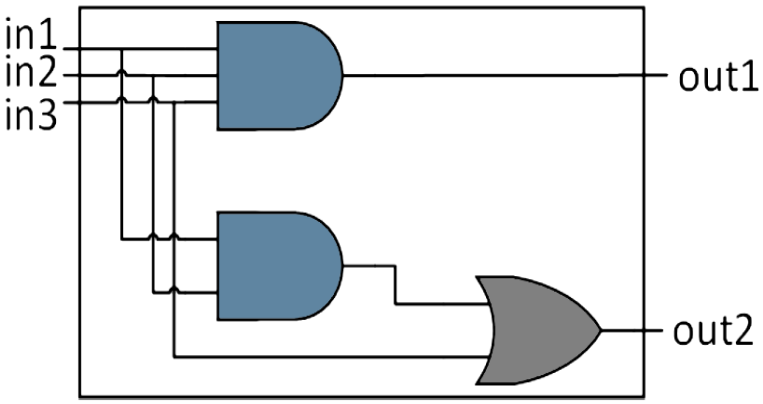
POLITECNICO
MILANO 1863



PROVA FINALE DI RETI LOGICHE [085877]

Studente: Francesco Paterna
Matricola: 843367
Codice Persona: 10520319

1.Introduzione

La seguente documentazione descriverà in maniera dettagliata, il lavoro svolto per la risoluzione della prova finale di Reti Logiche relativa all'anno accademico 2018/2019.
Partirò introducendo e commentando le specifiche date; in seguito mostrerò, partendo dai primi ragionamenti su cui è stato costruito il codice, la relativa implementazione e concluderò analizzando i test di verifica effettuati.
Dedicherò inoltre uno spazio per le relative ottimizzazioni.

2. Specifica

Descrizione generale
Sia dato uno spazio bidimensionale definito in termini di dimensione orizzontale e verticale, e siano date le posizioni di N punti, detti "centroidi", appartenenti a tale spazio. Si vuole implementare un componente HW descritto in VHDL che, una volta fornite le coordinate di un punto appartenente a tale spazio, sia in grado di valutare a quale/i dei centroidi risulti più vicino (Manhattan distance).
Lo spazio in questione è un quadrato (256x256) e le coordinate nello spazio dei centroidi e del punto da valutare sono memorizzati in una memoria (la cui implementazione non è parte del progetto). La vicinanza al centroide viene espressa tramite una maschera di bit (maschera di uscita) dove ogni suo bit corrisponde ad un centroide: il bit viene posto a 1 se il centroide è il più vicino al punto fornito, 0 negli altri casi. Nel caso il punto considerato risulti equidistante da 2 (o più) centroidi, i bit della maschera d'uscita relativi a tali centroidi saranno tutti impostati ad 1.
Degli N centroidi K<=N sono quelli su cui calcolare la distanza dal punto dato. I K centroidi sono indicati da una maschera di ingresso a N bit: il bit a 1 indica che il centroide è valido (punto dal quale calcolare la distanza) mentre il bit a 0 indica che il centroide non deve essere esaminato. Si noti che la maschera di uscita è sempre a N bit e che i bit a 1 saranno non più di K.

Dati
I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0.

- L'indirizzo 0 è usata per memorizzare il numero di centroidi (Maschera di ingresso: definisce quale centroide deve essere esaminato);
- Gli indirizzi dal 1 al 16 sono usati per memorizzare le coordinate a coppie X e Y dei centroidi:
    - 1 - Coordinata X 1^{o} Centroide
    - 2 - Coordinata Y 1^{o} Centroide
    - 3 - Coordinata X 2^{o} Centroide
    - 4 - Coordinata Y 2^{o} Centroide
    - …
    - 15 - Coordinata X 8^{o} Centroide
    - 16 - Coordinata Y 8^{o} Centroide
- Gli indirizzi 17 e 18 sono usati per memorizzare le Coordinate X e Y del punto da valutare
- L'indirizzo 19 è usato per scrivere, alla fine, il valore della maschera di uscita

Note ulteriori sulla specifica

1. I centroidi nella maschera vengono elencati dal bit meno significativo -posizione più a destra - (Centroide 1) al più significativo (Centroide 8);
2. Il valore della maschera risultante dall'identificazione del/dei centroide/i più vicino/i segue la stessa regola vista al punto precedente. Il bit meno significativo rappresenta il Centroide 1 mentre quello più significativo il Centroide 8;
3. Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero.

ESEMPIO:

La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I dati dall'indirizzo 0 a 18 sono ingressi per l'elaborazione mentre il dato in posizione 19 è il risultato dell'elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

Indirizzo Memoria	Valore	Commento
0	185	/* Maschera 10111001
1	75	// X centroide 1 (da considerare vedi Maschera)
2	32	// Y centroide 1 (da considerare vedi Maschera)
3	111	// X centroide 2 (da NON considerare vedi Maschera)
4	213	// Y centroide 2 (da NON considerare vedi Maschera)
5	79	// X centroide 3 (da NON considerare vedi Maschera)
6	33	// Y centroide 3 (da NON considerare vedi Maschera)
7	1	// X centroide 4 (da considerare vedi Maschera)
8	33	// Y centroide 4 (da considerare vedi Maschera)
9	80	// X centroide 5 (da considerare vedi Maschera)
10	35	// Y centroide 5 (da considerare vedi Maschera)
11	12	// X centroide 6 (da considerare vedi Maschera)
12	254	// Y centroide 6 (da considerare vedi Maschera)
13	215	// X centroide 7 (da NON considerare vedi Maschera)
14	78	// Y centroide 7 (da NON considerare vedi Maschera)
15	211	// X centroide 8 (da considerare vedi Maschera)
16	121	// Y centroide 8 (da considerare vedi Maschera)
17	78	// X del punto da valutare
18	33	// Y del punto da valutare
19	17	// Maschera di uscita 00010001 (OUTPUT)

Interfaccia del Componente

Il componente da descrivere deve avere la seguente interfaccia.

```
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;
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_start è il segnale di START generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

APPENDICE: Descrizione Memoria
NOTA: **La memoria è già istanziata all'interno del Test Bench e non va sintetizzata**

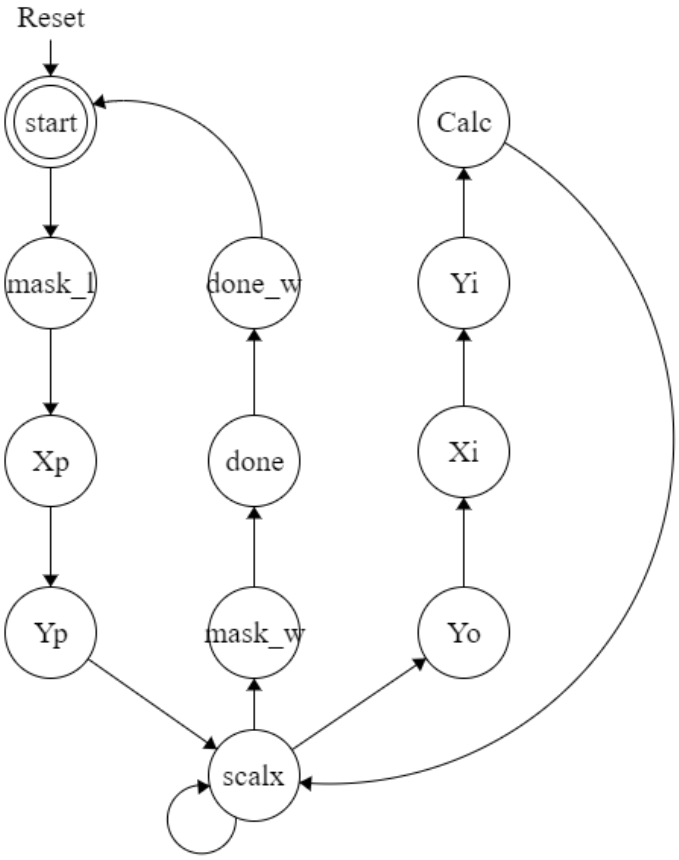
La memoria e il suo protocollo può essere estratto dalla seguente descrizione VHDL che fa parte del test bench e che è derivata dalla User guide di VIVADO disponibile al seguente link:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf

```
-- Single-Port Block RAM Write-First Mode (recommended template)

-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di;
        else
          do <= RAM(conv_integer(addr));
        end if;
      end if;
    end if;
  end process;
end syn;
```

3. Analisi della Specifica

Dopo una prima analisi, possiamo osservare come la specifica da implementare sia comparabile al del gioco delle bocce; dove il pallino è rappresentato dal punto da valutare, le bocce sono rappresentate dai centroidi, e la maschera d'ingresso rappresenta il numero delle bocce che vogliamo utilizzare (da 0 fino ad un massimo di 8).
Data la presenza di un segnale di reset, e la natura molto sequenziale del gioco, siamo orientati come approccio verso una macchina a stati finiti.

Gli stati utilizzati sono 13+1[1] e sono elencati nel seguente schema:
(per facilitare la leggibilità, alcuni stati presentano un nome diverso nel progetto; i nomi originali sono stati trascritti in blu)



Start: il componente è pronto e attende il segnale i_start = '1'

Mask_l: carica la maschera d'ingresso [mask_loader]

Xp: Carica coordinata x del punto da valutare [x_point_loader]

Yp: Carica coordinata y del punto da valutare [y_point_loader]

Scalx: Seleziona un centroide valido e ne carica la variabile x

Yo: Carica coordinata y del centroide da valutare [Load_y]

Xi: Calcola |Xp – Xo| [Calc_Xi]

Yi: Calcola |Yp - Yo| [Calc Yi]

Calc: Calcola la distanza Xi + Yi e verifica se è la più piccola distanza manhattan registrata, in caso positivo, aggiorna la maschera d'uscita [Calc_Man]

Mask_w: Scrive la maschera d'uscita [Mask_Writer]

Done: Porta ad '1' il segnale di o_done e porta i segnali di lettura e scrittura memoria a '0'

Done_w: Attende che il segnale i_start scenda a '0' per riportarci alo stato di start

[1] Vedremo in seguito che, per consentire alla memoria di inviare i dati, abbiamo elaborato un delay. Tutti i passaggi verso stati che necessitano di lettura a memoria, vengono preceduti da uno stato denominato WAIT_MEM

Tale approccio presenta tre problematiche da risolvere:

- Come selezionare in maniera efficiente i centroidi appartenenti alla maschera d'ingresso
- Come gestire il segnale di reset
- Come gestire i tempi della memoria

1) Per la selezione efficiente dei centroidi, sfrutteremo un vettore K settato di default a "00000001" Ad ogni giro effettueremo un AND bit a bit, tra K e la Maschera di Input [MI]; se il risultato sarà pari a K, il componente procederà a caricare le coordinate di quel centroide, altrimenti passerà direttamente a verificare il centroide successivo.
Per verificare il centroide successivo, effettuerò uno Shift a sinistra della maschera K, e verrà rieseguito l'AND bit a bit. Questo ciclo si concluderà quando lo Shift a sinistra esaurirà il vettore K portandolo a "0000000". Appena il ciclo si conclude viene lanciata la funzione di scrittura della maschera, contenuta nello stato Mask_Writer.
Questo processo viene effettuato dallo stato **SCALX** della fsm. (**S**elect **C**entroid **A**nd **L**oad **X**)

Primo Step

0	0	1	1	1	1	0	1	Maschera Input **[MI]**
+	+	+	+	+	+	+	+	**AND**
0	0	0	0	0	0	0	1	Vettore **[K]**
=	=	=	=	=	=	=	=	
0	0	0	0	0	0	0	1	**CETROIDE VALIDO**

(Eseguo shift a sinistra del vettore K)

Secondo Step

0	0	1	1	1	1	0	1	Maschera Input **[MI]**
+	+	+	+	+	+	+	+	**AND**
0	0	0	0	0	0	1	0	Vettore **[K]** shiftato
=	=	=	=	=	=	=	=	
0	0	0	0	0	0	0	0	**CENTROIDE NON VALIDO**

….

2) Per la gestione del segnale di reset mi ricollego alla specifica del progetto:

“i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;”

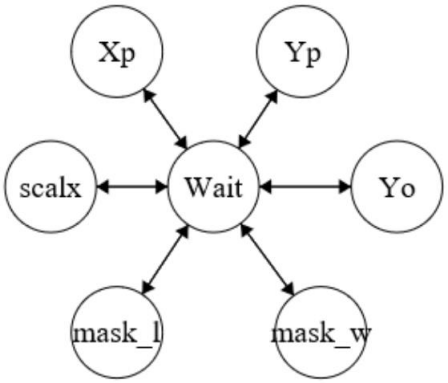
Poiché il segnale inizializza la macchina, occorre resettare tutte le variabili globali ed i segnali di output ai valori di default ed in seguito riportare la macchina a stati finiti sullo stato “START”, ogni volta che il segnale di i_rst viene riportato ad ‘1’.

```
if(i_rst = '1') then
    o_en <=  ‘0’;                    Riporta a 0 il segnale di lettura a memoria
    o_we <=  ’0’;                    Riporta a 0 il segnale di lettura a memoria
    o_done <=  ‘0’;                  Riporta a 0 il segnale di fine elaborazione
    MO := "00000000";                Valore Maschera d'Uscita
    Dmin := 510;                     Distanza Manhattan minima (255+255)
    k := "00000001";                 Selezionatore di Centroidi
    address := x"0001";              Puntatore sulla coordinata X del primo centroide
    p_state <= START;                Riporta la macchina allo stato iniziale
    state <= START;                  Riporta la macchina allo stato inizile
end if;
```

3) Per consentire alla memoria il tempo di inviare i dati, abbiamo elaborato uno stato di delay denominato WAIT_MEM

WAIT_MEM intermedierà tutti gli stati che necessitano di ricevere dati dalla memoria, nell’ordine:

- Mask_Loader — Caricamento Maschera di Input
- X_Point_Loader — Caricamento Coordinata X Punto da Valutare
- Y_Point_Loader — Caricamento Coordinata Y Punto da Valutare
- ScalX — Caricamento Coordinata X Centroide
- Load_Y — Caricamento Coordinata Y Centroide
- Mask_Writer — Caricamento Maschera di Output



4. TestBenches

Per verificare il corretto funzionamento del nostro componente utilizzeremo delle testbenches.
Effettueremo prima delle simulazoni post-sintesi, osservando attentamente il Waveform Viewer, al fine di accertarci che tutti i segnali ricevuti e trasmessi corrispondano a quanto richiesto dalla specifica;
successivamente ripeteremo il processo in modalità behavioral, per avere una visione chiara anche del comportamento delle variabili, sfruttando dei report[2] stampati sulla tcl console di Vivado.
Oltre al test ufficiale, fornitoci dal docente, ho creato altri test[3] al fine di garantire il pieno funzionamento del componente.
Tutte le mie testbenches sono varianti del Testbench ufficiale, allegato alla pagina successiva.

Prima di analizzare i risultati, lascio un elenco di tutte le testbench alternative utilizzate:

TESTBENCH ALTERNATIVE GENERICHE

- Bestcorner Punto da valutare in posizione (128,128), 4 Centroidi ai 4 angoli della mappa
- Busy Gli 8 centroidi ed il punto da valutare si trovano tutti in posizione (42,42)
- Surrounded Un punto da valutare circondato da 8 centroidi

TEST BENCH SPECIALIZZATE

- Reset Testiamo il comportamento con un segnale di reset durante l'esecuzione
- Leonida Testiamo che la funzione modulo funzioni corettamente, evitando gli overflow
- MaxDist Ci accertiamo che il componente regga la massima distanza manhattan
- Tentation Ci accertiamo che la funzione SCALX selezioni correttamente i centroidi
- Empty_Mask Testiamo il componente con una maschera vuota
- InTheCorner Verificamo che le posizioni contententi uno 0, non causino errori di calcolo

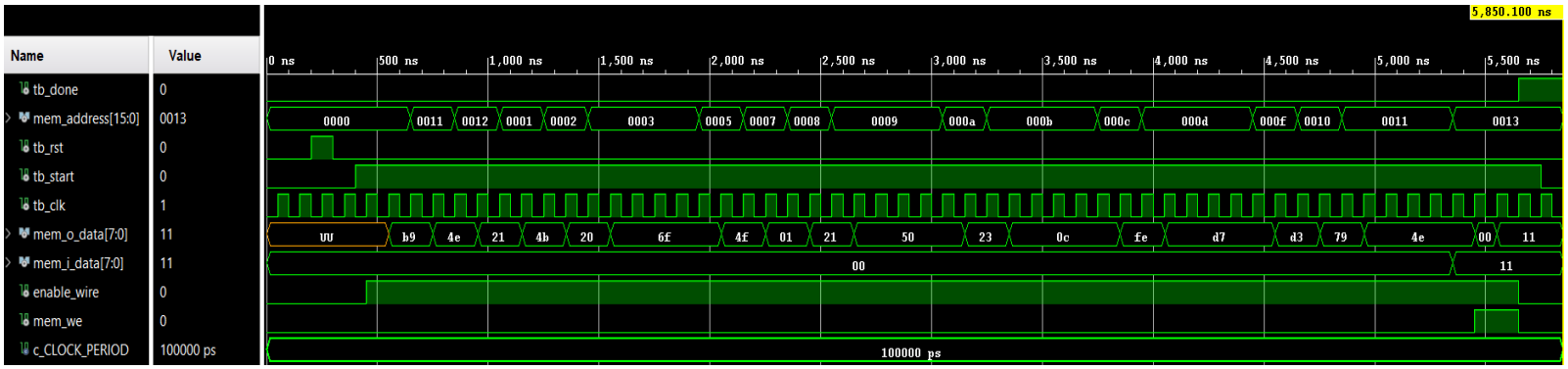
[2] I report statement sono stati disattivati nella versione definitiva rendendoli dei commenti, nel caso vogliate riattivarli occorre rimuovere i due trattini --
(https://en.wikibooks.org/wiki/Programmable_Logic/VHDL_General_Syntax#Comments)

[3] Tutte le mie testbenches alternative ed il progetto saranno resi disponibili sul mio profilo Github al termine del periodo di consegna del progetto, al fine di evitare plagi.(https://github.com/FrancescoPaterna)

Testbench Ufficiale

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
--
--

entity project_tb is
end project_tb;

architecture projecttb of project_tb is
constant c_CLOCK_PERIOD  : time := 100 ns;
signal   tb_done                  : std_logic;
signal   mem_address: std_logic_vector (15 downto 0) := (others => '0');
signal   tb_rst                   : std_logic := '0';
signal   tb_start                 : std_logic := '0';
signal   tb_clk                   : std_logic := '0';
signal   mem_o_data,mem_i_data          : std_logic_vector (7 downto 0);
signal   enable_wire                    : std_logic;
signal   mem_we                         : std_logic;

type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);

-- come da esempio su specifica
signal RAM: ram_type := (
0 => std_logic_vector(to_unsigned( 185 , 8)), --maschera input
1 => std_logic_vector(to_unsigned( 75 , 8)),
2 => std_logic_vector(to_unsigned( 32 , 8)),
3 => std_logic_vector(to_unsigned( 111 , 8)),    --coordinate centroidi
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)), -- Coordinate punto
18 => std_logic_vector(to_unsigned( 33 , 8)), -- Da Valutare
others => (others =>'0'));

component 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 component project_reti_logiche;

begin
UUT: project_reti_logiche
port map (
i_clk                => tb_clk,
i_start       => tb_start,
i_rst                => tb_rst,
i_data               => mem_o_data,
o_address            => mem_address,
o_done               => tb_done,
o_en                 => enable_wire,
o_we                 => mem_we,
o_data               => mem_i_data
);

p_CLK_GEN : process is
begin
wait for c_CLOCK_PERIOD/2;
tb_clk <= not tb_clk;
end process p_CLK_GEN;


MEM : process(tb_clk)
begin
if tb_clk'event and tb_clk = '1' then
if enable_wire = '1' then
if mem_we = '1' then
RAM(conv_integer(mem_address))  <= mem_i_data;
mem_o_data                      <= mem_i_data after 2 ns;
else
mem_o_data <= RAM(conv_integer(mem_address)) after 2 ns;
end if;
end if;
end if;
end process;


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 until tb_done = '1';
wait for c_CLOCK_PERIOD;
tb_start <= '0';
wait until tb_done = '0';

-- Maschera di output = 00010001
assert RAM(19) = std_logic_vector(to_unsigned( 17 , 8)) report "TEST
FALLITO" severity failure;

assert false report "Simulation Ended!, TEST PASSATO" severity
failure;
end process test;

end projecttb;
```

RISULTATI GENERATI DALLA TESTBENCH UFFICIALE



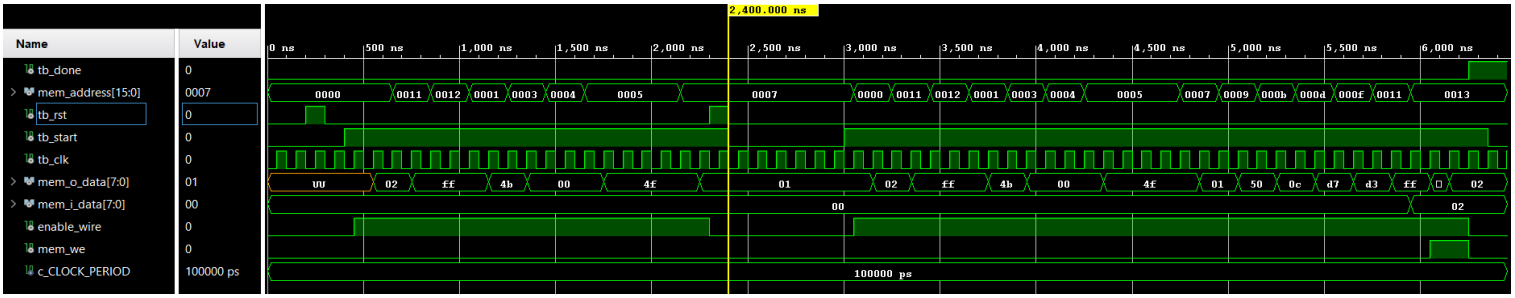
Notiamo dal Waveform Viewer come il componente abbia richiesto correttamente la maschera d'ingresso B9(HEX) = 185(DEC); le coordinate del punto da valutare 4E(HEX) = 78(DEC), 21(HEX) = 33(DEC) e le coordinate di ogni singolo centroide.
Notiamo inoltre come abbia correttamente inviato la maschera di output corretta, la 11(HEX) = 00010001(BIN).

Leggendo le note in modalità Behavioral osserviamo:

Note: --------RESET--------
Note: MASCHERA INPUT : 185
Note: COORDINATA XP : 78
Note: COORDINATA YP : 33
Note: COORDINATA XO : 75
Note: COORDINATA YO : 32
Note: XI CALCOLATO: 3
Note: YI CALCOLATO: 1
Note: DISTANZA MANHATTAN CALCOLATA: 4
Note: SCARTATO K : 2
Note: SCARTATO K : 4
Note: COORDINATA XO : 1
Note: COORDINATA YO : 33
Note: XI CALCOLATO: 77
Note: YI CALCOLATO: 0
Note: DISTANZA MANHATTAN CALCOLATA: 77
Note: COORDINATA XO : 80
Note: COORDINATA YO : 35
Note: XI CALCOLATO: 2
Note: YI CALCOLATO: 2
Note: DISTANZA MANHATTAN CALCOLATA: 4
Note: COORDINATA XO : 12
Note: COORDINATA YO : 254
Note: XI CALCOLATO: 66
Note: YI CALCOLATO: 221
Note: DISTANZA MANHATTAN CALCOLATA: 287
Note: SCARTATO K : 64
Note: COORDINATA XO : 211
Note: COORDINATA YO : 121
Note: XI CALCOLATO: 133
Note: YI CALCOLATO: 88
Note: DISTANZA MANHATTAN CALCOLATA: 221
Note: MASCHERA OUTPUT : 17
Failure: Simulation Ended!, TEST PASSATO
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:05 . Memory (MB): peak = 2505.027 ; gain = 0.000

Notiamo come il componente abbia scartato i centroidi giusti, 2(DEC) = 00000010(BIN), 4(DEC) = 00000100(BIN) e 64(DEC) = 01000000(BIN); ed abbia selezionato correttamente il primo ed il quinto vettore, avendo entrambi la distanza Manhattan minima dal centroide.
Il componente funziona alla perfezione, ad ha eseguito tutto il processo in soli 58 cicli di clock.

RISULTATI TESTBENCH ALTERNATIVA “RESET”



Il test alternativo di maggiore importanza, è quello relativo all’arrivo di un segnale di Reset durante l’esecuzione.
Ho elaborato la Testbench in modo da lanciare un segnale di reset dopo 23 cicli di clock.

```
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 1800 ns;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    tb_start <= '0';
    wait for 600 ns;
    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';
```

La memoria della nostra testbench contiene i seguenti dati:

Maschera d’ingresso: 2(DEC) = 00000010(BIN)
Coordinate del punto da valutare: 255(DEC) , 255(DEC)
Coordinate unico centroide valido: 0(DEC), 0(DEC)

Notiamo come in presenza del segnale i_rst = ‘1’, a 2300ns, la macchina venga resettata correttamente, rimettendosi in attesa del segnale di start. All’arrivo del nuovo segnale di start, riprende da zero, e porta a termine il test senza problemi; infatti possiamo osservare come la maschera d’uscita elaborata sia 2(HEX) = 00000010(BIN), l’unico centroide valido.

Risparmio al lettore l’analisi delle altre Testbenches, concludendo questo capitolo con l’analisi di “RESET”, che dal punto di vista delle funzionalità, risulta quella più interessante.

5.Ottimizzazioni

Sono possibili molteplici ottimizzazioni per questo progetto:

1. **Recupero di alcuni cicli di clock dovuti al passaggio di stato WAIT_MEM**

Analizzando il Waveform Viewer osserviamo come posso esserci dei disallineamenti dei cicli di clock durante i passaggi di stato tra WAIT_MEM e SCALX, questo si traduce con un possibile ritardo di 2 cicli di clock in attesa dei fronti di salita.
Tuttavia, poiché il componente termina l'elaborazione con una velocità molto minore dei 100 cicli di clock, non è una perdita eccessiva

2. **Portare a 0 il segnale o_en per centroide non appartenete alla maschera d'ingresso**

Si potrebbe impedire alla memoria inviare dati ad un blocco di memoria probabilmente vuoto, o comunque contenente un'informazione futile all'elaborazione;
tuttavia ho preferito mantenere un segnale o_en sempre alto a meno di un segnale di reset (anche perché non vi era alcuna limitazione nella specifica richiesta).
In tal caso basterebbe portare il segnale o_en a '0' al termine della funzione dello stato LOAD_Y, e riportarlo ad '1' nello stato SCALX, solamente nel caso in cui il centroide appartenga alla maschera d'ingresso.

3. **Elaborare una funzione di controllo della maschera d'ingresso nello stato SCALX, in modo da anticipare il passaggio allo stato MEM_WRITER**

Si potrebbe implementare un controllo della maschera d'ingresso nella funzione scalx.
Ogni volta che un centroide valido viene analizzato, si dovrebbe uno XOR tra la Maschera d'ingresso ed il vettore K prima di essere shiftato a sinistra.
Nel caso in cui MI risultasse pari a 00000000, si potrebbe passare direttamente alla stato MASK_WRITER prima di portare il vettore K a 00000000;
Tuttavia poiché il nostro componente termina la propria esecuzione entro i 100 cicli di clock, ho preferito non implementare tale funzione.