

Progetto di Reti Logiche

Cristian Sbrolli

Matricola: 887811

Anno Accademico 2019/2020

Prof. Gianluca Palermo

Esercitatore Antonio Rosario Miele

**INTRODUZIONE**

Il componente HW da realizzare ha lo scopo di effettuare la codifica di un indirizzo, in base alla sua appartenenza o meno ad alcuni insiemi di indirizzi noti (Working Zones, ognuna di dimensione 4 per questo progetto). Il processo di codifica, a basso livello di dettagli, è illustrato in figura:

La seconda fase mi ha portato a compiere delle scelte, in quanto le WZ possono essere salvate o meno in dei registri. Salvandole, si migliora l’efficienza del componente in codifiche successive nella stessa esecuzione, dato che le WZ non cambiano a meno del reset, rimettendoci però in area, dato che il salvarle richiede almeno 64 FF aggiuntivi (8 bit per ogni WZ). Non salvandole si migliora l’area occupata dal componente, quindi guadagnando anche in fattore economica, e si guadagna anche in scalabilità (sul numero di WZ), peggiorando però le prestazioni su codifiche successive nella stessa esecuzione. Nella mia implementazione ho scelto la seconda opzione, questo anche perché, per le mie competenze, risulta complicato confrontare l’efficienza temporale delle due versioni menzionate, mentre è più immediato verificarne l’area che occupano in termini di FF e LUT. Quindi ho svolto il progetto cercando di ottimizzare il componente in modo da occupare pochi componenti base, comunque ottimizzando al massimo, nei limiti della scelta implementativa, anche i di cicli di clock impiegati per la codifica. Scegliendo quindi di non salvare le WZ in memoria, il processo sopra figurato si può vedere più dettagliatamente:

Codifica indirizzo appartenente a una WZ

Si

Fa parte della WZ?

Lettura indirizzo WZ

Lettura indirizzo da codificare

No

Codifica indirizzo non appartenente a nessuna WZ

Si

Tutte le WZ checkate?

No

**ARCHITETTURA**

**E SCELTE PROGETTUALI**

Ho scelto di descrivere il componente con un unico modulo monoprocesso, sensibile al clock e al segnale di reset, che descrive una macchina a stati, e di usare principalmente variabili invece che segnali, così da rendere il codice più semplice (da capire, ma anche per debuggare e ottimizzare), migliorandone la leggibilità. La macchina a stati implementata nel codice è descritta in seguito:

Reset\*

GETIND

WAITIND

WAITSTART

DONE

CHECKWZ

\*Il segnale di RESET porta la FSM nello stato iniziale WAITSTART, indipendentemente dallo stato attuale, si potrebbe inserire una freccia da ogni stato che va in WAITSTART, ma ho evitato per mantenere il grafo leggibile.

**Descrizione degli stati:**

**WAITSTART:** è lo stato di partenza dopo il reset, si occupa di inizializzare alcune variabili e di settare i segnali per richiedere alla memoria l’indirizzo da codificare.

**WAITIND:** visto che il dato richiesto alla memoria sarà pronto da leggere solo dopo due cicli di clock, questo stato funge da “stato di attesa”. In realtà in questo stato viene già richiesto il prossimo indirizzo alla memoria, ossia l’indirizzo della prima WZ, e similmente verrà fatto nel prossimo stato (vedere la successiva parte relativa alle ottimizzazioni).

**GETIND:** in questo stato viene letto il valore dell’indirizzo da codificare.

**CHECKWZ:** in questo stato si analizza l’appartenenza dell’indirizzo ad una WZ, sottraendo l’indirizzo base della WZ all’indirizzo da codificare, controllando che il risultato sia minore o uguale a 3. Questo è possibile perché la sottrazione è effettuata tra unsigned, quindi anche in caso di risultato che avrebbe segno negativo, leggendolo come unsigned si ottiene un valore in modulo sicuramente più grande di 3, in quanto si ottiene un valore binario con almeno il bit più significativo ad 1. È da notare che questo metodo non crea problemi rappresentando il risultato su 8 bit, in quanto minuendo e sottraendo sono entrambi codificati su 7 bit, anche se rappresentati su 8. In caso di riconoscimento dell’appartenenza alla WZ, viene codificato l’indirizzo come appartenente alla WZ analizzata (usando il risultato della sottrazione per calcolare l’offset rispetto all’indirizzo base dalla WZ), alzato il WE della memoria, e la macchina passa allo stato DONE; altrimenti viene controllato se ci sono ancora WZ da analizzare: in caso positivo, lo stato resta invariato e viene aggiornato l’indirizzo della prossima WZ che verrà analizzata, in caso negativo viene codificato l’indirizzo come non appartenente a nessuna WZ ( 0 & indirizzo stesso) e la macchina passa allo stato DONE.

**DONE:** in questo stato viene alzato il segnale o\_done, poi la macchina resta su questo stato finché non viene abbassato il segnale di start.

**OTTIMIZZAZZIONI EFFETTUATE**

Inizialmente, per gestire la lettura in memoria, che richiede un ciclo di clock di attesa per avere il dato pronto da leggere, avevo usato uno stato di attesa al quale si passava dopo ogni richiesta di lettura alla memoria, perdendo così un ciclo di clock per lettura. Successivamente ho notato che in quel ciclo di clock di attesa si può già richiedere alla memoria un altro indirizzo, e ripetendo questa strategia, si può fare in modo di perdere solo un ciclo di clock per la prima lettura richiesta, mentre da quella in poi, ogni ciclo di clock si può leggere un dato, avendolo richiesto due cicli prima. In questo modo, nello stato di WAITIND già viene richiesto l’indirizzo base della prima WZ, e allo stesso modo nello stato di GETIND viene richiesto l’indirizzo base della seconda WZ, poi nello stato di CHECKWZ viene richiesto un indirizzo per ciclo. Questo mi ha permesso di ridurre a quasi la metà i cicli di clock necessari per una conversione.

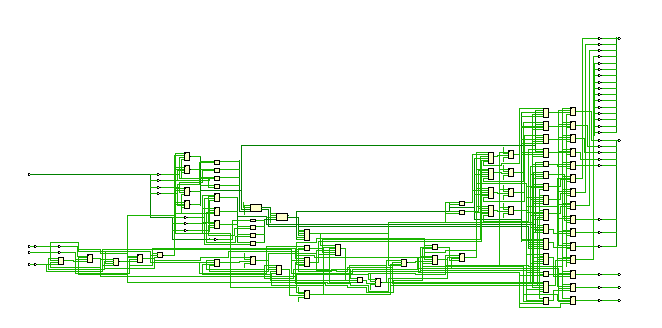
Un'altra ottimizzazione è stata l’eliminazione di uno stato che aveva la funzione di codifica, in quanto richiedeva un flag per codificare l’indirizzo come appartenente a una WZ o meno. Spostando la codifica all’interno dello stato di CHECKWZ è stato possibile eliminare quel flag e lo stato di codifica, riducendo ulteriormente l’utilizzo di entrambi LUT e FF.

**RISULTATI SPERIMENTALI**

**SINTESI**

La sintesi è stata eseguita senza warning o errori rilevanti (solo il warning sui constrainsts), in seguito è riportato l’utilizzo di risorse e la schematic risultante:

****

****