Elaborato Sis e Verilog

Corso di Architettura degli Elaboratori A.A. 2023/2024
Prof. Franco Fummi, Prof. Michele Lora

Tommi Bimbato VR500751, Antonio Iovine VR504083

9 febbraio 2024

Indice

Sommario

Questa relazione presenta una descrizione dettagliata del progetto sviluppato in SIS e Verilog per il gioco della ”Morra Cinese”. Vengono illustrati i concetti principali, le scelte progettuali e i risultati ottenuti. In allegato alla presente relazione sono presenti i codici SIS e Verilog inerenti al progetto. L’elaborato è stato eseguito da Tommi Bimbato e Antonio Iovine durante il primo semestre del corso di Architettura degli Elaboratori tenuto dai professori Franco Fummi e Michele Lora nell’A.A. 2023/2024.

Capitolo 1

Introduzione

1.1 Approccio progettuale

L'elaborato è stato inizialmente analizzato e sviluppato a partire da una risoluzione algorithmica delle specifiche e dei requisiti di output *(Linguaggio C)*. Successivamente, è stato elaborato un possibile schema di funzionamento del sistema (FSMD), sono state individuate le componenti più affini al calcolo combinatorio e la loro controparte nell'esecuzione in FSM (vedere sezione 5.3). Una volta suddivise le funzioni tra Datapath e controllore (FSM), si è descritto in codice verilog il comportamento generale della macchina. Si è passati contemporaneamente alla progettazione a *gate level* dei componenti del datapath confrontando il comportamento del modello in Verilog e i singoli blocchi che compongono il Datapath.

1.2 Analisi delle specifiche

1.2.1 Specifiche di input e output

Le specifiche impongono questa configurazione di input e output principali:

- **Input**
  - `PRIMO`: segnale a 2 bit che rappresenta la scelta del giocatore 1 (tabella 1.1)
  - `SECONDO`: segnale a 2 bit che rappresenta la scelta del giocatore 2 (tabella 1.1)
  - `INIZIA`: segnale a 1 bit che funge da input per il reset e l'inizio della partita
- **Output**
  - `MANCHE`: segnale a 2 bit che rappresenta il vincitore della manche (tabella 1.2)
  - `PARTITA`: segnale a 2 bit che rappresenta la fine della manche (tabella 1.3)

1.2.2 Specifiche di funzionamento e regole del gioco

Dinamica di gioco

Quando il segnale `INIZIA` viene impostato a 1, il sistema si resetta e si prepara per una nuova serie di manche. Il numero massimo di manche che possono essere giocate in una partita è determinato dalla concatenazione dei bit di `PRIMO` e `SECONDO`, a cui viene aggiunto il valore minimo di manche giocabili (4). La partita si divide in manche, durante le quali ciascuno dei due giocatori sceglie una tra tre mosse: `SASSO`, `CARTA` e `FORBICE`. Questa scelta avviene simultaneamente per entrambi i giocatori, e viene rappresentata attraverso gli input `PRIMO` e `SECONDO`. Una volta che entrambi i giocatori hanno effettuato la loro mossa, il sistema elabora le scelte e determina il vincitore della manche e lo descrive tramite l'output `MANCHE`.

PRIMO/SECONDO	Codifica in bit
Sasso	01
Carta	10
Forbice	11
Mossa non valida	00

Tabella 1.1: Codifica di PRIMO e SECONDO

MANCHE	Codifica in bit
Vittoria G1	01
Vittoria G2	10
Pareggio	11
Manche annullata	00

Tabella 1.2: Codifica di MANCHE

PARTITA	Codifica
Vince PRIMO	01
Vince SECONDO	10
Pareggio	11
Partita in corso	00

Tabella 1.3: Codifica di PARTITA

Regola della mossa ripetuta da un uscente vinciore

In addizione alla regola standard per la definizione del vincitore della manche è stata aggiunta nelle specifiche una regola che si attiva alla ripetizione della mossa da parte del vincitore uscente da una manche. Se un giocatore vince una manche, nella successiva non potrà riutilizzare la stessa mossa, in caso contrario la manche sarà nulla fintanto che il giocatore non cambierà mossa.

Vincitore in caso di raggiungimento delle partite massime

Il vincitore della partita è il primo giocatore che raggiunge un vantaggio di due manche sull'altro giocatore. In termini progettuali questa affermazione *potrebbe* nascondere un ambiguità: il sistema è stato quindi progettato affinchè in caso di raggiungimento del numero di manche massime con un giocatore in vantaggio di una sola manche la vittoria sarà assegnata anche se non è soddisfatta la specifica $Vantaggio = 2$.

Capitolo 2

FSDM (Finish State Machine with Datapath)

Il sistema di controllo tramite FSM gestisce il controllo a partire dall'elaborazione del datapath delle singole manche. Gli ingressi del sistema sono direttamente collegati al datapath che calcola, per ogni manche, il vincitore. Questo dato (sotto forma di segnale a 2 bit), vine passato alla FSM che si occupa di gestire il flusso dei segnali di output. Il Datapath passa inoltre all'FSM 2 segnali di controllo che si attivano al superamento delle 4 manche minime giocate e al raggiungimento del numero massimo di manche giocabili[1].

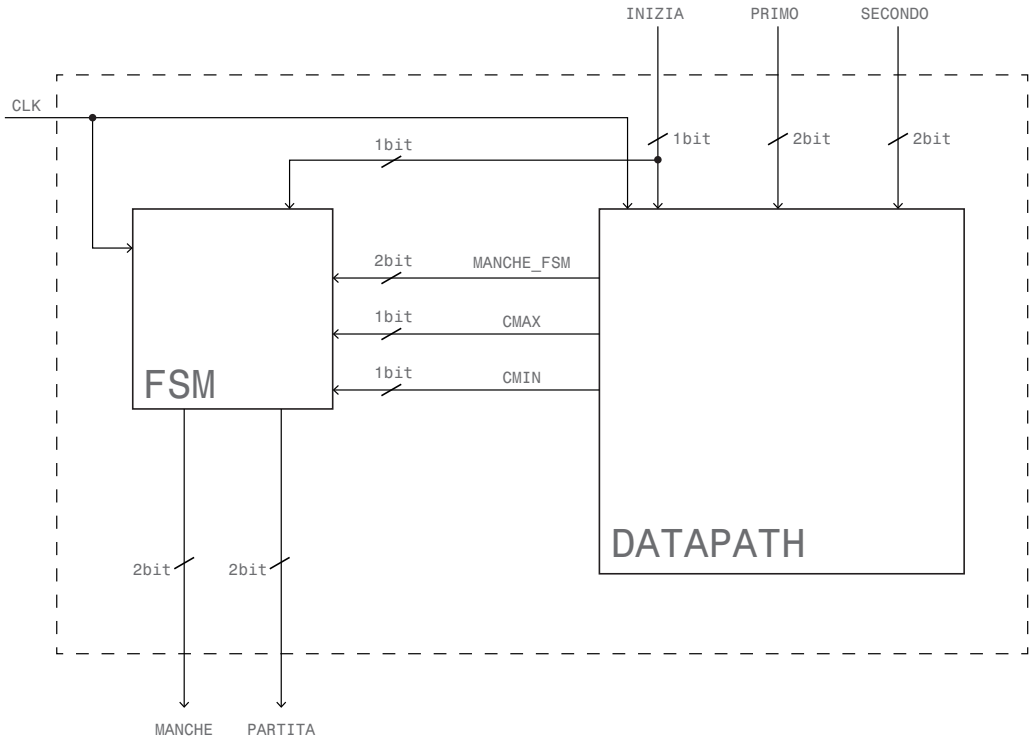

Figura 2.1: Schema di funzionamento generale dell'FSMD

2.1 Datapath

Il Datapath è il componente che si occupa di elaborare le scelte dei giocatori e determinare il vincitore della manche. Nonostante abbiano approcci differenti nella descrizione hardware, sia utilizzando SIS a livello di gate che scrivendo codice Verilog in uno stile

[1] Nello specifico `CMIN` sale ad 1 quando sono state giocate 4 manche e `CMAX` sale ad 1 quando sono state giocate le manche massime.

behavioural, il datapath presenta un comportamento coerente e equivalente in entrambe le rappresentazioni.

2.1.1 Funzioni Principali del Datapath

Reset e Inizializzazione dei Registri

La funzione di reset e inizializzazione dei registri permette al sistema di tornare ad una configurazione coerente e prevedibile all'inizio di una nuova partita. Il segnale `INIZIA` è collegato direttamente al Datapath e viene utilizzato per la sopracitata funzione.

Settaggio del Numero di Manche Massime

Il Datapath è incaricato di impostare il numero massimo di manche per la partita. Questo valore è basato sulla concatenazione dei bit rappresentanti le scelte dei giocatori (`PRIMO` e `SECONDO`) e un valore minimo di manche giocabili (4). In concreto questa funzione viene eseguita registrando in un registro a 5 bit il valore di `PRIMO` e `SECONDO` concatenati e sommando 4.

Calcolo combinatorio del vincitore della *manche* e applicazione delle regole interne

Una delle funzioni più cruciali del Datapath è determinare la mossa vincente tra le due prese in input (`PRIMO` e `SECONDO`). Durante la progettazione a gate level (ambiente SIS), per garantire la correttezza e comprensibilità del percorso dei segnali, abbiamo adottato una strategia di codifica degli input `PRIMO` e `SECONDO`. Questi input, rappresentanti le mosse dei giocatori, sono stati convertiti in segnali a 3 bit ciascuno, dove ogni bit corrisponde ad una mossa[2]. La codifica inziale, (tabella 1.1), consente una rappresentazione chiara e compatta delle mosse ma difficilmente confrontabile direttamente a gate level. Per fare questo è stato modellato il componente `"2b_to_3b.blif"` (vedere sezione 4.1) che converte la codifica binaria di PRIMO e SECONDO in una codifica a 3 bit. Il Datapath gestisce anche le regole che coinvolgono la ripetizione della mossa da parte del vincitore uscente attraverso la memorizzazione in un registro dell'ultimo vincitore e della mossa utilizzata per la vittoria, confrontandolo quando necessario con la mossa giocata nella manche corrente.

Conteggio delle Manche Giocate e Invio di Segnali di Controllo all'FSM

Il Datapath tiene traccia del numero di manche giocate e invia segnali di controllo specifici all'FSM. Questi segnali contribuiscono al corretto flusso di gioco e determinano il passaggio agli stati successivi dell'FSM, in particolare dal Datapath escono il segnale di controllo `CMIN` e `CMAX`. All'interno del datapath è presente un contatore a 5 bit che confronta la sua configurazione con il registro che contiene il numero massimo di manche impostato all'inizio della partita. Il contatore di manche giocate reagisce anche alla possibilità di manche annullata: in questo caso il componente viene bypassato e il conteggio rimane statico fino alla sucessiva manche valida.

2.2 Finish State Machine

L'FSM è il componente che si occupa di gestire il flusso dei segnali di output e di controllo, è stata ideata ad 8 stati e secondo il modello della macchina di Mealy. Il controllore FSM procede a decretare il vincitore se le condizioni per farlo sono soddisfatte tenendo conto dei vantaggi che accumulano i rispettivi giocatori. Lo stato iniziale dell'FSM è `S0` (tabella 2.1), da questo varia in base all'esito del segnale che determina il vincitore da ogni manche in arrivo dal Datapath. Quando sussistono le condizioni per la vittoria della partita o terminano le manche giocabili il controllore decreta il vincitore tramite L'output `PARTITA` e torna allo stato iniziale qualora l'input principale `INIZIA` salga ad 1 stimolando il reset. Il *controllore FSM* prende in input 4 segnali:

[2] In ambiente verilog questa codifica non è stata necessaria vista la più alta astrazione dell'ambiente stesso e la possibilità di confrontare le mosse direttamente a 2 bit con un costrutto *case*

- MANCHE_FSM: segnale a 2 bit che rappresenta il vincitore della manche da passare alla FSM.
- CMIN: segnale a 1 bit che si attiva al raggiungimento delle 4 manche minime.
- CMAX: segnale a 1 bit che si attiva al raggiungimento delle manche massime.
- INIZIA: segnale a 1 bit che funge da input per il reset e l'inizio della partita e che permette di riportare la FSM allo stato iniziale.

Ed emette in output i 2 segnali principali:

- MANCHE: segnale a 2 bit che rappresenta il vincitore della manche.
- PARTITA: segnale a 2 bit che decreta il vincitore dell'intera partita.

La codifica binaria degli stati è stata impostata secondo il seguente schema:

Nome stato	**Codifica**	**Descrizione**
S0	000	INIT / Pareggio
S1	001	Vantaggio di PRIMO = 1
S2	010	Vantaggio di PRIMO = 2
S3	011	Vantaggio di PRIMO = 3
S4	100	Vantaggio di SECONDO = 1
S5	101	Vantaggio di SECONDO = 2
S6	110	Vantaggio di SECONDO = 3
S7	111	Partita conclusa

Tabella 2.1: Codifica degli stati dell'FSM

La combinazione degli input secondari (MANCHE_FSM, CMIN, CMAX), l'ingresso principale INIZIA e lo stato corrente dell'automa determinano lo stato prosimo (δ) e gli output (secondo la funzione di output λ). In uscita alla FSM ci sono le due uscite principali del sistema: MANCHE (2bit) e PARTITA (2bit). I passaggi di stato e le uscite sono descritti in dettaglio nella tabella 2.2 e nel diagramma dei passaggi di stato (figura 2.2).

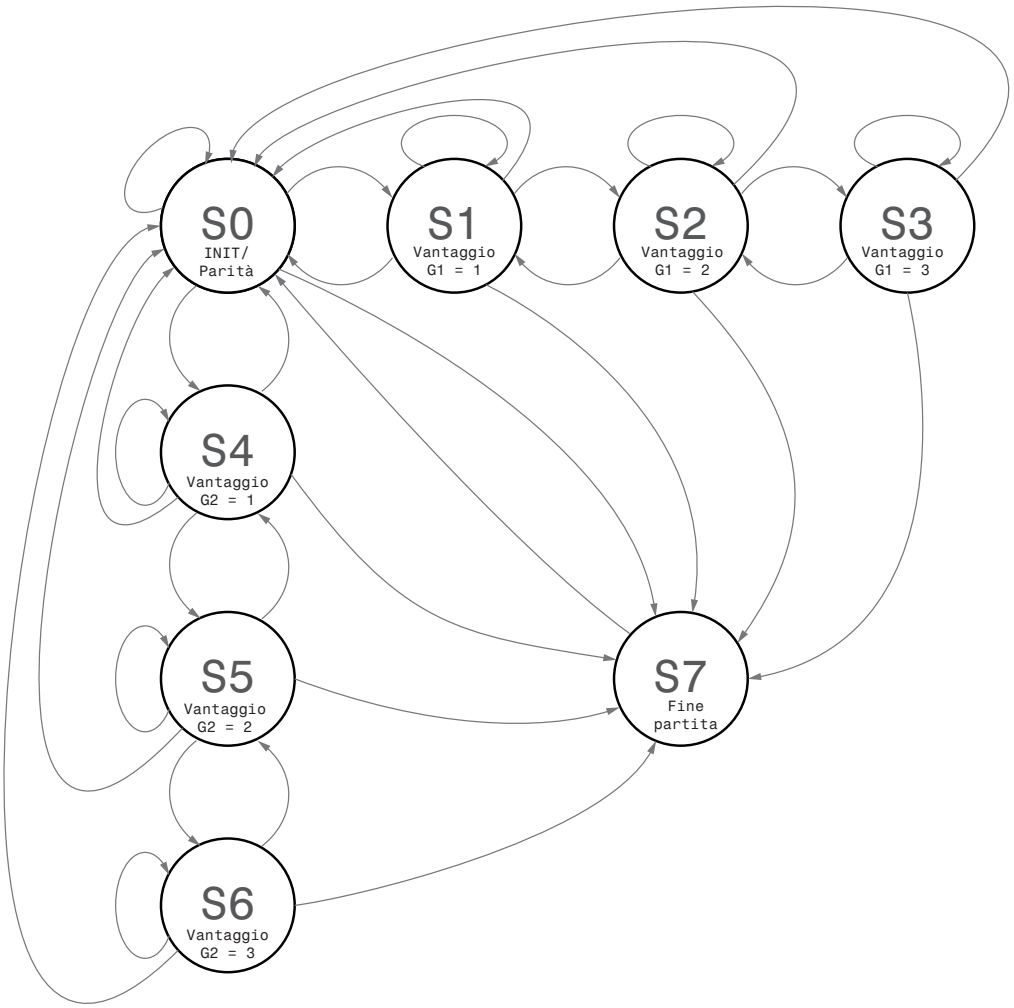

Figura 2.2: State Transition Graph

Input				Stato Attuale	Stato Prossimo	Output	
MANCHE_FSM	-	CMIN - CMAX	- INIZIA			MANCHE -	PARTITA
00	-	-	0	S0	S0	00	00
11	0	0	0	S0	S0	11	00
00	-	-	0	S1	S1	00	00
11	0	0	0	S1	S1	11	00
00	-	-	0	S2	S2	00	00
11	0	0	0	S2	S2	11	00
00	-	-	0	S3	S3	00	00
11	0	0	0	S3	S3	11	00
01	-	0	0	S0	S1	01	00
01	0	0	0	S1	S2	01	00
01	0	0	0	S2	S3	01	00
10	0	0	0	S3	S2	10	00
10	-	0	0	S2	S1	10	00
10	-	0	0	S1	S0	10	00
00	-	-	0	S4	S4	00	00
11	0	0	0	S4	S4	11	00
00	-	-	0	S5	S5	00	00
11	0	0	0	S5	S5	11	00
00	-	-	0	S6	S6	00	00
11	0	0	0	S6	S6	11	00
10	-	0	0	S0	S4	10	00
10	0	0	0	S4	S5	10	00
10	0	0	0	S5	S6	10	00
01	0	0	0	S6	S5	01	00
01	-	0	0	S5	S4	01	00
01	-	0	0	S4	S0	01	00
11	-	1	0	S0	S7	11	11
01	-	1	0	S0	S7	01	01
10	-	1	0	S0	S7	10	10
11	-	1	0	S1	S7	11	01
01	1	-	0	S1	S7	01	01
10	-	1	0	S1	S7	10	11
11	1	-	0	S2	S7	11	01
01	1	-	0	S2	S7	01	01
10	-	1	0	S2	S7	10	01
11	1	-	0	S3	S7	11	01
01	1	-	0	S3	S7	01	01
10	1	-	0	S3	S7	10	01
11	-	1	0	S4	S7	11	10
01	-	1	0	S4	S7	01	11
10	1	-	0	S4	S7	10	10
11	1	-	0	S5	S7	11	10
01	-	1	0	S5	S7	01	10
10	1	-	0	S5	S7	10	10
11	1	-	0	S6	S7	11	10
01	1	-	0	S6	S7	01	10
10	1	-	0	S6	S7	10	10
--	-	-	1	S0	S0	00	00
--	-	-	1	S1	S0	00	00
--	-	-	1	S2	S0	00	00
--	-	-	1	S3	S0	00	00
--	-	-	1	S4	S0	00	00
--	-	-	1	S5	S0	00	00
--	-	-	1	S6	S0	00	00
--	-	-	1	S7	S0	00	00
--	-	-	0	S7	S7	00	00

Tabella 2.2: Descrizione input e output dei passaggi di stato dell'FSM

Capitolo 3

Verilog

3.1 Design.sv

Contesualmente alla presente relazione è stato allegato il codice `design.sv` contenente la descrizione hardware del sistema in linguaggio Verilog. Nella prima parte della descrizione hardware vengono dichiarate le porte di input e output del sistema e i registri necessari. Oltre agli input e output principali definiti nelle specifiche, sono stati aggiunti dei registri per il controllo del numero di manche settate e giocate, la memorizzazione dell'ultima mossa vincente e i segnali di controllo scambiati tra il datapath e l'FSM. Vengono inoltre specificati i parametri locali di tutti possibili stati della FSM.

```
module MorraCinese (
// Inputs
input [1:0] PRIMO,
input [1:0] SECONDO,
input INIZIA,
input clk,
// Outputs
output reg [1:0] MANCHE = 2'b00,
output reg [1:0] PARTITA = 2'b00
);

reg [1:0] MOSSA_PRECEDENTE = 2'b00;
reg [4:0] NUMERO_PARTITE = 5'b00000;
reg [4:0] CONTATORE = 5'b00000;
reg [1:0] MANCHE_REG = 2'b00;
reg [2:0] STATO = 3'b000;
reg [2:0] STATO_PROSSIMO;
reg CMIN = 1'b0;
reg CMAX = 1'b0;

reg [1:0] MANCHE_FSM;
integer i = 1; // Debug purpose

localparam S0 = 3'b000,
            S1 = 3'b001,
            S2 = 3'b010,
            S3 = 3'b011,
            S4 = 3'b100,
            S5 = 3'b101,
            S6 = 3'b110,
            S7 = 3'b111;
```

Viene descritto un blocco sincorno che al variare del Clock aggiorna lo stato della FSM con lo stato prossimo calcolato nell'FSM.

```
always @(posedge clk) begin : UPDATE_STATE
  STATO = STATO_PROSSIMO;
end;
```

Ad ogni fronte di salita del segnale di clock (`posedge clk`) si procede con il reset iniziale in risposta al segnale `INIZIA`. Durante il reset, vengono inizializzati i registri e i segnali di controllo del sistema e lo stato dell'FSM. Successivamente, mediante una serie di condizioni e case statement si gestiscono le diverse combinazioni di mosse dei giocatori e vengono applicate le regole del gioco. La variabile `MANCHE_FSM` rappresenta lo stato della manche, mentre `MOSSA_PRECEDENTE` memorizza la mossa vincente precedente. Si procede con conteggio delle manche e la verifica del raggiungimento delle manche minime e massime e vengono eventualmente inviati alla FSM i segnali di controllo `CMIN` e `CMAX`. La descrizione include anche ritardi (#1), questa pausa è stata inserita dopo la sezione di reset iniziale quando il segnale INIZIA è attivo. L'obiettivo è dare il tempo al sistema di completare l'operazione di reset prima di procedere con le operazioni successive.[1]

```
38  always @(posedge clk) begin : DATAPATH
39  if (INIZIA) begin
40    MOSSA_PRECEDENTE  = 2'b00;
41    PARTITA           = 2'b00;
42    MANCHE_REG        = 2'b00;
43    MANCHE_FSM        = 2'b00;
44    MANCHE            = 2'b00;
45    CMAX              = 1'b0;
46    CMIN              = 1'b0;
47    NUMERO_PARTITE    = {PRIMO, SECONDO} + 4;
48    CONTATORE         = 5'b00000;
49    STATO             = 3'b000;
50  end else begin
51    // Datapath core logic
52    if ({MOSSA_PRECEDENTE, MANCHE_REG} == {PRIMO, 2'b01} || {MOSSA_PRECEDENTE,
          MANCHE_REG} == {SECONDO, 2'b10}) begin
53      MANCHE_FSM = 2'b00;
54    end else begin
55      case ({PRIMO, SECONDO})
56        4'b0101, 4'b1010, 4'b1111: begin
57          MANCHE_FSM = 2'b11;
58          MOSSA_PRECEDENTE = 2'b00;
59        end
60        4'b0111, 4'b1001, 4'b1110, 4'b0100, 4'b1000, 4'b1100 : begin
61          MANCHE_FSM = 2'b01;
62          MOSSA_PRECEDENTE = {PRIMO};
63        end
64        4'b0110, 4'b1011, 4'b1101, 4'b0001, 4'b0010, 4'b0011: begin
65          MANCHE_FSM = 2'b10;
66          MOSSA_PRECEDENTE = {SECONDO};
67        end
68        4'b0000: begin
69          MANCHE_FSM = 2'b00;
70          MOSSA_PRECEDENTE = 2'b00;
71        end
72      endcase
73      if (MANCHE_FSM != 2'b00) begin
74        MANCHE_REG = MANCHE_FSM;
75        CONTATORE = CONTATORE + 1;
76      end
77    end
78
79    if (CONTATORE == 4)
80      CMIN = 1'b1;
81    if (CONTATORE == NUMERO_PARTITE)
82      CMAX = 1'b1;
83  end
84  #10;
85 end
```

Viene descritto un blocco sincrono che elabora lo stato della FSM e calcola lo stato prossimo in base alle condizioni di input e allo stato corrente. Parallelamente configura

[1] Poiché la simulazione in Verilog è discreta e avviene a passi di tempo discreti (cicli di clock), è necessario introdurre un ritardo per garantire che il sistema si trovi in uno stato coerente prima di eseguire ulteriori operazioni.

le uscite principali al valore corretto (MANCHE e PARTITA). L'elaborazione è vincolata al segnale INIZIA (collegato sia al Datapath che alla FSM), l'FSM elabora gli stati prossimi anche in correlazione a quest'ultimo.

```
87  always @(posedge clk) begin : FSM
88    if (!INIZIA) begin
89      case (STATO)
90        S0: begin
91          MANCHE = MANCHE_FSM;
92          case (MANCHE_FSM)
93            2'b00: begin
94              PARTITA = 2'b00;
95              STATO_PROSSIMO = S0;
96            end
97            2'b01: begin
98            . . .
```

La descrizione hardware in verilog termina con la chiusura di tutti gli "switch case" aperti e il termine del modulo. Viene descritto il funzionamento nel momento in cui INIZIA è attivo, ovvero il reset dello stato prossimo ad S0 e delle uscite principali.

```
348         . . .
349         S7: begin
350           STATO_PROSSIMO = S7;
351           MANCHE = 2'b00;
352           PARTITA = 2'b00;
353         end
354       endcase
355     end else if (INIZIA) begin
356       MANCHE = 2'b00;
357       PARTITA = 2'b00;
358       STATO_PROSSIMO = S0;
359     end
360   end
361 endmodule
```

3.2 testbench.sv

Il file allegato testbench.sv contiene la descrizione del testbench per stimolare in maniera controllata il testing del sistema.

Nella prima parte del codice vengono dichiarati e collegate tutte le uscite e le entrate necessarie allo svolgimento del test. Vengono inoltre dichiarati i segnali di controllo per la simulazione e i file di output per la memorizzazione dei risultati attraverso le task Simulate e Output. Si definisce inoltre un clock temporizzato a 10 unità di tempo per la simulazione.

```
1   module MorraCinese_TB;
2   reg [1:0] PRIMO;
3   reg [1:0] SECONDO;
4   reg INIZIA;
5   reg [1:0] MANCHE;
6   reg [1:0] PARTITA;
7   reg clk;
8
9   integer out;
10  integer script;
11
12 MorraCinese test14 (
13   .PRIMO(PRIMO),
14   .SECONDO(SECONDO),
15   .INIZIA(INIZIA),
16   .clk(clk),
17   .MANCHE(MANCHE),
18   .PARTITA(PARTITA)
19   );
20
```

```
  task Simulate();
    $fdisplay(script, "simulate %b %b %b %b %b", PRIMO[1], PRIMO[0], SECONDO[1],
        SECONDO[0], INIZIA);
  endtask

  task Output();
    $fdisplay(out, "Outputs: %b %b %b %b", MANCHE[1],MANCHE[0],PARTITA[1],
        PARTITA[0]);
    endtask



  always
  #10 clk = ~clk;
```

Successivamente all'interno di un blocco `initial` vengono inizializzati i file di output e il clock, e vengono invocate le task `Simulate` e `Output` per stimolare il sistema e memorizzare i risultati sincronzzate agli input da "inviare" al modello.

```
  initial begin

    script = $fopen("testbench.script", "w");
    out = $fopen("output_verilog.txt", "w");

    $dumpfile("tb.vcd");
    $dumpvars(1);

    $fdisplay(script,"rl FSMD.blif");

    clk = 1'b0;

    INIZIA = 1'b1;
    PRIMO = 2'b11;
    SECONDO = 2'b01;
    Simulate();
    #20;
    Output();   //1


    INIZIA = 1'b0;
    PRIMO = 2'b11;
    SECONDO = 2'b10;
    Simulate();
    #20;
    Output();   //2

    PRIMO = 2'b01;
    SECONDO = 2'b01;
    Simulate();
    #20;
    Output();   //3
    . . .
```

```
  $finish;
  $fclose(script);
  $fclose(out);

end
endmodule
```

Capitolo 4

SIS

4.1 Architettura del sistema

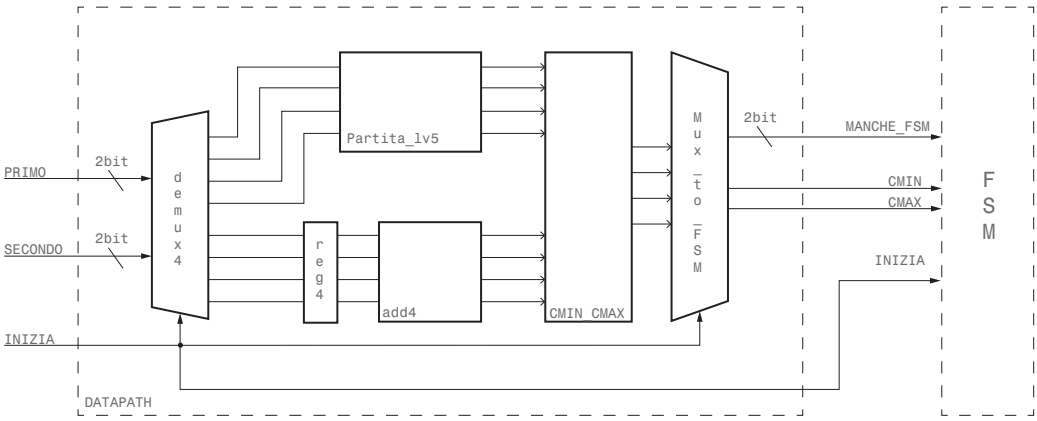

Figura 4.1: Schema di funzionamento generale del Datapath in SIS

Componenti del modello SIS

La descrizione hardware del sistema è stata sviluppata in SIS utilizzando la descrizione a livello di gate. Si è scelto di procedere con la descrizione a *gate level* per poter maneggiare la simulazione ad un livello di astrazione piu basso e per poter avere un controllo più preciso sulle risorse utilizzate. Nello specifico sono stati modellati inizialmente i seguenti componenti:

- **Porte Logiche ed elementi base:**
  - **and.blif**: Porta logica AND.
  - **not.blif**: Porta logica NOT.
  - **notAnd.blif**: Porta logica NAND.
  - **nor.blif**: Porta logica NOR.
  - **xnor.blif**: Porta logica XNOR.
  - **xor.blif**: Porta logica XOR.
  - **or.blif**: Porta logica OR.
  - **or_3b.blif**: Porta logica OR a 3 bit.
  - **uno.blif**: Costante 1.
  - **zero.blif**: Costante 0.
- **Operatori Matematici, di Confronto e contatori:**

  - **add4.blif**: Modulo a 5 bit che aggiunge la costante 4.
  - **cont5bit.blif**: Contatore a 5 bit.
  - **countmanche.blif**: Contatore di manche giocate.
  - **maggiore.blif**: Confronto maggiore a 1 bit.
  - **maggiore5b.blif**: Confronto maggiore a 5 bit.
  - **sum1bit.blif**: Sommatore a 1 bit.
  - **sum1bit_no.blif**: Sommatore a 1 bit (senza riporto).
  - **sum2bit.blif**: Sommatore a 2 bit.
- **Multiplexer e Demultiplexer:**
  - **Mux.blif**: Multiplexer a 1 bit.
  - **Mux_2b.blif**: Multiplexer a 2 bit.
  - **Mux_5b.blif**: Multiplexer a 5 bit.
  - **Mux_to_FSM.blif**: Filtro in output dal datapath che reagisce in base al valore di INIZIA.
  - **demux1bit.blif**: Demultiplexer a 1 bit.
  - **demux4bit.blif**: Demultiplexer a 4 bit.
- **Registri:**
  - **Reg.blif**: Registro a 1 bit.
  - **Reg_2b.blif**: Registro a 2 bit.
  - **Reg_4b.blif**: Registro a 4 bit.
  - **Reg_5b.blif**: Registro a 5 bit.

E successivamente implementati in blocchi più complessi:

- **Blocchi Finali e Altri Moduli:**
  - **2b_to_3b.blif**: Convertitore delle codifiche di PRIMO e SECONDO in valori a 3 bit per le mosse.
  - **CMIN_CMAX.blif**: Implementazione dei segnali di controllo diretti alla FSM per il conteggio del numero di manche giocate.
  - **Check_Mossa.blif**: Multiplexer modificato per il controllo della mossa del giocatore.
  - **Check_mossa_2b.blif**: Versione a 2 bit di Check_Mossa.blif che funziona per due flussi (PRIMO e SECONDO).
  - **Gate_3b.blif**: [da scrivere!].
  - **Mossa_ill.blif**: Implementazione di una mossa ripetuta.
  - **tavolino.blif**: Assegna la vittoria in caso di mossa ”sconosciuta” all’altro giocatore.
  - **Controllo_Partita.blif**: Modulo che elabora l’annullamento della manche in caso di mossa non permessa.
  - **Partita_lv1.blif**: raggruppamento 1.
  - **Partita_lv2.blif**: raggruppamento 2.
  - **Partita_lv3.blif**: raggruppamento 3.
  - **Partita_lv4.blif**: raggruppamento 4.
  - **Partita_lv5.blif**: raggruppamento 5.
  - **DATAPATH.blif**: insieme dei moduli del datapath.
  - **FSM.blif**: Macchina a stati finiti.
  - **FSMD.blif**: Macchina a stati finiti con datapath.

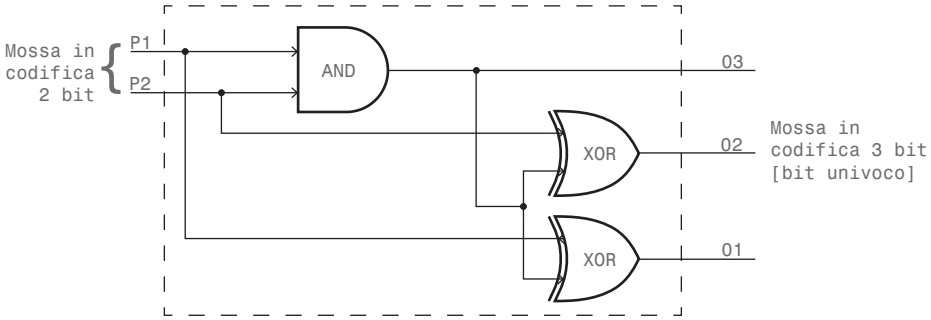

Figura 4.2: Funzionamento del modulo `2b_to_3b.blif`

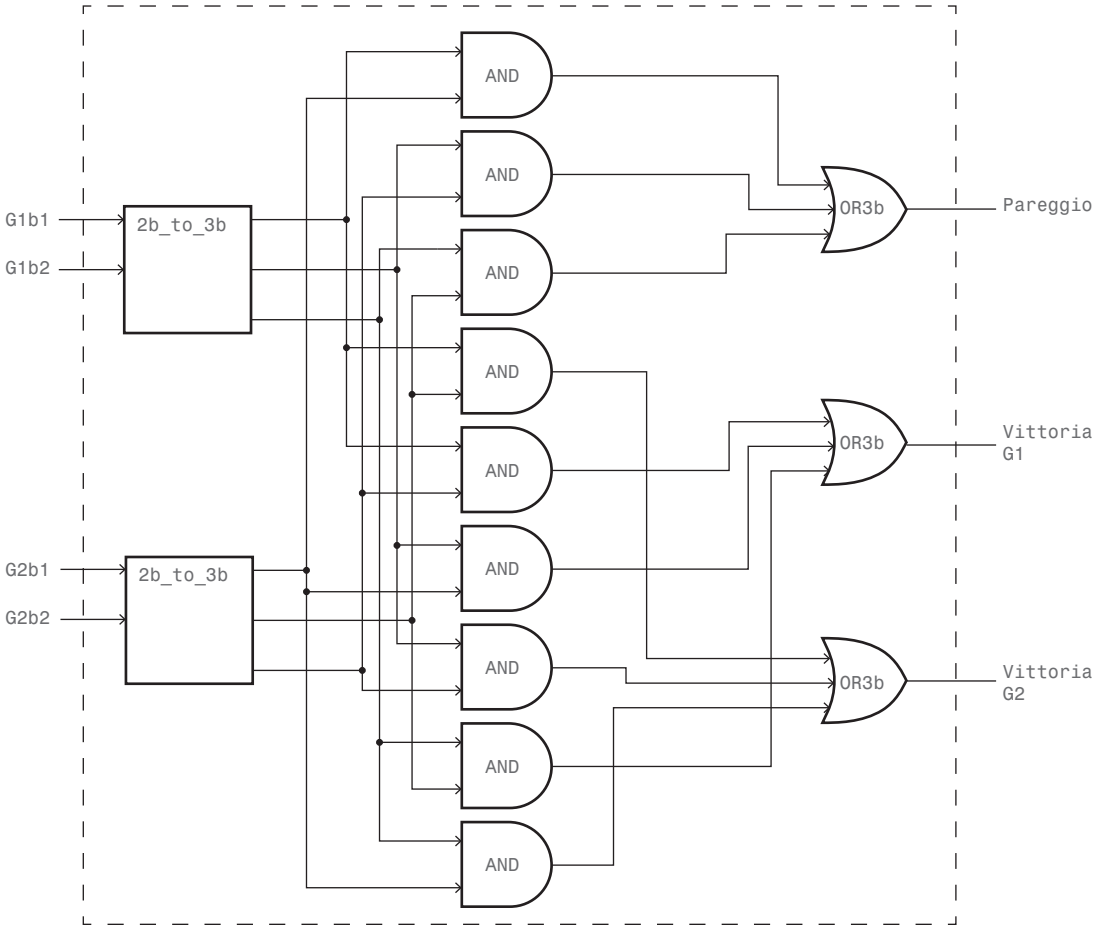

Figura 4.3: Funzionamento del modulo `Partita_lv1.blif`

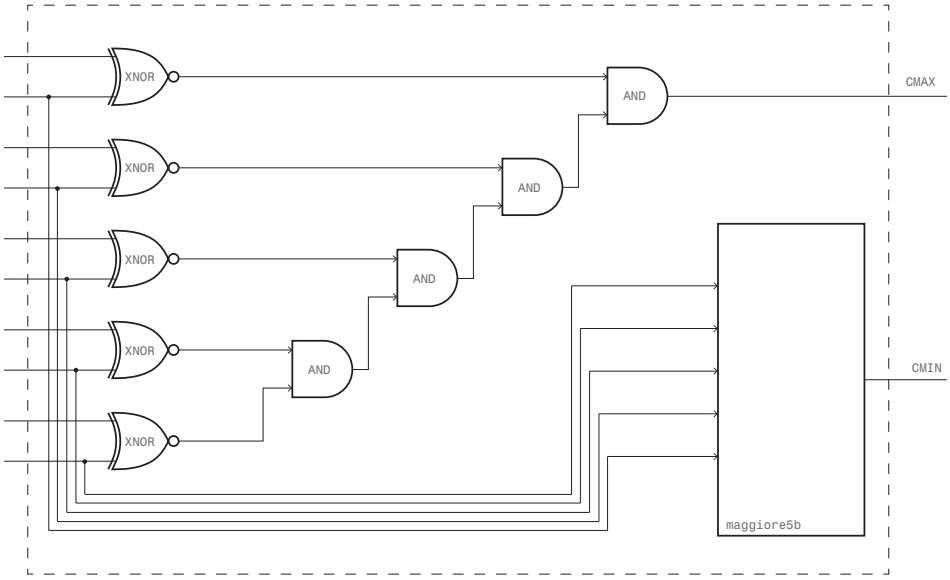

Figura 4.4: Funzionamento del modulo `CMIN_CMAX.blif`

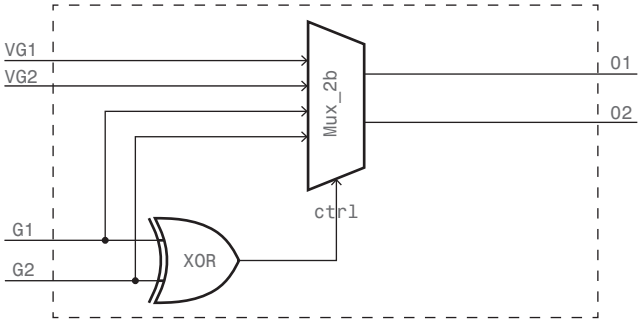

Figura 4.5: Funzionamento del modulo `tavolino.blif`

4.2 Statistiche del circuito e mapping tecnologico

Sono stati effettuati due mapping tecnologici del circuito, uno prima dell'ottimizzazione e uno dopo. L'ottimizzazione ha portato ad una riduzione dei nodi da 474 a 62 e dei letterali (in somma di prodotti) da 969 a 419.

```
sis> print_stats
FSMD            pi= 5   po= 7   nodes=474       latches=16
lits(sop)= 969
```

Figura 4.6: statistiche dopo ottimizzazione

```
sis> print_stats
FSMD            pi= 5   po= 7   nodes= 62       latches=16
lits(sop)= 419
```

Figura 4.7: statistiche prima dell'ottimizzazione

Il modello è stato anche mappato sulla libreria tecnologica `synch.genlib` e sono stati ottenuti i seguenti risultati:

```
sis> read_blif FSMD.blif
sis> read_library synch.genlib
sis> map -m 0 -s
warning: unknown latch type at node '{[45]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[46]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[47]}' (RISING_EDGE assumed)
WARNING: uses as primary input drive the value (0.20,0.20)
WARNING: uses as primary input arrival the value (0.00,0.00)
WARNING: uses as primary input max load limit the value (999.00)
WARNING: uses as primary output required the value (0.00,0.00)
WARNING: uses as primary output load the value 1.00
>>> before removing serial inverters <<<
# of outputs:          23
total gate area:       6088.00
maximum arrival time: (46.00,46.00)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-46.00,-46.00)
total neg slack:      (-593.40,-593.40)
# of failing outputs:  23
>>> before removing parallel inverters <<<
# of outputs:          23
total gate area:       5976.00
maximum arrival time: (43.60,43.60)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-43.60,-43.60)
total neg slack:      (-549.00,-549.00)
# of failing outputs:  23
# of outputs:          23
total gate area:       5800.00
maximum arrival time: (42.80,42.80)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-42.80,-42.80)
total neg slack:      (-542.20,-542.20)
# of failing outputs:  23
sis>
```

Figura 4.8: Statistiche dopo il mapping

Capitolo 5

Scelte progettuali ed eventuali chiarimenti

5.1 Definizione del comportamento in caso di mossa *"sconosciuta"* al datapath

Nel caso in cui uno dei giocatori effettui una mossa "nulla" (non riconosciuta come mossa effettiva dalla codifica del datapath, eg. `PRIMO` o `SECONDO = 00`) durante una manche, l'altro giocatore vincerà automaticamente. Tuttavia, è importante notare che questa regola non si applica nel caso in cui un giocatore, proveniente da una manche precedentemente vincente, utilizzi nuovamente la stessa mossa. In tale situazione, la priorità viene data alla regola presente nelle specifiche che non consente la ripetizione della mossa vincente (per il medesimo giocatore) e rende non valida la manche.

5.2 Comportamento del sistema in relazione al segnale `INIZIA`

Il segnale `INIZIA` è stato implementato come un segnale di reset del sistema. Tuttavia per la sua natura di segnale **in ingresso** la macchina è stata progettata affinchè al termine di una partita l'elaboratore attenda di ricevere in input il segnale `INIZIA` per procedere al reset e iniziare una nuova partita. Al termine di una partia, dopo aver decretato il vincitore (output `PARTITA` $\neq$ `00`), dal ciclo di clock successivo il sistema manderà in output `MANCHE = 00 PARTITA = 00` fintanto che non viene nuovamente attivato lo stato di reset.

5.3 Approfondimento sulla suddivisione delle funzioni tra Datapath e FSM

La scelta di suddividere le funzioni tra Datapath e FSM è stata guidata da un'analisi delle funzioni chiave che dovrebbe svolgere il sistema, cercando di bilanciare la complessità tra il Datapath e il controllore FSM. In particolare alcune funzioni, a nostro avviso, combaciavano con la natura combinatoria del Datapath.

Il confronto delle mosse e la verifica della loro validità, per esempio, coinvolgono principalmente operazioni logiche e confronti diretti tra i dati di input. Queste operazioni sono più naturalmente implementate in un sistema combinatorio, dove le uscite dipendono solo dagli ingressi attuali senza considerare uno stato interno.[1] La progettazione a *gate level* per quanto riguarda la simulazione in SIS, ci ha permesso di gestire il flusso dei dati in maniera puntuale e precisa permettendoci di testare ogni blocco di porte logiche singolarmente per poi unirlo in un sistema più complesso.

[1] Si noti che nonostante nel Datapath vengano elaborati direttamente gli input senza tener conto di un eventuale stato del sistema, questo non implica l'assenza di registri funzionali al confronto con dati elaborati in cicli di clock precedenti al calcolo (come per esempio il confronto di una mossa vincente con quella dello stesso giocatore nella manche precedente).

In un sistema a stati finiti sequenziale il focus principale è sulla gestione dello stato attuale del sistema, gli input ricevuti dal Datapath (nel caso dell'elaborato in questione e in generale sul modello di Mealy) e l'elaborazione dello stato prossimo. Lo spostamento di operazioni logiche complesse dal datapath all'FSM avrebbe reso la logica di controllo inutilmente più complessa, influenzando la chiarezza della progettazione e la manutenibilità del codice. Nel presente elaborato infatti abbiamo optato per una definizione della FSM ad 8 stati, che ci ha permesso di mantenere un controllo preciso sul flusso di gioco e di gestire in modo chiaro i segnali di output mantenendo una logica combinatoria (all'interno del Datapath) chiara e lineare.