Prova finale (Progetto di Reti Logiche)

Luca Colasanti – Cod. Persona 10627030 – Matr. 891653
Stefano D'Angelo – Cod. Persona 10607664 – Matr. 890033

Corso di Reti Logiche
A.A. 2019 / 2020
Professore: William Fornaciari
Scaglione A - E

Indice

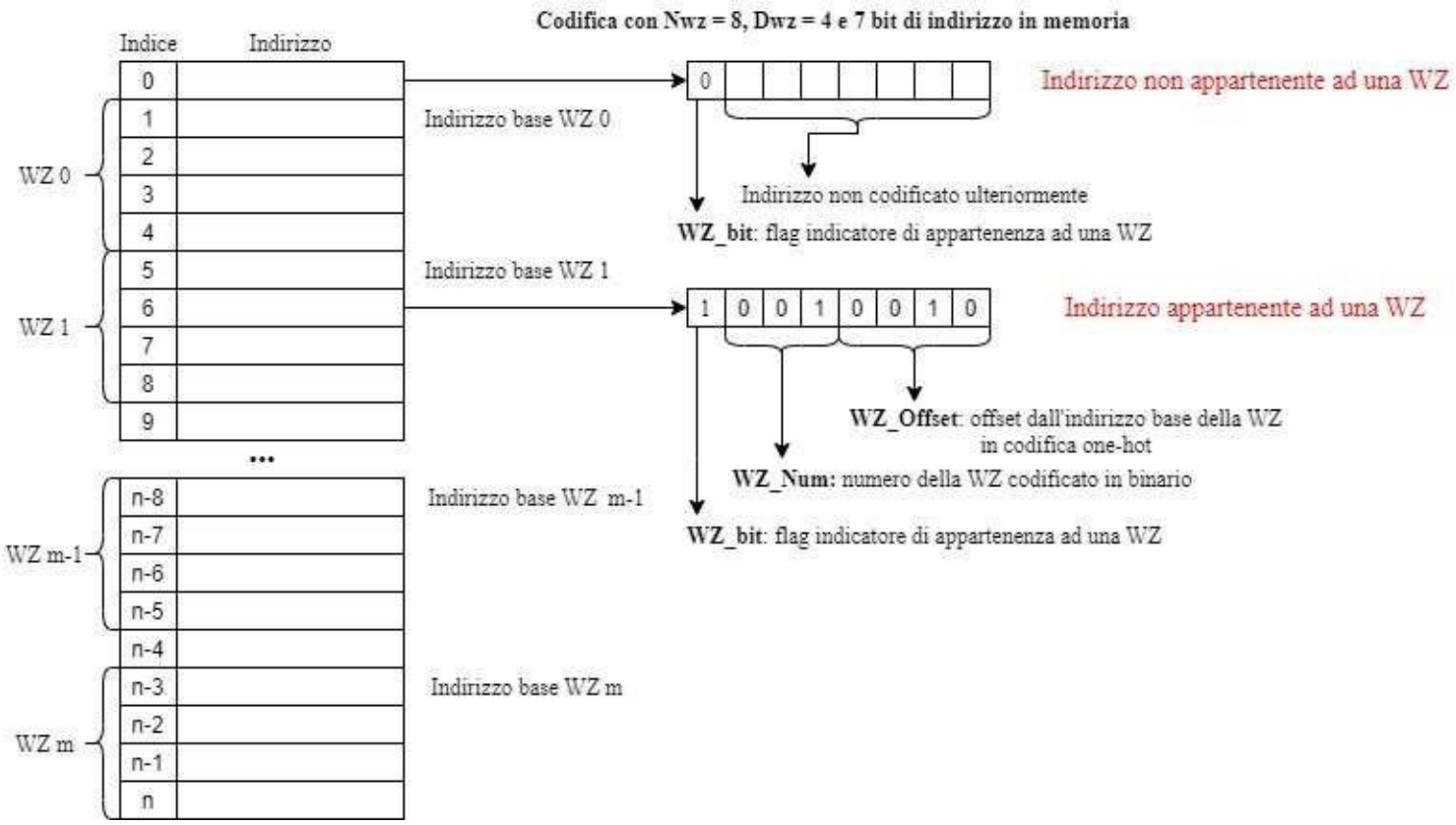
1 Introduzione

La prova di quest'anno richiede di progettare un componente HW che, interagendo con una memoria RAM, simuli il comportamento descritto nel metodo di codifica denominato *"Working Zone"*. Tale metodo è stato pensato con la finalità di ridurre il consumo energetico dovuto ai pin di input–output di un chip organizzando il bus indirizzi con il presupposto che in un dato istante di tempo il programma prediliga utilizzare solo una porzione dello spazio di indirizzamento a sua disposizione. A tal fine il bus trasmette solamente l'offset dall'indirizzo di memoria base della Working Zone, ovvero un intervallo di indirizzi contigui in memoria, assieme ad un identificatore di quest'ultima.

L'obiettivo perseguito durante lo studio e l'implementazione della soluzione è stato quello di mantenere il codice quanto più semplice da comprendere, vista la semplicità di parametri con cui ci si è trovati a lavorare. Naturalmente questo non va a discapito della capacità del programma di assolvere completamente alla richiesta formulata nelle specifiche, prevedendo opportuni accorgimenti relativi a possibili errori o casi particolari che possono incorrere con gli indirizzi.

Il modulo da progettare interagisce con una memoria RAM con indirizzamento a 16 bit, in cui vengono definite 8 Working Zone (NWZ = 8), ognuna di dimensione 4 bit (DWZ = 4).

Nello specifico la soluzione è stata elaborata come macchina a stati che salva in opportuni registri il risultato delle sottrazioni tra gli indirizzi base delle otto WZ e l'indirizzo da codificare per procedere poi a verificare se uno dei valori risultanti dalla differenza è minore della dimensione delle WZ; In caso affermativo l'indirizzo appartiene a quest'ultima WZ. A seconda del risultato dell'analisi precedente, si procede alla codifica dell'indirizzo in uscita in base alla sua appartenenza a una WZ o meno, come illustrato nell'immagine sottostante.

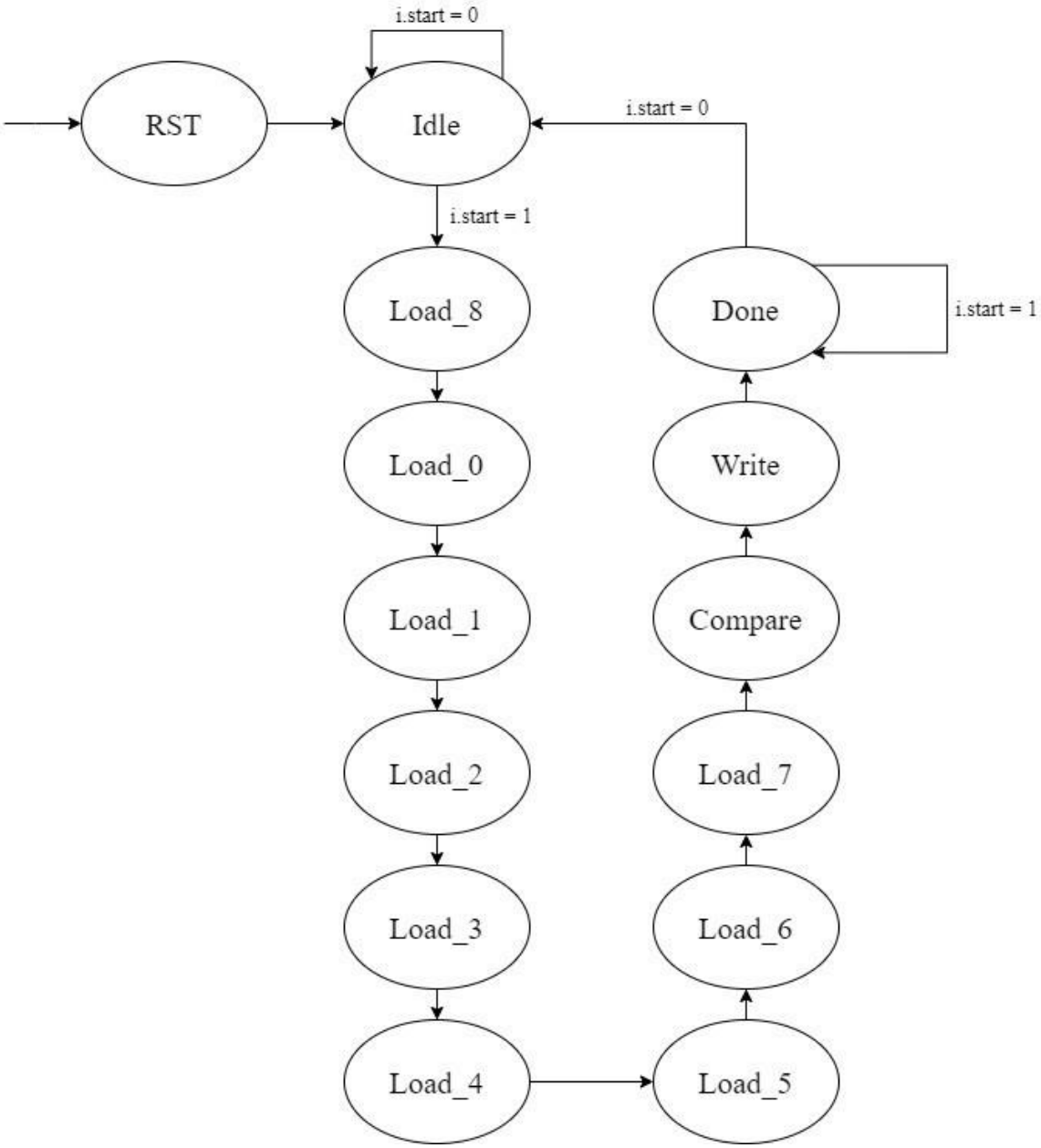


Da specifica gli indirizzi base contenuti in memoria, assieme all'indirizzo da codificare, sono definiti con 7 degli 8 bit a disposizione, tuttavia, dato l'utilizzo di sottrazioni per confrontare gli offset e valutare l'appartenenza di un indirizzo ad una WZ, abbiamo deciso di adottare segnali *std_logic_vector* di 8 bit anche per gli indirizzi base delle WZ. In questo modo non incorriamo nel rischio che un segnale da 7 bit venga male interpretato come un valore negativo secondo il complemento a 2.

Ulteriori accorgimenti hanno riguardato la strutturazione del codice secondo un paradigma stabile ed assodato quale quello descritto in "Introduzione al linguaggio VHDL: Aspetti teorici ed esempi di progettazione" del prof. Carlo Brandolese: l'utilizzo di tre processi con funzioni ben definite, analizzate nel dettaglio nella sezione 2, facilita la comprensione del codice e soprattutto evidenzia le singole aree del programma che si occupano di specifici aspetti, garantendo una maggiore facilità di manutenzione ed estensione.

Data la quantità limitata di dati e di indirizzi con cui risulta necessario operare, un'ulteriore scelta di sviluppo è stata l'adozione di un pattern di soluzione relativo specificamente al caso in analisi, come risulta evidente dal codice: preferenza d'uso di costrutti if-then-else per gestire la traduzione in one-hot degli offset ad esempio. Se da una parte questo rappresenta uno svantaggio nella necessità di dover riscrivere l'algoritmo per dati e input di dimensioni maggiori, il cuore della computazione resta il medesimo (questo aspetto sarà ripreso più nel dettaglio nella sezione "Ottimizzazioni") ed abbiamo ritenuto più consono prediligere una soluzione non appesantita dalle necessità e dai controlli richiesti da situazioni più generiche.

2 Architettura



La macchina a stati finiti si articola in vari stati di funzionamento: parte in stato di RST in modo da resettare i vari segnali in uscita a valori di default e si entra poi nel ciclo vero e proprio in cui si analizza l'appartenenza dell'indirizzo da codificare ad una WZ, partendo dallo stato IDLE. Analizziamo gli stati più in dettaglio:

- RST: è lo stato in cui la macchina si trova non appena viene avviata. In questo stato *o_address* ed *o_data* sono inizializzati a valori non significativi, e i segnali *o_en*, *o_we* ed *o_done* sono posti a zero. Non è necessario inizializzare i segnali temporanei ***addr0,…,addr8*** in quanto l'unico modo di modificare tali valori è quello di passare dagli stati LOAD0,…,LOAD8. I test effettuati verificano che quest'ultimo insieme di inizializzazioni sarebbe solo ridondante (vedi sezione 4). Dallo stato di RST si passa allo stato di IDLE dopo un ciclo di clock;

- IDLE: questo è lo stato in cui la macchina si trova nell'attesa di ricevere il segnale *i_start* = 1, che comincia il processo di codifica. Poiché si può giungere a questo stato anche quando abbiamo appena terminato la precedente esecuzione, impostiamo *o_done* = 0;
- LOAD_8: carichiamo in *o_address* la codifica in binario senza segno del numero 8 per richiedere in memoria la lettura dell'indirizzo da codificare. A tal fine alziamo *o_en* = 1 e salviamo in un vettore temporaneo *addr8* il valore letto in memoria;
- LOAD_0 - LOAD_1 - … - LOAD_7: in tutti questi stati si carica in *o_address* la codifica in binario senza segno del numero della LOAD (che indica il numero della WZ) e si procede a caricare negli opportuni vettori temporanei *addr0,…,addr7* la differenza tra l'indirizzo da codificare contenuto in *addr8* e *i_data*, ovvero il valore letto dalla memoria sulla base di *o_address*. Naturalmente si avrà *o_en* = 1 e *o_we* = 0 ad ogni stato e si procede nella lettura in ordine crescente di indirizzi di memoria;
- COMPARE: in questo stato si verifica a cascata se qualcuno dei valori tra *addr0* e *addr7* è minore o uguale a 3 (in binario), ovvero se l'indirizzo da codificare appartiene alla WZ corrispondente. Qualora l'indirizzo appartenga alla WZ n con offset dal suo indirizzo di base m (<4) si procede a codificare *o_data* con la modalità descritta nell'overview del programma (1 & WZ_Num & WZ_Off). Se invece l'indirizzo non appartiene a nessuna WZ, *o_data* non viene modificato e rispecchia la codifica per indirizzi esterni alla WZ, ovvero viene restituito *addr8* così com'è (abbiamo assunto che il bit più significativo di *addr8* sia sempre a 0, in quanto i valori delle WZ vanno da 0 a 127, essendo la codifica basata su 7 bit più un flag aggiuntivo);
- WRITE: si carica in *o_address* la codifica in binario del numero 9, che sarebbe la posizione in memoria in cui caricare l'indirizzo codificato e si pongono *o_en* = 1 e *o_we* = 1 per procedere alla scrittura del valore elaborato;
- DONE: poiché la computazione è terminata alziamo il segnale *o_done* e, appena *i_start* torna a 0, ci portiamo nuovamente in IDLE in attesa di cominciare una nuova computazione.

Per gestire l'eventualità di segnali di reset in ogni momento della computazione, da qualsiasi stato si può passare allo stato RST qualora *i_rst* = 1, in modo da resettare gli indirizzi base delle WZ e poi riprendere la computazione. Si sottintende quindi la presenza di un arco che da ogni stato porta in RST al soddisfacimento della condizione sopra citata.

Tale schema risolutivo agisce sulla premessa, espressa nella specifica, che le WZ siano sempre fornite non sovrapposte, di valore significativo e conformi alle indicazioni, per cui non ci si preoccupa di verificare la correttezza dei dati.

La macchina si snoda nella computazione attraverso l'utilizzo di tre process:

- State: si occupa di scandire l'avanzamento degli stati della macchina in funzione del ciclo di clock (si lavora sul fronte di discesa come indicato dalla condizione *falling_edge(i_clk)* all'interno dello statement if) o di eventuali segnali di *i_rst* = 1 asincroni;
- Delta: si occupa dei passaggi da uno stato al successivo a seconda dei segnali di clock e altri segnali d'ingresso, qualora siano presenti condizioni particolari;
- Lambda: gestisce le operazioni di lettura, manipolazione e confronto di segnali temporanei e dati, assieme alla produzione dell'output.

3 Risultati sperimentali

Dal report di sintesi possiamo ricavare una visione generale della performance ed efficienza del programma elaborato in termini prevalentemente di area occupata, vista la non necessità di effettuare l'analisi di timing.

Come già largamente discusso il programma non si propone di eccellere in termini di occupazione d'area, ma ha il focus di essere di quanto più possibile leggibile ed efficiente in lettura. In ogni caso

è possibile ricavare che la soluzione circuitale di Vivado fa uso di 58 Look-Up Tables (LUT) e 14 Flip-Flop (FF). Da una breve analisi ricaviamo che l'occupazione della FPGA xc7a200tfbg484-1 corrisponde a meno di 1‰ per le LUT ed una percentuale ancor più bassa per i FF. Un tale indice di occupazione evidenzia come la semplicità dell'applicazione sia rispecchiata nella minuscola occupazione della scheda ed anche un uso minimale di FF, dato dall'unica necessità di memorizzare gli indirizzi base delle WZ assieme a poche altre informazioni temporanee.

Si segnala oltretutto la risoluzione della maggior parte degli errori che emergono in post-sintesi, fatta eccezione per alcuni warning relativi ad Inferring Latches. A tal proposito il problema è stato sottoposto all'attenzione di un tutor: il suo consiglio è stato di inizializzare in ogni stato il valore di ogni segnale, il che ha aumentato di conseguenza il numero di LUT utilizzate oltre il centinaio. Nonostante la modifica permaneva un warning di Inferring Latch relativo a *o_address*, che tuttavia non è reinizializzabile a fine di ogni stato poiché utilizzato nelle operazioni di lettura e scrittura. Una sua modifica di fatto generava numerosi altri warning e non dava più esito positivo ai test. Di conseguenza abbiamo optato di annullare le modifiche attuate su consiglio del tutor e mantenere il codice come elaborato inizialmente, poiché la risoluzione dei test non risultava intaccata dai warning presenti. Altri warning di minore importanza che si sono presentati sono "Sequential Element Unused" e "No Constraints selected for write", che a seguito di una ricerca su Internet sono risultati essere dei bug relativi a Vivado (vedi sezione 7).

Per la progettazione della soluzione si è fatto prevalentemente utilizzo dell'applicazione "Vivado 2019.2 WebPACK Edition" senza particolari settaggi o ottimizzazioni.

4 Simulazioni

Oltre ai testbench forniti (no WZ e con WZ), il programma è stato ampiamente sottoposto sia a test generati casualmente che a test relativi a specifiche situazioni critiche per il funzionamento. Per la prima tipologia segnaliamo l'utilizzo di un programma Java elaborato per la generazione di test, di cui riportiamo con maggiore dettaglio i casi più specifici e di maggiore interesse:

- WZ adiacenti: tale test presuppone di verificare l'efficacia del programma nel riconoscimento corretto dei limiti delle WZ e verificare quindi l'eventuale presenza di errori nel ramo if che analizza a quale WZ appartiene l'indirizzo da codificare (Lambda process). Tale test è utile anche nella verifica che l'operazione di sottrazione in signed tra indirizzo da codificare e indirizzi base della WZ funzioni opportunamente. Rappresenta uno dei punti fondamentali del programma, vista la mancanza di WZ sovrapposte o scorrette da specifica;
- WZ agli estremi della memoria: rappresenta un caso limite di particolare rilievo e va a stimolare nello specifico il meccanismo di riconoscimento della WZ di appartenenza (presente nel process Lambda). Si rivela fondamentale per verificare che il programma funzioni perfettamente a fronte di indirizzi codificati su 7 bit (come da specifica);
- i_rst alzato nel mezzo della computazione: non si è voluta escludere nella programmazione della soluzione l'eventualità di reset asincroni e questo test pone sotto esame proprio tale funzionalità del programma. Come già espresso nella sezione 2 in questi casi il programma deve portarsi nello stato di reset, riportare a valori non significativi *o_data*, *o_address* e azzerare i segnali di *o_en*, *o_we* e *o_done* per poi tornare nello stato di IDLE ed essere pronto a ricominciare la computazione;
- Molteplici istanze di i_rst =1 in una singola computazione: sull'onda del precedente test è rilevante anche testare il comportamento di stress dovuto a un segnale di reset ripetuto. Come da progetto, il programma segue a riportarsi ad ogni istanza nello stato di reset senza generare valori non significativi o computare erroneamente. L'elaborazione dell'informazione non viene terminata qualora il reset la interrompa prima della conclusione;
- Test multi-start: test di rilievo che stressa la robustezza del programma di fronte all'eventualità di due segnali di start (due computazioni differenti) agenti su contenuti di RAM

differenti, ma senza la presenza di un reset tra le due. Il programma vuole far sì che l'acquisizione di dati dalla memoria avvenga correttamente. È una criticità di minore rilevanza nel nostro caso visto che l'acquisizione delle WZ avviene ad ogni computazione.

5 Ottimizzazioni

Il progetto ha cercato di perseguire la maggior leggibilità e comprensione ottenibile del codice, tuttavia sono diversi gli obiettivi che si sarebbero potuti scegliere in alternativa: un focus maggiore su velocità di computazione, uso dell'area o simili. Le ottimizzazioni applicate di particolare rilievo sono:

- La decisione di memorizzare non gli indirizzi base delle WZ, ma direttamente la differenza tra l'indirizzo da codificare ed esse stesse, in modo da evitare di dover provvedere a quest'operazione in un secondo momento a seguito della memorizzazione degli indirizzi di base delle WZ per intero;
- L'utilizzo di un pattern basato sull'utilizzo di tre differenti process per l'esecuzione del programma, ai fini di una maggiore leggibilità e pulizia del codice.

Segnaliamo tra le altre anche diverse proposte di ottimizzazione che perseguono obiettivi differenti:

- Al fine di ottimizzare l'utilizzo di registri e componenti della scheda programmabile la soluzione può prevedere di non conservare staticamente gli indirizzi base delle WZ, ma limitarsi ad estrarli e mantenerli solamente per il ciclo in cui si provvede alle operazioni di confronto con l'indirizzo da codificare. Il guadagno che si riscontra può essere in termini di "silicio" e di espansione del codice, mantenendo prestazioni equiparabili;
- Si può cercare di rendere il codice più generale in modo da favorire una sua futura espansione, incorrendo sicuramente in un generale aumento dei tempi di computazione e di occupazione della memoria, vista la necessità di implementare maggiori controlli e segnali sufficientemente grandi;

6 Conclusioni

L'attività di progetto è stata svolta attraverso una stretta collaborazione di entrambi i membri del gruppo, nello specifico nella decisione delle caratteristiche della soluzione, ma anche dal punto di vista della revisione e ricerca di ottimizzazioni sia nel programma che nella stesura della relazione. Seppur in dimensione limitata tale attività ha affinato lo spirito di collaborazione e la capacità di lavorare in squadra, oltre ad aver rappresentato un'adeguata applicazione delle nozioni assimilate nel corso di Reti Logiche. La necessità di cimentarsi nello studio dei rudimenti di un nuovo linguaggio, quale il VHDL, ha rappresentato inoltre un ulteriore orizzonte di scoperta e uno stimolo nell'approfondire in futuro ulteriori applicazioni.

Il programma elaborato propone una soluzione quanto più completa rispetto alle specifiche fornite e cerca di raggiungere tale fine mantenendo una struttura del codice pulita, ma soprattutto veloce ed efficiente, cercando di non perdersi in controlli o passaggi facilmente evitabili. Seppur non votato all'ottimizzazione della memoria piuttosto che del tempo di computazione, il compromesso raggiunto risulta essere adeguato in entrambe le specifiche senza diventare troppo oneroso.

7 References

https://forums.xilinx.com/t5/Implementation/Vivado-warning-Constraints-18-5210-No-constraints-selected-for/td-p/930337

https://forums.xilinx.com/t5/Synthesis/Sequential-element-is-unused-and-will-be-removed-from-module/td-p/537437