

---

---

# Prova Finale

## (Progetto di reti logiche)

---

---

Daniele Chiappalupi

Matricola: 868120

Codice Persona: 10536115

Professore William Fornaciari

Tutor Davide Zoni

Anno Accademico 2018/2019

# INDICE

---

<b>1. Introduzione</b>	<b>1</b>
------------------------	----------

---

1.1. Componenti	1
-----------------	---

<b>2. Design</b>	<b>2</b>
------------------	----------

---

2.1. Rappresentazione schematica	2
----------------------------------	---

2.2. Segnali	2
--------------	---

2.3. Stato idle	3
-----------------	---

2.4. Stato start	3
------------------	---

2.5. Stati takex, takey, mem	3
------------------------------	---

2.6. Stati c...x, c...y, c..., c...d	3
--------------------------------------	---

2.7. Stati comp_min, comp	3
---------------------------	---

2.8. Stati stop, done	3
-----------------------	---

<b>3. Testing</b>	<b>4</b>
-------------------	----------

---

3.1. Tutti i centroidi attivi	4
-------------------------------	---

3.2. Nessun centroide attivo	4
------------------------------	---

3.3. Solo un centroide attivo	4
-------------------------------	---

3.4. Testing generale randomico	5
---------------------------------	---

<b>4. Ottimizzazioni</b>	<b>5</b>
--------------------------	----------

---

## 1. Introduzione

Lo scopo del progetto è la realizzazione di un componente hardware in VHDL che, dato uno spazio bidimensionale definito in termini di dimensione orizzontale e verticale e le posizioni di otto *centroidi* appartenenti a tale spazio, sia in grado di trovare il/i centroide/i a distanza minima rispetto ad un punto di riferimento, anch'esso immerso nello spazio sopra descritto.

Ulteriori specifiche aggiungono che, degli otto centroidi in ingresso, saranno da considerare solo quelli resi validi da una maschera di ingresso a 8 bit.

### 1.1. Componenti

L'interfaccia del componente, come indicato dalla specifica, è la seguente:

```
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 componente dovrà interfacciarsi con una memoria RAM da 65536 Byte, all'interno della quale i dati sono organizzati secondo lo schema sottostante:

[0]	Maschera di Ingresso
[1]	Coordinata X - Centroide 1
[2]	Coordinata Y - Centroide 1
[3]	Coordinata X - Centroide 2
[4]	Coordinata Y - Centroide 2
...	...
[15]	Coordinata X - Centroide 8
[16]	Coordinata Y - Centroide 8
[17]	Coordinata X - Punto di Riferimento
[18]	Coordinata Y - Punto di Riferimento
[19]	Maschera di Output
...	...

Di conseguenza, ogni dato ha la dimensione di 8 bit, e il componente dovrà scrivere il risultato della propria computazione nell'indirizzo 19 della memoria.

## 2. Design

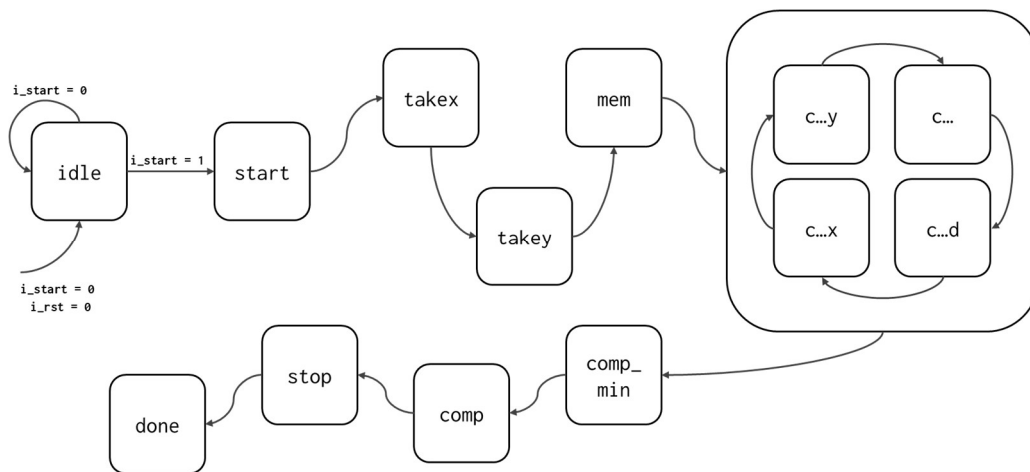
Date le specifiche del problema, è ragionevole pensare di risolverlo attraverso la realizzazione di una macchina a stati finiti (*FSM*). L'idea di base è molto semplice: dopo aver salvato la posizione del punto di riferimento e la maschera dei centroidi da considerare utilizzando degli opportuni segnali, per ogni centroide attivo sarà necessario calcolare la distanza dal punto iniziale. Dopodiché, per semplicità di progetto, piuttosto che aggiornare il risultato ad ogni distanza valutata, ho scelto di proseguire dedicando uno stato al calcolo della distanza minima tra tutte quelle compute, e successivamente di rendere attivi nella maschera di uscita tutti i centroidi che sono da considerare ed hanno distanza uguale a quella minima appena calcolata. Gli stati usati sono i seguenti:

```
type fsm_type is (idle, start, takex, takey, mem, c0x, c0y, c0, c0d, c1x, c1y, c1, c1d, c2x, c2y, c2, c2d, c3x, c3y, c3, c3d, c4x, c4y, c4, c4d, c5x, c5y, c5, c5d, c6x, c6y, c6, c6d, c7x, c7y, c7, c7d, comp_min, comp, stop, done);
```

Segue uno schema della macchina ed una breve descrizione dei singoli stati e dei segnali usati.

### 2.1. Rappresentazione Schematica

Per ragioni di semplicità e leggibilità, non propongo il vero schema ma riporto uno pseudo-diagramma degli stati della macchina. Si noti che la specifica è stata interpretata in modo che, nel caso in cui venga abbassato il segnale di start durante la computazione, la macchina si riporti nello stato iniziale, pronta per riceverne uno nuovo. La ridondanza del segnale *i\_start* nella rappresentazione del primo stato evidenzia proprio questa interpretazione. Inoltre, si osservi che, una volta iniziata l'esecuzione, gli stati vengono percorsi tutti in qualsiasi situazione (tranne nel caso in cui si alzi il segnale di reset o si abbassi quello di start): le motivazioni di questa scelta saranno spiegate nella sezione 4.



### 2.1. Segnali

```
signal ss, ss_next : fsm_type;
signal c_attivi, c_attivi_tmp, risultato, risultato_tmp : std_logic_vector(7 downto 0);
signal d0, d1, d2, d3, d4, d5, d6, d7, d0_tmp, d1_tmp, d2_tmp, d3_tmp, d4_tmp, d5_tmp, d6_tmp, d7_tmp : unsigned(8 downto 0);
signal x, y, xc, yc, x_tmp, y_tmp, xc_tmp, yc_tmp : unsigned(7 downto 0);
signal min, min_tmp : unsigned(8 downto 0);
```

- **ss, ss\_next**: sono i segnali che governano i cambiamenti di stato; ss rappresenta lo stato corrente, ss\_next quello futuro;
- **c\_attivi**: è il segnale nel quale viene salvata la maschera dei centroidi da considerare;
- **risultato**: è il segnale nel quale verrà inserita la maschera di uscita;

- **d0, d1, d2, d3, d4, d5, d6, d7**: sono i segnali nei quali vengono memorizzate le distanze dei vari centroidi dal punto di riferimento;
- **x, y**: sono i segnali dove vengono salvate le coordinate del punto di riferimento;
- **xc, yc**: sono i segnali dove, di volta in volta, vengono salvate le coordinate del centroide in analisi;
- **min**: è il segnale dove viene memorizzata la distanza minima del problema;
- **...tmp**: sono tutti i segnali necessari per far sì che i valori vengano conservati ad ogni ciclo di clock senza che nessun latch venga inferito;

## 2.2. Stato **idle**

È lo stato iniziale della macchina: ci si trova in questa situazione quando si è in attesa del segnale di start che fa partire la computazione. Se, durante l'esecuzione, il segnale di reset viene portato ad 1 o il segnale di start viene portato a 0, la macchina tornerà in questo stato.

## 2.3. Stato **start**

È lo stato di inizio della computazione: vengono inizializzati il risultato (a "00000000") e il minimo (a "11111111"). Inoltre, viene abilitata la memoria per la lettura dei dati all'indirizzo 0, dove è presente la maschera dei bit da utilizzare, che verrà memorizzata in **c\_attivi** nello stato successivo.

## 2.4. Stati **takex, takey, mem**

I primi due sono gli stati in cui viene abilitata la memoria per la lettura delle coordinate x ed y del punto di riferimento. Siccome i dati saranno visibili al ciclo di clock successivo, la coordinata x verrà memorizzata nello stato **takey**, mentre la coordinata y nello stato **mem**.

## 2.5. Stati **c...x, c...y, c..., c...d**

Sono gli stati che servono a calcolare la distanza dei vari centroidi dal punto di riferimento. Gli stati **c...x**, **c...y** e **c...** servono ad abilitare la memoria a leggere le coordinate del centroide ed a memorizzarle, mentre **c...d** è lo stato in cui viene computata la distanza.

Si noti che i segnali nei quali vengono memorizzate le distanze sono lunghi 9 bit: questa scelta è dovuta al fatto che, per salvare la distanza tra il punto di riferimento ed il centroide, possono essere necessari più di 8 bit (ad esempio, nel caso in cui il punto di riferimento sia nella posizione (0, 0) e il centroide in analisi sia nella posizione (255, 255): la distanza verrebbe uguale a 510, e 8 bit non basterebbero per salvare tale numero). Ciò significa che, essendo gli operatori degli unsigned lunghi 8 bit, dovrò estenderli concatenando uno '0' al loro inizio.

## 2.6. Stati **comp\_min, comp**

In questi due stati vengono calcolati, rispettivamente, la distanza minima tra tutte quelle calcolate e il risultato finale della computazione. Osservando il primo, è facile dedurre il motivo per cui tutte le distanze dei centroidi inattivi siano inizializzate al valore massimo (il sistema di valutazione, infatti, confronta tra loro tutte le distanze possibili). Nel secondo, la computazione è fatta nel modo più elementare, segnando tutti i centroidi che sono attivi ed hanno distanza uguale a quella minima.

## 2.7. Stati **stop, done**

Sono gli stati finali: vengono usati per scrivere in memoria il risultato ed inviare il segnale di done.

### 3. Testing

Il progetto è stato esaustivamente testato sottoponendolo a più di 100 casi di test. Infatti, oltre a tutti i fondamentali casi limite appositamente creati per testare il programma in condizioni speciali, ho provveduto a scrivere un programma in C (che verrà brevemente trattato nel paragrafo 3.4) capace di generare test in maniera randomica. Tutti i casi di test effettuati hanno dato esito positivo sia in simulazione *behavioural*, sia in simulazione *post-sintesi functional*, sia in simulazione *post-sintesi timing*, dove ho utilizzato un vincolo di clock a periodo 100 nanosecondi. Di seguito vengono descritti nel dettaglio gli end-case testati.

#### 3.1. Tutti i centroidi attivi

È un caso limite estremamente rilevante: partendo da tutti i centroidi attivi, si possono creare diverse combinazioni particolari (di seguito elencate) che potrebbero mettere in discussione il corretto funzionamento del componente.

- Centroidi sovrapposti e punto di riferimento nella loro stessa posizione;
- Centroidi sovrapposti e punto di riferimento in una posizione diversa;
- Alcuni centroidi in posizione con distanza maggiore di 256 dal punto di riferimento ed altri centroidi con distanza uguale a 255 dal punto di riferimento;
- Centroidi posizionati tutti agli estremi dello spazio bidimensionale;

Tutti questi test vanno a stimolare e verificare il corretto funzionamento della macchina a stati nelle fasi di calcolo delle distanze e computazione del risultato. Essendo tutti i centroidi attivi, svolgendo i casi di test sopracitati si verifica che ogni stato reagisca coerentemente al proprio scopo. Il terzo caso è fondamentale per controllare che le distanze siano correttamente calcolate e confrontate anche quando superano la rappresentazione ad 8 bit.

#### 3.2. Nessun centroide attivo

È il caso duale al precedente: è ovviamente necessario verificare che, nel caso in cui non ci siano punti attivi, il componente restituisca il risultato corretto: nello stato *comp* deve essere compiuto un efficiente controllo della maschera dei centroidi attivi, indipendentemente dalle loro distanze (anche se queste non vengono calcolate nel caso in cui il centroide non sia attivo) e questo caso di test lo verifica. Inoltre, utilizzando degli opportuni segnali di controllo nella finestra di debug delle simulazioni in Xilinx, è possibile verificare che le varie distanze non siano effettivamente calcolate quando il centroide non è attivo.

#### 3.3. Solo un centroide attivo

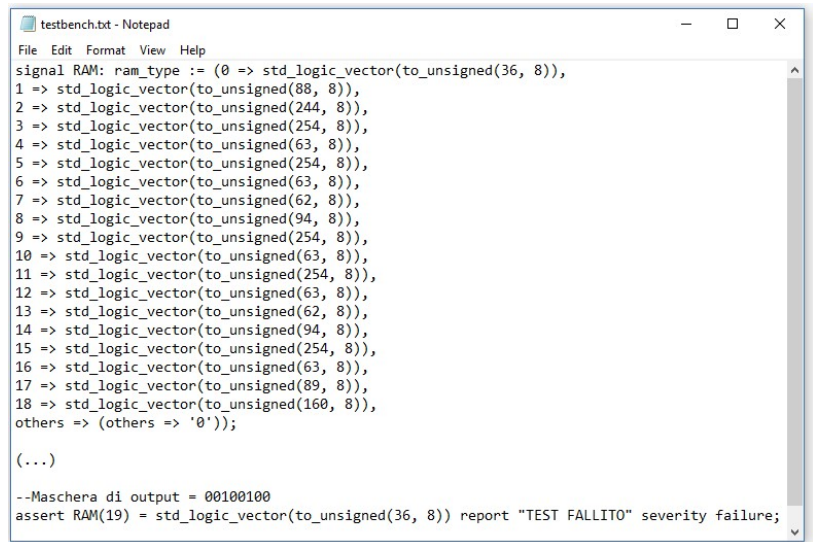
È importante verificare che solo il centroide corretto venga considerato ai fini del risultato e che solo di quello venga calcolata la distanza. Questo caso di test è stato ripetuto per ogni centroide: così facendo si è verificato il corretto funzionamento di tutti gli stati di computazione della macchina. Inoltre, mettendo l'unico centroide attivo ed il punto di riferimento ad estremi opposti dello spazio bidimensionale, è possibile analizzare ulteriormente il componente nei casi in cui la distanza non sia rappresentabile con 8 bit.

### 3.4. Testing generale randomico

Per verificare l'effettivo funzionamento della macchina in condizioni generali è stato programmato un eseguibile tramite il linguaggio di programmazione C, capace di generare testbenches in maniera randomica.

Il suo utilizzo è molto semplice: il programma genera un file di testo con tutte le righe di codice necessarie per il testbench, ovvero l'assegnamento

dei valori agli indirizzi della memoria e l'asserzione finale che verifica il corretto esito del test. Una volta generato, basta copiare ed incollare queste righe al posto di quelle precedentemente implementate nel testbench fornito su beep per averne uno nuovo e perfettamente funzionante (nell'immagine vi è un esempio di output del programma).



```
testbench.txt - Notepad
File Edit Format View Help
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned(36, 8)),
1 => std_logic_vector(to_unsigned(88, 8)),
2 => std_logic_vector(to_unsigned(244, 8)),
3 => std_logic_vector(to_unsigned(254, 8)),
4 => std_logic_vector(to_unsigned(63, 8)),
5 => std_logic_vector(to_unsigned(254, 8)),
6 => std_logic_vector(to_unsigned(63, 8)),
7 => std_logic_vector(to_unsigned(62, 8)),
8 => std_logic_vector(to_unsigned(94, 8)),
9 => std_logic_vector(to_unsigned(254, 8)),
10 => std_logic_vector(to_unsigned(63, 8)),
11 => std_logic_vector(to_unsigned(254, 8)),
12 => std_logic_vector(to_unsigned(63, 8)),
13 => std_logic_vector(to_unsigned(62, 8)),
14 => std_logic_vector(to_unsigned(94, 8)),
15 => std_logic_vector(to_unsigned(254, 8)),
16 => std_logic_vector(to_unsigned(63, 8)),
17 => std_logic_vector(to_unsigned(89, 8)),
18 => std_logic_vector(to_unsigned(160, 8)),
others => (others => '0'));

(...)

--Maschera di output = 00100100
assert RAM(19) = std_logic_vector(to_unsigned(36, 8)) report "TEST FALLITO" severity failure;
```

### 4. Ottimizzazioni

Il componente è stato costruito seguendo il principio base della semplicità, ed è stato sottoposto a progressive migliorie in fase di simulazione post-sintesi timing: una delle scelte più importanti effettuate in questo stadio è stata quella di eseguire tutti gli stati della macchina indipendentemente dalla configurazione della maschera di ingresso. Infatti, inizialmente negli stati computazionali centrali del componente erano presenti dei cicli if/elsif che indirizzavano l'esecuzione solo negli stati dei centroidi attivi. Questo tipo di approccio risultava particolarmente poco efficiente dal punto di vista delle simulazioni post-sintesi timing, di conseguenza si è adottato un codice più leggero, che passa in ogni esecuzione in tutti gli stati. Nel caso in cui un centroide non sia attivo, la sua distanza viene impostata al valore massimo (511), che è comunque maggiore della massima distanza possibile tra un centroide ed un punto di riferimento (se, ipoteticamente, un centroide è posizionato in (0,0) e il punto di riferimento in (255,255), la loro distanza sarà pari a 510).

Il risultato di questa scelta è sia un risparmio non indifferente di area e pesantezza sintattica dato dall'eliminazione dei cicli if/elsif, sia una maggiore prevedibilità per quanto riguarda il tempo di esecuzione del processo. Infatti, è sempre eseguito lo stesso numero di stati, e ne risulta che le durate delle varie computazioni siano tutte simili tra loro.

Un'altra importante scelta di design è stata quella di dividere le valutazioni su ogni centroide in quattro stati diversi: come spiegato in precedenza, perché i dati letti dalla memoria siano disponibili al componente c'è bisogno di aspettare un ciclo di clock, e la stessa cosa vale per quanto riguarda l'utilizzo di eventuali risultati di computazione all'interno degli stati. Di conseguenza, snodare le operazioni in diversi stati aumenta la leggibilità e la semplicità del componente.

Infine, è rilevante spendere alcune parole sull'approccio alla computazione del risultato. Come già scritto precedentemente, si è dedicato un intero stato al calcolo della distanza minima, ed un altro alla determinazione del risultato. Anche in questo caso, c'era la possibilità di aggiornare i segnali durante la computazione degli altri stati, ma si è preferita una soluzione più semplice ed efficace.