Architettura degli Elaboratori

Relazione Laboratorio SIS

A cura di:
Alessio Zattoni (Matr. VR457153), Joni Mae Rosales (Matr. VR456121), Angelo Borgonovi (Matr. VR457071)

A.A. 2020/2021

"Progettazione di un circuito bancomat"

INDICE

TRACCIA

Si progetti il circuito sequenziale che controlla l'erogazione di denaro di un bancomat. Il circuito ha 4 ingressi nel seguente ordine:

- `BANCOMAT_INSERITO`(1 bit)
- `CODICE(`4 bit)
- `CASH_RICHIESTO`(10 bit)
- `CASH_DISPONIBILE`(16 bit)

Gli output sono i seguenti e devono seguire il seguente ordine:

- `REINSERIRE_CODICE`(1 bit)
- `ABILITAZIONE_EROGAZIONE`(1 bit)
- `BLOCCO_BANCOMAT`(1 bit)
- `CASH_DA_EROGARE`(10bit)

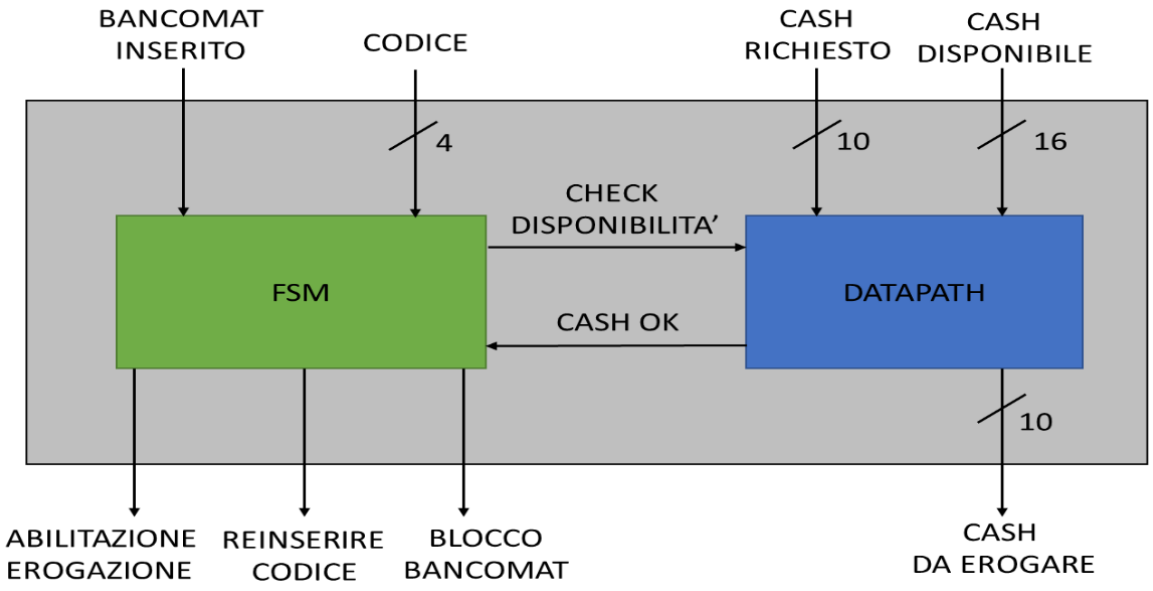
Input e output devono essere definiti nell'ordine sopra specificato (da sinistra verso destra). Il meccanismo è guidato come segue:

- Il segnale di ingresso BANCOMAT_INSERITO (da considerare derivante da un circuito esterno che rileva la presenza di un bancomat valido inserito nel macchinario) se uguale a 1 abilita la codifica dei numeri inseriti tramite il segnale di ingresso CODICE. Se uguale a 0, disabilita (pone a zero) tutte le uscite del circuito.
- Una volta che il bancomat è stato inserito, viene inserito nel circuito il codice di autenticazione tramite il segnale di ingresso `CODICE` composto da 3 numeri inseriti in 3 istanti consecutivi, con range 0..9, codificati quindi con 4 bit.
- Una volta accertato che la sequenza numerica corrisponde a 5 5 0, il circuito riceve l'ammontare del cash richiesto dall'ingresso `CASH_RICHIESTO` (ammontare da 0 a 1023 euro, codificato con 10 bit) e attiva il controllo della disponibilità di banconote nella cassaforte, tramite il segnale interno `CHECK_DISPONIBILITA`(1bit). Il controllo verifica se il cash richiesto è

inferiore a ¼ del cash disponibile in cassaforte, quest'ultimo ricevuto dal segnale di ingresso `CASH_DISPONIBILE`. Se inferiore allora il circuito abilita il segnale interno `CASH_OK`, il quale fa abilitare il segnale di uscita `ABILITAZIONE_EROGAZIONE`, e riporta sul segnale di uscita `CASH_DA_EROGARE` l'importo richiesto. Altrimenti, tutti questi segnali rimangono posti a 0.

- Se il codice viene inserito in modo errato, il circuito abilita l'uscita `REINSERIRE_CODICE`. Se il codice viene inserito in modo errato per 3 volte consecutive, il circuito abilita l'uscita `BLOCCO_BANCOMAT`.

Lo schema generale del circuito deve rispettare la FSMD riportata di seguito:



ARCHITETTURA GENERALE CIRCUITO

Il circuito è composto da una fsm(fsm.blif), che svolge i controlli e un datapath(datapath.blif) che svolge i calcoli. L'insieme della fsm e datapath crea la FSMD(FSMD.blif). La comunicazione tra i due circuiti avviene attraverso segnali intermedi. Il circuito una volta mappato può essere realizzato come ASIC o FPGA.

FSM

Il controllore è rappresentato graficamente con una macchina a stati finiti(FSM) della tipologia Mealy. Gli input sono (BANCOMAT INSERITO,CODICE0, CODICE 1, CODICE 2, CODICE 3, OK). Gli output sono(REINSERIRE CODICE, ABILITAZIONE EROGAZIONE, BLOCCO BANCOMAT, CHECK DISPONIBILITÀ , CHECK BLOCCO). Di seguito verranno descritti i vari input ed output:

INPUT

- **BANCOMAT INSERITO:** Questo input indica la presenza di un bancomat inserito. Il circuito che realizza questo input è dato a priori e verrà descritto nella sezione delle scelte progettuali.

- **CODICE(0,1,2,3):** Sono 4 bit utilizzati per inserire il pin del bancomat.

- **OK:** Segnale che attesta se il controllo sulla disponibilità ha dato esito positivo.

OUTPUT

- **REINSERIRE CODICE:** Indica che il pin inserito è errato.

- **ABILITAZIONE EROGAZIONE:** Quando è a 1 indica che il denaro deve essere erogato. Quando a 0 invece non deve avvenire nessuna erogazione.

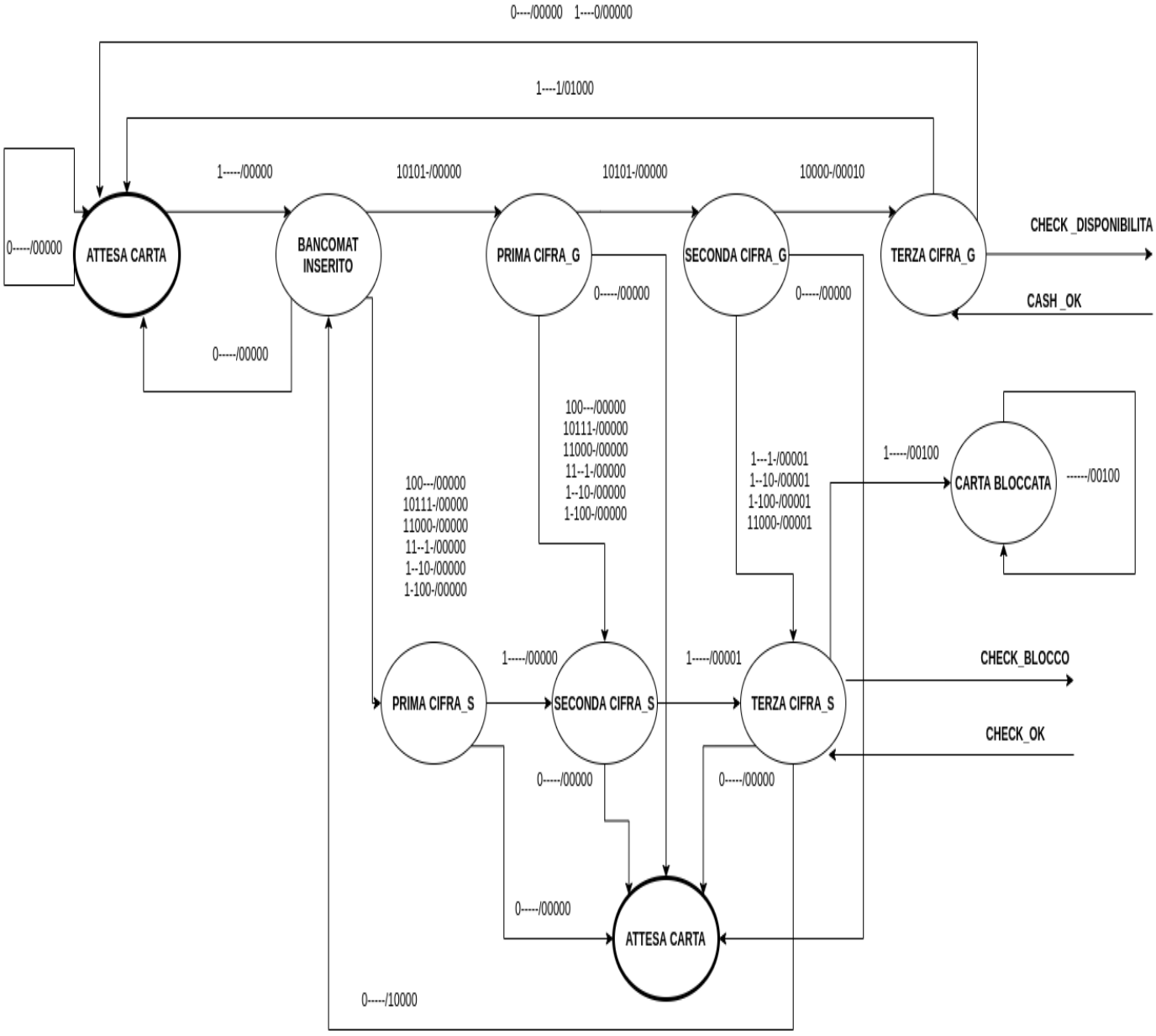
- **BLOCCO BANCOMAT:** Quando questo segnale è a 1 il bancomat deve essere bloccato.

- **CHECK DISPONIBILITÀ:** Quando questo bit è a 1, indica che deve avvenire il controllo della disponibilità.

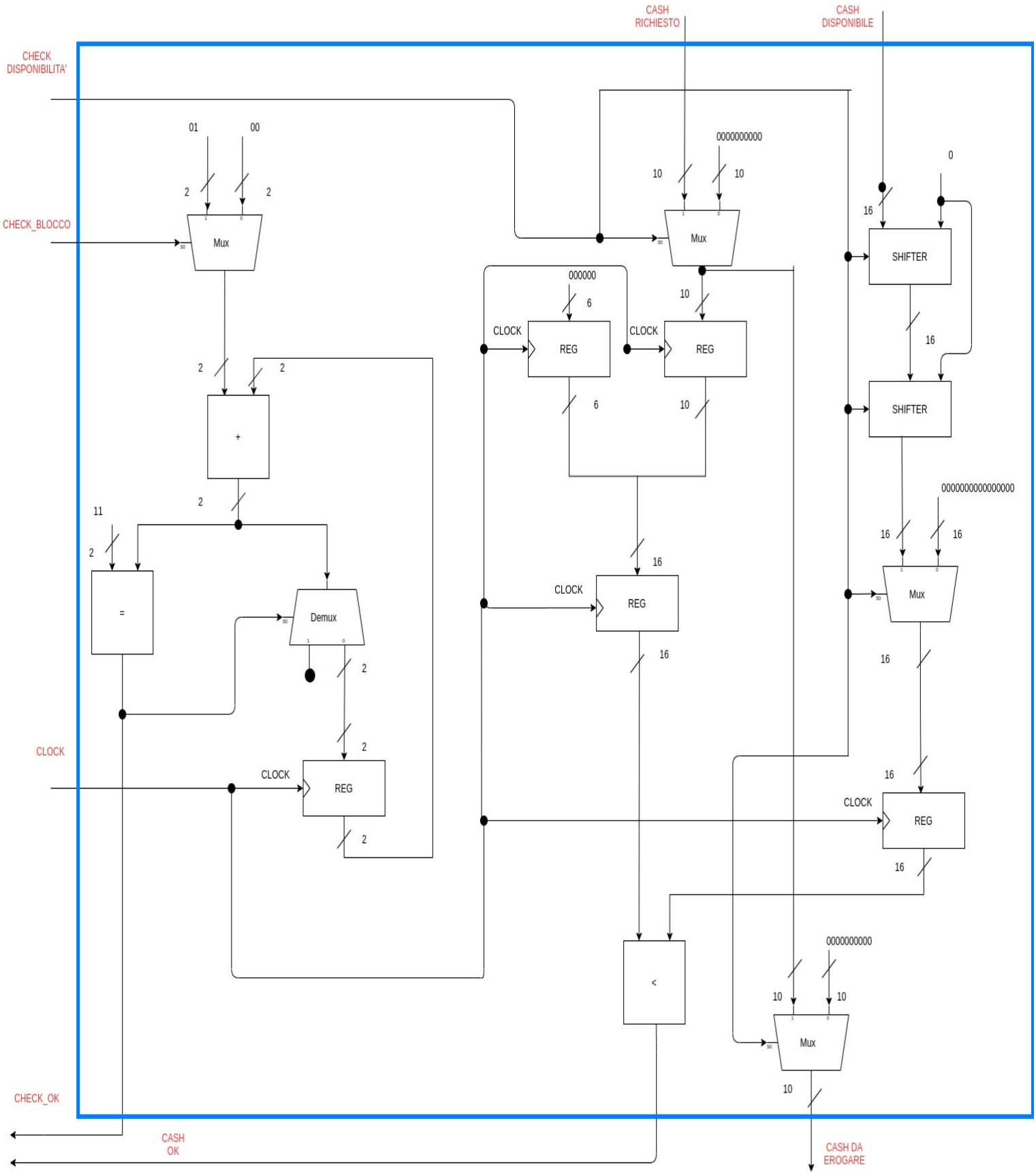
- **CHECK BLOCCO:** Quando questo bit è a 1, indica che deve avvenire il controllo sul numero di tentativi.

STG

Qui di seguito viene riportato l'STG (State Transition Graph), che rappresenta la macchina a stati finiti relativa al controllore del circuito. I segnali **check disponibilità** e **check blocco** sono indirizzati come input per il datapath. I segnali **cash ok** e **check ok** invece provengono dal datapath e servono per indicare se i due check sono venuto con successo o meno(0-1).



DATAPATH


CASH RICHIESTO
CASH DISPONIBILE
CHECK DISPONIBILITA'
CHECK_BLOCCO
01
00
2
2
Mux
0000000000
10
10
Mux
0
16
SHIFTER
000000
6
10
CLOCK
REG
CLOCK
REG
16
SHIFTER
2
2
+
2
6
10
0000000000000000
11
2
=
Demux
2
16
16
16
CLOCK
REG
Mux
16
16
CLOCK
2
2
CLOCK
REG
2
16
CLOCK
REG
16
<
0000000000
10
10
Mux
10
CHECK_OK
CASH OK
CASH DA EROGARE

COMPONENTI

DATAPATH

- 2 multiplexer 10bit.
- 1 multiplexer 16bit.
- 1 multiplexer 2bit.
- 1 demultiplexer a 2bit.
- 2 registri 16bit.
- 1 registro 6 bit.
- 1 registro 2 bit.
- 1 registro 10 bit.
- 2 shifter 16bit.
- 1 accumulatore 2 bit.
- 1 comparatore 2bit.
- 1 minore a 16bit.
- 1 costante 10bit (0000000000).
- 1 costante 16 bit (0000000000000000).
- 1 costante 6bit (000000).
- 1 costante 2bit (11).

FSMD

- 1 multiplexer 10bit.
- 1 costante 10bit(0000000000).
- Datapath.
- Fsm.

CICLO DATAPATH

Il datapath è suddiviso in due circuiti che fanno due operazioni differenti. Il primo si occupa di controllare che la cifra richiesta sia minore di ¼ della disponibilità totale (rappresentato nella parte destra). Questo circuito prende in input due segnali: **cash richiesto** (10 bit) e **cash disponibile** (16bit). Se il segnale **controllo disponibilità** è a 1, il multiplexer fa passare i 10 bit, altrimenti fa passare 10 zeri. Successivamente i bit vanno in un registro a 10 bit, per poi essere inseriti in un registro più grande da 16 bit. Per avere un valore corretto, il cablaggio prevede che le prime 6 posizioni siano collegate ad un altro registro che contiene 6 zeri, mentre le restanti 10 al registro prima citato. Questa operazione permette di passare da un numero a 10 bit ad uno a 16 bit; questa operazione ci servirà dopo per confrontare questo numero con il **cash disponibile** che è a 16 bit. Dalla parte opposta il **cash disponibile** viene immesso in uno shifter, il quale è collegato al segnale **check disponibilità**. Quando il segnale si alza lo shifter sposta i 16 bit a destra e aggiunge uno 0 in testa. Questa operazione viene effettuata per ben due volte, simulando l'operazione di moltiplicazione per 0,25(1/4). L'output del secondo shifter è collegato ad un multiplexer. Quando quest'ultimo riceve il segnale **check disponiblità** a 1 fa passare il valore dello shifter, altrimenti produce 16 zeri. Il valore viene inserito in un registro. I due output così prodotti vengono passati in input ad un circuito che controlla se il primo valore (**cash richiesto**) è minore del secondo(**cash disponibile**). Se è vero il segnale **cash ok** vale 1 sennò resta a 0. Questo segnale servirà successivamente alla fsm per attivare l'erogazione. Inoltre contemporaneamente in un altro multiplexer viene inserito nel ingresso 1 il **cash richiesto**, mentre nel ingresso 0 tutti 0. Se il segnale di selezione vale 1 verrà erogato il **cash richiesto**, altrimenti non viene erogato nulla. Il secondo circuito tiene traccia del numero di tentativi, quando arriva a 3 tentativi consecutivi il circuito invia un segnale alla fsm per il blocco della carta. Questo circuito è composto da un multiplexer, che riceve come segnale di selezione **check blocco**. Se vale 0 fa passare il valore 00, altrimenti il valore 01. Successivamente l'output del multiplexer andrà in un accumulatore. Questo accumulatore sommerà al valore residuo, il valore passato dal multiplexer. L'output del accumulatore viene diviso in due percorsi(fan-out). Il primo va ad un comparatore, il quale controllerà che l'input ricevuto sia uguale a 11(3), così facendo si possono controllare il numero di tentativi. Se è vero, il segnale **check ok** viene messo a 1 altrimenti rimane a 0. Questo segnale verrà utilizzato dalla fsm per bloccare o meno il bancomat. Il secondo percorso va collegato ad un demultiplexer, il quale ha come segnale di selezione l'output del comparatore. Così facendo se vale 0, il valore viene memorizzato in un registro per un successivo accumulo, altrimenti se vale 1 il valore viene resettato(l'uscita non è collegata), quindi viene passato il valore 0.

MINIMIZZAZIONE CIRCUITO

FSM

Minimizzazione degli stati

Per prima cosa carichiamo il file e proviamo a minimizzare gli stati con il comando **state_minimize stamina.** La procedura di minimizzazione degli stati non ha portato miglioramenti. Prima ci sono 9 stati, dopo ce ne sono ancora 9. Quindi si riesce a dedurre che non ci siano stati equivalenti.

Prima:

```
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$ sis
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif fsm.blif
sis> print_stats
FSM_BANCOMAT     pi= 6   po= 5   nodes=  5       latches= 0
lits(sop)=    0  #states(STG)=   9
```

Dopo:

```
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$ sis
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif fsm.blif
sis> state_minimize stamina
Running stamina, written by June Rho, University of Colorado at Boulder
Number of states in original machine : 9
Number of states in minimized machine : 9
```

Codifica degli stati

La codifica degli stati viene effettuata con il comando **state_assign jedi,** il quale assegna in maniera automatica la codifica degli stati. Successivamente tramite il comando write_blif <nome file>.blif si salva la fsm codificata.

Assegnazione codifica:

```
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$ sis
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif fsm.blif
sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> write_blif fsm.blif
```

Codifica:

```
.code ATTESA_CARTA 1111
.code BANCOMAT_INSERITO 1101
.code PRIMA_CIFRA_G 0001
.code PRIMA_CIFRA_S 1001
.code SECONDA_CIFRA_G 0011
.code SECONDA_CIFRA_S 1011
.code TERZA_CIFRA_G 1000
.code TERZA_CIFRA_S 1010
.code CARTA_BLOCCATA 1110
```

Minimizzazione letterali

In questa fase si cerca di minimizzare il numero di letterali tramite lo script rugged. La minimizzazione porta da 79 letterali in forma di somma di prodotti a 40. Un guadagno di 39 letterali. Inoltre i nodi da 9 sono passati a 8.

Prima:

```
sis> print_stats
FSM_BANCOMAT     pi= 6   po= 5   nodes=  9       latches= 4
lits(sop)=  79  #states(STG)=   9
```

Dopo:

```
sis> source script.rugged
sis> print_stats
FSM_BANCOMAT    pi= 6   po= 5   nodes=  8       latches= 4
lits(sop)=  40  #states(STG)=   9
```

DATAPATH

Minimizzazione letterali

Per prima cosa si deve caricare la rete con il comando **read_blif,** successivamente tramite lo script rugged si effettua la minimizzazione. Dopo la minimizzazione c'è stato un guadagno massivo, infatti si è passati da 1396 letterali a 87 con un guadagno di 1309 letterali. Inoltre i nodi sono passati da 146 a 28.

Prima:

```
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif datapath.blif
Warning: network `SHIFTER16BIT', node "B7" does not fanout
Warning: network `DATAPATH', node "Q0" does not fanout
Warning: network `DATAPATH', node "Q1" does not fanout
Warning: network `DATAPATH', node "[172]" does not fanout
Warning: network `DATAPATH', node "[181]" does not fanout
sis> print_stats
DATAPATH        pi=28   po=12   nodes=146       latches=34
lits(sop)=1396
```

Dopo:

```
sis> read_blif datapath.blif
Warning: network `SHIFTER16BIT', node "B7" does not fanout
Warning: network `DATAPATH', node "Q0" does not fanout
Warning: network `DATAPATH', node "Q1" does not fanout
Warning: network `DATAPATH', node "[385]" does not fanout
Warning: network `DATAPATH', node "[394]" does not fanout
sis> source script.rugged
sis> print_stats
DATAPATH        pi=28   po=12   nodes= 28       latches=26
lits(sop)=  87
```

FSMD

Minimizzazione letterali

Per prima cosa si deve caricare la rete con il comando **read_blif,** successivamente tramite lo script rugged si effettua la minimizzazione. Dopo la minimizzazione c'è stato un guadagno, infatti si è passati da 168 letterali a 60 con un guadagno di 108 letterali. Inoltre i nodi sono passati da 57 a 17.

Prima:

```
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif FSMD.blif
Warning: network `fsm.blif', node "CODICE0" does not fanout
Warning: network `fsm.blif', node "CODICE1" does not fanout
Warning: network `fsm.blif', node "CODICE2" does not fanout
Warning: network `fsm.blif', node "CODICE3" does not fanout
Warning: network `FSM_BANCOMAT', node "OK" does not fanout
Warning: network `DATAPATH', node "B14" does not fanout
Warning: network `DATAPATH', node "B15" does not fanout
Warning: network `FSMD', node "B14" does not fanout
Warning: network `FSMD', node "B15" does not fanout
Warning: network `FSMD', node "C_OK" does not fanout
Warning: network `FSMD', node "C10" does not fanout
sis> print_stats
FSMD            pi=31   po=13   nodes= 57       latches=30
lits(sop)= 168
```

Dopo:

```
sis> source script.rugged
sis> print_stats
FSMD            pi=31   po=13   nodes= 17       latches= 4
lits(sop)=  60
```

MAPPING

Il mapping viene eseguito caricando la libreria **synch.genlib,** successivamente viene mappato il circuito per area e contemporaneamente viene mostrato in output a video il risultato con il comando **map -s.** L'area è passata da 1720 a 1336, con un guadagno di 384 sull'area. Il ritardo è aumentato passando da 10.20 a 13.80. Avendo scelto un mapping che favorisce l'area, questo risultato è quello che ci aspettavamo. Dopo il mapping si può notare un aumento di letterali, questo cambiamento è ciò che ci aspettavamo dall'operazione di mapping.

Prima:

```
sis> map -s
warning: unknown latch type at node '{[463]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[464]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[465]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[466]}' (RISING_EDGE assumed)
WARNING: uses as primary input drive the value (0.20,0.20)
WARNING: uses as primary input arrival the value (0.00,0.00)
WARNING: uses as primary input max load limit the value (999.00)
WARNING: uses as primary output required the value (0.00,0.00)
WARNING: uses as primary output load the value 1.00
>>> before removing serial inverters <<<
# of outputs:          17
total gate area:       1720.00
maximum arrival time: (10.20,10.20)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-10.20,-10.20)
total neg slack:      (-155.80,-155.80)
# of failing outputs:  17
>>> before removing parallel inverters <<<
# of outputs:          17
total gate area:       1720.00
maximum arrival time: (10.20,10.20)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-10.20,-10.20)
total neg slack:      (-155.80,-155.80)
# of failing outputs:  17
# of outputs:          17
total gate area:       1720.00
maximum arrival time: (10.20,10.20)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-10.20,-10.20)
total neg slack:      (-155.80,-155.80)
# of failing outputs:  17
```

Dopo:

```
sis> read_library synch.genlib
sis> map -s
warning: unknown latch type at node '{[4390]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[4391]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[4392]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[4393]}' (RISING_EDGE assumed)
WARNING: uses as primary input drive the value (0.20,0.20)
WARNING: uses as primary input arrival the value (0.00,0.00)
WARNING: uses as primary input max load limit the value (999.00)
WARNING: uses as primary output required the value (0.00,0.00)
WARNING: uses as primary output load the value 1.00
>>> before removing serial inverters <<<
# of outputs:           17
total gate area:        1336.00
maximum arrival time: (13.80,13.80)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-13.80,-13.80)
total neg slack:      (-128.80,-128.80)
# of failing outputs:  17
>>> before removing parallel inverters <<<
# of outputs:           17
total gate area:        1336.00
maximum arrival time: (13.80,13.80)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-13.80,-13.80)
total neg slack:      (-128.80,-128.80)
# of failing outputs:  17
# of outputs:           17
total gate area:        1336.00
maximum arrival time: (13.80,13.80)
maximum po slack:     (-4.80,-4.80)
minimum po slack:     (-13.80,-13.80)
total neg slack:      (-128.80,-128.80)
# of failing outputs:  17
```

Statistiche dopo il mapping:

```
sis> print_stats
FSMD            pi=31   po=13   nodes= 40       latches= 4
lits(sop)=  80
```

RIDUZIONE RITARDI

Successivamente abbiamo provato a ridurre il cammino critico tramite il comando **reduce_depth**, ottenendo un miglioramento sul ritardo con conseguente aumento del numero di letterali. Il ritardo da 13.80 è arrivato a 12.40. L'aera da 1336 è diventata 1584. Facendo un confronto, un aumento di 248 letterali comparato con una diminuzione di 1.4 secondi, ci fa capire che il circuito migliore è quello minimizzato per area, ponendo vincoli sul ritardo. Questa scelta è dovuta alla natura del circuito, dato che anche se aumenta il tempo di risposta di qualche secondo, non essendo un dispositivo in cui il tempo è essenziale (per esempio dispositivo che controlla l'uscita degli airbag), il circuito assolve il suo compito. Invece diminuendo l'area si riescono a creare circuiti più piccoli e meno costosi. In questo caso quest'ultima caratteristica è da preferire.

Riduzione cammino critico:

```
sis> reduce_depth
sis> map -s
>>> before removing serial inverters <<<
# of outputs:          17
total gate area:       1584.00
maximum arrival time: (12.40,12.40)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-12.40,-12.40)
total neg slack:      (-126.40,-126.40)
# of failing outputs:  17
>>> before removing parallel inverters <<<
# of outputs:          17
total gate area:       1584.00
maximum arrival time: (12.40,12.40)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-12.40,-12.40)
total neg slack:      (-126.40,-126.40)
# of failing outputs:  17
# of outputs:          17
total gate area:       1584.00
maximum arrival time: (12.40,12.40)
maximum po slack:     (-5.80,-5.80)
minimum po slack:     (-12.40,-12.40)
total neg slack:      (-126.40,-126.40)
# of failing outputs:  17
```

SIMULAZIONE CIRCUITO

Qui di seguito viene riportata una simulazione del circuito, sia in fase di erogazione, sia in fase di blocco. Con un unico script **(collaudofsmd.script)** tramite il comando source <nome script>.estensione, abbiamo simulato il funzionamento del circuito.

SCRIPT COLLAUDOFSMD.SCRIPT

 # inserisco il pin giusto, cash disponibile e cash richiesto
sim 1 0 0 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0
sim 1 0 1 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0
sim 1 0 1 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0
sim 1 0 0 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
#inserisco il pin sbagliato la prima volta
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
#inserisco il pin sbagliato una seconda volta
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
#inserisco il pin sbagliato una terza volta(negli ultimi 3 stati sono in carta bloccata)
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0
sim 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0

Simulazione circuito:

```
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$
(base) alessio@allen93:~/Scrivania/sis elaborato/SIS$ sis
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif FSMD.blif
Warning: network `fsmc.blif', node "CODICE0" does not fanout
Warning: network `fsmc.blif', node "CODICE1" does not fanout
Warning: network `fsmc.blif', node "CODICE2" does not fanout
Warning: network `fsmc.blif', node "CODICE3" does not fanout
Warning: network `FSM_BANCOMAT', node "OK" does not fanout
Warning: network `SHIFTER16BIT', node "B7" does not fanout
Warning: network `DATAPATH', node "Q0" does not fanout
Warning: network `DATAPATH', node "Q1" does not fanout
Warning: network `DATAPATH', node "[202]" does not fanout
Warning: network `DATAPATH', node "[211]" does not fanout
Warning: network `FSMD', node "C_OK" does not fanout
Warning: network `FSMD', node "C10" does not fanout
Warning: network `FSMD', node "Q0" does not fanout
Warning: network `FSMD', node "Q1" does not fanout
Warning: network `FSMD', node "[202]" does not fanout
Warning: network `FSMD', node "[211]" does not fanout
sis> source collaudofsmd.script

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 11010000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 00010000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 00110000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 10000000000000000000000000000000000000

Network simulation:
Outputs: 0 1 0 0 1 1 1 0 0 0 0 1 0
Next state: 11110000000111000010000000000111110100

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 11010000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 10010000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 10110000000000000000000000000000000000

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 10100000000000000000000000000000000000

Network simulation:
Outputs: 1 0 0 0 0 0 0 0 0 0 0 0
```

```
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1001000000000000000000000000000000000001

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1011000000000000000000000000000000000001

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1010000000000000000000000000000000000001

Network simulation:
Outputs: 1 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1101000000000000000000000000000000000010

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1001000000000000000000000000000000000010

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1011000000000000000000000000000000000010

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 1010000000000000000000000000000000000010

Network simulation:
Outputs: 0 0 1 0 0 0 0 0 0 0 0 0 0
Next state: 1110000000000000000000000000000000000000

Network simulation:
Outputs: 0 0 1 0 0 0 0 0 0 0 0 0 0
Next state: 1110000000000000000000000000000000000000

Network simulation:
Outputs: 0 0 1 0 0 0 0 0 0 0 0 0 0
Next state: 1110000000000000000000000000000000000000
```

Il quinto stato dall'alto è lo stato in cui viene erogato il denaro. Successivamente, nel 10° stato si alza il primo bit a 1, quindi il circuito ha rilevato una combinazione errata. Nel 15° stato dall'altro viene rilevato un secondo tentativo di inserimento errato. Il terzultimo stato ha rilevato per la terza volta una combinazione sbagliata, quindi si alza il bit di blocco bancomat. I due stati successivi indicano che ora la macchina si trova nello stato carta bloccata (loop), quindi il bit di blocco della carta rimane altro.

Note:
I warning rilevati dopo la lettura dei vari file non influiscono con il funzionamento, dato che si riferiscono ad uscite non collegate.

SCELTE PROGETTUALI

- Nella fsm, le transizioni tra i vari stati avvengono solamente se il bancomat è inserito(segnale a 1), sennò se vale 0 in qualunque stato ci si trovi, il circuito rimanda allo stato di reset(**attesa carta**).

- Nella fsm, quando si entra nello stato **carta bloccata,** il bit di blocco rimane a 1 per indicare il bancomat bloccato. Inoltre in questo stato si entra in un loop, fino a quando il tecnico non sbloccherà la carta.

- Nel datapath l'erogazione del denaro avviene nella transizione tra lo stato **terza cifra giusta** e **attesa carta**.

- Nel datapath il demultiplexer nell'uscita 1 produce 00(valore di default). In sis l'uscita non è stata collegata con un altro componente, dato che se sis non rileva il valore assume 0 di default.

- Nella FSMD la parte del blocco bancomat è gestita dal circuito che rileva la carta(questo circuito ci è stato dato a priori). Se tale circuito rileva che il blocco bancomat è a 1, allora abbassa il bit di **bancomat inserito**,così facendo si va nello stato **carta bloccata** e il bit di **blocco bancomat** rimane alto. Se invece il bit è a 0, il bancomat rimane a 1 e lo stato successivo sarà **bancomat inserito**, in questo caso viene alzato il bit **reinserire codice**.

- Dato che il segnale **bancomat inserito** è un segnale che viene richiesto come input, non è stato possibile realizzare il circuito che implementi la produzione automatica di tale segnale, con la relativa parte del blocco bancomat. Tale circuito può essere costruito tramite un multiplexer che ha come segnale di selezione il bit **blocco bancomat.** Se il bit è a 0, bancomat inserito rimane a 1. Se blocco bancomat è a 1 invece, bancomat inserito va a 0 facendo uscire la carta(i valori 1 e 0 sono i due ingressi del multiplexer).

- Nel datapath, nella parte che serve per fare il check della disponibilità, sono stati inseriti dei registri per mantenere uno storico dei movimenti delle varie operazioni. Così facendo si riesce a mantenere in memoria il prelievo precedente a quello attuale(solo se lo si memorizza in un registro a parte che non si azzera).