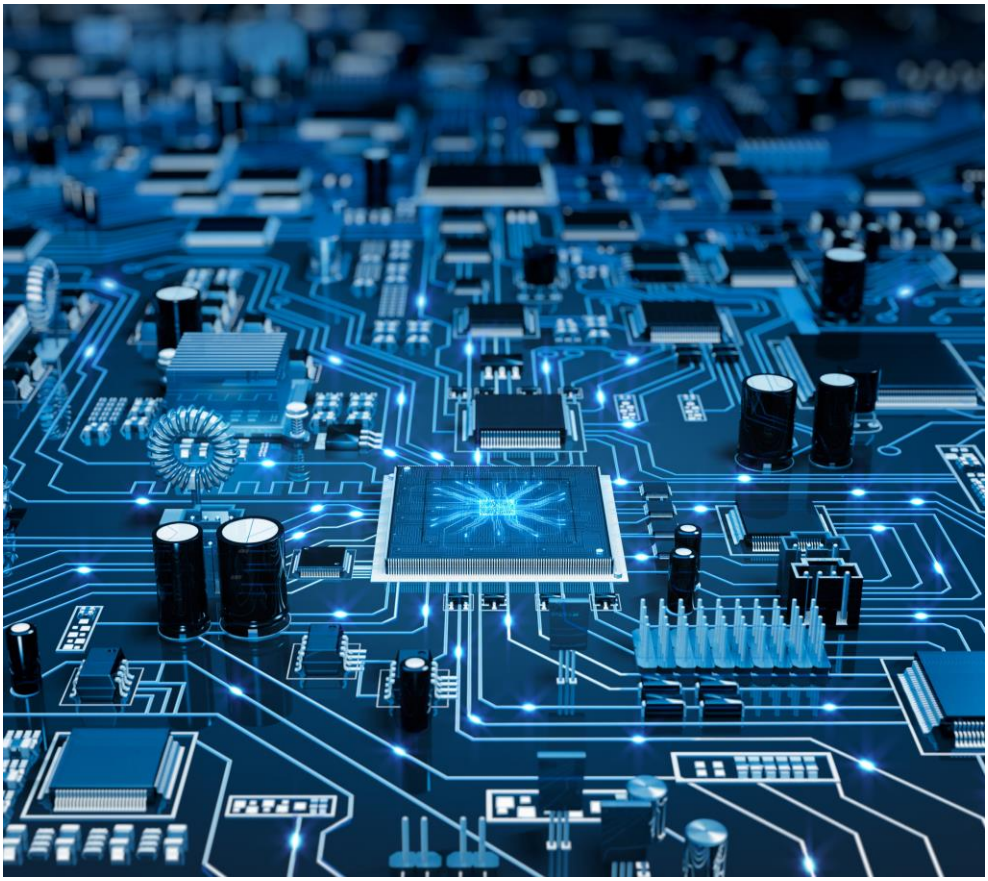
# PROVA ANALE

# PROGETTO RETI LOGICHE



Marco D'Antini
Codice persona 10603556

Reti Logiche
Anno 2020/2021
Professore: William Fornaciari

# Indice

# SPECIFICA PROGETTO

Il progetto è ispirato al metodo di equalizzazione dell'istogramma in un' immagine. Metodo utilizzato per ricalibrare il contrasto di un'immagine quando essa appare troppo omogenea nella distribuzione dei colori.

In questa versione è stato sviluppato un algoritmo in versione semplificata applicabile a immagini di dimensione massima 128x128 pixel e scala di colori da massimo 256 livelli.

L'algoritmo è così implementato:

- DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE
- SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE +1)))
- TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL
- NEW_PIXEL_VALUE = MIN( 255 , TEMP_PIXEL)

La memoria utilizzata per memorizzare l'immagine è così strutturata:

| Valore grandezza colonna | Valore grandezza riga | Input primo pixel | ... | Input ultimo pixel | Output primo pixel | ... | Output ultimo pixel |
|---|---|---|---|---|---|---|---|
| [0] | [1] | [2] | | | [2+(N_COL*N_RIG)] | | |

Figura 1: raffigurazione impiego memoria

Inoltre sono presenti dei segnali funzionali che implementano la funzione di avvio del processo (`i_start`), di fine elaborazione (`o_done`) e di reset del modulo che lo riporta allo stato iniziale dell'esecuzione (`i_rst`);

Il modulo può elaborare più immagini in successione ma solamente in modo sequenziale.

Il modulo parte con un segnale `START` a valore '1' e quando termina l'esecuzione il segnale `DONE` sarà '1', ciò dovrà far scendere `START` a '0'. L'esecuzione non potrà ripartire fin quando anche il segnale `DONE` assumerà nuovamente il valore '0'.

# SCELTE PROGETTUALI

Nella fase iniziale di progettazione la mia strategia è stata realizzare su carta e penna la macchina che descrive gli stati di elaborazione del circuito. Infatti l'approccio utilizzato è il Behavioural, con cui ho iniziato a riportare gli stati realizzati dalla carta al codice.

In maniera grossolana ho individuato 4 macrostati. Questi sono:

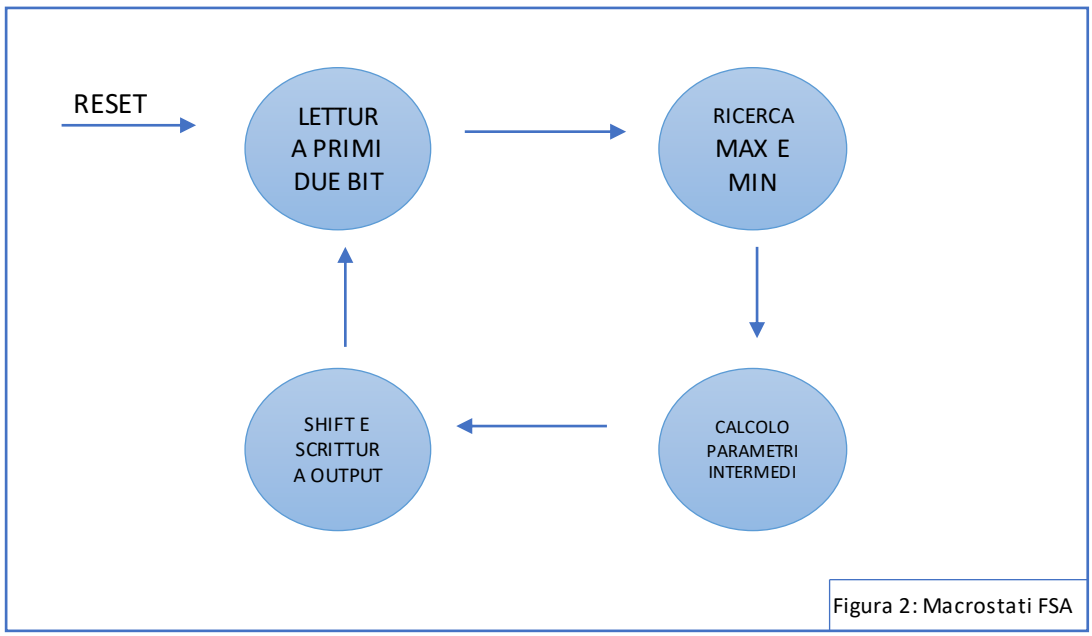


Figura 2: Macrostati FSA

- lettura dei primi due bit e calcolo della grandezza dell'immagine
- un primo ciclo che legge in bit indicati dalla dimensione e individua massimo e minimo valore dei pixel
- calcolo dei parametri del necessari al nuovo valore
- un secondo ciclo che legge nuovamente la i pixel dell'immagine e con il valore finale scrive anche in memoria

Di seguito riporto la raffigurazione degli stati in maniera più dettagliata.

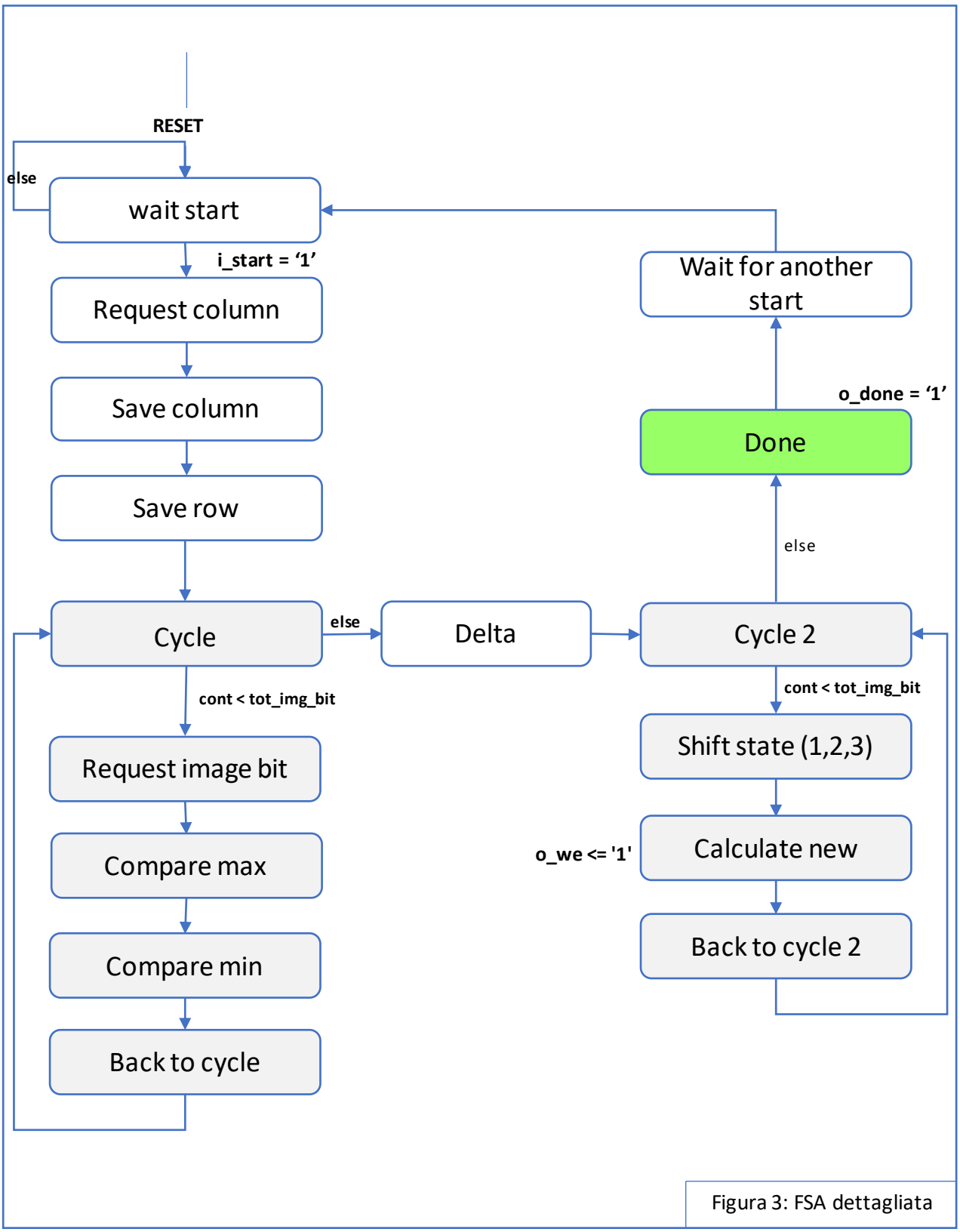



Figura 3: FSA dettagliata

Descrizione degli stati:

- **WAIT_START:** la macchina è in attesa di un segnale di start per partire. Quando il segnale `i_start = '1'` passa allo stato successivo altrimenti in autoanello torna su `WAIT_START`. A computazione finita la macchina torna in questo stato e assegno `o_done = '1'`;
- **REQ_COL:** richiedo dalla memoria il valore della colonna;
- **SAVE_COL:** salvo nel segnale `n_col` il valore richiesto allo stato precedente e richiedo il valore della riga;
- **SAVE_ROW:** salvo nel segnale `n_row` il valore di riga e calcolo il totale dei pixel dell'immagine;
- **CYCLE:** stato che contiene la condizione verso l'interno del ciclo (`cycle_1` per l'individuazione del massimo e minimo valore) e la fine del ciclo;
- **REQ_IMG_BIT:** internamente al primo ciclo richiedo il valore attuale del pixel;
- **COMP_MAX:** ricevo il valore richiesto allo stato precedente e comparo con la variabile temporanea `max` assegnata a 0 inizialmente;
- **COMP_MIN:** comparo con la variabile temporanea `min` assegnata a 255 inizialmente e incremento il contatore;
- **BACK_CYCLE:** stato ausiliario che salva il contatore e torna al ciclo `cycle_1`;
- **DELTA:** calcolo il `delta` e lo `shift_value` operando con un controllo a soglia per il calcolo del $\log 2$; quest'ultimo è implementato con un AND logico tra il vettore `bit_shift` e un vettore di 8 bit con un solo '1' in posizione diversa ad ogni confronto per ottenere un valore diverso;

```
Esempio: if (bit_shift AND "10000000") =  "10000000" then
              shift_level<= 1;
         else if (bit_shift AND "01000000") =  "01000000" then
              shift_level<= 2; …
```

Figura 4: estratto di codice confronto AND

- **CYCLE_2:** stato che contiene la condizione verso l'interno del ciclo (`cycle_2` per il calcolo e la scrittura del nuovo valore pixel) e la fine del ciclo che è anche la fine della computazione del programma;
- **SHIFT_STATE_1:** richiedo il valore del pixel della memoria;
- **SHIFT_STATE_2:** calcolo `temp - min`;
- **SHIFT_STATE_3:** opero lo shift a sinistra e salvo nella variabile `temp_pixel`;
- **CALC_NEW:** assegno ad `o_address` l'indirizzo di memoria su cui dovrà scrivere, cambio i bit `o_en ='1'` e `_we = '1'`, calcolo il valore del nuovo pixel e lo scrivo in memoria. Infine incremento il contatore;
- **BACK_CYCLE_2:** salva il valore del contatore e torna allo stato `cycle_2`;
- **DONE:** la computazione è finita allora `o_done = '1'`;
- **WAIT_FOR_ANOTHER_START**: stato ausiliario che mi riporta a **wait_for_start** se `i_start='0'` o rimane su se stesso in autoanello.

# RISULTATI SINTESI

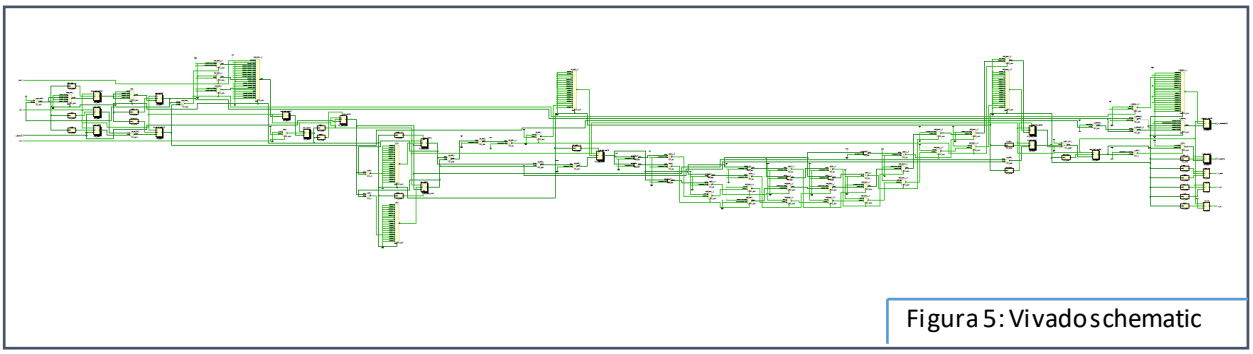


Figura 5: Vivado schematic

Come anticipato la sintesi del componente non produce errori e la simulazione Post Synthesis functional passa tutti i test elencati precedentemente.

Allego i risultati della sintesi riguardanti i componenti utilizzati ottenuti con il comando `report_utilization`:

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 204 | 0 | 134600 | 0.15 |
| LUT as Logic | 204 | 0 | 134600 | 0.15 |
| LUT as Memory | 0 | 0 | 46200 | 0.00 |
| Slice Registers | 157 | 0 | 269200 | 0.06 |
| Register as Flip Flop | 157 | 0 | 269200 | 0.06 |
| Register as Latch | 0 | 0 | 269200 | 0.00 |
| F7 Muxes | 0 | 0 | 67300 | 0.00 |
| F8 Muxes | 0 | 0 | 33650 | 0.00 |

Come desiderato il codice non inserisce dei Latch.

Figura 6: Comando report_utilization

Infine allego i risultati di utilizzo temporale ottenuti con il comando `report_timing:`

```
Slack:                    inf
  Source:                 n_row_reg[3]/C
                            (rising edge-triggered cell FDRE)
  Destination:            tot_img_bit_reg[12]/D
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        6.008ns  (logic 2.769ns (46.089%)  route 3.239ns (53.911%))
  Logic Levels:           8  (CARRY4=4 FDRE=1 LUT4=1 LUT6=2)
```

Figura 7: Comando report_timing

Da questo dato emerge che il circuito in media utilizza 6ns per commutare tutte le sue porte logiche, ciò vuol dire che per i restanti 4 ns i segnali restano stabili.
In base a quanto emerso posso concludere che la frequenza dei 100 MHz è stata rispettata e si potrebbe andare ad una frequenza maggiore.

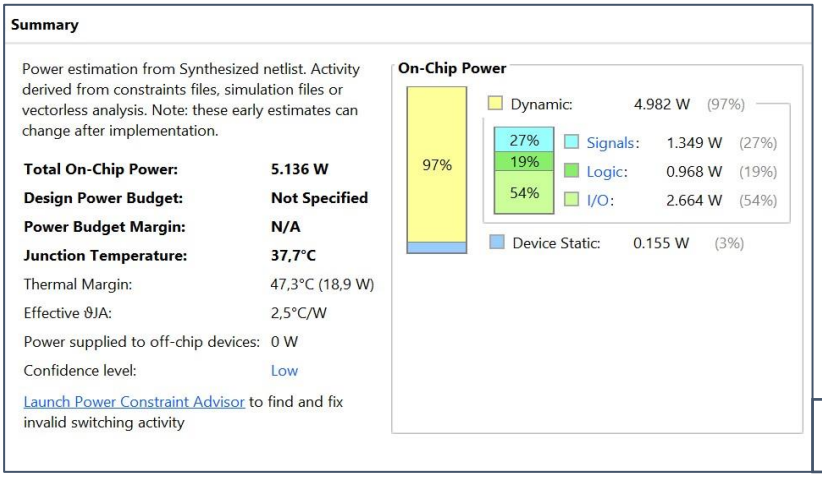



Figura 8: Consumo di energia

# TESTING

Come testbench iniziale ho utilizzato quello fornito dal professore su Beep. Una volta passato quello ho modificato lo stesso per coprire alcuni casi di interesse quali:

- un'immagine composta da tutti pixel con valore 0;
- una da tutti pixel con valore 255;
- una mista tra 255 e 0.

Successivamente ho utilizzato uno script in Python realizzato da un mio collega per generare casi di test che fornissero immagini di dimensione fino a 128x128.

Ho ottenuto la totalità dei test passati in una batteria di 15'000 casi di test in Behavoiral e post Synthesis functional.

Ho infine ripetuto i test, ma con un testbench che fornisse casi di reset, ottenendo nuovamente la totalità dei test passati.

```
s_read <= true;
wait for c_CLOCK_PERIOD;
s_read <= false;
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';
wait for c_CLOCK_PERIOD;
```

Figura 8: estratto di codice Testbench

# CONCLUSIONI

A fine del lavoro svolto valuto quest'esperienza molto positivamente.

Ho avuto l'occasione di sperimentare un approccio pratico a ciò che di norma rimane solo teorico.

Grazie all'utilizzo di un applicativo aziendale come Vivado e ad un linguaggio un po' diverso da quelli visti fino ad ora come VHDL ho avuto modo di conoscere una realtà professionale che rappresenta un'altra faccia dell'ingegneria informatica.

In particolare è stato interessante capire cosa c'è dietro al processo di ottimizzazione di un modulo, sia strutturale che temporale, ottenuto tramite il cambio di approccio e le modifiche al codice derivate dall'attività di debug dei segnali in linea temporale, molto diversa da quella vista in altri linguaggi di programmazione.