Progetto d'esame - Battaglia navale

Carlo Vassallo - matricola: 892613

Architettura degli elaboratori I - anno 2016/17

Specifiche del progetto

Il progetto ha come obbiettivo quello di simulare, in logisim, un circuito che permetta a due utenti di giocare a battaglia navale.

Input/output:

- **Display:** ogni giocatore ha a disposizione un display che mostra una griglia 4x6 ove può visualizzare lo stato delle proprie navi.
- **7segment Display:** due display a 7 segmenti che permettono ai due giocatori di visualizzare il proprio numero di navi rimanenti.
- **Keypad:** due tastiere composte da 11 tasti, contenenti le lettere dalla 'A' alla 'D', i numeri da 1 a 5, e un tasto 'attacca'. Attraverso questi ogni giocatore puo selezionare le coordinate dove posizionare le navi e/o attaccare l'avversario. La selezione viene confermata mediante il tasto 'attacca'.
- **New Game Button:** un bottone che permette di iniziare una nuava partita.
- **Led di stato:**: al lato di ciascun display è posto un led, esso puo assumere due colori:
  - *verde:* se deteniamo il turno di gioco
  - *rosso:* se l'avversario detiene il turno

Fasi di gioco:

1. **Preparazione alla battaglia:** in questa fase i giocatori posizioneranno le navi a disposizione (per un totale di 6 caselle riempite) selezionando la posizione di destinazione e premendo invio per ogni casella da occupare. All'avvio della partita il primo giocatore posizionera le navi. Il suo turno in questa fase terminerà automaticamente al posizionamento della 6° nave. Quando anche il 2 giocatore avrà posizionato l'ultima nave il gioco passerà alla 2° fase.
2. **Battaglia:** durante questo stadio i due giocatori, a turno, selezioneranno meidante il keypad le coordinate ove sferrare il proprio attacco. Quest'ultimo avrà effetto sulla griglia dell'avversario che potrà visulaizzare sul proprio display se è stato colpito o meno. Il turno viene autamaticamente passato dopo ogni attacco. La fase di battaglia termina quando uno dei due giocatori affonda tutte le navi dell'avversario.
3. **Fine del gioco:** uno dei giocatori ha vinto e la partita viene automaticamente resettata.

Memorizzazione:

- **Griglie dei giocatori:** ciascun campo di gioco è salvato in memoria. Ogni coordinata puo assumere 3 divesi valori:
  - *00* in presenza di acqua
  - *01* in presenza di un segmento di nave integro
  - *10* in presenza di un segmento di nave colpita
- **Stato:** Lo stato del gioco è rapprensentato su 3 bit, ciscuno con un proprio sognificato:
  - *1° bit:* rappresenta la fase di gioco; 0 preparazione alla battaglia, 1 battaglia.
  - *2° bit:* turno del giocatore; 0 per il primo giocatore, 1 per il secondo.
  - *3° bit:* stato della mossa, 0 se in attesa, 1 se è stata sferrata. Questo bit è fondamentale, infatti su di esso si basa il principio di progettazione.

Principio di progettazine:

Ogni componente è stato sviluppato in funzione dello stato, piu in particolare sullo stato della mossa, infatti esso definisce come e quando i componenti devono agire. Durante lo stato di 'mossa in attesa' essi si limitano a fornire in outut il valore ricevuto in input. Nel momento in cui il giocatore effettua la propria mossa, e quindi lo stato passa in stato di 'mossa sferrata', i componenti forniranno il valore corrispondente all'effetto della mossa.

Componenti

Ogni componente di questo progetto, fatta eccezione per le porte logiche, led e display a 7 segmenti, è stato realizzato personalmente.

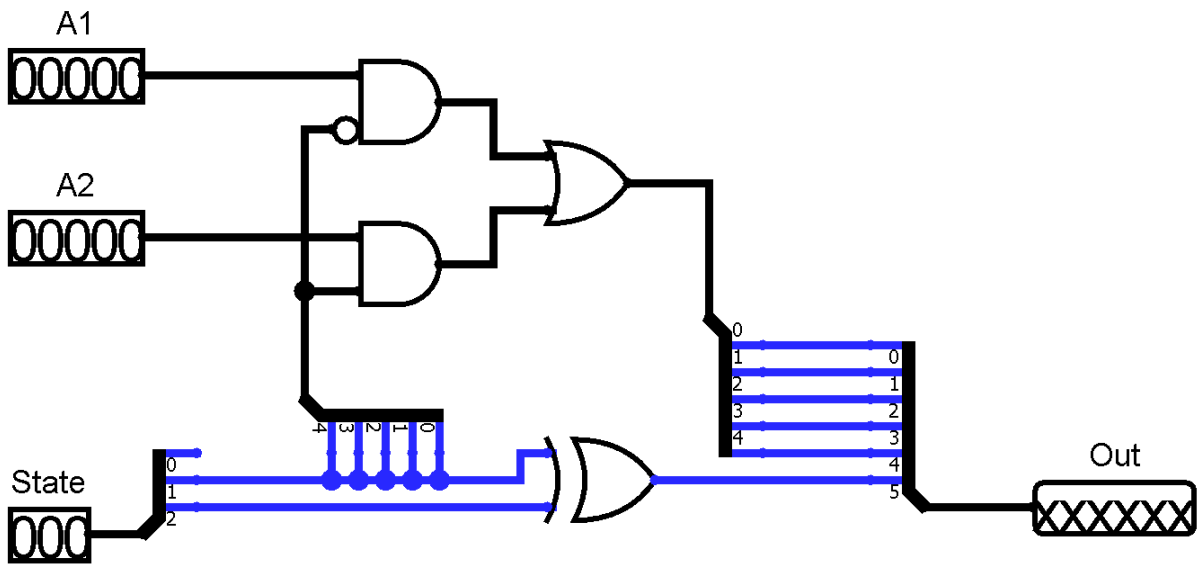
Componenti base:

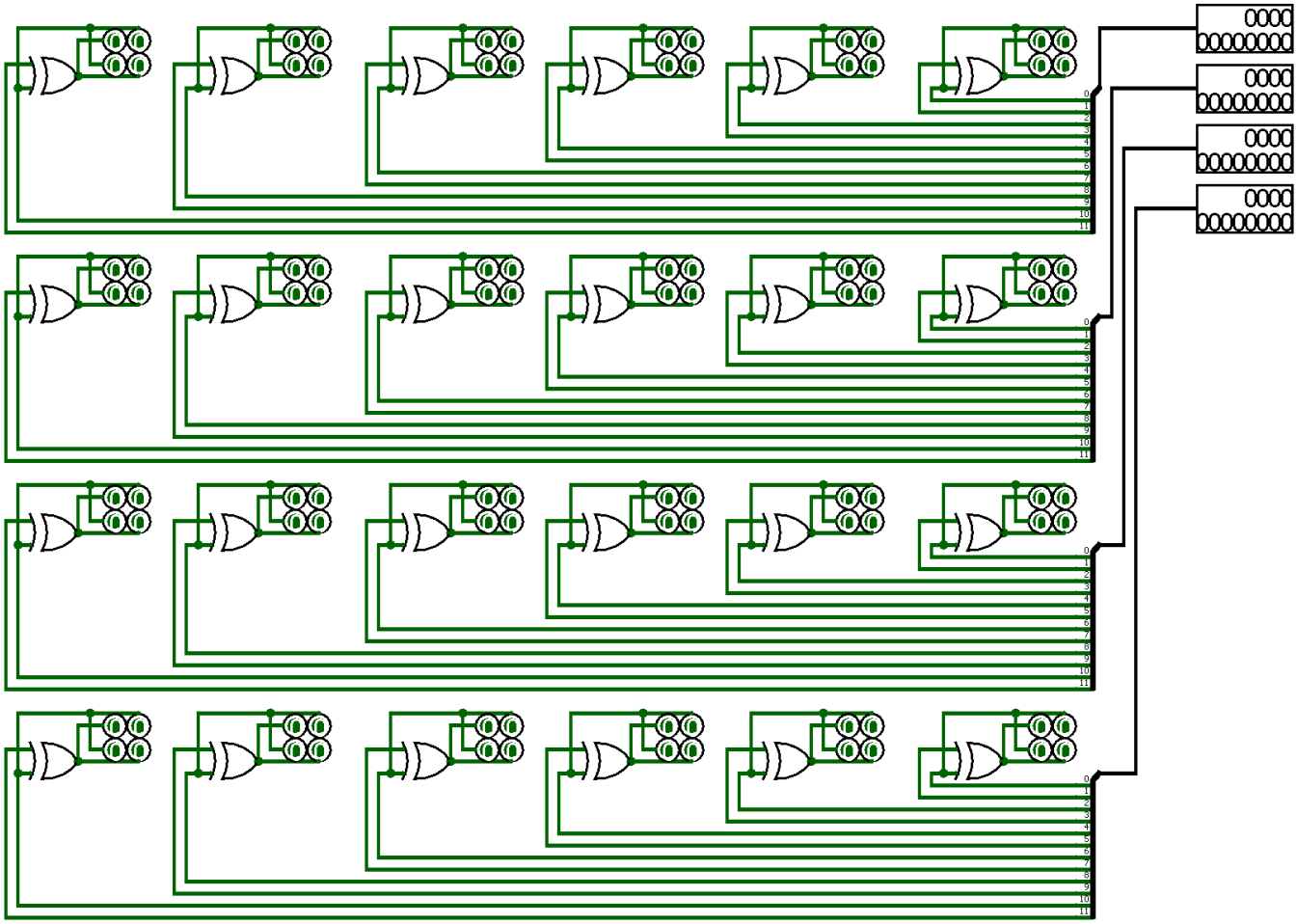
- **Flip-Flop D**
- **Decoder** a 6 bit
- **Multiplexer** da 64 vie da 2 bit
- **Adder** a 4 bit

Componenti:

- **Display** (x2)
- **Memoria**
- **Keypad** (x2)
- **StateManager**
- **ShipCountManager**
- **AttackProcessor**
- **AttackKeeper**

Display





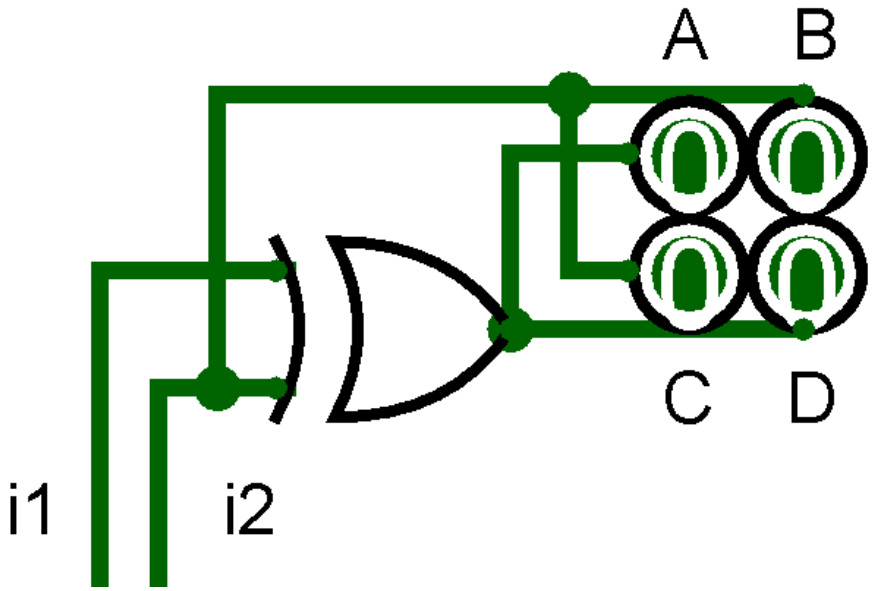
Questo componente ha il compito di mostrare ai giocatori lo stato delle proprie navi ed è composto da:

- **96 led:** in gruppi da quatto che rappresentano le celle della griglia di gioco. Ogni cella puo assumere la colorazione interamente blu, se contiene acqua, interamente grigia se contiene un segmento di nave, metà grigia e metà blu a formare una croce se rappresenta un segmento di nave che è stato colpito. Per ottenere 2 colori è stato settato come colore del led spento il colore celeste e grigio per il led acceso. La tensione su ogni led è gestita dal driver del display.
- **Display Driver:** questo componente, che possiamo immaginare come lo chassis del display, ha il compito di accendere a dovere ogni led in modo da rispecchiare la griglia di gioco salvata in memoria. Questo driver ha 4 input da 12bit ciascuno, ovvero la rappresentazione binaria di ogni riga della griglia di gioco. In oltre presenta 96 pin di output posizionati a dovere in modo da potervici posizionare sopra i led, immaginando che siano saldati sopra. Internamente questo componente si limita a leggere dagli input il valore di ogni cella (ovvero ogni gruppo di 4 led) e accendere a dovere i led. Come da specifiche in memoria ogni cella è rappresentata su 2 bit come:
  - *00* acqua
  - *01* nave
  - *10* nave colpita

  Per fare ciò è stato creato un circuito cosi composto:

i1	i2		A	B	C	D
0	0		0	0	0	0
0	1		1	1	1	1
1	0		1	0	0	1
1	1		x	x	x	x

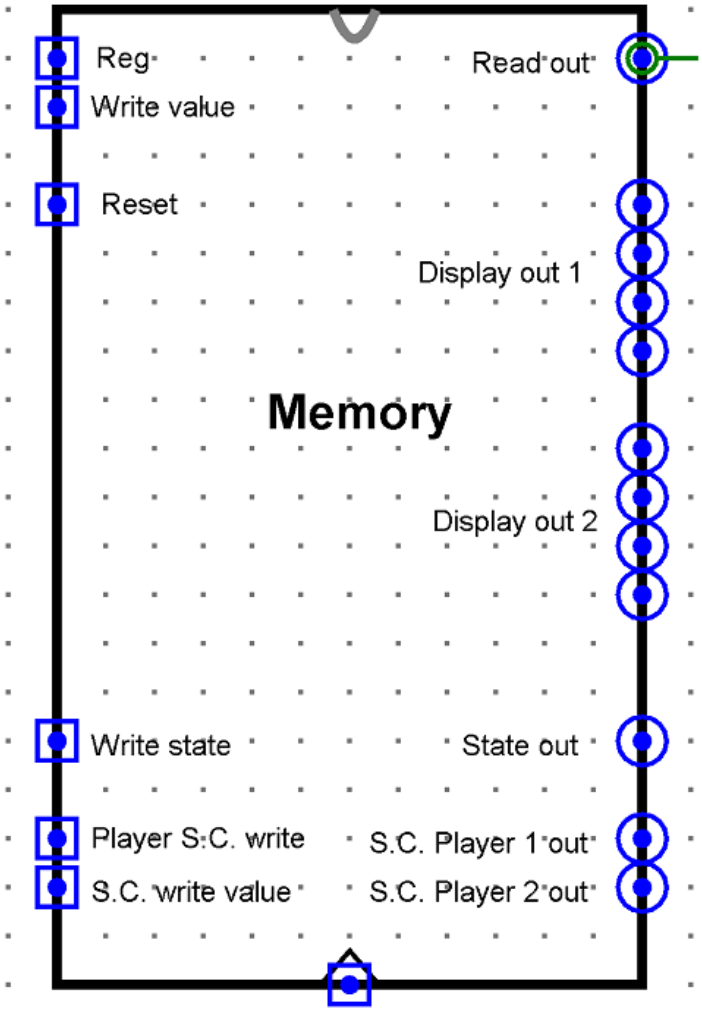
- B = C = i2
- A = D = (!i1 AND i1) OR (i1 AND !i2) = i1 XOR i2

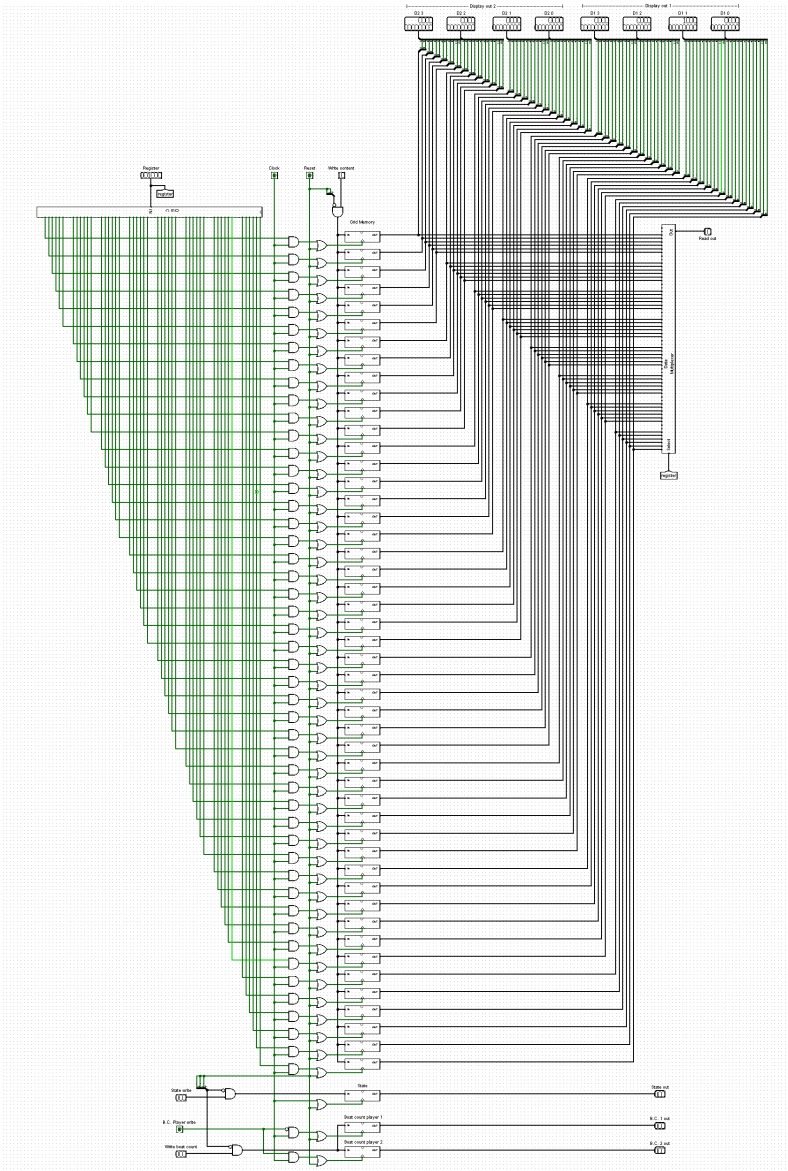


Considerazione

Durante la progettazione del display ho incontrate delle difficolta puramente strumentali. Infatti logism non mette a disposizione led che possano assumere più di 2 colori. Dopo delle ricerche ho trovato una soluzione che prevedeva la sovrapposizione di led semi trasparenti in modo da sovrappore più colori e attraverso le possibili combinazioni formare svariati colorazioni. Tuttavia logism non permette la sovrapposizione di led, infatti la soluzione sopracitata prevedeva lo sfruttamento di quello che sembra essere un bug. Dovendo scartare questa opzione ho pensato di utilizzare quattro led per ogni cella come sopra descritto. A questo punto collegare i led direttamente alla non avrebbe reso la sensazione di un display in quanto la distanza fra le celle sarebbe stata ampia, motivo per cui ho creato il driver con gli input sulla parte superiore del componente, invece che sui lati come è solito fare.

Memory


Reg
Write value
Reset
Write state
Player S.C. write
S.C. write value
Read out
Display out 1
Memory
Display out 2
State out
S.C. Player 1 out
S.C. Player 2 out



La memoria ha il compito di memorizzare i dati del gioco, quali le due griglie dei giocatori, il numero delle navi posizionate e/o rimanenti, e lo stato della partita. La struttura del circuito è molto simile ad un register file di una cpu. Memorizzazione:

- **Griglie dei giocatori:** ogniuna delle due griglie è salvata in una memoria indirizzabile composta da 48 celle da 2 bit ciascuna. Gli indirizzi sono da 6bit di cui:
  - *1° bit* rappresenta il giocatore; 0 per il 1°, 1 per il 2°
  - *2°, 3° bits* raprresentano la riga sulla griglia
  - *4°, 5°, 6° bits* raprresentano la collonna sulla griglia.

  Ogni cella puo contentere:

  - *00* in presenza di acqua
  - *01* in presenza di un segmento di nave integro
  - *10* in presenza di un segmento di nave colpita
- **Stato:** lo stato del gioco, rapresentato da un led è salvato in una memoria da 3 bits con il seguente formato:
  - *1° bit* rappresenta la fase di gioco; 0 posizionamento delle navi, 1 partita.
  - *2° bit* turno del giocatore; 0 per il primo giocatore, 1 per il secondo.
  - *3° bit* stato della mossa, 0 se in attesa, 1 se è stata inviata.
- **Contatore Navi:** questi contatori, uno per giocatore, hanno un molteplice utilizzo: quello di tenere traccia delle sezioni di nave che sono state posizionate dai giocatori durante la fase di preparazione, e quello di tenere traccia delle sezioni di nave ancora integre per giocatore. Questo ci permette di reagire al completamento della fase di preparazione e alla vitoria di un giocatore, nonche di visualizzare lo score di ogni giocatore.

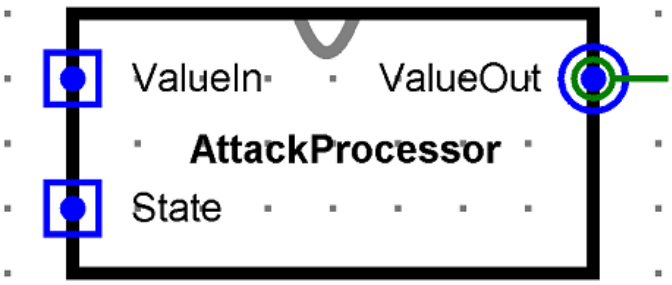
- **Griglie di gioco:** le due griglie di gioco sono memorizzate in una struttura ispirata al register file; infatti la scrittura avviene specificando l'indirizzo di registro e il suo valore e analogamente la lettura avviene specificando il registro da leggere. Per fare cio sono stati creati 64 registri da 2 bit, 1 decoder da 64 vie a 2 bit e 1 multiplexer da 64 vie a 2 bit. Il piano di indirizzamento e cosi formato: il 1° bit, più significativi, indica il giocatore, il 2° e 3° indicano la riga sulla griglia, il 4°, 5°, 6° indicano la colonna sulla griglia. Essendo 6 le righe e 3 i bit utilizzati per rappresentarle, per ogni riga rimangono 2 indirizzi inutilizzati per un totale di 16 indirizzi superflui che non sono stati collegai a nessun registro. I dati qui memorizzati devono essere rappresentati sul display, motivo per cui sono state create delle uscite dedicate.
- **Stato della partita:** lo stato della partita è memorizzato in un registro da 3 bit appositamente creato, esso è accessibile in scrittura e lettura attraverso un ingresso e un uscita dedicate.
- **Cotatore navi:** Il contatore delle navi di ogni giocatore è memorizzato in un registro dedicato a 3 bit. Essi sono accessibili in scrittura, attraverso un ingresso di selezione per specificare su quale giocatore agire e un ingresso ove immettere il nuovo valore da memorizzare. Per le operazioni di lettura invece sono presenti 2 uscite dedicate per ogni contatore.

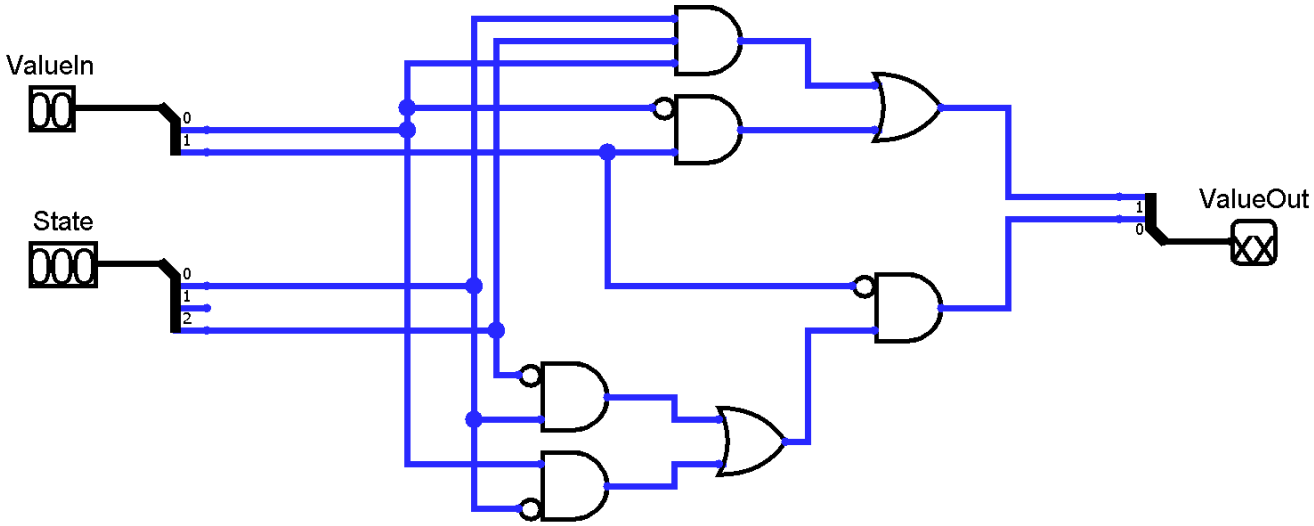
In oltre la memoria ha un input di reset, che setta a 0 ogni registro in modo da poter iniziare una nuova partita.

Considerazioni

La progettazione di questo componente non è stata particolarmente difficoltosa in quanto è per la maggiorparte ispirata al registerfile, le difficolta riscontrate sono state piu che altro relative alla costruzione fisica inquanto sia per il multiplexer e il decoder, sia per il componente in se è stato necessario collegare molti fili, operazione delicata che spesso ha portato a degli errori involtari e di conseguenza ad una fase di "debugging". Ho deciso di utilizzare un piano di indirizzamento "bucato" perche evitarlo mi avrebbe costretto a sviluppare un circuito per la traduzione degli indirizzi o qunatomeno avrebbe reso molto piu complicato il circito all'interno del keypad.Un problema che ho dovuto affrontare è stato quello relativo alla gestione di logisim dei latch-set-clear, infatti logism non supporta a pieno circuiti di quel tipo che inizialmente risultano essere un uno stato di errore e che spesso causano un errore del sistema detto 'Oscilation'. Per ovviare a questo problema ho collegato ogni memoria a un bottone in modo tale da essere sicuro che ogni valore contenuto nei registri sia inizializzato a inizio partita, soluzione non sufficente in quanto l'oscillazione continuava a presentarsi in modo apparentemente casuale. Dopo le dovute indagini ho capito che il problema era in parte causata dal collegamento logico del clock con i segnali di reset. Non riuscendo, tuttavia, a isolare in modo specifico la causa esatta del problema ho dovuto cercare per esaustione la combinazione che meglio gestiva il problema, ovvero quella riportata nel curcuito.

AttackProcessor





Questo componente è adibito al processing dell attacco. Come da principio durante gli stati di attesa l output equivarrà a l'input. Durante la fase di attacco questo componente distingue due casi:

- *Stato di preparazione:* in questo stato (1° bit di stato a 0) l'AttackProcessor fornirà in output 01 (valore che rappresenta una sezione di nave) se il valore in input è 00 (valore che rappresenta l'acqua) in modo da permettere il posizionamento delle navi.
- *Stato di gioco:* in questo stato (1° di stato a 1) l'AttackProcessor fornirà in output il valore 10 (valore che rappresenta una sezione di nave affondata) se il valore in input è 01 (sezione di nave) in modo da permettere l'affondamento delle navi.

i1	**i0**	**s2**	**s1**		**o1**	**o0**
0	0	0	0		0	0
0	0	0	1		0	1
0	0	1	0		0	0
0	0	1	1		0	0
0	1	0	0		0	1
0	1	0	1		0	1
0	1	1	0		0	1
0	1	1	1		1	0
1	0	0	0		1	0
1	0	0	1		1	0
1	0	1	0		1	0
1	0	1	1		1	0
1	1	0	0		x	x
1	1	0	1		x	x
1	1	1	0		x	x
1	1	1	1		x	x

	00	**01**	**11**	**00**
00	0	0	0	0
01	0	0	*1*	0
11	x	x	x	x
10	*1*	*1*	*1*	*1*

- o1 = (i0 AND s2 AND s0) OR (i1 AND !i0)

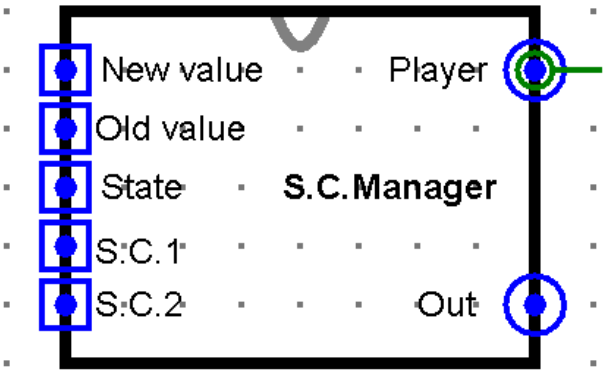
	00	**01**	**11**	**10**
00	0	*1*	0	0
01	*1*	*1*	0	*1*
11	x	x	x	x
10	0	0	0	0

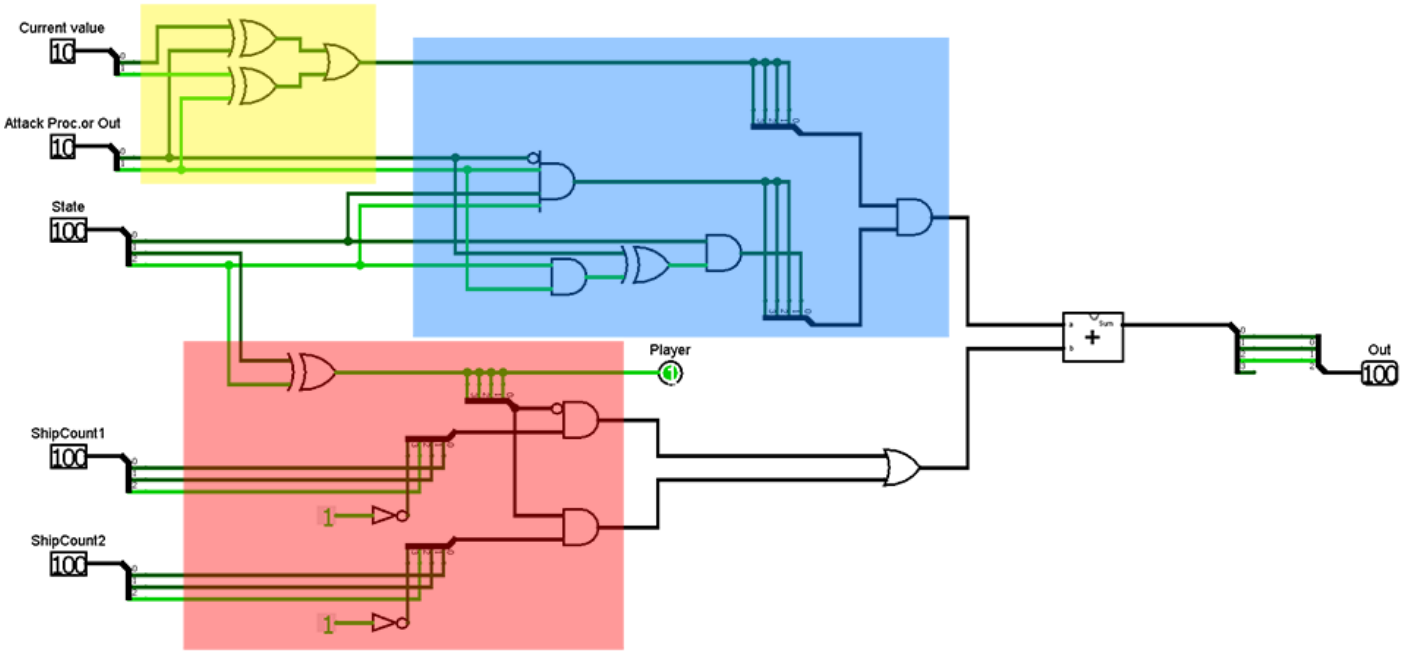
- o0 = (!i1 AND !s2 AND s0) OR (!i1 AND i0 AND !s0) =
  = !i1 AND ((!s2 AND s0) OR (i0 AND !s0))

Considerazioni

Lo sviluppo di questo componente, una volta definito il principio di progettazione e quindi il comportamento dell'AttackProcessor, si è ridotto a una semplice tabella di verità.

ShipCountManager





Lo ShipCountManager è adibito alla gestione del contatore delle navi di ogni giocatore. Il suo compito è quello di incrementare o decrementare il corretto contatore in base alla fase di gioco. Per lo scopo è stato realizzato un sommatore a 4 bit. Il concetto di base è quello di sommare zero in stato di attesa mentre sommare +1 o -1 in base alla situazione; per fare cio, essendo i contatori a 3 bit e dovendovi sommarci un valore negativo, internamente espandiamo il valore a 4 bit in modo da poter utilizzare il -1 (1111) in complento a 2. Il circuito è diviso in 2 parti:

- *Selezione dell'operazione* (blu) che seleziona +1 in stato di preparazione alla partita e -1 in stato di gioco, in modo tale da incrementare il contatore ogni volta che un giocatore posiziona una nave e decrementarlo ogni volta che una nave viene affondata. Il cicuito prende quindi in input lo stato e l'output del AttackProcessor in modo tale da essere consapevole dell'effetto della mossa corrente e agire di conseguenza. In supplemento a quanto detto vi è un circuito (riportato in giallo) che gestisce la situazione in cui un giocatore attacchi o posizioni una nave in una cella ove vi è gia una nave affondata o una nave propria nave posizionata. Per fare cio, sono stati messi in XOR il valore corrente e il valore prossimo, il risultato in AND con la selezione dell'operazione, nei casi sopra citati, porta a 0 l output in modo tale da sommare 0 e non modificare il contatore.
- *Selezione del contatore:* (rosso) che in base allo stato della partita e al giocatore corrente seleziona l'appropriato contatore, overro il medesimo del giocatore se siamo nella fase di preparazione, o il contatore opposto al giocatore che detiene il turno se siamo in fase di partita.

Selezione dell'operazione:

s2	**s0**	**o1**	**o0**		**A3**	**A2**	**A1**	**A0**	
0	1	0	1		0	0	0	1	+1
1	1	1	0		1	1	1	1	-1

- A3 = A2 = A1 = s0 AND s2 AND o1 AND !o0
- A0 = (!s2 AND s0 AND !o1 AND o0) OR (s2 AND s0 AND o1 AND o0) =

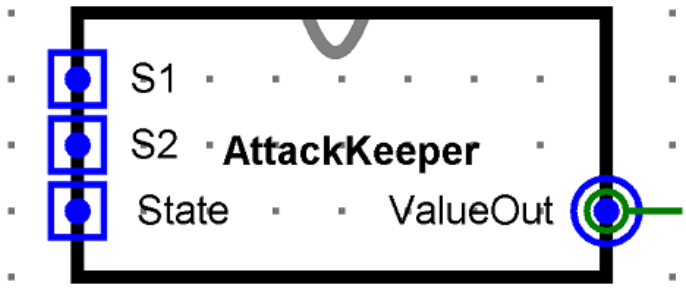
  = s0 ((!s2 AND !o1 AND o0) OR (s2 AND o1 AND o0))

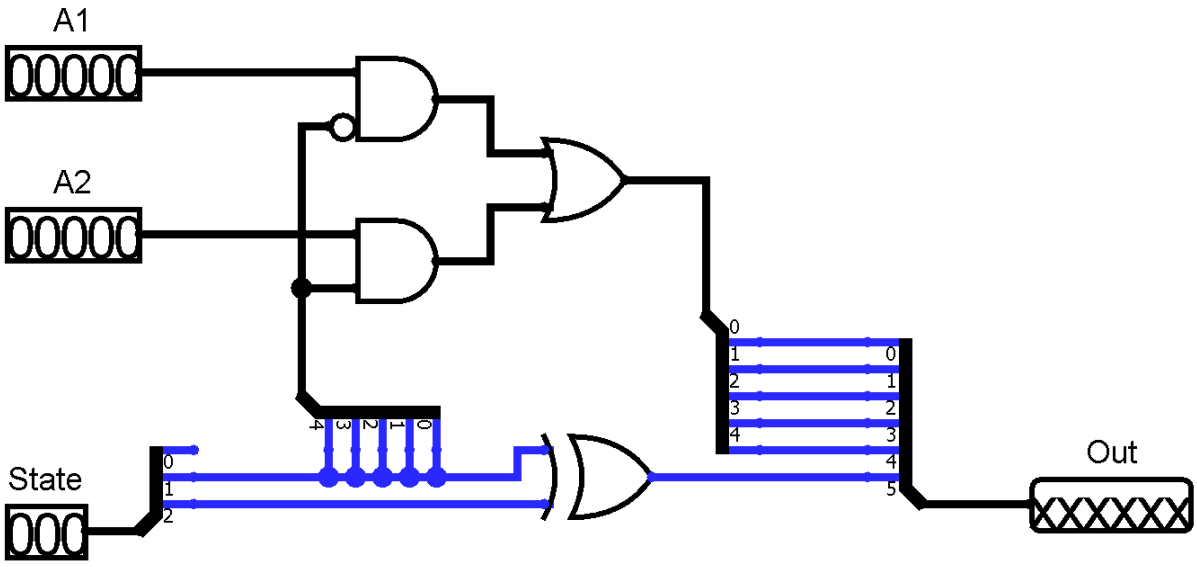
  = s0 AND ((s2 AND o1) XOR o0)

Selezione del constatore:

Mettendo in XOR i primi 2 bit dello stato otteniamo il giocatore su cui agire, questa informazione viene utilizzita per selezionare contatori attraverso lo stesso meccanismo del multiplexer.

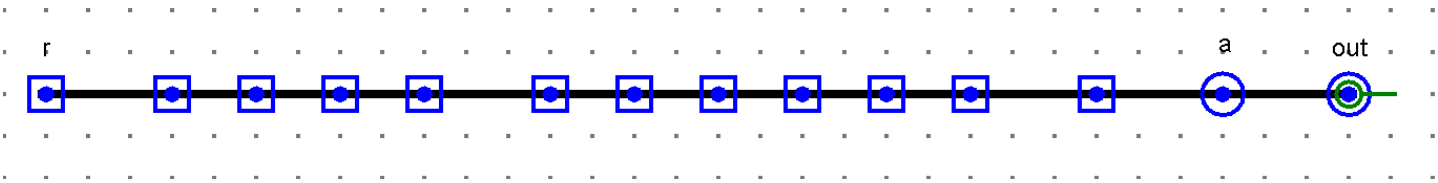
AttackKeeper

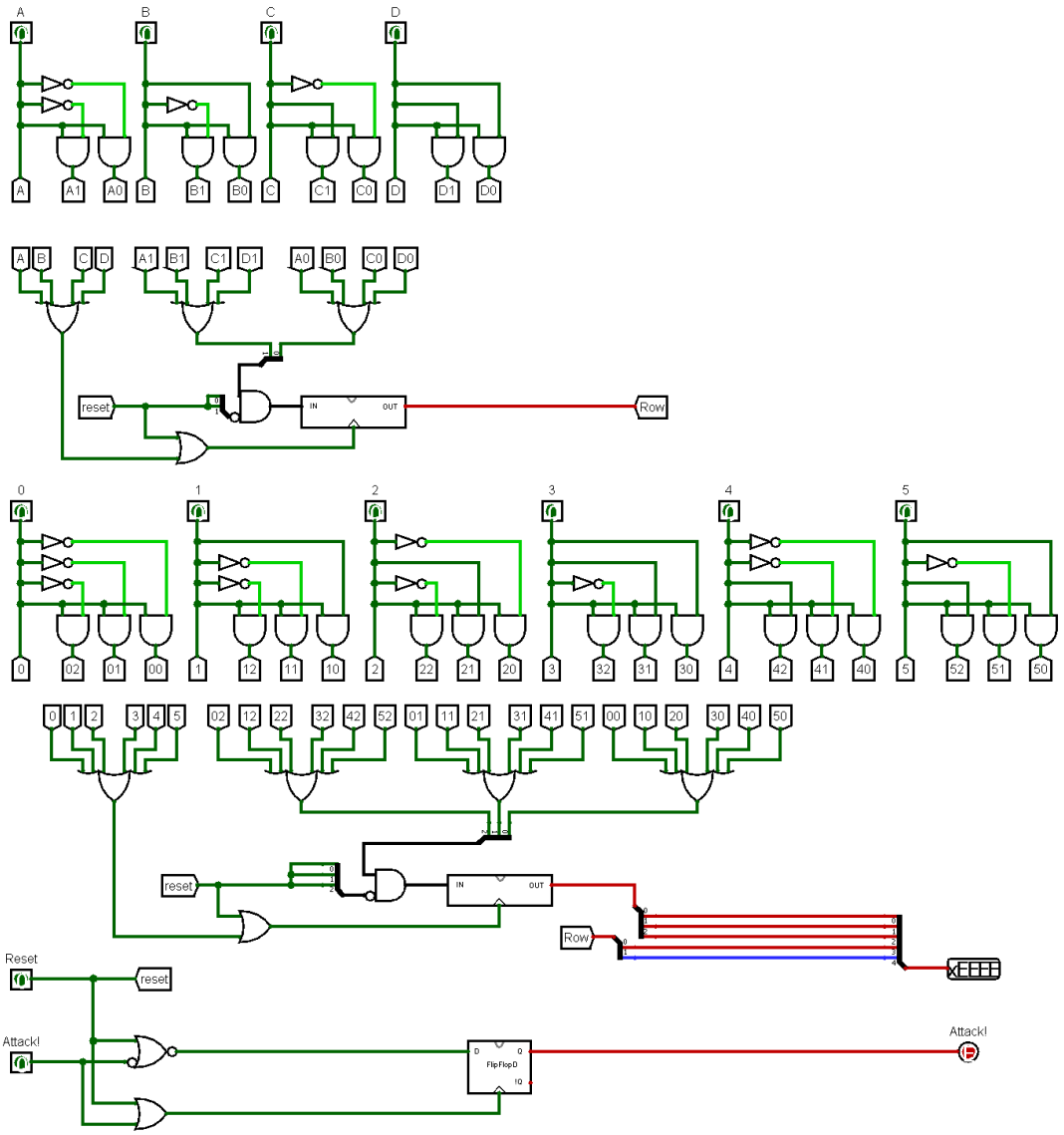




Questo è un semplice componente che in base allo stato del gioco seleziona opportunamente l'attacco da eseguire; esso riceve in input i due output delle keypad e fornisce in output quello corrispondente al giocatore che detiene il turno appendendovici in prima posizione il bit che rappresenta la griglia su cui agire, ovvero quella del giocatore corrente se siamo in stato di preparazione oppure quella dell'avversario se sivamo in stato di gioco. Il circuito seleziona l'input attraverso il medesimo meccanismo di un comune multiplexer, mentre calcola il bit del giocatore attraverso un XOR fra stato di gioco e turno. (2° e 3° bit di stato).

Keypad



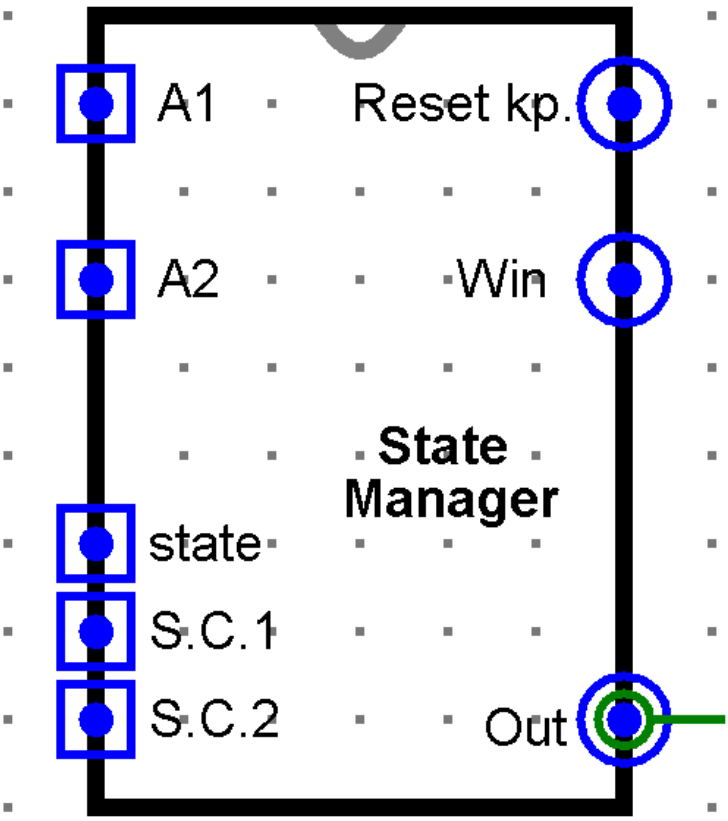


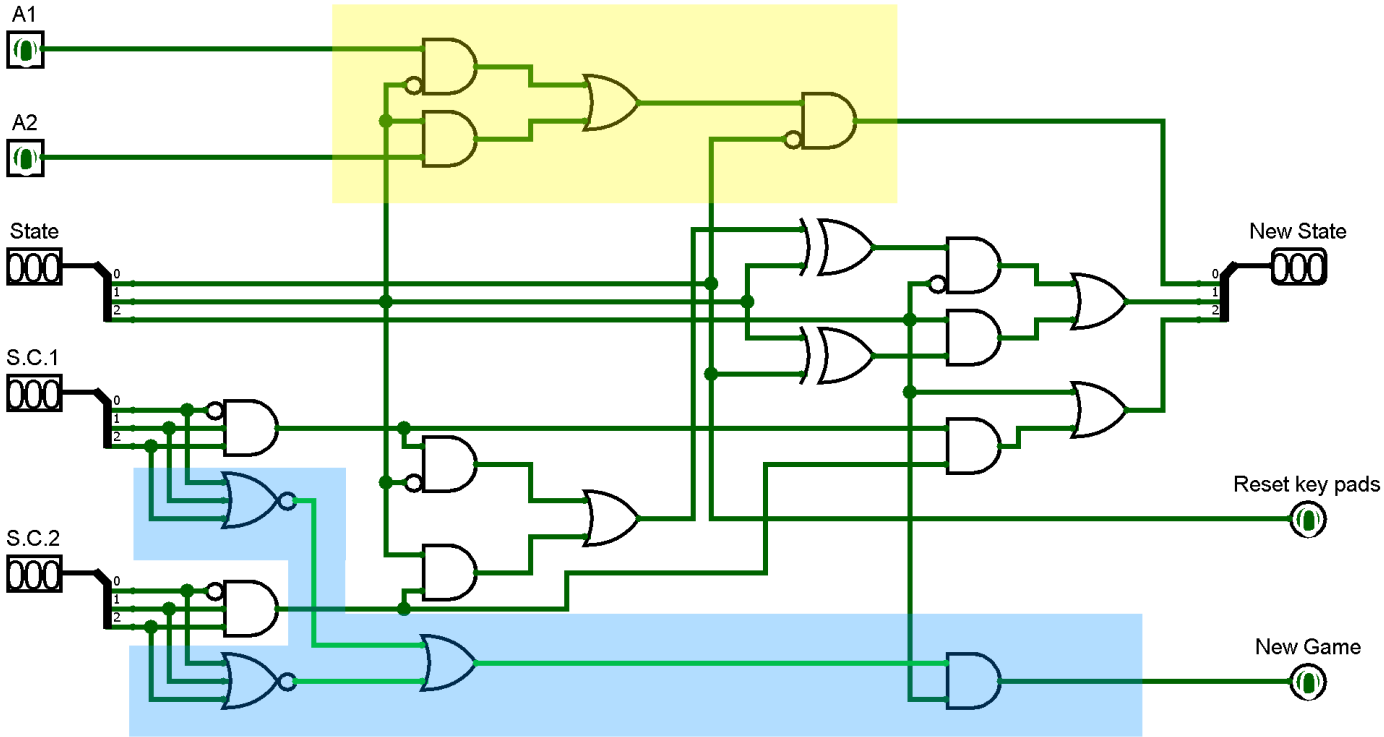
Il keypad è adibito alla selezione dell'attacco e alla modifica di stato in stato di attacco; è composto da 2 registri e un flipflop. I due registri memorizzano la selezione delle cordinate ove attaccare. Ogni bottone, mediante un opportuno circuito, salva nel registro il valore ad esso corrispondente. Cosi facendo all' 1°input corrispondente al tasto 'A' corrdisponderà il valore 00 e cosi via. L'ultimo input è adibito al submit dell'attacco, esso si limita a salvare che la mossa è stata lanciata. Ciuascuna delle memorie è collegata al segnale di reset in modo tale che al passaggio di turno oppure all'avvio di una nuova partita le memorie vengano azzerate. I 3 valori delle memorie sono unite cosi come definito nelle specifiche e portati in output.

Considerazioni

Per quanto riguarda l'aspetetto esteriore del componente, ho optato per una singola linea ansiche un rettangolo in quanto distrae meno dalla tastiera. inizialmente avevo deciso di posizionare i pin di output sopra il componente cosi come fatto per il display driver, tuttavia i tasti non risultavono tutti cliccabili sovrapponendoli al componente. La scelta di salvare il click del tasto attacca invece che collegarlo direttamente all'output deriva dal fatto che i componenti che necessitano di tale informazione non riuscivano a regire al cambio di valore in tempo tale da scrivere in memoria, problema accentuato dall'asincronia rispetto al clock.

StateManager





Questo componente ha il compito di aggiornare adeguatamnte lo stato. Per farlo tiene conto di 5 variabili. lo stato corrente del gioco, i due contatori correnti, e i due valori di attacco dei due giocatori.

- *Calcolo dello stato di attacco (giallo):* Seleziona il flag di attacco dell giocatore corrente e lo nega se se il giocatore aveva gia attaccato.
- *Calcolo del turno e dello stato di gioco:* Questo sotto-circuito calcola il turno del giocatore, tenendo conto dello stato del gioco, e lo stato del gioco, che muta in base allo stato dei contatori.

cf	s2	s1	s0		s1

0	0	0	1		0
0	0	0	1		0
0	0	1	0		1
0	0	1	1		1
0	1	0	0		0
0	1	0	1		1
0	1	1	0		1
0	1	1	1		0
1	0	0	0		1
1	0	0	1		1
1	0	1	0		0
1	0	1	1		0
1	1	0	0		0
1	1	0	1		1
1	1	1	0		1
1	1	1	1		0

		00	**01**	**11**	**10**
00		0	0	*1*	*1*
01		0	*1*	0	*1*
11		0	*1*	0	*1*
10		*1*	*1*	0	0

s1 = (!cf AND !s2 AND s1) OR (s2 AND s1 AND !s0) OR (s2 AND !s1 AND s0) OR (cf AND !s2 AND !s1) =

```
= !s2((!cf AND s1) OR (cf AND !s1)) OR s2((s1 AND!s0) OR (!s1 AND s0)) =

=  !s2(cf XOR s1) OR s2(s1 XOR s0)
```

s2 = (cf21 AND cf2) OR s2

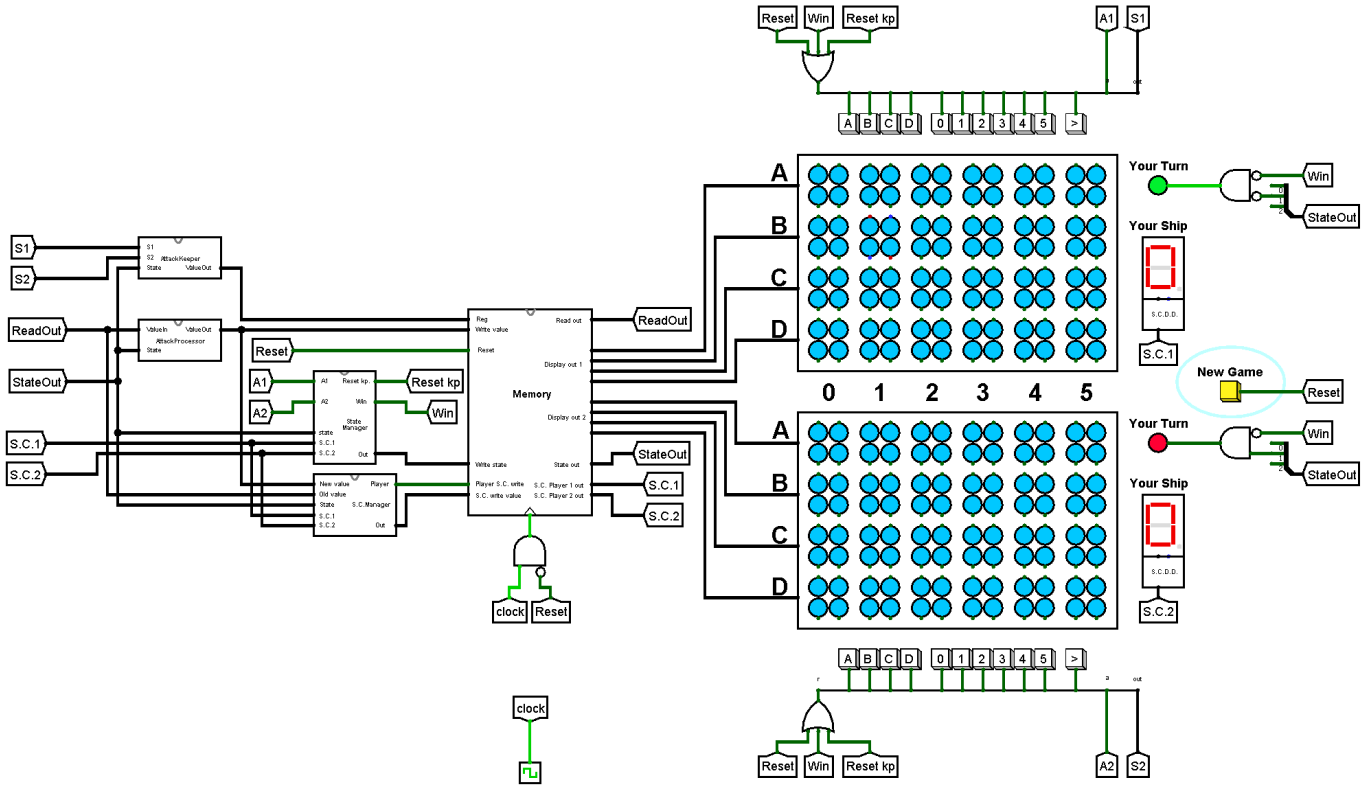
Il primo bit di stato deve essere a 1 solo se siamo gia in stato di partita oppure se i due contatori sono stati incrementati fino a 4.

- *Controllo fine gioco (blu):* controlla se i due contatori sono a 0 e se lo stato di gioco è in partita. se queste condizioni si verificano allora l'output 'new game' sarà a 1 e resttera il gioco analogamente al bottone new game.

Considerazioni

Lo sviluppo di questa parte è stata una delle piu difficili, infatti, diversante dagli altri componenti, non ho potuto scrivere una tabella di verità perche le variabili da tenere conto sono stroppe e si sarebbe venuta a creare una tabella molto grande. Per ovviare al porblema ho diviso il circuito in sottoparti; degne di nota sono quelle riguardanti gli stati di gioco s2 e s1. infatti per fare cio, ho creato un sotto componente che si accorgesse che il contatore de giocatore corrente è stato riempito, in modo tale da dover considerare una sola variabile ansiche 6, riducendo notevolmente la tabella di verità - sopra riportata -.

Interazione dei componenti



I componenti sopra descritti interagioscono fra loro principalemente mediante lo stato, infatti, come da principio i componenti reagiscono allo stato di attacco. il perno fondamentale del circuito e quindi lo stato. le informazioni sulla partita sono tutte memorizzate nella memoria che è presentata visivamente all'utente mediante i due display di gioco, per quanto riguarda le griglie, attraverso i due display a 7 segmenti per quanto riguarda il numero di navi, e attraverso un led per il turno. l'interazione utente-circuito avviene mediante il keypad, che trasmettera la selezione all'AttackKeeper il cui output verra usato dalla memoria come indirizzo di lettura/scrittura, al che AttackProcessor utilizzera questo valore come valore da processare e valutando anche lo stato corrente metterà in output il dovuto valore, valore su cui reagirà lo ShipCountManager aggiornando a dovere i contatori. Tutte queste interazioni pero non hanno alcun effetto fintanto che lo stato non muta in stato di attacco. Questo evento è gestito dallo StateManager che presa visione del effetivo attacco da parte dell'utente mediante il keypad metterà a 1 il valore del'ultimo bit dello stato, scatenando una reazione in tutti gli altri componenti che collaborando fra loro andranno a scrivere un nuovo valore all'interno della memoria, che a sua volta causera un cambiamento sulle periferiche di output. Vi è un bottone comune hai due giocatori che una volta premuto resetta il valore delle memorie permettendo una nuova partita. come gia spiegato questo risolve i problemi relativi all'inizializzazione dei valori nella memoria. Tutto il circuito è sincronizzato con un clock. La presenza del clock del bottone di reset, se non opportunamente gestiti portavano all'errore di oscillazione, che è stato risolto impedendo al clock di propagarsi sui registi qunado vi è il segnale di reset su di essi.

Possibili sviluppi futuri

Un possibile scenario per una versione 2.0 è quello di implementare un giocatore virtuale. Una possibile strategia sarebbe quella di creare un circuito che generi casualmente una selezione e che metta a 1 il flag d attacco solo quando il valore corrispondente alla selezione generata sia una valore accettabile. Questo potrebbe essere implementato senza troppe difficolta in quanto la selezione è diretta alla memoria, quindi una volta generato una selezione casule il valore corrispondente sarebbe subito disponibile sulla porta di lettura della memoria.