

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 56 FF aggiuntivi (7 bit per ogni WZ, il primo bit è sempre 0 da specifica). Non salvandole si migliora l’area occupata dal componente, quindi guadagnando anche sul costo della soluzione, e si guadagna anche in scalabilità (all’aumentare del numero di WZ l’altra soluzione aumenta linearmente il numero di FF richiesti), 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 in un contesto realistico, dove entra in gioco anche il tempo di elaborazione di altri componenti interagenti col nostro, mentre è più immediato verificare l’area occupata dal componente 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 da debuggare e ottimizzare), migliorandone la leggibilità. La macchina a stati implementata nel codice è descritta in seguito:

Reset\*

GETIND

CHECKWZ

WAITSTART

DONE

\*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 le variabili e di settare i segnali per richiedere alla memoria l’indirizzo da codificare.

**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 quindi che il risultato sia minore o uguale a 3. In caso di riconoscimento dell’appartenenza alla WZ, viene codificato l’indirizzo come appartenente alla WZ analizzata in quel ciclo di clock (usando il risultato della sottrazione per calcolare l’offset rispetto all’indirizzo base dalla WZ), viene alzato il WE della memoria, e la macchina passa allo stato DONE; altrimenti, se l’indirizzo non appartiene alla WZ, viene controllato se ci sono ancora WZ da analizzare: in caso positivo, lo stato resta invariato e viene aggiornato l’indirizzo della 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 E SCELTE EFFETTUATE**

Inizialmente, avevo scelto di lavorare sul fronte di salita del clock, e questo mi portava a dover aspettare un ciclo di clock in più per la lettura di dati dalla RAM, dovuto al ritardo di 1 ns che essa introduce nel restituire il dato richiesto. Successivamente ho deciso di lavorare sul fronte di discesa del clock, il che, dato il minimo periodo di 100ns, garantisce la lettura del dato al ciclo successivo rispetto al ciclo in cui esso è stato richiesto. Questo mi ha permesso di eliminare uno stato che usavo per attendere il dato, così riducendo notevolmente il numero di cicli richiesti per la codifica e anche il numero di FF ed LUT.

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.

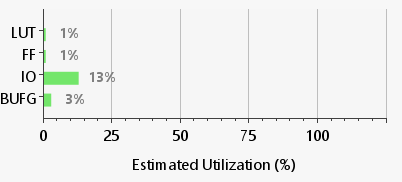
Un’ottimizzazione importante riguarda la verifica dell’appartenenza dell’indirizzo alla WZ nello stato CHECKWZ: inizialmente, prima di sottrarre, controllavo che l’indirizzo fosse maggiore dell’indirizzo base della WZ, per evitare sottrazioni con risultato negativo. In realtà, ai fini del controllo successivo, ossia che il risultato sia minore o uguale a 3, è corretto anche non effettuare il confronto tra i due operandi, in quanto, essendo la sottrazione effettuata in binario tra unsigned, nel caso di risultato negativo si otterrà (leggendo come unsigned) un valore sicuramente maggiore di 3, per via del bit più significativo sicuramente ad 1. Inoltre, essendo la sottrazione rappresentata su 8 bit, ma tra valori effettivamente a 7 bit (anche se rappresentati su 8), non si hanno problemi di overflow che possono far perdere il bit più significativo ad 1, il che comprometterebbe il funzionamento del metodo.

**RISULTATI SPERIMENTALI**

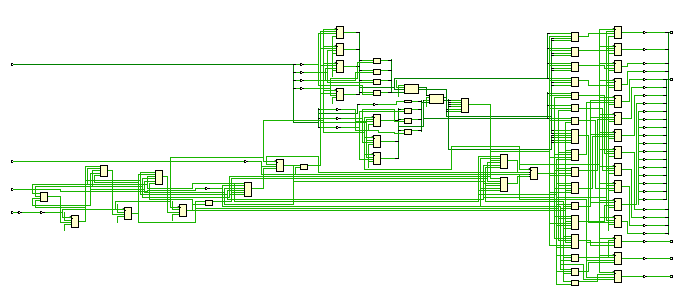
**SINTESI**

La sintesi è stata eseguita senza warning o errori rilevanti, in seguito è riportato l’utilizzo di risorse:

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource** | **Estimation** | **Available** | **Utilization %** |
| LUT | 28 | 134600 | 0,02 |
| FF | 27 | 269200 | 0,01 |
| IO | 38 | 285 | 13,33 |
| BUFG | 1 | 32 | 3,13 |

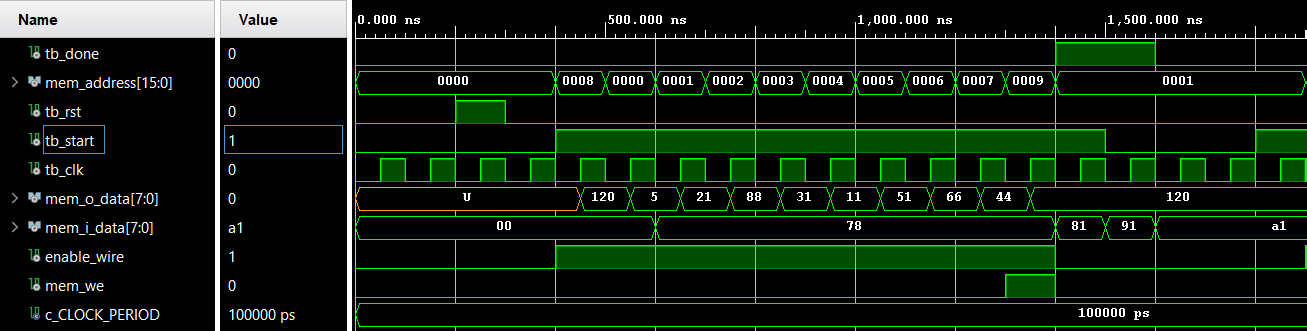
****

Come accennato nell’introduzione, ho svolto il progetto cercando di ridurre il più possibile le risorse utilizzate, ottenendo buoni risultati, con un basso utilizzo di LUT e FF. In seguito, è riportata la schematics del componente sintetizzato:

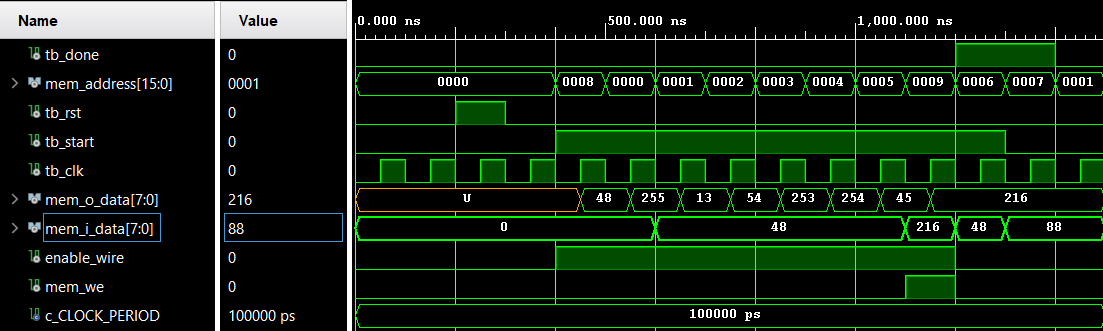


**SIMULAZIONI**

In seguito, sono analizzate due simulazioni, una in cui l’indirizzo non appartiene a nessuna WZ, e una un cui invece appartiene a una WZ. Per il primo caso, è riportata una delle simulazioni effettuate:



Si può vedere che per la codifica, in questo caso, sono necessari 10 cicli di clock da quando **start** viene alzato a quando **done** viene alzato, che è il numero minimo di cicli possibili per un’implementazione come questa, in cui non vengono salvate le WZ. Come già menzionato, è qui evidente il funzionamento del componente sul fronte di discesa del clock, che permette di leggere correttamente il dato richiesto nel ciclo precedente alla memoria. Invece nel caso in cui l’indirizzo appartiene a una WZ si ha:



Come si può vedere, da quando **start** va alto a quando viene alzato **done** il numero di cicli di clock necessari per codificare nel caso di appartenenza a una WZ sono N+2, dove N è il numero della WZ in ordine di analisi (la prima è all’indirizzo 0, l’ottava all’indirizzo 7). I due cicli aggiuntivi sono relativi agli stati di START e DONE.