

Sistema di controllo di un ascensore

BY ANTONIO BERNARDINI

Table of contents

1 Homework	1
1.1 Introduzione	1
1.2 Gestione delle chiamate ai piani e movimento	2
1.3 Codifiche	2
1.4 Ingressi	3
1.5 Come realizzare il circuito	3
1.6 Suggerimenti	3
2 Soluzione	3
2.1 Introduzione	3
2.2 Scelte progettuali	4
2.3 Display a 7 segmenti per i piani	4
2.4 Display a 16 segmenti per gli stati	6
2.5 Bufferizzazione delle richieste in attesa	8
2.6 Gestione dei reset	9
2.7 Rilevatore per un generico piano	9
2.8 Segnale per l'arrivo a destinazione	10
2.9 Gestione delle richieste	10
2.10 Bufferizzazione della direzione	14
2.11 Automa a stati finiti per la gestione degli stati	15
2.11.1 Display a 7 segmenti per il timer	16
2.11.2 Gestione del timer	17
2.12 Automa a stati finiti per la gestione dei piani	18
2.13 Conclusione	19

1 Homework

1.1 Introduzione

Per questo homework è stato richiesto di progettare e realizzare il sistema di controllo di un ascensore in un palazzo di 3 piani, cui si aggiunge il piano terra ed un ultimo *piano VIP*, descritto più in basso. L'ascensore può essere nei seguenti stati:

- *Attesa*: l'ascensore aspetta che venga chiamato a un piano.
- *Movimento verso l'alto o il basso*: l'ascensore è in movimento in una direzione specifica.
- *Arresto*: l'ascensore si ferma all'arrivo ad un piano in cui è stata effettuata una prenotazione.
- *Apertura delle porte*: le porte si aprono e restano aperte per 9 secondi.

Ad ogni piano è installato un sensore di arrivo che, quando rileva la presenza dell'ascensore, solleva un segnale A_x per indicare che esso è arrivato al piano x .

1.2 Gestione delle chiamate ai piani e movimento

Il passaggio ad un piano può essere prenotato sia dal piano stesso, sia dall'interno dell'ascensore, con l'apposita pulsantiera. È possibile prenotare il passaggio a qualsiasi piano in qualsiasi stato, anche mentre l'ascensore è in movimento. La chiamata ad un determinato piano viene bufferizzata in un elemento di memoria appositamente dedicato e permane nello stato di "richiesta effettuata" fino al momento in cui il l'ascensore arriva al piano coinvolto dalla richiesta.

Se già in movimento, l'ascensore procede nel suo senso di marcia fino a quando non ha raggiunto l'ultimo piano per cui è presente una chiamata, in quella direzione. Ad esempio, se l'ascensore ha raggiunto il secondo piano in salita e sono presenti due chiamate, una al terzo ed una al primo, l'ascensore darà priorità al terzo piano poiché raggiungibile nello stesso senso di marcia.

Viceversa, quando l'ascensore raggiunge un piano e sono unicamente presenti chiamate da piani raggiungibili nel senso di marcia opposto, esso invertirà il senso di marcia.

Se, al momento della chiusura delle porte, non è presente alcuna chiamata, l'ascensore rimarrà (a porte chiuse) nello stato di attesa. Se in tale stato l'ascensore riceve una chiamata dal piano in cui è presente, si limiterà ad aprire le porte.

Questo funzionamento di base è modificato dall'ultimo (quarto) piano: il *piano VIP*. Se è stata ricevuta una richiesta di passaggio al *piano VIP*, se l'ascensore sta procedendo in quella direzione (quindi verso l'alto), *salterà tutte le altre fermate* fino al raggiungimento del *piano VIP*. Viceversa, se si sta muovendo in direzione opposta (quindi verso il basso), raggiungerà soltanto il prossimo piano per cui è stata ricevuta una richiesta di passaggio e poi invertirà il senso di marcia.

1.3 Codifiche

L'ascensore deve tenere traccia dei seguenti stati, utilizzando le seguenti codifiche:

z_2	z_1	z_0	Stato
0	0	0	Attesa
0	0	1	Movimento verso l'alto
0	1	0	Movimento verso il basso
0	1	1	Porte aperte
1	0	0	Arresto

Table 1. Codifica degli stati dell'ascensore

Lo stato di arresto corrisponde al momento in cui l'ascensore arriva ad un piano per cui è stata richiesta la fermata, segnalato dal segnale A_x . Dopo essersi arrestato, l'ascensore dovrà aprire le porte e mantenerle aperte per 9 secondi.

L'ascensore deve anche tenere traccia del piano in cui si trova in un determinato istante temporale. Per questo motivo, vengono utilizzate le seguenti variabili:

y_2	y_1	y_0	Piano
0	0	0	Terra
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	VIP

Table 2. Codifica dei piani dell'ascensore

All'avvio del sistema si può assumere che l'ascensore si trovi al piano T nello stato di attesa.

1.4 Ingressi

Gli ingressi al sistema di controllo dell'ascensore sono di due famiglie: la prima famiglia è costituita dai segnali A_x precedentemente citati che vengono sollevati quando un ascensore raggiunge un piano. Tali segnali valgono 1 solo nel momento in cui l'ascensore è rilevato, per poi tornare a zero.

La seconda famiglia è costituita dai segnali P_x che indicano la chiamata al piano x . Tali segnali valgono 1 soltanto quando il tasto di chiamata viene premuto, per poi tornare a zero. È quindi necessario bufferizzare tali segnali in opportuni modi.

1.5 Come realizzare il circuito

La soluzione circuitale deve essere inserita all'interno del file `circuit.dig`, che può essere modificato utilizzando l'editor e simulatore di circuiti [Digital](#). Nel file `circuit.dig` sono già specificati gli ingressi e le uscite del circuito.

1.6 Suggerimenti

1. Per misurare il tempo, si può utilizzare un *clock* impostato ad una frequenza opportuna e “contare” il passaggio del tempo con un numero opportuno di stati. Ad esempio, per contare 5 secondi, si può utilizzare un clock impostato a 1Hz e utilizzare 5 stati. Il clock in `circuit.dig` è già impostato a 1Hz.
2. È possibile *decomporre* una singola macchina a stati in più macchine a stati che controllino il funzionamento di parti differenti del sistema in maniera coordinata. Lo stato di una macchina a stati può essere utilizzata come input di un'altra. Allo stesso modo, l'output di una macchina a stati può diventare l'input di un'altra. In questo modo, è possibile studiare separatamente i problemi e comporre una soluzione finale.
3. Le transizioni di stato non sono necessariamente dovute al cambiamento di una sola variabile in ingresso. I “caratteri in input” in questo caso possono corrispondere ad un valore booleano calcolato da una funzione booleana arbitraria, che processa segnali in input al sistema e variabili memorizzate in opportuni elementi di memoria in maniera consona. Alcune transizioni possono essere “automatiche”, semplicemente legate alla ricezione di un certo numero di colpi di clock.

2 Soluzione

2.1 Introduzione

Analizzando il file `circuit.dig` per la prima volta notiamo la presenza di 10 input, rispettivamente P_0, \dots, P_3, P_{VIP} per le chiamate e A_0, \dots, A_3, A_{VIP} per gli arrivi, di 6 transizioni, rispettivamente y_2, y_1, y_0 per i piani e z_2, z_1, z_0 per gli stati, e infine del *clock* impostato ad 1Hz. Inoltre non è presente un output esplicito, infatti le transizioni, prima di diventare tali, “passano” attraverso dei flip-flop D per definizione di automa a stati finiti e, nel caso dell'ascensore, essa non ha un vero e proprio

output, ma occorrerà che mostri sia il piano che lo stato in un determinato istante temporale, pertanto tali transizioni fungono anche da output. Ci troviamo davanti ad un problema che può essere certamente modellato con le macchine di Mealy e/o Moore, ma non direttamente. Infatti occorre semplificare il problema utilizzando la famosa tecnica del *divide et impera* e vedremo in questo documento come fare. Buona lettura.

2.2 Scelte progettuali

Per quanto riguarda le scelte progettuali, vorrei partire dalla fine. In particolare implementeremo un sistema di visualizzazione dei piani e degli stati dell'ascensore utilizzando un display a 7 segmenti per i piani e 4 display a 16 segmenti per gli stati. Infatti mantenendo gli output del file `circuit.dig` avremmo una codifica binaria dei piani e degli stati e questo comporterebbe una più difficile lettura da parte di un utilizzatore medio che si presume non sia un ingegnere. Successivamente, come richiesto dalle specifiche, dovremmo bufferizzare le chiamate P_x in degli appositi elementi di memoria. Per esempio, dei flip-flop SR sono più che sufficienti per fare ciò. Capiremo come gestire i segnali di RESET dei flip-flop SR e andremo a generare diversi segnali “personalizzati”, come DESTINATION e CK_2 , utili per risolvere piccoli problemi. Successivamente cercheremo di analizzare il problema della gestione delle richieste utilizzando appositi segnali UP, STOP, DOWN per gestire le priorità attraverso un circuito dedicato, e allo stesso tempo andremo a bufferizzare la direzione precedente dell'ascensore, sempre in appositi elementi di memoria. Infine ci occuperemo della realizzazione degli automi a stati finiti, sia per gli stati che per i piani, con annesso il circuito dedicato per la gestione del Timer. Iniziamo!

2.3 Display a 7 segmenti per i piani

Sembra assurdo iniziare la progettazione partendo dalla fine, ma non è così. Infatti, come citato precedentemente, non è un scelta causale perchè nello stesso modo un cui un utilizzatore medio avrà una lettura semplificata dei piani e dello stato dell'ascensore, anche un ingegnere in fase di debugging può trarne beneficio. Infatti è semplicissimo sbagliare la posizione di 1 bit e leggere, ad esempio, il piano $y_2 = 0, y_1 = 1, y_0 = 0$ (piano 2) al posto del piano $y_2 = 1, y_1 = 0, y_0 = 0$ (*piano VIP*). Questo discorso vale anche per gli stati naturalmente.

Pertanto iniziamo la nostra progettazione implementando un display a 7 segmenti per la visualizzazione dei piani. Per fare ciò è chiaro che occorre convertire un input binario (cioè y_2, y_1, y_0) in un numero decimale corrispondente. Pertanto possiamo implementare un semplice decodificatore che per definizione fa proprio questo. Naturalmente ci conviene implementare il circuito combinatorio utilizzando una ROM (Read Only Memory) per rendere la progettazione più pulita possibile. Capito questo, è necessario sapere che un display a 7 segmenti non è altro che un insieme di diodi led connessi che vengono accesi in base al valore decodificato di input. In generale un display a 7 segmenti si presenta come segue:

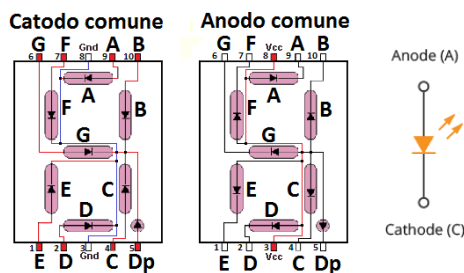


Figure 1. Display a 7 segmenti

In particolare utilizzeremo la configurazione a catodo comune che, come si intuisce dal nome, presenta dei diodi led che condividono tutti lo stesso catodo connesso a GND. Come mostrato in Fig. 1 ogni input del display a 7 segmenti accenderà uno o più diodi led illuminando di conseguenza una o più linee. Pertanto, per rappresentare correttamente ogni numero, possiamo fare riferimento alla seguente immagine:



Figure 2. Codifiche dei numeri per un display a 7 segmenti

Per esempio per scrivere il numero 1 dovremmo settare $b = 1$ e $c = 1$. Quindi, come dicevo precedentemente, dovremmo implementare un decodificatore per fare in modo di attivare b, c o qualsiasi altro input utilizzando l'input binario y_2, y_1, y_0 . Pertanto possiamo costruire la seguente tabella di verità:

Piano	y_2	y_1	y_0	a	b	c	d	e	f	g	HEX
0	0	0	0	1	1	1	1	1	1	0	0x7E
1	0	0	1	0	1	1	0	0	0	0	0x30
2	0	1	0	1	1	0	1	1	0	1	0x6D
3	0	1	1	1	1	1	1	0	0	1	0x79
PIV	1	0	0	0	1	1	0	0	1	1	0x33

Table 3. Tabella di verità del decodificatore per il display a 7 segmenti

A questo punto abbiamo tutto il necessario per la realizzazione del decodificatore, quindi poiché avremmo 3 bit di input e 7 bit di output dovremmo settare la ROM con le seguenti impostazioni:

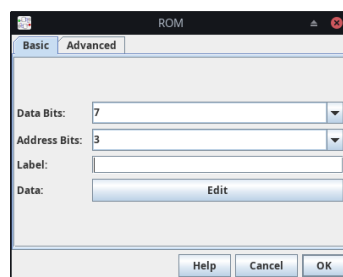


Figure 3. Impostazioni ROM per il display a 7 segmenti

inserendo nel campo **Data** la colonna HEX presente in Tabella 3. Pertanto il circuito finale è il seguente:

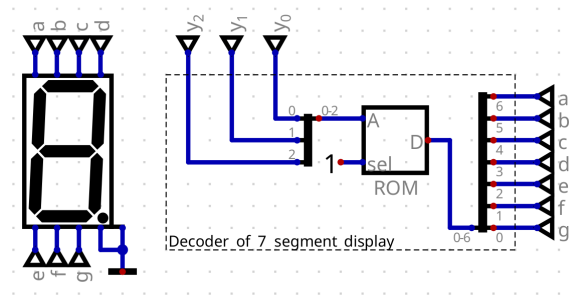


Figure 4. Decodificatore per il display a 7 segmenti

Infine l'input "dp" del display a 7 segmenti gestisce il punto, ma dato che non ci serve è stato collegato direttamente a GND.

2.4 Display a 16 segmenti per gli stati

Questa progettazione è simile alla precedente anche se c'è un fattore importante da tenere in considerazione: il numero di bit! Infatti poichè dobbiamo implementare 4 display a 16 segmenti avremmo bisogno di $16 \cdot 4 = 64$ bit. Tuttavia il simulatore Digital riesce a realizzare ROM per un massimo di 32 bit ciascuna. Pertanto andremo ad implementare due ROM. La prima per la gestione dei primi 32 bit "meno significativi" e la seconda per la gestione degli ultimi 32 bit "più significativi". Ma perchè occorre usare 4 display a 16 segmenti? Non basterebbe usare di nuovo un display a 7 segmenti? Beh, con un display a 7 segmenti visualizzeremmo solo i numeri da 0 a 4 proprio come i piani. Ma nel caso degli stati dobbiamo dare qualche informazione in più. Infatti possiamo fare la seguente associazione:

Stato	D_1	D_2	D_3	D_4
Attesa	<i>W</i>	<i>A</i>	<i>I</i>	<i>T</i>
Movimento verso l'alto	<i>U</i>	<i>P</i>		
Movimento verso il basso	<i>D</i>	<i>O</i>	<i>W</i>	<i>N</i>
Porte aperte	<i>O</i>	<i>P</i>	<i>E</i>	<i>N</i>
Arresto	<i>S</i>	<i>T</i>	<i>O</i>	<i>P</i>

Table 4. Conversione degli stati dell'ascensore nei caratteri rappresentabili nei 4 display a 16 segmenti

Per fortuna $\frac{3}{4}$ degli stati utilizzeranno esattamente tutti e 4 i display e solo nel caso dello stato UP si utilizzeranno 2 display per la rappresentazione dei singoli caratteri. Continuiamo la nostra disquisizione tenendo presente che in generale un display a 16 segmenti si presenta come segue:

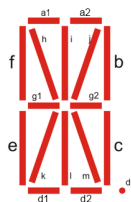


Figure 5. Display a 16 segmenti

quindi per le rappresentazioni corrette dei caratteri possiamo fare riferimento alla seguente immagine:

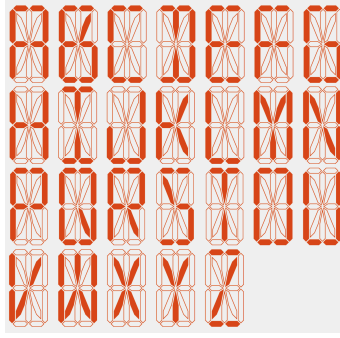


Figure 6. Codifiche dei caratteri per un display a 16 segmenti

Siamo pronti per costruire la tabella di verità per i primi 2 display cioè per i primi 32 bit “meno significativi”:

Display D_1																					
Stato	z_2	z_1	z_0	a_1	a_2	b	c	d_2	d_1	e	f	g_1	g_2	h	i	j	m	l	k	Carattere	HEX
WAIT	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	0	1	0	1	W	0xA8CC
UP	0	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	U	0x00FC
DOWN	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	0	D	0x483F
OPEN	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	O	0x00FF
STOP	1	0	0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	0	S	0x063B
Display D_2																					
Stato	z_2	z_1	z_0	a_1	a_2	b	c	d_2	d_1	e	f	g_1	g_2	h	i	j	m	l	k	Carattere	HEX
WAIT	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0	0	0	0	A	0x03CF
UP	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	P	0x03C7
DOWN	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	O	0x00FF
OPEN	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	P	0x03C7
STOP	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	T	0x4803

Table 5. Tabella di verità del decodificatore per i display D_1 e D_2 a 16 segmenti

Si noti che nella conversione da binario ad esadecimale ho considerato la sequenza invertita, ossia k, l, m, \dots in modo tale che il bit in posizione 0 sia a_1 mentre il bit in posizione 15 sia k (considerando sempre gruppi di 4 bit nella conversione da binario a esadecimale). Questo perchè nel simulatore Digital, quando si usa il display a 16 segmenti, il bit in posizione 0 è proprio a_1 pertanto occorre codificare in modo opportuno i valori in esadecimale da inserire nella ROM. Continuiamo dunque con i restanti 2 display cioè per i secondi 32 bit “più significativi”:

Display D_3																					
Stato	z_2	z_1	z_0	a_1	a_2	b	c	d_2	d_1	e	f	g_1	g_2	h	i	j	m	l	k	Carattere	HEX
WAIT	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	0	0	1	0	I	0x4833
UP	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0x0000
DOWN	0	1	0	0	0	1	1	0	0	1	1	0	0	0	1	0	1	0	1	W	0xA8CC
OPEN	0	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	E	0x03F3
STOP	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	O	0x00FF
Display D_4																					
Stato	z_2	z_1	z_0	a_1	a_2	b	c	d_2	d_1	e	f	g_1	g_2	h	i	j	m	l	k	Carattere	HEX
WAIT	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	T	0x4803
UP	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0x0000
DOWN	0	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	0	N	0x24CC
OPEN	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	0	N	0x24CC
STOP	1	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	P	0x03C7

Table 6. Tabella di verità del decodificatore per i display D_3 e D_4 a 16 segmenti

A questo punto abbiamo tutto il necessario per la realizzazione del decodificatore, quindi poichè avremmo 3 bit di input e 32 bit di output dovremmo settare le due ROM con le seguenti impostazioni:

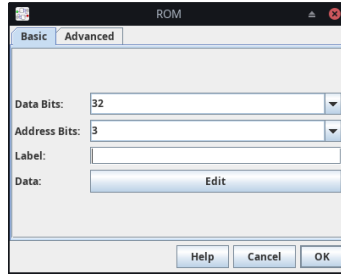



Figure 7. Impostazioni ROM per il display a 16 segmenti


inserendo nel campo **Data** la colonna HEX presente sia in Tabella 5 che in Tabella 6 come segue:



Data

File	
Address...	Value
0x0	3CFA8CC
0x1	3C700FC
0x2	FF483F
0x3	3C700FF
0x4	4803063B
0x5	0
0x6	0
0x7	0

OK



Data

File	
Address...	Value
0x0	0x48034833
0x1	0
0x2	24CCA8CC
0x3	24CC03F3
0x4	3C700FF
0x5	0
0x6	0
0x7	0

OK

ROM per D_1 e D_2 ROM per D_3 e D_4

Figure 8. Campo Data delle due ROM

come si può notare i valori in esadecimale sono stati disposti in modo tale che per il display D_1 completiamo i primi 16 bit della prima ROM e per il display D_2 completiamo gli ultimi 16 bit della prima ROM per un totale di 32 bit. Lo stesso vale per i display D_3 e D_4 ma per la seconda ROM. Pertanto il circuito finale è il seguente:

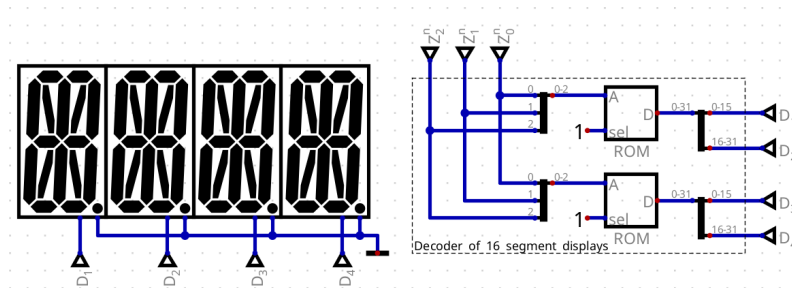


Figure 9. Decodificatore per il display a 16 segmenti

Infine l'input "dp" del display a 16 segmenti gestisce il punto, ma dato che non ci serve è stato collegato direttamente a GND.

2.5 Bufferizzazione delle richieste in attesa

Entriamo nel vivo della progettazione ed in particolare utilizziamo dei flip-flop SR per bufferizzare le chiamate P_x . Un flip-flop SR è un circuito che si comporta come segue: quando l'ingresso SET = 1 e l'ingresso RESET = 0 ne segue che l'uscita $Q = 1$, mentre quando l'ingresso SET = 0 e l'ingresso

$RESET = 1$ ne segue che l'uscita $Q = 0$. Infine il caso $SET = 1$ e $RESET = 1$, nello stesso momento, non è ammissibile. Questo tipo di flip-flop è perfetto nel nostro caso perchè quando arriva una generica chiamata P_x questa si trasforma in una richiesta R_x (quindi $Q = R_x$) che rimane in attesa prima di essere servita (cioè rimane in attesa finchè $RESET = 0$). Pertanto avremmo che tutte le chiamate P_0, \dots, P_3, P_{VIP} vengono collegate in 5 flip-flop SR rispettivamente agli ingressi di SET, in modo che, quando arriva una generica chiamata si attivi la relativa richiesta in attesa.

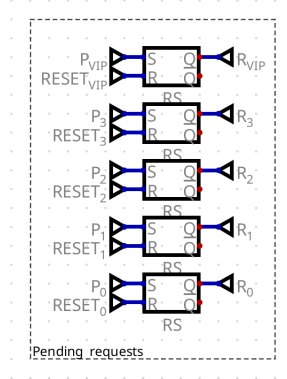


Figure 10. Bufferizzazione delle richieste in attesa

2.6 Gestione dei reset

Ma chi colleghiamo agli ingressi dei RESET? Potrebbe venirci in mente di collegare gli arrivi A_0, \dots, A_3, A_{VIP} , ma questo non va bene perchè gli arrivi si comportano come sensori per la rilevazione del passaggio per un generico piano ma non sono indicatori del fatto che una generica richiesta R_x , e quindi una generica chiamata P_x , sia stata servita o meno. Quindi occorre implementare una circuiteria che ci dica se la richiesta è stata servita correttamente in modo che si possa procedere all'effettivo reset dei flip-flop SR interessati. Ma quand'è che dobbiamo resettare i flip-flop SR? O meglio, quand'è che una richiesta viene servita? Beh, una richiesta viene servita se siamo arrivati ad un piano x e se sono passati 9 secondi nello stato di apertura porte. Pertanto è chiaro che i bit y_2, y_1, y_0 , che rappresentano un generico piano, sono da mettere in AND (con opportuni ingressi negati) tra di loro ed inoltre dobbiamo aggiungere un segnale TIMEOUT che rappresenta il bit di overflow nel momento in cui vengono contati 9 secondi. In particolare, usando un Timer ed impostando un numero di bit pari a 4 e come massimo valore di conteggio $(8)_{10} = (1000)_2$ nel momento in cui si passa a $(9)_{10} = (1001)_2$ ci avanza un bit per generare l'overflow che verrà assegnato al segnale TIMEOUT, ma di questo ne discuteremo meglio più avanti quando verrà creato l'automa a stati finiti per la gestione degli stati dell'ascensore.

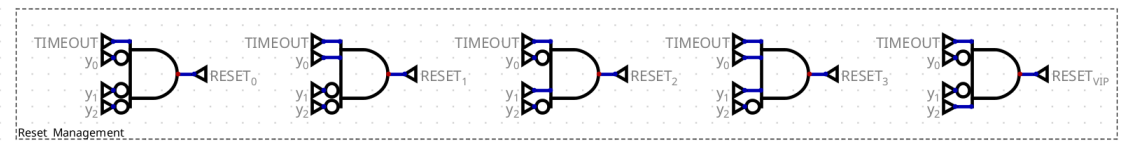


Figure 11. Gestione dei reset

2.7 Rilevatore per un generico piano

In generale sappiamo che i segnali di arrivo A_0, \dots, A_3, A_{VIP} differiscono dalle chiamate P_0, \dots, P_3, P_{VIP} per il fatto che essi sono unici. Infatti non ci possono essere due segnali di arrivo attivi nello stesso momento. Pertanto possiamo costruire un rilevatore che si attiva solo quando si passa per un generico piano e per farlo è sufficiente mettere in OR tutti i segnali di arrivo A_0, \dots, A_3, A_{VIP} proprio per la loro definizione di unicità.

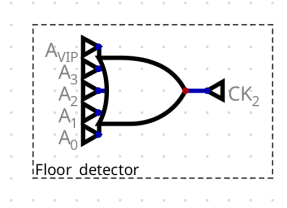


Figure 12. Rilevatore per un generico piano

2.8 Segnale per l'arrivo a destinazione

Un altro segnale utile da implementare è DESTINATION, ossia un segnale che vale 1 quando l'ascensore arriva a destinazione, altrimenti vale 0. Per realizzarlo dovremmo soddisfare una generica richiesta R_x in concomitanza di un generico segnale di arrivo A_x . Questo si traduce in 5 porte AND più l'aggiunta di una porta OR (praticamente si tratta di un Multiplexer, però implementando singolarmente le 6 porte totali si risparmiano input che non sarebbero usati, infatti 5 non è una potenza di 2). Infatti proprio per l'unicità dei segnali A_x solo una delle porte AND sarà attivata a rotazione. Naturalmente l'arrivo a destinazione non implica che una richiesta R_x sia servita.

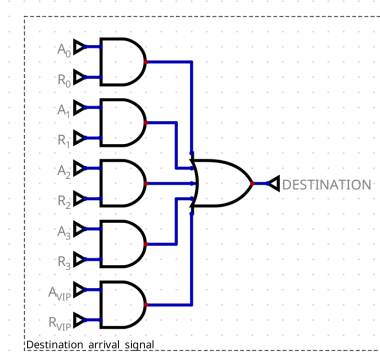


Figure 13. Segnale per l'arrivo a destinazione

2.9 Gestione delle richieste

Arrivati a questo punto occorre trovare un modo per “sapere” se l'ascensore deve andare verso l'alto, verso il basso o rimanere fermo. Pertanto sicuramente sappiamo che ci dovranno essere 3 segnali, ossia UP, STOP e DOWN, che, in base ad una generica richiesta R_x e sapendo a quale piano è arrivata l'ascensore, verranno opportunamente attivati. Ma come facciamo a sapere a che piano è arrivata l'ascensore? Beh, semplicemente usando i segnali di arrivo A_0, \dots, A_3, A_{VIP} (attenzione a non confondere questa logica con la logica del segnale DESTINATION). Ma in che modo? Considerando, per esempio, il segnale UP sappiamo che l'ascensore deve andare verso l'alto solo se $\exists A_x > x \wedge R_x = 1$ (il simbolo \wedge rappresenta la AND logica), cioè solo se mi arriva una richiesta R_x e l'ascensore arriva ad un piano x con un segnale $A_x > x$. Per esempio, se $x = 0$ (piano 0) e mi arriva una richiesta $R_1 = 1$ per sapere se l'ascensore è arrivata al piano 1 avremmo $A_1 = 1$ e quindi $A_1 > x$. Quindi:

$$\begin{aligned} \text{UP} = 1 &\iff \exists A_x > x \wedge R_x = 1 \\ \text{STOP} = 1 &\iff \exists A_x = x \wedge R_x = 1 \\ \text{DOWN} = 1 &\iff \exists A_x < x \wedge R_x = 1 \end{aligned}$$

Pertanto l'unica rete logica iterativa che ci permette di realizzare i confronti $A_x > x$, $A_x = x$, $A_x < x$ è il comparatore che avrà come uscite i segnali UP, STOP, DOWN. Tale comparatore dovrà gestire tutte le possibili richieste R_x in base ai piani di arrivo A_x per un totale di $2^5 \cdot 5 = 160$ casi (infatti abbiamo 4 piani più il piano terra). Inoltre i segnali di arrivo A_x dovranno essere bufferizzati, in funzione della rilevazione di un generico piano, per sapere quale segnale attivare tra UP, STOP,

DOWN in base all'arrivo di una nuova richiesta R_x . Ed ecco qui che ci viene in aiuto il circuito di Fig. 12. In particolare usando un flip-flop D con segnale di *clock* CK_2 possiamo far passare l'ingresso D in uscita Q (cioè $Q = D$) solo quando viene rilevato un nuovo piano. Inoltre per semplificarci la vita possiamo codificare i segnali A_x passando da 5 bit a 3 bit. Quest'ultima scelta è molto utile per la costruzione delle tabelle di verità del comparatore per fare in modo di avere 8 bit in input (cioè le 5 richieste R_x più 3 bit della codifica dei segnali di arrivo A_x) e 3 bit in output (cioè UP, STOP, DOWN).

Iniziamo a realizzare il codificatore dei segnali di arrivo A_x , che avrà la seguente tabella di verità:

A_{VIP}	A_3	A_2	A_1	A_0	a_2	a_1	a_0	HEX
0	0	0	0	0	0	0	0	0x0
0	0	0	0	1	0	0	0	0x0
0	0	0	1	0	0	0	1	0x1
0	0	1	0	0	0	1	0	0x2
0	1	0	0	0	0	1	1	0x3
1	0	0	0	0	1	0	0	0x4

Table 7. Tabella di verità del codificatore per i segnali di arrivo

per implementare il codificatore verrà impiegata una ROM (Read Only Memory) pertanto occorre settare le seguenti impostazioni:

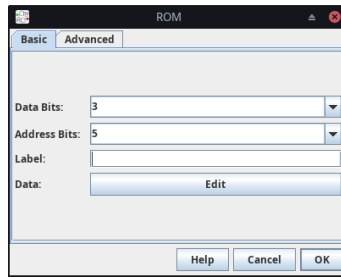


Figure 14. Impostazioni ROM per il codificatore

inserendo nel campo **Data** la colonna HEX presente in Tabella 7. Successivamente tutti i bit di uscita saranno memorizzati in un flip-flop D, con segnale di *clock* CK_2 , con le seguenti impostazioni:

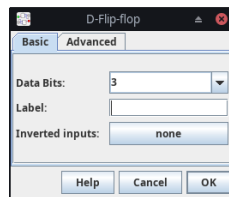


Figure 15. Impostazioni flip-flop D

e l'uscita Q , avente 3 bit, verrà usata come input del comparatore insieme alle richieste R_x . Al termine di questa parte il circuito è il seguente:

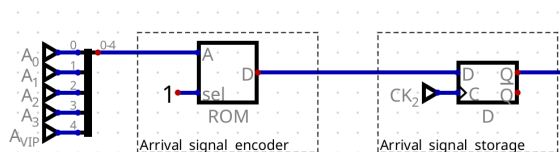


Figure 16. Gestione delle richieste incompleta

Ora riportiamo le relazioni che legano i segnali di uscita UP, STOP, DOWN, con i relativi ingressi in 5 tabelle di verità ciascuna composta da 32 bit. La prima tabella di verità rappresenta le 32 possibili richieste R_x nel caso in cui $a_2=0, a_1=0, a_0=0$ (quindi siamo nel caso in cui non si attiva nessun segnale di arrivo, oppure si attiva il segnale di arrivo A_0):

a_2	a_1	a_0	R_0	R_1	R_2	R_3	R_{VIP}	UP	STOP	DOWN	HEX
0	0	0	0	0	0	0	0	0	0	0	0x0
0	0	0	0	0	0	0	1	1	0	0	0x4
0	0	0	0	0	0	1	0	1	0	0	0x4
0	0	0	0	0	0	1	1	1	0	0	0x4
0	0	0	0	0	1	0	0	1	0	0	0x4
0	0	0	0	0	1	0	1	1	0	0	0x4
0	0	0	0	0	1	1	0	1	0	0	0x4
0	0	0	0	0	1	1	1	1	0	0	0x4
0	0	0	0	1	0	0	0	1	0	0	0x4
0	0	0	0	1	0	0	1	1	0	0	0x4
0	0	0	0	1	0	1	0	1	0	0	0x4
0	0	0	0	1	0	1	1	1	0	0	0x4
0	0	0	0	1	1	0	0	1	0	0	0x4
0	0	0	0	1	1	0	1	1	0	0	0x4
0	0	0	0	1	1	1	0	1	0	0	0x4
0	0	0	0	1	1	1	1	1	0	0	0x4
0	0	0	1	0	0	0	0	0	1	0	0x2
0	0	0	1	0	0	0	1	1	1	0	0x6
0	0	0	1	0	0	1	0	1	1	0	0x6
0	0	0	1	0	0	1	1	1	1	0	0x6
0	0	0	1	0	1	0	0	1	1	0	0x6
0	0	0	1	0	1	0	1	1	1	0	0x6
0	0	0	1	0	1	1	0	1	1	0	0x6
0	0	0	1	0	1	1	1	1	1	0	0x6
0	0	0	1	1	0	0	0	1	1	0	0x6
0	0	0	1	1	0	0	1	1	1	0	0x6
0	0	0	1	1	0	1	0	1	1	0	0x6
0	0	0	1	1	0	1	1	1	1	0	0x6
0	0	0	1	1	1	0	0	1	1	0	0x6
0	0	0	1	1	1	0	1	1	1	0	0x6
0	0	0	1	1	1	1	0	1	1	0	0x6
0	0	0	1	1	1	1	1	1	1	0	0x6

Table 8. Tabella di verità del comparatore per $a_2=0, a_1=0, a_0=0$

La seconda tabella di verità rappresenta le 32 possibili richieste R_x nel caso in cui $a_2=0, a_1=0, a_0=1$ (quindi siamo nel caso in cui si attiva il segnale di arrivo A_1):

a_2	a_1	a_0	R_0	R_1	R_2	R_3	R_{VIP}	UP	STOP	DOWN	HEX
0	0	1	0	0	0	0	0	0	0	0	0x0
0	0	1	0	0	0	0	1	1	0	0	0x4
0	0	1	0	0	0	1	0	1	0	0	0x4
0	0	1	0	0	0	1	1	1	0	0	0x4
0	0	1	0	0	1	0	0	1	0	0	0x4
0	0	1	0	0	1	0	1	1	0	0	0x4
0	0	1	0	0	1	1	0	1	0	0	0x4
0	0	1	0	0	1	1	1	1	0	0	0x4
0	0	1	0	1	0	0	0	0	1	0	0x2
0	0	1	0	1	0	0	1	1	1	0	0x6
0	0	1	0	1	0	1	0	1	1	0	0x6
0	0	1	0	1	0	1	1	1	1	0	0x6
0	0	1	0	1	1	0	0	1	1	0	0x6
0	0	1	0	1	1	0	1	1	1	0	0x6
0	0	1	0	1	1	1	0	1	1	0	0x6
0	0	1	0	1	1	1	1	1	1	0	0x6
0	0	1	1	0	0	0	0	0	0	1	0x1
0	0	1	1	0	0	0	1	1	0	1	0x5
0	0	1	1	0	0	1	0	1	0	1	0x5
0	0	1	1	0	0	1	1	1	0	1	0x5
0	0	1	1	0	1	0	0	1	0	1	0x5
0	0	1	1	0	1	0	1	1	0	1	0x5
0	0	1	1	0	1	1	0	1	0	1	0x5
0	0	1	1	0	1	1	1	1	0	1	0x5
0	0	1	1	1	0	0	0	0	1	1	0x3
0	0	1	1	1	0	0	1	1	1	1	0x7
0	0	1	1	1	0	1	0	1	1	1	0x7
0	0	1	1	1	0	1	1	1	1	1	0x7
0	0	1	1	1	1	0	0	1	1	1	0x7
0	0	1	1	1	1	0	1	1	1	1	0x7
0	0	1	1	1	1	1	0	1	1	1	0x7
0	0	1	1	1	1	1	1	1	1	1	0x7

Table 9. Tabella di verità del comparatore per $a_2=0, a_1=0, a_0=1$

La terza tabella di verità rappresenta le 32 possibili richieste R_x nel caso in cui $a_2=0, a_1=1, a_0=0$ (quindi siamo nel caso in cui si attiva il segnale di arrivo A_2):

a_2	a_1	a_0	R_0	R_1	R_2	R_3	R_{VIP}	UP	STOP	DOWN	HEX
0	1	0	0	0	0	0	0	0	0	0	0x0
0	1	0	0	0	0	0	1	1	0	0	0x4
0	1	0	0	0	0	1	0	1	0	0	0x4
0	1	0	0	0	0	1	1	1	0	0	0x4
0	1	0	0	0	1	0	0	0	1	0	0x2
0	1	0	0	0	1	0	1	1	1	0	0x6
0	1	0	0	0	1	1	0	1	1	0	0x6
0	1	0	0	0	1	1	1	1	1	0	0x6
0	1	0	0	1	0	0	0	0	0	1	0x1
0	1	0	0	1	0	0	1	1	0	1	0x5
0	1	0	0	1	0	1	0	1	0	1	0x5
0	1	0	0	1	0	1	1	1	0	1	0x5
0	1	0	0	1	1	0	0	0	1	1	0x3
0	1	0	0	1	1	0	1	1	1	1	0x7
0	1	0	0	1	1	1	0	1	1	1	0x7
0	1	0	0	1	1	1	1	1	1	1	0x7
0	1	0	1	0	0	0	0	0	0	1	0x1
0	1	0	1	0	0	0	1	1	0	1	0x5
0	1	0	1	0	0	1	0	1	0	1	0x5
0	1	0	1	0	0	1	1	1	0	1	0x5
0	1	0	1	0	1	0	0	0	1	1	0x3
0	1	0	1	0	1	0	1	1	1	1	0x7
0	1	0	1	0	1	1	0	1	1	1	0x7
0	1	0	1	0	1	1	1	1	1	1	0x7
0	1	0	1	1	0	0	0	0	0	1	0x1
0	1	0	1	1	0	0	1	1	0	1	0x5
0	1	0	1	1	0	1	0	1	0	1	0x5
0	1	0	1	1	0	1	1	1	0	1	0x5
0	1	0	1	1	0	1	1	1	0	1	0x5
0	1	0	1	1	1	0	0	0	1	1	0x3
0	1	0	1	1	1	0	1	1	1	1	0x7
0	1	0	1	1	1	1	0	1	1	1	0x7
0	1	0	1	1	1	1	1	1	1	1	0x7
0	1	0	1	1	1	1	1	1	1	1	0x7

Table 10. Tabella di verità del comparatore per $a_2=0, a_1=1, a_0=0$

La quarta tabella di verità rappresenta le 32 possibili richieste R_x nel caso in cui $a_2=0, a_1=1, a_0=1$ (quindi siamo nel caso in cui si attiva il segnale di arrivo A_3):

a_2	a_1	a_0	R_0	R_1	R_2	R_3	R_{VIP}	UP	STOP	DOWN	HEX
0	1	1	0	0	0	0	0	0	0	0	0x0
0	1	1	0	0	0	0	1	1	0	0	0x4
0	1	1	0	0	0	1	0	0	1	0	0x2
0	1	1	0	0	0	1	1	1	1	0	0x6
0	1	1	0	0	1	0	0	0	0	1	0x1
0	1	1	0	0	1	0	1	1	0	1	0x5
0	1	1	0	0	1	1	0	0	1	1	0x3
0	1	1	0	0	1	1	1	1	1	1	0x7
0	1	1	0	1	0	0	0	0	0	1	0x1
0	1	1	0	1	0	0	1	1	0	1	0x5
0	1	1	0	1	0	1	0	0	1	1	0x3
0	1	1	0	1	0	1	1	1	1	1	0x7
0	1	1	0	1	1	0	0	0	0	1	0x1
0	1	1	0	1	1	0	1	1	0	1	0x5
0	1	1	0	1	1	1	0	0	1	1	0x3
0	1	1	0	1	1	1	1	1	1	1	0x7
0	1	1	1	0	0	0	0	0	0	1	0x1
0	1	1	1	0	0	0	1	1	0	1	0x5
0	1	1	1	0	0	1	0	0	1	1	0x3
0	1	1	1	0	0	1	1	1	1	1	0x7
0	1	1	1	0	1	0	0	0	0	1	0x1
0	1	1	1	0	1	0	1	1	0	1	0x5
0	1	1	1	0	1	1	0	0	1	1	0x3
0	1	1	1	0	1	1	1	1	1	1	0x7
0	1	1	1	1	0	0	0	0	0	1	0x1
0	1	1	1	1	0	0	1	1	0	1	0x5
0	1	1	1	1	0	1	0	0	1	1	0x3
0	1	1	1	1	0	1	1	1	1	1	0x7
0	1	1	1	1	1	0	0	0	0	1	0x1
0	1	1	1	1	1	0	1	1	0	1	0x5
0	1	1	1	1	1	1	0	0	1	1	0x3
0	1	1	1	1	1	1	1	1	1	1	0x7
0	1	1	1	1	1	1	1	1	1	1	0x7
0	1	1	1	1	1	1	1	1	1	1	0x7

Table 11. Tabella di verità del comparatore per $a_2=0, a_1=1, a_0=1$

La quinta tabella di verità rappresenta le 32 possibili richieste R_x nel caso in cui $a_2=1, a_1=0, a_0=0$ (quindi siamo nel caso in cui si attiva il segnale di arrivo A_{VIP}):

a_2	a_1	a_0	R_0	R_1	R_2	R_3	R_{VIP}	UP	STOP	DOWN	HEX
1	0	0	0	0	0	0	0	0	0	0	0x0
1	0	0	0	0	0	0	1	0	1	0	0x2
1	0	0	0	0	0	1	0	0	0	1	0x1
1	0	0	0	0	0	1	1	0	1	1	0x3
1	0	0	0	0	1	0	0	0	0	1	0x1
1	0	0	0	0	1	0	1	0	1	1	0x3
1	0	0	0	0	1	1	0	0	0	1	0x1
1	0	0	0	0	1	1	1	0	1	1	0x3
1	0	0	0	1	0	0	0	0	0	1	0x1
1	0	0	0	1	0	0	1	0	1	1	0x3
1	0	0	0	1	0	1	0	0	0	1	0x1
1	0	0	0	1	0	1	1	0	1	1	0x3
1	0	0	0	1	1	0	0	0	0	1	0x1
1	0	0	0	1	1	0	1	0	1	1	0x3
1	0	0	0	1	1	1	0	0	0	1	0x1
1	0	0	0	1	1	1	1	0	1	1	0x3
1	0	0	1	0	0	0	0	0	0	1	0x1
1	0	0	1	0	0	0	1	0	1	1	0x3
1	0	0	1	0	0	1	0	0	0	1	0x1
1	0	0	1	0	0	1	1	0	1	1	0x3
1	0	0	1	0	1	0	0	0	0	1	0x1
1	0	0	1	0	1	0	1	0	1	1	0x3
1	0	0	1	0	1	1	0	0	0	1	0x1
1	0	0	1	0	1	1	1	0	1	1	0x3
1	0	0	1	1	0	0	0	0	0	1	0x1
1	0	0	1	1	0	0	1	0	1	1	0x3
1	0	0	1	1	0	1	0	0	0	1	0x1
1	0	0	1	1	0	1	1	0	1	1	0x3
1	0	0	1	1	1	0	0	0	0	1	0x1
1	0	0	1	1	1	0	1	0	1	1	0x3
1	0	0	1	1	1	1	0	0	0	1	0x1
1	0	0	1	1	1	1	1	0	1	1	0x3

Table 12. Tabella di verità del comparatore per $a_2 = 1, a_1 = 0, a_0 = 0$

per implementare il comparatore verrà impiegata una ROM (Read Only Memory) pertanto occorre settare le seguenti impostazioni:

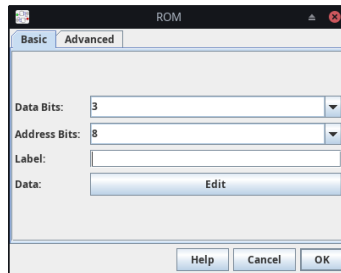


Figure 17. Impostazioni ROM per il comparatore

inserendo nel campo **Data** la colonna HEX delle precedenti 5 tabelle. Pertanto il circuito finale è il seguente:

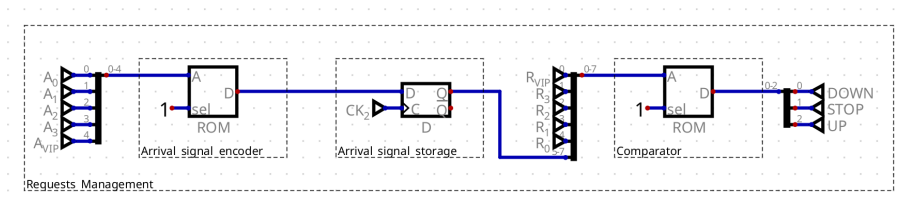


Figure 18. Gestione delle richieste

2.10 Bufferizzazione della direzione

A questo punto la questione è un po' delicata. Dobbiamo trovare un modo per memorizzare (e già questo ci dà un'indicazione importante) da quale direzione l'ascensore arriva ad un generico piano x . Cioè se è arrivata dai piani alti oppure dai piani bassi. O meglio, se una volta arrivata ad un piano

x essa era ad un piano $x + i$ oppure $x - i$, con $i \in [0, 4]$ (dove l'estremo superiore 4 sarebbe il *piano VIP*). Quindi ci serve un segnale che deve essere attivato e memorizzato quando la direzione è verso il basso (il caso verso l'alto si manifesta per dualità perchè se l'ascensore non ha la direzione verso il basso deve per forza averla verso l'alto), per indicare al piano di arrivo che ci siamo arrivati andando verso il basso. Tale segnale, che indicheremo con WAS_{DOWN} , può essere memorizzato in un flip-flop SR, ma da chi viene settato e/o resettato? Avremo bisogno di altri due segnali dedicati. Il segnale che si occuperà di settare $WAS_{DOWN} = 1$ dovrà essere DIR_{DOWN} che dipenderà dall'automa a stati finiti degli stati dell'ascensore, quindi ne parleremo più avanti. Mentre il segnale che si occuperà di settare $WAS_{DOWN} = 0$ (cioè di resettarlo) è proprio il segnale $DOWN$ che abbiamo creato in Fig. 18, il quale è sempre settato a 0 tranne nel caso in cui avviene una richiesta R_x verso il basso. Questo vuol dire che il flip-flop SR dovrà avere l'ingresso di RESET negativo.

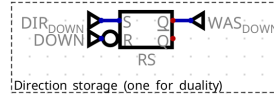


Figure 19. Bufferizzazione della direzione

2.11 Automa a stati finiti per la gestione degli stati

Una volta creati tutti i segnali di cui abbiamo discusso finora è chiaro che essi vadano inseriti come input per l'automa a stati finiti dedicato alla gestione degli stati dell'ascensore. Quindi, ricapitolando, avremmo in input i segnali CK_2 , $DESTINATION$, UP , $STOP$, $DOWN$, WAS_{DOWN} più il segnale $TIMEOUT$, che non abbiamo trattato ancora in modo approfondito, e infine il segnale di richiesta R_{VIP} per la gestione dedicata al *piano VIP*. Naturalmente a questi si aggiungono le transizioni z_2, z_1, z_0 che rientrano in input per definizione. Tuttavia, anche se l'ascensore non ha un output esplicito, possiamo comunque fare in modo che l'automa a stati finiti generi degli output per noi. In particolare nel circuito di Fig. 19 parlavamo del segnale DIR_{DOWN} . Tale segnale sarà un output dell'automa a stati finiti insieme al segnale DIR_{UP} , che ci tornerà utile per la creazione dell'automa a stati finiti per la gestione dei piani dell'ascensore. Dunque, come si intuisce, è chiaro che gli stati dell'automa a stati finiti per la gestione degli stati dell'ascensore ha come stati proprio la codifica che troviamo in Tabella 1. Pertanto possiamo costruire una relazione tra questi stati utilizzando il seguente automa a stati finiti:

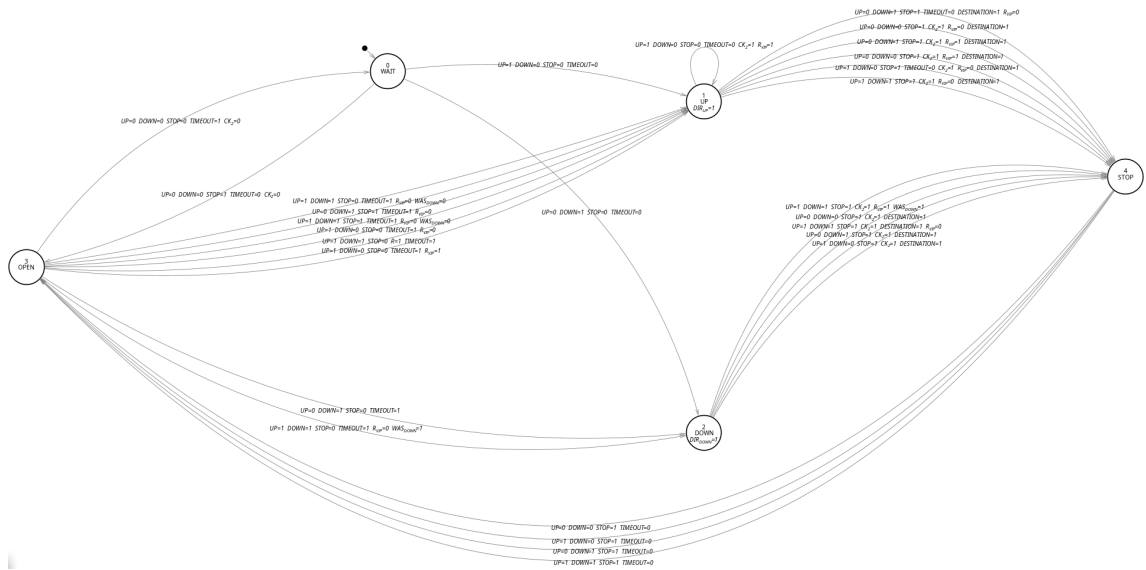


Figure 20. Automa a stati finiti per la gestione degli stati

consultabile direttamente dal file `FSM-states.fsm` attraverso il simulatore Digital. Pertanto, andando nell'apposito menù `Create > State Transition Table` e successivamente, sempre dall'apposito menù, su `Create > Circuit Variants > Circuit with LUTs` si ottiene il seguente circuito sequenziale:

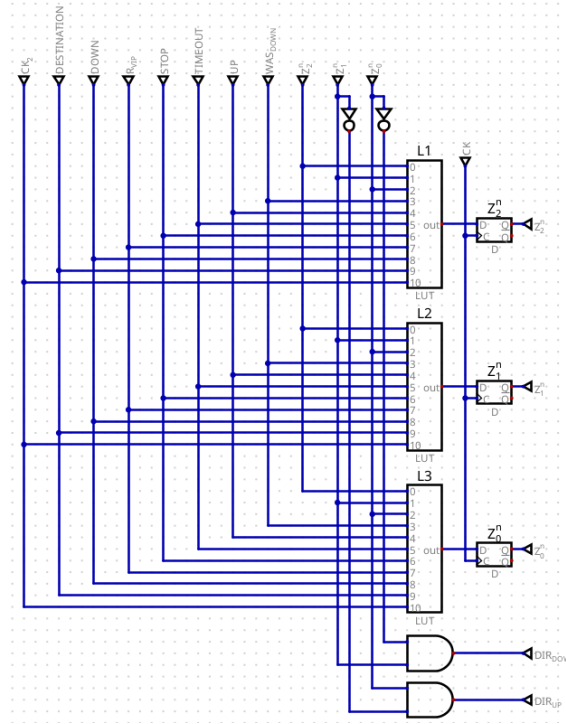


Figure 21. Circuito sequenziale (incompleto) dell'automa a stati finiti per la gestione degli stati

Il circuito mostrato in Fig. 21 è stato leggermente modificato rispetto a quello generato automaticamente da Digital. In particolare sono stati tolti i bottoni di input e di output, perchè utilizziamo il componente *tunnel* per portare un determinato dato da una parte all'altra del circuito.

2.11.1 Display a 7 segmenti per il timer

Anche in questo caso andremo ad implementare un decodificatore per mostrare il conteggio del Timer su un display a 7 segmenti a catodo comune. La progettazione è molto simile a quella già precedentemente realizzata per la visualizzazione del piano corrente però, in questo caso, ancora una volta, cambia il numero di bit da decodificare. Pertanto possiamo costruire la seguente tabella di verità (si noti che partendo da 0 e arrivando ad 8 sono passati effettivamente 9 secondi):

Secondi	t_3	t_2	t_1	t_0	a_t	b_t	c_t	d_t	e_t	f_t	g_t	HEX
0	0	0	0	0	1	1	1	1	1	1	0	0x7E
1	0	0	0	1	0	1	1	0	0	0	0	0x30
2	0	0	1	0	1	1	0	1	1	0	1	0x6D
3	0	0	1	1	1	1	1	1	0	0	1	0x79
4	0	1	0	0	0	1	1	0	0	1	1	0x33
5	0	1	0	1	1	0	1	1	0	1	1	0x5B
6	0	1	1	0	1	0	1	1	1	1	1	0x5F
7	0	1	1	1	1	1	1	0	0	0	0	0x70
8	1	0	0	0	1	1	1	1	1	1	1	0x7F

Table 13. Tabella di verità del decodificatore per il display a 7 segmenti

A questo punto abbiamo tutto il necessario per la realizzazione del decodificatore, quindi poichè avremmo 4 bit di input e 7 bit di output dovremmo settare la ROM con le seguenti impostazioni:

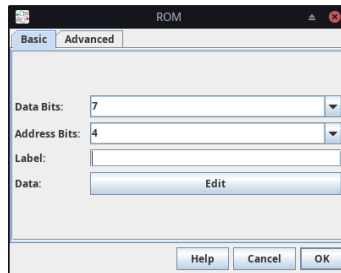


Figure 22. Impostazioni ROM per il display a 7 segmenti

inserendo nel campo **Data** la colonna HEX presente in Tabella 13. Pertanto il circuito finale è il seguente:

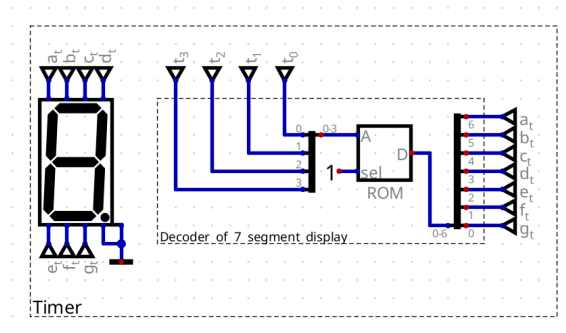


Figure 23. Decodificatore per il display a 7 segmenti

Infine l'input "dp" del display a 7 segmenti gestisce il punto, ma dato che non ci serve è stato collegato direttamente a GND.

2.11.2 Gestione del timer

Ma quand'è che si deve attivare il Timer? Beh, solo nello stato di apertura porte (cioè OPEN) che corrisponde a $z_2 = 0, z_1 = 1, z_0 = 1$. Quindi possiamo creare il seguente circuito che si collega direttamente al precedente:

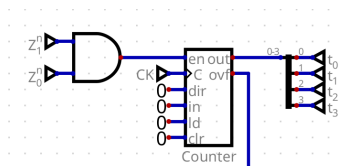


Figure 24. Gestione del timer nell'automa a stati finiti per la gestione degli stati dell'ascensore

Pertanto il circuito sequenziale completo per la gestione degli stati dell'ascensore è il seguente:

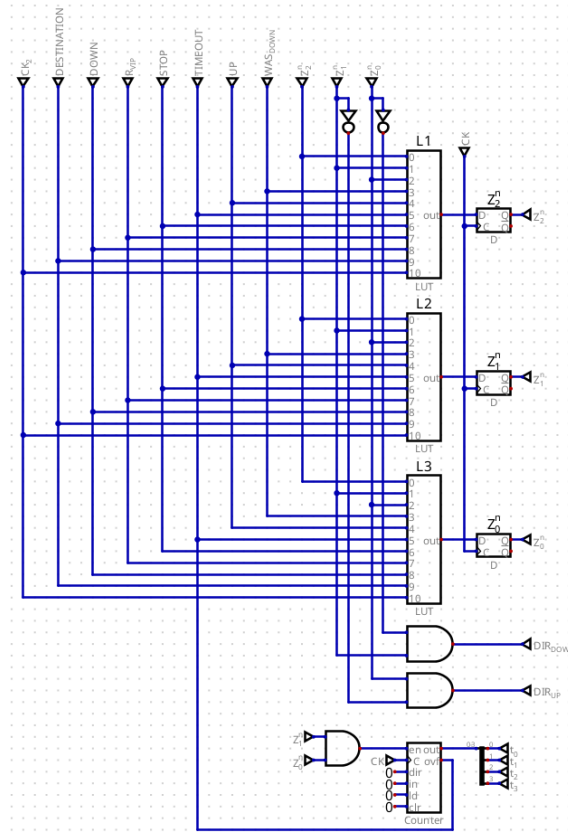


Figure 25. Circuito sequenziale completo dell'automa a stati finiti per la gestione degli stati

collegando opportunamente l'uscita di overflow del Timer al segnale TIMEOUT. Quindi in caso di overflow, cioè quando il Timer arriva a contare $(9)_{10}$, avremo $\text{TIMEOUT} = 1$.

2.12 Automa a stati finiti per la gestione dei piani

Siamo quasi giunti al termine della nostra progettazione. Ci rimane solo la realizzazione dell'automa a stati finiti per la gestione dei piani dell'ascensore. Esso avrà in input il segnale CK_2 insieme agli output generati dall'automa a stati finiti per la gestione degli stati dell'ascensore, ossia DIR_{DOWN} e DIR_{UP} . Naturalmente a questi si aggiungono le transizioni y_2, y_1, y_0 che rientrano in input per definizione. Dunque, come si intuisce, è chiaro che gli stati dell'automa a stati finiti per la gestione dei piani dell'ascensore ha come stati proprio la codifica che troviamo in Tabella 2. Pertanto possiamo costruire una relazione tra questi stati utilizzando il seguente automa a stati finiti:

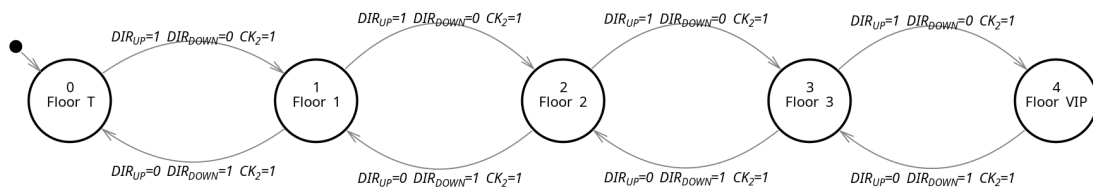


Figure 26. Automa a stati finiti per la gestione dei piani

consultabile direttamente dal file `FSM-floor.fsm` attraverso il simulatore Digital. Pertanto, andando nell'apposito menù **Create > State Transition Table** (consiglio di modificare le variabili da z_* ad y_* con un click destro del mouse sul nome) e successivamente, sempre dall'apposito menù, su **Create > Circuit Variants > Circuit with LUTs** si ottiene il seguente circuito sequenziale:

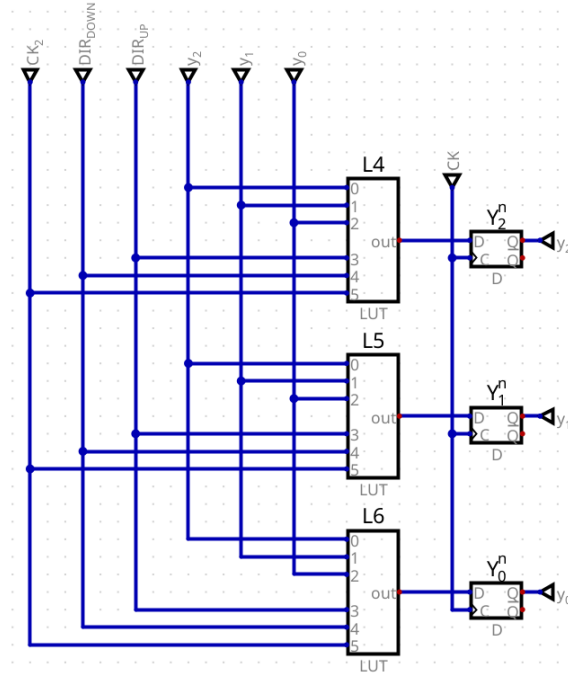


Figure 27. Circuito sequenziale completo dell'automata a stati finiti per la gestione dei piani

Anche in questo caso il circuito mostrato in Fig. 27 è stato leggermente modificato rispetto a quello generato automaticamente da Digital. In particolare sono stati tolti i bottoni di input e di output, perchè utilizziamo il componente *tunnel* per portare un determinato dato da una parte all'altra del circuito.

2.13 Conclusione

Innanzitutto concludo mostrando il circuito finale completo che realizza il sistema di controllo per l'ascensore:

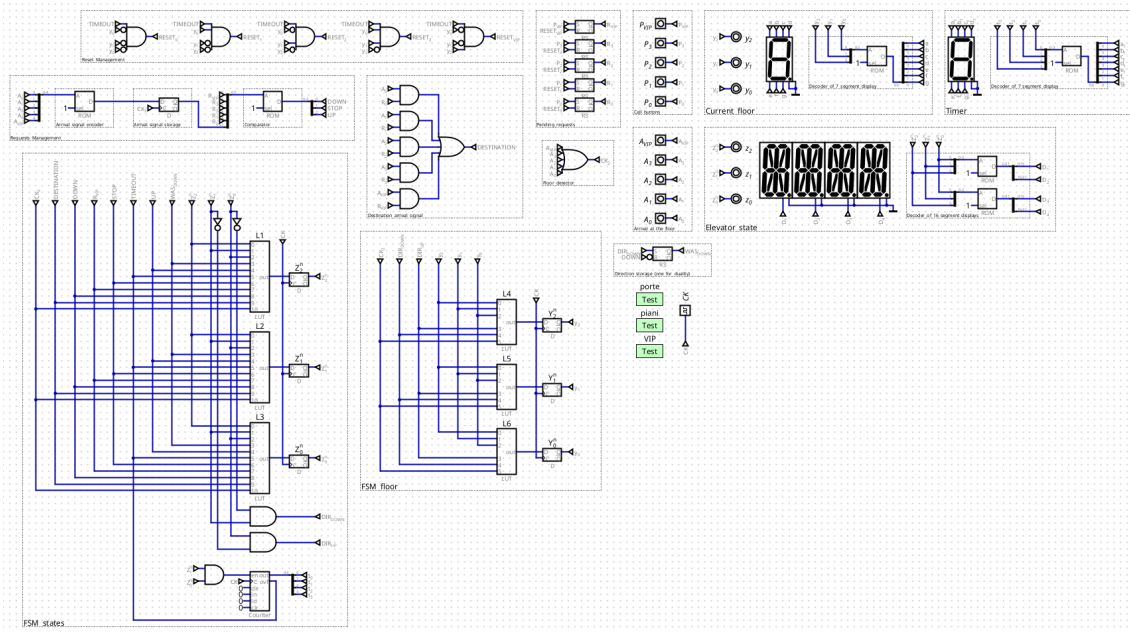


Figure 28. Circuito completo per la gestione di un sistema di controllo di un'ascensore

consultabile direttamente dal file `circuit.dig` attraverso il simulatore Digital. Infine voglio

precisare che una progettazione del genere al giorno d'oggi è molto “old school” nel senso che probabilmente, sia per una questione di costi che per una questione di tempo, è più facile impiegare delle schede embedded programmabili. Tuttavia ammetto che si tratta comunque di un esercizio per pensare fuori dagli schemi canonici, esattamente ciò che un ingegnere deve saper fare. Sarebbe interessante portare avanti la progettazione cercando di ottimizzare al minimo il numero di porte per poi prototipare l'intero progetto su PCB utilizzando software come KiCAD, aggiungendo dei veri sensori, al posto dei segnali $A_0, \dots A_3, A_{VIP}$, dei motori, e in generale tutto ciò che serve per la realizzazione di un'ascensore.