UNIVERSITÀ
di VERONA

ELABORATO SIS

LABORATORIO DI ARCHITETTURA DEGLI ELABORATORI
2022/2023

“Gestione di un parcheggio”

COMPONENTI DEL GRUPPO:

TERZIU FABIO VR471449

ZANRE FULBERT VR490128

INDICE:

1 ARCHITETTURA GENERALE DEL CIRCUITO:

1.1 Introduzione

L' elaborato consiste nella gestione di un parcheggio automatizzato per automobili, ed in questa prima fase vedremo la sua introduzione per poi mostrare più nel dettaglio il suo funzionamento.

Cominciamo col dire che il parcheggio è suddiviso in tre settori:

- Settore “A”, 31 posti disponibili massimi;
- Settore “B”, 31 posti disponibili massimi;
- Settore “C”, 24 posti disponibili massimi;

Al momento d'ingresso l'utente deve dichiarare in quale settore desidera parcheggiare.
Analogamente al momento dell'uscita deve dichiarare da quale settore proviene.
Il parcheggio rimane libero durante la notte, permettendo a tutte le macchine di entrare ed uscire a piacimento, quindi senza la necessità di dover dichiarare il parcheggio del quale si vuole usufruire.

La mattina il dispositivo viene acceso da un operatore tramite l'inserimento del codice a 5 bit ‘11111’.
Al ciclo successivo il sistema attende l'inserimento del numero di automobili presenti nel settore “A”, memorizzandone il valore.
Nei due cicli successivi avviene lo stesso per i settori “B” e “C”.
Se dovesse venir inserito un valore superiore a quello del massimo numero di posti consentiti, il parcheggio verrà considerato pieno.
A questo punto, ricevuti i valori d'ingresso che indicano quanti e quali posti macchina sono occupati, il sistema inizia il suo funzionamento autonomo, quindi ogni volta un utente si dovrà avvicinare alla posizione di ingresso o uscita e premere un pulsante relativo al settore in cui intende parcheggiare oppure ha già parcheggiato.
Il dispositivo si spegne quando riceve in input la sequenza ‘00000’.

1.2 Tabella I/O e schema generale circuito

Il circuito ha 2 ingressi e 3 uscite:

Ingressi:

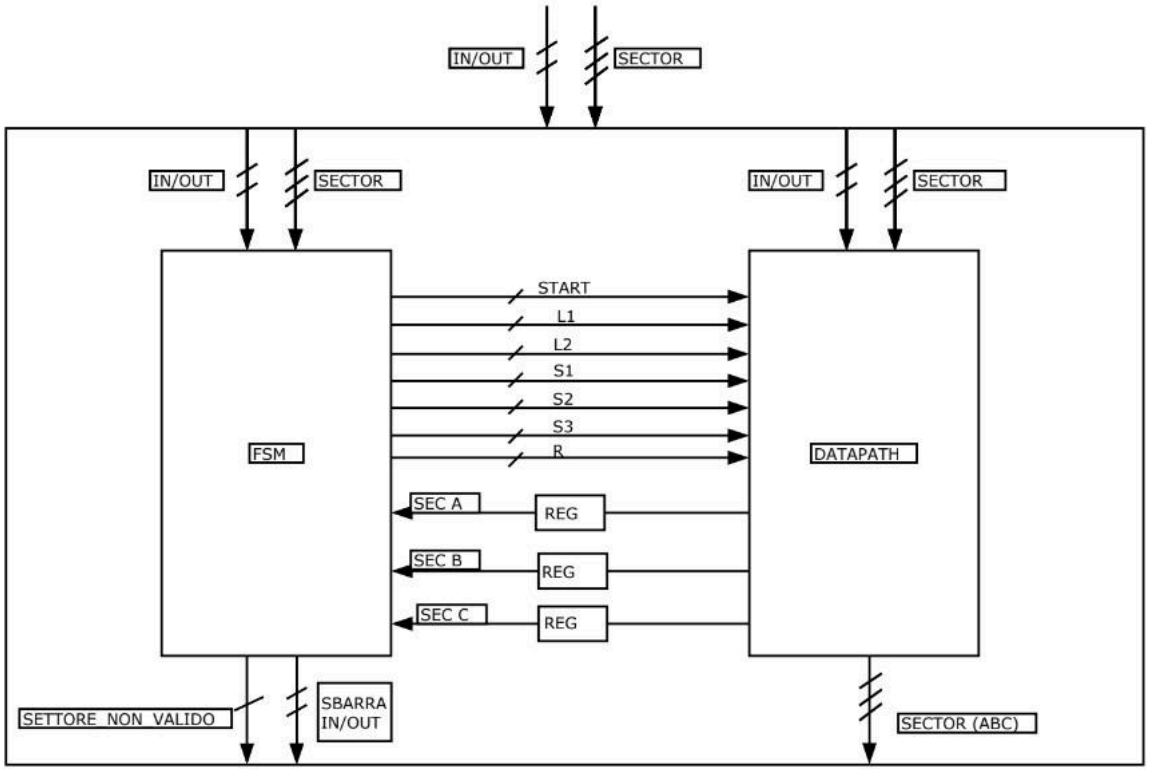
- IN/OUT (2 bit): se l'utente è in ingresso il sistema riceve in input la codifica 01, se invece è in uscita riceverà la codifica 10.
  Le restanti codifiche a 00 e 11 vengono interpretate come anomalie di sistema e la richiesta verrà ignorata, non aprendo quindi alcuna sbarra.
- SECTOR (3 bit): i settori sono indicati con codifica one-hot, ovvero una stringa di 3 bit in cui uno solo assume valore 1 mentre gli altri 0. Le restanti codifiche invece verranno interpretate come errori di inserimento e la richiesta verrà ignorata.

Uscite:

- SETTORE_NON_VALIDO (1 bit): questo bit assumerà valore 1 qualora il settore inserito non fosse valido.
- SBARRA IN/OUT (1 bit a sbarra): 10 sbarra di ingresso alzata, 01 sbarra di uscita alzata, 00 entrambe abbassate ed infine 11 entrambe alzate.
  La sbarra rimane aperta solo per un ciclo di clock, anche se la richiesta del ciclo successivo dovesse venire ignorata.
- SECTOR_(A/B/C) (1 bit a settore): questo bit assume il valore 1 se tutti i posti nel settore sono occupati, mentre 0 se ci sono ancora posti liberi.
  Se l'utente dovesse chiedere di entrare in un parcheggio già pieno, il sistema non aprirà alcuna sbarra.

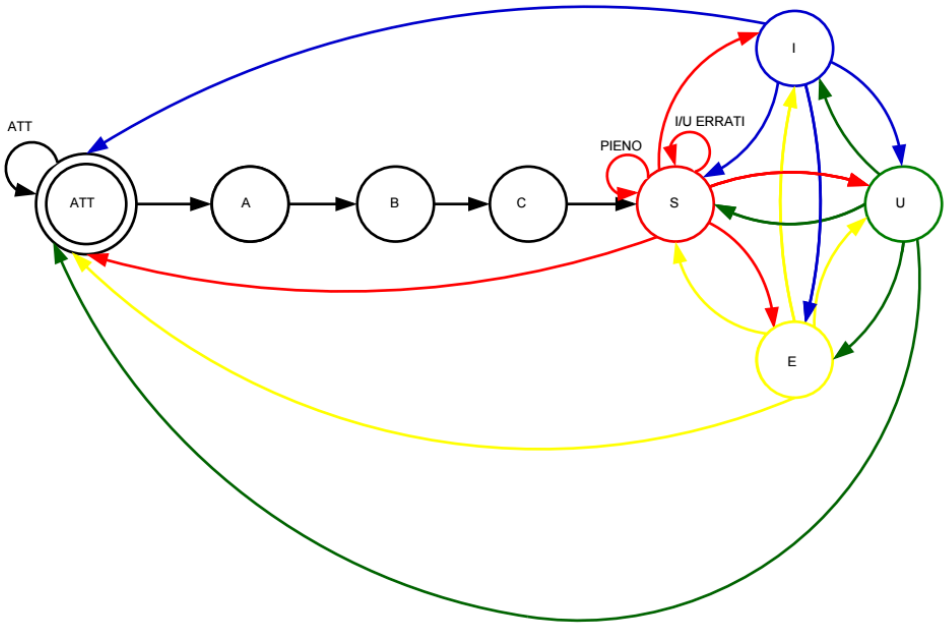
Schema generale del circuito:

Di seguito viene riportato lo schema del circuito, ovvero l'FSMD che riceve 5 bit in ingresso e ne restituisce 6 in uscita.
L'FSMD mette in comunicazione il controllore FSM e l'elaboratore Datapath, relazione che andremo a vedere e commentare in seguito.



2 FSM:

2.1 STG



Un' STG è un grafo delle transazioni grazie alla quale si rappresenta graficamente una FSM. In essa, essendo la parte di controllo del FSMD, si vanno a creare degli stati, ognuno con una funzione specifica ed il passaggio da uno stato all'altro avviene tramite codifiche.

La nostra STG iniziale è composta da 8 stati:
Lo stato iniziale 'ATT' è lo stato di attesa, in cui la macchina è considerata spenta.
Da questo stato, si può andare soltanto nello stato 'A' con la codifica '11111', dove l'operatore inserirà il numero di posti occupati nel parcheggio A.
Lo stesso avviene subito dopo con B e C per poi finire nello stato di start 'S'.
A questo punto, la macchina inizia il suo funzionamento autonomo avendo già tutti i dati di cui ha bisogno.
Dallo stato'S' si può finire in 'I', 'U', 'E' ed 'ATT' e lo stesso vale per questi ultimi.
Nello stato 'I' si finisce soltanto se una macchina desidera fare un ingresso e si entra con codifica '01' nei rispettivi parcheggi A=100, B=010, C=001, e a questo punto si aprirà la sbarra d'ingresso con codifica 10.
Attenzione però perchè se il parcheggio dove si desidera entrare dovesse risultare pieno, e ciò lo capiamo dai selettori in ingresso (pa=1, pb=1, pc=1) ricevuti dal datapath, si ritornerebbe allo stato di start e non verrà aperta alcuna sbarra.
Si rimane/ritorna sempre nello stato di start se la codifica di ingresso/uscita dovesse essere errata (quindi 00, 11).
Nello stato 'U' invece si finisce soltanto se l'utente desidera uscire dal parcheggio con la seguente codifica di uscita '10' e settore A=100, B=010, C=001.
Rimane ora soltanto lo stato 'E', ovvero lo stato dove ci si finisce se la codifica di settore dovesse essere errata, quindi nessuna delle tre nominate in precedenza, e a questo punto viene alzato il primo bit di in output che sta a indicare proprio l'errore.

L’’errore persisterà finché non viene inserita una codifica con la combinazione di settore esatta oppure finché non si spegne la macchina.
Come detto in precedenza, la macchina si può spegnere dallo stato ‘S’, ‘I’, ‘U’, ‘E’, con codifica ‘00000’ e si ritornerà nello stato di attesa.

2.2 STG (codifica)

```
.model fsm
.inputs out in sa sb sc pa pb pc
.outputs err sb.in sb.out s1 s2 s3 l1 l2 start r

.start_kiss
.i 8
.o 10
.p 94
.s 8
.r ATT
```

```
#preaccensione+spegnimento
----0--- ATT ATT 0000000000
---0---- ATT ATT 0000000000
--0----- ATT ATT 0000000000
-0------ ATT ATT 0000000000
0------- ATT ATT 0000000000
11111--- ATT A 0000000000
-------- A B 0001000110
-------- B C 0001101010
-------- C START 0001111110
00000--- START ATT 0001110001
00000--- ERR ATT 0001110001
00000--- I ATT 0001110001
00000--- U ATT 0001110001

#nulla (i/u errato)
00001--- START START 0001110010
00010--- START START 0001110010
00100--- START START 0001110010
11001--- START START 0001110010
11010--- START START 0001110010
11100--- START START 0001110010
00001--- I START 0001110010
00010--- I START 0001110010
00100--- I START 0001110010
11001--- I START 0001110010
11010--- I START 0001110010
11100--- I START 0001110010
00001--- U START 0001110010
00010--- U START 0001110010
00100--- U START 0001110010
11001--- U START 0001110010
11010--- U START 0001110010
11100--- U START 0001110010
00001--- ERR START 0001110010
00010--- ERR START 0001110010
00100--- ERR START 0001110010
11001--- ERR START 0001110010
11010--- ERR START 0001110010
11100--- ERR START 0001110010
```

```
#errore settore
01000--- START ERR 1001110010
1-000--- START ERR 1001110010
--011--- START ERR 1001110010
--101--- START ERR 1001110010
--110--- START ERR 1001110010
--111--- START ERR 1001110010
01000--- I ERR 1001110010
1-000--- I ERR 1001110010
--011--- I ERR 1001110010
--101--- I ERR 1001110010
--110--- I ERR 1001110010
--111--- I ERR 1001110010
01000--- U ERR 1001110010
1-000--- U ERR 1001110010
--011--- U ERR 1001110010
--101--- U ERR 1001110010
--110--- U ERR 1001110010
--111--- U ERR 1001110010
01000--- ERR ERR 1001110010
1-000--- ERR ERR 1001110010
--011--- ERR ERR 1001110010
--101--- ERR ERR 1001110010
--110--- ERR ERR 1001110010
--111--- ERR ERR 1001110010

#ingresso + pieno
011001-- START START 0001110010
01010-1- START START 0001110010
01001--1 START START 0001110010
011001-- I START 0001110010
01010-1- I START 0001110010
01001--1 I START 0001110010
011001-- U START 0001110010
01010-1- U START 0001110010
01001--1 U START 0001110010
011001-- ERR START 0001110010
01010-1- ERR START 0001110010
01001--1 ERR START 0001110010
```

```
#ingresso + libero
011000-- START I 0100110110
01010-0- START I 0101011010
01001--0 START I 0101101110
011000-- U I 0100110110
01010-0- U I 0101011010
01001--0 U I 0101101110
011000-- ERR I 0100110110
01010-0- ERR I 0101011010
01001--0 ERR I 0101101110
011000-- I I 0100110110
01010-0- I I 0101011010
01001--0 I I 0101101110

#uscita
10100--- START U 0010110110
10010--- START U 0011011010
10001--- START U 0011101110
10100--- I U 0010110110
10010--- I U 0011011010
10001--- I U 0011101110
10100--- ERR U 0010110110
10010--- ERR U 0011011010
10001--- ERR U 0011101110
10100--- U U 0010110110
10010--- U U 0011011010
10001--- U U 0011101110

.end_kiss

.end
```

Inputs (8 bit):

1. out/in: input principale, indica se si desidera entrare oppure uscire con rispettive codifiche: ‘01’, ‘10’
2. sa/sb/sc: input principale, indica il settore con seguente rispettiva codifica: A=100, B=010, C=001.
3. pa/pb/pc: ingresso ricevuto dal datapath, mostra quali dei settori sono pieni oppure ancora liberi. Sono input indipendenti l’uno dall’altro. pa=1(parcheggio A pieno), pa=0(parcheggio A con posti disponibili). Lo stesso vale per B e C.

Outputs (10 bit):

1. err: output principale, che si alza se e soltanto se la codifica dell’input sa/sb/sc è errata.
2. sb.in/sb.out: output principale, 10 indica la sbarra d’ingresso aperta, mentre 01 la sbarra d’uscita aperta. 00 sono entrambe chiuse. Le sbarre si aprono soltanto quando un utente desidera effettuare un ingresso oppure un’uscita.

   In fine abbiamo aggiunto i seguenti output da dare in ingresso al datapath:
3. s1/s2/s3: output (autonomi l’uno dall’altro) che serve per capire se ci troviamo nei primi tre cicli di clock dove attendiamo l’inserimento dei valori oppure dal quarto ciclo dove la macchina lavora in autonomia. Quando i bit sono abbassati significa che attendono l’inserimento dei posti macchina oppure che aggiorno il valore della codifica, altrimenti sono a 1.
4. l1/l2: output per qualsiasi funzione riguardante i tre settori. A=01, B=10, C=11.
5. start: output a 1 indica quando la macchina è attiva, quindi quando non è nello stato di attesa.
6. r: output a 1 soltanto quando si passa da uno stato a quello di attesa

La codifica del STG nel file .blif comincia con la keyword:
“.model .inputs .outputs”**,** per indicare rispettivamente il nome del progetto, gli ingressi e le uscite.

Dopo la keyword start_kiss, vengono definiti:

- .i (numero ingressi)
- .o (numero uscite)
- .p (numero transazioni)
- .s (numero stati)
- .r (stato di reset)

Definiti questi valori, si passa alla scrittura delle combinazioni per passaggio da stato a stato e bisogna scriverle nel seguente ordine:
ingresso - stato attuale - stato prossimo - uscita

Esempio: 11111- - - ATT A 0000000000 (accensione macchina)

2.3 Minimizzazione degli stati

La minimizzazione degli stati consiste nel saper ridurre il numero degli stessi, attraverso degli appositi criteri di equivalenza che ci permettono di raggrupparli per caratteristiche simili.
Questa funzione si lancia con il comando *state_minimize stamina*

Osserviamo il prima ed il dopo la minimizzazione:

```
sis> print_stats
FSM             pi= 8   po=10   nodes= 10       latches= 0
lits(sop)=   0  #states(STG)=   8
```

```
sis> print_stats
FSM             pi= 8   po=10   nodes= 10       latches= 0
lits(sop)=   0  #states(STG)=   5
```

Possiamo notare infatti che gli stati da 8 sono passati a 5.

2.4 Codifica degli stati

Attraverso il comando *state_assign jedi* viene effettuata in automatico la codifica effettiva degli stati, operazione necessaria per continuare.

```
sis> print_stats
FSM             pi= 8   po=10   nodes= 13       latches= 3
lits(sop)= 312  #states(STG)=   5
sis>
```

Come possiamo vedere in figura, sono stati creati nodi, latch di memoria e letterali.

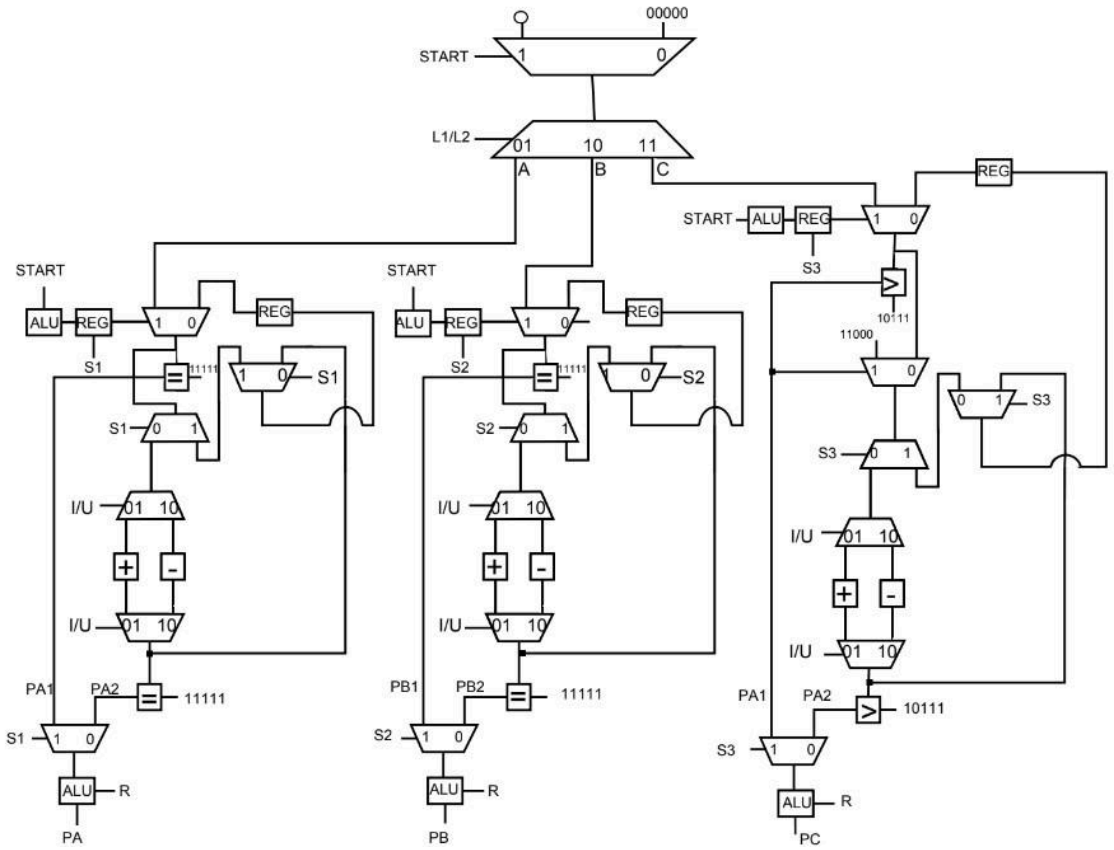
3 Datapath:

3.1 Schema generale

Conclusa la parte di controllo, arriva il momento della parte di elaborazione, ovvero il datapath.
Il datapath non ha un approccio comportamentale come quello della FSM, bensì ragiona in termini di unità funzionali connesse tra loro, quindi possiamo dire un approccio strutturale. Esso lavora con programmi e circuiti a flussi, diversamente dal FSM che lavora sulla logica condizionale.

Il seguente datapath, è composto principalmente da quattro parti:

1. La parte superiore comanda l'avvio del circuito.
2. Quella di sinistra lavora sui dati del parcheggio A.
3. La centrale sui dati del parcheggio B.
4. Quella invece di destra lavora sui dati del parcheggio C.



3.2 Tabella I/O

Il datapath ha come obiettivo quello di dare in uscita quando e se i parcheggi sono pieni oppure liberi. (PA,PB,PC)
Per fare ciò, oltre ai due principali ingressi visti in precedenza:

- IN/OUT (2 bit)
- SECTOR (3 bit)

necessitiamo di ulteriori ingressi ricevuti dal FSM, per poter comunicare con essa in modo ottimale.
Andiamo ora ad analizzarli:

- START(1 bit): questo ingresso ci serve per capire quando il dispositivo è considerato 'acceso', dunque questo bit sarà abbassato soltanto quando ci troveremo nello stato di attesa.

- L1, L2 (1 bit ciascuno): input che lavorano in combinazione per determinare se dobbiamo lavorare con la parte del circuito che gestisce il parcheggio A, B oppure C, con le seguenti codifiche:
  00 quando la macchina è spenta
  01 settore A
  10 settore B
  11 settore C

- S1, S2, S3 (un bit ciascuno): questi tre ingressi sono rivolti rispettivamente ai settori A, B, C.
  Differenziano la prima parte di immissione posti macchina, con la seconda dove il funzionamento dev'essere autonomo.
  Quando i bit sono abbassati significa che attendono l'inserimento dei posti macchina e quindi è attiva soltanto una parte del circuito oppure indicano l'aggiornamento della codifica, quando invece sono a 1 si va a creare un anello chiuso in modo da poter lavorare con i dati finora ricevuti e fare eventuali somme/sottrazioni, ingressi/uscite.

- r (un bit): questo ingresso, usato come selettore (come anche quelli precedenti), ci è servito per una singola funzione.
  Questo bit è a 1 soltanto quando vogliamo andare nello stato di attesa.
  Necessitiamo di questo selettore in modo che quando si immette la codifica di spegnimento '00000', i settori pieni diano come output 0, proprio per simulare la macchina spenta.

```
.model datapath
.inputs V0 V1 V2 V3 V4 S1 S2 S3 L1 L2 START r
.outputs PA PB PC



.subckt Mux I0=V0 I1=V1 I2=V2 I3=V3 I4=V4 S=START O0=O0 O1=O1 O2=O2 O3=O3 O4=O4
.subckt Demuxabc I0=O0 I1=O1 I2=O2 I3=O3 I4=O4 S1=L1 S2=L2 A0=A0 A1=A1 A2=A2 A3=A3 A4=A4 B0=B0 B1=B1 B2=B2 B3=B3 B4=B4 C0=C0 C1=C1 C2=C2 C3=C3 C4=C4

.subckt SA X0=A0 X1=A1 X2=A2 X3=A3 X4=A4 S=S1 PA=PA START=START r=r

.subckt SB X0=B0 X1=B1 X2=B2 X3=B3 X4=B4 S=S2 PB=PB START=START r=r

.subckt SC X0=C0 X1=C1 X2=C2 X3=C3 X4=C4 S=S3 PC=PC START=START r=r


.search Mux.blif
.search Demuxabc.blif
.search SA.blif
.search SB.blif
.search SC.blif

.end
```

```
sis> print_stats
datapath        pi=12   po= 6   nodes=256       latches=18
lits(sop)=1206
```

3.3 Analisi dei componenti

Prima però vediamo brevemente le tipologie di componenti utilizzati:

- Multiplexer: dove un segnale di controllo determina quale dei 2^m ingressi va in uscita. Questo componente, come anche tutti gli altri, va creato su misura per soddisfare la propria esigenza.
  Noi per esempio ne abbiamo utilizzati 4, perché differiscono di bit di ingresso e tipologia di segnale di controllo.
  In seguito viene riportato un esempio di multiplexer che tramite un selettore proveniente dal FSM starta il datapath, quindi fa lo switch soltanto se in ingresso avrà '11111' ed il selettore a 1:

```
.model Mux
.inputs S I0 I1 I2 I3 I4
.outputs O0 O1 O2 O3 O4

.names S I0 O0
11 1
.names S I1 O1
11 1
.names S I2 O2
11 1
.names S I3 O3
11 1
.names S I4 O4
11 1
.end
```

- Demultiplexer: al contrario del multiplexer, tramite un selettore determina quale delle uscite riceve l'ingresso.
  Un esempio del demultiplexer utilizzato per controllare la parte di circuito A, B e C, dove in base ad un selettore a due bit sceglie se dare il valore in uscita ad una delle tre parti:

```
.model Demuxabc
.inputs I0 I1 I2 I3 I4 S1 S2
.outputs A0 A1 A2 A3 A4  B0 B1 B2 B3 B4  C0 C1 C2 C3 C4

.names I0 S1 S2 A0
101 1
.names I1 S1 S2 A1
101 1
.names I2 S1 S2 A2
101 1
.names I3 S1 S2 A3
101 1
.names I4 S1 S2 A4
101 1
.names I0 S1 S2 B0
110 1
.names I1 S1 S2 B1
110 1
.names I2 S1 S2 B2
```

```
110 1
.names I3 S1 S2 B3
110 1
.names I4 S1 S2 B4
110 1
.names I0 S1 S2 C0
111 1
.names I1 S1 S2 C1
111 1
.names I2 S1 S2 C2
111 1
.names I3 S1 S2 C3
111 1
.names I4 S1 S2 C4
111 1

.end
```

- Sommatore: unità aritmetica per la somma.
  Inizialmente bisogna partire da un sommatore base ad un bit, per poi richiamarlo tante volte quanti bit lo si vuole usare.
  Ogni singola operazione ha come uscita di supporto un eventuale riporto della somma tra bit chiamato 'COUT' che poi finisce come un ulteriore bit da prendere in considerazione 'CIN' nell'operazione successiva.
  L'ultimo riporto invece non viene utilizzato.
  Un esempio di sommatore a 5 bit:

```
.model Sommatore
.inputs A0 A1 A2 A3 A4 B0 B1 B2 B3 B4 CIN
.outputs O0 O1 O2 O3 O4 COUT

.subckt Somma1bit A=A4 B=B4 CIN=CIN O=O4 COUT=C3
.subckt Somma1bit A=A3 B=B3 CIN=C3 O=O3 COUT=C2
.subckt Somma1bit A=A2 B=B2 CIN=C2 O=O2 COUT=C1
.subckt Somma1bit A=A1 B=B1 CIN=C1 O=O1 COUT=C0
.subckt Somma1bit A=A0 B=B0 CIN=C0 O=O0 COUT=COUT

.search Somma1bit.blif
.end
```

- Registro: componente che lavora/ fa parte della memoria della macchina.

```
.model registro1
.inputs A
.outputs OUT
.latch A OUT re NIL 0
.end
```

  dove 'A' è l'ingresso del latch, 'OUT' l'uscita, 're' è la tipologia (in questo caso 'rising edge'), 'NIL' il segnale di clock (significa che il registro usa il clock generale del circuito) e '0' è lo stato iniziale del latch.

- Unità di confronto: i comparatori possono essere principalmente di maggioranza, minoranza e uguaglianza.
  Soffermandoci su quello maggioranza, esso utilizza un sistema per calcolare se due ingressi siano uno maggiore dell'altro. Usufruisce della funzione XOR, una porta logica che fornisce in uscita 1 quando almeno un bit di ingresso è a 1.

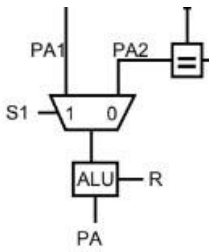
```
.model Maggiore
.inputs A4 A3 A2 A1 A0 B4 B3 B2 B1 B0
.outputs M
.subckt Xor A=A4 B=B4 O=O4
.subckt Xor A=A3 B=B3 O=O3
.subckt Xor A=A2 B=B2 O=O2
.subckt Xor A=A1 B=B1 O=O1
.subckt Xor A=A0 B=B0 O=O0

.names A4 A3 A2 A1 A0 O4 O3 O2 O1 O0 M
1----1---- 1
-1---01--- 1
--1--001-- 1
---1-0001- 1
----100001 1
.search Xor.blif
.end
```

  Una volta comparati bit per bit entrambi gli ingressi, si procede nello scrivere la tabella di verità (.names) dove in uscita avremo 1 se il valore del primo ingresso sarà maggiore del secondo.

- Alu: Arithmetic-Logic unit è un componente ibrido che va creato da 0 secondo le proprie esigenze.
  Prendiamo d'esempio una alu utilizzata nel nostro circuito:

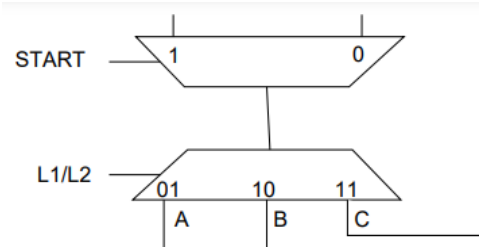
```
.model aluout
.inputs I r
.outputs O

.names I r O
-1 0
00 0
.end
```

  Arriva in ingresso un singolo bit, che indica se il parcheggio è pieno oppure libero, comandato da un selettore 'R' che se a 0, mi da in uscita ciò che ricevo in ingresso, altrimenti mi restituisce 0.



3.4 Parte di controllo

Formata soltanto da due componenti, un multiplexer ed un demultiplexer:



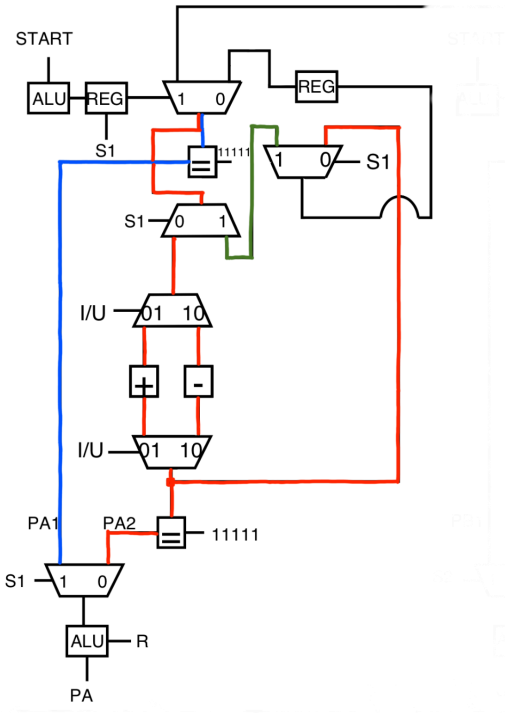
Questa parte si occupa di gestire l'intero circuito:

- il multiplexer commuta soltanto se inserita la codifica di accensione '11111' nel momento in cui il selettore start è a 1. A quel punto rimane attiva per tutto il ciclo di funzionamento, e tornerà allo stato di riposo soltanto quando il selettore sarà a 0.
- Una volta fatto lo switch, i bit passano al primo demultiplexer che ha il compito di suddividere il circuito in tre macro parti perchè necessitano di lavorare separatamente dato che gestiscono dati differenti.
  Se il selettore dovesse inviare la codifica 01 verrà messa in funzione la parte del circuito che comanda il parcheggio A, 10 per B e 11 per C.

3.5 Settori A & B

I settori A e B hanno entrambi 31 posti massimi disponibili, perciò, oltre alla nomenclatura dei componenti e I/U, il circuito di B possiamo dire sia uguale a quello di A.
Per questo motivo ci soffermeremo a capire il funzionamento di A dato che lo stesso discorso vale per B.

3.5.1 Analisi circuito A:



```
.model SA
.inputs X0 X1 X2 X3 X4 S START r
.outputs PA


.subckt Alu I=START O=ALU
.subckt Reg I=ALU O=REG S=S
.subckt MuxA I0=D0 I1=D1 I2=D2 I3=D3 I4=D4 X0=X0 X1=X1 X2=X2 X3=X3 X4=X4 S=REG O0=A0 O1=A1 O2=A2 O3=A3 O4=A4

.subckt Comp PA=PA1 I0=A0 I1=A1 I2=A2 I3=A3 I4=A4 X0=V1 X1=V1 X2=V1 X3=V1 X4=V1

.subckt DemuxS I0=A0 I1=A1 I2=A2 I3=A3 I4=A4 S=S A0=C0 A1=C1 A2=C2 A3=C3 A4=C4 B0=B0 B1=B1 B2=B2 B3=B3 B4=B4


.subckt demuxIO I0=B0 I1=B1 I2=B2 I3=B3 I4=B4 S1=X0 S2=X1 A0=Y0 A1=Y1 A2=Y2 A3=Y3 A4=Y4 B0=Z0 B1=Z1 B2=Z2 B3=Z3 B4=Z4
.subckt Sommatore A0=Y0 A1=Y1 A2=Y2 A3=Y3 A4=Y4 B0=V0 B1=V0 B2=V0 B3=V0 B4=V1 CIN=V0 COUT=COUT O0=N0 O1=N1 O2=N2 O3=N3 O4=N4
.subckt Sottrattore A0=Z0 A1=Z1 A2=Z2 A3=Z3 A4=Z4 B0=V0 B1=V0 B2=V0 B3=V0 B4=V1 BIN=V0 BOUT=BOUT O0=M0 O1=M1 O2=M2 O3=M3 O4=M4
.subckt Muxio I0=N0 I1=N1 I2=N2 I3=N3 I4=N4 X0=M0 X1=M1 X2=M2 X3=M3 X4=M4 S1=X0 S2=X1 O0=P0 O1=P1 O2=P2 O3=P3 O4=P4


.subckt MuxA I0=P0 I1=P1 I2=P2 I3=P3 I4=P4 S=S X0=C0 X1=C1 X2=C2 X3=C3 X4=C4 O0=Q0 O1=Q1 O2=Q2 O3=Q3 O4=Q4

.subckt Registro5bit A0=Q0 A1=Q1 A2=Q2 A3=Q3 A4=Q4 OUT0=D0 OUT1=D1 OUT2=D2 OUT3=D3 OUT4=D4

.subckt Const0 C0=V0
.subckt Const1 C1=V1


.subckt Comp PA=PA2 I0=P0 I1=P1 I2=P2 I3=P3 I4=P4 X0=V1 X1=V1 X2=V1 X3=V1 X4=V1
.subckt aluout I=HA r=r O=PA
.subckt Mux1bit I0=PA2 I1=PA1 S=S O0=HA
```

Il circuito è diviso principalmente in due parti:

1. Controllo primi cicli di clock (inserimento posti/percorso blu);
2. Anello chiuso per funzionamento autonomo (ingressi e uscite auto/percorso rosso e verde);

Cominciamo col dire che il primo multiplexer in cima al circuito, comanda queste due parti. Esso è controllato da un registro ed una alu.
La alu riceve in ingresso il bit di start, che viene negato per poi mandarlo in ingresso al registro comandato da S1 facendo così commutare il multiplexer.
S1 è il selettore identificativo per la parte di circuito A, così come S2 per B ed S3 per C. Questo bit inizialmente è a 1, dunque il multiplexer senza commutare manda la codifica in ingresso ad un comparatore e ad un demultiplexer:

- Il valore viene comparato con la codifica a 5 bit 11111 (numero massimo di posti consentiti), se dovesse essere uguale darà in uscita 1 (posti pieni) altrimenti 0. Questo valore poi andrà in ingresso ad un multiplexer che invierà direttamente questo valore in uscita, perché anch'esso comandato da S1.
- La stessa codifica come abbiamo detto arriva in ingresso anche a un demultiplexer, sempre comandato da S1. Il valore seguirà il percorso verde (vedere circuito) fino a tornare nel multiplexer principale.

Il multiplexer abbiamo detto che commuta se il selettore S1 va a 0, ed andrà a 0 se una macchina volesse entrare oppure uscire. A questo punto inizierebbe il funzionamento autonomo creandosi così in circuito chiuso.

Una volta commutato a 0, si prosegue col percorso rosso, e la nostra codifica entrerà così in un demultiplexer che tramite un selettore determina se fare una somma di 1 oppure una sottrazione di 1. Verrà sommato 1 se il selettore avrà la codifica 01, e sottratto 1 quando la codifica del selettore sarà 10.

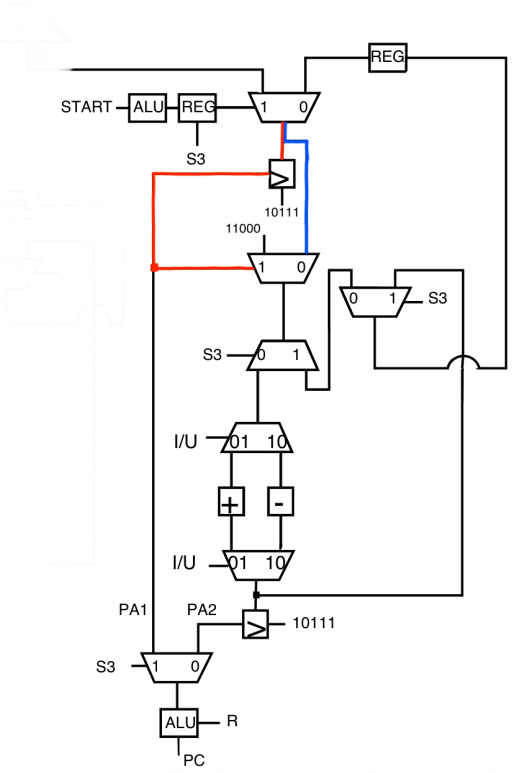
La codifica in output andrà ad un multiplexer “antagonista” sempre comandato dallo stesso selettore, per avere così soltanto un output in uscita con la codifica aggiornata (somma o sottrazione).
Questo valore poi viene innanzitutto reimmesso nel circuito per creare una costante ciclica con lo stesso valore di posti sempre aggiornato. Poi va anche in ingresso ad un comparatore che compara anch'esso con 11111.
Il valore in uscita, 0 (ancora libero), 1 (parcheggio pieno), lo riceve multiplexer ‘finale’, che manderà questo bit in uscita.

Rimane soltanto un componente però tra quest’ ultimo multiplexer e l’effettiva uscita finale, ovvero una alu, descritta in precedenza.
Essa è comandata da un selettore ‘r’, e se quest'ultimo assumerà il valore 1, l’output sarà sempre 0, altrimenti restituisce quello ricevuto in ingresso.

3.6 Settore C

Il settore C differenzia dagli altri due perché appunto può contenere al massimo 24 macchine.
Ciò significa che vanno sostituiti i comparatori di uguaglianza con comparatori di maggioranza e bisognerà aggiungere una parte di circuito che blocca l’inserimento di auto superiori alle 24, per il resto, il circuito rimane lo stesso.

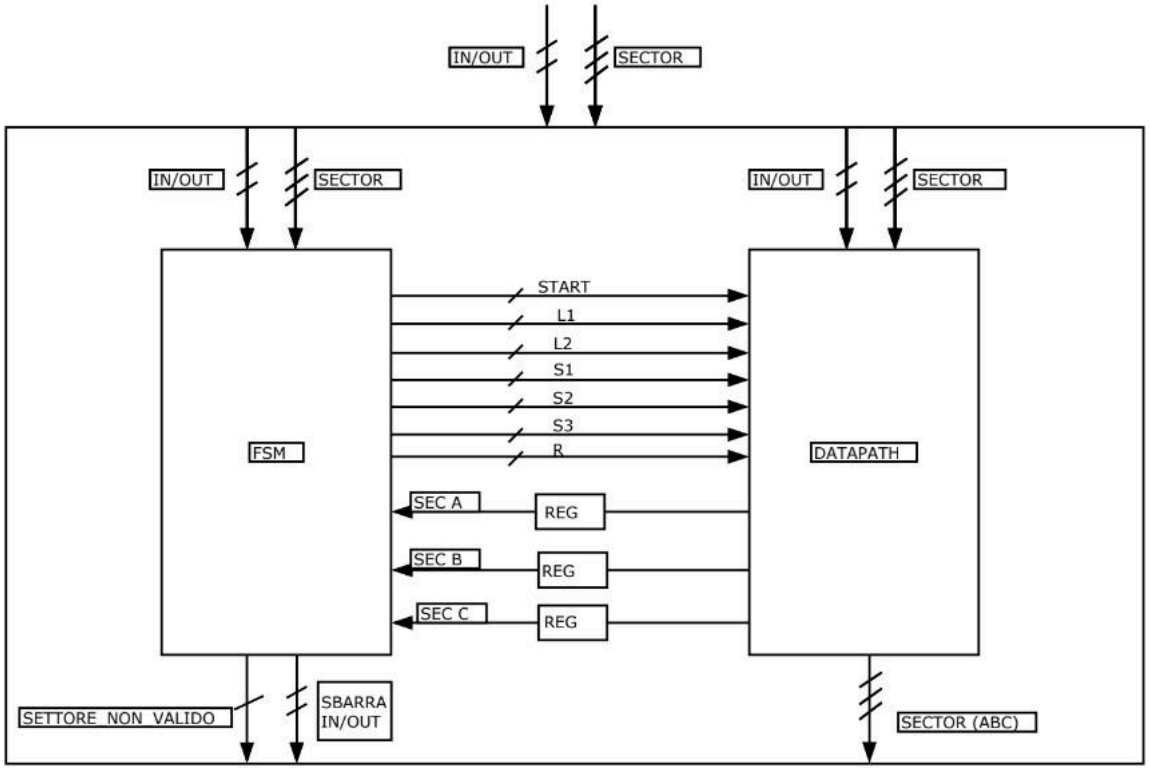
3.6.1 Analisi circuito C:



```
.model SC
.inputs X0 X1 X2 X3 X4 S START r
.outputs PC


.subckt Alu I=START O=ALU
.subckt Reg I=ALU S=S O=REG
.subckt MuxA I0=D0 I1=D1 I2=D2 I3=D3 I4=D4 X0=X0 X1=X1 X2=X2 X3=X3 X4=X4 S=REG O0=A0 O1=A1 O2=A2 O3=A3 O4=A4

.subckt Maggiore M=PA1 A4=A0 A3=A1 A2=A2 A1=A3 A0=A4 B4=V1 B3=V0 B2=V1 B1=V1 B0=V1

.subckt MuxA I0=A0 I1=A1 I2=A2 I3=A3 I4=A4 X0=V1 X1=V1 X2=V0 X3=V0 X4=V0 S=PA1 O0=H0 O1=H1 O2=H2 O3=H3 O4=H4

.subckt DemuxS I0=H0 I1=H1 I2=H2 I3=H3 I4=H4 S=S A0=C0 A1=C1 A2=C2 A3=C3 A4=C4 B0=B0 B1=B1 B2=B2 B3=B3 B4=B4


.subckt demuxIO I0=B0 I1=B1 I2=B2 I3=B3 I4=B4 S1=X0 S2=X1 A0=Y0 A1=Y1 A2=Y2 A3=Y3 A4=Y4 B0=Z0 B1=Z1 B2=Z2 B3=Z3 B4=Z4
.subckt Sommatore A0=Y0 A1=Y1 A2=Y2 A3=Y3 A4=Y4 B0=V0 B1=V0 B2=V0 B3=V0 B4=V1 CIN=V0 COUT=COUT O0=N0 O1=N1 O2=N2 O3=N3 O4=N4
.subckt Sottrattore A0=Z0 A1=Z1 A2=Z2 A3=Z3 A4=Z4 B0=V0 B1=V0 B2=V0 B3=V0 B4=V1 BIN=V0 BOUT=BOUT O0=M0 O1=M1 O2=M2 O3=M3 O4=M4
.subckt Muxio I0=N0 I1=N1 I2=N2 I3=N3 I4=N4 X0=M0 X1=M1 X2=M2 X3=M3 X4=M4 S1=X0 S2=X1 O0=P0 O1=P1 O2=P2 O3=P3 O4=P4


.subckt MuxA I0=P0 I1=P1 I2=P2 I3=P3 I4=P4 S=S X0=C0 X1=C1 X2=C2 X3=C3 X4=C4 O0=Q0 O1=Q1 O2=Q2 O3=Q3 O4=Q4

.subckt Registro5bit A0=Q0 A1=Q1 A2=Q2 A3=Q3 A4=Q4 OUT0=D0 OUT1=D1 OUT2=D2 OUT3=D3 OUT4=D4

.subckt Const0 C0=V0
.subckt Const1 C1=V1


.subckt Maggiore M=PA2 A4=P0 A3=P1 A2=P2 A1=P3 A0=P4 B4=V1 B3=V0 B2=V1 B1=V1 B0=V1
.subckt aluout I=HC r=r O=PC
.subckt Mux1bit I0=PA2 I1=PA1 S=S O0=HC
```

Come abbiamo detto, il circuito del settore C anziché utilizzare due comparatori di uguaglianza, ne userà due di maggioranza.
Maggiore di 10111, ovvero maggiore di 23. Se la comparazione risulta essere quindi 11000 o più, l'output sarà 1.
Essendo che la codifica per definire il numero 24, non utilizza a pieno tutti e cinque i bit, si potrebbe incorrere nella problematica di avere più di 24 auto in un settore che ne può contenere al massimo 24 e quindi anche errori di calcolo.
Per provvedere a ciò, è stato aggiunto un multiplexer che 'blocca' codifiche superiori a 11000.
Il multiplexer ha un ingresso con costante 11000, un altro ingresso dove arriva la nostra codifica ed un selettore.
Il selettore sarebbe l'uscita del comparatore di maggioranza.
Il multiplexer è stato programmato in modo tale che se il selettore è 1 (quindi >23), restituisco come output la serie costante di 5 bit 11000 altrimenti se il selettore è a 0 mando in uscita la mia codifica in ingresso essendo sicuro sia <24.
Per il resto il circuito rimane tale e quale ai due precedenti.

4 FSMD:



4.1 Unione tra FSM e Datapath

L'FSMD non è altro che l'unione tra la parte di controllo ed elaborazione.
A livello di programmazione vengono richiamate queste due macro parti in un unico file .blif ed allacciate uscite ed ingressi precedentemente progettati per mettere in comunicazione le due parti.
Fatto questo e definiti gli ingressi e le uscite avremo ciò:

```
.model fsmd
.inputs IN OUT SECTOR1 SECTOR2 SECTOR3
.outputs SETTORE_NON_VALIDO SBARRA_IN SBARRA_OUT SECTOR_A SECTOR_B SECTOR_C


.subckt FSM out=IN in=OUT sa=SECTOR1 sb=SECTOR2 sc=SECTOR3 pa=pa pb=pb pc=pc err=SETTORE_NON_VALIDO sb.in=SBARRA_IN sb.out=SBARRA_OUT s1=s1 s2=s2 s3=s3 l1=L1 l2=L2 start=start r=r

.subckt datapath V0=IN V1=OUT V2=SECTOR1 V3=SECTOR2 V4=SECTOR3 S1=s1 S2=s2 S3=s3 START=start L1=L1 L2=L2 PA=SECTOR_A PB=SECTOR_B PC=SECTOR_C r=r


.subckt registro1 A=SECTOR_A OUT=pa
.subckt registro1 A=SECTOR_B OUT=pb
.subckt registro1 A=SECTOR_C OUT=pc


.search FSM.blif
.search datapath.blif

.end
```

Nota. Non è bastato soltanto allacciare ingressi e uscite tra loro, perché ci siamo ritrovati di fronte a un warning di ciclo continuo. Per risolvere il problema, sono stati aggiunti tre registri tra Datapath ed FSM perchè non non si poteva fare un collegamento diretto.

4.2 Ottimizzazione circuito

Terminato il lavoro e simulato il circuito per constatarne il funzionamento, si è passato all'ottimizzazione del circuito, ottenuta attraverso il richiamo di uno script chiamato *script.rugged*, che contiene un insieme di funzioni volte a semplificare nodi, registri, tempi e area del circuito.
Riportiamo le differenza tra il prima e il dopo:

```
sis> print_stats
FSMD            pi= 5   po= 9   nodes=269       latches=24
lits(sop)=1518
```

```
sis> source script.rugged
sis> write_blif FSMD.blif
sis> print_stats
FSMD            pi= 5   po= 9   nodes=119       latches=24
lits(sop)= 499
```

Come possiamo vedere, i letterali e i nodi sono diminuiti drasticamente.

4.3 Mapping

Conclusa l'FSMD e fatte le dovute simulazioni, si procede a crearne una copia per lanciare i comandi del mapping per visualizzarne le statistiche.
Prima di ciò descriviamo brevemente cosa significa mappare un circuito.
Mappare un circuito significa descrivere le funzionalità hardware della FPGA (field programmable gate array).
Il mapping è un'operazione che va eseguita dopo l'ottimizzazione del circuito, in quanto potrebbe portare a gravi errori se svolta prima, perché l'hardware non sarebbe più modificabile.

Richiamando la libreria *synch.genlib*, attraverso i comandi *map -m 0* e *map -n 1* vengono rispettivamente ridotti area e ritardo complessivo.

Mapping per area:

```
sis> print_stats
FSMDmap         pi= 5   po= 9   nodes=277       latches=24
lits(sop)= 664
```

Mapping per ritardo:

```
sis> print_stats
FSMDmap         pi= 5   po= 9   nodes=480       latches=24
lits(sop)= 888
```

Evinciamo dunque che conviene il mapping per area in quanto accresce meno le statistiche del circuito.

5 CONCLUSIONI:

5.1 Scelte progettuali

I segnali che mettono in comunicazione il controllore e l'elaboratore evidenziano come quest'ultimo, progettato da zero, rappresenti il vero punto cardine delle scelte personali adottate nella realizzazione del progetto.

Un'altra scelta progettuale è stata quella di mettere a 0 tutti e cinque i bit di uscita dopo lo spegnimento della macchina.
Abbiamo scelto di procedere in questo modo perché la simulazione presumeva uno spegnimento, dunque non sembrava consono lasciare alzati i bit delle sbarre e quelli dei settori ancora pieni.

5.2 Riepilogo del lavoro

Il lavoro è stato impegnativo ma anche molto gratificante.
La parte critica per noi è stato il datapath, perché ha presentato diversi problemi, tra cui cicli continui, sezioni del circuito non sincronizzate correttamente e perdite di dati.
Tuttavia, affrontando ogni difficoltà con dedizione, siamo riusciti a risolverle tutte e a comprendere sempre meglio la logica del sistema.
In generale ci riteniamo molto soddisfatti del lavoro svolto, che ci ha permesso di arricchire le nostre conoscenze, inizialmente soltanto teoriche.