UNIVERSITÀ DEGLI STUDI DI VERONA

DIPARTIMENTO DI INFORMATICA

Elaborato SIS e Verilog

Laboratorio di Architettura degli Elaboratori

Relazione a cura di:
Rebonato Mattia - VR500232
Pellizzari Andrea - VR502128

Anno accademico 2023-2024

Sommario

1. Introduzione e specifiche

Si richiede la progettazione di un circuito sequenziale atto alla gestione di partite, tra due giocatori, del gioco *'morra cinese'*.

Il circuito ha perciò **3 ingressi**:

- **PRIMO** [2 bit]: grazie al quale si codifica la mossa scelta dal primo giocatore. (Vi è la possibilità che il giocatore 1 scelga una delle 3 mosse, oppure di giocare una mossa non valida).

- **SECONDO** [2 bit]: grazie al quale si codifica la mossa scelta dal secondo giocatore. (Vi è la possibilità che il giocatore 2 scelga una delle 3 mosse, oppure di giocare una mossa non valida)

- **INIZIA** [1 bit]: codifica la possibilità di riportare il sistema alla configurazione iniziale (per scelte progettuali può essere modificato in ogni momento della partita). Questo segnale di input agisce perciò come un vero e proprio *segnale di reset*.

Inoltre, il circuito conta di **2 uscite**:

- **MANCHE** [2 bit]: grazie al quale si codifica il risultato dell' ultima manche giocata.

- **PARTITA** [2 bit]: codifica il risultato della partita se si raggiunge la fine.

Dalle indicazioni fornite nella specifica del progetto, la partita può giungere al termine in seguito alle seguenti eventualità:

- *Vittoria del giocatore 1*.
- *Vittoria del giocatore 2*.
- *Raggiungimento del numero limite di manche* stabilito nella fase di scelta. La seguente eventualità determina un pareggio tra i due giocatori.

2. Struttura e architettura generale e scelte progettuali

La linea generale fornita nelle specifiche del progetto ci ha permesso di definire la struttura del circuito:
Il circuito è stato suddiviso in una ***FSM*** (che funge da *controllore*) e un ***Datapath*** (che svolge i compiti di *elaborazione*).
FSM e *Datapath* comunicano (definendo perciò il modello FSMD) grazie a segnali di stato (Check, Primo, Secondo) e grazie ai segnali di controllo (Stato, Manche).

Il *Datapath* riceve come input *PRIMO*, *SECONDO* e *INIZIA*. All' interno dell' *elaboratore* vengono quindi svolte le seguenti funzioni:

- Controllo del ***raggiungimento del numero minimo di partite*** (4, per definizione nelle specifiche del progetto). Nel caso in cui fossero state giocate almeno 4 partite, allora viene impostato il segnale di controllo tra FSM e Datapath *'check'* a 1, segnalando alla *FSM* tale eventualità.

- Controllo del ***raggiungimento del numero massimo di partite*** (viene deciso dagli utenti nella prima fase della partita*). Per codificare questa eventualità gli *input 'PRIMO'* e *'SECONDO'* vengono settati a 00 (tali scelte non inficiano nella corretta funzione del circuito, in quanto da tali input non dipende più nessuna scelta, in quanto si è raggiunta l' ultima manche).

- ***Controllo della validità della mossa***. Nel caso si verificasse tale eventualità, l' input *'PRIMO'* verrà impostato a 00.

- ***Controllo della correttezza della mano***.

La *FSM* non riceve nessun input dagli utenti, ed è associata alla funzione di controllo sulle seguenti situazioni:

- ***Controllo sulla fase di partita***. La *FSM* effettua controlli sulla situazione del punteggio della partita in corso, quindi controlla il vantaggio (per scelta progettuale viene preso come riferimento il vantaggio del primo giocatore).

- ***Controllo sulla vittoria della manche***.

** ('nella prima fase della partita'): a tale fase di selezione viene associato uno stato nella FSM (denominato 'Scelta').*

3. Specifica del modello FSMD

Le richieste progettuali sopracitate ci hanno spinto a definire il modello FSMD strutturato come segue.

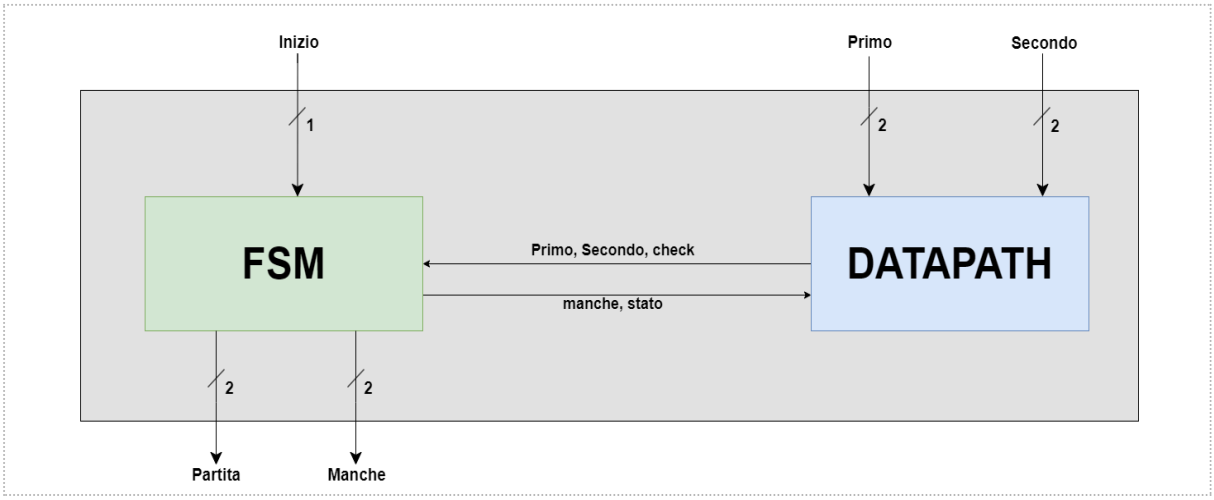

Figura 1*: Schema FSMD del circuito.*

I segnali di stato ‘check’, ‘Primo’ e ‘Secondo’ vengono inviati dal Datapath alla FSM con lo scopo di condividere tali informazioni, dato che esse sono indicative anche nella componente di controllo (FSM), in quanto sulla base di questi ingressi è possibile determinare le ‘transizioni tra gli stati’** (insieme al valore ‘Inizio’).

Il segnale di controllo ‘manche’ fornisce al datapath un indicazione sull’ esito dell’ ultima manche giocata. Il segnale ‘stato’, invece funge da indicatore per il datapath rispetto allo stato in cui si trova il sistema, in quanto lo stesso datapath non riesce a determinare in che situazione della partita ci si trova (in quanto questo compito spetta proprio alla componente di controllo (FSM)).
Tali segnali sono usati sostanzialmente come ‘segnali di selezione per demultiplexer’*** che prendono decisioni sulla memorizzazione all’ interno di registri che fanno riferimento all’ ultima mossa giocata e all’ ultimo vincitore di una manche.

Grazie ai segnali sopra-citati è possibile un interazione tra Datapath e FSM. Di seguito sono approfondite le modalità operative e di cooperazione dei due componenti del modello.

*** (‘transizioni tra gli stati’): la relazione tra i segnali di stato indicati e le transizioni tra gli stati è verificabile nella successiva sezione, dedicata alla specifica dell’ architettura del controllore.*
**** (‘segnali di selezione per demultiplexer’): l’ utilizzo dei segnali di controllo dati dall FSM come segnali di selezione per demultiplexer è definita per completo nella sezione dedicata all’ architettura del datapath.*

4. Architettura del controllore

Il controllore implementato è una macchina a stati finiti (*FSM*) di Mealy.
La *FSM* presenta 8 stati, in grado di soddisfare le richieste della specifica del progetto grazie all' utilizzo di 4 segnali di input e 3 di output.

Un circuito sequenziale modellato utilizzando una macchina a stati finiti è talvolta definito come una 6-upla M = (S, I, O, δ, λ, s) dove:

- ***S*** - ***Insieme degli stati***: '*Inizio*', '*Scelta*', '*Primo+1*', '*Primo+2*', '*Primo-1*', '*Primo-2*', '*Pareggio*', '*End*'.
- ***I*** - ***Insieme degli ingressi***: '*Primo*', '*Secondo*', '*Check*', '*Inizio*'.
- ***O*** - ***Insieme delle uscite****:* '*Partita*', '*Manche*'.
- ***δ*** - ***Funzione di stato prossimo***: definita successivamente tramite ***STT*** (*State Transition Table*) e ***STG*** *(State Transition Graph)*.
- ***λ*** - ***Funzione d'uscita*** definita successivamente tramite ***STT*** (*State Transition Table*) e ***STG*** *(State Transition Graph).*
- ***s*** - ***Stato di reset***: '*Inizio*'.

Gli 8 stati della *FSM* svolgono le seguenti funzioni:

- **Inizio**: Questo stato rappresenta la configurazione iniziale nella quale si trova il circuito una volta inizializzato. Perciò rappresenterà successivamente anche lo stato di reset, ossia lo stato a cui farà riferimento il circuito nell' eventualità che il segnale input detto *'di reset'* venga azionato.

- **Scelta**: Questo stato funge da fase iniziale di scelta, nella quale i segnali di input Primo e Secondo vengono uniti per definire il numero di manche totali da giocare nella partita. Per scelte progettuali date dalla specifica iniziale del progetto tali segnali di input verranno utilizzati inoltre come valori codificanti le mosse dei giocatori per la prima manche della partita.

- **Primo+1**: Rappresenta la situazione in cui il giocatore 1 ha un vantaggio di una manche rispetto al giocatore 2.
- **Primo+2**: Rappresenta la situazione in cui il giocatore 1 ha un vantaggio di due manche rispetto al giocatore 2.
- **Primo-1**: Rappresenta la situazione in cui il giocatore 2 ha un vantaggio di una manche rispetto al giocatore 1.
- **Primo-2**: Rappresenta la situazione in cui il giocatore 2 ha un vantaggio di due manche rispetto al giocatore 1.

- **Pareggio**: Questo stato rappresenta la situazione in cui vi è una fase di pareggio tra i due giocatori.

- **End**: Il circuito transita in questo stato nel momento in cui la partita termina (le eventualità sono molteplici:
    - Il giocatore 1 vince la partita.
    - Il giocatore 2 vince la partita.
    - Si raggiunge il numero di manche limite settato nella fase di scelta.

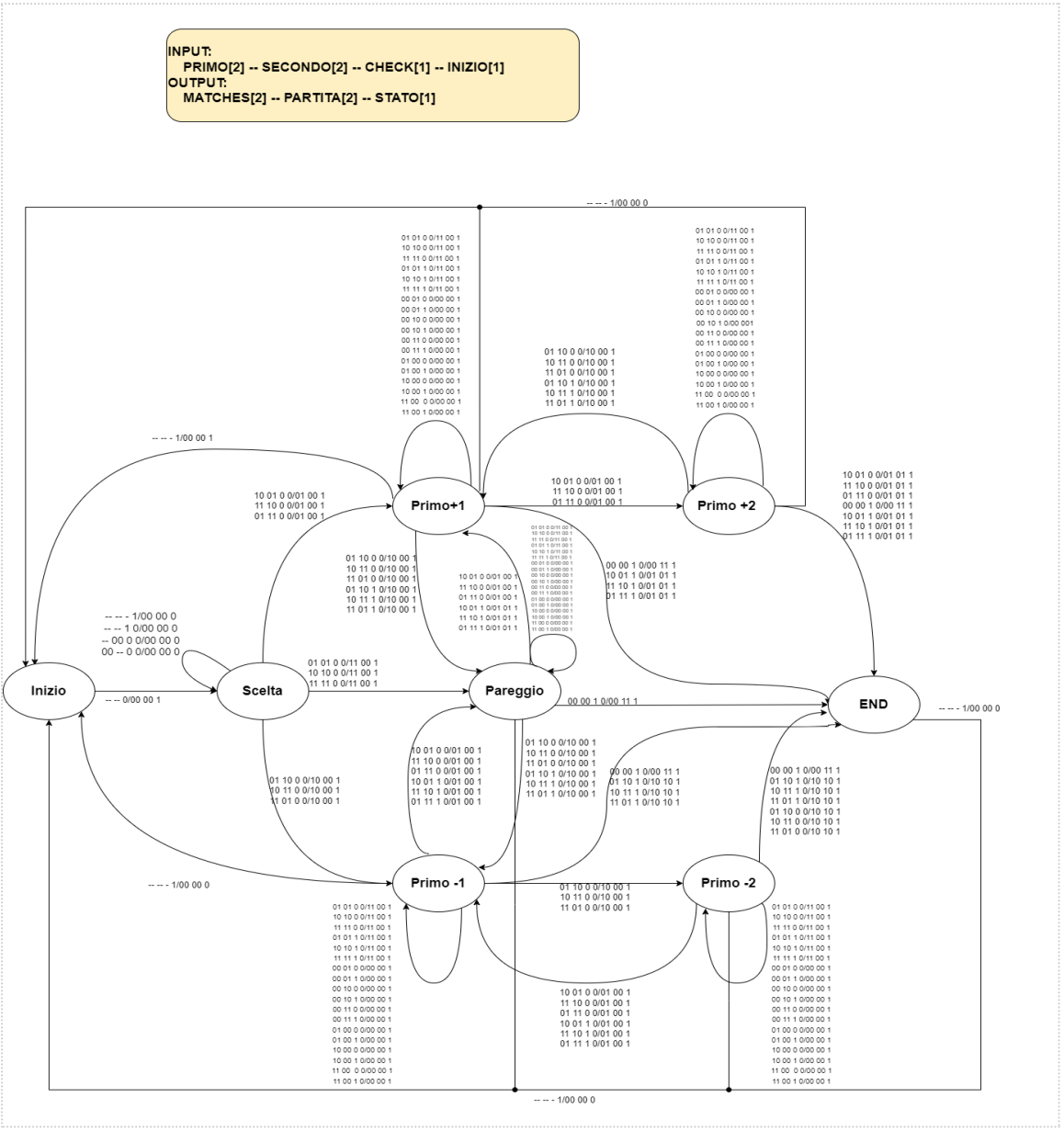

Figura 2*: STG (State Transition Graph) del controllore (FSM).*

Dal grafo ***STG*** *(State Transition Graph)* si può notare che nel momento in cui il segnale di input *'Inizio'* è 1, viene effettuata 'un' operazione di ***reset***, riportando il circuito alla configurazione iniziale, nello stato *'Inizio'*.

Di seguito è riportata la ***STT*** *(State Transition Table)* , uno strumento grazie al quale è stato possibile sviluppare in maniera più agevole il controllore nei linguaggi SIS e Verilog.

	01 01 0 0	10 10 0 0	11 11 0 0	01 01 1 0	11 11 1 0	00 01 0 0	00 01 1 0	00 10 0 0	00 10 1 0	00 11 0 0	00 11 1 0	01 00 0 0
INIZIO	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0
SCELTA	PAREGGIO/11 00 0	PAREGGIO/11 00 0	PAREGGIO/11 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0
PRIMO+1	PRIMO+1/11 00 0	PRIMO+1/11 00 0	PRIMO+1/11 00 0	PRIMO+1/11 00 0	PRIMO+1/11 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0
PRIMO+2	PRIMO+2/11 00 0	PRIMO+2/11 00 0	PRIMO+2/11 00 0	PRIMO+2/11 00 0	PRIMO+2/11 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0
PRIMO-1	PRIMO-1/11 00 0	PRIMO-1/11 00 0	PRIMO-1/11 00 0	PRIMO-1/11 00 0	PRIMO-1/11 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0
PRIMO-2	PRIMO-2/11 00 0	PRIMO-2/11 00 0	PRIMO-2/11 00 0	PRIMO-2/11 00 0	PRIMO-2/11 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0
PAREGGIO	PAREGGIO/11 00 0	PAREGGIO/11 00 0	PAREGGIO/11 00 0	PAREGGIO/11 00 0	PAREGGIO/11 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0
END												

01 00 1 0	10 00 0 0	10 00 1 0	11 00 0 0	11 00 1 0	10 01 0 0	11 10 0 0	01 11 0 0	00 00 1 0	10 01 1 0	11 10 1 0	01 11 1 0	10 11 1 0
SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0
SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0
PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+1/00 00 0	PRIMO+2/01 00 0	PRIMO+2/01 00 0	PRIMO+2/01 00 0	END/00 11 0	END/01 01 0	END/01 01 0	END/01 01 0	PAREGGIO/10 00 0
PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	PRIMO+2/00 00 0	END/01 01 0	END/01 01 0	END/01 01 0	END/00 01 0	END/01 01 0	END/01 01 0	END/01 01 0	PRIMO+1/10 00 0
PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PRIMO-1/00 00 0	PAREGGIO/01 00 0	PAREGGIO/01 00 0	PAREGGIO/01 00 0	END/00 11 0	PAREGGIO/01 00 0	PAREGGIO/01 00 0	PAREGGIO/01 00 0	END/10 10 0
PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-2/00 00 0	PRIMO-1/01 00 0	PRIMO-1/01 00 0	PRIMO-1/01 00 0	END/00 11 0	PRIMO-1/01 00 0	PRIMO-1/01 00 0	PRIMO-1/01 00 0	END/10 10 0
PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PAREGGIO/00 00 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	END/00 11 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	PRIMO+1/01 00 0	PRIMO-1/10 00 0

11 01 1 0	10 10 1 0	01 10 0 0	10 11 0 0	11 01 0 0	01 10 1 0	00 00 0 0	- - - - 1
SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	SCELTA/00 00 0	/	INIZIO/00 00 1
SCELTA/00 00 0	SCELTA/00 00 0	PRIMO-1/10 00 0	PRIMO-1/10 00 0	PRIMO-1/10 00 0	SCELTA/00 00 0	/	INIZIO/00 00 1
PAREGGIO/10 00 0	PRIMO+1/11 00 0	PAREGGIO/10 00 0	PAREGGIO/10 00 0	PAREGGIO/10 00 0	PAREGGIO/10 00 0	/	INIZIO/00 00 1
PRIMO+1/10 00 0	PRIMO+2/11 00 0	PRIMO+1/10 00 0	PRIMO+1/10 00 0	PRIMO+1/10 00 0	PRIMO+1/10 00 0	/	INIZIO/00 00 1
END/10 10 0	PRIMO-1/11 00 0	PRIMO-2/10 00 0	PRIMO-2/10 00 0	PRIMO-2/10 00 0	END/10 10 0	/	INIZIO/00 00 1
END/10 10 0	PRIMO-2/11 00 0	END/10 10 0	END/10 10 0	END/10 10 0	END/10 10 0	/	INIZIO/00 00 1
PRIMO-1/10 00 0	PAREGGIO/11 00 0	PRIMO-1/10 00 0	PRIMO-1/10 00 0	PRIMO-1/10 00 0	PRIMO-1/10 00 0	/	INIZIO/00 00 1
						/	INIZIO/00 00 1

***Figure 3, 4, 5)**: STT (State Transition Table) della FSM.*

5. Architettura del datapath

Architettura del datapath.

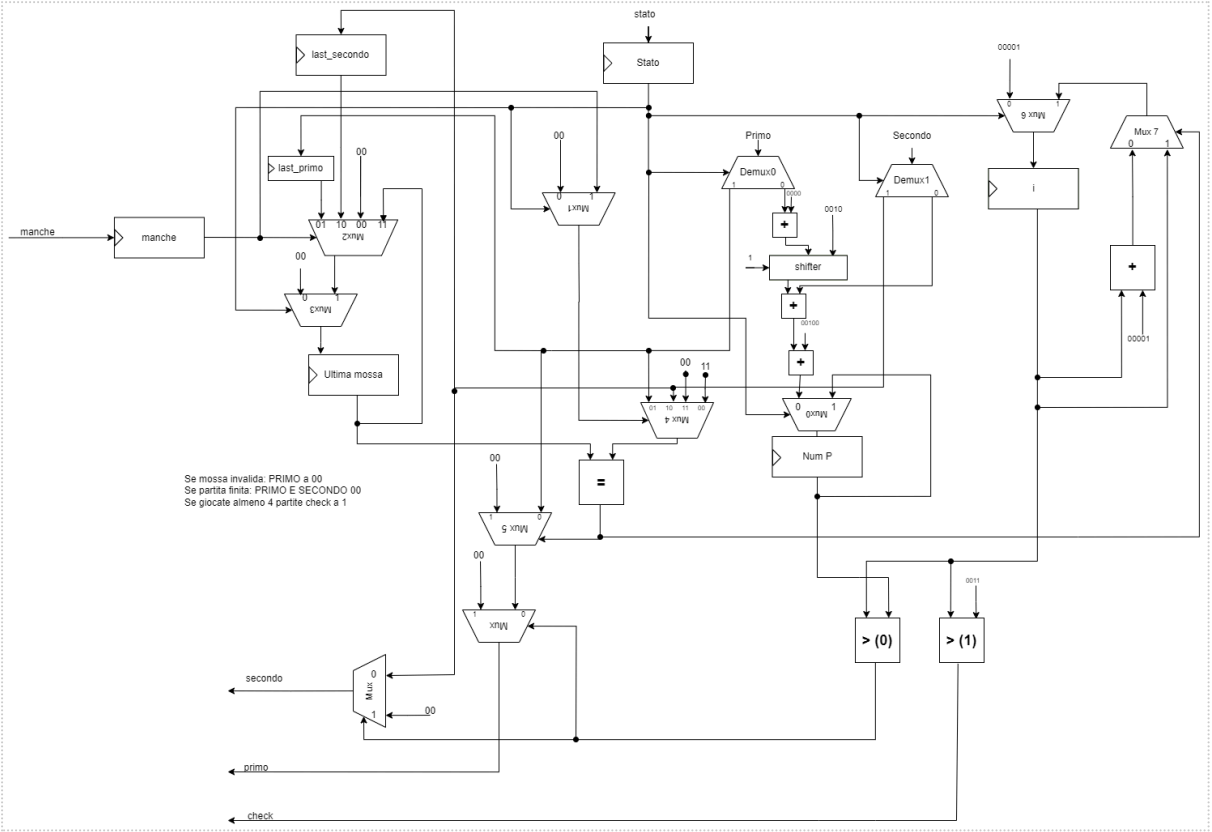

***Figura 6)**: Datapath del circuito.*

Il datapath si sviluppa come segue:

- ***Registro 'Stato'***: il registro 'Stato' prende come input il valore del segnale 'stato' (che assume il valore 0 se ci troviamo nello stato 'Inizio', altrimenti il suo valore sarà 1), input del datapath e lo memorizza. L' utilizzo di questo registro si è reso necessario dal momento in cui il valore dello stato altrimenti andrebbe perso, al passare del ciclo di clock.

- ***Demux0***: Il Demultiplexer 'Demux0' prende in input il valore del segnale di input 'Primo', e come segnale di selezione il valore del registro 'Stato'.

- ***Demux1***: Il Demultiplexer 'Demux1' prende in input il valore del segnale di input 'Secondo', e come segnale di selezione il valore del registro 'Stato'.

- ***Sommatore1***: Prende come input il valore di Primo, in caso 'Demux0' commuti l' uscita 0 e somma ad esso il valore '0000'. Esso serve a rendere il valore in ingresso a 4 bit, ed evitare eventuali durante l' implementazione nei linguaggi SIS e Verilog.

- ***Shifter1***: Prende in input il valore di output del sommatore ‘1’ ed effettua uno shift di due bit a sinistra.

- ***Sommatore2***: Prende in input il valore di output dello shifter ‘1’ e somma il valore di output del Demux1, in caso commuti l’ uscita 0.

- ***Sommatore3***: Prende in input il valore del sommatore ‘2’ e somma il valore ‘00100’.

- ***Mux0***: Multiplexer che prende in input il valore in uscita dal sommatore 3 (come ingresso codificato con il valore 0), e il valore del registro ‘Num_p’ (come ingresso codificato con il valore 1). Nel caso ci si trovi nello stato ‘Inizio’ darà in output (ossia memorizzerà nel registro ‘Num_p’) il valore dato dal Sommatore ‘3’ (questa azione rappresenta la scelta del numero di manche da giocare), altrimenti darà in output il valore di ‘Num_p’ (ossia non sovrascriverà il valore presente all’ interno del registro).

- ***Registro ‘Num_p’***: Registro a 5 bit la cui funzione è memorizzare il numero di manche da giocare.

- ***Comparatore ‘maggiore (0)’***: Comparatore a 5 bit che si occupa di controllare che “I” sia maggiore di “MaxP”, in tal caso i segnali di stato primopass e secondopass assumeranno valore 00, indicando per appunto che sono già state giocate tutte le partite massime previste e che la partita finirà in pareggio.

- ***Comparatore ‘maggiore (1)’***: Comparatore a 5 bit che si occuperà di controllare se il numero delle partite “i” è maggiore di “00011”, in tal caso il segnale di stato “check” assumerà valore 1 ed indicherà che la partita, nel caso in cui si avrà un vantaggio di 2, quest’ultima potrà terminare con la vittoria del giocatore in vantaggio di 2.

- ***Mux4***: Multiplexer che decide quale giocare ha vinto la manche precedente e instrada la mossa corretta verso il componente di uguaglianza per validare o meno la mossa.

- ***Mux1***: Multiplexer che prende in input il valore del segnale di controllo manche e ‘00’, se ci troviamo nello stato ‘Inizio’ darà in output 00, altrimenti il valore dell’ input manche.

- ***Registro ‘last_secondo’***: Registro a 2 bit che si occuperà di memorizzare la mossa attuale di secondo per il prossimo ciclo di clock, in modo da confrontarla successivamente con quella attuale di secondo.

- ***Registro ‘last_primo’***: Registro a 2 bit che si occuperà di memorizzare la mossa attuale di primo per il prossimo ciclo di clock, in modo da confrontarla successivamente con quella attuale di primo.

- ***Mux2***: Multiplexer a 4ingressi a 2 bit ognuno, si occupa di popolare il registro “Ultima mossa” decidendo che valore assegnarli, nel caso in cui l’ultima manche valida fosse fosse stata vinta da “01” passerà il valore del registro “last_primo”, nel caso in cui invece l’ultima manche valide fosse stata vinta da “10” passerà il valore del registro “last_secondo”, negli altri casi 00.

- ***Mux3****:* Decide in base allo stato se resettare o meno il registro “ultima_mossa”.

- ***Registro ‘ultima_mossa’***: Registro a 2 bit che memorizza il valore della mossa vincente null’ ultima manche valida.

- ***Comparatore di uguaglianza***: Confronta il valore del registro 'ultima_mossa' con quello del giocatore vincente nella manche. Tramite questo componente si verifica che un giocatore non vinca utilizzando la stessa mossa per più volte di fila.

- ***Sommatore4:*** Effettua la somma tra il valore nel registro 'i' e il valore '00001'.

- ***Mux7***: Multiplexer che prende in input il valore '00001' come ingresso '0' e il valore dell' output del multiplexer 7 come ingresso '1'.

- ***Mux6***: Multiplexer che prende in input il valore '00001' come ingresso '0' e il valore dell' output del multiplexer 7 come ingresso '1'.

- ***Registro 'i'***: Registro a 5 bit che memorizza il numero della manche in corso di svolgimento.

- **Mux5:** Nel caso in cui la mossa sia invalida, mette 'primopass' a 00

- ***Mux8 e Mux9***: Multiplexer che settano a '00' i valori rispettivamente di 'primopass' e 'secondopass', nel caso in cui il valore del registro 'i' abbia oltrepassato il valore del registro 'Num_p'. Questa scelta si aggiunge alle precedenti come scelta progettuale, in quanto grazie a questa scelta è possibile codificare la fine della partita per superamento del numero di partite stabilito inizialmente.

6. Implementazione in linguaggio Verilog:

L' implementazione del sistema tramite Verilog, in stile behavioral, ha portato alla seguente organizzazione dei file (all' interno della cartella VR500232_VR502128) :

- verilog/
  - **design.sv**: contiene il modello Verilog del sistema.
  - **testbench.sv**: contiene il "testbench" che verrà utilizzato per testare la correttezza del design.sv.
  - **output_verilog.txt**: risultato ottenuto lanciando il testbench, utilizzato successivamente per controllare la correttezza del progetto sis.

7. Implementazione in linguaggio SIS

L' implementazione del sistema tramite SIS ha portato alla seguente organizzazione dei file (all' interno della cartella VR500232_VR502128) :

- sis/
  - **FSMD.blif:** componente generale che permette la connessione e la comunicazione tra FSM e Datapath.
  - **testbench.script:** file di test ottenuto dalla simulazione in Verilog.
  - **output_sis.txt:** file ottenuto dopo la simulazione del progetto SIS, tramite input del file 'testbench.script', tale file di input è stato ottenuto dalla simulazione in Verilog.
  - **output_verilog.txt:** file ottenuto dal verilog, utilizzato per controllare la corretta esecuzione del sis.
  - non_ottimizzato
  - **datapath.blif:** componente che contiene la connessione tra i componenti, al fine di implementare il datapath esposto nelle sezioni precedenti.
  - **fsm.blif:** componente che svolge la funzione il controllore. A questo componente è stato applicato lo State Transition Graph, precedentemente trattato.
  - **fsm_jedi.blif:** fsm con stati assegnati attraverso il comando

    ```
    state_assign jedi
    ```

  - **demux2out2bit.blif:** componente che svolge la funzione di demultiplexer. Dispone di 2 output a 2 bit, con un segnale di selezione a 2 bit.
  - **demux2out5bit.blif:** componente che svolge la funzione di demultiplexer. Dispone di 2 output a 5 bit, con un segnale di selezione a 2 bit.
  - **comparatore2bit.blif:** componente che funge da comparatore di uguaglianza tra 2 valori a 2 bit.

- **maggiore5bit.blif:** componente che funge da comparatore di maggioranza tra 2 valori a 5 bit.

- **multiplexer2ingressi1bit.blif:** componente che svolge la funzione di multiplexer. Dispone di 2 ingressi ad 1 bit, con un segnale di selezione a 1 bit.

- **multiplexer2ingressi2bit.blif:** componente che svolge la funzione di multiplexer. Dispone di 2 ingressi a 2 bit, con un segnale di selezione a 1 bit.

- **multiplexer2ingressi5bit.blif:** componente che svolge la funzione di multiplexer. Dispone di 2 ingressi a 5 bit, con un segnale di selezione a 1 bit.

- **multiplexer4ingressi1bit.blif:** componente che funge da multiplexer. Dispone di 4 ingressi a 1 bit, con un segnale di selezione a 2 bit.

- **multiplexer4ingressi2bit.blif:** componente che funge da multiplexer. Dispone di 4 ingressi a 2 bit, con un segnale di selezione a 2 bit.

- **registro.blif:** componente che svolge la funzione di registro ad 1 bit. Verrà successivamente utilizzato come componente per i registri a più bit.

- **registro2.blif:** componente che funge da registro a 2 bit.

- **registro5.blif:** componente che funge da registro a 5 bit.

- **shifter2bitleft.blif:** componente che funge da shifter. Effettua lo shift a sinistra di 2 posizioni.

- **sommatore1bit.blif:** componente che effettua la somma tra 2 valori ad 1 bit.

- **sommatore4bit.blif:** componente che effettua la somma tra 2 valori a 4 bit.

- **sommatore5bit.blif:** componente che effettua la somma tra 2 valori a 5 bit.

- **sommatore_primo_secondo.blif**

- **uno.blif:** componente utilizzato per rappresentare il valore binario ‘1’.

- **xnor.blif:** tabella di verità “xnor”, utilizzata in alcuni componenti

- **xor.blif:** tabella di verità “xor”, utilizzata in alcuni componenti

- **zero.blif:** componente utilizzato per rappresentare il valore binario ‘0’.

8. Mapping e ritardi

Statistiche del circuito prima di effettuare l'ottimizzazione per area:

- Numero di letterali: 1546
- Numero di nodi: 106

***Figura 7**): Statistiche del circuito prima di effettuare l' ottimizzazione per area.*

Statistiche del circuito dopo aver lanciato la semplificazione *full-simplify*:

- Numero di letterali: 628
- Numero di nodi: 106

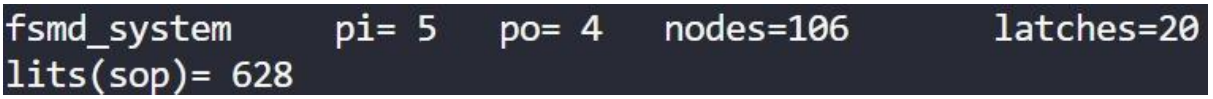

***Figura 8**): Statistiche del circuito dopo aver lanciato la semplificazione full-simplify.*

Statistiche del circuito dopo aver lanciato lo script '*script.rugged'*:

- Numero di letterali: 455
- Numero di nodi: 60

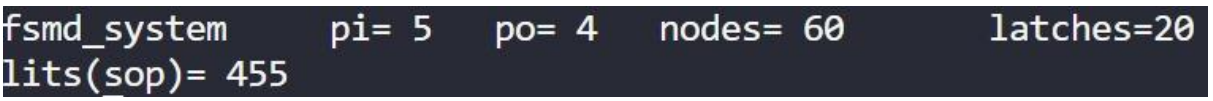

***Figura 9**): Statistiche del circuito dopo aver lanciato lo script 'script.rugged'.*

Dopo aver ottimizzato il circuito si è proceduto ad effettuare la mappatura del circuito, per verificare le statistiche riguardanti area e ritardo, una volta ottimizzato il circuito. Una volta applicata la libreria synch.genlib si ottengono le statistiche nella successiva immagine.

```
>>> before removing serial inverters <<<
# of outputs:           24
total gate area:        7800.00
maximum arrival time: (58.00,58.00)
maximum po slack:     (-4.20,-4.20)
minimum po slack:     (-58.00,-58.00)
total neg slack:      (-606.60,-606.60)
# of failing outputs:  24
>>> before removing parallel inverters <<<
# of outputs:           24
total gate area:        7768.00
maximum arrival time: (58.20,58.20)
maximum po slack:     (-4.20,-4.20)
minimum po slack:     (-58.20,-58.20)
total neg slack:      (-608.40,-608.40)
# of failing outputs:  24
# of outputs:           24
total gate area:        7096.00
maximum arrival time: (55.60,55.60)
maximum po slack:     (-4.20,-4.20)
minimum po slack:     (-55.60,-55.60)
total neg slack:      (-584.00,-584.00)
# of failing outputs:  24
```

***Figura 10**): Statistiche riguardanti area e ritardo, una volta ottimizzato il circuito.*