ELABORATO SIS-VERILOG 2023/2024

ARCHITETTURA DEGLI ELABORATORI

VR500307: FRANCESCO FRISON

VR504966: MARCO MAZZUCATO

DESCRIZIONE DELL'ELABORATO

Si richiede la progettazione di un circuito sequenziale per la gestione del gioco "**Morra Cinese**".

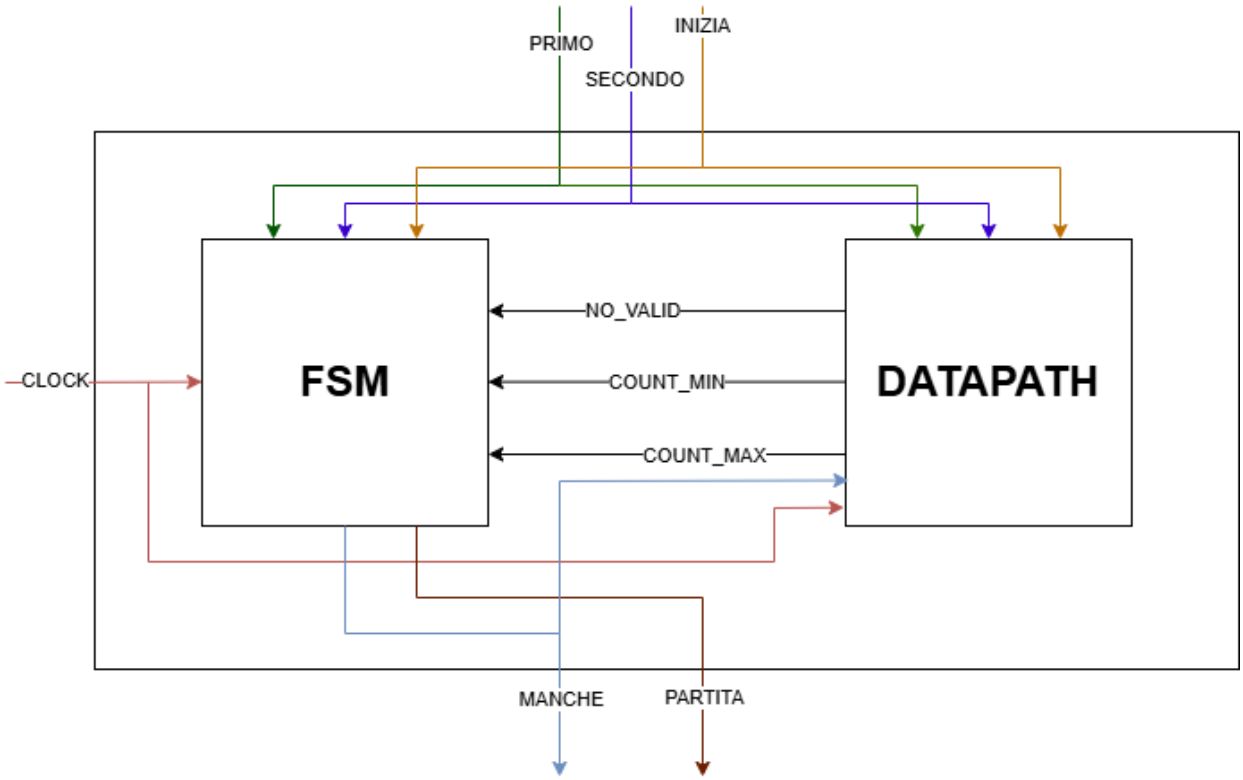
Due giocatori inseriscono le proprie mosse, che possono essere **sasso, carta, forbici oppure mossa non valida**. Il giocatore vincente di ogni manche viene calcolato in base alle classiche regole (sasso batte forbici, forbici batte carta, carta batte sasso).

Per vincere la partita occorre che siano state giocate un minimo di 4 manche, ed un massimo di 19. Il numero di manche giocabili è inserito all'avvio del gioco. Vince il giocatore che riesce a guadagnare un vantaggio maggiore di due rispetto l'altro, a patto che siano state giocate il minimo di partite.

SPECIFICHE DEL PROGETTO

- Se non si verifica la condizione di vittoria ma si è raggiunto il numero massimo di manche giocabili, si termina forzatamente la partita e si decreta il vincitore in base ai punteggi totali;
- La manche è decretata come valida se uno dei due giocatori vince sull'altro, oppure avviene un pareggio;
- **Il giocatore che ha vinto la manche precedente non può scegliere la stessa mossa che ha utilizzato per vincere**, altrimenti la manche sarà decretata non valida;
- **In caso di pareggio, alla manche successiva entrambi i giocatori possono giocare qualunque mossa**.

ARCHITETTURA DEL CIRCUITO (FSMD)

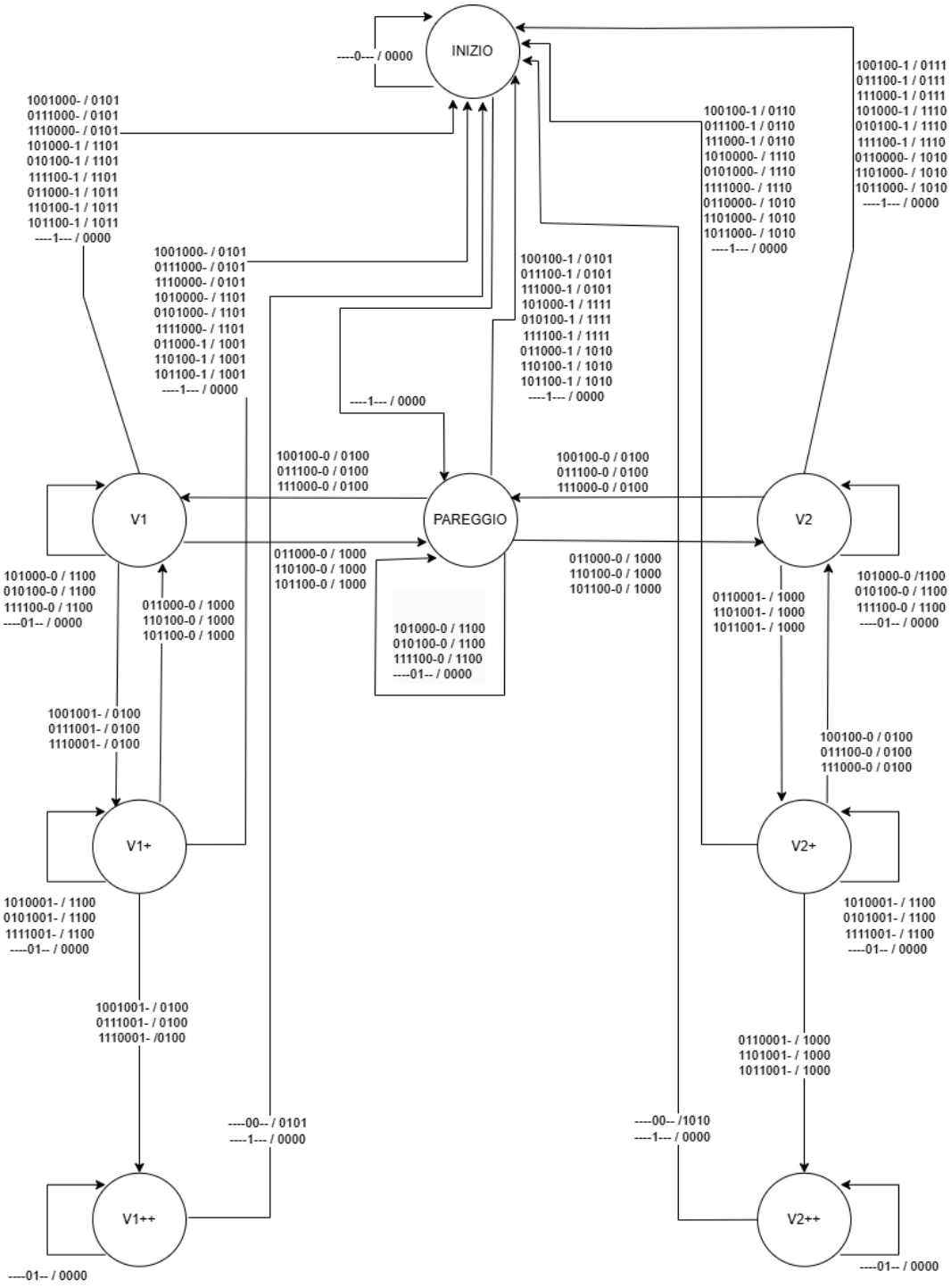


Abbiamo creato i seguenti segnali di stato provenienti dal Datapath per aiutare la FSM a cambiare di stato correttamente:

- **NO_VALID**: segnale ad 1 bit che specifica se la manche che è stata inserita dai giocatori deve essere considerata valida o meno. Viene quindi valorizzato a 1 se la manche non è valida, 0 altrimenti;
- **COUNT_MIN**: segnale ad 1 bit che specifica se ci si trova sotto il minimo di partite da giocare. Viene valorizzato a 1 fino a che non sono state giocate almeno 4 partite, diventa 0 quando le partite minime sono state raggiunte;
- **COUNT_MAX**: segnale ad 1 bit che dichiara se è stato raggiunto il massimo di partite giocabili. Decreta quindi se la partita deve terminare forzatamente, valendo 1, oppure no, valendo 0.

MANCHE viene utilizzato nel Datapath per sapere a quale giocatore confrontare la mossa che ha inserito precedentemente, in modo da poter determinare la validità della manche successiva. Inoltre, è anche utilizzato per tenere conto del numero delle manche valide giocate.

DIAGRAMMA DEGLI STATI DEL CONTROLLORE (FSM)



SEMANTICA DEI BIT DELLA FSM:

- **INPUT:**
  - PRIMO [2]
  - SECONDO [2]

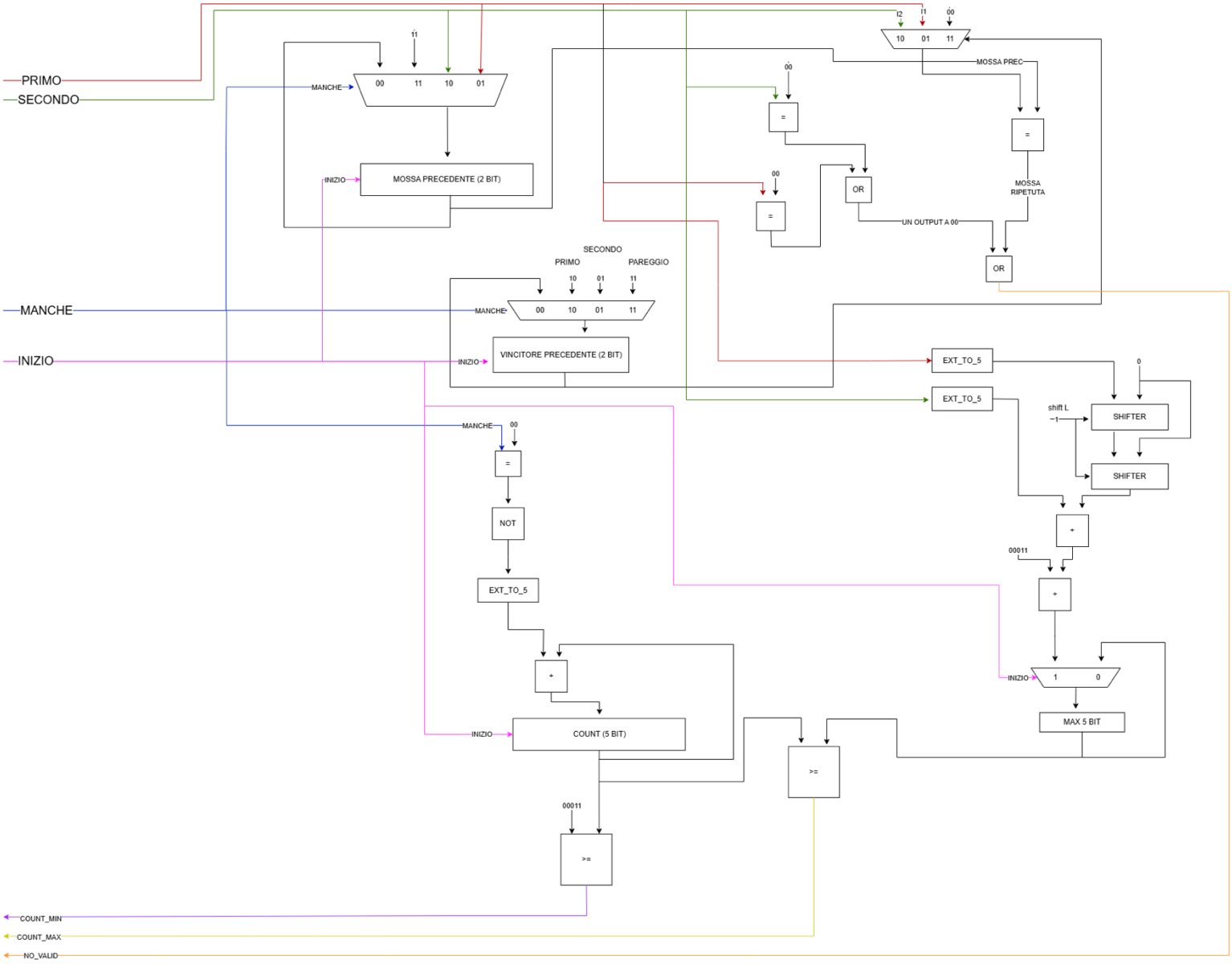
    - INIZIA [1]
    - NO_VALID [1]
    - COUNT_MIN [1]
    - COUNT_MAX [1]
- **OUTPUT:**
    - MANCHE [2]
    - PARTITA [2]

SEMANTICA DEGLI STATI:

- **INIZIA**: stato iniziale della FSM e anche stato di reset. Se viene impostato *INIZIA* ad 1, passa allo stato di **PAREGGIO**, altrimenti rimane sempre in **INIZIA**;
- **PAREGGIO**: stato in cui i giocatori sono in pareggio, quindi nessun vantaggio. Se la partita deve terminare (*INIZIA* oppure *COUNT_MAX* attivo) la partita termina con vincita del giocatore che ha vinto l'ultima manche;
- **V1**: stato in cui il giocatore 1 è in vantaggio. Nel caso in cui *COUNT_MIN* è 0, se dovesse vincere la manche il giocatore 1, allora esso vincerebbe anche la partita. Se, invece, *COUNT_MIN* vale 1, la partita non termina e si passa a **V1+**;
- **V1+**: stato di doppio vantaggio per il giocatore 1. Se, durante questo stato, *COUNT_MIN* diventasse 0, oppure *COUNT_MAX* diventasse 1, la partita terminerebbe con vincita 1. Se vince il giocatore 2, torna a V1. Infine, se vince nuovamente il giocatore 1, e *COUNT_MIN* è ancora attivo, si passa a **V1++**;
- **V1++**: ultimo stato di vantaggio possibile per 1 (causa minimo 4 partite). In questo stato, qualunque sia il risultato della manche, a patto che sia valida, vince il giocatore 1 (siccome, se si è qui, la prossima manche è la quarta, e quindi COUNT_MIN tornerà a 0);
- **V2**, **V2+** e **V2++**: stati con funzionamento simile a quelli appena descritti, con la differenza che riguardano il giocatore 2.

A termine partita, lo stato torna ad **INIZIA**, in modo da poter giocare una nuova partita senza problemi.

ARCHITETTURA DEL DATAPATH



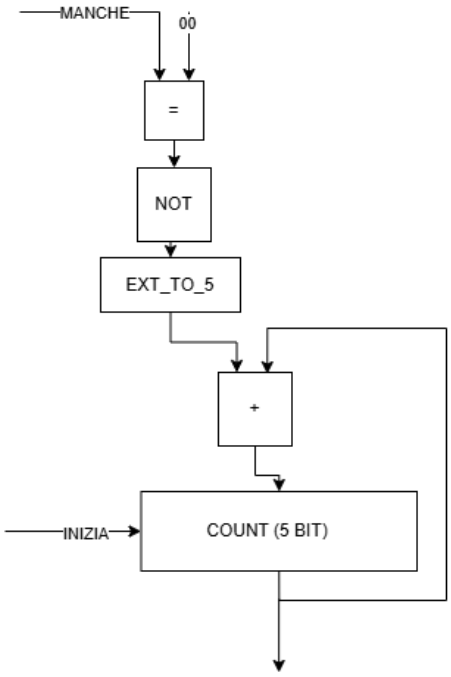
Questo Datapath utilizza i segnali inseriti come input, quindi **PRIMO**, **SECONDO** e **INIZIA**, e anche il segnale **MANCHE** che viene utilizzato come segnale di controllo. Esso, infatti, è utilizzato come selettore per salvare il vincitore della manche e la sua mossa, in modo da poter confrontare tali registri successivamente. Tutti i registri sono regolati dal segnale di **CLOCK** (non inserito nel disegno per renderlo più leggibile).

Grazie a questo circuito, il Datapath è in grado di produrre i 3 bit necessari alla FSM, quali **NO_VALID**, **COUNT_MIN** e **COUNT_MAX**.

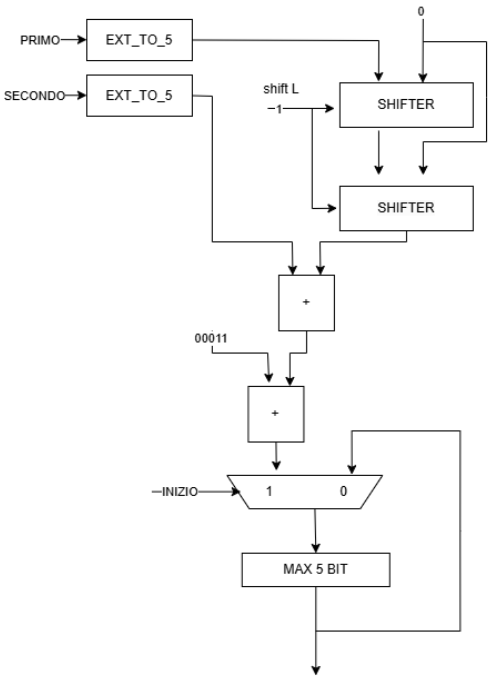
CALCOLO COUNT

Per tenere conto del numero delle manche giocate, il nostro Datapath utilizza il valore di **MANCHE**.
Esegue un confronto tra esso ed il valore ’00’, producendo quindi 1 se la manche non è valida, 0 altrimenti.
Tale valore viene quindi negato, esteso a 5 bit e poi sommato al valore del registro, prima di aggiornarlo.
Inoltre, se dovesse arrivare il segnale di reset (**INIZIA**), il registro **COUNT** viene reimpostato al suo valore inziale ’00000’.

Successivamente, per calcolare **COUNT_MIN**, viene semplicemente eseguito il seguente confronto: COUNT < 4. Tale risultato produce il segnale utilizzato per determinare se il minimo di partite è stato raggiunto o meno.



CALCOLO NUMERO MASSIMO DI MANCHE



Prende i segnali di **PRIMO** e **SECONDO**. Esegue uno shift dei bit di **PRIMO** due volte, inserendo 0 nello spazio che si crea. Successivamente, tale valore e **SECONDO** vengono sommati tra di loro, e poi aggiunto 3. Infine, questo valore viene inserito nel registro MAX (a 5 bit) **se e solo se** il segnale di **INIZIO** vale 1, altrimenti il valore di MAX rimane invariato.
NB: al massimo abbiamo deciso di aggiungere 3 anziché’ 4 siccome abbiamo voluto utilizzare il componente del maggiore uguale, avendolo già utilizzato nel circuito.

STATISTICHE PRIMA E DOPO L'OTTIMIZZAZIONE

STATISTICHE PRIMA

DATAPATH PRIMA

```
datapath         pi= 7   po= 3   nodes=103       latches=14
lits(sop)= 597  lits(fac)= 483
```

FSM PRIMA

```
fsm_nonOttimizzata      pi= 8   po= 4   nodes=  4       latches= 0
lits(sop)=   0  lits(fac)=   0  #states(STG)=   8
```

FSMD PRIMA

```
morraCinese      pi= 5   po= 4   nodes=127       latches=17
lits(sop)= 758  lits(fac)= 628
```

STATISTICHE DOPO

DATAPATH DOPO

È stato lanciato lo "*script.rugged*" una volta, ed è stato raggiunto il massimo della ottimizzazione.

```
datapath         pi= 7   po= 3   nodes= 30       latches=14
lits(sop)= 191  lits(fac)= 153
```

FSM DOPO

È stata eseguita la "*state_minimize stamina*" e "*state_assign jedi*".

```
fsm_nonOttimizzata      pi= 8   po= 4   nodes=  7       latches= 3
lits(sop)=1637  lits(fac)= 562  #states(STG)=   8
```

Successivamente, lo "*script.rugged*" tre volte per raggiungere queste statistiche.

```
fsm_nonOttimizzata      pi= 8   po= 4   nodes= 24       latches= 3
lits(sop)= 161  lits(fac)= 145  #states(STG)=   8
```

FSMD DOPO

È stato lanciato lo "*script.rugged*" due volte per terminare l'ottimizzazione.

```
morraCinese      pi= 5   po= 4   nodes= 53       latches=17
lits(sop)= 338  lits(fac)= 291
```

MAPPING TECNOLOGICO

Valori del Mapping tecnologico:

```
sis> map -m 0 -s
warning: unknown latch type at node '{[16]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[17]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[18]}' (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:       6584.00
maximum arrival time: (57.60,57.60)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-57.60,-57.60)
total neg slack:      (-743.20,-743.20)
# of failing outputs:  21
>>> before removing parallel inverters <<<
# of outputs:          21
total gate area:       6504.00
maximum arrival time: (57.60,57.60)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-57.60,-57.60)
total neg slack:      (-743.20,-743.20)
# of failing outputs:  21
# of outputs:          21
total gate area:       6248.00
maximum arrival time: (57.40,57.40)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-57.40,-57.40)
total neg slack:      (-738.60,-738.60)
# of failing outputs:  21
```

Dopo aver lanciato il comando “*map -m 0 -s*” per eseguire il mapping tecnologico sulla FSMD, si può notare che **l’area totale è pari a 6248.00 con un ritardo di 57.40**. Tale ottimizzazione è stata resa possibile grazie alla libreria “*synch.genlib*” di SIS.

SCELTE PROGETTUALI

- Al primo input per iniziare la partita, gli output sono sempre “00” e “00”;
- **La partita inizia con l’attivazione di inizia ad 1, e si resetta allo stesso modo**;
- Durante lo stato iniziale, qualunque input senza inizia attivo darà sempre come output “00” e “00”, senza cambiare di stato;
- **In caso di parità ed una manche giocabile rimanente, il vincitore di tale manche sarà anche il vincitore della partita**.
- Riguardo il design di Verilog, è stato aggiunto un bit aggiuntivo “m” per semplificare ulteriormente il codice, evitando ripetizioni di controlli *if*. Inoltre, è stato aggiunto anche il bit “aggiornaRegistriPrec” per aggiornare correttamente i registri che salvano giocatore e mossa precedente. *Questo è dovuto al fatto che, altrimenti, tale blocco always non veniva sempre eseguito nell’ordine corretto, rischiando in alcuni casi di de-sincronizzare i controlli del modulo*;
- Al termine di una partita, si torna allo stato iniziale, e per ricominciare a giocare si può inserire inizia ad 1, con il relativo numero di manche massime.