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](#_Toc34601746)

[Architettura 2](#_Toc34601747)

[2.1 Idle 2](#_Toc34601748)

[2.2 Ask 2](#_Toc34601749)

[2.3 Wait\_mem1 2](#_Toc34601750)

[2.4 Save 2](#_Toc34601751)

[2.5 Ask\_addr 2](#_Toc34601752)

[2.6 Wait\_mem2 2](#_Toc34601753)

[2.7 Analize 2](#_Toc34601754)

[2.8 Enc\_wrt 2](#_Toc34601755)

[2.9 Wait\_mem3 2](#_Toc34601756)

[2.10 Done 2](#_Toc34601757)

[2.11 Wait1 2](#_Toc34601758)

[Risultati sperimentali 4](#_Toc34601759)

[Testing 5](#_Toc34601760)

[Conclusione 6](#_Toc34601761)

# Introduzione

**[**nella prima parte dell’introduzione credo sia necessario una specie di abstract, da scrivere ad esempio come paragrafo 1.1 .

In questo abstract spieghiamo in modo sintetico in cosa consiste il progetto e ci allegherei due disegnini (come suggerito dal prof nelle regole) che, tramite due esempi concreti, mostrano l’addr prima, le wz, e come si trasforma l’addr in output).

Ad esempio, scriverei:

L’obiettivo del progetto è quello di verificare tramite un componente programmato in linguaggio VHDL (non so se abbia senso dire “componente etc etc “) la presenza o meno di un indirizzo “addr” di 7 bit all’interno di un 8 working zone “WZ”: se l’indirizzo è presente in una delle celle della working zone viene restituito invariato aggiuggendo uno ‘0’ come MSB ad “addr, altrimenti viene restituito un indirizzo codificato come mostrato in 1.2 (oppure glielo dico già qua, ma volevo essere sintetico in questo “abstract”).

1.2: spieghiamo meglio la codifica e allegherei i due disegnini che mostrano due esempi concreti nel caso di appartenenza o meno….**]**

**[**quello che hai scritto tu, lo farei seguire a quanto scritto sopra tipo in un nuovo paragrafo 1.3 “scelte progettuali” (il nome l’ho preso da altre relazione, quindi magari lo chiamiamo in un altro modo…), insomma come ci siamo approcciati al progetto.**]**

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 secondo il linguaggio VHDL **[**se scritto in 1.1, non c’è bisogno di ripetere “secondo il linguaggio VHDL”, ovviamente**]**. 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 scorporare**[**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.

Si rimane inermi in questo stato 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, da qui si transita in Save.

## 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 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; restiamo in questo stato finché i\_start è alto, quando i\_start viene portato basso allora si abbassa o\_done e si transita 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

La seguente è la macchina a stati del progetto che risolve la codifica delle WZ:

**[**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

# Risultati sperimentali

Non ho capito: dobbiamo metterci ad esempio la foto degli schematic pre e/o post sintesi; numero FF e LUT? Il clock minimo per cui funziona? Il tempo minimo massimo impiegato tra i vari test bench? HELP ☹

# Testing

Il componente [è giusto da dire?] è stato sottoposto a vari test bench (qui di seguito mostrati), comportandosi correttamente a tutti sia in pre-stintesi sia in post-sintesi.

[devo dirgli cosa si intende con correttamente secondo te??????

Inizialmente si è testato il componente con i due test bench d’esempio proposti insieme alla richiesta del progetto: uno dei due forniva in input una WZ e un addr presente in essa; l’altro forniva una WZ e un addr non presente in essa.

Si è poi deciso di generare tramite un programma scritto in linguaggio C [effettivamente abbiamo ancora il generatore, no?... gli scrivo di questo senza fare cenno ai 300k che secondo me lo mettono anche altri e altrimenti poi pensa magari che abbiamo copiato o ci siamo passati dei testbench…] un generatore di WZ e addr (indirizzi sia presenti e non nelle working zones), rispettando la specifica, e il relativo test bench scritto in VHDL che prende WZ e addr dal file creato dal generatore, così da verificare il corretto funzionamento del componente non solo per i due esempi sopra citati, bensì un numero consistente di casi.

Verificato che il componente è in grado di riconoscere se un addr è in WZ o meno, restituendo rispettivamente l’output corretto secondo la specifica, si è deciso di scrivere una serie di test bench che controllano il corretto funzionamento del componente [l’ho ripetuto mille volte…] in caso di start o reset forniti sia all’inizio della simulazione, sia alla fine di essa (cominciando così una nuova esecuzione).

In particolare, sono stati scritti 4 test bench sia per il caso di start doppio, sia per il reset doppio [corretto da dire?]: questi propongono i casi “inWZ-inWZ”, “inWZ-outWZ”, “outWZ-outWZ”, “outWZ-inWZ”, fornendo un diverso addr dopo il secondo start/reset e lasciando invariate le working zones.

[foto di un esempio per start e uno per reset]

Infine, si è scritto un test “finale” finalizzato al testare il corretto funzionamento del componente [yeee, milionesima volta che l’ho scritto] a seguito di una serie di reset forniti “casualmente” durante l’esecuzione, lasciando invariati addr e WZ.

[la foto mi sembra abbastanza esplicativa, metterei solo quella e basta, senza spiegare dove abbiamo messo i reset…. O magari sì dai, sotto la foto scriviamo dove abbiamo messo i reset, meglio prevenire 😉]

[foto]

# Conclusione

“max mezza pagina” dice il Palermo… che cazzo ci scrivo però? ☹