Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo - Anno 2019/2020

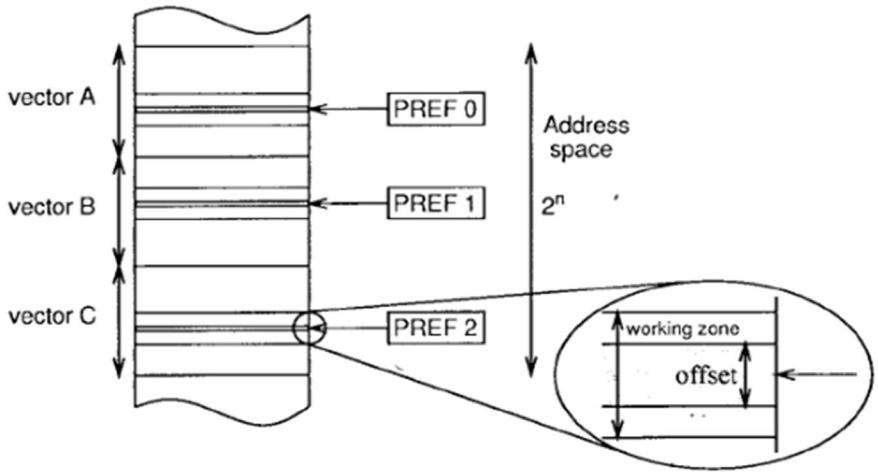
Francesco Puoti
Codice Persona: 10595640 Matricola: 886675

Indice

1. Introduzione e specifiche di Progetto

1.1. Introduzione

Scopo del progetto è l'implementazione di un modulo che, interfacciandosi con una memoria RAM, riesca ad elaborare la codifica di un dato indirizzo, utilizzando la "Working-zone encoding for reducing the energy in microprocessor address buses".
Obiettivo di quest'ultima è quello di diminuire il consumo energetico derivante dagli "input-output pins", che rappresenta una parte sostanziale del consumo energetico totale di un microprocessore. Infatti, la capacità associata a pin esterni è dalle cento alle mille volte maggiore rispetto a quella associata a un nodo interno[1]. E per un microprocessore, la maggior parte dei pin esterni è di collegamento con bus dati e bus indirizzi. Il metodo working_zone è pensato per il secondo i questi. Innanzitutto, occorre specificare che con il termine working_zone si indicano intervalli, con dimensione fissa, di indirizzi di memoria, ognuno con un indirizzo di base, memorizzato in opportune celle della RAM.



In figura[1] una rappresentazione grafica di una memoria con tre working-zone

---

[1] E. Musoll, T. Lang and J. Cortadella, "Working-zone encoding for reducing the energy in microprocessor address buses," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems , vol. 6, no. 4, pp. 568-572, Dec. 1998.

Idea del metodo è che tutti gli indirizzi di memoria, appartenenti a una delle working_zone, vengano trasmessi codificati, indicando numero della working_zone di appartenenza e spiazzamento rispetto all'indirizzo base.
Il processo di codifica ha tre passi principali:

a. Il componente legge da memoria l'indirizzo da codificare.
b. Confronta il suddetto indirizzo con gli indirizzi base delle working-zone, analizzando, per ognuno di quest'ultimi, l'offset.
c. Se l'offset è minore, o anche uguale, alla dimensione prestabilita delle working-zone, allora si procede alla codifica dell'indirizzo: un bit addizionale, rispetto all'indirizzo, chiamato Working-zoneBit viene posto a '1'. I bit di indirizzo vengono, invece, divisi in: numero della working-zone al quale l 'indirizzo appartiene, codificato in binario, e l'offset rispetto all'indirizzo base della working-zone, codificato secondo la tecnica

OneHot. Quest'ultima consiste nell'avere un vettore di bit, posti tutti a '0', tranne uno, posto per l'appunto a '1': la posizione di quest'ultimo indicherà lo spiazzamento. Nel caso di non appartenenza ad alcuna delle Working-Zone previste, il Working-zoneBit viene posto a '0' e l'indirizzo è trasmesso senza modifiche.

1.2. Descrizione Memoria

Nel caso specifico di tale progetto, la memoria è così costituita: un array di dimensione 2^n dove n=16 (quindi gli indirizzi vanno da 0 a 655635) e dimensione delle celle 8 bit. Anche l 'indirizzo, che è da specifica di 7 bit, viene memorizzato su 8 bit: il valore dell'ottavo bit sarà sempre zero. Segue una rappresentazione grafica.

Indirizzi di Memoria	Contenuto della cella
0	Indirizzo base WZ 0
1	Indirizzo base WZ 1
2	Indirizzo base WZ 2
…	…
7	Indirizzo base WZ 7
8	Indirizzo da codificare
9	Risultato della codifica della macchina
…	…
655635	…

1.3. Esempio dell'algoritmo Working-Zone

Facendo riferimento alla rappresentazione data della memoria nel paragrafo precedente, consideriamo il caso seguente:

Indirizzi di Memoria	Contenuto della cella
0	43
1	15
2	64
3	96
4	114
5	28
6	82
7	86
8	Indirizzo da codificare
9	Valore codificato in output

Caso 1:

indirizzo da codificare: 97
esso appartiene alla WZ numero 3.
Valore codificato in output sarà: 178, ovvero [1 – 011 – 0010].

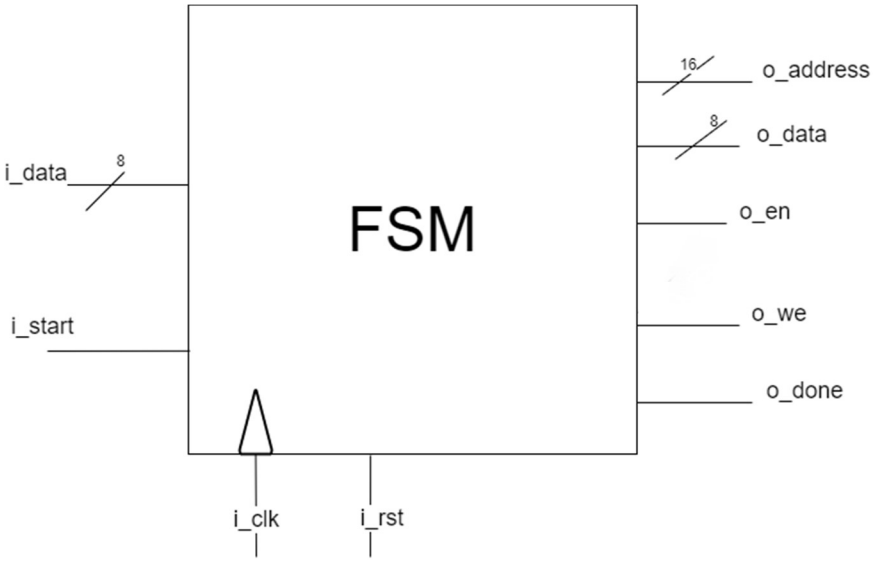
Caso 2:

indirizzo da codificare :121
esso non appartiene ad alcuna WZ.
Valore codificato in output sarà: 121 ovvero [0 – 1111001].

2. Architettura

Il componente è stato implementato utilizzando un automa a stati finiti (FSM), più precisamente una macchina di Mealy. Il circuito è di tipo sequenziale, in quanto i valori logici delle uscite sono determinati sia dai valori logici presenti negli ingressi, sia dallo stato precedente della stessa uscita. Pertanto, il circuito sarà composto da più elementi di memoria.

2.1. Descrizione interfaccia componente



In figura: rappresentazione dell'interfaccia del componente

In particolare:

- **i_clk**: segnale di CLOCK in ingresso generato dalla RAM;
- **i_start**: segnale di START generato dalla RAM;
- **i_rst**: segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- **i_data**: segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- **o_address**: segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- **o_done**: segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- **o_en**: segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- **o_we**: segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- **o_data**: segnale (vettore) di uscita dal componente verso la memoria.

2.2. Processi della macchina e scelte implementative

La scelta progettuale ha optato per la descrizione architetturale della FSM tramite tre processi (state_reg, deltaProcess, lambdaProcess).

- **state_reg**:
  descrive l'evoluzione della macchina a seguito dei cambiamenti dei segnali clock e reset:
  → il fronte di salita del clock produce un aggiornamento dello stato corrente, in base a quanto deciso dal deltaProcess, e di tutti i registri interni;
  → il segnale di reset, al quale la macchina è sensibile in maniera asincrona rispetto al clock, produce, per l'appunto, un azzeramento dello stato della macchina.

- **deltaProcess**:
  descrive l'evoluzione dello stato prossimo della macchina in funzione sia dello stato presente, sia degli ingressi e dei valori memorizzati nei registri.

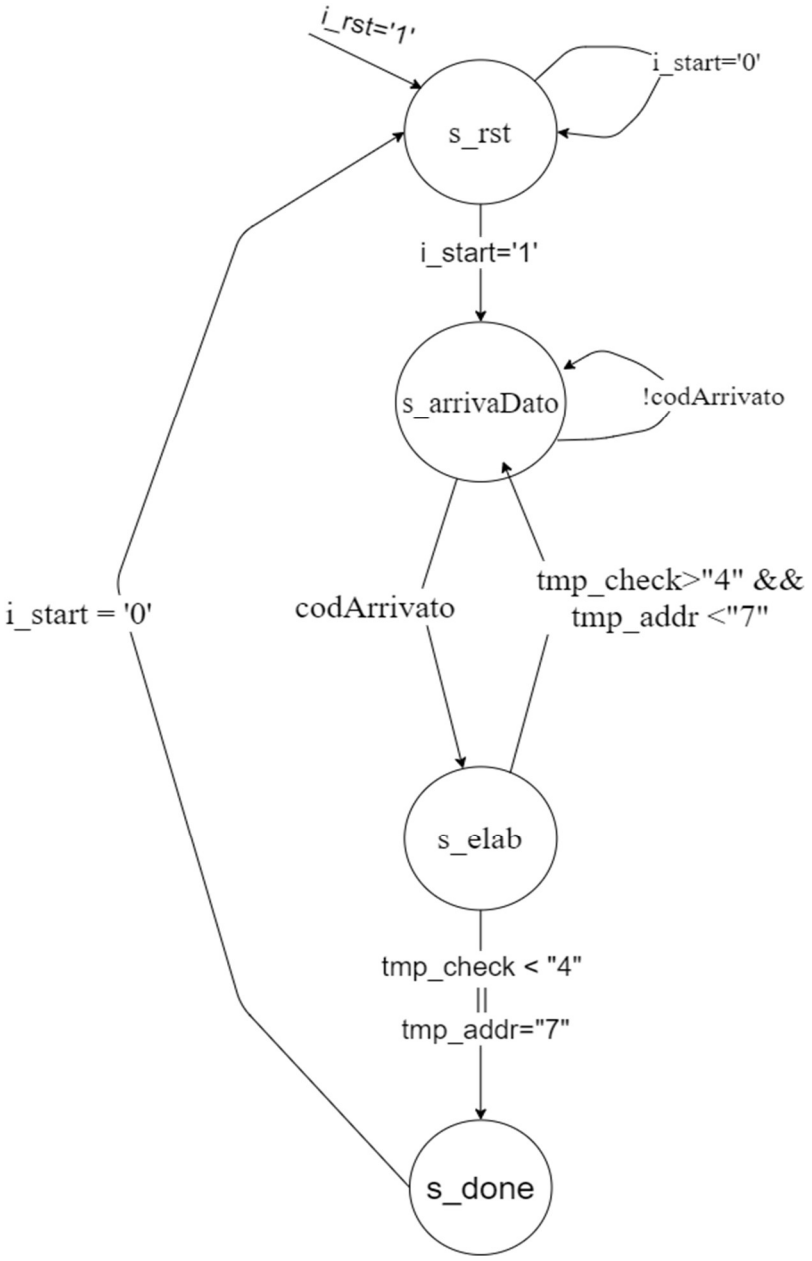
- **lambdaProcess**:
  descrive le uscite e l'evoluzione dei registri interni della macchina.
  Da notare gli assegnamenti all'inizio del processo, indipendenti dallo stato in cui si trova la macchina, che hanno come obiettivo la creazione di registri in fase di sintesi. Infatti, senza gli assegnamenti:

  ```
  next_tmpcod <= tmp_cod;
  next_tmpcheck <= tmp_check;
  next_tmpaddr <= tmp_addr;
  next_codArrivato <= codArrivato;
  ```

  non si sarebbero avuti registri, ma dei latch, che avrebbero potuto produrre effetti di propagazione indesiderata dei segnali.
  Per quanto riguarda invece i segnali di uscita, essi vengono portati a valori non significativi, in modo da poter controllare in maniera deterministica il comportamento della macchina, assegnando, quando di interesse, determinati valori a determinati segnali.

  ```
  o_address <= (others => '-');
  o_data <= (others => '-');
  o_en <= '0';
  o_we <= '-';
  o_done <= '0';
  ```

2.3. Descrizione degli stati della macchina a stati



In figura: rappresentazione grafica della macchina a stati.
Sono state omesse le uscite della macchina per facilitare la lettura.

- **s_rst**:
  stato nel quale perviene la macchina quando il segnale i_rst è posto a '1'. Prepara l'indirizzo per la lettura e aspetta di ricevere il segnale di start alto, così da partire subito nella codifica e quindi next_state<=s_arrivaDato.

- **s_arrivaDato**:
  la macchina riceve il dato in ingresso tramite il segnale i_data e lo salva in un registro:
  → se il dato in ingresso proviene dalla cella contenente l'indirizzo da codificare (codArrivato=false), allora il dato in ingresso viene salvato nel registro tmp_cod.
  → se il segnale proviene, quindi, da una delle celle contenenti un WZ address (codArrivato=true), allora la macchina calcola subito lo spiazzamento tra 'indirizzo da codificare e l'ultimo indirizzo arrivato, salvando il risultato nel registro tmp_check.
  next_state<=s_elab.

- **s_elab**:
  è in tale stato che viene gestito il core della logica applicativa della macchina. Viene analizzato il tmp_check.
  ➔ se tmp_check<=3 allora si procede alla codifica Viene fatto uno shift_left di un vettore di bit, di dimensione 4, di un numero di posizioni pari al tmp_check. Viene assegnato allora il segnale di o_data tramite una concatenazione del bit WZ_Bit='1', WZ_Num=tmp_addr, WZ_Offset.
  ➔ se l'offset indica una non appartenenza alla working zone, la macchina procede con il verificare l'indirizzo della cella di memoria della RAM alla quale era arrivata:
  - se tutte le celle di memoria della RAM, nelle quali sono memorizzate le working zone, non sono state ispezionate, allora si ritorna nello stato di lettura dati.
  - se tutte le celle sono state ispezionate, allora assegna o_data <= tmp_cod. Infatti, essendo l'indirizzo compreso tra 0 e 127, avrà sempre il most significant bit (equivalente al WZ_bit) pari a 0.

  Finita la codifica si ha la transizione nello stato "s_done".

- **s_done**:
  stato di termine computazione, dove la macchina notifica alla memoria la fine dell'elaborazione. Prossimo stato sarà s_rst, dove la macchina aspetterà il via per una nuova computazione.

3. Risultati Sperimentali

3.1. Report di sintesi (analisi del Vivado Synthesis Report)

Si può subito notare la codifica sequenziale degli stati della macchina

```
---------------------------------------------------------------------------------
INFO: [Device 21-403] Loading part xc7a200tfbg484-1
INFO: [Synth 8-802] inferred FSM for state register 'current_state_reg' in module 'project_reti_logiche'
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------------
                   s_rst |                               00 |                               00
            s_arrivadato |                               01 |                               01
                  s_elab |                               10 |                               10
                  s_done |                               11 |                               11
---------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module
```

Come si può notare nella seguente figura, sono stati utilizzati 22 Flip-Flop di tipo D:
→ 3 FF per tmp_addr_reg;
→ 16 FF, 8 per tmp_cod_reg e 8 per tmp_check_reg;
→ 1 FF per il boolean codArrivato.
→ 2 FF per la codifica degli stati.

Inoltre, nella fase di mapping, i blocchi di logica forniti sono stati posizionati su 106 celle dell'FPGA:

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     2|
|3     |LUT1   |     1|
|4     |LUT2   |    18|
|5     |LUT3   |     5|
|6     |LUT4   |     9|
|7     |LUT5   |     3|
|8     |LUT6   |     7|
|9     |FDCE   |    14|
|10    |FDRE   |     8|
|11    |IBUF   |    11|
|12    |OBUF   |    27|
+------+-------+------+
```

Da notare, inoltre, come, in fase di sintesi, si presentino 13 warnings: 12 di si riferiscono al segnale o_address, in quanto i bit (15 downto 4) non vengono mai assegnati direttamente, e quindi fissati a 0 dal tool; il restante warning è perché non sono stati aggiunti costraints.

```
Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   106|
+------+---------+-------+------+
```

4. Test Bench

4.1. Simulazioni (TestBench di esempio forniti e altri)

1) Test bench "tb_pfrl_2020_in_wz":
   Behavioural Simulation, tempo impiegato **1,550.000ns**
   Post-Synthesis Functional Simulation, tempo impiegato **1,550.100ns**

2) Test Bench "tb_pfrl_2020_no_wz":
   Behavioural Simulation, tempo impiegato **2,250.000ns**
   Post-synthesis Timing Simulation, tempo impiegato **2,250.100ns**

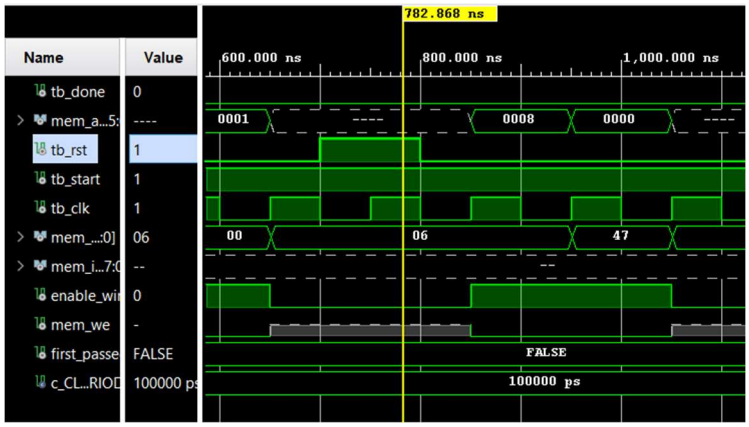
Oltre ai test bench forniti con le specifiche di progetto, sono stati elaborati ulteriori casi di test. Sono stati elaborati:

3) Test caso limite: indirizzo da codificare = 127 e uno degli indirizzi WZ, memorizzato nella cella 7 della RAM, uguale a 124. Tempo impiegato **2350ns.**

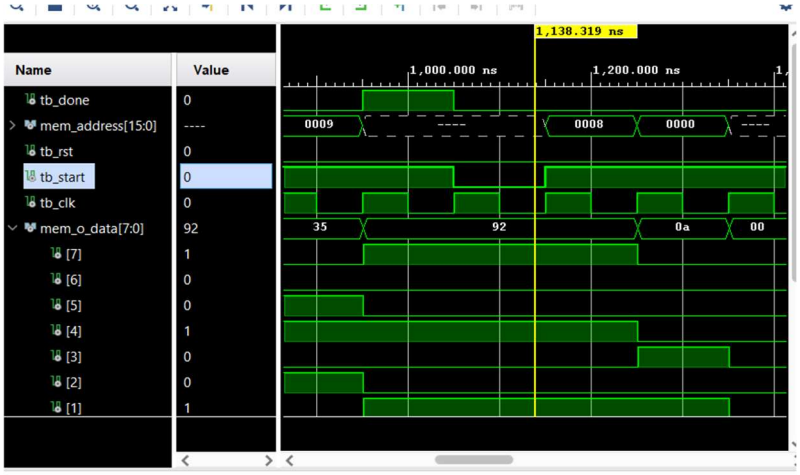
4) Test caso limite: indirizzo da codificare = 0, WZ_ADDR=0, memorizzato nella cella 0 della RAM. Così facendo, è stato verificato anche il tempo impiegato dalla macchina nel caso di WZ_ADDR presente nella cella 0: **950 ns**.

5) Test per controllare che la macchina ignori gli indirizzi con spiazzamento negativo, rispetto al WZ_ADDR: indirizzo da codificare = 11, indirizzo WZ=13. Essendo però i segnali dichiarati come unsigned, la sottrazione, che nel caso di analisi del segno darebbe un numero negativo, dà un valore di spiazzamento molto elevalo, in tal caso [11111110] = 254.

6) Test per analisi del comportamento macchina nel caso di reset asincrono: si noti come la macchina, non appena ricevuto il reset asincrono, riporti tutti i propri segnali a un valore di default, permettendo anche di modificare i dati all'interno della RAM, senza dover aspettare la fine del processo e dare un secondo segnale di start.



7) Test per analisi del comportamento nel caso di un secondo segnale di start a 1, a seguito di un cambiamento di indirizzi nelle celle della RAM:
si vede chiaramente il passaggio tb_done='l' a fine della prima codifica della macchina; tb_done='0' a seguito di i_start='0'; a questo punto, ricevuto i_start='1', la macchina parte con la successiva codifica.



5. Conclusioni

Il progetto è stato svolto con l'obiettivo di ottimizzare la macchina, cercando di diminuire il numero di cicli di clock impiegati per svolgere il processo di codifica, e il numero di FlipFlop usati, e schematizzare gli stati in base alle fasi principali del metodo Working_zone. Ciò ha portato, anche, a una riduzione del numero degli stati. Possiamo notare, infatti, come lo stato s_elab, oltre ad avere l'algoritmo di confronto tra il wzAddr e l'indirizzo da codificare, contenga anche la logica di codifica e di abilitazione alla lettura e scrittura in RAM. La macchina precedente aveva uno stato di elaborazione, nel quale si analizzava lo spiazzamento tra wzAddr e indirizzo da codificare e poi vi era uno stato di codifica aggiuntivo (s_codifica). Si perdevano, quindi, due cicli di clock: uno per passare da s_elab a s_codifica e uno per dare tempo a s_codifica di effettuare le operazioni, su un dato già disponibile in s_elab. Inoltre, vi era uno stato s_letturaIndirizzo, che preparava i segnali di output per leggere da memoria, quindi un altro ciclo di clock perso. Pertanto, esso è stato eliminato, aggiungendo la logica di preparazione segnali dove opportuno, ed è stato lasciato solo lo stato di ricezione del dato.
Per diminuire il numero di FlipFlop usati, si è cercato di minimizzare i segnali dei registri al minimo numero di bit necessario per mantenere le informazioni di interesse. Esempio è il registro tmp_addr: esso contiene l'indirizzo della cella della RAM , e quindi, in questo caso, del numero della WorkingZone, alla quale la macchina è giunta. Invece di fare un segnale da 16 bit, si è optato per uno da 3 bit, considerato che le working_zone sono memorizzate nelle prime 7 celle della RAM.
Per concludere quindi, si hanno uno stato in cui si aspetta il segnale di start, uno di lettura dati da memoria, uno di analisi del dato ricevuto, uno stato di conclusione del processo.