UNIVR Dipartimento di Informatica

Elaborato SIS – Verilog

Laboratorio Architettura degli Elaboratori

Progettazione Morra Cinese

A.A. 2023/2024

Matricole:
Quaresima Luca VR500114
Pedretti Simone VR500040

INDICE

Specifiche

Si progetti un dispositivo per la gestione di partite della *morra cinese*, conosciuta anche come *sasso-carta-forbici.*

Due giocatori inseriscono una mossa, che può essere *carta, sasso, o forbici*. Ad ogni manche, il giocatore vincente è decretato dalle seguenti regole:

- Sasso batte forbici.
- Forbici batte carta.
- Carta batte sasso.

Nel caso in cui i due giocatori scelgano la stessa mossa, la manche finisce in pareggio.

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

- Si devono giocare un **minimo** di **quattro manche**;
- Si possono giocare un **massimo** al ciclo di clock in cui viene iniziata la partita;
- Vince il primo giocatore a riuscire a **vincere due manche 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 non sarebbe valida ed andrebbe ripetuta (quindi, non conteggiata);
- Ad ogni manche, **in caso di pareggio la manche viene conteggiata**. Alla manche successiva, entrambi i giocatori possono usare tutte le mosse.

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 [2bit]: 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. Ad esempio, se si inserissero i valori PRIMO = 00 e SECONDO = 00, si indicherebbe di giocare esattamente quattro manche. Se si inserisse il valore PRIMO = 00 e SECONDO = 01, si indicherebbe di giocare al più 5 manche (le 4 obbligatorie, più il valore 1 indicato da 0001). Se si inserissero i valori PRIMO = 10 e SECONDO = 01, si indicherebbe di giocare tredici manche (le 4 obbligatorie, più il valore 9 indicato da 1001). 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.

Architettura generale del circuito FSMD

Il circuito è composto da una **FSM** (controllore) e un **DATAPATH** (elaboratore) che comunicano tra loro.

I file che contengono la rappresentazione di queste componenti hanno come nomi *3_FSM_stati_assegnati.blif* e *DATAPATH.blif*.

Il file che permette il collegamento tra la macchina a stati finiti e l'elaboratore ha il nome di *FSMD.blif*.

Lo schema generale è il seguente:

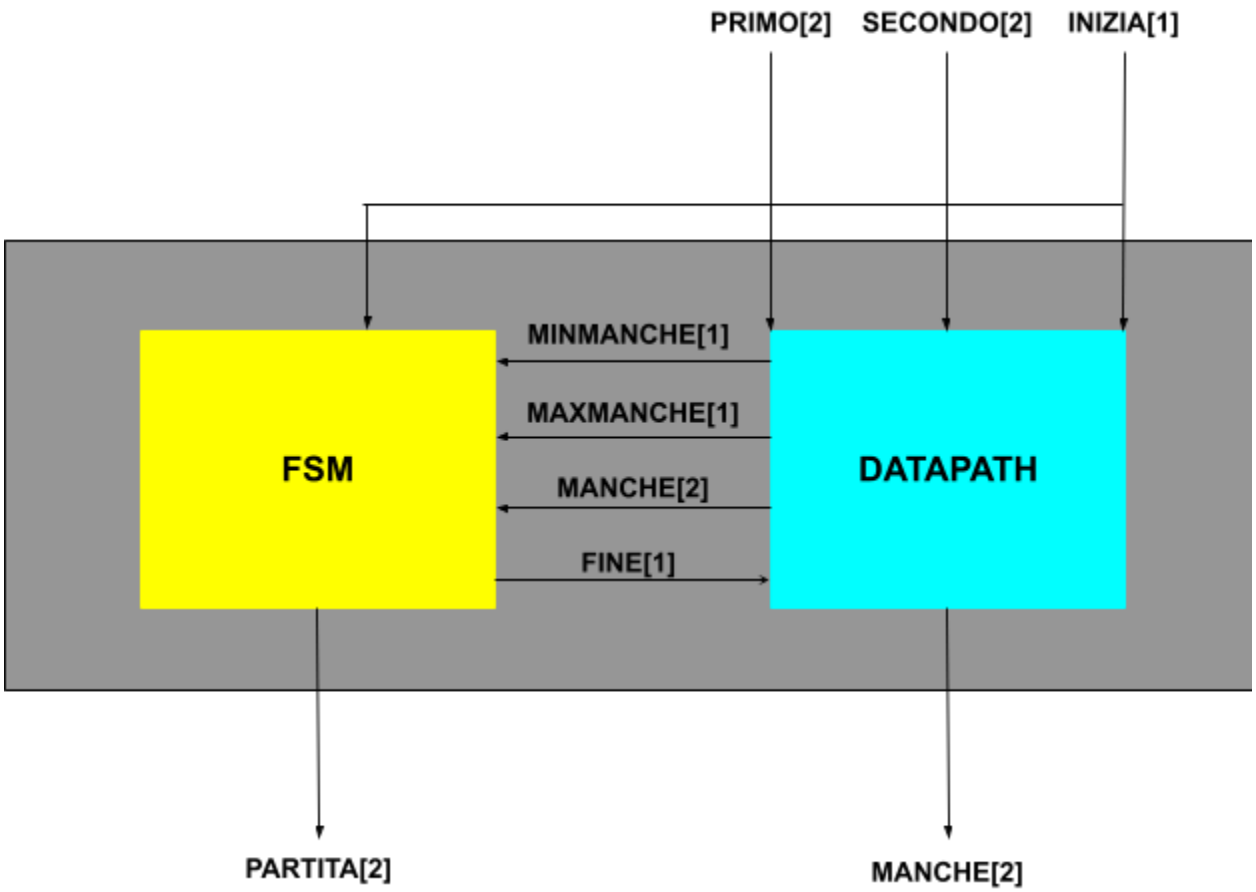
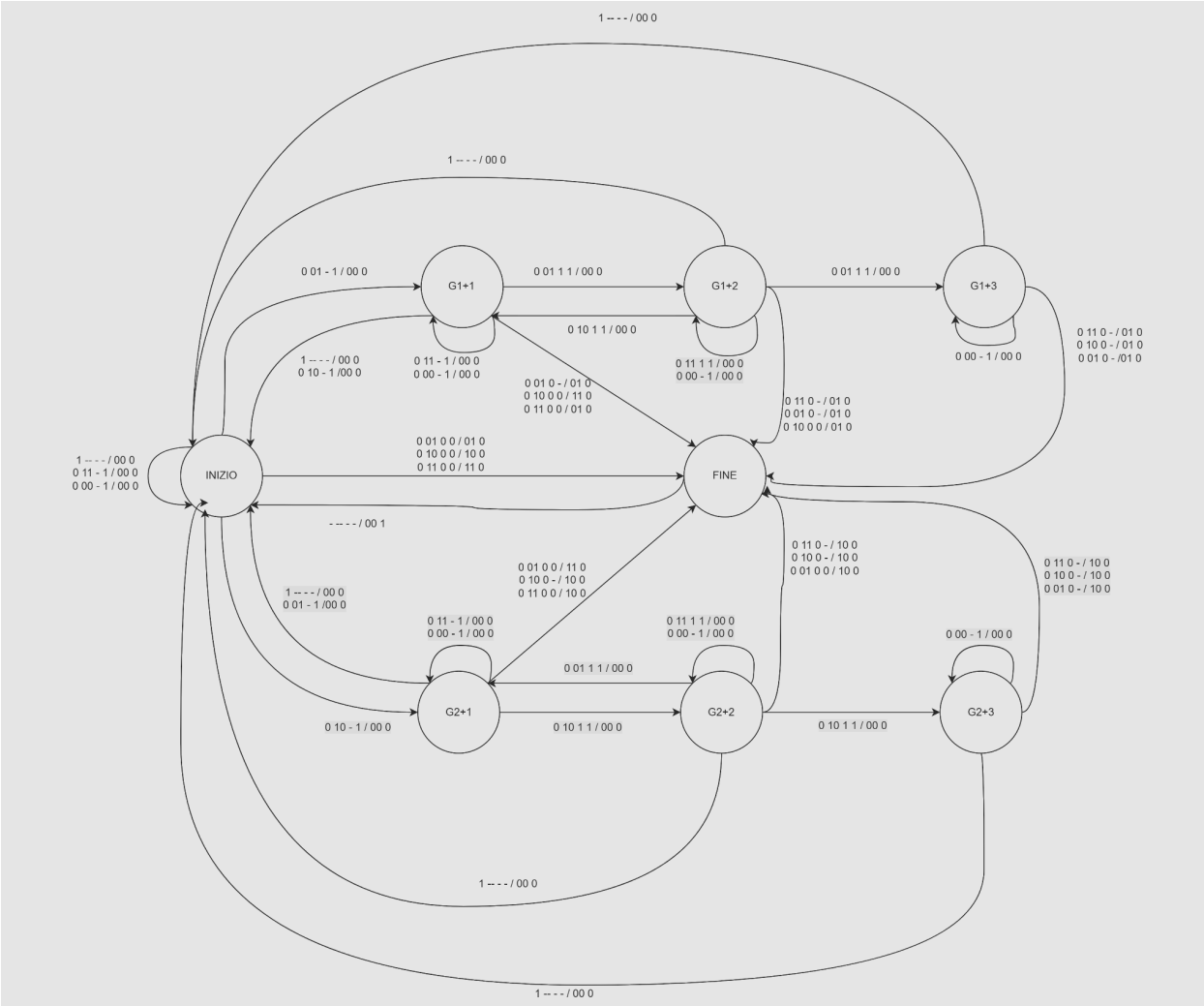


Diagramma del Controllore

Il **controllore** del gioco **Morra Cinese** è una macchina a stati finiti del tipo **Mealy**.
I segnali che si presentano sono i seguenti:

Segnali di input	**Segnali di output**
INIZIA[1] MANCHE[2] MINMANCHE[1] MAXMANCHE[1]	PARTITA[2] FINE[1]



La **FSM** presenta 8 stati:

- **INIZIO**: nello stato iniziale, si avanza, a seconda del giocatore vittorioso, in **G1+1** se vince G1, in **G2+1** se vince G2.
  Si avanza a **FINE** se uno dei due giocatori vince, o si pareggia, e contemporaneamente viene raggiunto il massimo delle manche
  Si rimane nello stato se viene giocata una manche invalida, o se la manche viene pareggiata.
  Si resetta, rimanendo in **INIZIO**, se si inserisce INIZIA=1.

- **G1+1**: si avanza in **G1+2** se vince G1 e contemporaneamente il minimo e il massimo delle manche non sono ancora stati raggiunti.
  Si avanza a **FINE** se vince G1 e contemporaneamente viene raggiunto il minimo o il massimo delle manche, oppure se la manche viene pareggiata e contemporaneamente si raggiunge il massimo delle manche.
  Si rimane in **G1+1** se la manche viene pareggiata e maxmanche=0, o se si gioca una manche invalida.
  Si retrocede a **INIZIO** se vince G2 e contemporaneamente il minimo e il massimo delle manche non sono ancora stati raggiunti.
  Si resetta, tornando a **INIZIO**, inserendo INIZIA=1.

- **G1+2**: si avanza in **G1+3** se vince G1 e contemporaneamente il minimo e il massimo delle manche non sono ancora stati raggiunti.
  Si avanza a **FINE** se vince G1 e contemporaneamente viene raggiunto il minimo o il massimo delle manche, oppure se la manche viene pareggiata e contemporaneamente si raggiunge il massimo delle manche.
  Si rimane in **G1+2** se la manche viene pareggiata, minmanche=1, minmanche=1 opure se viene giocata una manche invalida.
  Si retrocede a **G1+1** se vince G2 e contemporaneamente minmanche=1 e maxmanche=1.
  Si resetta, tornando a **INIZIO**, inserendo INIZIA=1.

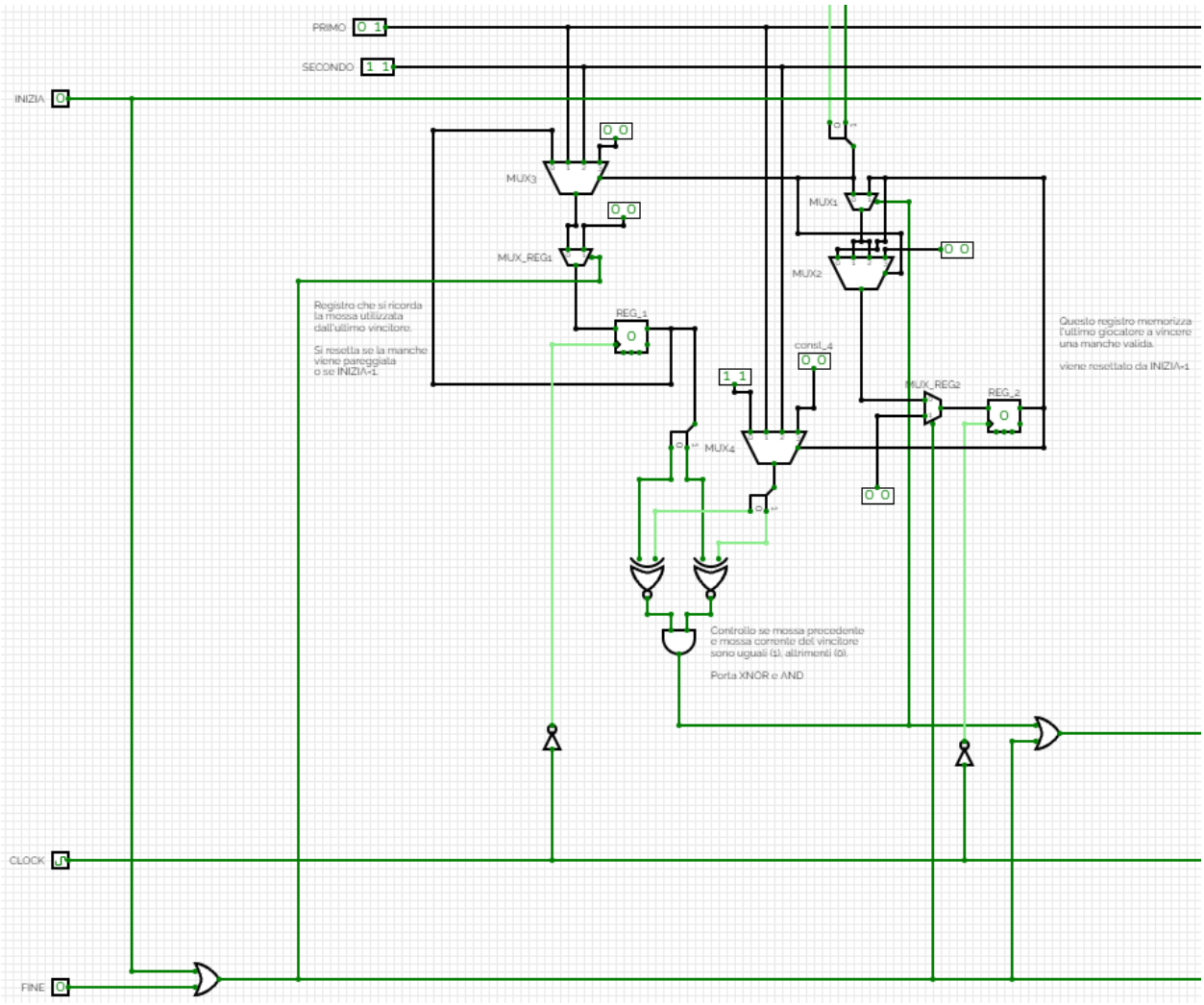
- **G1+3**: si avanza a **FINE** se viene raggiunto il minimo di manche.
  Si rimane in **G1+3** se si gioca una manche invalida.
  SI resetta, tornando a **INIZIO**, inserendo INIZIA=1.

  Il discorso è simmetrico per quanto riguarda gli stati **G2+1, G2+2, G2+3**.

- **FINE:** Indipendentemente dagli input, si avanza allo stato **INIZIO.**

Architettura del Datapath

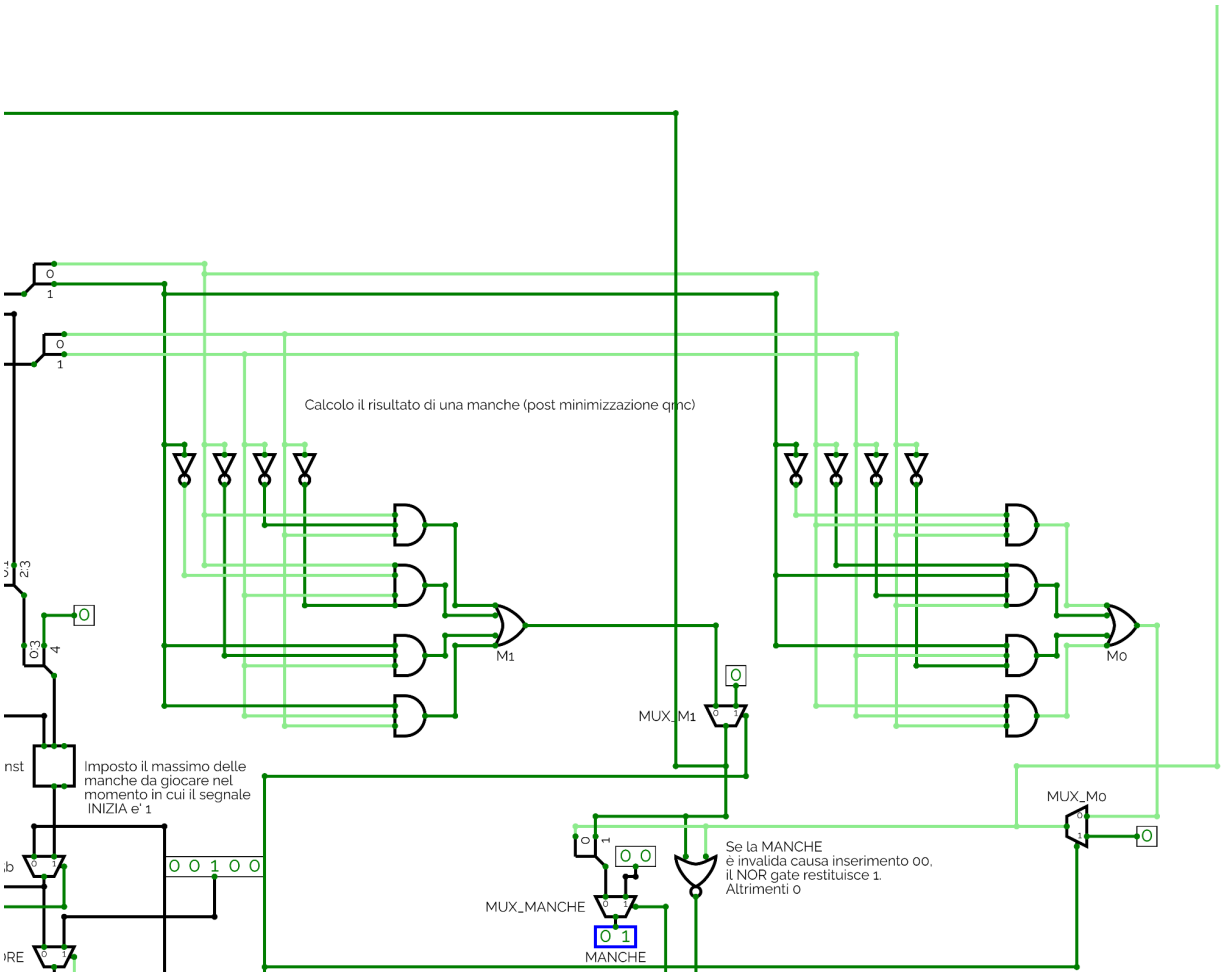
Controllo della mossa del vincitore precedente



- **REG_1**: registro a 2 bit. Memorizza la mossa del vincitore della manche precedente. Collegato al segnale di clock tramite porta NOT: falling edge register.
- **MUX3**: 4 ingressi a 2 bit e come selettore usa il risultato della manche. In caso di manche nulla il valore di REG_1 rimane lo stesso; con manche pareggiata il registro viene azzerato; con manche vinta da uno dei due giocatori, il valore del registro diventa la mossa che il vincitore ha utilizzato.
- **MUX_REG1**: 2 ingressi a 2 bit; azzera il REG_1 quando la porta OR (in basso a sinistra) ha come output 1. Altrimenti passa al registro l'output del MUX3.
- **REG_2**: registro a 2 bit. Memorizza il vincitore dell'ultima manche valida (solo quando la manche viene vinta, quindi può assumere i valori 01 o 10). Collegato al segnale di clock tramite porta NOT: falling edge register.

- **MUX1**: 2 ingressi a 2 bit; ha come selettore la validità della manche: 1 manche invalida e passa il valore di REG_2, 0 manche valida e passa il valore di manche.
- **MUX2**: 4 ingressi a 2 bit; ha come selettore manche: 00 passa il valore di REG_2, 01 e 10 passa l'output del MUX1, 11 passa la costante 00.
- **MUX_REG2**: uguale a MUX_REG1.
- **MUX4**: 4 ingressi a 2 bit; ha come selettore REG_2: 00 passa 11 così è sicuro che non dia manche invalida, 01 passa PRIMO, 10 passa SECONDO, 11 passa 00.
- **XNOR e AND**: confrontano il valore di REG_1 e l'output di MUX4; danno 1 se la manche è invalida, 0 altrimenti.
- **OR**: ha come input l'output della porta AND e l'output della porta OR (in basso a sinistra).

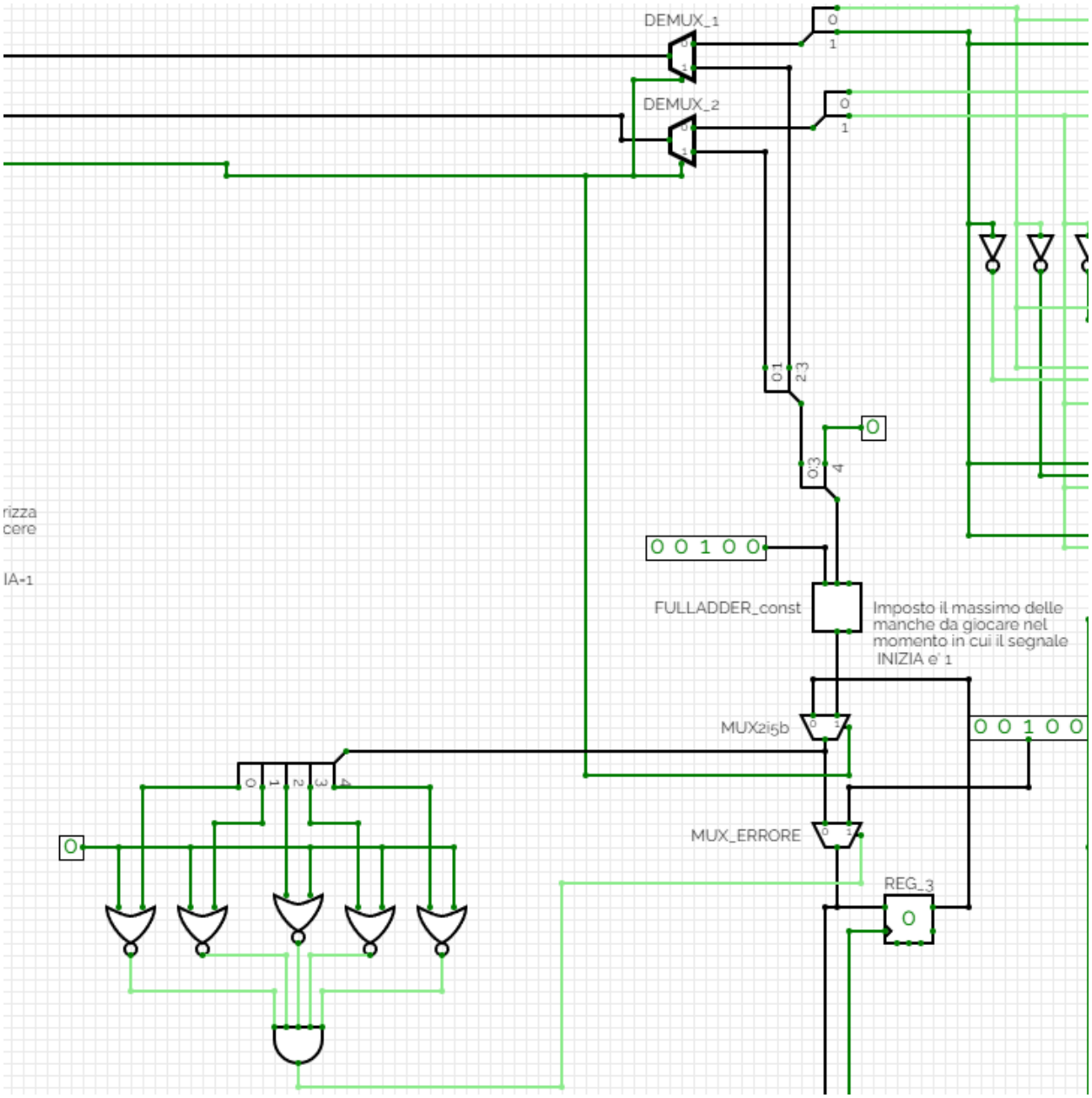
Calcolo vincitore Manche



Il calcolo del vincitore della Manche è stato trattato come una minimizzazione di Quine-McCluskey (vedere file *risultato_manche_non_minimizzato.blif*).
I bit vengono poi filtrati da **MUX_M1** e **MUX_M0**, che hanno come segnale di **select** il controllo che la manche sia valida (vedere **Controllo mossa precedente**).

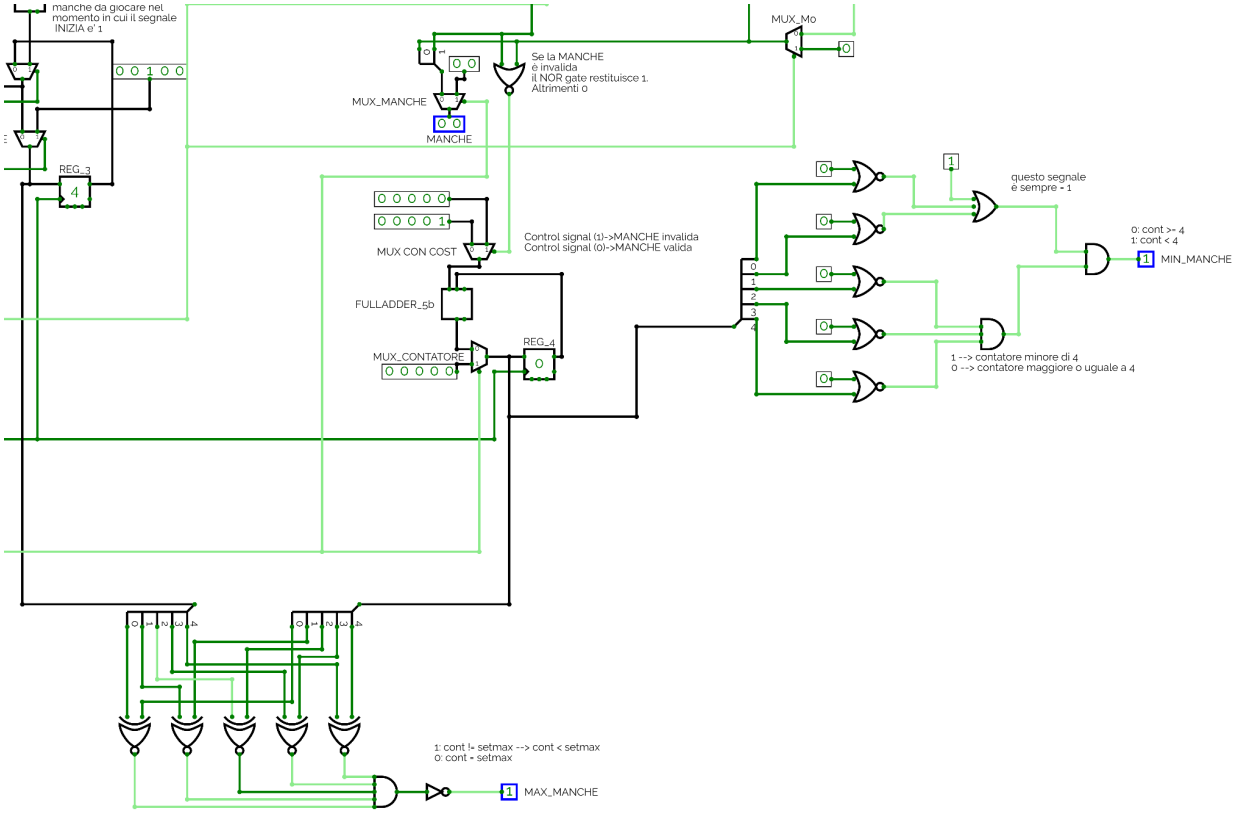
A questo punto, i bit uscenti dai due Mux vengono uniti e filtrati dal **MUX_MANCHE**, che riceve come segnale di **select** il segnale **fine** uscente dalla fsm.
I bit del risultato della Manche vanno anche al **Controllo mossa precedente.**
Se il segnale di fine, o il segnale INIZIA è 1, i registri del **Controllo mossa precedente** vengono azzerati, e quindi il risultato della manche non verrà memorizzato.
Se la partita è finita, o viene resettata, il risultato della manche sarà 00.
Se la manche è invalida, per qualsiasi motivo, la porta **NOR** restituirà 1, che impedirà al registro che conta il numero di manche di incrementarsi.

Calcolo del valore massimo di manche giocabili + controllo errore



- **DEMUX_1 e DEMUX_2**: 2 uscite a 2 bit; hanno come selettore INIZIA e come input rispettivamente PRIMO e SECONDO. Se INIZIA = 1, viene “presa” la strada per il calcolo del numero massimo di manche giocabili. Se INIZIA = 0, viene presa la strada per il calcolo del vincitore della manche (vedere **Calcolo vincitore manche**).
- **FULLADDER_const**: full adder a 5 bit. Aumenta di 4 (numero minimo di manche giocabili) la concatenazione tra PRIMO e SECONDO.
- **MUX2i5b**: 2 ingressi a 5 bit. Ha come selettore INIZIA: 1 passa il valore di FULLADDER_const, 0 passa il valore di REG_3.
- **NOR e AND** (a sinistra): questo è il controllo per un eventuale errore da parte dell’utente. Se l’utente inizia la prima partita con INIZIA = 0, quindi REG_3 = 0, l’output di questo controllo sarà 1. 0 altrimenti.
- **MUX_ERRORE**: 2 ingressi a 5 bit. Ha come selettore l’output del controllo dell’errore. 1 passa 00100, quindi automaticamente il numero minimo di manche giocabili. 0 passa l’output di MUX2i5b.
- **REG_3**: registro a 5 bit. Memorizza il numero di manche giocabili.

Comparatori REG_3 e Minmanche e conteggio Manche valide.



Se la manche è invalida la porta **NOR** restituisce 1.
-**MUX_CON_COST:** riceve come segnale di **select** l'output della porta **NOR** sopracitata. Restituisce 1 se la manche è valida, 0 se invalida.
In questo modo, il contatore **FULLADDER_5b,** si incrementa solamente se la manche giocata è valida.
L'output del contatore viene filtrato dal **MUX_CONTATORE**: se la partita è finita, o è stata resettata **FULLADDER_5b** viene azzerato, altrimenti, continua con il conteggio delle manche valide, che vengono memorizzate nel **REG_4.**

Comparatore Minmanche:
Riceve in input la costante 4 e l'output del **REG_4** (ossia il contatore delle manche valide).
Se il conteggio delle manche è maggiore o uguale al minimo delle manche, viene restituito 0, altrimenti 1.

Comparatore Maxmanche:
Riceve in input l'output del **REG_3** (il massimo delle manche giocabili) e l'output del **REG_4** (il contatore delle manche valide).
Se il conteggio delle manche è uguale al massimo delle manche, viene restituito 0, altrimenti 1.

Statistiche del Circuito

- **Prima dell'ottimizzazione**

```
sis> read_blif FSMD_no.blif
Warning: network `DATAPATH', node "[61]" does not fanout
Warning: network `DATAPATH', node "[167]" does not fanout
Warning: network `FSMD_no', node "[61]" does not fanout
Warning: network `FSMD_no', node "[167]" does not fanout
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes=152       latches=17
lits(sop)=1044
```

Le statistiche del circuito sono le seguenti:

Numero nodi: 152
Numero letterali: 1044

- **Dopo l'ottimizzazione**

```
sis> full_simplify
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes=152       latches=17
lits(sop)= 332
sis> source script.rugged
sis> print_stats
unknown command 'print_stats'
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes= 39       latches=17
lits(sop)= 327
sis> source script.rugged
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes= 35       latches=17
lits(sop)= 317
sis> source script.rugged
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes= 35       latches=17
lits(sop)= 315
sis> source script.rugged
sis> print_stats
FSMD_no         pi= 5   po= 4   nodes= 35       latches=17
lits(sop)= 315
sis> write_blif FSMD.blif
```

Le statistiche post-ottimizzazione sono le seguenti:

Numero di nodi: 35.
Numero di letterali: 315.

Mapping: gate e ritardi

Dopo aver ottimizzato il circuito lo si mappa così da visualizzare le statistiche riguardo area e ritardo. È stata quindi assegnata la libreria **synch.genlib** e il circuito mappato presenta le seguenti statistiche:

```
>>> before removing serial inverters <<<
# of outputs:          21
total gate area:       5584.00
maximum arrival time: (48.80,48.80)
maximum po slack:     (-6.80,-6.80)
minimum po slack:     (-48.80,-48.80)
total neg slack:      (-536.60,-536.60)
# of failing outputs:  21
>>> before removing parallel inverters <<<
# of outputs:          21
total gate area:       5440.00
maximum arrival time: (46.60,46.60)
maximum po slack:     (-6.80,-6.80)
minimum po slack:     (-46.60,-46.60)
total neg slack:      (-503.60,-503.60)
# of failing outputs:  21
# of outputs:          21
total gate area:       5248.00
maximum arrival time: (46.00,46.00)
maximum po slack:     (-6.80,-6.80)
minimum po slack:     (-46.00,-46.00)
total neg slack:      (-495.80,-495.80)
# of failing outputs:  21
sis>
```

Il **total gate area** (area) è di 5248.00, mentre il **maximum arrival time** (cammino critico, ritardo) è di 46.00.

Scelte progettuali

- Nel momento in cui la partita è terminata e ci troviamo nello stato di **FINE**, se l'utente inserisce INIZIA=1, il massimo delle partite viene fornito dalla concatenazione di PRIMO e SECONDO, come da specifica.
  Se invece l'utente inserisce INIZIA=0, il massimo delle partite rimane quello della partita precedente.
  In questo modo l'utente inizierà la partita successiva appena "arrivato" nello stato **INIZIO**.

- Se INIZIA = 0 e REG_3 = 0 (questo può avvenire solo all'inizio della simulazione), REG_3 viene automaticamente impostato a 4 (controllo errore).

- Segnali di controllo tra FSM e DATAPATH: **FINE** (da FSM a DATAPATH) considerato come una sorta di flag, alzato a 1 quando nella FSM si passa dallo stato di FINE a INIZIO; **MINMANCHE** (da DATAPATH a FSM) abbassato a 0 quando viene raggiunto il numero minimo di manche giocabili (4); **MAXMANCHE** (da DATAPATH a FSM) abbassato a 0 quando viene raggiunto il numero di manche giocabili nella partita, il quale è stato stabilito precedentemente; **MANCHE** (da DATAPATH a FSM) serve per muoversi tra gli stati della FSM.