Politecnico di Torino

Elettronica dei sistemi digitali
LAB 06
Algorithmic State Machine
Hardware-based Cryptographic Protocol

Gruppo C03

Iaconeta Andrea
Giura Andrea
Valvo Edmondo

August 6, 2022

Index

1 Design procedure

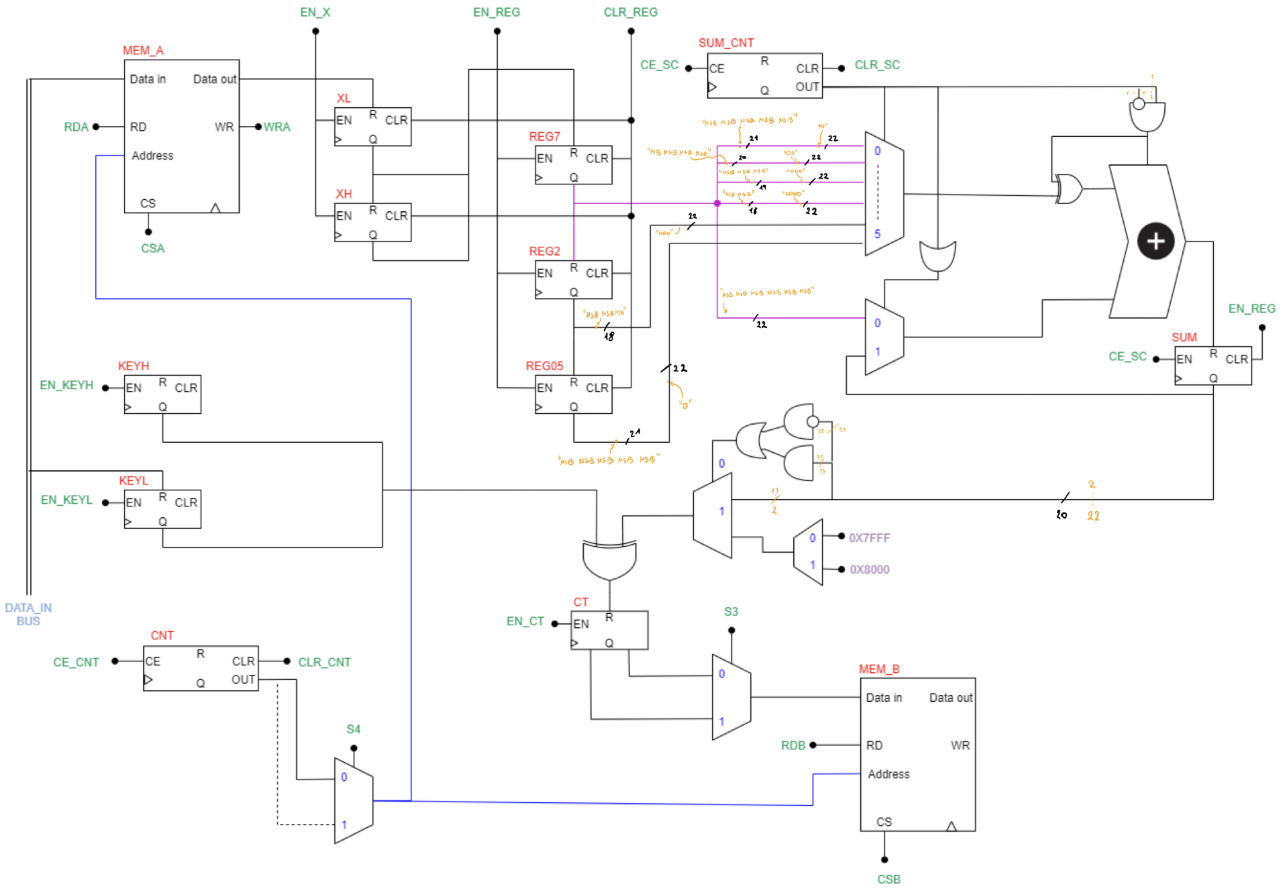
L'obiettivo del progetto è realizzare un circuito logico che processa dati audio ottenuti in ingresso da un convertitore analogico digitale. In una prima fase di acquisizione il dispositivo salva i byte all'interno di una memoria con parallelismo a 8 bit, i dati devono essere leggibili in formato Big Endian. In una seconda fase, invece, gli stessi dati vengono recuperati dalla memoria e criptati attraverso un filtro FIR che utilizza un semplice algoritmo di codifica basato sull'xor dei dati con una chiave, anch'essa letta dall'esterno durante il salvataggio dei dati. I dati criptati vengono salvati in un'altra memoria, questa volta in formato Little Endian, una volta terminata la conversione la macchina attende l'handshake con il mondo esterno ed è quindi pronta a ripartire.

Il progetto dell'Algorithmic State Machine (ASM) si suddivide in più parti, in primis è stato definito lo pseudocodice per delineare le operazioni che il dispositivo deve eseguire.

```
//carico memA e resetto registri
for i in 0 to 1023{
memA(i)=datain;
reg7=0;
reg2=0;
reg05=0;
}


//carico la key
regkeyH=datain;
regkeyL=datain;

for(i=0;i<511;i++){
        x(i)=memA(2*i) & memA(2*i+1);
        reg7=x(i)  reg2=reg7  reg05=reg2;//pipeline registri

        //calcolo CT(i) parall=22
        PT(i)=7.75*reg7-2*reg2+0.5*reg05;

        if(PT(i)>max){
                PT(i)=0x7FFF;
        }else if(PT(i)<min){
                PT(i)=0x8000;
        }else{
                PT(i)=to_16bit(PT(i));
        }

        CT(i)=PT(i) XOR (regkeyH & regkeyL);

        MemB(2*i)=CT(7 downto 0);
        MemB(2*i+1)=CT(15 downto 8);

}
```

Successivamente sono stati progettati il datapath e l'unità di controllo seguendo le specifiche di progetto, nei paragrafi seguenti è spiegato nel dettaglio il funzionamento e l'interazione tra le due parti.

2 Datapath



2.1 Top Entity

Dopo aver realizzato la descrizione VHDL delle varie sezioni, è stata creata una top entity per assemblare l'intero progetto, di seguito si riporta il file che comprende tutte le sotto entity del datapath e dell'unità di controllo.

2.1.1 VHDL Code

FIR_top_level.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity FIR_top_level is
port(top_start,top_clock,top_reset:in std_logic;
                top_datain:signed(7 downto 0);
                done:out std_logic);

end FIR_top_level;


architecture logica of FIR_top_level is
--inclusione dei vari componenti
        component memA is
                port(datain:in signed(7 downto 0);
                                indirizzo:in std_logic_vector(9 downto 0);
                                cs,clk,wr,rd: in std_logic;
                                dataout:out signed(7 downto 0));
```

```
end component;
--blocco di concatenazione X
component X is
        port(DOA: in signed(7 downto 0);
                clk,clr,en_x:in std_logic;
                Xtot:out signed(15 downto 0));
end component;
--pipeline dei registri
component blocco_reg is
        port(X_tot:in signed (15 downto 0);
                clr_reg,clk,en_reg:in std_logic;
                reg7q,reg2q,reg05q:out signed(15 downto 0));
end component;
--mux in ingresso al sommatore
component blocco_mux is
        port(s1:in std_logic_vector(2 downto 0);
                q7,q2,q05:in signed(15 downto 0);
                feedback:in signed(21 downto 0);
                parz,B:out signed(21 downto 0));
end component;
--adder
component bit21_add_sub is
        port(x21,y21:in signed(21 downto 0);
                counter_in:std_logic_vector(2 downto 0);
                PT:out signed(21 downto 0);
                carry_out:out std_logic);
end component;
--saturazione del PT
component satura_PT is
        port( PT : in signed ( 21 downto 0);
                PT_sat : out signed ( 15 downto 0));
end component;
--concatenazione della chiave
component key is
        port ( Data_in : in signed (7 downto 0);
                 clk : in std_logic;
                 clr_reg : in std_logic;
                 EN_H : in std_logic;
                 EN_L : in std_logic;
                 Key_out : out signed(15 downto 0));
END component;
--calcolatore del ct con xor
component CT_calculator is
        port ( PT_sat : in signed (15 downto 0);
                 clk : in std_logic;
                 clr_ct : in std_logic;
                 EN_ct : in std_logic;
                 s3 : in std_logic;
                 key : in signed( 15 downto 0);
                 CT_out : out signed ( 7 downto 0));
end component;
--memoria B
component memB is
        port(datain:in signed(7 downto 0);
        indirizzo:in std_logic_vector(9 downto 0);
        cs,clk,wr,rd: in std_logic;
        dataout:out signed(7 downto 0));
end component;
--generazione dell'indirizzo a partire dal conteggio
component Address_make is
        port(CE_CNT,clk,CLR_CNT,flag,s4:in std_logic;
```

```
                address,cnt10:buffer std_logic_vector(9 downto 0));
        end component;
        --conteggio di somma
        component counter_3bits is
                port(CE_CNT,clk,CLR_CNT:in std_logic;
                        C_out:buffer std_logic_vector(2 downto 0));
        end component;
        --registro in uscita dal sommatore
        component regn is
                GENERIC ( N: integer:=8);
                PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                                Clock, Resetn, EN: IN STD_LOGIC;
                                Q: OUT std_logic_vector(N-1 DOWNTO 0));
        END component;



        component fsm is
                PORT ( clk_fsm : in std_logic;
                                rst,start : in std_logic;

                                cnt : in std_logic_vector (9 downto 0);
                                sum_cnt : in std_logic_vector (2 downto 0);

                                csa,wra,rda,en_x,clr_reg,en_reg,clr_sc,ce_sc,en_ct,
                                en_keyh,en_keyl,clr_cnt,ce_cnt,s3,s4,csb,wrb,done_out : out std_logic;

                                flag_OUT : out std_logic
                  );
        END component;

        --dichiarazione dei segnali intermedi
        signal cnt_fsm:std_logic_vector(9 downto 0);
        signal sum_cnt_fsm:std_logic_vector(2 downto 0);
        signal csa_fsm,wra_fsm,rda_fsm,en_x_fsm,clr_reg_fsm,en_reg_fsm,
                clr_sc_fsm,ce_sc_fsm,en_ct_fsm,en_keyh_fsm,en_keyl_fsm,
                clr_cnt_fsm,ce_cnt_fsm,s3_fsm,s4_fsm,csb_fsm,wrb_fsm,
                done_out_fsm,flag_OUT_fsm :  std_logic;


        signal mid_indirizzo:std_logic_vector(9 downto 0);

        signal doa2,ct_hl,dob:signed (7 downto 0);

        signal Xtot_blocco_reg,reg7q_mid,reg2q_mid,reg05q_mid,
                        PT_sat_mid,key_out_mid: signed(15 downto 0);

        signal PT_mid,parz_mid,B_mid,PTQ:signed(21 downto 0);

        signal carryout:std_logic;


begin



fsm_map: fsm port map (
        top_clock,top_reset,top_start,

        cnt_fsm,sum_cnt_fsm,
```

```
        csa_fsm,wra_fsm,rda_fsm,en_x_fsm,clr_reg_fsm,en_reg_fsm,
        clr_sc_fsm,ce_sc_fsm,en_ct_fsm,en_keyh_fsm,en_keyl_fsm,
        clr_cnt_fsm,ce_cnt_fsm,s3_fsm,s4_fsm,csb_fsm,wrb_fsm,
        done_out_fsm,flag_OUT_fsm);

memA_map: memA port map(datain=>top_datain,indirizzo=>mid_indirizzo,
        cs=>csa_fsm,clk=>top_clock,wr=>NOT(wra_fsm),rd=>rda_fsm,dataout=>doa2);

X_map:X port map(DOA=>doa2,clk=>top_clock,clr=>clr_reg_fsm,
                en_x=>en_x_fsm,Xtot=>Xtot_blocco_reg);

blocco_reg_map:blocco_reg port map(X_tot=>Xtot_blocco_reg,clr_reg=>clr_reg_fsm,clk=>top_clock,
                en_reg=>en_reg_fsm,reg7q=>reg7q_mid,reg2q=>reg2q_mid,reg05q=>reg05q_mid);


blocco_mux_map: blocco_mux port map(s1=>sum_cnt_fsm,q7=>reg7q_mid,q2=>reg2q_mid,q05=>reg05q_mid,
                        feedback=>PTQ,parz=>parz_mid,B=>B_mid);

--viene inserito un registro tra uscita del sommatore e ingresso 'feedback' del secondo mux
regn_map:regn generic map(N=>22)
        port map(R=>std_logic_vector(PT_mid),Clock=>top_clock,
                Resetn=>clr_reg_fsm,EN=>ce_sc_fsm,signed(Q)=>PTQ);

bit21_add_sub_map: bit21_add_sub port map(x21=>parz_mid,y21=>B_mid,counter_in=>sum_cnt_fsm,
                        PT=>PT_mid, carry_out=>carryout);

satura_PT_map: satura_PT port map(PT=>PT_mid,PT_sat=>PT_sat_mid);

key_map: key port map(Data_in=>top_datain,clk=>top_clock,clr_reg=>clr_reg_fsm,
                        EN_H=>en_keyh_fsm,EN_L=>en_keyl_fsm,Key_out=>key_out_mid);

CT_calculator_map:CT_calculator port map(PT_sat=>PT_sat_mid,clk=>top_clock,clr_ct=>clr_reg_fsm,
                        EN_ct=>en_ct_fsm,s3=>s3_fsm,key=>key_out_mid,CT_out=>ct_hl);

memB_map:memB port map(datain=>ct_hl,indirizzo=>mid_indirizzo,cs=>csb_fsm,clk=>top_clock,
                        wr=>NOT(wrb_fsm),rd=>'0',dataout=>dob);

Address_make_map:Address_make port map(CE_CNT=>ce_cnt_fsm,clk=>top_clock,clr_cnt=>clr_cnt_fsm,
                        flag=>flag_OUT_fsm,s4=>s4_fsm,address=>mid_indirizzo,cnt10=>cnt_fsm);

counter_3bits_map: counter_3bits port map(CE_CNT=>ce_sc_fsm,clk=>top_clock,
                        clr_cnt=>clr_sc_fsm,C_out=>sum_cnt_fsm);

end architecture;
```

2.2 Key registers

Per eseguire la concatenazione della chiave, in quanto il dato in ingresso è su 8 bit, per la rappresentazione Big Endian, viene prima salvata la parte MSB e poi la parte LSB. I segnali di enable rispettivamente EN_L e EN_H, vengono forniti dall'unità di controllo.
Il segeunte blocco implementa la concatenazione:

2.2.1 VHDL Code

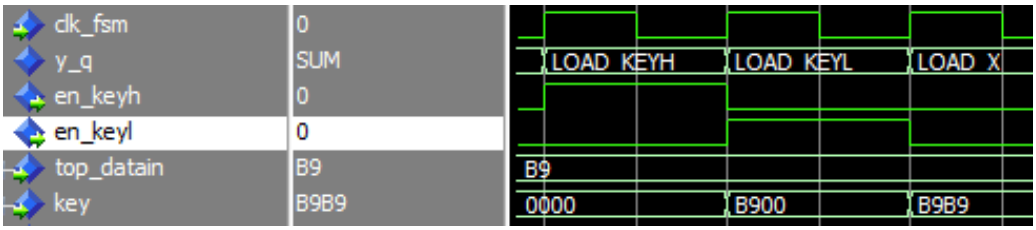
Key.vhd

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

```
--blocco per la concatenazione della Key
ENTITY key IS
        port ( Data_in : in signed (7 downto 0);
                clk : in std_logic;
                clr_reg : in std_logic;
                EN_H : in std_logic;
                EN_L : in std_logic;
                Key_out : out signed(15 downto 0));
END key;

ARCHITECTURE struct OF key IS
        component regn
                GENERIC ( N: integer:=7);
                PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                        Clock, Resetn, EN: IN STD_LOGIC;
                        Q: OUT std_logic_vector(N-1 DOWNTO 0));
        end component;

        BEGIN
                --rappresentazione key --> Big Endian
                --MSB-LSB

                --registro per i bit meno significativi
                KEYL: regn generic map (N => 8)
                      port map ( std_logic_vector(Data_in), clk, clr_reg, EN_L,
                        signed(Q)=>Key_out(7 downto 0));

                --registro per i bit pi significativi
                KEYH: regn generic map (N => 8)
                      port map (std_logic_vector(Data_in), clk, clr_reg, EN_H,
                        signed(Q)=>Key_out(15 downto 8));

end struct;
```

2.2.2 Waves

Di seguito è riportato un frammento della simulazione Modelsim riguardante il caricamento della chiave, per semplicità si è utilizzato lo stesso valore in datain, sia per la parte LSB che MSB:



2.3 Load A and Address

Nella fase di caricamento della memoria A, viene utilizzato il "cnt10" ovvero un contatore a 10 bit, da 0 a 1023 per indirizzare l'intera memoria. Per la lettura dei valori X e la scrittura della memoria B, invece, lo stesso contatore viene utilizzato con $0 <= cnt <= 511$:

$$x[cnt] = x[2 * cnt] \ \ \& \ \ x[2 * cnt + 1] \tag{1}$$

si utilizza il c_int, formato dai 9 bit meno significativi del contatore + il not flag. Ovvero si moltiplica per 2(shift a dx) e poi si concatena il not flag.

Il blocco che implementa tale operazione è il seguente:

2.3.1 VHDL Code

Address_make.vhd

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

--blocco utilizzato per "creare" l'indirizzo
entity Address_make is
        port(CE_CNT,clk,CLR_CNT,flag,s4:in std_logic;
                address,cnt10:buffer std_logic_vector(9 downto 0));

end Address_make;

architecture struct of Address_make is

        component counter_10bits is
                port(CE_CNT,clk,CLR_CNT:in std_logic;
                        C_out:buffer std_logic_vector(9 downto 0));
        end component;

        component bitN_mux2to1 is
                  port(x8, y8: in std_logic_vector (9 downto 0);
                               s8: in std_logic;
                               m8: out std_logic_vector (9 downto 0));
        end component;

        signal  c_int : std_logic_vector(9 downto 0);

        begin
                --contatore a 10 bit
                --da 0 a 1023
                C1: counter_10bits port map (CE_CNT,clk,CLR_CNT,cnt10);

                c_int <= cnt10(8 downto 0) & not flag;

                M1: bitN_mux2to1 port map(cnt10, c_int, s4, address);


end architecture;
```

Di seguito si riporta un estratto della struttura della memoria A, si rimanda alla sezione 5 per visualizzare tutti i codici VHDL utilizzati:

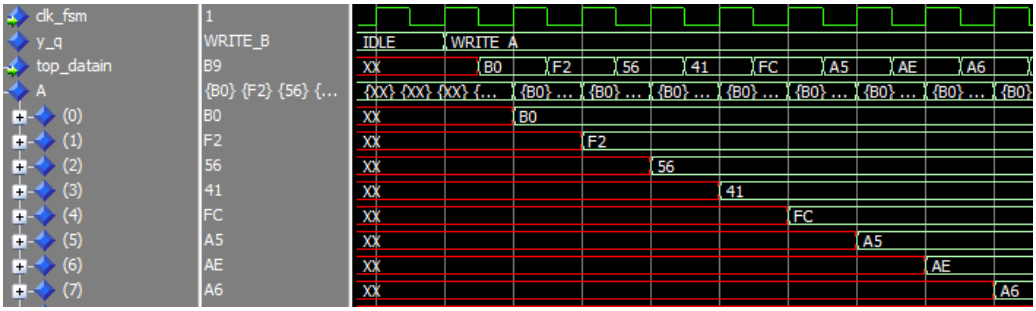
memA.vhd

```
entity memA is
        port(datain:in signed(7 downto 0);
                indirizzo:in std_logic_vector(9 downto 0);
                cs,clk,wr,rd: in std_logic;
                dataout:out signed(7 downto 0));
end memA;

architecture logica of memA is
--definizione  della memoria come array
--di parole dove ciascuna parola n byte di tipo signed
type matrice is array(0 to 1023) of signed(7 downto 0);

signal A:matrice;

begin
        process(clk)
                begin
                --scrive solo se gli appropriati segnali di controllo vengono assegnati
```

```
                        if(clk' event and clk='1' and wr='0' and cs='1') then
                        A(to_integer(unsigned(indirizzo)))<=datain;

                --legge solo se gli appropriati segnali di controllo vengono assegnati
                        elsif(clk' event and clk='1' and rd='1' and cs='1') then
                        dataout<=A(to_integer(unsigned(indirizzo)));
                        end if;
        end process;
end architecture;
```

2.3.2 Waves

Si mostra un frammento della simulazione Modelsim riguardante il caricamento dei dati nella memoria A:



2.4 Read A - Big endian

Essendo la lettura sincrona con il clock, si sono utilizzati due registri per effettuare la concatenazione di X come mostra la figura 2.4.2. Dato che ci vogliono due colpi di clock per estrarre l'intera X si esegue la seguente procedura:

- Si inizializzano (in stati precedenti) i due registri a 0.
- Al primo colpo di clock si carica il primo dato,che corrisponde al most significant byte, nel primo registro e il dato (0) che si trovava precedentemente in tale registro viene passato al secondo registro.
- Al secondo colpo di clock il most significant byte viene inviato al secondo registro e sul primo si carica il least significant byte.

2.4.1 VHDL Code

Per una migliore leggibilità del report verranno mostrati i file più importanti e/o alcuni frammenti di codice, per una visione integrale fare riferimento alla sezione 5.

X.vhd

```
        entity X is
        port(DOA: in signed(7 downto 0);
                        clk,clr,en_x:in std_logic;
                        Xtot:out signed(15 downto 0));
        end X;

        --definizione dei due segnali di uscita, uno per
        --il least significant byte e l'altro per il most significant byte
        signal xlq:std_logic_vector(7 downto 0);
        signal xhq:std_logic_vector(7 downto 0);

        begin

                --i due registri che contengono le due parti di X
                reg1:regn port map(R=>std_logic_vector(DOA),Clock=>clk,
                        Resetn=>clr,EN=>en_x,Q=>xlq);
```

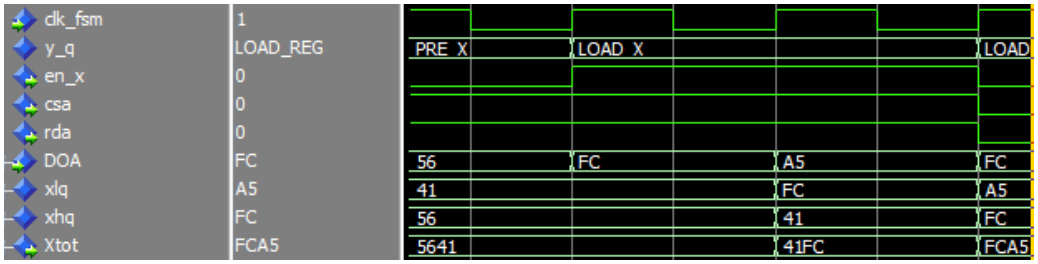
```
reg2:regn port map(R=>std_logic_vector(xlq),Clock=>clk,
        Resetn=>clr,EN=>en_x,Q=>xhq);

--concatenazione per ottenere il segnale X completo
--su 16 bit in formato big endian
Xtot<=signed(xhq & xlq);
```

blocco_reg.vhd

```
entity blocco_reg is
port(X_tot:in signed (15 downto 0);
                clr_reg,clk,en_reg:in std_logic;
                reg7q,reg2q,reg05q:out signed(15 downto 0));
end blocco_reg;

signal q7,q2,q05:std_logic_vector (15 downto 0);

--in questo blocco si hanno tre registri che contengono
--i tre componenti necessari per il calcolo di PT

--ciascun registro ha la sua uscita che verrmanipolata
-- poi nel blocco antecedente il sommatore
begin

        reg1:regn port map(R=>std_logic_vector(X_tot),Clock=>clk,Resetn=>clr_reg,
                        EN=>en_reg,Q=>q7);
        reg7q<=signed(q7);

        reg2:regn port map(R=>q7,Clock=>clk,Resetn=>clr_reg,EN=>en_reg,Q=>q2);
        reg2q<=signed(q2);

        reg3:regn port map(R=>q2,Clock=>clk,Resetn=>clr_reg,EN=>en_reg,Q=>q05);
        reg05q<=signed(q05);
```

2.4.2 Waves

In figura, viene mostrata la lettura, il caricamento e la concatenazione di X nei registri xh e xl:



2.4.2

2.5 PT computation

Il calcolo di PT corrisponde a:

$$PT[i] = 7.75 \cdot x[i] - 2 \cdot x[i-1] + 0.5 \cdot x[i-2] \tag{2}$$

Tuttavia per facilità di esecuzione si è deciso di eseguire l'espressione equivalente:

$$PT[i] = \frac{1}{4} \cdot (31 \cdot x[i] - 8 \cdot x[i-1] + 2 \cdot x[i-2]) \tag{3}$$

Risulta fondamentale notare che 31 corrisponde a 11111 in binario, questo implica che una prodotto con tale valore equivale alla somma di 5 versioni shiftate del segnale da moltiplicare, per evitare overflow si è scelto di lavorare con un **parallelismo a 22 bit**.
Per effettuare tale calcolo si è deciso di porre di fronte al sommatore due multiplexer, uno a sei vie e uno a due vie.
Il mux 2 to 1 ha come ingressi:

- la prima versione del segnale x(i), ovvero quella moltiplicata per 00001 ed estesa su 22 bit.
- il segnale di "feedback" del sommatore che corrisponde al parziale della somma che incrementa ad ogni iterazione.

Mentre invece gli ingressi del mux 6 to 1 sono:

- la seconda versione del segnale x(i), ovvero quella moltiplicata per 00010 ed estesa su 22 bit.
- la terza versione del segnale x(i), ovvero quella moltiplicata per 00100 ed estesa su 22 bit.
- la quarta versione del segnale x(i), ovvero quella moltiplicata per 01000 ed estesa su 22 bit.
- la quinta versione del segnale x(i), ovvero quella moltiplicata per 10000 ed estesa su 22 bit.
- il segnale x(i-1) moltiplicato per 8 (shiftato a sinistra di 3) ed esteso su 22 bit
- il segnale x(i-2) moltiplicato per 2 (shiftato a sinistra di 1) ed esteso su 22 bit

Il segnale di controllo del mux a sei vie corrisponde all'uscita del contatore mentre invece quello del secondo mux deve essere sempre 1 ad eccezione della prima "iterazione" in cui il contatore è a 0. Di conseguenza quest'ultimo mux viene pilotato da un segnale ottenuto come OR di ciascun bit del segnale in arrivo dal contatore:

$$s = cnt(2) + cnt(1) + cnt(0) \tag{4}$$

Tramite questa combinazione dei mux si ottengono correttamente gli addendi ad ogni colpo di clock.
Al primo addendo (quello in uscita dal mux a sei vie) viene effettuata una bitwise XOR con un segnale i cui bit sono tutti e 22 equivalenti al segnale SUM. Tale segnale deve essere a 0 in caso di somma e 1 in caso di sottrazione che si ha solo quando il contatore raggiunge 4, proprio per questo si ottiene tramite:

$$SUM = [NOT(cnt(0))] \cdot [NOT(cnt(1))] \cdot [cnt(2)] \tag{5}$$

A questo punto basta aggiungere come carry in il valore del segnale SUM di modo che in caso di sottrazione il cambio di segno del primo addendo venga effettuato correttamente.
Infine si aggiunge un registro all'uscita del sommatore la cui uscita va sia al mux precedentemente descritto che al blocco responsabile della saturazione di PT.

2.5.1 VHDL Code

In sequenza vengono mostrate le implementazioni del blocco contenente i mux antecedenti il sommatore e dell'adder che esegue sottrazioni e somme necessarie al calcolo di PT:

blocco_mux.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity blocco_mux is
        port(s1:in std_logic_vector(2 downto 0);
                q7,q2,q05:in signed(15 downto 0);
                feedback:in signed(21 downto 0);
                parz,B:out signed(21 downto 0));
end blocco_mux;

architecture logica of blocco_mux is

        component bit21_mux6to1 is
                port(u21,v21,z21,w21,t21,k21:in std_logic_vector(21 downto 0);
                        s21:in std_logic_vector(2 downto 0);
                        o21:out std_logic_vector(21 downto 0));
        end component;

        component bit21_mux2to1 is
                port(x21,y21:in std_logic_vector(21 downto 0);
```

```
                        s21:in std_logic;
                        o21:out std_logic_vector(21 downto 0));
        end component;

        signal in1,in2,in3,in4,in5,in6,in7:signed(21 downto 0);
        signal controllo2:std_logic;

        begin
                --si hanno 7 ingressi

                --i primi 5 sono i termini che
                -- una volta sommati corrispondo ad una moltiplicazione per 31
                in1<= q7(15)& q7(15)& q7(15) & q7(15) & q7(15) & q7(15) & q7;
                in2<= q7(15)& q7(15)& q7(15) & q7(15) & q7(15)  & q7 & '0';
                in3<=  q7(15)& q7(15) & q7(15) & q7(15)  & q7 & '0' & '0';
                in4<=  q7(15)& q7(15) & q7(15)  & q7 & '0' & '0' & '0';
                in5<=   q7(15)& q7(15)  & q7 & '0' & '0' & '0' & '0';


                --il sesto termine che corrisponde a x(i-1)
                -- gioltiplicato per 8 ed esteso su 22 bit
                in6<= q2(15) & q2(15) & q2(15) & q2 & "000";

                -- il settimo termine che corrisponde a x(i-2)
                --moltiplicato per 2 ed esteso su 22 bit
                in7<= q05(15) & q05(15) & q05(15) & q05(15) & q05(15) & q05 & '0';

                mux1:bit21_mux6to1 port map(u21=>std_logic_vector(in2),v21=>std_logic_vector(in3),
                        z21=>std_logic_vector(in4),w21=>std_logic_vector(in5),
                        t21=>std_logic_vector(in6),k21=>std_logic_vector(in7),
                        s21=>s1,signed(o21)=>parz);

                --controllo 2 che a partire dalla seconda iterazione
                 --equivale sempre a 1, scegliendo il feedback che torna dal sommatore
                controllo2<=s1(0) OR s1(1) OR s1(2);
                mux2:bit21_mux2to1 port map(x21=>std_logic_vector(in1),y21=>std_logic_vector(feedback),
                        s21=>controllo2,signed(o21)=>B);


end architecture;
```

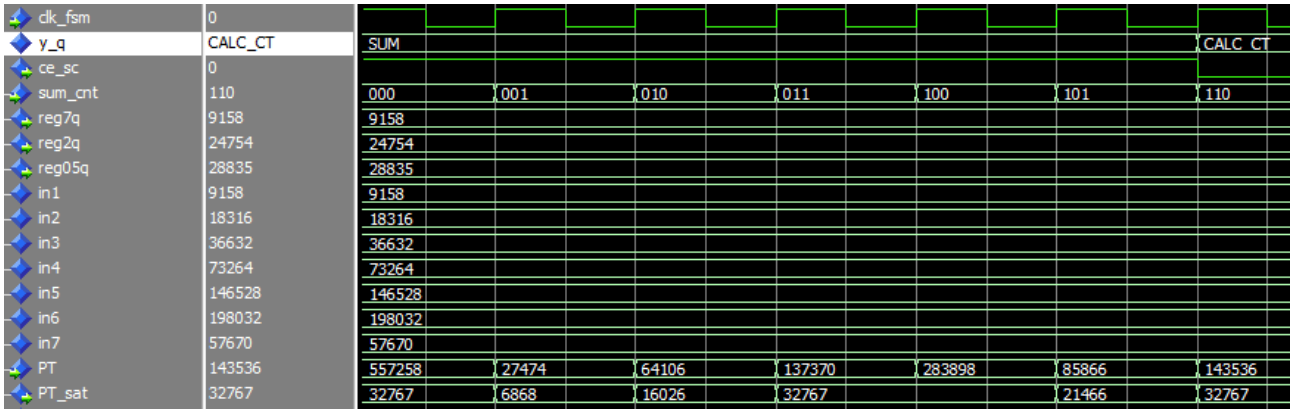
bit21_add_sub.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bit21_add_sub is
        port(x21,y21:in signed(21 downto 0);
                counter_in:std_logic_vector(2 downto 0);
                PT:out signed(21 downto 0);
                carry_out:out std_logic);
end bit21_add_sub;


architecture logica of bit21_add_sub is
        component bit21_adder is
                port(a21,b21: in SIGNED (21 downto 0);
                        cin:in std_logic;
                        cout: out std_logic;
                        s21:out SIGNED (21 downto 0));
        end component;
```

```
        signal vector: signed(21 downto 0);
        signal operazione:std_logic;
        signal x_mod:signed (21 downto 0);


        begin
                --in primis si ricava il segnale operazione
                --che andrd 1 solo quando il counter vale 4
                --ovvero quando si sottrae 8*X(i-1)
                operazione<=counter_in(2) AND (NOT counter_in(1)) AND (NOT counter_in(0));


                --tramite la xor si cambia ciascun bit se
                --operazione=1 e quindi si ha una sottrazione
                vector<=(others=>operazione);
                x_mod<=x21 XOR vector;

                --operazione poi serve da carry_in dato che
                --in caso di sottrazione al segnale cambiato va sommato 1
                somma:bit21_adder port map(a21=>x_mod,b21=>y21,cin=>operazione,
                        cout=>carry_out,s21=>PT);

end architecture;
```

2.5.2 Waves

In figura si riporta un intero ciclo di calcolo, sono presenti i 7 ingressi dei multiplexer e il PT parziale. L'ultima onda rappresenta il PT finale calcolato e saturato (vedi sezione 2.6).



2.6 PT Saturation and CT

Dato che il numero da memorizzare nella memoria B è su 16 bit, potrebbe avvenire saturazione durante le operazioni sul PT:

- Se PT[i] è maggiore del massimo valore positivo rappresentabile su 16 bit deve saturare a PT[i] = 0x7FFF.
- Se PT[i] è minore del minimo valore negativo rappresentabile su 16 bit deve saturare a PT[i] = 0x8000.

2.6.1 VHDL Code

Vengono mostrati i frammenti fondamentali che realizzano la saturazione e il calcolo di CT (vedi sez. 5 per codice integrale):

satura_PT.vhd

```
        entity satura_PT is
                port( PT : in signed ( 21 downto 0);
                        PT_sat : out signed ( 15 downto 0));
```

```
end satura_PT;

signal pt_max, pt_min, intern: signed(15 downto 0);
signal selector : std_logic;

begin
        --0x7fff
        --max numero positivo su 16 bit
        pt_max <= "0111111111111111";

        --0x8000
        --min numero negativo su 16 bit
        pt_min <= "1000000000000000";

        --se il bit di segno e' 1 (selettore del mux)
        --seleziona 0x8000
        --altrimenti seleziona 0x7fff
        M1: bitN_mux2to1 generic map ( N => 16)
                port map (std_logic_vector(pt_max), std_logic_vector(pt_min),
                        PT(21), signed(m8)=>intern);


        --selettore del 2 mux
        --e' 0 quando i bit sono uguali
        -- 1 altrimenti
        selector <= not (((not PT(21)) and (not PT(20)) and (not PT(19))
                         and (not PT(18)) and (not PT(17)))
                         or (PT(21) and PT(20) and PT(19) and PT(18) and PT(17)));

        --se selector = 1
        --seleziona il PT saturato
        --ovvero uscita del mux precedente
        --altrimenti seleziona i 16 bit del PT (uscita del sommatore)
        M2: bitN_mux2to1 generic map (N => 16)
                port map (std_logic_vector(PT(17 downto 2)), std_logic_vector(intern),
                        selector, signed(m8)=>PT_sat);
```

L'immagine che segue mostra la realizzazione della tecnica di criptaggio "One Time Pad", che è basata sull'XOR bit a bit tra il plaintext(PT) e la chiave:

$$CT[i] = PT[i] \bigoplus Key \tag{6}$$

CT_calculator.vhd

```
entity CT_calculator is
        port ( PT_sat : in signed (15 downto 0);
                clk : in std_logic;
                clr_ct : in std_logic;
                EN_ct : in std_logic;
                s3 : in std_logic;
                key : in signed( 15 downto 0);
                CT_out : out signed ( 7 downto 0));
end CT_calculator;

signal R, Qout : signed ( 15 downto 0);

begin
        --ingresso del registro
        --calcolo vero e proprio di CT
        R <= PT_sat xor key;
```

```
--registro utilizzato per salvare CT
R1: regn generic map ( N => 16)
        port map (R=> std_logic_vector(R), clock=>Clk, resetn=>clr_ct,
                EN=>EN_ct,signed(Q)=>Qout);

--mux utilizzato per "dividere" il CT
--se s3  --> parte LSB (ultimi 8 bit)
--se s3  --> parte MSB (primi 8 bit)
--tale blocco implementa la "conversione" da Big a Little Endian
M1: bitN_mux2to1 generic map (N => 8)
        port map (x8=>std_logic_vector(Qout(7 downto 0)),
                y8=>std_logic_vector(Qout(15 downto 8)),s8=>s3, signed(m8)=>CT_out);
```

2.7 Write B - Little endian

Al seguito del criptaggio di CT, la memorizzazione in memoria B deve avvenire nel formato Little Endian. Si è utilizzato un mux, con ingressi gli 8 bit LSB e gli 8 bit MSB; quando il selettore, che è collegato a flag, è 0 seleziona i bit meno significativi altrimenti quelli più significativi. Siccome flag è utilizzato anche in address, questo permette di scrivere in B prima la parte LSB e poi la parte MSB rispettando il formato Little Endian.

2.7.1 VHDL Code

Il seguente frammento di codice, mostra come è stata implementata tale parte, s3 = flag, che viene gestito dalla "control unit".

```
--mux utilizzato per "dividere" il CT
--se s3 è 0 --> parte LSB (ultimi 8 bit)
--se s3 è 1 --> parte MSB (primi 8 bit)
--tale blocco implementa la "conversione" da Big a Little Endian
M1: bitN_mux2to1 generic map (N => 8)
        port map (x8=>std_logic_vector(Qout(7 downto 0)),
                y8=>std_logic_vector(Qout(15 downto 8)),s8=>s3, signed(m8)=>CT_out);
```

Struttura della memoria B:

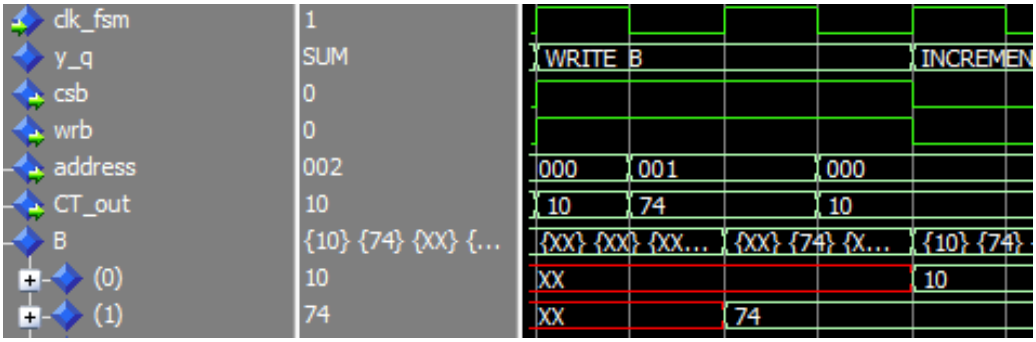
memB.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memB is
port(datain:in signed(7 downto 0);
                indirizzo:in std_logic_vector(9 downto 0);
                cs,clk,wr,rd: in std_logic;
                dataout:out signed(7 downto 0));
end memB;


architecture logica of memB is
--descrizione equivalente a memA
type matrice is array(0 to 1023) of signed(7 downto 0);

signal B:matrice;

begin
        process(clk)
                begin
```

```
                        if(clk' event and clk='1' and wr='0' and cs='1') then
                        B(to_integer(unsigned(indirizzo)))<=datain;
                        elsif(clk' event and clk='1' and rd='1' and cs='1') then
                        dataout<=B(to_integer(unsigned(indirizzo)));
                        end if;
        end process;
end architecture;
```

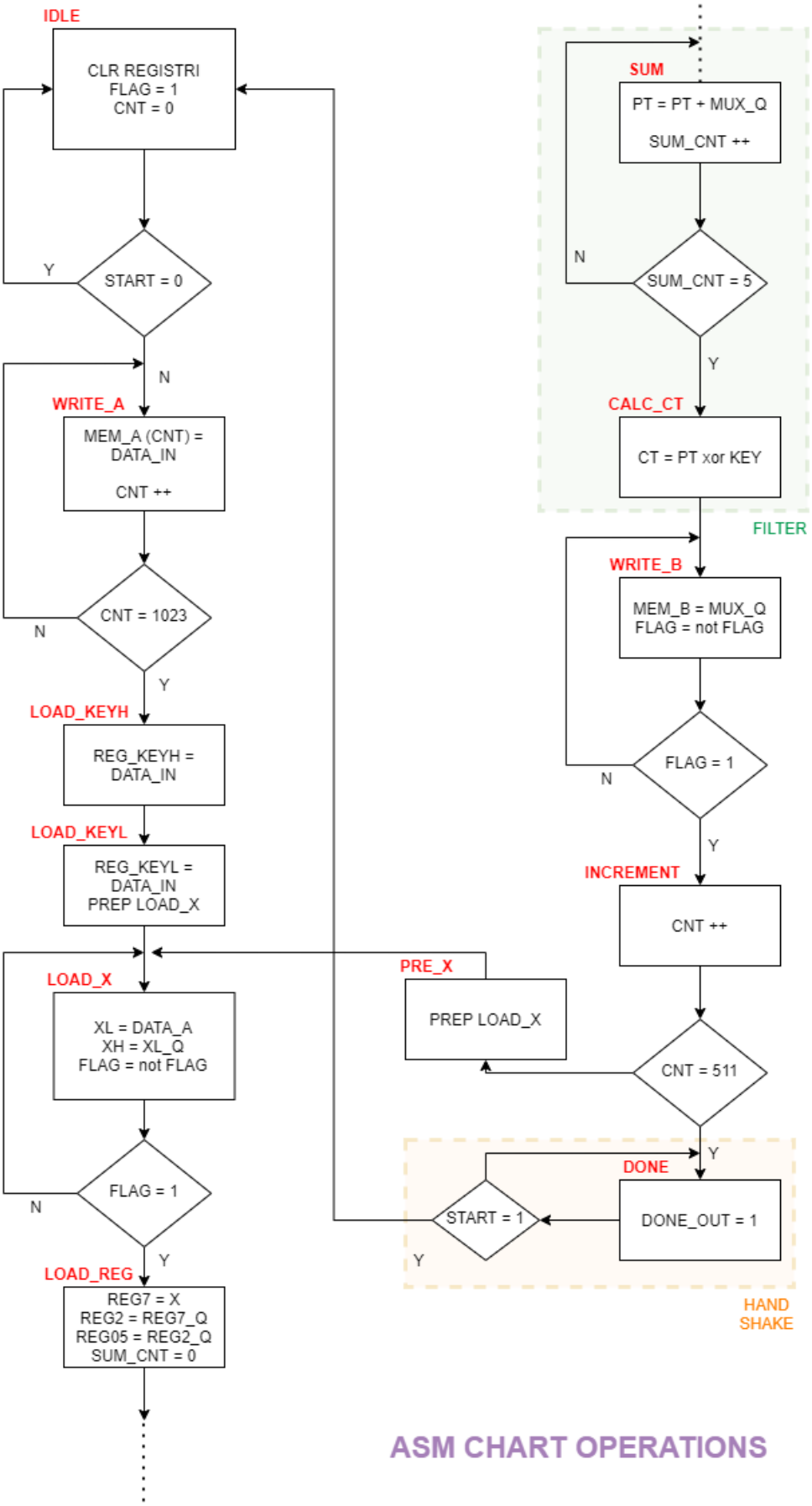
2.7.2 Waves

In figura è riportata la scrittura nella memoria B, si può vedere come il dispositivo salva prima il byte più significativo nella seconda locazione di memoria, e successivamente quello meno significativo nella prima.

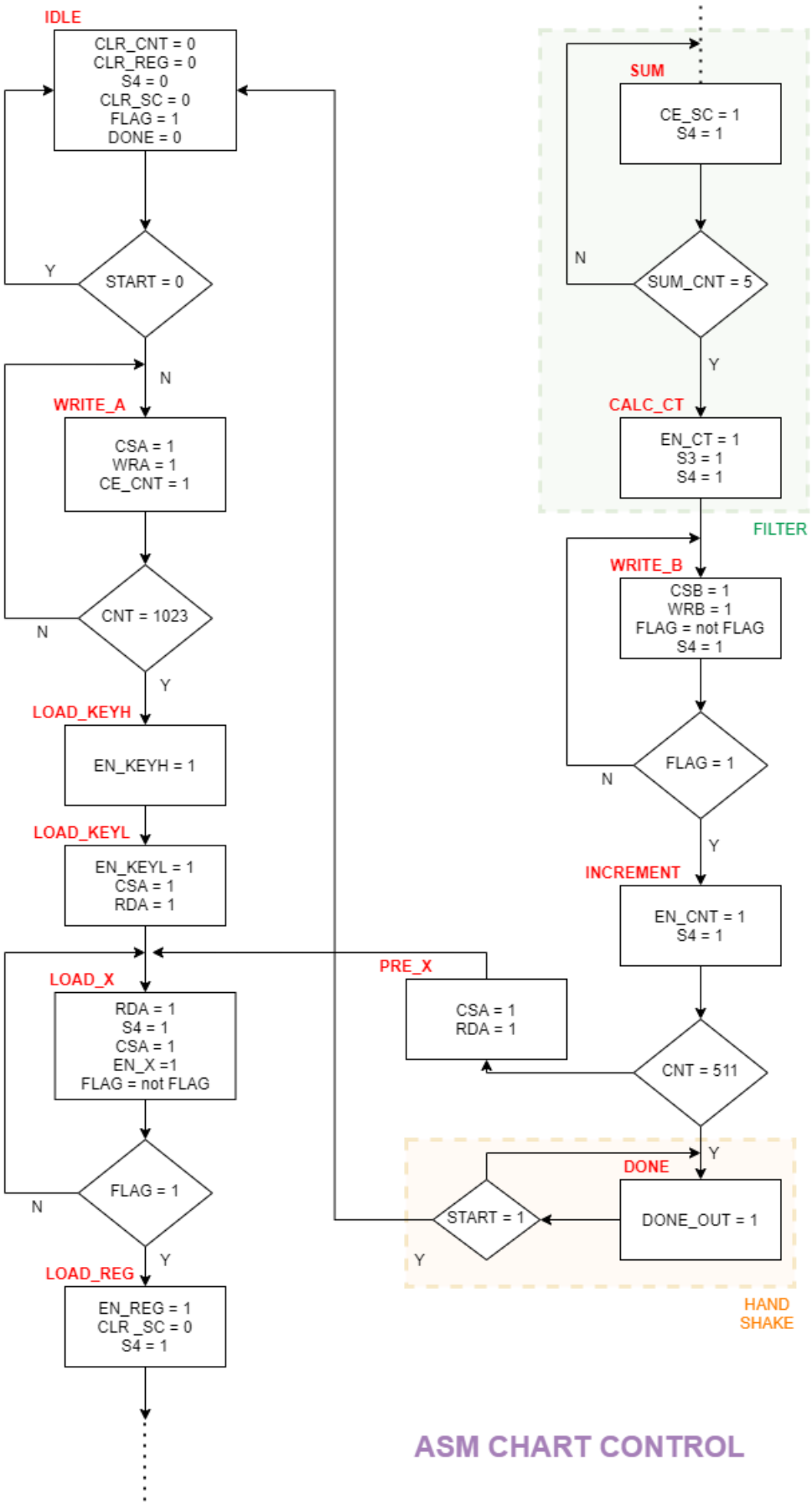


3 ASM chart

3.1 Operation chart



3.2 Control chart



3.3 State machine description

- IDLE

  Stato in cui la macchina attende l'attivazione del segnale di start dall'esterno, durante l'idle vengono eseguite tutte le operazioni preliminari come il reset dei contatori e dei registri, l'inizializzazione del flag e dei selettori dei multiplexer.

- WRITE_A

  Attiva la memoria A in modalità lettura per salvare i dati provenienti dal bus dataIn. Viene anche abilitato il contatore degli indirizzi in modo che ad ogni colpo di clock venga puntata la correta locazione di memoria in cui scrivere il byte in ingresso.

- LOAD_KEYH and LOAD_KEYL

  Similmente allo stato precedente in entrambi gli stati viene abilitato rispettivamente il registro KEYH e il registro KEYL. Lo stato LOAD_KEYL sostituisce lo stato PRE_X nel primo ciclo di calcolo, in pratica oltre a salvare il byte meno significativo della chiave, provvede ad abilitare in anticipo la lettura dalla memoria A in modo che i dati siano pronti in tempo per il prossimo stato.

- LOAD_X

  Mantiene attiva la lettura dalla memoria A e salva nel registro XL il dato proveniente dalla locazione puntata dall'indirizzo. Tramite il controllo del flag lo stato viene eseguito 2 volte: la prima volta carica il byte più significativo, la seconda volta esegue lo shift del byte appena salvato spostandolo in XH e carica il byte meno significativo in XL.

- LOAD_REG

  Abilita i 3 registri in pipeline caricando nel primo il valore di X caricato nello stato precedente.

- SUM

  Attiva il contatore di somma e il registro SUM, la macchina non cambia stato fino a quando il contatore non raggiunge il valore 5 in modo che si possa compiere l'intero ciclo di somma.

- CALC_CT

  Viene abilitato il registro CT e quindi salvata l'xor tra la chiave e il risultato del FIR.

- WRITE_B and INCREMENT

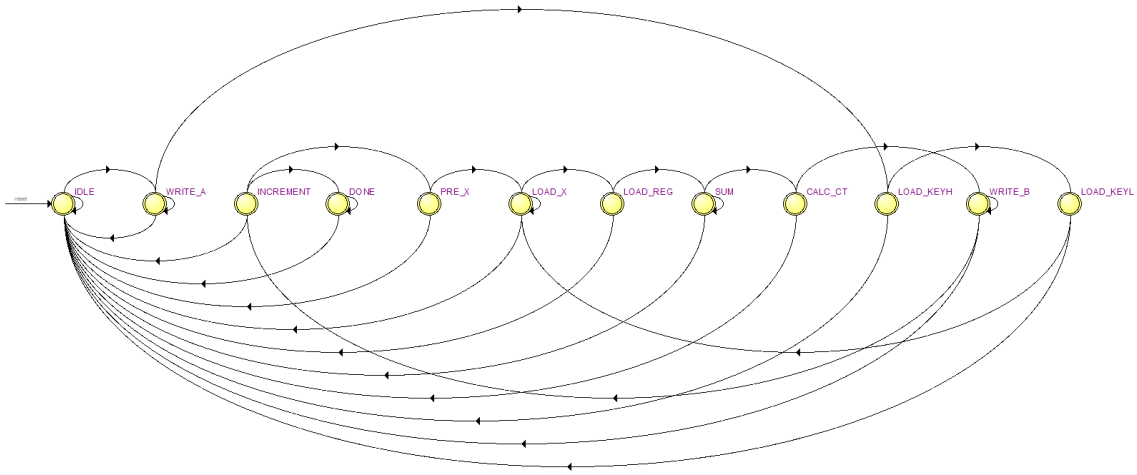
  Attiva la scrittura della memoria B e salva il dato nella locazione puntata dall'indirizzo. Tramite il controllo del flag lo stato viene eseguito 2 volte: la prima volta carica il byte meno significativo, la seconda, grazie al controllo del multiplexer carica quello più significativo.

- PRE_X

  Abilita in anticipo la lettura dalla memoria A in modo che i dati siano pronti in tempo per il prossimo stato.

- DONE

  Si occupa di sincronizzare il lavoro tra mondo esterno e ASM tramite il protocollo handshake, quindi setta il DONE a 1 e attende che il controllo riporti il segnale di start a 0 prima di tornare nello stato di IDLE.



fsm.vhd

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


ENTITY fsm IS
        PORT ( clk_fsm : in std_logic;
                rst,start : in std_logic;

                cnt : in std_logic_vector (9 downto 0);
                sum_cnt : in std_logic_vector (2 downto 0);

                csa,wra,rda,en_x,clr_reg,en_reg,clr_sc,ce_sc,en_ct,
                en_keyh,en_keyl,clr_cnt,ce_cnt,s3,s4,csb,wrb,done_out : out std_logic;

                flag_OUT : out std_logic);
END fsm;


ARCHITECTURE struct OF fsm IS

        TYPE State_type IS (IDLE,WRITE_A,LOAD_KEYH,LOAD_KEYL,LOAD_X,LOAD_REG,SUM,
                            CALC_CT,WRITE_B,INCREMENT,PRE_X,DONE);
        SIGNAL y_q, y_d : State_type := IDLE; -- y_Q is present state, y_D is next state

        signal flag : std_logic := '1';

        begin
        s3 <= not flag;
        flag_OUT <= flag;

                state_proc:process (y_q,rst,start,cnt,sum_cnt,flag)
                        begin
                                --struttura del pallogramma
                        case y_q is
                                when IDLE => if (start = '0') then y_d <= IDLE;
                                                else y_d <= WRITE_A;
                                                end if;

                                when WRITE_A => if (cnt = "1111111111") then y_d <= LOAD_KEYH;
                                                else y_d <= WRITE_A;
                                                end if;

                                when LOAD_KEYH => y_d <= LOAD_KEYL;

                                when LOAD_KEYL => y_d <= LOAD_X;

                                when LOAD_X => if (flag = '1') then y_d <= LOAD_REG;
                                                else y_d <= LOAD_X;
                                                end if;

                                when LOAD_REG => y_d <= SUM;

                                when SUM => if (sum_cnt = "101") then y_d <= CALC_CT;
                                                else y_d <= SUM;
                                                end if;

                                when CALC_CT => y_d <= WRITE_B;
```

```
                when WRITE_B => if (flag = '1') then y_d <= INCREMENT;
                                else y_d <= WRITE_B;
                                end if;

                when INCREMENT => if (cnt = "0111111111") then y_d <= DONE;
                                  else y_d <= PRE_X;
                                  end if;

                when PRE_X => y_d <= LOAD_X;

                when DONE => if (start = '1') then y_d <= DONE;
                                else y_d <= IDLE;
                                end if;

                when others => y_d <= IDLE;

        end case;

end process state_proc;

clkpr : process (clk_fsm)
                --passagio allo stato futuro al colpo di clk
        begin

        if(clk_fsm' event and clk_fsm = '1') then
                if ( rst = '0') then y_q <= IDLE;
                        else y_q <= y_d;
                        end if;
        end if;

end process clkpr;


outputs:process (y_q,clk_fsm) -- state register
        begin
        --default values
        csa <= '0';
        wra <= '0';
        rda <= '0';
        en_x <= '0';
        clr_reg <= '1';
        en_reg <= '0';
        clr_sc <= '1';
        ce_sc <= '0';
        en_ct <= '0';
        en_keyh <= '0';
        en_keyl <= '0';
        clr_cnt <= '1';
        ce_cnt <= '0';

        s4 <= '0';
        csb <= '0';
        wrb <= '0';

        done_out <= '0';

        case y_q is
                when IDLE =>
                        clr_cnt <= '0';
                        clr_reg <= '0';
                        clr_sc <= '0';
```

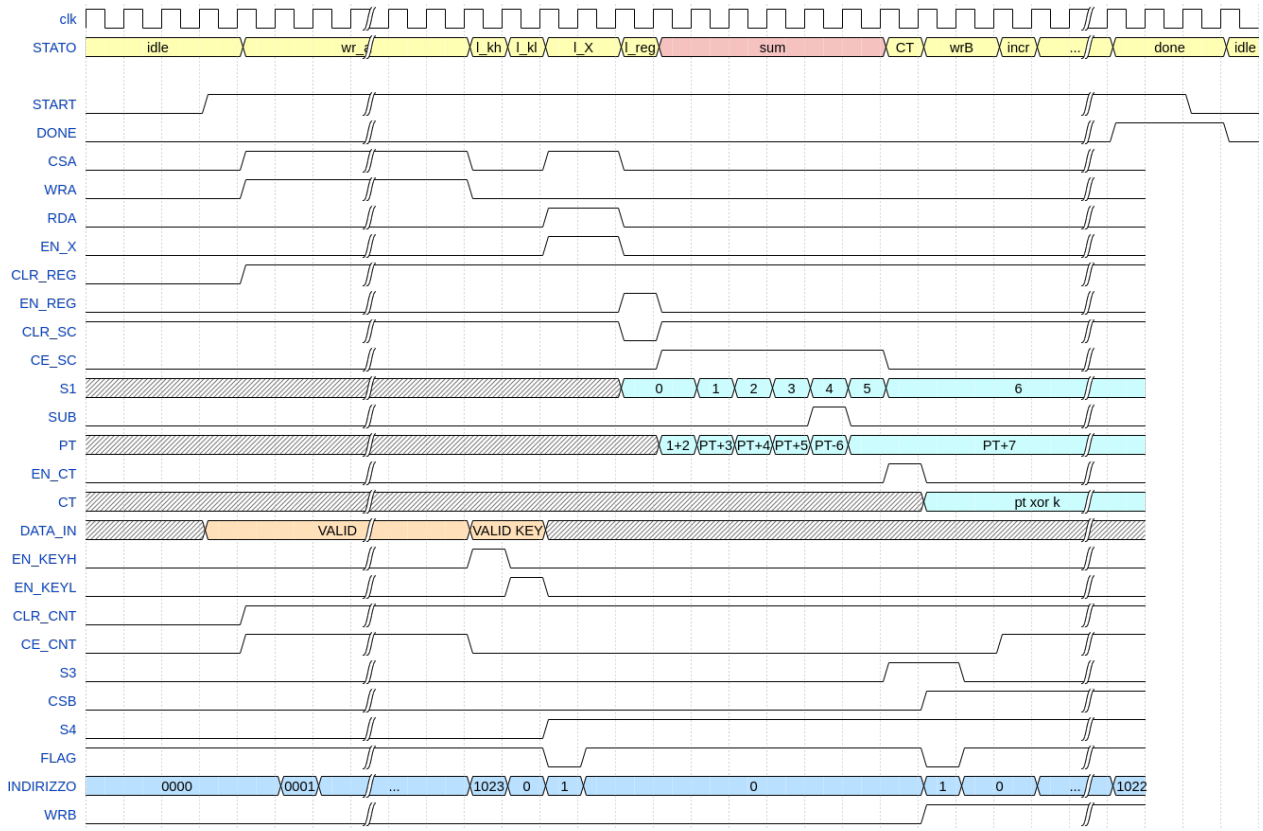
```
            flag <= '1';

        when WRITE_A =>
            csa <= '1';
            wra <= '1';
            ce_cnt <= '1';
            flag <= '1';

        when LOAD_KEYH =>
            en_keyh <= '1';
            flag <= '1';

        when LOAD_KEYL =>
            en_keyl <= '1';
            flag <= '1';
            rda<='1';
            csa<='1';

        when LOAD_X =>
            rda <= '1';
            s4 <= '1';
            csa <= '1';
            en_x <= '1';
            if(clk_fsm' event and clk_fsm = '0') then
            flag <= not flag;
            end if;

        when LOAD_REG =>
            en_reg <= '1';
            clr_sc <= '0';
            s4 <= '1';
            en_x <= '0';
            flag <= '1';

        when SUM =>
            ce_sc <= '1';
            s4 <= '1';
            flag <= '1';

        when CALC_CT =>
            en_ct <= '1';
            --s3 <= '1';
            s4 <= '1';
            flag <= '1';

        when WRITE_B =>
            csb <= '1';
            wrb <= '1';
            s4 <= '1';
            if(clk_fsm' event and clk_fsm = '0') then
            flag <= not flag;
            end if;


        when INCREMENT =>
            ce_cnt <= '1';
            s4 <= '1';
            flag <= '1';

        when PRE_X =>
            s4 <= '1';
```

```
                flag <= '1';
                csa <= '1';
                rda <= '1';

            when DONE =>
                done_out <= '1';
                flag <= '1';

            when others => flag <= '1';
        end case;
    end process outputs;
end struct;
```

4 Timing and Test Bench

In fase di progettazione per verificare la corretta temporizzazione del progetto, è stata disegnata l'evoluzione temporale dei segnali più importanti, nella figura si può osservare la divisione in quattro fasi principali:

- la fase di idle in cui il sistema è completamente resettato e statico
- la fase di write A
- la fase di calcolo e di salvataggio dei dati
- la fase di done in cui il dispositivo aspetta l'handshake con il mondo esterno.



I risultati attesi sono stati riscontrati una volta eseguita la simulazione in Modelsim, infatti il diagramma Wave del programma è pressochè identico alle onde disegnate durante la progettazione, a meno di correzioni minori effettuate durante il debug del codice VHDL.

4.1 Testbench

Per il test delle funzionalità del circuito è stato creato un testbench che genera i segnali di controllo (reset, start e stop) e i valori in ingresso su dataIn. E' stato scritto in c++ un codice per la generazione di un file di testo con 512 valori casuali a 16 da fornire in ingresso alla memoria A dopo essere divisi in due byte secondo le specifiche BIG Endian. Il file di testo viene aperto e letto dal testbench che riga per riga mette il valore generato casualmente in ingresso alla memoria A. In questo modo è stato possibile testare a pieno il sommatore e le altre parti del circuito complessivo.

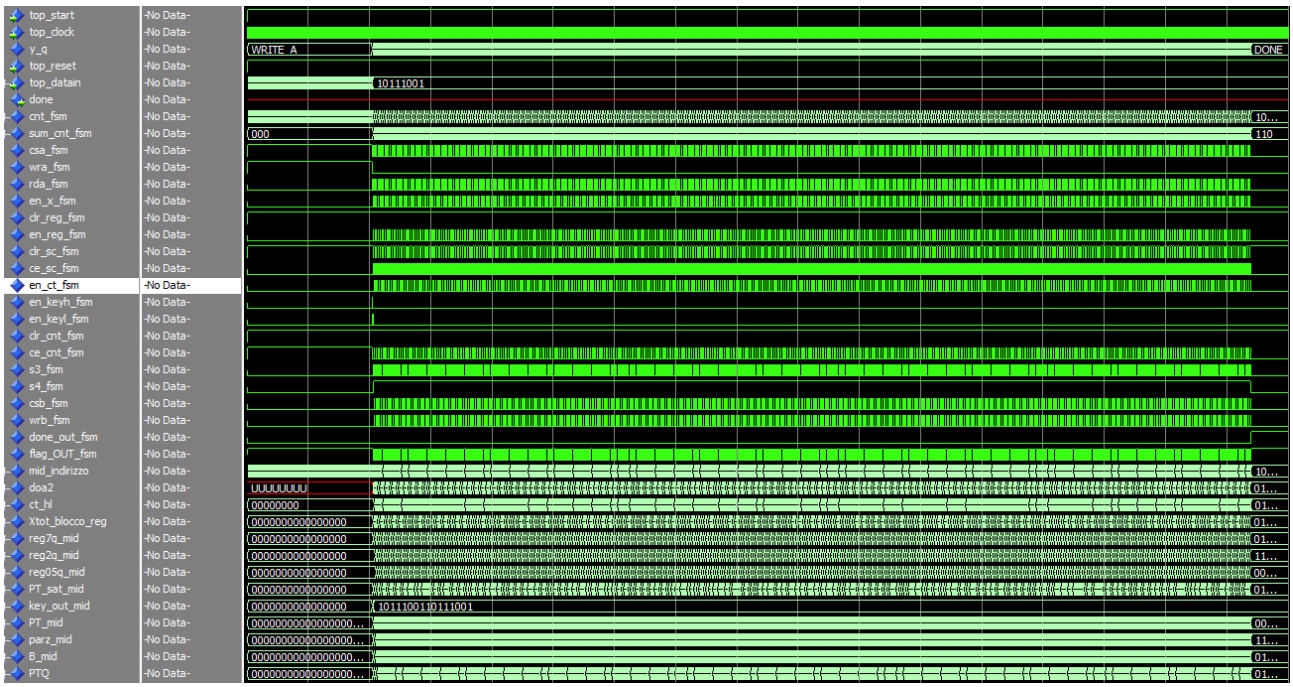
tb_FIR_file.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use STD.Textio.all;


entity tb_FIR_file is
end tb_FIR_file;

architecture testing of tb_FIR_file is
        --dichiarazione del componente da testare
        component FIR_top_level is
                port(top_start,top_clock,top_reset:in std_logic;
                                top_datain:signed(7 downto 0);
                                done:out std_logic);
        end component;

        signal parti,orologio,ripristina,fermate:std_logic;
        signal ingresso:signed(7 downto 0);

begin
        --creazione del clock (2 ns di periodo)
        clk: process
                begin
                orologio<='1';
                wait for 1 ns;
                orologio<='0';
                wait for 1 ns;
        end process;

        --creazione dei segnali di controllo della macchina
        ripristina<='0','1' after 5 ns;
        parti<='0','1' after 6 ns;

        --processo di lettura del file di ingresso
        i: process
                file fin : text is in "datain.txt";
                variable buf : line;
                variable buf_bit : bit_vector(7 downto 0);


        begin
        --lettura e formattazione dei valori sul file
                wait for 7 ns; --attesa per il segnale di start
                while not (endfile(fin)) loop
                        readline(fin, buf);
                        read(buf, buf_bit);
                        --scrittura del valore sul bus di ingresso
                        ingresso <= signed(to_stdlogicvector(buf_bit));

                        --temporizzazione della scrittura con il clk
```

```
                wait for 2 ns;
            end loop;
            wait;
        end process i;

        --mapping dei segnali con il FIR
        dut_map:FIR_top_level port map
                (top_start=>parti,top_clock=>orologio,top_reset=>ripristina,
                 top_datain=>ingresso,done=>fermate);

end testing;
```

4.2 Waves

Anche in questo caso è chiara la divisione delle quatro fasi sopra citate, la figura riporta l'intero processo di conversione a partire dal caricamento dei dati fino al completamento di tutte le 512 operazioni e del salvataggio di tutti i risultati nella memoria B in formato Little Endian.



5 VHDL Code - Subsections

5.1 Sezione 2.3

memA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memA is
port(datain:in signed(7 downto 0);
                indirizzo:in std_logic_vector(9 downto 0);
                cs,clk,wr,rd: in std_logic;
                dataout:out signed(7 downto 0));
end memA;

architecture logica of memA is
--definizione  della memoria come array
--di parole dove ciascuna parola n byte di tipo signed
type matrice is array(0 to 1023) of signed(7 downto 0);

signal A:matrice;

begin
        process(clk)
                begin
                --scrive solo se gli appropriati segnali di controllo vengono assegnati
                        if(clk' event and clk='1' and wr='0' and cs='1') then
                        A(to_integer(unsigned(indirizzo)))<=datain;

                --legge solo se gli appropriati segnali di controllo vengono assegnati
                        elsif(clk' event and clk='1' and rd='1' and cs='1') then
                        dataout<=A(to_integer(unsigned(indirizzo)));
                        end if;
        end process;
end architecture;
```

counter_10bits.vhd

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

--contatore da 0 1023
entity counter_10bits is
        port(CE_CNT,clk,CLR_CNT:in std_logic;
                C_out:buffer std_logic_vector(9 downto 0));
end counter_10bits;

architecture logica of counter_10bits is

        signal p,u:std_logic_vector(8 downto 0);

        component t_flipflop is
                port(T,clock,reset:in std_logic;
                        Q:buffer std_logic);
        End component;

        begin
                --ff 1
                tff0:t_flipflop port map(T => CE_CNT, clock => clk, reset => CLR_CNT, Q => C_out(0));
```

```
                p(0) <= C_out(0) AND CE_CNT;

                --genera 8 flip flop
                generazione:for i in 1 to 8 generate
                        tffi:t_flipflop port map( T => p(i-1), clock => clk,
                                        reset => CLR_CNT, Q => C_out(i));
                        p(i) <= C_out(i) AND p(i-1);
                end generate;

                --flip flop 9
                tff9:t_flipflop port map(T => p(8), clock => clk, reset => CLR_CNT, Q => C_out(9));

end architecture;
```

t_flipflop.vhd

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

Entity t_flipflop is
port(T,clock,reset:in std_logic;
                Q:buffer std_logic);
End t_flipflop;

Architecture logica of t_flipflop is
--signal tmp:std_logic;

 begin
 --tmp<=NOT Q;
 process(clock,reset)
        begin
                IF (reset='0') then
                        Q<='0';
                ELSIF(Clock' EVENT AND Clock = '1' AND T='1') THEN
                        Q<=((NOT Q));
                END IF;
 END process;
END logica;
```

5.2 Sezione 2.4

X.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity X is
        port(DOA: in signed(7 downto 0);
                clk,clr,en_x:in std_logic;
                Xtot:out signed(15 downto 0));
end X;

architecture logica of X is

        component regn is
                GENERIC ( N: integer:=8);
                PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                        Clock, Resetn, EN: IN STD_LOGIC;
                        Q: OUT std_logic_vector(N-1 DOWNTO 0));
        END component;
```

```
        --definizione dei due segnali di uscita, uno per
        --il least significant byte e l'altro per il most significant byte
        signal xlq:std_logic_vector(7 downto 0);
        signal xhq:std_logic_vector(7 downto 0);

        begin

                --i due registri che contengono le due parti di X
                reg1:regn port map(R=>std_logic_vector(DOA),Clock=>clk,
                        Resetn=>clr,EN=>en_x,Q=>xlq);
                reg2:regn port map(R=>std_logic_vector(xlq),Clock=>clk,
                        Resetn=>clr,EN=>en_x,Q=>xhq);

                --concatenazione per ottenere il segnale X completo
                --su 16 bit in formato big endian
                Xtot<=signed(xhq & xlq);


end architecture;
```

blocco_reg.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity blocco_reg is
        port(X_tot:in signed (15 downto 0);
                clr_reg,clk,en_reg:in std_logic;
                reg7q,reg2q,reg05q:out signed(15 downto 0));
end blocco_reg;

architecture logica of blocco_reg is
        component regn is
                GENERIC ( N: integer:=16);
                PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                        Clock, Resetn, EN: IN STD_LOGIC;
                        Q: OUT std_logic_vector(N-1 DOWNTO 0));
        END component;

        signal q7,q2,q05:std_logic_vector (15 downto 0);

        --in questo blocco si hanno tre registri che contengono
        --i tre componenti necessari per il calcolo di PT

        --ciascun registro ha la sua uscita che verrmanipolata
        -- poi nel blocco antecedente il sommatore
        begin

                reg1:regn port map(R=>std_logic_vector(X_tot),Clock=>clk,Resetn=>clr_reg,
                                EN=>en_reg,Q=>q7);
                reg7q<=signed(q7);

                reg2:regn port map(R=>q7,Clock=>clk,Resetn=>clr_reg,EN=>en_reg,Q=>q2);
                reg2q<=signed(q2);

                reg3:regn port map(R=>q2,Clock=>clk,Resetn=>clr_reg,EN=>en_reg,Q=>q05);
                reg05q<=signed(q05);

end architecture;
```

5.3 Sezione 2.5

blocco_mux.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity blocco_mux is
        port(s1:in std_logic_vector(2 downto 0);
                q7,q2,q05:in signed(15 downto 0);
                feedback:in signed(21 downto 0);
                parz,B:out signed(21 downto 0));
end blocco_mux;

architecture logica of blocco_mux is

        component bit21_mux6to1 is
                port(u21,v21,z21,w21,t21,k21:in std_logic_vector(21 downto 0);
                        s21:in std_logic_vector(2 downto 0);
                        o21:out std_logic_vector(21 downto 0));
        end component;

        component bit21_mux2to1 is
                port(x21,y21:in std_logic_vector(21 downto 0);
                        s21:in std_logic;
                        o21:out std_logic_vector(21 downto 0));
        end component;

        signal in1,in2,in3,in4,in5,in6,in7:signed(21 downto 0);
        signal controllo2:std_logic;

        begin
                --si hanno 7 ingressi

                --i primi 5 sono i termini che
                -- una volta sommati corrispondo ad una moltiplicazione per 31
                in1<= q7(15)& q7(15)& q7(15) & q7(15) & q7(15) & q7(15) & q7;
                in2<= q7(15)& q7(15)& q7(15) & q7(15) & q7(15)  & q7 & '0';
                in3<=  q7(15)& q7(15) & q7(15) & q7(15)  & q7 & '0' & '0';
                in4<=  q7(15)& q7(15) & q7(15)  & q7 & '0' & '0' & '0';
                in5<=   q7(15)& q7(15)  & q7 & '0' & '0' & '0' & '0';


                --il sesto termine che corrisponde a x(i-1)
                -- gioltiplicato per 8 ed esteso su 22 bit
                in6<= q2(15) & q2(15) & q2(15) & q2 & "000";

                -- il settimo termine che corrisponde a x(i-2)
                --moltiplicato per 2 ed esteso su 22 bit
                in7<= q05(15) & q05(15) & q05(15) & q05(15) & q05(15) & q05 & '0';

                mux1:bit21_mux6to1 port map(u21=>std_logic_vector(in2),v21=>std_logic_vector(in3),
                        z21=>std_logic_vector(in4),w21=>std_logic_vector(in5),
                        t21=>std_logic_vector(in6),k21=>std_logic_vector(in7),
                        s21=>s1,signed(o21)=>parz);

                --controllo 2 che a partire dalla seconda iterazione
                 --equivale sempre a 1, scegliendo il feedback che torna dal sommatore
                controllo2<=s1(0) OR s1(1) OR s1(2);
                mux2:bit21_mux2to1 port map(x21=>std_logic_vector(in1),y21=>std_logic_vector(feedback),
                        s21=>controllo2,signed(o21)=>B);
```

```
end architecture;
```

bit21_add_sub.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bit21_add_sub is
        port(x21,y21:in signed(21 downto 0);
                counter_in:std_logic_vector(2 downto 0);
                PT:out signed(21 downto 0);
                carry_out:out std_logic);
end bit21_add_sub;


architecture logica of bit21_add_sub is
        component bit21_adder is
                port(a21,b21: in SIGNED (21 downto 0);
                        cin:in std_logic;
                        cout: out std_logic;
                        s21:out SIGNED (21 downto 0));
        end component;

        signal vector: signed(21 downto 0);
        signal operazione:std_logic;
        signal x_mod:signed (21 downto 0);


        begin
                --in primis si ricava il segnale operazione
                --che andrd 1 solo quando il counter vale 4
                --ovvero quando si sottrae 8*X(i-1)
                operazione<=counter_in(2) AND (NOT counter_in(1)) AND (NOT counter_in(0));


                --tramite la xor si cambia ciascun bit se
                --operazione=1 e quindi si ha una sottrazione
                vector<=(others=>operazione);
                x_mod<=x21 XOR vector;

                --operazione poi serve da carry_in dato che
                --in caso di sottrazione al segnale cambiato va sommato 1
                somma:bit21_adder port map(a21=>x_mod,b21=>y21,cin=>operazione,
                        cout=>carry_out,s21=>PT);

end architecture;
```

mux6to1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux6to1 is
port(u,v,z,w,t,k:in std_logic;
                s6:in std_logic_vector(2 downto 0);
                o:out std_logic);
end mux6to1;

architecture logica of mux6to1 is
```

```
SIGNAL p1,p2,p3,p4: std_logic;

            component mux2to1 is
                port(x, y, s: in std_logic;
                    m: out std_logic);
        end component;

                begin

                 Mux1: mux2to1 PORT MAP(x=>u, y=>v, s=>s6(0), m=>p1);

                Mux2: mux2to1 PORT MAP(x=>z, y=>w, s=>s6(0), m=>p2);

                 Mux3: mux2to1 PORT MAP(x=>t, y=>k, s=>s6(0), m=>p3);

                 Mux4: mux2to1 PORT MAP(x=>p1, y=>p2, s=>s6(1), m=>p4);

                        Mux5: mux2to1 PORT MAP(x=>p4, y=>p3, s=>s6(2), m=>o);

end architecture;
```

bit21_mux6to1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bit21_mux6to1 is
        port(u21,v21,z21,w21,t21,k21:in std_logic_vector(21 downto 0);
                s21:in std_logic_vector(2 downto 0);
                o21:out std_logic_vector(21 downto 0));
end bit21_mux6to1;

architecture logica of bit21_mux6to1 is
        component mux6to1 is
                port(u,v,z,w,t,k:in std_logic;
                        s6:in std_logic_vector(2 downto 0);
                        o:out std_logic);
        end component;

        begin
                a:for i in 0 to 21 generate
                muxi:mux6to1 port map(u=>u21(i),v=>v21(i),z=>z21(i),w=>w21(i),
                        t=>t21(i),k=>k21(i),s6=>s21,o=>o21(i));
                end generate;
end architecture;
```

bit21_mux2to1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bit21_mux2to1 is
port(x21,y21:in std_logic_vector(21 downto 0);
                s21:in std_logic;
                o21:out std_logic_vector(21 downto 0));
end bit21_mux2to1;

architecture logica of bit21_mux2to1 is

component mux2to1 is
```

```
    port(x, y, s: in std_logic;
        m: out std_logic);
end component;

begin
a:for i in 0 to 21 generate
        muxi:mux2to1 port map(x=>x21(i),y=>y21(i),s=>s21,m=>o21(i));
end generate;
end architecture;
```

bit21_adder.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bit21_adder is
        port(a21,b21: in SIGNED (21 downto 0);
                cin:in std_logic;
                cout: out std_logic;
                s21:out SIGNED (21 downto 0));
end bit21_adder;

architecture logica of bit21_adder is
        signal c_middle: std_logic_vector (20 downto 0);


        component full_adder is
                PORT(ci,a,b: IN std_logic;
                        somma,co:OUT std_logic);
        end component;
        begin
                fa0:full_adder port map(ci=>cin,a=>a21(0),
                        b=>b21(0),somma=>s21(0),co=>c_middle(0));

                generazione: for i in 1 to 20 generate
                                fa1to20:full_adder port map(ci=>c_middle(i-1),a=>a21(i),
                                        b=>b21(i),somma=>s21(i),co=>c_middle(i));
                end generate;

                fa21: full_adder port map(ci=>c_middle(20),a=>a21(21),
                        b=>b21(21),somma=>s21(21),co=>cout);


end logica;
```

full_adder.vhd

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

entity full_adder is
        PORT(ci,a,b: IN std_logic;
                 somma,co:OUT std_logic);
END full_adder;

architecture logica of full_adder is
        component mux2to1 is
        port(x, y, s: in std_logic;
        m: out std_logic);
        end component;

        signal p0 : std_logic;
```

```
begin
        p0<= a XOR b;
        mapping: mux2to1 port map(x=>b,y=>ci,s=>p0,m=>co);
        somma<= ci XOR p0;
end logica;
```

counter_3bits.vhd

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

--contatore a 3 bit
--utilizzato all'interno del blocco di somma
--per scandire i vari ingressi dei mux ad ogni colpo di clock
--uscita utilizzata anche come segnale di SUM/SUB del sommatore
entity counter_3bits is
        port(CE_CNT,clk,CLR_CNT:in std_logic;
                C_out:buffer std_logic_vector(2 downto 0));
end counter_3bits;

architecture logica of counter_3bits is

        signal p,u:std_logic_vector(1 downto 0);

        component t_flipflop is
                port(T,clock,reset:in std_logic;
                        Q:buffer std_logic);
        End component;

        begin

                tff0:t_flipflop port map(T => CE_CNT, clock => clk, reset => CLR_CNT, Q => C_out(0));
                p(0) <= C_out(0) AND CE_CNT;

                tff1:t_flipflop port map( T => p(0), clock => clk, reset => CLR_CNT, Q => C_out(1));
                p(1) <= C_out(1) AND p(0);

                tff2:t_flipflop port map(T => p(1), clock => clk, reset => CLR_CNT, Q => C_out(2));

end architecture;
```

5.4 Sezione 2.6

satura_PT.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--blocco per la saturazione di PT
entity satura_PT is
        port( PT : in signed ( 21 downto 0);
              PT_sat : out signed ( 15 downto 0));
end satura_PT;

architecture struct of satura_PT is

        component bitN_mux2to1
                GENERIC ( N: integer := 10);
```

```
                port(x8, y8: in std_logic_vector (N-1 downto 0);
                        s8: in std_logic;
                        m8: out std_logic_vector (N-1 downto 0));
        end component;

        signal pt_max, pt_min, intern: signed(15 downto 0);
        signal selector : std_logic;

        begin
                --0x7fff
                --max numero positivo su 16 bit
                pt_max <= "0111111111111111";

                --0x8000
                --min numero negativo su 16 bit
                pt_min <= "1000000000000000";

                --se il bit di segno e' 1 (selettore del mux)
                --seleziona 0x8000
                --altrimenti seleziona 0x7fff
                M1: bitN_mux2to1 generic map ( N => 16)
                        port map (std_logic_vector(pt_max), std_logic_vector(pt_min),
                                PT(21), signed(m8)=>intern);


                --selettore del 2 mux
                --e' 0 quando i bit sono uguali
                -- 1 altrimenti
                selector <= not (((not PT(21)) and (not PT(20)) and (not PT(19))
                                and (not PT(18)) and (not PT(17)))
                                or (PT(21) and PT(20) and PT(19) and PT(18) and PT(17)));

                --se selector = 1
                --seleziona il PT saturato
                --ovvero uscita del mux precedente
                --altrimenti seleziona i 16 bit del PT (uscita del sommatore)
                M2: bitN_mux2to1 generic map (N => 16)
                        port map (std_logic_vector(PT(17 downto 2)), std_logic_vector(intern),
                                selector, signed(m8)=>PT_sat);

end struct;
```

CT_calculator.vhd

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

--blocco per il calcolo di CT
entity CT_calculator is
        port ( PT_sat : in signed (15 downto 0);
                clk : in std_logic;
                clr_ct : in std_logic;
                EN_ct : in std_logic;
                s3 : in std_logic;
                key : in signed( 15 downto 0);
                CT_out : out signed ( 7 downto 0));
end CT_calculator;

architecture struct of CT_calculator is

        component regn
```

```
                GENERIC ( N: integer:=7);
                PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                        Clock, Resetn, EN: IN STD_LOGIC;
                        Q: OUT std_logic_vector(N-1 DOWNTO 0));
        end component;

        component bitN_mux2to1
                GENERIC ( N: integer := 10);
                port(x8, y8: in std_logic_vector (N-1 downto 0);
                        s8: in std_logic;
                        m8: out std_logic_vector (N-1 downto 0));
        end component;

        signal R, Qout : signed ( 15 downto 0);

        begin
                --ingresso del registro
                --calcolo vero e proprio di CT
                R <= PT_sat xor key;

                --registro utilizzato per salvare CT
                R1: regn generic map ( N => 16)
                        port map (R=> std_logic_vector(R), clock=>Clk, resetn=>clr_ct,
                                EN=>EN_ct,signed(Q)=>Qout);

                --mux utilizzato per "dividere" il CT
                --se s3  --> parte LSB (ultimi 8 bit)
                --se s3  --> parte MSB (primi 8 bit)
                --tale blocco implementa la "conversione" da Big a Little Endian
                M1: bitN_mux2to1 generic map (N => 8)
                        port map (x8=>std_logic_vector(Qout(7 downto 0)),
                                y8=>std_logic_vector(Qout(15 downto 8)),s8=>s3, signed(m8)=>CT_out);

end struct;
```

regn.vhd

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

ENTITY regn IS
        GENERIC ( N: integer:=8);
        PORT (R: IN std_logic_vector(N-1 DOWNTO 0);
                        Clock, Resetn, EN: IN STD_LOGIC;
                        Q: OUT std_logic_vector(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' AND EN ='1') THEN
                                Q <= R;
                        END IF;
        END PROCESS;
```

```
END Behavior;
```

bitN_mux2to1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bitN_mux2to1 is
        GENERIC ( N: integer := 10);
    port(x8, y8: in std_logic_vector (N-1 downto 0);
        s8: in std_logic;
        m8: out std_logic_vector (N-1 downto 0));
end bitN_mux2to1;

architecture arch_mux8 of bitN_mux2to1 is
    component mux2to1 is
        port(x, y, s: in std_logic;
            m: out std_logic);
        end component;
    begin
    a: for i in 0 to N-1 generate
        hi: mux2to1 port map(x =>  x8(i), y => y8(i), s => s8, m => m8(i));
    end generate;

end arch_mux8;
```

mux2to1.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
    port(x, y: in std_logic;
         s : in std_logic;
        m: out std_logic);
end mux2to1;

architecture arch_mux of mux2to1 is
    begin
            h1: m <= (NOT (s) AND x) OR (s AND y);
end arch_mux;
```

memB.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memB is
port(datain:in signed(7 downto 0);
                indirizzo:in std_logic_vector(9 downto 0);
                cs,clk,wr,rd: in std_logic;
                dataout:out signed(7 downto 0));
end memB;


architecture logica of memB is
--descrizione equivalente a memA
type matrice is array(0 to 1023) of signed(7 downto 0);

signal B:matrice;
```

```
begin
        process(clk)
                begin
                        if(clk' event and clk='1' and wr='0' and cs='1') then
                        B(to_integer(unsigned(indirizzo)))<=datain;
                        elsif(clk' event and clk='1' and rd='1' and cs='1') then
                        dataout<=B(to_integer(unsigned(indirizzo)));
                        end if;
        end process;
end architecture;
```