UNIVERSITÀ
di VERONA

Elaborato SIS e Verilog

Corso di Architettura degli Elaboratori

Prof. Fummi Franco, Prof. Lora Michele

A.A. 2023/2024

Ariton Maria Gemma – VR500028
Darie Alexandru – VR500116

Indice

1. Specifiche

Si richiede di progettare un dispositivo per la gestione delle partite del gioco della **Morra Cinese**. Tale dispositivo andrà sviluppato come un circuito sequenziale FSMD sia in SIS che in Verilog.

Due giocatori inseriscono una mossa tra sasso, carta oppure forbice; il vincitore è decretato dalle seguenti regole:

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

Se entrambi i giocatori scelgono la stessa mossa, la manche finirà in pareggio.

Ogni partita si compone 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 giocabili viene settato al ciclo di clock in cui viene iniziata la partita;
- A vincere sarà il giocatore che avrà **due** manche vinte in più del proprio avversario, a patto di aver giocato **almeno** quattro manche;
- Ad ogni manche, il giocatore **vincente** della manche **precedente** non può ripetere l'ultima mossa utilizzata. Nel caso lo facesse, la manche viene considerata non valida (e non conteggiata) ed andrebbe ripetuta;
- Ad ogni manche, in caso di **pareggio** la manche viene **conteggiata**. Alla manche successiva, entrambi i giocatori possono usare tutte le mosse (vedere 7 per ulteriori scelte a riguardo).

Il circuito ha **tre** ingressi:

- **PRIMO** [2 bit]: mossa scelta dal primo giocatore.
  Le mosse hanno i seguenti codici:
  - **00**: nessuna mossa;
  - **01**: Sasso;
  - **10**: Carta;
  - **11**: Forbice;
- **SECONDO** [2 bit]: mossa scelta dal secondo giocatore. Le mosse hanno gli stessi codici del primo giocatore.
- **INIZIA** [1 bit]: quando vale 1, riporta il sistema alla configurazione iniziale. Inoltre, la concatenazione degli ingressi **PRIMO** e **SECONDO** viene usata per specificare il numero massimo di manche oltre le quattro obbligatorie. Quando vale 0, la manche prosegue normalmente.

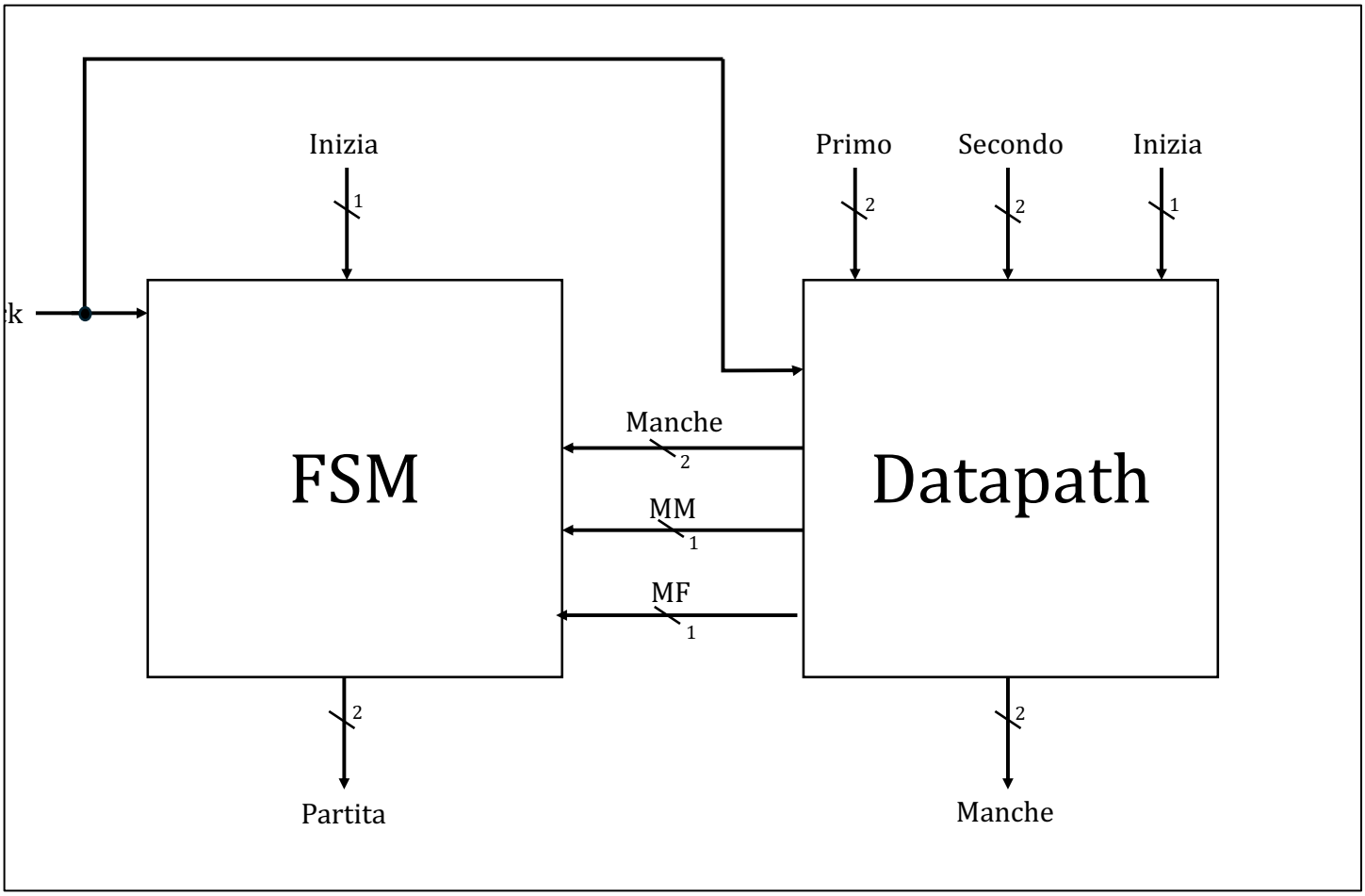
Il circuito ha **due** uscite:

- **MANCHE** [2 bit]: fornisce 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 il risultato della partita con la seguente codifica:
  - **00**: la partita non è terminata;
  - **01**: la partita è terminata, ed ha vinto il giocatore 1;
  - **10**: la partita è terminata, ed ha vinto il giocatore 2;
  - **11**: la partita è terminata in pareggio.

2. Architettura generale

Il nostro circuito è costituito da una FSM (controllore) e da un Datapath (elaboratore). Nella figura sottostante è riportata l'FSMD.



La FSMD è articolata da 3 segnali in input condivisi tra FSM e Datapath:
- **Inizia** [1 bit]: è usato come segnale di reset della partita, quando è a 1 vengono calcolate anche le manche massime da giocare;
- **Primo** [2 bit]: sono i 2 bit che denotano quale mossa il giocatore ha scelto;
- **Secondo** [2 bit]**:** uguale a Primo;

In output vengono restituiti 2 segnali:
- **Partita** [2 bit]: rappresenta l'esito della partita ed è restituito dalla FSM;
- **Manche** [2 bit]: rappresenta l'esito della manche ed è restituito dal Datapath;

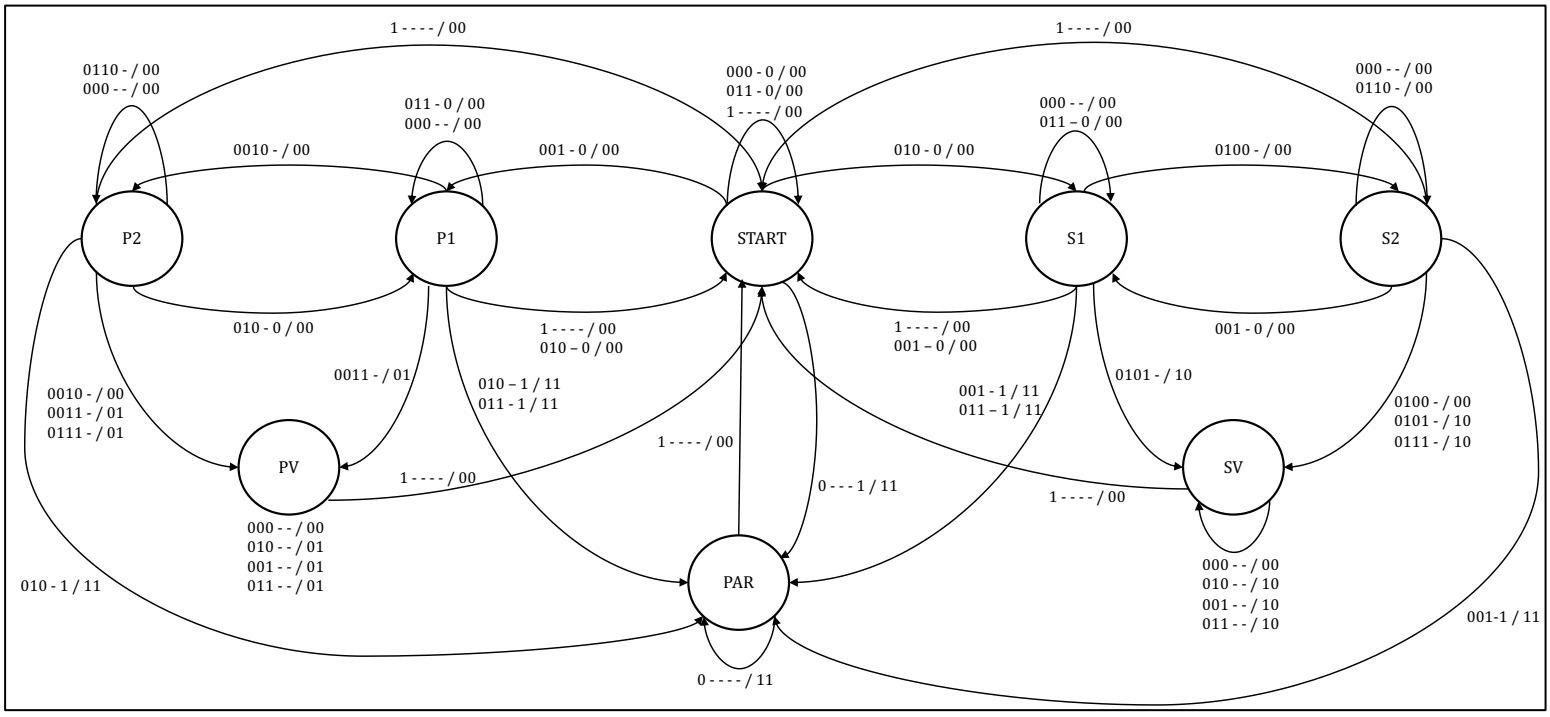
3. Controllore

Come detto nel capitolo precedente, il segnale di input della FSM è **Inizia**, che ha funzione di segnale di reset.

La FSM è composta da 8 stati:

- **Start:** stato di partenza, a cui si ritorna ogniqualvolta lo stato **Inizia** diventi 1 da qualunque altro stato;
- **S1**: stato che rappresenta il vantaggio di uno del secondo giocatore rispetto al primo;
- **S2**: stato che rappresenta il vantaggio di due del secondo giocatore rispetto al primo;
- **SV**: stato di vittoria finale del secondo giocatore;
- **P1**: stato che rappresenta il vantaggio di uno del primo giocatore rispetto al secondo;
- **P2**: stato che rappresenta il vantaggio di due del primo giocatore rispetto al secondo;
- **PV**: stato di vittoria finale del primo giocatore;
- **PAR**: stato che rappresenta la conclusione in pareggio della partita;

Sottostante è riportato lo **State Transition Graph** del controllore



3.1 Implementazione in SIS

```
.model CONTROLLO
.inputs inz m1 m2 MM MF
.outputs p1 p2
.start_kiss
.i 5
.o 2
.p 42
.s 8
.r START

1---- START START 00
010-0 START S1 00
001-0 START P1 00
000-0 START START 00
011-0 START START 00
0---1 START PAR 11
0100- S1 S2 00
1---- S1 START 00
001-0 S1 START 00
001-1 S1 PAR 11
0101- S1 SV 10
011-0 S1 S1 00
011-1 S1 PAR 11
000-- S1 S1 00
1---- P1 START 00
010-0 P1 START 00
010-1 P1 PAR 11
0010- P1 P2 00
0011- P1 PV 01
011-0 P1 P1 00
011-1 P1 PAR 11
000-- P1 P1 00
0---- PAR PAR 11
1---- PAR START 00
001-0 S2 S1 00
1---- S2 START 00
0100- S2 SV 00
0101- S2 SV 10
001-1 S2 PAR 11
000-- S2 S2 00
0110- S2 S2 00
0111- S2 SV 10
1---- SV START 00
000-- SV SV 00
010-- SV SV 10
001-- SV SV 10
011-- SV SV 10
010-0 P2 P1 00
1---- P2 START 00
0010- P2 PV 00
0011- P2 PV 01
010-1 P2 PAR 11
000-- P2 P2 00
0110- P2 P2 00
0111- P2 PV 01
000-- PV PV 00
010-- PV PV 01
001-- PV PV 01
011-- PV PV 01
1---- PV START 00
.end_kiss
```

3.2 Implementazione in Verilog

La FSM in Verilog è stata implementata seguendo lo STG già riportato in precedenza, viene di seguito riportata solo la parte iniziale e finale del codice, in quanto il resto del codice ha la stessa struttura.

```
always @ (stato, manche, in) begin : FSM
  if (in == 1'b1) begin
    nextStato = 3'b000;
    partita = 2'b00;
  end
else begin
  case (stato)
  //START
    3'b000: begin
      if (manche == 2'b01 && mf == 1'b0) begin
        nextStato = 3'b001;
        partita = 2'b00;
      end
      else if (manche == 2'b10 && mf == 1'b0) begin
        nextStato = 3'b100;
        partita = 2'b00;
      end
      else if (mf == 1'b1) begin
        nextStato = 3'b111;
        partita = 2'b11;
      end
      else begin
        nextStato = 3'b000;
        partita = 2'b00;
      end
      end
```

.....

```
  //SV
   3'b101: begin
      if(manche == 2'b00) begin
          nextStato = 3'b101;
          partita = 2'b00;
      end
      else begin
          nextStato = 3'b101;
          partita = 2'b10;
      end
      end


  //PAR
   3'b111: begin
      nextStato = 3'b111;
      partita = 2'b11;
      end

  endcase
 end
end
endmodule
```

4. Datapath

4.1 Componenti

Segue l'elenco di tutti i componenti che sono stati utilizzati nello sviluppo del circuito:

- ***Registro a 2 bit (2):***
  - *PP*: memorizza la mossa proibita del primo giocatore, in caso di sua vittoria;
  - *PS*: memorizza la mossa proibita del secondo giocatore, in caso di sua vittoria;
- ***Registro a 5 bit (2):***
  - *CMP*: utilizzato come contatore per memorizzare le manche giocate fino a quel punto;
  - *TOTALE*: memorizza il numero di manche totali da giocare, deciso qualvolta il segnale INIZIA sia a 1;
- ***Demultiplexer a 2 bit (2):***
  Utilizzati per modificare il comportamento del circuito in base al segnale di INIZIA.
  - *Se INIZIA è 1, le mosse di Primo e Secondo verranno utilizzate per settare le manche totali;*
  - *Se INIZIA è 0, le mosse di Primo e Secondo verranno utilizzate nel gioco principale;*
- ***Multiplexer a 4 ingressi da 2 bit (2):***
  Utilizzati per decretare la mossa proibita in base al segnale di output di CV.
  - *Multiplexer riferente a Primo:*
    - 00: viene passata la mossa precedente salvata in PP, in quanto 00 è mossa non valida;
    - 01: vince il Primo giocatore, quindi viene passato nel registro PP il segnale di Primo;
    - 10: vince il Secondo giocatore, quindi Primo ha come unica mossa proibita 00 (mossa non valida);
    - 11: la manche finisce in pareggio quindi entrambi i giocatori non hanno mosse proibite;
  - *Multiplexer riferente a Secondo:*
    Ragionamento simile tranne per:
    - 01: vince il Primo giocatore, quindi Secondo ha come unica mossa proibita 00 (mossa non valida);
    - 10: vince il Secondo giocatore, quindi viene passato nel registro PS il segnale di Primo;
- ***Multiplexer a 2 ingressi da 5 bit (3):***
  Ciascuno svolge solo una funzione tra le sottostanti:
  - Salvare nel registro TOTALE le mosse totali quando il segnale INIZIA è 1, oppure mantenere le stesse mosse totali quando INIZIA è 0;
  - Se il segnale di output da CV è 00, allora la manche non viene conteggiata e viene di conseguenza salvato nel registro CMP il contatore delle mosse non aumentato di 1,

nel caso in cui CV sia diverso da 00 viene aumentato il contatore di mosse e salvato nel reg. CMP;
- Se INIZIA è a 0, vengono eseguiti i passaggi sopra elencati, se è 1, viene azzerato il registro CMP (in quanto inizia una nuova partita);

- ***Multiplexer a 2 ingressi da 1 bit (1):***
  In base al segnale INIZIA si ha come segnale di output MF (che denota il raggiungimento delle manche finali) oppure 00 (nel caso in cui INIZIA sia a 1);

- ***Multiplexer a 2 ingressi da 2 bit (1):***
  In base al segnale INIZIA viene passato in output il segnale uscente da CV (nel caso in cui INIZIA sia a 0) oppure 00 (nel caso INIZIA sia a 1), questo perché se INIZIA è a 1, si ha come output 11 (usato poi per azzerare i registri PP e SP), quindi questo multiplexer corregge 11 in 00 come output manche.

- ***Multiplexer a 4 ingressi da 5 bit (2):***
  In base al segnale di output dei due demultiplexer, viene scelta la codifica della mossa inserita, per poi essere passata come segnale di input al sommatore;

- ***Sommatore a 2 ingressi a 5 bit (3):***
  Prende in input i segnali di output dei multiplexer soprastanti e li somma, per passare il risultato al secondo sommatore come segnale di input, sommato poi alla codifica della cifra 4, 00100;
  L'ultimo sommatore viene usato per incrementare di 1 il registro CMP;

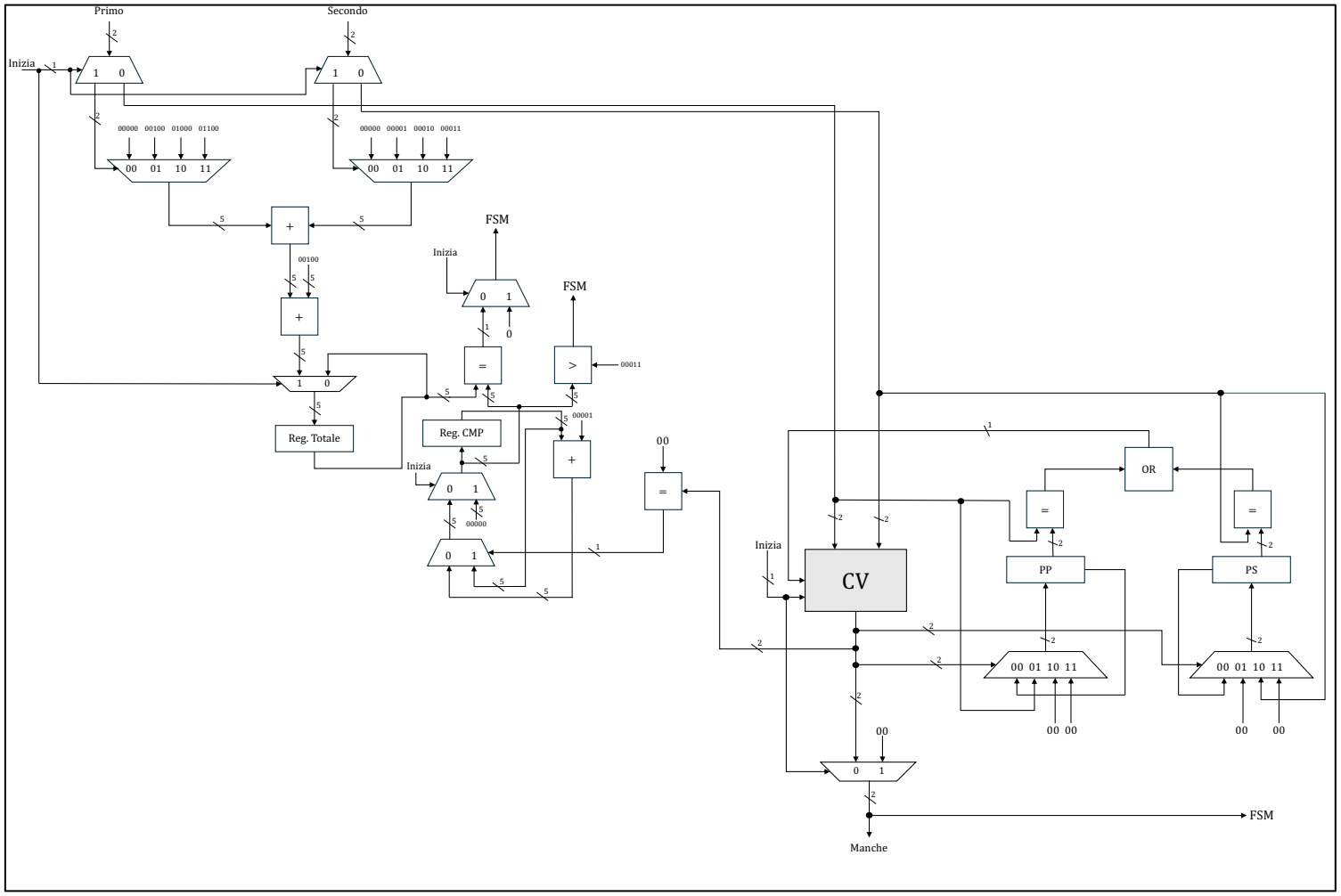
- ***Comparatore di uguaglianza a 2 bit (3):***
  Due vengono usati per controllare se il valore nel reg. PP e PS sono uguali alle mosse di Primo e Secondo, rispettivamente.
  Uno controlla se il segnale di output di CV è uguale a 00 (manche non valida), per passare poi l'esito come segnale di controllo al multiplexer;

- ***Comparatore di uguaglianza a 5 bit (1):***
  Controlla se il valore nel reg. CMP è uguale al valore nel reg. TOTALE;

- ***Comparatore di maggioranza a 5 bit (1):***
  Controlla se il valore nel reg. CMP è maggiore della codifica di 3 (00011), quindi se si sono giocate le quattro manche minime;

- ***OR (1):***
  Prende in input i segnali di output dei comparatori di uguaglianza, controlla se uno dei due segnali è a 1 (quindi uno dei due giocatori ha messo la mossa proibita) e lo usa come segnale di input nel componente CV;

Segue il grafico del Datapath:



4.1.2 Componente CV

Il componente CV (Controllo Vincitore) prende in input:

- le mosse di entrambi i giocatori;
- il segnale INVALIDO derivante dal componente OR;

Come segnali di output si ha il vincitore tra i due.

Quindi il componente controlla chi dei due giocatori sarà il vincente nel caso in cui INIZIA e INVALIDO siano a 0 e lo darà in output.

- Nel caso in cui INVALIDO sia a 1, il componente darà come output 00, ovvero manche non valida in quanto vuol dire che uno dei due giocatori che ha vinto al ciclo precedente ha rimesso la stessa mossa usata per vincere.
- Quando INIZIA è a 1, viene dato in output 11, ovvero pareggio, in quanto sarà utilizzato poi per azzerare i registri in preparazione alla nuova partita (consultare cap. 7 per approfondimenti).

4.2 Implementazione in SIS

Segue il codice riguardante l'implementazione del Datapath in SIS:

```
.model DATAPATH
.inputs p1 p2 s1 s2 inz
.outputs mo1 mo2 MM MF

.subckt CV p1=dp1 p2=dp2 s1=ds1 s2=ds2 inv=ugu rst=inz m1=m1 m2=m2
.subckt UGUALE_2B a1=dp1 a2=dp2 b1=rp1 b2=rp2 o=up
.subckt UGUALE_2B a1=ds1 a2=ds2 b1=rs1 b2=rs2 o=us
.subckt OR a=up b=us o=ugu
.subckt CONST0 o=zr
.subckt MUX4da2 a1=rp1 a2=rp2 b1=p1 b2=p2 c1=zr c2=zr d1=zr d2=zr ctr1=m1 ctr2=m2 o1=wrp1 o2=wrp2
.subckt MUX4da2 a1=rs1 a2=rs2 b1=zr b2=zr c1=s1 c2=s2 d1=zr d2=zr ctr1=m1 ctr2=m2 o1=wrs1 o2=wrs2
.subckt REG2 i1=wrp1 i2=wrp2 o1=rp1 o2=rp2
.subckt REG2 i1=wrs1 i2=wrs2 o1=rs1 o2=rs2

.subckt MUX_2B a1=m1 a2=m2 b1=zr b2=zr ctr=inz o1=mo1 o2=mo2

# uguale tra manche del cv e costante 0, da in output se la manche è valida o meno (mv)
.subckt UGUALE_2B a1=m1 a2=m2 b1=zr b2=zr o=mv

# mux, se ctr ,che prende mv, è 1 prende registro cmp non incrementato, se ctr è 0 prende cmp incrementato (incr)
.subckt MUX_2_da5 a1=incr1 a2=incr2 a3=incr3 a4=incr4 a5=incr5 b1=cmp1 b2=cmp2 b3=cmp3 b4=cmp4 b5=cmp5 ctr=mv o1=c1 o2=c2 o3=c3 o4=c4 o5=c5

# mux, se inizia è a 0 prende l'output del mux prima, se a 1 prende la costante di 5 bit di 0
.subckt MUX_2_da5 a1=c1 a2=c2 a3=c3 a4=c4 a5=c5 b1=zr b2=zr b3=zr b4=zr b5=zr ctr=inz o1=d1 o2=d2 o3=d3 o4=d4 o5=d5

# registro cmp (manche completate)
.subckt REG5 i1=d1 i2=d2 i3=d3 i4=d4 i5=d5 o1=cmp1 o2=cmp2 o3=cmp3 o4=cmp4 o5=cmp5

# uguale, prende manche totali (del registro tot, rt) e le confronta a quelle completate, da in output se le manche massime sono finite o meno
.subckt UGUALE_5B a1=rt1 a2=rt2 a3=rt3 a4=rt4 a5=rt5 b1=d1 b2=d2 b3=d3 b4=d4 b5=d5 o=tMF
.subckt MUX_1B a=tMF b=zr ctr=inz o=MF

.subckt CONST1 o=uno

# maggiore o uguale, prende cmp e lo confronta con la costante 00011, dà in output se le manche minime sono finite o meno
.subckt maggiore_5 A1=d1 A2=d2 A3=d3 A4=d4 A5=d5 B1=zr B2=zr B3=zr B4=uno B5=uno O=MM

# registro totale, prene in input il risultato tra la somma di 00100 e il registro psc (quindi psc aumentato)
.subckt REG5 i1=muxt1 i2=muxt2 i3=muxt3 i4=muxt4 i5=muxt5 o1=rt1 o2=rt2 o3=rt3 o4=rt4 o5=rt5

# sommatore, prende 1 0 0 0 0 in input il registro cmp e la costante 00001, li somma e dà output il registro cmp incrementato (incr)
.subckt sommatore5 A1=cmp1 A2=cmp2 A3=cmp3 A4=cmp4 A5=cmp5 B1=zr B2=zr B3=zr B4=zr B5=uno CIN=zr O1=incr1 O2=incr2 O3=incr3 O4=incr4 O5=incr5 COUT=COUT1

.subckt MUX_2_da5 a1=rt1 a2=rt2 a3=rt3 a4=rt4 a5=rt5 b1=psca1 b2=psca2 b3=psca3 b4=psca4 b5=psca5 ctr=inz o1=muxt1 o2=muxt2 o3=muxt3 o4=muxt4 o5=muxt5

# sommatore tra gli input dei giocatori
.subckt sommatore5 A1=si1 A2=si2 A3=si3 A4=si4 A5=si5 B1=zr B2=zr B3=uno B4=zr B5=zr CIN=zr O1=psca1 O2=psca2 O3=psca3 O4=psca4 O5=psca5 COUT=COUT2

# sommatore manche max
.subckt sommatore5 A1=ipri1 A2=ipri2 A3=ipri3 A4=ipri4 A5=ipri5 B1=isec1 B2=isec2 B3=isec3 B4=isec4 B5=isec5 CIN=zr O1=si1 O2=si2 O3=si3 O4=si4 O5=si5

#multiplexer giocatore uno
.subckt MUX4da5 a1=zr a2=zr a3=zr a4=zr a5=zr b1=zr b2=zr b3=uno b4=zr b5=zr c1=zr c2=uno c3=zr
c4=zr c5=zr d1=zr d2=uno d3=uno d4=zr d5=zr ctr1=uscp1 ctr2=uscp2 o1=ipri1 o2=ipri2 o3=ipri3 o4=ipri4 o5=ipri5

#multiplexer giocatore due
.subckt MUX4da5 a1=zr a2=zr a3=zr a4=zr a5=zr b1=zr b2=zr b3=zr b4=zr b5=uno c1=zr c2=zr c3=zr
c4=uno c5=zr d1=zr d2=zr d3=zr d4=uno d5=uno ctr1=uscs1 ctr2=uscs2 o1=isec1 o2=isec2 o3=isec3 o4=isec4 o5=isec5

#demux giocatore uno
.subckt demux a=p1 b=p2 in=inz x1=uscp1 x2=uscp2 y1=dp1 y2=dp2

#demux giocatore due
.subckt demux a=s1 b=s2 in=inz x1=uscs1 x2=uscs2 y1=ds1 y2=ds2
```

4.3 Implementazione in Verilog

Segue il codice riguardante l'implementazione del Datapath in Verilog:

```
always @(posedge clk) begin : DP
  if (in == 1'b1) begin
      pp = 2'b00;
      ps = 2'b00;
      cmp = 5'b00000;
      stato = 3'b000;
      totale = 5'b00100+primo*3'b100+secondo;
      manche = 2'b00;
  end
  else if (pp == primo || ps == secondo || primo == 2'b00 || secondo == 2'b00) begin
      manche = 2'b00;
  end
  else if (primo == secondo) begin
      manche = 2'b11;
  end
  else if (primo == 2'b01) begin
      if (secondo == 2'b10) begin
          manche = 2'b10;
      end
      else begin
          manche = 2'b01;
      end
  end
  else if (primo == 2'b10) begin
      if (secondo == 2'b01) begin
          manche = 2'b01;
      end
      else begin
          manche = 2'b10;
      end
  end
  else if (primo == 2'b11) begin
      if (secondo == 2'b01) begin
          manche = 2'b10;
      end
      else begin
          manche = 2'b01;
      end
  end

  if (manche != 2'b00) begin
      cmp = cmp + 1;
  end

  if (cmp == totale && in == 1'b0) begin
```

```
        mf = 1'b1;
    end else begin
        mf = 1'b0;
    end

    if (cmp > 3'b011) begin
        mm = 1'b1;
    end else begin
        mm = 1'b0;
    end

    if (manche == 2'b01) begin
        pp = primo;
        ps = 2'b00;
    end
    else if (manche == 2'b10) begin
        pp = 2'b00;
        ps = secondo;
    end
    else if (manche == 2'b11) begin
        pp = 2'b00;
        ps = 2'b00;
    end
    $display("manche: %b, mm: %b, mf: %b, next: %b, partita: %b", manche, mm, mf, nextStato, partita);
end
```

5. Statistiche del circuito

5.1 Statistiche pre-ottimizzazione e post

Seguono le statistiche della FSMD pre-ottimizzazione:

```
sis> read_blif morra.blif
Warning: network `DATAPATH', node "COUT1" does not fanout
Warning: network `DATAPATH', node "COUT2" does not fanout
Warning: network `DATAPATH', node "[120]" does not fanout
Warning: network `MORRACINESE', node "COUT1" does not fanout
Warning: network `MORRACINESE', node "COUT2" does not fanout
Warning: network `MORRACINESE', node "[120]" does not fanout
sis> print_stats
MORRACINESE     pi= 5   po= 4   nodes=118       latches=17
lits(sop)= 882
sis>
```

Seguono le statistiche della FSMD post-ottimizzazione:

```
sis> full_simplify
sis> source script.rugged
sis> write_blif ../FSMD.blif
sis> print_stats
MORRACINESE     pi= 5   po= 4   nodes= 39       latches=17
lits(sop)= 317
sis>
```

Come si nota, l'ottimizzazione ha ridotto i nodi da 118 a 39 e i letterali da 882 a 317.

5.1 Mapping

Viene eseguito il mapping sulla libreria ***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 '{[14]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[15]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[16]}' (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:          21
total gate area:       5140.00
maximum arrival time: (56.60,56.60)
maximum po slack:     (-4.00,-4.00)
minimum po slack:     (-56.60,-56.60)
total neg slack:      (-529.60,-529.60)
# of failing outputs:  21
>>> before removing parallel inverters <<<
# of outputs:          21
total gate area:       5124.00
maximum arrival time: (56.80,56.80)
maximum po slack:     (-4.00,-4.00)
minimum po slack:     (-56.80,-56.80)
total neg slack:      (-530.40,-530.40)
# of failing outputs:  21
# of outputs:          21
total gate area:       5028.00
maximum arrival time: (55.80,55.80)
maximum po slack:     (-4.00,-4.00)
minimum po slack:     (-55.80,-55.80)
total neg slack:      (-521.00,-521.00)
# of failing outputs:  21
sis>
```

Come si nota, in seguito al mapping si ha:

- **Total Gate Area**: 5028;
- **Maximum arrival time (ritardo)**: 55.80;

6. Simulazioni

6.1 Testbench

In Verilog, oltre all'implementazione della FSM e del Datapath già riportati in precedenza, è necessario anche un testbench per stimolare il testing del sistema.
Il testbench contiene nella parte iniziale la dichiarazione e collegamento degli input e output, i segnali di controllo per la simulazione e i file dove salvare l'output. Segue poi la sezione dove vengono create da noi un certo numero di partite decidendo le combinazioni delle mosse dei due giocatori e, di conseguenza, anche gli esiti delle manche, così da controllare il corretto funzionamento del sistema.
Di seguito viene riportato il codice tipo della struttura di una breve partita:

```
// Partita 1
inizia = 1'b1;
primo = 2'b00; secondo = 2'b00; // max 4 manche
$fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
#20
$fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);

  // Manche 1 : carta vs forbice
  inizia = 1'b0;
  primo = 2'b10; secondo = 2'b11;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // ha vinto il secondo
  // vantaggio 1

  // Manche 2: forbice vs carta
  primo = 2'b11; secondo = 2'b10;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // ha vinto il primo
  // vantaggio 0

  // Manche 3: forbice vs sasso
  primo = 2'b11; secondo = 2'b01;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // Manche non valida quindi non conteggiata, primo ha messo mossa proibita
  // vantaggio 0

  // Manche 3: forbice vs carta
  primo = 2'b11; secondo = 2'b10;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // Manche non valida quindi non conteggiata, primo ha messo AGAIN la mossa proibita
  // vantaggio 0

  // Manche 3: carta vs sasso
  primo = 2'b10; secondo = 2'b01;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // ha vinto il primo
  // vantaggio di 1

  // Manche 4: sasso vs carta
  primo = 2'b01; secondo = 2'b10;
  $fdisplay(tbf, "simulate %b %b %b %b %b", primo[1], primo[0], secondo[1], secondo[0], inizia);
  #20
  $fdisplay(outf, "Outputs: %b %b %b %b", manche[1], manche[0], partita[1], partita[0]);
  // ha vinto il secondo
  // vantaggio di 0

// Partita 1 terminata in pareggio
```

6.2 Risultati finali

Dalle simulazioni si ottengono 2 file di output, uno per SIS e uno per Verilog:
Vengono sotto riportati

Output SIS:

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

Output Verilog:

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

Come si nota, non c'è alcuna differenza tra i due quindi gli output sono identici.

7. Scelte Progettuali e ulteriori info

- Nell'eventualità che si completino tutte le manche da giocare e nessuno dei due giocatori si trova in vantaggio di due punti rispetto all'altro, la partita viene terminata in pareggio;
- Qualora INIZIA sia a 1, e quindi si utilizza la combinazione di Primo e Secondo per decidere le manche totali da giocare, abbiamo deciso di utilizzare 5 bit nella codifica per ovviare al problema del overflow nel caso in cui le manche totali arrivino a 19 (10011);
- Nel Datapath, passiamo il segnale del contatore delle manche giocate direttamente ai comparatori di uguaglianza e maggioranza, senza farlo passare per il reg. CMP. Questo per ovviare ai problemi di ritardo causati dal dover attendere un altro ciclo affinché il contenuto del registro venga passato ai comparatori;
- Ad ogni manche, il giocatore non vincente avrà come unico vincolo 00 (manche non valida), lo stesso si applica nel caso in cui sia un pareggio;
- Nel caso di pareggio, alla manche successiva i due giocatori non avranno nessun vincolo di mossa oltre a 00;
- Se INIZIA è 1, il segnale di output di CV sarà 11, in quanto nel multiplexer verrà passato nei registri PP e PS, il valore 00 (azzeramento dei registri) e viene impiegato un multiplexer nel per "correggere" l'output delle manche derivante dal componente CV;
- Il circuito in SIS non ottimizzato genera alcuni warning riguardanti alcune uscite che non vengono usate. Possiamo anche ignorarli, in quanto non ostacolano il funzionamento del circuito;
  Tali warning sono generati a causa del COUT nel sommatore;
- In caso in cui la manche risulti 00, alla manche successiva restano i vincoli precedenti;
- In caso di pareggio, ad entrambi i giocatori NON viene assegnato nessun punto;