Prova Finale (Progetto di Reti Logiche)

Scaglione P-ZZZZ – Prof. Gianluca Palermo – A.A. 2019-2020

Marco Petri (10569751) e g-picc

Indice

Introduzione

1.1 Il problema

Il problema trattato in questo progetto è quello della traduzione di un indirizzo in funzione di un dato insieme di indirizzi chiamati Working Zones (WZ): si hanno 8 WZ di dimensione 4 che non possono essere fra loro sovrapposte, nemmeno parzialmente. L'indirizzo da tradurre, di dimensione 7 bit, verrà lasciato inalterato aggiungendo solamente un bit '0' in testa nel caso in cui non sia presente in una delle WZ; mentre verrà codificato solo nel caso in cui risulti appartenente a una WZ, concatenando un '1' al numero in binario naturale delle WZ (3 bit) e all'offset nella WZ codificato con codifica One Hot (4 bit).

Ad esempio, se una WZ ha come indirizzo base 30 (di conseguenza costituito dagli indirizzi 30, 31, 32, 33) ed è la numero 4 e si vuole codificare gli indirizzi 31 e 35 (supponendo che nelle altre 7 WZ non ci sia l'indirizzo 35), allora avremo le seguenti codifiche:

Codifica del numero 31:

WZ_BIT	W_NUM			WZ_OFFSET			
1	1	0	0	0	0	1	0

Codifica del numero 35:

WZ_BIT	INDIRIZZO						
0	0	1	0	0	0	1	1

1.2 Scelte progettuali

Il problema in questione ha inizialmente portato a due modalità di risoluzione differenti: salvare le WZ in registri oppure non salvarle così da occupare meno spazio. Il salvataggio delle working zones, sebbene sia oneroso dal punto di vista della memoria, risulta essere più conveniente dal punto di vista della complessità temporale, perciò abbiamo optato per una soluzione in cui salvare tutte le working zones mediante signal secondo un'architettura behavioral. Infatti, grazie a questo approccio, basta salvare una sola volta le WZ negli appositi registri e nel caso di conversioni successive (senza che venga dato un reset a fine conversione) a fronte di un nuovo segnale di start dopo la prima esecuzione da un segnale di reset, si avrà bisogno solamente di ricaricare l'indirizzo da codificare e, una volta codificato, scriverlo all'interno della memoria.

Nella progettazione della macchina a stati si è quindi deciso di separare la parte del codice che si occupa di caricare tutte le WZ dalla parte del codice che si occupa di caricare l'indirizzo da codificare, benché l'operazione sia fondamentalmente la stessa con la sola differenza

dell'indirizzo richiesto; è stata effettuata una suddivisione funzionale degli stati, perciò ogni stato è caratterizzato dall'effettuare solamente una tipologia di operazione.
Per comunicare con la memoria bisogna tenere conto che essa esegue le operazioni richieste nel periodo in cui il clock è alto, allora il minimo numero di stati per leggere o scrivere da memoria è 2 oppure 3: 2 se si utilizza un'architettura con flip flop operanti sul fronte di discesa, mentre 3 nel caso in cui si operi sul fronte di salita.
In questo caso, si è preferito operare sul fronte di salita, per cui utilizzando 3 clock: richiesta dell'indirizzo, attesa della memoria e salvataggio/analisi.

Architettura

2.1 Idle

Lo stato Idle è lo stato iniziale in cui si trova la macchina dopo uno stato di reset, questo stato non fa alcunché fintanto che il segnale i_start è basso, quando questo viene portato ad alto si opera una transizione per andare allo stato Ask per iniziare la codifica.

2.2 Ask

Lo stato Ask è lo stato in cui si richiede una working zone in lettura alla memoria, utilizzando un registro per tenere conto dell'indirizzo da richiedere al momento della richiesta.

2.3 Wait_mem1

Lo stato Wait_mem1 è uno stato in cui attendiamo che la memoria esegua la richiesta di lettura dell'indirizzo e scriva tale valore negli ingressi del nostro modulo.

2.4 Save

Lo stato Save è lo stato in cui si effettua il salvataggio della n-esima WZ che abbiamo appena richiesto all'interno dell'apposito registro che utilizziamo per memorizzarla. Si ritorna allo stato Ask nel caso in cui il valore del registro count sia inferiore a 8 (salvando quindi tutte le WZ in appositi registri) e lo si incrementa.

2.5 Ask_addr

Lo stato Ask_addr è lo stato in cui si effettua la richiesta alla memoria dell'indirizzo da codificare.

2.6 Wait_mem2

Lo stato Wait_mem2 è lo stato in cui aspettiamo che la memoria gestisca la nostra richiesta di lettura dell'indirizzo da codificare.

2.7 Analize

Lo stato Analize salva il valore delle variabili necessarie per comporre l'indirizzo da inviare alla macchina per la scrittura, distinguendo quindi il caso in cui l'indirizzo è all'interno o all'esterno di una WZ.

2.8 Enc_wrt

Lo stato Enc_wrt è lo stato in cui scriviamo l'indirizzo codificato sulla memoria avendolo prima composto con i campi precedentemente calcolati.

2.9 Wait_mem3

Lo stato Wait_mem3 è lo stato in cui aspettiamo che la memoria gestisca la nostra richiesta di scrittura dell'indirizzo da codificare.

2.10 Done

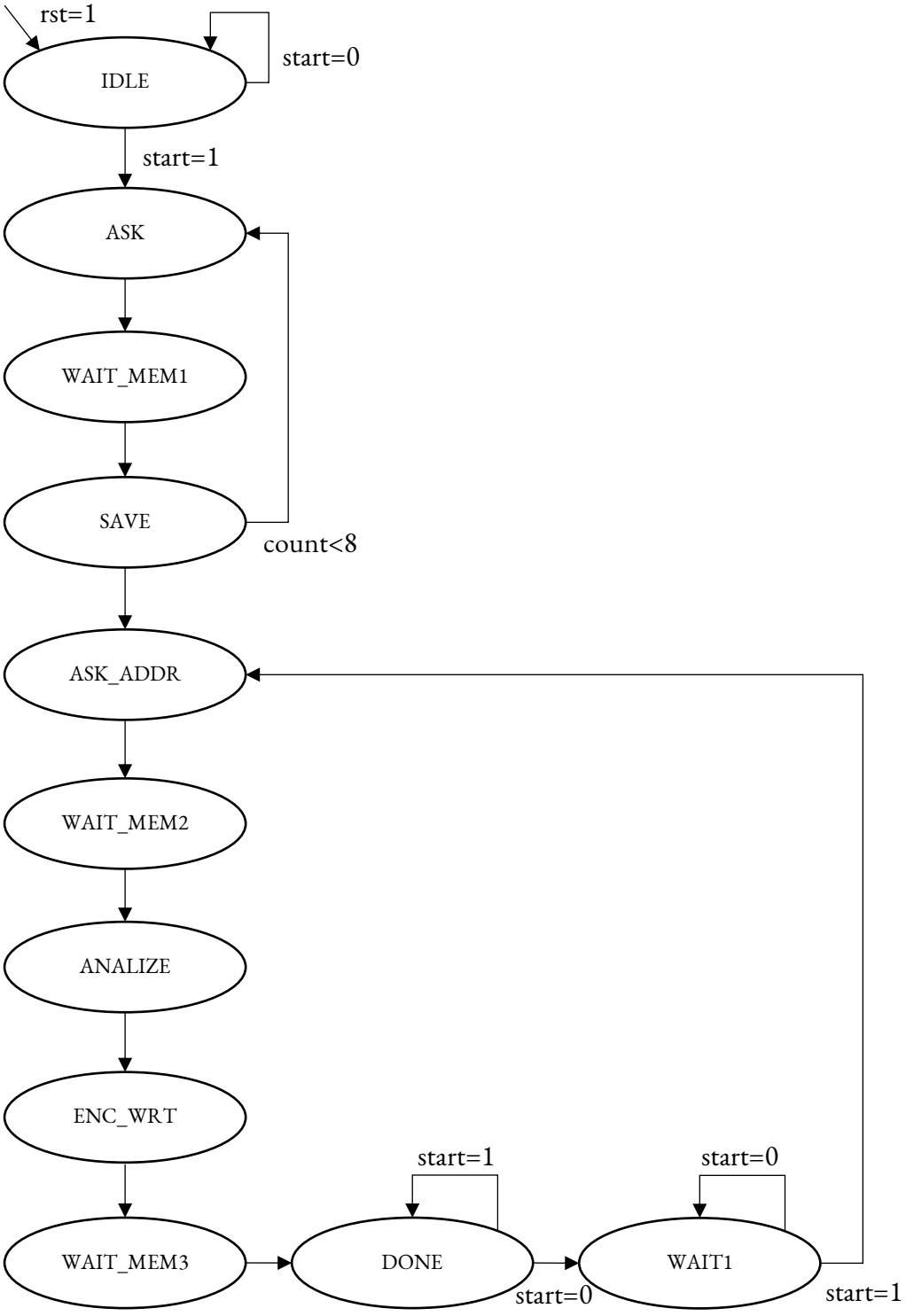
Lo stato Done alza il segnale Done a 1 poiché la codifica è completata e siamo sicuri che la RAM abbia scritto tale valore, oltretutto restiamo in questo stato finché i_start è alto, quando i_start viene portato basso allora si abbassa o_done e si transisce su Wait1.

2.11 Wait1

Lo stato Wait1 aspetta che venga dato un segnale i_start alto per reiniziare la codifica dal caricamento dell'indirizzo da codificare poiché le WZ sono già salvate.

2.12 Macchina a stati

Si riporta il modello della macchina a stati utilizzato per lo sviluppo del progetto:

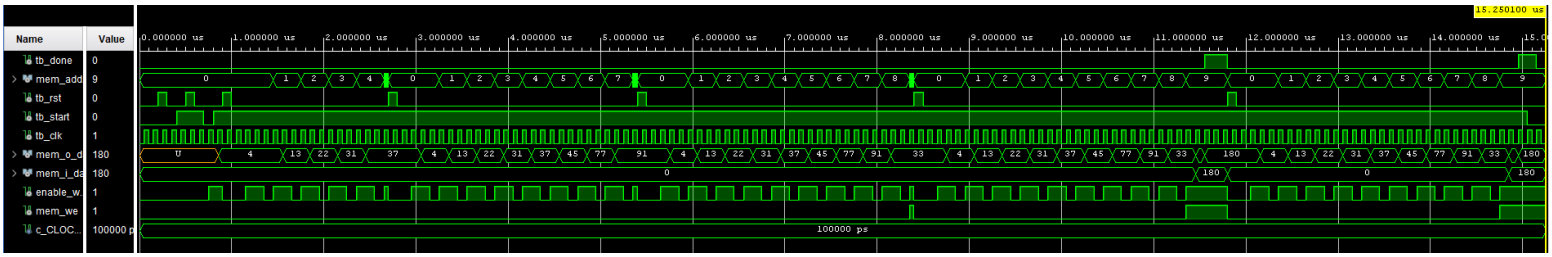


Testing e risultati sperimentali

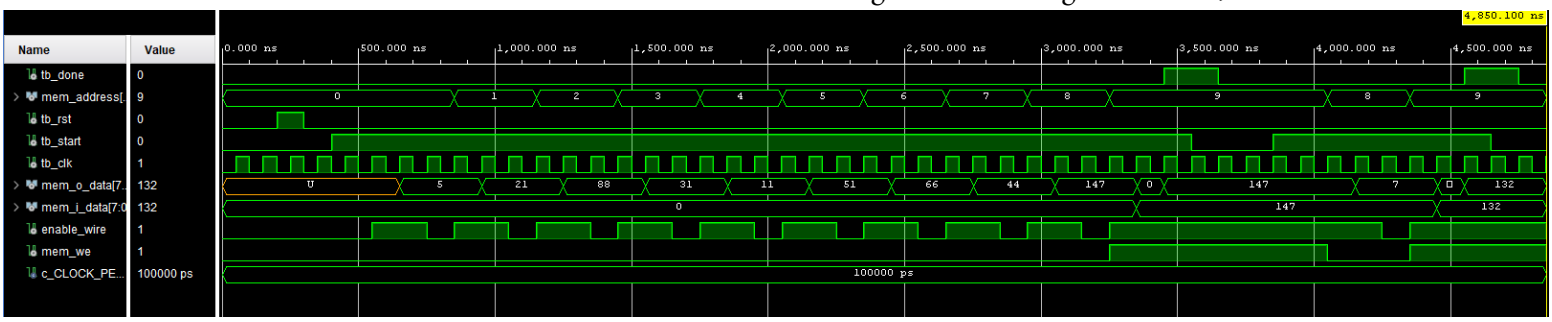
3.1 Testing

Il componente è stato sottoposto a diversi testbench di due tipologie: correttezza della specifica e stabilità. Con i primi si è voluto verificare il corretto comportamento del componente, sia in pre-sintesi che in post-sintesi: non sono di seguito riportati in quanto non particolarmente rilevanti, ma si osserva che si è verificato un corretto comportamento del componente. I secondi sono una serie di 9 test, dei quali si riportano i grafici di quelli ritenuti più importanti relativamente allo stress del componente:

1. Reset asincroni: questo test si basa sulla verifica del corretto comportamento del componente a seguito di reset asincroni forniti durante un'esecuzione. I reset sono stati forniti in diversi stati del componente per verificarne il corretto comportamento;

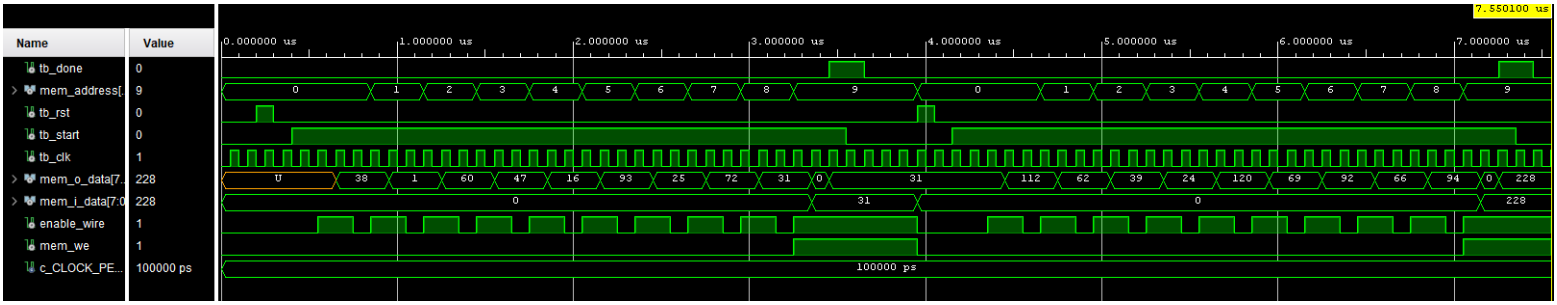


2. Doppio start OUT-IN: questo test si basa sulla verifica del corretto funzionamento del programma in seguito alla codifica di due indirizzi (diversi) fuori da WZ e successivamente dentro una WZ senza che venga dato alcun segnale di reset;



3. Doppio start OUT-OUT: questo test ha la stessa idea del precedente, con la differenza che viene effettuata la codifica successiva di due indirizzi all'esterno delle WZ diversi;
4. Doppio start IN-OUT: questo test ha la stessa idea del precedente, con la differenza che prima viene effettuata la codifica di un indirizzo interno ad una WZ e successivamente uno esterno alle WZ;
5. Doppio start IN-IN: questo test ha la stessa idea del precedente, con la differenza che viene effettuata la codifica successiva di due indirizzi all'interno delle WZ diversi;

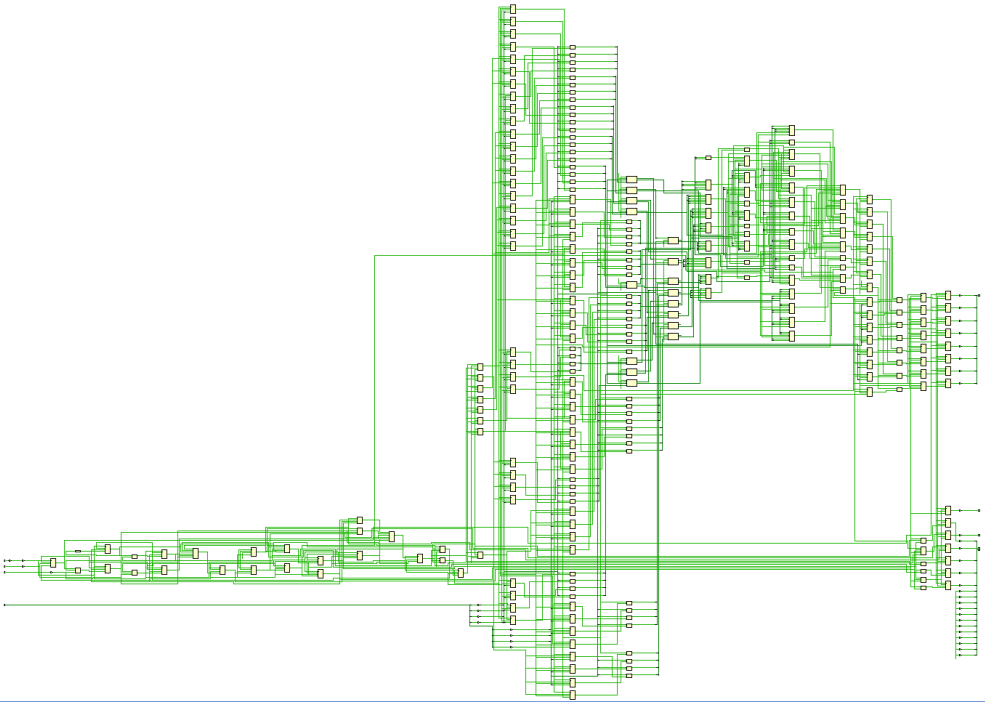
6. Reset OUT-IN: questo test si basa sulla verifica del funzionamento del programma nel caso in cui a fine computazione venga dato un segnale di reset e quindi potrebbero essere cambiate tutte o parte delle WZ presenti nella memoria oltre all'indirizzo da codificare. Prima viene effettuata una codifica con l'indirizzo all'esterno delle WZ e successivamente un test con un indirizzo (diverso) all'interno di una WZ:



7. Reset OUT-OUT: questo test ha la stessa idea del precedente, con la differenza che viene effettuata la codifica di due indirizzi (diversi) all'esterno dalle WZ;
8. Reset IN-OUT: questo test ha la stessa idea del precedente, con la differenza che prima viene effettuata una codifica con l'indirizzo all'interno di una WZ e successivamente un test con un indirizzo all'esterno di una WZ;
9. Reset IN-IN: questo test ha la stessa idea del precedente, con la differenza che nelle codifiche viene effettuata la codifica di due indirizzi (diversi) all'interno dalle WZ.

3.2 Risultati sperimentali

La sintesi del componente descritto al punto precedente realizzato in linguaggio VHDL, effettuata con VIVADO, porta ad una rete avente 117 Flip Flop e 130 LUT: i Flip Flop sono dati dall'utilizzo di diversi registri di cui uno a 16 bit e gli altri suddivisi in registri da 8 bit, 4 bit, 3 bit e 1 bit. Questi registri sono quelli che abbiamo utilizzato per mantenere lo stato delle WZ e per salvare gli altri valori da dare alle costanti che utilizziamo nel nostro progetto.
Si riporta lo schematic effettuato da Vivado relativamente alla post-sintesi:



Conclusione

Siccome i test che sono stati effettuati hanno ottenuto un risultato positivo sia in pre-sintesi che in post-sintesi funzionale possiamo dedurre con un buon livello di certezza che il componente sia esente da errori progettuali. Inoltre, abbiamo verificato che il componente funzioni anche per valori inferiori a 100ns del clock, quindi a frequenze maggiori, per avere un margine di sicurezza migliore del componente. Per quanto riguarda gli obiettivi prefissati, possiamo ritenere che la progettazione sia stata efficace: possiamo osservare dai testbench come la codifica di un indirizzo successivamente ad un'altra codifica (avendo già in memoria le WZ) richieda molto meno tempo rispetto a dover ricaricare gli indirizzi in memoria, favorendo quindi una esecuzione più veloce a discapito della memoria utilizzata, piuttosto che un componente utilizzante poca memoria a discapito della velocità di codifica degli indirizzi.