Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

Progetto di Reti Logiche

A.A. 2020/2021

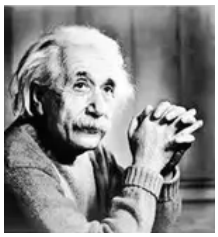
Leonardo Panseri
10664030

Febbraio 2021

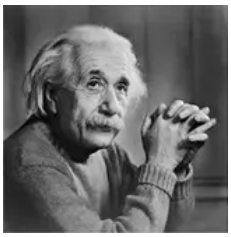
Indice

1 Introduzione

L'obiettivo del progetto è la realizzazione di un circuito digitale che sia in grado di applicare una versione semplificata del metodo di equalizzazione dell'istogramma di un'immagine. Questo metodo serve a ricalibrare il contrasto di un'immagine distribuendo su tutto l'intervallo di intensità i valori dei pixel che sono simili tra loro. La semplificazione è data dall'ipotesi di considerare solo immagini in scala di grigi a 256 livelli.



(a) Immagine non equalizzata



(b) Immagine equalizzata

Figura 1: Esempio di applicazione del metodo di equalizzazione dell'istogramma

L'immagine originale è memorizzata sequenzialmente in una RAM istanziata nel Test Bench, l'immagine con l'istogramma equalizzato deve essere memorizzata nella stessa RAM, a partire dall'indirizzo successivo a quello dell'ultimo byte dell'originale. Il componente deve compiere i seguenti passi:

1. Individuare i pixel con valore massimo MAX e minimo MIN
2. Calcolare il livello di shift necessario secondo la formula

$$SHIFT\_LVL = 8 - \lfloor \log_2(MAX - MIN + 1) \rfloor$$

3. Trasformare ogni pixel dell'originale secondo la formula

$$NEW\_PIXEL = \min(255, (CURRENT\_PIXEL - MIN) \ll SHIFT\_LVL)$$

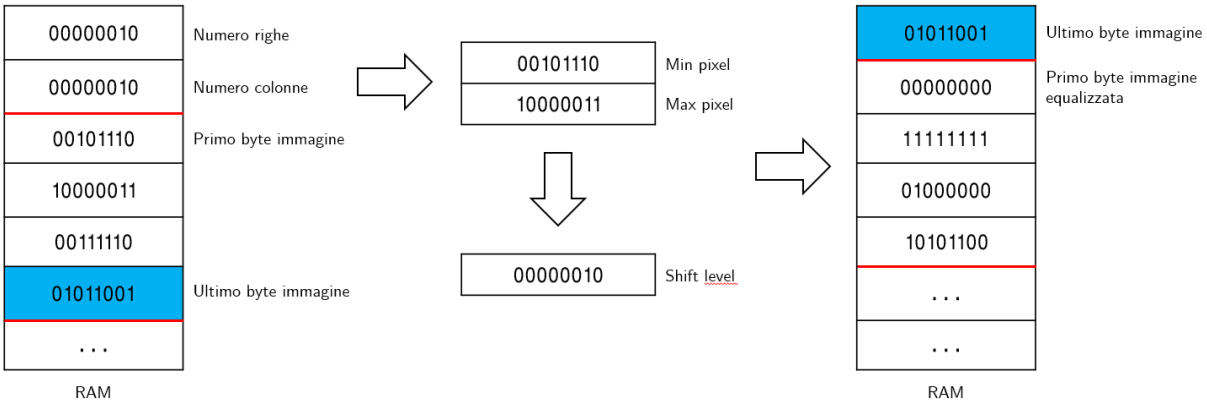


Figura 2: Esempio di funzionamento

L'implementazione deve essere scritta in VHDL e deve essere correttamente simulabile in pre-sintesi e in post-sintesi su FPGA target xc7a200tfbg484-1. Inoltre, l'implementazione deve essere in grado di codificare altre immagini senza ricevere segnale di reset, ma l'immagine da codificare non può essere modificata durante l'esecuzione.

2 Architettura

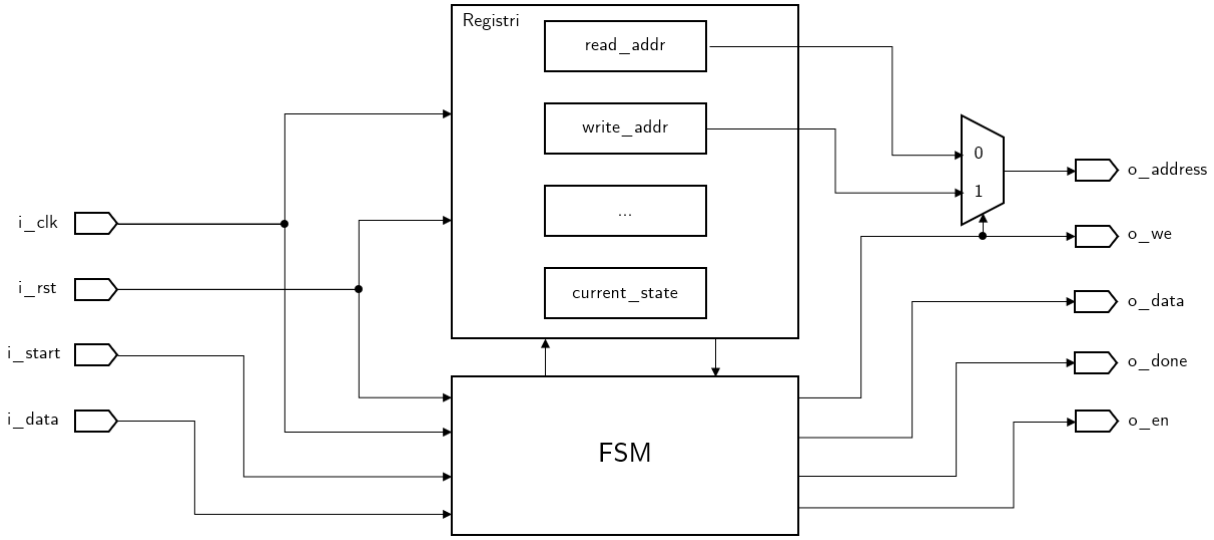


Figura 3: Schema sintetico dei moduli

L'architettura del componente si divide in due moduli principali: il modulo dei registri, in cui vengono salvate le informazioni necessarie e il modulo della macchina a stati finiti (da qui in poi FSM) che gestisce l'avanzamento delle operazioni. È stato scelto un approccio *single process*, per diminuire la complessità dell'architettura e aumentare la leggibilità del codice. La maggiore criticità con tale approccio è data dal fatto che il cono di logica di ogni stato venga eseguito una volta per ogni ciclo di clock. Per questo motivo, eventuali percorsi critici troppo lunghi (con ritardo più grande del periodo di clock) potrebbero causare glitch nell'esecuzione. Per testare questa criticità, sono stati svolti molti test con periodo di clock inferiore a quello specificato (fino a 15 ns), non rilevando mai comportamenti errati.

Le porte di uscita sono collegate direttamente alla logica interna dell'FSM, tranne *o_address*, l'indirizzo di lettura/scrittura della RAM. Questo segnale è collegato al registro appropriato grazie a un multiplexer con ingresso di selezione *o_we* che controlla il cambio di modalità lettura/scrittura della RAM.

2.1 Registri

Questo modulo sequenziale è realizzato interamente con descrizione comportamentale. Sono presenti sei registri con reset asincrono, gestiti da un solo *process*, e un registro con reset sincrono, gestito da un altro *process*:

- *current_state*: 4 bit, memorizza lo stato corrente della FSM
- *read_addr*: 16 bit, indirizzo di lettura dalla RAM
- *write_addr*: 16 bit, indirizzo di scrittura sulla RAM
- *max_pixel*: 8 bit, valore assunto dal pixel con intensità massima nell'immagine originale
- *min_pixel*: 8 bit, valore assunto dal pixel con intensità minima nell'immagine originale
- *shift_lvl*: 4 bit, valore di shift necessario per calcolare le nuove intensità dei pixel dell'immagine con l'istogramma equalizzato
- *end_addr*: 16 bit, indirizzo successivo a quello dell'ultimo byte dell'immagine originale. Questo è l'unico registro con reset sincrono, in quanto viene coinvolto in una moltiplicazione e il *Report Methodology* della *RTL Analysis* di Vivado ha messo in evidenza la necessità di implementare un registro sincrono, per poter ottimizzare la logica in fase di sintesi

Tutti i registri sono realizzati tramite flip-flop di tipo D. Si vuole mettere in evidenza la scelta del margine di clock su cui i registri vengono aggiornati: per i registri asincroni si è scelto il margine discendente, in quanto i nuovi valori vengono assegnati all'interno della logica della FSM, anch'essa aggiornata sul margine discendente; mentre il registro sincrono viene aggiornato sul margine ascendente, per evitare di dover aspettare un intero ciclo di clock prima di completare l'aggiornamento.

2.2 FSM

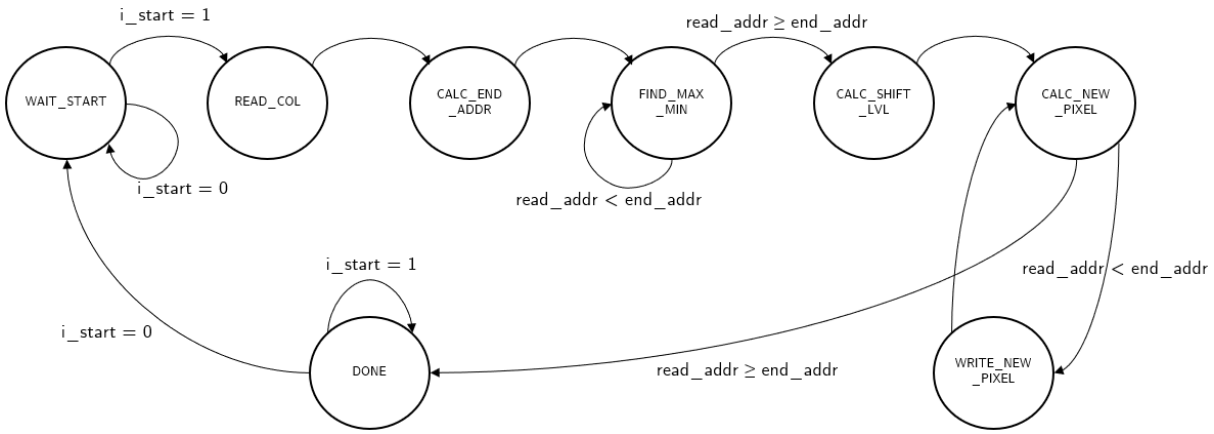


Figura 4: Diagramma degli stati della FSM

Questo modulo è realizzato interamente con descrizione comportamentale, usando lo stesso *process* dei registri con reset asincrono. Lo stato corrente tiene traccia dell'avanzamento delle operazioni e seleziona il cono di logica da eseguire. Segue una lista degli stati con una breve spiegazione delle operazioni compiute in ciascuno di essi:

- *WAIT_START*: stato di reset, si attende che il segnale *i_start* venga portato ad 1 per iniziare l'esecuzione
- *READ_COL*: viene letto il numero di colonne dell'immagine dalla RAM. Si noti che, per evitare di usare flip-flop non strettamente necessari, questo risultato intermedio viene memorizzato nei byte meno significativi del registro *end_addr*
- *CALC_END_ADDR*: viene letto il numero di righe e si calcola e memorizza l'indirizzo successivo a quello finale dell'immagine originale

$$END\_ADDR = COL * ROW + 2$$

- *FIND_MAX_MIN*: si leggono uno a uno i pixel dell'immagine originale e si individuano in parallelo i pixel, con valore maggiore e minore, che verranno poi memorizzati
- *CALC_SHIFT_LVL*: viene calcolato il livello di shift tramite confronti successivi per evitare di dover implementare una ALU che sia in grado di svolgere il logaritmo
- *CALC_NEW_PIXEL*: per ogni pixel dell'immagine originale, viene calcolato il nuovo valore e viene posto in uscita per essere scritto sulla RAM
- *WRITE_NEW_PIXEL*: si attende che la scrittura venga completata e si prepara la RAM per la lettura del prossimo pixel
- *DONE*: l'equalizzazione dell'istogramma è completata, si attende che venga eseguita la procedura di restart come da specifica

Si evidenzia la scelta progettuale di fornire alla RAM l'indirizzo di scrittura/lettura necessario nello stato precedente, rispetto a quello dell'operazione vera e propria. Ciò, insieme alla scelta di aggiornare lo stato corrente sul margine discendente del segnale di clock, inversamente alla RAM che è aggiornata sul margine ascendente, permette di ottimizzare al massimo i tempi di esecuzione, rimuovendo la necessità di aspettare un'intero ciclo di clock prima di leggere/scrivere i dati della RAM.

3 Risultati Sperimentali

3.1 Sintesi

La sintesi effettuata con software Vivado 2020.2, utilizzando strategia standard su FPGA Artix-7 xc7a200tfbg484-1, non presenta alcun tipo di warning. Vengono utilizzati i seguenti componenti:

- 178 LUT (0,13% del totale)
- 98 FF (0,04% del totale)

Durante la fase di progetto, è stata posta particolare attenzione nell'usare i flip-flop strettamente necessari e nell'evitare la creazione di latch indesiderati.

3.2 Simulazioni

Di seguito vengono riportate tutte le simulazioni ritenute importanti per dimostrare la correttezza dell'implementazione.

3.2.1 Immagine da 0 byte

Questo caso di test controlla che non si verifichino errori quando in input viene fornita un'immagine che ha il numero di colonne, il numero di righe o entrambi a 0.

3.2.2 Stesso valore per tutti i pixel

Questo caso di test controlla che la situazione in cui tutti i pixel hanno intensità uguale sia gestita adeguatamente, cioè che l'immagine risultante sia completamente bianca come da specifica. Inoltre il test, copre anche i casi in cui tutti i pixel hanno intensità al limite dell'intervallo (tutti 0 o tutti 255).

3.2.3 Delta massimo

Questo caso di test controlla che non si verifichino errori quando il pixel massimo ha valore 255 e il pixel minimo ha valore 0. Si è rivelato particolarmente utile nel mettere in luce la necessità di usare una variabile intermedia di 9 bit per calcolare il delta senza incorrere in problemi.

3.2.4 Più immagini senza reset

Questo caso di test controlla che l'architettura sia in grado di codificare correttamente più immagini una di seguito all'altra senza ricevere reset.

3.2.5 Più immagini con reset

Questo caso di test controlla che l'architettura sia in grado di codificare correttamente più immagini una di seguito all'altra ricevendo reset tra ogni immagine.

3.2.6 Reset durante la computazione

Questo caso di test ferma la computazione di un'immagine con un reset. È stato utile ad identificare una svista nel codice che causava il mancato ripristino allo stato iniziale corretto del registro contenente il valore massimo dei pixel.

3.2.7 Casuali

Mille test generati casualmente per coprire una grande varietà di casi.

4 Conclusione

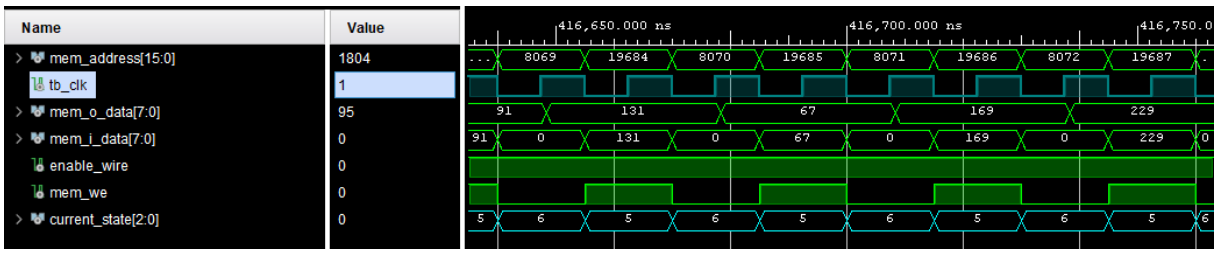


Figura 5: Waveform simulata post sintesi

Come dimostrato dai risultati sperimentali, l'architettura sopra descritta rispetta la specifica e vanta prestazioni considerevoli anche in seguito alla sintesi su FPGA. Si è infatti verificato, sperimentalmente, che il periodo di clock può essere abbassato senza problemi fino a un ordine di grandezza in meno rispetto a quello dato dalla specifica, portando notevoli

miglioramenti nel tempo di equalizzazione dell'istogramma di immagini di grandi dimensioni. Come si vede in Figura 5, il componente è in grado di scrivere un pixel equalizzato ogni due cicli di clock, senza sprecare tempo in stato di attesa. Si reputa che questa sia la performance migliore ottenibile, perché si è limitati da un solo canale di accesso alla RAM, che impedisce la parallelizzazione delle operazioni.

L'approccio *single process* si è dimostrato una scelta di design solida, in quanto ha permesso di descrivere l'architettura in modo semplice e intuitivo, evitando di dover utilizzare molti segnali interni di controllo.