Progetto finale di Reti Logiche

Weger Marco - Matricola n° 888201

Anno Accademico 2019/2020

Contenuto

1 Introduzione

Per questo progetto mi sono posto l'obiettivo di descrivere un componente che rispetti le specifiche sia in pre sintesi che in post sintesi. Ho voluto scrivere del codice di facile lettura e che si adatti in modo semplice e rapido a qualsiasi tipo di modifica del pattern e/o della dimensione della memoria e del suo contenuto (più dettagli in seguito). La FPGA consigliata non ci pone particolari vincoli di area nonostante ciò ho voluto dare particolare riguardo sia ai tempi di esecuzione che all'area occupata. Per quanto riguarda la frequenza di clock non sono andato alla ricerca di una massimizzazione in quanto la specifica fissa il periodo di clock a 100 ns.

1.1 Obiettivi aggiuntivi

Dopo essermi assicurato di rispettate le richieste della specifica fornita mi sono posto i seguenti obiettivi:

1. Minimizzare il tempo trascorso dal momento che il segnale di start viene ricevuto al momento di invio del segnale di done;
2. Disattivare il segnale di enable della memoria tra le varie esecuzioni;
3. Descrivere un componente in grado di funzionare anche nel caso ci fossero reset asincroni durante l'esecuzione di una codifica;
4. Rendere il componente adattabile ad un'eventuale modifica della lunghezza dell'indirizzo della cella di memoria tramite una costante;
5. Rendere il componente adattabile ad un'eventuale modifica della dimensione di una singola cella di memoria tramite una costante (ADDR);
6. Rendere il componente adattabile ad un'eventuale modifica del numero di elementi in una working-zone tramite una costante (WZ_OFFSET);
7. Rendere il componente adattabile ad un'eventuale modifica del numero di working-zone tramite una costante (WZ_NUM).

Al fine di raggiungere i sopracitati obiettivi ho assunto che l'indirizzo da codificare e l'indirizzo codificato vengano sempre salvati in successione in celle immediatamente consecutive all'ultimo indirizzo di working-zone (es. se ci fosse 16 working-zone RAM(16) conterrebbe l'indirizzo da codificare e RAM(17) l'indirizzo codificato). Tutte le ottimizzazioni descritte in seguito sono state valutate sulla base dei dati forniti dalla specifica e non tengono conto dell'eventuale crescita sproposita delle costanti sopracitate.

1.2 Obiettivo velocità

La limitazione più stringente in termini di velocità è data dall'accesso alla RAM e dai suoi ritardi. Al fine di minimizzare i tempi di letture e confronti ho

optato per un componente che al momento della lettura di un dato imposti contemporaneamente la successiva richiesta. In questo modo posso garantire l'esecuzione dei confronti tra l'indirizzo e le N working-zone in N+1 cicli di clock. Occorreranno poi 2 cicli di clock per la scrittura del dato codificato in memoria e la notificazione di elaborazione completata. Un ulteriorie ciclo di clock mi è servito a garantire che il segnale *o_mem* non rimanesse attivo tra un'esecuzione e la successiva. Ricapitolando, nel caso pessimo in cui l'indirizzo letto non si trova in nessuna delle work-zone

$$T_{\text{esecuzione}} = (N + 1) \cdot T_{\text{clock}} + 3 \cdot T_{\text{clock}}$$

Nella valutazione del tempo di esecuzione va preso in considerazione il fatto che il segnale di start potrebbe arrivare sul fronte di discesa del clock, qualora succedesse l'esecuzione ritarderà di $T_{\text{clock}}/2$. Nella specifica forntia l'esecuzione massima dovrà durare 1200 ns (nel caso in cui *i_start* fosse allineato al fronte di salita del clock), 1250 ns altrimenti.

1.3 Obiettivo codice semplice e riutilizzabile

Ho deciso di realizzare una macchina a stati finiti al fine di rendere facilmente comprensibile ed eventualmente modificabile ogni stato. Tutti i segnali sono gestiti da un'unica entity ad eccezione di un contatore generico utilizzato per scandire le working-zone. Le seguenti costanti garantiscono un alto livello di riutilizzabilità (tra parentesi i valori assegnati da specifica):

- **SIZE_MEM** (16): dimensione dell'indirizzo di memoria;
- **SIZE_ADDR** (8): dimensione di una cella di memoria;
- **SIZE_WZ** (4): estensione della singola working-zone;
- **COUNT_WZ** (3): dimensione del vettore *WZ_NUM*;
- **N_WZ** (8): numero di working-zone.

Viene naturale notare la relazione tra *COUNT_WZ* e *N_WZ*, nonostante ciò ho voluto esplicatre costanti differenti in modo da far fronte a situazioni in cui il numero di working-zone non è potenza di 2.

1.4 Funzionamento in sintesi

Una soluzione che memorizza tramite registri i valori delle working-zone non avrebbe migliorato in modo significativo i tempi di esecuzione peggiorando però l'area occupata pertando ho optato per la in memoria a ogni esecuzione. La singola esecuzione di una codifica può essere descritta attraverso un numero finito di step (che poi diventeranno una macchina a stati finiti):

1. Reset ed attesa del segnale di start (*i_start=1*);

2. Abilitazione della memoria e richiesta dell'indirizo da codificare (salvato in un registro);
3. Richiesta della i-esima working-zone e confronto con l'indirizzo salvato, eventuale codifica e passaggio a step successivo (passo ripetuto per i compreso tra 0 e il numero di working-zone);
4. Scrittura dell'indirizzo codificato in memoria;
5. Invio segnale di elaborazione completata (*o_done=1*) e attesa feedback (*i_start=0*), il dato è disponibile fin dal momento in cui *o_done* viene portato a 1;

1.5 Note aggiuntive sulla specifica

Per la sintesi è stata scelta l'FPGA xc7a200tfbg484-1.

2 Architettura

2.1 Macchina a stati finiti

Il funzionamento del componente è scandito dalla macchina a stati finiti in figura 1, la funzionalità di ogni singolo stato è descritta nella tabella 1. La macchina è sincronizzata con il segnale *i_clk* e sensibile al segnale *i_start*.

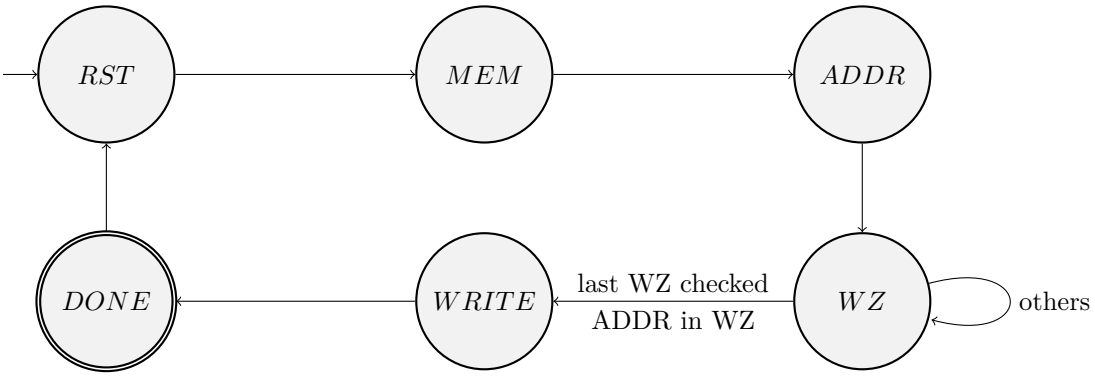

Figura 1: macchina a stati finiti implementata.

Stato	Descrizione
RST	Stato di reset della macchina. Da questo stato partiranno anche eventuali rieseccuzioni senza reset. Tutti i segnali vengono settati al loro valore di default (compreso *o_done* che potrebbe trovarsi alto) e il componente si prepara a leggere il l'indirizzo da codificare. La memoria è disattivata.
MEM	Utilizzato esclusivamente per abilitare la memoria prima di iniziare le letture.
ADDR	L'indirzzo da codificare viene prelevato e salvato in un apposito registro. Il contatore che scandisce le working-zone viene resettato e la memoria viene preparata per la lettura della prima working-zone.
WZ	Unico stato ciciclico della macchina. La condizione di uscita dal ciclo è il raggiungimento dell'ultima working-zone (caso in cui l'indirizzo non subirà modifiche) oppure l'identificazione della working-zone di appartenza (indirizzo da codificare). Nell'ultima esecuzione dello stato la memoria viene preparata per la scrittura del dato (codificato o meno).
WRITE	Il segnale *o_we* viene alzato per permettere alla memoria di prelevare il dato (*o_we*).
DONE	Viene notificato il termine dell'esecuzione tramite *o_done* per poi tornare nello stato RST in attesa di una nuova esecuzione.

Tabella 1: stati della macchina a stati finiti implementata.

3 Sintesi

3.1 Registri sintetizzati

Analizzando il report di sintesi e lo schema prodotto si può notare che gli stati della macchina sono codificati con la notazione "one hot". Di seguito sono ricapitolati i registri sintetizzati.

N° bit	N° registri	Modulo	Contenuto
8	1	main	Salvataggio dell'indirizzo da confrontare con le working-zone.
1	3	main	Gestione di segnali a singolo bit: *o_done*, *o_en*, *o_we*.
6	1	main	Codifica lo stato corrente della macchina secondo la notazione "one hot".
3	1	counter	Contatore a 3 bit usato per scandire le working-zone.

Tabella 2: registri sintetizzati e loro contenuto.

3.2 Area occupata

Tramite le funzionalità di reportistica di VIVADO si possono analizzare il numero di LUT e di Flip Flop (a singolo bit) utilizzati.[1]

Risorsa	Utilizzo	Disponibilità	Utilizzo in %
Look Up Table	30	134600	0,02229
Flip Flop	20	269200	0,00743

Tabella 3: area utilizzata dal componente in relazione con l'FPGA consigliata.

3.3 Report di timing

La specifica non richiede la valutazione del Worst Negative Slack (WNS) di conseguenza verrà considerato massimo. Analizzando i test bench forniti ho notato che la RAM ha un ritardo di esecuzione (T_{RAM}) di 1 ns. Dalle precendenti considerezanio segue il minimo periodo calcolabile:

$$T_{\min} = T_{\text{RAM}} + T_{\text{clock}} - WNS \approx 1ns + 100ns - 100ns \approx 1ns$$

Ne consegue la massima frequenza di clock: $f_{\max} = 1/T_{\min} \approx 1Ghz$.

3.4 Warnings in post sintesi

Tutti i warning che si sono presentati durante lo sviluppo del componente sono stati risolti senza particolari difficoltà.

[1] Calcoli basati sulla dimensione della FPGA citata nella sezione 1.5

3.5 Note aggiuntive sulla sintesi

Per la sintesi ho usato la versione 2019.2 di XILINX VIVADO con i settaggi di default.

4 Simulazioni

Oltre a testare il componente con i test bench forniti dalla specifica ho ritenuto opportuno valutare alcuni casi critici del funzionamento del componente (*descritti nella sezione 4.3*).

4.1 Indirizzo non nella working-zone (test bench fornito)

4.1.1 Dati

4.1.2 Risultati in pre sintesi

4.1.3 Risultati in post sintesi

4.2 Indirizzo nella working-zone (test bench fornito)

4.2.1 Dati

4.2.2 Risultati in pre sintesi

4.2.3 Risultati in post sintesi

4.3 Altri test bench

5 Implementazione

Il funzionamento del componente è stato testato con successo anche post implementazione. Non sono state fatte ulteriori analisi sull'implementazione.

6 Conclusione

Ricapitolando ho descritto un componente con le seguenti caratteristiche:

- funzionante in pre/post sintesi e post implementazione;
- ottimizzato sotto il punto di vista dell'area utilizzata;
- ottimizzato sotto il punto di vista del tempo di esecuzione;
- altamente personalizzabile e configurabile mediante l'uso di costanti;
- utilizzo di LUT[2] par al 0,02229%;
- utilizzo di FF[2] par al 0,00743%.

[2]Calcoli basati sulla dimensione della FPGA citata nella sezione 1.5