Progetto di Reti Logiche

Obiettivo:

L'obiettivo del progetto è la descrizione e la sintesi di un componente hardware in VHDL, il quale adopera una memoria in cui legge i dati iniziali e nella quale dovrà scrivere l'output. In particolare, nella memoria sono presenti delle coordinate di centroidi in uno spazio bidimensionale e, dopo aver misurato la distanza di ognuno di essi da un altro centroide assegnato, il componente stabilisce quali di questi sono i più vicini a quest'ultimo.

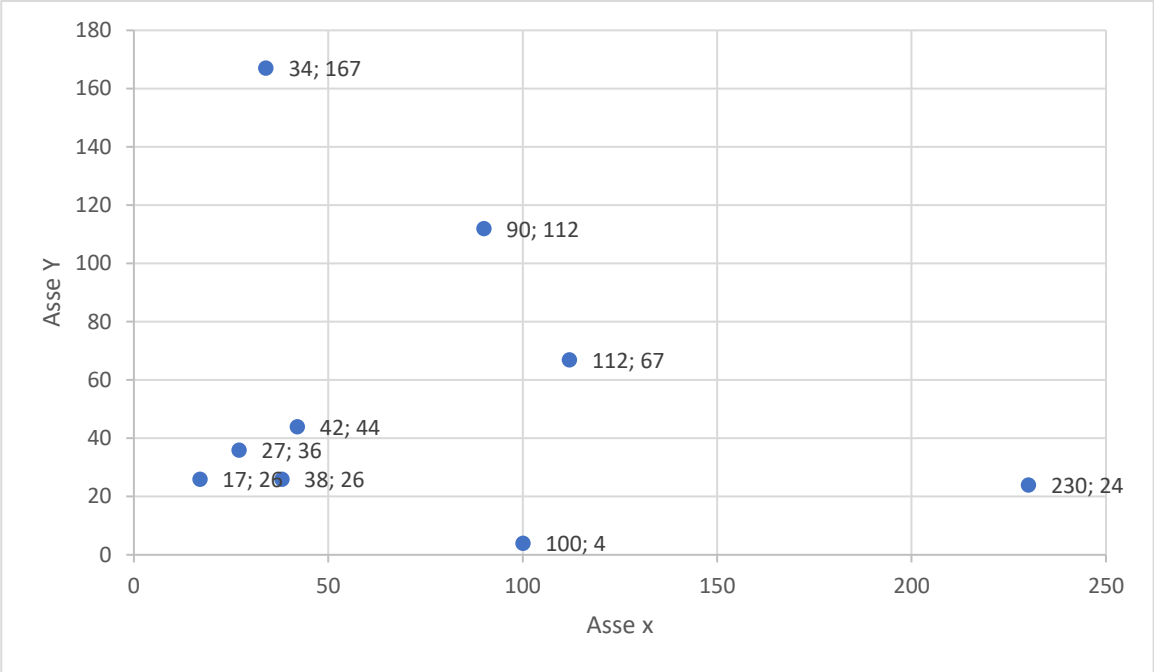
Introduzione:

Il componente possiede, dalle specifiche, queste porte in ingresso e in uscita per interfacciarsi con la memoria.

```
entity project_reti_logiche is
    Port ( i_clk : in STD_LOGIC;
           i_start : in STD_LOGIC;
           i_rst : in STD_LOGIC;
           i_data : in STD_LOGIC_VECTOR (7 downto 0);
           o_address : out STD_LOGIC_VECTOR (15 downto 0);
           o_done : out STD_LOGIC;
           o_en : out STD_LOGIC;
           o_we : out STD_LOGIC;
           o_data : out STD_LOGIC_VECTOR (7 downto 0));
end project reti logiche;
```

Il numero di centroidi da analizzare varia a seconda dei casi, poiché in memoria è anche memorizzata una maschera che indica quali di questi debbano essere controllati: nel caso in cui tutti i bit della maschera fossero a '1', allora tutti i centroidi dovranno essere controllati; viceversa, se tutti fossero a '0', nessuno dovrà essere verificato. Il valore massimo che la maschera può assumere è 255, che corrisponde a 8 bit e quindi a 8 centroidi.

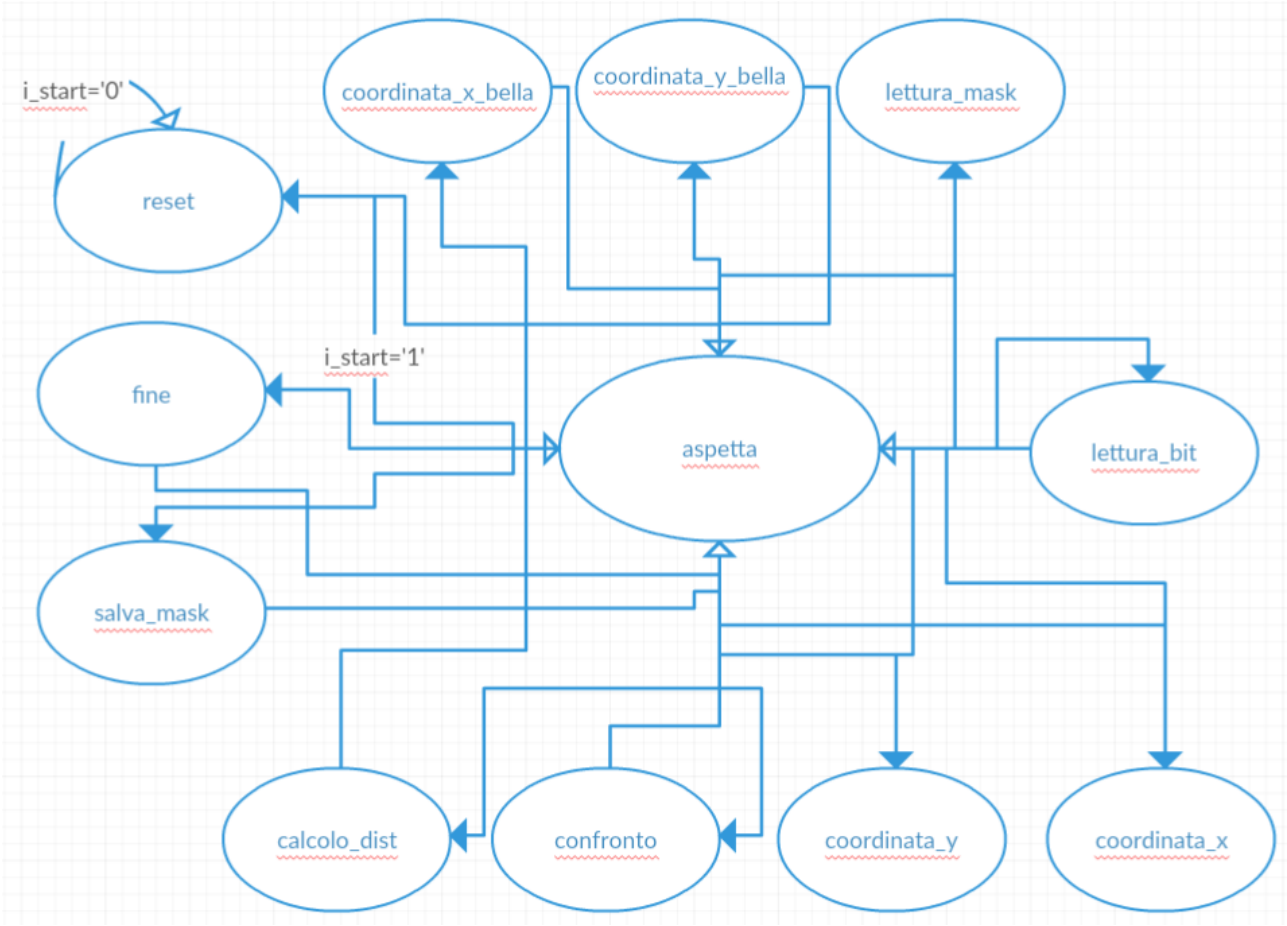
Di seguito è rappresentata una possibile configurazione della situazione iniziale: il centroide da cui misurare la distanza è in posizione (34; 167) e l'output desiderato sarà una maschera di uscita con un solo '1' in corrispondenza del punto (90, 112), il più vicino:



Modellizzazione del componente:

Per analizzare il comportamento del componente HW è utile modellizzarlo con un automa a stati finiti (FSM); questo modello è reso utilizzabile anche dalla presenza di un segnale start e un altro di reset, che gestiscono il suo funzionamento.

Di seguito, uno schema dell'automa con archi e nodi. Le frecce vuote indicano l'arrivo in *aspetta* da ogni stato, quelle piene il contrario, cioè da *aspetta* agli altri stati. Per chiarezza dello schema non sono state esplicitate le frecce che da ogni stato conducono a *reset*, nel caso in cui il segnale *"i_rst"* fosse pari a '1'.



L'automa si compone di 12 stati, in ordine:

- *"reset":* è lo stato di partenza; il segnale "i_start", se posto a '1', sancisce l'inizio della computazione. In questo stato dell'automa, tutti i segnali vengono inizializzati.
- *"coordinata_x_bella":* in questo stato avviene la lettura della coordinata x del centroide da cui bisogna misurare tutte le distanze.
- *"coordinata_y_bella":* in questo stato avviene la lettura della coordinata y del centroide da cui bisogna misurare tutte le distanze.
- *"lettura_mask":* quando si giunge a questo stato, il componente legge dalla memoria la maschera d'ingresso e la memorizza nel segnale appositamente creato, "mask_in". Essa determinerà, successivamente, quali centroidi andranno analizzati.
- *"lettura_bit":* qui viene analizzata l'i-esimo bit della maschera d'ingresso precedentemente salvata; tramite l'utilizzo di un flag ("flag_bit"), l'automa sceglierà se leggere le coordinate

del centroide i-esimo (“flag_bit”=’1’) oppure di leggere il bit successivo della maschera d’ingresso, rimanendo quindi in questo stato.

- *“coordinata_x”:* in questo stato viene letta la coordinata x del centroide i-esimo da analizzare.
- *“coordinata_y”:* in questo stato viene letta la coordinata y del centroide i-esimo da analizzare.
- *“calcolo_dist_xy”:* qua si trovano le distanze lungo la x e lungo la y: $|x_{bella} - x_{centroide\_i\text{-}esimo}|$ (lo stesso per la y); successivamente si giunge nello stato
- *“calcolo_dist”:* in cui le distanze lungo la x e lungo la y vengono sommate tra loro, per trovare la distanza totale.
- *“confronto”:* in questo stato si confronta la distanza appena trovata tramite la somma compiuta nel precedente stato e quella minima finora trovata; se la prima fosse maggiore della seconda, allora si porrebbe il flag “flag_wr” a ‘0’; qualora fossero uguali, “flag_wr” verrebbe posto a ‘1’; altrimenti, se la prima fosse minore della seconda, tutti i bit della maschera d’uscita assumerebbero come valore ‘0’, viceversa “flag_wr” ‘1’ e infine la distanza minima verrebbe aggiornata con la distanza appena trovata (dal centroide i-esimo).
- *“salva_mask”:* qui è dove, in base al valore di “flag_wr”, si scrive sulla maschera d’uscita ‘1’ o ‘0’, rispettivamente nel caso in cui “flag_wr” sia pari a ‘1’ o a ‘0’. Inoltre, tramite il contatore “cnt_mask”, si effettua un controllo sul numero di centroidi ancora da analizzare; nel caso questo segnale sia uguale a 7 (partendo da 0), si dovrà passare allo stato:
- *“fine”:* dove si salva la maschera di output in memoria e si pone il bit “o_done” a ‘1’; lo stato successivo sarà quello di reset, in attesa di un segnale “i_start” posto uguale a ‘1’ per cominciare una nuova computazione. In questo stato, “o_done” viene posto a ‘1’ per segnalare il termine dell’analisi da parte del componente.
- *“aspetta”:* questo è uno stato di transizione in cui l’automa passa dopo ogni altro stato; è stato introdotto per permettere ai segnali provenienti dalla memoria di aggiornarsi dopo l’ingresso nello stato successivo rispetto a quello in cui sono necessari.

Transition Table

Current state	i_start	flag_bit	cnt_mask	stato_vecchio	stato_prossimo
reset	0	x	x	x	reset
reset	1	x	x	x	aspetta
coordinata_x_bella	x	x	x	x	aspetta
coordinata_y_bella	x	x	x	x	aspetta
lettura_mask	x	x	x	x	aspetta
lettura_bit	x	x	x	x	aspetta
coordinata_x	x	x	x	x	aspetta
coordinata_y	x	x	x	x	aspetta
calcolo_dist_xy	x	x	x	x	aspetta
calcolo_dist	x	x	x	x	aspetta
confronto	x	x	x	x	aspetta
salva_mask	x	x	x	x	aspetta
fine	x	x	x	x	aspetta
aspetta	x	x	x	reset	coordinata_x_bella
aspetta	x	x	x	coordinata_x_bella	coordinata_y_bella
aspetta	x	x	x	coordinata_y_bella	lettura_mask
aspetta	x	x	x	lettura_mask	lettura_bit
aspetta	x	0	x	lettura_bit	lettura_bit
aspetta	x	1	x	lettura_bit	coordinata_x
aspetta	x	x	x	coordinata_x	coordinata_y
aspetta	x	x	x	coordinata_y	calcolo_dist_xy
aspetta	x	x	x	calcolo_dist_xy	calcolo_dist
aspetta	x	x	x	calcolo_dist	confronto
aspetta	x	x	x	confronto	salva_mask
aspetta	x	x	<=7	salva_mask	lettura_bit
aspetta	x	x	8	salva_mask	fine

I segnali che consentono il comportamento da FSM del componente sono 3:

- *"stato_ vecchio":* tiene memoria dello stato precedente in cui si trovava l'automa; è stato aggiunto perché sancisce il comportamento dell'automa quando si trova nello stato *aspetta*
- *"stato_corrente":* stato in cui si trova l'automa nell'attuale ciclo di clock;
- *"stato_prossimo":* stato in cui l'automa opererà al ciclo di clock successivo.

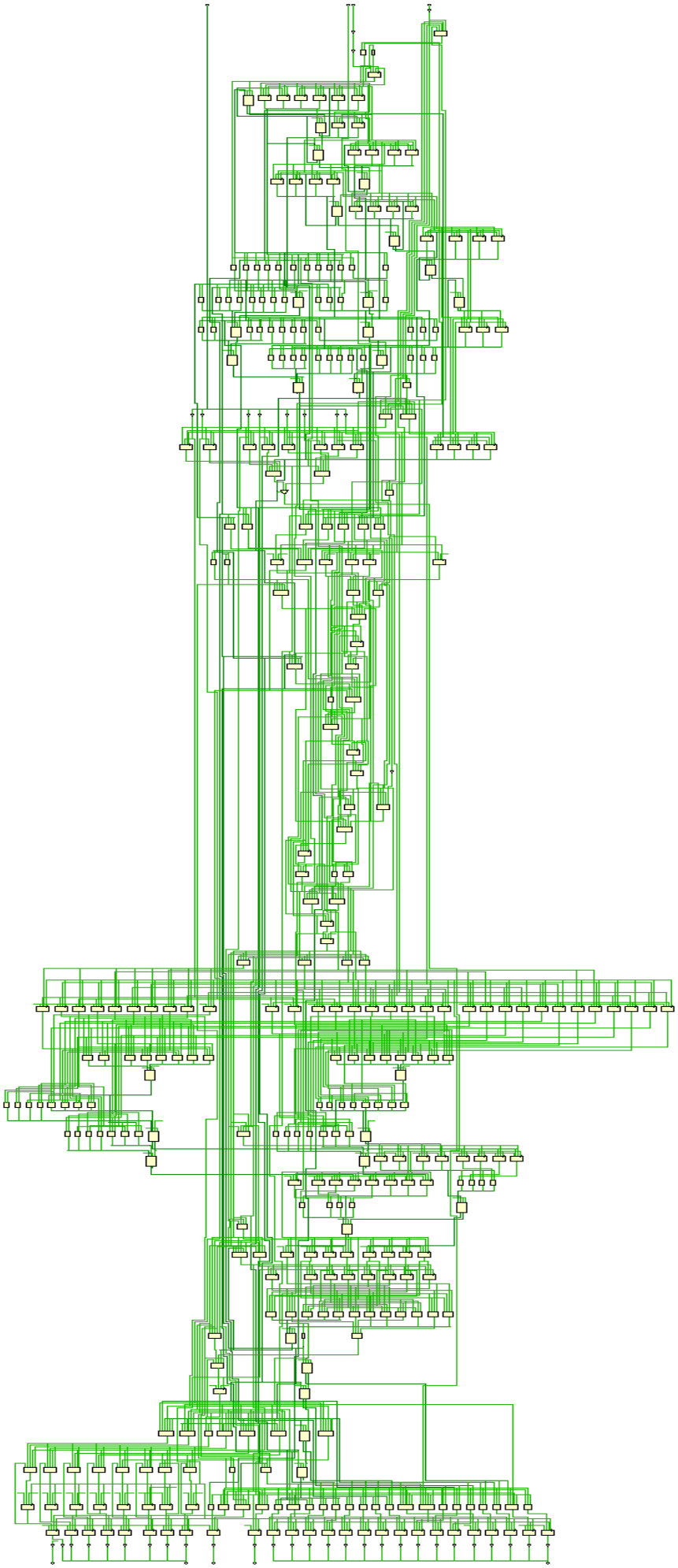
Altri segnali sono:

- *"cnt_mask":* tiene conto del numero di bit della maschera d'ingresso già analizzati; indica, nello stato *salva_mask*, quando finisce la computazione.

- *"flag_bit":* assume il valore i-esimo della maschera d'ingresso; se pari a '1', lo stato successivo a *lettura_bit* sarà *coordinata_x*, altrimenti rimarrà *lettura_bit*.
- *"flag_wr":* flag che vale '1' se nello stato *salva_mask* il bit i-esimo verrà posto a '1', '0' altrimenti.
- *"distanza":* è invece il segnale che nello stato *reset* è inizializzato al valore massimo; successivamente verrà utilizzato per essere confrontato con *"distanza_temp"* al fine di assegnare un valore alto a "flag_wr" nei casi in cui *"distanza_temp<distanza"* o, viceversa, basso se *"distanza_temp>distanza"*.
- *"distanza_temp":* valore della somma delle distanze "parziali" *"dist_x"* e *"dist_y"* dell'i-esimo centroide.

Gli altri segnali sono banali, il loro significato può essere desunto dal loro nome.

Di seguito la rappresentazione schematica del componente tramite Vivado:



Scelte implementative:

Il componente utilizza un singolo processo sia per la parte sequenziale che per quella combinatoria; in particolare, dopo aver controllare che "i_rst" non fosse pari a '1', sono presenti altri due "if-then": il primo

```
if (i_clk'event and i_clk='1') then
```

permette al componente di aggiornare il proprio stato corrente e quello precedente; dopodiche il secondo

```
if (i_clk'event and i_clk='0') then
```

contiene un "case-when" di tutti gli stati che "stato_corrente" può valere; sono aggiornati i segnali e viene inoltre, al termine di ogni "when", aggiornato "stato_prossimo".

Testbench:

Il primo testbench utilizzato per controllare il corretto funzionamento del componente aveva una configurazione della memoria casuale.

```
RAM <= (0 => std_logic_vector(to_unsigned( 185 , 8)),
          1 => std_logic_vector(to_unsigned( 75 , 8)),
          2 => std_logic_vector(to_unsigned( 32 , 8)),
          3 => std_logic_vector(to_unsigned( 111 , 8)),
          4 => std_logic_vector(to_unsigned( 213 , 8)),
          5 => std_logic_vector(to_unsigned( 79 , 8)),
          6 => std_logic_vector(to_unsigned( 33 , 8)),
          7 => std_logic_vector(to_unsigned( 1 , 8)),
          8 => std_logic_vector(to_unsigned( 33 , 8)),
          9 => std_logic_vector(to_unsigned( 80 , 8)),
          10 => std_logic_vector(to_unsigned( 35 , 8)),
          11 => std_logic_vector(to_unsigned( 12 , 8)),
          12 => std_logic_vector(to_unsigned( 254 , 8)),
          13 => std_logic_vector(to_unsigned( 215 , 8)),
          14 => std_logic_vector(to_unsigned( 78 , 8)),
          15 => std_logic_vector(to_unsigned( 211 , 8)),
          16 => std_logic_vector(to_unsigned( 121 , 8)),
          17 => std_logic_vector(to_unsigned( 78 , 8)),
          18 => std_logic_vector(to_unsigned( 33 , 8)),
  others => (others =>'0'));
```

Successivamente la configurazione della memoria è stata modificata per coprire i seguenti casi di test:

1) Maschera di input = ‘00000000’: tramite questo corner test è verificata la maschera di output = ‘00000000’.
2) Maschera di input = ‘11111111’ e centroidi tutti a distanza congruente dal centroide analizzato: in questo caso è stata verificata la maschera di output = ‘11111111’.
3) Segnale di reset posto a 1 durante la computazione: in questo test, con una configurazione della memoria casuale, è stato provato il funzionamento del segnale “i_rst”, che riporta l’automa nello stato di reset:

```
test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    tb_start <= '0';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
```

4) Tre configurazioni di memoria consecutive: per provare il corretto funzionamento dell’automa al termine dell’analisi; è stato utilizzato un “if-elsif” con un contatore per modificare lo stato della memoria:

```
if (count=0) then
    count <= 1;
elsif (count=1) then
    RAM <= (0 => std_logic_vector(to_unsigned( 185 , 8)),
                1 => std_logic_vector(to_unsigned( 75 , 8)),
                2 => std_logic_vector(to_unsigned( 32 , 8)),
                3 => std_logic_vector(to_unsigned( 111 , 8)),
                4 => std_logic_vector(to_unsigned( 213 , 8)),
                5 => std_logic_vector(to_unsigned( 79 , 8)),
                6 => std_logic_vector(to_unsigned( 33 , 8)),
                7 => std_logic_vector(to_unsigned( 1 , 8)),
                8 => std_logic_vector(to_unsigned( 33 , 8)),
                9 => std_logic_vector(to_unsigned( 80 , 8)),
                10 => std_logic_vector(to_unsigned( 35 , 8)),
                11 => std_logic_vector(to_unsigned( 12 , 8)),
                12 => std_logic_vector(to_unsigned( 254 , 8)),
                13 => std_logic_vector(to_unsigned( 215 , 8)),
                14 => std_logic_vector(to_unsigned( 78 , 8)),
                15 => std_logic_vector(to_unsigned( 211 , 8)),
                16 => std_logic_vector(to_unsigned( 121 , 8)),
                17 => std_logic_vector(to_unsigned( 78 , 8)),
                18 => std_logic_vector(to_unsigned( 33 , 8)),
     others => (others =>'0'));
 count <= 2;
elsif (count=2) then
    RAM <= (0 => std_logic_vector(to_unsigned( 0 , 8)),
                1 => std_logic_vector(to_unsigned( 75 , 8)),
                2 => std_logic_vector(to_unsigned( 32 , 8)),
                3 => std_logic_vector(to_unsigned( 111 , 8)),
                4 => std_logic_vector(to_unsigned( 213 , 8)),
                5 => std_logic_vector(to_unsigned( 79 , 8)),
                6 => std_logic_vector(to_unsigned( 33 , 8)),
                7 => std_logic_vector(to_unsigned( 1 , 8)),
                8 => std_logic_vector(to_unsigned( 33 , 8)),
                9 => std_logic_vector(to_unsigned( 80 , 8)),
                10 => std_logic_vector(to_unsigned( 35 , 8)),
                11 => std_logic_vector(to_unsigned( 12 , 8)),
                12 => std_logic_vector(to_unsigned( 254 , 8)),
                13 => std_logic_vector(to_unsigned( 215 , 8)),
                14 => std_logic_vector(to_unsigned( 78 , 8)),
                15 => std_logic_vector(to_unsigned( 211 , 8)),
                16 => std_logic_vector(to_unsigned( 121 , 8)),
                17 => std_logic_vector(to_unsigned( 78 , 8)),
                18 => std_logic_vector(to_unsigned( 33 , 8)),
     others => (others =>'0'));
     count <= 3;
```

5) Test consecutivi e segnale di reset alto prima del termine della computazione: una combinazione dei punti 4 e 3 per un maggior controllo.

Conclusioni:

Il componente supera la Behavioral Simulation, va in post-sintesi e garantisce un output corretto anche per la Post-Synthesis Functional Simulation e la Post-Synthesis Timing Simulation con un periodo di clock di 100ns.

Un altro possibile svolgimento del progetto avrebbe potuto essere una descrizione del componente tramite l'utilizzo di due processi al posto di uno solo; nel primo ci sarebbe stata la parte sequenziale, nel secondo quella combinatoria. Questa suddivisione avrebbe comportato, probabilmente, una maggior chiarezza nella descrizione del componente, tuttavia sarebbe stata soggetta a un maggior rischio di errore che ne avrebbe pregiudicato il corretto funzionamento, in particolare quello di post-sintesi.