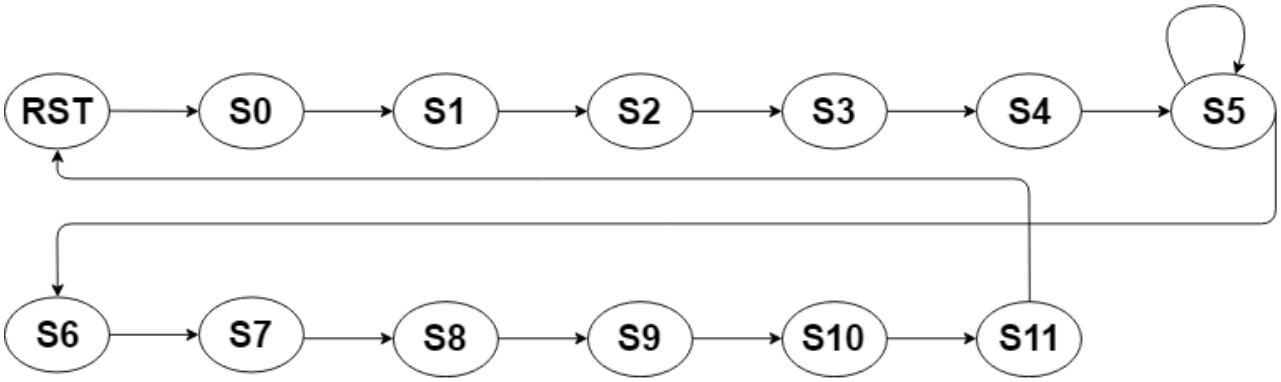
Repo del progetto di reti logiche

Federico Comolli
Matricola: 843616
Cod. Persona: 10496119

Seconda consegna (consegnato la prima volta in data 5/06/2018 con esito negativo)

DESCRIZIONE ANALITICA DELL'IMPLEMENTAZIONE

Per realizzare al meglio la consegna, ho analizzato a lungo la macchina a stati finiti in grado di calcolare l'area minima che racchiude tutti i pixel con valore maggiore o uguale alla soglia.
E' riportato di seguito uno schema sintetico della macchina descritta nel progetto usando il linguaggio VHDL accompagnato da una descrizione analitica del comportamento di ogni singolo stato.



La macchina a stati finiti è descritta utilizzando un unico processo sensibile al segnale i_clk.

La macchina inizialmente si trova nello stato di reset; quando viene asserito i_start la macchina passa allo stato S0. La macchina torna allo stato di reset ogni volta che sul fronte di salita del segnale i_clk, il segnale i_rst vale '1'.
Nello stato RST viene asserito o_en (necessario per poter accedere alla memoria, sia in lettura che in scrittura) e vengono impostati a '0' i segnali o_address e curr_address; curr_address è un segnale interno che viene usato per aggiornare il segnale di output o_address.

```
if(i_rst = '1' or stato = rst) then
            stato <= S0;
            o_en <= '1';
            curr_address <="0000000000000000";
            o_address <= "0000000000000000";
end if;
```

Nello stato S0 vengono resettati tutti i contatori (in questo modo è possibile fare leggere alla FSM due o più aree consecutive, avendo ovviamente l'accortezza di asserire il segnale i_rst tra un'area e la successiva).

In questo stato il segnale o_address viene impostato al valore decimale ‘2’ (l’indirizzo di memoria dove è scritto il numero di colonne dell’immagine in ingresso) e il segnale interno curr_address viene impostato al valore decimale ‘3’ (il valore successivo di o_address).
Sono impostati al valore ‘0’ i seguenti segnali: cont_righe (un segnale interno che viene usato come iteratore sulle righe dell’immagine), cont_colonne (un segnale interno che viene usato come iteratore sulle colonne dell’immagine), n_righe (il segnale che tiene memorizzato il numero di righe dell’immagine), n_colonne (il segnale che tiene memorizzato il numero di colonne dell’immagine), soglia (il segnale che tiene memorizzato la soglia per includere un pixel o meno nel calcolo dell’area), base e altezza (due segnali interni che permettono di calcolare l’area moltiplicandoli tra loro), area, o_done, o_we (il segnale di uscita che permette di scrivere in memoria solo se è asserito) e o_data (il segnale di output verso la memoria).

```
if(i_start = '1' and stato = S0) then
            stato <= S1;
            curr_address<= "0000000000000011";
            o_address <= "0000000000000010";
            --resetto tutti i contatori
            cont_righe <= "00000001";
            cont_colonne <= "00000000";
            n_righe <= "00000000";
            n_colonne <= "00000000";
            soglia <= "00000000";
            base <= "00000000";
            altezza <= "00000000";
            area <= "0000000000000000";
            o_done <= '0';
            o_we <= '0';
            o_data <= "00000000";
end if;
```

Sul fronte di salita del ciclo di clock successivo si passa allo stato S1.
Lo stato S1 ha lo scopo di incrementare di un’unità i segnali o_address e curr_address.
o_address assume il valore decimale ‘3’ (in modo che tra due cicli di clock è possibile leggere il valore delle righe) e curr_address assume il valore successivo. Ogni volta che viene aggiornato o_address viene anche incrementato curr_address, in questo modo è possibile incrementare il segnale o_address.

```
if(stato = S1) then
            stato <= S2;
            curr_address <= curr_address + "0000000000000001";
            o_address <=curr_address;
end if;
```

Nello stato S1 lo stato viene impostato a S2.
In questo stato inizia la lettura dei valori dalla memoria RAM. Nello stato S2 viene letto il numero di colonne dell'immagine dall'indirizzo di memoria '2'. Tale valore viene memorizzato nel segnale interno n_colonne; viene inoltre incrementato il valore di o_address con la coppia di operazione che viene qui ulteriormente ricordata:
viene assegnato a o_address il valore di curr_address e viene incrementato di '1' il valore del segnale interno curr_address.

```
if(stato = S2) then
        stato <= S3;
        n_colonne <= i_data;
        curr_address <= curr_address + "0000000000000001";
        o_address <= curr_address;
end if;
```

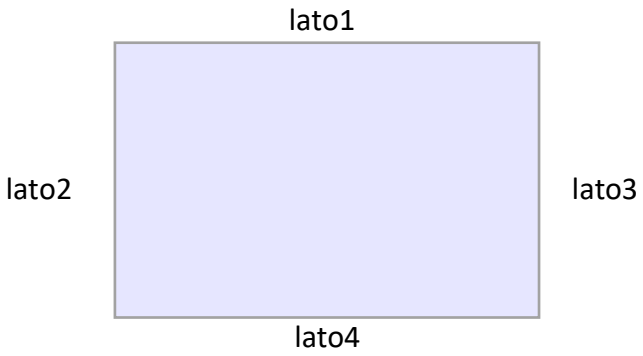
Nello stato S2 lo stato viene impostato a S3.
In questo stato viene letto il numero di righe dell'immagine dall'indirizzo di memoria '3'. Tale valore viene memorizzato nel segnale interno n_righe.
Viene inoltre incrementato il valore di o_address con la coppia di operazione precedentemente descritta.
Il segnale interno stato viene impostato al valore S4.

```
if(stato = S3) then
        stato <= S4;
        n_righe <= i_data;
        curr_address <= curr_address + "0000000000000001";
        o_address <= curr_address;
end if;
```

Nello stato S4 viene letto il valore della soglia in base al quale bisogna includere o meno i pixel nel calcolo dell'area richiesta. Tale valore viene memorizzato nel segnale interno soglia e viene incrementato di '1' il valore del segnale o_address con la corrispondente coppia di operazione.
Viene inoltre incrementato di un'unità il segnale interno cont_colonne, il quale inizia ad iterare lungo tutte le colonne dell'immagine da leggere.
In questo stato vengono resettati al valore '0' quattro segnali che compaiono qua per la prima volta: lato1, lato2, lato3 e lato4.
Questi quattro segnali servono per calcolare l'area richiesta e vengono aggiornati nello stato successivo in base ad una logica descritta più avanti.
Infine, viene impostato a S5 il segnale interno stato.

```
if(stato = S4) then
            lato1 <= n_righe;
            lato4 <= "00000000";
            lato2 <= n_colonne;
            lato3 <= "00000000";
            cont_colonne <= cont_colonne + "00000001";
            stato <= S5;
            soglia <= i_data;
            curr_address <= curr_address + "0000000000000001";
            o_address <= curr_address;
end if;
```

Lo stato S5 racchiude l'intera logica del calcolo dell'area.
La consegna della prova chiede di calcolare l'area del rettangolo MINIMO che racchiude TUTTI i pixel con un valore superiore o uguale al valore della soglia letto nell'indirizzo '4' della RAM.



Ai fini del calcolo dell'area si ipotizzi che quattro segnali sopra raffigurati contengano:

- lato1 il valore più piccolo del segnale cont_righe a cui viene letto un pixel maggiore o uguale della soglia;
- lato2 il valore più piccolo del segnale cont_colonne a cui viene letto un pixel maggiore o uguale della soglia;
- lato3 il valore più grande del segnale cont_colonne a cui viene letto un pixel maggiore o uguale della soglia;
- lato4 il valore più grande del segnale cont_righe a cui viene letto un pixel maggiore o uguale della soglia.

Ad ogni ciclo di clock viene incrementato di un'unità il valore del segnale o_address con la coppia di operazione precedentemente descritta.
Viene inoltre incrementato di una unità, ad ogni ciclo di clock in cui la FSM rimane nello stato S5, il segnale interno cont_colonne; quando però cont_colonne vale quanto il segnale n_colonne, allora viene resettato al valore '0' e viene invece incrementato di una unità il valore cont_righe. In questo modo non esistono due cicli di clock in cui la FSM rimane nello stato S5 e in cui entrambi i valori dei segnali interni cont_righe e cont_colonne rimangono inalterati.
Sulla base di questi due iteratori e sulla base del valore di i_data in ingresso alla FSM (in base al valore di o_address) vengono aggiornati opportunamente i quattro segnali lato1, lato2, lato3 e lato4.

A lettura dell'immagine terminata viene garantito che questi quattro segnali rispettano le proprietà sopra descritte.
La FSM rimane nello stato S5 fino a quando il segnale cont_colonne assume il valore del segnale n_colonne e, contemporanemante, il segnale cont_righe assume il valore del segnale n_righe.
A questo punto è terminata la lettura dell'immagine dalla memoria RAM e la FSM procede con il calcolo dell'area richiesta e passa quindi allo stato S6.
Come ultima operazione viene impostato a '0' il valore del segnale o_address (il primo indirizzo di memoria in cui viene effettuata la scrittura del byte meno significativo dell'area calcolata (su 2 byte); curr_address viene invece impostato, come al solito, al valore successivo di o_address.
Per maggiore chiarezza viene di seguito riportato il frammento di codice corrispondente allo stato S5 appena descritto.

```
if(stato = S5) then
            if(i_data >= soglia) then
               if(n_righe /= 0 and n_colonne /= 0) then
                  --aggiorno i lati
                  if(cont_righe < lato1) then
                     lato1 <= cont_righe;
                  end if;
                  if(cont_righe > lato4) then
                     lato4 <= cont_righe;
                  end if;
                  if(cont_colonne < lato2) then
                     lato2 <= cont_colonne;
                  end if;
                  if(cont_colonne > lato3) then
                     lato3 <= cont_colonne;
                  end if;
               end if;
            end if;
            cont_colonne <= cont_colonne + "00000001";
            if (cont_colonne = n_colonne) then
               cont_colonne <= "00000001";
               cont_righe <= cont_righe + "00000001";
            end if;
            curr_address <= curr_address + "0000000000000001";
            o_address <= curr_address;
            --se ho letto tutta l'immagine vado nello stato S6 e o_address lo imposto a zero
            if(((cont_colonne = n_colonne) and (cont_righe= n_righe)) or n_colonne = "00000000" or
n_righe = "00000000") then
               stato <= S6;
               curr_address <= "0000000000000001";
               o_address <= "0000000000000000";
            end if;
end if;
```

Come si può osservare, se o il segnale n_righe o n_colonne hanno il valore pari a ‘0’, allora si passa immediatamente allo stato successivo e l’area calcolata avrà un valore nullo.
Con lo stato S5 termina la lettura dalla memoria RAM e inizia il processing dell’area da scrivere nella memoria.
A tal proposito, nello stato S6 vengono calcolati i valori del due segnali interni base e altezza.
Come base si intende la base (in numero di righe di pixel) del rettangolo sopra raffigurato;
come altezza, invece, si intende l’altezza (in numero di colonne di pixel) del rettangolo sopra raffigurato.
Questo stato potrebbe essere compattato con il suo stato successivo S7; ma come scelta implementativa, al fine di ridurre il minimo periodo di clock a cui è garantita l’implementazione della FSM, ho deciso di separare l’operazione di calcolo dell’area in due operazione più semplici, eseguite in due cicli di clock successivi. Il calcolo dell’area richiede infatti di eseguire prima due sottrazioni e, successivamente, richiede di eseguire una moltiplicazione tra i risultati delle due sottrazioni. Essendo due operazioni sequenziali e molto onerose in termini di tempo, ho deciso di separarle in due cicli di clock differenti.
In questo stato viene asserito a ‘1’ il segnale o_we, necessario per poter scrivere dei valori nella memoria RAM.
Compiuta questa semplice operazione il segnale stato viene impostato al valore S7.
Per aumentare la chiarezza espositiva viene riportato anche qua, come in tutti gli altri stati, il frammento di codice vhdl corrispondente al comportamento sopra descritto.

```
if(stato = S6) then
            stato <= S7;
            o_we <= '1';
            if(lato4>=lato1) then
               if(lato3>=lato2) then
                  base <= ((lato4-lato1)+1);
                  altezza <= ((lato3-lato2)+1);
               end if;
            end if;
end if;
```

Lo stato S8 conclude l’operazione di calcolo dell’area sopra descritta.
Il segnale area è di 16 bit, coerentemente con il valore massimo dell’area che può essere calcolata da questa FSM. A tal proposito si propone la seguente banale riflessione:
i valori delle colonne e quello delle righe sono espressi su 8 bit (valore massimo possibile 255). Di conseguenza l’area va da un minimo di ‘0’ (se n_righe o n_colonne sono uguali a ‘0’ oppure se la soglia è maggiore di tutti i pixel dell’immagine proposta) ad un massimo di 255*255 se sia n_righe che n_colonne assumono il valore massimo consentito (255) e inoltre la soglia è minore di tutti i pixel “più esterni” dell’immagine proposta. Il valore 255*255 = 65025 può essere espresso con un minimo di log(65025) = 15,9887 → 16 bit.
Questo è il motivo per cui il segnale area ha 16 bit e viene scritta in memoria in due byte distinti.

```
if(stato = S7) then
            stato <= S8;
            area <= base*altezza;
end if;
```

Nello stato S8 viene scritto il primo byte in memoria; nell'indirizzo '0' della memoria RAM vengono scritti i bit da '0' a '7' del segnale interno area.
Viene ricordato che il segnale o_address è stato impostato a '0' alla fine dello stato S5, coerentemente con quanto descritto.

```
if(stato = S8) then
            stato <= S9;
            o_data <= area(7 downto 0);
end if;
```

Nello stato S9 viene scritto il secondo ed ultimo byte in memoria; viene impostato il segnale o_address al valore '1' e vengono scritti in tale indirizzo della memoria RAM i bit da '8' a '15' del segnale interno area.
Viene asserito il segnale o_done, necessario affinché i test-bench possano controllare la correttezza dei valori scritti in memoria e viene impostato a S10 il segnale stato.

```
if(stato = S9) then
            stato <= S10;
            o_data <= area(15 downto 8);
            o_done <= '1';
            o_address <= curr_address;
end if;
```

Nello stato S10, il penultimo, non viene fatto nulla di significativo per quanto riguarda la computazione e la scrittura dell'area richiesta; tuttavia è molto importante questo stato perché consente di lasciare il tempo necessario per poter scrivere nella memoria RAM entrambi i byte, prima di asserire a '0' i segnali o_done e o_we.
Tali segnali sono infatti resettati nello stato successivo (ultimo stato) S11. Una volta che il segnale o_done viene asserito a '0', il test-bench inizierà ad eseguire i due assert che controlleranno la correttezza dell'area computata dalla FSM.
Nello stato S11 il segnale stato viene impostato al valore rst, in modo tale che la FSM è quindi pronta a computare nuovamente un nuovo valore di area partendo da un'immagine letta dalla memoria RAM.

ANALISI NUMERICA DELLA SINTESI E DELL'IMPLEMENTAZIONE

Di seguito sono riportate le caratteristiche tecniche della sintesi e dell'implementazione della FSM descritta.

La sintesi della rete ha 206 LUT e 150 FF, l'implementazione ha 219 LUT e 150 FF.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
✔ synth_1	constrs_1	synth_design Complete!								206	150
✔ impl_1	constrs_1	route_design Complete!	0.120	0.000	0.141	0.000	0.000	0.149	0	219	150

E' garantito che la sintesi e l'implementazione avvengano correttamente fino a un periodo di clock del testbench di 5.6 ns.

```
create_clock -period 5.6 -name i_clk [get_ports i_clk]
```

SIMULAZIONI IN PRESINTESI E IN POST SINTESI EFFETTUATE

Al fine di testare il corretto funzionamento del circuito progettato, ho simulato in pre-sintesi i quattro test-bench proposti dal Dr. Zoni e anche in post-sintesi, ottenendo valutazioni positive. Al fine di assicurarmi che il circuito sia in grado di computare correttamente tutte le area che gli vengano proposte, mi sono servito di un programma in Java che generasse test-bench randomici. E' di seguito riportato il codice Java da me scritto.

```
public class Generator {
    private static final int NUM_OF_TEST = 100;
    private static final String FORMAT = ".txt";
    public static void main(String[] args) {
        int column;
        int row;
        int thresh;
        int area;
        ArrayList<Integer> aree = new ArrayList<>();
        int i;
        ArrayList<Integer> RAM = new ArrayList<>();
        Random rand = new Random();


        PrintWriter writer = null;
        try {
            /****/
            for(i=0; i<NUM_OF_TEST; i++) {
                column = rand.nextInt(256);
                row = rand.nextInt(256);
                thresh = rand.nextInt(256);
                RAM.clear();
                for (int j = 0; j < row * column; j++) {
                    if (rand.nextBoolean() && /****/){
                        RAM.add(rand.nextInt(256));
                    } else RAM.add(0);
                }
                //calcolo dell'area
                int righe;
                int colonne;
                int lato1 = row;
                int lato4 = 0;
                int lato2 = column;
                int lato3 = 0;
                int cont = 0;
                area = 0;
```

```
                for (righe = 1; righe <= row; righe++) {
                    for (colonne = 1; colonne <= column; colonne++) {
                        if (RAM.get(cont) >= thresh) {
                            if (righe < lato1) lato1 = righe;
                            if (righe > lato4) lato4 = righe;
                            if (colonne < lato2) lato2 = colonne;
                            if (colonne > lato3) lato3 = colonne;
                        }
                        cont++;
                    }
                }
                int base = 0;
                int altezza = 0;
                if (lato4 >= lato1){
                    if (lato3 >= lato2) {
                        base = (lato3-lato2)+1;
                        altezza = (lato4-lato1)+1;
                    }
                }
                area = base*altezza;

                if(i==0) writer.write("if count = "+(i)+" then\n");
                writer.write("--test # "+(i+1)+"\n"+"RAM <= (");
                writer.write("2 => \""+String.format("%8s",Integer.toBinaryString(column)).replace('
','0')+"\", ");
                writer.write("3 => \""+String.format("%8s",Integer.toBinaryString(row)).replace('
','0')+"\", ");
                writer.write("4 => \""+String.format("%8s",Integer.toBinaryString(thresh)).replace('
','0')+"\", ");
                for(int index =0; index<RAM.size(); index++){
                    if(RAM.get(index)!=0)
                        writer.write((5+index)+" =>
\""+String.format("%8s",Integer.toBinaryString(RAM.get(index))).replace(' ','0')+"\", ");
                }
                writer.write("others => (others =>'0'));\n");
                if(i==NUM_OF_TEST-1) writer.write("end if;\n");
                    else writer.write("count <= "+(i+1)+";\nelsif count = "+(i+1)+" then\n");
                aree.add(area);
            }

            writer.write("         else\n");
            writer.write("mem_o_data <= RAM(conv_integer(mem_address)) after 1 ns;\n");
            /*****/
            writer.write("begin\n");
            for(int c =0; c<aree.size(); c++){
                writer.write("--test # "+(c+1)+"\nwait for 100 ns;\nwait for c_CLOCK_PERIOD;\ntb_rst
<= '1';\nwait for c_CLOCK_PERIOD;\ntb_rst <= '0';\n");
                writer.write("wait for c_CLOCK_PERIOD;\ntb_start <= '1';\nwait for
c_CLOCK_PERIOD;\ntb_start <= '0';\nwait until tb_done = '1';\nwait until tb_done = '0';\nwait until
rising_edge(tb_clk);\n");
                String risultato = String.format("%16s",
Integer.toBinaryString(aree.get(c))).replace(' ','0');
                writer.write("assert RAM(1) = \""+risultato.subSequence(0, 8)+"\" report \"FAIL high
bits\" severity failure;\n");
                writer.write("assert RAM(0) = \""+risultato.subSequence(8, 16)+"\" report \"FAIL
high bits\" severity failure;\n\n\n");
            }

            writer.write("assert false report \"Simulation Ended!, test passed\" severity
failure;\n");
            writer.write("end process test;\n");
            writer.write("end projecttb;\n");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                writer.close();
            } catch (Exception e) {
            }
        }
    }
}
```

Tale programma mi ha permesso di generare un numero arbitrario di test in sequenza (impostando opportunamente il valore della costante NUM_OF_TEST); in particolare ho generato 100 aree randomiche ogni giorno, per esattamente 15 giorni.
Ho eseguito la simulazione in pre e post-sintesi delle 100 aree ogni giorno, arrivando ad un totale di 1500 aree computate correttamente sia in post-sintesi che in pre-sintesi.
Metto alla luce che, come nei test-bench a noi forniti dal Dr. Zoni, ho inserito un ritardo di 1 ns nel test-bench per assegnare i valori uscenti dalla memoria RAM al segnale mem_o_data.
Il circuito da me proposto non è in grado di computare correttamente l'area in post-sintesi senza tale ritardo; a tal proposito propongo di seguito una descrizione in grado di computarla correttamente anche in assenza di tale ritardo. Tale descrizione aggiorna il segnale interno o_address sul fronte di salita del segnale di clock, mentre legge i valori di i_data sul fronte di discesa del ciclo di clock; procedendo in questo modo riesce a leggere correttamente i valori di i_data indipendentemente dal ritardo introdotto dalla sintesi del circuito.
Non ho consegnato questa seconda versione del circuito per scelta, in quanto la complessità implementativa è maggiore e ho preferito proporre una soluzione più semplice e chiara ma ugualmente funzionante.

```
architecture Behavioral of project_reti_logiche is

    type STATUS is ( rst, S0, S1, S2, S3, S4, S5, S6, S7, S8, S9 ,S10 , S11, S12);
    signal stato: STATUS := rst;
    signal n_righe: std_logic_vector(7 downto 0);
    signal n_colonne: std_logic_vector(7 downto 0);
    signal soglia: std_logic_vector(7 downto 0);
    signal cont_righe: std_logic_vector(7 downto 0) := "00000001";
    signal cont_colonne: std_logic_vector(7 downto 0) := "00000001";
    signal curr_address: std_logic_vector(15 downto 0);
    signal lato1: std_logic_vector(7 downto 0) := "00000000";
    signal lato2: std_logic_vector(7 downto 0) := "00000000";
    signal lato3: std_logic_vector(7 downto 0) := "00000000";
    signal lato4: std_logic_vector(7 downto 0) := "00000000";
    signal base: std_logic_vector(7 downto 0);
    signal altezza: std_logic_vector(7 downto 0);
    signal area: std_logic_vector(15 downto 0);

begin
    --processo logica sequenziale
    process1: process(i_clk, i_data)
    begin
        if( rising_edge(i_clk) ) then
            --ogni volta che siamo sul fronte di salita del clock
            --se viene asserito i_rst lo stato diventa lo sato iniziale S0
            if(i_rst = '1' or stato = rst) then
                stato <= S0;
                o_en <= '1';
                curr_address <="0000000000000000";
                o_address <= "0000000000000000";
            end if;
```

```
--se viene asserito i_start viene cambiato stato solo
   se la rete è nello stato iniziale S0
if(i_start = '1' and stato = S0) then
   stato <= S1;
   curr_address<= "0000000000000011";
   o_address <= "0000000000000010";
   --resetto tutti i contatori
   --n_righe <= "00000000";
   --n_colonne <= "00000000";
   --soglia <= "00000000";
   base <= "00000000";
   altezza <= "00000000";
   area <= "0000000000000000";
   o_done <= '0';
   o_we <= '0';
   o_data <= "00000000";
   cont_righe <= "00000001";
   cont_colonne <= "00000000";
end if;

if(stato = S1) then
   stato <= S2;
   curr_address <= curr_address + "0000000000000001";
   o_address <=curr_address;
end if;

--leggo n_colonne
if(stato = S2) then
   stato <= S3;
   --n_colonne <= i_data;
   curr_address <= curr_address + "0000000000000001";
   o_address <= curr_address;
end if;

--leggo n_righe
if(stato = S3) then
   stato <= S4;
   --n_righe <= i_data;
   curr_address <= curr_address + "0000000000000001";
   o_address <= curr_address;
end if;

--leggo la soglia
if(stato = S4) then
   stato <= S5;
   --soglia <= i_data;
   curr_address <= curr_address + "0000000000000001";
   o_address <= curr_address;
   cont_colonne <= cont_colonne + "00000001";
end if;

if(stato = S5) then
   stato <= S6;
   curr_address <= curr_address + "0000000000000001";
   cont_colonne <= cont_colonne + "00000001";
end if;

--leggo l'immagine rimanendo nello stato S6
if(stato = S6) then
   curr_address <= curr_address + "0000000000000001";
   o_address <= curr_address;
   if(((cont_colonne = n_colonne) and (cont_righe= n_righe))
      or n_colonne = "00000000" or n_righe = "00000000")
      then
      stato <= S7;
      curr_address <= "0000000000000001";
      o_address <= "0000000000000000";
   end if;
   cont_colonne <= cont_colonne + "00000001";
   if (cont_colonne = n_colonne) then
      cont_colonne <= "00000001";
      cont_righe <= cont_righe + "00000001";
   end if;
end if;

--ho letto l'intera immagine e calcolo base e altezza
if(stato = S7) then
   stato <= S8;
   o_we <= '1';
   if(lato4>=lato1) then
      if(lato3>=lato2) then
         base <= ((lato4-lato1)+1);
         altezza <= ((lato3-lato2)+1);
      end if;
   end if;
end if;

--calcolo l'area in S8
if(stato = S8) then
   stato <= S9;
   area <= base*altezza;
end if;

--scrivo il primo byte in memoria
if(stato = S9) then
   stato <= S10;
   o_data <= area(7 downto 0);
end if;

--scrivo il secondo byte in memoria
if(stato = S10) then
   stato <= S11;
   o_data <= area(15 downto 8);
   o_done <= '1';
   o_address <= curr_address;
end if;

if(stato = S11) then
   --o_done <= '0';
   stato <= S112;
   curr_address <= "0000000000000010";
   --o_we <= '0';
end if;

--stato finale
if(stato = S12) then
   o_done <= '0';
   o_we <= '0';
   stato <= rst;
end if;

  end if;
end process;
```

```
--secondo processo
   process2: process(i_clk)
   begin
      if(falling_edge(i_clk)) then

         if(stato = S2) then
            n_colonne <= i_data;
         end if;

         if(stato = S3) then
            n_righe <= i_data;
         end if;


         if(stato = S4) then
            soglia <= i_data;
            lato1 <= n_righe;
            lato4 <= "00000000";
            lato2 <= n_colonne;
            lato3 <= "00000000";
         end if;

         --nello stato S6 aggiorno i contatori
         if(stato = S6) then
            if(i_data >= soglia) then
               if(n_righe /= 0 and n_colonne /= 0) then
                  if(cont_righe < lato1) then
                     lato1 <= cont_righe;
                  end if;
                  if(cont_righe > lato4) then
                     lato4 <= cont_righe;
                  end if;
                  if(cont_colonne < lato2) then
                     lato2 <= cont_colonne;
                  end if;
                  if(cont_colonne > lato3) then
                     lato3 <= cont_colonne;
                  end if;
               end if;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

Come già evidenziato, questa seconda soluzione presenta due processi, il primo che è sensibile al fronte di salita del ciclo di clock mentre il secondo che è sensibile invece al fronte di discesa di tale segnale. Il primo processo incrementa correttamente il segnale di o_address mentre il secondo legge sul fronte di discesa il valore corrispondente di i_data.