Guía de ejercicios # 1 - Lógica Digital

Organización de Computadoras 2017

UNQ

1 Introduccion

Las computadoras actuales resuelven todos los problemas en base a un conjunto pequeño de herramientas fundamentados en la lógica proposicional. Esto tiene origen en el siglo XIX, cuando el matemático George Boole propuso utilizar la lógica proposicional para **describir y resolver los problemas**. Tiempo después se pensó en **automatizar** esas expresiones mediante circuitos digitales que son los que hoy componen las computadoras.

1.1 Operadores elementales

El álgebra de Boole, como cualquier álgebra, usa variables y operaciones. En particular, las variables y operaciones son lógicas, por lo cual las variables pueden tomar el valor 1 (que representa al valor verdadero) y el 0 (que representa al valor falso).

A continuación un repaso de las tablas de verdad de los operadores clásicos.

Conjunción

p	**q**	$p \bullet q$
0	0	0
0	1	0
1	0	0
1	1	1

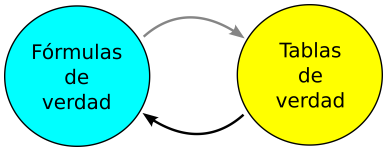
Disyunción

p	**q**	$p + q$
0	0	0
0	1	1
1	0	1
1	1	1

Negacion

p	$\overline{p}$
0	1
1	0

En esta sección pondremos en práctica la traduccion entre dos sistemas formales, las expresiones (o fórmulas) de verdad y las tablas de verdad.



Ejercicios

Escribir la tabla de verdad de las siguientes expresiones.

1.1.a $p \bullet q + \overline{p \bullet q}$

1.1.b $(\overline{p} + q) \bullet (p + \overline{q})$

1.1.c $(p \bullet q \bullet w) + \overline{p \bullet q \bullet w}$

1.1.d $(p \bullet q \bullet w) + p \bullet \overline{q \bullet w} + \overline{p \bullet q \bullet w}$

1.1.e $p \bullet (q \bullet \overline{w} + \overline{q} \bullet w)$

1.1.f $(p + q) \bullet (p + w) \bullet (\overline{p} + \overline{q})$

1.2 Formas normales

A menudo es útil contar con un mecanismo estructurado para deducir la fórmula de verdad a partir de una tabla de verdad de una relativa complejidad. Para eso presentamos la **suma de productos (SoP)** y el **producto de sumas (PoS)**

SoP La expresión SOP (*sum of product*) es una norma de escritura para las fórmulas de verdad. Como lo dice su nombre, es una suma (disyunción) de términos que son productos (conjunciones) entre literales. ¿Y qué es un literal? Bueno, es una variable o su negación.
Para construirla, se debe extraer el término que describe cada caso de la tabla de verdad que verifica la fórmula. Es decir, cada fila donde la salida vale 1. Con cada uno de esos casos se describe con un término con todas las variables, segun aparezcan afirmadas o negadas. Por ejemplo:

p	q	s	
0	0	0	
0	1	1	$\rightarrow \overline{p} \bullet q$
1	0	1	$\rightarrow p \bullet \overline{q}$
1	1	1	$\rightarrow p \bullet q$

Finalmente, los términos se componen con una disyunción: $(\overline{p} \bullet q) + (p \bullet \overline{q}) + (p \bullet q)$

PoS La expresión PoS (*Products of sums*) es una conjunción de disyunciones. A diferencia de la anterior, describe la fórmula a partir de los casos donde no se cumple, algo así como "**f vale cuando no ocurre el caso ... ni el caso...**"

El primer paso es rescatar los casos donde la formula vale **0**, por ejemplo:

p	q	s	
0	0	0	$\rightarrow \overline{p} \bullet \overline{q}$
0	1	1	
1	0	0	$\rightarrow p \bullet \overline{q}$
1	1	1	

Luego, esos casos se niegan y se unen con conjunción, ya que no debe cumplirse ninguno de ellos: $\overline{(\overline{p} \bullet \overline{q})} \bullet \overline{(p \bullet \overline{q})}$

Finalmente, aplicando la propiedad de De Morgan, los terminos negados se convierten en disyunciones: $(p+q) \bullet (\overline{p}+q)$

1.2.a Escribir la fórmula que se obtiene con **SOP** para la siguiente tabla. Luego, puede construir la tabla de verdad en base a la expresión obtenida para verificar el resultado del método SoP.

A	B	C	$F(A,B,C)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

1.2.b Escribir la fórmula que se obtiene con **PoS** para la siguiente tabla. Luego, puede construir la tabla de verdad en base a la expresión obtenida para verificar el resultado del método PoS.

A	B	C	$F(A,B,C)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

2 Circuitos combinatorios

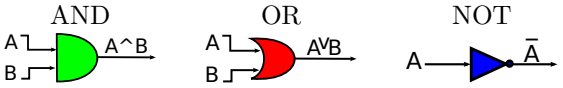
Fue Claude Shannon (matemático e ingeniero norteamericano, 1916-2001) quien aplicó las teorías de Boole a los circuitos compuestos de elementos que solo pueden adoptar dos estados estables, apareciendo entonces los llamados circuitos lógicos. Puede decirse entonces que el álgebra de Boole es el sistema matemático empleado en el diseño de circuitos lógicos.

Un circuito lógico es una composición de compuertas que traduce un conjunto de entradas en una salida, de acuerdo a una función lógica, que se actualiza inmediatamente luego de proveerse las entradas.

2.1 Compuertas elementales

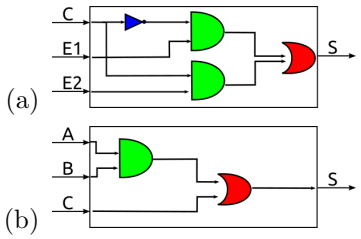
Existe un convenio gráfico para representar dispositivos que lleven a cabo funciones booleanas elementales y que, en función de las combinaciones diseñadas, se obtendrán funciones más complejas. Las siguientes son las compuertas elementales.

AND OR NOT



Ejercicios

2.1.a Construir la fórmula de verdad para los siguientes circuitos:



2.1.b Especifique la tabla de verdad de cada circuito del ejercicio anterior.

2.2 Otras compuertas

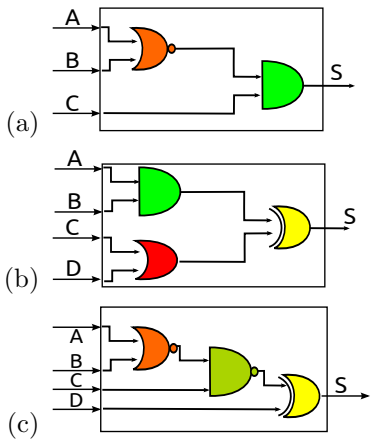
Además, es práctico contar con otras compuertas (que se construyen a partir de las anteriores).

NAND NOR XOR

$a \uparrow b = \overline{a \bullet b}$ $a \downarrow b = \overline{a + b}$ $a \oplus b = (a \bullet \overline{b}) + (\overline{a} \bullet b)$

Ejercicios

2.1.a Construir la fórmula de verdad para los siguientes circuitos:



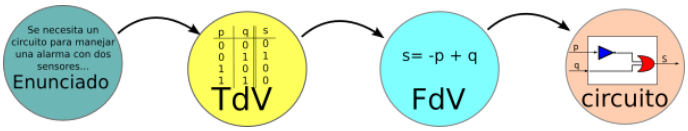
2.1.b Especifique la tabla de verdad de cada circuito del ejercicio anterior.

2.3 Diseño de circuitos

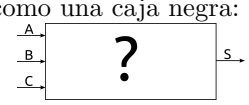
¿Cómo se construyen los circuitos? A partir de su fórmula de verdad ¿Y como se llega a la fórmula?

En general se cuenta con una descripción del problema a través de un enunciado en lenguaje natural (informal), que

debe ser abstraido (formalizado) en una tabla de verdad, para luego ser convertida en una fórmula de verdad y luego en un circuito.



Por ejemplo, se necesita un circuito de 3 entradas y una salida que compute la función "mayoría". Es decir, que si dos o más entradas valen 1, la salida debe valer 1, y un 0 en caso contrario. Para entender lo que se pide, es útil dibujarlo como una caja negra:



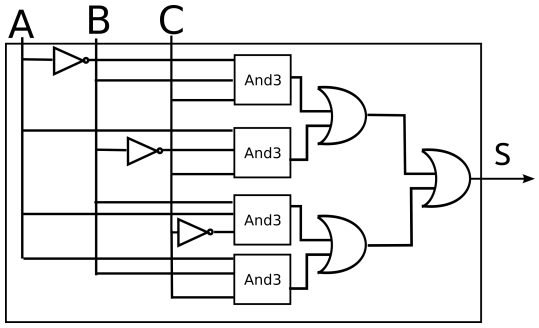
Y luego plantear la tabla de verdad:

A	B	C	Mayoría
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Usando el método SoP, se obtiene la fórmula:

$$(\overline{A} \bullet B \bullet C) + (A \bullet \overline{B} \bullet C) + (A \bullet B \bullet \overline{C}) + (A \bullet B \bullet C)$$

Finalmente, dibujar el circuito:



Ejercicios

2.3.a Diseñar un circuito **And3** de 3 entradas y una salida que compute la conjunción entre sus entradas.

2.3.b Diseñe un circuito de 3 entradas y una salida, cuya salida sea tal que la cantidad de unos entre las entradas y las salidas sea par (por ejemplo, para la entrada 101 la salida es 0 y para la entrada 111 la salida es 1). Plantee la solución como una tabla de verdad y derive el circuito de esta última. Pruebe el circuito obtenido con al menos 4 casos, es decir, simule que se ingresan valores (0 o 1) en las entradas y verifique si la salida obtenida es la esperada.

2.3.c Haga un circuito con 4 entradas y una salida tal que la salida sea 1 si y sólo si hay exactamente 2 entradas en 1. Pruebe el circuito obtenido con al menos 4 casos, es decir, simule que se ingresan valores (0 o 1) en las entradas y verifique si la salida obtenida es la esperada.

2.3.d Haga el circuito de un comparador de 1 bit. Debe tener 2 entradas (a y b) y 5 salidas (S_1 , S_2 , S_3 , S_4 , S_5), tales que:

- $S_1 = 1 \leftrightarrow a > b$
- $S_2 = 1 \leftrightarrow a \geq b$
- $S_3 = 1 \leftrightarrow a = b$
- $S_4 = 1 \leftrightarrow a \leq b$
- $S_5 = 1 \leftrightarrow a < b$

2.3.e Un número de 4 bits es Miti-Miti si sus dos bits más significativos son iguales a los 2 menos significativos. 0101 es Miti-Miti, pero 0110 no. Se pide hacer un circuito que devuelva 1 si la cadena de entrada es Miti-Miti.

2.3.f Construir un circuito que a partir de un número de 3 bits x_0, x_1, x_2 tenga 4 salidas que computen el resultado de hacer $3 * x$. Si el resultado no entra en 4 bits, debe devolver 1 en sus cuatro salidas.

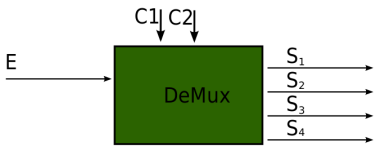
Ejemplos:

- $0, 0, 0 \rightarrow 0, 0, 0, 0$ porque $3 * 0 = 0$
- $0, 0, 1 \rightarrow 3$ porque $3 * 1 = 3$
- $1, 1, 1 \rightarrow$ porque $3 * 7 = 21$ y 21 no se puede representar con 4 bits

3 Circuitos famosos

Existe un conjunto de circuitos que son usados muchas veces en la construccion de una computadora. Esos son los que diseñaremos en esta sección.

3.a Un **demultiplexor** es un circuito usado para proyectar el valor que se recibe en la línea de entrada en una de las líneas de salida, dependiendo del estado de las dos líneas de control. Gráficamente es como sigue:



Si el demultiplexor tiene 2 líneas de control, 1 línea de entrada y 4 líneas de salida.

- Si c_1 y c_0 valen 0 entonces se proyecta en s_1 y las restantes salidas valen 0
- Si $c_1 = 0$ y $c_0 = 1$ entonces se proyecta en s_2 y las restantes salidas valen 0
- Si $c_1 = 1$ y $c_0 = 0$ entonces se proyecta en s_3 y las restantes salidas valen 0
- Si c_1 y c_0 valen 1 entonces se proyecta en s_4 y las restantes salidas valen 0

Diseñar el circuito.

3.b Dibujar el circuito de un decodificador de 2 líneas de entrada (e_i) y 4 líneas de salida (s_i), cuya tabla de verdad es la siguiente:

e_1	e_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

3.c **Usando el circuito anterior**, dibujar un demultiplexor de 1 línea de entrada, 2 líneas de control y 4 líneas salidas.

3.d Diseñar un *full adder* de 1 bit, es decir, un circuito capaz de sumar dos cadenas de 1 bit cada una y un carry de entrada, y cuyo resultado es una cadena de un bit y un carry de salida.

3.e Diseñar un restador de 1 bit:

3.f Diseñar un restador con carry de 1 bit, que tiene 3 entradas: los operandos mas el borrow de entrada (el anterior pidió 1) y tiene 2 salidas, el resultado de la resta y el nuevo borrow.

3.g Diseñar un *full adder* de 2 bits reusando circuitos que conozca. Un *full adder* de 2 bits tiene como entrada un carry anterior y dos cadenas de 4 bits cada una (es decir, 9 entradas en total), y su salida es una cadena de 4 bits y un carry de salida (es decir, 5 salidas en total).

3.h Diseñar un *restador* de 4 bits reusando circuitos que conozca.

3.i Rehacer el circuito 2.3.f usando sumadores.

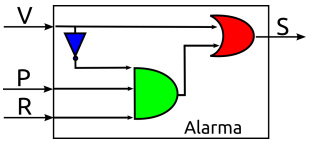
4 Autoevaluacion

1. Probá el circuito **demultiplexor** que definiste en el ejercicio 3.a con al menos 4 casos, es decir, simulá que se ingresan valores (0 o 1) en las entradas y verifique si la salida obtenida es la esperada.

2. Probá el circuito **decodificador** que definiste en el ejercicio b con al menos 4 casos.

3. Probá el circuito **comparador de un bit** que definiste en el ejercicio d con al menos 2 casos.

4. Se tiene un circuito y se sabe que su función es la de disparar una alarma en una habitación con dos sensores: uno en la única ventana y otro en la única puerta. Si se abre la ventana, la alarma debe sonar de inmediato, pero si se abre la puerta entonces comienza a correr un reloj que activa la entrada R luego de 5 segundos. El circuito es el siguiente:



¿Cómo se comprueba que el circuito cumple con su función?

5. ¿Para qué sirve una tabla de verdad?

References

[1] Williams Stallings, *Computer Organization and Architecture*, octava edición, Editorial Prentice Hall, 2010. **Apéndice A: Logica Digital**

[2] A. Tanenbaum, *Organización de Computadoras*, cuarta edición, Editorial Pearson. **Capitulo 3, secciones 1 y 2**