Prova Finale (Progetto di Reti Logiche)

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

Marco Petri (10569751) e Giuseppe Piccirillo (10568059)

Indice

## Tabella dei contenuti

[Introduzione 1](#_Toc35023212)

[1.1 Il problema 1](#_Toc35023213)

[1.2 Scelte progettuali 1](#_Toc35023214)

[Architettura 3](#_Toc35023215)

[2.1 Idle 3](#_Toc35023216)

[2.2 Ask 3](#_Toc35023217)

[2.3 Wait\_mem1 3](#_Toc35023218)

[2.4 Save 3](#_Toc35023219)

[2.5 Ask\_addr 3](#_Toc35023220)

[2.6 Wait\_mem2 3](#_Toc35023221)

[2.7 Analize 3](#_Toc35023222)

[2.8 Enc\_wrt 3](#_Toc35023223)

[2.9 Wait\_mem3 4](#_Toc35023224)

[2.10 Done 4](#_Toc35023225)

[2.11 Wait1 4](#_Toc35023226)

[2.12 Macchina a stati 5](#_Toc35023227)

[Testing e risultati sperimentali 6](#_Toc35023228)

[3.1 Testing 6](#_Toc35023229)

[3.2 Risultati sperimentali 7](#_Toc35023230)

[Conclusione 8](#_Toc35023231)

# Introduzione

## 1.1 Il problema

Il problema che si tratta in questo progetto è quello della traduzione di un indirizzo in funzione di un dato insieme di indirizzi chiamati Working Zones (WZ) che hanno una certa dimensione che per questo progetto è pari a 4 e la cardinalità dell’insieme è pari a 8, ovvero abbiamo 8 WZ di dimensione 4 che non possono essere fra loro sovrapposte, nemmeno parzialmente. L’indirizzo che ha 7 bit verrà trasmesso così com’è aggiungendo un bit 0 in testa se non appartenente ad una WZ mentre verrà codificato se appartenente a una WZ concatenando un 1 al numero in binario naturale delle WZ all’offset nella WZ codificato con codifica One Hot.

Se ad esempio una WZ è 30 ed è la numero 4 e dobbiamo codificare gli indirizzi 31 e 35 (supponendo che le altre WZ siano lontane e quindi 35 non appartenente ad alcuna WZ) 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

La risoluzione del problema della codifica di un indirizzo dato il pool poteva essere eseguito secondo la nostra analisi in due modi differenti: salvando le working zone in registri oppure non salvandole per occupare meno spazio. Il salvataggio delle working zone 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 zone mediante signal secondo un’architettura behavioral. Mediante questo approccio possiamo sfruttare nel caso di conversioni successive senza che venga dato un reset il fatto di aver già presenti in appositi registri tutte le WZ per la codifica, quindi a fronte di un nuovo segnale di start dopo la prima esecuzione da un segnale di reset, avremo bisogno solamente di ricaricare l’indirizzo da codificare e di codificarlo e scriverlo all’interno della memoria.

Nella progettazione della macchina a stati abbiamo 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, così la scrittura della macchina a stati viene effettuata con stati divisi per funzionalità in cui ogni stato è caratterizzato dallo svolgere solamente un’operazione.

Per comunicare con la memoria dobbiamo tenere conto che essa esegue le operazioni da noi 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 nel caso in cui si operi sul fronte di discesa, mentre 3 nel caso in cui si utilizzi un’architettura con flip flop operanti sul fronte di salita del clock.

Abbiamo 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.

## 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 in 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:

rst=1

start=1

start=1

start=1

start=0

start=0

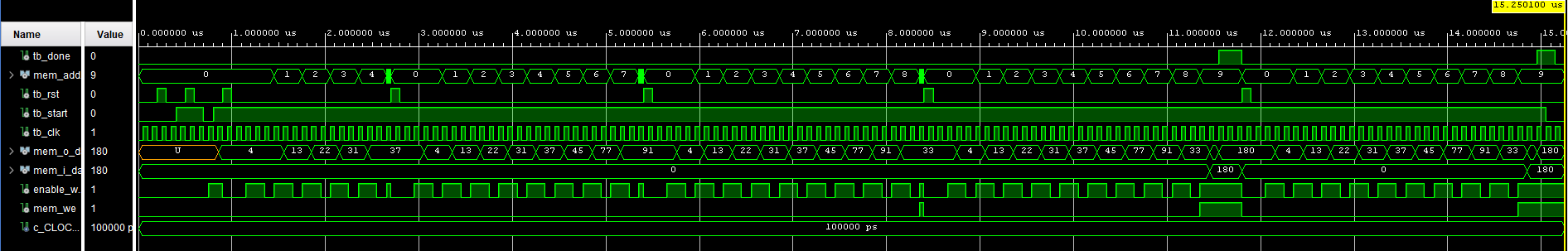
start=0

# Testing e risultati sperimentali

## 3.1 Testing

Il componente è stato sottoposto diversi testbench di due tipologie: correttezza della specifica e stabilità. Con i primi si è voluto verificare il corretto comportamento del componente si pre-sintesi e in post-sintesi, pertanto poiché non particolarmente rilevanti non sono di seguito riportati ma ci si limita ad osservare che con i test effettuati 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;



1. **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;

A picture containing indoor, sky

Description automatically generated

1. **Doppio start OUT-OUT**: questo test ha la stessa idea del precedente, viene effettuata la codifica successiva di due indirizzi all’esterno delle WZ diversi;
2. **Doppio start IN-OUT**: questo test ha la stessa idea del precedente, prima viene effettuata la codifica di un indirizzo interno ad una WZ e successivamente uno esterno alle WZ;
3. **Doppio start IN-IN**: questo test ha la stessa idea del precedente, viene effettuata la codifica successiva di due indirizzi all’interno delle WZ diversi;
4. **Reset OUT-IN**: questo test si basa sulla verifica di un 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:

A picture containing indoor

Description automatically generated

1. **Reset OUT-OUT**: questo test ha la stessa idea del precedente, nelle codifiche viene effettuata la codifica di due indirizzi (diversi) all’esterno dalle WZ;
2. **Reset IN-OUT**: questo test ha la stessa idea del precedente, 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;
3. **Reset IN-IN**: questo test ha la stessa idea del precedente, nelle codifiche viene effettuata la codifica di due indirizzi (diversi) all’interno dalle WZ.

## 3.2 Risultati sperimentali

La sintesi effettuata con VIVADO del componente descritto al punto precedente realizzato in linguaggio VHDL 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.

# 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. Abbiamo verificato che il componente funzioni anche per valori inferiori a 100ns del clock per avere un margine di sicurezza migliore del componente. Per quanto riguarda gli obiettivi che ci siamo posti possiamo ritenere che la progettazione sia stata efficace, possiamo osservare dai testbench come la codifica di un indirizzo successivamente ad un’altra codifica (quindi avendo già in memoria le WZ) richieda molto meno tempo rispetto a dover caricare 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.