



Laboratorio Architettura degli Elaboratori Elaborato SIS e Verilog

Università di Verona
Imbriani Paolo - Grasu Iuliana Alexandra
VR500437 - VR501064

Marzo 2024

Contents

1	Introduzione e specifiche del progetto	3
1.1	Analisi dei requisiti	4
1.1.1	SIS	4
1.1.2	Verilog	4
2	Architettura del dispositivo	5
3	Diagramma del controllore	7
3.1	Progettazione STG	7
4	Architettura del Datapath	9
5	Ottimizzazione	12
5.1	Ottimizzazione per area: prima e dopo	12
5.2	Mapping Tecnologico	12
6	Scelte progettuali	13

1 Introduzione e specifiche del progetto

In questa relazione verrà riportato come progettare un sistema digitale prendendo come esempio la gestione di partite della *morra cinese*, conosciuta anche come *sasso-carta-forbici*. I nostri obiettivi includono l'ottimizzazione della performance, riduzione dell'area e il ritardo il più possibile.

Per renderla più avvincente, ogni partita si articola di più manche, con le seguenti regole:

- Si devono giocare un minimo di quattro manche;
- Si possono giocare un massimo di diciannove manche. Il numero massimo di manche, viene settato al ciclo di clock in cui viene iniziata la partita
- Vince il primo giocatore che avrà un punteggio di due manche in più del proprio avversario, a patto di aver giocato almeno quattro partite;
- Il giocatore vincente della manche precedente non può ripetere l'ultima mossa utilizzata. Nel caso lo facesse, la manche non sarebbe valida ed andrebbe ripetuta (quindi non conteggiata);
- In caso di pareggio la manche viene conteggiata. Alla manche successiva, entrambi i giocatori possono usare tutte le mosse.

Ci vengono dati anche **i tre ingressi**:

- PRIMO [2 bit]: mossa scelta dal primo giocatore
- SECONDO [2 bit]: mossa scelta dal secondo giocatore
- INIZIA [1 bit]: bit di reset

e le **due uscite**:

- MANCHE [2 bit]: fornisce il risultato dell'ultima manche.
- PARTITA [2 bit]: fornisce il risultato della partita.

1.1 Analisi dei requisiti

Quando si vuole creare un sistema digitale (o in generale, progettare) il punto di partenza è l'analisi dei requisiti. Questo comporta riunire le conoscenze e gli strumenti che verranno utilizzati per la costruzione del progetto. Non si vuole mai essere colti impreparati durante la fase di progettazione, che è indubbiamente il nodo più critico del progetto.

Per realizzare questo progetto ci vengono forniti due linguaggi per la descrizione hardware.

- SIS
- Verilog

1.1.1 SIS

SIS è uno strumento per la sintesi, simulazione e l'ottimizzazione di circuiti combinatori e sequenziali. Date le informazioni di input e output (insieme agli eventuali suoi mintermini ovvero al loro *ON-set*), SIS produce, sintetizzando il comportamento del circuito dandoci la possibilità di osservarne i cambiamenti o di saperne le caratteristiche. In SIS sono stati integrati molti programmi e algoritmi diversi, consentendo all'utente di scegliere tra una varietà di tecniche per ogni fase del processo. SIS funge sia da quadro all'interno del quale è possibile testare e confrontare vari algoritmi, sia da strumento per la sintesi automatica e l'ottimizzazione di circuiti sequenziali.

La versione di SIS utilizzata in questo progetto è la **1.3.6**, rilasciata, circa, nel 2005.

1.1.2 Verilog

Verilog è un linguaggio di descrizione dell'hardware (HDL, Hardware Description Language) utilizzato per la progettazione e la simulazione di circuiti digitali. Una volta aver realizzato la codifica in Verilog, è possibile sintetizzarlo in un circuito elettronico reale, che può essere implementato su FPGA (Field Programmable Gate Arrays) o ASIC (Application-Specific Integrated Circuit).

Verilog consente la creazione di "testbench", ovvero ambienti di simulazione

utilizzati per verificare e validare il corretto funzionamento di un design. Questo è essenziale durante lo sviluppo per garantire il corretto funzionamento del circuito rispetto ai vari input e condizioni e sarà fondamentale per la realizzazione del progetto richiesto.

2 Architettura del dispositivo

L'intero progetto è stato basato sul modello *controllore* e *datapath* (FSMD). Durante la prima fase della progettazione ci siamo mantenuti ad un livello di astrazione *funzionale* in modo da:

- Definire l'insieme di operazioni necessarie per espletare le attività indicate dalle specifiche;
- Identificare le particolari operazioni svolte dalle unità funzionali, la necessità di memorizzazione di dati o risultati "temporanei" ed "intermedi";
- Capire come progettare la sezione "operativa" che esegue le istruzioni immagazzinandole in una memoria costituita da unità funzionali e registri. (Ovvero il *data path*);
- Identificare eventuali segnali di controllo che devono **garantire** la corretta attivazione delle unità funzionali presenti nel datapath;
- Definire il progetto del controllore (Macchina a stati finiti) che genera i segnali di controllo per il datapath.

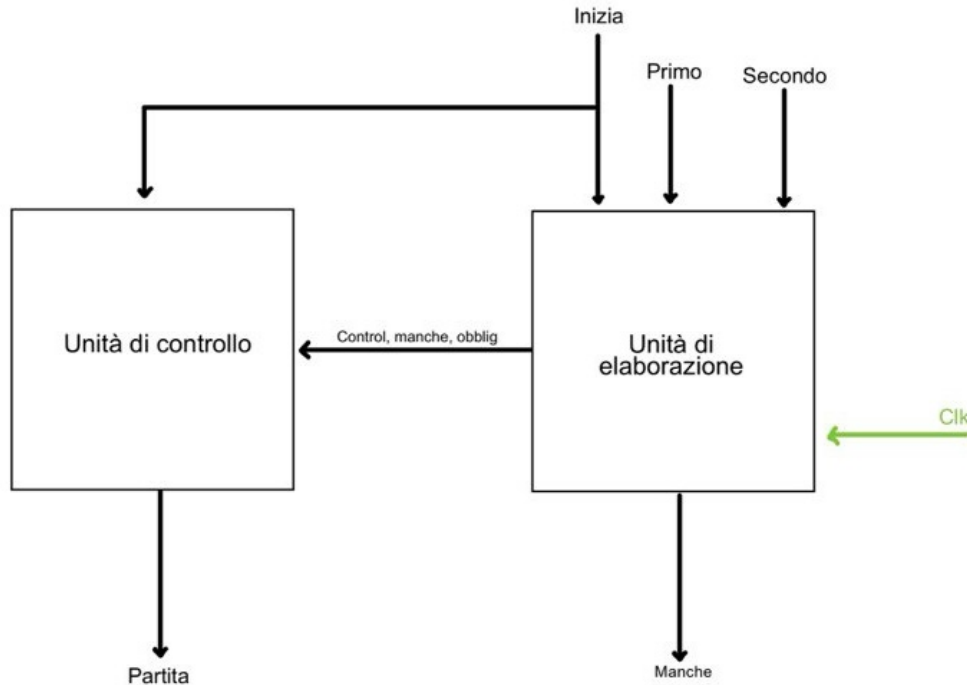


Figure 1: Architettura finale

I segnali di stato, invece, sono i seguenti:

- Control: Indica quando una partita ha raggiunto le manche massime. Se questo segnale è uguale a 1, vuol dire che le manche sono arrivate a 0.
- Manche: il valore della manche viene calcolato nel datapath e poi spedito alla FSM. Sono i bit che ci permettono di avanzare da uno stato all'altro e capire chi, dei due giocatori, sta vincendo senza necessariamente segnarsi il punteggio;
- Obblig: Obblig - *diminutivo di "obbligatorio"*, indica quando sono state "superate" le 4 manche obbligatorie. Se Obblig è 0, chiunque abbia un vantaggio di 2 rispetto all'altro, vince automaticamente la partita.

3 Diagramma del controllore

La progettazione generale del circuito comporta la creazione della *Macchina a Stati Finiti* anche chiamata *FSM* (Finite State Machine). Quest'ultima può essere descritta graficamente mediante un *diagramma degli stati*, o *grafo di transizione dello stato* (State Transition Graph o STG).

Nel nostro contesto, adotteremo la *macchina di Mealy*, che descrive il funzionamento di una FSM usando degli archi orientati che rappresentano le transizioni alle quali verrà associato anche un simbolo d'uscita.

3.1 Progettazione STG

Durante la progettazione dello *State Transition Graph*, ci siamo più volte chiesti quale sia il modo migliore per rappresentare il circuito. Abbiamo dovuto passare dalla descrizione "a parole" al grafo degli stati. Una macchina a stati è *fisicamente realizzabile* quando soddisfa le due seguenti condizioni:

- Il numero degli stati è finito;
- Il comportamento della macchina in un istante *non dipende da eventi futuri*.

Si tenga presente che non esiste una corrispondenza unica tra un circuito sequenziale e una FSM (o viceversa), poiché allo stesso circuito possono corrispondere più FSM e la stessa FSM può essere trasformata in più circuiti sequenziali.

In principio, si definisce l'insieme degli stati S - stabilendone, fra l'altro, la cardinalità - e si identificano la *funzione di stato prossimo*, ovvero δ , e la *funzione di uscita*, ovvero γ .

Questa è stata la nostra interpretazione per la consegna a noi assegnata.

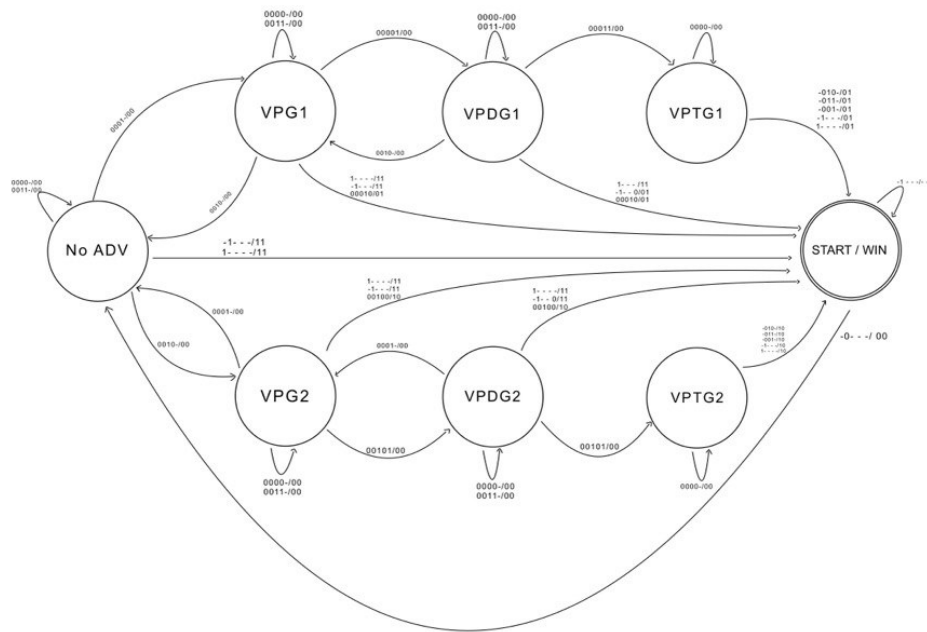


Figure 2: Diagramma degli stati

Come già precedentemente detto, il nostro controllore è una macchina a stati finiti di Mealy:

input	output
Inizia, Control, Manche, Obblig	Partita

La FSM ha 8 stati:

- STARTEND: Il dispositivo inizialmente è spento e bisogna inserire un input per poter farlo partire. Ci spostiamo quando il numero di manche da giocare è diverso da 0. Questo stato indica quando siamo stati nello stato di Inizio o Fine di una partita.
- NOADV: stato nel quale entrambi i giocatori non hanno vantaggio uno sull'altro.
- VPUG1/VPUG2: stati speculari per i quali G1/G2 sono in ”**VantaggioPiùUno**” sull'avversario. Stato nel quale si arriva alla fine delle manche. Se la partita va in pareggio, ma un giocatore vince la prossima manche e Obblig=0, il G1/G2 vince.
- VPDG1/VPUG2: ulteriori stati speculari i quali G1/G2 sono in ”**VantaggioPiùDue**” rispetto all'avversario.
- VPTG1/VPTG2: ”**VantaggioPiùTre**” caso più estremo possibile. Infatti, la partita si continua anche nel caso di 3-0 o 0-3. (Teoricamente G1/G2 avrebbe vinto a tavolino, ma si giocano sempre tutte e 4 le manche).

4 Architettura del Datapath

La prima operazione eseguita dal Datapath è decidere dove indirizzare gli input PRIMO E SECONDO. Questi due input, in base a un demultiplexer che ha come selettore INIZIA vengono spediti nelle varie componenti del circuito. Se INIZIA=1 vengono mandati nel *ContaManche*, componente che si occupa di settare le manche iniziali nel caso in cui la partita sia terminata

(oppure se è stata interrotta bruscamente). Se INIZIA=0, PRIMO E SECONDO vengono inviati al *VietaMosse* e al *CalcolaManche*.

Quindi il nostro circuito si può dividere in tre componenti principali (anche se non delimitate graficamente nel circuito):

1. *ContaManche*, salva le manche in un registro; viene sottratto il valore e poi risalvato all'interno del registro se la manche giocata è valida. Altrimenti, il valore di NumeroManche rimane lo stesso. Ogni volta viene controllato se sono state superate le 4 manche obbligatorie. Se vengono superate, impostiamo il segnale Obblig = 0 della FSM che ci aiuta a determinare il vincitore direttamente. Inoltre, se le manche sono terminate e quindi sono uguali a 0, mandiamo il segnale alla FSM dove CONTROL=1.
2. *VietaMosse* è una componente che "salva" la mossa del giocatore che ha vinto. Se uno dei due giocatori vince la manche precedente ed utilizza la mossa vietata, la manche non è valida e bisogna rigiocarla.
3. *CalcolaManche* è la componente più importante perché è quella che fa funzionare la maggior parte del circuito, calcolando la manche attuale in base al valore di PRIMO e SECONDO.

input	output
Primo, Secondo, Inizia	Manche, Obblig, Control

5 Ottimizzazione

5.1 Ottimizzazione per area: prima e dopo

Statistiche del circuito prima dell'ottimizzazione per area:

```
sis> print_stats
FSMD          pi= 5   po= 4   nodes=147   latches=17
lits(sop)=1123
```

Figure 4: Prima dell'ottimizzazione

147 nodi e 1123 letterali.

```
sis> print_stats
FSMD          pi= 5   po= 4   nodes= 55   latches=17
lits(sop)= 319
```

Figure 5: Dopo dell'ottimizzazione

Dopo aver lanciato due volte il comando "full_simplify" e due volte lo "script.rugged", abbiamo ottenuto 55 nodi e 319 letterali. Il circuito ha subito un miglioramento del 73%.

5.2 Mapping Tecnologico

Dopo aver ottimizzato l'intero circuito per area, lo abbiamo mappato con la libreria synch.genlib.

```

>>> before removing serial inverters <<<
# of outputs:      21
total gate area:    5696.00
maximum arrival time: (63.60,63.60)
maximum po slack:   (-4.60,-4.60)
minimum po slack:   (-63.60,-63.60)
total neg slack:    (-623.40,-623.40)
# of failing outputs: 21
>>> before removing parallel inverters <<<
# of outputs:      21
total gate area:    5696.00
maximum arrival time: (63.60,63.60)
maximum po slack:   (-4.60,-4.60)
minimum po slack:   (-63.60,-63.60)
total neg slack:    (-623.40,-623.40)
# of failing outputs: 21
# of outputs:      21
total gate area:    5536.00
maximum arrival time: (63.20,63.20)
maximum po slack:   (-4.60,-4.60)
minimum po slack:   (-63.20,-63.20)
total neg slack:    (-617.00,-617.00)
# of failing outputs: 21

```

Figure 6: Mapping Tecnologico del circuito

Abbiamo ottenuto un'area di 5536.00 e un ritardo da 63.20.

6 Scelte progettuali

- La FSM si occupa di capire soltanto l'andamento della partita. Il processo di elaborazione è stato interamente affidato al Datapath. Inviemo un ulteriore segnale di controllo di nome "Controllo" che determina quando sono terminate le partite.
- Se un giocatore usa una mossa vietata, la manche non è valida. Questo vale anche per chi utilizza "nessuna mossa" ovvero 00. Anche in questo caso la manche non è valida e bisogna rifarla.
- Quando Inizia = 1, le manche sono inizializzate a 0 e stessa cosa per le partite. Se la partita viene interrotta a causa di Inizia = 1, il dispositivo torna alla configurazione iniziale Start/End. Tuttavia, dobbiamo aspettare un ciclo di clock aggiuntivo per permettere alla FSM di spostarsi da Start/End a NOADV. (Se vogliamo utilizzare le manche di questo Inizia come numero di manche effettivo per la prossima partita.)