**Lavoro di gruppo di:**

**848834 - 10523360 - Ali El Wahsh  
847218 - 10520143 - Mathyas Giudici**

DOCUMENTAZIONE ALLEGATA AL PROGETTO DI RETI LOGICHE

I. PREFAZIONE 2

II. SEGNALI GESTIONE STATI FSM 2

III. SEGNALI GESTIONE MEMORIA e contatori 2

IV. SEGNALI DI SUPPORTO 3

V. DESCRIZIONE STATI 3

A. RESET 3

B. INIZIALE 3

C. CONFRONTO 3

D. STATO\_MOLTIPLICA 3

E. SALVA 3

F. ASPETTA 3

VI. implementazione architettura 4

VII. TESTBENCH E TESTING 6

VIII. SCHEMA LOGICO 7

# PREFAZIONE

Al fine di poter implementare una rete logica in grado di calcolare l’area del rettangolo minimo che circoscrive la figura in memoria (dato un valore di soglia) si decide di sviluppare il progetto verso un’ottimizzazione in tempo di esecuzione.

Vi è da considerare che il problema presentato ha una complessità θ(n), dove n è il numero di elementi della memoria RAM in cui vi è salvata l’immagine. Infatti non vi è modo di determinare quale sia il rettangolo più grande senza aver scandito almeno una volta per intero l’immagine.

*Nell’algoritmo da noi implementato vi è una scansione sequenziale della memoria; tale algoritmo implica che bisogna arrivare fino all’ultimo indirizzo disponibile per aver effettuato tutti i confronti, di soglia, necessaria ad aver trovato il rettangolo più grande. Un altro algoritmo potrebbe essere quello di visita “a spirale”; questo algoritmo ridurrebbe sensibilmente il tempo nel caso medio (rettangolo “aderente” al bordo) dove si avrebbe una complessità logaritmica; tuttavia il caso pessimo (rettangolo non esistente) implicherebbe lo stesso una visita di tutta la memoria e complessità lineare.*

Se da un lato si ha un vantaggio nella velocità della risposta del sistema si perderà dal punto di vista di dimensionale; il circuito sintetizzato a partire dal codice VHDL occuperà una considerevole area.

# SEGNALI GESTIONE STATI FSM

In prima istanza si decide di definire un nuovo tipo “stato”.

*type state\_type is (reset,iniziale,confronto,stato\_moltiplica,salva,aspetta);*

In questo modo si possono definire i seguenti segnali:

*signal stato\_corrente, stato\_prossimo : state\_type;*

il primo rappresenta lo stato in cui si trova la macchina, mentre il secondo rappresenta lo stato in cui la macchina si troverà al successivo fronte di salita.

# SEGNALI GESTIONE MEMORIA e contatori

*signal soglia, colonne, righe: std\_logic\_vector(7 downto 0);*

Nello stato “iniziale” della macchina vengono in questi segnali vengono caricati i rispettivi valori salvati in memoria. Questi segnali non verranno modificati per tutta l’esecuzione.

*signal nord, sud, west, est: std\_logic\_vector(7 downto 0);*

Questi segnali sono usati per salvare le coordinate massime; sono di supporto per il calcolo dell’area del rettangolo.

*signal coordc, coordr: std\_logic\_vector (7 downto 0);*

Rappresentano rispettivamente la coordinata di colonna e coordinata di riga su cui si sta lavorando durante lo stato “confronto”.

*signal prossimo\_address: std\_logic\_vector (15 downto 0);*

Rappresenta il prossimo indirizzo (di memoria) che verrà caricato in *o\_address* al successivo ciclo di clock.

# SEGNALI DI SUPPORTO

*signal phase: std\_logic\_vector(1 downto 0);*

Questo segnale tiene traccia del caricamento dei valori durante lo stato “iniziale”.

*signal moltiplica: std\_logic\_vector(15 downto 0);*

Dopo lo stato “moltiplica” contiene il risultato del calcolo dell'area del rettangolo.

*signal productphase: std\_logic\_vector (1 downto 0);*

Questo segnale tiene traccia della fase di caricamento in memoria del risultato della moltiplicazione.

# DESCRIZIONE STATI

L’entità project\_reti\_logiche è stata da noi implementata come una macchina a stati finiti (FSM).

La macchina è articolata in sei stati:

## RESET

Questo è lo stato iniziale della macchina. Vi si giunge attraverso il segnale di *reset* del testbench. In questo stato si attende il segnale di *start*; dopo il quale vengono inizializzati i segnali interni alla macchina e si passa allo stato successivo.

## INIZIALE

Una volta terminata l’inizializzazione dei segnali all’interno dello stato di reset si giunge in questo stato dove vengono salvati nei segnali *colonne, righe, soglia* i valori salvati in memoria che rimarranno costanti per tutta la durata dell’esecuzione.

## CONFRONTO

Questo è lo stato più importante della macchina, in questo stato avviene il confronto tra gli elementi salvati in memoria con il valore di soglia. Se il confronto da esito positivo vengono aggiornati di conseguenza i valori dei segnali di *nord, sud, est* e *west*.

Al termine della lettura in memoria si passa al successivo stato per il calcolo dell’area.

## STATO\_MOLTIPLICA

In questo stato avviene la moltiplicazione per il calcolo dell’area del rettangolo in cui vi è contenuta l’immagine secondo la logica:

*moltiplica <= (sud - nord + "00000001") \* (est - west + "00000001");*

Il più *“00000001”* viene introdotto perché il calcolo dell’area deve tenere conto degli elementi che compongono il “bordo”.

## SALVA

In questo stato vi si giunge dopo aver calcolato l’area richiesta; in questo stato si procede al salvataggio in memoria dell’area. Si innalza anche il segnale per il testbench *done*.

## ASPETTA

È l’ultimo stato della macchina. Permette di abbassare il livello *done* e permettere al testbench di eseguire la valutazione della correttezza dell’area calcolata.

# implementazione architettura

*architecture Behavioral of project\_reti\_logiche is*

*begin*

*funzione: process(i\_clk,i\_rst)*

*begin*

*if (i\_rst='1') then*

*stato\_corrente<= reset;*

*elsif (i\_clk'event and i\_clk='1') then*

*stato\_corrente <= stato\_prossimo;*

*end if;*

*if (i\_clk'event and i\_clk='0') then*

*case stato\_corrente is*

*when reset =>*

*if(i\_start = '0') then*

*stato\_prossimo <= reset;*

*else*

*nord <= "ZZZZZZZZ";*

*sud <= "ZZZZZZZZ";*

*west <= "ZZZZZZZZ";*

*est <= "ZZZZZZZZ";*

*o\_address <="0000000000000010";*

*coordr <="00000001";*

*coordc <="00000001";*

*phase <= "00";*

*o\_en <= '1';*

*o\_we <= '0';*

*o\_done <= '0';*

*stato\_prossimo <= iniziale;*

*end if;*

*when iniziale =>*

*case phase is*

*when "00" =>*

*colonne <= i\_data;*

*o\_address <="0000000000000011";*

*o\_en <= '1';*

*o\_we <= '0';*

*phase <= "01";*

*when "01" =>*

*righe <= i\_data;*

*o\_address <="0000000000000100";*

*o\_en <= '1';*

*o\_we <= '0';*

*phase <= "10";*

*when "10" =>*

*soglia <= i\_data;*

*phase <= "00";*

*if (righe="00000000" or colonne="00000000") then*

*stato\_prossimo <= stato\_moltiplica;*

*o\_address <="0000000000000000";*

*prossimo\_address <="0000000000000001";*

*else*

*stato\_prossimo <= confronto;*

*o\_address <="0000000000000101";*

*prossimo\_address <="0000000000000110";*

*o\_en <= '1';*

*o\_we <= '0';*

*end if;*

*if (righe = “00000000” or colonne = “00000000”) then*

*stato\_prossimo <= stato\_moltiplica;*

*else*

*stato\_prossimo <= confronto;*

*when others => stato\_prossimo <= aspetta;*

*end case;*

*when confronto =>*

*if(i\_data >= soglia) then*

*if (nord = "ZZZZZZZZ") then*

*nord <= coordr;*

*sud <= coordr;*

*est <= coordc;*

*west <= coordc;*

*else*

*sud <= coordr;*

*if(coordc < west) then*

*west <= coordc;*

*elsif(coordc > est) then*

*est <= coordc;*

*end if;*

*end if; -- fine if dove setto*

*end if; -- fine controllo soglia*

*if (coordr = righe and coordc = colonne) then*

*o\_address <= "0000000000000000";*

*prossimo\_address <= "0000000000000001";*

*o\_en <= '0';*

*o\_we <= '0';*

*stato\_prossimo <= stato\_moltiplica;*

*else*

*o\_address <= prossimo\_address;*

*prossimo\_address <= prossimo\_address + "0000000000000001";*

*o\_en <= '1';*

*o\_we <= '0';*

*if(coordc = colonne) then*

*coordc <= "00000001";*

*coordr <= coordr + "00000001";*

*else*

*coordc <= coordc + "00000001";*

*end if;*

*end if;*

*when stato\_moltiplica =>*

*if(nord = "ZZZZZZZZ" ) then*

*moltiplica <= "0000000000000000";*

*else*

*moltiplica <= (sud - nord + "00000001") \* (est - west + "00000001");*

*end if;*

*productphase <= "00";*

*stato\_prossimo <= salva;*

*when salva =>*

*case productphase is*

*when "00" =>*

*o\_en <= '1';*

*o\_we <= '1';*

*o\_data <= moltiplica (7 downto 0);*

*productphase <= "01";*

*when "01" =>*

*o\_en <= '0';*

*o\_we <= '0';*

*o\_address <= prossimo\_address;*

*productphase <= "10";*

*when "10" =>*

*o\_en <= '1';*

*o\_we <= '1';*

*o\_done <= '1';*

*o\_data <= moltiplica (15 downto 8);*

*productphase <= "11";*

*when "11" =>*

*o\_en <= '1';*

*o\_we <= '1';*

*o\_done <= '0';*

*stato\_prossimo <= aspetta;*

*when others => stato\_prossimo <= aspetta;*

*end case;*

*when aspetta =>*

*o\_done <= '0';*

*when others => o\_done <= '0';*

*end case;*

*end if;*

*end process;*

*end Behavioral;*

# TESTBENCH E TESTING

# VirtualBox_Windows%2010_27_02_2018_11_53_07.pngVirtualBox_Windows%2010_27_02_2018_11_52_18.pngVirtualBox_Windows%2010_27_02_2018_11_52_02.pngSCHEMA LOGICO