POLITECNICO
DI TORINO

ASM Chart Design

Digital Filter

Introduzione

Il progetto prevede l'ideazione di un circuito la cui funzione è quella di salvare, all'interno di una memoria denominata MEM_A, dei dati in ingresso. Una volta effettuato questo caricamento la macchina procede rielaborando tali dati eseguendo l'equazione in figura e salvando i risultati all'interno di un'altra memoria, MEM_B. Contemporaneamente al caricamento nella seconda memoria, la macchina esegue il calcolo della media aritmetica dei dati precedentemente salvati all'interno della MEM_A.

$$Y(n) = (-1)^n [0.25X(n) + X(n-1) - 2X(n-3)]$$

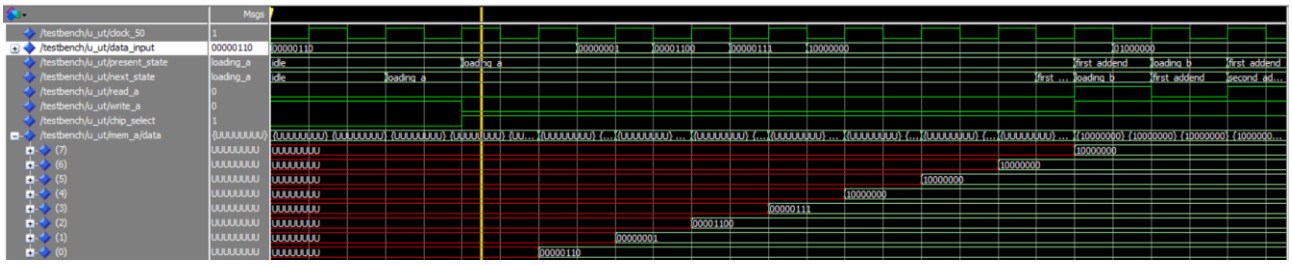
La macchina progettata è pertanto una ASM, la cui Control Unit fornisce, di stato in stato, i segnali necessari affinché le operazioni di caricamento e somma avvengano con il timing corretto.

Caricamento memoria A

Nel momento in cui la macchina viene accesa, il primo stato è quello di Idle, durante il quale si attende il segnale di "*Start*" che sarà anche segnale responsabile dell'iniziale reset dei componenti della macchina.

Una volta percepito "*Start*" la macchina passa allo stato di caricamento della prima memoria. MEM_A viene accesa e configurata in scrittura. Contemporaneamente viene attivato un segnale di enable per il counter che passa l'indirizzo corretto alla memoria. Durante questa fase i dati vengono aggiornati ad ogni fronte di salita del clock e l'indirizzo aumenta di una unità con lo stesso timing di modo che la memoria venga riempita completamente senza creare spazi vuoti tra le celle.

Il completo riempimento della memoria viene riconosciuto dall'attivazione di un segnale, *Terminal_Count*, una volta che il counter è arrivato al massimo valore. La macchina passa quindi al primo degli stati che porteranno al calcolo del valore *Y(n)*, dove *n* rappresenta l'indirizzo di memoria dove il dato verrà salvato all'interno della MEM_B.



Simulazione caricamento MEM_A

L'immagine precedente mostra una simulazione effettuata su Modelsim, avendo diminuito per semplicità la dimensione della memoria ad 1byte. Non appena il caricamento è terminato, la macchina passa allo stato successivo, prendendo il primo dato necessario per il calcolo di *Y(n)*.

Calcolo *Y(n)*

Al fine di calcolare il valore si *Y(n)* si è scelto di creare quattro stati differenti, uno per ogni operazione necessaria a compiere il calcolo completo. Tre di questi stati sono responsabili della corretta scelta dell'addendo, mentre l'ultimo è dedicato al cambio del segno della somma calcolata.

In ognuno degli stati che leggono dalla MEM_A un nuovo addendo, la memoria viene accesa e impostata in modalità lettura. Solo dopo aver compiuto completamente il calcolo di *Y(n)*, la macchina passa allo stato di caricamento della MEM_B, per poi tornare al primo stato necessario a computare il nuovo dato da inserire.

Facendo riferimento al Datapath della macchina, si può notare che i blocchi che contribuiscono al calcolo di *Y(n)* sono due *Mux*, uno *Shift Register*, un *Adder* ed un *Register.*

Cambio del segno Il primo Mux è collegato all'ingresso dello Shift Register e, a seconda del valore del segnale di selezione, fa passare il dato proveniente dalla memoria oppure zero, nel caso ci si trovi nello stato di modifica del segno. Difatti per effettuare il cambio del segno di *Y(n)* si è scelto di sottrarre a zero il valore di Y appena calcolato. Nonostante il cambio del segno sia da applicare in caso di indirizzo dispari, poiché quando viene preso il terzo addendo l'adder esegue una sottrazione tra *"Input 1"* (terzo addendo) ed *"Input 2"* (somma dei primi due addendi), il risultato ottenuto possiede modulo corretto e segno corretto in caso di indirizzo dispari, opposto in caso di indirizzo pari. Proprio per questo motivo si è scelto di far passare la macchina nello stato di cambio segno solo per indirizzo pari.

Ingresso dei dati Il secondo Mux è direttamente collegato al Counter dell'indirizzo di memoria e modula il valore dell'indirizzo che viene passato, a seconda del valore del segnale di selezione relativo al mux stesso. Questo segnale cambia a seconda dello stato in cui ci si trova, di modo che venga passato l'indirizzo di memoria necessario.

È stata fatta la scelta di istanziare un unico Shift Register, progettato con più segnali di enable che attivano lo shifting a destra o a sinistra a seconda dell'operazione da compiere per fornire il dato corretto all'Adder.

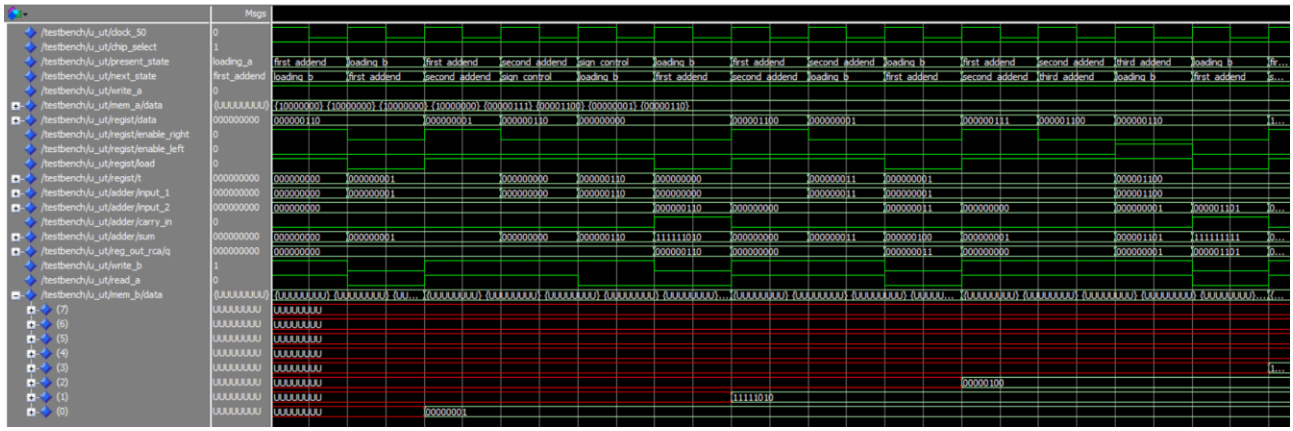
La macchina è stata progettata di modo che, ad ogni fronte di salita del segnale di clock, lo Shift Register potesse caricare un nuovo dato. È stato studiato il timing di modo che la macchina lavori, quindi, in pipeline, evitando di aspettare che ogni operazione venga eseguita prima di caricare i dati successivi alla prossima, ma bensì caricando già i dati successivi contemporaneamente all'operazione di addizione.

Per quanto concerne le varie addizioni si è reso necessario variare il parallelismo interno aumentando il numero di bit degli addendi a nove. In questo modo l'analisi dei valori ottenuti, soprattutto al fine della successiva saturazione dei dati è stata resa più semplice.

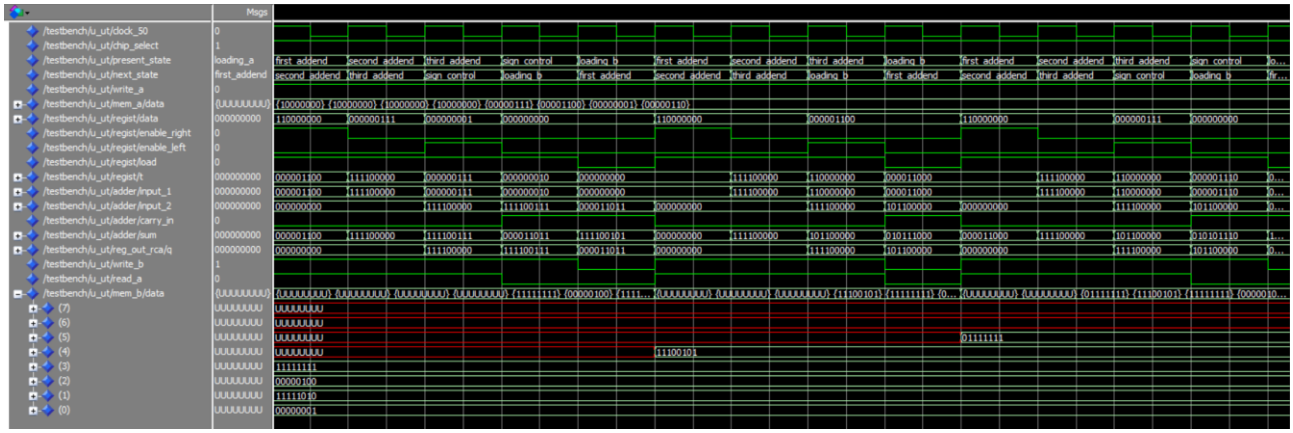
Indirizzi negativi È necessario portare in evidenza la scelta di gestione dei segnali con indirizzo negativo. Infatti si è scelto di evitare di far eseguire alla macchina somme con termini che sarebbero dovuti essere, secondo specifiche di progetto, posti uguali a zero, ma nei casi che vanno da indirizzo pari a zero a indirizzo pari a due vengono presi solo i dati non presentanti termini con indici negativi. Per una migliore comprensione si può fare riferimento alla figura seguente, dove si nota in maniera immediata la mancanza dei termini con indirizzo negativo nei casi prima esposti.

$$Y(0) = 0.25X(0) + X(-1) - 2X(-3) = 0.25X(0)$$
$$Y(1) = -[0.25X(1) + X(0) - 2X(-2)] = -[0.25X(1) + X(0)]$$
$$Y(2) = 0.25X(2) + X(1) - 2X(-1)| = 0.25X(2) + X(1))$$

Di seguito delle immagini che mostrano il comportamento della macchina nei casi sopra citati.



Simulazione Modelsim caricamento primi tre dati in MEM_B



Simulazione Modelsim caricamento dati in MEM_B

Caricamento memoria B

Per compiere il caricamento del dato calcolato mediante le operazioni precedentemente descritte, è stato creato uno stato *Loading_B* nel quale la memoria viene configurata con le stesse condizioni dello stato *Loading_A.*

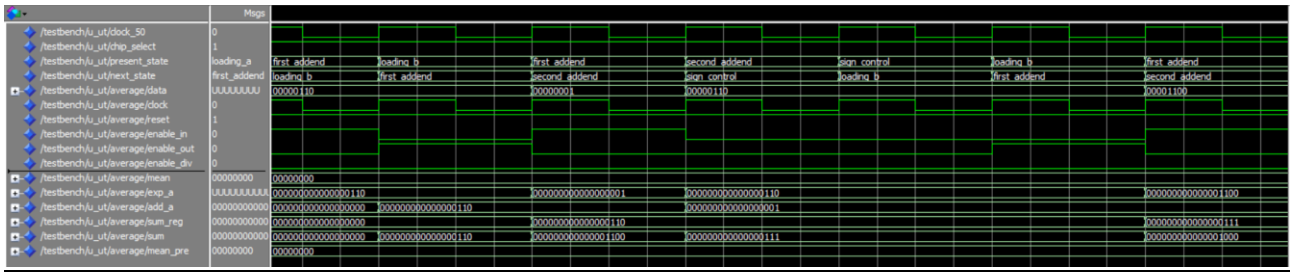
A differenza dello stato di caricamento della memoria A è stato istanziato un Mux in ingresso alla memoria responsabile della saturazione dei dati in ingresso. La *Control Unit* è responsabile della scelta del valore corretto da assegnare al segnale di selezione di questo Mux. A seconda del suo valore infatti il Mux manda in uscita il segnale in uscita dal registro posto in coda all'Adder, oppure "01111111" o "10000000", se il segnale dal Mux è rispettivamente più grande del massimo o più piccolo del minimo valore rappresentabile su 8 bit.

Calcolo della media

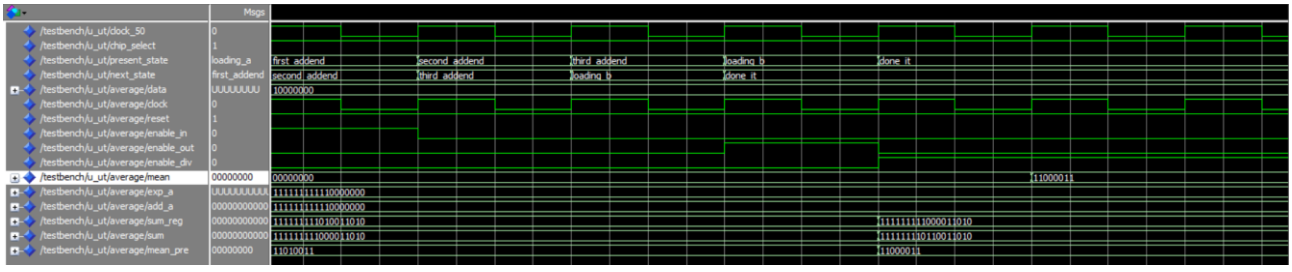
In maniera contemporanea ai passaggi descritti sinora, al circuito vengono passati dalla Control Unit dei segnali tali da far attivare il blocco per il calcolo della media aritmetica dei valori precedentemente caricati all'interno della memoria A.

Facendo riferimento al Datapath si può evincere la struttura del circuito mediante il quale la media viene calcolata. Tale circuito è composto da due registri, uno Shift Register responsabile della divisione finale, ed un Adder.

La struttura è molto simile a quella descritta per il calcolo di *Y(n)*, con la differenza che, in questo caso, un nuovo dato viene passato in ingresso ogni volta che si passa per lo stato che legge dalla memoria il primo addendo per calcolare Y(n). Negli stati seguenti i segnali di enable dei componenti vengono mantenuti a zero. In questo modo il circuito esegue la somma solo quando un nuovo dato passa in ingresso al registro che precede l'Adder. Infine la divisione, poiché deve essere compiuta per una potenza di due, viene effettuata mediante un registro cui vengono passati in ingresso solamente i bit corrispondenti al parallelismo del registro. Il risultato corrisponde all'implementazione di uno shift register, senza effettivamente aver effettuato la sua istanza. Il segnale di enable di questo registro viene attivato solo nel momento in cui la macchina ha terminato tutte le operazioni ed entra nello stato *"Done"*, di modo che in ultima azione venga mandato in output il valore della media e che la macchina aspetti che il segnale di *Start* sia equivalente allo zero logico per poter ripartire.



Inizio calcolo della media

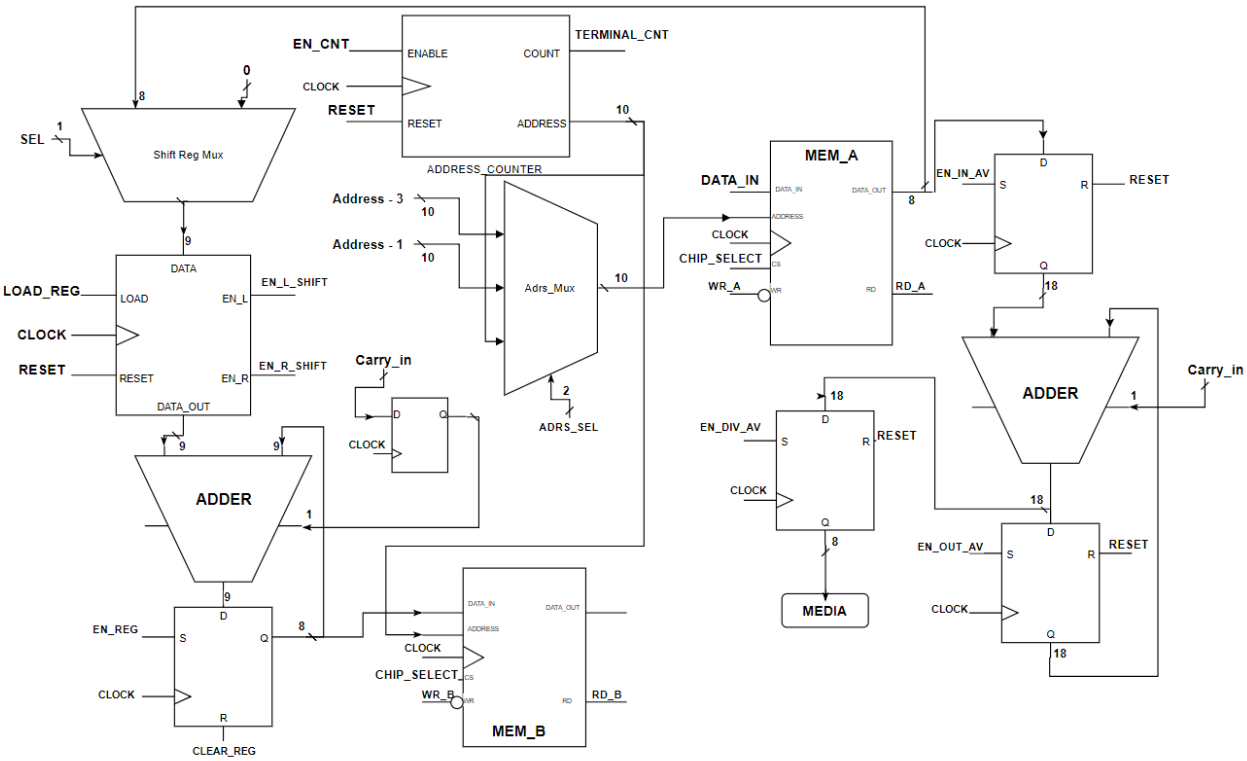


Fine calcolo media

PSEUDO CODICE

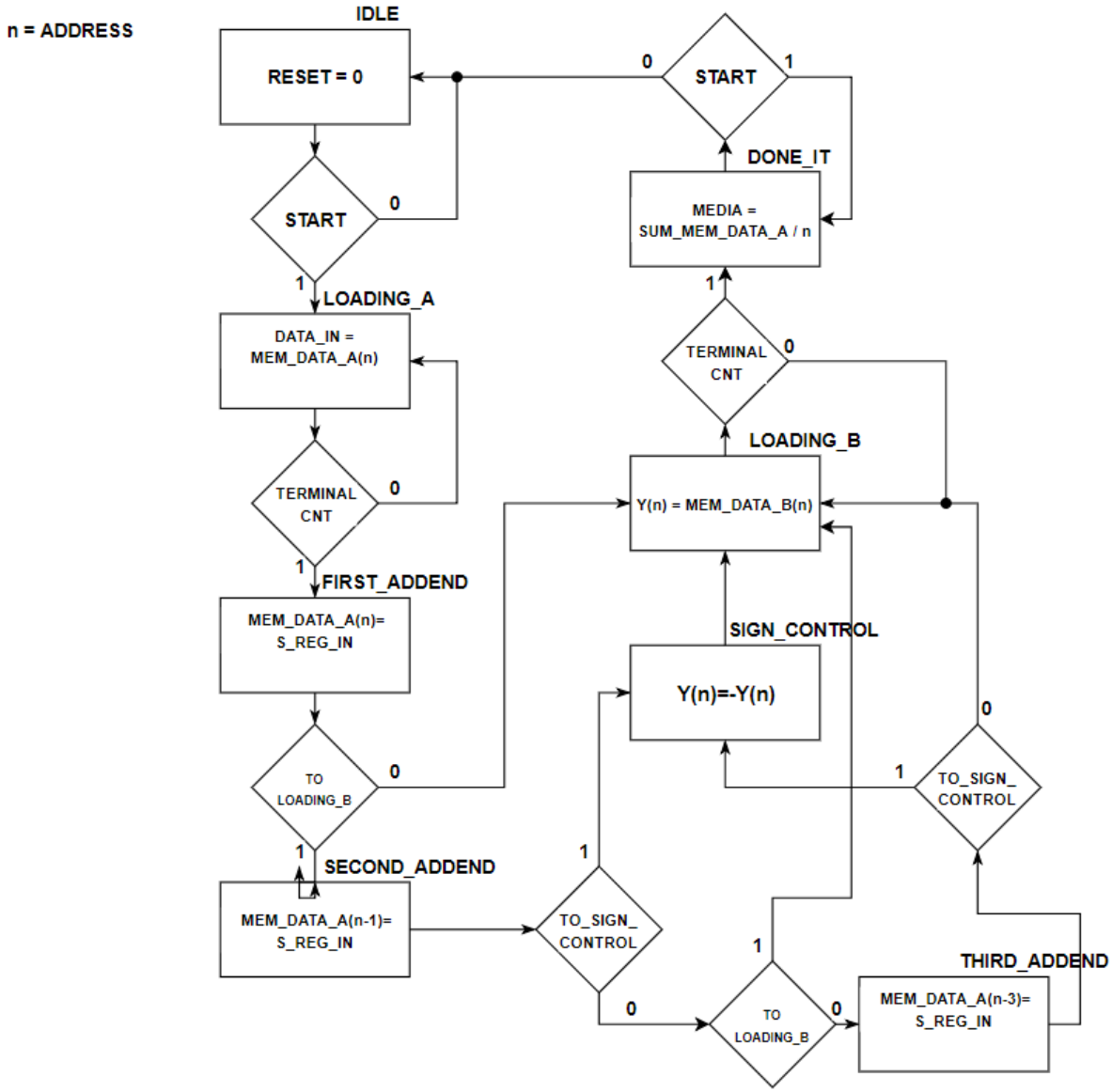
```
#include <stdio.h>
#include <stdlib.h>

#DEFINE MEMORY_A 1024 8
#DEFINE MEMORY_B 1024 8

float get_input_values()
{
   // codice ingresso dati input
   return data_in

}

void save_memory_A(float data_in)
{
   // salvataggio memoria A con ingresso di data_in
}

void save_memory_B(float data)
{
   //
}

int shift_reg_right(int x)
{
   //
}

int shift_reg_left(int x)
{
   //
}

int main()
{
   // Reset machine
   float data_in [1023];
   int n;
   bool done = 0;
   bool start = 0;

 IDLE:  if (start==0)
          goto IDLE;

 data_in = get_input_values();

 for (n=0 ; n<1024 ; n++){

   save_memory_A(data_in);
 }



for (n=0; n<1024; n++)

 {
   // variabili locali impostate a 0
   float x(-3) = 0;
   float x(-2) = 0;
   float x(-1) = 0;
   //
   float A , B , C;
    A = shift_reg_right(x(n)); // shift a destra del dato
    B = x(n-1);
    C = shift_reg_left(x(n-3));  //shift a sinistra del dato


    y(n)= A + B - C;

    if( (n%2)==0 ) // divisione per 2 ha resto nullo
    {
      save_memory_B(-y(n));
      }
    else

      save_memory_B(y(n));
      }

   for (i=0; i<1024; i++)
{
      OUT = OUT + MEMORY_A(i);

}
   OUT = OUT / 1024;
   done = 1;

   return 0;
}
```

DATAPATH

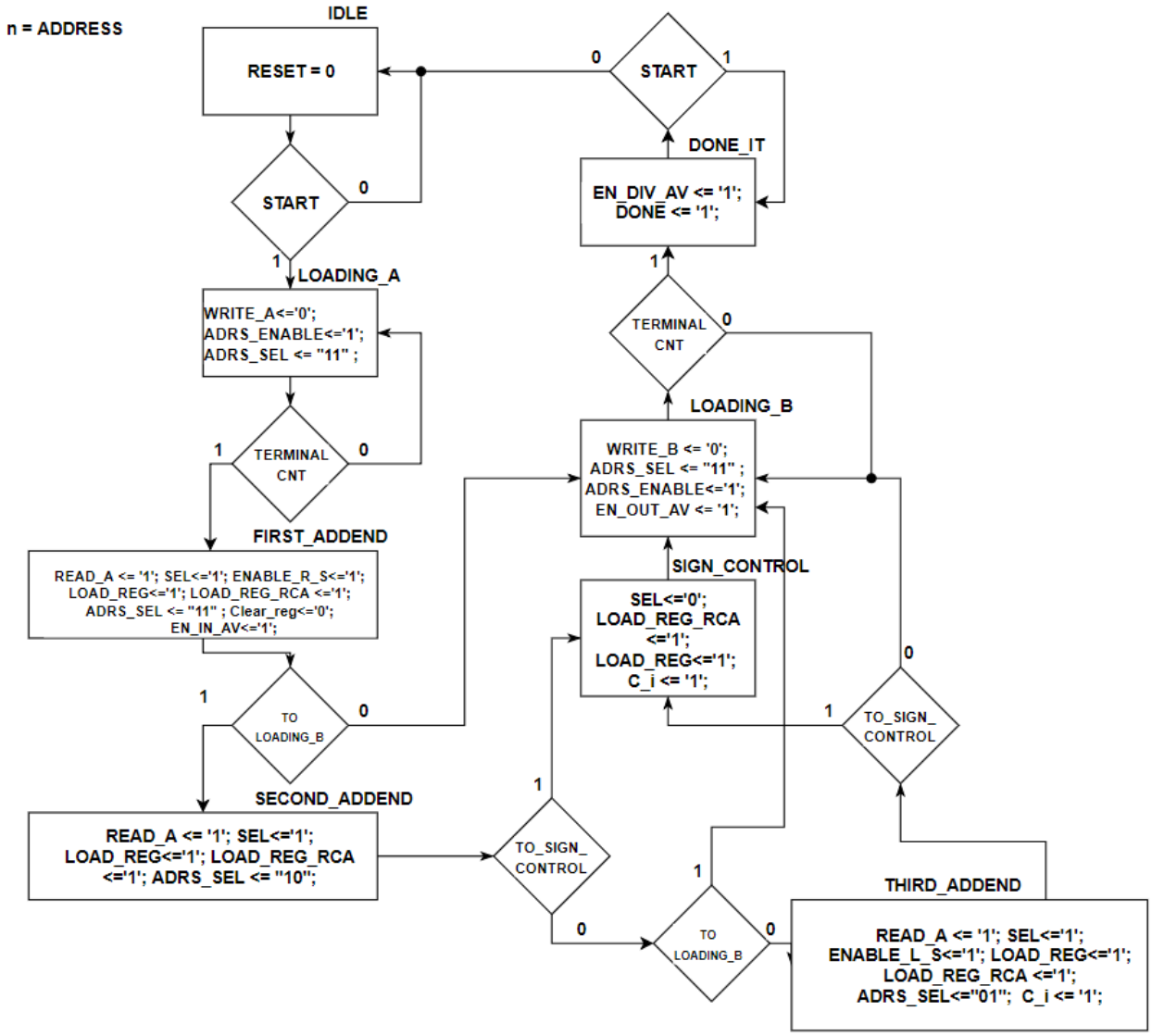

EN_CNT
ENABLE
COUNT
TERMINAL_CNT
CLOCK
RESET
RESET
ADDRESS
10
ADDRESS_COUNTER
0
8
SEL
1
Shift Reg Mux
9
Address - 3
10
Address - 1
10
Adrs_Mux
10
2
ADRS_SEL
DATA
LOAD_REG
LOAD
EN_L
EN_L_SHIFT
CLOCK
RESET
RESET
EN_R
EN_R_SHIFT
DATA_OUT
Carry_in
D
Q
CLOCK
9
9
ADDER
1
9
D
EN_REG
S
Q
8
CLOCK
R
CLEAR_REG
DATA_IN
DATA_OUT
ADDRESS
CLOCK
CHIP_SELECT
CS
WR_B
WR
RD
RD_B
MEM_B
MEM_A
DATA_IN
DATA_IN
DATA_OUT
ADDRESS
CLOCK
CHIP_SELECT
CS
WR_A
WR
RD
RD_A
8
EN_IN_AV
D
S
R
RESET
CLOCK
Q
18
ADDER
Carry_in
1
18
D
EN_DIV_AV
S
R
RESET
CLOCK
Q
8
MEDIA
18
D
EN_OUT_AV
S
R
RESET
CLOCK
Q
18

ASM CHART

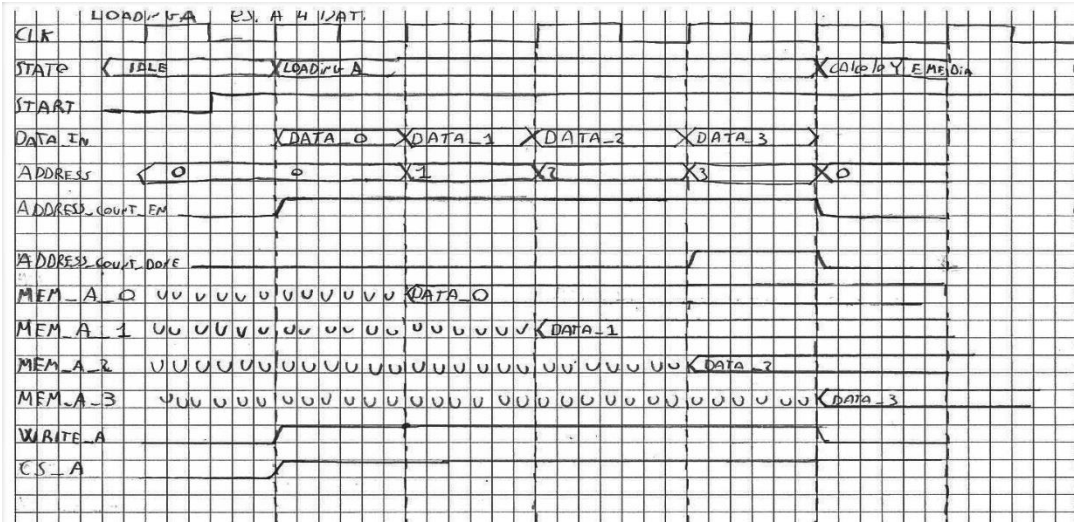
n = ADDRESS



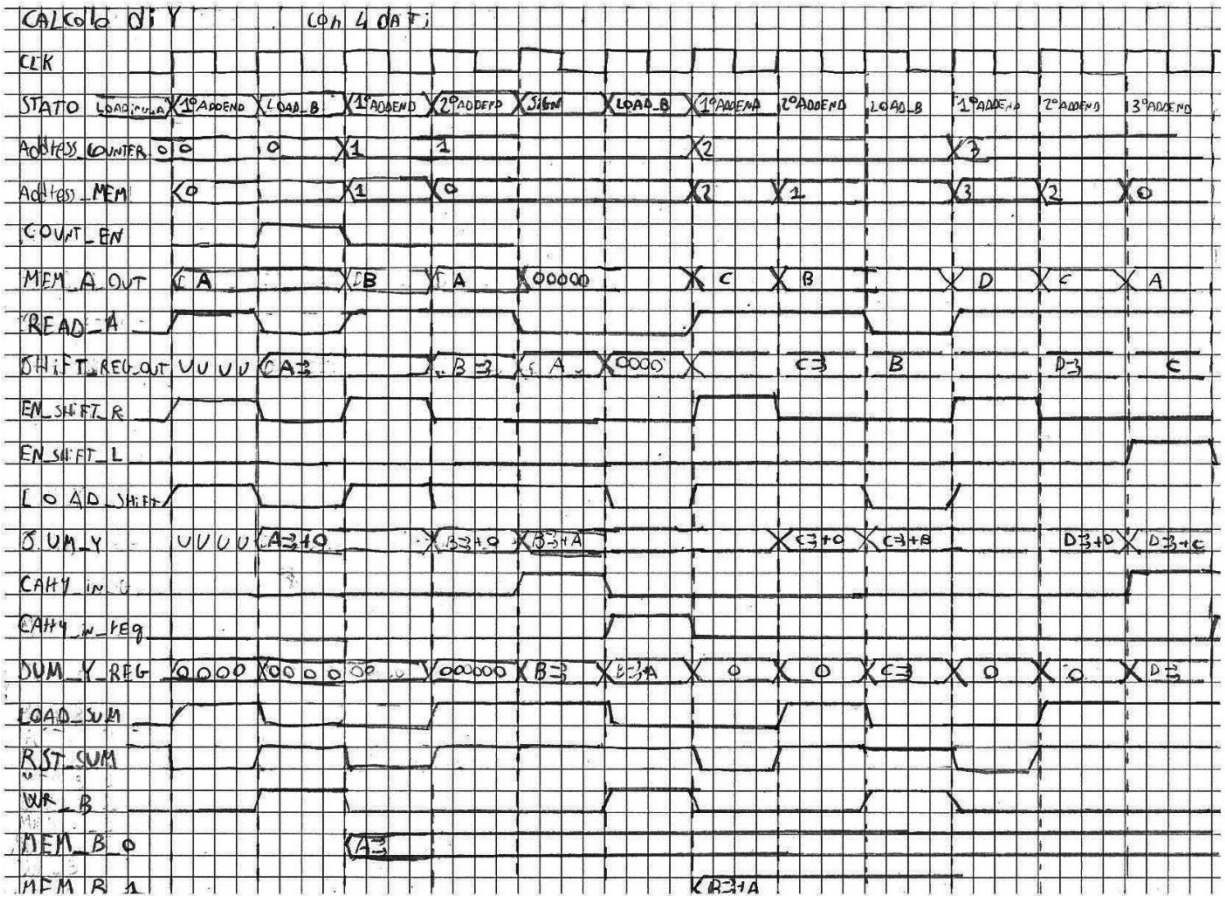
CONTROL UNIT



HANDMADE TIMING



Caricamento memoria A



Calcolo Y(n)

VHDL

Digital_Filter

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Digital_Filter IS
PORT ( CLOCK_50, START, RESET: IN STD_LOGIC;
                    DATA_INPUT : IN SIGNED(7 DOWNTO 0);
                    DONE : OUT STD_LOGIC;
                    MEDIA : OUT SIGNED(7 DOWNTO 0)
                   );
END Digital_Filter;

ARCHITECTURE STRUCTURE OF Digital_Filter IS
COMPONENT Memory
        PORT ( DATA_IN : IN SIGNED(7 DOWNTO 0);
                         ADDRESS : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                         CS, CLK, WR, RD:  IN STD_LOGIC;
                         DATA_OUT : OUT SIGNED(7 DOWNTO 0)
                    );
END COMPONENT;

COMPONENT Adrs_Counter
                    PORT (CLOCK, RESET, ENABLE_COUNT :  IN STD_LOGIC;
                                    ADRS : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
                                COUNT : OUT STD_LOGIC
                                );
END COMPONENT;

COMPONENT Mux2to1_9bit
PORT ( IN_1 : IN SIGNED(8 DOWNTO 0);
                         SEL : IN STD_LOGIC;
                         U : OUT SIGNED(8 DOWNTO 0)
                         );
END COMPONENT ;

COMPONENT Mux_3_to_1
        PORT( ADRS: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                        SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        U : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
                     );
END COMPONENT;

COMPONENT Shift_register
                    PORT ( Data : IN SIGNED(8 DOWNTO 0);
                                CLOCK, RESET, ENABLE_RIGHT, ENABLE_LEFT, LOAD: IN STD_LOGIC;
                                T : OUT SIGNED(8 DOWNTO 0)
                                );
END COMPONENT;

COMPONENT regn
                                GENERIC ( N : integer);
                    PORT (R : IN SIGNED(N-1 DOWNTO 0);
                                Clock, Resetn, LOAD : IN STD_LOGIC;
                                        Q : OUT SIGNED(N-1 DOWNTO 0)
                                        );
END COMPONENT;

COMPONENT flipflop
  PORT (D, Clock, Resetn : IN STD_LOGIC;
```

```
      Q : OUT STD_LOGIC);
END COMPONENT;

COMPONENT RCA
          GENERIC (P : INTEGER);
          PORT (INPUT_1, INPUT_2 : IN SIGNED(P-1 DOWNTO 0);
                              CARRY_IN : IN STD_LOGIC;
                              CARRY_OUT : OUT STD_LOGIC;
                              SUM : OUT SIGNED(P-1 DOWNTO 0)
                              );
END COMPONENT;

COMPONENT Memory_mux
          PORT( IN_1: IN SIGNED(7 DOWNTO 0);
                              SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                              U : OUT SIGNED(7 DOWNTO 0)
                    );
END COMPONENT;

COMPONENT Aritchmetic_mean
PORT ( DATA : IN SIGNED (7 DOWNTO 0);
                    CLOCK, RESET, ENABLE_IN, ENABLE_OUT, ENABLE_DIV: IN STD_LOGIC;
                    OK : BUFFER STD_LOGIC;
                    MEAN                    : OUT SIGNED (7 DOWNTO 0)
                    );
END COMPONENT;



TYPE State_type IS (IDLE, LOADING_A, LOADING_B, FIRST_ADDEND, SECOND_ADDEND,
THIRD_ADDEND,SIGN_CONTROL, DONE_IT);
SIGNAL PRESENT_STATE, NEXT_STATE : State_type;
SIGNAL ADDRESS_CYCLE,ADRS_ENABLE,
                    READ_B, WRITE_B, READ_A,WRITE_A,
                    CHIP_SELECT ,AVERAGE_DONE,
                    ENABLE_L_S, ENABLE_R_S, LOAD_REG,TERMINAL_CNT, LOAD_REG_RCA, LOADED, CHANGED,
SEL, C_i, RCA_C_i, RCA_C_o, Clear_reg,
 EN_IN_AV, EN_OUT_AV, EN_DIV_AV, TO_SIGN_CONTROL, TO_LOADING_B :  STD_LOGIC;

SIGNAL ADRS, ADRS_TO_MEM : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL ADRS_SEL, S_MUX_B : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL MEM_DATA_A,MEM_DATA_B: SIGNED(7 DOWNTO 0);
SIGNAL MEM_DATA, OUT_RCA, RCA_IN_1, RCA_IN_2, S_REG_MUX_OUT : SIGNED(8 DOWNTO 0);
BEGIN
--FSM
STATE_TABLE_PROCESS : PROCESS(PRESENT_STATE, START, TERMINAL_CNT, TO_SIGN_CONTROL,
TO_LOADING_B)
          BEGIN
                    CASE PRESENT_STATE IS
                              WHEN IDLE => IF (START = '1') THEN NEXT_STATE <= LOADING_A;
                              ELSE NEXT_STATE <= IDLE;
                              END IF;
                              WHEN LOADING_A => IF (TERMINAL_CNT = '1') THEN NEXT_STATE <= FIRST_ADDEND;
                              ELSE NEXT_STATE <= LOADING_A;
                              END IF;
                              WHEN FIRST_ADDEND => IF (TO_LOADING_B = '1' )THEN NEXT_STATE <= LOADING_B;
                              ELSE NEXT_STATE <= SECOND_ADDEND;
                              END IF;
                              WHEN SECOND_ADDEND =>IF (TO_SIGN_CONTROL = '1') THEN NEXT_STATE <=
SIGN_CONTROL;
                                        ELSIF(TO_LOADING_B = '1') THEN NEXT_STATE <= LOADING_B;
                                        ELSE NEXT_STATE <= THIRD_ADDEND;
                                        END IF;
```

```
                        WHEN THIRD_ADDEND => IF (TO_SIGN_CONTROL = '1') THEN NEXT_STATE <=
SIGN_CONTROL;
                        ELSE NEXT_STATE <= LOADING_B;
                    END IF;
                        WHEN SIGN_CONTROL => NEXT_STATE <= LOADING_B;
                        WHEN LOADING_B => IF (TERMINAL_CNT = '1') THEN NEXT_STATE <= DONE_IT;
                        ELSE NEXT_STATE <= FIRST_ADDEND;
                        END IF;
                        WHEN DONE_IT  => IF (START = '0') THEN NEXT_STATE <= IDLE ;
                        ELSE NEXT_STATE <= DONE_IT;
                        END IF;
                END CASE;
        END PROCESS;

PROCESS (CLOCK_50, RESET)
BEGIN
        IF RESET = '0' THEN PRESENT_STATE <= IDLE;
        ELSIF (CLOCK_50'EVENT AND CLOCK_50='1') THEN
        PRESENT_STATE <= NEXT_STATE;
        END IF;
END PROCESS;

--CONTROL UNIT
STATE_ASSIGNEMENT_PROCESS : PROCESS (PRESENT_STATE, ADRS,ADRS(0), OUT_RCA)
        BEGIN
        --DEFAULT VALUE
        CHIP_SELECT <= '1';         ADRS_ENABLE<='0'; Clear_reg<='1';TO_LOADING_B <= '0';
        TO_SIGN_CONTROL <= '0';
        READ_B<='0';       WRITE_B<='1';      READ_A<='0';     WRITE_A<='1';
        ENABLE_R_S<='0';           ENABLE_L_S<='0';      LOAD_REG<='0';       LOAD_REG_RCA <='0';       C_i <= '0';
        EN_IN_AV<='0'; EN_DIV_AV <= '0'; EN_OUT_AV <= '0';
        CASE PRESENT_STATE IS
                WHEN IDLE =>  Clear_reg<='0';
                WHEN LOADING_A => WRITE_A<='0'; ADRS_ENABLE<='1'; ADRS_SEL <= "11";
                WHEN FIRST_ADDEND =>READ_A<='1'; SEL<='1'; ENABLE_R_S<='1'; LOAD_REG<='1';
LOAD_REG_RCA <='1'; ADRS_SEL <= "11" ; Clear_reg<='0'; EN_IN_AV<='1';
                 IF (ADRS = "0000000000" ) THEN TO_LOADING_B <= '1'; END IF;
                WHEN SECOND_ADDEND => READ_A <= '1'; SEL<='1'; LOAD_REG<='1'; LOAD_REG_RCA <='1';
ADRS_SEL <= "10";
                        IF (ADRS = "0000000001") THEN TO_SIGN_CONTROL <= '1';
                                ELSIF(ADRS = "0000000010") THEN TO_LOADING_B <= '1';
                                  ELSE TO_LOADING_B <= '0'; TO_SIGN_CONTROL <= '0';
                                END IF;
                WHEN THIRD_ADDEND => READ_A <= '1'; SEL<='1'; ENABLE_L_S<='1'; LOAD_REG<='1';
LOAD_REG_RCA <='1'; ADRS_SEL<="01";  C_i <= '1';
                                IF (ADRS(0) = '0') THEN TO_SIGN_CONTROL <= '1'; END IF;
                WHEN SIGN_CONTROL => SEL<='0'; LOAD_REG_RCA <='1'; LOAD_REG<='1';  C_i <= '1';
                WHEN LOADING_B => WRITE_B <= '0'; ADRS_SEL <= "11" ; ADRS_ENABLE<='1';  EN_OUT_AV <= '1';
                                IF (OUT_RCA(8 DOWNTO 7) = "10") THEN S_MUX_B <= "10";
                                 ELSIF (OUT_RCA(8 DOWNTO 7) = "01") THEN S_MUX_B <= "01";
                                  ELSE S_MUX_B <= "00";
                                 END IF;
                WHEN DONE_IT => EN_DIV_AV <= '1'; DONE <= '1';
                WHEN OTHERS =>  CHIP_SELECT <= '0';

        END CASE;
        END PROCESS;


--DATA PATH
ADRS_CNT  :  Adrs_Counter  PORT  MAP(CLOCK=>CLOCK_50,RESET=>RESET,ENABLE_COUNT=>ADRS_ENABLE,
COUNT => TERMINAL_CNT, ADRS=>ADRS);
```

ADRS_mux : Mux_3_to_1 PORT MAP(ADRS=>ADRS, SEL=>ADRS_SEL, U=>ADRS_TO_MEM);

MEM_A : Memory PORT MAP (DATA_IN =>DATA_INPUT, ADDRESS=>ADRS_TO_MEM, CS=>CHIP_SELECT, CLK=>CLOCK_50, WR=>WRITE_A, RD=>READ_A, DATA_OUT=>MEM_DATA_A);

MEM_DATA(8) <=MEM_DATA_A(7);
MEM_DATA(7 DOWNTO 0) <= MEM_DATA_A(7 DOWNTO 0);

REGIST : Shift_register PORT MAP (Data=>S_REG_MUX_OUT,CLOCK=>CLOCK_50, RESET=>START, ENABLE_LEFT=>ENABLE_L_S,ENABLE_RIGHT=>ENABLE_R_S, LOAD=>LOAD_REG , UPLOADED=>LOADED, T=>RCA_IN_1);

MUX : Mux2to1_9bit PORT MAP (IN_1=>MEM_DATA, SEL=>SEL, U=>S_REG_MUX_OUT);

C_in_REG : flipflop PORT MAP (D=>C_i, CLOCK=>CLOCK_50, Resetn=>START, Q=>RCA_C_i);

ADDER : RCA GENERIC MAP (P => 9)
PORT MAP (INPUT_1=>RCA_IN_1, INPUT_2=>RCA_IN_2, CARRY_IN=>RCA_C_i, CARRY_OUT=>RCA_C_o, SUM=>OUT_RCA);

REG_OUT_RCA : regn GENERIC MAP (N => 9)
PORT MAP (R=>OUT_RCA, CLOCK=>CLOCK_50, Resetn=>Clear_reg, LOAD=>LOAD_REG_RCA,Q=>RCA_IN_2);

MUX_TO_B : Memory_mux PORT MAP (IN_1=>OUT_RCA(7 DOWNTO 0), SEL=>S_MUX_B, U=>MEM_DATA_B);

MEM_B : Memory PORT MAP (DATA_IN=>MEM_DATA_B , ADDRESS=>ADRS, CS=>CHIP_SELECT, CLK=>CLOCK_50, WR=>WRITE_B, RD=>READ_B);

AVERAGE : Aritchmetic_mean PORT MAP (DATA=> MEM_DATA_A, CLOCK=>CLOCK_50, RESET=>START, ENABLE_IN=>EN_IN_AV, ENABLE_OUT=>EN_OUT_AV, ENABLE_DIV=>EN_DIV_AV, MEAN=>MEDIA);

END STRUCTURE;

Address_Counter

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Adrs_Counter IS
        PORT (CLOCK :  IN STD_LOGIC;
                       RESET, ENABLE_COUNT : IN STD_LOGIC;
                       COUNT : OUT STD_LOGIC;
                       ADRS : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
                       );
END Adrs_Counter;
ARCHITECTURE COUNTING OF Adrs_Counter IS
SIGNAL COUNT_VAR : INTEGER RANGE 0 TO 1024:=1;

        BEGIN
        COUNTING_PROCESS : PROCESS (CLOCK, RESET)
                BEGIN
                COUNT <= '0';
                IF RESET = '0' THEN COUNT_VAR <= 0;
                ELSIF (CLOCK'EVENT AND CLOCK='1') THEN
                        IF (ENABLE_COUNT = '1') THEN
                                COUNT_VAR <= COUNT_VAR +1;
                        IF (COUNT_VAR = 1023) THEN
                COUNT_VAR <= 0;
                END IF;
                END IF;
                END IF;
```

```
                IF (COUNT_VAR = 1023 ) THEN
                COUNT <= '1';
                END IF;
        END PROCESS;
        ADRS <= std_logic_vector(to_unsigned(COUNT_VAR, 10));

END COUNTING;
```

Average

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Adrs_Counter IS
        PORT (CLOCK :  IN STD_LOGIC;
                        RESET, ENABLE_COUNT : IN STD_LOGIC;
                        COUNT : OUT STD_LOGIC;
                        ADRS : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
                        );
END Adrs_Counter;
ARCHITECTURE COUNTING OF Adrs_Counter IS
SIGNAL COUNT_VAR : INTEGER RANGE 0 TO 1024:=1;

        BEGIN
        COUNTING_PROCESS : PROCESS (CLOCK, RESET)
                BEGIN
                COUNT <= '0';
                IF RESET = '0' THEN COUNT_VAR <= 0;
                ELSIF (CLOCK'EVENT AND CLOCK='1') THEN
                        IF (ENABLE_COUNT = '1') THEN
                                COUNT_VAR <= COUNT_VAR +1;
                        IF (COUNT_VAR = 1023) THEN
                COUNT_VAR <= 0;
                END IF;
                END IF;
                END IF;
                IF (COUNT_VAR = 1023 ) THEN
                COUNT <= '1';
                END IF;
        END PROCESS;
        ADRS <= std_logic_vector(to_unsigned(COUNT_VAR, 10));

END COUNTING;
```

FlipFlop

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY flipflop IS
PORT (D, Clock, Resetn : IN STD_LOGIC;
    Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
```

```
PROCESS (Clock, Resetn)
BEGIN
IF (Resetn = '0') THEN -- asynchronous clear
Q <= '0';
ELSIF (Clock'EVENT AND Clock = '1') THEN
Q <= D;
END IF;
END PROCESS;
END Behavior;
```

Full_Adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all;

ENTITY full_adder IS
        PORT(C_i : IN STD_LOGIC;
                A, B : IN STD_LOGIC ;
                S, C_o : OUT STD_LOGIC
                        );
END full_adder;

ARCHITECTURE Behavior OF full_adder IS
        SIGNAL xor_out : STD_LOGIC ;

        COMPONENT Mux2to1
                PORT (     In1 , In2, Sel : IN STD_LOGIC ;
                                        U : OUT STD_LOGIC
                        );
         END COMPONENT ;

         BEGIN
         xor_out <= A XOR B ;
         MUX : Mux2to1 PORT MAP(In1=>B, In2=>C_i, Sel=>xor_out, U=>C_o);
         S <= (C_i XOR xor_out) ;

 END Behavior;
```

Memory

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY MEMORY IS
PORT ( DATA_IN : IN SIGNED(7 DOWNTO 0);
                ADDRESS : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                CS, CLK, WR, RD :  IN STD_LOGIC;
                DATA_OUT : OUT SIGNED(7 DOWNTO 0)
                );
END MEMORY;


ARCHITECTURE BEHAVIOR OF MEMORY IS
TYPE MEM_STRUCTURE IS ARRAY(1023 DOWNTO 0) OF SIGNED(7 DOWNTO 0);
SIGNAL DATA : MEM_STRUCTURE;
BEGIN
--WRITE PROCESS
```

```
	MEM_WRITE : PROCESS (CLK, WR, RD, CS)
	BEGIN
		IF(clk'event and clk='1') THEN
			IF (CS = '1' AND NOT WR = '1' AND RD = '0') THEN
				DATA(TO_INTEGER(UNSIGNED(ADDRESS))) <= DATA_IN;
			END IF;
		END IF;
	END PROCESS;

--READ
			DATA_OUT <= DATA(TO_INTEGER(UNSIGNED(ADDRESS))) WHEN (CS = '1' AND
RD = '1' AND NOT WR = '0');

END BEHAVIOR;
```

Memory_Mux

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY Memory_mux IS
	PORT( IN_1: IN SIGNED(7 DOWNTO 0);
			SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			U : OUT SIGNED(7 DOWNTO 0)
		 );
END Memory_mux;

ARCHITECTURE BHV OF Memory_mux IS

BEGIN
MUX_OUT : PROCESS (SEL, IN_1)
BEGIN
	CASE SEL IS
		WHEN "01" => U <= "01111111";
		WHEN "10" => U <= "10000000";
		WHEN OTHERS => U <= IN_1;
	END CASE;
END PROCESS;
END BHV;
```

Mux_3_to_1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY Mux_3_to_1 IS
	PORT( ADRS: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
			SEL : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			U : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
		 );
END Mux_3_to_1;

ARCHITECTURE BHV OF Mux_3_to_1 IS
SIGNAL ADDRESS : INTEGER RANGE -3 TO 1023:=0;
BEGIN
```

```
MUX_OUT : PROCESS (SEL, ADRS)
BEGIN
	CASE SEL IS
		WHEN "11" => ADDRESS <= TO_INTEGER(UNSIGNED(ADRS));
		WHEN "10" => ADDRESS <= TO_INTEGER(UNSIGNED(ADRS)) -1;
		WHEN "01" => ADDRESS <= TO_INTEGER(UNSIGNED(ADRS)) -3;
		WHEN OTHERS => ADDRESS <= TO_INTEGER(UNSIGNED(ADRS));
	END CASE;
END PROCESS;
U <=STD_LOGIC_VECTOR(TO_UNSIGNED(ADDRESS, 10));
END BHV;
```

RCA

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY RCA IS
	GENERIC (P : INTEGER);
	PORT (INPUT_1, INPUT_2 : IN SIGNED(P-1 DOWNTO 0);
			CARRY_IN : IN STD_LOGIC;
			CARRY_OUT : OUT STD_LOGIC;
			SUM : OUT SIGNED(P-1 DOWNTO 0)
			);
END RCA;

ARCHITECTURE BHV OF RCA IS
	COMPONENT full_adder
			PORT(C_i : IN STD_LOGIC;
				A, B : IN STD_LOGIC ;
				S, C_o : OUT STD_LOGIC
				);
END COMPONENT;
SIGNAL XOR_INPUT : SIGNED(P-1 DOWNTO 0);
SIGNAL C_OUT : STD_LOGIC_VECTOR(P-1 DOWNTO 0);
BEGIN
XOR_INPUT(0) <= INPUT_2(0) XOR CARRY_IN;

	F_ADD_0 : full_adder PORT MAP(C_i=>CARRY_IN, A=>INPUT_1(0), B=>XOR_INPUT(0),
C_O=>C_OUT(0), S=>SUM(0));
RCA_CYCLE : FOR i IN 1 TO P-1 GENERATE
	XOR_INPUT(i) <= INPUT_2(i) XOR CARRY_IN;
	F_ADD : full_adder PORT MAP (C_i=>C_OUT(i-1), A=>INPUT_1(i), B=>XOR_INPUT(i),
C_o=>C_OUT(i), S=>SUM(i));
END GENERATE;
CARRY_OUT<=C_OUT(P-1);
END BHV;
```

REGn

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY regn IS
        GENERIC ( N : integer);
                PORT (R : IN SIGNED(N-1 DOWNTO 0);
                        Clock, Resetn, LOAD : IN STD_LOGIC;
                                Q : OUT SIGNED(N-1 DOWNTO 0)
                                );
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN

        PROCESS (Clock, Resetn)
BEGIN

        IF (Resetn = '0') THEN
                Q <= (OTHERS => '0');
        ELSIF (Clock'EVENT AND Clock = '1') THEN
                IF (LOAD = '1') THEN
                        Q <= R;
                END IF;
        END IF;
END PROCESS;
END Behavior;
```

Shift_Register

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Shift_register IS
                PORT ( Data : IN SIGNED(8 DOWNTO 0);
                        CLOCK, RESET, ENABLE_RIGHT, ENABLE_LEFT, LOAD: IN STD_LOGIC;
                        T : OUT SIGNED(8 DOWNTO 0)
                        );
END Shift_register;

ARCHITECTURE BEHAVIOR OF Shift_register IS
BEGIN
        PROCESS (RESET, CLOCK)
        BEGIN

        IF(RESET = '0') THEN
                T <= (OTHERS => '0');
         ELSIF(rising_edge(CLOCK)) THEN
                IF (LOAD = '1') THEN
                        IF  (ENABLE_left = '1' AND LOAD = '1' ) THEN
                                T(0) <= '0';
                                T(1) <= DATA(0);
                                T(2) <= DATA(1);
                                T(3) <= DATA(2);
                                T(4) <= DATA(3);
```

```
                    T(5) <= DATA(4);
                    T(6) <= DATA(5);
                    T(7) <= DATA(6);
                    T(8) <= DATA(7);
                    UPLOADED <= '1';
                ELSIF (ENABLE_right = '1' AND LOAD = '1') THEN
                    T(0) <= DATA(2);
                    T(1) <= DATA(3);
                    T(2) <= DATA(4);
                    T(3) <= DATA(5);
                    T(4) <= DATA(6);
                    T(5) <= DATA(7);
                    T(6) <= DATA(8);
                    T(7) <= DATA(8);
                    T(8) <= DATA(8);
                 UPLOADED <= '1';
                ELSE
                T <= DATA;
                UPLOADED <= '1';
                 END IF;
            END IF;
        END IF;
        END PROCESS;
END BEHAVIOR;
```

TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY TESTBENCH IS
END TESTBENCH;

ARCHITECTURE BHV OF TESTBENCH IS
  COMPONENT Digital_Filter
   PORT ( CLOCK_50: IN STD_LOGIC;
                DATA_INPUT : IN SIGNED(7 DOWNTO 0);
                START, RESET: IN STD_LOGIC;
                DONE : OUT STD_LOGIC;
                MEDIA : OUT SIGNED(7 DOWNTO 0)
               );
END component;
SIGNAL DATA_IN, DATA_OUT : SIGNED(7 DOWNTO 0);
SIGNAL CLOCK,START,RESET : STD_LOGIC;
SIGNAL ADRS : STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
 U_UT    :    Digital_Filter    PORT    MAP(CLOCK_50=>CLOCK,START=>START,RESET=>RESET,
DATA_INPUT=>DATA_IN);
  PROCESS
  BEGIN
    CLOCK <= '0', '1' AFTER 10 ns;
    WAIT FOR 20 ns;
END PROCESS;
```

```
START_PROCESS: PROCESS
BEGIN
  START <= '0';
  WAIT FOR 30 ns;
  START<='1';
  WAIT FOR 1000 ns ;
END PROCESS;

RESET_PROCESS: PROCESS
BEGIN
  RESET<='1';
  WAIT FOR 10 ns;
  RESET <='0';
  WAIT FOR 20 ns;
  RESET <= '1' ;
  WAIT FOR 900000 ns;
END PROCESS;

  DATA_IN_PROCESS: PROCESS  -- (-1)^n { 0,25*x(n) + x(n-1) -2*x(n-3) }
BEGIN
    DATA_IN <= "00000110";
    WAIT FOR 80 ns;
    DATA_IN <= "00000001";
    WAIT FOR 20 ns;
    DATA_IN <= "00001100";
    WAIT FOR 20 ns;
    DATA_IN <= "00000111";
    WAIT FOR 20 ns;
    DATA_IN <= "10000000";
    WAIT FOR 20 ns;
    DATA_IN <= "10000000";
    WAIT FOR 20 ns;
    DATA_IN <= "10000000";
    WAIT FOR 20 ns;
    DATA_IN <= "10000000";
    WAIT FOR 20 ns;
    DATA_IN <= "10000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;
    DATA_IN <= "01000000";
    WAIT FOR 20 ns;

END PROCESS  ;


END BHV;
```