Elaborato SIS/VERILOG
Laboratorio di Architettura degli Elaboratori

Hamdane, Walid VR500517

Andrzejak, Patryk VR500315

Anno Accademico 2023/2024

Contents

1 Specifiche

Viene chiesto di sviluppare in SIS/VERILOG un dispositivo di gestione della **morra cinese** con le regole classiche. Ogni partita si articola in più manche, il numero di manche viene determinato a partire da un minimo di quattro, viene poi sommato il numero ottenuto dalla concatenazione degli input dei giocatori al primo turno, quidni fino ad un massimo di 19 (1111 = 15 +4 = 19). Vince il primo giocatore che passate le 4 manche minime ottiene un vantaggio di 2 sull'avversario. Viene inoltre aggiunta una limitazione per cui il giocatore vincente di una manche non può riutilizzare la stessa mossa alla manche successiva.

Il circuito ha 3 ingressi:

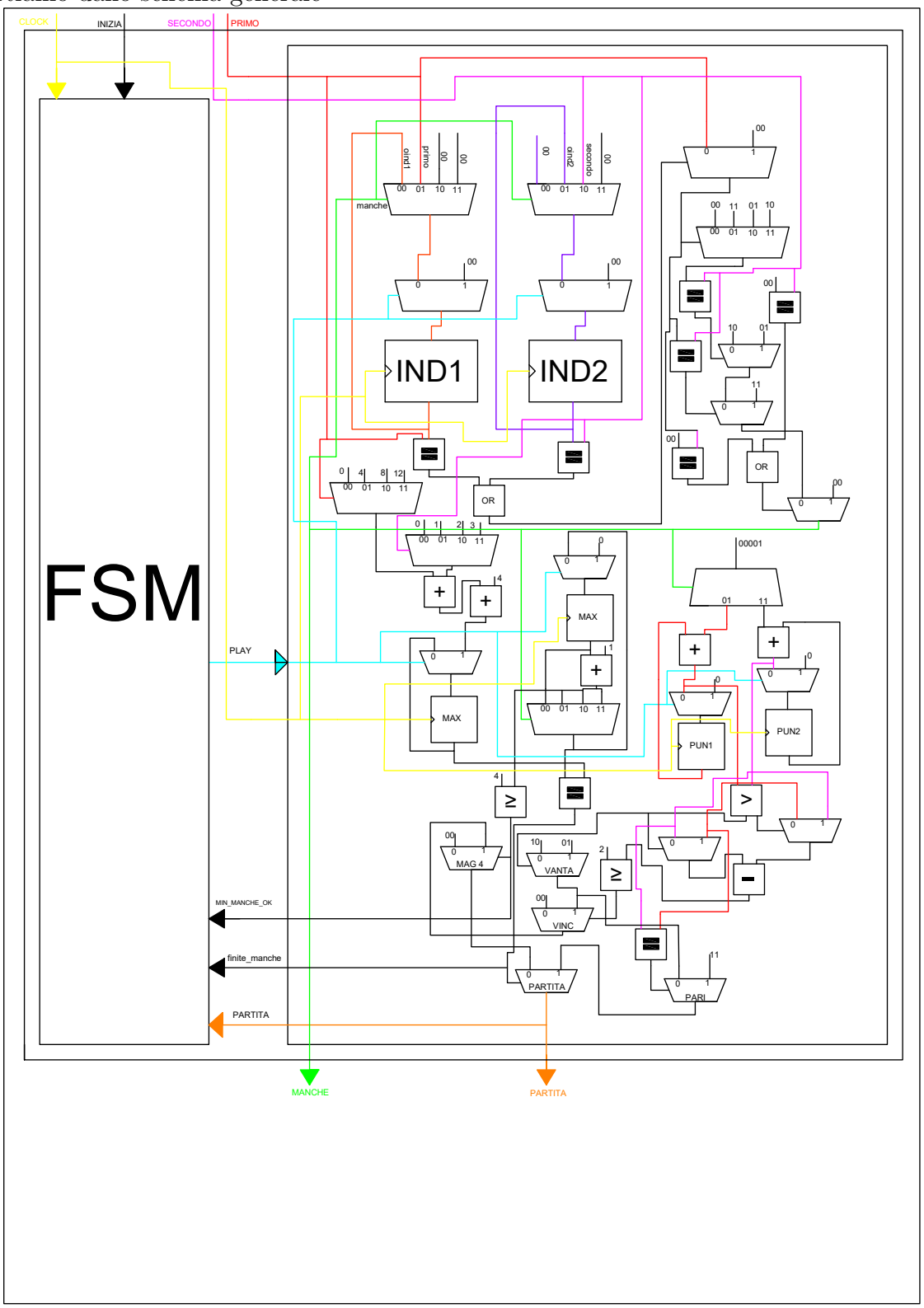
- PRIMO [2] Mossa scelta dal giocatore 1 con la seguente codifica:
  - **00** NESSUNA MOSSA.
  - **01** SASSO.
  - **10** CARTA.
  - **11** FORBICE.
- SECONDO [2] Mossa scelta dal giocatore 2 con la seguente codifica:
  - **00** NESSUNA MOSSA.
  - **01** SASSO.
  - **10** CARTA.
  - **11** FORBICE.
- INIZIO [1] Se 1 riporta il sistema allo stato iniziale.

Il circuito ha 2 uscite:

- MANCHE [2] risultato manche.
  - **00** MANCHE NON VALIDA.
  - **01** MANCHE VINTA DA P1.
  - **10** MANCHE VINTA DA P2.
  - **11** MANCHE PARI.
- PARTITA [2] risultato partita
  - **00** PARTITA NON FINITA.
  - **01** PARTITA VINTA DA P1.
  - **10** PARTITA VINTA DA P2.
  - **11** PARTITA PARI.

2 Progettazione e scelte

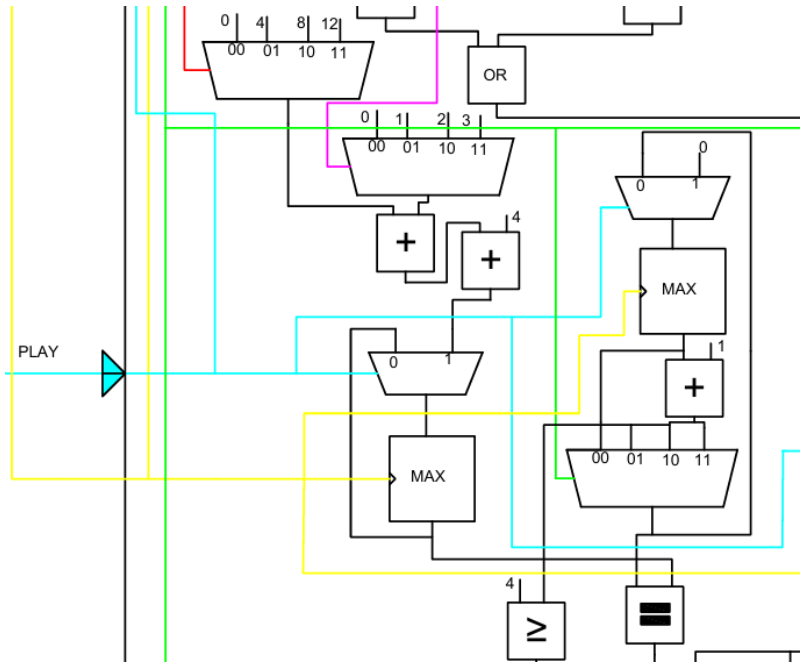
Partiamo dallo schema generale



Il progetto include una FSM minima dato che la maggior parte dell'elaborazione viene lasciata al DATAPATH, data la dimensione per comprenderlo meglio dividiamo concettualmente lo schema in 4 parti :

- COUNTER MANCHE
- CALCOLO COMBINATORIO MANCHE
- STORING PUNTEGGI
- CONTROLLO CONDIZIONI FINE PARTITA

2.1 COUNTER MANCHE



I componenti principali sono:

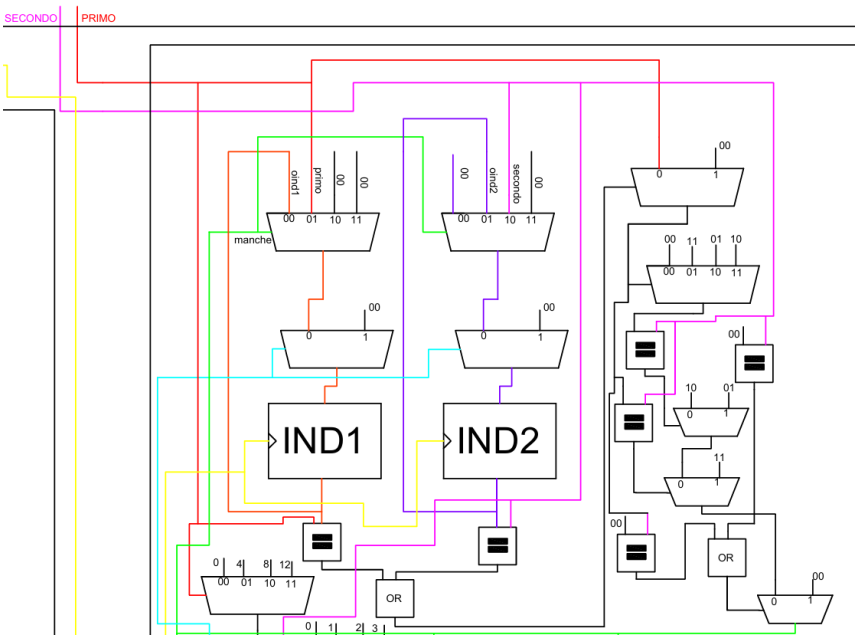
- 2 REGISTRI 5 bit, uno contiene in ogni momento il valore della manche massime inizializzate nel primo ciclo, legge solo quando il **multiplexer** sopra riceve 1 dal valore PLAY (gestito da FSM), il secondo viene azzerato da un **multiplexer** a inizio partita (INIZIA) e viene incrementato con la manche valida che selezione da un multiplexer che in caso di manche non valida 00, ritorna il valore non incrementato al registro.

- 2 MULTIPLEXER, multiplexer con 4 ingressi da 5 bit, gli ingressi sono costanti assegnate, in base al valore di PRIMO e SECONDO escono valori diversi, per PRIMO è il valore di primo seguito da tutti 0, per SECONDO è il valore di SECONDO anticipato da 3 0, in questo modo riusciamo a passare da ingressi a 2 bit ai valori da concatenare che devono essere a 5 bit, passano per un primo **sommatore**, per ottenere la concatenazione, poi in un **secondo sommatore** che somma la costante 4.

- COMPARATORE, confronta i valori dei due registri e ritorna 1 quando si arriva a fine partita per raggiungimento manche massime

- OPERATORE CONFRONTO, ritorna 1 quando si raggiungono le 4 manche minime.

2.2 CALCOLO COMBINATORIO MANCHE



Divisibile a sua volta in 2 parti la prima (destra dell'immagine) calcola il vincitore della manche con la seguente logica:

1. PRIMO fa da selettore ad un multiplexer con 4 ingressi a 2 bit, questo fa uscire il valore che deve avere SECONDO per la condizione vittoria P1, ad esempio se P1=CARTA(10) allora uscira SASSO(01)

2. il valore di uscita del multiplexer viene confrontato con SECONDO, se sono uguali da 1 e quindi vittoria manche P1, in caso contrario vince la manche P2.

3. il pareggio viene considerato a parte con un **comparatore** che confronta PRIMO e SECONDO e in caso siano uguali MANCHE = 11 (pareggio) ignorando il calcolo precedente, dando dunque priorità al pareggio.

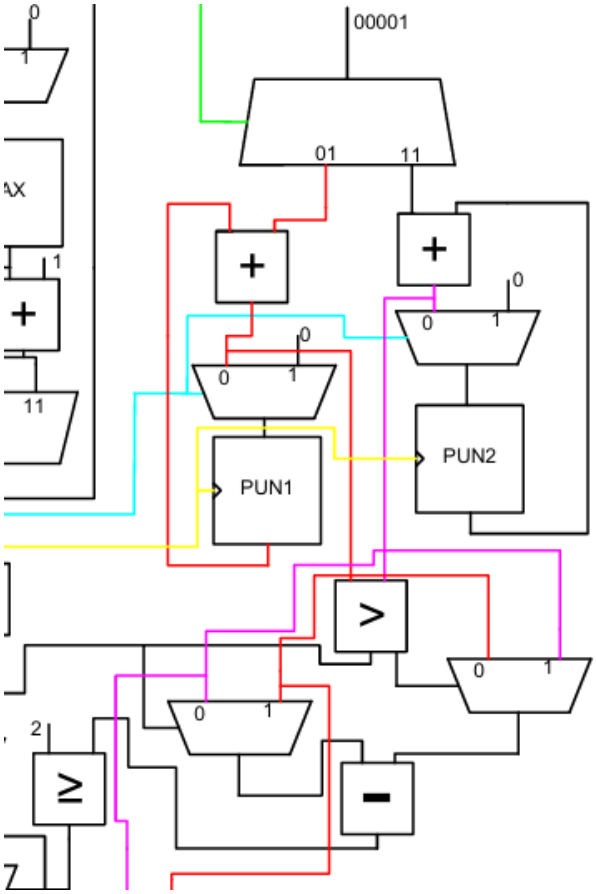
La seconda invece utilizza due registri per salvare le mosse vietate ai giocatori, se ad esempio vince la manche P1, quindi MANCHE = 01 il multiplexer sopra **IND1** fa uscire il valore di PRIMO che viene letto da registro per il prossimo ciclo, invece quello di **IND2** azzera il registro.

In caso di pareggio vengono azzerati entrambi, mentre in caso di manche non valida vengono mantenuti i limiti precedenti, se un giocatore gioca una mossa vietata, viene inviato unsegnale che agisce sul valore di PRIMO passato alla parte di calcolo generando quindi MANCHE non valida.

Entrambi i registri utilizzano un MUX che in caso di PLAY=1 li azzera.

Se uno dei due giocatori gioca 00 la manche non viene considerata valida.

2.3 STORING PUNTEGGI



Abbiamo deciso di salvare in 2 registri a 5 bit i punteggi dei giocatori.

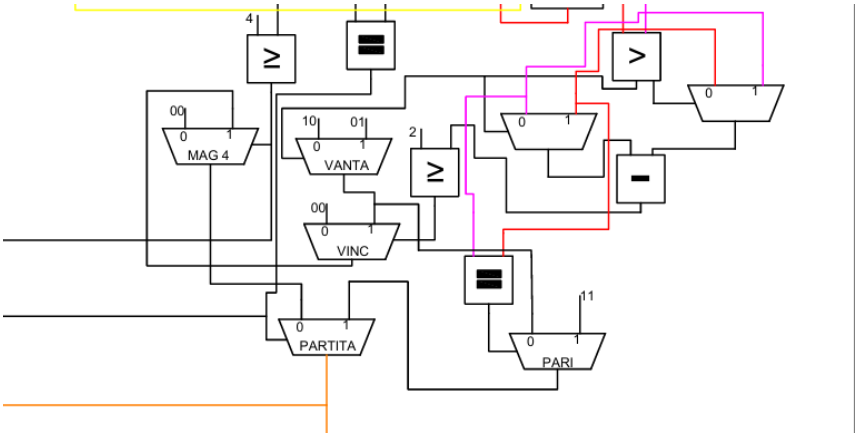
Questi vengono incrementati in base al segnale MANCHE, agisce su un DEMULTIPLEXER con un ingresso a 5 bit che contiene la costante 1, e due uscite.

Questo 1 viene sommato al valore precedente del registro del giocatore vincente, mentre all'altra uscita viene inviato 0.

La manche in pareggio viene considerata valida, ma i punteggi vengono mantenuti invariati.

Viene poi ad ogni ciclo controllata la differenza tra i due punteggi (eventualmente incrementati), il componente " $>$ " controlla $P1 > P2$, se vero si effettua la sottrazione P1-P2 e lo si confronta con 2 per la condizione di vantaggio minimo uguale 2, altrimenti se falso gli operandi della sottrazione vengono invertiti e diventa $P2 - P1$.

2.4 CONTROLLO CONDIZIONI FINE PARTITA



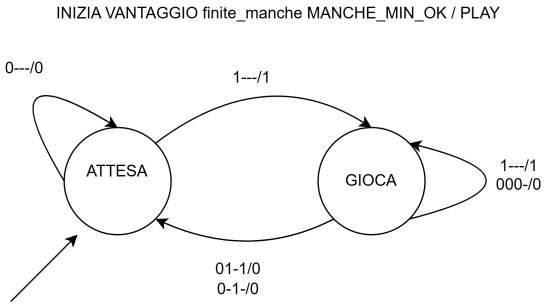
Questa parte finale è dedicata al calcolo delle condizioni di fine partita per generare i segnali MIN_MANCHE_OK e $finite_manche$ che vanno alla FSM e il segnale output del sistema PARTITA.

Un $\geq$ controlla se si sono giocate le manche minime e generea $finite_manche$ che va alla FSM e al MUX **MAG4**.

Il calcolo della sezione precedente trova il giocatore in vantaggio e lo seleziona in **VANTA**, questo però viene bloccato dal MUX **VINC.** finche non si ha anche il vantaggio minimo di 2, la presenza di questi 2 segnali veri da il vincitore su PARTITA e conclude la partita.

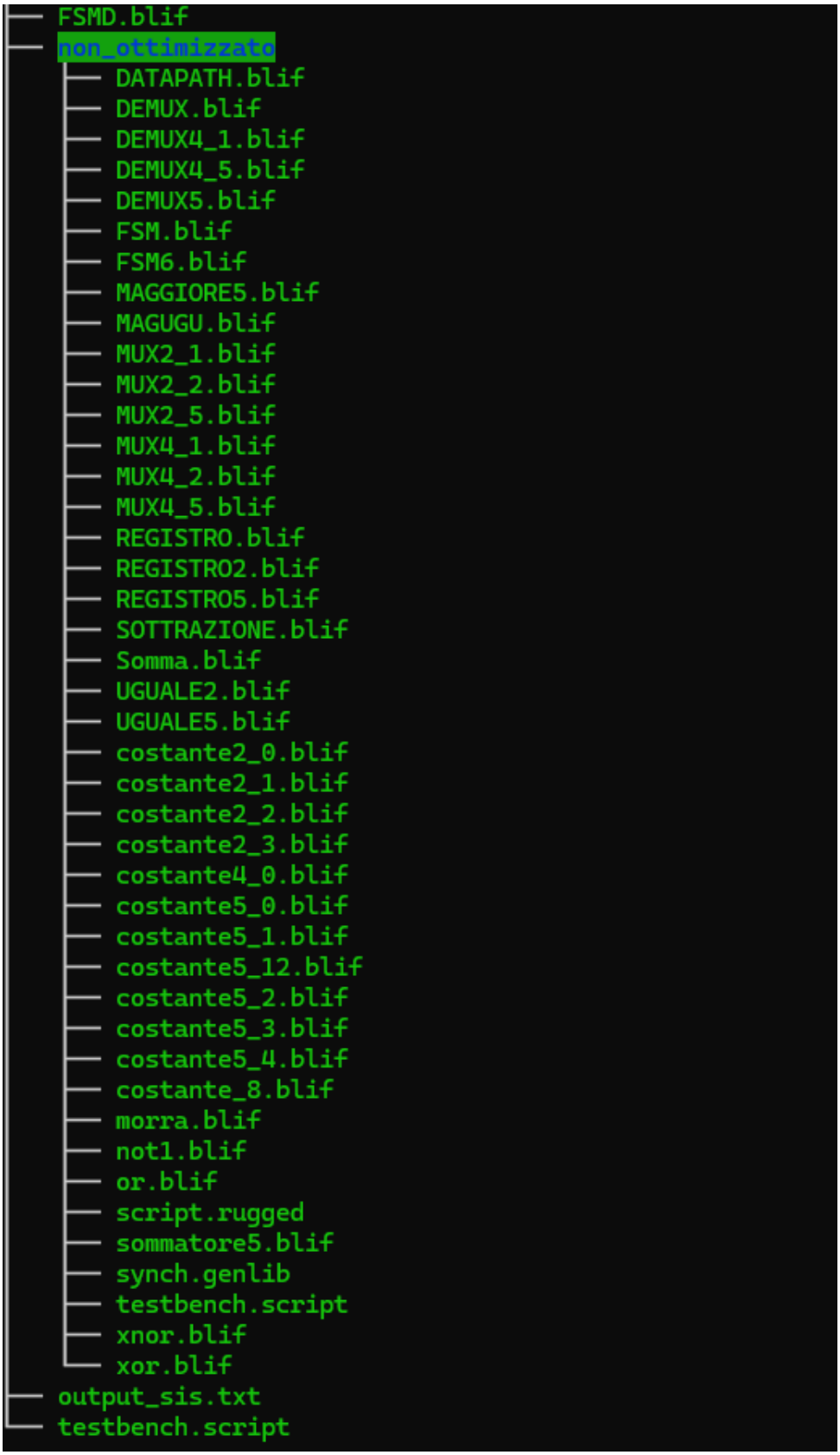
Se si arriva alle manche massime, controllate da un comparatore che confronta i 2 registri della manche, abbiamo deciso di considerare vincitore anche il giocatore che finite la manche è in vantaggio ma non di 2 punti rispetto all'avversario, o in caso di punteggi uguali, viene dato pareggio.

2.5 FSM



3 SIS

Struttura cartella



```
sis> print_stats
FSMD            pi= 5   po= 4   nodes= 99       latches=24
lits(sop)= 585
sis>
```

Figure 1: pre state assign jedi

```
sis> print_stats
FSMD            pi= 5   po= 4   nodes= 95       latches=24
lits(sop)= 555
```

Figure 2: post state assign jedi

```
sis> map -m 0 -s
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:          28
total gate area:       8556.00
maximum arrival time: (76.40,76.40)
maximum po slack:     (-5.60,-5.60)
minimum po slack:     (-76.40,-76.40)
total neg slack:      (-895.20,-895.20)
# of failing outputs:  28
>>> before removing parallel inverters <<<
# of outputs:          28
total gate area:       8428.00
maximum arrival time: (75.00,75.00)
maximum po slack:     (-5.60,-5.60)
minimum po slack:     (-75.00,-75.00)
total neg slack:      (-853.40,-853.40)
# of failing outputs:  28
# of outputs:          28
total gate area:       7756.00
maximum arrival time: (72.20,72.20)
maximum po slack:     (-5.60,-5.60)
minimum po slack:     (-72.20,-72.20)
total neg slack:      (-819.80,-819.80)
# of failing outputs:  28
sis>
```

Figure 3: post mapping

4 VERILOG

```
// Code your design here
module MorraCinese (
    input clk,
    input[1:0] PRIMO,
    input[1:0] SECONDO,
    input INIZIA,
        output reg [1:0] MANCHE,
    output reg [1:0] PARTITA
);
parameter NESSUNA = 2'b00;
parameter SASSO = 2'b01;
parameter CARTA = 2'b10;
parameter FORBICE = 2'b11;
parameter W1 = 2'b01;
parameter W2 = 2'b10;
parameter DRAW = 2'b11;
parameter NA = 2'b00;

reg stato = 1'b0;
reg stato_prossimo = 1'b0;
reg play = 1'b0;
reg [4:0] MAX = 5'b0;
reg [4:0] PT1 = 5'b0;
reg [4:0] PT2 = 5'b0;
reg [1:0] IND1 = 2'b0;
reg [1:0] IND2 = 2'b0;
reg [4:0] ROUND = 5'b0;



// FSM
    always @(clk) begin : UPDATE
    stato = stato_prossimo;
    end

    always @(stato, PARTITA, INIZIA ) begin : FSM
    case(stato)
    1'b0 :
    if (INIZIA)begin
        play = 1;
        stato_prossimo = 1'b0;
    end else begin
        stato_prossimo = 1'b1;
        play = 0;
    end
    1'b1:
        if( ~INIZIA && PARTITA == 2'b00 )begin
            stato_prossimo  = 1'b1;
            play = 0;
        end else begin
            stato_prossimo = 1'b0;
            play = 1;
        end
    endcase
    end

```

```
57  always @(posedge clk) begin : DATAPATH
58
59    //RESET IN CASO play SIA 1 E DETERMINO ANCHE IL NUMERO DI
          MANCHE
60    if(play)begin
61        MAX = {PRIMO, SECONDO} + 4;
62        PT1 = 0;
63        PT2 = 0;
64        IND1 = 0;
65        IND2 = 0;
66        ROUND = 0;
67        PARTITA = 0;
68    end
69    //PARTE COMBINATORIA PER DETERMINARE IL RISULTATO DELLA MANCHE
70    if ( PRIMO == IND1 || SECONDO == IND2 || PRIMO == NESSUNA ||
          SECONDO == NESSUNA )begin
71        MANCHE = NA;
72        IND1 = IND1;
73        IND2 = IND2;
74    end
75    else if(PRIMO == SECONDO )begin
76        MANCHE = DRAW;
77        IND1 = NESSUNA;
78        IND2 = NESSUNA;
79    end
80    else if((PRIMO == SASSO && SECONDO == FORBICE )  ||  (PRIMO ==
          CARTA && SECONDO == SASSO )  || (PRIMO == FORBICE && SECONDO
           == CARTA ))
81    begin
82        MANCHE = W1;
83        IND1  = PRIMO;
84        IND2 = NESSUNA;
85    end
86    else if((SECONDO == SASSO && PRIMO == FORBICE )  ||  (SECONDO
          == CARTA && PRIMO == SASSO )  || (SECONDO == FORBICE &&
          PRIMO == CARTA ))
87      begin
88        MANCHE = W2;
89        IND1 = NESSUNA;
90        IND2 = SECONDO;
91    end
92
93    // A STO PUNTO HO IL RISULTATO DELLA MANCHE AGGIORNO I REGISTRI
          NECESSARI
94    if(MANCHE == NA) begin
95        ROUND = ROUND;
96        PT1 = PT1;
97        PT2 = PT2;
98    end
99    else if(MANCHE == W1)begin
100       ROUND = ROUND + 5'b00001;
101       PT1 = PT1 + 5'b00001;
102       PT2 = PT2;
103   end
104   else if(MANCHE == W2)begin
105       ROUND = ROUND + 5'b00001;
106       PT1 = PT1;
107       PT2 = PT2 + 5'b00001;
```

```
108     end
109     else if(MANCHE == DRAW) begin
110         ROUND = ROUND +1;
111     end
112
113     //aggiornamento del registro PARTITA
114 if(ROUND > 3 && (PT1-PT2>=5'b00010 || ROUND == MAX) && PT1>PT2)
        begin
115       PARTITA <= W1;
116     end
117     else if(ROUND > 3 && (PT2-PT1>=5'b00010 || ROUND == MAX) && PT2
          >PT1)begin
118       PARTITA <= W2;
119     end
120     else if(ROUND == MAX && PT1==PT2 )begin
121         PARTITA <= DRAW;
122     end
123
124     //FINE DATAPATH
125     end
126 endmodule
```

4.1 testbench.sv

```
1
2  module testebench();
3  integer tbf, outf ;
4
5  reg clk;
6  reg INIZIA;
7  reg[1:0] PRIMO, SECONDO;
8  wire[1:0] MANCHE, PARTITA;
9  MorraCinese MORRA(clk, PRIMO, SECONDO, INIZIA, MANCHE, PARTITA );
10 always #10 clk = ~clk;
11
12 initial begin
13     $dumpfile("dump.vcd");
14     $dumpvars(1);
15     tbf = $fopen("testbench.script", "w");
16     outf = $fopen("output_verilog.txt", "w");
17     $fdisplay(tbf, "read_blif␣FSMD.blif");
18     clk = 1'b0;
19     PRIMO = 2'b10;
20     SECONDO = 2'b10;
21     INIZIA = 1'b1;
22     $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
          SECONDO[1], SECONDO[0], INIZIA);
23     #20
24     $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
          PARTITA[1], PARTITA[0]);
25     #20
26     PRIMO = 2'b01;
27     SECONDO = 2'b11;
28     INIZIA = 1'b0;
29     $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
          SECONDO[1], SECONDO[0], INIZIA);
```

```
30      #20
31      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
32      #20
33      PRIMO = 2'b11;
34      SECONDO = 2'b01;
35      INIZIA = 1'b0;
36      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
37      #20
38      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
39      #20
40      PRIMO = 2'b11;
41      SECONDO = 2'b10;
42      INIZIA = 1'b0;
43      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
44      #20
45      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
46      #20
47      PRIMO = 2'b01;
48      SECONDO = 2'b01;
49      INIZIA = 1'b0;
50      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
51      #20
52      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
53      #20
54      PRIMO = 2'b01;
55      SECONDO = 2'b10;
56      INIZIA = 1'b0;
57      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
58      #20
59      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
60      #20
61          PRIMO = 2'b11;
62      SECONDO = 2'b01;
63      INIZIA = 1'b0;
64      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
65      #20
66      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
67      #20
68          PRIMO = 2'b11;
69      SECONDO = 2'b10;
70      INIZIA = 1'b0;
71      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
72      #20
73      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
74      #20
```

```
75          PRIMO = 2'b10;
76      SECONDO = 2'b01;
77      INIZIA = 1'b0;
78      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
79      #20
80      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
81      #20
82          PRIMO = 2'b11;
83      SECONDO = 2'b10;
84      INIZIA = 1'b0;
85      $fdisplay(tbf, "simulate␣%b␣%b␣%b␣%b␣", PRIMO[1], PRIMO[0],
            SECONDO[1], SECONDO[0], INIZIA);
86      #20
87      $fdisplay(outf, "Outputs:␣%b␣%b␣%b␣%b", MANCHE[1], MANCHE[0],
            PARTITA[1], PARTITA[0]);
88      #20
89
90
91      $fdisplay(tbf, "quit");
92      $fclose(tbf);
93      $fclose(outf);
94      $finish;
95  end
96
97
98  endmodule
```