COMPONENTI DEL GRUPPO:

Francesco Pontiggia	843986	10524642
Arianna Ricci	846769	10536266

Progetto di reti logiche AA 17/18

Francesco Pontiggia 843986

Arianna Ricci 846769

Sommario

Presentazione del progetto e dell'algoritmo.

Il progetto consiste in un unico componente strutturato come macchina a stati con reset sincrono. La macchina a stati, ricevuto il segnale di start, legge dalla memoria il numero di righe e di colonne dell'immagine e il valore soglia. In seguito, scorre tutte le celle dell'immagine in ordine sequenziale, riga per riga, confrontando il valore contenuto con il valore soglia, mantenendo in alcuni segnali interni al componente i valori della prima e ultima riga contenenti un valore soprasoglia, e della prima e ultima colonna contenenti un valora soprasoglia. Dopo aver scorso l'immagine, calcola la lunghezza dei lati del rettangolo contenente l'immagine come differenza dei valori dell'ultima e prima colonna (e riga), e poi fa il prodotto tra le due lunghezze. Scrive in memoria quel valore (agli indirizzi 0 e 1) e poi si rimette nello stato iniziale in attesa di un nuovo segnale di start.

Il componente evita di scorrere tutta la griglia solo se essa ha 0 righe o 0 colonne (vale a dire, se non ha celle, quindi l'output è necessariamente 0). Non valuta invece se il valore

soglia è pari a 0 (vale a dire, se tutte le celle della griglia appartengono all'immagine, quindi l'output è il numero di righe peri l numero di colonne). Tutto ciò si può riscontrare chiaramente nei tempi dei test.

Il componente viene sintetizzato e implementato correttamente.

NB: alcuni dei segnali seguenti (*next_num_colonne_tmp, next_num_righe_tmp, next_indirizzo*) e degli stati seguenti (per esempio *S5_soprasoglia_buffer, S5_aggiornatore_buffer, S9*) sono stati introdotti con lo scopo di ridurre il più possibile il clock minimo in post implementation timing simulation(e di passare i testbench stessi in post-implementation): si potrebbero eliminare, a patto di alzare il clock minimo.

Segnali

La macchina a stati si avvale dei seguenti segnali interni:

- **Current_state**, **next_state**: stato presente e stato prossimo;
- **Indirizzo**, **next_indirizzo**: indirizzo della memoria a cui la macchina vuole leggere il valore di una cella, e indirizzo immediatamente successivo (16 bit, inizializzati rispettivamente a 4 e 5: infatti la macchina inizia a leggere dall'indirizzo 5);
- **Num_colonne**, **num_righe**: numero di colonne e righe dell'immagine, vengono lette dalla memoria, indirizzi 2 e 3(8 bit);
- **Prodotto**: area del rettangolo minimo, valore da scrivere in memoria alla fine della computazione (16 bit);
- **Diff_righe**, **diff_colonne**: lunghezza dei due lati del rettagolo, differenza tra il numero dell'ultima riga (colonna) e prima riga (colonna) contenente un valore soprasoglia (8 bit);
- **Num_colonne_tmp, num_righe_tmp**: contatori della colonna e della riga a cui la macchina si trova ad ogni passo dello scorrimento delle celle dell'immagine (8 bit, inizializzati entrambi a 1);
- **Next_num_colonne_tmp**, **next_num_righe_tmp**: valori successivi dei contatori num_colonne_tmp, num_righe_tmp (8 bit, inizializzati entrambi a 2);
- **Max_riga, min_riga, max_colonna, min_colonna**: numero dell'ultima riga(colonna) e della prima riga (colonna) contenenti un valore soprasoglia; la loro differenza (sommato 1) dà la lunghezza dei lati del rettangolo (8 bit);
- **Valore_soglia**: il valore di soglia per determinare quali celle appartengono all'immagine e quali appartengono allo sfondo, viene letto dall'indirizzo 4 di memoria (8 bit);
- **Uno**: tiene memoria se la macchina ha già trovato nella computazione un valore soprasoglia oppure no; è necessario per la correttezza dell'algoritmo (infatti quando la macchina trova il primo valore soprasoglia inizializza max_riga, min_riga, max_colonna, min_colonna, mentre quando trova i successivi valori soprasoglia aggiorna questi stessi segnali confrontandoli con i contatori della posizione attuale a cui si trova la macchina nello scorrimento delle celle della griglia dell'immagine(1 bit, inizializzato a 0);

COMPONENTI DEL GRUPPO:

Francesco Pontiggia	843986	10524642
Arianna Ricci	846769	10536266

Stati

Gli stati della macchina (e le relative azioni eseguite) sono i seguenti:

1. **S0**: stato iniziale, aspetta il segnale di start dalla memoria, altrimenti resta in loop su sé stesso;
2. **S1**: ha appena ricevuto il segnale di start, manda alla memoria la richiesta di leggere il valore contenuto all'indirizzo 2
3. **S2**: legge il numero di colonne, cioè il valore contenuto all'indirizzo 2, e lo salva in *num_colonne*; chiede di leggere il valore all'indirizzo 3;
4. **S3**: legge il numero di righe, cioè il valore contenuto all'indirizzo 3, e lo salva in *num_righe*; chiede di leggere il valore all'indirizzo 4;
5. **S3a**: valuta se il numero di righe è nullo: se sì, va direttamente al calcolo dell'area, se no, prosegue;
6. **S3b**: valuta se il numero di colonne è nullo: se sì, va direttamente al calcolo dell'area, se no, prosegue;
7. **S4**: legge il valore di soglia e lo salva in *valore_soglia*; inizia l'iterazione su tutte le celle dell'immagine chiedendo il valore contenuto nella prima cella dell'immagine.
8. **S5**: aggiorna i segnali indirizzo e *next_indirizzo*;
9. **S5_reader**: chiede alla memoria di leggere il valore contenuto nella cella all'indirizzo del segnale *indirizzo*;
10. **S5_confrontatore**: confronta il valore contenuto nella cella con il valore soglia; se superiore, allora passa in S5_soprasoglia, se no passa semplicemente in S5_aggiornatore, cioè aggiorna soltanto i contatori della posizione corrente nella griglia dell'immagine;
11. **S5_soprasoglia**: aggiorna, se necessario, i valori dei segnali *max_riga, min_riga, max_colonna, min_colonna*; in particolare si avvale del segnale *uno* (si veda descrizione del segnale *uno*);
12. **S5_soprasoglia_buffer**: stato buffer;
13. **S5_aggiornatore**: stato per aggiornare i contatori; se la macchina si trova alla fine di una riga della griglia dell'immagine, allora va in S5_aggiornatore1, altrimenti in S5_aggiornatore_colonne;
14. **S5_aggiornatore1**: valuta se la macchina si trova alla fine della griglia (si ricordi che si può avere questo stato solo se la macchina è alla fine di una riga, quindi basta valutare soltanto se la macchina sta scorrendo l'ultima riga); se sì, va in S6, se no va in S5_aggiornatore_righe;
15. **S5_aggiornatore_righe**: aggiorna i contatori delle righe *num_righe_tmp, next_num_righe_tmp* aumentando di 1 il loro valore e portando ad 1 e 2 i valori dei contatori delle colonne *num_colonne_tmp, next_num_colonne_tmp*;
16. **S5_aggiornatore_colonne**: aggiorna i contatori delle colonne *num_colonne_tmp, next_num_colonne_tmp* aumentando di 1 il suo valore;
17. **S5_aggiornatore_buffer**: stato buffer;
18. **S5_restart**: fa ripartire il ciclo tornando in S5;
19. **S6**: calcola *diff_righe* e *diff_colonne*; se però *uno* è uguale a 0, allora significa che non c'è nessuna cella con valore soprasoglia, quindi imposta a zero entrambi i segnali.
20. **S7**: calcola il valore di prodotto;

21. **S7_buffer**: stato buffer (per eseguire la moltiplicazione di prodotto servono due cicli di clock);
22. **S8**: scrive gli 8 bit più significativi del segnale *prodotto* all'indirizzo 0 della memoria;
23. **S8a**: scrive gli 8 bit meno significativi del segnale *prodotto* all'indirizzo 1 della memoria;
24. **Stato_finale**: porta a 1 *o_done*, per comunicare alla memoria che la computazione è terminata, secondo quanto indicato nelle specifiche;
25. **S9**: riporta la macchina nello stato S0, pronta a ricevere di nuovo il segnale di start.

Testbench forniti e prestazioni

Il componente passa tutti i test forniti in presintesi, postsintesi (functional e timing) e post-implementation (functional e timing). Di seguito si indicano le prestazioni nei testbench forniti: in ogni cella della tabella si ha rispettivamente il clock minimo a cui il componente passa il dato test e il tempo complessivo necessario con quel clock (tra parentesi le unità di misura). I tempi di computazione sono presi a partire dal segnale di start.

	Behavioral	Functional post synthesis	Timing post synthesis	Functional post implementation	Timing post implementation
Testbench	2 (ps) / 4,773 (ns)	105 (ps)/248,241 (ns)	5(ns)/11932,5(ns)	105 (ps)/248,241 (ns)	5 (ns)/ 11935,4(ns)
Testbench_delay	1 (ns)/ 2386,5(ns)	5 (ns)/11932,5 (ns)	5(ns)/11932,5(ns)	5(ns)/11932,5(ns)	5(ns)/11937,5(ns)
Testbench2	2(ps)/6,117(ns)	105(ps)/318,129(ns)	5(ns)/15292,5(ns)	5(ns)/15292,5(ns)	5(ns)/14777,5(ns)
Testbench2_delay	1(ns)/3058,5 (ns)	5(ns)/15292,5(ns)	5(ns)/15292,5(ns)	5(ns)/15292,5(ns)	5(ns)/15097,5(ns)
Testbench3	2(ps)/5,125(ns)	105(ps)/266,545(ns)	5(ns)/12812,5(ns)	5(ns)/12812,5(ns)	5(ns)/12967,5(ns)
Testbench3_delay	1(ns)/2562,5(ns)	5(ns)/12812,5(ns)	5(ns)/12812,5(ns)	5(ns)/12812,5(ns)	9(ns)/23061,5(ns)
Testbench4	2(ps)/4,837(ns)	105(ps)/251,569(ns)	5(ns)/12092,5 (ns)	5(ns)/12092,5 (ns)	5(ns)/12097,5(ns)
Testbench4_delay	1(ns)/2418,5(ns)	5(ns)/12092,5(ns)	5(ns)/12092,5(ns)	5(ns)/12092,5(ns)	5(ns)/21765,5(ns)

Testbench aggiuntivi e prestazioni

Di seguito si indicano i test aggiuntivi a cui abbiamo sottoposto il nostro componente (per semplicità si indicano solo le parti di codice che differiscono dal testbench 3 fornito, preso come riferimento): il componente li passa tutti.

COMPONENTI DEL GRUPPO:

Francesco Pontiggia	843986	10524642
Arianna Ricci	846769	10536266

Nessuna riga

L'immagine ha 0 righe.

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

signal RAM: ram_type := (2 => "00011000", **3 =>"00000000",**4 => "00000010", 30 => "00000011", 31 => "00000011", 32 => "00000011", 33 => "00000011", 36 => "00000111", 37 => "00000111", 38 => "00000111", 39 => "00000111", 42 => "00001011", 43 => "00001011", 44 => "00001011", 45 => "00001011", 48 => "00001111", 54 => "00000011", 60 => "00000111", 66 => "00011111", 72 => "00011001", 78 => "00000011", 79 => "00000011", 80 => "00000011", 84 => "00000111", 85 => "00000111", 86 => "00000111", 90 => "00011111", 91 => "00011111", 92 => "00011111", 96 => "00011001", 102 => "00000011", 108 => "00000111", 114 => "00011111", 120 => "00011001", 126 => "00000011", 132 => "00000111", 133 => "00000111", 134 => "00000111", 135 => "00000111", 138 => "00001011", 139 => "00001011", 140 => "00001011", 141 => "00001011", 144 => "00001111", 145 => "00001111", 146 => "00001111", 147 => "00001111", others => (others =>'0'));

assert RAM(1) = "00000000" report "FAIL high bits" severity failure;

assert RAM(0) = "00000000" report "FAIL low bits" severity failure;

Nessuna colonna

L'immagine ha 0 colonne.

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

signal RAM: ram_type := (**2 => "00000000",** 3 =>"00000111",4 => "00000010", 30 => "00000011", 31 => "00000011", 32 => "00000011", 33 => "00000011", 36 => "00000111", 37 => "00000111", 38 => "00000111", 39 => "00000111", 42 => "00001011", 43 => "00001011", 44 => "00001011", 45 => "00001011", 48 => "00001111", 54 => "00000011", 60 => "00000111", 66 => "00011111", 72 => "00011001", 78 => "00000011", 79 => "00000011", 80 => "00000011", 84 => "00000111", 85 => "00000111", 86 => "00000111", 90 => "00011111", 91 => "00011111", 92 => "00011111", 96 => "00011001", 102 => "00000011", 108 => "00000111", 114 => "00011111", 120 => "00011001", 126 => "00000011", 132 => "00000111", 133 => "00000111", 134 => "00000111", 135 => "00000111", 138 => "00001011", 139 => "00001011", 140 => "00001011", 141 => "00001011", 144 => "00001111", 145 => "00001111", 146 => "00001111", 147 => "00001111", others => (others =>'0'));

assert RAM(1) = "00000000" report "FAIL high bits" severity failure;

assert RAM(0) = "00000000" report "FAIL low bits" severity failure;

Solo primo e ultimo

La griglia ha dimensione normale, solo la prima e l'ultima cella sono soprasoglia, l'immagine ha dimensione 168(24x7).

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

signal RAM: ram_type := (2 => "00011000", 3 =>"00000111",4 => "00000010", **5 => "00000011", 172 => "00000011", others => (others =>'0'));**

assert RAM (1) = "00000000" report "FAIL high bits" severity failure;

assert RAM (0) = "10101000" report "FAIL low bits" severity failure;

COMPONENTI DEL GRUPPO:

Francesco Pontiggia	843986	10524642
Arianna Ricci	846769	10536266

Solo 1

Solo un elemento appartiene all'immagine.

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

signal RAM: ram_type := (2 => "00011000", 3 =>"00000111",4 => "00000010", **80 => "00000011", others => (others =>'0'));**

assert RAM (1) = "00000000" report "FAIL high bits" severity failure;

assert RAM (0) = "00000001" report "FAIL low bits" severity failure;

Il massimo

La griglia ha dimensione 255x255, e tutte le celle sono soprasoglia.

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

signal RAM: ram_type:= (**2 => "11111111", 3 =>"11111111",4 => "00000010", others => (others =>'1'));**

assert RAM (1) = "11111110" report "FAIL high bits" severity failure;

assert RAM (0) = "00000001" report "FAIL low bits" severity failure;

Due cicli

Vengono eseguiti due cicli.

```
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;
tb_start <= '0';
wait until tb_done = '1';
wait until tb_done = '0';
wait until rising_edge(tb_clk);

assert RAM (1) = "00000000" report "FAIL high bits" severity failure;
assert RAM (0) = "01101110" report "FAIL low bits" severity failure;

wait for c_CLOCK_PERIOD;
tb_start <= '1';
wait for c_CLOCK_PERIOD;
tb_start <= '0';
wait until tb_done = '1';
wait until tb_done = '0';
wait until rising_edge(tb_clk);
```

```
assert RAM (1) = "00000000" report "FAIL high bits" severity failure;
assert RAM (0) = "01101110" report "FAIL low bits" severity failure;

assert false report "Simulation Ended! test passed" severity failure;
end process test;

end projecttb;
```

Riparti

L'esecuzione viene fermata e fatta ripartire.

```
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;
tb_start <= '0';

wait for 10*c_CLOCK_PERIOD;

--mid-computation reset signal: computation is supposed to start again
tb_rst <= '1';
wait for c_CLOCK_PERIOD;
tb_rst <= '0';
wait for c_CLOCK_PERIOD;
tb_start <= '1';
wait for c_CLOCK_PERIOD;
tb_start <= '0';
wait until tb_done = '1';
wait until tb_done = '0';
wait until rising_edge(tb_clk);

assert RAM (1) = "00000000" report "FAIL high bits" severity failure;
assert RAM (0) = "00010101" report "FAIL low bits" severity failure;


assert false report "Simulation Ended! test passed" severity failure;
end process test;

end projecttb;
```

Start prima di reset

Viene dato un segnale di start prima di reset: la computazione inizia solo dopo la corretta sequenza reset-start.

```
test: process is
begin
```

```
--
wait for 75 ns;
tb_start <= '1'; --Start 1
wait for c_CLOCK_PERIOD;
tb_start <= '0'; --Start 0
--
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;
tb_start <= '0';
wait until tb_done = '1';
wait until tb_done = '0';
wait until rising_edge(tb_clk);

assert RAM (1) = "00000000" report "FAIL high bits" severity failure;
assert RAM (0) = "01101110" report "FAIL low bits" severity failure;


assert false report "Simulation Ended! test passed" severity failure;
end process test;

end projecttb;
```

Infine, la tabella delle prestazioni relative ai test aggiuntivi (si usa la stessa notazione della tabella relativa ai test forniti)

	Behavioral	Functional post synthesis	Timing post synthesis	Functional post implementation	Timing post implementation
Testbench (nessuna_riga)	2(ps)/41(ps)	105(ps)/2,177(ns)	5(ns)/102,500(ns)	105(ps)/2,177(ns)	5(ns)/107,5(ns)
Testbench (nessuna_colonna)	2(ps)/45(ps)	105(ps)/2,385(ns)	5(ns)/112,500(ns)	105(ps)/2,385(ns)	5(ns)/117,5(ns)
Testbench (solo primo e ultimo)	2(ps)/4,789(ns)	105(ps)/249,073(ns)	5(ns)/11972,5(ns)	5(ns)/11972,5(ns)	9(ns)/21549,5(ns)
Tetsbench (solo 1)	2(ps)/4,781(ns)	105(ps)/248,657(ns)	5(ns)/11952,5(ns)	5(ns)/11952,5(ns)	5(ns)/11947,5(ns)
Testbench (il massimo)	2(ps)/2341,961(ns)	105(ps)/121782,017(ns)	5(ns)/5854902,5(ns)	5(ns)/5854902,5(ns)	5(ns)/5854917,5(ns)
Testibench (due cicli)	2(ps)/10,255(ns)	105(ps)/533,201(ns)	5(ns)/25632,5(ns)	5(ns)/25632,5(ns)	5(ns)/25937,5(ns)
Testbench (riparti)	2(ps)/4,863(ns)	105(ps)/252,921(ns)	5(ns)/12157,5(ns)	5(ns)/12157,5(ns)	9(ns)/21894,5(ns)
Testbench (start improprio)	2(ps)/105,133(ns)	105(ps)/366,949(ns)	5(ns)/12932,5(ns)	5(ns)/12932,5(ns)	5(ns)/13087,5(ns)