

A.A. 2023-24

# LABORATORIO 6 ESD

## ASM – Automatic Irrigation System

Data: 08/05/2024

Gruppo: C10 – Fabio Calabrese [294296], Marc Helou [293359],  
Alessandro Spongano [286129]

L'implementazione del controllore automatico d'irrigazione proposta si basa sul seguente pseudocodice:

```
cnt_1 = 0
for cnt_1 in 0 to 1023
    MEM_A(cnt_1) = DATA_EXT
    MEM_B(cnt_1) = 0
    cnt_1 += 1
end for
cnt_1 = 0
cnt_2 = 0
flag_counter = 0
IRRIGATION_ALARM = 0
data_sample_20 = MEM_A(cnt_1 + 1)&MEM_A(cnt_1)
data_sample_40 = MEM_A(cnt_1 + 3)&MEM_A(cnt_1 + 2)
cnt_1 += 4
while cnt_2 < 1023
    average = (data_sample_20 + data_sample_40)/2
    MEM_B(cnt_2) = average(7 downto 0)
    MEM_B(cnt_2 + 1) = average(15 downto 8)
    IF |data_sample_40| >= |data_sample_20|
        highest = data_sample_40
    ELSE
```

```

highest = data_sample_20

MEM_B(cnt_2 + 2) = highest(7 downto 0)
MEM_B(cnt_2 + 3) = highest(15 downto 8)

IF |average| > |threshold|
    IF flag_counter < 10
        flag_counter += 1
        IF flag_counter = 10
            IRRIGATION_ALARM = 1
    ELSE
        flag_counter = 0

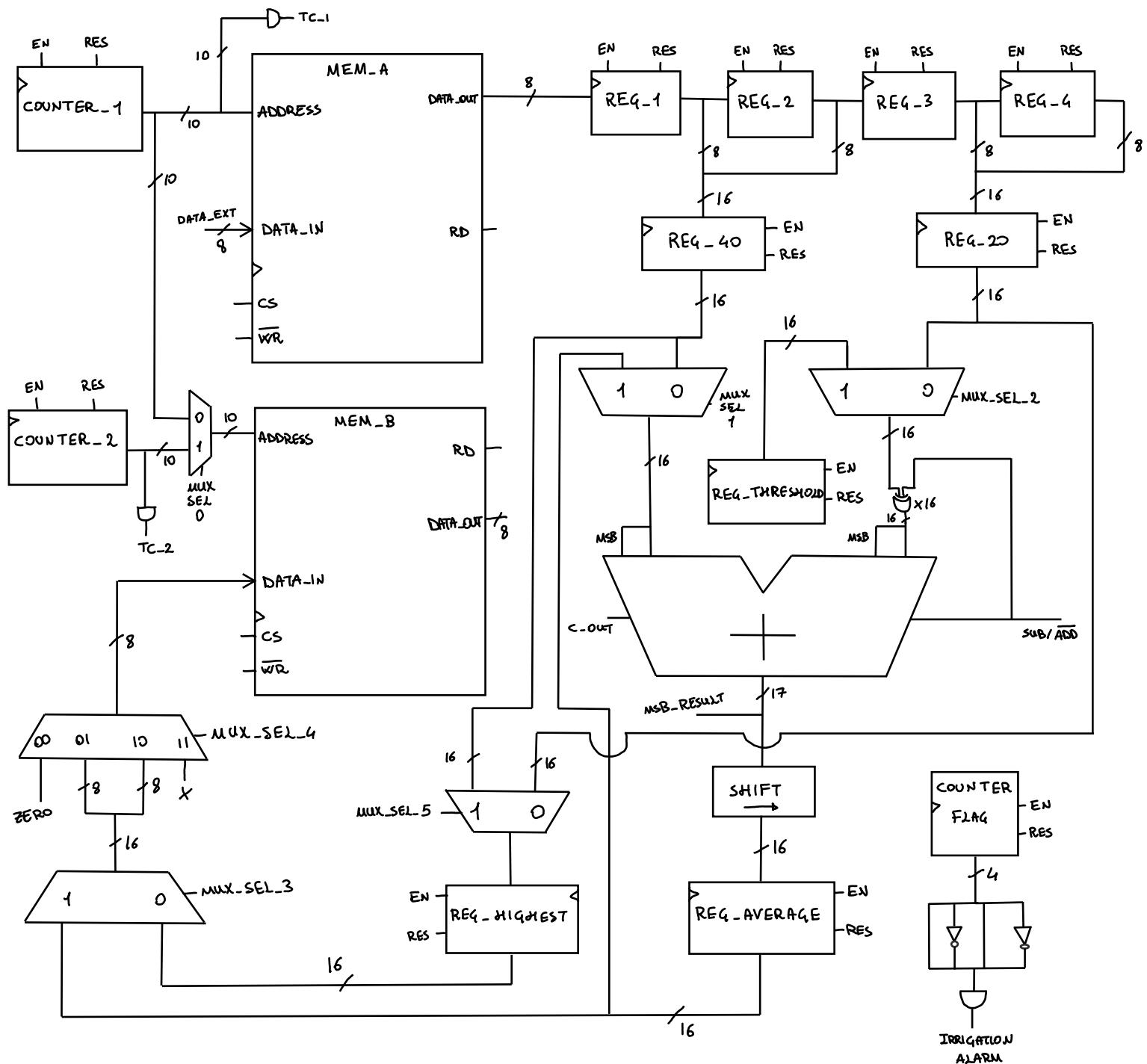
data_sample_20 = MEM_A(cnt_1 + 1)&MEM_A(cnt_1)
data_sample_40 = MEM_A(cnt_1 + 3)&MEM_A(cnt_1 + 2)
cnt_1 += 4
cnt_2 += 4

end while

```

Il codice è stato tradotto in un circuito con il seguente Datapath:

Datapath:



Al fine di comprendere la logica dell'algoritmo e del controllo del circuito mostrato, si considerino i seguenti come i valori predefiniti dei segnali di controllo:

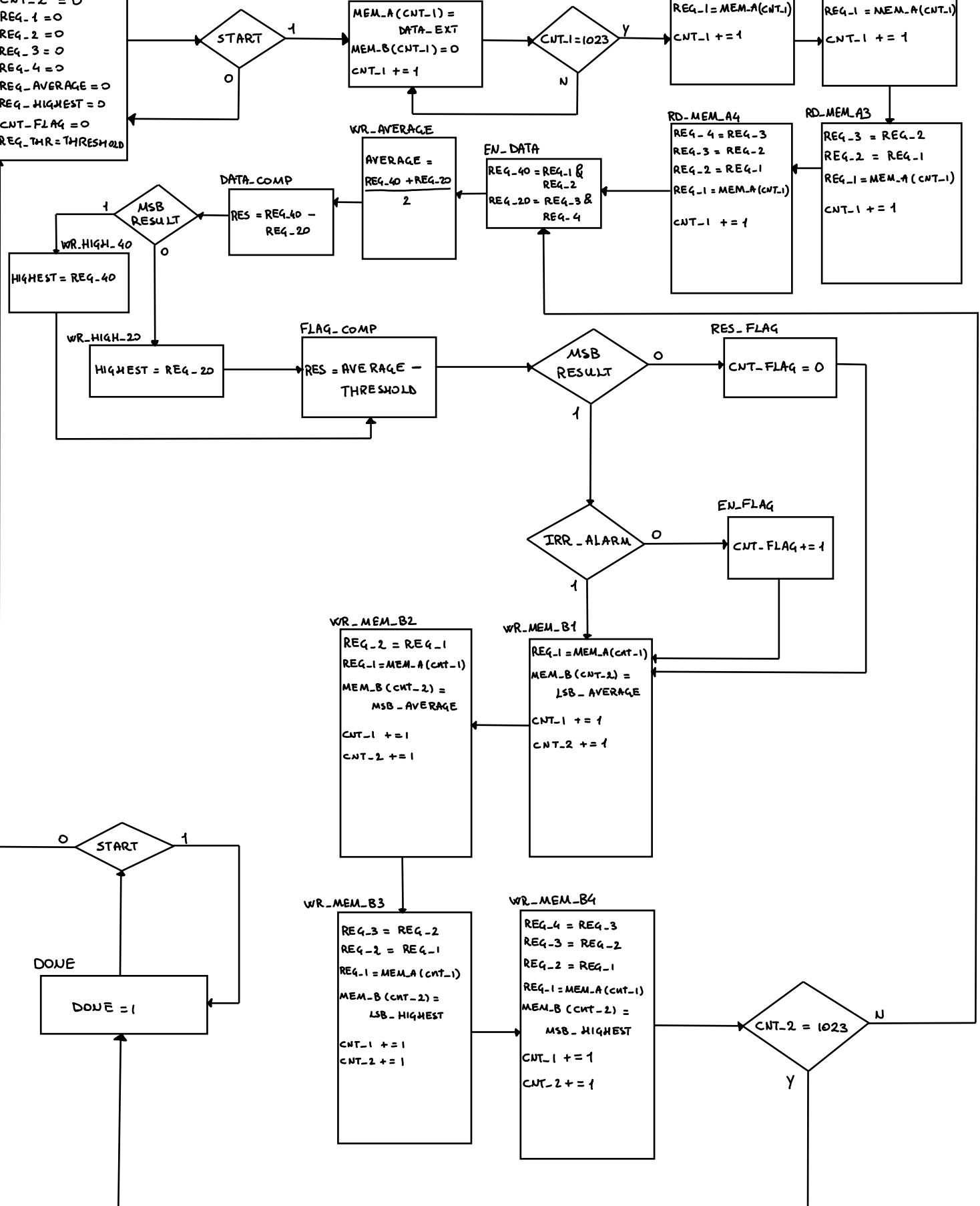
```
cs_a <= '0'; cs_b <= '0'; wr_a_n <= '1'; wr_b_n <= '1'; rd_a <= '0'; rd_b <= '0';
en_reg1 <= '0'; en_reg2 <= '0'; en_reg3 <= '0'; en_reg4 <= '0'; en_reg40 <= '0'; en_reg20 <= '0';
mux_sel_0 <= '0'; mux_sel_1 <= '0'; mux_sel_2 <= '0'; mux_sel_3 <= '0'; mux_sel_4 <= "00"; mux_sel_5 <= '0'; sub_add_n <= '0';
en_average <= '0'; en_thr <= '0'; en_high <= '0'; en_cnt_1 <= '0'; en_cnt_2 <= '0'; en_cnt_flag <= '0'; reset_flag_counter <= '0';
done_sig <= '0'; reset_DP <= '0';
```

Di seguito si riportano le ASM chart che descrivono l'algoritmo implementato dal circuito e i controlli che vengono cambiati ed attivati dalla macchina in ogni particolare stato. Nella ASM chart di controllo all'interno di uno stato sono stati riportati solamente i segnali che cambiano rispetto al valore predefinito:

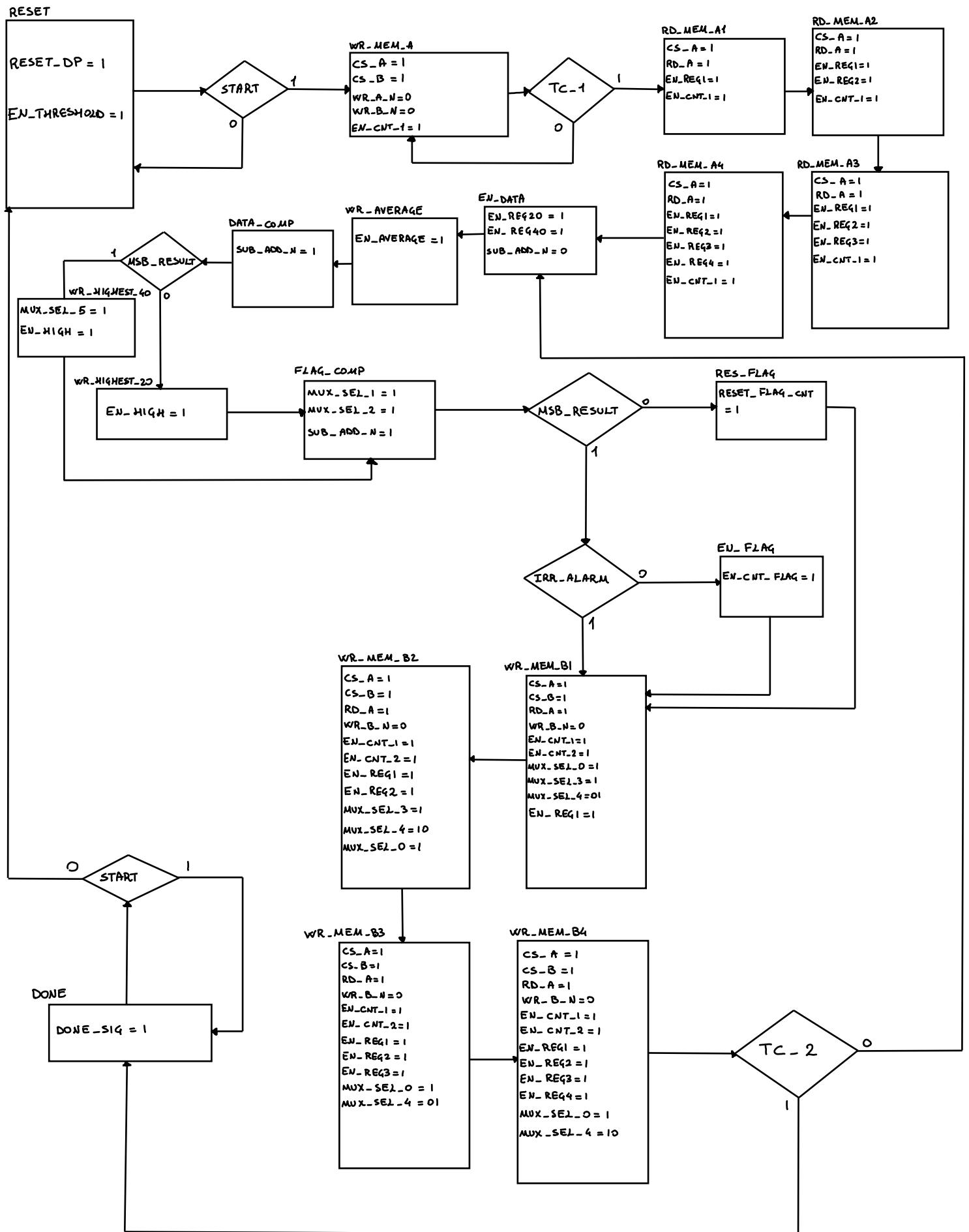
# ASM chart algoritmica

RESET

```
CNT_1 = 0
CNT_2 = 0
REG_1 = 0
REG_2 = 0
REG_3 = 0
REG_4 = 0
REG_AVERAGE = 0
REG_HIGHEST = 0
CNT_FLAG = 0
REG_THR = THRESHOLD
```



# ASM chart di controllo



La macchina a stati finiti di Moore che controlla il circuito proposto si compone di un totale di 19 stati. Inizialmente si riempie la MEM\_A con i dati provenienti dall'esterno rappresentanti il potenziale matriciale misurato. Ciclicamente li si legge, quattro “parole” alla volta, per effettuarne la media, determinarne il più negativo ed eventualmente registrare una media superiore (in modulo) alla soglia in arrivo dall'esterno e memorizzata su un registro a sola lettura. I dati rappresentanti la media e la misura più negativa vengono infine scritti nella MEM\_B mentre nuovi dati vengono estratti dalla MEM\_A per una nuova valutazione. L'algoritmo si conclude una volta che l'intera MEM\_B viene riempita.

Una volta registrato il valore attivo di START, il circuito inizia il riempimento delle memorie a partire dal colpo di clock successivo sfruttando la scrittura sincrona dei register file. I dati in arrivo dall'esterno sono sincronizzati con il fronte di salita del clock e vengono registrati un byte per colpo di clock all'interno della MEM\_A. Lo stesso counter\_1 che fa da indirizzo alle locazioni della MEM\_A viene usato per inizializzare a 0 tutte le locazioni della MEM\_B durante questa fase, seppur non espressamente richiesto.

Terminata la fase iniziale di riempimento delle memorie inizia il lavoro sui dati. Sfruttando la lettura asincrona dei register file, si passano i dati (parole da 8 bit STD\_LOGIC\_VECTOR) a partire dalla locazione 0 della MEM\_A in una pipeline di quattro registri da 8 bit che aiuteranno a parallelizzare correttamente i valori SIGNED da 16 bit. Da notare è che anche nell'ultimo stato di lettura della MEM\_A (RD\_MEM\_A4) viene tenuto ad ‘1’ l'enable del counter\_1 in modo che al prossimo giro di lettura il dato nella locazione 4 sia immediatamente disponibile alla registrazione nel REG\_1.

I dati in uscita ai quattro registri vengono parallelizzati e portati dentro ai due registri REG\_40 e REG\_20 che conterranno i dati del potenziale della prima misurazione SIGNED su 16 bit.

La prima operazione effettuata sui dati è il calcolo della media. I dati vengono estesi con segno su 17 bit, il dato in ingresso al sommatore da destra rappresentante la misura a -20 cm viene prima passato in una batteria di XOR col segnale di SUB\_ADD\_N che per il calcolo della media è impostato a ‘0’. L'estensione del parallelismo dei dati e della loro somma è la misura precauzionale che è stata adottata per evitare overflow e ottenere sempre un risultato di somma corretto; infatti, sommando due numeri con lo stesso segno il risultato potrebbe non essere rappresentabile sui 16 bit originali. In uscita al blocco combinatorio del sommatore viene effettuata la divisione per due del risultato, perdendo il LSB tramite un semplice shift logico del bus (senza uno shifter fisico o alcun gate). Il dato rappresentante la media arriva infine al REG\_AVERAGE e viene registrato al fronte di clock successivo.

A questo punto vengono effettuati due confronti sfruttando il sommatore. Il segnale di SUB\_AND\_N viene impostato ad ‘1’ in modo che si effettui la sottrazione in complemento a 2 tra il dato misurato a -40 cm e quello misurato a -20 cm. In uscita al sommatore si avrà il risultato della sottrazione e in modo da decidere quale dei due dati sia il più negativo viene inviato alla macchina di controllo il MSB del risultato che fa quindi da status flag. In caso sia pari ad ‘1’, significa che il sottraendo (dato a -20 m) è minore in modulo del minuendo (dato a -40) m e quindi quest'ultimo viene inviato al registro REG\_HIGHEST che al fronte successivo lo registrerà.

Si è arrivati a scegliere il MSB del risultato della sottrazione come segnale di status per la macchina poiché nonostante l'estensione di segno e il caso “critico” in cui uno dei dati da confrontare fosse ‘0’, esso indicava sempre in maniera corretta l'avvenuto confronto.

La stessa operazione di confronto viene quindi effettuata tra la media e la soglia girando adeguatamente i mpx\_1 e mpx\_2 e tenendo ad ‘1’ il SUB\_AND\_N. Nel caso di una media misurata maggiore in modulo della soglia viene data l’abilitazione al counter\_flag a meno che il segnale di status IRRIGATION\_ALARM non sia già attivo, in questo caso il contatore avrà già raggiunto “1010” e quindi saranno state misurate 10 medie superiori alla soglia consecutive.

È da notare che durante l’uso del sommatore per le somme e i confronti, il segnale di status del MSB del risultato cambia in maniera puramente combinatoria in base ai dati in ingresso al sommatore, il suo valore può quindi essere usato immediatamente per la decisione da effettuare. Nelle due fasi di confronto si otterranno risultati “spuri” in uscita al blocco combinatorio e naturalmente non verranno registrati nel REG\_AVERAGE, l’unico registro connesso al sommatore.

Nell’ultima fase dell’algoritmo si usano quattro colpi di clock per scrivere in ordine la media e il valore più negativo della prima misurazione usando il counter\_2 come indirizzo alla MEM\_B e impostando il mpx\_4 in modo che passino gli 8 LSB prima e gli 8 MSB dopo del dato da registrare in memoria. Durante gli stessi quattro giri di clock vengono estratti dalla MEM\_A i dati ottenuti durante la misurazione successiva, come nei primi quattro stati dell’algoritmo, in modo da poter subito effettuarne l’elaborazione.

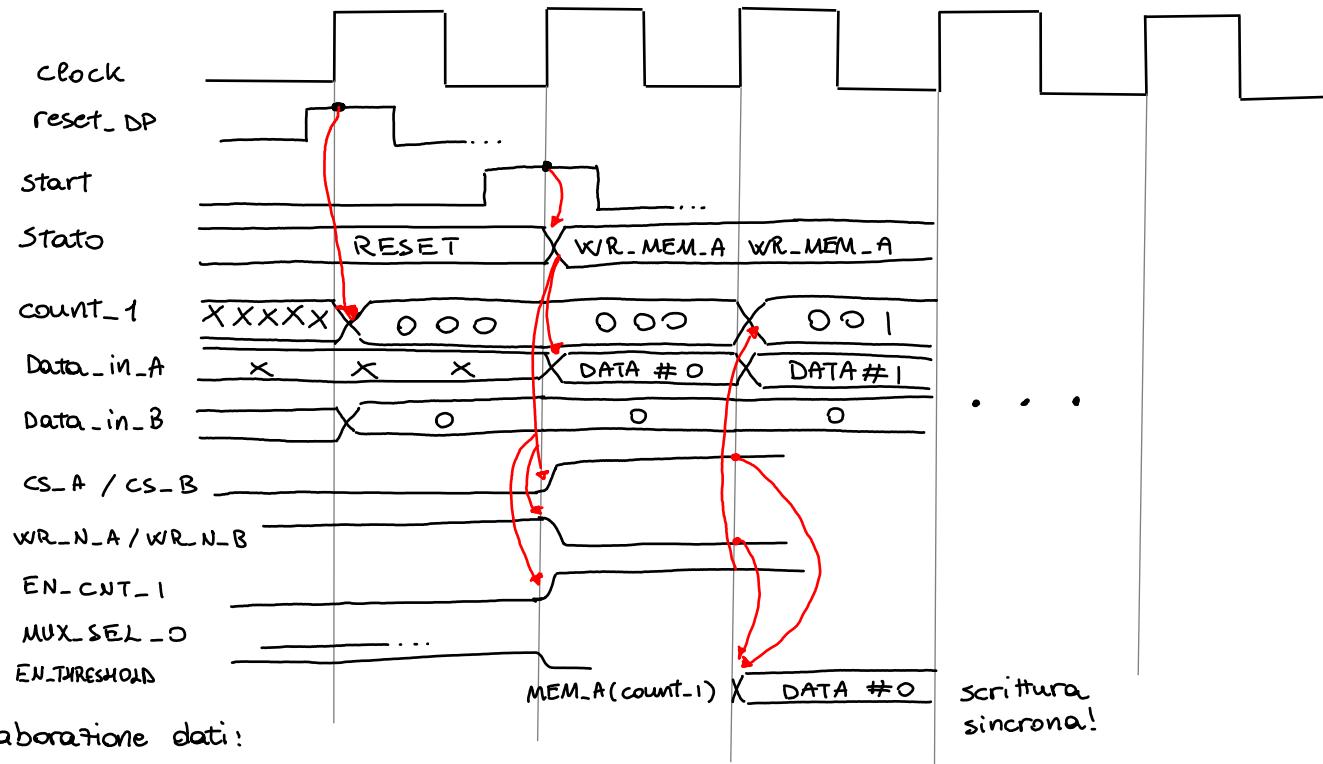
Una volta riempita la MEM\_B viene portato ad ‘1’ il DONE\_SIG e si resta in attesa che il segnale di START torni a ‘0’ per ricominciare l’algoritmo. Durante l’ultima scrittura in MEM\_B il counter\_1 ricomincia da 0 il suo conteggio e punta nuovamente alle prime quattro locazioni della MEM\_A ma ciò non inficia ovviamente alla conclusione dell’algoritmo.

Nello stato di RESET viene asserito ad ‘1’ il segnale RESET\_DP che inizializza ogni registro e contatore prima di una nuova elaborazione dei dati. In aggiunta, è stato considerato un segnale di reset “utente” in ingresso alla macchina che può interrompere in qualsiasi momento il funzionamento della stessa. L’unico registro che viene abilitato durante questo stato è quello contenente la soglia, in modo che il suo valore sia disponibile alla partenza delle operazioni successive. Tale registro può essere resettato solo manualmente dall’utente.

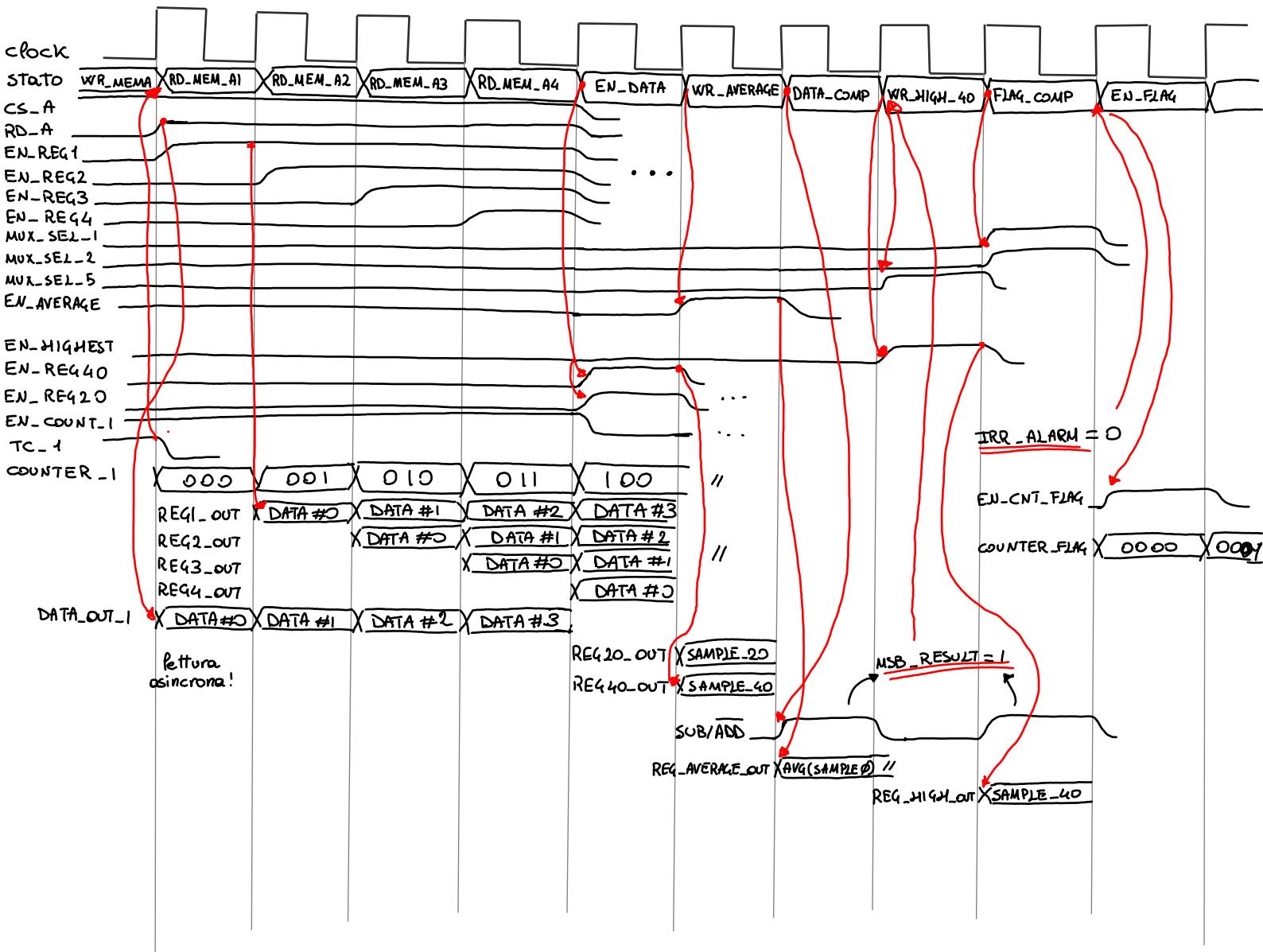
Di seguito si riporta il timing manuale delle principali fasi dell’algoritmo:

- Partenza e registrazione dati:

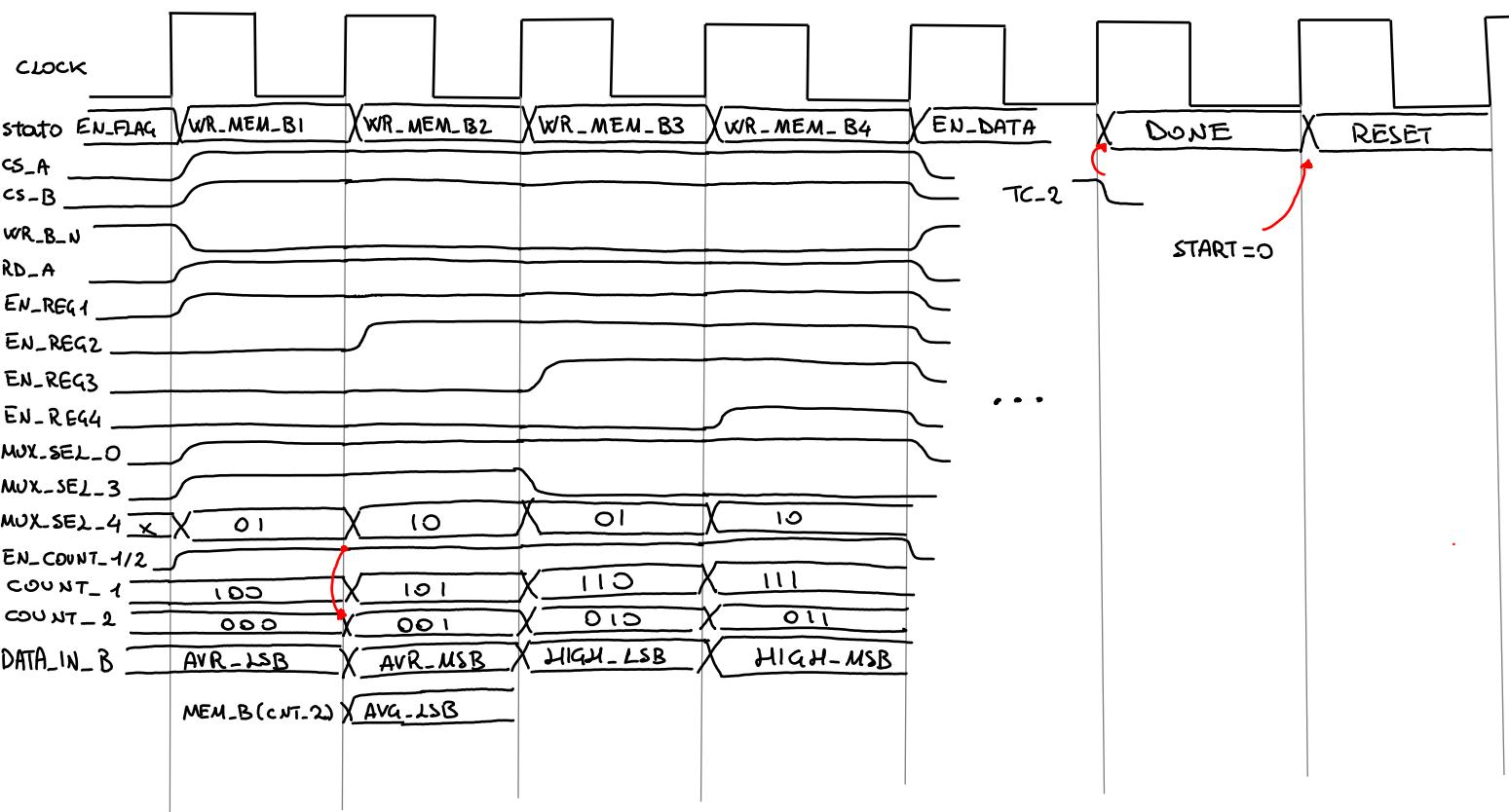
## Timing:



- Elaborazione dati:



- Scrrittura dei dati e fine dell'algoritmo



Nel disegnare il timing manuale si è considerato che il MSB della sottrazione durante entrambi i confronti fosse uguale ad '1' e che il flag di allarme non fosse ancora attivo.

Si riportano di qui i codici VHDL usati per l'implementazione della macchina. Nella top-level entity della macchina si può notare che l'IRRIGATION\_ALARM è stato implementato come segnale BUFFER poiché è un'uscita del circuito e deve essere usato per una decisione dell'algoritmo.

### - T-FlipFlop:

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 ENTITY t_flipflop IS
5 PORT (
6   T : IN STD_LOGIC;
7   clk,reset : IN STD_LOGIC;
8   Q : BUFFER STD_LOGIC
9 );
10 END t_flipflop;
11
12 ARCHITECTURE behavior OF t_flipflop IS
13
14 BEGIN
15
16   ff : PROCESS(T, clk,reset)
17
18   BEGIN
19     IF (reset = '1') THEN      -- reset asincrono
20       Q <= '0';
21
22     ELSIF (clk'EVENT AND clk = '1') THEN
23       IF ( T = '1' ) THEN
24         Q <= NOT Q;
25       END IF;
26
27     END IF;
28
29   END PROCESS;
30
31 END behavior;
32
33
34
35
36
37

```

### - Multiplexer 2 a 1

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4 --creazione del multiplexer2to1 a partire da entity:
5
6 ENTITY multiplexer2to1 IS
7
8 PORT ( x, y : IN STD_LOGIC ; -- voglio un ingresso x,y a 1 bit
9        s : IN STD_LOGIC;
10        m: OUT STD_LOGIC); -- voglio che l'uscita sia x o y
11
12 END multiplexer2to1;
13
14 -- definizione dell'architettura
15
16 ARCHITECTURE behavior OF multiplexer2to1 IS
17 BEGIN
18
19   m <= (NOT (s) AND x) OR (s AND y); -- struttura interna
20
21 END behavior;

```

### - Full Adder

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 ENTITY fulladder IS
5 PORT(a, b, cin: IN STD_LOGIC;
6      f, cout: OUT STD_LOGIC);
7
8 END fulladder;
9
10
11 ARCHITECTURE behavior OF fulladder IS
12
13 COMPONENT multiplexer2to1
14   PORT (x,y,s:in STD_LOGIC; -- voglio un ingresso x,y a 1 bit
15        m: OUT STD_LOGIC);-- voglio che l'uscita sia x o y
16
17 END COMPONENT;
18
19 SIGNAL selettore: STD_LOGIC;
20
21 BEGIN
22
23   selettore <= a XOR b;
24   f <= cin XOR selettore;
25
26   mul: multiplexer2to1 PORT MAP( x => b, y => cin, s => selettore, m => cout);
27
28 END ARCHITECTURE;

```

## - Multiplexer 2 a 1 su 8 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 --creazione del multiplexer2_1 a 8bit a partire da entity:
6
7 ENTITY multiplexer2to1_8bit IS
8   GENERIC( n : INTEGER := 8);
9   PORT (x: IN SIGNED(n-1 DOWNTO 0);
10        y: IN SIGNED(n-1 DOWNTO 0);
11        s: IN STD_LOGIC;
12        m: OUT SIGNED(n-1 DOWNTO 0));
13
14 END multiplexer2to1_8bit;
15
16
17 -- definisco l'architettura del multiplexer
18
19
20 ARCHITECTURE behavior OF multiplexer2to1_8bit IS
21
22
23   COMPONENT multiplexer2to1
24     PORT (x, y : IN STD_LOGIC;
25            s : IN STD_LOGIC;
26            m: OUT STD_LOGIC);
27
28   END COMPONENT;
29
30   BEGIN
31
32   -- istanzio 8 mux2to1.
33   G1 : FOR i IN 0 TO n-1 GENERATE
34     mux: multiplexer2to1 PORT MAP(x(i), y(i), s, m(i));
35
36   END GENERATE;
37
38 END behavior;
40

```

## - Multiplexer 2 a 1 su 10 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 --creazione del multiplexer2_1 a 10bit a partire da entity:
6
7 ENTITY multiplexer2to1_10bit IS
8   GENERIC( n : INTEGER := 10);
9   PORT (x: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
10        y: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
11        s: IN STD_LOGIC;
12        m: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
13
14 END multiplexer2to1_10bit;
15
16
17 -- definisco l'architettura del multiplexer
18
19
20 ARCHITECTURE behavior OF multiplexer2to1_10bit IS
21
22
23   COMPONENT multiplexer2to1
24     PORT (x, y : IN STD_LOGIC;
25            s : IN STD_LOGIC;
26            m: OUT STD_LOGIC);
27
28   END COMPONENT;
29
30
31   BEGIN
32
33   -- istanzio 10 mux 2to1.
34   G1 : FOR i IN 0 TO n-1 GENERATE
35     mux: multiplexer2to1 PORT MAP(x(i), y(i), s, m(i));
36
37   END GENERATE;
38
39 END behavior;
40

```

## - Multiplexer 4 a 1 su 8 bit

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4 -- creazione di multiplexer4to1 a 8 bit a partire da entity.
5
6 ENTITY multiplexer4to1_8bit IS
7
8     PORT(u, v, w, x: IN SIGNED(7 DOWNTO 0);          -- ingressi del mu4to1.
9         selectore: IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- selettore, costituito da 2 bit.
10        f: OUT SIGNED(7 DOWNTO 0)                  -- uscita del mu4to1.
11    );
12
13
14 END multiplexer4to1_8bit;
15
16 -- definizione dell'architettura.
17 ARCHITECTURE behavior OF multiplexer4to1_8bit IS
18
19 -- richiamo il componente multiplexer2to1 a 8 bit.
20 COMPONENT multiplexer2to1_8bit
21
22     GENERIC( n : INTEGER := 8);
23     PORT (x: IN SIGNED(n-1 DOWNTO 0);
24            y: IN SIGNED(n-1 DOWNTO 0);
25            s: IN STD_LOGIC;
26            m: OUT SIGNED(n-1 DOWNTO 0));
27
28     END COMPONENT;
29
30 -- definisco il segnale.
31
32 SIGNAL output_1, output_0: SIGNED( 7 DOWNTO 0 ) ; -- segnale di uscita di ciascun multiplexer2to1.
33
34 BEGIN
35
36     mui: multiplexer2to1_8bit port map ( x=> u, y=> v, s=> selectore(0), m=> output_1);
37     mu2: multiplexer2to1_8bit port map ( x=> w, y=> x, s=> selectore(0), m=> output_0);
38     mu3: multiplexer2to1_8bit port map ( x=> output_1, y=> output_0, s=> selectore(1), m=> f);
39
40 END ARCHITECTURE;
```

## - Multiplexer 2 a 1 su 16 bit

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4 --creazione del multiplexer2to1 a 16bit a partire da entity:
5
6 ENTITY multiplexer2to1_16bit IS
7
8     GENERIC( n : INTEGER := 16);
9     PORT (x: IN SIGNED(n-1 DOWNTO 0);
10           y: IN SIGNED(n-1 DOWNTO 0);
11           s: IN STD_LOGIC;
12           m: OUT SIGNED(n-1 DOWNTO 0));
13
14 END multiplexer2to1_16bit;
15
16 -- definisco l'architettura del multiplexer
17
18 ARCHITECTURE behavior OF multiplexer2to1_16bit IS
19
20
21     COMPONENT multiplexer2to1
22
23         PORT (x, y : IN STD_LOGIC;
24                s : IN STD_LOGIC;
25                m: OUT STD_LOGIC);
26
27     END COMPONENT;
28
29
30
31 BEGIN
32
33     -- istanzio 16 mux 2to1.
34     G1 : FOR i IN 0 TO n-1 GENERATE
35         mux: multiplexer2to1 PORT MAP(x(i), y(i), s, m(i));
36
37     END GENERATE;
38
39
40 END behavior;
```

## - Registro da 8 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5
6 ENTITY regn_8bit IS
7   GENERIC ( N : integer:= 8);
8
9   PORT (
10     R : IN SIGNED(N-1 DOWNTO 0);
11     Clock,Reset,enable : IN STD_LOGIC;
12     Q : OUT SIGNED(N-1 DOWNTO 0)
13   );
14
15 END regn_8bit;
16
17
18 ARCHITECTURE Behavior OF regn_8bit IS
19
20 BEGIN
21
22   PROCESS (Clock, Reset)
23
24   BEGIN
25
26     IF (Reset = '1') THEN
27       Q <= (OTHERS => '0');      -- reset asincrono.
28
29     ELSIF (Clock'EVENT AND Clock = '1') THEN
30       IF enable = '1' THEN
31         Q <= R;
32       END IF;
33
34     END IF;
35
36   END PROCESS;
37
38
39 END Behavior;
40

```

## - Registro da 16 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5
6 ENTITY regn_16bit IS
7   GENERIC ( N : integer:= 16);
8
9   PORT (
10     R : IN SIGNED(N-1 DOWNTO 0);
11     Clock,Reset,enable : IN STD_LOGIC;
12     Q : OUT SIGNED(N-1 DOWNTO 0)
13   );
14
15 END regn_16bit;
16
17
18 ARCHITECTURE Behavior OF regn_16bit IS
19
20 BEGIN
21
22   PROCESS (Clock, Reset)
23
24   BEGIN
25
26     IF (Reset = '1') THEN      -- reset asincrono.
27       Q <= (OTHERS => '0');
28
29     ELSIF (Clock'EVENT AND Clock = '1') THEN
30       IF enable = '1' THEN
31         Q <= R;
32       END IF;
33
34     END IF;
35
36   END PROCESS;
37
38
39 END Behavior;
40

```

## - Up-counter su 10 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 -- Il count_1 è il contatore che funge da puntatore per la memoria A ( e per la fase di inizializzazione della memoria B ).  

5 -- Nel progetto finale abbiamo usato lo stesso componente per count_2.
6
7 ENTITY count_1 IS
8   GENERIC ( n : INTEGER := 10);
9   PORT (
10     enb : IN STD_LOGIC;
11     clock,reset_c : IN STD_LOGIC;
12     Qs : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
13   );
14 END count_1;
15
16
17 ARCHITECTURE behavir OF count_1 IS
18
19   COMPONENT t_flipflop
20
21   PORT (
22     T : IN STD_LOGIC;
23     clk,reset : IN STD_LOGIC;
24     Q : OUT STD_LOGIC
25   );
26
27 END COMPONENT;
28
29
30
31 SIGNAL qsignal : STD_LOGIC_VECTOR(n-1 DOWNTO 0);
32 SIGNAL and_link : STD_LOGIC_VECTOR(n-2 DOWNTO 0);
33
34 BEGIN
35
36   -- Il contatore è realizzato con 10 t_flipflop ( a noi interessa contare fino a 1023 ).  

37
38   tf1 : t_flipflop PORT MAP ( T => enb, clk => clock, reset => reset_c, Q => qsignal(0));
39
40   and_link(0) <= enb AND qsignal(0);
41
42
43
44   G : FOR i IN 1 TO n-2 GENERATE
45
46     tf : t_flipflop PORT MAP ( T => and_link(i-1), clk => clock, reset => reset_c, Q => qsignal(i));
47     and_link(i) <= and_link(i-1) AND qsignal(i);
48
49   END GENERATE;
50
51   tf_last : t_flipflop PORT MAP ( T => and_link(n-2) , clk => clock, reset => reset_c, Q => qsignal(n-1));
52
53   Qs <= qsignal;
54
55 END ARCHITECTURE;
56

```

## - Up-counter su 4 bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 -- Il flag_counter sarà utilizzato per dare il segnale di allarme.
5
6 ENTITY flag_counter IS
7   GENERIC ( n : INTEGER := 4);
8   PORT (
9     enb : IN STD_LOGIC;
10    clock,reset_c : IN STD_LOGIC;
11    Qs : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
12  );
13 END flag_counter;
14
15
16 ARCHITECTURE behavir OF flag_counter IS
17
18   COMPONENT t_flipflop
19
20   PORT (
21     T : IN STD_LOGIC;
22     clk,reset : IN STD_LOGIC;
23     Q : OUT STD_LOGIC
24   );
25
26 END COMPONENT;
27
28
29
30 SIGNAL qsignal : STD_LOGIC_VECTOR(n-1 DOWNTO 0);
31 SIGNAL and_link : STD_LOGIC_VECTOR(n-2 DOWNTO 0);
32
33
34 BEGIN
35
36   -- Il contatore è realizzato con 4 t_flipflop ( a noi interessa contare fino a 10 ).  

37
38   tf1 : t_flipflop PORT MAP ( T => enb, clk => clock, reset => reset_c, Q => qsignal(0));
39
40   and_link(0) <= enb AND qsignal(0);
41
42
43
44   G : FOR i IN 1 TO n-2 GENERATE
45
46     tf : t_flipflop PORT MAP ( T => and_link(i-1), clk => clock, reset => reset_c, Q => qsignal(i));
47     and_link(i) <= and_link(i-1) AND qsignal(i);
48
49   END GENERATE;
50
51   tf_last : t_flipflop PORT MAP ( T => and_link(n-2) , clk => clock, reset => reset_c, Q => qsignal(n-1));
52
53   Qs <= qsignal;
54
55 END ARCHITECTURE;
56

```

## - Register file

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 ENTITY register_file IS
6 PORT( data_in : IN STD_LOGIC_VECTOR(7 downto 0);    -- dati in ingresso.
7       address : IN STD_LOGIC_VECTOR(9 downto 0);    -- indirizzo.
8       cs, wr_n, rd, clock : IN STD_LOGIC;           -- chip select, write_n, read, clock.
9       data_out : OUT STD_LOGIC_VECTOR(7 downto 0));   -- dati in uscita.
10
11 END register_file;
12
13 ARCHITECTURE behavior OF register_file IS
14
15 TYPE register_type IS ARRAY(0 to 1023) OF STD_LOGIC_VECTOR(7 DOWNTO 0); -- definisco la dimensione della memoria
16
17 SIGNAL reg_file : register_type; -- istanzio un segnale di memoria
18
19 BEGIN
20
21
22 write_reg : PROCESS(clock)      -- processo per la scrittura sincrona
23
24 BEGIN
25
26 IF (clock'EVENT AND clock = '1') THEN
27
28   IF (cs = '1') THEN          -- seleziono la memoria.
29
30     IF (wr_n = '0') THEN    -- fase di scrittura.
31
32       reg_file(TO_INTEGER(UNSIGNED(address))) <= data_in;
33
34     END IF;
35   END IF;
36 END IF;
37
38 END PROCESS write_reg;
39
40 -- processo per la lettura asincrona
41
42 read_reg : PROCESS(rd,address)      -- processo per la lettura asincrona.
43
44 BEGIN
45 IF (cs = '1') THEN
46
47   IF (rd = '1') THEN
48
49
50
51     data_out <= reg_file(TO_INTEGER(UNSIGNED(address)));
52   ELSE
53     data_out <= "UUUUUUUU"; -- per evitare la creazione di latch.
54
55   END IF;
56   ELSE
57     data_out <= "UUUUUUUU";
58
59   END IF;
60
61 END IF;
62
63
64
65
66
67 END read_reg;
68
69
70 END behavior;
```

## - Ripple carry adder su 17 bit

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 -- Dichiarazione delle porte di ingresso e di uscita del ripple carry adder.
6
7 ENTITY ripple_carry_adder IS
8
9   GENERIC ( n: INTEGER := 16 );
10  PORT ( x_in: IN SIGNED(n DOWNTO 0);
11        y_in : IN SIGNED(n DOWNTO 0);
12        c_in : IN STD_LOGIC;
13        c_out : OUT STD_LOGIC;
14        s_out : OUT SIGNED(n DOWNTO 0)
15      );
16
17 END ripple_carry_adder;
18
19 -- Architettura del ripple carry.
20 ARCHITECTURE behavior OF ripple_carry_adder IS
21
22 -- richiamo il componente "fulladder".
23 COMPONENT fulladder
24
25   PORT(a, b, cin: IN STD_LOGIC;
26        f, cout: OUT STD_LOGIC);
27
28 END COMPONENT;
29
30
31 SIGNAL cout_intermedi : STD_LOGIC_VECTOR(n-1 DOWNTO 0);           -- carry in uscita di ciascun full adder
32
33
34
35
36 BEGIN
```

```

37  -- Il ripple_carry a 17 bit è costituito da 17 full adder.
38
39  fu_primo: fulladder PORT MAP( a => x_in(0), b => y_in(0), cin => c_in, f => s_out(0), cout => cout_intermedi(0));
40
41  G2 : FOR i IN 1 TO n-1 GENERATE
42
43      fu: fulladder PORT MAP( a => x_in(i), b => y_in(i), cin => cout_intermedi(i-1), f => s_out(i), cout => cout_intermedi(i));
44
45  END GENERATE;
46
47  fu_final: fulladder PORT MAP( a => x_in(n), b => y_in(n), cin => cout_intermedi(n-1), f => s_out(n), cout => c_out);
48
49 END ARCHITECTURE;
50

```

- Entità top level che descrive l'intero circuito e la macchina a stati finiti

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.numeric_std.all;
4
5 -- VHDL del datapath ed unità di controllo.
6
7 -- Entità del circuito.
8
9 ENTITY irrigation_system IS
10
11  PORT(clock_asm, reset_asm, start : IN STD_LOGIC; -- clock della macchina, reset esterno asincrono, e segnale di start.
12      done_sig : OUT STD_LOGIC; -- done_sig è il segnale dato in uscita, solo dopo aver concluso tutte le operazioni.
13      irr_alarm : BUFFER STD_LOGIC; -- segnale di allarme ; si accende quando il valore "AVERAGE" è maggiore ( in valore assoluto ) della soglia.
14      threshold : IN SIGNED(15 DOWNTO 0); -- si suppone che la soglia venga impostata dall' utente.
15      data_ext : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- dati in ingresso ( misurati dai sensori ). 
16      data_out_MEMB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- segnale non necessario ai fini del progetto, inserito solamente per come è stata strutturata la memoria e per completezza.
17
18  -- Architettura.
19
20  ARCHITECTURE behavior OF irrigation_system IS
21
22  -- richiamo tutti i componenti necessari per la realizzazione del circuito.
23
24  COMPONENT ripple_carry_adder IS
25      GENERIC ( n: INTEGER := 16 );
26      PORT (
27          x_in : IN SIGNED(n DOWNTO 0);
28          y_in : IN SIGNED(n DOWNTO 0);
29          c_in : IN STD_LOGIC;
30          c_out : OUT STD_LOGIC;
31          s_out : OUT SIGNED(n DOWNTO 0)
32      );
33
34  END COMPONENT;
35
36  COMPONENT regn_16bit
37      GENERIC ( N : integer:= 16 );
38      PORT (
39          R : IN SIGNED(N-1 DOWNTO 0);
40          Clock, Reset, enable : IN STD_LOGIC;
41          Q : OUT SIGNED(N-1 DOWNTO 0)
42      );
43
44  END COMPONENT;
45
46  COMPONENT regn_8bit
47      GENERIC ( N : integer:= 8 );
48
49      PORT (
50          R : IN SIGNED(N-1 DOWNTO 0);
51          Clock, Reset, enable : IN STD_LOGIC;
52          Q : OUT SIGNED(N-1 DOWNTO 0)
53      );
54
55  END COMPONENT;
56
57  COMPONENT register_file
58      PORT( data_in : IN STD_LOGIC_VECTOR(7 downto 0);
59            address : IN STD_LOGIC_VECTOR(9 downto 0);
60            cs, wr_n, rd, clock : IN STD_LOGIC;
61            data_out : OUT STD_LOGIC_VECTOR(7 downto 0)
62        );
63
64  END COMPONENT;
65
66  COMPONENT multiplexer4to1_8bit
67      PORT(u, v, w, x: IN SIGNED(7 DOWNTO 0); -- ingressi del mu4to1.
68                  selettore: IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- selettore, costituito da 2 bit.
69
70
71
72
73
74
75
76
77
78
79

```

```

80          f: OUT SIGNED(7 DOWNTO 0)      -- uscita del mu4tol.
81      );
82  END COMPONENT;
83
84  COMPONENT multiplexer2to1_16bit
85  GENERIC( n : INTEGER := 16);
86  PORT (x: IN SIGNED(n-1 DOWNTO 0);
87         y: IN SIGNED(n-1 DOWNTO 0);
88         s: IN STD_LOGIC;
89         m: OUT SIGNED(n-1 DOWNTO 0));
90
91  END COMPONENT;
92
93  COMPONENT multiplexer2to1_10bit
94  GENERIC( n : INTEGER := 10);
95  PORT (x: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
96         y: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
97         s: IN STD_LOGIC;
98         m: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
99
100 END COMPONENT;
101
102 COMPONENT flag_counter
103 GENERIC ( n : INTEGER := 4);
104 PORT (
105     enb : IN STD_LOGIC;
106     clock_reset_c : IN STD_LOGIC;
107     qs : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
108 );
109 END COMPONENT;
110
111 COMPONENT count_
112 GENERIC ( n : INTEGER := 10);
113 PORT (
114     enb : IN STD_LOGIC;
115     clock_reset_c : IN STD_LOGIC;
116     qs : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
117 );
118 END COMPONENT;
119
120
121
122
123
124
125
126
127
128
129
130
131
132 TYPE state_type IS (RESET, WR_MEMA, RD_MEMA1, RD_MEMA2, RD_MEMA3, RD_MEMA4, EN_DATA, WR_AVR, DATA_COMP,      -- tutti gli stati possibili
133 WR_HIGH_40, WR_HIGH_20, FLAG_COMP, RES_FLAG, EN_FLAG, WR_MEMB1, WR_MEMB2, WR_MEMB3, WR_MEMB4,
134 DONE);
135
136 SIGNAL state : state_type;      -- stato che viene gestito nell'implementazione a due processi
137
138 -- ogni segnale e bus del DP (i segnali seguono fedelmente l'ASM di controllo e il datapath).
139 SIGNAL reset_DP, reset_flag_counter, reset_final_flag_counter, cs_a, cs_b, wr_a_n, wr_b_n, rd_a, rd_b, en_reg1, en_reg2, en_reg3, en_reg4,
140 en_reg40, en_reg20,
141 mux_sel_0, mux_sel_1, mux_sel_2, mux_sel_3, mux_sel_5, sub_add_n, msb_result, en_average,
142 en_thr, en_high, en_cnt_1, en_cnt_2, en_cnt_flag, tc_1, tc_2, carry_out : STD_LOGIC;
143
144 SIGNAL mux_sel_4 : STD_LOGIC_VECTOR(1 DOWNTO 0);
145
146 SIGNAL reg1_in, reg1_out, reg2_out, reg3_out, reg4_out, mux4_out, lsb_mux4, msb_mux4, zero_mux4 : SIGNED(7 DOWNTO 0);
147
148 SIGNAL data_in_b, data_out_a : STD_LOGIC_VECTOR(7 DOWNTO 0);
149
150 SIGNAL counted_1, counted_2, address_a, address_b : STD_LOGIC_VECTOR(9 DOWNTO 0);
151
152 SIGNAL cnt_flag : STD_LOGIC_VECTOR(3 DOWNTO 0);
153
154 SIGNAL reg40_in, reg20_in, data_40, data_20, mux1_out, mux2_out, mux3_out, mux5_out, regAvr_in, regAvr_out,
155 reghigh_out, regThr_out, xor_gate_out : SIGNED(15 DOWNTO 0);
156
157 SIGNAL in_left_add, in_right_add, out_add : SIGNED(16 DOWNTO 0);
158
159
160
161 BEGIN
162  state_transition : PROCESS(clock_asm)      -- descrizione delle transizioni di stato
163
164  BEGIN
165
166    IF (clock_asm'EVENT AND clock_asm = '1') THEN
167
168      IF reset_asm = '1' THEN
169        state <= RESET;
170
171      ELSE
172
173        CASE state IS
174
175          WHEN RESET => IF start = '1' THEN state <= WR_MEMA;
176              ELSE state <= RESET;
177              END IF;
178
179          WHEN WR_MEMA => IF tc_1 = '1' THEN state <= RD_MEMA1;
180              ELSE state <= WR_MEMA;
181              END IF;
182
183          WHEN RD_MEMA1 => state <= RD_MEMA2;
184
185          WHEN RD_MEMA2 => state <= RD_MEMA3;
186
187          WHEN RD_MEMA3 => state <= RD_MEMA4;
188
189          WHEN RD_MEMA4 => state <= EN_DATA;
190
191          WHEN EN_DATA => state <= WR_AVR;
192
193        END CASE;
194
195      END IF;
196

```

```

197 WHEN WR_AVR => state <= DATA_COMP;
198 WHEN DATA_COMP => IF msb_result = '1' THEN
199   state <= WR_HIGH_40;
200   ELSE
201     state <= WR_HIGH_20;
202   END IF;
203
204 WHEN WR_HIGH_40 => state <= FLAG_COMP;
205
206 WHEN WR_HIGH_20 => state <= FLAG_COMP;
207
208 WHEN FLAG_COMP => IF msb_result = '1' THEN
209   IF irr_alarm = '1' THEN
210     state <= WR_MEMB1;
211   ELSE
212     state <= EN_FLAG;
213   END IF;
214   ELSE
215     state <= RES_FLAG;
216   END IF;
217
218 WHEN RES_FLAG => state <= WR_MEMB1;
219
220 WHEN EN_FLAG => state <= WR_MEMB1;
221
222 WHEN WR_MEMB1 => state <= WR_MEMB2;
223
224 WHEN WR_MEMB2 => state <= WR_MEMB3;
225
226 WHEN WR_MEMB3 => state <= WR_MEMB4;
227
228 WHEN WR_MEMB4 => IF tc_2 = '1' THEN
229   state <= DONE;
230   ELSE
231     state <= EN_DATA;
232   END IF;
233
234 WHEN DONE => IF start = '1' THEN
235
236   state <= DONE;
237   ELSE
238     state <= RESET;
239   END IF;
240
241 END CASE;
242
243 END IF;
244
245 END IF;
246
247
248 END PROCESS state_transition;
249
250 state_outputs : PROCESS(state) -- definisco i segnali attivati in ogni strato mettendone prima i valori di default
251
252 BEGIN
253
254   cs_a <= '0'; cs_b <= '0'; wr_a_n <= '1'; wr_b_n <= '1'; rd_a <= '0'; rd_b <= '0';
255   en_reg1 <= '0'; en_reg2 <= '0'; en_reg3 <= '0'; en_reg4 <= '0'; en_reg40 <= '0'; en_reg20 <= '0';
256   mux_sel_0 <= '0'; mux_sel_1 <= '0'; mux_sel_2 <= '0'; mux_sel_3 <= '0'; mux_sel_4 <= '00';
257   en_average <= '0'; en_thr <= '0'; en_high <= '0'; en_cnt_1 <= '0'; en_cnt_2 <= '0';
258   en_cnt_flag <= '0'; reset_flag_counter <= '0';
259   done_sig <= '0'; reset_DP <= '0';
260
261 CASE state IS
262
263   WHEN RESET =>
264     reset_DP <= '1';
265     en_thr <= '1';
266
267   WHEN WR_MEMA =>
268
269     cs_a <= '1'; cs_b <= '1'; wr_a_n <= '0'; wr_b_n <= '0'; en_cnt_1 <= '1';
270     -- scrivo in memoria A e B, incremento il contatore 1.
271
272   -- la sequenza di lettura iniziale legge un byte per volta per 4 colpi di clock, 4 incrementi del contatore 1.
273   WHEN RD_MEMA1 =>
274
275     --
276     cs_a <= '1'; rd_a <= '1'; en_reg1 <= '1'; en_cnt_1 <= '1';
277
278   WHEN RD_MEMA2 =>
279     cs_a <= '1'; rd_a <= '1'; en_reg1 <= '1'; en_reg2 <= '1'; en_cnt_1 <= '1';
280
281   WHEN RD_MEMA3 =>
282     cs_a <= '1'; rd_a <= '1'; en_reg1 <= '1'; en_reg2 <= '1'; en_reg3 <= '1'; en_cnt_1 <= '1';
283
284   WHEN RD_MEMA4 =>
285     cs_a <= '1'; rd_a <= '1'; en_reg1 <= '1'; en_reg2 <= '1'; en_reg3 <= '1'; en_reg4 <= '1';
286     en_cnt_1 <= '1';
287
288   WHEN EN_DATA =>
289     en_reg20 <= '1'; en_reg40 <= '1';
290     -- dati raggruppati in registri da 16 bit, si ricostruisce il dato per intero.
291
292   WHEN WR_AVR =>
293     en_average <= '1';
294     -- si registra il valor medio
295
296   WHEN DATA_COMP =>
297     sub_add_n <= '1';
298     -- per effettuare la comparazione tra data_20 e data_40 è necessario convertire il sommatore in sottrattore.
299
300   WHEN WR_HIGH_40 =>
301     mux_sel_5 <= '1'; en_high <= '1';
302     -- registro data_40 se è più grande in modulo.
303
304   WHEN WR_HIGH_20 =>
305     en_high <= '1';
306     -- registro data_20 se è più grande in modulo.
307
308
309
310
311
312
313

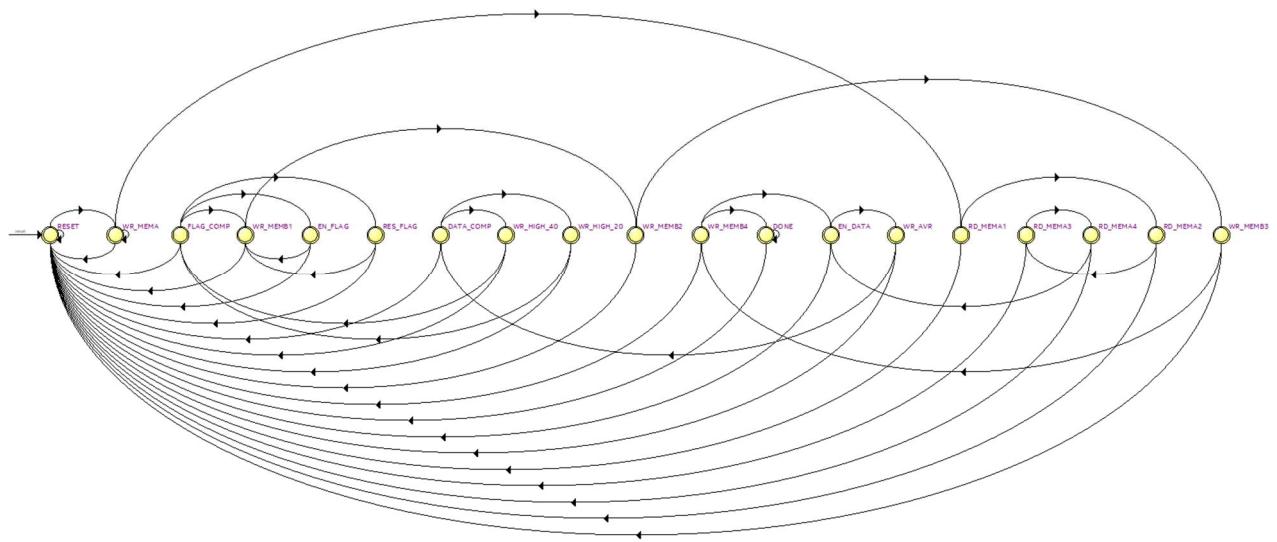
```

```

314 |
315 |         WHEN FLAG_COMP =>
316 |             mux_sel_1 <= '1'; mux_sel_2 <= '1'; sub_add_n <= '1';
317 |
318 |         WHEN RES_FLAG =>
319 |             reset_flag_counter <= '1'; -- resetto il contatore flag se non si ottengono 10 valori consecutivi soddisfacenti la condizione imposta.
320 |
321 |         WHEN EN_FLAG =>
322 |             en_cnt_flag <= '1'; -- incremento il contatore se ottengo un valore sottosoglia.
323 |
324 |         WHEN WR_MEMB1 =>
325 |             cs_a <= '1'; cs_b <= '1'; rd_a <= '1'; wr_b_n <= '0'; en_cnt_1 <= '1'; en_cnt_2 <= '1'; en_reg1 <= '1'; mux_sel_0 <= '1';
326 |             mux_sel_3 <= '1'; mux_sel_4 <= "01";
327 |
328 |         WHEN WR_MEMB2 =>
329 |             cs_a <= '1'; cs_b <= '1'; rd_a <= '1'; wr_b_n <= '0'; en_cnt_1 <= '1'; en_cnt_2 <= '1'; en_reg1 <= '1'; en_reg2 <= '1';
330 |             mux_sel_0 <= '1'; mux_sel_3 <= '1'; mux_sel_4 <= "10";
331 |
332 |         WHEN WR_MEMB3 =>
333 |             cs_a <= '1'; cs_b <= '1'; rd_a <= '1'; wr_b_n <= '0'; en_cnt_1 <= '1'; en_cnt_2 <= '1'; en_reg1 <= '1';
334 |             en_reg2 <= '1'; en_reg3 <= '1';
335 |             mux_sel_0 <= '1'; mux_sel_4 <= "01";
336 |
337 |         WHEN WR_MEMB4 =>
338 |             cs_a <= '1'; cs_b <= '1'; rd_a <= '1'; wr_b_n <= '0'; en_cnt_1 <= '1'; en_cnt_2 <= '1'; en_reg1 <= '1';
339 |             en_reg2 <= '1'; en_reg3 <= '1'; en_reg4 <= '1';
340 |             mux_sel_0 <= '1'; mux_sel_4 <= "10";
341 |
342 |         WHEN DONE =>
343 |             done_sig <= '1';
344 |
345 |     END CASE;
346 | END PROCESS state_outputs;
347 |
348 | -- Datapath
349 |
350 | count1 : count_1 PORT MAP ( enb => en_cnt_1, clock => clock_asm, reset_c => reset_DP, Qs => counted_1);
351 | tc_1 <= counted_1(9) AND counted_1(8) AND counted_1(7) AND counted_1(6) AND counted_1(5) AND counted_1(4) AND counted_1(3) AND counted_1(2) AND counted_1(1) AND counted_1(0);
352 | -- quando il contatore 1 raggiunge "111111111" allora tc_1 = '1'
353 | count2 : count_1 PORT MAP ( enb => en_cnt_2, clock => clock_asm, reset_c => reset_DP, Qs => counted_2);
354 | tc_2 <= counted_2(9) AND counted_2(8) AND counted_2(7) AND counted_2(6) AND counted_2(5) AND counted_2(4) AND counted_2(3) AND counted_2(2) AND counted_2(1) AND counted_2(0);
355 | -- quando il contatore 2 raggiunge "111111111" allora tc_2 = '1'.
356 |
357 | reset_final_flag_counter <= (reset_DP OR reset_flag_counter);
358 | -- il reset del contatore di flag avviene sia all'inizio dell'algoritmo che quando si incontra una media sottosoglia
359 |
360 | counter_flag : flag_counter PORT MAP ( enb => en_cnt_flag, clock => clock_asm, reset_c => reset_final_flag_counter, Qs => cnt_flag);
361 | irr_alarm <= cnt_flag(3) AND NOT(cnt_flag(2)) AND cnt_flag(1) AND NOT(cnt_flag(0)); -- "1010".
362 |
363 | address_a <= counted_1;
364 | MEM_A : register_file PORT MAP ( data_in => data_ext, address => address_a, cs => cs_a, wr_n => wr_a_n, rd => rd_a,
365 |                                     clock => clock_asm, data_out => data_out_a);
366 |
367 | data_in_b <= STD_LOGIC_VECTOR(mux4_out);
368 |
369 | MEM_B : register_file PORT MAP ( data_in => data_in_b, address => address_b, cs => cs_b, wr_n => wr_b_n, rd => rd_b,
370 |                                     clock => clock_asm, data_out => data_out_MEMB);
371 |
372 | reg1_in <= SIGNED(data_out_a);
373 |
374 | -- 4 registri posti in sequenza.
375 | reg1 : regn_8bit PORT MAP ( R => reg1_in, Clock => clock_asm, Reset => reset_DP, enable => en_reg1, Q => reg1_out);
376 | reg2 : regn_8bit PORT MAP ( R => reg1_out, Clock => clock_asm, Reset => reset_DP, enable => en_reg2, Q => reg2_out);
377 | reg3 : regn_8bit PORT MAP ( R => reg2_out, Clock => clock_asm, Reset => reset_DP, enable => en_reg3, Q => reg3_out);
378 | reg4 : regn_8bit PORT MAP ( R => reg3_out, Clock => clock_asm, Reset => reset_DP, enable => en_reg4, Q => reg4_out);
379 |
380 | -- ricostruzione dati per intero e registrazione.
381 | reg40_in <= reg1_out & reg2_out;
382 | reg20_in <= reg3_out & reg4_out;
383 |
384 | reg40 : regn_16bit PORT MAP ( R => reg40_in, Clock => clock_asm, Reset => reset_DP, enable => en_reg40, Q => data_40);
385 | reg20 : regn_16bit PORT MAP ( R => reg20_in, Clock => clock_asm, Reset => reset_DP, enable => en_reg20, Q => data_20);
386 |
387 | -- registri di soglia, valor medio e valore massimo.
388 | regThr : regn_16bit PORT MAP ( R => threshold, Clock => clock_asm, Reset => reset_asm, enable => en_thr, Q => regThr_out);
389 | regAvr : regn_16bit PORT MAP ( R => regAvr_in, Clock => clock_asm, Reset => reset_DP, enable => en_average, Q => regAvr_out);
390 | regHigh : regn_16bit PORT MAP ( R => mux5_out, Clock => clock_asm, Reset => reset_DP, enable => en_high, Q => regHigh_out);
391 |
392 | -- tutti i mux necessari.
393 | mpx0 : multiplexer2to1_10bit PORT MAP ( x => counted_1, y => counted_2, s => mux_sel_0, m => address_b);
394 |
395 | mpx1 : multiplexer2to1_16bit PORT MAP ( x => data_40, y => regAvr_out, s => mux_sel_1, m => mux1_out);
396 | mpx2 : multiplexer2to1_16bit PORT MAP ( x => data_20, y => regThr_out, s => mux_sel_2, m => mux2_out);
397 |
398 | mpx3 : multiplexer2to1_16bit PORT MAP ( x => regHigh_out, y => regAvr_out, s => mux_sel_3, m => mux3_out);
399 |
400 | zero_mux4 <= "00000000"; -- il dato che viene inserito in MEM_B per inizializzarla
401 | lsb_mux4 <= mux3_out(7 DOWNTO 0);
402 | msb_mux4 <= mux3_out(15 DOWNTO 8);
403 | mpx4 : multiplexer4to1_8bit PORT MAP ( u => zero_mux4, v => lsb_mux4, w => msb_mux4, x => "UUUUUUUU", selettore => mux_sel_4, f => mux4_out);
404 |
405 | -- permette il passaggio di un byte alla volta in memoria B.
406 |
407 | mpx5 : multiplexer2to1_16bit PORT MAP ( x => data_20, y => data_40, s => mux_sel_5, m => mux5_out);
408 | -- permette il passaggio di uno dei due valori, affinché venga registrato in memoria B il valore massimo.
409 |
410 | in_left_add <= mux1_out(15) & mux1_out; -- estensione di segno
411 |
412 | G1 : FOR i IN 0 TO 15 GENERATE
413 |     xor_gate_out(i) <= sub_add_n XOR STD_LOGIC(mux2_out(i));
414 |
415 | END GENERATE;
416 |
417 | in_right_add <= xor_gate_out(15) & xor_gate_out; -- estensione di segno
418 |
419 | adder : ripple_carry_adder PORT MAP ( x_in => in_left_add, y_in => in_right_add, c_in => sub_add_n, c_out => carry_out, s_out => out_add);
420 |
421 | msb_result <= out_add(16); -- msb del risultato usato come status flag per la fase di transizione degli stati.
422 |
423 | regAvr_in <= out_add(16 DOWNTO 1); -- shifting logico.
424 |
425 | END behavior;

```

State Machine View di Quartus:



Al fine di effettuare una simulazione esaustiva su Modelsim, nel testbench si è provata ogni combinazione dei segnali di RESET e START al fine di assicurarsi che l'algoritmo partisse, finisse e ricominciasse in maniera corretta.

In ingresso al circuito si sono inviati 8 bit alla volta i dati rappresentanti tre misurazioni diverse, impostandone i valori più critici in modo da controllare che le operazioni aritmetiche di media e confronto e successivamente la scrittura nella MEM\_B avvenissero esattamente.

Il codice usato per la simulazione è il seguente:

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5
6  ENTITY tbirrigation_system IS
7  END tbirrigation_system;
8
9
10 ARCHITECTURE behavior OF tbirrigation_system IS
11
12 COMPONENT irrigation_system
13   PORT(clock_asm, reset_asm, start : IN STD_LOGIC;
14        done_sig : OUT STD_LOGIC;
15        irr_alarm : BUFFER STD_LOGIC;
16        threshold : IN SIGNED(15 DOWNTO 0);
17        data_ext : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
18        data_out_MEMB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
19
20 END COMPONENT;
21
22
23
24 SIGNAL clock, reset, strt, finito, allarme : STD_LOGIC;
25 SIGNAL data_in : STD_LOGIC_VECTOR(7 DOWNTO 0);
26 SIGNAL data_out_B : STD_LOGIC_VECTOR(7 DOWNTO 0);
27 SIGNAL soglia : SIGNED( 15 DOWNTO 0 ) ;
28
29 BEGIN
30
31   strt <= '0', '1' after 6 ps, '0' after 10 ps, '1' after 6700 ps, '0' after 6800 ps;-- , '1' AFTER 1 ps;
32   -- reset <= '0', '1' AFTER 2 ps, '0' AFTER 6 ps;
33   soglia <= "1111111111111100";
34   clock_process : PROCESS
35
36     BEGIN
37       clock <= '0';
38       WAIT FOR 1 ps;
39       clock <= '1';
40       WAIT FOR 1 ps;
41
42     END PROCESS clock_process;
43
44   dati_inA : PROCESS
45
46     BEGIN
47
48       data_in <= "00000000"; -- -32768
49       WAIT FOR 2 ps;
50       data_in <= "10000000";
51       WAIT FOR 2 ps;
52       data_in <= "00000100"; -- -32764
53       WAIT FOR 2 ps;
54       data_in <= "10000000"; -- media = -32766 (1000000000000010)
55       WAIT FOR 2 ps;
56       data_in <= "00000000";
57       WAIT FOR 2 ps;
58       data_in <= (others => ('0')); -- 0
59       WAIT FOR 2 ps;
60
61       data_in <= "11101011"; -- -20
62       WAIT FOR 2 ps;
63       data_in <= (others => ('1')); -- media = -10
64       WAIT FOR 2 ps;
65       data_in <= "11110000"; -- -16
66       WAIT FOR 2 ps;
67       data_in <= (others => ('1'));
68       WAIT FOR 2 ps;
69       data_in <= "00000000"; -- 0
70       WAIT FOR 2 ps;
71       data_in <= (others => ('0')); -- media = -8 (11111111111100)
72       WAIT FOR 2 ps;
73
74     END PROCESS dati_inA;
75
76   dut : irrigation_system PORT MAP( clock_asm => clock, reset_asm => reset, start => strt, done_sig => finito, irr_alarm => allarme ,threshold => soglia, data_ext => data_in );
77
78
79 END ARCHITECTURE;

```

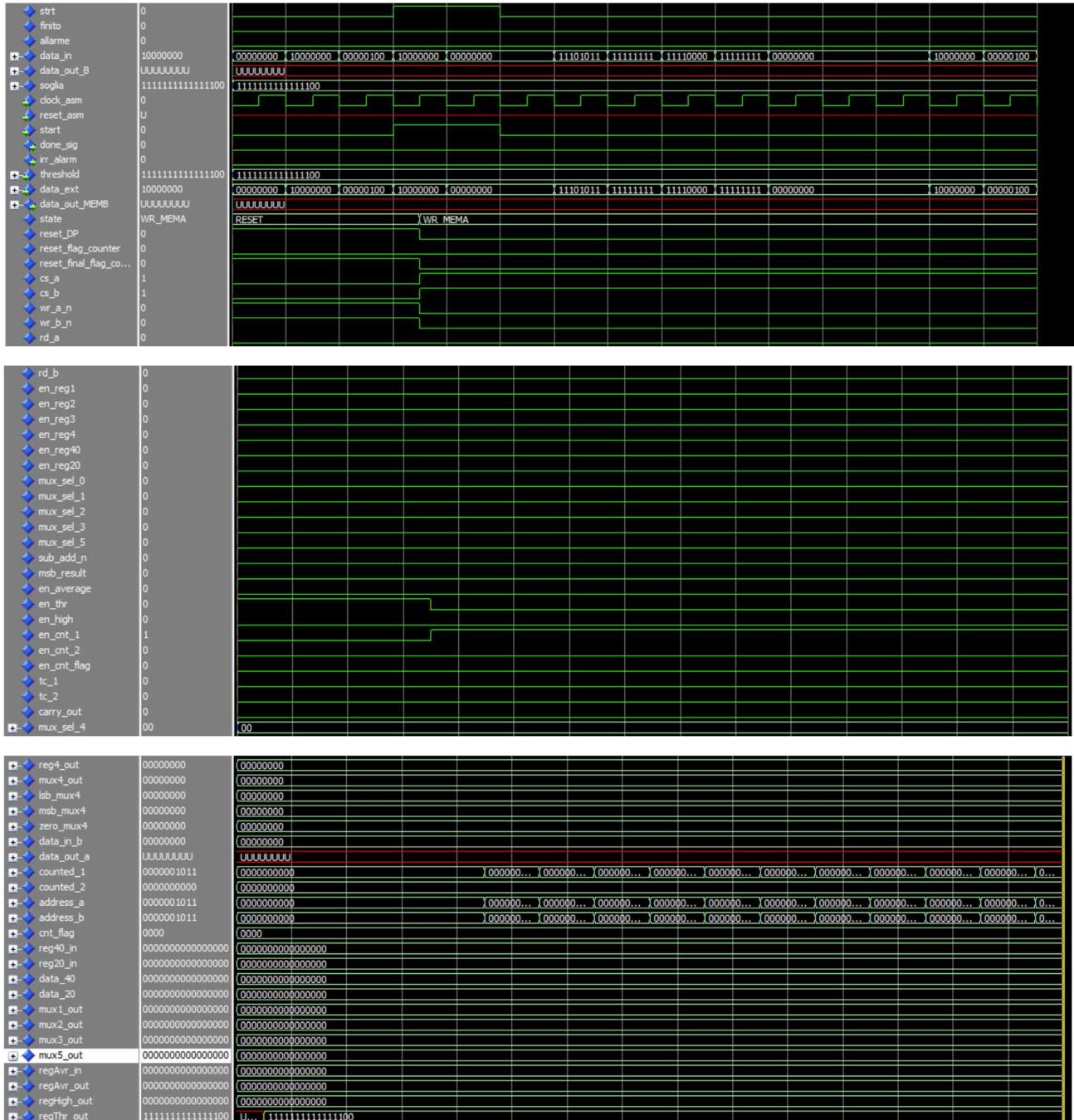
Si può notare che nell'ultimo tentativo effettuato il segnale di START si è asserito ad '1' per due volte al fine di testare il protocollo "handshake" con il segnale di DONE.

Si può evincere dal codice mostrato che per simulare l'arrivo dei dati esterni è stato scritto un processo che ciclicamente cambia il byte in ingresso per un totale di tre misurazioni. I dati arrivano quindi sin dal primo colpo di clock, senza aspettare che lo START venga registrato ad '1'. Questa è stata una semplice scelta di comodità per la simulazione e non causa problemi nella corretta scrittura ed elaborazione dei dati che avvengono dal colpo successivo all'attivazione dello START.

Di seguito si riportano frammenti della simulazione di Modelsim. Viene mostrata la fase iniziale dell'algoritmo, dopodiché vengono elaborati i primi dati presenti nella MEM\_A e infine si mostra la

fine del riempimento della MEM\_B. Il pezzo di simulazione che è stato preso in considerazione per l'elaborazione è quello in cui nel REG\_20 viene registrato un potenziale di 0 kPa mentre nel REG\_40 vengono registrati - 20 kPa. Si può notare come la media venga registrata con il valore di -10 kPa e il valore HIGHEST viene correttamente indicato come il dato del REG\_40. Inoltre il CNT\_FLAG viene aumentato di 1 poiché è stata registrata una media sotto la soglia impostata di -4 kPa.

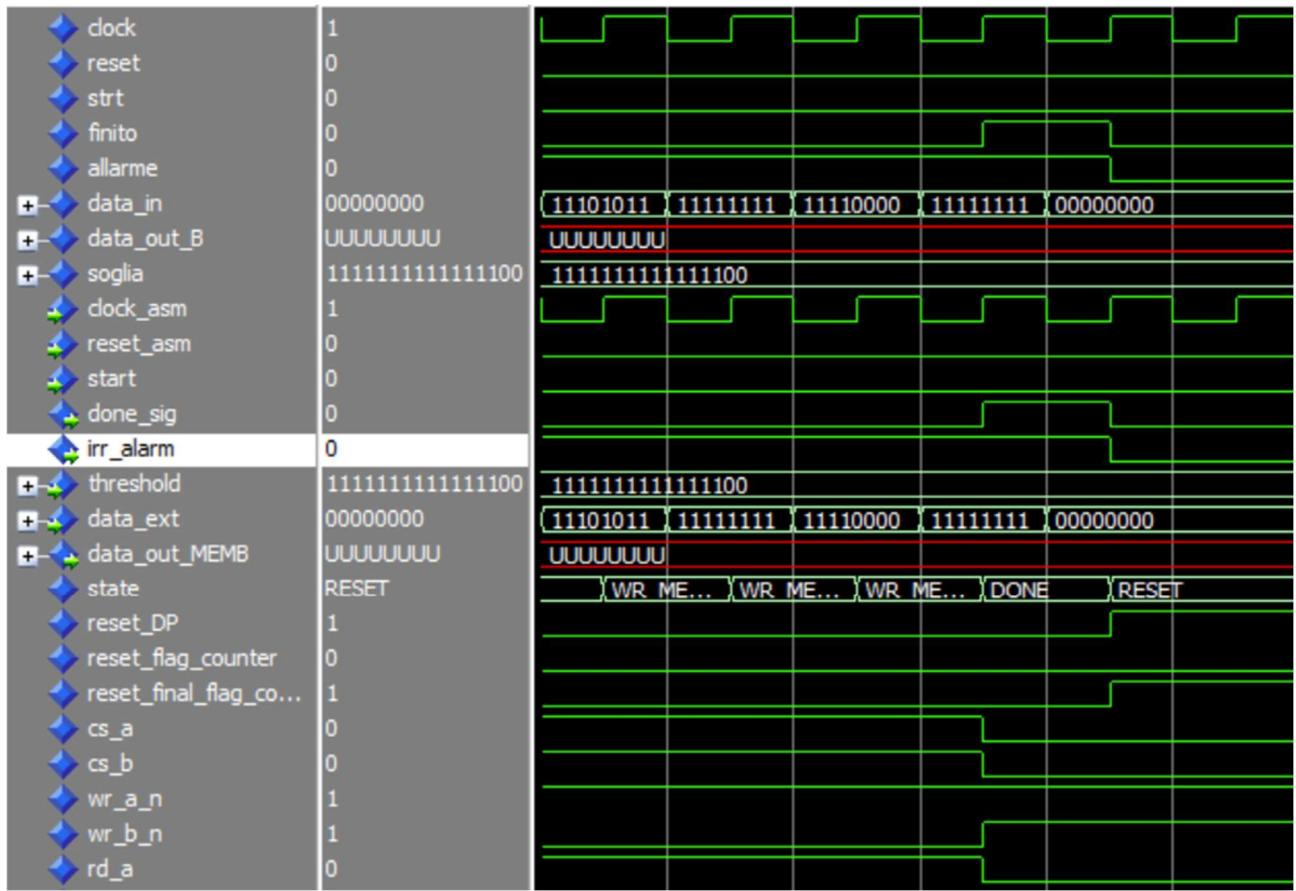
- Fase di registrazione dei dati:

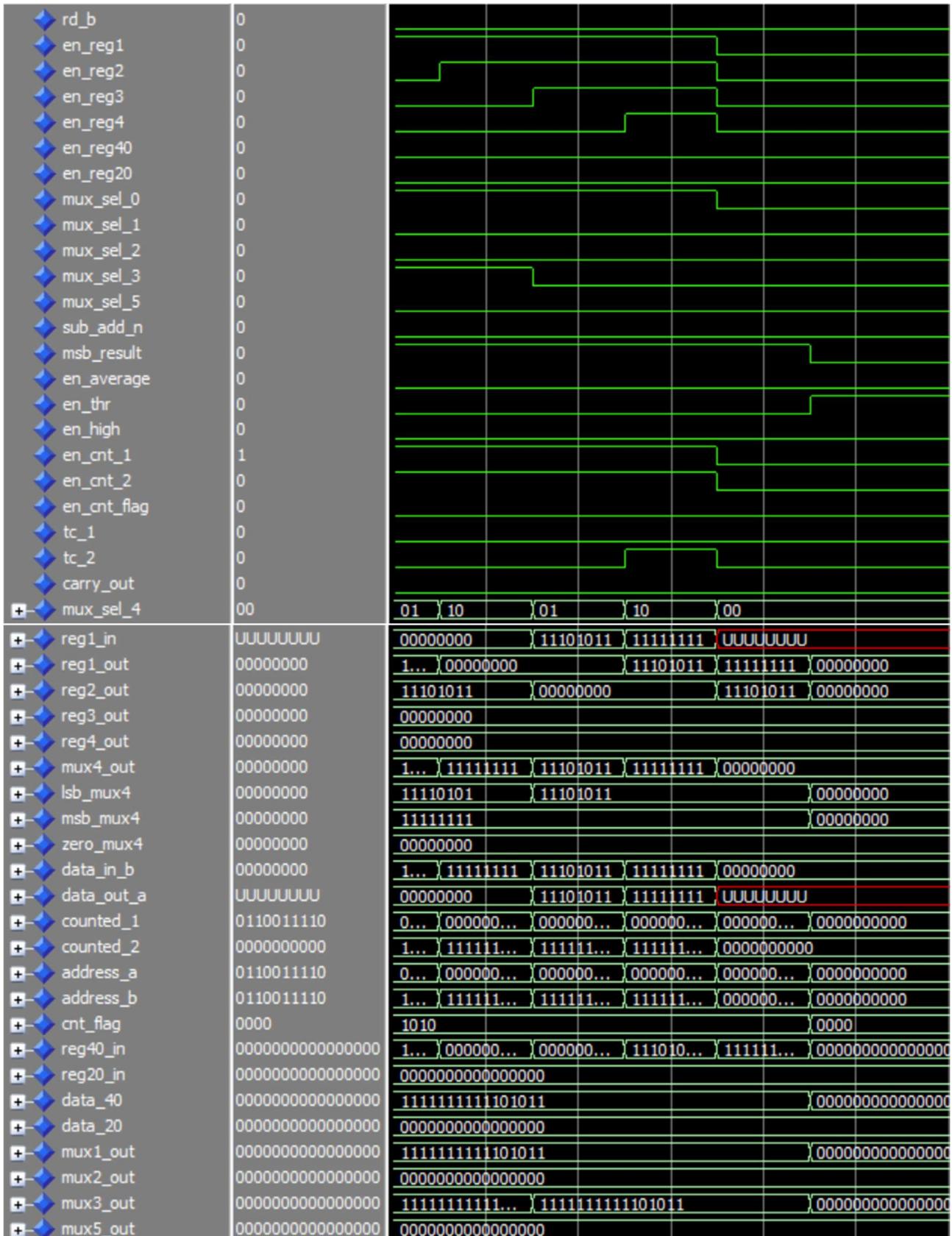


- Fase di elaborazione e scrittura



- Fase di allarme e fine algoritmo:





+ regAvr_in	00000000000000000000	111111111110101			00000000000000000000
+ regAvr_out	00000000000000000000	1111111111110101			00000000000000000000
+ regHigh_out	00000000000000000000	11111111111101011			00000000000000000000
+ regThr_out	111111111111111100	1111111111111100			
+ xor_gate_out	00000000000000000000	0000000000000000			
+ in_left_add	00000000000000000000...	11111111111101011			00000000000000000000
+ in_right_add	00000000000000000000...	0000000000000000			
+ out_add	00000000000000000000...	11111111111101011			00000000000000000000