Progetto finale di Reti Logiche

Weger Marco - Matricola n° 888201

Anno Accademico 2019/2020

Contenuto

1 Introduzione

Per questo progetto mi sono posto l'obiettivo di descrivere un componente che rispetti le specifiche sia in pre sintesi che in post sintesi. Ho voluto scrivere del codice di facile lettura e che si adatti in modo semplice e rapido a qualsiasi tipo di modifica del pattern e/o della dimensione della memoria e del suo contenuto (più dettagli verranno forniti in seguito). Per quanto riguarda la frequenza di clock non sono andato alla ricerca di una massimizzazione in quanto la specifica fissa il periodo di clock a 100 ns.

1.1 Obiettivi aggiuntivi

Dopo essermi assicurato di rispettate le richieste della specifica fornita mi sono posto i seguenti obiettivi:

1. Minimizzare il tempo trascorso dal momento che il segnale di start viene ricevuto al momento di invio del segnale di done;
2. Disattivare il segnale di enable della memoria tra le varie esecuzioni;
3. Descrivere un componente in grado di funzionare anche nel caso ci fossero reset asincroni durante l'esecuzione di una codifica;
4. Rendere il componente adattabile ad un'eventuale modifica della lunghezza dell'indirizzo della cella di memoria tramite una costante;
5. Rendere il componente adattabile ad un'eventuale modifica della dimensione di una singola cella di memoria tramite una costante (ADDR);
6. Rendere il componente adattabile ad un'eventuale modifica del numero di elementi in una working-zone tramite una costante (WZ_OFFSET);
7. Rendere il componente adattabile ad un'eventuale modifica del numero di working-zone tramite una costante (WZ_NUM).

Al fine di raggiungere i sopracitati obiettivi ho assunto che l'indirizzo da codificare e l'indirizzo codificato vengano sempre salvati in successione in celle immediatamente consecutive all'ultimo indirizzo di working-zone (ad esempio se ci fossero 16 working-zone, RAM(16) conterrebbe l'indirizzo da codificare e RAM(17) l'indirizzo codificato).

Tutte le ottimizzazioni descritte in seguito sono state valutate sulla base dei dati forniti dalla specifica e non tengono conto dell'eventuale crescita sproposita delle costanti sopracitate.

1.2 Obiettivo velocità

La limitazione più stringente in termini di velocità è data dall'accesso alla RAM e dai suoi ritardi. Al fine di minimizzare i tempi di letture e confronti ho optato per un componente che al momento della lettura di una cella imposti

contemporaneamente la successiva richiesta. In questo modo posso garantire l'esecuzione dei confronti tra l'indirizzo e le N working-zone in N+1 cicli di clock. Occorreranno poi 2 cicli di clock per la scrittura del dato codificato in memoria e la notificazione di elaborazione completata. Un ulteriorie ciclo di clock mi è servito a garantire che il segnale *o_en* non rimanga attivo tra un'esecuzione e la successiva. Ricapitolando, nel caso pessimo in cui l'indirizzo letto non si trova in nessuna delle work-zone:

$$T_{\text{esecuzione}} = (N + 1) \cdot T_{\text{clock}} + 3 \cdot T_{\text{clock}}$$

Nella valutazione del tempo di esecuzione va preso in considerazione il fatto che il segnale di start potrebbe arrivare sul fronte di discesa del clock, qualora succedesse l'esecuzione ritarderà di $T_{\text{clock}}/2$. Nella specifica forntia l'esecuzione massima dovrà durare 1200 ns (nel caso in cui *i_start* fosse allineato al fronte di salita del clock), 1250 ns altrimenti.

1.3 Obiettivo codice semplice e riutilizzabile

Ho deciso di realizzare una macchina a stati finiti al fine di rendere il codice facilmente comprensibile ed eventualmente modificabile, anche parzialmente. Tutti i segnali sono gestiti da un'unica entity ad eccezione di un contatore generico utilizzato per scandire le working-zone. Le seguenti costanti garantiscono un alto livello di riutilizzabilità (tra parentesi i valori assegnati da specifica):

- **SIZE_MEM** (16): dimensione dell'indirizzo di memoria;
- **SIZE_ADDR** (8): dimensione di una cella di memoria;
- **SIZE_WZ** (4): estensione della singola working-zone;
- **COUNT_WZ** (3): dimensione del vettore *WZ_NUM*;
- **N_WZ** (8): numero di working-zone.

Viene naturale notare la relazione tra *COUNT_WZ* e *N_WZ*, nonostante ciò ho voluto esplicatre costanti differenti in modo da far fronte a situazioni in cui il numero di working-zone non è potenza di 2.

1.4 Funzionamento in sintesi

Una soluzione che memorizza tramite registri i valori delle working-zone avrebbe migliorato i tempi di esecuzione compromettendo l'adattabilità del componente a situazione con un maggior numero di working-zone e/o una RAM più grande; pertando ho optato per scandire la memoria a ogni esecuzione. La singola esecuzione di una codifica può essere descritta attraverso un numero finito di step (che poi diventeranno una macchina a stati finiti):

1. Reset ed attesa del segnale di start (*i_start=1*);

2. Abilitazione della memoria e richiesta dell'indirizzo da codificare (salvato in un registro);
3. Richiesta della i-esima working-zone e confronto con l'indirizzo salvato, eventuale codifica e passaggio a step successivo (passo ripetuto per i compreso tra 0 e il numero di working-zone);
4. Scrittura dell'indirizzo codificato in memoria;
5. Invio segnale di elaborazione completata (*o_done=1*) e attesa feedback (*i_start=0*), il dato è disponibile fin dal momento in cui *o_done* viene portato a 1;

1.5 Note aggiuntive sulla specifica

Non sono stati considerati casi di working-zone sovrapposte in modo parziale o totale. Il componente scandisce in modo sequenziale le working-zone quindi viene identifica come valida quella nell'indirizzo RAM più basso (la scansione parte dall'indirizzo 0). Per la sintesi è stata scelta l'FPGA xc7a200tfbg484-1.

2 Architettura

2.1 Macchina a stati finiti

Il funzionamento del componente è scandito dalla macchina a stati finiti in figura 1, la funzionalità di ogni singolo stato è descritta nella tabella 1. La macchina è sincronizzata con il segnale *i_clk* e sensibile al segnale *i_start*.

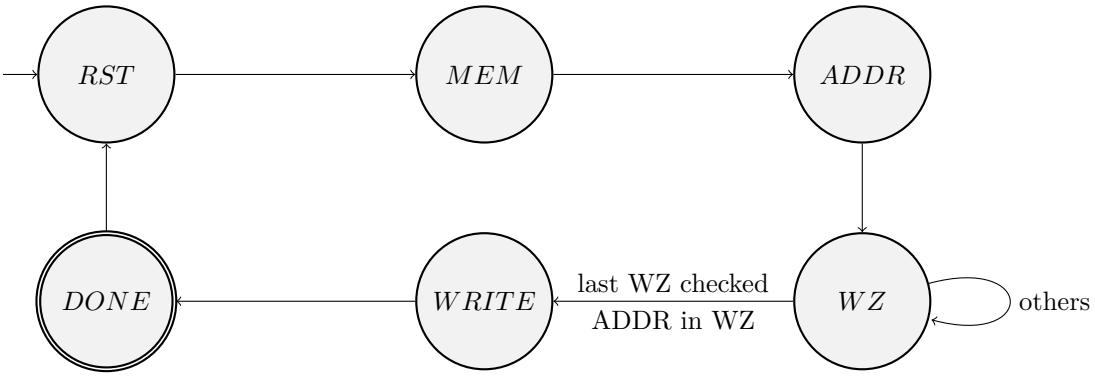


Figura 1: macchina a stati finiti implementata.

Stato	Descrizione
RST	Stato di reset della macchina. Da qui partono eventuali rieseccuzioni senza reset. Tutti i segnali vengono settati al loro valore di default (compreso *o_done* che potrebbe trovarsi alto) e il componente si prepara a leggere il l'indirizzo da codificare. La memoria è disattivata.
MEM	Utilizzato esclusivamente per abilitare la memoria prima di iniziare le letture.
ADDR	L'indirzzo da codificare viene prelevato e salvato in un apposito registro. Il contatore che scandisce le working-zone viene resettato e la memoria viene preparata per la lettura della prima working-zone.
WZ	Unico stato ciciclico della macchina. La condizione di uscita dal ciclo è il raggiungimento dell'ultima working-zone (caso in cui l'indirizzo non subirà modifiche) oppure l'identificazione della working-zone di appartenza (indirizzo da codificare). Nell'ultima esecuzione dello stato la memoria viene preparata per la scrittura del dato (codificato o meno).
WRITE	Il segnale *o_we* viene alzato per permettere alla memoria di prelevare il dato (*o_we*).
DONE	Viene notificato il termine dell'esecuzione tramite *o_done* per poi tornare nello stato RST in attesa di una nuova esecuzione.

Tabella 1: stati della macchina a stati finiti implementata.

2.2 Schema funzionale

Analizzando i componenti più significativi dello schema prodotto da Vivado si possono notare le seguenti informazioni:

- il segnale *o_data* (il valore finale dell'elaborazione) è derivato da 3 multiplexer:
  - **next_addr_i**: discrimina l'appartenza o meno alla working-zone;
  - **next_addr0_i**: discrimina l'eventuale offset nella working-zone;
  - **current_addr_i**: è comandato dall'enable della memoria;
- i segnali a singolo bit *o_done*, *o_en* e *o_we* sono gestiti dagli omonimi flip flop;
- la macchina a stati finiti è gestita da un registro a 3 bit (**current_state_reg**);
- l'indirizzo da codificare è memorizzato in un registro di 7 bit (**current_addr_reg**);
- tutti i flip flop utilizzati sono di tipo D;
- il multiplexer che governa il segnale *o_address* è gestito da segnali costanti ad eccezione dei bit che ne scandiscono i valori da 0 a 9 (**o_address_i___1**).

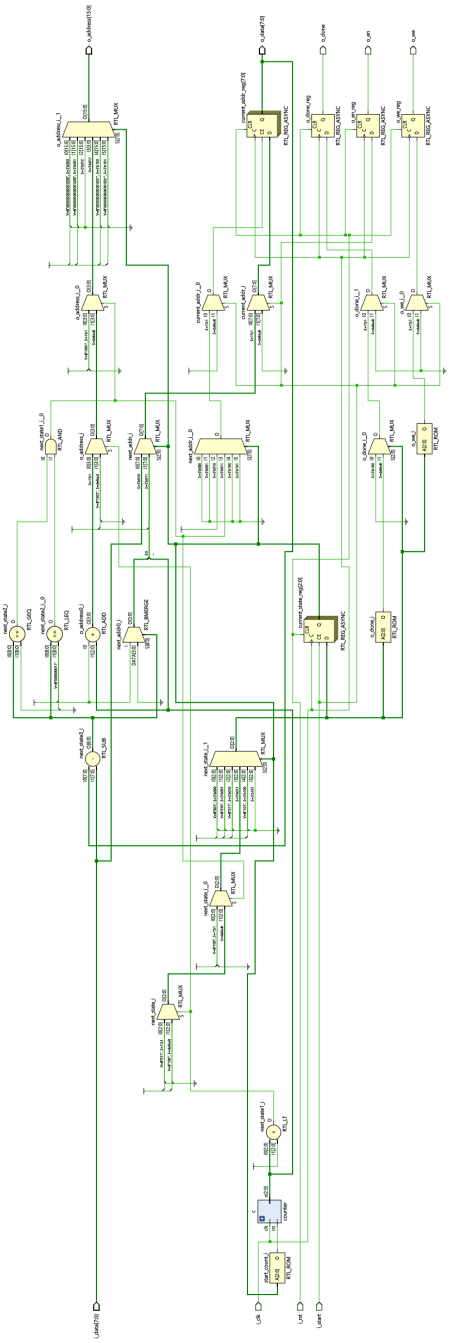


Figura 2: schema funzionale completo prodotto da Vivado.

3 Sintesi

3.1 Registri sintetizzati

Analizzando il report di sintesi e lo schema prodotto si può notare che gli stati della macchina sono codificati con la notazione "one hot" al fine di velocizzare le transizioni. Di seguito sono ricapitolati i registri sintetizzati.

N° bit	N° registri	Modulo	Contenuto
8	1	main	Salvataggio dell'indirizzo da confrontare con le working-zone.
1	3	main	Gestione di segnali a singolo bit: *o_done*, *o_en*, *o_we*.
6	1	main	Codifica lo stato corrente della macchina secondo la notazione "one hot".
3	1	counter	Contatore a 3 bit usato per scandire le working-zone.

Tabella 2: registri sintetizzati e loro contenuto.

3.2 Area occupata

Tramite le funzionalità di reportistica di VIVADO si possono analizzare il numero di LUT e di Flip Flop (a singolo bit) utilizzati.

Risorsa	Utilizzo	Disponibilità	Utilizzo in %
Look Up Table	30	134600	0,02229
Flip Flop	20	269200	0,00743

Tabella 3: area utilizzata dal componente in relazione all'FPGA citata nella sezione 1.5.

3.3 Report di timing

La specifica non richiede la valutazione del Worst Negative Slack (WNS) di conseguenza verrà considerato massimo. Analizzando i test bench forniti ho notato che la RAM ha un ritardo di esecuzione (T_{RAM}) di 1 ns. Dalle precendenti considerezanio segue il minimo periodo calcolabile:

$$T_{\min} = T_{\text{RAM}} + T_{\text{clock}} - WNS \approx 1ns + 100ns - 100ns = 1ns$$

Ne consegue la massima frequenza di clock: $f_{\max} = 1/T_{\min} \approx 1Ghz$.

3.4 Warnings

Tutti i warning che si sono presentati durante lo sviluppo del componente sono stati risolti senza particolari difficoltà.

3.5 Note aggiuntive sulla sintesi

Per la sintesi ho usato la versione 2019.2 di XILINX VIVADO con i settaggi di default.

4 Simulazioni

Oltre a testare il componente con i test bench forniti dalla specifica ho ritenuto opportuno valutare alcuni casi critici del funzionamento del componente (*descritti nella sezione 4.3*).

4.1 Indirizzo non nella working-zone (test bench fornito)

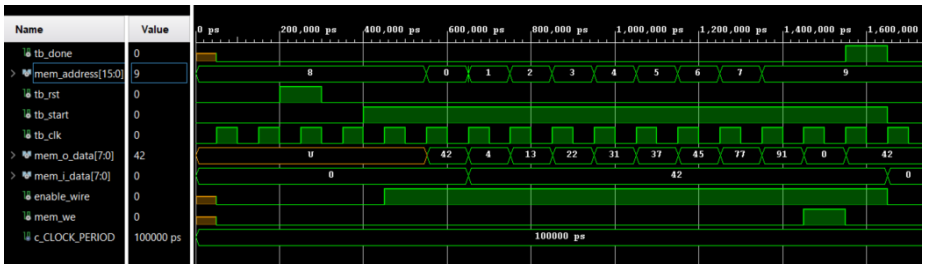
Da questo test bench mi aspetto che tutta la memoria venga scandita e infine ADDR venga proposto senza variazioni. Potremo inoltre valutare il tempo massimo per una singula esecuzione.

4.1.1 Dati

- Working-zone: [4,13,22,31,37,45,77,91];
- ADDR: 42.

4.1.2 Risultati in pre sintesi

Figura 3: valori dei segnali della entity durante la simulazione pre sintesi.



Questo primo test mi conferma il corretto valore dei segnali in seguito al reset. Trascuriamo per semplicità il fatto che il segnale di start arrivi sul fronte di discesa del clock e valutiamo i segnali ad ogni ciclo:

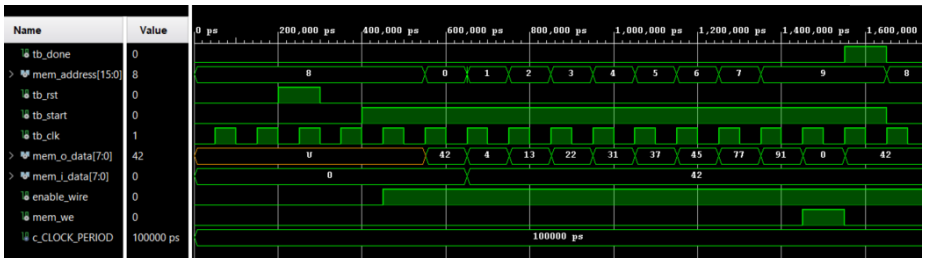
1. Il segnale *enable_wire* viene alzato, mi aspetto che al prossimo clock venga caricato l'ADDR in *mem_o_data*;
2. ADDR viene salvato nell'registro locale, alla memoria viene richiesta la prima WZ;
3. Il confronto tra ADDR e WZ0 non è positivo, si passa al successivo;
4. Il confronto tra ADDR e WZ1 non è positivo, si passa al successivo;
5. Il confronto tra ADDR e WZ2 non è positivo, si passa al successivo;
6. Il confronto tra ADDR e WZ3 non è positivo, si passa al successivo;
7. Il confronto tra ADDR e WZ4 non è positivo, si passa al successivo;

8. Il confronto tra ADDR e WZ5 non è positivo, si passa al successivo;
9. Il confronto tra ADDR e WZ6 non è positivo, si passa al successivo;
10. Il confronto tra ADDR e WZ7 non è positivo, i confronti sono terminati quindi *mem_address* viene correttamento portato a 9 (indirizzo dove deve scrivere) e in *mem_i_data* viene messo 42 (ADDR non modificato);
11. Il segnale *mem_we* viene alzato per scrivere in memoria il valore finale;
12. La scrittura è terminata, si alza *o_done*.

Una volta riportato *tb_start* a '0' il componente torna in modo corretto nella situazione iniziale (*ulteriori test riguardanti esecuzioni in successione nella sezione 4.3*). Il tempo di esecuzione, come mi aspettavo, è di 12 cicli di clock.

4.1.3 Risultati in post sintesi

Figura 4: valori dei segnali della entity durante la simulazione post sintesi.



Non avendo latch e/o segnali indefiniti il test post sintesi è equivalente alla situazione in pre sintesi a meno di alcuni *dont-care* che precedeno il reset iniziale (trascurabili).

4.2 Indirizzo nella working-zone (test bench fornito)

Da questo test bench possiamo verificare che i confronti con le working-zone e la codifica dell'indirizzo avvengano correttamente.

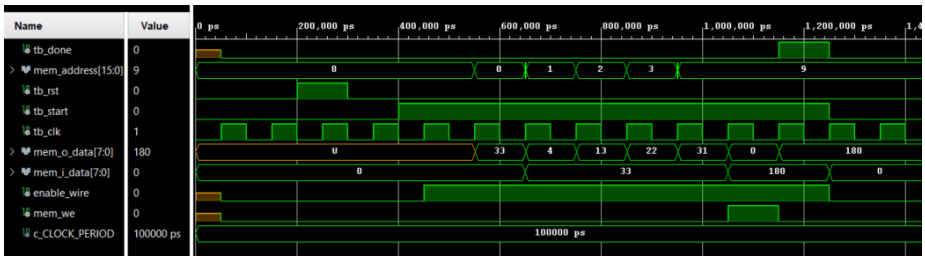
4.2.1 Dati

- Working-zone: [4,13,22,31,37,45,77,91];
- ADDR: 33.

L'indirizzo si trova nella quarta working-zone quindi in output mi aspetto 180 (1-011-0100).

4.2.2 Risultati in pre sintesi

Figura 5: valori dei segnali della entity durante la simulazione pre sintesi.



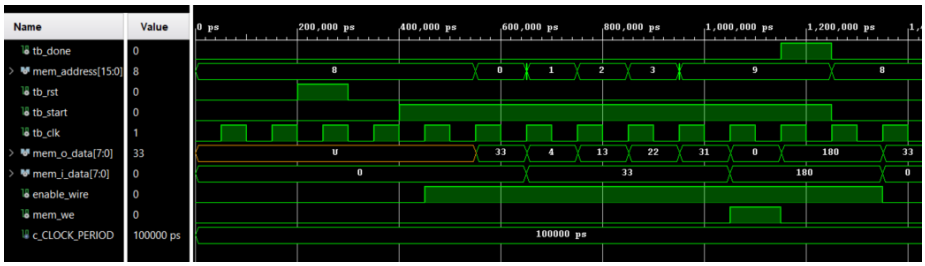
Trascuriamo per semplicità il fatto che il segnale di start arrivi sul fronte di discesa del clock e valutiamo i segnali ad ogni ciclo:

1. Il segnale *enable_wire* viene alzato, mi aspetto che al prossimo clock venga caricato l'ADDR in *mem_o_data*;
2. ADDR viene salvato nell'registro locale, alla memoria viene richiesta la prima WZ;
3. Il confronto tra ADDR e WZ0 non è positivo, si passa al successivo;
4. Il confronto tra ADDR e WZ1 non è positivo, si passa al successivo;
5. Il confronto tra ADDR e WZ2 non è positivo, si passa al successivo;
6. Il confronto tra ADDR e WZ3 è positivo, *mem_address* viene correttamento portato a 9 (indirizzo dove deve scrivere) e in *mem_i_data* viene messo 180 (ADDR nella quarta working-zone con offset 3);
7. Il segnale *mem_we* viene alzato per scrivere in memoria il valore finale;
8. La scrittura è terminata, si alza *o_done*.

Una volta riportato *tb_start* a '0' il componente torna in modo corretto nella situazione iniziale (*ulteriori test riguardanti esecuzioni in successione nella sezione 4.3*). Il tempo di esecuzione è di 8 cicli di clock, quattro in meno rispetto al test bench precedenze dato che non sono stati necessari gli ultimi confronti.

4.2.3 Risultati in post sintesi

Figura 6: valori dei segnali della entity durante la simulazione post sintesi.



Non avendo latch e/o segnali indefiniti il test post sintesi è equivalente alla situazione in pre sintesi a meno di alcuni *dont-care* che precedeno il reset iniziale (trascurabili).

4.3 Altri test bench

Ho realizzato ulteriori test al fine di verificare le seguenti situazioni:

- ADDR nella prima working-zone (confine);
- ADDR nell'ultima working-zone (confine);
- ADDR nella working-zone con offset 0 (confine);
- ADDR nella working-zone con offset 3 (confine);
- riesecuzione di una codifica senza invio del segnale di reset;
- invio del segnale di reset nel mentre di un'esecuzione.

Tutti i test bench, compresi quelli nelle sezioni 4.1 e 4.2, hanno dato esito positivo sia in pre che in post sintesi. I test bench sono stati infine eseguiti in post implementazione confermando i risultati precedenti. Non sono state fatte ulteriori analisi sulla situazione in post implementazione.

5 Conclusione

Ricapitolando ho descritto un componente con le seguenti caratteristiche:

- funzionante in pre/post sintesi e in post implementazione;
- ottimizzato sotto il punto di vista dell'area utilizzata;
- ottimizzato sotto il punto di vista del tempo di esecuzione;
- altamente personalizzabile e configurabile mediante l'uso di costanti;
- utilizzo di LUT[1] par al 0,02229%;
- utilizzo di FF[1] par al 0,00743%.

[1] Calcoli basati sulla dimensione della FPGA citata nella sezione 1.5