

POLITECNICO DI TORINO



# RELAZIONE LABORATORIO 6

ELETTRONICA DEI SISTEMI DIGITALI

## GRUPPO B-14:

Leonardo Vencato  
Luca Sasselli  
Marco Pomponio  
Riccardo Midena

# PROGETTAZIONE

## ANALISI DELLA CONSEGNA

Data la richiesta di utilizzare un solo componente sommatore, è stata fin da subito evidente la necessità di iterare ciascun addendo di  $Y_n$  in colpi di clock successivi memorizzando il valore in un registro fino a calcolo completato. Per implementare questa operazione si è scelto di utilizzare una macchina di Mealey, poiché permette di ridurre drasticamente i colpi di clock necessari al completamento di una singola operazione e nel caso in questione non è sembrato complicare eccessivamente il progetto. Visto che la formula del filtro richiede 3 somme algebriche, salvo ottimizzazioni, saranno necessari 3 colpi di clock per ciascun risultato.

Si è tuttavia notato che tale formula, utilizzando bus da 8-bit, conduce ad overflow/underflow nel 75% dei casi. Si è quindi deciso di indirizzare le operazioni di ottimizzazione verso l'identificazione preventiva di questi eventi.

## OTTIMIZZAZIONE

Prima di tutto è stata eseguita un'analisi approfondita sulla formula del filtro: dal momento che è richiesto l'uso di un solo sommatore, e visto che il risultato deve essere scritto su 8 bit [-128; 127], si è modificata la formula facendo i seguenti raccoglimenti:

$$Y_n = 0.5Y_{n-1} - [0.25Y_{n-2} - (2X_{n-1} - 4X_n)]$$

Si nota che per via della dimensione degli addendi, è necessario usare un sommatore da 10 bit per evitare di incorrere in overflow già nel calcolo delle somme parziali.

Con questo arrangiamento viene eseguita per prima l'operazione più "critica", cioè quella che con più probabilità potrebbe portare il risultato fuori dal range di 8-bit. Infatti, continuando con l'analisi matematica:

$$\text{Se } (0.5Y_{n-1} = 0 \text{ AND } 0.25Y_{n-2} = 0) \rightarrow Y_n = (2X_{n-1} - 4X_n)$$

$$\text{Se } (2X_{n-1} - 4X_n) > 222 \rightarrow Y_n > 127 \rightarrow Y_n = 127$$

$$\text{Se } (2X_{n-1} - 4X_n) < -223 \rightarrow Y_n < -128 \rightarrow Y_n = -128$$

$$\text{Se } [0.25Y_{n-2} - (2X_{n-1} - 4X_n)] > 191 \rightarrow Y_n < -128 \rightarrow Y_n = -128$$

$$\text{Se } [0.25Y_{n-2} - (2X_{n-1} - 4X_n)] < -191 \rightarrow Y_n > 127 \rightarrow Y_n = 127$$

Inoltre si semplifica l'hardware visto che il filtro calcolerà soltanto sottrazioni. Con queste condizioni e considerando una distribuzione uniforme per i valori di  $X_n$  nel suo range, è possibile calcolare il valore di  $Y_n$  già alla prima operazione nel 57% circa dei casi, nella seconda nel 6% dei casi, e infine verranno eseguite tutte e tre le operazioni nel 37% delle volte. Questo algoritmo eseguirà in media 1.8095 operazioni per ogni  $X_n$ .

Tuttavia l'implementazione prevede l'uso di comparatori a 10 bit, componenti grandi e relativamente lenti dato che al loro interno devono eseguire una sottrazione. Inoltre, dato che le specifiche richiedono l'uso di un solo elemento in grado di eseguire i calcoli, ci si aspetta che il committente richieda un'ottimizzazione a livello di area e consumi. Per questo motivo si è scelto di modificare i valori di controllo da 222, -223, 191 e -191 rispettivamente a 255, -256, 255 e -256, facilmente implementabili con poche porte logiche.

Le prestazioni dell'algoritmo calano di poco a livello teorico, ma guadagna sicuramente in frequenza di clock non dovendo attendere il tempo di propagazione dei comparatori.

Un'ulteriore ottimizzazione, anche se sensibilmente meno importante, è che il circuito possa by-passare la logica prima introdotta nel caso in cui entrambi i valori di  $Y$  sono nulli. Tale circostanza si verifica sempre per il primo valore calcolato, tuttavia escluso questo si verifica in poco più dello 0.01% dei casi. Nonostante queste considerazioni si è deciso comunque di implementare questa piccola ottimizzazione poiché si verifica sempre una volta e non costituisce un dispendio di hardware significativo.

Le nuove prestazioni teoriche sono:

- $Y_n$  calcolato alla prima operazione nel 50% circa dei casi.
- $Y_n$  calcolato alla seconda operazione nel 2% circa dei casi.
- $Y_n$  calcolato alla terza operazione nel 48% circa dei casi.

Più precisamente l'algoritmo esegue in media 1.9734 operazioni per ogni  $X_n$ .

## VERIFICA

Per verificare il corretto funzionamento del circuito è stato scritto un testbench apposito che replica, con un'implementazione di alto livello, il comportamento del DUT fornendogli allo stesso tempo i valori di ingresso e i segnali di controllo. I valori in uscita vengono quindi confrontati con quelli elaborati dal testbench ed eventuali discrepanze vengono memorizzate in un file di log per facilitare il debug.

I dati in ingresso sono generati attraverso un algoritmo di xor-shifting su 8-bit che produce un flusso di dati pseudo-casuali consentendo una verifica di tipo Montecarlo. Sono state svolte simulazioni di sufficiente durata e con un sufficiente range di valori da poter essere sufficientemente sicuri del funzionamento del circuito.

Vari test hanno confermato l'efficienza del circuito confermando le aspettative teoriche: il circuito impiega 1024 colpi di clock per il caricamento, e una media di circa 2018 colpi di clock per elaborare tutti i dati.

## NOTA AGGIUNTIVA

Nel testo del progetto non è specificato il modo in cui un dispositivo esterno può leggere il contenuto della RAM\_B dopo l'elaborazione dei dati. A questo scopo si è reso disponibile all'esterno del filtro i bus DATA\_OUT, ADDRESS\_DATA\_OUT, e il segnale RD\_DATA\_OUT. Come si nota dallo schema a livello RT, sulla porta ADDRESS della RAM\_B vi è un multiplexer in modo da essere controllata dal circuito del filtro durante l'elaborazione dei dati (BUSY=1), e dall'esterno quando l'elaborazione è completa (BUSY=0).

# ELABORATI

## BREVE DESCRIZIONE DEL FILTRO REALIZZATO

La control unit del filtro ha soltanto 6 stati: standby, loading\_memory, operation\_1, operation\_2, operation\_3 e all\_done.

Nello stato di standby il filtro attende che il segnale START vada a 1, una volta campionato START la control unit passa nello stato di loading\_memory in cui il segnale LOAD\_MEM\_A va a 0, e viene abilitato l'address counter che aumentando di valore ad ogni colpo di clock permette il caricamento dei dati nella RAM\_A.

Quando il segnale COUNTER\_OVERFLOW viene campionato ad 1 tutti i 1024 dati sono stati caricati, e la control unit passa nello stato operation\_1. A questo punto i 3 stati di operation si alternano elaborando così i valori caricati:

- Durante operation\_1 i segnali di controllo si regolano in modo eseguire il calcolo  $2X_{n-1} - 4X_n$ .
- Durante operation\_2 viene eseguito  $0.25Y_{n-2} - Temp$ .
- Durante operation\_3 viene calcolato  $0.5Y_{n-1} - Temp$ .

In ognuna delle 3 fasi, quando il risultato si è propagato completamente generando i segnali di OVER-X e UNDER-X viene prodotto in modo combinatorio il segnale NEXT\_VALUE. Questo segnale indica quando è ad 1 che il calcolo del valore  $Y_n$  è completo secondo l'algoritmo descritto precedentemente. Al colpo di clock successivo il valore verrà caricato nella RAM\_B, l'address counter aumenterà di 1 estraendo il valore  $X_n$  successivo, e i registri si scambieranno i loro contenuti.

Tutto questo andrà avanti fin quando non verrà campionato un overflow dell'address counter: a questo punto tutti i valori sono stati elaborati e la control unit entrerà nello stato all\_done portando il segnale di DONE ad 1. Stato da cui uscirà quando START andrà a 0 e la control unit si resetterà tornando in standby.

Durante tutte le fasi di caricamento ed elaborazione il segnale START deve rimanere ad 1. Nel caso in cui venga campionato uno 0 il filtro interromperà immediatamente ciò che stava facendo tornando nello stato di standby.

## PSEUDOCODICE

```
Xn_1=0;
Yn_1=0;
Yn_2=0;
ADDR=0;

--attendi lo start
while START=0 do
end while;

--carica i valori
for ADDR=0 to ADDR=1023 do
    X[ADDR]=DATA_IN;
end for;

--elabora i dati
```

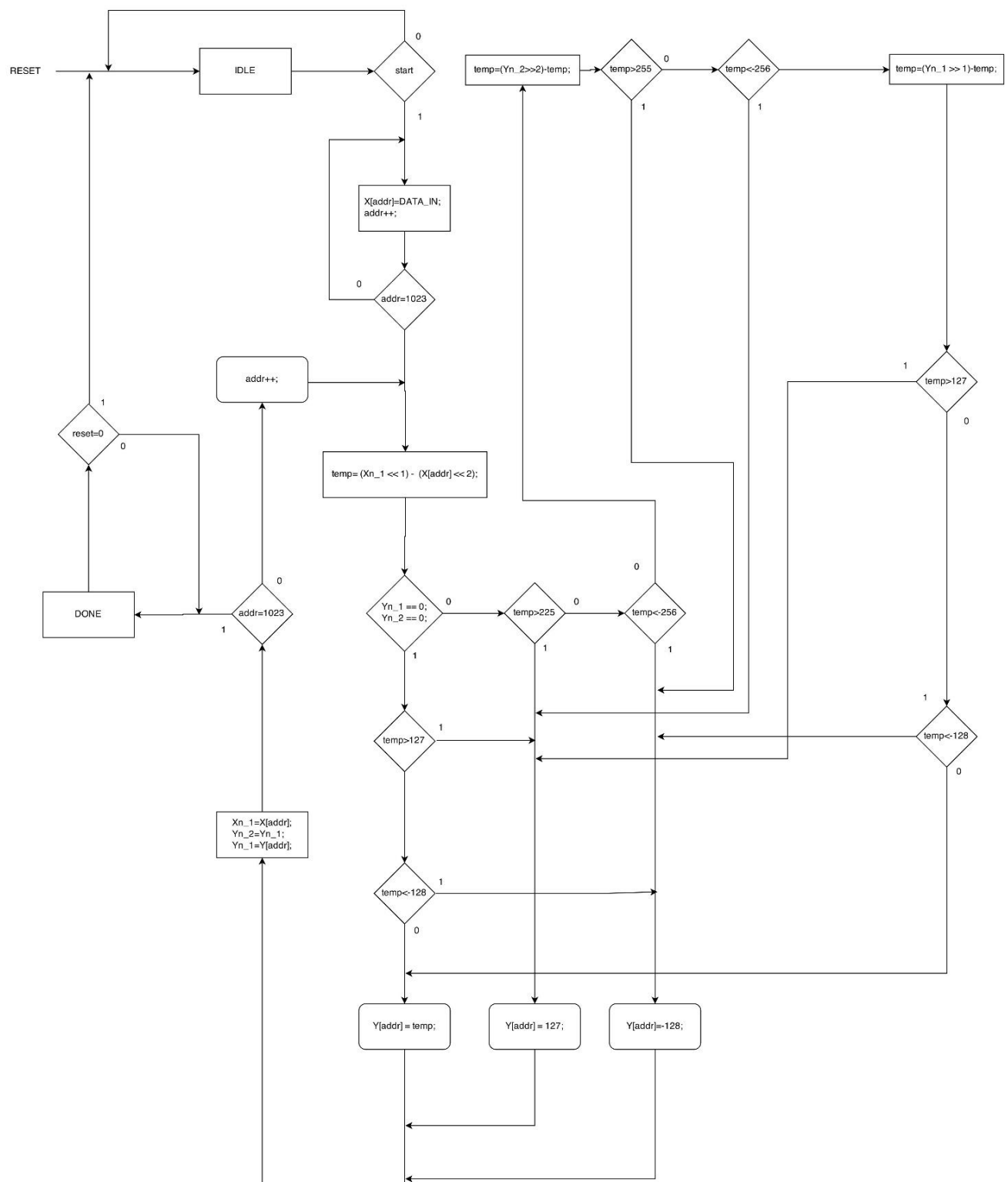
```

for ADDR=0 to ADDR=1023 do
    TEMP=(Xn_1<<1)-(X[ADDR]<<2);
    if Yn_1==0 and Yn_2==0 then -- se gli altri operandi sono 0 salta
        if TEMP>127 then
            Y[ADDR]=127;
        else if TEMP<-128 then
            Y[ADDR]=-128;
        else
            Y[ADDR]= TEMP;
        end if;
    else if TEMP>222 then -- TEMP>255
        Y[ADDR]=127;
    else if TEMP<-223 then -- TEMP<-256
        Y[ADDR]=-128;
    else
        TEMP=(Yn_2>>2)-TEMP;
        if TEMP>191 then -- TEMP>255
            Y[ADDR]=-128;
        else if TEMP<-191 then -- TEMP<-256
            Y[ADDR]=127;
        else
            TEMP=(Yn_1>>1)-TEMP;
            if TEMP>127 then
                Y[ADDR]=127;
            else if TEMP<-128 then
                Y[ADDR]=-128;
            else
                Y[ADDR]= TEMP;
            end if;
        end if;
    end if;
    Xn_1=X[ADDR];
    Yn_2=Yn_1;
    Yn_1=Y[ADDR];
end for;

--segnala che hai finito e attendi
DONE=1;
while START=1 do
end while;

```

# ASM CHART COMPORTAMENTALE

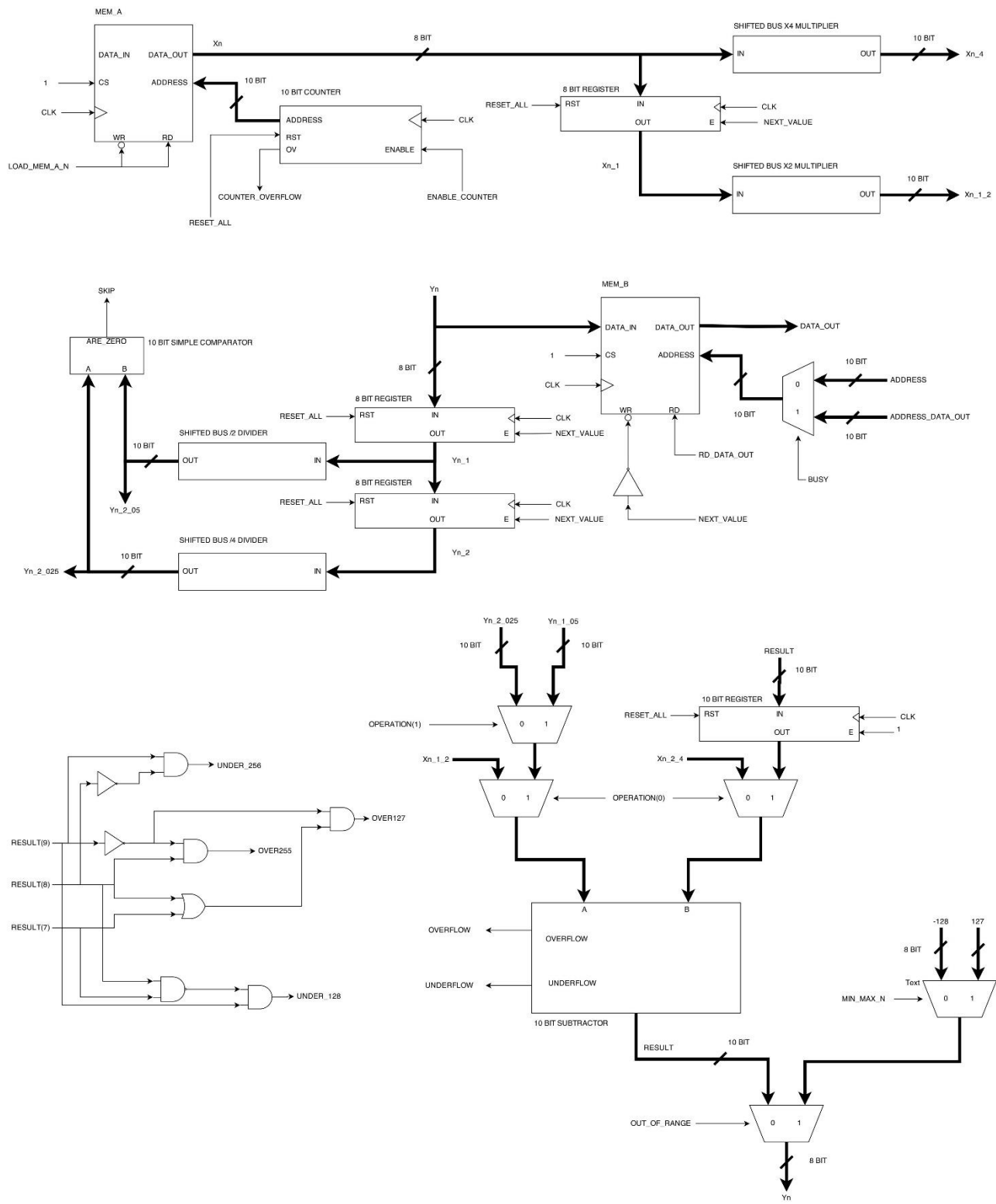


The diagram is a complex state machine with three main operational sections: OPERATION\_1, OPERATION\_2, and OPERATION\_3.

- OPERATION\_1 (Top):** Starts with a 'Reset' input leading to a state where `RESET_ALL_N=0`, `ENABLE_COUNTER=0`, `LOAD_MEM_A_N=1`, `NEXT_VALUE=0`, `OUT_OF_RANGE=0`, `MIN_MAXN=0`, `BUSY=0`, `DONE=0`, and `OPERATION=00`. A 'START' decision leads to 'LOADING\_MEMORY' or back to 'START'. 'LOADING\_MEMORY' sets `RESET_ALL_N=1`, `ENABLE_COUNTER=1`, `LOAD_MEM_A_N=0`, `NEXT_VALUE=0`, `OUT_OF_RANGE=0`, `MIN_MAXN=0`, `BUSY=1`, `DONE=0`, and `OPERATION=00`. It then checks for 'START' and 'COUNTER\_OVERFLOW'. If 'COUNTER\_OVERFLOW' occurs, it leads to a 'SKIP' decision. The 'SKIP' decision leads to either 'ENABLE\_COUNTER=1; NEXT\_VALUE=1' or 'ENABLE\_COUNTER=0; NEXT\_VALUE=0'. This is followed by a series of overflow/underflow checks for 127, 128, 255, and 256 bits, leading to 'OUT\_OF\_RANGE' or 'MIN\_MAXN' updates.
- OPERATION\_2 (Middle):** Starts with a 'START' decision leading to `RESET_ALL_N=1`, `LOAD_MEM_A_N=1`, `BUSY=1`, `DONE=0`, and `OPERATION=00`. It then checks for 'NEXT\_VALUE' and 'COUNTER\_OVERFLOW'. If 'COUNTER\_OVERFLOW' occurs, it leads to a decision on 'OVERFLOW=1' or 'UNDERFLOW=1' or 'OVER255=1' or 'UNDER\_256=1'. This leads to 'ENABLE\_COUNTER=1; NEXT\_VALUE=1; OUT\_OF\_RANGE=1' or 'ENABLE\_COUNTER=0; NEXT\_VALUE=0; OUT\_OF\_RANGE=0'.
- OPERATION\_3 (Bottom):** Starts with a 'START' decision leading to `RESET_ALL_N=1`, `ENABLE_COUNTER=0`, `NEXT_VALUE=0`, `LOAD_MEM_A_N=1`, `OUT_OF_RANGE=0`, `MIN_MAXN=0`, `BUSY=0`, `DONE=1`, and `OPERATION=00`. It then checks for 'START' and 'COUNTER\_OVERFLOW'. If 'COUNTER\_OVERFLOW' occurs, it leads to a decision on 'OVER255=1' or 'UNDERFLOW=1' or 'UNDER\_256=1' or 'OVERFLOW=1'. This leads to 'MIN\_MAXN=1' or 'MIN\_MAXN=0'.

The diagram includes various decision points (diamonds) and state transitions (arrows) based on the values of various signals like `ENABLE_COUNTER`, `NEXT_VALUE`, `OUT_OF_RANGE`, `MIN_MAXN`, and `DONE`.

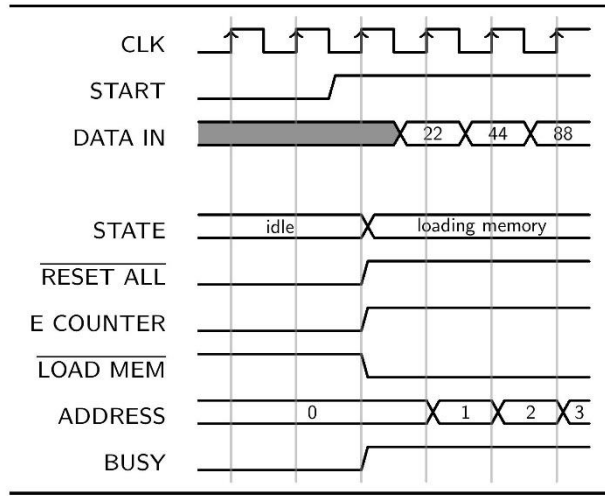
# SCHEMA RTL



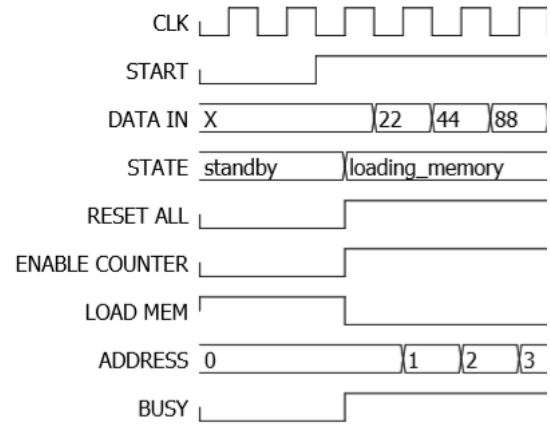


TIMING

CARICAMENTO DATI

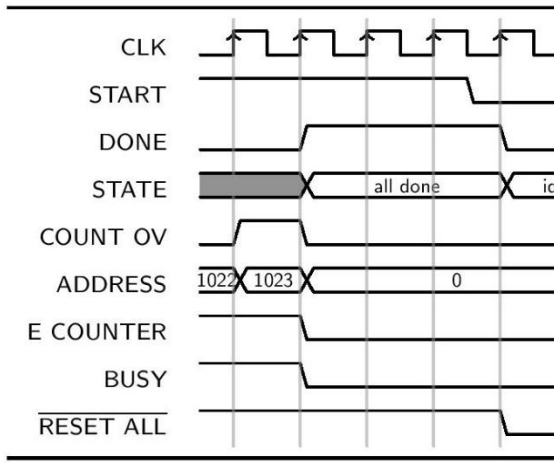


TIMING PREVISTO

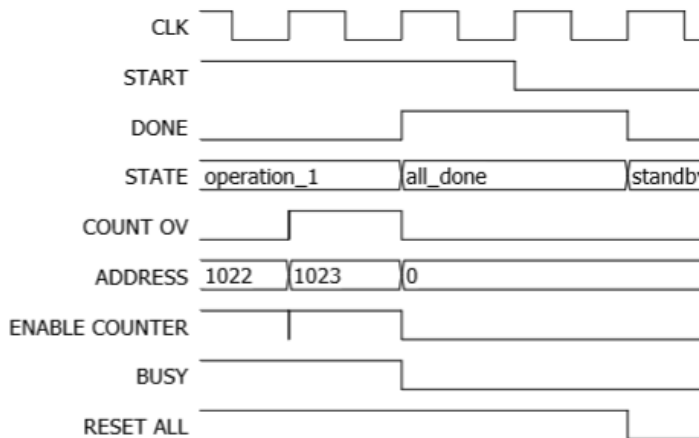


TIMING SIMULATO (MODELSIM)

FINE PROCESSO

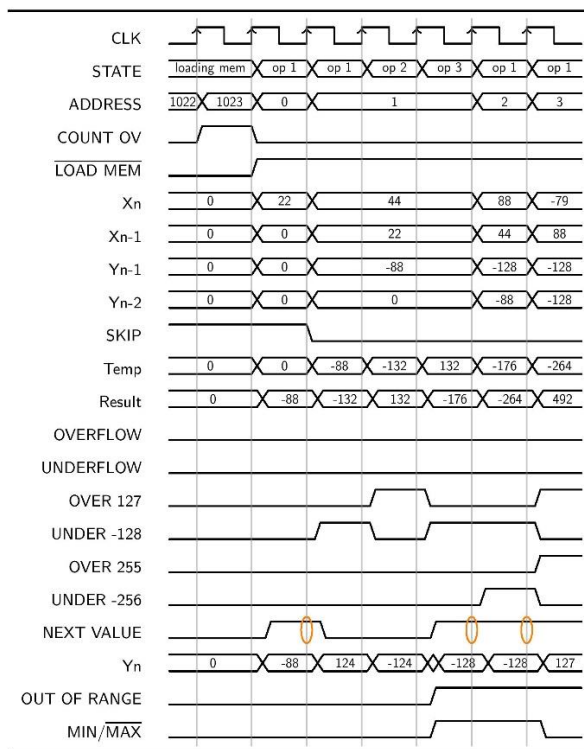


TIMING PREVISTO

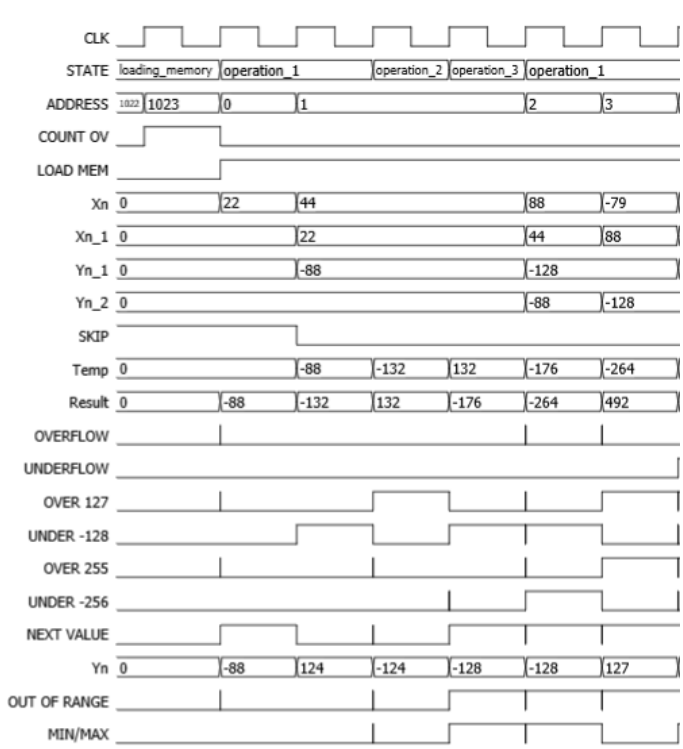


TIMING SIMULATO (MODELSIM)

## ELEBORAZIONE



TIMING PREVISTO



TIMING SIMULATO (MODELSIM)

Nel timing previsto, le zone cerchiate corrispondono agli istanti in cui  $Y_n$  è stato calcolato e salvato in memoria. Si noti come il primo e il terzo valore vengano calcolati in un solo colpo di clock mentre il secondo ne richieda tre.

## SORGENTE VHDL

### MAIN COMPONENT

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_misc.all;

ENTITY filter IS
PORT ( CLK, RSTN, START, RD_DATA_OUT : IN STD_LOGIC;
        DATA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        ADDRESS_DATA_OUT : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
        DATA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        DONE: OUT STD_LOGIC);
END filter;

ARCHITECTURE Structural OF filter IS

COMPONENT ram
    PORT (DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);

```

```

        ADDRESS: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
        CS, CLK, WRN, RD: IN STD_LOGIC;
        DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

COMPONENT register_nbit IS
    GENERIC (N: INTEGER := 8);
    PORT (R : IN STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          Clk, Rstn, E : IN STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR((N-1) DOWNTO 0));
END COMPONENT;

COMPONENT counter_nbit IS
    GENERIC (N: INTEGER := 10);
    PORT ( Clk, Clrn, E : IN STD_LOGIC;
          COUNT : BUFFER STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          MAX_VAL: OUT STD_LOGIC);
END COMPONENT;

COMPONENT subtractor_nbit IS
    GENERIC (N: INTEGER := 10);
    PORT ( A, B : IN STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          DIFF: BUFFER STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          OVERFLOW, UNDERFLOW: OUT STD_LOGIC);
END COMPONENT;

--CONTROLL SIGNALS
    --FROM CONTROL UNIT TO DATA PATH
    SIGNAL RESET_ALL_N, ENABLE_COUNTER, LOAD_MEM_A_N, NEXT_VALUE,
    OUT_OF_RANGE, MIN_MAXN, BUSY: STD_LOGIC;
    SIGNAL OPERATION: STD_LOGIC_VECTOR(1 DOWNTO 0);
    --FROM DATA PATH TO CONTROL UNIT
    SIGNAL OVER255, OVER127, UNDER_256, UNDER_128, OVERFLOW, UNDERFLOW,
    COUNTER_OVERFLOW, SKIP: STD_LOGIC;

--BUS
    SIGNAL Xn, Xn_1, Yn, Yn_1, Yn_2: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL Xn_4, Xn_1_2, Temp, Result, Yn_1_05, Yn_2_025, ADDRESS:
    STD_LOGIC_VECTOR(9 DOWNTO 0);

--OTHERS
    SIGNAL OPERAND_A, OPERAND_B, ADDRESS_MEMB: STD_LOGIC_VECTOR(9 DOWNTO 0);
    SIGNAL NEXT_VALUE_N: STD_LOGIC;

--CONTROL UNIT STATES
    TYPE State_type IS (STANDBY, LOADING_MEMORY, OPERATION_1, OPERATION_2,
    OPERATION_3, ALL_DONE);
    SIGNAL STATE, NEXT_STATE : State_type;

BEGIN

--DATA PATH

SKIP<=nor_reduce(Yn_1_05) AND nor_reduce(Yn_2_025);
OVER255<= (NOT Result(9)) AND Result(8);

```

```

OVER127<= (NOT Result(9)) AND (Result(8) OR Result(7));
UNDER_256<= Result(9) AND (NOT Result(8));
UNDER_128<= Result(9) AND (Result(8) NAND Result(7));
NEXT_VALUE_N<= NOT NEXT_VALUE;

--MEMORY DECLARATION
Mem_A: ram PORT MAP(DATA_IN, ADDRESS, '1', CLK, LOAD_MEM_A_N,
LOAD_MEM_A_N, Xn);
ADDRESS_MEMB<=ADDRESS WHEN BUSY='1' ELSE
                ADDRESS_DATA_OUT;
Mem_B: ram PORT MAP(Yn, ADDRESS_MEMB, '1', CLK, NEXT_VALUE_N,
RD_DATA_OUT, DATA_OUT);

--REGISTER DECLARATION
Reg_Xn_1: register_nbit GENERIC MAP(8) PORT MAP(Xn, CLK, RESET_ALL_N,
NEXT_VALUE, Xn_1);
Reg_Yn_1: register_nbit GENERIC MAP(8) PORT MAP(Yn, CLK, RESET_ALL_N,
NEXT_VALUE, Yn_1);
Reg_Yn_2: register_nbit GENERIC MAP(8) PORT MAP(Yn_1, CLK, RESET_ALL_N,
NEXT_VALUE, Yn_2);
Reg_Temp: register_nbit GENERIC MAP(10) PORT MAP(Result, CLK,
RESET_ALL_N, '1', Temp);

--ADDRESS COUNTER
Addr_Cont: counter_nbit GENERIC MAP(10) PORT MAP(CLK, RESET_ALL_N,
ENABLE_COUNTER, ADDRESS, COUNTER_OVERFLOW);

--MULTIPLICATION AND DIVISIONS
Xn_4<= Xn&"00";
Xn_1_2<= Xn_1(7)&Xn_1&"0";
Yn_1_05<=Yn_1(7)&Yn_1(7)&Yn_1(7)&Yn_1(7 DOWNTO 1);
Yn_2_025<=Yn_2(7)&Yn_2(7)&Yn_2(7)&Yn_2(7)&Yn_2(7 DOWNTO 2);

--ALU
OPERAND_A<=Xn_1_2 WHEN OPERATION(0)='0' ELSE
                Yn_2_025 WHEN OPERATION(1)='0' ELSE
                Yn_1_05 WHEN OPERATION(1)='1';
OPERAND_B<=Xn_4 WHEN OPERATION(0)='0' ELSE
                Temp;
Sub: subtractor_nbit GENERIC MAP(10) PORT MAP(OPERAND_A, OPERAND_B,
Result, OVERFLOW, UNDERFLOW);

--RESULT
Yn<=Result(7 DOWNTO 0) WHEN OUT_OF_RANGE='0' ELSE
        "01111111" WHEN MIN_MAXN='0' ELSE
        "10000000" WHEN MIN_MAXN='1';

--CONTROL UNIT
PROCESS (STATE, START, NEXT_VALUE, COUNTER_OVERFLOW) -- state table
BEGIN
    CASE STATE IS
        WHEN STANDBY=>
            IF (START='1') THEN
                NEXT_STATE<=LOADING_MEMORY;

```

```

        ELSE
            NEXT_STATE<=STANDBY;
        END IF;
    WHEN LOADING_MEMORY=>
        IF START='0' THEN
            NEXT_STATE<=STANDBY;
        ELSIF (COUNTER_OVERFLOW='1') THEN
            NEXT_STATE<=OPERATION_1;
        ELSE
            NEXT_STATE<=LOADING_MEMORY;
        END IF;
    WHEN OPERATION_1=>
        IF START='0' THEN
            NEXT_STATE<=STANDBY;
        ELSIF (NEXT_VALUE='1') THEN
            IF COUNTER_OVERFLOW='1' THEN
                NEXT_STATE<=ALL_DONE;
            ELSE
                NEXT_STATE<=OPERATION_1;
            END IF;
        ELSE
            NEXT_STATE<=OPERATION_2;
        END IF;
    WHEN OPERATION_2=>
        IF START='0' THEN
            NEXT_STATE<=STANDBY;
        ELSIF (NEXT_VALUE='1') THEN
            IF COUNTER_OVERFLOW='1' THEN
                NEXT_STATE<=ALL_DONE;
            ELSE
                NEXT_STATE<=OPERATION_1;
            END IF;
        ELSE
            NEXT_STATE<=OPERATION_3;
        END IF;
    WHEN OPERATION_3=>
        IF START='0' THEN
            NEXT_STATE<=STANDBY;
        ELSIF COUNTER_OVERFLOW='1' THEN
            NEXT_STATE<=ALL_DONE;
        ELSE
            NEXT_STATE<=OPERATION_1;
        END IF;
    WHEN ALL_DONE=>
        IF START='0' THEN
            NEXT_STATE<=STANDBY;
        ELSE
            NEXT_STATE<=ALL_DONE;
        END IF;
    WHEN OTHERS =>
        NEXT_STATE<=STANDBY;
    END CASE;
END PROCESS; -- state table

PROCESS (CLK) -- state flip-flops
BEGIN

```

```

    IF RSTN='0' THEN
        STATE<=STANDBY;
    ELSIF (CLK'EVENT AND CLK='1') THEN
        STATE<=NEXT_STATE;
    END IF;
END PROCESS;

PROCESS (STATE, OVER127, UNDER_128, OVER255, UNDER_256, OVERFLOW,
UNDERFLOW, SKIP)
BEGIN
    CASE STATE IS
        WHEN STANDBY=>
            RESET_ALL_N<='0';
            ENABLE_COUNTER<='0';
            LOAD_MEM_A_N<='1';
            NEXT_VALUE<='0';
            OUT_OF_RANGE<='0';
            MIN_MAXN<='0';
            BUSY<='0';
            DONE<='0';
            OPERATION<="00";
        WHEN LOADING_MEMORY=>
            RESET_ALL_N<='1';
            ENABLE_COUNTER<='1';
            LOAD_MEM_A_N<='0';
            NEXT_VALUE<='0';
            OUT_OF_RANGE<='0';
            MIN_MAXN<='0';
            BUSY<='1';
            DONE<='0';
            OPERATION<="00";
        WHEN OPERATION_1=>
            RESET_ALL_N<='1';
            IF (SKIP='1' OR OVERFLOW='1' OR UNDERFLOW='1' OR
OVER255='1' OR UNDER_256='1') THEN
                ENABLE_COUNTER<='1';
                NEXT_VALUE<='1';
            ELSE
                ENABLE_COUNTER<='0';
                NEXT_VALUE<='0';
            END IF;
            LOAD_MEM_A_N<='1';
            IF SKIP='1' THEN
                IF (OVER127='1' OR UNDER_128='1' OR
OVERFLOW='1' OR UNDERFLOW='1') THEN
                    OUT_OF_RANGE<='1';
                ELSE
                    OUT_OF_RANGE<='0';
                END IF;
                IF ((OVER127='1' OR OVERFLOW='1') AND
UNDERFLOW='0') THEN
                    MIN_MAXN<='0';
                ELSIF ((UNDER_128='1' OR UNDERFLOW='1') AND
OVERFLOW='0') THEN
                    MIN_MAXN<='1';
                ELSE

```

```

        MIN_MAXN<='0';
    END IF;
ELSE
    IF (OVER255='1' OR UNDER_256='1' OR
OVERFLOW='1' OR UNDERFLOW='1') THEN
        OUT_OF_RANGE<='1';
    ELSE
        OUT_OF_RANGE<='0';
    END IF;
    IF ((OVER255='1' OR OVERFLOW='1') AND
UNDERFLOW='0') THEN
        MIN_MAXN<='0';
    ELSIF ((UNDER_256='1' OR UNDERFLOW='1') AND
OVERFLOW='0') THEN
        MIN_MAXN<='1';
    ELSE
        MIN_MAXN<='0';
    END IF;
    END IF;
    BUSY<='1';
    DONE<='0';
    OPERATION<="00";
    WHEN OPERATION_2=>
        RESET_ALL_N<='1';
        IF (OVERFLOW='1' OR UNDERFLOW='1' OR OVER255='1' OR
UNDER_256='1') THEN
            ENABLE_COUNTER<='1';
            NEXT_VALUE<='1';
            OUT_OF_RANGE<='1';
        ELSE
            ENABLE_COUNTER<='0';
            NEXT_VALUE<='0';
            OUT_OF_RANGE<='0';
        END IF;
        LOAD_MEM_A_N<='1';
        IF ((OVER255='1' OR OVERFLOW='1') AND UNDERFLOW='0')
THEN
            MIN_MAXN<='1';
        ELSIF ((UNDER_256='1' OR UNDERFLOW='1') AND
OVERFLOW='0') THEN
            MIN_MAXN<='0';
        ELSE
            MIN_MAXN<='0';
        END IF;
        BUSY<='1';
        DONE<='0';
        OPERATION<="01";
        WHEN OPERATION_3=>
            RESET_ALL_N<='1';
            ENABLE_COUNTER<='1';
            NEXT_VALUE<='1';
            LOAD_MEM_A_N<='1';
            IF (OVERFLOW='1' OR UNDERFLOW='1' OR OVER127='1' OR
UNDER_128='1') THEN
                OUT_OF_RANGE<='1';
            ELSE
                OUT_OF_RANGE<='0';

```

```

        END IF;
        IF ((OVER127='1' OR OVERFLOW='1') AND UNDERFLOW='0')
THEN
        MIN_MAXN<='0';
        ELSEIF ((UNDER_128='1' OR UNDERFLOW='1') AND
OVERFLOW='0') THEN
        MIN_MAXN<='1';
        ELSE
        MIN_MAXN<='0';
        END IF;
        BUSY<='1';
        DONE<='0';
        OPERATION<="11";
        WHEN ALL_DONE=>
        RESET_ALL_N<='1';
        ENABLE_COUNTER<='0';
        NEXT_VALUE<='0';
        LOAD_MEM_A_N<='1';
        OUT_OF_RANGE<='0';
        MIN_MAXN<='0';
        BUSY<='0';
        DONE<='1';
        OPERATION<="00";
        END CASE;
    END PROCESS;

END Structural;

```

## RAM

```

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

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

ARCHITECTURE Structural OF ram IS
    TYPE Memory_Type IS ARRAY (1023 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL Memory: Memory_Type;
BEGIN

    PROCESS (CLK)
    BEGIN
        IF (CLK'EVENT AND CLK='1') THEN
            IF (CS='1' AND WRN='0') THEN
                Memory(to_integer(unsigned(ADDRESS)))<=DATA_IN;
            END IF;
        END IF;
    END PROCESS;
END Structural;

```



```

DATA_OUT<= Memory(to_integer(unsigned(ADDRESS))) WHEN (CS='1' AND RD='1')
ELSE
    (OTHERS=>'0');

END Structural;

```

## COUNTER

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY counter_nbit IS
    GENERIC (N: INTEGER := 10);
    PORT ( Clk, Clrn, E : IN STD_LOGIC;
          COUNT : BUFFER STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          MAX_VAL: OUT STD_LOGIC);
END counter_nbit;

ARCHITECTURE Structural OF counter_nbit IS

    COMPONENT Tflipflop
    PORT ( Clk, Rstn, T : IN STD_LOGIC;
          Q : BUFFER STD_LOGIC);
    END COMPONENT;

    SIGNAL FFIN: STD_LOGIC_VECTOR(N DOWNTO 0);

    BEGIN

        FFIN(0)<=E;
        FFIN(N DOWNTO 1)<= COUNT((N-1) DOWNTO 0) AND FFIN((N-1) DOWNTO 0);
        MAX_VAL<=FFIN(N);

        count_gen: FOR I IN 0 TO (N-1) GENERATE
            FF: Tflipflop PORT MAP (Clk, Clrn, FFIN(I), COUNT(I));
        END GENERATE;
    END Structural;

```

## REGISTER

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY register_nbit IS
    GENERIC (N: INTEGER :=8);
    PORT (R : IN STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          Clk, Rstn, E: IN STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR((N-1) DOWNTO 0));
END register_nbit;

ARCHITECTURE Behavior OF register_nbit IS
    BEGIN
        PROCESS (Clk, Rstn)
            BEGIN

```

```

        IF (Rstn = '0') THEN
            Q <= (OTHERS => '0');
        ELSIF (Clk'EVENT AND Clk = '1') THEN
            IF E='1' THEN
                Q <= R;
            END IF;
        END IF;
    END PROCESS;
END Behavior;

```

## SUBTRACTOR

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_misc.all;

ENTITY subtractor_nbit IS
    GENERIC (N: INTEGER := 10);
    PORT ( A, B : IN STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          DIFF: BUFFER STD_LOGIC_VECTOR((N-1) DOWNTO 0);
          OVERFLOW, UNDERFLOW: OUT STD_LOGIC);
END subtractor_nbit;

ARCHITECTURE Structural OF subtractor_nbit IS

    COMPONENT single_full_adder
        PORT (A, B, CI: IN STD_LOGIC;
              S, CO: OUT STD_LOGIC);
    END COMPONENT;

    SIGNAL CARRY: STD_LOGIC_VECTOR(N DOWNTO 0);
    SIGNAL B_N: STD_LOGIC_VECTOR((N-1) DOWNTO 0);
    SIGNAL OV: STD_LOGIC;

    BEGIN

        B_N<= NOT B;
        CARRY(0)<='1';

        fulladd_gen: FOR I IN 0 TO (N-1) GENERATE
            FF: single_full_adder PORT MAP (A(I), B_N(I), CARRY(I), DIFF(I),
            CARRY(I+1));
        END GENERATE;

        OV<=CARRY(N) XOR CARRY(N-1);
        OVERFLOW<= (NOT A(N-1)) AND OV;
        UNDERFLOW<= A(N-1) AND OV;

    END Structural;

```

## FULL ADDER

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

```

```

ENTITY single_full_adder IS
    PORT (A, B, CI: IN STD_LOGIC;
          S, CO: OUT STD_LOGIC);
END single_full_adder;

ARCHITECTURE add OF single_full_adder IS
SIGNAL G: STD_LOGIC;
BEGIN
    G <= A XOR B;
    S <= G XOR CI;
    CO <= CI WHEN G='1' ELSE
        B;
END add;

```

## T FLIP-FLIP

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Tflipflop IS
PORT ( Clk, Rstn, T : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC);
END Tflipflop;

ARCHITECTURE Structural OF Tflipflop IS
BEGIN
    PROCESS (Clk, Rstn)
    BEGIN
        IF (Rstn='0') THEN
            Q<='0';
        ELSIF (Clk='1' AND Clk'EVENT) THEN
            IF(T='1') THEN
                IF(Q='0') THEN
                    Q<='1';
                ELSE
                    Q<='0';
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Structural;

```

## TEST-BENCH

```

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

ENTITY filter_testbench IS

END;

ARCHITECTURE Behaviour OF filter_testbench IS
    TYPE Ram IS ARRAY(1023 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);

```

```

CONSTANT CLK_period : TIME := 1 ms;
CONSTANT delay_short : TIME := 1 ns;
CONSTANT delay_long : TIME := 1 ms;
CONSTANT seed : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001011";

SIGNAL CLK, RSTN, START, RD_DATA_OUT, DONE : STD_LOGIC;
SIGNAL DATA_IN, DATA_OUT : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL ADDRESS_DATA_OUT : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL RAM1, RAM2 : Ram;

COMPONENT filter IS
    PORT (CLK, RSTN, START, RD_DATA_OUT : IN STD_LOGIC;
          DATA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          ADDRESS_DATA_OUT : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
          DATA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
          DONE : OUT STD_LOGIC);
END COMPONENT;

BEGIN
    dut : filter PORT MAP (CLK, RSTN, START, RD_DATA_OUT, DATA_IN,
ADDRESS_DATA_OUT, DATA_OUT, DONE);

    clock_gen : PROCESS
        BEGIN
            CLK <= '0';
            WAIT FOR CLK_period/2;
            CLK <= '1';
            WAIT FOR CLK_period/2;
        END PROCESS;

    run : PROCESS
        FILE outfile : TEXT IS OUT "outfile.txt";
        VARIABLE buf : LINE;
        VARIABLE a, b, c, d, shift_a, shift_b, errors, temp_hits,
result : INTEGER := 0;
        VARIABLE average : REAL;
        VARIABLE rand_temp : STD_LOGIC_VECTOR(7 DOWNTO 0) :=
seed;

        VARIABLE temp : STD_LOGIC := '0';
        VARIABLE temp_div : SIGNED(7 DOWNTO 0);

        BEGIN
            START <= '0';
            RSTN <= '1';
            RD_DATA_OUT <= '0';

            errors := 0;
            temp_hits := 0;

            WAIT FOR delay_long;

            RSTN <= '0';
            WAIT FOR delay_short;
            RSTN <= '1';

            WAIT FOR delay_long;

            START <= '1';

```

```

        WAIT FOR delay_long;

        FOR i IN 0 TO 1023 LOOP
            temp := rand_temp(rand_temp'LENGTH-1)
XOR rand_temp(rand_temp'LENGTH-2);
            rand_temp(rand_temp'LENGTH-1 DOWNTO 1)
:= rand_temp(rand_temp'LENGTH-2 DOWNTO 0);
            rand_temp(0) := temp;

            RAM1(i) <= rand_temp;
            DATA_IN <= rand_temp;

            WAIT FOR delay_long;
        END LOOP;

        WHILE DONE='0' LOOP
            WAIT UNTIL CLK='1';
            temp_hits := temp_hits + 1;
        END LOOP;

        START<='0';
        RD_DATA_OUT <= '1';

        FOR i IN 0 TO 1023 LOOP
            ADDRESS_DATA_OUT <=
STD_LOGIC_VECTOR(TO_UNSIGNED(i, ADDRESS_DATA_OUT'LENGTH));
            WAIT FOR delay_long;
            RAM2(i) <= DATA_OUT;
            WAIT FOR delay_long;
        END LOOP;

        FOR i IN 0 TO 1023 LOOP
            IF i>1 THEN
                b:=TO_INTEGER(SIGNED(RAM2(i-2)));
            ELSE
                b:=0;
            END IF;

            IF i>0 THEN
                a:=TO_INTEGER(SIGNED(RAM2(i-1)));
                d:=TO_INTEGER(SIGNED(RAM1(i-1)));
            ELSE
                a:=0;
                d:=0;
            END IF;

            c:=TO_INTEGER(SIGNED(RAM1(i)));

            temp_div:=to_signed(a, 8);
            shift_a:=to_integer(temp_div(7)&temp_div(7
downto 1));

            temp_div:=to_signed(b, 8);

            shift_b:=to_integer(temp_div(7)&temp_div(7)&temp_div(7 downto 2));

            result:=shift_a-shift_b-4*c+2*d;
            IF result>127 THEN
                result:=127;
            ELSIF result<-128 THEN
                result:=-128;

```

```

        END IF;

        IF result/=TO_INTEGER(SIGNED(RAM2(i))) THEN
            write(buf, STRING("ERROR: cell "));
            write(buf, i);
            write(buf, STRING(" is "));
            write(buf, TO_INTEGER(SIGNED(RAM2(i))));
            write(buf, STRING(", should be "));
            write(buf, result);
            WRITELINE (outfile, buf);
            errors := errors + 1;
        END IF;
    END LOOP;

    write(buf, STRING("TOTAL ERRORS: "));
    write(buf, errors);
    WRITELINE (outfile, buf);
    write(buf, STRING("The algorithm took "));
    write(buf, temp_hits);
    write(buf, STRING(" clock hits to be completed,
with an average of "));
    average := REAL(temp_hits)/1024.0;
    write(buf, average);
    write(buf, STRING(" operations per memory cell."));
    WRITELINE (outfile, buf);

    WAIT FOR delay_long*20;
END PROCESS;

END;

```