Architettura degli Elaboratori

Relazione Elaborato SIS - Verilog

Bortolaso Mattia - VR500026
Colombo Matteo - VR500130

SOMMARIO

Architettura generale del circuito

L'elaborato assegnato prevede la realizzazione di una macchina in grado di gestire partite di MorraCinese.
Il circuito è composto da una FSM (controllore) e un DATAPATH (elaboratore) che comunicano tra loro. Il progetto è stato sviluppato utilizzando tecnologie differenti: SIS e Verilog. I file che contengono la rappresentazione di queste componenti sono presenti nelle rispettive cartelle, sis e verilog. Nella prima cartella con i nomi di fsm.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; nella seconda cartella, FSM e Datapath sono descritti nel file design.sv. Nelle pagine seguenti verranno descritte nel dettaglio le due sezioni della macchina analizzando le componenti utilizzate.

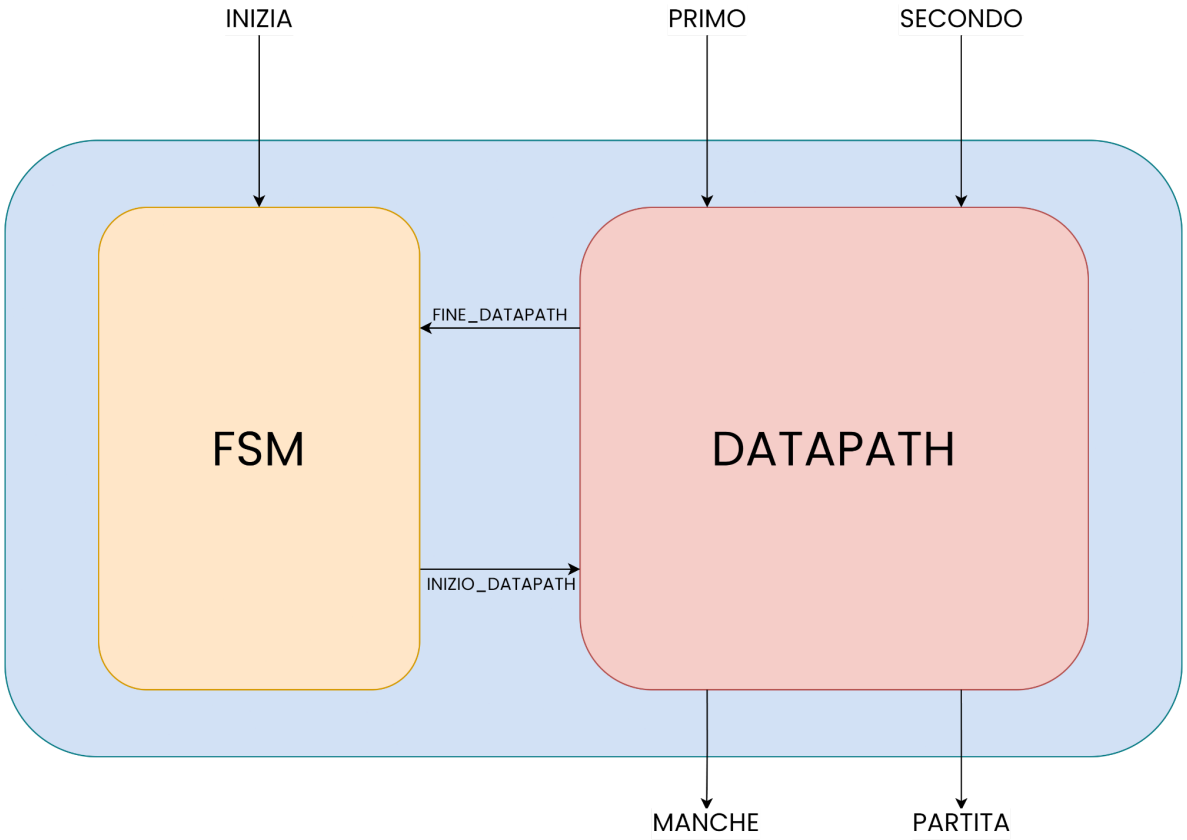


Diagramma della macchina a stati finiti

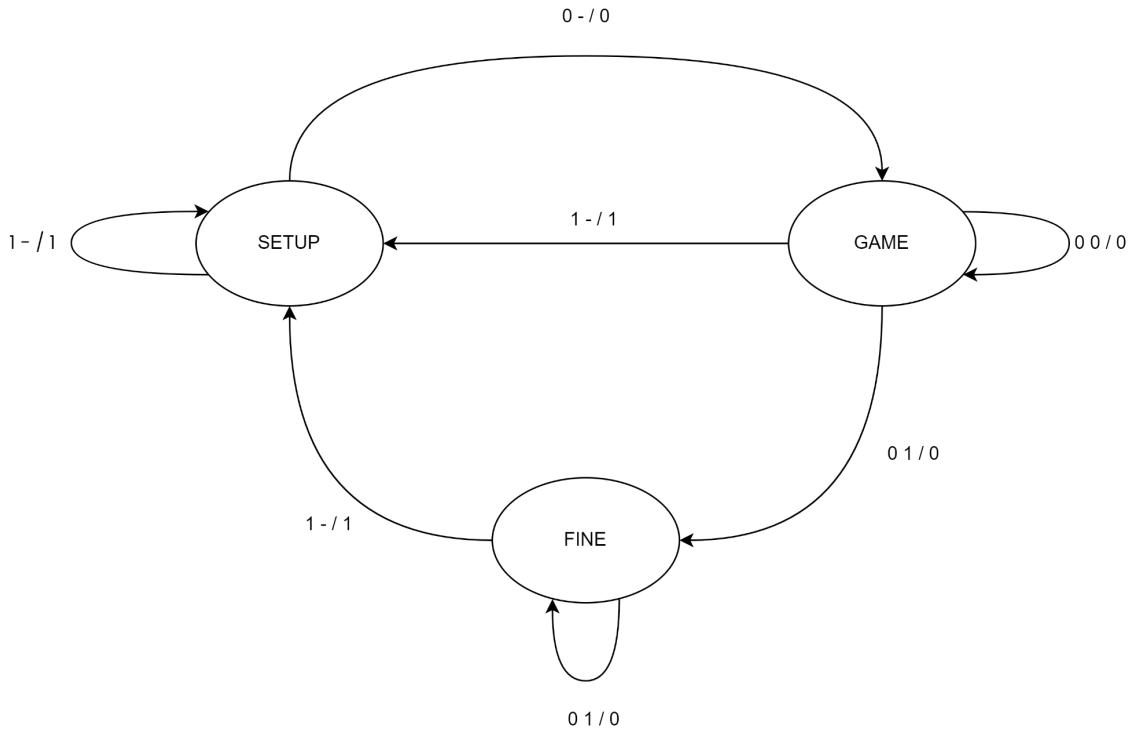
Il controllore della macchina per la vittoria al gioco muraglia cinese è una macchina a stati finiti del tipo Mealy. Si presentano i seguenti segnali:

Segnali di input	Segnali di output
INIZIO[1]	inizio_datapath[1]
fine_datapath[1]	

La FSM presenta 3 stati:

- **SETUP:** è lo stato di reset in cui il controllore fin quando trova il segnale INIZIO alzato dice al datapath di impostare il numero manche massime che si possono giocare durante la partita. Non appena il segnale INIZIO si abbassa passa allo stato di GAME e la partita inizia.
- **GAME:** in questo stato si giocano le varie manche gestite dal datapath, se il segnale INIZIO torna a 1 il circuito si resetta e torna nello stato di setup, se il segnale di fine_datapath è 1 passa allo stato di FINE.
- **FINE:** stato in cui la FSM riceve dal datapath il segnale di fine partita e attende un nuovo segnale di RESET per cominciarne un'altra.

In qualunque momento è possibile ripristinare la macchina tramite il bit di INIZIO.



Architettura del datapath

Il Datapath è composto da due sezioni:

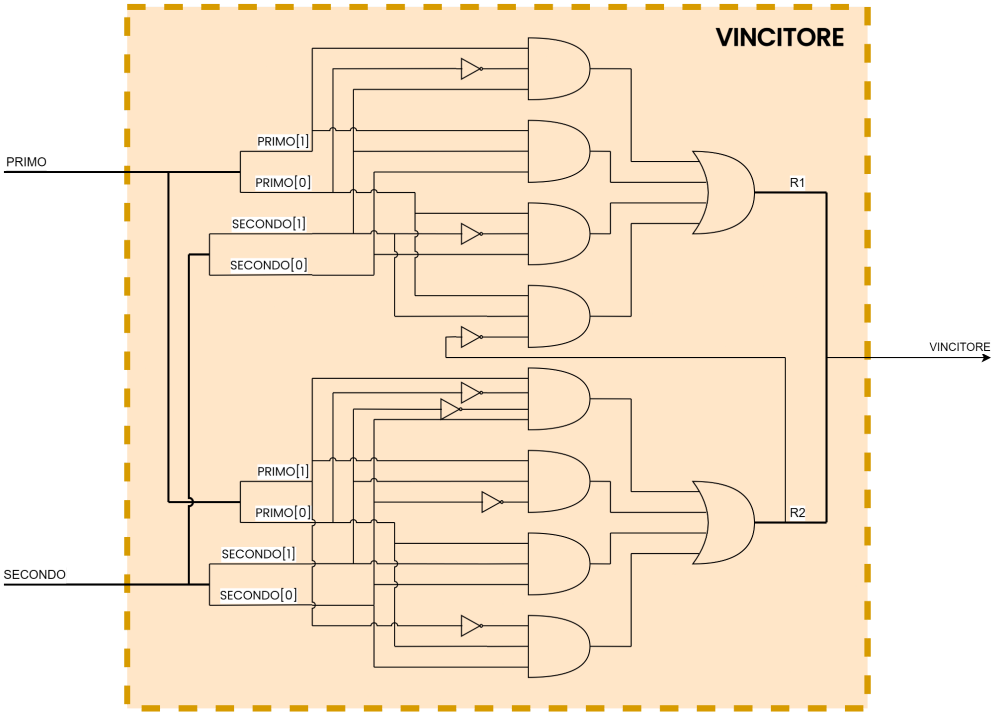
- **VINCITORE MANCHE**: si occupa dell'elaborazione delle mosse giocate dai due giocatori e alla dichiarazione del vincitore della manche in base ai controlli sulle ultime mosse giocate.
- **VINCITORE PARTITA**: decreta il vincitore della partita basandosi sul numero di manche massime impostate e in base al vantaggio tra i due giocatori.

Vincitore Manche

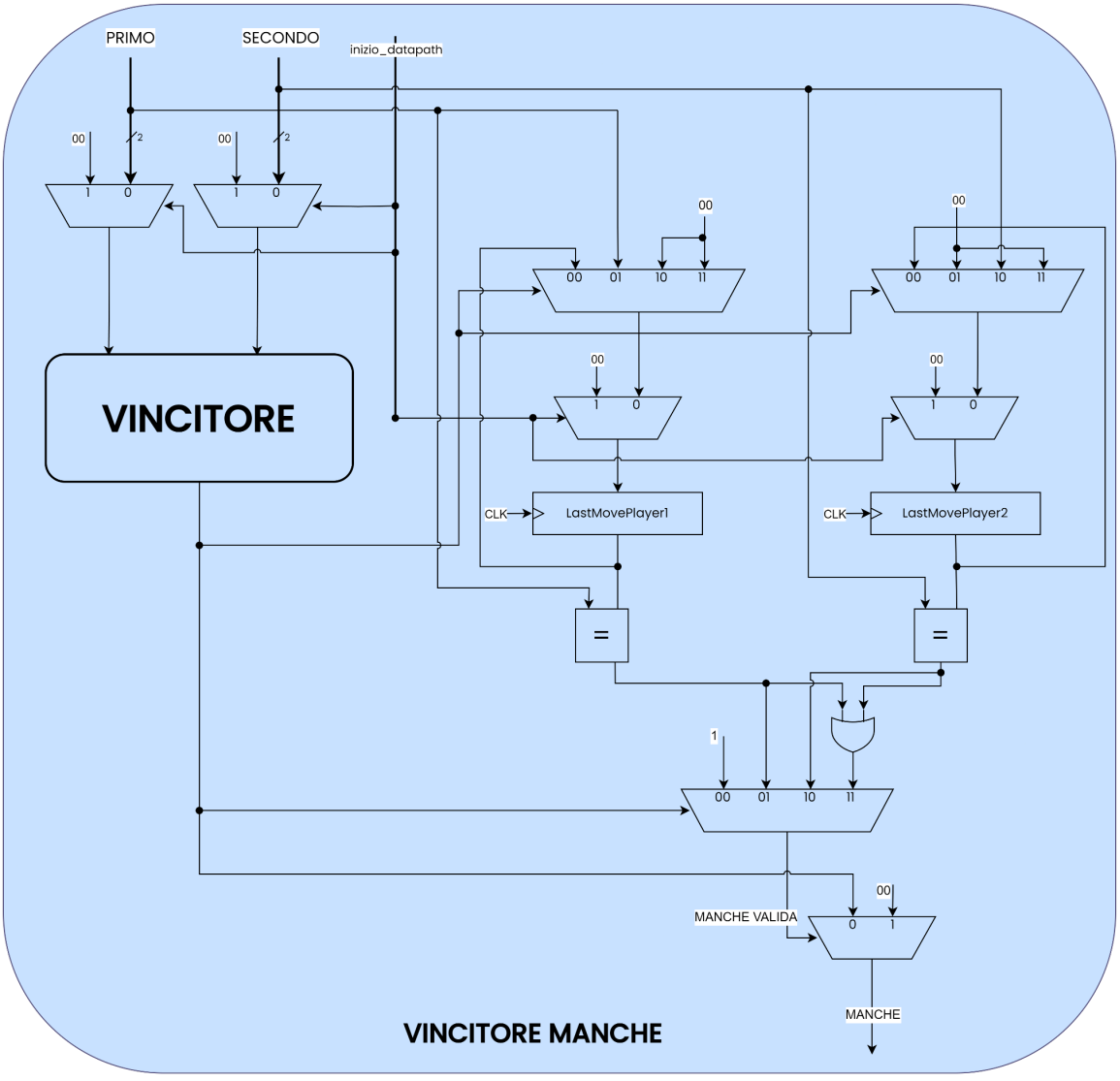
Questa parte di datapath si occupa inizialmente di decretare il vincitore della manche confrontando le mosse giocate dai giocatori, successivamente verifica se la manche è valida in base al vincolo: "*il giocatore vincente della manche precedente non può ripetere l'ultima mossa*". Il controllo viene effettuato per mezzo di due registri che memorizzano la mossa vincente in base a chi dei due giocatori vince. Se vince il giocatore 1, nel registro LastMovePlayer1 viene memorizzata la mossa utilizzata da G1, e in LastMovePlayer2 viene inserito 00. Viceversa in caso di vittoria di G2. Quando è pareggio entrambi i registri vengono scritti a 00: quindi i due giocatori possono giocare le mosse che vogliono per vincere la manche successiva.

Entrando più nel dettaglio, i componenti utilizzati sono:

- **VINCITORE**: un componente logico composto da varie porte logiche in grado di decretare il vincitore della manche unicamente confrontando le mosse giocate. Accetta in input 2 ingressi da 2 bit ciascuno che corrispondono a PRIMO e SECONDO, e un output a 2 bit VINCITORE.



- **Quattro multiplexer** da **2 ingressi** controllati dal segnale inizio_datapath che servono a resettare i registri ed evitare che PRIMO e SECONDO entrino nel componente VINCITORE quando il segnale INIZIO (setup del numero max di round) viene alzato.
- Coppia di **registri LastMovePlayer1** e **LastMovePlayer2** utilizzati per tenere traccia delle ultime mosse utilizzate dai due giocatori.
- **Due multiplexer a 4 ingressi**, uno per registro, utilizzati prima dei registri LastMovePlayer per poter decidere quali valori inserire nei registri. Quando VINCITORE è 00 significa che uno dei due giocatori ha inserito una mossa non valida, quindi la manche non è valida e i registri si riscrivono. Quando VINCITORE vale 01 significa che ha vinto il giocatore 1 e quindi va scritta la mossa utilizzata nel relativo registro G1 e 00 nel registro di G2. Viceversa nel caso di vittoria del G2. Quando è pareggio entrambi i registri vengono scritti a 00, togliendo il vincolo ad entrambi i giocatori.
- Coppia di **comparatori** che confrontano i valori dei registri con la mossa appena giocata dai due giocatori, restituendo in output 1 nel caso di uguaglianza.
  Per il pareggio i segnali dei due comparatori sono stati combinati con una porta OR.
- **Multiplexer a 4 ingressi** ciascuno controllato da VINCITORE, ha la funzione di far passare il segnale del comparatore corrispondente al vincitore della giocata.
- **Multiplexer a 2 ingressi** che in base al segnale ricevuto dal comparatore corrispondente al vincitore, decreta se la manche è valida o meno. Se è valida lascia passare il segnale VINCITORE, in caso contrario 00.



Vincitore Partita

Questa parte del datapath effettua il calcolo dei round massimi, tiene conto dei round, calcola il vantaggio e decide chi vince la partita. Serve per calcolare l'output partita e rispettare tutti i vincoli come il vantaggio maggiore di due da parte di G1 o G2, il numero minimo di partite impostato a 4 e di non sforare le 19 partite massime.

- **REGISTRO VANTAGGIO:** in ingresso a questo registro abbiamo un multiplexer a due ingressi che viene pilotato da **INIZIO**, se **INIZIO** è zero riscrive il registro, se INIZIO è uno scrive nel registro 0100 (4 in decimale) in modo da avere 6 se G2 ha un vantaggio di due e 2 se G1 ha un vantaggio di due. Dopo il registro abbiamo utilizzato un sottrattore e un sommatore per decrementare di uno il valore del registro e aumentare di uno il valore del registro per poi decidere che valore utilizzare tramite un multiplexer a 4 ingressi pilotato da **MANCHE**, su 00 e 11 troviamo il vantaggio non manipolato,su 01 abbiamo il registro sottratto e su 10 il registro dopo la somma, l'uscita la chiamiamo Vantaggio_Mux.

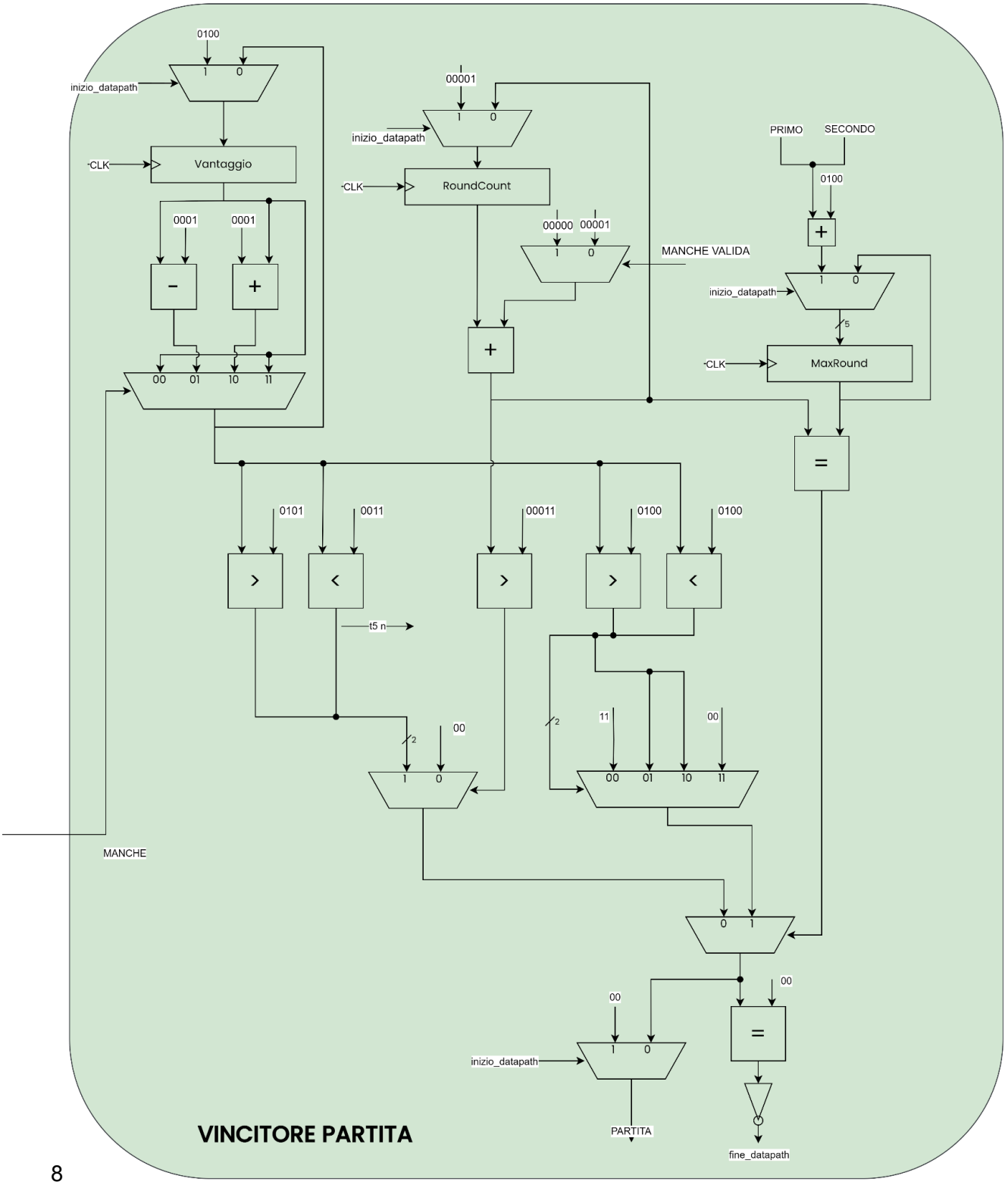
- **CONTEGGIO ROUND MASSIMI:** Uniamo **PRIMO** e **SECONDO** in un unico segnale da 4 bit e li mandiamo all'ingresso di un sommatore dove gli sommiamo 0100 (4 in decimale) e l'uscita la mandiamo al bit 1 di un multiplexer controllato da **inizio_datapath** mentro al bit 0 andiamo a riscrivere il registro. Dopo questo mux abbiamo un registro chiamato **MaxRound** che tiene conto dei round massimi che si possono giocare. L'uscita del registro **MaxRound** la compariamo all'uscita del registro Sommatore_Partite, questo segnale lo chiameremo Max_Round_Check.

- **CONTEGGIO ROUND:** Abbiamo un multiplexer pilotato da inizio_datapath, su 0 abbiamo l'uscita del sommatore mentre su 1 abbiamo 00001 (1 in decimale), l'uscita di questo multiplexer entra in un registro chiamato **RoundCount**. L'output di questo registro è collegata ad un sommatore, abbiamo un multiplexer pilotato da **MANCHE VALIDA** dove sull'ingresso 0 abbiamo 00001 e su 1 abbiamo 00000, l'uscita è collegata al sommatore, il multiplexer serve per decidere se sommare uno o zero in base alla validità della manche. L'uscita del sommatore, che chiamiamo Sommatore_Partite va all'ingresso di un maggiore che controlla se le partite giocate sono maggiori di 4 e la partita può finire, l'uscita del maggiore la chiameremo Partite_Min_Giocate.

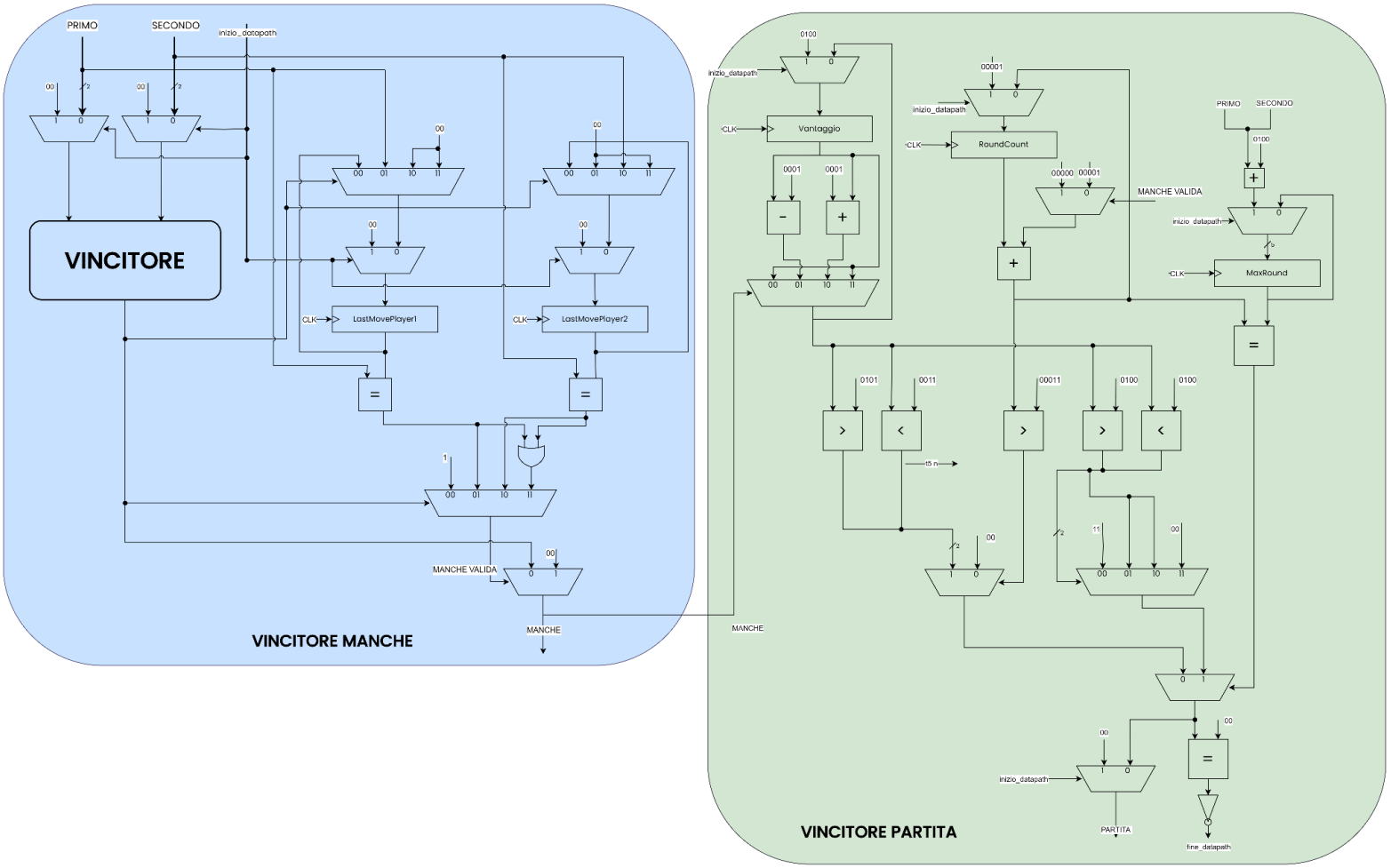
- **CONTROLLO VANTAGGIO:** Utilizziamo il maggiore e il minore per controllare il vantaggio che abbiamo impostato e decretare se il vincitore è G1 o G2. Utilizziamo un maggiore che prende Vantaggio_Mux e controlla se è maggiore di 0101 (5 in decimale), utilizziamo un minore per controllare se lo stesso valore è minore di 0011 (3 in decimale), combiniamo le uscite e le mandiamo a un multiplexer pilotato a 2 ingressi dove su 0 abbiamo 00 e su 1 abbiamo le nostre uscite combinate, l'uscita di questo multiplexer la chiameremo Controllo_1. Andiamo a vedere se l'uscita del multiplexer del registro vantaggio è maggiore di 0100 (4 in decimale) tramite un maggiore e andiamo anche a controllare se è minore di 0100 (4 in decimale) per poi concatenare le uscite e andare a pilotare un multiplexer a 4 ingressi, su questo multiplex abbiamo sull'ingresso 00 11 sull'ingresso 01 abbiamo il nostro segnale combinato che esce dal maggiore e minore su 01 abbiamo sempre il nostro segnale

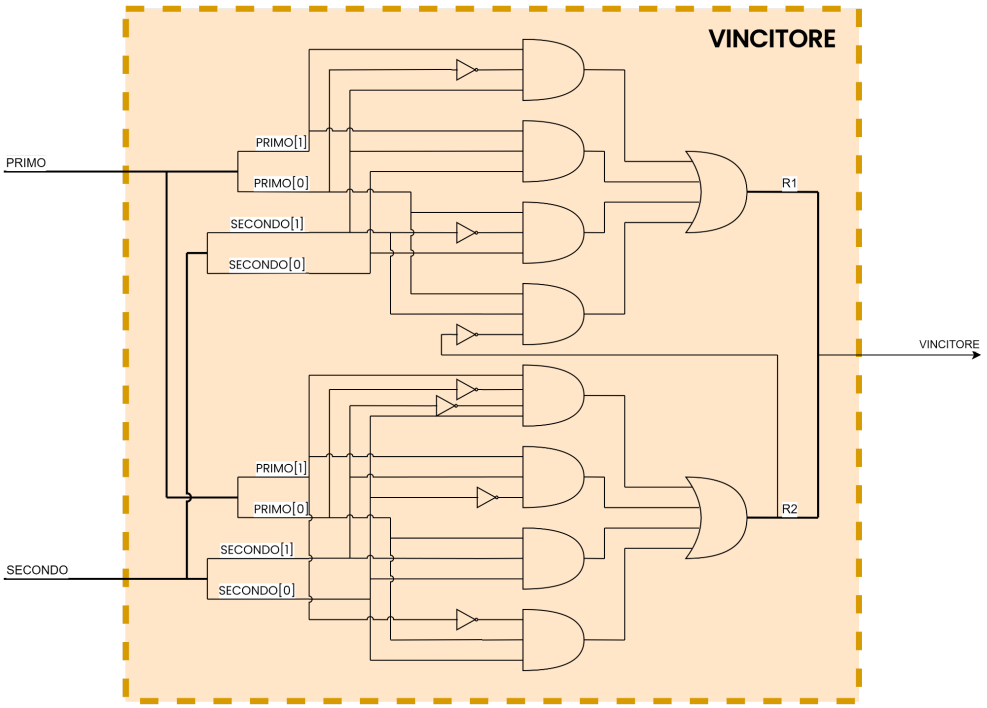
e su 11 abbiamo 00, questa uscita la chiameremo Controllo_2. Abbiamo dopo un mux che controllato da Partite_Min_Giocate che sul bit 0 ha Controllo_1 e sull'1 ha Controllo_2, l'uscita del mux la chiameremo Mux_Check_Exit.

- **CONTROLLI FINALI:** Prendiamo Mux_Check_Exit e usiamo un comparatore per controllare che è uguale a 0, e all'uscita del comparatore abbiamo una porta logica NOT per invertire l'output che chiameremo **fine_datapath**. Abbiamo un multiplexer pilotato da **inizio_datapath** che in ingresso sul bit 0 ha l'uscita di Mux_Check_Exit e sul bit 1 ha 00, questa sarà il nostro output **PARTITA**.



Schema completo del Datapath





Statistiche del circuito

Pre-ottimizzazione

Le statistiche del circuito prima di effettuare l'ottimizzazione indicano:

```
bsis> print_stats

FSMD            pi= 5   po= 4   nodes=168       latches=20
lits(sop)= 833

bsis> full_simplify
```

- Numero di nodi: 168
- Numero di letterali: 833

Post-ottimizzazione

L'ottimizzazione è stata eseguita facendo vari tentativi tramite l'esecuzione di varie sequenze dei comandi *full_simplify* e *source script.rugged* che permette la distruzione e ricostruzione della fsmd. L'ottimizzazione migliore che si è riusciti ad ottenere è di **322 letterali** e **44 nodi**.

```
bsis> full_simplify

bsis> print_stats

FSMD            pi= 5   po= 4   nodes=168       latches=20
lits(sop)= 361

bsis> source script.rugged

bsis> print_stats

FSMD            pi= 5   po= 4   nodes= 44       latches=18
lits(sop)= 331

bsis> full_simplify

bsis> print_stats

FSMD            pi= 5   po= 4   nodes= 44       latches=18
lits(sop)= 329

bsis> source script.rugged

bsis> print_stats

FSMD            pi= 5   po= 4   nodes= 44       latches=18
lits(sop)= 322
```

Mapping del circuito

Una volta effettuato il mapping del circuito per mezzo della libreria synch.genlib, sono stati ottenuti i seguenti risultati:

- Total Area: **4912.00**
- Gate Count: **129**
- Arrival time (cammino critico): **51.80**

```
>>> before removing serial inverters <<<
# of outputs:          22
total gate area:       5168.00
maximum arrival time: (60.20,60.20)
maximum po slack:     (-3.80,-3.80)
minimum po slack:     (-60.20,-60.20)
total neg slack:      (-588.20,-588.20)
# of failing outputs:  22
>>> before removing parallel inverters <<<
# of outputs:          22
total gate area:       5024.00
maximum arrival time: (53.00,53.00)
maximum po slack:     (-3.80,-3.80)
minimum po slack:     (-53.00,-53.00)
total neg slack:      (-571.40,-571.40)
# of failing outputs:  22
# of outputs:          22
total gate area:       4912.00
maximum arrival time: (51.80,51.80)
maximum po slack:     (-3.80,-3.80)
minimum po slack:     (-51.80,-51.80)
total neg slack:      (-563.60,-563.60)
# of failing outputs:  22

bsis> print_map_stats

Total Area              = 4912.00
Gate Count              = 129
Buffer Count            = 9
Inverter Count          = 27
Most Negative Slack     = -51.80
Sum of Negative Slacks  = -563.60
Number of Critical PO   = 22
```

Scelte progettuali

- In fase di progettazione è stato scelto di affidare l'intera parte di elaborazione al datapath e lasciare alla FSM il compito di passare da una fase di gioco a un'altra.
- Per decretare il vincitore unicamente dalle mosse dei due giocatori è stato creato un componente logico ad hoc composto da varie porte logiche.
- Per gestire il vantaggio tra i due giocatori si è scelto di utilizzare un registro impostato a 4 bit dove ogni volta che il G2 vince, viene incrementato di 1, mentre quando il G1 vince viene decrementato di 1. Si è scelto il valore di reset a 4 perché, dato che il numero minimo di manche per vincere è 4, nel caso in cui un giocatore vincesse le prime quattro manche, con un registro a 2 bit non si sarebbe riuscito a salvare il vantaggio accumulato.
- Per gestire il contatore dei round si è deciso di utilizzare un registro a 5 bit e un sommatore per incrementare ad ogni partita 0 o 1 in base alla validità della MANCHE.