library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity **RAM\_256x8** is

port ( Address : in unsigned (7 downto 0);

Data\_in : in signed(7 downto 0);

data\_out: out signed(7 downto 0);

cs, write\_0\_read\_1, clock : in std\_logic);

end RAM\_256x8;

architecture Struct of RAM\_256x8 is

begin

process(clock, cs, address, write\_0\_read\_1, data\_in) is

type ram\_array is array (0 to 255) of signed(7 downto 0) ;

variable  mem : ram\_array;

begin

if (cs='1') then

if (  write\_0\_read\_1='1' ) then

Data\_out<= mem(to\_integer(Address));

elsif(clock'event and clock= '1') then

if (write\_0\_read\_1='0') then

mem(to\_integer(unsigned(Address))) := Data\_in;

end if;

end if;

end if;

end process;

end Struct;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity **SRAM\_SW\_AR\_256x8** is

port( Address : in unsigned (7 downto 0);

Data\_in : in signed(7 downto 0);

Data\_out : out signed(7 downto 0);

cs, write\_0\_read\_1, clock: in std\_logic

);

end SRAM\_SW\_AR\_256x8;

architecture Struct of SRAM\_SW\_AR\_256x8 is

component RAM\_256x8 is

port (Address : in unsigned (7 downto 0);

Data\_in : in signed(7 downto 0);

Data\_out : out signed(7 downto 0);

cs, write\_0\_read\_1, clock : in std\_logic);

end component;

begin

Mema : RAM\_256x8 port map (Address, Data\_in, Data\_out, cs, write\_0\_read\_1, clock);

end Struct;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity **SRAM\_SW\_AR\_1024x8** is

port( Address : in unsigned (7 downto 0);

     DATA\_IN : in signed(7 downto 0);

DATA\_OUT\_A, DATA\_OUT\_B, DATA\_OUT\_C, DATA\_OUT\_D : out signed(7 downto 0);

CS : in std\_logic\_vector(3 downto 0);

Write\_0\_read\_1, clock : in std\_logic

);

end SRAM\_SW\_AR\_1024x8;

architecture Struct of SRAM\_SW\_AR\_1024x8 is

component SRAM\_SW\_AR\_256x8 is

port( Address : in unsigned (7 downto 0);

Data\_in : in signed(7 downto 0);

Data\_out : out signed(7 downto 0);

cs, write\_0\_read\_1, clock : in std\_logic

);

end component;

begin

Ram\_0 : SRAM\_SW\_AR\_256X8 port map (Address, data\_in, DATA\_OUT\_A, CS(0), Write\_0\_read\_1, Clock);

Ram\_1 : SRAM\_SW\_AR\_256X8 port map (Address, data\_in, DATA\_OUT\_B, CS(1), Write\_0\_read\_1, Clock);

Ram\_2 : SRAM\_SW\_AR\_256X8 port map (Address, data\_in, DATA\_OUT\_C, CS(2), Write\_0\_read\_1, Clock);

Ram\_3 : SRAM\_SW\_AR\_256X8 port map (Address, data\_in, DATA\_OUT\_D, CS(3), Write\_0\_read\_1, Clock);

end Struct;

library IEEE;

use IEEE.std\_logic\_1164.all;

entity **Decode**r is

port(

EN : in std\_logic;

D : out std\_logic\_vector( 3 downto 0);

sel : in std\_logic\_vector( 1 downto 0)

);

end Decoder;

Architecture behav of Decoder is

begin

process(EN, sel)

begin

if(EN = '1') then

if(sel = "00") then

D <= "0001";

elsif (sel = "01") then

D <= "0010";

elsif (sel = "10") then

D <= "0100";

else

D <= "1000";

end if;

else

D <= "0000";

end if;

end process;

end behav;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity **SRAM\_SW\_AR\_1024x8\_DEC** is

port( ADDRESS : in unsigned (9 downto 0);

     DATA\_IN : in signed(7 downto 0);

DATA\_OUT : out signed(7 downto 0);

CS, WRite\_0\_read\_1, clock : in std\_logic

);

end SRAM\_SW\_AR\_1024x8\_DEC;

architecture Stru of SRAM\_SW\_AR\_1024x8\_DEC is

signal Cs\_signal : std\_logic\_vector(3 downto 0);

signal DATA\_OUT\_0, DATA\_OUT\_1, DATA\_OUT\_2, DATA\_OUT\_3 : signed(7 downto 0);

component SRAM\_SW\_AR\_1024x8 is

port( Address : in unsigned (7 downto 0);

DATA\_IN : in signed(7 downto 0);

DATA\_OUT\_A, DATA\_OUT\_B, DATA\_OUT\_C, DATA\_OUT\_D : out signed(7 downto 0);

CS:in std\_logic\_vector(3 downto 0);

Write\_0\_read\_1, Clock : in std\_logic

);

end component;

component Decoder is

port(

EN : in std\_logic;

D : out std\_logic\_vector( 3 downto 0);

sel : in std\_logic\_vector( 1 downto 0)

);

end component;

component mux\_4\_to\_1\_8bit is   --multiplexer selettore di uscita dati

port(

sel : in std\_logic\_vector (1 downto 0);

y1, y2, y3, y4 : in signed (7 downto 0);

y\_sat : out signed (7 downto 0)

);

end component;

begin

Dec: Decoder port map (CS, Cs\_signal, std\_logic\_vector(address(9 downto 8)));

Meme : SRAM\_SW\_AR\_1024x8 port map (address(7 downto 0), Data\_in,  DATA\_OUT\_0,       DATA\_OUT\_1, DATA\_OUT\_2, DATA\_OUT\_3,       CS\_signal, write\_0\_read\_1, clock );

mux : mux\_4\_to\_1\_8bit port map (std\_logic\_vector(address(9 downto 8)),DATA\_OUT\_0, DATA\_OUT\_1, DATA\_OUT\_2, DATA\_OUT\_3, Data\_out);

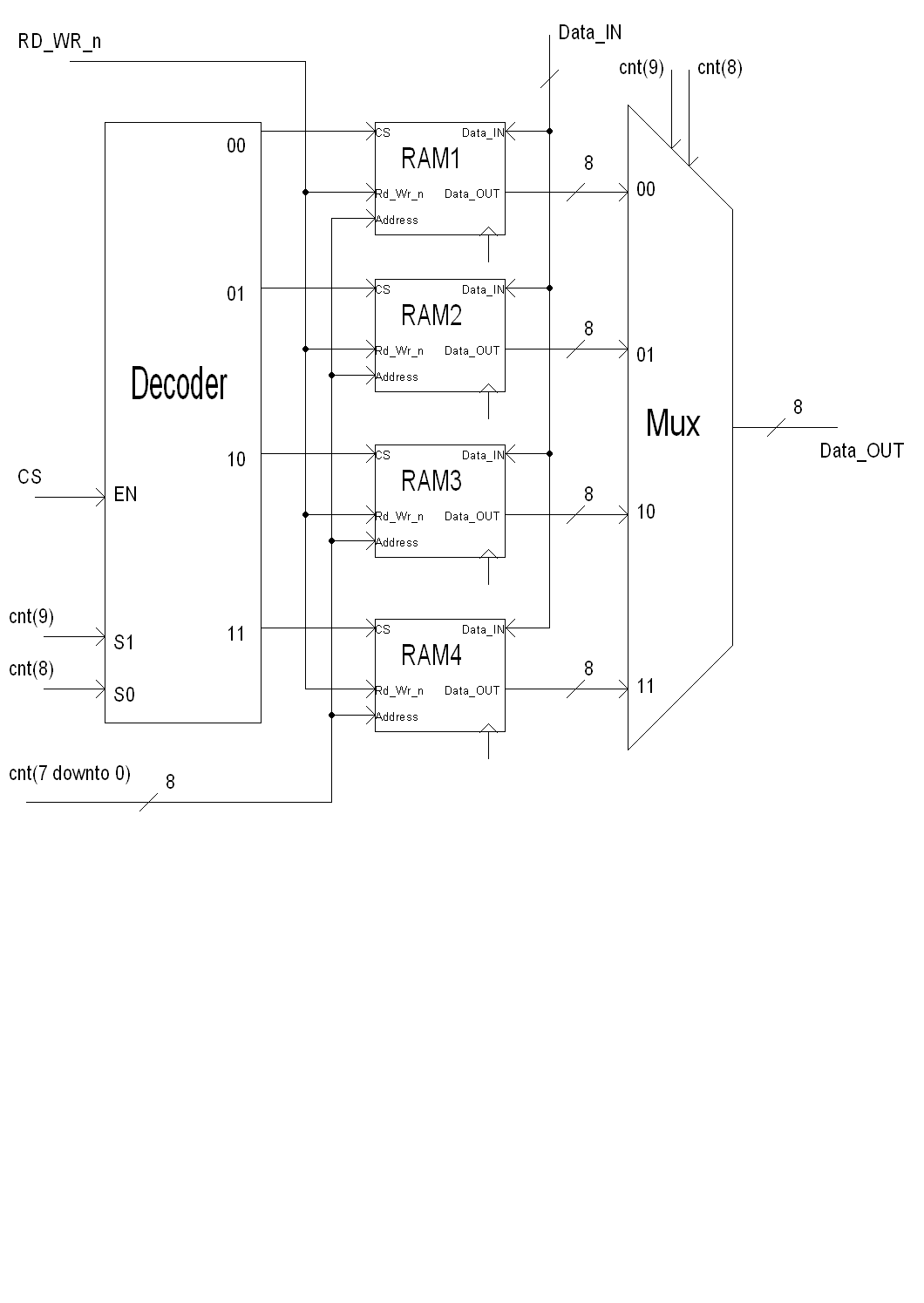
end Stru;

La composizione di questi componenti (più il multiplexer descritto dopo) forma una memoria Ram da 1024x8 bit. In particolare abbiamo usato un sistema strutturale, ovvero:

* abbiamo definito una Ram 256x8bit;
* abbiamo creato una struttura con quattro di queste memorie per avere in totale una memoria da 1024x8;
* l’ultimo componente invece usa la Ram, un Decoder e Multiplexer per gestire il funzionamento della memoria.

Il decoder utilizza i primi due bit del contatore per abilitare una sola delle Ram piccole, gli altri 8 bit del contatore vengono utilizzate per puntare la cella su cui scrivere il dato in ingresso o da dove prelevare il dato per mandarlo in uscita. Inoltre abbiamo dovuto utilizzare un mux per dover scegliere quale dato mandare in uscita fra le quattro Ram 256x8 selezionato sempre tramite i primi due bit del contatore.

Di seguito abbiamo riportato lo schema generale della memoria:



library ieee;

use ieee.STD\_LOGIC\_1164.all;

use ieee.numeric\_std.all;

--Multiplexer da usare per il saturatore. Parallelismo da 8 bit

entity mux\_4\_to\_1\_8bit is

port(

sel : in std\_logic\_vector (1 downto 0);

y1, y2, y3, y4 : in signed (7 downto 0);

y\_sat : out signed (7 downto 0)

);

end mux\_4\_to\_1\_8bit;

architecture behav of mux\_4\_to\_1\_8bit is

begin

process (sel, y1, y2, y3, y4)

begin

if (sel= "00") then

y\_sat <= y1;

elsif(sel = "01") then

y\_sat <= y2;

elsif( sel = "10") then

y\_sat <= y3;

else

y\_sat <= y4;

end if;

end process;

end behav;

library ieee;

use ieee.STD\_LOGIC\_1164.all;

use ieee.numeric\_std.all;

--Multiplexer da usare per scegliere i dati da far sommare. Parallelismo da 10 bit

entity mux\_4\_to\_1\_10bit is

port(

sel : in std\_logic\_vector (1 downto 0);

Data\_00 : in signed (9 downto 0);

Data\_01 : in signed(9 downto 0);

Data\_10\_11 : in signed (9 downto 0);

y : out signed (9 downto 0)

);

end mux\_4\_to\_1\_10bit;

architecture behav of mux\_4\_to\_1\_10bit is

begin

process (sel, Data\_00, Data\_01, Data\_10\_11)

begin

if (sel= "00") then

y <= Data\_00;

elsif(sel = "01") then

y <= Data\_01;

elsif( sel = "10") then

y <= Data\_10\_11;

else

y <= Data\_10\_11;

end if;

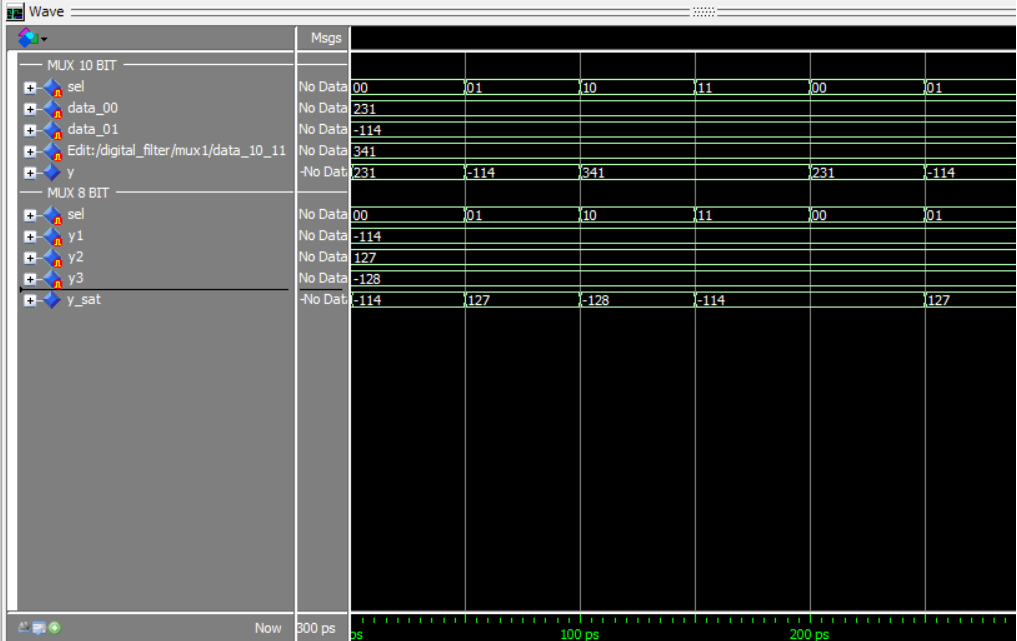
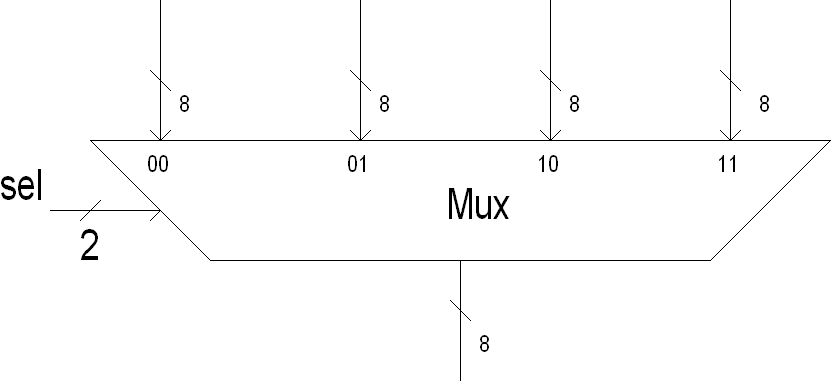
end process;

end behav;

Un generico mux il quale in base ad un comando di selezione sceglie fra i dati in ingresso quale mandare in uscita.

Di seguito abbiamo riportato un mux con 4 dati in ingresso, con parallelismo da 8 bit.

Nel grafico su modelsim invece la simulazione dei due mux usati nel progetto.



library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Sommatore con parallelismo da 10 bit

entity **Adder\_Subtractor\_10\_bit** is

generic (n : integer := 10);

port(

Add\_n\_Sub : IN std\_logic;

a, b : IN signed (n-1 downto 0);

y  : Out signed (n-1 downto 0)

);

end Adder\_Subtractor\_10\_bit;

architecture behav of Adder\_Subtractor\_10\_bit is

begin

process (a,b, Add\_n\_Sub)

begin

if(Add\_n\_Sub = '1') then

y<= a-b;

else

y <= a+b;

end if;

end process;

end behav;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Sommatore con parallelismo da 18 bit

entity **Adder\_Subtractor\_18\_bit** is

generic (n : integer := 18);

port(

Add\_n\_Sub : IN std\_logic;

a, b : IN signed (n-1 downto 0);

y  : Out signed (n-1 downto 0)

);

end Adder\_Subtractor\_18\_bit;

architecture behav of Adder\_Subtractor\_18\_bit is

begin

process (a,b, Add\_n\_Sub)

begin

if(Add\_n\_Sub = '1') then

y<= a-b;

else

y <= a+b;

end if;

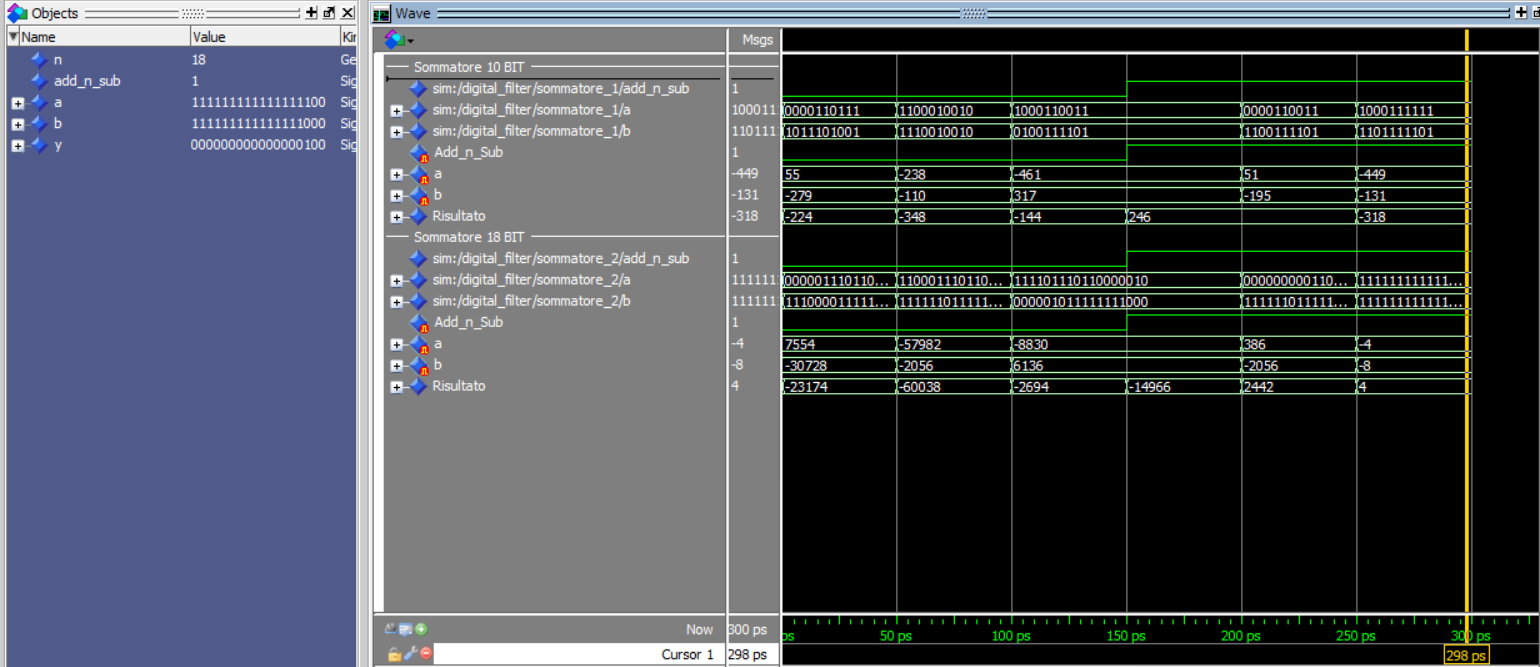
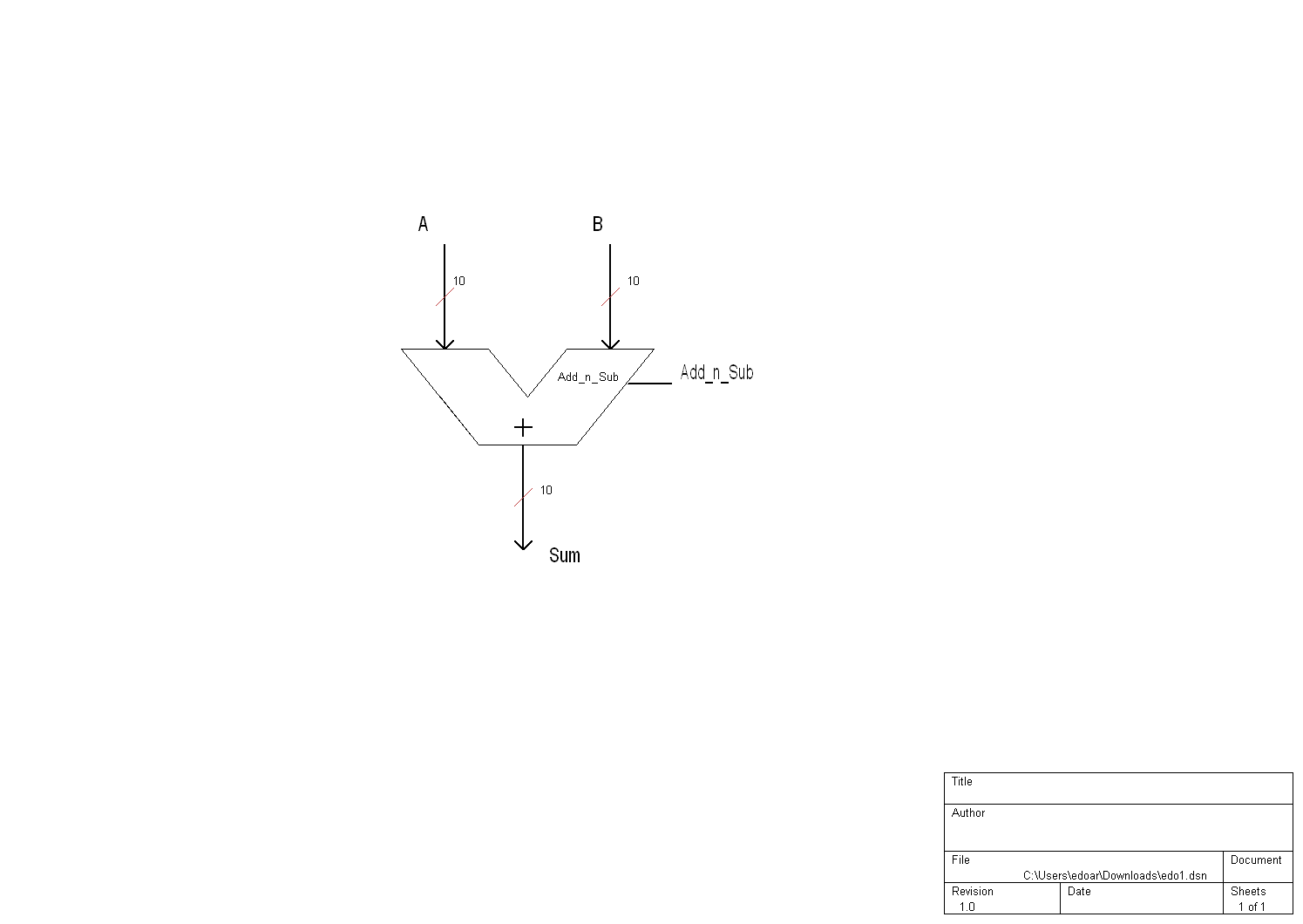
end process;

end behav;

Un generico sommatore il quale in base al segnale di Add\_n\_Sub somma o fa la differenza dei due dati in ingresso. In particolare se il segnale vale ‘0’ allora fa l’operazione di somma, invece se vale ‘1’ fa l’operazione di sottrazione.

Di seguito abbiamo riportato il sommatore, con i dati di ingresso e uscita con parallelismo da 8 bit,  utilizzato nel circuito utilizzato per valutare il dato da inserire nella Memoria .

Nel grafico su modelsim invece la simulazione dei due sommatori usati nel progetto.



library ieee;

use ieee.STD\_LOGIC\_1164.all;

use ieee.numeric\_std.all;

entity **counter\_10\_bit\_sincrono** is

generic ( N : integer:=10);

port

(

Cnt\_EN\_1, CLK, Clear\_1: in std\_logic;

  cnt: buffer unsigned(N-1 downto 0)

);

end counter\_10\_bit\_sincrono;

architecture struct of counter\_10\_bit\_sincrono is

begin

process (clk, clear\_1, cnt\_en\_1)

begin

if (clear\_1 = '1') then

cnt <= (others => '0');

elsif((CLK' event and CLK='1') and Cnt\_en\_1 = '1') then

                 cnt <= cnt + 1;

           end if;

end process;

end struct;

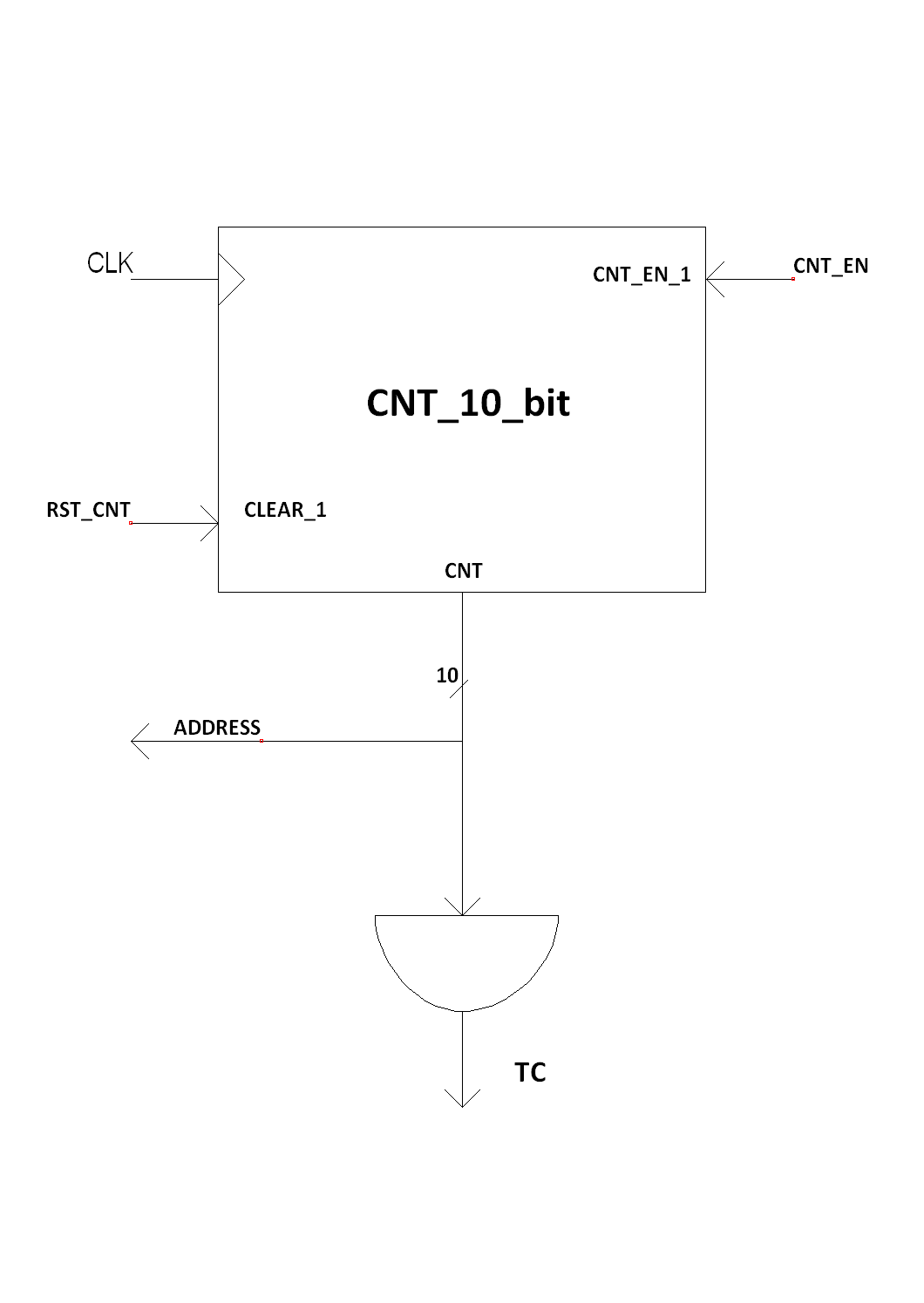
Il componente qui descritto è un contatore modulo 10 che conta da “0000000000” a “1111111111”, ovvero da 0 a 1023.

Esso viene utilizzato per puntare gli indirizzi di memoria della Ram\_A e della Ram\_B sia nella fase di lettura sia nella fase di scrittura. L’ LSB viene utilizzato per decidere se l’operazione (-1)n, in particolare quando esso è ‘0’ vuol dire che ‘n’ è pari quindi viene svolta l’operazione (0+y) invece quando è ‘1’ vuol dire che ‘n’ è dispari quindi viene svolta l’operazione (0-y) al fine di cambiare segno ad y.

Inoltre  quando il contatore arriva alla fine del suo conteggio, ovvero ha contato 1024 operazioni, il segnale TC (collegato a tutti i bit del contatore tramite una porta AND) diventa ‘1’ e comunica alla CU che è finito il ciclo di operazioni e quindi cambia stato.

Il componente viene resettato a ‘0’ tramite il segnale di Rst\_Cnt il quale è una OR tra la il Rst e clear\_cnt, quest’ultimo è un segnale controllato dalla CU che setta il segnale a ‘1’ nel momento in cui c’è bisogno di resettare il contatore, ad esempio nel momento in cui è finita l’operazione di scrittura della Ram\_A.

Per quanto riguarda  riguarda l’abilitazione di conteggio esso è controllato dal segnale Cnt\_En che viene gestito dalla CU. Esso rimane ad ‘1’ durante tutta la fase di scrittura della Mem\_A, invece, successivamente, durante l’elaborazione dei dati si attiva ogni qualvolta viene terminata una intera elaborazione (ogni 4 colpi di clock).



Di seguito le descrizioni dei registri utilizzati nel progetto per salvare i dati durante le operazioni. Abbiamo usato:

* Registri da 8 bit per salvare i dati in output dalla Ram A e per salvare il valore della media;
* Un registro da 10 bit per salvare le operazioni derivanti dall’elaborazione dei dati della Ram A;
* Un registro da 18 bit utilizzato per aggiornare continuamente la somma di tutti gli elementi della Ram A.

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Registro con parallelismo da 8 bit

entity **Reg\_8\_bit** is

generic (n : integer := 8);

port ( D : in signed(n-1 downto 0);

      Rest\_1, Clock, EN\_1 : in std\_logic;

Q : out signed(N-1 downto 0)

);

end Reg\_8\_bit;

architecture Behav of Reg\_8\_bit is

begin

process(Rest\_1, Clock, EN\_1)

begin

if(Rest\_1 = '1') then

Q <= (others => '0');

elsif ((Clock'event and Clock = '1') and EN\_1 ='1') then

Q <= D;

end if;

end process;

end Behav;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Registro con parallelismo da 10 bit

entity **Reg\_10\_bit** is

generic (n : integer := 10);

port ( D : in signed(n-1 downto 0);

      Rest\_1, Clock, EN\_1 : in std\_logic;

Q : out signed(N-1 downto 0)

);

end Reg\_10\_bit;

architecture Behav of Reg\_10\_bit is

begin

process(Rest\_1, Clock, EN\_1)

begin

if(Rest\_1 = '1') then

Q <= (others => '0');

elsif (Clock'event and Clock = '1' and EN\_1 = '1') then

Q <= D;

end if;

end process;

end Behav;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--Registro con parallelismo da 18 bit

entity **Reg\_18\_bit** is

generic (n : integer := 18);

port ( D : in signed(n-1 downto 0);

      Rest\_1, Clock, EN\_1 : in std\_logic;

Q : out signed(N-1 downto 0)

);

end Reg\_18\_bit;

architecture Behav of Reg\_18\_bit is

begin

process(Rest\_1, Clock, EN\_1)

begin

if(Rest\_1 = '1') then

Q <= (others => '0');

elsif (Clock'event and Clock = '1' and EN\_1 ='1') then

Q <= D;

end if;

end process;

end Behav;