Riconoscitore di stringhe

by Antonio Bernardini

Table of contents

1 Homework

1.1 Introduzione

Per questo homework, occorre progettare una *rete sequenziale* in grado di accettare sequenze di caratteri che rispettino il formato $ba(dc)*bac$, dove la notazione $*$ indica che possono esserci n ripetizioni della sottosequenza dc con $n \in [0, \infty)$. La sequenza di caratteri in input è infinita.

L'alfabeto di input è costituito dai caratteri $I = \{a, b, c, d\}$ rappresentati con le seguenti codifiche:

I	x_1	x_0
a	0	0
b	0	1
c	1	1
d	1	0

Table 1. Tabella con le codifiche dei caratteri di input

L'alfabeto di output è costituito dai caratteri $O = \{\text{no}, \text{sì}\}$ rappresentati con le seguenti codifiche:

O	z
no	1
sì	0

Table 2. Tabella con le codifiche dei caratteri di output

La rete dovrà essere costruita come *macchina di Mealy*. La *rete sequenziale* dovrà essere in grado di accettare o rifiutare correttamente le stringhe ammissibili secondo la definizione dell'espressione ed inoltre deve essere in grado di accettare correttamente anche sequenze *sovrapposte*.

2 Soluzione

2.1 Introduzione

Abbiamo a disposizione tutte le indicazioni per procedere alla progettazione della *rete sequenziale* però come al solito cercherò di implementare una soluzione piuttosto compatta con l'ausio di moduli ROM (Read Only Memory).

2.2 Design della rete sequenziale

Sappiamo che l'alfabeto di input è $I = \{a, b, c, d\} = \{00, 01, 11, 10\}$, mentre l'alfabeto di output è $O = \{\text{no}, \text{sì}\} = \{1, 0\}$. Pertanto costruisco una relazione tra gli stati utilizzando la *macchina di Mealy*:

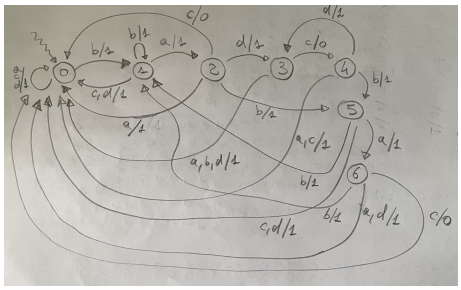


Figure 1. Relazione tra gli stati utilizzando la *macchina di Mealy*

Quindi, poichè ci sono 7 stati (in realtà lo stato 5 e lo stato 6 sono inutili perchè sono una ripetizione degli stati 1 e 2, tuttavia li ho lasciati perchè, avendo scelto di usare solo moduli ROM, avrei avuto sempre 5 bit d'ingresso e 1 bit d'uscita per ogni singolo modulo, come si vede in Fig. 2, pertanto eliminando tali stati non c'è una vera e propria ottimizzazione), abbiamo bisogno di 3 bit per la rappresentazione corretta degli stati (che indicheremo con y_2, y_1, y_0) e delle transizioni (che indicheremo con y_2', y_1', y_0'), che avviene tramite la seguente tabella:

x_1	x_0	y_2	y_1	y_0	y_2'	y_1'	y_0'	z
0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	0	1
0	1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	1
0	1	0	1	0	1	0	1	1
1	1	0	1	0	0	0	0	0
1	0	0	1	0	0	1	1	1
0	0	0	1	1	0	0	0	1
0	1	0	1	1	0	0	0	1
1	1	0	1	1	1	0	0	0
1	0	0	1	1	0	0	0	1
0	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	1	1
1	1	1	0	0	0	0	0	1
1	0	1	0	0	0	1	1	1
0	0	1	0	1	1	1	0	1
0	1	1	0	1	0	0	1	1
1	1	1	0	1	0	0	0	1
1	0	1	0	1	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	1	1	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	1	1	–	–	–	–
0	1	1	1	1	–	–	–	–
1	1	1	1	1	–	–	–	–
1	0	1	1	1	–	–	–	–

Table 3. Tabella delle codifiche degli input, degli stati, delle transizioni e dell'output

Ora normalemente si dovrebbe procedere con la minimizzazione delle transizioni y_2', y_1', y_0' e dell'output z utilizzando le mappe di Karnaugh, tuttavia come accennato pocanzi implementeremo

dei moduli ROM. Pertanto in totale avremo bisogno di 4 moduli ROM (3 per le transizioni ed 1 per l'output) configurando ogni modulo con le seguenti impostazioni:

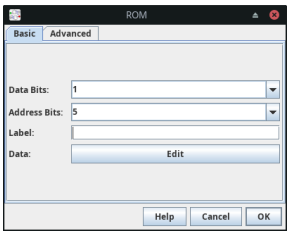
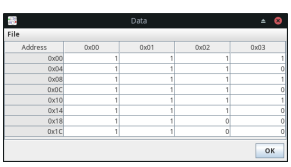


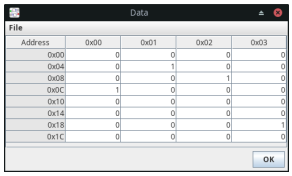
Figure 2. Impostazioni di ogni singolo modulo ROM

dove è possibile aggiungere un *Label* personalizzato per comprendere a cosa serve il singolo modulo. Dunque possiamo configurare gli indirizzi di memoria del modulo ROM utilizzando la Tabella 3. In particolare tutti i bit colorati in blu, presi per righe, devono essere convertiti in esadecimale (4 bit alla volta partendo dal bit meno significativo, ovvero y_0, e procedendo fino al bit più significativo, ovvero x_1). Mentre i bit colorati in verde devono essere presi per colonne perchè ogni ROM calcolerà una singola transizione y_i' o l'output z.

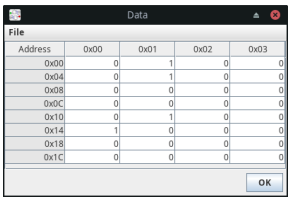
Pertanto per utilizzare le ROM sul software Digital occorre compilare ogni indirizzo di memoria di ogni ROM come segue:



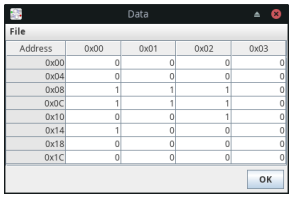
ROM per l'output z



ROM per la transizione y_2'



ROM per la transizione y_1'



ROM per la transizione y_0'

Figure 3. Configurazione dei moduli ROM della *rete sequenziale*

Chiaramente trattandosi di una *rete sequenziale* avremo bisogno di un modulo *clock*, in comune a 4 flip-flop D, ai quali verranno collegate in input le transizioni e, attraverso il componente *tunnel*, verranno portate le uscite Q_i agli stati y_2, y_1 e y_0 di input. Pertanto il circuito finale è il seguente:

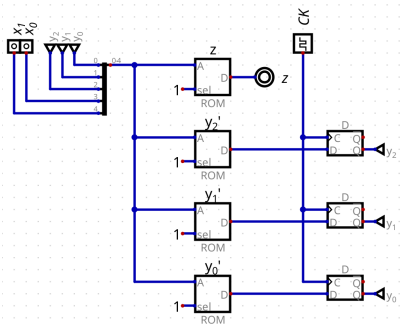


Figure 4. Circuito finale