Prova finale (Progetto di Reti Logiche) [051228]

Francesco Circhetta [10496325]

June 4, 2018

Abstract

Lo scopo del progetto è la realizzazione di un componente hardware in VHDL. Esso riceve in ingresso un'immagine in toni di grigio e, dopo aver applicato un algoritmo di thresholding ad una soglia data, scrive in output l'area del rettangolo minimo che racchiude completamente l'immagine B/W risultante.

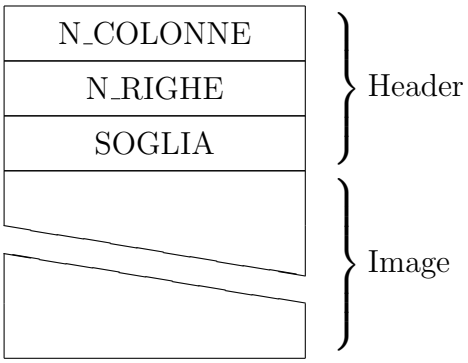
1 Introduzione

L'interfaccia del componente, così come presentata nelle specifiche [1], è la seguente:

```
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;
```

Il componente si interfaccia con un chip RAM nel quale è stata precaricata un'immagine RAW in toni di grigio, i relativi metadati e il valore soglia per il computo dell'area.

Il formato dell'immagine è il seguente:



dove **N_COLONNE** è la larghezza in px, **N_RIGHE** è l'altezza in px e **SOGLIA** è il threshold. L'immagine, header incluso, è collocata all'indirizzo `0x0002` ed è rappresentata come una matrice. Ogni pixel è memorizzato in un intero a 8 bit.

2 Design

Data la presenza di un segnale di start e di uno di reset, è ragionevole pensare che sia necessario un automa a stati finiti (FSM).

2.1 `IDLE` state

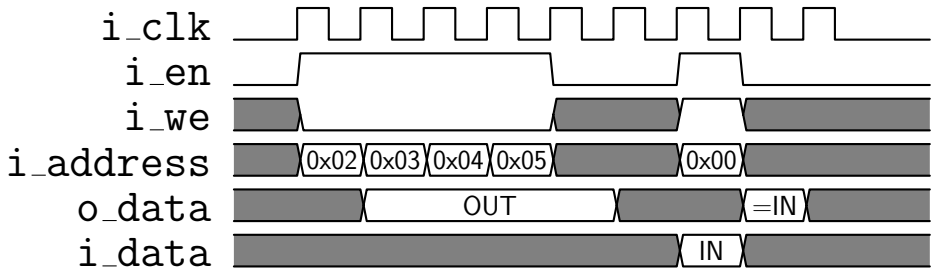
Lo stato iniziale è `IDLE`, nel quale la macchina attende il segnale di `i_start` proveniente dal testbench. Ogni altro stato confluisce ovviamente in `IDLE` ogni qualvolta `i_rst` è posto alto.

2.2 RAM interface

Innanzi tutto, per poter leggere l'immagine, è necessario memorizzare i suoi metadata. Ciò comporta la lettura dalla RAM delle locazioni di memoria `0x0002`, `0x0003`, `0x0004` e lo store dei contenuti.

Dunque, si analizza il protocollo del chip RAM. Dal codice fornito nelle specifiche si può concludere che ci si trova di fronte ad un chip di RAM sincrona.

Durante la lettura è sufficiente settare l'indirizzo per un solo ciclo di clock, ma il risultato non può essere utilizzato fino al successivo ciclo di clock. Ciò, tuttavia, è utile durante letture consecutive multiple, come si può evincere dall'immagine seguente:



2.3 `FETCH` state

La prima lettura dalla memoria RAM necessita di un ciclo di fetch. Il primo elemento che si vuole leggere è **N_COLONNE**, perciò `o_en` è posto alto e `o_address` è settato al primo indirizzo da leggere, `0x0002`. Il dato **N_COLONNE** sarà disponibile su `i_data` al ciclo di clock successivo.

2.4 `WIDTH, HEIGHT, THRES` states

Come si può evincere dai nomi di questi stati, è qui che avviene lo store del dati **N_COLONNE**, **N_RIGHE**, **SOGLIA**, letti dal bus. L'indirizzo viene via via incrementato ad ogni ciclo di clock, poichè è possibile accedere alla memoria RAM sequenzialmente.

2.5 `IMAGE` state

L'immagine è ora disponibile un pixel per ciclo di clock, seguendo una scansione lineare della matrice dell'immagine.

Contestualmente alla lettura viene applicato il thresholding rispetto al valore **SOGLIA** memorizzato durante lo stato `THRES`. Il valore ottenuto viene sottoposto ad un algoritmo di bounding box che si occupa di estrarre le coordinate delle righe e colonne che racchiudono completamente la figura risultante.

2.6 `DIM, AREA` state

Durante lo stato `DIM` vengono elaborati i risultati dell'algoritmo di bounding box. Quest'ultimo, infatti, ha restituito nel ciclo di clock precedente le due righe e le due colonne entro le quali l'immagine è completamente racchiusa. Per ottenere le dimensioni del rettangolo si eseguono le seguenti operazioni:

$$width = column_{max} - column_{min} + 1$$
$$height = row_{max} - row_{min} + 1$$

Al termine delle operazioni di somma e differenza, viene eseguita la moltiplicazione tra *width* e *column*, al fine di ottenere l'area del bounding box.

2.7 `AREAL, AREAH` states

Secondo le specifiche, il testbench richiede che il risultato della computazione sia salvato nella memoria RAM. Esso è un numero a 16 bit, pertanto va diviso in due byte: `0x0001` ne conterrà la parte più significativa, `0x0000` la parte meno significativa.

Il protocollo di comunicazione in scrittura è già stato analizzato e visualizzato nel paragrafo RAM interface

2.8 `DONE` state

Durante l'ultimo ciclo di clock la computazione è ormai terminata e viene posto alto il segnale in uscita `o_done`, la memoria RAM viene posta in IDLE abbassando il segnale `o_en`. Lo stato successivo è `IDLE` che predispone il componente ad una nuova elaborazione.

2.9 Transition table

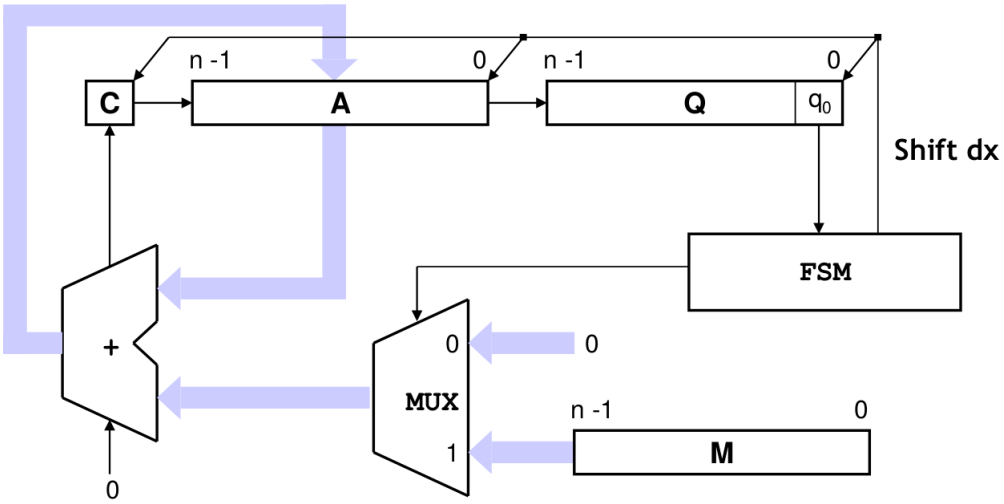
Current state	`i_start`	`i_rst`	`i_areadone`	`i_done`	Next state
*	x	1	x	x	IDLE
IDLE	1	x	x	x	FETCH
IDLE	0	x	x	x	IDLE
FETCH	x	x	x	x	WIDTH
WIDTH	x	x	x	x	HEIGHT
HEIGHT	x	x	x	x	THRES
THRES	x	x	x	x	IMAGE
IMAGE	x	x	x	1	DIM
IMAGE	x	x	x	0	IMAGE
DIM	x	x	x	x	AREA
AREA	x	x	1	x	AREAL
AREA	x	x	0	x	AREA
AREAL	x	x	x	x	AREAH
AREAH	x	x	x	x	DONE
DONE	x	x	x	x	IDLE

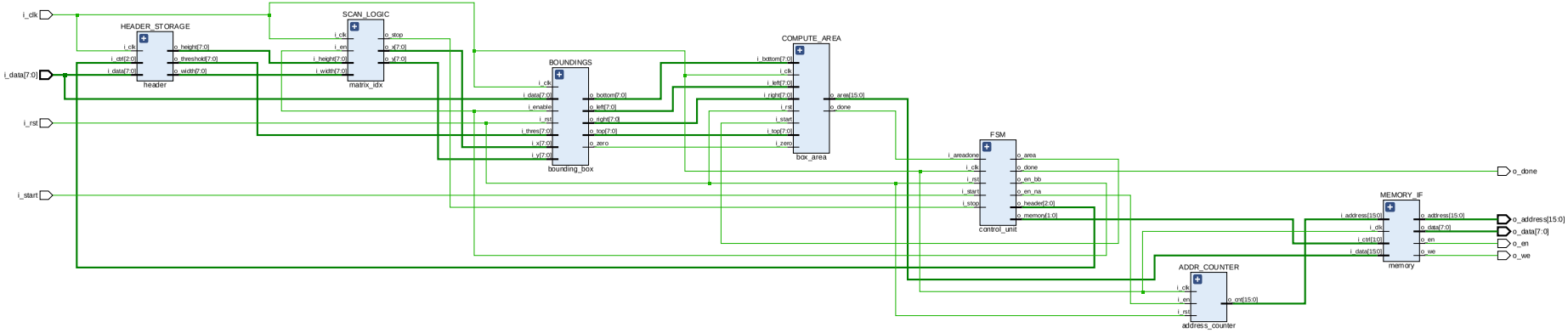
3 RTL schematic

Nella pagine seguenti è riportato lo schema finale RTL del componente così come presentato da Vivado in seguito all'elaborazione del listato VHDL.

Si nota subito la presenza del sotto-componente `FSM` le cui linee di uscita controllano il comportamento dei restanti blocchi. `HEADER_STORAGE` contiene i tre registri (3 bytes) nei quali viene salvato il contenuto dell'header dell'immagine. `SCAN_LOGIC` è il blocco che tiene traccia della posizione corrente nell'elaborazione della matrice dell'immagine. `BOUNDINGS` implementa in hardware l'algoritmo di bounding box e il confronto tra pixel in ingresso e valore soglia. `ADDR_COUNTER` è un contatore incrementale utilizzato per la lettura sequenziale della RAM, `MEMORY_IF` è semplice glue logic combinatoria.

Un occhio di riguardo merita il componente `COMPUTE_AREA`. Esso si avvale di un componente interno, ovvero un moltiplicatore sequenziale a 8 bit, il cui schema può essere osservato nell'immagine seguente.




i_clk
i_data[7:0]
i_rst
i_start
HEADER_STORAGE
i_clk
i_ctrl[2:0]
i_data[7:0]
o_height[7:0]
o_threshold[7:0]
o_width[7:0]
header
SCAN_LOGIC
i_clk
i_en
i_height[7:0]
i_width[7:0]
o_stop
o_x[7:0]
o_y[7:0]
matrix_idx
BOUNDINGS
i_clk
i_data[7:0]
i_enable
i_rst
i_thres[7:0]
i_x[7:0]
i_y[7:0]
o_bottom[7:0]
o_left[7:0]
o_right[7:0]
o_top[7:0]
o_zero
bounding_box
COMPUTE_AREA
i_bottom[7:0]
i_clk
i_left[7:0]
i_right[7:0]
i_rst
i_start
i_top[7:0]
i_zero
o_area[15:0]
o_done
box_area
FSM
i_areadone
i_clk
i_rst
i_start
i_stop
o_area
o_done
o_en_bb
o_en_na
o_header[2:0]
o_memory[1:0]
control_unit
o_done
ADDR_COUNTER
i_clk
i_en
i_rst
o_cnt[15:0]
address_counter
MEMORY_IF
i_address[15:0]
i_clk
i_ctrl[1:0]
i_data[15:0]
o_address[15:0]
o_data[7:0]
o_en
o_we
memory
o_addres[15:0]
o_data[7:0]
o_en
o_we

4 Ottimizzazioni

La parte più onerosa dell'elaborazione, in seguito all'algoritmo scelto per la computazione, è la lettura sequenziale dell'immagine. Pertanto le decisioni si sono orientate verso l'innalzamento della velocità massima di clock, anche a scapito di un piccolo aumento del numero di cicli necessari (numero comunque costante e che quindi non aumenta la complessità computazionale dell'algoritmo).

Il chip FPGA sul quale è stato modellato lo sviluppo del componente hardware mette a disposizione dei blocchi DSP capaci di svolgere in velocità elaborazioni complesse. Tuttavia si è scelto di non usarli, poichè energivori e difficili da pilotare e debuggare.

Durante la valutazione statica del timing, la prima difficoltà si è riscontrata nei numerosi livelli generati dalla sintesi di un moltiplicatore a 8 bit completamente combinatorio. Il ritardo di propagazione molto alto costringeva l'intero componente a lavorare ad una velocità di clock inferiore ai 120 MHz.

Si è dunque optato per la realizzazione di un moltiplicatore seriale a 8 cicli di clock. Esso prolunga il tempo di computazione della singola operazione di moltiplicazione, ma permette, dopo una successiva ottimizzazione del resto del componente, di più che raddoppiare la velocità di lettura dell'immagine, ovvero la fase più onerosa di tutto il processo.

Notevole attenzione ha successivamente richiesto la codifica degli stati della FSM: in seguito a successivi tentativi si è scoperto che la scelta iniziale del compilatore di seguire la codifica one-hot semplificava la logica della FSM stessa, ma introduceva nuove sfide dal punto di vista del routing.

La scelta manuale della codifica Gray ha permesso di recuperare alcune frazioni di ns dovute al ritardo di propagazione all'interno delle matrici di interconnessione del chip.

Al fine di innalzare ulteriormente la velocità di clock si è proceduto a verificare il numero di livelli della logica combinatoria componente per componente.

Si sono aggiunte ottimizzazioni come il precalcolo della condizione di stop e un ciclo aggiuntivo per il calcolo degli operandi della moltiplicazione finale.

5 Testbench

I 4 testbench forniti per l'autovalutazione sono stati riunificati in un unico VHDL in grado di modificare al volo il contenuto della RAM. Ciò ha consentito, tra l'altro, di testare la specifica della riutilizzabilità immediata del componente al termine della segnalazione tramite `o_done`.

Ai 4 test ne è stato aggiunto un quinto che spinge il componente verso due corner case: l'immagine fornita ha dimensioni massime consentite dalla specifica e dall'indirizzamento della memoria RAM (16 bit - 64 kB) e presenta un solo pixel acceso nell'ultima locazione di memoria.

L'unico pixel in posizione finale testa la propagazione del dato lungo la catena di registri e reti combinatorie in unione alla condizione di stop.

6 Risultati

Il componente supera correttamente la simulazione *Behavioral*, la simulazione *Post-Syntesis Functional* e *Timing* (con periodo di clock di 4 ns).

Ulteriori test sono stati effettuati nel corso della progettazione del componente anche *Post-Implementation*, arrivando alla conclusione che con il giusto lavoro di routing il componente potrebbe superare la velocità di clock di 250 MHz raggiunta in via teorica con la sintesi.

La stima della velocità massima di esercizio in sintesi, infatti, assume una lunghezza media per il percorso di propagazione dei segnali. Dall'analisi statica dei timings si nota come la suddivisione percentuale tra delay della logica e delay delle nets è nettamente in favore di queste ultime, grazie anche al lavoro di riduzione dei livelli di logica combinatoria.

References

[1] *Progetto Finale di Reti logiche*, available here.