



UNIVERSITÀ
di VERONA

Dipartimento
di **INFORMATICA**

Laboratorio di Architettura degli Elaboratori

Elaborato SIS e Verilog

A.A. 2023/2024

Danny Furfaro VR502846

Giorgia Zanini VR503052

Specifiche e scelte progettuali

Il progetto realizzato prevede la gestione di partite di morra cinese.

Ogni partita si articola di più manche, con le seguenti regole:

- Si devono giocare un minimo di quattro manche e un massimo di diciannove, deciso dai giocatori al momento del reset, in cui le mosse valgono come stringa di bit da sommare a 4 (00100) per ottenere il numero di manche massime per la partita che sta per svolgersi.
Il numero massimo di manche viene anche settato al ciclo di clock in cui viene iniziata la partita, mettendo però il reset a 1 all'inizio di essa;
nota progettuale: nel momento in cui il reset è a 1 le mosse non vengono viste come mosse effettive, ma solo come stringa di bit per determinare le manche massime, di conseguenza non verrà poi conteggiata la manche in cui il reset è a 1.
- Vince il primo giocatore a riuscire a vincere due manche in più del proprio avversario, a patto di aver giocato almeno quattro manche;
o, nel caso di raggiungimento delle manche massime, vince il giocatore che ha un vantaggio rispetto all'altro giocatore,
in caso non ci siano vantaggi, la partita termina in pareggio.
- Ad ogni manche, il giocatore vincente della manche precedente non può ripetere l'ultima mossa utilizzata (*nota progettuale*) solo per il ciclo di clock successivo.
Nel caso lo facesse, la manche non sarebbe valida ed andrebbe ripetuta (quindi, non conteggiata).
In caso di pareggio la manche viene conteggiata. Alla manche successiva, entrambi i giocatori possono usare tutte le mosse.

Il circuito comprende 5 bit di input che si dividono in 3 ingressi, così ordinati:

[2 bit] – mossa del giocatore 1

00 → nessuna mossa,
considerata come mossa non valida, invalida anche la manche

01 → sasso

10 → carta

11 → forbice

[2 bit] – mossa del giocatore 2

Le mosse hanno gli stessi codici del primo giocatore.

[1 bit] – reset

1 → riporta il sistema alla configurazione iniziale
e considera le mosse dei due giocatori come numero da sommare a 4 (00100) per
settare il numero di manche massime della partita che si sta per svolgere.

Nota progettuale: all'avvio inserire il reset a 1 in modo da settare il numero di
manche massime.

0 → la partita procede regolarmente.

Il circuito comprende inoltre 4 bit di output che si dividono in 2 uscite, così ordinati:

[2 bit] – manche (fornisce il risultato dell'ultima manche giocata)

00 → manche non valida

01 → manche vinta dal giocatore 1

10 → manche vinta dal giocatore 2

11 → manche pareggiata

[2 bit] – partita (fornisce il risultato della partita)

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

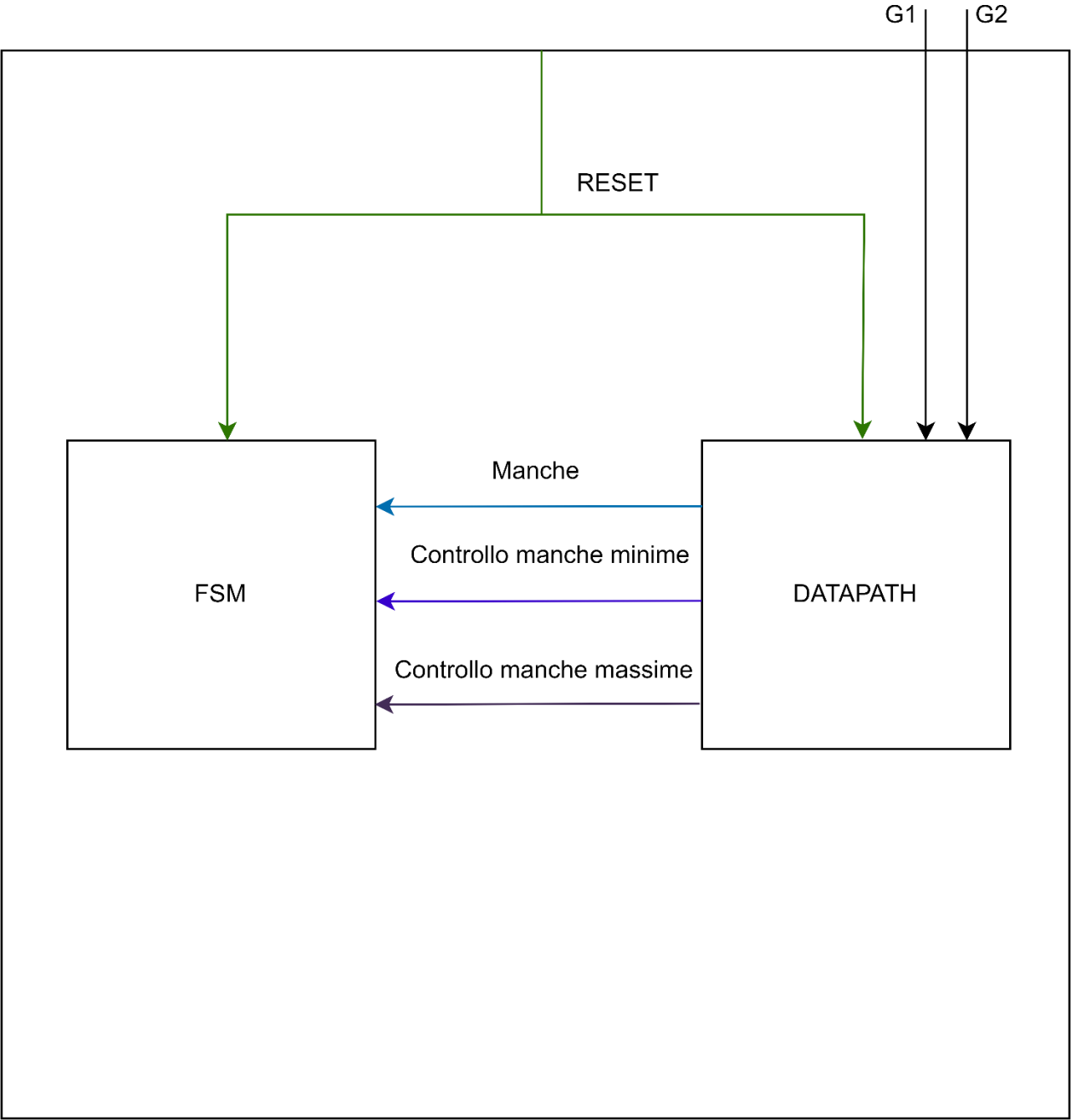
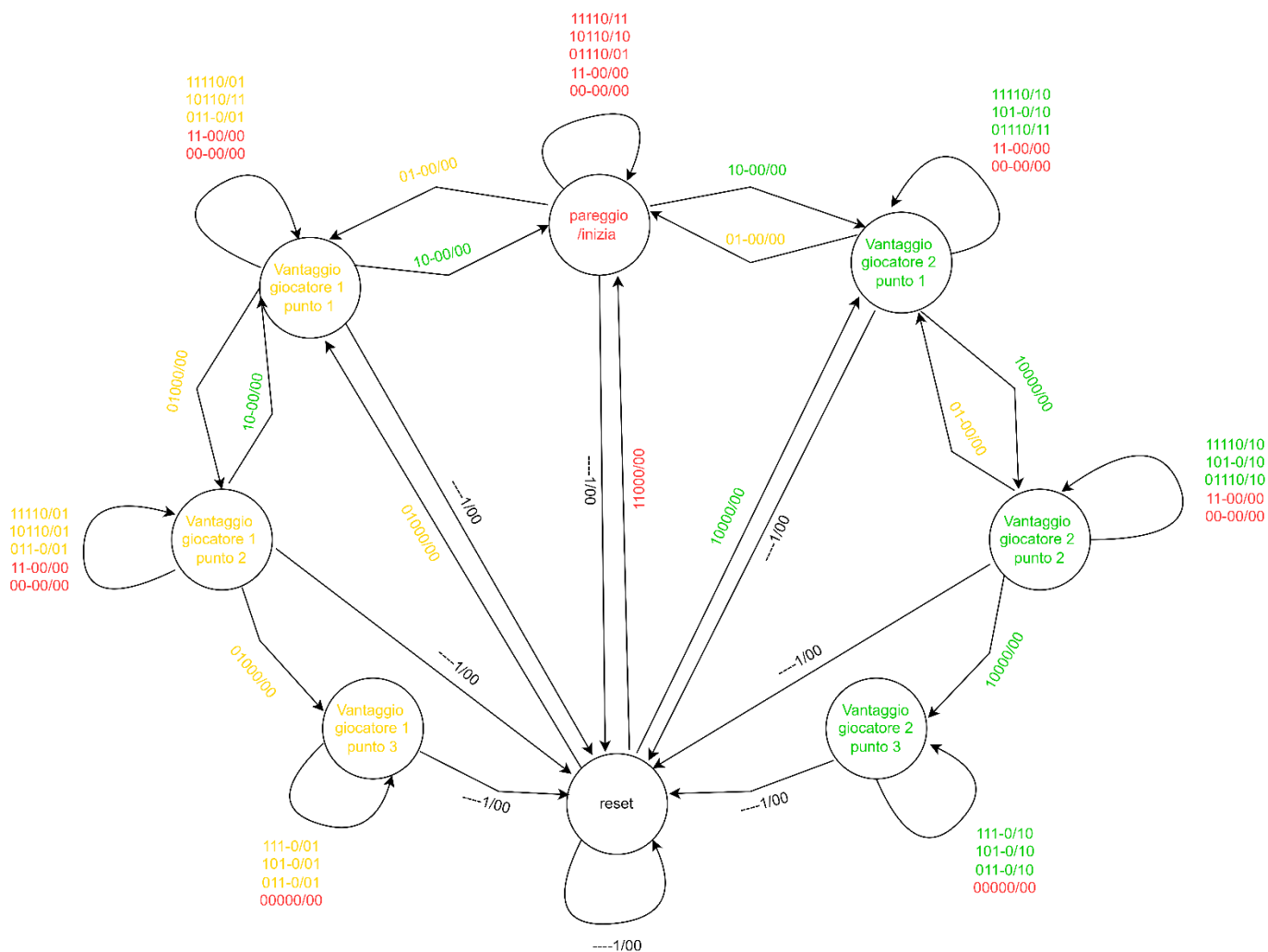


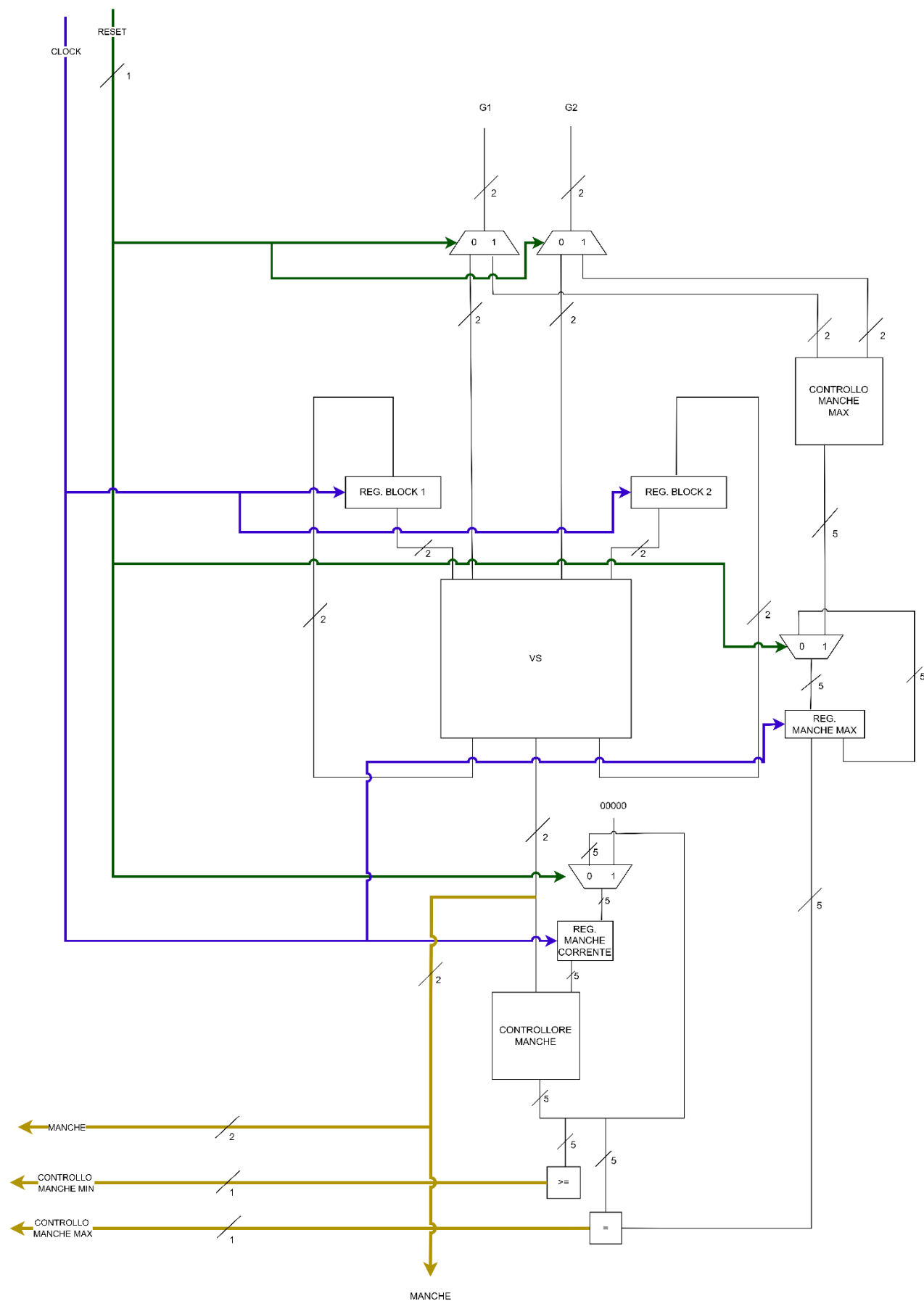
Diagramma degli stati del controllore - STG - FSM

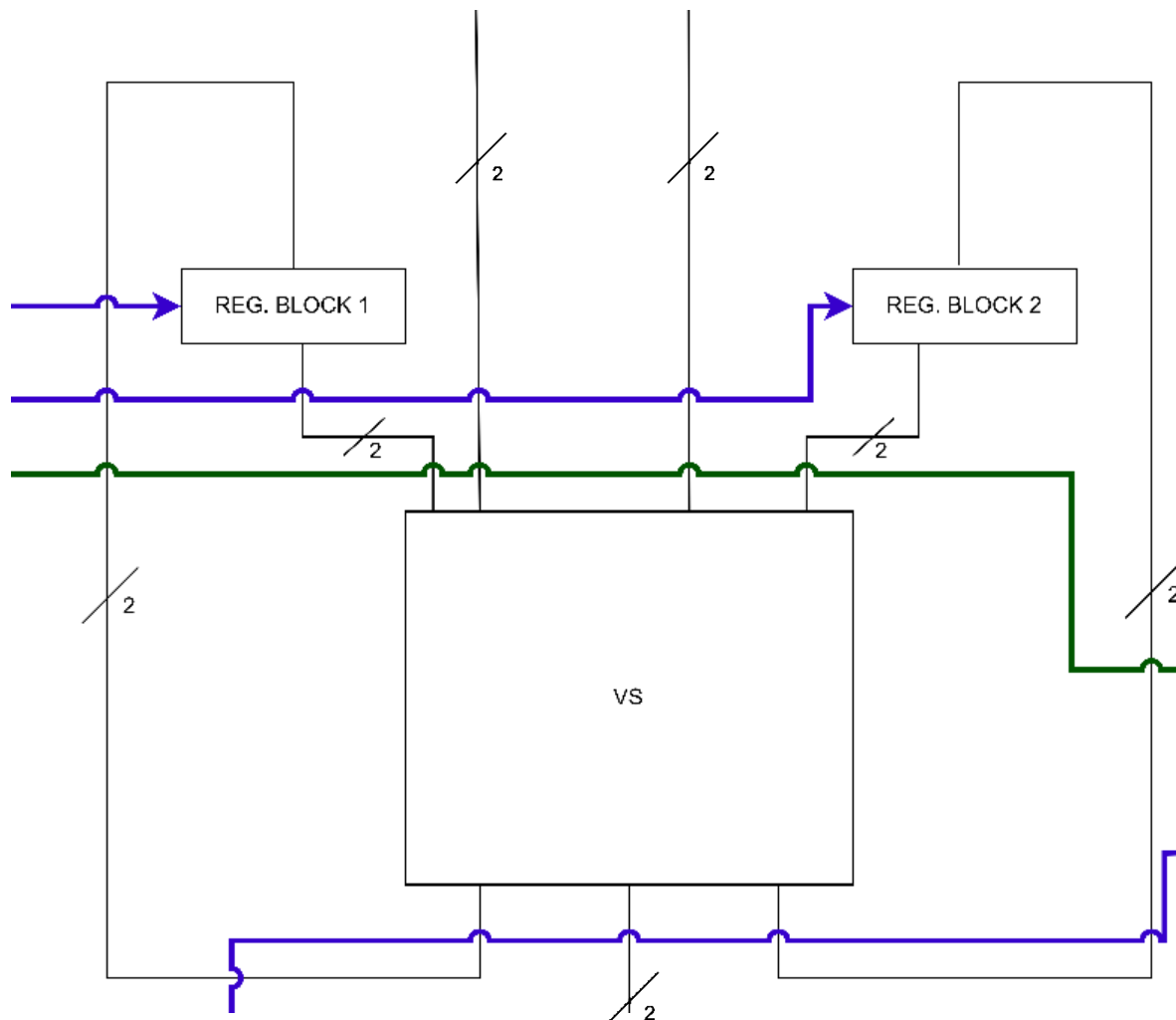


Nella FSM avviene la gestione del vantaggio dei giocatori.

La partita termina nello stato in cui si trova.

Architettura del Datapath





- gestione manche

VS

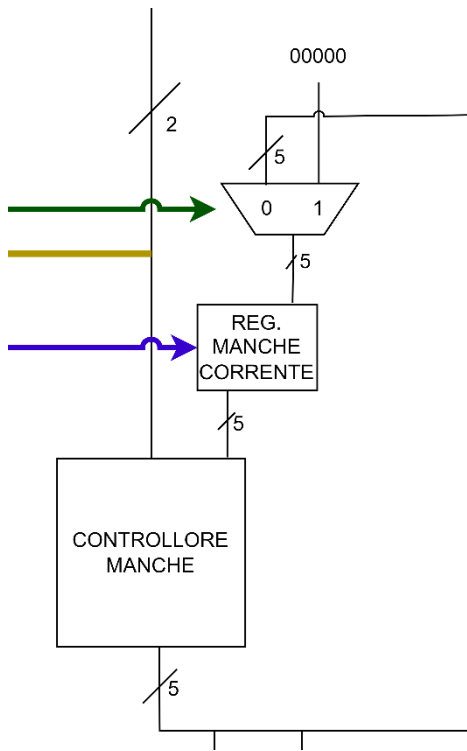
Questo componente ha 4 entrate:

- 2 sono le mosse effettive dei giocatori
- 2 due sono le mosse bloccanti, salvate nei registri che tengono in memoria la mossa del giocatore vincente dal ciclo di clock precedente

REG. BLOCK 1 – 2

Qui viene salvata la mossa del giocatore vincente, che poi al ciclo di clock successivo verrà confrontata con la mossa del rispettivo giocatore, per verificare se può giocare una la mossa.

In caso di pareggio, manche non valida o il giocatore perda la manche, nel suo rispettivo registro verrà salvato 00 (jolly).



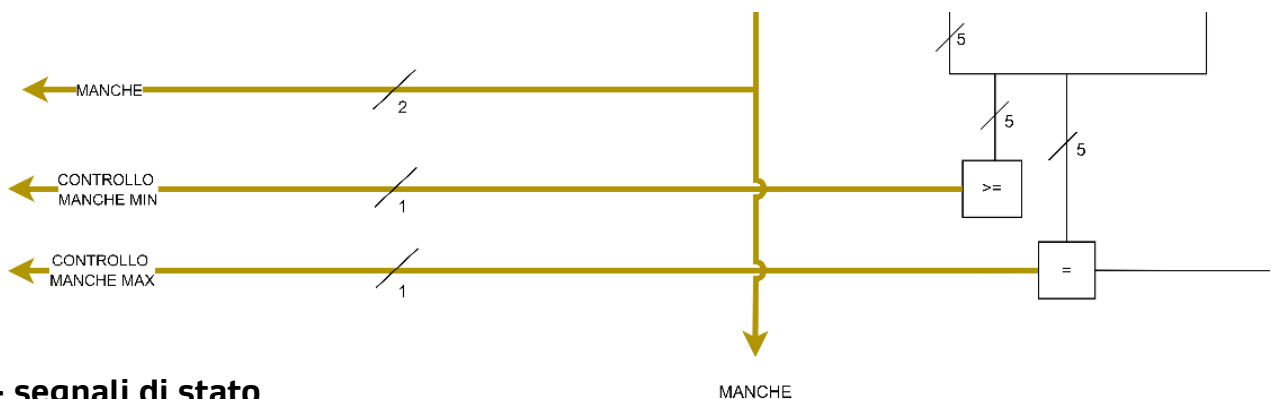
- contatore

MUX – REG. MANCHE CORRENTE

In caso di reset (1) lascia passare una stringa di 0 che andrà a resettare il registro delle manche correnti, altrimenti lascia passare il valore delle manche correnti, che andrà salvato nel registro.

CONTROLLORE MANCHE

Se la manche è valida aggiunge 1 al conteggio delle manche correnti, se non è valida manda in output lo stesso numero di manche correnti.



- segnali di stato

MANCHE

Output del componente VS che determina lo stato della manche e in caso chi la vince.

\geq → MIN

Controllo per verificare se si ha raggiunto il minimo di 4 manche valide.

$=$ → MAX

Controllo per verificare se si ha raggiunto il numero di manche massime impostato dai giocatori all'inizio della partita.

Ottimizzazione

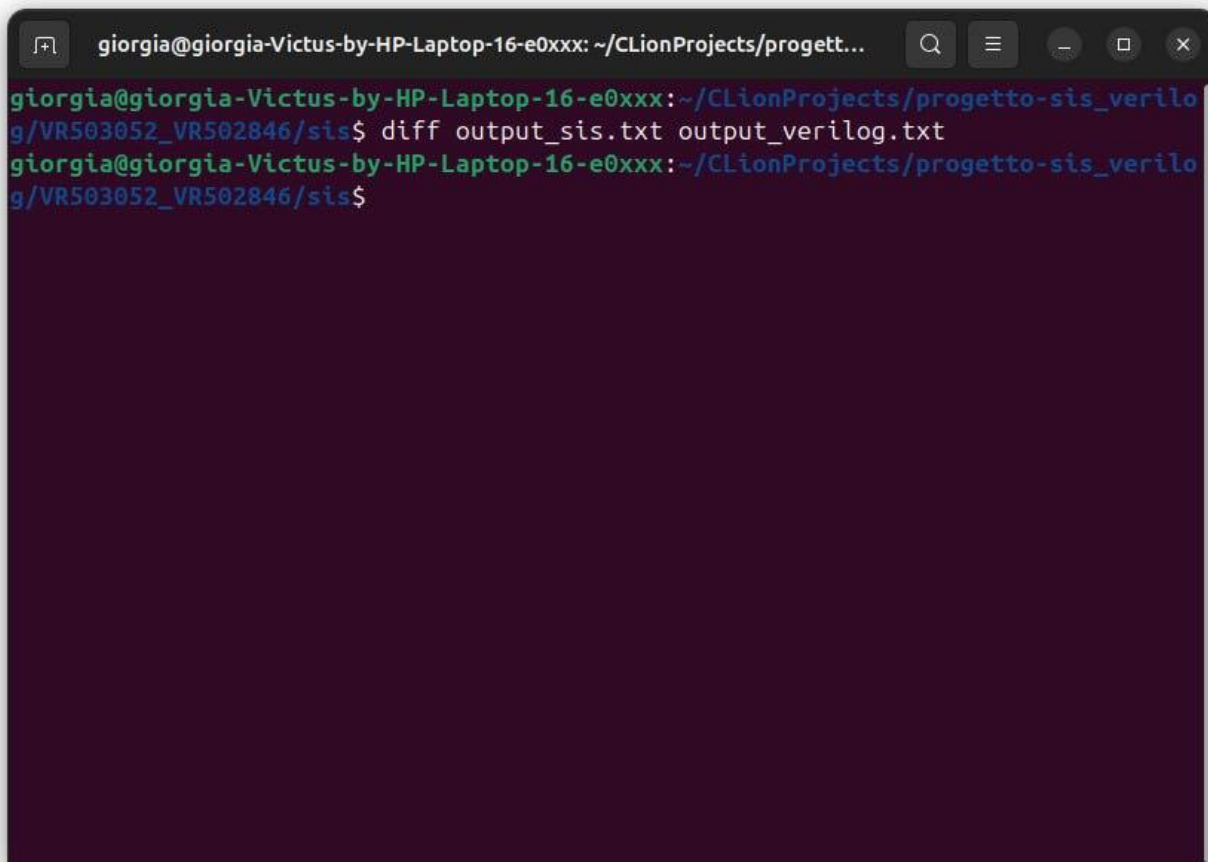
```
bsis> read_blif fsmd.blif
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 41   latches=17
lits(sop)=2947
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 41   latches=17
lits(sop)= 617
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 41   latches=17
lits(sop)= 607
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 41   latches=17
lits(sop)= 607
bsis> source script.rugged
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 48   latches=17
lits(sop)= 411
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 48   latches=17
lits(sop)= 411
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 48   latches=17
lits(sop)= 411
bsis> source script.rugged
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 49   latches=17
lits(sop)= 400
bsis> full_simplify
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 49   latches=17
lits(sop)= 400
bsis> source script.rugged
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 50   latches=17
lits(sop)= 399
bsis> source script.rugged
bsis> print_stats
fsmd      pi= 5   po= 4   nodes= 52   latches=17
lits(sop)= 399
bsis> write_blif ../FSMD.blif
```

Technology mapping

```
bsis> read_library synch.genlib
bsis> map -m 0 -s

warning: unknown latch type at node '{[19]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[20]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[21]}' (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: 7368.00
maximum arrival time: (76.40,76.40)
maximum po slack: (-8.00,-8.00)
minimum po slack: (-76.40,-76.40)
total neg slack: (-779.80,-779.80)
# of failing outputs: 21
>>> before removing parallel inverters <<<
# of outputs: 21
total gate area: 7304.00
maximum arrival time: (71.60,71.60)
maximum po slack: (-8.00,-8.00)
minimum po slack: (-71.60,-71.60)
total neg slack: (-746.20,-746.20)
# of failing outputs: 21
# of outputs: 21
total gate area: 6840.00
maximum arrival time: (69.20,69.20)
maximum po slack: (-8.00,-8.00)
minimum po slack: (-69.20,-69.20)
total neg slack: (-722.80,-722.80)
# of failing outputs: 21
```

Confronto degli output Sis-Verilog



```
giorgia@giorgia-Victus-by-HP-Laptop-16-e0xxx: ~/CLionProjects/progett...
giorgia@giorgia-Victus-by-HP-Laptop-16-e0xxx:~/CLionProjects/progetto-sis_verilog/VR503052_VR502846/sis$ diff output_sis.txt output_verilog.txt
giorgia@giorgia-Victus-by-HP-Laptop-16-e0xxx:~/CLionProjects/progetto-sis_verilog/VR503052_VR502846/sis$
```