Filtro Digitale

Elia Ribaldone, Andrea Trufini, Christian Fabiano

Docente: Maurizio Zamboni

Aprile - Maggio 2018

Indice

1 Introduzione

Il progetto da realizzare richiede la progettazione di un "filtro digitale" in grado di elaborare dei dati presenti in una memoria (MEM_A) e scrivere il risultato di tale elaborazione su di un'altra memoria (MEM_B), entrambe organizzate in 1024 word da 1 byte. L'operazione da eseguire, su di un i-esimo dato prelevato dalla MEM_A, è la seguente:

$$y(i) = (-1)^i \ [0.25x(i) + x(i-1) - 2x(n-3)]$$

Il circuito deve inoltre fornire in uscita la media dei campioni presenti sulla memoria A, e lo schema da realizzare in vista "top view" è il seguente:

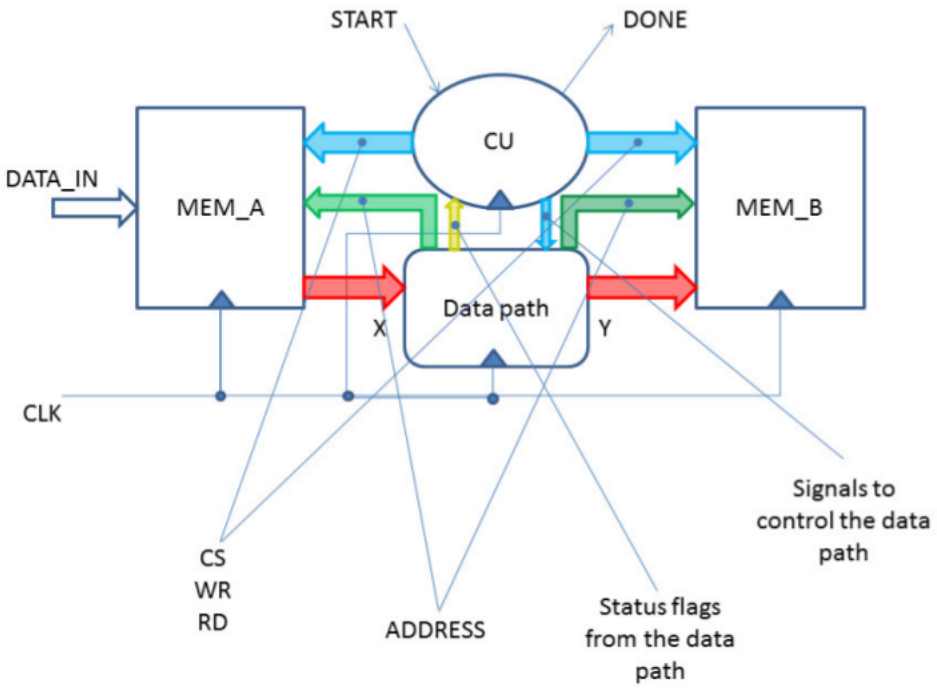

Figura 1: Top view

Le specifiche richiedono di utilizzare soltanto due sommatori per effettuare le operazioni necessarie, e che le memorie siano sincrone in scrittura ed asincrone in lettura. I passi operativi e le scelte di progetto effettuate vengono dettagliate nelle sezioni successive:

- Pseudocodice
- Data path
- ASM Chart
- Control ASM
- Timing
- VHDL
- Testbench e simulazione

2 Pseudocodice

Il linguaggio utilizzato per lo pseudocodice è, per semplicità di lettura, MATLAB, e il codice è preceduto da una descrizione verbale delle operazioni da svolgere.

2.1 Descrizione verbale algoritmo

1. Azzerare un contatore necessario per scorrere gli elementi di memoria da 0 fino a 1023;
2. Finché il contatore non raggiunge 1023, caricare un dato alla volta;
3. Prelevare l'iesimo dato dalla memoria;
4. Se l'indirizzo precedente all'i-esimo dato non è negativo, prelevare il secondo dato e nel frattempo svolgere il primo calcolo dell'espressione: dividere il numero per quattro e tenerlo da parte; contemporaneamente accumulare il numero estratto nel passo precedente in un altra locazione;
5. Se l'indirizzo corrispondente all'i-esimo dato sottratto di tre non è negativo, prelevare il terzo dato e nel frattempo svolgere il secondo calcolo dell'espressione: sommare al valore ottenuto in precedenza il valore prelevato dalla memoria nel punto 4;
6. Svolgere la terza parte del calcolo: sottrarre al valore ottenuto in precedenza il valore prelevato dalla memoria nel punto 5 moltiplicato per due;
7. Se il contatore è un numero dispari, invertire il numero trovato in precedenza;
8. Scrivere in memoria B il numero calcolato se compreso tra -128 e 127, altrimenti scrivere -128 se negativo, o 127 se positivo;
9. Incrementare di 1 il contatore;
10. Se il contatore non ha raggiunto il numero 1024, ritornare al punto 3;
11. Termine dell'operazione: fornire i dati in memoria B e dividere per 1024 il numero accumulato ad ogni ciclo nel punto 4.

2.2 Pseudocodice MATLAB

NOTA: a livello di pseudocodice MATLAB le operazioni non sono ancora divise in "stati": in realtà si è scelto di eseguire le operazioni di calcolo ed di lettura seguendo un approccio "pipeline", che rende la sequenza di operazioni leggermente diversa da quella descritta nello pseudocodice. Inoltre, la sintassi è in alcuni punti volutamente diversa da quella che sarebbe necessaria per compilare ed eseguire il programma, in quanto risulta di più facile comprensione.

```
%Inizializzazione registri
CNT_1024 = 0
REG_AVERAGE = 0

%Caricamento memoria
for CNT_1024 = 0:1023 %NOTA: l'incremento del contatore viene fatto in modo ...
    automatico
    MEM_A(CNT_1024, 1:8) = DATA_IN
    CNT_1024 = CNT_1024 + 1
end

%Scrittura dati in MEM_B
for CNT_1024 = 0:1023
    REG_FILTER = 0 %Reset somma
    REG_SAMPLES = MEM_A(CNT_1024, 1:8) %Read primo dato

    REG_FILTER = REG_SAMPLES/4; %Prima operazione
    REG_AVERAGE = REG_AVERAGE + REG_SAMPLES %Accumulo per media

    if(CNT_1024 - 1 >= 0)
        REG_SAMPLES = MEM_A(CNT_1024 - 1, 1:8) %Read secondo dato
        REG_FILTER = REG_FILTER + REG_SAMPLES %Seconda operazione
        if(CNT_1024 - 3 >= 0)
            REG_SAMPLES = MEM_A(CNT_1024 - 3, 1:8) %Read terzo dato
            REG_FILTER = REG_FILTER - 2*REG_SAMPLES %Terza operazione
        end
    end

    if((-1)^CNT_1014 < 0)
        REG_FILTER = -REG_FILTER %Inversione ogni 2 cicli
    end

    if(-128 <= REG_FILTER <= 127) %"Overflow" del dato
        MEM_B(CNT_1024, 1:8) = REG_FILTER
    elseif (REG_FILTER < 0)
        MEM_B(CNT_1024, 1:8) = -128
    else
        MEM_B(CNT_1024, 1:8) = 127
    end
end

REG_AVERAGE = REG_AVERAGE / 1024 %Divisione media
```

3 Data path

Il datapath, e la relativa unità di controllo, sono state progettate in modo da ridurre i cicli di clock necessari a svolgere l'operazione ed aumentare la massima frequenza di lavoro: viene effettuata una 'pipeline' delle fasi di aquisizione dati dalla memoria e relativa elaborazione, come descritto nella parte verbale dello pseudocodice.

3.1 Parallelismo

Il parallelismo scelto per il datapath è di 10 bit per le operazioni sui campioni, e 18 bit per l'operazione di media. La scelta di 10 bit è poiché le operazioni tra i tre dati diversi su 8 bit potrebbero dar luogo, nel caso in cui il terzo dato sia negativo, ad un numero rappresentabile su non meno di 10 bit. Esempio: 0 + 1 - 2* (-128) = 258 (non rappresentabile in C2 su 9 bit). I 18 bit per la media sono necessari perché l'operazione di divisione per 1024 viene fatta dopo che sono stati accumulati tutti i campioni, altrimenti si avrebbe una perdita di precisione eccessiva. Per cui, sommando 1024 dati da 8 bit ciascuno, sono necessari 18 bit.

3.2 Gestione indirizzi

Invece di usare tre contatori "sfasati" per la creazione degli indirizzi, si usa un singolo contatore in cui l'uscita va ad uno shift register a 3 registri, in modo da permettere all'unità di controllo, tramite un comando di selezione di un multiplexer a tre ingressi, di selezionare l'indirizzo corretto della MEM_A dal quale prelevare il dato. Per l'indirizzo di scrittura della MEM_B, viene usato lo stesso contatore.

3.3 Descrizione blocchi utilizzati

- **Counter_1024:** contatore a 10 bit utilizzato per la generazione degli indirizzi. Viene incrementato ad ogni ciclo di operazioni.

- **Registri e memorie:**

  Reg_Counter (1, 2 e 3): Registri a 10 bit usati come shift register per tenere memoria degli ultimi tre indirizzi. Il comando di shift viene dato ogni volta che il contatore si incrementa.

  Reg_Samples: Registro a 8 bit che sincronizza la lettura da MEM_A con le operazioni sincrone da svolgere. Separa l'operazione di lettura da quella di calcolo, realizzando un pezzo di pipeline.

  Reg_Filter: Registro a 10 bit che accumula il dato in uscita dal Sum_Filter. Viene resettato ad ogni incremento del contatore, ossia ad ogni nuova operazione. Poiché la MEM_B è su 8 bit, nel caso in cui il risultato dell'operazione non sia rappresentabile su 8 bit, si avrebbe un overflow nell'utilizzare celle di MEM_B per fare i calcoli. Per questo è stato necessario inserire un registro REG_FILTER.

  Reg_Average: Registro a 18 bit che accumula il valore di tutti i campioni prima di effettuare la divisione.

  MEM_A: Memoria RAM 1024x8 sincrona in scrittura, asincrona in lettura. Contiene i campioni da elaborare.

  MEM_B: Memoria RAM 1024x8 sincrona in scrittura, asincrona in lettura. Memoria in cui vengono scritti i risultati delle operazioni sui campioni.

- **Blocchi operazionali:**

  Sum_Filter: Sommatore ripple carry a 10 bit per svolgere le operazioni tra i campioni.

  Sum_Average: Sommatore ripple carry a 18 bit per svolgere l'operazione di media.

  Divider_by_1024: Blocco che effettua la divisione facendo uno shift di 10 sul dato in uscita da Reg_Average

  Three_Operation_Block: Blocco logico che ha in ingresso il campione proveniente da Reg_Samples, e svolge le operazioni di moltiplicazione per due e divisione per quattro del dato facendo uno shift del campione rispettivamente a sinistra e a destra.

- **Blocchi logici:**

  Valid_Address: Blocco logico in grado di capire quando l'indirizzo è compreso tra 0 e 2.

  Sample_CA_1: Porta XOR a 10 bit utilizzata per complementare il numero (quando si deve sottrarre il terzo dato).

  Block_Samples_n: Porta AND a 10 bit usata mentre si nega il valore a cicli alterni prima di scriverlo in MEM_B, equivale a sommare '0' al dato in Reg_Filter per quel colpo di clock.

  Ovf_Sign_10_to_8_Filter: Blocco usato per controllare preventivamente alla scrittura in MEM_B la corretta rappresentazione del dato su 8 bit e il segno del dato.

- **Mux_Address:** Multiplexer 3-1 a 10 bit, seleziona l'indirizzo da dare in ingresso alla MEM_A
- **Mux_Operation:** Multiplexer 3-1 a 10 bit che seleziona quale delle tre operazioni fatte da Three_Operation_Block dare in ingresso al sommatore.
- **Mux_Saturation:** Multiplexer 3-1 a 10 bit che manda in uscita verso MEM_A il numero proveniente da Reg_Filter in caso rappresentabile in C2 su 8 bit, -128 o 127 altrimenti.

3.4 Interfacciamento

L'interfaccia verso l'esterno è del tipo asincrono => sincrono, in quanto l'inizio dell'operazioni avviene solo dopo il campionamento di un segnale di START. La lettura della memoria B e della media è corretta solo quando il segnale DONE è a 1.

3.5 Schema datapath

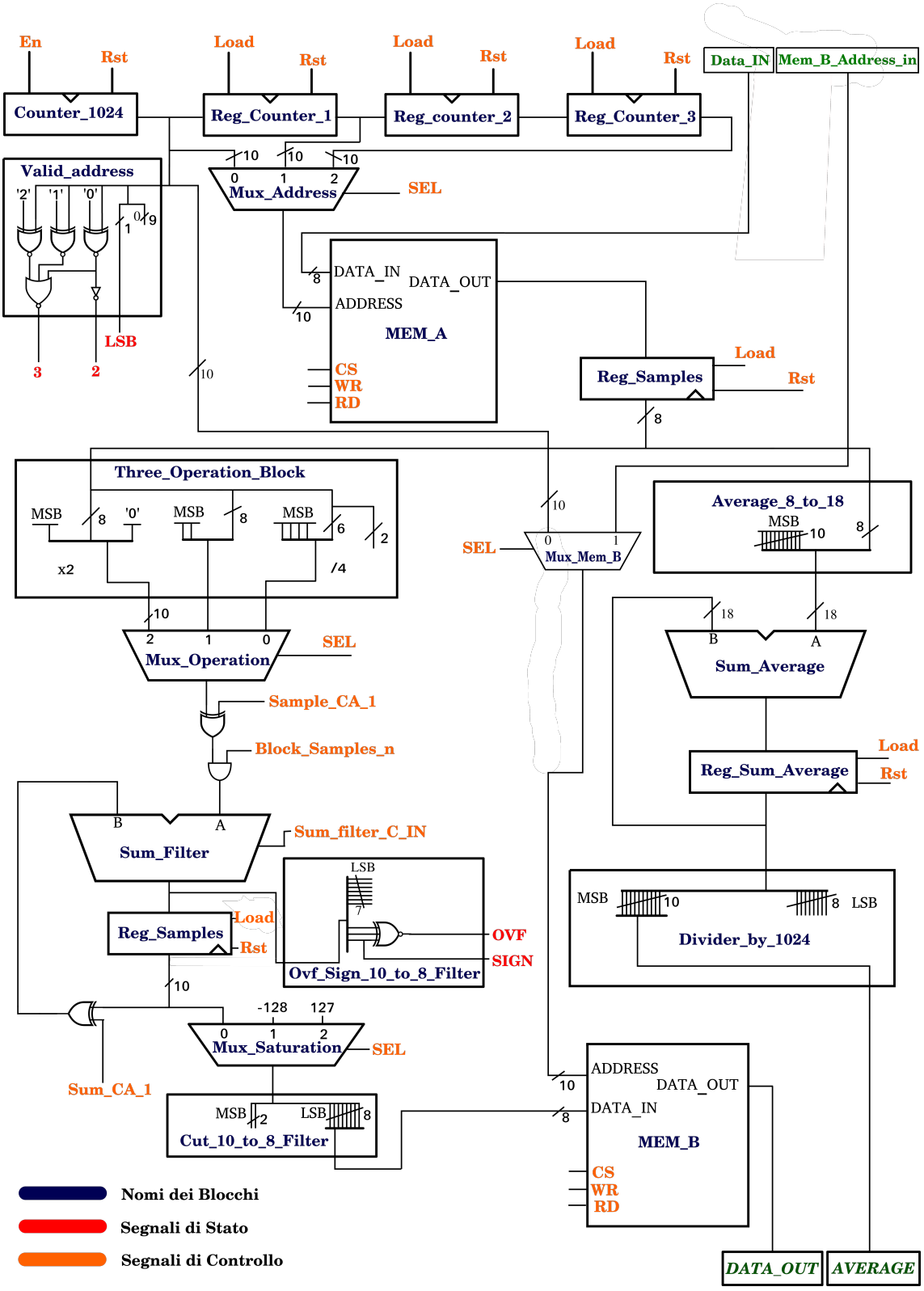

Figura 2: Datapath

4 ASM chart

4.1 Descrizione stati

- **Reset:**
  Stato di reset della macchina. Non è necessario azzerare MEM_A e MEM_B in quanto verranno entrambe sovrascritte. Viene azzerato il contatore e l'accumulatore per la media, e viene messo a 0 il segnle di "DONE"

- **S0_MEM_A:**
  Caricamento della MEM_A. Ad ogni colpo di clock viene memorizzato, nella locazione puntata dal contatore, il dato presente sull'ingresso DATA_IN fornito dall'esterno.

- **S1_Load_Data_1:**
  Dopo aver raggiunto il terminal count ed essersi azzerato, viene prelevato dalla memoria A il dato puntato dall'i-esimo indirizzo. Viene intanto azzerato Reg_Filter poiché è appena iniziato il calcolo del nuovo valore da scrivere in MEM_B

- **S2_Execute_Data_1:**
  Viene eseguito il passo uno del calcolo, ossia $y(i) = x(i)/4$. Viene accumulato il primo valore della media. Intanto viene caricato il dato puntato dall'i-esimo indirizzo sottratto di uno per l'esecuzione successiva

- **S3_Execute_Data_2:**
  Viene eseguito il passo due del calcolo, ossia $y(i) = x(i)/4 + x(i-1)$. Intanto viene caricato il terzo dato puntato dall'i-esimo indirizzo sottratto di tre per l'esecuzione successiva

- **S3_Invalid_2:**
  Nel caso in cui l'indirizzo i-1 sia negativo, non viene eseguito né il passo due né il passo tre del calcolo Esecuzione del passo uno del calcolo e dell'accumulazione media (in quanto in ramo alternativo dell'ASM)

- **S3_Invalid_3:**
  Nel caso in cui l'indirizzo i-3 sia negativo, non viene il passo tre del calcolo Esecuzione del passo due del calcolo (in quanto in ramo alternativo dell'ASM)

- **S4_Execute_Data_3:**
  Viene eseguito il passo tre del calcolo, ossia $y(i) = x(i)/4 + x(i-1) - 2 * x(i-3)$

- **S5_Invert:**
  Se Counter_1024 è dispari, inverte Reg_Filter.

- **S6_Write_Sum:**
  Se il dato calcolato è rappresentabile in C2 su 8 bit, lo scrive in MEM_B alla posizione puntata da Counter_1024

- **S6_Write_127:**
  Se il dato calcolato NON è rappresentabile in C2 su 8 bit, ed è positivo, scrive in MEM_B 127, ossia il più grande positivo rappresentabile con quel parallelismo.

- **S6_Write_Neg_128:**
  Se il dato calcolato NON è rappresentabile in C2 su 8 bit, ed è negativo, scrive in MEM_B -128, ossia il più grande negativo rappresentabile con quel parallelismo.

- **S7_Done**
  Al termine delle operazioni, viene abilitato il segnale "DONE" e il Mux_Mem_B farà indirizzare la MEM_B dall'esterno invece che dal contatore, per consentire la lettura dei dati raccolti.

4.2 ASM Chart

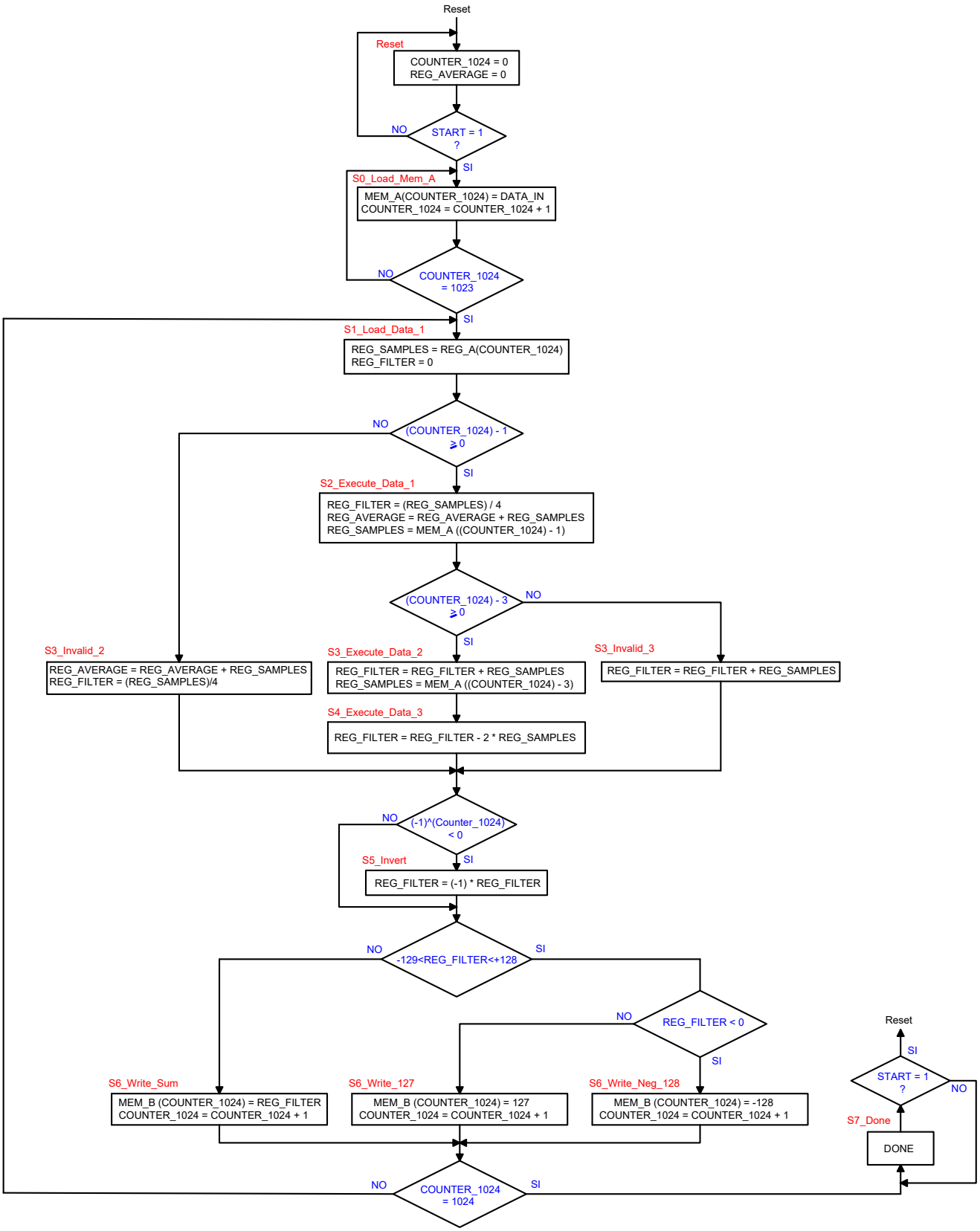

Figura 3: ASM Chart

5 Control ASM

Nel control ASM sono stati inseriti i comandi da dare in ogni stato dell'ASM chart. In ogni stato è stato sostituito al posto dell'azione da svolgere, il COMANDO per eseguire tale azione.

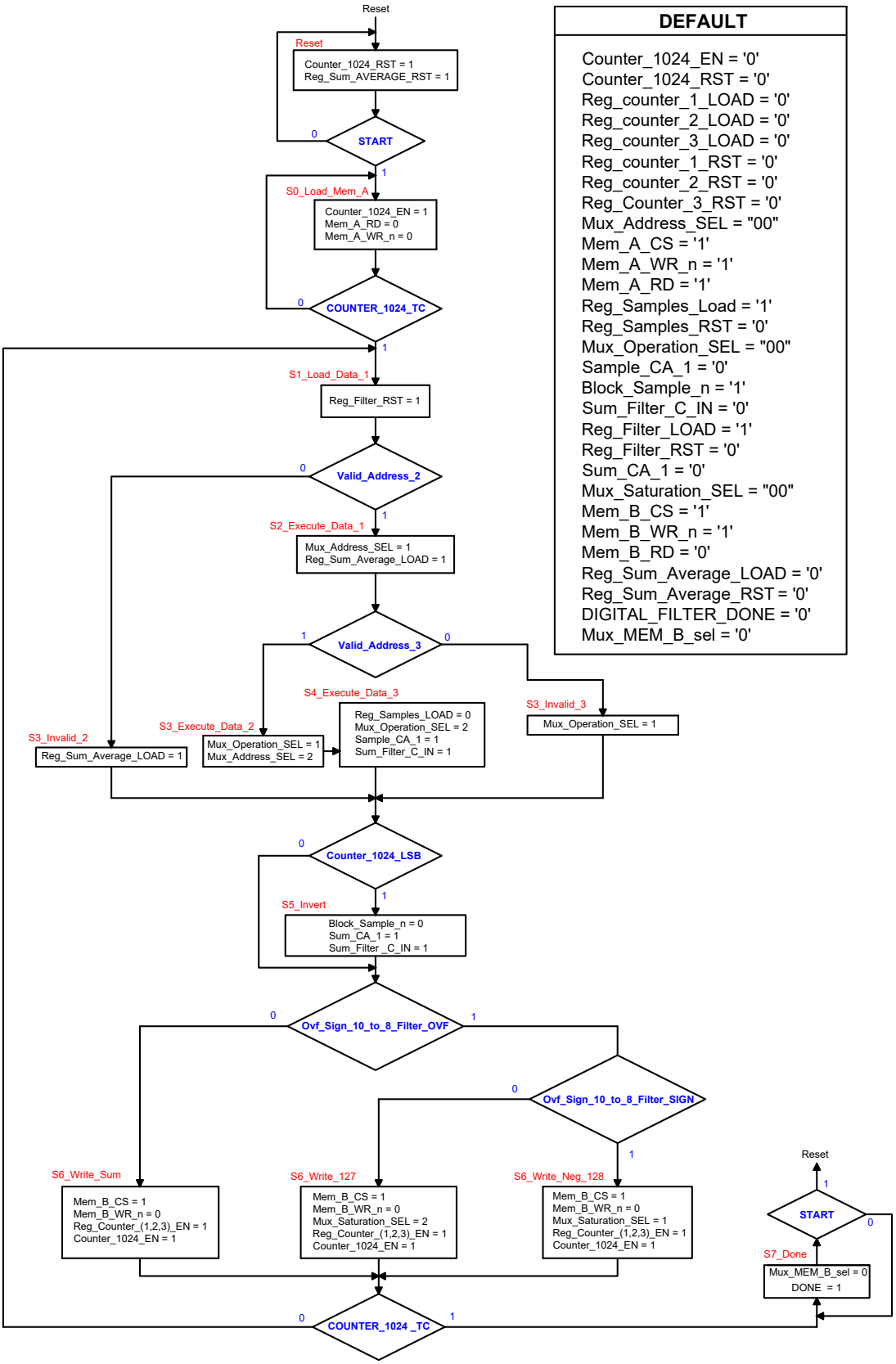

Figura 4: Control ASM

6 Timing

I frammenti di timing ritenuti più significativi sono la fase di caricamento della memoria e la prima fase di calcolo del circuito, quella in cui per i primi campioni estratti il procedimento devia leggermetne dal comportamento a regime. Sono stati inoltre inseriti per il timing numeri che mostrano il comportamento del circuito in caso di non rappresentabilità su 8 bit del risultato dell'operazione (Stato S7_write_127), prima della scrittura nella memoria B.
Dal confronto con la simulazione del timing effettuata da MODELSIM è emersa coerenza con le ipotesi fatte nella scrittura manuale del timing.
Per motivi di leggibilità sono stati omessi alcuni dei segnali che non cambiano durante il timing, ed il loro valore di default è specificato nella precedente pagina.

6.1 Timing caricamento memoria A

Per l'avvio delle operazioni, come descritto nella sezione 3.4, si aspetta di campionare il segnale di START.

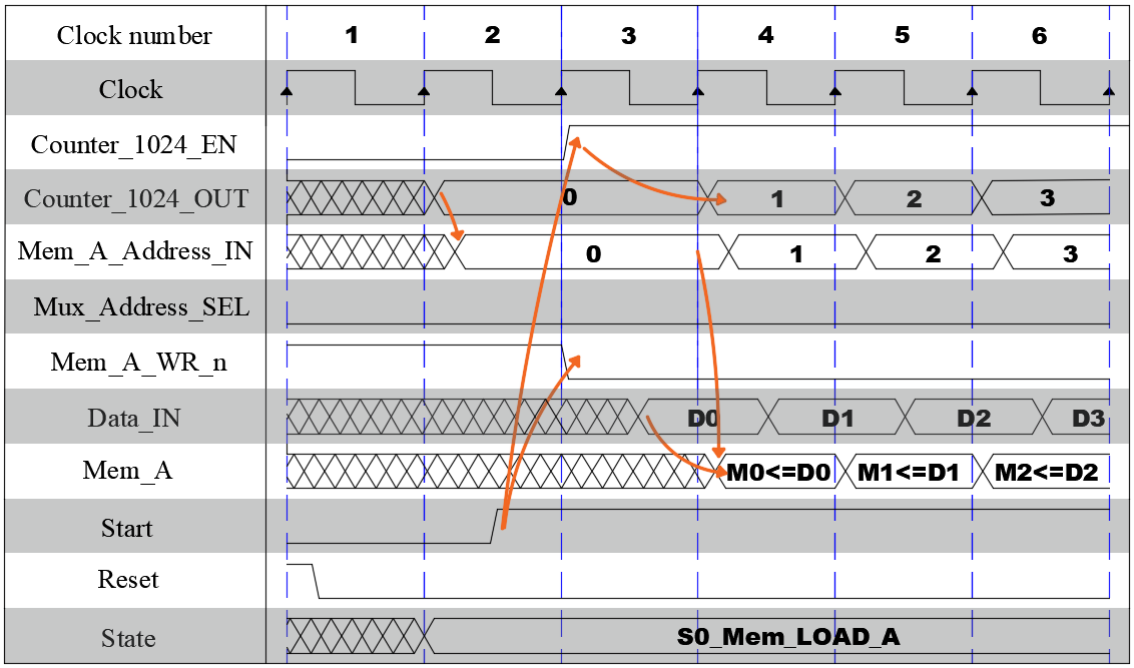

Figura 5: Timing caricamento MEM_A

6.2 Timing caricamento memoria A [SIMULAZIONE]

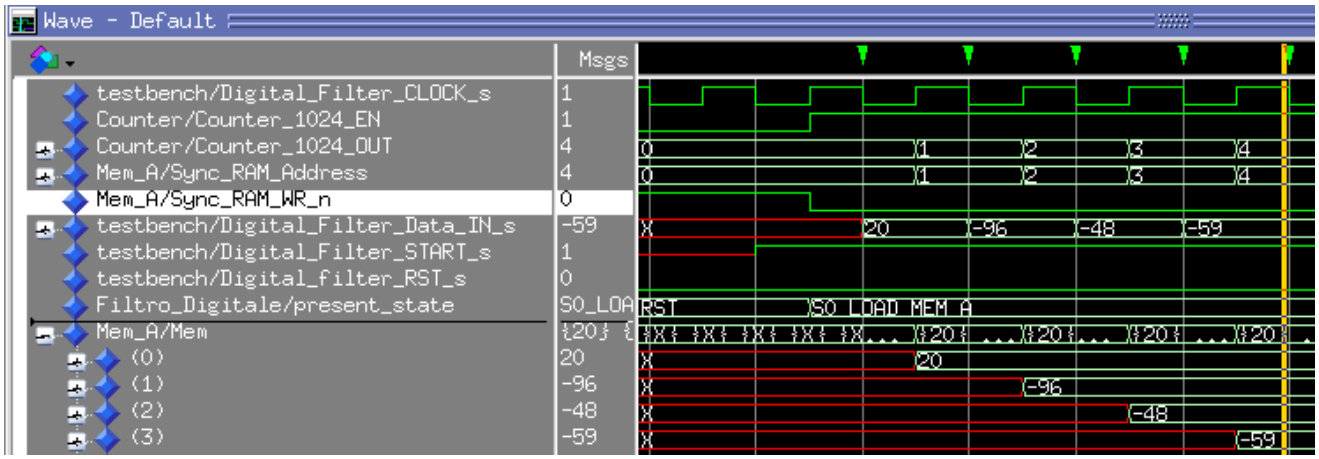

Figura 6: Timing caricamento MEM_A [SIMULAZIONE]

6.3 Svolgimento calcolo sui primi indirizzi

Come detto in precedenza, il timing mostra l'evoluzione dei segnali a partire da fine caricamento memoria e inizio calcolo. Per i primi tre dati gli indirizzi risulterebbero negativi quindi gli stati evolvono come descritto nell'ASM chart. Per questioni di leggibilità, il timing è stato diviso in due in corrispondenza del colpo di clock n° 9.
NOTA: st(x,y) è stata usata come notazione compatta per indicare Mem(x) <= y.

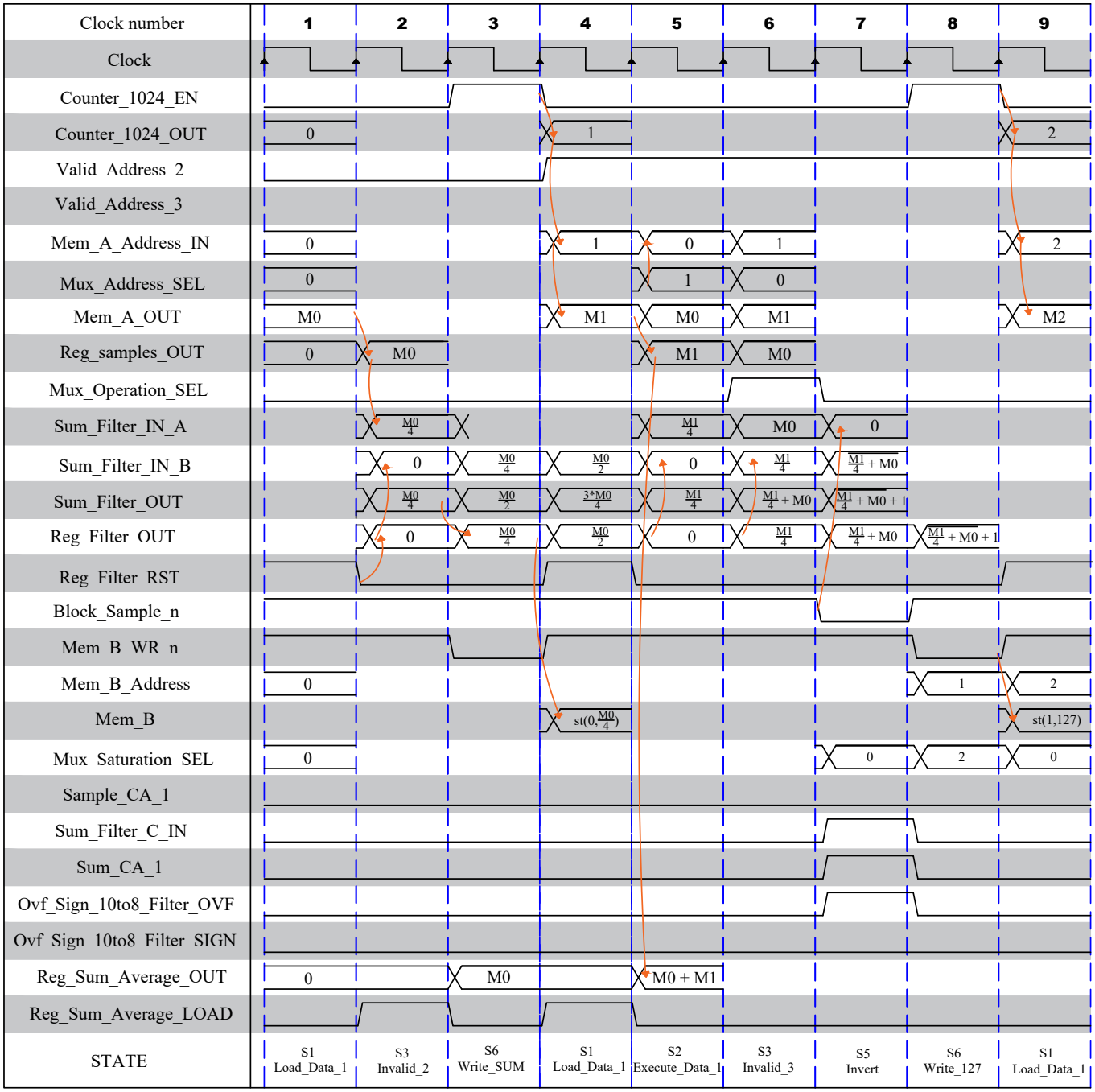

Figura 7: Timing inizio calcoli - 1

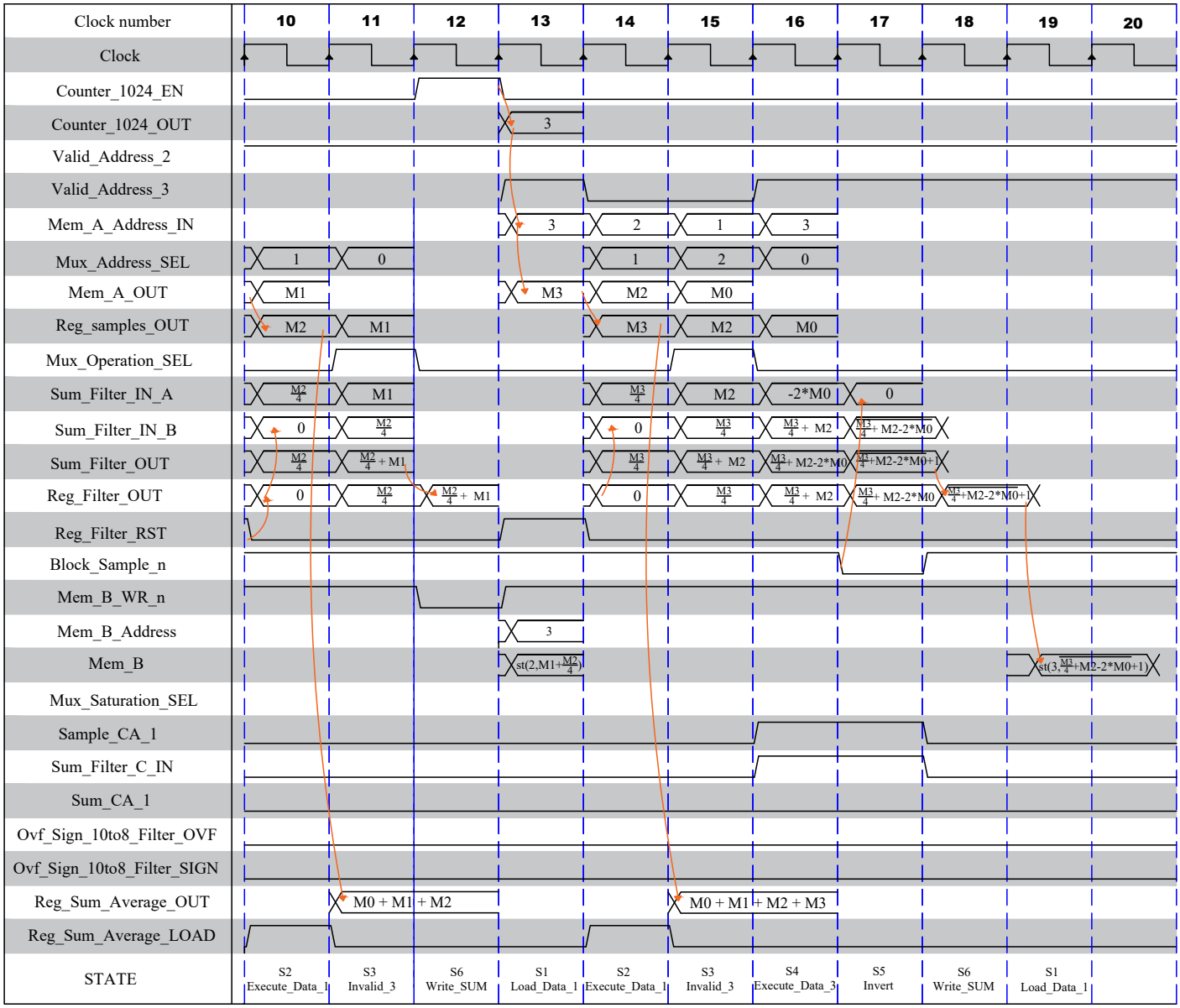

Figura 8: Timing inizio calcoli - 2

6.4 Svolgimento calcolo sui primi indirizzi [SIMULAZIONE]

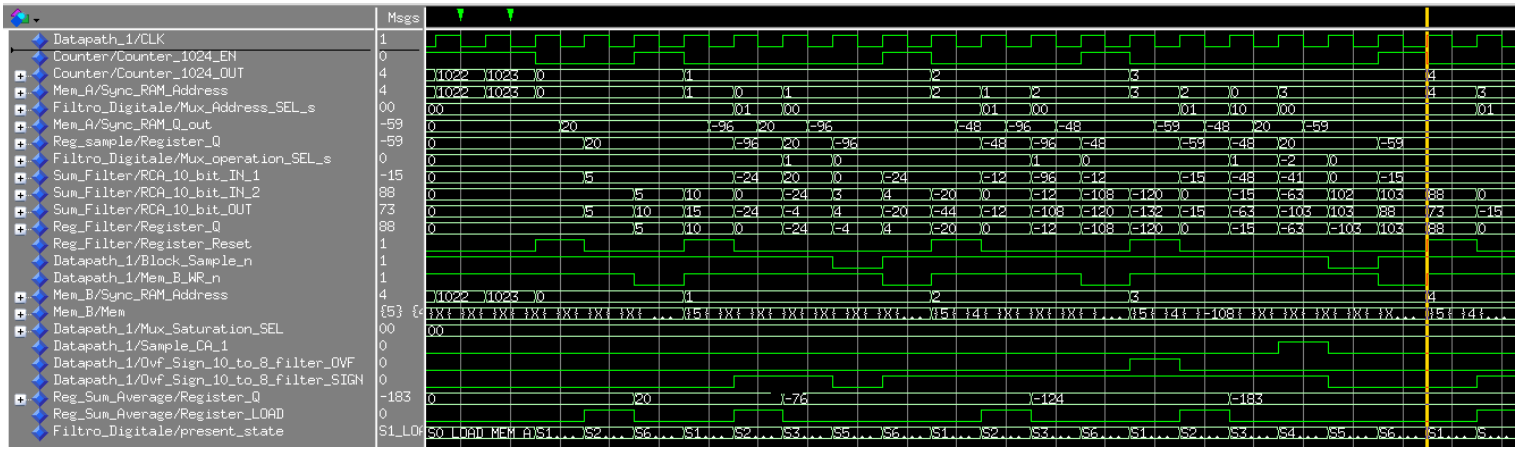

Figura 9: Timing inizio calcoli [SIMULAZIONE]

7 Testbench e simulazione

7.1 Schema simulazione

Per verificare la correttezza delle operazioni svolte dal circuito, si è voluto evitare di realizzare in VHDL tale controllo in quanto facendo una descrizione ad alto livello con lo stesso linguaggio gli output generati dal testbench in VHDL potrebbero più facilmente essere affetti dagli stessi errori del circuito.
Si è dunque proceduto nel seguente modo:

1. Creazione di un file MEM_A_BIN con numeri binari su 8 bit generati in modo casuale;
2. Esecuzione calcoli del circuito digital filter;
3. Scrittura su di un file MEM_B_BIN dei numeri binari generati dal circuito;
4. Comparazione dei file MEM_B_BIN con i file generati da 3 diversi procedimenti software: C, MATLAB, bash.

NOTA: l'operazione di divisione per 4 via software è stata fatta in codice in modo da prendere sempre il numero più basso quando c'è un eventuale resto (Esempio: $3/4 = 0$, $-3/4 = -1$), per rispecchiare il comportamento del circuito.

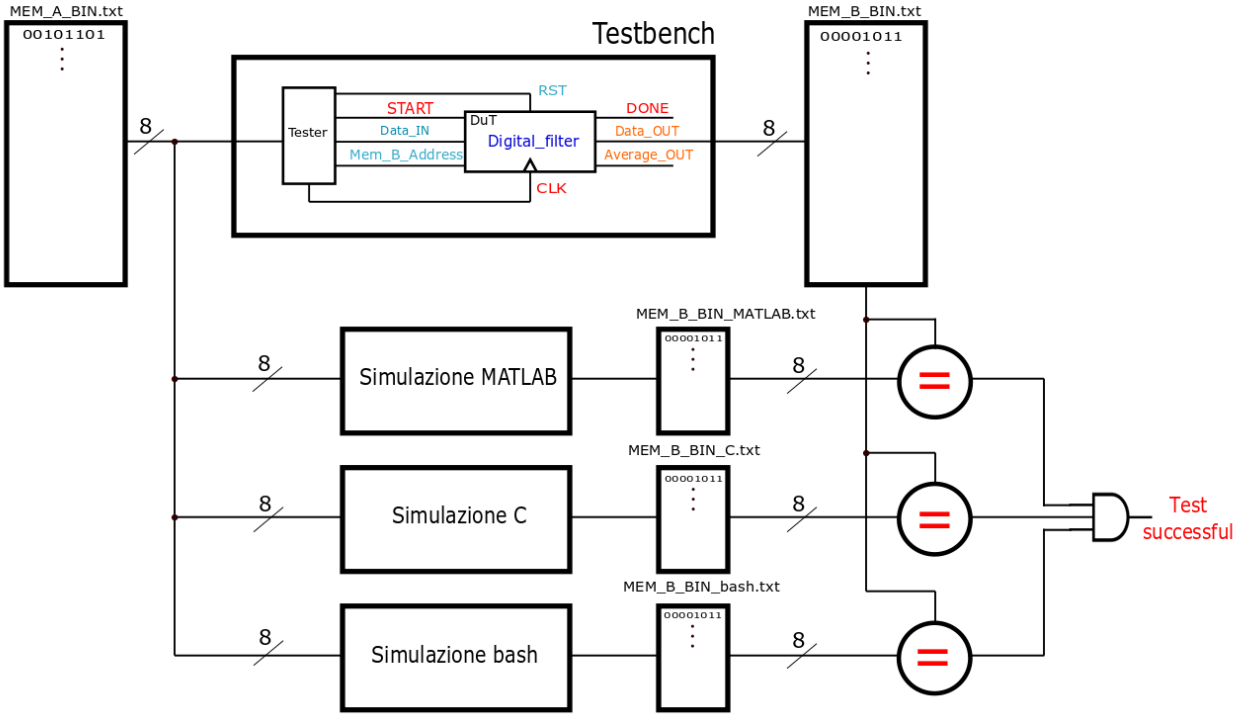

Figura 10: Schema della simulazione

Dopo diverse prove di congruenza tra dati generati dal circuito dal programma ed i dati generati in C/Matlab/bash si è ritenuto il circuito funzionante.

Seguono nelle pagine successive i codici di testbench e delle simulazioni software.

7.2 Simulazione e analisi timing su Modelsim

A scopo di mostrare le proporzioni tra i periodi impiegati per il caricamento della MEM_A e dell'esecuzione delle operazioni, viene riportata una simulazione di modelsim:

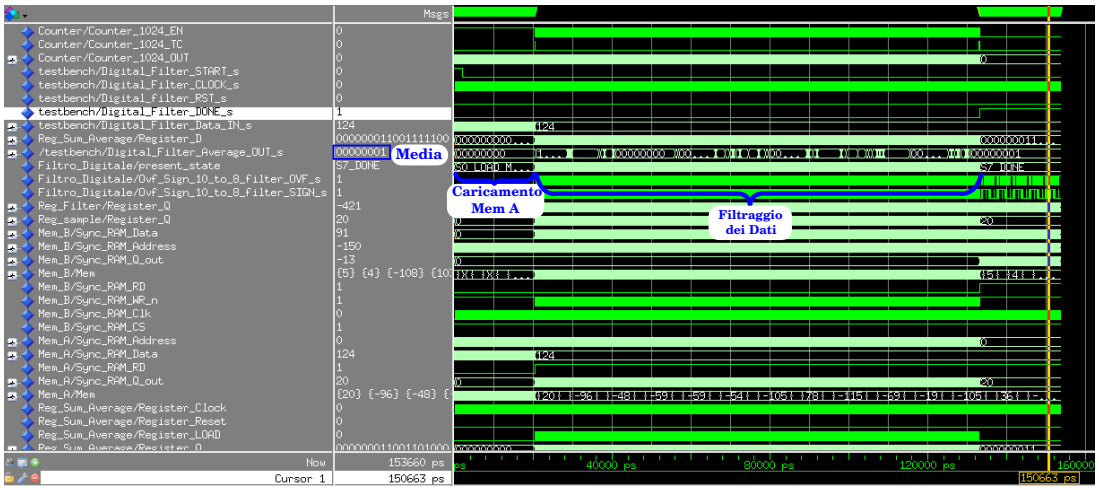

Figura 11: Modelsim - Simulazione del codice VHDL

Per quanto riguarda la massima frequenza di lavoro, con il circuito progettato è possibile raggiungere 81 MHz. Il ritardo maggiore è introdotto dal sommatore (che è di tipo ripple-carry, in quanto usandone uno diverso con questo parallelismo non si otterrebbero vantaggi considerevoli).

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	81.62 MHz	81.62 MHz	Digital_Filter_CLOCK	

Figura 12: Modelsim - Timing analysis

7.3 Testbench VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4  USE std.env.all;
5  USE std.Textio.all;
6
7  ENTITY TestBench IS
8  END TestBench;
9
10
11 ARCHITECTURE behaviour OF TestBench IS
12
13     -- definisco il componente
14     COMPONENT Digital_Filter
15         PORT(
16                 Digital_Filter_START: IN STD_LOGIC;
17                 Digital_Filter_CLOCK: IN STD_LOGIC;
18                 Digital_filter_RST: IN STD_LOGIC;
19                 Digital_Filter_Data_IN: IN SIGNED (7 DOWNTO 0);
20                 Digital_Filter_Data_OUT: OUT SIGNED (7 DOWNTO 0);
21                 Digital_Filter_Mem_B_Address : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
22                 Digital_Filter_Average_OUT: OUT SIGNED (7 DOWNTO 0);
23                 Digital_Filter_DONE: OUT STD_LOGIC
24             );
25     END COMPONENT;
26
27     -- segnali di INGRESSO
28     SIGNAL Digital_Filter_START_s ,Digital_Filter_CLOCK_s, Digital_filter_RST_s: ...
           STD_LOGIC;
29     SIGNAL Digital_Filter_Data_IN_s : SIGNED (7 DOWNTO 0);
30     SIGNAL Digital_Filter_Mem_B_Address_s : STD_LOGIC_VECTOR(9 DOWNTO 0);
31
32     -- segnali di USCITA
33     SIGNAL Digital_Filter_DONE_s : STD_LOGIC;
34     SIGNAL Digital_Filter_Data_OUT_s, Digital_Filter_Average_OUT_s: SIGNED (7 ...
           DOWNTO 0);
35
36     -- dafinizione dei file usati come input e output e del file con i dati di ...
           uscita giusti
37     CONSTANT name_file_data_in : STRING := "MEM_A_BIN.txt";
38     CONSTANT name_file_data_out : STRING := "MEM_B_BIN.txt";
39     CONSTANT name_file_data_out_dec : STRING := "MEM_B_DEC.txt";
40
41     -- TEMPI COSTANTI PER LA SIMULAZIONE
42     CONSTANT CLK_PERIOD : TIME := 20 ps;
43     CONSTANT N_CICLE : INTEGER := 500000 ;
44     CONSTANT SIM_TIME : TIME := N_CICLE*20 ps;
45
46     -- segnali per i tempi di simulazione
47     SIGNAL END_SIM : BOOLEAN := FALSE;
48
49
50 BEGIN
51
52     --  istanzio il DUT
53     Filtro_Digitale : Digital_Filter
54         PORT map
55         (
56             Digital_Filter_START => Digital_Filter_START_s,
57             Digital_Filter_CLOCK => Digital_Filter_CLOCK_s,
58             Digital_filter_RST => Digital_filter_RST_s,
59             Digital_Filter_Data_IN => Digital_Filter_Data_IN_s,
60             Digital_Filter_Data_OUT => Digital_Filter_Data_OUT_s ,
61             Digital_Filter_Mem_B_Address =>Digital_Filter_Mem_B_Address_s ,
62             Digital_Filter_Average_OUT => Digital_Filter_Average_OUT_s ,
63             Digital_Filter_DONE => Digital_Filter_DONE_s
64         );
```

```
65
66      -- PROCESSO CHE ESEGUE le seguenti istruzioni:
67      --      - legge il file input E LI d i dati in ingresso al DUT (ciclo)
68      --          - alla fine preleva i dati dalla memoria b e li scrive sul file ...
            di uscita
69
70      file_and_checker :
71      PROCESS
72
73          -- dichiaro i file utilizzati come testo
74          FILE file_data_in , file_data_out, file_data_out_dec : text ;
75
76          -- variabili per i file
77          VARIABLE line_va, line_dec: LINE;
78          VARIABLE number_va : SIGNED(7 DOWNTO 0);
79          --VARIABLE good_v : BOOLEAN;
80
81      BEGIN
82
83          -- files di input
84          file_open(file_data_in, name_file_data_in, read_mode);
85          file_open(file_data_out_dec, name_file_data_out_dec, write_mode);
86          -- file di output
87          file_open(file_data_out, name_file_data_out, write_mode);
88
89          WAIT FOR 6*CLK_PERIOD+CLK_PERIOD;
90
91              -- in questo ciclo vengonon presi i valori dal file file_data_in e ...
                    salvati nella Mem_A
92              Mem_A_write_loop:
93                  FOR i IN 0 TO 1023 LOOP
94                      readline(file_data_in, line_va);
95                      -- scrivo a terminale la riga per verificarne il contenuto
96                      -- per poter scrivere il contatore i a terminale devo ...
                            convertirlo in una stringa
97                      report "WRITE IN A " & integer'image(i) & " : " & line_va.all;
98                      read(line_va, number_va);
99                      Digital_Filter_Data_IN_s <= number_va;
100                     WAIT FOR CLK_PERIOD;
101                 END LOOP Mem_B_write_loop;
102             -- chiudo il file di input
103             file_close(file_data_in);
104
105             -- aspetto che il DUT ABBIA FINITO
106             Waiting_dut_done_loop:
107                 WHILE Digital_Filter_DONE_s = '0' LOOP
108                     WAIT FOR CLK_PERIOD;
109                 END LOOP Waiting_dut_done_loop;
110
111
112             -- leggo i valori dalla memoria b e li salvo sul file file_data_out
113             Saving_data_in_mem_B_loop:
114                 FOR i IN 0 TO 1023 LOOP
115                     -- trasformo il contatore da intero ad uno std_logic_vector ...
                            di 10 bit
116                     -- per puntare progressivamente l'indirizzo della memoria b ...
                            da salvare nel file
117                     Digital_Filter_Mem_B_Address_s <= ...
                            STD_LOGIC_VECTOR(TO_UNSIGNED(i,10)) ;
118                     WAIT FOR CLK_PERIOD;
119                     -- scrivo la prima riga in line_va
120                     write(line_va, Digital_Filter_Data_OUT_s);
121                     write(line_dec, to_integer(Digital_Filter_Data_OUT_s));
122                     -- scrivo a terminale la riga per verificarne il contenuto
123                     -- per poter scrivere il contatore i a terminale devo ...
                            convertirlo in una stringa
124                     report "READ IN B " & integer'image(i) & " : " & line_va.all;
125                     -- salvo line_va nel file
126                     writeline(file_data_out, line_va);
127                     writeline(file_data_out_dec, line_dec);
```

```
128
129                 END LOOP Saving_data_in_mem_B_loop;
130
131             -- chiudo il file di output
132             file_close(file_data_out);
133         finish(0);
134     END PROCESS file_and_checker;
135
136     -- PROCESSO DI CLOCK
137     CLK:
138     PROCESS
139     BEGIN
140         CLK_CICLE:
141         FOR i IN 0 TO N_CICLE LOOP
142             Digital_Filter_CLOCK_s <= '0' , '1' AFTER CLK_PERIOD/2;
143             WAIT FOR CLK_PERIOD;
144         END LOOP CLK_CICLE;
145         --- ferma la simulazione dopo N_CICLE cicli di clock
146         finish(0);
147     END PROCESS CLK;
148
149     -- PROCESSO DI RESET
150     RST:
151     PROCESS
152     BEGIN
153         Digital_filter_RST_s <= '1';
154         WAIT FOR CLK_PERIOD*2;
155         Digital_filter_RST_s <= '0';
156         WAIT FOR  SIM_TIME;
157     END PROCESS;
158
159     -- PROCESSO DI START
160     START:
161     PROCESS
162     BEGIN
163         WAIT FOR CLK_PERIOD*6;
164         Digital_filter_START_s <= '1';
165         WAIT FOR CLK_PERIOD*100;
166         Digital_filter_START_s <= '0';
167         WAIT FOR  SIM_TIME;
168     END PROCESS;
169     --- questa parte serve per arrestare la simulazione al SIM_TIME settato
170     stop_simulation :
171     PROCESS
172     BEGIN
173         WAIT FOR SIM_TIME; --run the simulation for this duration
174         END_SIM <= true;
175     END PROCESS;
176
177 END behaviour;
```

7.4 Simulazione MATLAB

```
clc;
clear;
close all;

%Creazione matrice binaria casuale
 for CNT_1024 = 1:1024
    MEM_A_BIN(CNT_1024, 1:8) = randi([0 1], 1, 8);
 end

 int16 MEM_A
%Conversione matrice MEM_A_BIN in decimale (MEM_A_BIN => MEM_A)
for CNT_1024 = 1:1024
    MEM_A(CNT_1024) = typecast(uint8(bin2dec(num2str(MEM_A_BIN(CNT_1024, ...
        1:8)))), 'int8');
end

REG_AVERAGE = int16(0);
%Scrittura dati in MEM_B
for CNT_1024 = 1:1024
    REG_FILTER = int16(0); %Reset somma
    REG_SAMPLES = int16(MEM_A(CNT_1024)) %Read primo dato

    REG_AVERAGE = REG_AVERAGE + REG_SAMPLES; %Accumulo per media
    REG_FILTER = idivide(REG_SAMPLES, 4, 'floor') %Prima operazione, divide ...
        troncando al num. pi basso

    if(CNT_1024 - 1 >= 1)
        REG_SAMPLES = int16(MEM_A(CNT_1024 - 1)); %Read secondo dato
        REG_FILTER = REG_FILTER + REG_SAMPLES; %Seconda operazione
        if(CNT_1024 - 3 >= 1)
            REG_SAMPLES = int16(MEM_A(CNT_1024 - 3)); %Read terzo dato
            REG_FILTER = REG_FILTER - 2*REG_SAMPLES; %Terza operazione
        end
    end

    if((-1)^(CNT_1024-1) < 0)
        REG_FILTER = -REG_FILTER; %Inversione ogni 2 cicli
    end

    if(REG_FILTER >= -128 && REG_FILTER <= 127) %"Overflow" del dato
        MEM_B(CNT_1024) = REG_FILTER;
    else
        if(REG_FILTER < 0)
        MEM_B(CNT_1024) = -128;
        elseif(REG_FILTER >0)
        MEM_B(CNT_1024) = 127;
        end
    end
end

REG_AVERAGE = idivide(REG_AVERAGE, 1024, 'floor') %Divisione media

%Conversione matrice MEM_B in binario (MEM_B => MEM_B_BIN)
for CNT_1024 = 1:1024
    num = MEM_B(CNT_1024)
    if(num < 0)
        MEM_B_BIN(CNT_1024, 1:8) = ['1', dec2bin(2^7 + num, 7)];
    else
        MEM_B_BIN(CNT_1024, 1:8) = dec2bin(num, 8);
    end
end

%Creazione file binari matrice A e B
dlmwrite("MEM_A.txt", MEM_A, 'delimiter', '\n');
dlmwrite("MEM_B.txt", MEM_B, 'delimiter', '\n');
dlmwrite("MEM_A_BIN.txt", MEM_A_BIN, 'delimiter', '');
dlmwrite("MEM_B_BIN_MATLAB.txt", MEM_B_BIN, 'delimiter', '');
```

7.5 Simulazione C

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <math.h>

#define row_dim 8
#define col_dim 1024
#define file_name_dim 100

void CA_2(int num, int *num_bin, int sign);
void CA_2_18(int num, int *num_bin, int sign);

int main(int argc, char ** argv ) {
    FILE *Mem_A_data;
    FILE *Mem_B_data;
    FILE *Mem_A_data_dec;
    FILE *Mem_B_data_dec;


    FILE *File_1;
    FILE *File_2;

    int Mem_A[col_dim], Mem_B[col_dim];
    int Mem_A_CA2[row_dim], Mem_B_CA2[row_dim];
    int i,j;
    int average, average_all, average_CA2[row_dim];
    int sign=1;
    int equal;
    int data_1, data_2;
    char file_1[file_name_dim];
    char file_2[file_name_dim];

    /*si utilizza un argomento da riga di comando per selezionare la modalit  ...
        di utilizzo del programma, ovvero:
     * - se  argv[0] = g: il programma funziona come generatore dei due file ...
         contenente i dati da introdurre in MEM_A
     *   e quelli teoricamente ottenuti per MEM_B;
     * - se argv[0] = c : il programma funziona come comparatore di due file, ...
         ovvero compara i dati che teoricamente
     *   essere presenti nella MEM_B e quelli che effettivamente sono stati ...
         ottenuti dalla simulazione circuitale
     *   argv[2] -> file dove vengono salvati i dati random
     *   argv[3] -> file dove vengono salvati i dati random in decimale
     *   argv[4] -> file dove vengono salvati i dati filtrati
     *   argv[4] -> file dove vengono salvati i dati filtratiin decimale

     *   */
    if(argc ==1){
        printf("- se argv[0] = g : il programma funziona come generatore dei \n ...
                    due file contenente i dati da introdurre in MEM_A \n ...
                   e quelli teoricamente ottenuti per MEM_B; \n- se argv[0] = ...
            c : il programma funziona come comparatore\n       di due file, ...
            ovvero compara i dati che teoricamente \n         essere presenti ...
            nella MEM_B e quelli che effettivamente\n        sono stati ...
            ottenuti dalla simulazione circuitale \nargv[2] -> file dove ...
            vengono salvati i dati random \nargv[3] -> file dove vengono ...
            salvati i dati random in decimale \nargv[4] -> file dove vengono ...
            salvati i dati filtrati\nargv[4] -> file dove vengono salvati i ...
            dati filtrati in decimale\n");
        return EXIT_FAILURE;
    }
    if(argc != 6) {
        printf("Errore nella riga di comando!\n");
        printf("h per l'help\n");
        return EXIT_FAILURE;
    }

```

```
55      if(argv[1][0] == 'g') {
56
57          printf("\nMODALITA' 0: GENERAZIONE DATI\n");
58          // file dove vengono salvati i dati random
59          Mem_A_data = fopen(argv[2], "w");
60              if (!Mem_A_data) {
61                  printf("Impossibile aprire il file 'Mem_A_data'");
62                  return EXIT_FAILURE;
63              }
64          // file dati random in decimale
65          Mem_A_data_dec = fopen(argv[3], "w");
66              if (!Mem_A_data_dec) {
67                  printf("Impossibile aprire il file 'Mem_A_data_dec'");
68                  return EXIT_FAILURE;
69              }
70
71          srand(time(NULL));          //generazione di valori casuali per la ...
                generazione dei dati da inserire in MEM_A
72          for (i = 0; i < col_dim; i++) {
73              sign = rand() % 2;          //vengono in questo modo generati 0 od ...
                    1 casualmente per il bit di segno
74              if (sign == 0)          //se il numero    negativo
75                  Mem_A[i] = (-1) * rand() % 129;   //saturo a -128
76              else                    //se il numero    positivo
77                  Mem_A[i] = rand() % 128;   //saturo a 127
78              if (Mem_A[i]==0)
79                  sign=0;
80
81              CA_2(Mem_A[i], Mem_A_CA2, sign);   //funzione che esegue il ...
                    complemento a 2 se il numero    negativo
82              for (j = 0; j < row_dim; j++)
83                  fprintf(Mem_A_data, "%d", Mem_A_CA2[j]);
84
85              fprintf(Mem_A_data, "\n");
86
87              fprintf(Mem_A_data_dec, "%d\n", Mem_A[i]);
88          }
89          fclose(Mem_A_data);
90          fclose(Mem_A_data_dec);
91
92          for (i = 0; i < 1024; i++) {
93              /* se il numero    negativo e non    divisibile per 4 nel sommatore ...
                    viene approssimato per difetto
94               * quindi    necessario sottrarre uno al numero trovato usando gli ...
                     interi, perch  con essi il numero
95               * sempre approssimato per difetto ma in modulo. */
96              if (Mem_A[i] < 0 && Mem_A[i] % 4 != 0) {
97                  Mem_B[i] = (Mem_A[i] / 4) - 1;
98                  // printf("%d\n", Mem_B[i]);
99              } else
100                 Mem_B[i] = (Mem_A[i] / 4);
101
102             if (i > 0)
103                 Mem_B[i] += Mem_A[i - 1];
104             if (i > 2)
105                 Mem_B[i] += (-2) * Mem_A[i - 3];
106             if (i%2 != 0)
107                 Mem_B[i] = Mem_B[i]*(-1);
108             if (Mem_B[i] > 127)
109                 Mem_B[i] = 127;
110             if (Mem_B[i] < -128)
111                 Mem_B[i] = -128;
112         }
113
114         /* scrittura dei dati teoricamente calcolati e che dovrebbero essere ...
                presenti in MEM_B */
115         Mem_B_data = fopen(argv[4], "w");
116             if (!Mem_B_data) {
117                 printf("Impossibile aprire il file 'Mem_B_data'");
118                 return EXIT_FAILURE;
119             }
120         Mem_B_data_dec = fopen(argv[5], "w");
```

```
121             if (!Mem_B_data_dec) {
122                 printf("Impossibile aprire il file 'Mem_B_data_dec'");
123                 return EXIT_FAILURE;
124             }
125 
126         if (Mem_B_data) {
127                 for (i = 0; i < col_dim; i++) {
128                     if (Mem_B[i] > 0)
129                         sign = 1;
130                     else
131                         sign = 0;
132 
133                 CA_2(Mem_B[i], Mem_B_CA2, sign);
134                 for (j = 0; j < row_dim; j++)
135                     fprintf(Mem_B_data, "%d", Mem_B_CA2[j]);
136 
137                 fprintf(Mem_B_data, "\n");
138 
139                 fprintf(Mem_B_data_dec, "%d\n", Mem_B[i]);
140                 }
141             fclose(Mem_B_data);
142             fclose(Mem_B_data_dec);
143         }else
144             printf("Impossibile aprire il file 'Mem_A_out'");
145 
146         average_all = 0;            //calcolo della media
147         for(i=0; i<col_dim; i++)
148             average_all += Mem_A[i];
149 
150         average = average_all/col_dim;
151         if(average<0 && average_all%1024!=0)
152             average -= 1;
153         if(average > 0)
154             sign = 1;
155         else
156             sign = 0;
157         CA_2_18(average_all, average_CA2, sign);
158 
159         printf("\nI file 'MEM_A_data' e 'MEM_B_data' sono stati generati!\n");
160 
161         printf("\nLa media e': \n");
162         printf("\t- Decimale: %d", average);
163         printf("\n");
164         printf("\t- Media CA2: ");
165         for (j = 0; j < 8; j++)
166             printf("%d", average_CA2[j]);
167         printf("\n");
168 
169     }else{
170 
171         /* SECONDA MODALITA' SCELTA DA RIGA DI COMANDO*/
172         printf("MODALITA' 1: CONFRONTO RISULTATI\n\n");
173         printf("Inserire il nome del file 1: ");
174             scanf("%s", file_1);
175         printf("\nInserisci il nome del file 2: ");
176             scanf("%s", file_2);
177 
178         File_1 = fopen(file_1, "r");
179         if (!File_1) {
180             printf("Impossibile aprire il file %s!\n", file_1);
181             return EXIT_FAILURE;
182         }
183 
184         File_2 = fopen(file_2, "r");
185         if (!File_2) {
186             printf("Impossibile aprire il file %s!\n", file_2);
187             return EXIT_FAILURE;
188         }
189 
190         equal = 1;
191         for(i=0; i<col_dim && equal == 1; i++) {
192                 fscanf(File_1, "%x", &data_1);
```

```
193                 fscanf(File_2, "%x", &data_2);
194
195             if (data_1 != data_2)
196                 equal = 0; //   trovati dati diversi
197
198         }
199
200         if(equal == 0)
201             printf("\nI due file non coincidono!\n");
202         else
203             printf("\nCircuito funzionante, i due file coincidono!\n");
204     }
205
206     return 0;
207 }
208
209 void CA_2(int num, int *num_bin, int sign){
210     int i,j;
211     int neg;
212     if (num==0){
213         for(i=0; i< 8 ; i++)
214             num_bin[i] = 0;
215         return;
216     }
217
218     if(num<0 && num!=0)
219         num=-num;
220     for(i=(row_dim - 1); i>=0; i--){            // conversione dei numeri i binario
221         if(num%2 == 0)
222             num_bin[i] = 0;
223         else
224             num_bin[i] = 1;
225         num /= 2;
226     }
227
228     neg = 0;
229     if(sign == 0)                                          // ---------- CA2, se ...
        il numero    negativo
230         for(i=(row_dim - 1); i>=0 && neg == 0; i--) {
231             if (num_bin[i] == 1) {
232                 neg = 1;
233                 for (j = i - 1; j >= 0; j--)
234                     num_bin[j] = 1 - num_bin[j];
235
236             }
237         }
238
239     num_bin[0] = 1 - sign;  //si complementa il bit di segno
240 }
241
242 void CA_2_18(int num, int *num_bin, int sign){
243     int i,j;
244     int neg;
245
246     num = abs(num);
247     for(i=(18 - 1); i>=0; i--){            // conversione dei numeri i binario
248         if(num%2 == 0 || num == 0 )
249             num_bin[i] = 0;
250         else
251             num_bin[i] = 1;
252         num /= 2;
253     }
254
255     neg = 0;
256     if(sign == 0)                                          // ---------- CA2, se ...
        il numero    negativo
257         for(i=(18 - 1); i>=0 && neg == 0; i--) {
258             if (num_bin[i] == 1) {
259                 neg = 1;
260                 for (j = i - 1; j >= 0; j--)
261                     num_bin[j] = 1 - num_bin[j];
262
```

```
263                 }
264             }
265 
266         num_bin[0] = 1 - sign;  //si complementa il bit di segno
267     }
```

7.6 Simulazione bash

```
#!/bin/bash
# $file_bin_to_create    il file binario da creare
# $file_row    il numero di righe del file
# $n_bits    il numero di bit per riga
# $file_filtred    il file binario di uscita
file_bin_to_create=$1
file_row=$2
n_bits=$3
file_filtred=$4

# creo un file col prefisso uguale a quello passato solo con dec finale
file_bin_to_create_dec=$(echo "$file_bin_to_create" | cut -d '.' -f ...
    1)"_dec."$(echo "$file_bin_to_create" | cut -d '.' -f 2)
# creo un file col prefisso uguale a quello passato solo con dec finale
file_filtred_dec=$(echo "$file_filtred" | cut -d '.' -f 1)"_dec."$(echo ...
    "$file_filtred" | cut -d '.' -f 2)

if [ -f $file_bin_to_create ]
then
    rm $file_bin_to_create
fi

if [ -f $file_bin_to_create_dec ]
then
    rm $file_bin_to_create_dec
fi

if [ -f $file_filtred ]
then
    rm $file_filtred
fi

if [ -f $file_filtred_dec ]
then
    rm $file_filtred_dec
fi

for ((i=1; i<=$file_row; i++))
do
    num_bin=0
    for((j=1; j<=$n_bits; j++))
    do
        bit=$((RANDOM % 2))
        num_bin=$num_bin$bit
        echo -n "$bit" >> $file_bin_to_create
    done
    num_bin_dec=$((2#$num_bin))
    [ "$num_bin_dec" -gt 127 ] && ((num_bin_dec=$num_bin_dec-256))
    echo "$num_bin_dec" >> $file_bin_to_create_dec
    echo >> $file_bin_to_create
done

count=$(( 0 ))
uno=$(( 0 ))
due=$(( 0 ))
quattro=$(( 0 ))
tre=$(( 0 ))
media=$((0))

for binary in `cat $file_bin_to_create`
do
    # trasformo in decimale
    number=$((2#$binary))
    [ "$number" -gt 127 ] && ((number=$number-256))
    #echo "$number"

    media=$(($media + $number))

```

```
67      if [ $count -gt 3 ]
68      then
69          # shifto i valori
70          uno=$(($due))
71          due=$(($tre))
72          tre=$(($quattro))
73          quattro=$(($number))
74
75          #eseguo sempre la stessa operazione a regime
76          to_print=$(($quattro/4 + $tre - 2*$uno))
77
78      else
79          if [ $count -eq 0 ]
80          then
81              to_print=$(( $number/4))
82              uno=$(($number))
83          else
84              if [ $count -eq 1 ]
85              then
86                  to_print=$(( $number/4 + $uno))
87                  due=$(($number))
88              else
89                  if [ $count -eq 2 ]
90                  then
91                      to_print=$(( $number/4 + $due))
92                      tre=$(($number))
93                  else
94                      if [ $count -eq 3 ]
95                      then
96                          to_print=$(( $number/4 + $tre - 2*$uno))
97                          quattro=$(($number))
98                      fi
99                  fi
100             fi
101         fi
102     fi
103
104     echo "$to_print"
105     # se il primo valore    negativo e minore di zero
106     # se inoltre non    divisibile per 4
107     # allora devo sottrarci uno perch  la divisione per 4 in binario
108     # approssima sempre per difetto
109     if [ $number -lt 0 ] ; then
110         sup=$((-$number%4))
111         echo $sup
112         if [ $sup -ne 0 ] ; then
113             to_print=$(($to_print-1))
114         fi
115     fi
116     echo "$to_print"
117
118     rest=$(( $count%2 ))
119     if [ $rest -ne 0 ]; then
120         to_print=$(( $to_print*(-1) ))
121     fi
122
123     if [ $to_print -lt -128 ]; then
124         to_print=-128
125     fi
126     if [ $to_print -gt 127 ]; then
127         to_print=127
128     fi
129
130     echo "$to_print" >> $file_filtred_dec
131
132     # riporto il numero al valore positivo precedente se l'aveva
133     [ $to_print -lt 0 ] && to_print=$(( $to_print + 256 ))
134
135     # ritrasformo in binario
136     to_print_bin=$(echo "obase=2;$to_print" | bc)
137
138     if [ $to_print -lt 0 ]
```

```
139     then
140         # devo mettere tutti gli 8 valori
141         # i bit aggiunti devono essere tutti 1
142         while [ ${#to_print_bin} -lt 8 ]
143         do
144             to_print_bin="1"$to_print_bin
145         done
146     else
147         # devo mettere tutti gli 8 valori
148         # i bit aggiunti devono essere tutti 0
149         while [ ${#to_print_bin} -lt 8 ]; do
150             to_print_bin="0"$to_print_bin
151         done
152     fi
153
154     echo "$to_print_bin" >> $file_filtred
155
156     #incremento il contatore
157     count=$(($count+1))
158 done
159
160
161     echo "la la somma   : $media"
162     media=$(($media/1024)
163     echo "la media   : $media"
164     # riporto il numero al valore positivo precedente se l'aveva
165     [ $media -lt 0 ] && media=$(( $media + 256 ))
166
167     # ritrasformo in binario
168     media_bin=$(echo "obase=2;$media" | bc)
169     echo "la media   : $media_bin"
```

8 VHDL

8.1 Digital_filter.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Digital_Filter IS
6      PORT
7      (
8          Digital_Filter_START: IN STD_LOGIC;
9          Digital_Filter_CLOCK: IN STD_LOGIC;
10         Digital_filter_RST: IN STD_LOGIC;
11         Digital_Filter_Data_IN: IN SIGNED (7 DOWNTO 0);
12         Digital_Filter_Mem_B_Address : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
13         Digital_Filter_Data_OUT: OUT SIGNED (7 DOWNTO 0);
14         Digital_Filter_Average_OUT: OUT SIGNED (7 DOWNTO 0);
15         Digital_Filter_DONE: OUT STD_LOGIC
16     );
17 END Digital_Filter;
18
19 ARCHITECTURE Structural OF Digital_Filter IS
20
21     COMPONENT ASM_chart_Datapath
22         PORT
23         ( --- ingressi datapath
24                 CLK : IN STD_LOGIC;
25                 Data_IN : IN SIGNED(7 DOWNTO 0);
26                 MEM_B_ADDRESS_IN : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
27
28                 -- uscite datapath
29                 Data_OUT : OUT SIGNED(7 DOWNTO 0);
30                 Average : OUT SIGNED(7 DOWNTO 0);
31
32                 --- ingressi di controllo
33                 Counter_1024_EN, Counter_1024_RST : IN STD_LOGIC;
34                 Reg_counter_1_LOAD, Reg_counter_1_RST : IN STD_LOGIC;
35                 Reg_counter_2_LOAD, Reg_counter_2_RST : IN STD_LOGIC;
36                 Reg_counter_3_LOAD, Reg_counter_3_RST : IN STD_LOGIC;
37                 Mux_Address_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
38                 Mem_A_CS, Mem_A_WR_n, Mem_A_RD: IN STD_LOGIC;
39                 Reg_samples_Load, Reg_samples_RST : IN STD_LOGIC;
40                 Mux_operation_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
41                 Sample_CA_1 : IN STD_LOGIC;
42                 Block_Sample_n : IN STD_LOGIC;
43                 Sum_Filter_C_IN : IN STD_LOGIC;
44                 Reg_Filter_LOAD, Reg_Filter_RST : IN STD_LOGIC;
45                 Sum_CA_1 : IN STD_LOGIC;
46                 Mux_Saturation_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
47                 Mem_B_CS, Mem_B_WR_n, Mem_B_RD: IN STD_LOGIC;
48                 Reg_Sum_Average_LOAD, Reg_Sum_Average_RST : IN STD_LOGIC;
49                 Mux_MEM_B_sel : IN STD_LOGIC;
50
51                 --- uscite di stato
52                 Counter_1024_TC : OUT STD_LOGIC;
53                 Valid_Address_2, Valid_Address_3, Counter_LSB : OUT STD_LOGIC;
54                 Ovf_Sign_10_to_8_filter_OVF, Ovf_Sign_10_to_8_filter_SIGN: OUT STD_LOGIC
55         );
56     END COMPONENT;
57
58     -- Segnali per il portmap
59         SIGNAL      Counter_1024_EN_s, Counter_1024_RST_s: STD_LOGIC;
60         SIGNAL     Reg_counter_1_LOAD_s, Reg_counter_1_RST_s : STD_LOGIC;
61         SIGNAL      Reg_counter_2_LOAD_s, Reg_counter_2_RST_s : STD_LOGIC;
62         SIGNAL      Reg_counter_3_LOAD_s, Reg_counter_3_RST_s : STD_LOGIC;
63         SIGNAL      Mux_Address_SEL_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
64         SIGNAL      Mem_A_CS_s, Mem_A_WR_n_s, Mem_A_RD_s: STD_LOGIC;
65         SIGNAL      Reg_samples_Load_s, Reg_samples_RST_s : STD_LOGIC;
66         SIGNAL      Mux_operation_SEL_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
67         SIGNAL      Sample_CA_1_s : STD_LOGIC;
68         SIGNAL      Block_Sample_n_s : STD_LOGIC;
69         SIGNAL      Sum_Filter_C_IN_s : STD_LOGIC;
70         SIGNAL      Reg_Filter_LOAD_s, Reg_Filter_RST_s : STD_LOGIC;
```

```
71          SIGNAL      Sum_CA_1_s : STD_LOGIC;
72          SIGNAL      Mux_Saturation_SEL_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
73          SIGNAL      Mem_B_CS_s, Mem_B_WR_n_s, Mem_B_RD_s: STD_LOGIC;
74          SIGNAL      Reg_Sum_Average_LOAD_s, Reg_Sum_Average_RST_s: STD_LOGIC;
75          SIGNAL      Mux_MEM_B_sel_s : STD_LOGIC;
76
77          -- Uscite di stato
78          SIGNAL      Counter_1024_TC_s : STD_LOGIC;
79          SIGNAL      Valid_Address_2_s, Valid_Address_3_s, Counter_LSB_s : STD_LOGIC;
80          SIGNAL      Ovf_Sign_10_to_8_filter_OVF_s, Ovf_Sign_10_to_8_filter_SIGN_s: STD_LOGIC;
81
82      type states IS (RST, S0_LOAD_MEM_A, S1_LOAD_DATA_1, S2_EXECUTE_DATA_1, S3_EXECUTE_DATA_2,
83                           S2_INVALID_2, S3_INVALID_3, S4_EXECUTE_DATA_3, S5_INVERT, S6_WRITE_SUM,
84                           S6_WRITE_127, S6_WRITE_128, S7_DONE);
85
86      SIGNAL present_state: states;
87      SIGNAL next_state: states;
88
89
90  BEGIN
91
92      Datapath_1 : ASM_chart_Datapath PORT MAP
93      (
94          CLK => Digital_Filter_CLOCK,
95          Data_IN => Digital_Filter_Data_IN,
96          MEM_B_ADDRESS_IN => Digital_Filter_Mem_B_Address,
97
98          -- uscite datapath
99          Data_OUT => Digital_Filter_Data_OUT,
100         Average => Digital_Filter_Average_OUT,
101
102         --- ingressi di controllo
103         Counter_1024_EN => Counter_1024_EN_s,
104         Counter_1024_RST => Counter_1024_RST_s,
105         Reg_counter_1_LOAD => Reg_counter_1_LOAD_s,
106         Reg_counter_1_RST => Reg_counter_1_RST_s,
107         Reg_counter_2_LOAD => Reg_counter_2_LOAD_s,
108         Reg_counter_2_RST => Reg_counter_2_RST_s,
109         Reg_counter_3_LOAD => Reg_counter_3_LOAD_s,
110         Reg_counter_3_RST => Reg_counter_3_RST_s,
111         Mux_Address_SEL => Mux_Address_SEL_s,
112         Mem_A_CS => Mem_A_CS_s,
113         Mem_A_WR_n => Mem_A_WR_n_s,
114         Mem_A_RD => Mem_A_RD_s,
115         Reg_samples_Load => Reg_samples_Load_s,
116         Reg_samples_RST => Reg_samples_RST_s,
117         Mux_operation_SEL => Mux_operation_SEL_s,
118         Sample_CA_1 => Sample_CA_1_s,
119         Block_Sample_n => Block_Sample_n_s,
120         Sum_Filter_C_IN => Sum_Filter_C_IN_s,
121         Reg_Filter_LOAD => Reg_Filter_LOAD_s,
122         Reg_Filter_RST => Reg_Filter_RST_s,
123         Sum_CA_1 => Sum_CA_1_s,
124         Mux_Saturation_SEL => Mux_Saturation_SEL_s,
125         Mem_B_CS => Mem_B_CS_s,
126         Mem_B_WR_n => Mem_B_WR_n_s,
127         Mem_B_RD => Mem_B_RD_s,
128         Reg_Sum_Average_LOAD => Reg_Sum_Average_LOAD_s,
129         Reg_Sum_Average_RST => Reg_Sum_Average_RST_s,
130         Mux_MEM_B_sel => Mux_MEM_B_sel_s,
131
132
133         --- uscite di stato
134         Counter_1024_TC => Counter_1024_TC_s,
135         Valid_Address_2 => Valid_Address_2_s,
136         Valid_Address_3 => Valid_Address_3_s,
137         Counter_LSB => Counter_LSB_s,
138         Ovf_Sign_10_to_8_filter_OVF => Ovf_Sign_10_to_8_filter_OVF_s,
139         Ovf_Sign_10_to_8_filter_SIGN => Ovf_Sign_10_to_8_filter_SIGN_s
140     );
141
142
143         State_registers:
144         PROCESS(Digital_Filter_Clock)
145             BEGIN
```

```
146                 if Digital_Filter_Clock'event and Digital_Filter_Clock = '1' then -- Le ...
                        operazioni vengono fatte solo sul fronte alto di salita del clock
147                     IF Digital_Filter_RST = '1' THEN
148                         present_state <= RST;
149                     ELSE
150                         present_state <= next_state;
151                     END IF;
152                 end if;
153         END PROCESS;
154
155
156         State_transitions:
157         PROCESS( present_state,
158                  Digital_Filter_START,
159                  Counter_1024_TC_s,
160                  Valid_Address_2_s,
161                  Valid_Address_3_s,
162                  Counter_LSB_s,
163                  Ovf_Sign_10_to_8_filter_OVF_s,
164                  Ovf_Sign_10_to_8_filter_SIGN_s
165                  )
166         BEGIN
167             next_state <= RST;
168             CASE present_state IS
169
170                 WHEN RST =>
171                     IF (Digital_Filter_START = '1') THEN
172                         next_state <= S0_LOAD_MEM_A;
173                     END IF;
174
175                 WHEN S0_LOAD_MEM_A =>
176                     IF (Counter_1024_TC_s = '1') THEN
177                         next_state <= S1_LOAD_DATA_1;
178                     ELSE
179                         next_state <= S0_LOAD_MEM_A;
180                     END IF;
181
182                 WHEN S1_LOAD_DATA_1 =>
183                     IF (Valid_Address_2_s = '1') THEN
184                         next_state <= S2_EXECUTE_DATA_1;
185                     ELSE
186                         next_state <= S2_INVALID_2;
187                     END IF;
188
189                 WHEN S2_EXECUTE_DATA_1 =>
190                     IF (Valid_Address_3_s = '1') THEN
191                         next_state <= S3_EXECUTE_DATA_2;
192                     ELSE
193                         next_state <= S3_INVALID_3;
194                     END IF;
195
196                 WHEN S3_EXECUTE_DATA_2 =>
197                         next_state <= S4_EXECUTE_DATA_3;
198
199                 WHEN S4_EXECUTE_DATA_3 | S2_INVALID_2 | S3_INVALID_3=>
200                     IF (Counter_LSB_s = '1') THEN
201                         next_state <= S5_INVERT;
202                     ELSE
203                         IF (Ovf_Sign_10_to_8_filter_OVF_s = '1') THEN
204                             IF (Ovf_Sign_10_to_8_filter_SIGN_s = '1') THEN -- Il segno  uno ...
                                    se negativo
205                                 next_state <= S6_WRITE_128;
206                             ELSE
207                                 next_state <= S6_WRITE_127;
208                             END IF;
209                         ELSE
210                             next_state <= S6_WRITE_SUM;
211                         END IF;
212                     END IF;
213
214                 WHEN S5_INVERT =>
215                     IF (Ovf_Sign_10_to_8_filter_OVF_s = '1') THEN
216                         IF (Ovf_Sign_10_to_8_filter_SIGN_s = '1') THEN -- Il segno  uno se ...
                                negativo
217                             next_state <= S6_WRITE_128;
218                         ELSE
```

```
219                             next_state <= S6_WRITE_127;
220                         END IF;
221                     ELSE
222                         next_state <= S6_WRITE_SUM;
223                     END IF;
224
225                 WHEN S6_WRITE_SUM | S6_WRITE_127 | S6_WRITE_128 =>
226                     IF (Counter_1024_TC_s = '1') THEN
227                         next_state <= S7_DONE;
228                     ELSE
229                         next_state <= S1_LOAD_DATA_1;
230                     END IF;
231
232                 WHEN S7_DONE =>
233                     IF (Digital_Filter_START = '0') THEN -- Altrimenti RESET
234                         next_state <= S7_DONE;
235                     ELSE
236                         next_state <= RST;
237                     END IF;
238                 WHEN OTHERS => next_state <= RST;
239
240             END CASE;
241         END PROCESS;
242
243         Output_decode :
244         PROCESS(present_state)
245         BEGIN
246         -- Valori di default
247                     Counter_1024_EN_s <= '0';
248                     Counter_1024_RST_s <= '0';
249                     Reg_counter_1_LOAD_s <= '0';
250                     Reg_counter_2_LOAD_s <= '0';
251                     Reg_counter_3_LOAD_s <= '0';
252                     Reg_counter_1_RST_s <= '0';
253                     Reg_counter_2_RST_s <= '0';
254                     Reg_Counter_3_RST_s <= '0';
255                     Mux_Address_SEL_s <= "00";
256                     Mem_A_CS_s <= '1';
257                     Mem_A_WR_n_s <= '1';
258                     Mem_A_RD_s <= '1';
259                     Reg_Samples_Load_s <= '1';
260                     Reg_Samples_RST_s <= '0';
261                     Mux_Operation_SEL_s <= "00";
262                     Sample_CA_1_s <= '0';
263                     Block_Sample_n_s <= '1';
264                     Sum_Filter_C_IN_s <= '0';
265                     Reg_Filter_LOAD_s <= '1';
266                     Reg_Filter_RST_s <= '0';
267                     Sum_CA_1_s <= '0';
268                     Mux_Saturation_SEL_s <= "00";
269                     Mem_B_CS_s <= '1';
270                     Mem_B_WR_n_s <= '1';
271                     Mem_B_RD_s <= '0';
272                     Reg_Sum_Average_LOAD_s <= '0';
273                     Reg_Sum_Average_RST_s   <= '0';
274                     DIGITAL_FILTER_DONE <= '0';
275                     Mux_MEM_B_sel_s <= '0';
276
277             CASE present_state IS
278                 WHEN RST =>
279                     Counter_1024_RST_s <= '1';
280                     Reg_counter_1_RST_s <= '1';
281                     Reg_counter_2_RST_s <= '1';
282                     Reg_counter_3_RST_s <= '1';
283                     Reg_samples_RST_s <= '1';
284                     Reg_Filter_RST_s <= '1';
285                     Reg_Sum_Average_RST_s   <= '1';
286
287                 WHEN S0_LOAD_MEM_A => Counter_1024_EN_s <= '1';
288                                       Mem_A_WR_n_s <= '0';
289                                       Mem_A_RD_s <= '0';
290
291                 WHEN S1_LOAD_DATA_1 => Reg_Filter_RST_s <= '1';
292
293                 WHEN S2_EXECUTE_DATA_1 => Mux_Address_SEL_s <= "01";
294                                           Reg_Sum_Average_LOAD_s <= '1';
```

```
295
296                 WHEN S3_EXECUTE_DATA_2 => Mux_Operation_SEL_s <= "01";
297                                           Mux_Address_SEL_s <= "10";
298
299                 WHEN S2_INVALID_2 => Reg_Sum_Average_LOAD_s <= '1';
300
301                 WHEN S3_INVALID_3 => Mux_Operation_SEL_s <= "01";
302
303                 WHEN S4_EXECUTE_DATA_3 => Reg_Samples_LOAD_s <= '0';
304                                           Mux_Operation_SEL_s <= "10";
305                                           Sample_CA_1_s <= '1';
306                                           Sum_Filter_C_IN_s <= '1';
307
308                 WHEN S5_INVERT => Block_Sample_n_s <= '0';
309                                   Sum_CA_1_s <= '1';
310                                   Sum_Filter_C_IN_s <= '1';
311
312                 WHEN S6_WRITE_SUM => Mem_B_CS_s <= '1';
313                                      Mem_B_WR_n_s <= '0';
314                                      Counter_1024_EN_s <= '1';
315                                      Reg_counter_1_LOAD_s <= '1';
316                                      Reg_counter_2_LOAD_s <= '1';
317                                      Reg_counter_3_LOAD_s <= '1';
318
319                 WHEN S6_WRITE_127 => Mem_B_CS_s <= '1';
320                                      Mem_B_WR_n_s <= '0';
321                                      Counter_1024_EN_s <= '1';
322                                      Reg_counter_1_LOAD_s <= '1';
323                                      Reg_counter_2_LOAD_s <= '1';
324                                      Reg_counter_3_LOAD_s <= '1';
325                                      Mux_Saturation_SEL_s <= "10";
326
327                 WHEN S6_WRITE_128 => Mem_B_CS_s <= '1';
328                                      Mem_B_WR_n_s <= '0';
329                                      Counter_1024_EN_s <= '1';
330                                      Reg_counter_1_LOAD_s <= '1';
331                                      Reg_counter_2_LOAD_s <= '1';
332                                      Reg_counter_3_LOAD_s <= '1';
333                                      Mux_Saturation_SEL_s <= "01";
334
335                 WHEN S7_DONE => DIGITAL_FILTER_DONE <= '1';
336                                 Mux_MEM_B_sel_s <= '1';
337                                 Mem_B_RD_s <= '1';
338                                 Mem_B_CS_s <= '1';
339
340
341             END CASE;
342         END PROCESS;
343
344 END Structural;
```

8.2 ASM_chart_Datapath.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY ASM_chart_Datapath IS
6      PORT( --- ingressi datapath
7              CLK : IN STD_LOGIC;
8              MEM_B_ADDRESS_IN : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
9              Data_IN : IN SIGNED(7 DOWNTO 0);
10
11             -- uscite datapath
12             Data_OUT : OUT SIGNED(7 DOWNTO 0);
13             Average : OUT SIGNED(7 DOWNTO 0);
14
15             --- ingressi di controllo
16             Counter_1024_EN, Counter_1024_RST : IN STD_LOGIC;
17             Reg_counter_1_LOAD, Reg_counter_1_RST : IN STD_LOGIC;
18             Reg_counter_2_LOAD, Reg_counter_2_RST : IN STD_LOGIC;
19             Reg_counter_3_LOAD, Reg_counter_3_RST : IN STD_LOGIC;
20             Mux_Address_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
```

```
21              Mem_A_CS, Mem_A_WR_n, Mem_A_RD: IN STD_LOGIC;
22              Reg_samples_Load, Reg_samples_RST : IN STD_LOGIC;
23              Mux_operation_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
24              Sample_CA_1 : IN STD_LOGIC;
25              Block_Sample_n : IN STD_LOGIC;
26              Sum_Filter_C_IN : IN STD_LOGIC;
27              Reg_Filter_LOAD, Reg_Filter_RST : IN STD_LOGIC;
28              Sum_CA_1 : IN STD_LOGIC;
29              Mux_Saturation_SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
30              Mem_B_CS, Mem_B_WR_n, Mem_B_RD: IN STD_LOGIC;
31              Reg_Sum_Average_LOAD, Reg_Sum_Average_RST : IN STD_LOGIC;
32              Mux_MEM_B_sel : IN STD_LOGIC;
33
34              --- uscite di stato
35              Counter_1024_TC : OUT STD_LOGIC;
36              Valid_Address_2, Valid_Address_3, Counter_LSB : OUT STD_LOGIC;
37              Ovf_Sign_10_to_8_filter_OVF, Ovf_Sign_10_to_8_filter_SIGN: OUT STD_LOGIC
38          );
39      END ASM_chart_Datapath;
40
41  ARCHITECTURE Structural OF ASM_chart_Datapath IS
42
43      COMPONENT Counter_1024
44          PORT
45          (  Counter_1024_EN : IN std_logic; -- Segnale di Enable
46              Counter_1024_RST : IN std_logic; -- Segnale di Reset
47              Counter_1024_CLK : IN std_logic; -- Segnale di Clock
48              Counter_1024_TC : OUT std_logic; -- Segnale di fine conta
49              Counter_1024_OUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
50          );
51      END COMPONENT;
52
53      COMPONENT Valid_address
54          PORT ( Valid_address_in : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
55                   Valid_address_2, Valid_address_3 : OUT STD_LOGIC;
56                   Valid_address_counter_LSB : OUT STD_LOGIC
57                 );
58      END COMPONENT;
59
60
61
62      COMPONENT Three_operation_block
63          PORT ( Three_operation_block_IN : IN SIGNED(7 DOWNTO 0);
64                   Three_operation_block_DIV4,
65                   Three_operation_block_MULT2,
66                   Three_operation_block_transparent : OUT SIGNED(9 DOWNTO 0)
67                 );
68      END COMPONENT;
69
70      COMPONENT Register_n
71          GENERIC (N : integer := 8);
72          PORT (
73              Register_D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0); -- Segnale che viene riportato in ...
                    uscita al colpo di clock successivo
74              Register_Clock, Register_Reset, Register_LOAD : IN std_logic;
75              Register_Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) -- Segnale di uscita
76          );
77      END COMPONENT;
78
79      COMPONENT Register_n_signed
80          GENERIC (N : integer := 8);
81          PORT (
82              Register_D : IN SIGNED(N-1 DOWNTO 0); -- Segnale che viene riportato in uscita al ...
                    colpo di clock successivo
83              Register_Clock, Register_Reset, Register_LOAD : IN std_logic;
84              Register_Q : OUT SIGNED(N-1 DOWNTO 0) -- Segnale di uscita
85          );
86      END COMPONENT;
87
88
89      COMPONENT RCA_18_bit
90      GENERIC (N : integer := 18);
91          PORT ( RCA_18_bit_in_1, RCA_18_bit_in_2: IN SIGNED(N-1 DOWNTO 0); -- Ingressi su N bit
92                   RCA_18_bit_c_in: IN STD_LOGIC; -- Carry_in da sommare al resto
93                   RCA_18_bit_out: OUT SIGNED(N-1 DOWNTO 0); -- Risultato della somma su N bit
```

```
94                  RCA_18_bit_c_out, RCA_18_bit_overflow: BUFFER STD_LOGIC -- Segnali di carry ...
                        out e overflow (Buffer poich uno serve per l'altro)
95          );
96      END COMPONENT;
97
98      COMPONENT RCA_10_bit IS
99      GENERIC (N : integer := 10);
100         PORT ( RCA_10_bit_in_1, RCA_10_bit_in_2: IN SIGNED(N-1 DOWNTO 0); -- Ingressi su N bit
101                 RCA_10_bit_c_in: IN STD_LOGIC; -- Carry_in da sommare al resto
102                 RCA_10_bit_out: OUT SIGNED(N-1 DOWNTO 0); -- Risultato della somma su N bit
103                 RCA_10_bit_c_out, RCA_10_bit_overflow: BUFFER STD_LOGIC -- Segnali di carry ...
                        out e overflow (Buffer poich uno serve per l'altro)
104         );
105     END COMPONENT;
106
107     COMPONENT Mux_3_to_1_10bit
108         PORT( Mux_3_to_1_10bit_IN_0, Mux_3_to_1_10bit_IN_1, Mux_3_to_1_10bit_IN_2 : IN ...
                STD_LOGIC_VECTOR (9 DOWNTO 0);
109                 Mux_3_to_1_10bit_SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
110                 Mux_3_to_1_10bit_OUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
111             );
112     END COMPONENT;
113
114     COMPONENT Mux_3_to_1_10bit_signed
115         PORT( Mux_3_to_1_10bit_IN_0, Mux_3_to_1_10bit_IN_1, Mux_3_to_1_10bit_IN_2 : IN SIGNED ...
                (9 DOWNTO 0);
116                 Mux_3_to_1_10bit_SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
117                 Mux_3_to_1_10bit_OUT : OUT SIGNED (9 DOWNTO 0)
118             );
119     END COMPONENT;
120
121     COMPONENT Ovf_Sign_10_to_8_filter
122         PORT ( Ovf_Sign_10_to_8_filter_IN : IN SIGNED(9 DOWNTO 0);
123                  Ovf_Sign_10_to_8_filter_OVF, Ovf_Sign_10_to_8_filter_SIGN : OUT STD_LOGIC
124                 );
125     END COMPONENT;
126
127     COMPONENT Cut_10_to_8_filter
128         PORT( Cut_10_to_8_filter_IN : IN SIGNED(9 DOWNTO 0);
129                 Cut_10_to_8_filter_OUT : OUT SIGNED(7 DOWNTO 0)
130               );
131     END COMPONENT;
132
133     COMPONENT Average_8_to_18
134         PORT( Average_8_to_18_IN : IN SIGNED(7 DOWNTO 0);
135                 Average_8_to_18_OUT : OUT SIGNED(17 DOWNTO 0)
136               );
137     END COMPONENT;
138
139     COMPONENT Sync_RAM
140         GENERIC( M : NATURAL  := 3;
141                  N : NATURAL := 1024
142                );
143         PORT( Sync_RAM_Address : IN STD_LOGIC_VECTOR (M-1 DOWNTO 0);
144                 Sync_RAM_Data : IN SIGNED (N-1 DOWNTO 0);
145                 Sync_RAM_WR_n, Sync_RAM_Clk, Sync_RAM_RD, Sync_RAM_CS: IN STD_LOGIC;
146                 Sync_RAM_Q_out : OUT SIGNED (N-1 DOWNTO 0)
147               );
148     END COMPONENT;
149
150     COMPONENT  Mux_2_to_1_10bit IS
151         PORT( Mux_2_to_1_10bit_IN_0, Mux_2_to_1_10bit_IN_1: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
152                 Mux_2_to_1_10bit_SEL : IN STD_LOGIC;
153                 Mux_2_to_1_10bit_OUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
154             );
155     END COMPONENT;
156
157 SIGNAL Counter_1024_OUT_s,
158          Reg_Counter_1_OUT_s,
159          Reg_Counter_2_OUT_s,
160          Reg_Counter_3_OUT_s,
161          Mux_MEM_B_OUT_s,
162          Mux_Address_OUT_s: STD_LOGIC_VECTOR(9 DOWNTO 0);
163
164 SIGNAL Mem_A_Data_OUT_s,
165          Reg_Samples_OUT_s,
```

```
166           Average_s: SIGNED(7 DOWNTO 0);
167
168   SIGNAL Three_Operation_Block_DIV4_s,
169           Three_Operation_Block_MULT2_s,
170           Three_Operation_Block_Transparent_s,
171           Mux_Operation_OUT_s,
172           Sample_CA1_OUT_s,
173           Block_Sample_n_OUT_s,
174           Sum_Filter_OUT_s,
175           Reg_Filter_OUT_s,
176           Sum_CA1_OUT_s,
177           Mux_Saturation_OUT_s : SIGNED(9 DOWNTO 0);
178
179   SIGNAL Cut_10_to_8_Filter_OUT_s : SIGNED(7 DOWNTO 0);
180
181   SIGNAL Average_8_to_18_OUT_s,
182           Sum_Average_OUT_s,
183           Reg_Sum_Average_OUT_s: SIGNED(17 DOWNTO 0);
184
185   BEGIN
186
187       Counter : Counter_1024 PORT MAP ( Counter_1024_EN => Counter_1024_EN,
188                                         Counter_1024_RST => Counter_1024_RST,
189                                         Counter_1024_CLK => CLK,
190                                         Counter_1024_TC => Counter_1024_TC,
191                                         Counter_1024_OUT => Counter_1024_OUT_s
192                                        );
193
194       Reg_Counter_1 : Register_n GENERIC MAP ( N => 10 )
195                                  PORT MAP ( Register_D => Counter_1024_OUT_s,
196                                             Register_Clock => CLK,
197                                             Register_Reset => Reg_Counter_1_RST,
198                                             Register_LOAD => Reg_Counter_1_LOAD,
199                                             Register_Q => Reg_Counter_1_OUT_s
200                                            );
201
202       Reg_Counter_2 : Register_n GENERIC MAP ( N => 10 )
203                                  PORT MAP ( Register_D => Reg_Counter_1_OUT_s,
204                                             Register_Clock => CLK,
205                                             Register_Reset => Reg_Counter_2_RST,
206                                             Register_LOAD => Reg_Counter_2_LOAD,
207                                             Register_Q => Reg_Counter_2_OUT_s
208                                            );
209
210       Reg_Counter_3 : Register_n GENERIC MAP ( N => 10 )
211                                  PORT MAP ( Register_D => Reg_Counter_2_OUT_s,
212                                             Register_Clock => CLK,
213                                             Register_Reset => Reg_Counter_3_RST,
214                                             Register_LOAD => Reg_Counter_3_LOAD,
215                                             Register_Q => Reg_Counter_3_OUT_s
216                                            );
217
218       Mux_Address : Mux_3_to_1_10bit PORT MAP ( Mux_3_to_1_10bit_IN_0 => Counter_1024_OUT_s,
219                                                 Mux_3_to_1_10bit_IN_1 => Reg_Counter_1_OUT_s,
220                                                 Mux_3_to_1_10bit_IN_2 => Reg_Counter_3_OUT_s,
221                                                 Mux_3_to_1_10bit_SEL => Mux_Address_SEL,
222                                                 Mux_3_to_1_10bit_OUT => Mux_Address_OUT_s
223                                               );
224
225       Valid_Add : Valid_Address PORT MAP ( Valid_address_in => Counter_1024_OUT_s,
226                                            Valid_address_2 => Valid_Address_2,
227                                            Valid_address_3 => Valid_Address_3,
228                                            Valid_address_counter_LSB => Counter_LSB
229                                  );
230
231       Mem_A : Sync_RAM GENERIC MAP ( M => 10, N => 8 )
232                           PORT MAP ( Sync_RAM_Address  => Mux_Address_OUT_s,
233                                      Sync_RAM_Data => Data_IN,
234                                      Sync_RAM_WR_n => Mem_A_WR_n,
235                                      Sync_RAM_Clk => CLK,
236                                      Sync_RAM_RD => Mem_A_RD,
237                                      Sync_RAM_CS => Mem_A_CS,
238                                      Sync_RAM_Q_out => Mem_A_Data_OUT_s
239                                     );
240
241       Reg_sample : Register_n_signed GENERIC MAP ( N => 8 )
```

```
                                    PORT MAP ( Register_D => Mem_A_Data_OUT_s,
                                               Register_Clock => CLK,
                                               Register_Reset => Reg_Samples_RST,
                                               Register_LOAD => Reg_Samples_LOAD,
                                               Register_Q => Reg_Samples_OUT_s
                                              );

    Three_Operations_block : Three_Operation_block
       PORT MAP ( Three_operation_block_IN => Reg_samples_OUT_s,
                  Three_operation_block_DIV4 => Three_Operation_Block_DIV4_s,
                  Three_operation_block_MULT2 => Three_Operation_Block_MULT2_s,
                  Three_operation_block_transparent  => Three_Operation_Block_Transparent_s
                );

    Mux_operation : Mux_3_to_1_10bit_signed
            PORT MAP( Mux_3_to_1_10bit_IN_0 => Three_Operation_Block_DIV4_s,
                      Mux_3_to_1_10bit_IN_1 => Three_Operation_Block_Transparent_s,
                      Mux_3_to_1_10bit_IN_2 => Three_Operation_Block_MULT2_s,
                      Mux_3_to_1_10bit_SEL => Mux_Operation_SEL,
                      Mux_3_to_1_10bit_OUT =>   Mux_Operation_OUT_s
                    );

    --sample CA1
    Sample_CA1_OUT_s <= (Mux_Operation_OUT_s(9) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(8) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(7) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(6) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(5) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(4) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(3) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(2) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(1) XOR Sample_CA_1) &
                        (Mux_Operation_OUT_s(0) XOR Sample_CA_1);

    --block sample
    Block_Sample_n_OUT_s <= (Sample_CA1_OUT_s(9) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(8) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(7) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(6) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(5) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(4) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(3) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(2) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(1) AND Block_Sample_n) &
                            (Sample_CA1_OUT_s(0) AND Block_Sample_n);

    Sum_Filter : RCA_10_bit PORT MAP ( RCA_10_bit_in_1 => Block_Sample_n_OUT_s,
                                       RCA_10_bit_in_2 => Sum_CA1_OUT_s,
                                       RCA_10_bit_c_in => Sum_Filter_C_IN,
                                       RCA_10_bit_out => Sum_Filter_OUT_s
                                      );

    Reg_Filter : Register_n_signed GENERIC MAP ( N => 10 )
                                    PORT MAP ( Register_D => Sum_Filter_OUT_s,
                                               Register_Clock => CLK,
                                               Register_Reset => Reg_Filter_RST,
                                               Register_LOAD => Reg_Filter_LOAD,
                                               Register_Q => Reg_Filter_OUT_s
                                              );

    Ovf_sign : Ovf_Sign_10_to_8_Filter
                    PORT MAP(Ovf_Sign_10_to_8_Filter_IN => Sum_Filter_OUT_s,
                             Ovf_Sign_10_to_8_Filter_OVF => Ovf_Sign_10_to_8_filter_OVF,
                             Ovf_Sign_10_to_8_Filter_SIGN => Ovf_Sign_10_to_8_filter_SIGN
                            );

    Sum_CA1_OUT_s <= (Reg_Filter_OUT_s(9) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(8) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(7) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(6) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(5) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(4) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(3) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(2) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(1) XOR Sum_CA_1) &
                     (Reg_Filter_OUT_s(0) XOR Sum_CA_1);
```

```
318
319     Mux_Saturation : Mux_3_to_1_10bit_signed
320                         PORT MAP ( Mux_3_to_1_10bit_IN_0 => Reg_Filter_OUT_s,
321                                    Mux_3_to_1_10bit_IN_1 => "1110000000", -- -128
322                                    Mux_3_to_1_10bit_IN_2 => "0001111111", -- 127
323                                    Mux_3_to_1_10bit_SEL => Mux_Saturation_SEL,
324                                    Mux_3_to_1_10bit_OUT => Mux_Saturation_OUT_s
325                                  );
326
327     Cut_10_to_8 : Cut_10_to_8_Filter PORT MAP ( Cut_10_to_8_filter_IN => Mux_Saturation_OUT_s,
328                                                 Cut_10_to_8_filter_OUT => Cut_10_to_8_filter_OUT_s
329                                               );
330
331     Mem_B : Sync_RAM GENERIC MAP ( M => 10, N => 8 )
332                              PORT MAP ( Sync_RAM_Address => Mux_MEM_B_OUT_s,
333                                         Sync_RAM_Data => Cut_10_to_8_filter_OUT_s,
334                                        Sync_RAM_WR_n => Mem_B_WR_n,
335                                         Sync_RAM_Clk => CLK,
336                                         Sync_RAM_RD => Mem_B_RD,
337                                         Sync_RAM_CS => Mem_B_CS,
338                                         Sync_RAM_Q_out => Data_OUT
339                                        );
340
341     Mux_MEM_B : Mux_2_to_1_10bit PORT MAP( Mux_2_to_1_10bit_IN_0 => Counter_1024_OUT_s,
342                                            Mux_2_to_1_10bit_IN_1 => MEM_B_ADDRESS_IN,
343                                            Mux_2_to_1_10bit_SEL => Mux_MEM_B_sel,
344                                            Mux_2_to_1_10bit_OUT => Mux_MEM_B_OUT_s
345                                         );
346
347     Average_8_to_18_block : Average_8_to_18
348                               PORT MAP ( Average_8_to_18_IN => Reg_Samples_OUT_s,
349                                          Average_8_to_18_OUT => Average_8_to_18_OUT_s
350                                        );
351
352     Sum_Average : RCA_18_bit PORT MAP ( RCA_18_bit_in_1 => Average_8_to_18_OUT_s,
353                                                            RCA_18_bit_in_2 => Reg_Sum_Average_OUT_s,
354                                                            RCA_18_bit_c_in => '0',
355                                                            RCA_18_bit_out => Sum_Average_OUT_s
356                                                          );
357
358     Reg_Sum_Average : Register_n_signed GENERIC MAP ( N => 18 )
359                                         PORT MAP ( Register_D => Sum_Average_OUT_s,
360                                                    Register_Clock => CLK,
361                                                   Register_Reset => Reg_Sum_Average_RST,
362                                                    Register_LOAD => Reg_Sum_Average_LOAD,
363                                                    Register_Q => Reg_Sum_Average_OUT_s
364                                                   );
365
366     Average_s <= Reg_Sum_Average_OUT_s(17 DOWNTO 10);
367     Average <= Average_s;
368 END Structural;
```

8.3 Sync_RAM.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Sync_RAM IS
6      GENERIC( M : NATURAL  := 10;
7               N : NATURAL := 8
8             );
9      PORT( Sync_RAM_Address : IN STD_LOGIC_VECTOR (M-1 DOWNTO 0);
10           Sync_RAM_Data : IN SIGNED (N-1 DOWNTO 0);
11           Sync_RAM_WR_n, Sync_RAM_Clk, Sync_RAM_RD, Sync_RAM_CS: IN STD_LOGIC;
12           Sync_RAM_Q_out : OUT SIGNED (N-1 DOWNTO 0)
13         );
14 END Sync_RAM;
15
16 ARCHITECTURE RTL OF Sync_RAM IS
17
18     TYPE RAM_Array IS ARRAY (0 TO (2**M - 1)) OF SIGNED(N-1 DOWNTO 0);
19     SIGNAL Mem : RAM_Array;
```

```
20
21  BEGIN
22
23      P0 : PROCESS(Sync_RAM_clk) IS
24          BEGIN
25              IF Sync_RAM_clk = '1' AND Sync_RAM_clk'event THEN
26                  IF Sync_RAM_CS = '1' THEN
27                      IF Sync_RAM_WR_n = '0' THEN
28                          Mem(to_integer(unsigned(Sync_RAM_Address))) <= Sync_RAM_Data;
29                      END IF;
30                  END IF;
31              END IF;
32          IF Sync_RAM_RD = '1' THEN
33              Sync_RAM_Q_out <= Mem(to_integer(unsigned(Sync_RAM_Address)));
34          ELSE
35              Sync_RAM_Q_out <= (OTHERS => '0');
36          END IF;
37      END PROCESS;
38
39  END RTL;
```

8.4 Register_n.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   -- Regustri a B but
5   ENTITY Register_n IS
6       GENERIC (N : integer := 8);
7       PORT (
8           Register_D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0); -- Segnale che viene riportato in ...
                uscita al colpo di clock successivo
9           Register_Clock, Register_Reset, Register_LOAD: IN std_logic;
10          Register_Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) -- Segnale di uscita
11      );
12  END Register_n;
13
14  ARCHITECTURE Behavior OF Register_n IS
15  BEGIN
16      PROCESS(Register_Clock)
17      BEGIN
18          IF (Register_Clock'EVENT AND Register_Clock = '1') THEN
19              IF (Register_Reset = '1') THEN
20                  Register_Q <= (OTHERS => '0');
21              ELSIF Register_LOAD = '1' THEN
22                  Register_Q <= Register_D;
23              END IF;
24          END IF;
25      END PROCESS;
26  END Behavior;
```

8.5 RCA_10_bit.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   -- Sommatore su 8 bit
6   ENTITY RCA_10_bit IS
7       GENERIC (N : integer := 10);
8       PORT (
9           RCA_10_bit_IN_1, RCA_10_bit_IN_2: IN SIGNED(N-1 DOWNTO 0); -- Ingressi su N bit
10          RCA_10_bit_C_IN: IN STD_LOGIC; -- Carry_in da sommare al resto
11          RCA_10_bit_OUT: OUT SIGNED(N-1 DOWNTO 0); -- Risultato della somma su N bit
12          RCA_10_bit_C_OUT, RCA_10_bit_overflow: BUFFER STD_LOGIC -- Segnali di carry out e ...
                overflow (Buffer poich uno serve per l'altro)
13      );
14  END RCA_10_bit;
15
```

```
16 ARCHITECTURE Structure OF RCA_10_bit IS
17
18     COMPONENT FA
19         PORT (
20             x, y, c_in : IN STD_LOGIC; -- x e y: bit da sommare, c_in: carry in
21             sum, c_out : OUT STD_LOGIC -- sum: somma di x e y, c_out: carry out
22         );
23     END COMPONENT;
24
25     -- Segnale intermedio per il PORT MAP
26     SIGNAL RCA_10_bit_c_ripple: SIGNED(N-2 DOWNTO 0); -- 'carry ripple': carry di supporto ...
           all'interno del RCA
27
28 BEGIN
29
30     -- Primo full adder con il carry_in
31     FA_0 : FA PORT MAP (
32                               x => RCA_10_bit_in_1(0),
33                               y => RCA_10_bit_in_2(0),
34                               c_in => RCA_10_bit_c_in,
35                               sum => RCA_10_bit_out(0),
36                               c_out => RCA_10_bit_c_ripple(0)
37                 );
38
39     -- Full adder in mezzo
40     FA_v : for I in 1 to N-2 generate
41                 FA_X: FA    PORT MAP (
42                               x => RCA_10_bit_in_1(I),
43                               y => RCA_10_bit_in_2(I),
44                               c_in => RCA_10_bit_c_ripple(I-1),
45                               sum => RCA_10_bit_out(I),
46                               c_out => RCA_10_bit_c_ripple(I));
47              end generate FA_v;
48
49     -- Ultimo full_adder con il carry_out
50     FA_9 : FA PORT MAP (
51                               x => RCA_10_bit_in_1(N-1),
52                               y => RCA_10_bit_in_2(N-1),
53                               c_in => RCA_10_bit_c_ripple(N-2),
54                               sum => RCA_10_bit_out(N-1),
55                               c_out => RCA_10_bit_c_out
56                 );
57
58     -- Si ha overflow se i due carry dell'ultimo full adder sono diversi
59     -- Se i due carry dell'ultimo full adder sono uguali non si ha overflow e quindi si ha il ...
           carry out
60     RCA_10_bit_overflow <= RCA_10_bit_c_ripple(N-2) XOR RCA_10_bit_c_out;
61
62 END Structure;
```

8.6 RCA_18_bit.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  -- Sommatore su 8 bit
6  ENTITY RCA_18_bit IS
7      GENERIC (N : integer := 18);
8      PORT (
9          RCA_18_bit_in_1, RCA_18_bit_in_2: IN SIGNED(N-1 DOWNTO 0); -- Ingressi su N bit
10         RCA_18_bit_c_in: IN STD_LOGIC; -- Carry_in da sommare al resto
11         RCA_18_bit_out: OUT SIGNED(N-1 DOWNTO 0); -- Risultato della somma su N bit
12         RCA_18_bit_c_out, RCA_18_bit_overflow: BUFFER STD_LOGIC -- Segnali di carry out e ...
               overflow (Buffer poich uno serve per l'altro)
13     );
14 END RCA_18_bit;
15
16 ARCHITECTURE Structure OF RCA_18_bit IS
17
18     COMPONENT FA
19         PORT (
20             x, y, c_in : IN std_logic; -- x e y: bit da sommare, c_in: carry in
```

```
21              sum, c_out : OUT std_logic -- sum: somma di x e y, c_out: carry out
22          );
23      END COMPONENT;
24
25      -- Segnale intermedio per il PORT MAP
26      SIGNAL RCA_18_bit_c_ripple: SIGNED(N-2 DOWNTO 0); -- 'carry ripple': carry di supporto ...
            all'interno del RCA
27
28  BEGIN
29
30      -- Primo full adder con il carry_in
31      FA_0 : FA PORT MAP (
32                              x => RCA_18_bit_in_1(0),
33                              y => RCA_18_bit_in_2(0),
34                              c_in => RCA_18_bit_c_in,
35                              sum => RCA_18_bit_out(0),
36                              c_out => RCA_18_bit_c_ripple(0)
37                  );
38
39      -- Full adder in mezzo
40      FA_v : for I in 1 to N-2 generate
41                  FA_X: FA    PORT MAP (
42                              x => RCA_18_bit_in_1(I),
43                              y => RCA_18_bit_in_2(I),
44                              c_in => RCA_18_bit_c_ripple(I-1),
45                              sum => RCA_18_bit_out(I),
46                              c_out => RCA_18_bit_c_ripple(I));
47              end generate FA_v;
48
49      -- Ultimo full_adder con il carry_out
50      FA_17 : FA PORT MAP (
51                              x => RCA_18_bit_in_1(N-1),
52                              y => RCA_18_bit_in_2(N-1),
53                              c_in => RCA_18_bit_c_ripple(N-2),
54                              sum => RCA_18_bit_out(N-1),
55                              c_out => RCA_18_bit_c_out
56                  );
57
58      -- Si ha overflow se i due carry dell'ultimo full adder sono diversi
59      -- Se i due carry dell'ultimo full adder sono uguali non si ha overflow e quindi si ha il ...
            carry out
60      RCA_18_bit_overflow <= RCA_18_bit_c_ripple(N-2) XOR RCA_18_bit_c_out;
61
62  END Structure;
```

8.7 Average_8_to_18.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY Average_8_to_18 IS
6       PORT( Average_8_to_18_IN : IN SIGNED(7 DOWNTO 0);
7             Average_8_to_18_OUT : OUT SIGNED(17 DOWNTO 0)
8           );
9   END Average_8_to_18;
10
11  ARCHITECTURE Behaviour OF Average_8_to_18 IS
12  BEGIN
13      Average_8_to_18_OUT <= Average_8_to_18_IN(7) &
14                                  Average_8_to_18_IN(7) &
15                                  Average_8_to_18_IN(7) &
16                                  Average_8_to_18_IN(7) &
17                                  Average_8_to_18_IN(7) &
18                                  Average_8_to_18_IN(7) &
19                                  Average_8_to_18_IN(7) &
20                                  Average_8_to_18_IN(7) &
21                                  Average_8_to_18_IN(7) &
22                                  Average_8_to_18_IN(7) &
23                                  Average_8_to_18_IN(7 DOWNTO 0);
24  END Behaviour;
```

8.8 Counter_1024.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY Counter_1024 IS
5      PORT
6      (
7          Counter_1024_EN : IN std_logic; -- Segnale di Enable
8          Counter_1024_RST : IN std_logic; -- Segnale di Reset
9          Counter_1024_CLK : IN std_logic; -- Segnale di Clock
10         Counter_1024_TC : OUT std_logic; -- Segnale di fine conta
11         Counter_1024_OUT : BUFFER STD_LOGIC_VECTOR (9 DOWNTO 0)
12     );
13 END Counter_1024;
14
15 ARCHITECTURE Structural OF Counter_1024 IS
16
17     COMPONENT sync_4_bit_counter
18         PORT
19         (
20             count_4_enable : IN std_logic; -- Segnale di Enable
21             count_4_reset: IN std_logic; -- Segnale di Reset
22             count_4_clock : IN std_logic; -- Segnale di Clock
23             count_4_out: BUFFER std_logic_vector(3 DOWNTO 0); -- Uscita del contatore
24             count_4_carry: OUT std_logic -- Vale uno quando ha finito di contare
25         );
26     END COMPONENT;
27
28     COMPONENT sync_2_bit_counter
29         PORT
30         (
31             count_2_enable : IN std_logic; -- Segnale di Enable
32             count_2_reset: IN std_logic; -- Segnale di Reset
33             count_2_clock : IN std_logic; -- Segnale di Clock
34             count_2_out: BUFFER std_logic_vector(1 DOWNTO 0); -- Uscita del contatore
35             count_2_carry: OUT std_logic -- Vale uno quando ha finito di contare
36         );
37     END COMPONENT;
38
39     SIGNAL carry_to_enable : std_logic_vector (2 DOWNTO 0);
40     -- Segnale per il mapping che collega all'enable del FF successivo il 'carry' del precedente
41
42 BEGIN
43
44         -- FF bit meno significativi
45         count_4_0 : sync_4_bit_counter PORT MAP
46         (
47             count_4_enable => Counter_1024_EN,
48             count_4_reset => Counter_1024_RST,
49             count_4_clock => Counter_1024_CLK,
50             count_4_out => Counter_1024_OUT(3 DOWNTO 0),
51             count_4_carry => carry_to_enable(0)
52         );
53
54         count_4_1 : sync_4_bit_counter PORT MAP
55         (
56             count_4_enable => carry_to_enable(0),
57             count_4_reset => Counter_1024_RST,
58             count_4_clock => Counter_1024_CLK,
59             count_4_out => Counter_1024_OUT(7 DOWNTO 4),
60             count_4_carry => carry_to_enable(1)
61         );
62
63         count_2_0 : sync_2_bit_counter PORT MAP
64         (
65             count_2_enable => carry_to_enable(1),
66             count_2_reset => Counter_1024_RST,
67             count_2_clock => Counter_1024_CLK,
68             count_2_out => Counter_1024_OUT(9 DOWNTO 8),
69             count_2_carry => carry_to_enable(2)
70         );
71
72         Counter_1024_TC <= Counter_1024_OUT(0) AND
73                                 Counter_1024_OUT(1) AND
```

```
74                                      Counter_1024_OUT(2) AND
75                                      Counter_1024_OUT(3) AND
76                                      Counter_1024_OUT(4) AND
77                                      Counter_1024_OUT(5) AND
78                                      Counter_1024_OUT(6) AND
79                                      Counter_1024_OUT(7) AND
80                                      Counter_1024_OUT(8) AND
81                                      Counter_1024_OUT(9);
82
83  END Structural;
```

8.9 Cut_10_to_8_filter.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY Cut_10_to_8_filter IS
6       PORT( Cut_10_to_8_filter_IN : IN SIGNED(9 DOWNTO 0);
7             Cut_10_to_8_filter_OUT : OUT SIGNED(7 DOWNTO 0)
8           );
9   END Cut_10_to_8_filter;
10
11  ARCHITECTURE Behaviour OF Cut_10_to_8_filter IS
12  BEGIN
13      Cut_10_to_8_filter_OUT <= Cut_10_to_8_filter_IN(7 DOWNTO 0);
14  END Behaviour;
```

8.10 Divider_by_1024.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY Divider_by_1024 IS
5   PORT( Divider_by_1024_IN : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
6         Divider_by_1024_OUT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
7       );
8   END Divider_by_1024;
9
10  ARCHITECTURE Behaviour OF Divider_by_1024 IS
11  BEGIN
12      Divider_by_1024_OUT <= Divider_by_1024_IN(19 DOWNTO 10);
13  END Behaviour;
```

8.11 Mux_2_to_1_10bit.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY Mux_2_to_1_10bit IS
6       PORT( Mux_2_to_1_10bit_IN_0, Mux_2_to_1_10bit_IN_1: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
7             Mux_2_to_1_10bit_SEL : IN STD_LOGIC;
8             Mux_2_to_1_10bit_OUT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
9           );
10  END Mux_2_to_1_10bit;
11
12  ARCHITECTURE Behavior OF Mux_2_to_1_10bit IS
13
14  BEGIN
15      Mux_2_to_1_10bit_OUT <= Mux_2_to_1_10bit_IN_0 WHEN Mux_2_to_1_10bit_SEL = '0' ELSE
16                              Mux_2_to_1_10bit_IN_1;
17  END Behavior;
```

8.12 Mux_2_to_1_10bit_signed.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Mux_2_to_1_10bit_signed IS
6      PORT( Mux_2_to_1_10bit_IN_0, Mux_2_to_1_10bit_IN_1: IN SIGNED (9 DOWNTO 0);
7              Mux_2_to_1_10bit_SEL : IN STD_LOGIC;
8              Mux_2_to_1_10bit_OUT : OUT SIGNED (9 DOWNTO 0)
9          );
10 END Mux_2_to_1_10bit_signed;
11
12 ARCHITECTURE Behavior OF Mux_2_to_1_10bit_signed IS
13
14 BEGIN
15     Mux_2_to_1_10bit_OUT <= Mux_2_to_1_10bit_IN_0 WHEN Mux_2_to_1_10bit_SEL = '0' ELSE
16                                     Mux_2_to_1_10bit_IN_1;
17 END Behavior;
```

8.13 Mux_3to1_10bit.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY Mux_3_to_1_10bit IS
5      PORT( Mux_3_to_1_10bit_IN_0, Mux_3_to_1_10bit_IN_1, Mux_3_to_1_10bit_IN_2 : IN ...
           STD_LOGIC_VECTOR (9 DOWNTO 0);
6              Mux_3_to_1_10bit_SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
7              Mux_3_to_1_10bit_OUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
8          );
9  END Mux_3_to_1_10bit;
10
11 ARCHITECTURE Behavior OF Mux_3_to_1_10bit IS
12
13 BEGIN
14     Mux_3_to_1_10bit_OUT <= Mux_3_to_1_10bit_IN_0 WHEN Mux_3_to_1_10bit_SEL = "00" else
15                     Mux_3_to_1_10bit_IN_1 WHEN Mux_3_to_1_10bit_SEL = "01" else
16                     Mux_3_to_1_10bit_IN_2;
17
18 END Behavior;
```

8.14 Mux_3to1_10bit_signed.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Mux_3_to_1_10bit_signed IS
6      PORT( Mux_3_to_1_10bit_IN_0, Mux_3_to_1_10bit_IN_1, Mux_3_to_1_10bit_IN_2 : IN SIGNED (9 ...
           DOWNTO 0);
7              Mux_3_to_1_10bit_SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
8              Mux_3_to_1_10bit_OUT : OUT SIGNED (9 DOWNTO 0)
9          );
10 END Mux_3_to_1_10bit_signed;
11
12 ARCHITECTURE Behavior OF Mux_3_to_1_10bit_signed IS
13
14 BEGIN
15     Mux_3_to_1_10bit_OUT <= Mux_3_to_1_10bit_IN_0 WHEN Mux_3_to_1_10bit_SEL = "00" else
16                     Mux_3_to_1_10bit_IN_1 WHEN Mux_3_to_1_10bit_SEL = "01" else
17                     Mux_3_to_1_10bit_IN_2;
18
19 END Behavior;
```

8.15 Ovf_Sign_10_to_8_filter.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Ovf_Sign_10_to_8_filter IS
6      PORT ( Ovf_Sign_10_to_8_filter_IN : IN SIGNED(9 DOWNTO 0);
7               Ovf_Sign_10_to_8_filter_OVF, Ovf_Sign_10_to_8_filter_SIGN : OUT STD_LOGIC
8             );
9  END Ovf_Sign_10_to_8_filter;
10
11 ARCHITECTURE Behaviour OF Ovf_Sign_10_to_8_filter IS
12 BEGIN
13     --- questo segnale ci dice quando il numero non  rappresentabile su 8 bit, ci si verifica se
14     --- i tre bit pi significativi non sono uguali, ossia i due bit pi significativi devono
15     --- essere uguali all'ottavo bit che viene considerato come bit di segno. TROVO LA F LOGICA
16     --- NOT((a*b*c) + NOT(a*b*c))=  ->  UNO SE I BIT NON SONO TUTTI UGLUALI
17     --- = NOT(a*b*c)*(a+b+c) = (not a + not b + not c)*(a+b+c) = (a xor b)+ (a xor c) + (b ...
           xor c)
18
19     Ovf_Sign_10_to_8_filter_OVF <=
20             (
21             ( Ovf_Sign_10_to_8_filter_IN(9) XOR Ovf_Sign_10_to_8_filter_IN(8) )OR
22             ( Ovf_Sign_10_to_8_filter_IN(9) XOR Ovf_Sign_10_to_8_filter_IN(7) )OR
23             ( Ovf_Sign_10_to_8_filter_IN(9) XOR Ovf_Sign_10_to_8_filter_IN(7) )
24             );
25
26
27
28     Ovf_Sign_10_to_8_filter_SIGN <= Ovf_Sign_10_to_8_filter_IN(9);
29
30 END Behaviour;
```

8.16 sync_2_bit_counter.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY sync_2_bit_counter IS
5      PORT
6      (
7          count_2_enable : IN std_logic; -- Segnale di Enable
8          count_2_reset: IN std_logic; -- Segnale di Reset
9          count_2_clock : IN std_logic; -- Segnale di Clock
10         count_2_out: BUFFER std_logic_vector(1 DOWNTO 0); -- Uscita del contatore
11         count_2_carry: OUT std_logic -- Vale uno quando ha finito di contare
12     );
13 END sync_2_bit_counter;
14
15 ARCHITECTURE Structural OF sync_2_bit_counter IS
16
17     COMPONENT T_FF
18         PORT
19         (
20             T : IN std_logic; -- Segnale di Toggle
21             T_clock: IN std_logic; -- Segnale di Clock
22             T_reset : IN std_logic; -- Segnale di Reset
23             T_Q: BUFFER std_logic -- Uscita Q del FF,   buffer perch la uso anche come ingresso
24         );
25     END COMPONENT;
26
27     SIGNAL Q_inside: std_logic_vector (1 DOWNTO 0); -- Segnale interno che contiene le uscite ...
           iesime
28     SIGNAL T_inside: std_logic_vector (1 DOWNTO 0); -- Segnale interno che contiene gli ...
           ingressi T iesimi
29
30 BEGIN
31
32     -- T_inside   il segnale T dell'iesimo T_Flip_Flop
33     -- Q_inside   il segnale Q dell'iesimo T_Flip_Flop
```

```
34
35      T_inside(0) <= count_2_enable; -- Il toggle del primo flip flop   l'enable.
36      T_inside(1) <= T_inside(0) AND Q_inside(0); -- Il toggle del secondo flip flop   l'AND ...
            tra l'uscita precedente e il toggle precedente.
37
38      -- PORT MAP dei vari T_FF
39
40
41      create_t_flip_flops : FOR i IN 0 TO 1 GENERATE
42          T_i : T_FF PORT MAP
43          (
44              T => T_inside(i),
45              T_clock => count_2_clock,
46              T_reset => count_2_reset,
47              T_Q => Q_inside(i)
48          );
49      END GENERATE;
50
51      -- Prelevo l'uscita dal segnale Q interno
52      count_2_out <= Q_inside;
53
54      -- Segnale di Carry out per usare contatori in parallelo
55      count_2_carry <= T_inside(1) AND Q_inside(1);
56
57  END Structural;
```

8.17 sync_4_bit_counter.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY sync_4_bit_counter IS
5       PORT
6       (
7           count_4_enable : IN std_logic; -- Segnale di Enable
8           count_4_reset: IN std_logic; -- Segnale di Reset
9           count_4_clock : IN std_logic; -- Segnale di Clock
10          count_4_out: BUFFER std_logic_vector(3 DOWNTO 0); -- Uscita del contatore
11          count_4_carry: OUT std_logic -- Vale uno quando ha finito di contare
12      );
13  END sync_4_bit_counter;
14
15  ARCHITECTURE Structural OF sync_4_bit_counter IS
16
17      COMPONENT T_FF
18          PORT
19          (
20              T : IN std_logic; -- Segnale di Toggle
21              T_clock: IN std_logic; -- Segnale di Clock
22              T_reset : IN std_logic; -- Segnale di Reset
23              T_Q: BUFFER std_logic -- Uscita Q del FF,   buffer perch la uso anche come ingresso
24          );
25      END COMPONENT;
26
27      SIGNAL Q_inside: std_logic_vector (3 DOWNTO 0); -- Segnale interno che contiene le uscite ...
            iesime
28      SIGNAL T_inside: std_logic_vector (3 DOWNTO 0); -- Segnale interno che contiene gli ...
            ingressi T iesimi
29
30  BEGIN
31
32      -- T_inside   il segnale T dell'iesimo T_Flip_Flop
33      -- Q_inside   il segnale Q dell'iesimo T_Flip_Flop
34
35      T_inside(0) <= count_4_enable; -- Il toggle del primo flip flop   l'enable.
36      T_inside(1) <= T_inside(0) AND Q_inside(0); -- Il toggle del secondo flip flop   l'AND ...
            tra l'uscita precedente e il toggle precedente.
37      T_inside(2) <= T_inside(1) AND Q_inside(1); -- E cos via
38      T_inside(3) <= T_inside(2) AND Q_inside(2);
39
40      -- PORT MAP dei vari T_FF
41
42
```

```
43      create_t_flip_flops : FOR i IN 0 TO 3 GENERATE
44          T_i : T_FF PORT MAP
45          (
46              T => T_inside(i),
47              T_clock => count_4_clock,
48              T_reset => count_4_reset,
49              T_Q => Q_inside(i)
50          );
51      END GENERATE;
52
53      -- Prelevo l'uscita dal segnale Q interno
54      count_4_out <= Q_inside;
55
56      -- Segnale di Carry out per usare contatori in parallelo
57      count_4_carry <= T_inside(3) AND Q_inside(3);
58
59  END Structural;
```

8.18 Three_operation_block.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY Three_operation_block IS
6       PORT ( Three_operation_block_IN : IN SIGNED(7 DOWNTO 0);
7                Three_operation_block_DIV4,
8                Three_operation_block_MULT2,
9                Three_operation_block_transparent : OUT SIGNED(9 DOWNTO 0)
10             );
11  END Three_operation_block;
12
13  ARCHITECTURE Structural OF Three_operation_block IS
14  BEGIN
15
16      Three_operation_block_DIV4 <= (Three_operation_block_IN(7) &
17                                             Three_operation_block_IN(7) &
18                                             Three_operation_block_IN(7) &
19                                             Three_operation_block_IN(7) &
20                                             Three_operation_block_IN(7 DOWNTO 2));
21
22      Three_operation_block_MULT2 <= Three_operation_block_IN(7) & Three_operation_block_IN & '0';
23
24      Three_operation_block_transparent <= Three_operation_block_IN(7) &
25                                                   Three_operation_block_IN(7) &
26                                                   Three_operation_block_IN;
27
28  END Structural;
```

8.19 Valid_address.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY Valid_address IS
5       PORT ( Valid_address_in : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
6                Valid_address_2, Valid_address_3 : OUT STD_LOGIC;
7                Valid_address_counter_LSB : OUT STD_LOGIC
8              );
9   END Valid_address;
10
11  ARCHITECTURE Behaviour OF Valid_address IS
12
13  BEGIN
14      --- l'indirizzo del primo contatore, ossia il secondo da utilizzare  valido se il ...
            contatore  diverso da 0
15      --- quindi lo  se non ho tutti i bit a 0, ossia dev'esserci almeno un bit a 1 perch sia ...
            valido
16      Valid_address_2 <= Valid_address_in(0) OR
```

```
17                              Valid_address_in(1) OR
18                              Valid_address_in(2) OR
19                              Valid_address_in(3) OR
20                              Valid_address_in(4) OR
21                              Valid_address_in(5) OR
22                              Valid_address_in(6) OR
23                              Valid_address_in(7) OR
24                              Valid_address_in(8) OR
25                              Valid_address_in(9);
26
27      --- l'indirizzo 3  valido se il contatore  maggiore di 3 ossia se
28      --- almeno uno dei bit 9-2  a 1  oppure   tutti i bit 9-2 sono a 0 ma l'1 E il 2 sono a 1
29      Valid_address_3 <= (Valid_address_in(0) AND Valid_address_in(1) ) OR
30                              Valid_address_in(2) OR
31                              Valid_address_in(3) OR
32                              Valid_address_in(4) OR
33                              Valid_address_in(5) OR
34                              Valid_address_in(6) OR
35                              Valid_address_in(7) OR
36                              Valid_address_in(8) OR
37                              Valid_address_in(9);
38      Valid_address_counter_LSB <= Valid_address_in(0);
39  END Behaviour;
```

8.20 T_FF.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY T_FF IS
5       PORT
6       (
7           T : IN std_logic; -- Segnale di Toggle
8           T_clock: IN std_logic; -- Segnale di Clock
9           T_reset : IN std_logic; -- Segnale di Reset
10          T_Q: BUFFER std_logic -- Uscita Q del FF,   buffer perch la uso anche come ingresso
11      );
12  END T_FF;
13
14
15  ARCHITECTURE Structure OF T_FF IS
16
17      COMPONENT FF_D -- FLIP FLOP
18          PORT
19          (
20              D, Clock, Reset: IN std_logic; -- Reset asincrono
21              Q : OUT std_logic
22          );
23      END COMPONENT;
24
25      SIGNAL T_inside: std_logic;
26
27  BEGIN
28
29      T_inside <= T_Q XOR T; -- Al posto del multiplexer, l'ingresso D   dato da XOR tra uscita ...
            Q e segnale toggle T
30
31      FF_1 : FF_D PORT MAP
32      (
33          D => T_inside, -- All'ingresso del T flip flop ci va l'uscita in XOR con l'ingresso.
34          Clock => T_clock,
35          Reset => T_reset,
36          Q => T_Q
37      );
38
39  END Structure;
```

8.21 FA.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY FA IS
6      PORT (
7          x, y, c_in : IN std_logic; -- x e y: bit da sommare, c_in: carry in
8          sum, c_out : OUT std_logic -- sum: somma di x e y, c_out: carry out
9      );
10 END FA;
11
12 ARCHITECTURE Behavior OF FA IS
13
14     SIGNAL P,G: std_logic; -- P: propagate, G: generate
15
16 BEGIN
17
18     P <= x XOR y; -- Se P=1, il propagate propaga in uscita il carry_in
19     G <= x AND y; -- Se G=1, il generate forza il carry out a 1
20     sum <= c_in XOR P;
21     c_out <= G OR (P AND c_in);
22
23 END Behavior;
```

8.22 FF.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY FF_D IS
6      PORT (
7          D, Clock, Reset: IN std_logic; -- Reset asincrono
8          Q : OUT std_logic
9      );
10 END FF_D;
11
12 ARCHITECTURE Behavior OF FF_D IS
13 BEGIN
14         PROCESS(Clock, Reset)
15         BEGIN
16             IF (Reset = '1') THEN
17                 Q <= '0';
18             ELSIF (Clock'EVENT AND Clock = '1') THEN
19                 Q <= D;
20             END IF;
21         END PROCESS;
22 END Behavior;
```