Laboratorio di Architettura degli Elaboratori I

Corso di laurea triennale in Informatica
Università degli Studi di Milano, A.A. 2019-2020

Nicola Basilico

7 Gennaio 2020

Simulazione d'esame (Turno A)

- L'esame ha una durata di 2 ore.
- È possibile consultare il libro di testo, appunti e la documentazione di Logisim.
- È proibito l'accesso ad Internet con qualsiasi mezzo.
- Verranno corretti solo gli esercizi che non generano errori.
- I sorgenti vanno uploadati su `https://upload.di.unimi.it/`

Esercizio 1

Upload: `esercizio1.circ`

Si sintetizzi un circuito per la funzione logica definita da E, preferibilmente dopo averla semplificata:

$$E = \Big(a \oplus (b \wedge c)\Big) \vee \Big(\overline{a \oplus \big((a \vee b \wedge c) \wedge (\bar{a} \vee b \wedge c)\big)}\Big) \wedge d$$

Nota: date due espressioni booleane E_1 ed E_2, $\overline{E_1}$ indica la negazione di E_1 (NOT); $E_1 \wedge E_2$ indica l'AND tra E_1 ed E_2; $E_1 \vee E_2$ indica l'OR tra E_1 ed E_2; $E_1 \oplus E_2$ indica l'OR esclusivo (XOR) tra E_1 ed E_2.

Soluzione

L'espressione E può essere semplificata nel seguente modo:

$$E = \Big(a \oplus (b \wedge c)\Big) \vee \Big(\overline{a \oplus \big((a \vee b \wedge c) \wedge (\bar{a} \vee b \wedge c)\big)}\Big) \wedge d$$

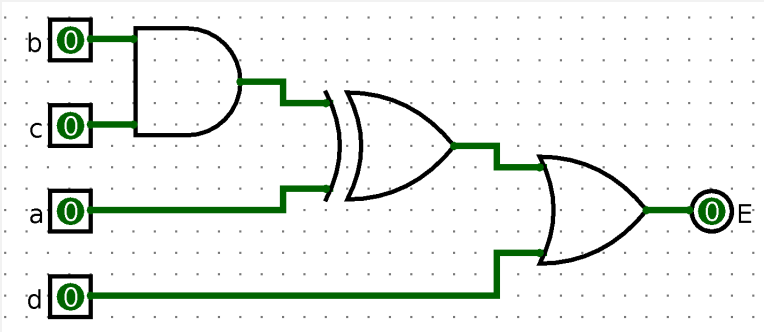
usando prop. Distributiva di AND, Inverso, Idempotenza

$$= \Big(a \oplus (b \wedge c)\Big) \vee \Big(\overline{a \oplus \big(b \wedge c\big)}\Big) \wedge d$$

usando prop. Assorbimento II di OR

$$= \Big(a \oplus (b \wedge c)\Big) \vee d$$

Il circuito che implementa l'espressione semplificata è quindi il seguente:



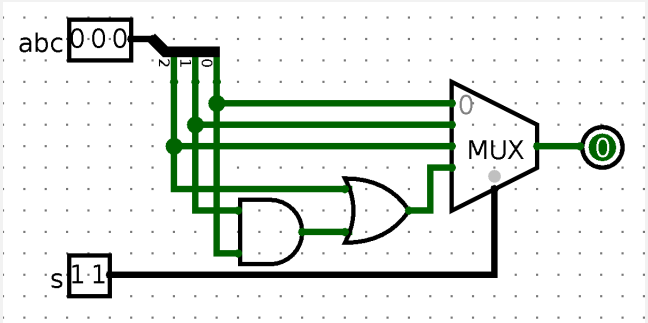
Esercizio 2

Upload: `esercizio2.circ`

Si implementi il circuito di un multiplexer a tre ingressi (a, b e c). Si estenda poi tale circuito in modo che il segnale di selezione non utilizzato per la scelta dell'input venga impiegato per per porre sull'uscita la funzione logica $a \vee c \wedge b$.

Soluzione

Un multiplexer a tre ingressi dovrà avere comunque due bit di selezione. Possiamo assumere che i segnali 00, 01 e 10 vadano a selezionare a, b e c, rispettivamente. Il segnale 11 resta inutilizzato e lo possiamo sfruttare per fare passare come quarta via il risultato della funzione combinatoria richiesta. Il circuito che risulta fa uso del componente *multiplexer* dalla libreria di Logisim:



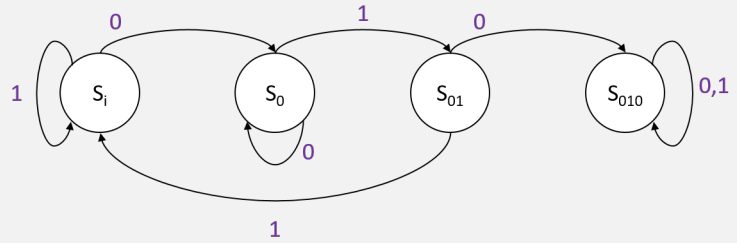
Esercizio 3

Upload: `esercizio3-4.circ`

Si realizzi un circuito M che riceva in ingresso una sequenza temporale di bit $b(t)$, un segnale asincrono di reset r e che emetta in uscita un segnale ad un bit ω. L'uscita, inizialmente posta a 0, passa stabilmente a 1 al primo riconoscimento della sotto-sequenza $(0, 1, 0)$. Una volta effettuato tale riconoscimento, il circuito non effettua più alcuna transizione a meno che non venga inviato il segnale r che riporta il circuito allo stato iniziale in modo asincrono.

Soluzione

Sintetizziamo M come una normale macchina a stati finiti, iniziamo costruendo il grafo delle transizioni. La macchina deve riconoscere la prima occorrenza della sequenza $(0, 1, 0)$, quindi dovremo usare 4 stati:

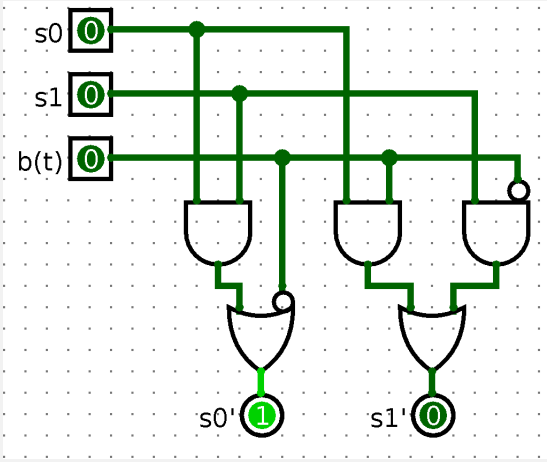


L'uscita ω sarà sempre pari a 0, tranne nello stato di riconoscimento S_{010} dove viene posta a 1. Si noti che, come da specifica, lo stato di riconoscimento è uno stato pozzo.
Assegniamo agli stati $S_i, S_0, S_{01}, S_{010}$ le codifiche $00, 01, 10, 11$, rispettivamente. Chiamiamo s_1, s_0 il secondo e il primo bit di stato, rispettivamente. La tabella delle transizioni è la seguente (con l'apostrofo in apice viene indicato il bit di stato prossimo):

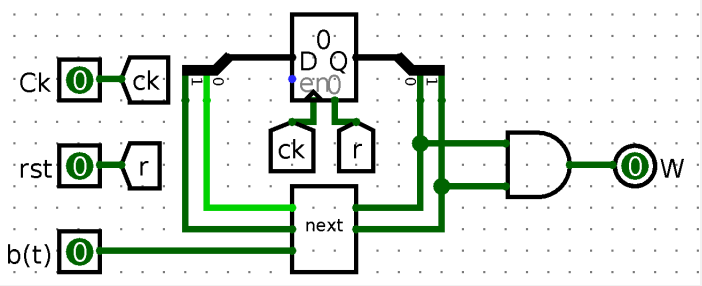
s_1	s_0	$b(t)$	s_1'	s_0'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

Il primo bit di stato s_0' ha tre maxtermini. Sfruttando la seconda forma canonica possiamo ottenere questa espressione: $s_0' = s_1 \wedge s_0 \vee \overline{b(t)}$.

Il secondo bit di stato s_1' ha 4 mintermini. Sfruttando la prima forma canonica possiamo ottenere questa espressione: $s_1' = s_0 \wedge b(t) \vee s_1 \wedge \overline{b(t)}$. La funzione di stato prossimo (che nel circuito chiameremo *next*) si implementa quindi con questo semplice circuito combinatorio:



Il circuito finale combina lo stato prossimo con un registro a due bit che usiamo come memoria dello stato corrente:



Il segnale di reset, essendo asincrono, può essere trattato direttamente nello hardware semplicemente collegandolo all'ingresso asynchronous clear del registro.

Esercizio 4

Upload: `esercizio3-4.circ`

Si ri-utilizzi il circuito prodotto nell'esercizio precedente in modo da realizzare un circuito esteso descritto di seguito.

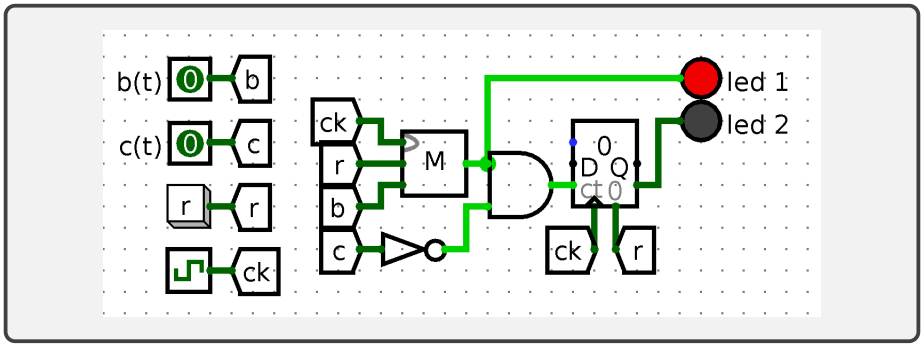
Il circuito riceve in ingresso due sequenze temporali di bit indicate con $b(t)$ e $c(t)$ e un segnale asincrono di reset r. In uscita sono presenti due led chiamati $led1$ e $led2$. Il circuito lavora attraversando due fasi.

- *Fase 1*: entrambi i led sono spenti e lo stream $c(t)$ viene ignorato (il circuito processa solo $b(t)$).
- *Fase 2*: $led1$ è acceso e lo stream $b(t)$ viene ignorato (il circuito processa solo $c(t)$).

Il circuito è inizialmente in fase 1. In questa fase, non appena viene riconosciuta la sotto-sequenza $(0, 1, 0)$ in $b(t)$, $led1$ viene acceso e si passa alla fase 2. Nella fase 2 il circuito conta quanti zeri compaiono nello stream $c(t)$. Non appena sono stati osservati 15 zeri (non necessariamente consecutivi), anche $led2$ viene acceso. Una volta che entrambi i due led sono accesi, l'unico modo per spegnerli è quello di riportare il circuito in fase 1 dando il segnale di reset asincrono.

Soluzione

La fase 1 nient'altro è che la macchina a stati finiti dell'esercizio precedente dove l'uscita ω (che rappresenta lo stato di riconoscimento di M) viene usata per accendere $led1$.
Per poter contare gli zeri basta aggiungere un contatore (componente *counter* dalla libreria di Logisim). Visto che dobbiamo contare al più fino a 15 poniamo a 4 il numero di data bits (il max value del contatore sarà quindi 15 o $0xf$ in esadecimale). Nell'ingresso *count* faremo arrivare il segnale $\omega \wedge \overline{c(t)}$. In questo modo il contatore incrementerà il suo valore solo quando $\omega = 1$ (abbiamo riconosciuto $(0, 1, 0)$ e siamo quindi in fase 2) e $c(t) = 0$. Il risultato è proprio quello di contare gli zeri dal momento in cui si è in fase 2, tutti gli input dati su $c(t)$ prima che $\omega = 1$ vengono ignorati. Visto che l'unico modo di spegnere i led deve essere il reset asincrono colleghiamo il segnale r all'asynchronous clear del contatore e settiamo a *stay on value* la proprietà *action on overflow*. Il circuito risultante è il seguente:


b(t)
b
c(t)
c
r
r
ck
ck
r
b
c
M
0
D Q
ct 0
ck
r
led 1
led 2