Progetto di reti logiche

A.A 2019/2020 - Prof. William Fornaciari

—

Matteo Bettini
10580728 - 888962
Politecnico di Milano
Ingegneria Informatica
Prova Finale

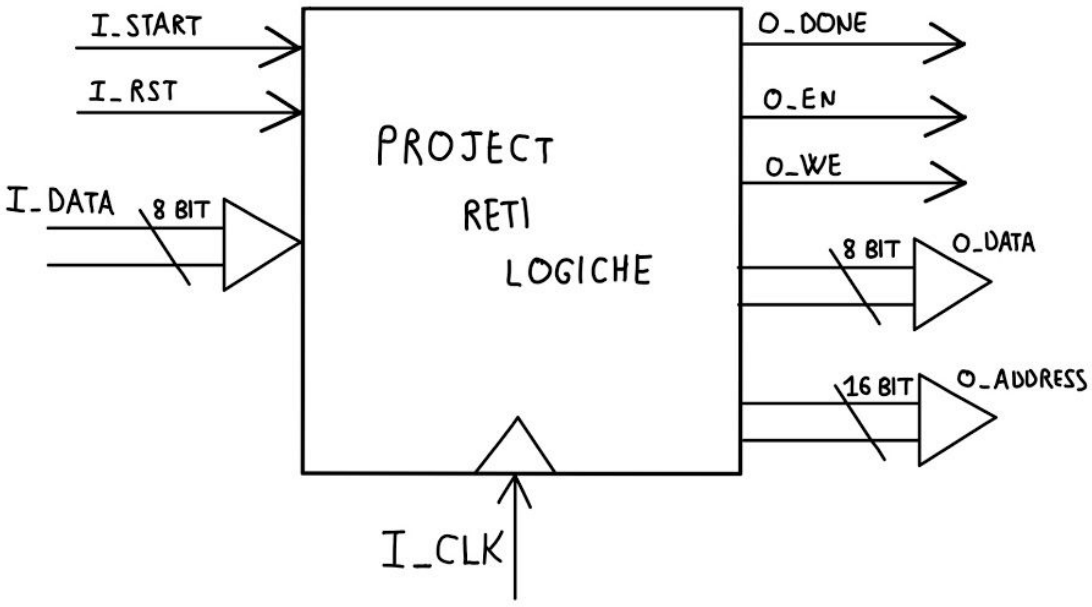
Indice

Introduzione e specifica

L'obiettivo di questo progetto è l'implementazione di un modulo hardware che riconosca l'appartenenza di un dato indirizzo a un certo intervallo di indirizzi detto "Working Zone" (WZ). Se l'indirizzo appartiene a una WZ allora sarà codificato in funzione di tale WZ e del proprio offset all'interno di essa per poi essere trasmesso in uscita, altrimenti verrà trasmesso senza essere cambiato.

Lo scopo è ridurre la dissipazione di potenza sul bus indirizzi. Infatti, nella maggior parte dei casi, le applicazioni usano più frequentemente determinati indirizzi rispetto ad altri. Identificando i set di indirizzi più frequenti, possiamo quindi costruire le WZ. Tramite la codifica degli indirizzi trasmessi sul bus in funzione della WZ a cui appartengono e del loro offset all'interno di tale WZ, viene diminuito nel caso medio il consumo di energia dovuto alla trasmissione di tali indirizzi.

Il componente descritto in questo progetto è una versione molto semplificata di tale algoritmo che opera con indirizzi di 7 bit. Per osservare i vantaggi descritti precedentemente, è infatti necessario analizzare componenti più complessi che operano in ambienti verosimili con un numero di bit molto maggiore.



Architettura

Ho suddiviso lo sviluppo del componente in due fasi distinte. Per prima cosa ho costruito il **datapath** che implementa l'algoritmo richiesto. Successivamente ho progettato la **macchina a stati** che regola le dinamiche del circuito.

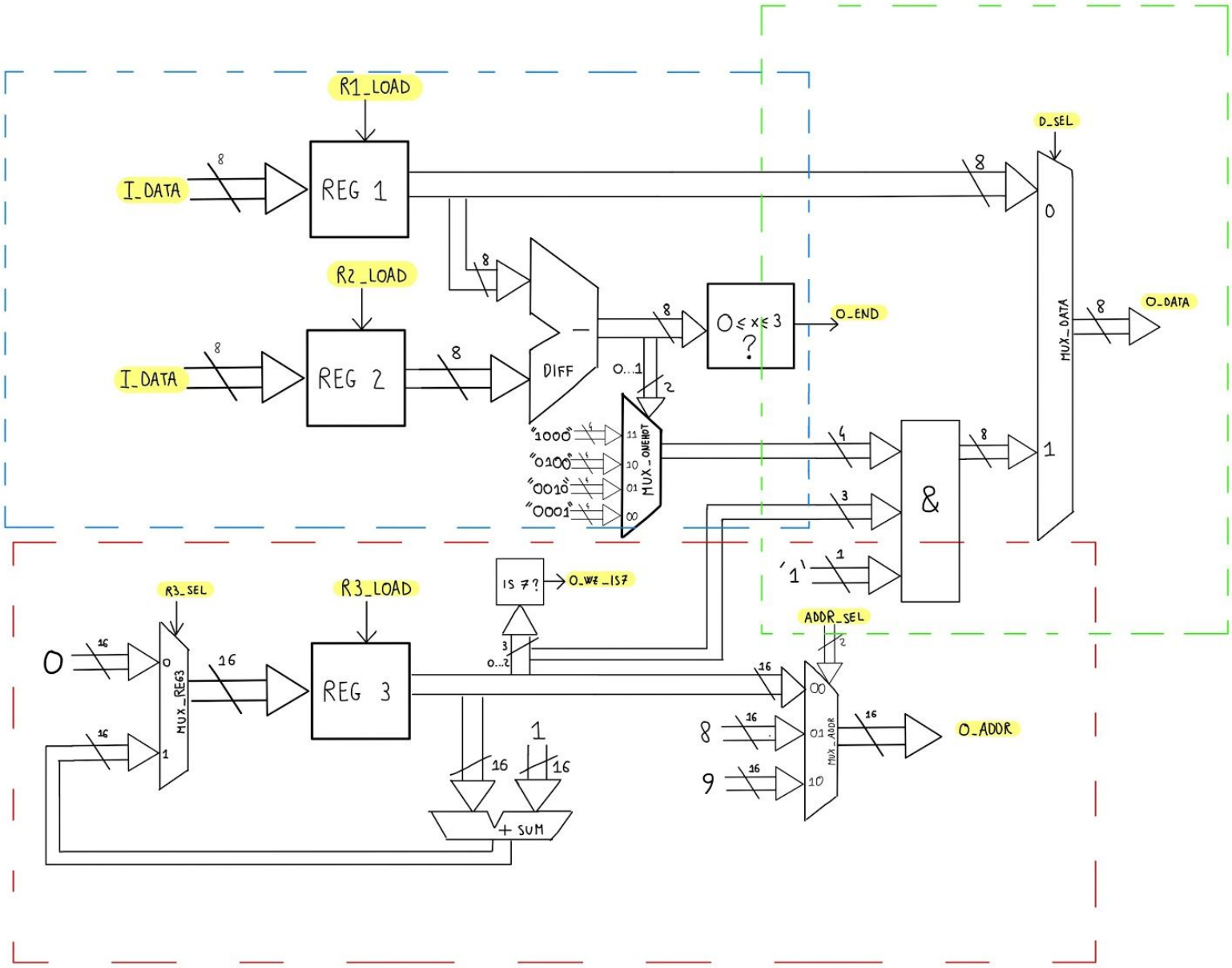
I. Datapath

Nella pagina successiva viene riportato in un disegno lo schema del datapath. Sostanzialmente può essere suddiviso in tre aree a seconda della funzione svolta.

- L'area blu si occupa di confrontare l'indirizzo target con gli indirizzi base delle WZ e di stabilire se un indirizzo appartiene o meno a una data WZ facendo la differenza tra i due; se la differenza è compresa tra 0 e 3 allora l'appartenenza è confermata e tale offset viene convertito in one-hot da un multiplexer (mux). I componenti usati sono: due registri a 8 bit ciascuno ("reg1" adibito al salvataggio dell'indirizzo da convertire e "reg2" che conterrà man mano gli indirizzi base delle WZ), un modulo "diff" che si occupa di fare la differenza tra gli indirizzi, un mux che esegue la codifica dell'offset (numero minimo di bit → one-hot) e un modulo custom che pone "o_end" a '1' se l'indirizzo appartiene alla WZ considerata altrimenti lo pone a '0'.

- L'area rossa invece è adibita a produrre l'indirizzo da utilizzare per l'accesso alla memoria. Svolge la semplice funzione di contare da 0 a 7. Si compone di: un registro a 16 bit "reg3" che conterrà i valori da 0 a 7, un mux per scegliere se caricare in "reg3" il valore 0 o l'output della somma, un sommatore "sum" che prende l'uscita di "reg3" e somma 1, un modulo custom che pone "o_wz_is7" a '1' quando l'output di "reg3" è 7 e un mux per scegliere l'indirizzo da utilizzare per l'accesso alla memoria (tra il valore di "reg3" e gli indirizzi delle celle 8 e 9 usati rispettivamente per leggere l'input e scrivere l'output).

- L'area verde serve a comporre l'indirizzo che poi andrà scritto nella cella 9 della memoria. Se l'indirizzo è stato catalogato come appartenente a una WZ allora il valore che si andrà a scrivere in memoria corrisponde all'ingresso 1 del "mux_data" ed è composto dalla concatenazione di '1', i 3 bit meno significativi di "reg3" e la codifica one-hot dell'offset. Se invece l'indirizzo target non appartiene a nessuna WZ verrà scelto l'ingresso 0 del "mux_data" che manda in uscita l'indirizzo target senza modificarlo. I componenti utilizzati sono: un modulo per la concatenazione e un mux per la scelta dell'output ("mux_data").

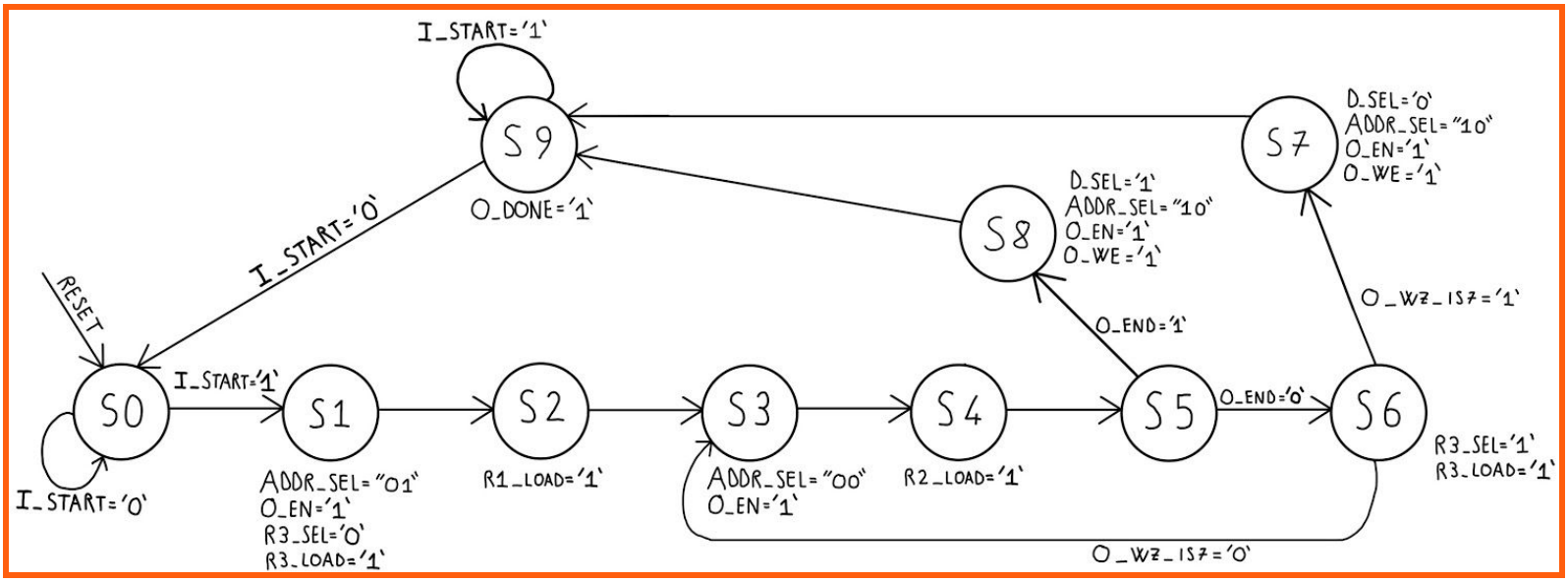


II. Macchina a stati

Nella pagina seguente è riportato il pallogramma della macchina a stati utilizzata. Essa è composta da 10 stati.

- **S0**: è lo stato iniziale della macchina e anche lo stato assunto a seguito del reset. La macchina sta in S0 finché il segnale di start è ‘0’ e appena esso diventa ‘1’ passa in S1.
- **S1**: pongo o_address a 8 e abilito la memoria in lettura (per leggere l’indirizzo target), inoltre viene dato il segnale per precaricare 0 in “reg3”. Passo in S2.
- **S2**: predispongo il caricamento dell’indirizzo target in “reg1” e passo in S3.

- **S3**: pongo o_address uguale al valore attuale del contatore (“reg3”) e abilito la memoria in lettura cosicché al ciclo dopo avrò su i_data l’indirizzo base di una delle WZ. Passo in S4.
- **S4**: predispongo il caricamento di i_data in “reg2” e passo in S5.
- **S5**: se il confronto tra l’indirizzo attuale e quello base della WZ va a buon fine passo in S8, altrimenti vado in S6.
- **S6**: predispongo il caricamento in “reg3” del suo valore incrementato di 1. Se ho appena confrontato l’indirizzo target con la settima e ultima WZ passo in S7 altrimenti torno in S3 pronto a considerare una nuova WZ.
- **S7**: l’indirizzo non appartiene a nessuna WZ quindi su o_data riporterò l’indirizzo target invariato. Per fare ciò abilito la cella 9 in scrittura e scrivo il contenuto di “reg1”. Passo in S9.
- **S8**: l’indirizzo target appartiene a una WZ. Abilito la cella 9 in scrittura e scrivo l’indirizzo codificato. Passo in S9.
- **S9:** Pongo o_done a ‘1’ e resto in S9 finché start non ritorna a ‘0’, quando ciò succede torno in S0.



III. Considerazioni sulle scelte progettuali e possibili alternative

La complessità temporale del componente così ottenuto è O(n) dove n è il numero delle WZ. Infatti, nel caso pessimo, cioè il caso in cui l’indirizzo non appartenga a nessuna WZ, prima di produrre un output, il modulo dovrà caricare gli indirizzi di tutte le WZ.

Questo design risulta più efficiente se vengono considerati certi presupposti di utilizzo nel caso medio.

Più nello specifico, se le WZ più frequenti sono quelle il cui indirizzo è nelle prime celle di memoria, l'accesso ad esse sarà quasi immediato e, di conseguenza, la codifica.

Un'altra possibile scelta di design consiste nello sviluppare un componente che, appena viene avviato, carica tutti gli indirizzi base delle WZ e l'indirizzo target in 9 registri, dopodichè, con 8 moduli sottrattori, procede a identificare la WZ di appartenenza. Se esso viene avviato senza essere resettato, controllerà tramite un apposito segnale se gli indirizzi base delle WZ sono già memorizzati e, se ottiene conferma, salterà la fase di caricamento degli stessi. Quando le WZ vengono cambiate tale modulo dovrà essere resettato.

Andiamo ora ad analizzare **vantaggi e svantaggi** di questi due approcci. Chiameremo "**PRL2**" il componente appena descritto (soluzione alternativa) e "**PRL1**" il componente sviluppato in questo progetto.

"PRL2" presenta indubbiamente una maggiore occupazione di spazio. Usando 10 registri (per le WZ, l'indirizzo target e il contatore) al posto dei 3 di "PRL1" e 7 moduli sottrattori in più rispetto a "PRL1". Inoltre, nel caso in cui dopo ogni codifica esso venga resettato, risulta estremamente inefficiente in quanto la complessità temporale sarà Θ(n) in tutti i casi (con n numero delle WZ). Un ulteriore svantaggio risiede nel fatto che il numero di registri e di sottrattori di "PRL2" cresce linearmente con il numero di WZ, è quindi necessario valutare tale trade-off che renderà il modulo utilizzabile solo con un numero ragionevole di WZ.

I vantaggi di "PRL2" sono però notevoli se vengono considerate alcune ipotesi d'uso. In particolare, se le WZ cambiano molto raramente e il componente non viene resettato al termine di ogni codifica, esso, nella maggior parte dei casi, sarà in grado di completare la computazione in un tempo costante (Θ(1)), in quanto gli indirizzi base delle WZ saranno già salvati nei registri e dovrà solo caricare l'indirizzo target ed eseguire i confronti.

In generale la soluzione da adottare dipende fortemente dai reali casi d'uso. La mia scelta di sviluppare "PRL1" è dovuta all'ipotesi che siano molto frequenti indirizzi appartenenti alle prime WZ e poco frequenti indirizzi non appartenenti a nessuna WZ. Inoltre in un sistema reale il numero delle WZ potrebbe essere molto elevato e quindi sfavorire "PRL2" per ragioni di occupazione spaziale. La complessità temporale di "PRL1" non è inoltre inficiata a fronte di reset molto frequenti.

Testing

Per effettuare i test ho usato il test bench fornito modificando i contenuti delle celle di memoria e l'output atteso. Sono stati effettuati i seguenti casi di test, che vanno a testare i limiti delle specifiche e situazioni potenzialmente critiche tenendo anche conto dell'implementazione (sia testing strutturale che black-box).

Un primo insieme di casi di test è stato effettuato per verificare le situazioni di **limiti inferiori**, sollecitando le zone blu e verde del datapath:

- l'indirizzo target è 0 e appartiene alla prima WZ con offset 0
- l'indirizzo target è 0 e non appartiene a nessuna WZ
- l'indirizzo target appartiene alla prima WZ con offset 0,1,2,3
- l'indirizzo target ha offset 4 dalla prima WZ

Altri test vanno a verificare i **limiti superiori** e, quindi, sollecitano anche la zona rossa:

- l'indirizzo target è 127 ed appartiene all'ultima WZ con offset 0
- l'indirizzo target è 127 ed appartiene all'ultima WZ con offset 3
- l'indirizzo target è 127 e ha offset 4 dall'ultima WZ
- l'indirizzo target appartiene all'ultima WZ con offset 0,1,2,3
- l'indirizzo target ha offset 4 dall'ultima WZ

I casi appena descritti sono poi stati testati anche in WZ diverse dalla prima e l'ultima e anche in condizioni in cui gli indirizzi delle WZ non siano solo in ordine crescente nella memoria, ma anche in ordine decrescente e sparso.

Inoltre, dato che nell'implementazione è presente una differenza tra indirizzi sono stati testati anche casi in cui l'offset tra l'indirizzo target e l'indirizzo base di una WZ sia -1,-2,-3 o -4 per verificare la corretta gestione di risultati negativi in uscita dal sottrattore "diff".

E' stata data per scontata la non sovrapposizione delle WZ. Nel caso di sovrapposizione e di appartenenza a due WZ il componente restituirà la codifica relativa alla WZ contenuta nella cella di memoria con indirizzo più basso.

Conclusioni

I test appena descritti sono stati fatti sul componente sintetizzato con FPGA target xc7a200tfbg484-1 e hanno prodotto i risultati attesi. Il tempo di clock usato è 100 ns. Il componente presenta un Worst Negative Slack di 96.031 ns. Il modulo sintetizzato fa uso di 42 Slice Register di tipo flip-flop e 0 di tipo latch. Il numero di cicli di clock impiegati per la computazione è variabile e dipende dal caso d'uso (vedi).