POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE

LAUREA MAGISTRALE IN INGEGNERIA DELL'AUTOMAZIONE

DIGYTAL PROGRAMMABLE SYSTEMS

Prof. Ing. Francesco De Leonardis

Sintesi e analisi del circuito digitale di un ATM (Automated Teller Machine) mediante Quartus II e ModelSim

D'ALESSANDRO Vito Ivano
VENEZIA Antonio

ANNO ACCADEMICO 2019-2020

Sommario

Con questo lavoro si è voluto realizzare il circuito digitale a bordo di un ATM bancario sfruttando software di simulazione quali Quartus II e ModelSim di cui l'azienda Intel è proprietaria dal 28 Dicembre 2015 (in precedenza i due software erano proprietari dell'azienda Altera). In prima istanza si è focalizzata l'attenzione sulla sintesi di una FSM (Finite State Machine) che permettesse di verificare il corretto inserimento della password neccessaria all'accesso al conto corrente. Le tipiche operazioni bancarie effettuate dall'utente quali prelievo e versamento di contante sono state simulate realizzando un sommatore/sottrattore. Il riscontro di ogni operazione effettuata dall'utente viene segnalato attraverso l'accensione di opportuni led.

Indice

Elenco delle figure **3**

Elenco delle tabelle **4**

1 Introduzione **5**

2 Principio di funzionamento **6**

3 Sintesi del circuito digitale **9**
3.1 Macchina a stati finiti 9
3.2 Divisore di freqeunza 10
3.3 Circuito di debouncing 11
3.4 Contatore 12
3.5 Display a 7 segmenti 13
3.6 Full Adder 14

4 Circuito digitale complessivo **18**
4.1 Messa in opera dell'ATM 20
4.1.1 Fase 1 - Inizializzazione 20
4.1.2 Fase 2 - Accesso al conto corrente 20
4.1.3 Fase 3 - Prelievo o versamento 20
4.1.4 Fase 4 - Ripristino 20
4.2 Risultati e discussione della simulazione effettuata in ModelSim 20
4.2.1 Caso 1: Password corretta, prelievo non consentito, prelievo consentito . . . 21
4.2.2 Caso 2: Password corretta, versamento non consentito, versamento consentito 22
4.2.3 Caso 3: Password corretta e versamento consentito 23
4.2.4 Caso 4: Password errata, password corretta e versamento consentito 24
4.2.5 Caso 5: Password errata per tre volte consecutive 25

5 TimeQuest Timing Analyzer **26**

6 Pin Planner **28**

7 Conclusioni **30**

Appendice **31**
Codice del progetto 31
Divisore di clock 38
Circuito antirimbalzo 39
Full Adder 40
Circuito sommatore/sottrattore a 4 bit 41

Elenco delle figure

1.1 Primo ATM installato in Italia 5
2.1 Schema illustrativo degli switch e push-button utilizzati 7
2.2 Diagramma di flusso del sistema realizzato 8
3.1.1 Macchina a stati finiti 9
3.2.1 Divisore di clock 10
3.2.2 Simulazione in ModelSim del divisore di frequenza 11
3.3.1 Microrimbalzo in un push-button 11
3.3.2 Schema a blocchi del circuito antirimbalzo 11
3.3.3 Simulazione del circuito antirimbalzo con divisore di frequenza 12
3.4.1 Contatore a 4 bit 13
3.5.1 Display a 7 segmenti 13
3.5.2 Mappa di Karnaugh display a 7 segmenti 13
3.5.3 Schema a blocchi display a 7 segmenti 14
3.6.1 Sommatore/sottrattore binario 15
3.6.2 Schema a blocchi del circuito sommatore/sottrattore realizzato 15
3.6.3 Simlazione del circuito sommatore/sottrattore in Modelsim 16
4.1 Schema logico di progetto dell'ATM bancario 18
4.2 Schema logico della FSM 19
4.2.1 Password corretta, prelievo non consentito, prelievo consentito 21
4.2.2 Password corretta, versamento non consentito, versamento consentito 22
4.2.3 Password corretta e versamento consentito 23
4.2.4 Password errata, password corretta e versamento consentito 24
4.2.5 Password errata per tre volte consecutive 25
5.1 Slack con clock di 50 MHz 26
5.2 Forme d'onda temporali con clock a 50 MHz 26
5.3 Slack con clock di 200 MHz 27
5.4 Slack con clock di 250 MHz 27
6.1 Vista dalll'alto della scheda FPGA considerata 28
6.2 Assegnazione dei pin ai segnali I/O 29

Elenco delle tabelle

2.1 Tabella di funzionamento dei push-button 6
3.2.1 Tabella di divisione del clock con contatore $Mod\,24$ 10
3.6.1 Tabella della verità somma binaria 14
3.6.2 Tabella della verità differenza binaria 14
4.2.1 Definizione delle 4 cifre della password 21

1 Introduzione

Lo sportello automatico, anche noto con il termine di uso internazionale ATM (dall'inglese Automated Teller Machine), è il sistema per il prelievo automatico di denaro contante dal proprio conto corrente bancario, attraverso l'uso di una carta di debito nei distributori collegati in rete telematica, anche fuori dagli orari di lavoro degli istituti di credito e in località diverse dalla sede della banca presso cui si ha il rapporto di conto corrente. Gli attuali ATM utilizzano, per il riconoscimento del cliente, una tessera plastificata con banda magnetica e microchip da inserire nell'apposito lettore posto nella parte anteriore dell'apparecchio e l'inserimento di un codice numerico segreto chiamato PIN (Personal Identification Number). Storicamente il primo sportello automatico è stato installato il 27 giugno 1967 presso la banca Barclays a Londra mentre per vedere il primo di questi apparecchi in funzione in Italia occorrerà attendere il 1976, quando viene installato su suolo italiano il primo ATM a Ferrara. I primi modelli di ATM funzionavano con voucher monouso che venivano letti e trattenuti dall'apparecchio. Successivamente, nel 1983 nasce la prima carta di debito bancaria per operare sul circuito BANCOMAT®, cui aderiscono le banche italiane, che consente il prelievo di contanti su qualunque apparecchio ATM presente sul territorio italiano. Dopo l'inizio del secondo millennio, al fine di rendere più sicure le transazioni, l'identificazione dell'utente è migrata dalla tecnologia a banda magnetica presente sulle carte a quella a microchip. Una volta che il cliente è stato riconosciuto mediante il corretto inserimento di una password fornita dalla banca è possibile accedere alle varie funzionalità dell'apparecchio. Nel caso in esame il cliente può:

- effettuare prelievo di denaro contante senza che però rimanga un saldo negativo sul conto corrente
- effettuare versamento di denaro contante nei limiti dei massimali imposti dal conto corrente

Figura 1.1: Primo ATM installato in Italia

2 Principio di funzionamento

La sportello automatico implementato prevede di riconoscere un PIN di accesso di 4 cifre comprese tra 0 e 3. La password per accedere al conto corrente è definita mediante 8 switch mentre l'inserimento della password da parte dell'utente avviene mediante l'utilizzo di 4 push-button di cui uno è utilizzato per l'inizializzazione della macchina (button di clear e indicato con bn(3)), un'altro è utilizzato per inserire la stringa di bit nel registro della macchina a stati finiti e/o terminare un'operazione (indicato con bn(0) e i restanti due button sono utilizzati per definire la coppia di bit da inserire nella macchina. I push-button indicati con bn(2), bn(1) e bn(0) sono posti in ingresso ad una porta OR in modo tale che quando si inserisce un numero diverso da zero non sia necessario premere il push-button bn(0) per inserire la cifra desiderata. La seguente tabella riassume quello che è stato precedentemente descritto:

	Clear	Cifra		Enter	
Caso	$btn(3)$	$btn(2)$	$btn(1)$	$btn(0)$	*Funzione*
1	1	X	X	X	Inizializzazione
2	0	0	0	1	Inserimento 0
3	0	0	1	X	Inserimento 1
4	0	1	0	X	Inserimento 2
5	0	1	1	X	Inserimento 3

Tabella 2.1: Tabella di funzionamento dei push-button

L'utente ha tre tentativi per inserire correttamente la password, superati i quali la macchina entra in fase di stop, fase in cui l'utente non potrà più accedere al conto corrente e sarà necessario reinizializzare la macchina per poter riprovare ad accedervi. Qualora invece l'utente riuscisse a inserire correttamente la password entro i 3 tentativi, accederà al conto corrente. Il capitale presente nel conto corrente è settato mediante altri 4 switch prima di effettuare una qualsiasi operazione. A questo punto l'utente potrà definire l'ammontare di denaro contante che vuole versare o prelevare dal conto corrente e l'operazione che intende compiere mediante ulteriori 5 switch. Le due possibili operazioni che possono essere scelte dall'utente sono le seguenti:

- se lo switch relativo all'operazione è settato a 1, si sceglie l'opzione di prelievo di denaro
- se lo switch relativo all'operazione è settato a 0, si sceglie l'opzione di versamento di denaro

Per procedere con l'operazione desiderata l'utente dovrà settare a 1 lo switch di enable e attendere la fine dell'operazione che è mostrata attraverso l'accensione di un led.

Nella seguente figura è mostrato uno schema del circuito digitale realizzato in cui si evidenziano i push-button e gli switch utilizzati per gli input del sistema:

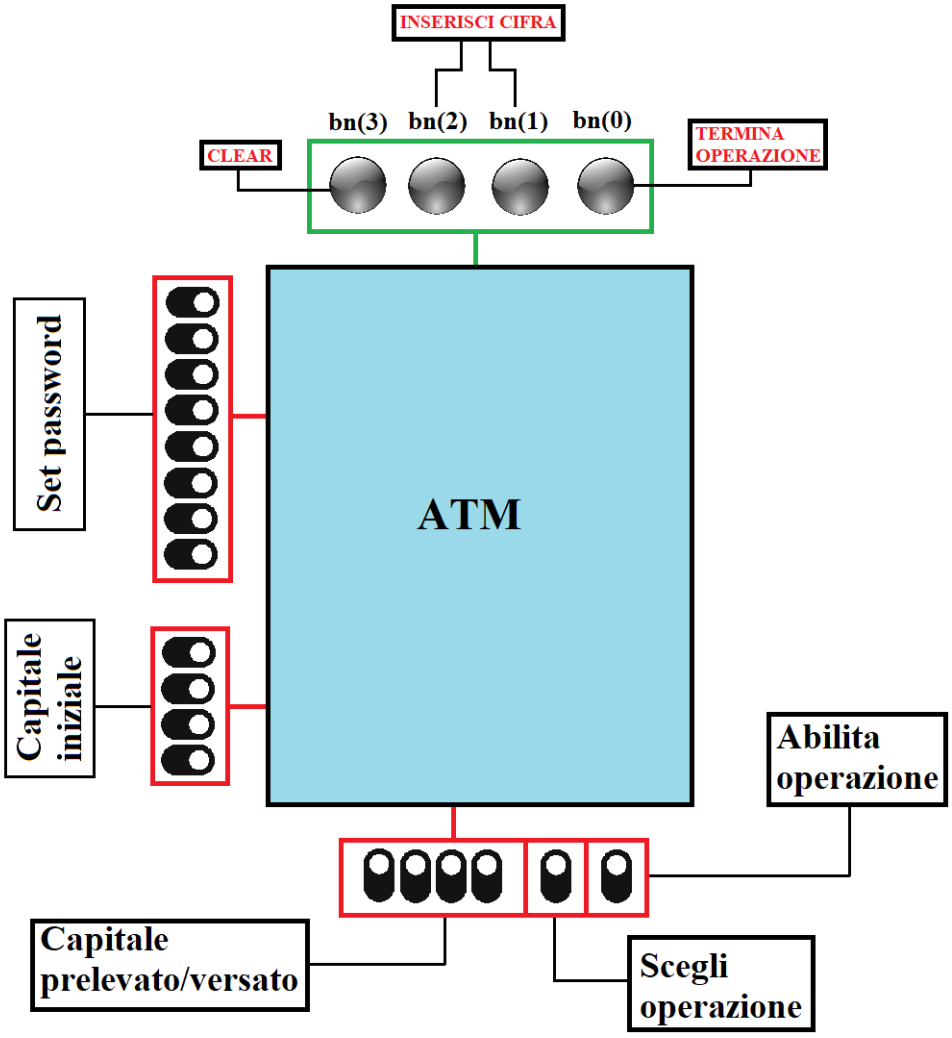

Figura 2.1: Schema illustrativo degli switch e push-button utilizzati

Un diagramma di flusso utile per comprendere il sistema realizzato è mostrato di seguito:

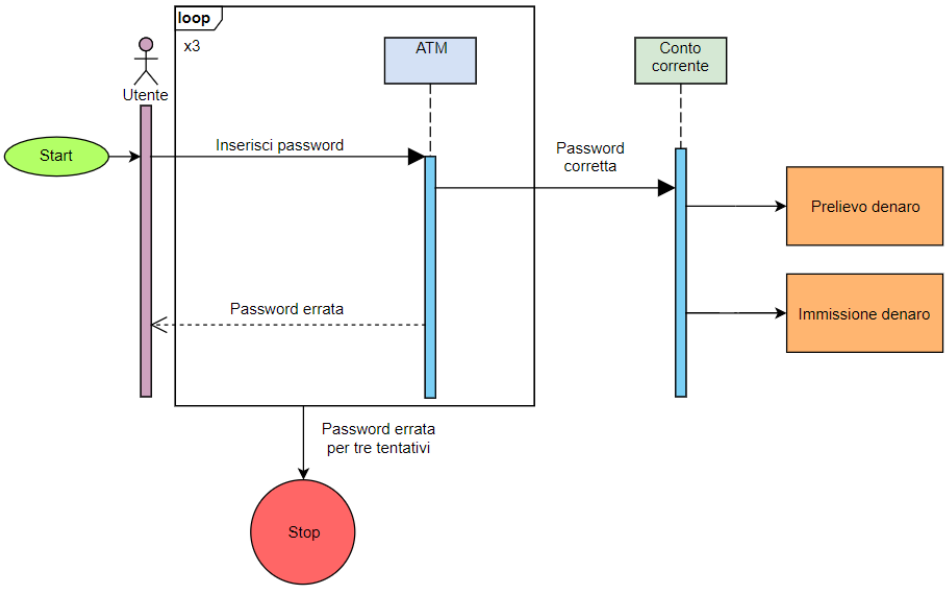

Figura 2.2: Diagramma di flusso del sistema realizzato

3 Sintesi del circuito digitale

3.1 Macchina a stati finiti

Per la realizzazione dell'ATM è stato necessario definire una macchina a stati finiti che permettesse il riconoscimento della password inserita dall'utente. La FSM implementata presenta 10 stati distinti:

- S_0, se in tale stato entra la prima cifra della password di accesso si passa a S_1, altrimenti si passa nello stato di errore E_1
- S_1, se in tale stato entra la seconda cifra della password di accesso si passa a S_2, altrimenti si passa nello stato di errore E_2
- S_2, se in tale stato entra la terza cifra della password di accesso si passa a S_3, altrimenti si passa nello stato di errore E_3
- S_3, se in tale stato entra l'utlima cifra della password di accesso si passa a S_4, altrimenti si passa nello stato di errore E_4
- S_4, stato in cui si giunge solo se è stata inserita correttamente la password
- E_1, stato di errore in cui si giunge quando la prima cifra inserita non è corretta
- E_2, stato di errore in cui si giunge quando la seconda cifra inserita non è corretta
- E_3, stato di errore in cui si giunge quando la terza cifra inserita non è corretta
- E_4, stato di errore in cui si giunge quando la quarta cifra inserita non è corretta
- *Stop*, stato in cui si giunge quando è stata inserita una password errata per 3 volte consecutive oppure quando l'utente ha inserito correttamente la password ed è termina l'operazione di prelievo o versamento

La macchina a stati descritta pocanzi è mostrata schematicamente nella seguente figura:

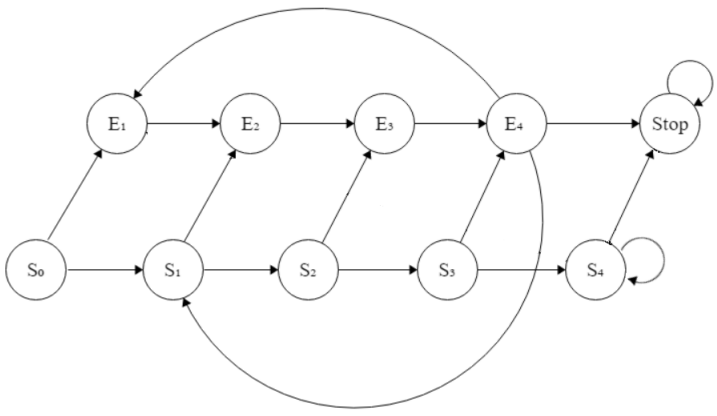

Figura 3.1.1: Macchina a stati finiti

3.2 Divisore di freqeunza

L'implementazione di un divisore di clock è stato necessario sia per la realizzazione del circuito di debouncing dei push-button utilizzati per inserire le cifre del PIN sia per l'accensione dei led. Ricordiamo infatti che la frequenza del clock della board utilizzata è di $50MHz$ quindi ha un periodo di $20ns$ con il quale sarebbe impercettibile l'intermittenza dei led all'occhio umano. Il divisore di clock è implementato mediante un contatore *mod* 24 attraverso cui si dimezza la frequenza di partenza del clock per ottenere la frequenza desiderata.
Nella seguente tabella è mostrata come varia la frequenza del clock per ogni divisione effettuata:

$q(i)$	$Frequenza[Hz]$	$Periodo[ms]$
	50000000.00	0.00002
0	25000000.00	0.00004
1	12500000.00	0.00008
2	6250000.00	0.00016
3	312500.00	0.00032
4	1562500.00	0.00064
5	781250.00	0.00128
6	390625.00	0.00256
7	195312.50	0.00512
...	...	...
17	190.73	5.24288
18	95.37	10.48576
19	47.68	20.97152
...	...	...
23	2.98	355.54432

Tabella 3.2.1: Tabella di divisione del clock con contatore Mod 24

Nella seguente figura è mostrato lo schema a blocchi che definisce il circuito digitale per la divisione del clock:

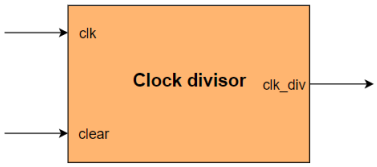

Figura 3.2.1: Divisore di clock

Per verificare il corretto funzionamento del circuito digitale implementato è stata realizzata una simulazione del circuito mediante $ModelSim$: in questo software è possibile definire le forme d'onda dei segnali di ingresso e avviando la simulazione si ottengono le forme d'onda dei segnali di uscita. Nel caso in esame è stato considerato un clock di frequenza pari a 50 MHz quindi con un periodo di 20 ns. In una prima fase è stato forzato il bit di *clear* a 1 in modo da inizializzare il circuito e successivamente è stato settato a 0. É stato considerato un divisore di frequenza tale da ottenere un segnale di clock in uscita di frequenza pari a 190 Hz quindi con un periodo di circa 5 ms.
La simulazione realizzata è mostrata nella seguente figura:

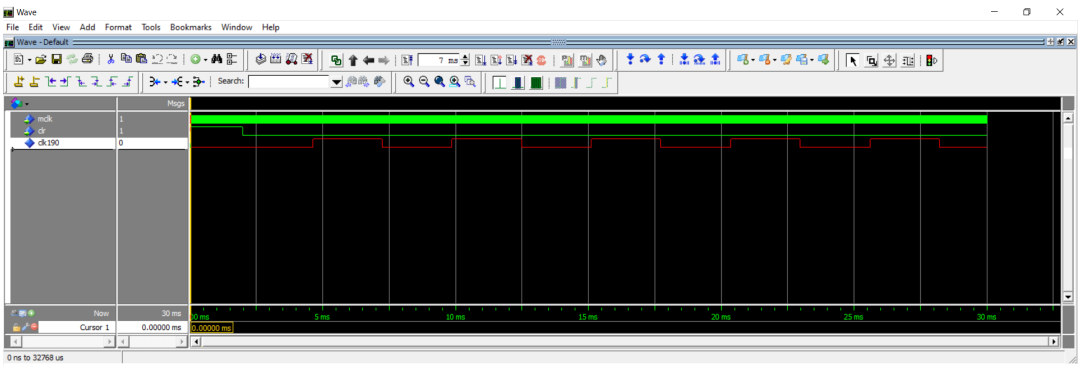

Figura 3.2.2: Simulazione in ModelSim del divisore di frequenza

3.3 Circuito di debouncing

Quando un interruttore viene chiuso (quindi viene premuto), tale chiusura non avviene in genere istantaneamente. A causa della flessibilità elastica della lamina metallica interna dell'interruttore, la chiusura produce infatti una serie di micro-rimbalzi (in inglese bounce), cioè una rapida sequenza di stati aperto/chiuso in successione.

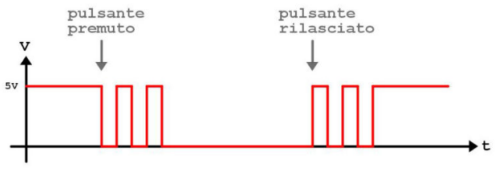

Figura 3.3.1: Microrimbalzo in un push-button

Nel caso in oggetto, poiché i push-button sono utilizzati per dare in ingresso alla macchina a stati finiti le cifre che costituiscono il PIN di accesso al conto corrente, il fenomeno del microrimbalzo non è trascurabile e per tale motivazione è necessario realizzare un circuito antirimbalzo (in inglese noto come "debouncing circuit").
Lo schema a blocchi del circuito è mostrato nella seguente figura:

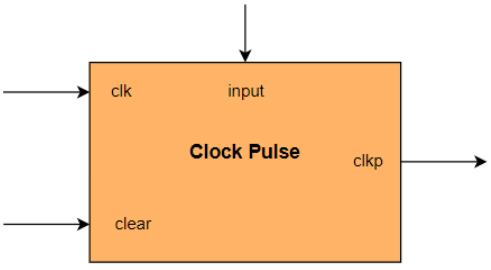

Figura 3.3.2: Schema a blocchi del circuito antirimbalzo

Anche in questo caso, per verificare il funzionamento del circuito è stata realizzata una simulazione in $ModelSim$, mostrata nella seguente figura:

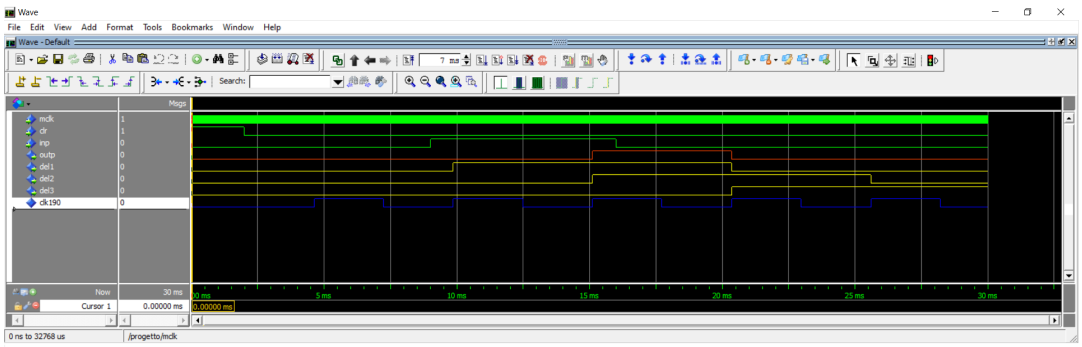

Figura 3.3.3: Simulazione del circuito antirimbalzo con divisore di frequenza

Nella precedente figura è possibile osservare diverse forme d'onda con differenti colori:

- le forme d'onda di input sono rappresentate con colore verde
- le forme d'onda relative ai segnali intermedi sono rappresentate con il colore giallo
- la forma d'onda del segnale di clock diviso ad una frequenza di 190 Hz è rappresentato con il colore blu
- la forma d'onda del segnale di uscita è rappresentata con il colore rosso

I 3 segnali intermedi (indicati in figura con $del1$, $del2$ e $del3$) permettono di determinare il segnale di output che evita i microrimbalzi dei push-button: il segnale $del1$ è identico al segnale di input che simula la pressione del push-button mentre gli altri due segnali, ovvero $del2$ e $del3$, sono ritardati di un periodo del clock a 190 Hz rispetto a $del1$. A questo punto per determinare il segnale di output si pongono i 3 segnali in ingresso ad una porta AND, in particolar modo:

$$outp = del_1 \ AND \ del_2 \ AND \ (NOT(del_3))$$

In questo modo il segnale di output sarà ad un valore logico alto soltanto quando l'uscita della porta AND è pari a 1, ovvero per segnali con frequenza inferiore a 190 Hz mentre tutte le volte che la frequenza di commutazione del segnale è superiore a tale frequenza, l'output risulterà essere nullo quindi è come se non fosse stato premuto il push-button.

3.4 Contatore

Il contatore è utilizzato per contare il numero di tentativi falliti nell'inserimento della password da parte dell'utente che vuole accedere al proprio conto corrente. Lo schema a blocchi è mostrato nella figura 3.4.1.
Dallo schema a blocchi del contatore a 4 bit è possibile descrivere il funzionamento di tale circuito: quando il clock si trova sul fronte di salita ed accade una particolare condizione, che nel nostro caso è l'errato inserimento del PIN da parte dell'utente, il contatore incrementerà di 1, altrimenti esso rimarrà inalterato. Osserviamo che anche per questo circuito è necessario un segnale di *clear* per la fase di inizializzazione.

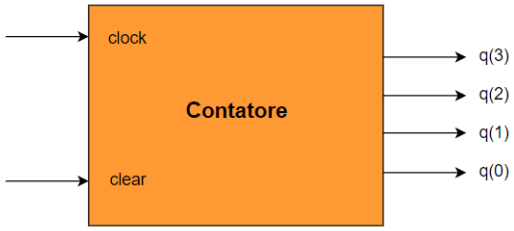

Figura 3.4.1: Contatore a 4 bit

3.5 Display a 7 segmenti

Il display a sette segmenti è un dispositivo elettronico in grado di visualizzare le 10 cifre numeriche, e in alcuni casi alcune lettere alfabetiche e simboli grafici, attraverso l'accensione di combinazioni di sette segmenti luminosi.

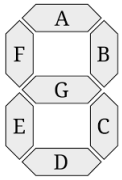

Figura 3.5.1: Display a 7 segmenti

La tabella della verità che mostra come devono essere accesi i 7 led luminosi per mostrare i numeri da 0 a 9 è rappresentata nella seguente tabella:

BCD inputs				segment outputs							display
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

Figura 3.5.2: Mappa di Karnaugh display a 7 segmenti

Nel caso in oggetto sono stati utilizzati 3 display a 7 segmenti presenti sulla board:

- il primo display è utilizzato per mostrare il numero di tentativi falliti nell'inserimento della password da parte dell'utente, numeri compresi in un range da 0 a 3.
- il secondo e terzo display sono utilizzati per mostrare a schermo il capitale presente sul conto corrente quando l'utente ha effettuato l'operazione

A titolo di esempio è mostrato lo schema a blocchi del primo display:

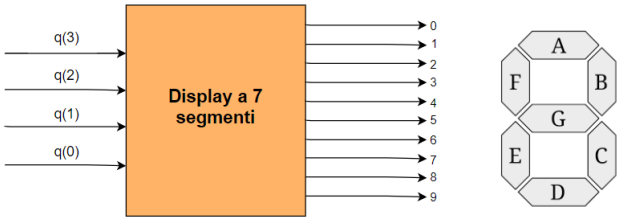

Figura 3.5.3: Schema a blocchi display a 7 segmenti

L'input del primo display coincide con l'output del contatore in modo che sul display venga visualizzato il numero di tentativi falliti nell'inserimento della password da parte dell'utente. Come si evince dalla precedente figura, in base ai 4 bit di ingresso viene abilitata una particolare word line costituita da 7 bit, ognuno dei quali connesso ad uno dei 7 led luminosi che costituiscono il display. Gli altri due display avranno come input il capitale finale (output del circuito sommatore/sottrattore).

3.6 Full Adder

La realizzazione del circuito sommatore/sottrattore è stata necessaria per simulare il prelievo o versamento di denaro da parte dell'utente una volta che è stata inserita correttamente la password. Prima di definire il circuito digitale realizzato per effettuare somma e differenza tra stringhe di bit, ricordiamo come si effettuano le operazioni aritmetiche nel sistema binario, in particolar modo somma e differenza.
La tabella della verità della somma aritmetica tra bit è mostrata nella seguente tabella:

Input		*Output*	
A	*B*	*Somma*	*Riporto*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 3.6.1: Tabella della verità somma binaria

La tabella della verità della differenza aritmetica tra bit è mostrata nella seguente tabella:

Input		*Output*	
A	*B*	*Differenza*	*Prestito*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Tabella 3.6.2: Tabella della verità differenza binaria

Lo schema del circuito sommatore/sottrattore binario a 4 bit è mostrato nella seguente figura:

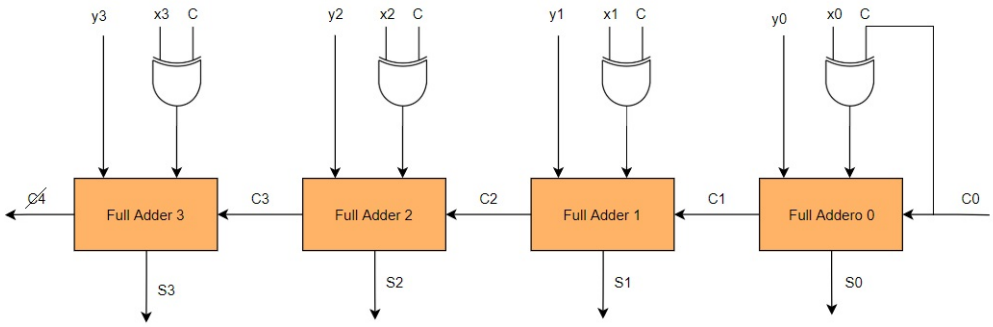

Figura 3.6.1: Sommatore/sottrattore binario

Poiché possiamo effettuare la differenza tramite la rappresentazione del numero negativo in complemento a 2, rendendo l'operazione una somma, è possibile utilizzare sia per la somma che per la differenza di stringhe di bit un full adder. Per far ciò si definisce un ingresso di controllo C_0 che si pone in XOR con il numero che si vuole sottrarre:

- quando C_0 è pari a 0 in uscita dalla XOR otteniamo l'ingresso inalterato
- quando C_0 è pari a 1 in uscita otteniamo l'ingresso complementato

In definitiva, se il bit di controllo è impostato a 1 si effettua la differenza tra le due stringhe di bit (indicate in figura 3.6.1 con x e y) mentre se è pari a 0 si effettua la somma tra le due stringhe di bit. Si rammenti che l'ultimo riporto va scartato ai fini del risultato.
Il circuito digitale realizzato su *Quartus* 2 è schematizzato nello schema a blocchi mostrato nella seguente figura:

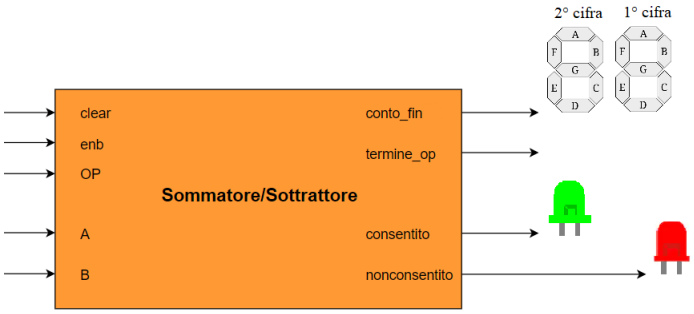

Figura 3.6.2: Schema a blocchi del circuito sommatore/sottrattore realizzato

Come si osserva in figura 3.6.2, gli ingressi del circuito sommatore/sottrattore sono i seguenti:

- *clr*, ingresso di clear per inizializzare la macchina
- *enb*, ingresso di enable utile per consentire alla macchina di effettuare l'operazione richiesta
- *OP*, bit di controllo che permette all'utente di scegliere quale operazione compiere (versamento se è pari a 0, prelievo se è pari a 1)

- A che indica il capitale iniziale presente nel conto corrente
- B che indica la somma di denaro che si vuole versare o prelevare

Gli output del circuito sono invece i seguenti:

- *conto_finale*, ovvero il capitale finale presente sul conto dopo che è stata effettuata l'operazione mostrato a schermo attraverso due display a 7 segmenti
- *termine_op*, flag che indica se l'operazione richiesta dall'utente è termina o meno
- *consentito*, led luminoso verde abilitato quando l'operazione richiesta dall'utente è consentita
- *nonconsentito*, led luminoso rosso abilitato quando l'operazione richiesta dall'utente non è consentita

É importante osservare che qualora l'operazione richiesta dall'utente non fosse consentita, la macchina chiederà all'utente di riprovare tale operazione finché l'operazione non sia consentita.
Per verificare il corretto funzionamento del circuito digitale realizzato è stata effettuata una simulazione in *ModelSim* mostrata nella figura 3.6.3 in cui sono stati analizzati i seguenti casi:

- segnale di clear settato a 1
- segnale di enable settato a 0
- versamento non consentito a causa di una somma di denaro che supera i massimali imposti per il conto corrente
- prelievo non consentito a causa di una somma di denaro superiore al capitale iniziale
- versamento e prelievo consentiti

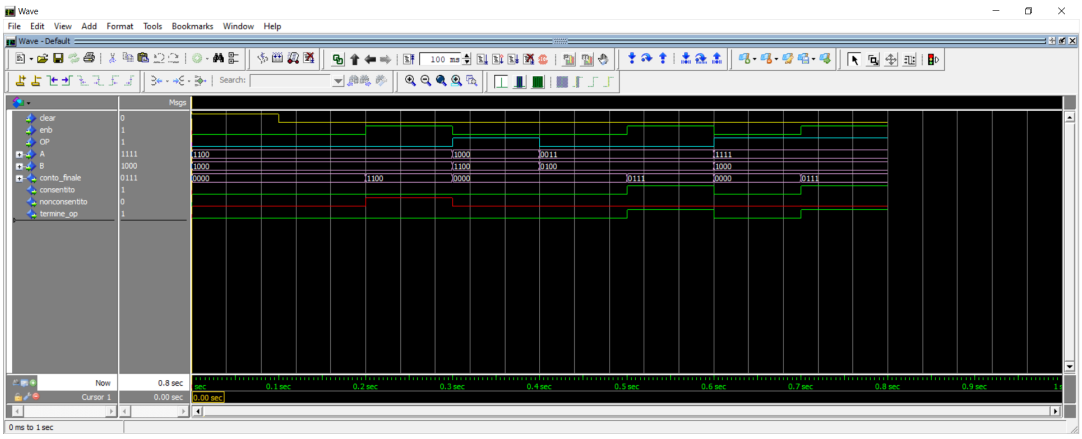

Figura 3.6.3: Simlazione del circuito sommatore/sottrattore in Modelsim

Come è possibile osservare nella precedente figura il primo caso analizzato è quello in cui il segnale di clear è impostato ad un livello logico alto: in tal caso la macchina è resettata perciò tutti i led sono disabilitati e il conto finale è nullo. Il secondo caso prevede di verificare il funzionamento dello switch di enable: si osserva che quando esso è ad un livello logico basso, sebbene venga richiesto alla macchina di effettuare una certa operazione, essa non porterà a termine quest'ultima dando come capitale finale capitale nullo.

Il caso successivo prevede che l'utente voglia versare una somma di denaro tale da superare i massimali imposti dalla banca: in tal caso il capitale finale coinciderà con quello iniziale, con l'accensione di un led luminoso indicante l'operazione non consentita(*non_consentito*). Quanto detto per il precedente caso accade anche quando l'utente ha intenzione di prelevare una somma di denaro superiore a quella inizialmente presente sul conto corrente. Nei casi precedentemente citati il flag *termine_op* permane ad uno stato logico basso poiché la macchina non ha effettuato alcuna operazione consentita.
Gli ultimi due casi analizzati nella simulazione sono quelli in cui l'utente voglia prelevare o versare una somma di denaro consentita: in tal caso la macchina effettuerà l'operazione dando in output il capitale finale, sarà abilitato il led luminoso verde *consentito* ad indicare che l'operazione richiesta è consentita e l'uscita indicata con *termine_op* si porterà a valore logico alto ad indicare che l'operazione è stata eseguita con successo.

4 Circuito digitale complessivo

Dopo aver analizzato singolarmente i blocchi digitali che costituiscono il circuito digitale complessivo, mediante la funzione *RTL Viewer* di *Quartus* è stato definito lo schema logico del progetto digitale realizzato mostrato nella figura 4.1.

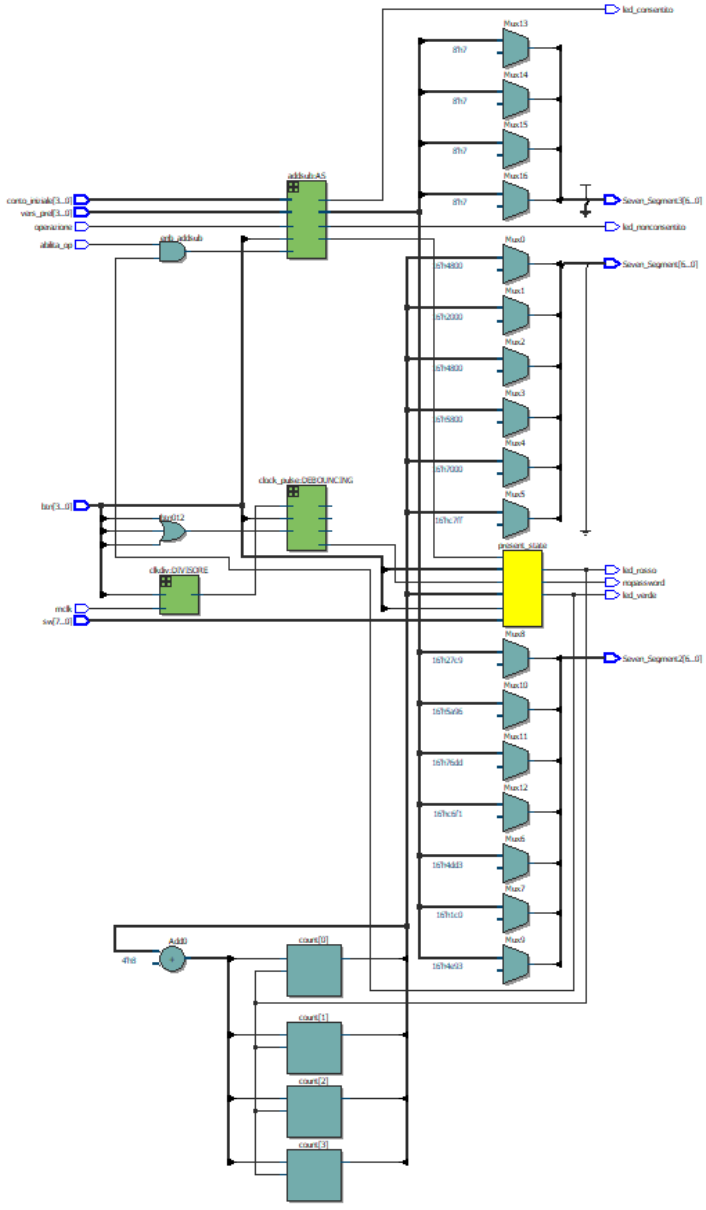

Figura 4.1: Schema logico di progetto dell'ATM bancario

Sempre mediante la funzione *RTL Viewer* è possibile visualizzare lo schema logico della FSM realizzata, mostrata nella seguente figura 4.2.

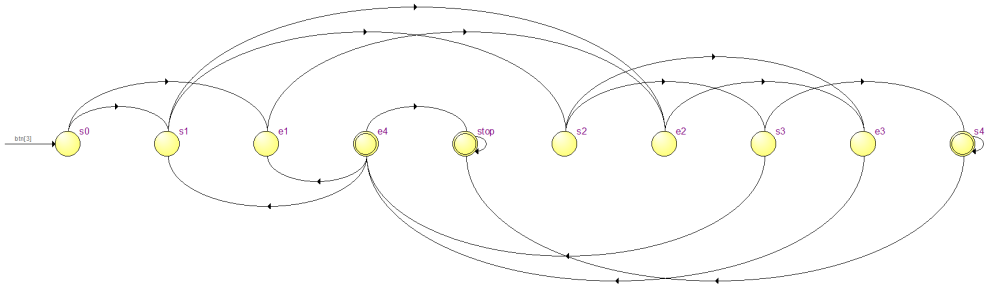

Figura 4.2: Schema logico della FSM

A sinistra della figura 4.1 si osservano gli input del circuito digitale realizzato, ovvero:

- 8 switch per settare la password ($sw[7:0]$)
- 4 switch per settare il capitale iniziale presente nel conto corrente ($conto\_iniziale[3:0]$)
- 4 switch per stabilire il capitale che si intende versare o prelevare ($vers\_prel[3:0]$)
- 1 switch per abilitare l'operazione di versamento o prelievo da conto corrente ($abilita\_op$)
- 1 switch per definire quale operazione si intende effettuare ($operazione$,se settato a 1 operazione di prelievo, se settato a 0 operazione di versamento)
- 4 push-button per inserire la password ($btn[3:0]$) di cui il button 3 è utilizzato per inizializzare la macchina, il button 2 e 1 sono utilizzati per inserire a seconda della combinazione la cifra della password e il button 0 è utilizzato per inserire la cifra precedentemente definita e/o terminare un'operazione
- segnale di clock a frequenza pari a 50 MHz ($mclk$)

A destra della figura 4.1 si osservano gli output, ovvero:

- led_rosso, abilitato se la password inserita è errata
- led_verde, abilitato se la password inserita è corretta
- $led\_consentito$, abilitato se l'operazione richiesta dall'utente è consentita
- $led\_nonconsentito$, abilitato se l'operazione richiesta dall'utente non è consentita
- $nopassword$, led abilitato quando la FSM entra nello stato di $Stop$
- $conto\_finale$ che indica il capitale presente nel conto corrente dopo che è stata effettuata l'operazione richiesta dall'utente
- $Seven\_Segment$, output del display a 7 segmenti che mostra il numero di tentativi falliti nell'inserimento della password
- $Seven\_Segment2$, output del display a 7 segmenti che mostra la prima cifra del capitale finale presente sul conto
- $Seven\_Segment3$, output del display a 7 segmenti che mostra la seconda cifra del capitale finale presente sul conto

4.1 Messa in opera dell'ATM

La messa in opera dell'ATM si articola principalmente su 5 fasi illustrate qui di seguito.

4.1.1 Fase 1 - Inizializzazione

Si setta a 1 il segnale di clear in modo da inizializzare la macchina per evitare eventuali malfunzionamenti della stessa. Successivamente si impostano gli input quali:

- password iniziale, $sw[7:0]$
- capitale iniziale, $conto\_iniziale[3:0]$
- capitale da prelevare o versare, $vers\_prel[3:0]$

4.1.2 Fase 2 - Accesso al conto corrente

Per accedere al conto corrente è necessario inserire la password corretta attraverso la pressione dei push-button secondo un'opportuna combinazione. Si hanno a disposizione 3 tentativi: se la password è inserita corettamente entro questi, un led verde segnalerà l'avvenuto accesso al conto corrente, viceversa un led rosso segnalerà il mancato accesso facendo entrare la FSM in uno stato di stop, da cui si potrà uscire inizializzando nuovamente la macchina (simulando così il blocco della carta). Il numero di tentativi effettuati viene mostrato su un apposito display a 7 segmenti, pilotato da un contatore a 4 bit in avanti.

4.1.3 Fase 3 - Prelievo o versamento

Se la password è stata inserita correttamente, si ha accesso al conto corrente, avendo quindi la possibilità di scegliere se versare o prelevare contante. La scelta dell' operazione è definita settando lo switch *operazione* (0 - versamento, 1 - prelievo), mentre il capitale da versare o prelevare è definito attraverso lo switch $vers\_prel$. Per confermare l'operazione scelta è necessario agire sullo switch *abilita_op*. Il risultato dell'operazione scelta è effettuata da un sommatore/sottrattore a 4 bit (quindi costituito da 4 Full Adder): se l'operazione è consentita, il capitale finale sarà disponibile in uscita, con l'accensione di un led verde (*led_consentito*), altrimenti l'operazione verrà annullata con l'accensione di un led rosso (*led_nonconsentito*), nei casi in cui l'utente voglia prelevare un capitale superiore al capitale iniziale oppure voglia versare una quantità di denaro tale da superare il massimale del conto corrente. Il capitale finale sarà quindi pari al capitale iniziale.

4.1.4 Fase 4 - Ripristino

Per riportare la FSM alla condizione inziale di funzionamento, basterà abilitare il push-button di clear *clr*.

4.2 Risultati e discussione della simulazione effettuata in ModelSim

Per verificare il corretto funzionamento del circuito digitale realizzato sono stati analizzati diversi casi particolari discussi di seguito. La password scelta è 2012, opportunamente settata con gli switch $sw[7:0]$. Ogni cifra della password è codificata da una coppia di bit quindi, nel caso in oggetto, il segnale *sw* sarà in binario pari a 10000110, come mostrato nella tabella 4.2.1.

	Password			
	Quarta cifra	Terza cifra	Seconda cifra	Prima cifra
Decimale	2	0	1	2
Binario	1 0	0 0	0 1	1 0

Tabella 4.2.1: Definizione delle 4 cifre della password

4.2.1 Caso 1: Password corretta, prelievo non consentito, prelievo consentito

Nella simulazione mostrata in figura 4.2.1, è stato analizzato il caso in cui l'utente inserisce correttamente la password ma voglia prelevare una somma di denaro contante superiore al capitale inizialmente presente sul conto corrente. In prima istanza è stata inizializzata la macchina mediante il segnale di clear (corrispondente al quarto bit dell'ingresso *btn*); successivamente sono state inserite le cifre corrette della password attraverso i push-button *btn* Quando tutte le cifre sono state correttamente inserite, si osserva che il *present_state* è pari a S_4, quindi l'utente è riuscito ad accedere al conto corrente. E' stato quindi settato a 1 il segnale indicato con *abilita_op* affinché il sommatore/sottrattore elaborasse l'operazione scelta dall'utente (in questo caso ha scelto di prelevare contante - segnale *operazione* pari a 1). Poiché il capitale iniziale (indicato con *conto_inziale*) è pari a 3 (0011 in binario) e il capitale che l'utente desidera versare è superiore ad esso (pari a 8, 1000 in binario), l'operazione richiesta dall'utente non può essere effettuata, con conseguente abilitazione del led *led_nonconsentito* e il capitale finale risulta essere pari al capitale iniziale. L'utente a questo punto dovrà portare a livello logico basso il segnale di abilitazione del sommatore/sottrattore, modificare il capitale che intende prelevare e riabilitare il segnale di abilitazione: nel caso in oggetto il capitale che si vuole prelevare è stato modificato a 1 (0001 in binario). In tal caso si osserva che è abilitato il led indicato con *led_consentito*, ad indicare che l'operazione richiesta è consentita, e il capitale finale sarà pari alla differenza tra quello iniziale e quello che si intende prelevare, ovvero 1 (ricordiamo che il capitale iniziale è rimasto invariato pari a 3 mentre quello che si intende prelevare è stato modificato a 2). Infine, l'utente dovrà premere il push-button *btn*(0) per chiudere l'interfaccia e portare la macchina nello stato di stop nel quale si abiliterà il led indicato con *nopass*.

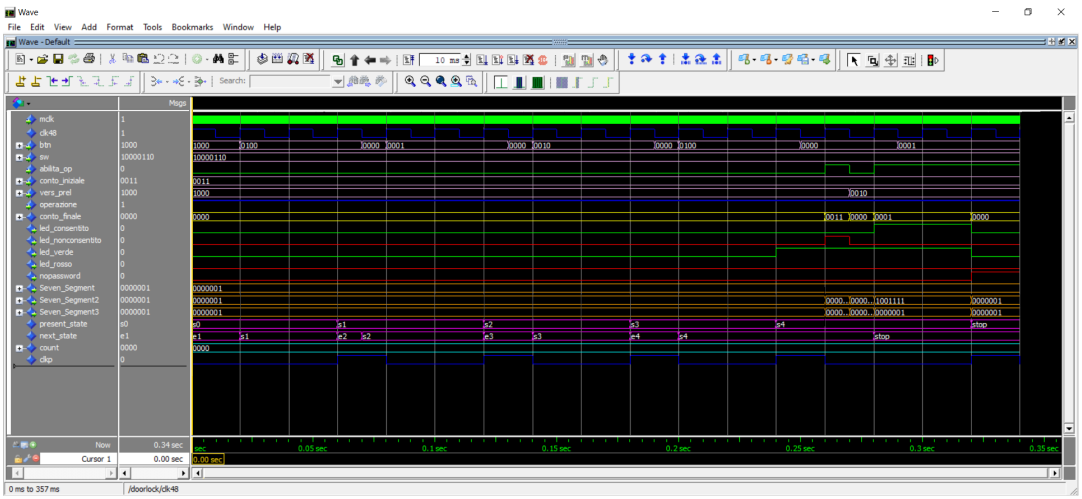

Figura 4.2.1: Password corretta, prelievo non consentito, prelievo consentito

4.2.2 Caso 2: Password corretta, versamento non consentito, versamento consentito

Nella simulazione mostrata in figura 4.2.2 è stato analizzato il caso in cui l'utente inserisca correttamente la password ma voglia versare una somma di denaro che supera i massimali del conto corrente imposti dalla banca. Come nel caso precedente, si procede portando a livello logico alto il segnale di clear per inizializzare la macchina. Successivamente l'utente inserisce correttamente la password (che ricordiamo essere 2012) e la macchina si porta nello stato S_4. A questo punto è abilitato il circuito sommatore/sottrattore portando a livello logico alto il segnale di abilitazione *abilita_op*.
Supponendo che sul conto corrente sia presente un capitale iniziale pari a 9 euro (1001 in binario) e che la somma di denaro che l'utente intende versare sia pari a 8 euro (1000 in binario), poiché la somma del capitale iniziale e di quello che intende versare supera il massimale del conto corrente imposto dalla banca, l'operazione non può essere effettuata, con conseguente abilitazione del led *led_nonconsentito*, con il capitale finale coincidente con quello iniziale.
L'utente a questo punto ha tre possibilità:

- terminare l'operazione, portando lo stato corrente della macchina a quello di stop
- prelevare anziché versare denaro contante
- versare meno denaro di quello precedentemente definito

In questa simulazione il capitale che si vuole versare viene modificato ad un valore pari a 6 (0110 in binario) in modo che non sia superato il massimale del conto corrente. L'operazione andrà a buon fine in corrispondenza dell'abilitazione del segnale *abilita_op*. Il capitale finale risulta essere pari a 15 (in binario 1111) con accensione del led *led_consentito*. Per chiudere l'interfaccia dell'ATM, all'utente basterà premere il push-button indicato con *btn*(0).

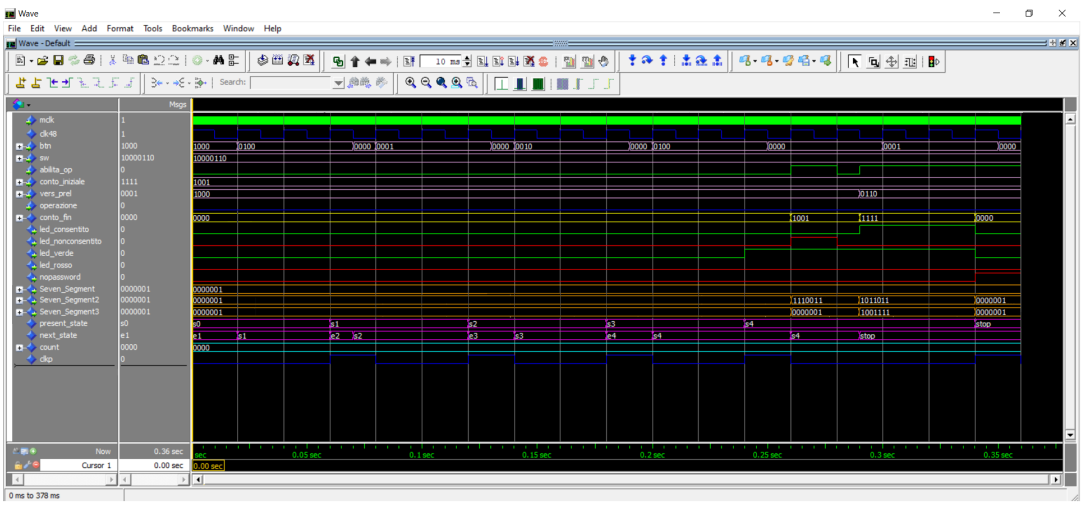

Figura 4.2.2: Password corretta, versamento non consentito, versamento consentito

Si osservi che nelle due precedenti simulazioni è possibile osservare anche il segnale indicato con *clkp* che rappresenta il clock impulsivo utilizzato dalla macchina per evitare i microrimbalzi dei push-button. Nelle successive simulazioni questo segnale (che è un segnale intermedio) verrà sottinteso.

4.2.3 Caso 3: Password corretta e versamento consentito

Nella simulazione mostrata in figura 4.2.3 è stato analizzato il caso in cui l'utente inserisca correttamente la password e scelga di versare una somma di denaro contante tale da non superare i massimali imposti dalla banca per il conto corrente. Il capitale iniziale è stato impostato a 3 euro (0011 in binario) mentre la somma di denaro che si intende versare è stata impostata a 8 euro (1000 in binario).
Dopo che l'utente ha inserito correttamente tutte e quattro le cifre della password, la macchina si porta nello stato S_4 e viene abilitato il *led_verde* indicante il corretto accesso al conto corrente. A questo punto l'utente abilita il circuito sommatore/sottrattore portando a livello logico alto il segnale *abilita_op*: poiché la somma tra il capitale iniziale e quello che si intende versare non è superiore ai massimali del conto corrente, viene abilitato il led *led_consentito*, che indica la possibilità di effettuare l'operazione, ottenendo così come capitale finale 11 euro (somma del capitale iniziale e quello versato).
A questo punto l'utente, per chiudere l'interfaccia dell'ATM, dovrà premere il push-button $btn(0)$, portando la macchina nello stato di stop.

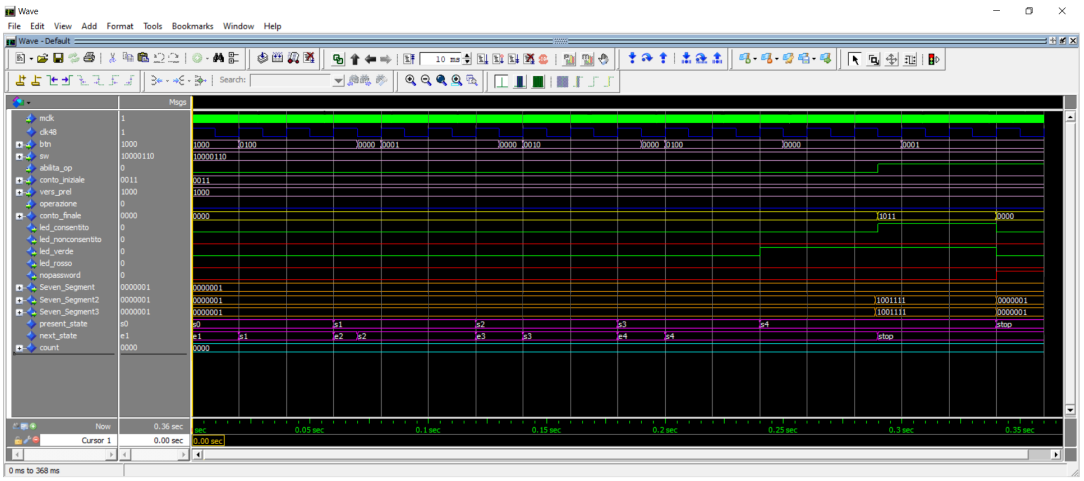

Figura 4.2.3: Password corretta e versamento consentito

4.2.4 Caso 4: Password errata, password corretta e versamento consentito

La simulazione mostrata in figura 4.2.4 descrive il comportamento della macchina nel caso in cui l'utente inserisca inizialmente una password errata ed al secondo tentativo inserisca la password corretta.
Si parte con l'inizializzazione della macchina portando a livello logico alto il segnale di clear associato al quarto push-button (quarto bit del segnale di input *btn*). La password è stata impostata come nei casi precedenti a 2012 (quindi 100001110 in binario).
Nel primo tentativo l'utente inserisce erroneamente la prima cifra della password (anziché inserire 2 inserisce 1) quindi la macchina si porterà allo stato di errore E_4 e verrà incrementato il segnale indicato con *count* che mostra il numero di tentativi falliti dall'utente per l'inserimento della password. L'uscita del contatore coincide con il segnale di input del display a 7 segmenti, infatti quando la macchina è nello stato S_4 il segnale *count* è pari a 0001 mentre il segnale *Seven_Segment* è pari a 1001111.
Nel secondo tentativo l'utente inserisce correttamente la password quindi la macchina si porta allo stato S_4 nel quale si ha l'accensione del *led_verde* che indica il corretto inserimento della password. A questo punto l'utente sceglie: quale operazione effettuare, capitale iniziale e capitale che intende prelevare o versare. Il capitale iniziale è posto a 8 euro (1000 in binario), l'utente quindi sceglie come operazione il versamento (poiché il segnale *operazione* è a livello logico basso) e il capitale che si vuole versare è pari a 6 euro (0110 in binario). L'operazione può essere effettuata (abilitazione *led_consentito*) e il capitale finale risulta essere pari alla somma di quello iniziale e quello versato, ovvero 14 euro (1110 in binario). A questo punto l'utente dovrà premere il push-button associato al primo bit del segnale di input *btn* per chiudere l'interfaccia della macchina.

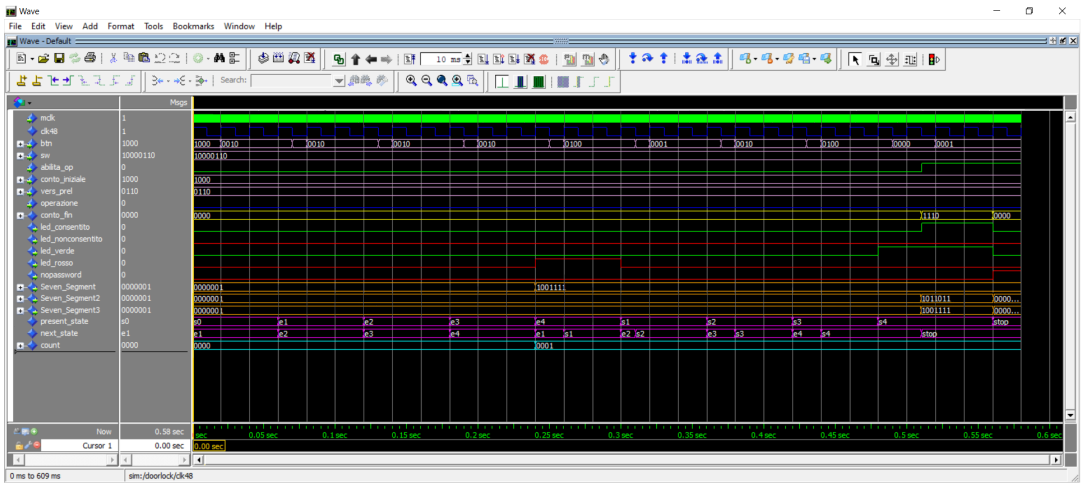

Figura 4.2.4: Password errata, password corretta e versamento consentito

4.2.5 Caso 5: Password errata per tre volte consecutive

L'ultimo caso analizzato in figura 4.2.5 mostra il comportamento della macchina quando l'utente inserisce per tre volte consecutive una password errata. Quello che si osserva dalla figura è che ogni qualvolta la macchina giunge nello stato di errore E_4 si ha l'incremento del contatore (indicato con *count*) e poiché esso è associato al display a 7 segmenti si ha anche il cambio di stato dei bit che definiscono la wordline del display.
Quando si giunge nello stato di errore E_4 e il contatore ha raggiunto valore pari a 3 (0011 in binario), la macchina passa dallo stato di errore allo stato di stop quindi all'utente non è più permesso inserire la password poiché ha utilizzato tutti e tre i tentativi disponibili per inserire correttamente la password.
Dalla figura osserviamo inoltre che ogni volta che la macchina si trova nello stato di errore E_4, viene abilitato il *led_rosso* che è un riscontro visivo utile all'utente per capire se ha inserito correttamente la password o meno.

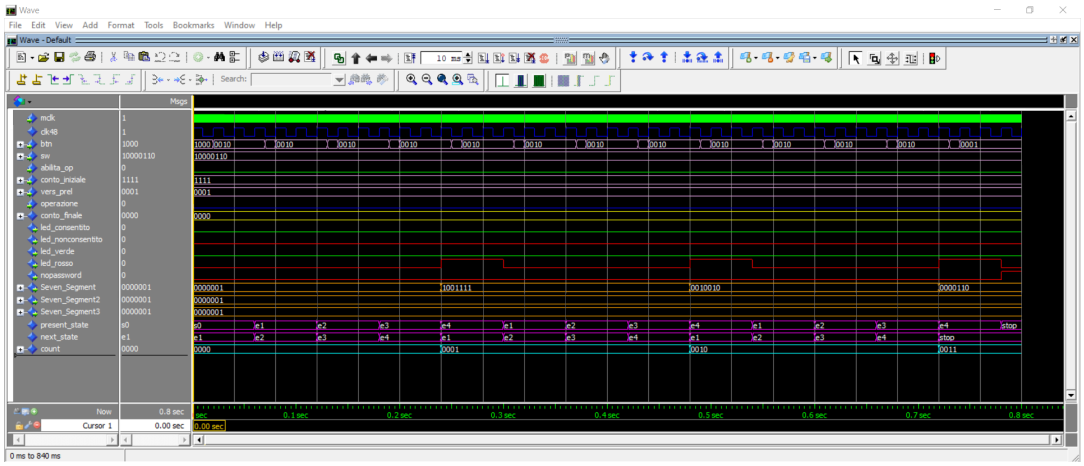

Figura 4.2.5: Password errata per tre volte consecutive

5 TimeQuest Timing Analyzer

Dopo aver realizzato su *Quartus II* il codice VHDL che permette di simulare il circuito digitale considerato e averne verificato il corretto funzionamento mediante *Modelsim*, è necessario verificare che la frequenza di funzionamento del circuito sia inferiore o tutt'al più uguale alla frequenza massima del clock della scheda FPGA considerata. Per poter verificare ciò è necessario utilizzare il tool *TimeQuest Timing Analyzer* integrato in *Quartus II*. Dopo aver creato la timing netlist, è necessario definire il periodo del clock della scheda: nel nostro caso la scheda FPGA utilizzata contiene un oscillatore con frequenza di 50 MHz. A questo punto per verificare che la frequenza massima di funzionamento del circuito realizzato sia inferiore alla frequenza massima della scheda, è necessario aprire il report denominato *ReportSetupSummary* in cui sono mostrati tutti i clock utilizzati nel circuito e per ognuno di essi lo slack.
Lo slack è il margine entro il quale viene soddisfatto o meno un requisito di tempo: uno slack positivo indica che il requisito temporale è soddisfatto (frequenza di funzionamento inferiore della frequenza del clock), uno slack negativo indica che esso non è soddisfatto (frequenza di funzionamento superiore della frequenza del clock) mentre uno slack nullo indica che la frequenza di funzionamento coincide perfettamente con la frequenza del clock.
In figura 5.1 è mostrato il *Report Setup Summary* nel caso in oggetto considerando una frequenza del clock di 50 MHz:

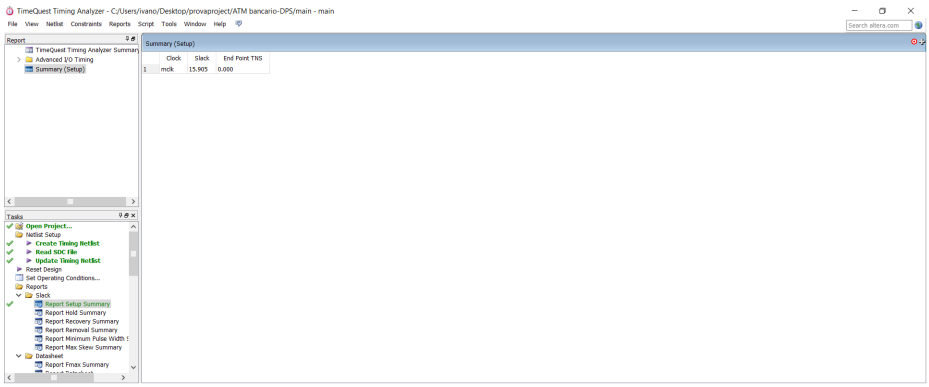

Figura 5.1: Slack con clock di 50 MHz

A questo punto cliccando con il tasto destro sul clock denominato *mclk* è possibile osservare anche più nel dettaglio l'analisi temporale precedente mediante la rappresentazione delle forme d'onda:

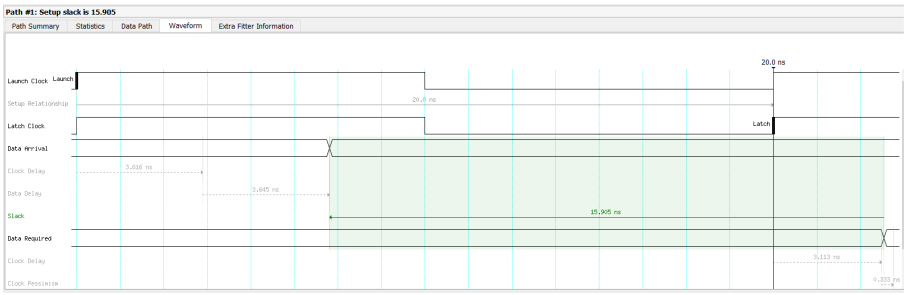

Figura 5.2: Forme d'onda temporali con clock a 50 MHz

A questo punto è necessario determinare la frequenza massima del clock entro la quale il circuito realizzato continua a funzionare correttamente. Per far ciò è necessario variare il clock di sistema.

Impostando il clock di sistema a 200 MHz otteniamo ancora uno slack positivo, come mostrato in 5.3:

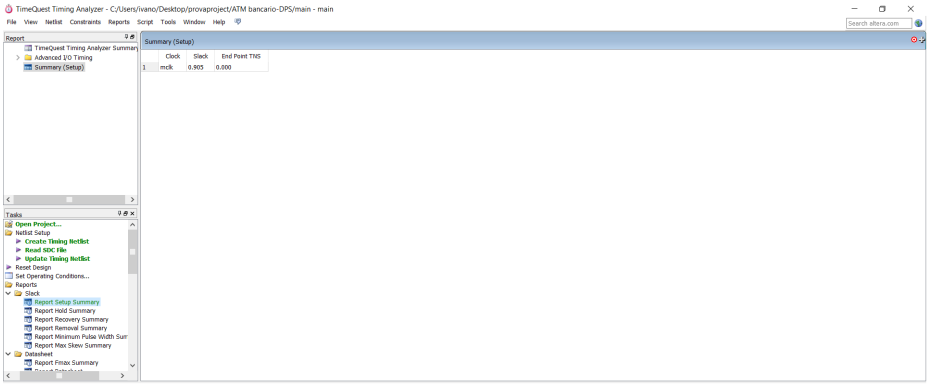

Figura 5.3: Slack con clock di 200 MHz

Aumentando la frequenza del clock a 250 MHz otteniamo uno slack negativo, come mostrato in figura 5.4:

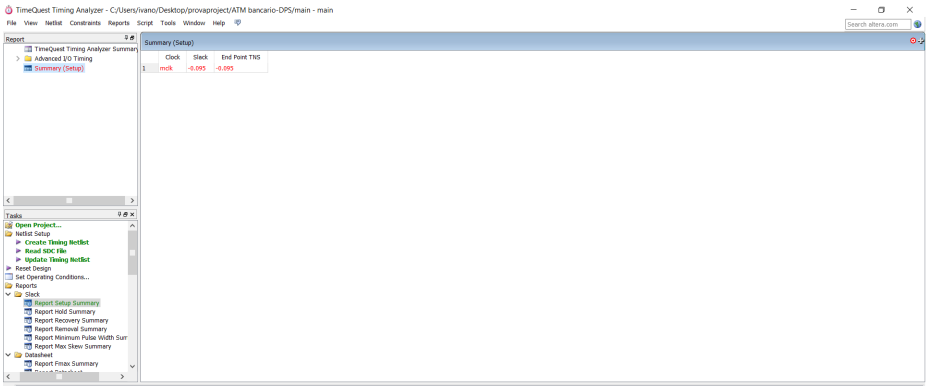

Figura 5.4: Slack con clock di 250 MHz

In definitiva è possibile affermare che il circuito digitale realizzato può funzionare per frequenze sicuramente inferiori a 200 MHz.

6 Pin Planner

Dopo aver analizzato la frequenza massima di funzionamento, è necessario associare ogni segnale definito nel codice VHDL ad uno specifico device presente sulla scheda FPGA. Per far questo è necessario aprire il tool *Pin Planner* di *Quartus II*. Ovviamente per conoscere quale pin corrisponde ad uno switch o ad un push-button è necessario utilizzare il datasheet della scheda FPGA utilizzata (essa deve essere definita come device anche su *Quartus II*, altrimenti non è possibile effetturare l'allocazione dei pin).
Nel caso in oggetto è stato considerata come scheda FPGA Altera CycloneV SE 5CSEMA5F31C6N. Nella seguente figura è mostrata la vista dall'alto della board considerata:

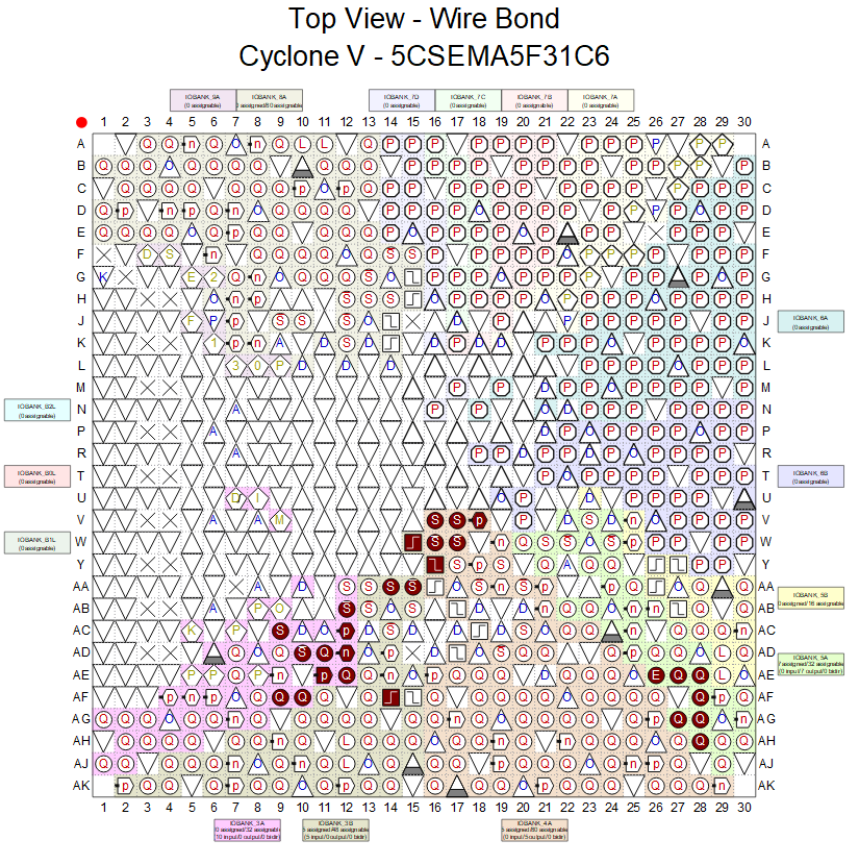

Figura 6.1: Vista dalll'alto della scheda FPGA considerata

Nel tool *Pin Planner* sono stati definiti per ogni segnale di ingresso e di uscita il pin a cui essi sono fisicamente connessi. I pin associati ai diversi segnali I/O sono mostrati in figura 6.2.
Come è possibile osservare, non tutti i segnali sono stati associati a dei pin perché la scheda utilizzata ha un numero limitato di switch (8 switch in particolare). In realtà per poter associare anche quei segnali a dei pin sarebbe stato possibile utilizzare le porte GPIO della scheda FPGA in modo da connettere ad essa una scheda millefori sulla quale sarebbero stati posizionati gli switch mancanti sulla scheda.

abilita_op	Input	PIN_AE12	3A	B3A_N0	PIN_AE12	2.5 V (default)		12mA (default)	
btn[3]	Input	PIN_Y16	3B	B3B_N0	PIN_Y16	2.5 V (default)		12mA (default)	
btn[2]	Input	PIN_W15	3B	B3B_N0	PIN_W15	2.5 V (default)		12mA (default)	
btn[1]	Input	PIN_AA15	3B	B3B_N0	PIN_AA15	2.5 V (default)		12mA (default)	
btn[0]	Input	PIN_AA14	3B	B3B_N0	PIN_AA14	2.5 V (default)		12mA (default)	
conto_iniziale[3]	Input				PIN_AH12	2.5 V (default)		12mA (default)	
conto_iniziale[2]	Input				PIN_AG15	2.5 V (default)		12mA (default)	
conto_iniziale[1]	Input				PIN_AJ14	2.5 V (default)		12mA (default)	
conto_iniziale[0]	Input				PIN_AE17	2.5 V (default)		12mA (default)	
led_consentito	Output	PIN_V16	4A	B4A_N0	PIN_V16	2.5 V (default)		12mA (default)	1 (default)
led_nonconsentito	Output	PIN_W16	4A	B4A_N0	PIN_W16	2.5 V (default)		12mA (default)	1 (default)
led_rosso	Output	PIN_V17	4A	B4A_N0	PIN_V17	2.5 V (default)		12mA (default)	1 (default)
led_verde	Output	PIN_V18	4A	B4A_N0	PIN_V18	2.5 V (default)		12mA (default)	1 (default)
mclk	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	2.5 V (default)		12mA (default)	
nopassword	Output	PIN_W17	4A	B4A_N0	PIN_W17	2.5 V (default)		12mA (default)	1 (default)
operazione	Input	PIN_AD10	3A	B3A_N0	PIN_AD10	2.5 V (default)		12mA (default)	
Seven_Segment[6]	Output	PIN_AH28	5A	B5A_N0	PIN_AH28	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[5]	Output	PIN_AG28	5A	B5A_N0	PIN_AG28	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[4]	Output	PIN_AF28	5A	B5A_N0	PIN_AF28	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[3]	Output	PIN_AG27	5A	B5A_N0	PIN_AG27	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[2]	Output	PIN_AE28	5A	B5A_N0	PIN_AE28	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[1]	Output	PIN_AE27	5A	B5A_N0	PIN_AE27	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment[0]	Output	PIN_AE26	5A	B5A_N0	PIN_AE26	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[6]	Output	PIN_AD27	5A	B5A_N0	PIN_AK12	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[5]	Output	PIN_AF30	5A	B5A_N0	PIN_AF18	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[4]	Output	PIN_AF29	5A	B5A_N0	PIN_AK11	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[3]	Output	PIN_AG30	5A	B5A_N0	PIN_AH10	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[2]	Output	PIN_AH30	5A	B5A_N0	PIN_AG17	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[1]	Output	PIN_AH29	5A	B5A_N0	PIN_AJ11	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment2[0]	Output	PIN_AJ29	5A	B5A_N0	PIN_AJ10	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[6]	Output	PIN_AC30	5B	B5B_N0	PIN_AC14	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[5]	Output	PIN_AC29	5B	B5B_N0	PIN_A8	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[4]	Output	PIN_AD30	5B	B5B_N0	PIN_AG25	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[3]	Output	PIN_AC28	5B	B5B_N0	PIN_AB15	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[2]	Output	PIN_AD29	5B	B5B_N0	PIN_AK7	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[1]	Output	PIN_AE29	5B	B5B_N0	PIN_AK8	2.5 V (default)		12mA (default)	1 (default)
Seven_Segment3[0]	Output	PIN_AB23	5A	B5A_N0	PIN_D10	2.5 V (default)		12mA (default)	1 (default)
sw[7]	Input	PIN_AC9	3A	B3A_N0	PIN_AC9	2.5 V (default)		12mA (default)	
sw[6]	Input	PIN_AE11	3A	B3A_N0	PIN_AE11	2.5 V (default)		12mA (default)	
sw[5]	Input	PIN_AD12	3A	B3A_N0	PIN_AD12	2.5 V (default)		12mA (default)	
sw[4]	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	2.5 V (default)		12mA (default)	
sw[3]	Input	PIN_AF10	3A	B3A_N0	PIN_AF10	2.5 V (default)		12mA (default)	
sw[2]	Input	PIN_AF9	3A	B3A_N0	PIN_AF9	2.5 V (default)		12mA (default)	
sw[1]	Input	PIN_AC12	3A	B3A_N0	PIN_AC12	2.5 V (default)		12mA (default)	
sw[0]	Input	PIN_AB12	3A	B3A_N0	PIN_AB12	2.5 V (default)		12mA (default)	
vers_prel[3]	Input				PIN_AH15	2.5 V (default)		12mA (default)	
vers_prel[2]	Input				PIN_AK13	2.5 V (default)		12mA (default)	
vers_prel[1]	Input				PIN_AK14	2.5 V (default)		12mA (default)	
vers_prel[0]	Input				PIN_AJ12	2.5 V (default)		12mA (default)	
<<new node>>									

Figura 6.2: Assegnazione dei pin ai segnali I/O

7 Conclusioni

Con questo progetto si è voluto realizzare il circuito logico di un Automated Teller Machine o ATM sfruttando il software di programmazione per schede FPGA *QuartusII*. Il corretto funzionamento logico della macchina è stato verificato con l'ausilio del software di simulazione *ModelSim* dove è stato possibile emulare gli input e verificare lo stato logico degli output della macchina, ad ogni colpo di clock, attraverso un'opportuna interfaccia grafica. E' stato necessario l'utilizzo di circuiti digitali quali il clock divisor e il clock pulse per l'implementazione via software di un divisore di frequenza e di un circuito antirimbalzo per i push-button utilizzati come dispositivi di input della macchina. Sviluppi futuri potrebbero prevedere ulteriori miglioramenti quali: l'utilizzo di un numero maggiore di switch per l'aumento della lunghezza della password da inserire o preferibilmente l'utilizzo di un tastierino numerico visto come periferica esterna alla macchina, l'utilizzo di un display LCD in sostituzione al display a 7 segmenti per poter mostrare un numero maggiore di cifre all'utente.

Appendice

Codice del progetto

```
--Digital Programmable Systems
--Prof.Ing Francesco DE LEONARDIS
--Progetto ATM bancario
--D'Alessandro Vito Ivano & Venezia Antonio
--Anno accademico 2019/2020


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

entity main is
port(
        --clock a 50 MHz
        mclk : in std_logic;

        --push-button
        btn : in std_logic_vector(3 downto 0);

        --switch per settare la password
        sw : in STD_LOGIC_VECTOR(7 downto 0);

        --switch per abilitare l'operazione di versamento/prelievo
        abilita_op : in STD_LOGIC:='0';

        --capitale iniziale presente nel conto corrente
        conto_iniziale : in STD_LOGIC_VECTOR(3 downto 0);

        --capitale che si vuole prelevare o versare
        vers_prel : in STD_LOGIC_VECTOR(3 downto 0);

        --indica quale operazione si sta effettuando(1 prelievo,0 immissione)
        operazione : in STD_LOGIC;

        --operazione consentita se questo led è acceso
        led_consentito : out STD_LOGIC:='0';

        --operazione non consentita se questo led è acceso
        led_nonconsentito : out STD_LOGIC:='0';

        --password inserita correttamente
        led_verde : out STD_LOGIC :='0';

        --password errata
        led_rosso : out STD_LOGIC :='0';

        --led abilitato se sono nella fase di stop
        nopassword : out std_logic;
```

```
        --display a 7 segmenti per contare numero di errori
        --nell'inserimento password
        Seven_Segment : out std_logic_vector(6 downto 0);

        --display a 7 segmenti per conto finale
        Seven_Segment2 : out std_logic_vector(6 downto 0);

        --display a 7 segmenti per conto finale
        Seven_Segment3 : out std_logic_vector(6 downto 0));

end main;

architecture main of main is

--Componente sommatore/sottrattore per operazioni di prelievo e immissione denaro

component addsub is
port(
        --segnale di clear
        clear : in std_logic;

        --segnale di enable per far effettuare l'operazione
        enb : in std_logic;

        --switch per scegliere quale operazione effettuare(1 se prelevo, 0 se verso denaro)
        OP: in std_logic;

        --A è il capitale iniziale, B è ciò che si vuole prelevare
        A,B  : in std_logic_vector(3 downto 0);

        --capitale finale
        conto : out std_logic_vector(3 downto 0);

        --led acceso se l'operazione scelta è consentita
        consentito : out std_logic;

        --led acceso se l'operazione scelta non è consentita
        nonconsentito : out std_logic;

        --flag per capire se l'operazione è terminata
        termine_op : out std_logic);
end component;

--Divisore di clock per passare da una frequenza di 50 MHz ad una frequenza di 48 Hz

component clkdiv is
port(
        --clock dell'FPGA a 50 MHz
        mclk : in std_logic;
```

```
        --clear
        clr : in std_logic;

        --clock a 48 Hz
        clk48  : out std_logic);

end component;

--Circuito di debouncing per i push-button

component clock_pulse is
port(
        --pressione dei push-button
        inp : in std_logic;

        --clock a 48 Hz
        cclk : in std_logic;

        --clear
        clr: in std_logic;

        --output di tipo impulsivo
        outp  : out std_logic);

end component;

--Macchina a stati finiti per verifica della correttezza della password inserita

type state_type is(s0,s1,s2,s3,s4,e1,e2,e3,e4,stop);

--Segnali utilizzati per il funzionamento del circuito digitale realizzato

signal present_state,next_state :state_type;
signal count : std_logic_vector(3 downto 0) :="0000";
signal fallimento : std_logic;
signal nopass : std_logic;
signal ledverde : std_logic:='0';
signal enb_addsub : std_logic:='0';
signal op_terminata : std_logic;
signal clear : std_logic;
signal clk48 : std_logic;
signal clkp : std_logic;
signal btn012 : std_logic;
signal bn : std_logic_vector(1 downto 0);
signal conto_fin : std_logic_vector(3 downto 0);

begin
enb_addsub <= (ledverde and abilita_op);
clear <= btn(3);
btn012 <= btn(0) or btn(1) or btn(2);
bn(1) <= btn(2);
```

```
bn(0) <= btn(1);

--Divisore di clock

DIVISORE : clkdiv port map(mclk,clear,clk48);

--Circuito antirimbalzo per i push-button

DEBOUNCING : clock_pulse port map(btn012,clk48,clear,clkp);

--Registro della FSM realizzata

sreg : process(clkp,clear)
        begin
            if (clear = '1') then
                    present_state <= s0;
            else
                    if clkp'event and clkp = '1' then
                            present_state <= next_state;
                    end if;
            end if;
            end process;

--Funzionamento della macchina a stati finiti in base alla password inserita

C1 : process(present_state,bn,sw, count,op_terminata)
        begin
            case present_state is
                    when s0 =>
                            if bn=sw(7 downto 6) then
                                    next_state <= s1;
                            else
                                    next_state <= e1;
                            end if;
                    when s1 =>
                            if bn = sw(5 downto 4) then
                                    next_state <= s2;
                            else
                                    next_state <= e2;
                            end if;
                    when s2 =>
                            if bn = sw(3 downto 2) then
                                    next_state <= s3;
                            else
                                    next_state <= e3;
                            end if;
                    when s3 =>
                            if bn = sw(1 downto 0) then
                                    next_state <= s4;
                            else
                                    next_state <= e4;
```

```
                        end if;
                when s4 =>
                        if op_terminata = '0' then
                                next_state <= s4;
                        else
                                next_state <= stop;
                        end if;
                when e1 =>
                        next_state <= e2;
                when e2 =>
                        next_state <= e3;
                when e3 =>
                        next_state <= e4;
                when e4 =>
                        if (count = "0011") then
                                next_state <= stop;
                        else
                                if bn = sw(7 downto 6) then
                                        next_state <= s1;
                                else
                                        next_state <= e1;
                                end if;
                        end if;
                when stop =>
                        next_state <= stop;
                when others =>
                        next_state <= s0;
            end case;
end process;

--Gli output di tale processo sono:
--led_verde abilitato se password corretta
--led rosso abilitato se password errata
--nopassword 1 se ho terminato i 3 tentativi di inserimento password

C2 : process(present_state,count)
        begin
                if present_state = s4 then
                        ledverde <= '1';
                else
                        ledverde <= '0';
                end if;
                if present_state = e4 then
                        fallimento <= '1';
                        count <= STD_LOGIC_VECTOR(unsigned(count)+1);
                else
                        fallimento <= '0';
                end if;
                if present_state = stop then
                        nopass <='1';
                else
```

```
                        nopass <= '0';
                end if;
end process;
led_rosso <= fallimento;
led_verde <= ledverde;
nopassword <=nopass;

--Display a 7 segmenti che mostra il numero di tentativi
--falliti nell'inserimento password

display : process(count)
        begin
            case count is
                    when "0000" =>
                                    Seven_Segment <= "0000001"; ---0
                    when "0001" =>
                                    Seven_Segment <= "1001111"; ---1
                    when "0010" =>
                                    Seven_Segment <= "0010010"; ---2
                    when "0011" =>
                                    Seven_Segment <= "0000110"; ---3
                    when "0100" =>
                                    Seven_Segment <= "1001100"; ---4
                    when others =>
                                    Seven_Segment <= "0000001"; ---0
              end case;
  end process;

--Sommatore/sottrattore per il calcolo del prelievo
--o versamento di capitale nel conto corrente

AS : addsub port map(
                        clear,
                        enb_addsub,
                        operazione,
                        conto_iniziale,
                        vers_prel,
                        conto_fin,
                        led_consentito,
                        led_nonconsentito,
                        op_terminata);


display1 : process(conto_fin)
        begin
            case conto_fin is
                    when "0000" =>
                                    Seven_Segment2 <= "0000001"; ---0
                                    Seven_Segment3 <= "0000001"; ---0
                    when "0001" =>
                                    Seven_Segment2 <= "1001111"; ---1
```

```
                                    Seven_Segment3 <= "0000001"; ---0
                when "0010" =>
                                    Seven_Segment2 <= "0010010"; ---2
                                    Seven_Segment3 <= "0000001"; ---0
                when "0011" =>
                                    Seven_Segment2 <= "0000110"; ---3
                                    Seven_Segment3 <= "0000001"; ---0
                when "0100" =>
                                    Seven_Segment2 <= "1001100"; ---4
                                    Seven_Segment3 <= "0000001"; ---0
                when "0101" =>
                                    Seven_Segment2 <= "1011011"; ---5
                                    Seven_Segment3 <= "0000001"; ---0
                when "0110" =>
                                    Seven_Segment2 <= "0011111"; ---6
                                    Seven_Segment3 <= "0000001"; ---0
                when "0111" =>
                                    Seven_Segment2 <= "1110000"; ---7
                                    Seven_Segment3 <= "0000001"; ---0
                when "1000" =>
                                    Seven_Segment2 <= "1111111"; ---8
                                    Seven_Segment3 <= "0000001"; ---0
                when "1001" =>
                                    Seven_Segment2 <= "1110011"; ---9
                                    Seven_Segment3 <= "0000001"; ---0
                when "1010" =>
                                    Seven_Segment2 <= "0000001"; ---0
                                    Seven_Segment3 <= "1001111"; ---1
                when "1011" =>
                                    Seven_Segment2 <= "1001111"; ---1
                                    Seven_Segment3 <= "1001111"; ---1
                when "1100" =>
                                    Seven_Segment2 <= "0010010"; ---2
                                    Seven_Segment3 <= "1001111"; ---1
                when "1101" =>
                                    Seven_Segment2 <= "0000110"; ---3
                                    Seven_Segment3 <= "1001111"; ---1
                when "1110" =>
                                    Seven_Segment2 <= "1001100"; ---4
                                    Seven_Segment3 <= "1001111"; ---1
                when "1111" =>
                                    Seven_Segment2 <= "1011011"; ---5
                                    Seven_Segment3 <= "1001111"; ---1
                when others =>
                                    Seven_Segment2 <= "0000001"; ---0
                                    Seven_Segment3 <= "0000001"; ---0
            end case;
  end process;
end main;
```

Divisore di clock

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;

--Divisore di clock

entity clkdiv is
port(
        --clock a 50 MHz
        mclk : in STD_LOGIC;

        --segnale di clear
        clr : in STD_LOGIC;

        --clock a 48 Hz
        clk48 : out STD_LOGIC);

end clkdiv;


architecture clkdiv of clkdiv is
signal q : STD_LOGIC_VECTOR(23 downto 0);
begin
        process(mclk,clr)
                begin
                        if clr = '1' then
                                q <= X"000000";
                        elsif mclk'event and mclk = '1' then
                                q <= q+1;
                        end if;
                end process;
        clk48 <= q(19);    --48 Hz
end clkdiv;
```

Circuito antirimbalzo

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

--Circuito antirimbalzo per i push-button

entity clock_pulse is
port(
        --pressione del button
        inp : in STD_LOGIC;

        --segnale di clock
        cclk : in STD_LOGIC;

        --segnale di clear
        clr : in STD_LOGIC;

        --output di tipo impulsato
        outp : out STD_LOGIC

    );

end clock_pulse;

architecture clock_pulse of clock_pulse is
signal delay1,delay2,delay3 : STD_LOGIC;
begin
        process(cclk,clr)
                begin
                        if clr='1' then
                                delay1 <= '0';
                                delay2 <= '0';
                                delay3 <= '0';
                        elsif cclk'event and cclk='1' then
                                delay1 <= inp;
                                delay2 <= delay1;
                                delay3 <= delay2;
                        end if;
        end process;
outp <= delay1 and delay2 and not delay3;
end clock_pulse;
```

Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

--Full Adder

entity Full_Adder is
port(
        --segnale di enable
        enable : in std_logic;

        --bit di ingresso
        X : in std_logic;
        Y : in std_logic;

        --riporto di ingresso
        Cin : in std_logic;

        --risultato
        sum : out std_logic;

        --riporto di uscita
    Cout : out std_logic);

end Full_Adder;

architecture bhv of Full_Adder is
begin
        process(enable,X,Y,Cin)
                begin
                        if(enable='1') then
                                sum <= (X xor Y) xor Cin;
                                Cout <= (X and Y) OR (X and Cin) OR (Y and Cin);
                        else
                                sum <= '0';
                                Cout <= '0';
                        end if;
        end process;
end bhv;
```

Circuito sommatore/sottrattore a 4 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

--Sommatore/sottratore a 4 bit

entity addsub is
port(
        --segnale di clear
        clear : in std_logic;

        --segnale di enable
        enb : in std_logic;

        --bit di controllo
        --differenza se OP=1
        --somma se OP=0
        OP : in std_logic;

        --capitale iniziale
        A : in std_logic_vector(3 downto 0);

        --capitale che si vuole
        --prelevare o versare
        B : in std_logic_vector(3 downto 0);

        --capitale finale
        conto : out std_logic_vector(3 downto 0);

        --led abilitato se
        --operazione consentita
        consentito : out std_logic;

        --led abilitato se
        --operazione non consentita
        nonconsentito : out std_logic;

        --flag per capire se è terminata
        --l'operazione
        termine_op : out std_logic);

end addsub;

architecture struct of addsub is

--Full Adder usato come componente

component Full_Adder is
  port( enable, X, Y, Cin : in std_logic;
        sum, Cout : out std_logic);
end component;
```

```
signal C1, C2, C3, C4: std_logic;
signal TMP: std_logic_vector(3 downto 0);
signal ovf : std_logic;
signal risultato : std_logic_vector(3 downto 0);

begin
        TMP(0)<= OP xor B(0);
        TMP(1)<= OP xor B(1);
        TMP(2)<= OP xor B(2);
        TMP(3)<= OP xor B(3);
        FA0:Full_Adder port map(enb,A(0),TMP(0),OP, risultato(0),C1);-- R0
        FA1:Full_Adder port map(enb,A(1),TMP(1),C1, risultato(1),C2);-- R1
        FA2:Full_Adder port map(enb,A(2),TMP(2),C2, risultato(2),C3);-- R2
        FA3:Full_Adder port map(enb,A(3),TMP(3),C3, risultato(3),C4);-- R3
        ovf <= C4 ;

--Processo di verifica dei risultati

verifica : process(OP,C4,ovf,risultato,A,enb,clear)
        begin
        if(clear='1') then
         consentito <= '0';
         nonconsentito <= '0';
         conto <= "0000";
         termine_op <= '0';
        else
                if(enb='1') then
                        if(OP='1') then
                                if(C4='1') then
                                        consentito <= '1';
                                        nonconsentito <= '0';
                                        conto <= risultato;
                                        termine_op <= '1';
                                else
                                        consentito <= '0';
                                        nonconsentito <= '1';
                                        conto <= A;
                                        termine_op <= '0';
                                end if;
                        else
                                if(ovf='0') then
                                        consentito <= '1';
                                        nonconsentito <= '0';
                                        conto <= risultato;
                                        termine_op <= '1';
                                else
                                        consentito <= '0';
                                        nonconsentito <= '1';
                                        conto <= A;
                                        termine_op <= '0';
```

```
                                end if;
                        end if;
                else
                        consentito <='0';
                        nonconsentito <='0';
                        conto <= "0000";
                        termine_op <= '0';
                end if;
        end if;
end process;
end struct;
```