Elaborato SIS – Laboratorio Architettura degli Elaboratori

Corso di laurea in Informatica

di Chiara Baldo VR500878 e Dalila Portaccio VR501508

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** di **diciannove manche**. Il numero massimo di manche viene settato 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 partite**;
- 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.

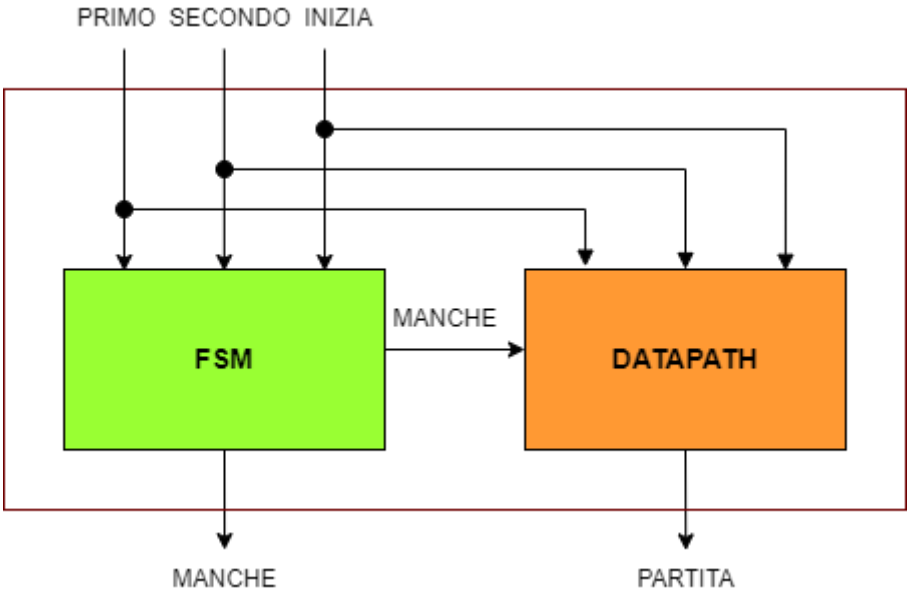
Il circuito ha tre ingressi:

- `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 partite oltre le quattro obbligatorie. Ad esempio, se si inserissero i valori `PRIMO = 00` e `SECONDO = 00`, si indicherebbe di giocare esattamente quattro partite. Se si inserisse il valore `PRIMO = 00` e `SECONDO = 01`, si indicherebbe di giocare al più 5 partite (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 partite (le 4 obbligatorie, più il valore 9 indicato da `1001`). Quando vale `0`, la partita prosegue normalmente.

Il circuito ha due uscite:

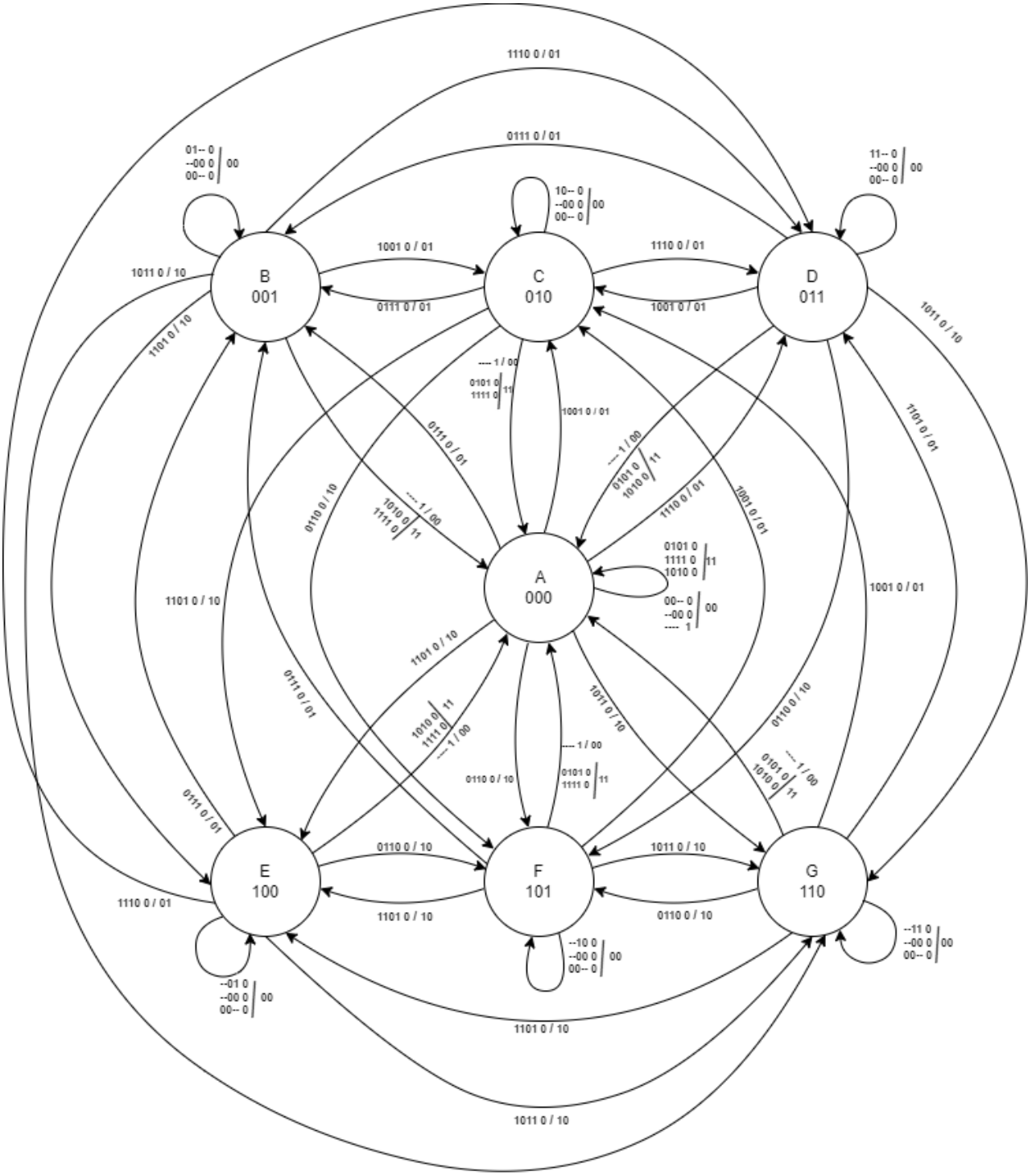
- `MANCHE [2 bit]`: fornisce il risultato dell'ultima manche giocata con la seguente codifica:
  - `00`: manche non valida;
  - `01`: manche vinta dal giocatore 1;
  - `10`: manche vinta dal giocatore 2;
  - `11`: manche pareggiata.
- **PARTITA [2 bit]**: fornisce il risultato della partita con la seguente 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



Il circuito è composto da una **FSM** che definisce i vari stati e le loro transizioni mandando come output il risultato della manche giocata al datapath e da, appunto, un **Datapath** che calcola le manche massime, aggiorna le manche giocate, e utilizza il segnale MANCHE proveniente dalla FSM per calcolare un eventuale vantaggio e l'esito di ogni partita giocata.

Diagramma degli stati (STG) della FSM



La FSM (finite state machine) riceve tre ingressi: `PRIMO` e `SECONDO`, da due bit, e `INIZIA`, da uno, e manda in uscita `MANCHE`, da due bit.

I sette stati da cui è composto determinano rispettivamente:

A: la manche è pareggiata;

B: la manche è stata vinta dal giocatore uno con sasso;

C: la manche è stata vinta dal giocatore uno con carta;

D: la manche è stata vinta dal giocatore uno con forbici;

E: la manche è stata vinta dal giocatore due con sasso;

F: la manche è stata vinta dal giocatore due con carta;

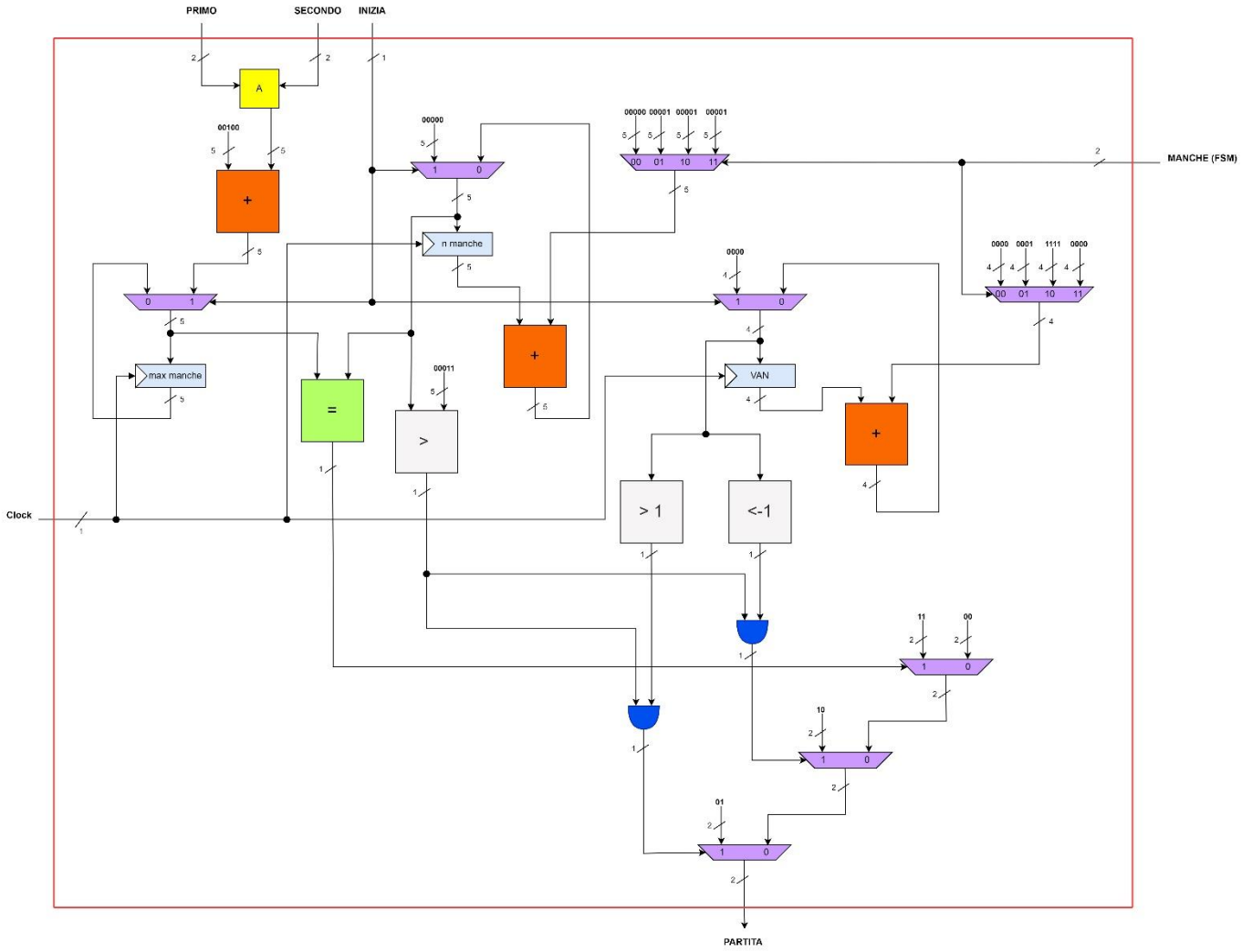
G: la manche è stata vinta dal giocatore due con forbici;

I numeri sotto alle lettere degli stati, nel grafo, rappresentano le codifiche degli stati stessi nell'FSM su Verilog.

Di seguito la State Transition Table di Mealy, per migliore leggibilità.

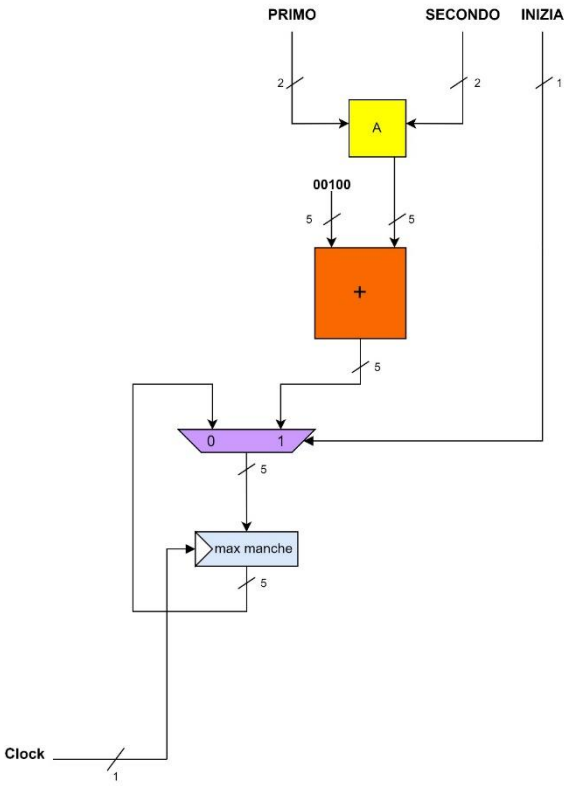
s	----1	00000	00010	00100	00110	01000	01010	01100	01110	10000	10010	10100	10110	11000	11010	11100	11110
A 000	A/00	A/00	A/00	A/00	A/00	A/00	A/11	F/10	B/01	A/00	C/01	A/11	G/10	A/00	E/10	D/01	A/11
B 001	A/00	B/00	B/00	B/00	B/00	B/00	B/00	B/00	B/00	B/00	C/01	A/11	G/10	B/00	E/10	D/01	A/11
C 010	A/00	C/00	C/00	C/00	C/00	C/00	A/11	F/10	B/01	C/00	C/00	C/00	C/00	C/00	E/10	D/01	A/11
D 011	A/00	D/00	D/00	D/00	D/00	D/00	A/11	F/10	B/01	D/00	C/01	A/11	G/10	D/00	D/00	D/00	D/00
E 100	A/00	E/00	E/00	E/00	E/00	E/00	E/00	F/10	B/01	E/00	E/00	A/11	G/10	E/00	E/00	D/01	A/11
F 101	A/00	F/00	F/00	F/00	F/00	F/00	A/11	F/00	B/01	F/00	C/01	F/00	G/10	F/00	E/10	F/00	A/11
G 110	A/00	G/00	G/00	G/00	G/00	G/00	A/11	F/10	G/00	G/00	C/01	A/11	G/00	G/00	E/10	D/01	G/00

Architettura del Datapath



Il datapath da noi progettato presenta una parte dove viene calcolato il massimo numero delle manche da giocare, una che conta il numero di manche giocate, un'altra dove viene tenuto il conto del vantaggio dei giocatori e un'ultima dove viene stabilito l'esito della partita.

Calcolo delle manche massime

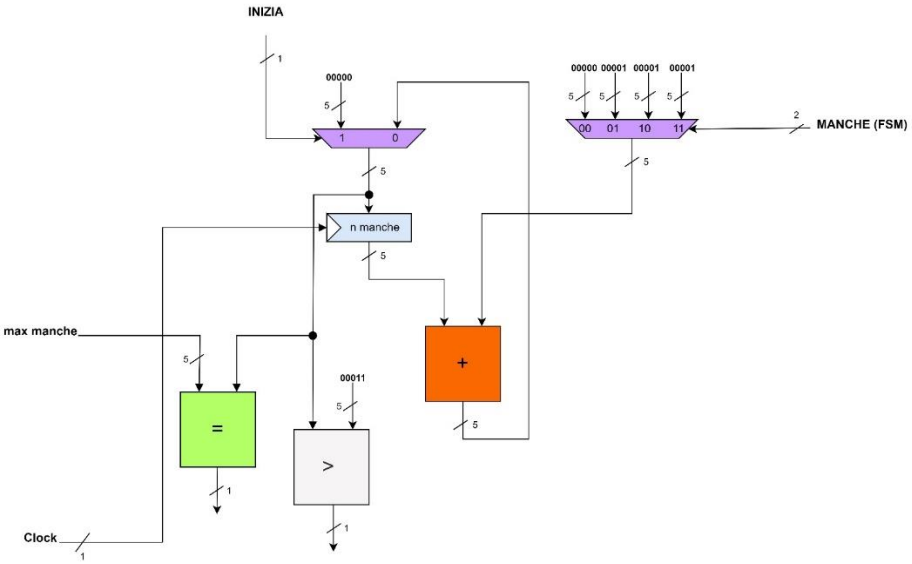


Il datapath riceve in input `INIZIA`, che varrà `1` ogni volta che il sistema verrà portato nello stato iniziale (e `0` durante il corso della partita), e `PRIMO` e `SECONDO`, che verranno "concatenati" in un unico numero con un componente da noi creato chiamato "aggregatore" e sommati a 4, determinando così il numero massimo di manche giocabili.

Il risultato verrà quindi mandato in un multiplexer (MUX) a due ingressi con selettore `INIZIA`; appena la partita ha inizio, quindi al primo ciclo di clock, `INIZIA` da come input `1` e la somma delle manche massime verrà mandata nel registro "**maxmanche**" il cui output è a sua volta collegato all'input del MUX scelto quando INIZIA è uguale a `0`.

Nei cicli di clock successivi INIZIA varrà appunto `0`, di conseguenza l'output del MUX (manche massime) non verrà aggiornato con i successivi input di PRIMO e SECONDO ma resterà del medesimo valore per tutto il corso della partita.

Calcolo delle manche giocate



Il nostro datapath riceve come input anche MANCHE, proveniente dall'FSM, che funge da selettore di un MUX a quattro ingressi collegato in output ad un sommatore cui è anche collegato il registro che tiene in memoria il numero di manche giocate. Il MUX darà come output 1 in 4 bit (`0001`) in caso di partita vinta da uno dei due giocatori (`01` o `10`) o pareggiata (`11`), invece in caso di manche non valida (`00`) l'output dato sarà 0 (`0000`) e quindi la manche non verrà conteggiata.

Il sommatore che riceve in input l'output del MUX sopracitato è collegato in output ad un altro MUX a due ingressi che ha come selettore `INIZIA`. Se `INIZIA` vale `1` il numero di manche viene azzerato, se vale `0` da come output il numero di manche sempre aggiornato.

Il numero di manche aggiornato viene memorizzato in un registro "**nmanche**" che è collegato a sua volta in output al sommatore.

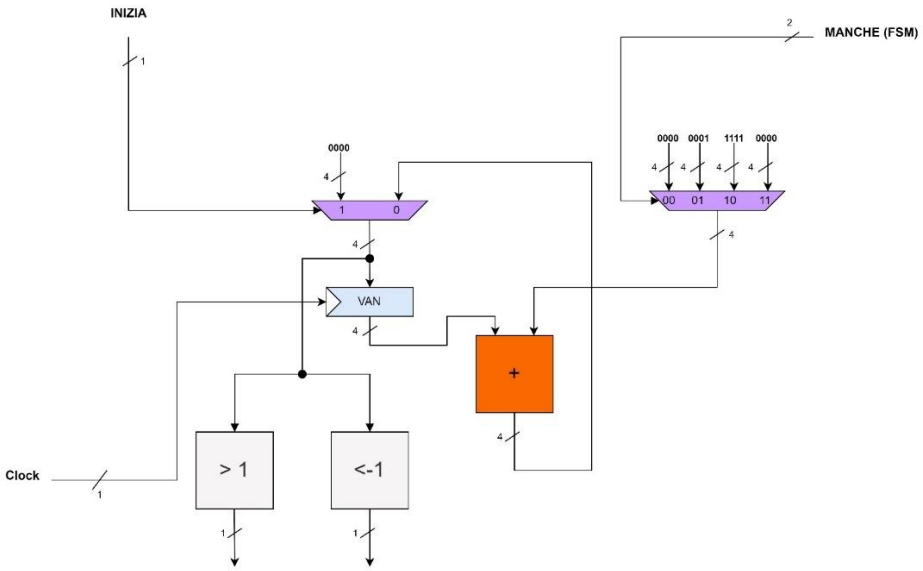
L'output del MUX che registra o aggiorna il registro nmanche è collegato anche a due operatori di confronto: uguale e maggiore.

L'operatore di confronto "**uguale**" riceve in input il valore che in ogni manche entrerà nel registro maxmanche e quello che entrerà nel registro nmanche, dando come output `1` se questi due inputs combaciano - quindi se il numero di manche massime giocate è stato raggiunto.

L'operatore di confronto "**maggiore**", invece, assicura di aver giocato almeno 4 manche, dando come output `1` se il numero di manche giocate supera il 3.

Queste ultime due componenti torneranno utili nella fase di calcolo dell'esito di `PARTITA`.

Il vantaggio



L'input `MANCHE` funge da selettore anche per un altro MUX a quattro ingressi, questa volta usato per determinare il vantaggio di uno dei due giocatori.

Se la manche giocata è pareggiata o non valida il MUX darà come output 0 a 4 bit (`0000`), se viene vinta dal giocatore uno darà 1 (`0001`) e darà -1 (`1111`) se vinta dal giocatore due.

L'output viene mandato ad un sommatore che è collegato in output ad un altro MUX a due ingressi che ha come selettore INIZIA. Se INIZIA vale `1`, dunque all'inizio di ogni partita, il vantaggio viene azzerato, se vale `0` da come output il valore del vantaggio aggiornato.

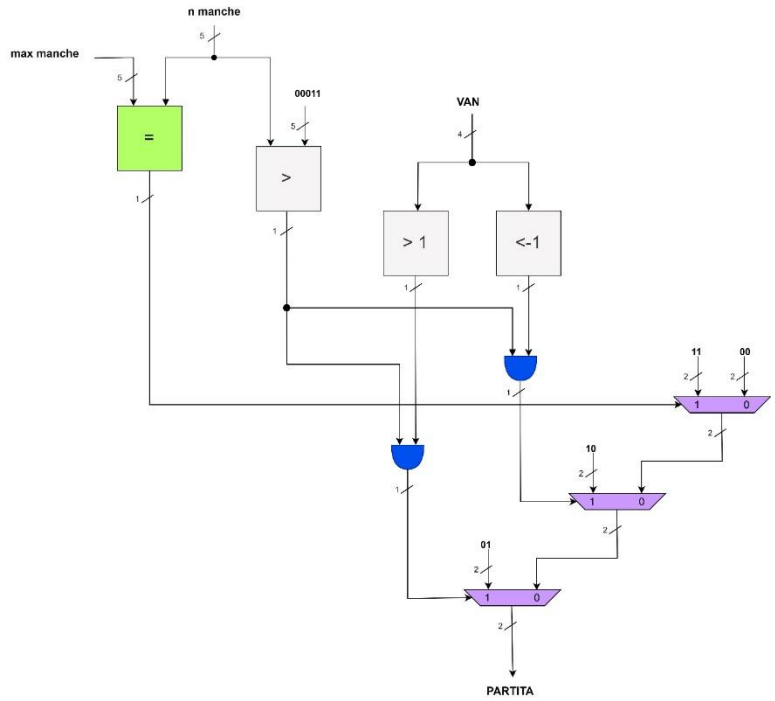
Il valore del vantaggio viene poi salvato nel registro "**VAN**" che è collegato in output al sommatore, che dunque aggiunge uno, toglie uno o lascia invariato il vantaggio in base all'esito della manche.

Se il valore del vantaggio è positivo significa che il giocatore 1 è in vantaggio, viceversa se è negativo è il giocatore 2 ad esserlo.

L'output del MUX a due ingressi che ad ogni manche aggiorna il registro VAN è collegato anche a due operatori di confronto: maggiore di uno ">1" e minore di meno "<-1", che serviranno per determinare se la partita è stata vinta da uno dei due giocatori.

In questo modo se uno dei due giocatori è in vantaggio di due, uno dei due operatori darà come output **1**, e questo implicherà poi, assieme ad altre condizioni, il termine della partita.

Esito della partita



Lo stato della partita viene gestito da due porte AND e tre MUX a due ingressi.

Il primo MUX (quello più in alto nel disegno del circuito) ha come selettore l'operatore di confronto uguale che da come output **1** quando il numero di manche giocate e quello delle manche massime corrispondono. Esso darà come output 11 (dunque che la partita termini in pareggio) se sono state giocate tutte le manche possibili, in caso contrario darà 00 (che la partita continui).

Il secondo MUX ha invece come selettore l'output di una porta AND che a sua volta riceve come inputs il valore dell'operatore "<-1" che da **1** se il giocatore 2 è in vantaggio di 2 e il valore dell'operatore ">" che da **1** se sono state svolte almeno 4 manche. Quindi se il giocatore due è in vantaggio di 2 e sono state fatte più di tre manche il MUX riceverà come selettore il valore di **1** e di conseguenza darà come output 10 (che la partita termini con la vittoria del giocatore due), in caso contrario darà come output lo stesso del MUX precedente.

Il terzo MUX funziona in modo analogo al secondo, ma determina un'eventuale vittoria del giocatore 1. Il selettore corrisponde quindi all'output di una porta AND che darà **1** se il giocatore uno è in vantaggio di due e sono state giocate almeno 4 partite, e in questo caso il MUX darà come output 01 (che la partita termini con la vittoria del giocatore uno), mentre in caso contrario l'output sarà lo stesso del MUX precedente.

L'output di quest'ultimo MUX è l'esito di `PARTITA`.

Statistiche del circuito prima e dopo l'ottimizzazione per area

Dopo aver eseguito sull'FSM (FSM.blif) la codifica automatica degli stati con *state_assign jedi*,

```
FSM             pi= 5   po= 2   nodes=  5       latches= 3
lits(sop)= 369  #states(STG)=   7
```

e dopo aver minimizzato il circuito della FSM con *source script.rugged*, opportunamente acclusa all'interno della directory di lavoro,

```
FSM             pi= 5   po= 2   nodes= 12       latches= 3
lits(sop)= 122  #states(STG)=   7
```

abbiamo "salvato" la nuova FSM in un nuovo file blif con il comando *write_blif FSMassegnata.blif*.

Abbiamo dunque collegato all'intero circuito dell'FSMD la nuova FSMassegnata (e minimizzata), rendendo il circuito funzionante; di seguito le statistiche pre-ottimizzazione.

```
FSMD            pi= 5   po= 4   nodes=137       latches=17
lits(sop)= 991
```

Abbiamo dunque eseguito l'ottimizzazione nuovamente con *source script.rugged*, fino ad arrivare a queste statistiche:

```
FSMD            pi= 5   po= 4   nodes= 45       latches=17
lits(sop)= 357
```

L'ottimizzazione è stata effettuata eseguendo una sola volta sul circuito, in quanto se il comando veniva eseguito più di una volta le statistiche del circuito iniziano ad aumentare invece che diminuire.

Numero di gate e ritardo

L'ultimo passaggio è stato dunque la mappatura su libreria tecnologica, che permette di visualizzare le statistiche di area e ritardo dell'intero circuito.

Dopo aver accluso la libreria synch.genlib all'interno della medesima directory dei file blif, l'abbiamo letta con il comando *read-library synch.genlib* e abbiamo eseguito l'effettiva mappatura ottimizzando il circuito per area con il comando *map -m 0*, per poi visualizzare le seguenti statistiche di circuito con il comando *map -s*:

```
>>> before removing serial inverters <<<
# of outputs:          21
total gate area:       5936.00
maximum arrival time: (49.80,49.80)
maximum po slack:     (-5.00,-5.00)
minimum po slack:     (-49.80,-49.80)
total neg slack:      (-596.40,-596.40)
# of failing outputs:  21
>>> before removing parallel inverters <<<
# of outputs:          21
total gate area:       5904.00
maximum arrival time: (48.60,48.60)
maximum po slack:     (-5.00,-5.00)
minimum po slack:     (-48.60,-48.60)
total neg slack:      (-594.00,-594.00)
# of failing outputs:  21
# of outputs:          21
total gate area:       5760.00
maximum arrival time: (47.60,47.60)
maximum po slack:     (-5.00,-5.00)
minimum po slack:     (-47.60,-47.60)
total neg slack:      (-581.80,-581.80)
# of failing outputs:  21
```

Scelte progettuali effettuate

Struttura generale del circuito:

- Abbiamo deciso di utilizzare la FSM come un'effettiva parte di "elaborazione" del circuito, piuttosto che come "controllore". Il motivo di questa scelta è dovuto al fatto che, progettando una FSM che cambia di stato in base alle mosse praticate dai giocatori (e dunque in base al vincitore della singola manche), è stato molto facile classificare come "non valida" la ripetizione della mossa vittoriosa del round precedente.
  In particolare, avendo creato uno stato di pareggio e uno stato di vittoria con ogni mossa per ogni giocatore, se per esempio il giocatore uno dopo aver vinto con sasso volesse ri-effettuare la mossa, per "impedirglielo" è sufficiente creare una transizione che riconduce allo stato "vittoria di G1 con sasso" che viene intrapresa quando l'input è 01-- e che restituisce 00 (mossa non valida).
  La seconda fase di elaborazione avviene dunque all'interno del datapath, che prende in input l'uscita MANCHE della FSM e la utilizza fino all'emissione di PARTITA.

- Nel caso almeno uno dei due giocatori decidesse di non giocare nessuna mossa (quindi mandasse in input `00`) abbiamo deciso di contare la manche come non valida, di conseguenza non verrà né conteggiata né saranno fatte transizioni di stato. Lo stesso accade qualora il giocatore che ha vinto la manche precedente scegliesse di rigiocarla subito dopo.

- Nel caso il registro nmanche raggiunga il registro maxmanche e qualora nessuno dei due giocatori abbia raggiunto un vantaggio di almeno due manche in quell'ultima manche, la partita terminerà in pareggio a prescindere dal vantaggio dei giocatori, se esso non è sufficiente a determinare una vittoria.

SIS:

- Il complemento "aggregatore" è stato creato da zero per unire due input da due bit in un solo input da cinque bit (cinque perché poi, sommate altre quattro manche, maxmanche può arrivare a diciannove, quantità cui sono necessari cinque bit per essere rappresentata); per farlo è stato necessario indicare che il bit più significativo sarebbe sempre stato zero; poi da quel bit, a seguire, ciascun bit sarebbe stato uno se il bit corrispondente di PRIMO o SECONDO fosse stato uno a sua volta.

- All'interno della sezione "Vantaggio" del datapath, il registro, il sommatore, il multiplexer e i due operatori di confronto "maggiore di uno" e "minore di meno uno" sono stati creati tenendo conto che potrebbero assumere valori negativi, e quindi utilizzando il complemento a due. Per facilitare l'operazione, non riuscendo a trovare un algoritmo che regolasse il funzionamento del complemento a due, abbiamo creato dei componenti ad hoc che emettono l'output solo nei casi contemplati dal funzionamento della morra cinese: ad, esempio, non potendo il registro VAN superare il valore quattro né il valore meno quattro, il "maggiore di uno" e il "minore di meno uno" non sono programmati per sapere quali output emettere quando l'input è superiore a quattro o

inferiore a meno quattro. Analogamente sono progettate le altre componenti che devono funzionare in complemento a due.

- Qualora il circuito, all'avvio, venisse simulato senza settare da principio INIZIA uguale a 1, la macchina fornirà correttamente l'output MANCHE ma emetterà sempre PARTITA come 00, in quanto il massimo delle manche viene deciso solo quando INIZIA è 1.

Verilog:

- Per praticità, il registro vantaggio è stato sostituito da un integer che svolge la medesima funzione.

- Su Verilog non è presente un segnale vero e proprio di reset, in quanto è l'input INIZIA a comportarsi da tale, ma è presente, ovviamente, un segnale di clock.

- Il circuito è composto da tre blocchi "always": il primo aggiorna le transizioni di stato, da stato presente a stato prossimo, e viene eseguito ogni volta che il clock si trova in fase di salita; il secondo regola il comportamento della FSM, e viene eseguito ogni volta che cambiano i valori di PRIMO, SECONDO e INIZIA (ovvero gli input della FSM); il terzo regola il comportamento del DATAPATH e viene eseguito ad ogni rising edge del clock.

Testbench:

Abbiamo progettato un testbench che simula l'andamento di tre partite.

1. La prima partita inizia con i valori di PRIMO e SECONDO rispettivamente 10 e 01, dunque, sommatogli quattro, abbiamo che il massimo di manche per questa partita sarà tredici.
   La partita comincia con una vittoria del giocatore uno e con un pareggio, per poi interrompersi improvvisamente con l'arrivo dell'input INIZIA=1, a dimostrazione del fatto che volta che ciò accade la partita viene completamente resettata.

2. La seconda partita ha un massimo di cinque manche, poiché quando INIZIA vale 1, PRIMO e SECONDO valgono rispettivamente 00 e 01.
   La partita comincia con un input non valido (00) da parte del giocatore uno, dunque la manche non viene conteggiata, e prosegue con due regolari vittorie da parte dello stesso giocatore; si può tuttavia constatare che la partita prosegue, poiché pur avendo raggiunto il vantaggio necessario a vincere ci troviamo solamente alla manche numero due. A questo punto il giocatore due vince una manche e subito dopo gioca una mossa non valida: subito dopo aver vinto con carta rigioca la mossa, perciò si resta nello stesso stato e il numero di manche non aumenta. Una manche in pareggio e infine, nella manche numero cinque (l'ultima possibile per una vittoria), il giocatore uno vince, riportando il vantaggio a due e dunque vincendo l'intera partita.

3. La terza partita avrà un massimo di quattro manche, in quanto sia PRIMO che SECONDO hanno valore 00 quando inizia vale 1.

La partita comincia con un pareggio e poi una vittoria a testa per ciascun giocatore; a questo punto si resta nella manche tre per altre due mosse, in quanto prima il giocatore uno gioca 00 e nella manche successiva il giocatore due gioca la mossa con cui aveva vinto la “prima” terza manche (forbice). Infine, entrambi i giocatori giocano sasso, mossa di pareggio, e la partita termina senza alcun vincitore, essendo VAN=0.