

# **Elaborato SIS – Laboratorio Architettura degli Elaboratori**

UniVR - Dipartimento di Informatica

## **Morra Cinese**

**Mattia Arganetto - VR502412**

**Fabio Irimie - VR501504**

1° Semestre 2023/2024

# Indice

<b>1</b>	<b>Specifiche</b>	<b>2</b>
<b>2</b>	<b>Architettura generale</b>	<b>3</b>
<b>3</b>	<b>Diagramma degli stati del controllore</b>	<b>4</b>
<b>4</b>	<b>Architettura del datapath</b>	<b>6</b>
<b>5</b>	<b>Simulazioni</b>	<b>10</b>
5.1	SIS . . . . .	11
5.2	Verilog . . . . .	12
5.3	Confronto . . . . .	13
<b>6</b>	<b>Statistiche del circuito</b>	<b>14</b>
6.1	Prima della minimizzazione . . . . .	14
6.2	Dopo la minimizzazione . . . . .	15
6.2.1	Minimizzazione degli stati . . . . .	15
6.2.2	Minimizzazione per area . . . . .	15
<b>7</b>	<b>Numero di gate e ritardo</b>	<b>16</b>
7.1	Prima della minimizzazione . . . . .	16
7.2	Dopo la minimizzazione per area . . . . .	17
<b>8</b>	<b>Scelte progettuali</b>	<b>17</b>
8.1	FSM . . . . .	17
8.2	Datapath . . . . .	18
8.2.1	Concatenatori . . . . .	18
8.2.2	Componenti riutilizzati . . . . .	18
8.3	Warning . . . . .	18

# 1 Specifiche

Si progetti un dispositivo per la gestione di partite di **Morra Cinese**, conosciuta anche come "sasso-carta-forbici". Il dispositivo dovrà essere modellato come un circuito sequenziale FSM (Finite State Machine + Datapath) in Sis e Verilog. Si considerino due giocatori che inseriscono una mossa che può essere sasso, carta o forbici. Ogni partita è composta da più manche e ad ogni manche il giocatore vincente è decretato dalle seguenti regole:

- Sasso batte Forbici
- Forbici batte Carta
- Carta batte Sasso

Nell'eventualità che i due giocatori scelgano la stessa mossa verrà decretato un pareggio. In aggiunta, ogni partita di **Morra Cinese** si articola su più manche, con le seguenti regole:

1. Si devono giocare un **minimo** di **quattro manche**;
2. Si possono giocare un **massimio** di **diciannove manche**. Il numero delle stesse viene settato al ciclo di clock in cui viene iniziata la partita;
3. Vince il primo giocatore che riesce a **vincere due manche in più del proprio avversario** (avendo giocato le quattro manche **minime**);
4. Ad ogni manche, il giocatore **vincente della manche precedente** è tenuto a non ripetere l'ultima mossa utilizzata. Nel caso lo facesse (indipendentemente dal risultato della manche attuale) la manche non sarebbe valida (non sarebbe conteggiata nel **mancheIdx**) e andrebbe quindi ripetuta;
5. Ad ogni manche, in caso di pareggio essa **viene conteggiata**. Alla manche successiva entrambi i giocatori possono usare **tutte le mosse**;

Il circuito ha **tre ingressi**:

- **PRIMO [2 bit]**: mossa selezionata dal primo giocatore. Le mosse hanno i seguenti codici:
  - **00**: Nessuna mossa utilizzata;
  - **01**: Sasso;
  - **10**: Carta;
  - **11**: Forbici;
- **SECONDO [2 bit]**: mossa selezionata dal secondo giocatore. Le mosse hanno codici identici a quelli del primo giocatore.
- **INIZIA [1 bit]**: quando il valore è uguale ad 1, riporta il sistema alla configurazione iniziale. Inoltre, la concatenazione degli ingressi **PRIMO** e **SECONDO** viene utilizzata per stabilire il numero massimo di manche (le **quattro manche** obbligatorie sommate al **valore concatenato di PRIMO e SECONDO**). Per fare un esempio inserendo i valori **PRIMO**

= **01** e **SECONDO** = **10** si dovrà sommare il numero **quattro** (in base due) al numero **0110** ottenendo un massimo di **dieci manche** per tale partita.

Il circuito ha **due uscite**:

- **MANCHE** [2 bit]: fornisce in output il risultato dell'ultima manche giocata con la seguente codifica:
  - **00**: manche non valida;
  - **01**: manche vinta dal giocatore 1;
  - **10**: manche vinta dal giocatore 2;
  - **11**: manche pareggiata;
- **PARTITA** [2 bit]: fornisce in output il risultato della partita con la seguente codifica:
  - **00**: la partita non è ancora terminata;
  - **01**: la partita è terminata, vittoria del **giocatore 1**;
  - **10**: la partita è terminata, vittoria del **giocatore 2**;
  - **11**: la partita è terminata in pareggio;

## 2 Architettura generale

Il circuito è costituito da FSM (controllore) e Datapath (elaboratore). Nella figura 1 è rappresentata l'architettura generale del circuito che prende in input i segnali **primo**, **secondo** e **inizia** condivisi tra FSM e Datapath e restituisce in output i segnali **manche** (dalla FSM) e **partita** (dal Datapath). Il segnale di clock è lo stesso per entrambi i componenti.

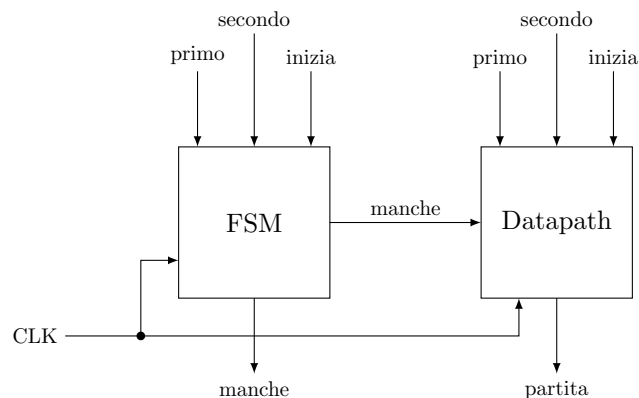


Figura 1: Architettura generale della FSMD

### 3 Diagramma degli stati del controllore

È riportata nella tabella 1 la State Transition Table del controllore con la rappresentazione di Mealy. Gli ingressi sono codificati nel seguente modo per agevolare la lettura della tabella:

- **Nessuna scelta:**  $N = 00$
- **Sasso:**  $S = 01$
- **Carta:**  $C = 10$
- **Forbice:**  $F = 11$

e gli stati codificati nel seguente modo:

- **Par:** Pareggio
- **PrS:** Primo giocatore vince con sasso
- **PrC:** Primo giocatore vince con carta
- **PrF:** Primo giocatore vince con forbice
- **SeS:** Secondo giocatore vince con sasso
- **SeC:** Secondo giocatore vince con carta
- **SeF:** Secondo giocatore vince con forbice

Dato l'elevato numero di ingressi nella seguente tabella è stato deciso di mettere in riga gli stati e in colonna gli ingressi per una migliore leggibilità.

	Par	PrS	PrC	PrF	SeS	SeC	SeF
- - 1	Par/00	Par/00	Par/00	Par/00	Par/00	Par/00	Par/00
N-0	Par/00	PrS/00	PrC/00	PrF/00	SeS/00	SeC/00	SeF/00
-N0	Par/00	PrS/00	PrC/00	PrF/00	SeS/00	SeC/00	SeF/00
SS0	Par/11	PrS/00	Par/11	Par/11	SeS/00	Par/11	Par/11
SC0	SeC/10	PrS/00	SeC/10	SeC/10	SeC/10	SeC/00	SeC/10
SF0	PrS/01	PrS/00	PrS/01	PrS/01	PrS/01	Prs/01	SeF/00
CS0	PrC/01	PrC/01	PrC/00	PrC/01	SeS/00	PrC/01	PrC/01
CC0	Par/11	Par/11	PrC/00	Par/11	Par/11	SeC/00	Par/11
CF0	SeF/10	SeF/10	PrC/00	SeF/10	SeF/10	SeF/10	SeF/00
FS0	SeS/10	SeS/10	SeS/10	PrF/00	SeS/00	SeS/10	SeS/10
FC0	PrF/01	PrF/01	PrF/01	PrF/00	PrF/01	SeC/00	PrF/01
FF0	Par/11	Par/11	Par/11	PrF/00	Par/11	Par/11	SeF/00

Tabella 1: State Transition Table del controllore (Mealy)

Nella figura 2 è rappresentato il State Transition Graph del controllore. Per una migliore leggibilità sono stati omessi gli archi che portano da uno stato in cui vince uno dei due giocatori a tutti gli altri stati in cui avrebbe vinto un giocatore, si è deciso invece di riportare il nome dello stato prossimo una seconda volta.

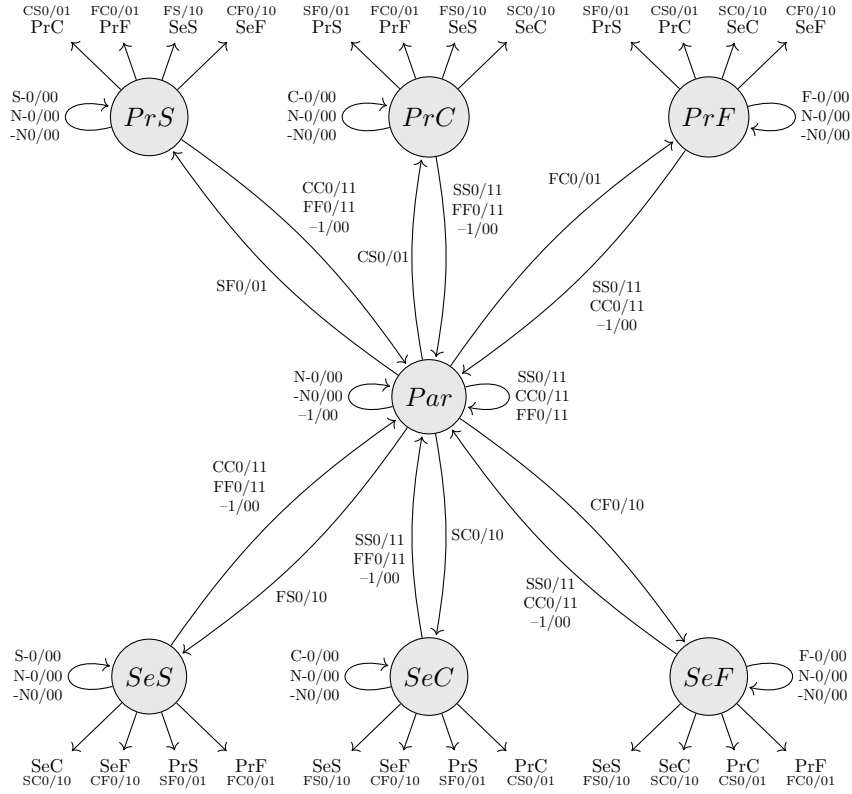


Figura 2: State Transition Graph del controllore

## 4 Architettura del datapath

I componenti che sono stati utilizzati per l'elaborazione del circuito sono i seguenti, presentati sottoforma di **Nome Componente** e **Numero di Occorrenze**.

- **Registro a 5 bit: 2**

Utilizzati per salvare in memoria i valori di:

- **mancheIdx**: Indice della manche corrente;
- **maxManche**: Numero massimo di manche che possono essere giocate in una singola partita;

- **Registro a 4 bit: 1**

Utilizzato per salvare in memoria il valore di **vantaggio**. Il vantaggio è codificato in complemento a 2 in modo da poter calcolare il vantaggio di entrambi i giocatori con un singolo registro. Quando il vantaggio assume valori positivi è il giocatore 1 ad avere il vantaggio, quando assume valori negativi è il giocatore 2 ad avercelo.

- **Mux a 4 Ingressi da 5 bit: 1**

Utilizzato per incrementare il **mancheIdx** durante la partita solo nel caso in cui la manche risulti valida. Il Mux dà in output 00001 se la manche è valida, 00000 altrimenti.

- **Mux a 4 Ingressi da 4 bit: 1**

Utilizzato per incrementare o decrementare il valore di **vantaggio**. Il Mux dà in output 0001 se il giocatore 1 ha vinto la manche, 1111 se il giocatore 2 ha vinto la manche, 0000 altrimenti. Visto che **vantaggio** è codificato in complemento a 2, il valore 1111 equivale a  $-1_{10}$ .

- **Mux a 2 Ingressi da 5 bit: 2**

Utilizzati per decidere quale valore inserire all'interno dei registri di **mancheIdx** e **maxManche**. Il Mux inizializza i registri con il valore 00000 se il valore di **inizia** è uguale a 1, altrimenti inserisce il nuovo valore di **mancheIdx** e **maxManche** in input.

- **Mux a 2 Ingressi da 4 bit: 1**

Utilizzato per decidere quale valore inserire all'interno del registro di **vantaggio**. Il Mux inizializza il registro con il valore 0000 se il valore di **inizia** è uguale a 1, altrimenti inserisce il nuovo valore di **vantaggio** in input.

- **Mux a 2 Ingressi da 2 bit: 2**

Utilizzati nel componente **Partita**. I due Mux insieme determinano quale valore di **partita** dovrà essere stampato in **output**.

- **Sommatore a 5 bit: 3**

Utilizzati per sommare 2 numeri e fornire in output un numero a 5 bit. Il loro scopo è determinare e modificare il valore di **maxManche** e **mancheIdx**.

- **Sommatore a 4 bit: 1**

Utilizzato per sommare 2 numeri e fornire in output un numero a 4 bit. La sua funzione è modificare il valore di **vantaggio**.

- **And: 2**

Utilizzati per verificare se due condizioni sono vere allo stesso tempo. Si trovano nel componente **Partita** del Datapath.

- **Maggiore uguale da 5 bit: 2**

Utilizzati rispettivamente per confrontare il valore di **mancheIdx** e **maxManche**, e per controllare se il valore di **mancheIdx** è maggiore o uguale a 4.

- **Maggiore uguale da 4 bit: 4**

Utilizzati per comparare il valore di **vantaggio** con alcune **Costanti**. Questo componente è stato progettato per tenere in considerazione la codifica in complemento a 2 di **vantaggio**.



- Utilizzato per concatenare il valore di **primo** e **secondo** in modo da poter avere un unico valore a 5 bit (aggiungendo uno 0 come bit più significativo) per poter calcolare **maxManche**.

- Utilizzati per concatenare i valori di controllo per determinare il vincitore della partita.

The diagram illustrates the hardware architecture for the game of Partita, showing two parallel processing paths for 'manche' and 'partita'.

**Manche Path:**

- Inputs:** A 2-bit input labeled 'manche' and a 5-bit input labeled '0000 0001 0001 0001'.
- ROM:** A 4-bit ROM with outputs 00, 01, 10, and 11.
- Calculation:** The 5-bit input is added to the ROM output (00) to produce a 5-bit result.
- Register:** The result is stored in a register labeled 'mancheIdx'.
- Output:** The output of 'mancheIdx' is a 5-bit signal labeled 'CLK'.

**Partita Path:**

- Inputs:** A 2-bit input labeled 'partita' and a 4-bit input labeled '0000 0001 1111 0000'.
- ROM:** A 4-bit ROM with outputs 00, 01, 10, and 11.
- Calculation:** The 4-bit input is added to the ROM output (00) to produce a 4-bit result.
- Register:** The result is stored in a register labeled 'vantaggio'.
- Output:** The output of 'vantaggio' is a 4-bit signal labeled 'CLK'.

**Common Components:**

- Counter:** A counter labeled 'C' receives a 5-bit input '00100' and outputs a 5-bit signal.
- MaxManche:** A register labeled 'maxManche' receives a 5-bit input and outputs a 5-bit signal.
- Partita:** A final output block labeled 'Partita' receives a 2-bit input and outputs a 2-bit signal labeled 'partita'.

Per una maggiore leggibilità del circuito è stata modellata una parte all'interno del componente **Partita** che riceve in input:

- vantaggio;
- mancheIdx;
- maxManche;
- inizia;

8

- partita.

Nella figura 4 è rappresentato il componente **Partita**.

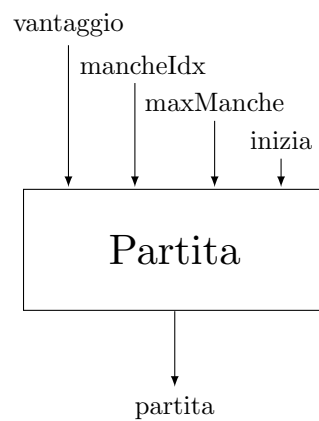


Figura 4: Componente Partita

Nella figura 5 è rappresentato il datapath del componente **Partita**.

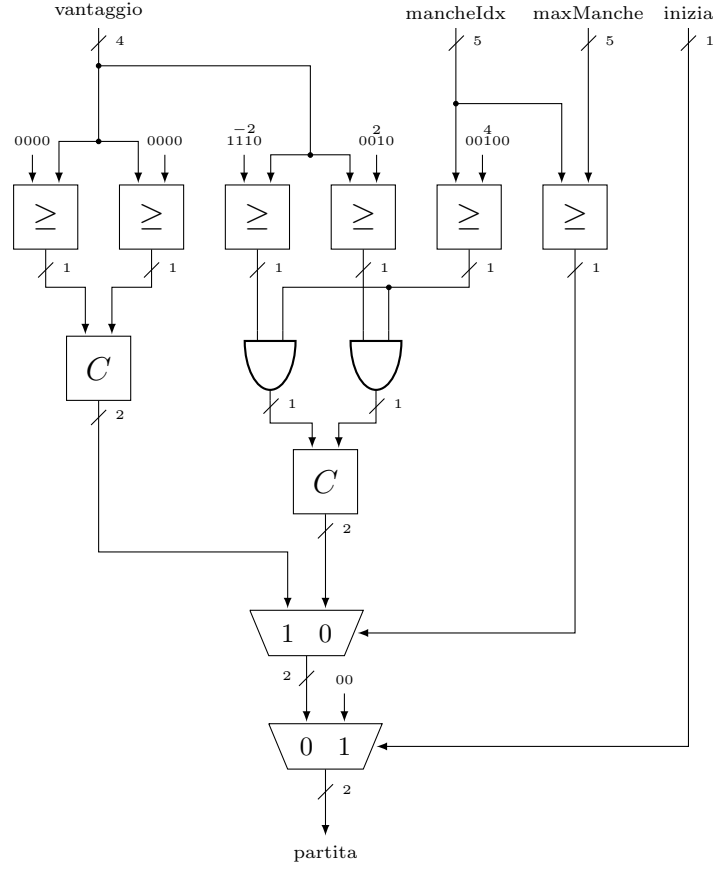


Figura 5: Architettura del componente Partita

## 5 Simulazioni

Gli ingressi sono: **primo[1]**, **primo[0]**, **secondo[1]**, **secondo[0]**, **inizia**.

Le uscite sono: **manche[1]**, **manche[0]**, **partita[1]**, **partita[0]**.

Abbiamo svolto una singola simulazione prendendo in considerazione **quattro partite**, ognuna con un diverso numero di **manche massime**. Questa simulazione è stata generata dal testbench in verilog in modo da verificare che gli output fossero identici tra il modello in verilog e quello in sis.

Le partite sono svolte nel seguente modo:

1. Partita da 6 manche massime finita in pareggio;
2. Partita da 5 manche massime vinta dal giocatore 2 con vantaggio  $-1$ ;
3. Partita da 19 manche massime vinta dal giocatore 1 con vantaggio  $+2$ ;
4. Partita da 9 manche massime vinta dal giocatore 1 con vantaggio  $+2$ ;

Partita 1			
Manche massime: 6			
#	Primo	Secondo	Van.
1	Sasso	Carta	-1
(1)	Forbici	Carta	-1
2	Sasso	Forbici	0
3	Forbici	Forbici	0
4	Forbici	Carta	1
5	Sasso	Carta	0
6	Forbici	Forbici	0
Pareggio			

Partita 2			
Manche massime: 5			
#	Primo	Secondo	Van.
1	Sasso	Forbici	1
2	Carta	Sasso	2
3	Sasso	Carta	1
4	Carta	Forbici	0
5	Forbici	Sasso	-1
Vittoria: <b>Secondo</b>			

Partita 3			
Manche massime: 19			
#	Primo	Secondo	Van.
1	Sasso	Forbici	1
2	Carta	Sasso	2
3	Forbici	Carta	3
(3)	Forbici	Sasso	3
4	Sasso	Carta	2
Vittoria: <b>Primo</b>			

Partita 4			
Manche massime: 9			
#	Primo	Secondo	Van.
1	Sasso	Forbici	1
2	Carta	Sasso	2
3	Forbici	Sasso	1
4	Carta	Carta	1
5	Forbici	Sasso	0
6	Sasso	Forbici	1
(6)	Sasso	Carta	1
(6)	Null	Forbici	1
(6)	Sasso	Forbici	1
7	Forbici	Forbici	1
8	Carta	Sasso	2
Vittoria: <b>Primo</b>			

## 5.1 SIS

Di seguito sono riportati i file di input e di output di sis per la simulazione.

testbench.script:

```
read_blif FSM.D.blif
simulate 0 0 1 0 1
simulate 0 1 1 0 0
simulate 1 1 1 0 0
simulate 0 1 1 1 0
simulate 1 1 1 1 0
simulate 1 1 1 0 0
simulate 0 1 1 0 0
simulate 1 1 1 1 0
simulate 0 0 0 1 1
simulate 0 1 1 1 0
simulate 1 0 0 1 0
simulate 0 1 1 0 0
simulate 1 0 1 1 0
simulate 1 1 0 1 0
simulate 1 1 1 1 1
simulate 0 1 1 1 0
simulate 1 0 0 1 0
simulate 1 1 1 0 0
simulate 1 1 0 1 0
simulate 0 1 1 0 0
simulate 0 1 0 1 1
simulate 0 1 1 1 0
simulate 1 0 0 1 0
simulate 1 1 0 1 0
simulate 1 0 1 0 0
simulate 1 1 0 1 0
simulate 0 1 1 1 0
simulate 0 1 1 0 0
simulate 0 0 1 1 0
simulate 0 1 1 1 0
simulate 1 1 1 1 0
simulate 1 0 0 1 0
quit
```

output\_sis.txt:

```
Outputs: 0 0 0 0
Outputs: 1 0 0 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 1 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 0 0 0
Outputs: 1 0 1 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 0 0 0
Outputs: 1 0 1 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 0 0
Outputs: 1 0 0 0
Outputs: 0 1 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 1
```

## 5.2 Verilog

Nel testbench in verilog sono stati inseriti dei `$display()` che mostrano i valori delle variabili in ogni ciclo di clock. Di seguito è riportato l'output dei display nel testbench in verilog:

```
# run -all
# Partita 1
# Primo: 00 Secondo: 10 Manche: 00 Partita: 00
# Primo: 01 Secondo: 10 Manche: 10 Partita: 00
# Primo: 11 Secondo: 10 Manche: 00 Partita: 00
# Primo: 01 Secondo: 11 Manche: 01 Partita: 00
# Primo: 11 Secondo: 11 Manche: 11 Partita: 00
# Primo: 11 Secondo: 10 Manche: 01 Partita: 00
```

```

# Primo: 01 Secondo: 10 Manche: 10 Partita: 00
# Primo: 11 Secondo: 11 Manche: 11 Partita: 11
#
# Partita 2
# Primo: 00 Secondo: 01 Manche: 00 Partita: 00
# Primo: 01 Secondo: 11 Manche: 01 Partita: 00
# Primo: 10 Secondo: 01 Manche: 01 Partita: 00
# Primo: 01 Secondo: 10 Manche: 10 Partita: 00
# Primo: 10 Secondo: 11 Manche: 10 Partita: 00
# Primo: 11 Secondo: 01 Manche: 10 Partita: 10
#
# Partita 3
# Primo: 11 Secondo: 11 Manche: 00 Partita: 00
# Primo: 01 Secondo: 11 Manche: 01 Partita: 00
# Primo: 10 Secondo: 01 Manche: 01 Partita: 00
# Primo: 11 Secondo: 10 Manche: 01 Partita: 00
# Primo: 11 Secondo: 01 Manche: 00 Partita: 00
# Primo: 01 Secondo: 10 Manche: 10 Partita: 01
#
# Partita 4
# Primo: 01 Secondo: 01 Manche: 00 Partita: 00
# Primo: 01 Secondo: 11 Manche: 01 Partita: 00
# Primo: 10 Secondo: 01 Manche: 01 Partita: 00
# Primo: 11 Secondo: 01 Manche: 10 Partita: 00
# Primo: 10 Secondo: 10 Manche: 11 Partita: 00
# Primo: 11 Secondo: 01 Manche: 10 Partita: 00
# Primo: 01 Secondo: 11 Manche: 01 Partita: 00
# Primo: 01 Secondo: 10 Manche: 00 Partita: 00
# Primo: 00 Secondo: 11 Manche: 00 Partita: 00
# Primo: 01 Secondo: 11 Manche: 00 Partita: 00
# Primo: 11 Secondo: 11 Manche: 11 Partita: 00
# Primo: 10 Secondo: 01 Manche: 01 Partita: 01
# ** Note: $finish      : testbench.sv(381)

```

### 5.3 Confronto

Di seguito è riportato il confronto tra i due output. Si nota che sono identici perchè il modello in sis è stato progettato in modo da rispettare il modello in Verilog.

output\_sis.txt:

```
Outputs: 0 0 0 0
Outputs: 1 0 0 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 1 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 0 0 0
Outputs: 1 0 1 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 0 0 0
Outputs: 1 0 0 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 0 0
Outputs: 1 0 0 0
Outputs: 0 1 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 1
```

output\_verilog.txt:

```
Outputs: 0 0 0 0
Outputs: 1 0 0 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 1 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 0 0 0
Outputs: 1 0 1 0
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 0 0 0 0
Outputs: 1 0 0 1
Outputs: 0 0 0 0
Outputs: 0 1 0 0
Outputs: 0 1 0 0
Outputs: 1 0 0 0
Outputs: 1 1 0 0
Outputs: 1 0 0 0
Outputs: 0 1 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 0 0 0 0
Outputs: 1 1 0 0
Outputs: 0 1 0 1
```

## 6 Statistiche del circuito

### 6.1 Prima della minimizzazione

Le statistiche della FSM prima della minimizzazione sono le seguenti:

```
sis> print_stats
controllo      pi= 5   po= 2   nodes= 2       latches= 0
lits(sop)=    0   #states(STG)= 7
sis> state_assign jedi
Running jedi, written by Bill Lin, UC Berkeley
sis> print_stats
controllo      pi= 5   po= 2   nodes= 5       latches= 3
lits(sop)= 379   #states(STG)= 7
```

Le statistiche del Datapath prima della minimizzazione sono le seguenti:

```
sis> print_stats
elaborazione    pi= 7    po= 2    nodes=150    latches=14
lits(sop)= 606
```

Le statistiche della FSM prima della minimizzazione sono le seguenti:

```
sis> print_stats
MorraCinese     pi= 5    po= 4    nodes=155    latches=17
lits(sop)= 985
```

## 6.2 Dopo la minimizzazione

### 6.2.1 Minimizzazione degli stati

```
sis> read_blif controllo.blif
sis> state_minimize stamina
Running stamina, written by June Rho, University of Colorado at Boulder
Number of states in original machine : 7
Number of states in minimized machine : 7
sis> state_assign jedi
Running jedi, written by Bill Lin, UC Berkeley
sis> stg_to_network
sis> print_stats
controllo       pi= 5    po= 2    nodes= 5    latches= 3
lits(sop)= 379 #states(STG)= 7
sis> full_simplify
sis> print_stats
controllo       pi= 5    po= 2    nodes= 5    latches= 3
lits(sop)= 152 #states(STG)= 7
sis> source script.rugged
sis> print_stats
controllo       pi= 5    po= 2    nodes= 11    latches= 3
lits(sop)= 109 #states(STG)= 7
sis> write_blif controlloMinimizzato.blif
```

### 6.2.2 Minimizzazione per area

Dopo aver minimizzato gli stati della FSM, abbiamo minimizzato il circuito dell'elaborazione eseguendo le seguenti operazioni:

```
sis> print_stats
elaborazione    pi= 7    po= 2    nodes=150    latches=14
lits(sop)= 606
sis> full_simplify
sis> print_stats
elaborazione    pi= 7    po= 2    nodes=150    latches=14
lits(sop)= 231
sis> source script.rugged
sis> print_stats
```



```

elaborazione    pi= 7    po= 2    nodes= 33    latches=14
lits(sop)= 188
sis> write_blif elaborazioneMinimizzato.blif

```

Dopo aver minimizzato sia la FSM che il Datapath, abbiamo minimizzato il circuito finale eseguendo le seguenti operazioni:

```

sis> print_stats
MorraCinese     pi= 5    po= 4    nodes= 44    latches=17
lits(sop)= 297
sis> full_simplify
sis> print_stats
MorraCinese     pi= 5    po= 4    nodes= 44    latches=17
lits(sop)= 285

```

Non abbiamo eseguito anche lo script *script.rugged* in quanto avremmo ottenuto un letterale e un nodo in più rispetto alla sola *full\_simplify*.

## 7 Numero di gate e ritardo

Di seguito sono riportate le statistiche del circuito implementato in sis riguardanti il numero di gate e il ritardo.

### 7.1 Prima della minimizzazione

Le statistiche di numero di gate e ritardo della FSMD prima della minimizzazione sono le seguenti:

```

sis> read_library synch.genlib
sis> map -m 0 -s -W
>>> before removing serial inverters <<<
# of outputs:      21
total gate area:    7268.00
maximum arrival time: (39.80,39.80)
maximum po slack:   (-3.40,-3.40)
minimum po slack:   (-39.80,-39.80)
total neg slack:    (-455.40,-455.40)
# of failing outputs: 21
>>> before removing parallel inverters <<<
# of outputs:      21
total gate area:    7268.00
maximum arrival time: (39.80,39.80)
maximum po slack:   (-3.40,-3.40)
minimum po slack:   (-39.80,-39.80)
total neg slack:    (-455.40,-455.40)
# of failing outputs: 21
# of outputs:      21
total gate area:    7044.00
maximum arrival time: (39.80,39.80)
maximum po slack:   (-3.40,-3.40)
minimum po slack:   (-39.80,-39.80)

```

```
total neg slack:      (-457.40,-457.40)
# of failing outputs: 21
```

Le statistiche di numero di gate e ritardo del Datapath prima della minimizzazione sono le seguenti:

- **Total gate area:** 7044 ;
- **Maximum arrival time (cammino critico):** 39.80 ;

## 7.2 Dopo la minimizzazione per area

Le statistiche di numero di gate e ritardo della FSM dopo la minimizzazione per area sono le seguenti:

```
sis> read_library synch.genlib
sis> map -m 0 -s -W
>>> before removing serial inverters <<<
# of outputs:      21
total gate area:    5564.00
maximum arrival time: (46.20,46.20)
maximum po slack:   (-3.20,-3.20)
minimum po slack:   (-46.20,-46.20)
total neg slack:    (-534.60,-534.60)
# of failing outputs: 21
>>> before removing parallel inverters <<<
# of outputs:      21
total gate area:    5500.00
maximum arrival time: (46.20,46.20)
maximum po slack:   (-3.20,-3.20)
minimum po slack:   (-46.20,-46.20)
total neg slack:    (-534.60,-534.60)
# of failing outputs: 21
# of outputs:      21
total gate area:    5244.00
maximum arrival time: (44.40,44.40)
maximum po slack:   (-3.20,-3.20)
minimum po slack:   (-44.40,-44.40)
total neg slack:    (-508.60,-508.60)
# of failing outputs: 21
```

Le statistiche di numero di gate e ritardo del Datapath dopo la minimizzazione per area sono le seguenti:

- **Total gate area:** 5244 ;
- **Maximum arrival time (cammino critico):** 44.40 ;

# 8 Scelte progettuali

## 8.1 FSM

Nello sviluppo di questo circuito abbiamo pensato di implementare il calcolo della manche all'interno della FSM in modo da avere un circuito più sempli-

ce e con meno componenti, risparmiandoci così di memorizzare all'interno del Datapath il vincitore di ogni manche e la mossa che ha usato per vincere.

## 8.2 Datapath

### 8.2.1 Concatenatori

I componenti dei concatenatori sono stati inseriti nel datapath a scopo puramente illustrativo, in quanto non sono stati utilizzati per la realizzazione del circuito siccome in sis è possibile gestire i bit singolarmente e quindi inserirli direttamente nei componenti che li utilizzano nell'ordine desiderato. In verilog, invece, è possibile utilizzare l'operatore di concatenazione.

### 8.2.2 Componenti riutilizzati

Per la realizzazione del datapath abbiamo cercato di riutilizzare il più possibile i componenti già realizzati per il circuito evitando di crearne di nuovi, ad esempio abbiamo utilizzato solo il componente **Maggiore Uguale** per tutti i controlli necessari all'interno del componente **Partita**.

Per quanto riguarda il resto dei componenti abbiamo dovuto crearne di nuovi in quanto il numero di bit del registro **vantaggio** e il numero di bit dei registri **mancheIdx** e **maxManche** sono diversi. Avremmo potuto considerare **vantaggio** da 5 bit per poter riutilizzare componenti già presenti, ma abbiamo preferito ottimizzare il circuito scegliendo il minor numero di bit necessari per ogni componente.

## 8.3 Warning

Simulando il circuito in sis vengono generati dei warning riguardanti le uscite che non vengono utilizzate, come ad esempio i riporti dei sommatore. Questi warning possono essere ignorati in quanto non influiscono sul funzionamento del circuito.