POLITECNICO
MILANO 1863

Progetto di Reti Logiche

William Zeni
matricola 10613915

Cristina Urso
matricola 10599689

9 maggio 2021

Progetto sostenuto presso il Politecnico di Milano dipartimento di Elettronica, Informazione e Bioingegneria, diretto dal professor Gianluca Palermo nell'anno 2020/21.

Indice

1 Introduzione

1.1 Scopo del progetto

Write somenthing here

1.2 Specifiche generali

Write somenthing here

1.3 Interfaccia del componente

Write somenthing here

1.4 Dati e Descrizione memoria

Write somenthing here

2 Desing Pattern

2.1 Scelte Progettuali

La struttura del progetto è stata suddivisa in due process principali: `UPDATE` e `STATES`. Il primo ha il compito di relazionarsi con la memoria e il secondo contiene gli stati della macchina impiegati nella equalizzazione delle immagini. In questo modo una porzione di codice è adibita esclusivamente alla computazione dei pixel (process `STATES`), mentre la rimanente si occupa dei segnali di output della *entity* (process `UPDATE`). Per permette un corretto dialogo tra i due process è nata l'esigenza di avere dei segnali "duplicati". I segnali con suffisso `'_cp'` sono stati introdotti per mantere in memoria i valori computati, mentre i segnali con suffisso `'_next'` sono stati implementati allo scopo di permettere agli stati di far richieste alla memoria. Nel particolare, per ogni ciclo di clock, durante *rising_edge*, il process `UPDATE` si risveglia, aggiornando i segnali. I segnali contenenti i valori da mantenere vengono reimpostati con i segnali `'_cp'`, mentre i segnali di output della *entity* sono aggiornati con i segnali `'_next'`.

2.2 Elenco Stati

2.2.1 START

Lo stato di `START` è stato pensato come stato di attesa iniziale. Questo stato viene invocato in due situazioni differenti: se il segnale di `i_rst` viene portato alto, oppure quando il segnale `i_start` viene riportato basso dopo la computazione di un immagine. Lo stato `START` non cambia fino a quando il segnale `i_start` non viene portato alto. In quel momento lo stato successivo viene impostato `INIT`.

2.2.2 INIT

Lo stato `INIT` è uno stato di transizione nel quale il processore si assicura che i segnali siano inizializzati con i valori opportuni. Successivamente imposta lo stato prossimo a `ABILIT_READ`.

2.2.3 ABILIT_READ

Lo stato `ABILIT_READ` è lo stato attraverso il quale abilitiamo la memoria alla sola lettura. Viene richiamato in momenti diversi del progetto e, in base allo stato chiamante, instrada lo stato prossimo a quello opportuno.

2.2.4 ABILIT_WRITE

Lo stato `ABILIT_WRITE` abilita la memoria alla scrittura. Viene invocato subito dopo aver computato il valore del nuovo pixel e in nessun altro momento. Instrada poi lo stato prossimo a `WRITE_PIXEL`.

2.2.5 WAIT_MEM

Lo stato `WAIT_MEM` è uno stato centrale durante la gestione del flusso di dati. Sostanzialmente "spreca" un ciclo di clock. Questo ci assicura sia in caso di scrittura, sia in caso di lettura, che i segnali in ingresso e in uscita siano letti o scritti correttamente. Nel caso specifico alla quale ci rifacciamo, alcune chiamate a questo stato potevano essere evitate. Questa informazione è emersa durante lo stress test a cui il processore è stato sottoposto. Tuttavia, si è preferito lasciarle per mantenere la stuttura del processore. Ciò dovrebbe permette una maggior robustezza, sebbene un aumento nella latenza della computazione.

2.2.6 GET_RC

Lo stato `GET_RC` è uno stato in preparazione al calcolo della dimensione dell'immagine e dei punti in cui bisognerà scrivere all'interno della memoria. Lo stato `GET_RC` viene invocato dopo l'abilitazione della memoria alla lettura. Questo stato si occupa del recuperare i valori dalla memoria e aggiornare i segnali `n_col` e `n_row`.

2.2.7 GET_DIM

Lo stato `GET_DIM` è lo stato che si preoccupa di aggiornare il segnale `dim_address` con il valore opportuno. Il calcolo $n\_col \cdot n\_row + 2$ aggiorna il segnale al primo bit libero per la scrittura.

2.2.8 READ_PIXEL

Lo stato `READ_PIXEL` è uno stato strettamente accopiato con lo stato `GET_MINMAX`. Richiede alla memoria il valore del pixel e aggiorna il segnale `curr_address` a quello successivo. In questo modo il valore del pixel sarà disponibile sul segnale `i_data` al *rising_edge* successivo.

2.2.9 GET_MINMAX

Write somenthing here.

2.2.10 GET_DELTA

Write somenthing here

2.2.11 CALC_SHIFT

Write somenthing here

2.2.12 GET_PIXEL

Write somenthing here

2.2.13 CALC_NEWPIXEL

Write somenthing here

2.2.14 WRITE_PIXEL

Write somenthing here.

2.2.15 DONE

Write somenthing here

2.2.16 WAITINGPIC

Write somenthing here

3 Risultati dei Test

Write somenthing here!

4 Conclusioni

Write somenthing here