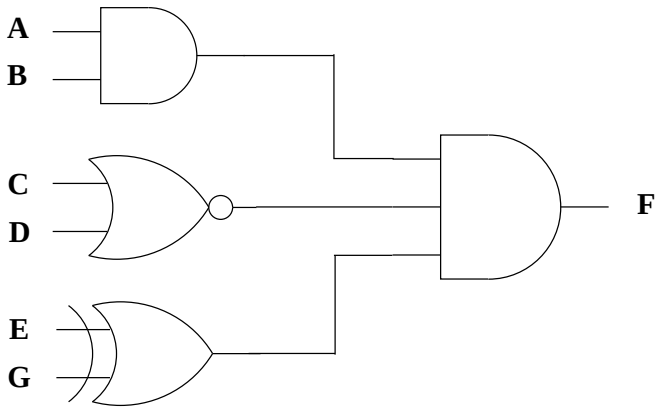
CIRCUITOS Y SISTEMAS DIGITALES

Departamento de Electronica y Comunicaciones
Universidad Pontifica de Salamanca en Madrid

Apuntes de clase



Juan González Gómez

Versión 0.3.7

Octubre-2002

Sobre estos apuntes

Estos apuntes se están realizando para cubrir el temario de la asignatura "*Circuitos y sistemas digitales*" , del Departamento de Electrónica y Comunicaciones, que se imparte en primero de Escuela y Facultad de Informática en la Universidad Pontificia de Salamanca en Madrid (**UPSAM**)[2].

Se han publicado bajo una **licencia libre**, de manera que se puedan *copiar*, *distribuir* y/o *modificar*. Y se ha hecho así por decisión del autor. El conocimiento siempre se ha difundido mediante las *copias*. En la edad media se realizaban copias a mano de los libros en los monasterios. Posteriormente con la aparición de la imprenta, ese proceso que tardaba muchos años se redujo drásticamente, permitiendo además obtener un número muchísimo mayor de *copias*, con lo que mayor cantidad de gente tenía acceso a los conocimientos. Actualmente tenemos prohibido copiar total o parcialmente los libros. Sólo hay que mirar las notas que aparecen en la contraportada. Y esta es una de las paradojas que existen hoy en día en el mundo de la enseñanza: El conocimiento lo puede transmitir el profesor oralmente, sin embargo, no es posible realizar copias del conocimiento que existe en los libros. Y si lo copias estás violando la ley.

Poco a poco, están apareciendo publicaciones y sobre todo software que permiten que se realicen copias. Es más, se incita a que se hagan estas copias, pues es la única manera de que se transmita el conocimiento. Y no sólo eso, sino que se permite su modificación, de manera que cada vez se vayan enriqueciendo más. Es un enfoque similar al del mundo científico: el descubrimiento de cada científico pasa a ser parte de la comunidad cientifica, para que otras personas los puedan utilizar para realizar nuevos descubrimientos.

Este es un asunto polémico, que visto desde una perspectiva científica tiene mucho sentido, pero visto desde una perspectiva comercial puede poner los pelos de punta a más de uno. A mí me gusta más el enfoque científico de la enseñanza. Y es ese conocimiento el que tiene que circular libremente, motivo por el cual estos apuntes tienen una licencia libre.

Licencia

Índice general

Índice de figuras

Capítulo 1

Introducción

Antes de entrar en los detalles de esta asignatura, es interesante tener una perspectiva mayor, para entender el contexto de esta asignatura, en qué fundamentos se basa y cómo se relaciona con el resto de asignaturas.

1.1. Electrónica

Esta asignatura trata sobre **Electrónica**. La Elecrónica estudia el comportamiento de los *electrones* en diversos medios, y se aplican estos conocimientos para conseguir que "***los electrones hagan lo que nosotros queramos***". Así por ejemplo, si construimos un circuito electrónico constituido por una pequeña bombilla, una pila y un interruptor (figura 1.1) y lo conectamos, lograremos que los electrones circulen por todo el circuito y que al atravesar la bombilla parte de ellos se conviertan en luz[1]. **¡¡Hemos conseguido que los electrones nos obedezcan!!**

Para "*dominar*" a los electrones, es necesario crear *circuitos electrónicos*, formados por materiales conductores (cables) que unen todos los componentes del circuito, de la misma manera que hay tuberías de agua que recorren nuestras casas, uniendo diferentes elementos: grifos, llaves de paso, el contador del agua...

El objetivo de la electrónica aplicada es construir circuitos electrónicos para que los electrones se comporten de la manera que a nosotros nos interese.

---

[1] No es el objetivo de estos apuntos el entrar en los detalles de los fenómenos físicos que subyacen en los cirucitos electrónicos. Se pretende que el alumno tenga una "intuición" de lo que está pasando.

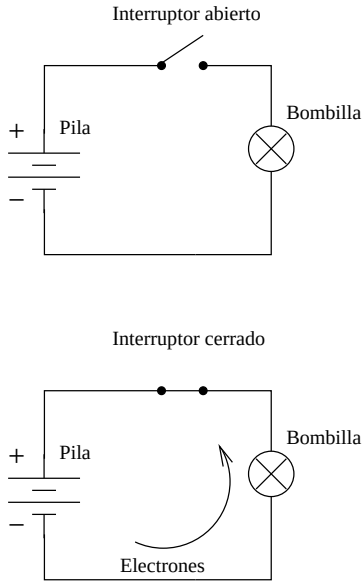


Figura 1.1: Un circuito electrónico muy simple: pila, interruptor y bombilla

1.2. Tipos de electrónica

1.2.1. Electrónica Analógica

Uno de los grandes retos del hombre es el de *manipular*, *almacenar*, *recuperar* y *transportar* la **información** que tenemos del mundo en el que vivimos, lo que nos permite ir progresando poco a poco, cada vez con más avances tecnológicos que facilitan nuestra vida y que nos permiten encontrar respuestas a preguntas que antes no se podían responder.

Ahora estamos viviendo un momento en el que esa capacidad de *manipulación*, *almacenamiento*, *recuperación* y *transporte* de la **información** está creciendo exponencialmente, lo que nos convierte en lo que los sociólogos llaman la "**Sociedad de la información**", y que tendrá (de hecho ya tiene) grandes implicaciones sociales.

Con la aparición de la electrónica las posibilidades para desarrollar esas capacidades aumentaron considerablemente. Para comprender los principios de la *electrónica analógica*, nos centraremos en un ejemplo concreto: **la manipulación, almacenamiento, recuperación y transporte de una voz humana**.

Cuando hablamos, nuestras cuerdas vocales vibran de una determinada manera, lo que originan que las moléculas del aire también lo hagan, chocando unas con otras y propagando esta vibración. Si no existiesen esas moléculas, como en el espacio, el sonido no se podría propagar.[2]

[2]Aunque en la mayoría de las películas de ciencia-ficción, cuando una nave destruye a otra en el espacio, se

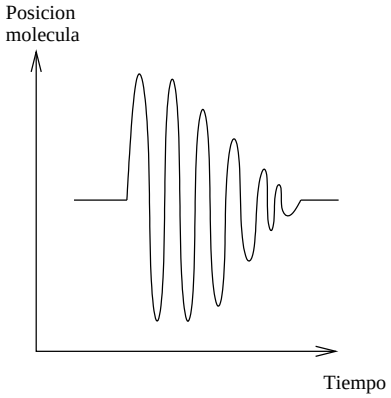


Figura 1.2: Un trozo de una señal acústica

Si medimos la vibración de una de estas moléculas, durante un intervalo corto de tiempo, y la pintamos, podría tener una pinta como la que se muestra en la figura 1.2. A esta vibración la llamaremos **señal acústica**.

Cuando esta señal acústica incide sobre un **micrófono**, aparece una **señal eléctrica** que tiene una forma ***análoga*** a la de la señal acústica. Las vibraciones de las moléculas se han convertido en variaciones del voltaje, que al final se traducen en vibraciones de los electrones. Es decir, que con los micrófonos lo que conseguimos es que **los electrones vibren de una manera análoga a cómo lo hacen las moléculas del aire** (ver figura 1.3).

Esta nueva señal eléctrica que aparece, se denomina **señal analógica**, puesto que es *análoga* a la señal acústica original. De esta manera, con señales eléctricas conseguimos imitar las señales del mundo real. Y lo que es más interesante, conseguimos que la información que se encuentra en la vibración de las moléculas del aire, pase a los electrones. Cuanto mejor sea el micrófono, más se parecerá la señal eléctrica a la acústica, y la información se habrá "copiado" con más fidelidad.

La **electrónica analógica** trata con este tipo de señales, análogas a las que hay en el mundo real, modificando sus características (ej. amplificándola, atenuándola, filtrándola...).

Fijémonos en el esquema de la figura 1.4. La persona que habla emite una **señal acústica** que es convertida en una **señal electrónica analógica** por el micrófono. Estas dos señales son muy parecidas, pero la que sale del micrófono es más pequeña. Por ello se introduce en un circuito electrónico, llamado amplificador, que la "agranda" (la ha ***manipulado***). A continuación esta señal se puede ***registrar*** en una cinta magnética de audio. Lo que se graba es una "copia" de la señal, pero ahora convertida a señal magnética. En cualquier momento la señal se puede

escucha un sonido de explosión.¡¡¡¡Fenómeno que es imposible!!!!!, pero que queda muy vistoso :-)

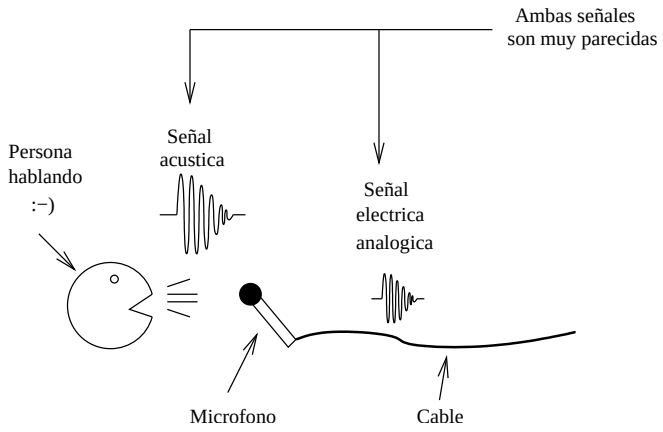


Figura 1.3: Conversión de una señal acústina en una señal eléctrica

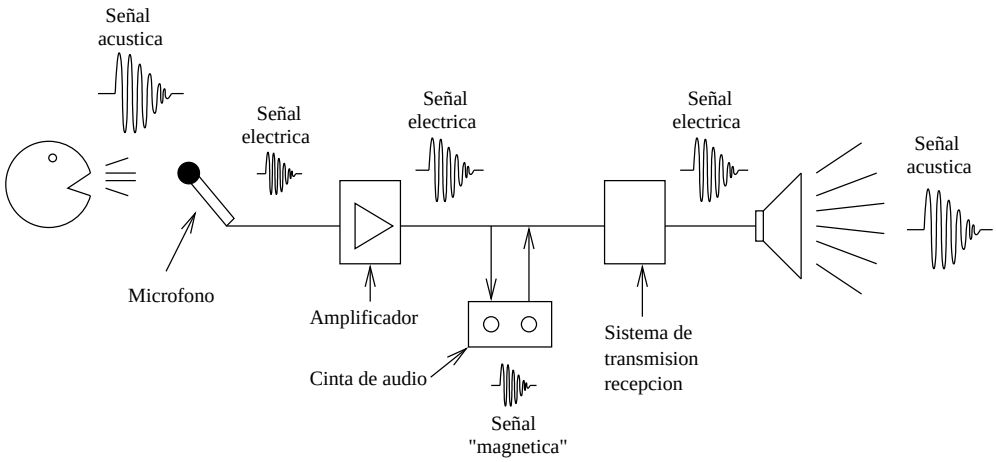


Figura 1.4: Un sistema de tratamiento de voz, con electrónica analógica

volver a ***recuperar***, convirtiéndose de señal magnética nuevamente a señal eléctrica. Una parte del sistema se ha llamado "sistema de transmisión-recepción" indicándose con esto que la señal eléctrica se puede ***transportar*** (Por ejemplo el sistema telefónico). Finalmente se introduce por un altavoz que relaliza la conversión inversa: pasar de una señal eléctrica a una acústica que se puede escuchar.

Los problemas de los sitemas analógicos son:

1. La información está ligada a la forma de la onda. Si esta se degrada, se pierde información
2. Cada tipo de señal analógica necesita de unos circuitos electrónicos particulares (No es lo mismo un sistema electrónico para audio que para vídeo, puesto que las señales tienen características completamente diferentes).

En las señales analógicas, la información se encuentra en la forma de la onda

---

1.2.2. Electrónica digital

Existe otra manera de *modificar*, *almacenar*, *recuperar* y *transportar* las señales, solucionando los problemas anteriores. Es un enfoque completamente diferente, que se basa en **convertir las señales en números**.

Existe un teorema matemático (teorema de muestreo de Nyquist) que nos garantiza que **cualquier señal se puede representar mediante números**, y que con estos números se puede ***reconstruir*** la señal original.

De esta manera, una señal digital, es una señal que está descrita por números. Es un conjunto de números. Y la **electrónica digital** es la que trabaja con señales digitales, o sea, con números. Son los números los que se *manipulan*, *almacenan*, *recuperan* y *transportan.*

Reflexionemos un poco. Estamos acostumbrados a escuchar el término televisión digital, o radio digital. ¿Qué significa esto? **¡¡¡Significa que lo que nos están enviando son números!!!!!** Que la información que nos envían está en los propios números y no en la forma que tenga la señal que recibidos. ¿Y qué es un sistema digital?, un sistema que trabaja con números. ¿Y un circuito digital? Un circuito electrónico que trabaja con números. ¡¡Y sólo con números!!

Si nos fijamos, con un ordenador, que es un sistema digital, podemos escuchar música o ver películas. La información que está almacenada en el disco duro son números.

En la figura 1.5 se muestra un sistema digital. La **señal acústica** se convierte en una **señal eléctrica**, y a través de un **conversor analógico-digital** se transforma en números, que son procesados por un **circuito digital** y finalmente convertidos de nuevo en una **señal electrónica**, a

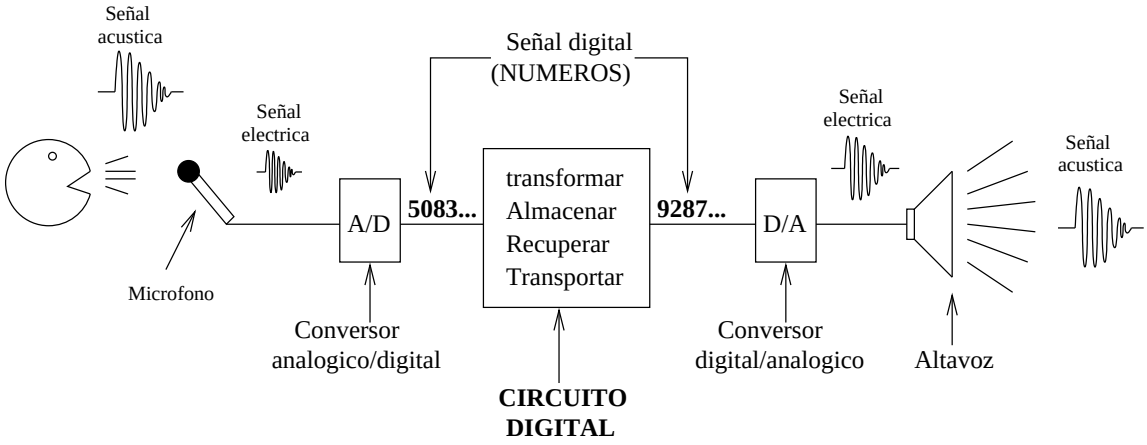


Figura 1.5: Sistema digital

través de un **conversor digital-analógico**, que al atravesar el altavoz se convierte en una **señal acústica**.

El utilizar circuitos y sistemas que trabajen sólo con números tiene una ventaja muy importante: se pueden realizar manipulaciones con independencia de la señal que se esté introduciendo: datos, voz, vídeo... Un ejemplo muy claro es internet. Internet es una red digital, especializada en la transmisión de números. Y esos números pueden ser datos, canciones, vídeos, programas, etc... La red no sabe qué tipo de señal transporta, "sólo ve números".

La electrónica digital trabaja con números. La información está en los números y no en la forma de señal. Cualquier señal siempre se puede convertir a números y recuperarse posteriormente.

1.3. Circuitos y sistemas digitales

Ya podemos entender de lo que trata esta asignatura. En ella estudiaremos y diseñaremos **circuitos digitales**, que manipulan números. Existen unos **números en la entrada** y nuestro circuitos generará otros **números de salida** (figura 1.6). Algunos números se considerarán como datos y otros se usarán para el control del propio circuito. No nos preocuparemos de dónde vienen estos números, pero ya sabemos que o bien vendrán de otro sistema digital, o bien de una *señal analógica* que se ha convertido a números (se ha digitalizado).

Un circuito digital realiza manipulaciones sobre los números de entrada y genera unos números de salida.

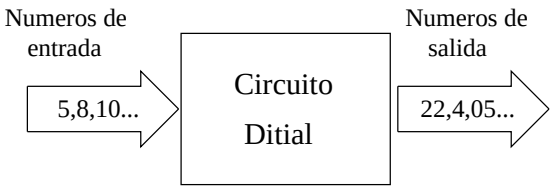


Figura 1.6: Un circuito digital genérico

1.4. Organización de los apuntes

En la introducción hemos visto la importancia que tienen los **números** en los *sistemas digitales*. En el capítulo 2 veremos las diferentes formas de representar un número y en concreto nos centraremos en el **sistema binario**. Para poder diseñar circuitos digitales, que manipulen números en binario, primero habrá que manejar las matemáticas que hay detrás: el **algebra de boole**, que se verá en el capítulo 3. Describiremos un tipo de circuitos, los **circuitos combinacionales**, mediante ***funciones booleanas*** y en el capítulo 4 veremos cómo se pueden implementar mediante **puertas lógicas**. En el capítulo 5 describiremos otros circuitos combinacionales más complejos, constituidos a partir de puertas lógicas, pero que se pueden considerar como componentes electrónicos: multiplexores, demultiplexores, codificadores, decodificadores, comparadores... y en el capítulo 7 cómo es posible realizar **operaciones aritméticas**. A partir del capítulo 8 se empiezan a ver **cirucitos secuenciales**, que se caracterizan porque pueden “recordar” o almacenar números. Los **biestables** nos permiten almacenar 1 bit de información y agrupándolos en **registros** (capítulo 9) almacenamos más información. Finalmente estudiaremos los **contadores** (capítulo 10) y los **autónomas finitos** (capítulo 11).

Capítulo 2

Sistemas de representación

2.1. Introducción

Hemos visto en el capítulo 1 cómo un **circuito digital trabaja con números** y sólo con números. El esquema general de estos circuitos se puede ver en la figura 2.1. Antes de entrar en la comprensión y diseño de estos circuitos, hay que estudiar **cómo se pueden representar esos números**, de manera que el circuito los entienda. Veremos que existen muchísimas formas de representar el mismo número (de hecho, existen infinitas formas), pero sólo unas pocas son las que nos interesarán para los *circuitos digitales*.

2.2. Conceptos

El concepto de número todos lo tenemos, pero **un mismo número se puede representar de muchas maneras**. Por ejemplo, el número 10, lo representamos mediante dos *dígitos*, el ’1’ y el ’0’. Si utilizásemos numeración romana, este mismo número lo representaríamos sólo con un único dígito ’X’. Pero está claro que ambas representaciones, “10” y “X” hacen referencia al mismo número diez.

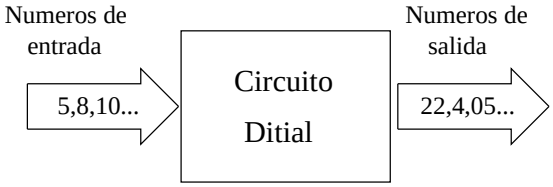


Figura 2.1: Un circuito digital genérico

Nosotros estamos acostumbrados a representar los números utilizando diez dígitos: ’0’, ’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’. Por eso nuestro sistema de representación se denomina **Sistema decimal** o **sistema en base diez**.

Analicemos con un poco más de detalle el *sistema decimal*, que es el que manejamos habitualmente. Vamos a representar el número “tres mil doscientos ochenta y uno”:

3281

Observamos lo siguiente:

- Está constituido por cuatro dígitos: ’3’,’2’,’8’ y ’1’.
- El **orden** en el que están colocados **es muy importante** y si se modifica, se está representando otro número.
- **Cuanto más a la izquierda está un dígito, más importante es**.

Este último punto es muy intuitivo. Imaginemos que el número 3281 representa el sueldo mensual de un ingeniero[1]. Si le preguntamos qué dígito es el que le gustaría modificar para tener un sueldo mayor, no dudaría en señalar al ’3’. “¡¡Ojalá me subieran en sueldo a 4281 euros!!” pensaría el ingeniero. Sin embargo, se echaría a reir si su jefe le dijese: “te subimos el sueldo a 3285 euros”.

El dígito ’3’ es más importante que todos los que tiene a su derecha. Tiene un **peso mayor** que el resto de dígitos. De hecho, este dígito ’3’ está representando al número tres mil. El dígito ’2’ por estar en tercera posición comenzado desde la derecha, representa el número doscientos, el ’8’ al ochenta y el ’1’ al uno. Podemos descomponer el número de la siguiente manera:

$$\begin{aligned} \mathbf{3281} &= 3000 + 200 + 80 + 1 = \\ &= \mathbf{3}\cdot 1000 + \mathbf{2}\cdot 100 + \mathbf{8}\cdot 10 + \mathbf{1} = \\ &\mathbf{3}\cdot 10^3 + \mathbf{2}\cdot 10^2 + \mathbf{8}\cdot 10^1 + \mathbf{1}\cdot 10^0 \end{aligned}$$

Observamos que cada dígito está multiplicando una pontencia de 10. Cuanto más a la izquierda se sitúe el dígito, mayor será la pontencia de diez por la que se multiplica.

En la figura 2.2 se muestra el número 3281 descompuesto en dígitos y pesos, y se indica cuál es el dígito de mayor peso y cuál es el de menor.

[1] Obviamente esto no se corresponde con la realidad :-)

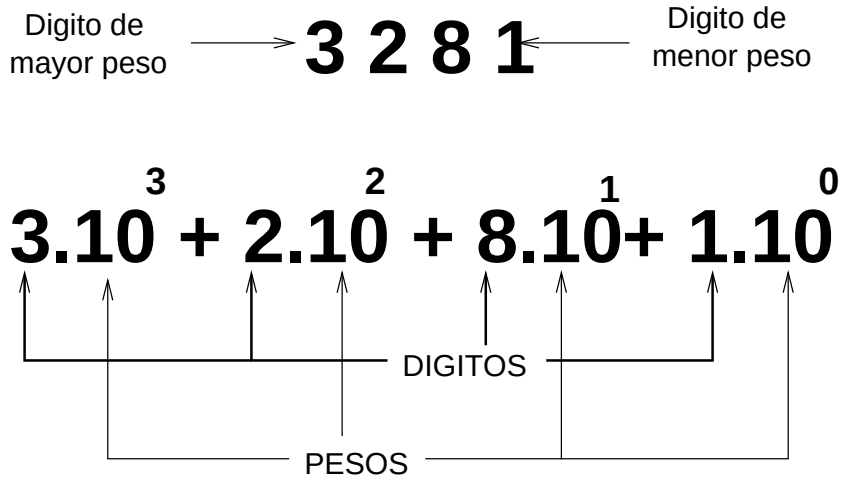


Figura 2.2: Dígitos y pesos del número 3281

Este sistema de representación también se llama **sistema en base diez** porque los pesos de los dígitos son potencias de 10: El dígito de más de la derecha tiene un peso de 10^0, los siguientes tienen pesos de 10^1, 10^2, 10^3, 10^4...

Nosotros representamos los números en el sistema decimal, que consta de diez dígitos diferentes, asignándoles un peso que es una potencia de diez, y que será mayor cuanto más a la izquierda se encuentre el dígito.

¿Qué nos impide que utilicemos unos sistemas de representación en los que los pesos de los dígitos, o incluso los dígitos sean diferentes de los del sistema decimal? Nada. Por ejemplo, podemos emplear un **sistema de representación octal** (Base 8), que utiliza sólo ocho dígitos (0,1,2...7) para representar cualquier número y los pesos de los diferentes dígitos serán potencias de 8. En este sistema, si escribimos los dígitos 352 no se corresponden con el número “trescientos cincuenta y dos” . Para calcular cuál es el número que representa hay que multiplicar cada dígito por su correspondiente peso, obteniendo **el número equivalente en el sistema decimal**.

$$\mathbf{352} = \mathbf{3}\cdot 8^2 + \mathbf{5}\cdot 8^1 + \mathbf{2}\cdot 8^0 =$$
$$3\cdot 64 + 5\cdot 8 + 2 = 248$$

El número 352 **en representación octal** es equivalente al número **248 del sistema decimal**. En el sistema octal, los dígitos tienen pesos que son potencias de 8, en lugar de potencias de 10 como en el **sistema decimal**. Para evitar confusiones cuando se trabaja con sistemas de representación diferentes, se emplea la siguiente notación:

$$352_8 = 248_{10}$$

El subíndice 8 indica que el número está representado en un sistema octal y con el subíndice 10 se indica que lo está en un sistema decimal.

2.3. Algunos sistemas de representación

2.3.1. Sistema octal (Base 8)

Ya lo hemos visto en el apartado de introducción. Utiliza ocho dígitos: 0,1,2,3,4,5,6 y 7 y los pesos son potencias de 8. No lo utilizaremos en esta asignatura.

2.3.2. Sistema binario (Base 2)

¿Se podrían utilizar sólo dos dígitos para representar cualquier numéro? Si, se denomina **sistema binario**. Este sistema de representación sólo utiliza los **dígitos 0** y **1 para representar cualquier número**. Fijémonos en lo interesante que resulta esto, *¡¡¡sólo con dos dígitos podemos representar cualquiera de los infinitos números!!!*

En el **sistema binario** los pesos de estos dígitos son pontencias de 2. Veamos un ejemplo del número binario 101001

$$101001 = \mathbf{1}\cdot 2^5 + \mathbf{0}\cdot 2^4 + \mathbf{1}\cdot 2^3 + \mathbf{0}\cdot 2^2 + \mathbf{0}\cdot 2^1 + \mathbf{1}\cdot 2^0 =$$
$$2^5 + 2^5 + 2^5 = 41$$

El número binario 101001 se corresponde con el número 41 en decimal.

El **sistema binario tiene mucha importancia** y **lo utilizaremos constantemente en esta asignatura**. Fijémonos en lo que significa esta forma de representación. Utilizando sólo dos dígitos, es posible representar cualquiera de los infinitos números. En la tecnología actual disponemos de un elemento, llamado **transistor**, que se puede encontrar en dos *estados* diferentes, abierto o cerrado[2], a los que le asociamos los dígitos 0 y 1. Todos los circuitos intregrados o chips se basan en estos transistores y trabajan internamente en binario. Todas las operaciones se realizan utilizando este sistema de representación, por eso es muy importante que lo conozcamos, para entender cómo funcionan los microprocesadores y los chips por dentro.

El sistema binaro utiliza sólo dos dígitos diferentes para representar cualquier número. El peso de los dígitos es una potencia de 2.

---

[2]El nombre técnico para estos estados es Corte y Saturación, pero es más intuitivo pensar en un transistor como en un pequeño interruptor que puede estar abierto o cerrado.

2.3.3. Sistema hexadecimal (Base 16)

¿Y sería posible utilizar más de 10 dígitos para representar los números?. También es posible. Ese es el caso del **sistema hexadecimal**, en el que se emplean **16 dígitos**: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F, donde las letras representan los números 10, 11, 12, 13, 14 y 15 respectivamente. Los pesos de los dígitos son pontencias de 16. Por ejemplo, el número hexadecimal FE2A se puede descomponer de la siguiente manera:

$$FE2A = F{\cdot}16^3 + E{\cdot}16^2 + 2{\cdot}16^1 + A{\cdot}16^0 =$$
$$15{\cdot}16^3 + 14{\cdot}16^2 + 2{\cdot}16^1 + 10{\cdot}16^0 = 65066$$

El **sistema hexadecimal** es muy curioso. Permite escribir números como los siguientes: CACA, DE, BACA :-). Se deja como ejercicio el obtener sus correspondientes números en el sistema decimal.

Este sistema, como veremos más adelante, se emplea para escribir números binarios de una manera más compacta, dado que el paso de hexadecimal a binario y vice-versa es inmediato.

2.4. Generalización

Dado un número de *m* dígitos ($a_m...a_0$), y usando un sistema en base *b*, se puede expresar en el **sistema decimal** utilizando la siguiente fórmula:

$$a_m a_{m-1}...a_2 a_1 a_0 = \sum_{i=m}^{0} a_i.b^i$$

Esta fórmula no es más que la generalización de los ejemplos expuestos en el apartado anterior. Si estamos trabajando con un sistema en base 7 (b=7) y el número que queremos convertir al sistema decimal tiene 4 dígitos (m=4), la fórmula de conversión sería:

$$a_3 a_2 a_1 a_0 = a_3{\cdot}7^3 + a_2{\cdot}7^2 + a_1{\cdot}7^1 + a_0{\cdot}7^0$$

En esta asignatura **nos centraremos en el sistema binario**, que será el que tendremos que comprender para utilizarlo en el diseño de circuitos digitales.

2.5. Tabla de conversión para los sistemas decimal- binario- hexadecimal

La tabla que se muestra a continuación representa las equivalencias entre diferentes números expresados en los sistemas **decimal**, **binario** y **hexadecimal**, que son los que más usaremos.

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ejercicios:

Hacer el ejercicio 1 de este capítulo.

2.6. Circuitos digitales y el Sistema binario

Ahora que ya tenemos un poco más claro el concepto de número y las diferentes formas que tenemos de representarlo, podemos retomar el esquema de un circuito digital (Figura 2.1) para precisarlo un poco más.

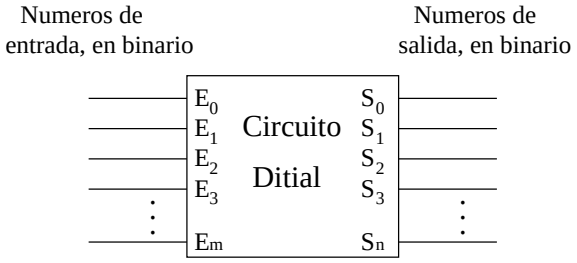


Figura 2.3: Un circuito digital genérico, con entradas y salidas binarias

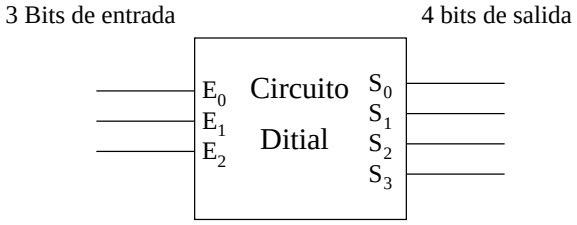


Figura 2.4: Un circuito digital con tres bits de entrada y 4 de salida

Con la tecnología que hay actualmente, los **circuitos digitales** manipulan números que están representados en binario. Así podemos decir que un **circuito digital** actual **tiene como entradas y salidas números en binario**. Es decir, números que vienen expresados con los dígitos '0' y '1'. En la figura 2.3 se ha dibujado un circuito digital genérico, en el que sus entradas y salidas se expresan en binario. Cada una de las entradas y salida representa un dígito binario. ¿Pero cual es el peso de este dígito? Eso nos lo indican los subíndices de las letras E y S. Así, la entrada E_0 se corresponde con el dígito de menor peso, la entrada E_1 con los dígitos de peso $2^1 = 2$, y así sucesivamente hasta la entrada *n* que es la de mayor peso. Lo mismo es aplicable a la salida.

En los circuitos digitales, los números que se procesan, están expresados en binario, tanto en la entrada como en la salida.

Un **dígito binario**, que puede ser '0' ó '1', recibe el nombre de **BIT**, del término ingles ***BI****nary digi****T*** (dígito binario). Utilizaremos los bits para indicar el tamaño de las entradas y salias de nuestros circuitos. Así por ejemplo podemos tener un circuito digital con 3 bits de entrada y 4 de salida. Este circuito se muestra en la figura 2.4.

Los circuitos digitales *sólo saben trabajar con números en binario*, sin embargo a los humanos nos es más cómodo trabajar en decimal. Trabajar con número binarios puede parecer "poco intuitivo". Vamos a ver cómo en determinadas ocasiones resulta muy intuitivo el trabajar con números binarios.

Imaginemos que en una habitación hay 5 bombillas situadas en la misma línea, y que cada

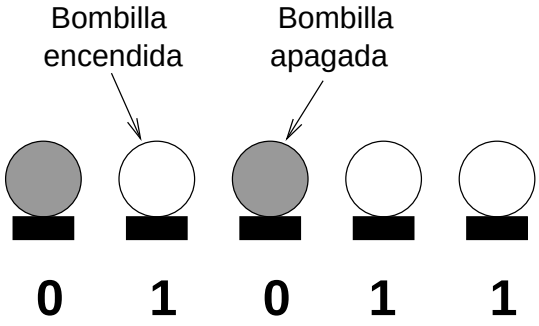


Figura 2.5: Utilización del sistema binario para expresar el estado de 5 bombillas

una de ellas puede estar encendida o apagada. ¿Cómo podríamos representar el estado de estas 5 bombillas mediante números? Una manera muy intuitiva sería utilizar el **sistema binario**, en el que utilizaríamos el **dígito 1** para indicar que **la bombilla está encendida** y el **dígito 0** para indicar que **está apagada**. Así el número 01011 nos indica que la primera bombilla está apagada, la segunda encendida, la tercera apagada y las dos últimas encendidas, como se muestra en la figura 2.5. Esta forma de representar el estado de las bombillas es bastante intuitivo. Este es un ejemplo en el que se puede ver que "pensar" en binario resulta más fácil que hacerlo directamente en decimal.

2.7. Sistema binario y sistema hexadecimal

El **sistema hexadecimal** se utiliza para **representar números binarios de una forma más compacta**. Cada dígito hexadecimal codifica 4 bits, de manera que un número hexadecimal de 4 bits permite representar un número binario de 16 bits. Veamos un ejemplo:

1011000111101101 = B1ED

Podemos ver cómo es mucho más cómodo utilizar el número hexadecimal que el binaro. Pero, *¿cómo se pasa de binario a hexadecimal o vice-versa?* El proceso es muy sencillo. Lo único que hay que conocer es la tabla del apartado 2.5. El número en binario hay que dividirlo en grupos de 4 bits empezando desde la derecha. La conversión del número binario anterior se haría de la siguiente manera:

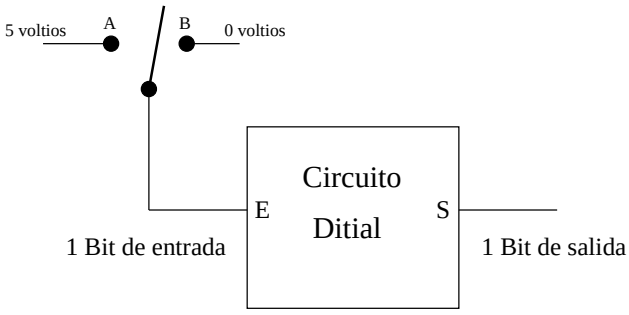
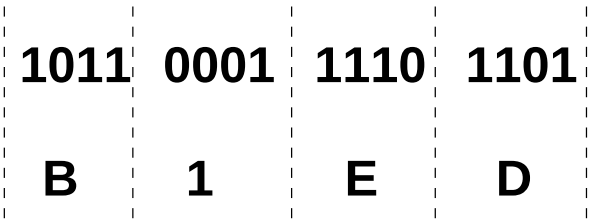


Figura 2.6: Cómo introducir dígitos binarios por un bit de la entrada de un circuito digital



Ejercicios:

Hacer los ejercicios 2 y 3 de este capítulo.

2.8. Bits y electrónica

Todavía nos queda una cosa por resolver. En la electrónica trabajamos con electrones, forzándolos a que hagan lo que nosotros queremos. En el caso de los circuitos digitales, lo que hacemos es operar con números. *¿Cómo conseguimos esto? ¿Cómo introducimos los números en los circuitos digitales?*

La solución a esto es ***asignar un voltaje a cada uno de los dos estados de un bit***. Lo normal, conocido como lógica TTL, es asignar el valor de 5 voltios al dígito ’1’ y 0 voltios al dígito ’0’. Esta asignación de valores depende de la tecnología empleada.

En la figura 2.6 se muestra un circuito digital que tiene un bit de entrada. Si queremos introducir un dígito ’1’ ponemos el interrupción en la posición A, de manera que por la entrada E llegan 5 voltios. Si queremos introducir un dígito ’0’ ponemos el interruptor en la posición B, por lo que llegan cero voltios.

En los circuitos digitales, se usan dos tensiones diferentes, una para representar el dígito '1' y otra para representar el dígito '0'. En la electrónica tradicional se usan 5 voltios para el digito '1' y 0 voltios para el digito '0'

2.9. Otros sistemas de representación

Para representar los números hemos visto que los circuitos digitales utilizan el sistema binario. Y hemos estado utilizando el **sistema binario natural**, en el que los bits tienen de peso potencias de 2, que es lo más habitual.

Sin embargo existen otros sistemas de representación que son binarios en el sentido de que sólo usan los dos dígitos '0' y '1', sin embargo tienen pesos diferentes. Algunos de estos sistemas, también conocidos como códigos son los siguientes:

1. **Código BCD**: Decimal Codificado en Binario. Es una manera de representar números decimales en binario. A cada dígito decimal se le asignan 4 bits, correspondientes a su número binario natural. Así por ejemplo para representar número decimal 21 en BCD, utilizaremos en total 8 bits, 4 para uno de los dos dígitos:

   $$21 = 0010\ 0001$$

   Los primeros 4 bits representan al dígito '2' y los 4 siguientes al dígito '1'.

2. **Código AIKEN**: Similar al BCD, pero con los pesos cambiados. Cada dígito decimal se representa mediante 4 bits, siendo los pesos de estos bits: 2, 4, 2 y 1.

3. **Código GRAY**: Son una familia de códigos que se caracterizan porque el paso de un número al siguiente implica que sólo se modifica un bit.

2.10. Terminología

BIT Dígito binaro. Un bit puede tomar los valores 0 ó 1. Es la abreviatura de las palabras inglesas de Binary digiT.

Byte Conjunto de 8 bits. El número más alto que se puede representar es el 11111111, que en decimal es 255.

2.11. Ejercicios resueltos

1. **Descomponer el número 63 en sus dígitos y pesos.**

   Solución:

   $$63 = 6{\cdot}10^1 + 3{\cdot}10^0$$

   Dígitos: ’6’ y ’3’ con pesos 10 y 1.

2. **Hacer lo mismo que en ejercicio 1, pero con el número 10358.**

   Solución:

   $$10358 = 1{\cdot}10^4 + 0{\cdot}10^3 + 3{\cdot}10^2 + 5{\cdot}10^1 + 8{\cdot}10^0$$

   Dígitos ’1’,’0’,’3’,’5’ y ’8’ con pesos 10000, 1000, 100, 10 y 1 respectivamente.

3. **Pasar los siguientes números al sistema decimal:**

   a) 1010111_2

   Solución:

   $1010111 = 1{\cdot}2^6 + 0{\cdot}2^5 + 1{\cdot}2^4 + 0{\cdot}2^3 + 1{\cdot}2^2 + 1{\cdot}2^1 + 1 \cdot 2^0 =$
   $2^6 + 2^4 + 2^2 + 2^1 + 2^0 =$
   64 + 16 + 4 + 2 +1 = 87

   b) $BABA_{16}$

   Solución:

   BABA=B$\cdot 16^3 + A{\cdot}16^2 + B{\cdot}16^1 + A{\cdot}16^0 =$
   $11{\cdot}4096$ + 10$\cdot$256 + 11$\cdot$16 + 10 =
   45056 + 2560 + 176 + 10 =47802

   c) 29_8

   Solución:

   $29_8 = 2{\cdot}8^1 + 9{\cdot}8^0$=2$\cdot$8 + 9 = 16 + 9 = 25

4. **Pasar de hexadecimal a binario:**

   a) FFFF

   Solución:

   FFFF = F - F - F - F = 1111-1111-1111-1111 = 1111111111111111

b) FA00

Solución:

FA00 = F-A-0-0 = 1111-1010-0000-0000 = 1111101000000000

c) 321C

Solución:

321C = 3-2-1-C = 0011-0010-0001-1100 = 11001000011100

2.12. Ejercicios

1. **Pasar los siguientes números a decimal**

   a) 347_8

   b) 2201_3

   c) $AF2_{16}$

   d) 10111_2

2. **Pasar de binario a hexadecimal**

   a) 0101101011111011

   b) 10010001110000101

   c) 1111000011110000

   d) 0101010110101010

3. **Pasar de hexadecimal a binario**

   a) FFFF

   b) 01AC

   c) 55AA

   d) 3210

Capítulo 3

ALGEBRA DE BOOLE

3.1. Introducción

Cuando trabajamos en **ingeniería**, utilizamos *ecuaciones* y *modelos matemáticos* que describen lo que estamos diseñando o analizando. Así por ejemplo, la ecuación

$$V_{max} = 2 \cdot W \cdot log_2 n$$

nos indica cuál es la velocidad máxima de transmisión por un canal que tiene un ancho de banda W y por el que se permiten n estados posibles de la señal transmitida, y será usada por un *Ingeniero de Telecomunicación* **para el diseño** de canales o sistemas de comunicación. Esa ecuación describe *una relación* entre ciertas variables, que son objeto de estudio del Ingeniero.

A lo mejor no entendemos el significado de esta ecuación. No sabemos lo que significa *ancho de banda* o *velocidad máxima de transmisión*, pero sí entendemos las operaciones que hay en ella: hay **productos** y **logaritmos**. Sin saber nada, y partiendo de los datos iniciales: $W = 2500$, n=4, seríamos capaces de calcular el valor de V_{max}:

$$V_{max} = 2 \cdot 2500 \cdot log_2 4 = 2 \cdot 2500 \cdot 2 = 10000$$

Sólo hay que introducir los datos en una calculadora y ya está.

De la misma manera, si un físico nos dice que la posición de cierta partícula viene determinada por la ecuación:

$$x = A \cdot \sin(wt + \varphi)$$

y nos da los siguientes datos: A=5, t=0 y $\varphi = 0$, sabemos calcular el valor de x, que será:

$$x = 5 \cdot \sin(w \cdot 0 + 0)$$

y por las propiedades de los *Números Reales*, que son los que estamos manejando, sabemos que "*algo por cero es cero*" y "*algo más cero es algo*":

$$x = 5 \cdot \sin(w \cdot 0 + 0) = 5 \cdot \sin(0 + 0) = 5 \cdot \sin(0) = 5 \cdot 0 = 0$$

¿Y por qué hemos sabido hacer eso? Porque conocemos las operaciones que el físico ha utilizado y además sabemos algunas propiedades de ellas.

En estas dos ecuaciones de ejemplo, **los números y las variables son Reales**. El conjunto de los *Números Reales* lo conocemos muy bien, así como todas las operaciones definidas en él. Estamos acostumbrados a trabajar con ellos desde pequeños, por eso este tipo de ecuaciones nos parecen *intuitivas* y sencillas, aunque no comprendamos lo que significan las variables usadas.

Hemos dicho que los *circuitos digitales* trabajan con números, y que estos números se expresan en *binario*. Veremos más adelante cómo **con un conjunto de ecuaciones podemos describir lo que hace un circuito**, que transforma los números de la entrada y los saca por la salida. Sin embargo, puesto que estos números vienen expresados en binario, **las variables y números utilizados NO SON REALES**.

Para describir un circuito digital utilizaremos ecuaciones

Para describir un circuito digital utilizaremos ecuaciones matemáticas. Sin embargo, estas ecuaciones tienen variables y números que NO SON REALES, por lo que NO podemos aplicar las mismas propiedades y operaciones que conocemos. Hay que utilizar nuevas operaciones y nuevas propiedades, definidas en el ALGEBRA DE BOOLE.

Por tanto, vamos a trabajar con unas ecuaciones a las que **NO estamos acostumbrados**. Son muy sencillas, pero al principio pueden resultar poco intuitivas. En este capítulo aprenderemos a trabajar con ellas.

3.2. Las operaciones del Álgrebra de Boole

En el *Álgebra de Boole* hay dos operaciones, denotadas con los símbolos + y · pero que ***¡¡no tienen nada que ver con las operaciones que todos conocemos de suma y producto!!***. ¡¡¡No

hay que confundirlas!!!!. El + y el · del *Algebra de Boole* se aplican a ***bits***, es decir, a números que sólo pueden ser el **'0'** ó el **'1'**.

3.2.1. La operación +

Esta operación se define de la siguiente manera:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

Las tres primeras operaciones nos resultan obvias, son iguales que la suma que conocemos, sin embargo la expresión $1 + 1 = 1$ nos puede resultar chocante. *¿¿Pero no me habían dicho toda la vida que 1+1=2??*, nos podemos estar preguntando. Sí, pero hay que recordar que aquí estamos utilizando otra operación que **NO ES LA SUMA**, la denotamos con el mismo símbolo '+', **¡¡pero no es una suma normal!! ¡¡Hay que cambiar el "chip"!! ¡¡Ahora estamos con Algebra de Boole!!**

Pasado el pánico inicial, si nos fijamos en esta nueva operación, notamos lo siguiente: **El resultado siempre es igual a '1' cuando alguno de los bits sumandos es igual a '1'**. O lo que es lo mismo, **El resultado de esta suma sólo da '0' si los dos bits que estamos sumando son iguales a cero**. En caso contrario valdrá '1'.

¿Y para qué nos sirve esta operación tan extraña? Veamos un ejemplo. Imaginemos que hay una sala grande a la que se puede acceder a través de dos puertas. En el techo hay una única lámpara y existen dos interruptores de luz, uno al lado de cada puerta de entrada. Como es lógico, *la luz se enciende cuando algunos de los dos interruptores (o los dos) se activan*. Esto lo podemos expresar mediante una *ecuación booleana*. Para denotar el estado de uno de los interruptores utilizaremos la **variable booleana A**, que puede valor '0' (Interruptor apagado) ó '1' (interruptor activado). Para el otro interruptor usaremos la **variable B**. Y para el estado de la luz, '0' (apagada) y '1' encendida, usaremos la **variable F**.

El estado en el que se encuentra la luz, en función de cómo estén los interruptores **viene dado por la ecuación booleana**:

$$F = A + B$$

que indica que F=1 (Luz encendida) si alguno de los interruptores está a '1' (activado).

Ya lo veremos más adelante, pero podemos ir adelantando unas propiedades muy interesantes. Si **A** es una variable boolena, se cumple:

- A + A = A
- 1 + A = 1
- 0 + A = A

3.2.2. La operación ·

Esta operación se define así:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

En este caso, la operación es más *intutitiva*, puesto que **es igual que el producto de *números Reales***. Si nos fijamos, vemos que **el resultado sólo vale '1' cuando los dos bits están a '1'**, o visto de otra manera, **el resultado es '0' cuando alguno de los dos bits es '0'**.

Vamos a ver un ejemplo. Imaginemos una caja de seguridad de un banco que sólo se abre cuando se han introducido dos llaves diferentes, una la tiene el director y la otra el jefe de seguridad. Si sólo se introduce una de ellas, la caja no se abrirá. Modelaremos el problema así. Utilizaremos la **variable A** para referirnos a una de las llaves ('0' no introducida, '1' introducida) y la **variable B** para la otra llave. Con la **variable F** expresamos el estado de la caja de seguridad ('0' cerrada y '1' abierta). El estado de la caja lo podemos expresar con la ecuación:

$$F = A \cdot B$$

que indica que la caja se abrirá (F=1) sólo si A=1 (una llave introducida) y B=1 (la otra llave introducida). En cualquier otro caso, F=0, y por tanto la caja no se abrirá.

Podemos ir adelantando algunas propiedades de esta operación:

- A·A=A
- A·0=0
- A·1=1

3.2.3. La negación

La operación de negación nos permite obtener el estado complementario del bit o variable booleana al que se lo aplicamos. Se define de la siguiente manera:

$$\overline{0} = 1$$
$$\overline{1} = 0$$

Es decir, que si se lo aplicamos a '0' obtenemos '1' y si se lo aplicamos al '1' obtenemos '0'. Esta operación nos permite cambiar el estado de una variable booleana. Si **A** es una variable boolena, $\overline{A}$ tiene el estado contrario.

3.3. Las propiedades del Álgebra de Boole

Las operaciones del *Álgebra de Boole* las podemos definir utilizando *tablas de verdad*:

- **Operación +**

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

- **Operación** $\cdot$

A	B	A$\cdot$B
0	0	0
0	1	0
1	0	0
1	1	1

Las **propiedades** del *Algebra de Boole* son las siguientes:

1. **Las operaciones + y $\cdot$ son CONMUTATIVAS**

   $A + B = B + A$

   $A \cdot B = B \cdot A$

2. **Elemento Neutro**

   A+0=A

   A·1=A

3. **Distributiva**

   $A + (B \cdot C) = (A + B) \cdot (A + C)$

   $A \cdot (B + C) = A \cdot B + A \cdot C$

4. **Elemento inverso**

   $A + \overline{A} = 1$

   A·$\overline{A}$=0

   Operación de **negación** definida por:

   $\overline{0} = 1$

   $\overline{1} = 0$

Ejercicios:

Para practicar e ir cogiendo soltura con el *Algebra de Boole* se recomienda hacer el **ejercicio 1** de este capítulo.

3.4. Teoremas importantes

Derivados de las propiedades fundamentales, existen una serie de *Teoremas* muy interesantes e importantes que usaremos a lo largo de todo el curso. Algunos los utilizaremos en la teoría y otros para los problemas.

- **Asociatividad**

  $A + B + C = (A + B) + C = A + (B + C)$

  $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$

- **Idempotencia**:

  $B + B = B$

  $B \cdot B = B$

- **Ley de Absorción**

  $A + A \cdot B = A$

  $A \cdot (A + B) = A$

  Este teorema es muy importante puesto que nos permite realizar simplificaciones en las expresiones.

- **Leyes de DeMorgan**

  $\overline{B_1 + B_2 + B_3 + ... + B_n} = \overline{B_1} \cdot \overline{B_2} \cdot \overline{B_3} \cdot ... \cdot \overline{B_n}$

  $\overline{B_1 \cdot B_2 \cdot B_3 \cdot ... \cdot B_n} = \overline{B_1} + \overline{B_2} + \overline{B_3} + ... + \overline{B_n}$

  Este teorema es también muy importante y lo usaremos constantemente. Vamos a hacer algunos ejemplos para aprender a utilizarlo:

  - **Ejemplo 1**: $\overline{A + B} = \overline{A} + \overline{B}$
  - **Ejemplo 2**: $\overline{A \cdot B + C.D} = \overline{A \cdot B} \cdot \overline{C \cdot D} = (\overline{A} + \overline{B}) \cdot (\overline{C} + \overline{D})$
  - **Ejemplo 3**: $\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C}$
  - **Ejemplo 4**: $\overline{A \cdot \overline{B} + \overline{C}} = \overline{A \cdot \overline{B}} \cdot \overline{\overline{C}} = (\overline{A} + \overline{\overline{B}}) \cdot C = (\overline{A} + B) \cdot C$

- **Teorema de Shannon**:

  $\overline{F(B_1, B_2, ..., B_n, +, \cdot)} = F(\overline{B_1}, \overline{B_2}, ..., \overline{B_n}, \cdot, +)$

  Este teorema es una generalización de las leyes de DeMorgan. Lo que nos dice es que si tenemos cualquier expresión booleana negada, es igual a la misma expresión en la que todas las variables estén negadas y en la que se sustituyan las operaciones + por . y viceversa.

  Veamos algunos ejemplos:

  - **Ejemplo 5**: $\overline{(B_1 + B_2) \cdot B_3} = (\overline{B_1} \cdot \overline{B_2}) + \overline{B_3}$

  En este este ejemplo se podrían haber aplicado las leyes de DeMorgan sucesivas veces, como hemos hecho en ejemplos anteriores, sin embargo podemos aplicar el *Teorema de Shannon*.

  - **Ejemplo 6**: $\overline{A.\overline{B} + \overline{C}}$=$(\overline{A} + B) \cdot C$
  - **Ejemplo 7**: $\overline{A \cdot \overline{B} \cdot \overline{C}} = \overline{A} + B + C$

- **Teorema de expansión**:

  $F(B_1, ..., B_n) = B_1 \cdot F(1, B_2, ..., B_n) + \overline{B_1} \cdot F(0, B_2, ..., B_n)$

  $F(B_1, ..., B_n) = [B_1 + F(0, B_2, ..., B_n)] \cdot [\overline{B_1} + F(1, B_2, ..., B_n)]$

  Este teorema es más teórico y no tiene aplicación directa en los problemas.

Ejercicios:

Hacer el ejercicio 2.

3.5. Funciones booleanas

3.5.1. Funciones reales y funciones booleanas

Hasta ahora hemos visto en qué operaciones se basa el Algebra de Boole y algunas de sus propiedades. Para aprender a trabajar con este nuevo tiepo de expresiones booleanas es necesario practicar, por eso se recomienda que se hagan los ejercicios propuestos.

Utilizando *expresiones booleanas*, vamos a definir **Funciones booleanas**, que son exactamente iguales a las funciones matemáticas a las que estamos habituados pero con la particularidad de que **las variables son booleanas** y que **los valores devueltos por la función también son booleanos,** es decir, **una función booleana sólo puede tomar los valores '0' ó '1'**.

Como hemos hecho antes, vamos a ver un ejemplo utilizando una función matemática de las que todos conocemos. Por ejemplo esta:

$$f(x) = x^2 + 1$$

Se trata de una *función Real* que tiene una *variable Real (x)*. Para cada valor de x, otenemos el valor de la función. Así por ejemplo podemos calcular los siguiente:

- $f(0) = 1$
- $f(1) = 2$
- $f(2) = 5$
- $f(3) = 10$

Como es una **función Real**, obtenemos como valores de la función **Números Reales**.

También podemos definir funciones reales de 2 ó más variables, como por ejemplo:

- $f(x, y) = x \cdot y + 3$. **Función de 2 variables**
- $g(x, y, z) = x \cdot y + z$. **Función de 3 variables**

Como estamos acostumbrados a trabajar con este tipo de funciones, nos resultan sencillas. Ahora vamos a definir **funciones booleanas**. Para ello hay que tener en mente que trabajaremos con variables booleanas y que por tanto usaremos las operaciones + y · del **Algebra de Boole**, y que como ya sabemos, nada tienen que ver con las operaciones suma y producto a las que estamos habituados.

Por ejemplo, sea la siguiente **función booleana de una variable**:

$$F(A) = \overline{A}$$

El valor devuelto por la función es el negado del que se le pasa por la variable. Como la variable A es booleana, sólo puede tomar los valores '0' y '1'. Los que la función F toma son:

- F(0)=$\overline{0}$=1
- F(1)=$\overline{1} = 0$

Vamos a definir una función un poco más compleja, usando dos variables booleanas, A y B:

$$F(A, B) = (A + B) \cdot \overline{B}$$

¿Cuando vale F(0,0)? sólo hay que sustituir en la función los valores de A y B por '0', obteniéndose:

- F(0,0)=(0+0)$\cdot\overline{0} = 0 \cdot 1 = 0$

Calcularemos el valor de F para el resto de valores de entrada de A y B:

- $F(0, 1) = (0 + 1) \cdot \overline{1} = 1 \cdot 0 = 0$
- $F(1, 0) = (1 + 0) \cdot \overline{0} = 1 \cdot 1 = 1$
- $F(1, 1)$ =Se deja como ejercicio para practicar (La solución es 0).

Fijándonos en esta función tan sencilla, podemos darnos cuenta de varias cosas:

1. Puesto que las variables de entrada A y B, sólo pueden tomar los valores '0' y '1', hay 4 casos distintos:

   a) A=0, B=0 $\Rightarrow F(0,0) = 0$

   b) A=0, B=1 $\Rightarrow F(0,1) = 0$

   c) A=1, B=0 $\Rightarrow F(1,0) = 1$

   d) A=1, B=1 $\Rightarrow F(1,1) = 0$

2. Antes de calcular los valores que toma la función, según lo que valgan A y B, se pueden aplicar algunas propiedades para obtener una **función más simplificada** (Como veremos en el apartado 3.7):

   $F(A,B) = (A+B) \cdot \overline{B}$ =$\{$Aplicando la propiedad distributiva$\}$=$A \cdot \overline{B} + B \cdot \overline{B} = A \cdot \overline{B}$

   Es más sencillo trabajar con esta función simplificada: $F(A,B) = A \cdot \overline{B}$

Las funciones booleanas pueden ser de muchas más variables, como en los siguientes ejemplos:

- $F(x,y,z) = x \cdot y + z$. Función booleana de **3 variables**
- $F(A,B,C,D) = \overline{A}.B + C.\overline{D}$. Función booleana de **4 variables**
- $F(E_4,E_3,E_2,E_1,E_0) = \overline{E_4} \cdot \overline{E_3} \cdot E_2 \cdot E_1 \cdot \overline{E_0}$. Función booleana de **5 variables**

Por cuestiones de comodidad, muchas veces no escribimos entre paréntesis las variables de la función, así por ejemplo podemos definir una **función de 3 variables** de la siguiente manera:

$$F = \overline{A} \cdot B + \overline{C}$$

Ejercicios:

Hacer el ejercicio 3

3.5.2. Funciones booleanas y tablas de verdad

Existen dos maneras de representar una función booleana. Una ya la conocemos, y es utilizado expresiones booleanas. Así por ejemplo se puede definir la función booleana siguiente:

$$F = A \cdot \overline{B}$$

y hemos visto cómo podemos obtener todos los valores de esta función.

Existe otra manera de especificar una función booleana y es utilizando las **tablas de verdad**. En ellas lo que estamos representando es el valor que debe tomar la función cuando las variables de entrada toman todos los valores posibles. Así por ejemplo yo puedo definir una función G de la siguiente manera:

A	**B**	**G**
0	0	0
0	1	1
1	0	0
1	1	1

¿Cuánto vale G si A=0 y B=1?. Miramos la tabla y vemos que G vale 1. Esta forma de definir funciones booleanas es muy sencilla. El número de filas de la tabla de verdad depende del número de variables que usemos.

Cuanto mayor número de variables, mayor cantidad de filas tendrá la tabla de verdad.

La regla que se cumple es la siguiente: "Si la función tienen n variables, la tabla de verdad tendrá 2^n filas". Veamos algunos ejemplos:

- Si una función tiene **2 variables**, su tabla de verdad tendrá **4 filas**
- Si la función tiene **3 variables**, la tabla tendrá 2^3 =**8 filas**
- Si la función tiene **4 variables**, la tabla tendrá 2^4 =**16 filas**
- .....

En la práctica no haremos tablas de verdad de más de 4 variables. Para eso están los ordenadores :-). Nosotros aprenderemos a definirlas y manejarlas.

Todavía hay algo que necesitamos conocer. **¿Qué relación hay entre una función definida mediante *expresiones booleanas* y una función definida mediante una *tabla de verdad*?** Es

decir, dada una tabla de verdad, ¿cómo podemos obtener la expresión booleana de la función? O dada una función mediante una expresión, ¿cómo obtenemos su tabla de verdad?.

Obtención de una tabla de verdad a partir de una expresión

Esto es bastante sencillo. Lo primero que hay que hacer es identificar el número de variables de la función, para conocer el tamaño de la tabla de verdad. A continuación escribimos números en binario en la parte de las variables. Finalmente vamos fila por fila obteniedo el valor de la función, utilizando la expresión.

Lo mejor es ver un ejemplo. Imaginemos que nos han dado la siguiente función, definida por la expresión:

$$F = \overline{A} \cdot B$$

1. La función tiene 2 variables, luego **la tabla de verdad tendrá** $2^2 = 4$ **filas**

2. Dibujamos una tabla de verdad con 4 filas, y ponemos en la parte de la izquierda el número de fila en *binario natural*, comenzando por la fila 0.

A	**B**	F
0	0	
0	1	
1	0	
1	1	

3. Aplicando la expresión, vamos calculando el valor de F. La primera fila se corresponde con F(0,0), la seguna con F(0,1), la tercera con F(1,0) y la última con F(1,1):

   - $F(0,0) = \overline{0} \cdot 0 = 1 \cdot 0 = 0$
   - $F(0,1) = \overline{0} \cdot 1 = 1 \cdot 1 = 1$
   - $F(1,0) = \overline{1} \cdot 0 = 0 \cdot 0 = 0$
   - $F(1,1) = \overline{1} \cdot 1 = 0 \cdot 1 = 0$

4. Ya podemos rellenar la tabla de verdad:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

Veamos otro ejemplo, ahora con una **función de 3 variables**:

$$G = A \cdot \overline{B} + C$$

1. Como la función tiene 3 variables, **la tabla de verdad tendrá** 2^3 **=8 filas**.

2. Dibujamos la tabla, poniendo en binario natural el número de fila, comenzando por 0:

A	**B**	**C**	**G**
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

3. Calculamos el valor de la función para cada una de las filas. El resultado se muestra a continuación, dejándose al lector su comprobación:

A	**B**	**C**	**G**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Ejercicios:

Hacer el ejercicio 4

Obtención de una expresión a partir de una tabla de verdad

Cuando diseñemos circuitos combinacionales, será muy normal que tengamos una tabla de verdad que haya que convertir a expresiones booleanas. El proceso es sencillo, sin embargo ocurre que ***dada una tabla de verdad se pueden obtener multitud de expresiones diferentes, todas ellas equivalentes***. Nuestra misión consistirá en obtener la expresión más simplificada posible. Esto lo iremos viendo en los siguientes apartados.

3.6. Formas canónicas

A partir de una tabla de verdad, podemos obtener múltiples expresiones para la misma función. Todas esas expresiones **son equivalentes** y podemos obtener unas expresiones de otras aplicando las propiedades del *Álgebra de Boole*.

Existen dos tipos de expresiones que se obtienen directamente de la tabla de verdad, de forma inmediata. Se denominan **formas canónicas**. ***Se caracterizan porque en todos los términos de estas expresiones aparecen todas las variables***.

3.6.1. Primera forma canónica

Una función que esté en **la primera forma canónica se caracteriza porque está formada por *sumas de productos***. Y recordemos que por ser una *forma canónica*, en todos sus términos se ecuentran todas sus variables.

Un ejemplo de una función de 3 variables, expresada en la primera forma canónica es la siguiente:

$$F = A \cdot B \cdot C + A \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C$$

Vemos que está constituida por la suma de tres términos y en cada uno de los términos están todas las variables.

La obtención de la primera forma canónica, a partir de una tabla de verdad es inmediato. El proceso se denomina “desarrollo de la tabla de verdad por unos”. Tomamos la tabla de verdad y sólo nos fijamos en las filas en las que la función vale ’1’, olvidándonos del resto. Por cada una de

estas filas tendremos un sumando, constituido por el producto de todas las variables, aplicando la siguiente regla:

Si una variable está a '0', en la fila escogida, usaremos la variable negada, y si está a '1' usaremos la variable sin negar.

Ejemplo:

Obtener la primera forma canónica, a partir de la siguiente tabla de verdad:

A	**B**	**C**	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Nos fijamos en las filas en las que F=1. Vemos que sólo hay tres filas, por tanto la función F se podrá expresar como suma de tres términos. Tomemos la primera fila en la que F=1. En ella vemos que A=0, B=0 y C=1, por tanto el primer término será $\overline{A} \cdot \overline{B} \cdot C$. Ahora nos fijamos en la siguiente fila en la que F=1: A=0, B=1 y C=1, por tanto el segundo término será: $\overline{A} \cdot B \cdot C$. Y por último nos fijamos en la última fila en la que F=1, en la que A=1, B=1 y C=1, por lo que el término será: $A \cdot B \cdot C$. **La función F será la suma de estos tres términos**:

$$F = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$$

Esta función está constituida por la suma de tres términos y en cada uno de ellos es el producto de las tres variables, bien negadas o no.

Vamos en algunos casos que esta expresión representa la misma función que la de la tabla de verdad:

1. Para A=0, B=1 y C=0, vemos en la tabla de verdad que F=0. Vamos a comprobarlo:

   $F = \overline{0} \cdot \overline{1} \cdot 0 + \overline{0} \cdot 1 \cdot 0 + 0 \cdot 1 \cdot 0 = 0 + 0 + 0 = 0$

2. Para A=0, B=1 y C=1, en la tabla de verdad F=1. Lo comprobamos:

   $F = \overline{0} \cdot \overline{1} \cdot 1 + \overline{0} \cdot 1 \cdot 1 + 0 \cdot 1 \cdot 1 = 1 \cdot 0 \cdot 1 + 1 \cdot 1 \cdot 1 + 0 = 1$

Se deja como ejercicio la comprobación para todos los demás casos.

Ejercicios:

Hacer los ejercicios 5 y 6.

Notación:

A cada uno de los sumandos de una expresión en la primera forma canónica, le corresponde una fila de la tabla de verdad, es decir, un número en decimal. Así en la función anterior:

$$F = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$$

el término $\overline{A} \cdot \overline{B} \cdot C$ representa la fila de la tabla de verdad en la que A=0, B=0 y C=1, que si se pone en decimal es el número 1.

De esta manera, esa función la podemos escribir de la siguiente manera:

$$F = \sum(1, 3, 7)$$

3.6.2. Segunda forma canónica

Un función en **la segunda forma canónica se caracteriza porque está formada por un *producto de sumas***. Y en todos sus términos deben aparecer todas sus variables, bien negadas o no. Por ejemplo:

$$F = (A + \overline{B} + C) \cdot (\overline{A} + B + C)$$

está constituida por dos términos que van multiplicados, y cada uno de ellos está formado por sumas.

La obtención de la segunda forma canónica, a partir de una tabla de verdad es inmediato. El proceso se denomina “desarrollo de la tabla de verdad por ceros”. Tomamos la tabla de verdad y sólo nos fijamos en las filas en las que la función vale ’0’, olvidándonos del resto. Por cada una de estas filas tendremos un término, constituido por la suma de todas las variables, aplicando la siguiente regla:

Si una variable está a '1', en la fila escogida, usaremos la variable negada, y si está a '0' usaremos la variable sin negar.

Es decir, que esta regla es justo la contraria que cuando estábamos trabajando con la primera forma canónica.

Ejemplo:

Obtener la segunda forma canónica, a partir de la siguiente tabla de verdad:

A	**B**	**C**	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Nos fijamos en las filas en las que F=0. En este ejemplo hay tres. Cada fila representa un término, que estará multiplicando al resto. Tomamos la primera fila en la que F=0 y vemos que A=0, B=1 y C=0. Aplicando la regla, el término que obtenemos es: $A + \overline{B} + C$. Para la siguiente fila en la que F=0, A=1, B=0 y C=0: $\overline{A} + B + C$ y finalmente, de la fila en la que A=1, B=1 y C=0 obtenemos: $\overline{A} + \overline{B} + C$. La función F desarrollada por la segunda forma canónica, queda:

$$F = (A + \overline{B} + C) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + C)$$

Se deja como ejercicio al lector el que compruebe que efectivamente esta expresión representa la misma función que la de la tabla de verdad.

Ejercicios:

Hacer los ejercicios 7 y 8.

Notación:

Para la segunda forma canónica se usa otra notación. La función F anterior está consituida por tres términos multiplicados. Si nos fijamos en el primero: $A + \overline{B} + C$, se corresponde con la fila de la tabla de verdad en la que A=0, B=1, C=0, que si lo ponemos en decimal es el número 2. De esta manera podemos usar la siguiente notación para representar a F:

$$F = \prod(2, 4, 6)$$

3.7. Simplificación de funciones booleanas

3.7.1. Introducción

En las matemáticas con *números Reales*, estamos muy acostumbrados a *simplificar*. De hecho es lo que nos han enseñado desde pequeños. Si una determinada expresión la podemos simplificar, ¿por qué no hacerlo?, así seguro que nos ahorramos cálculos. Por ejemplo, si vemos la siguiente ecuación:

$$\frac{4}{2}x - 2 = 2$$

lo primero que hacemos es simplificarla, aplicando primero que $\frac{4}{2} = 2$, quedando:

$$2x - 2 = 2$$

que todavía puede ser simplificada más, dividiendo por 2:

$$x - 1 = 1$$

Una vez simplificada es mucho más fácil trabajar.

Cuando estamos diseñando **circuitos digitales**, utilizaremos **funciones booleanas** para describirlos. Y antes de implementarlos, es decir, antes de convertir las ecuaciones a componentes electrónicos (puertas lógicas) tenemos que **simplificar al máximo**. Una de las misiones de los Ingenieros es **diseñar**, y otra muy importante es **optimizar**. ***No basta con realizar un circuito, sino que hay que hacerlo con el menor número posible de componentes electrónicos***. Y esto es lo que conseguimos si trabajamos con funciones simplificadas.

Las funciones booleanas se tienen que simplificar al máximo, para diseñar los circuitos con el menor número de componentes electrónicos.

Y este será uno de los grandes caballos de batalla de esta asignatura: **la simplificación de las funciones**. Esta simplificación la podemos realizar de dos maneras diferentes:

1. **Utilizando las propiedades y Teoremas del Algebra de Boole**. Se denomina método analítico de simplificación de funciones. Hay que manejar muy bien estas propiedades para poder eliminar la mayor cantidad de términos y variables.

2. **Utilizando el método de Karnaugh**. Es un método gráfico que si lo aplicamos bien, nos garantiza que obtendremos la función más simplificada posible, a partir de una tabla de verdad.

Normalmente las **formas canónicas** no son las expresiones más simplificadas.

3.7.2. Método analítico de simplificación de funciones

Desgraciadamente no exite tal métido :-(. Hay que basarse en la experiencia y en el conocimiento de las propiedades y teoremas del Algebra de Boole. Lo mejor es ver un ejemplo:

Ejemplo:

Simplificar la siguiente función:

$$F = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.B.\overline{C}$$

Vamos a intentar aplicar la propiedad distributiva, lo que normalmente llamamos sacar factor común. Operando con los términos 1 y 3:

$$\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C = \overline{A} \cdot C(\overline{B} + B) = \overline{A} \cdot C$$

Operando con los términos 2 y 4:

$$\overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot \overline{C} = B \cdot \overline{C} \cdot (\overline{A} + A) = B \cdot \overline{C}$$

La función que nos queda es:

$$F = \overline{A} \cdot C + B \cdot \overline{C}$$

Tanto la función inicial, como la que hemos obtenido son funciones equivalentes. Tienen la misma tabla de verdad, sin embargo, la segunda está mucho más simplificada: sólo tiene dos sumandos y cada sumando tiene sólo dos variables.

Ejemplo:

Simplificar la siguiente función:

$$G = E_2 \cdot \overline{E_1} \cdot E_0 \cdot (\overline{E_1} + E_2) \cdot E_1 + \overline{E_2} \cdot E_1$$

Si nos fijamos, vemos que podemos reordenar la función de manera que quede:

$$G = \overline{E_1} \cdot E_1 \cdot E_2 \cdot E_0 \cdot (\overline{E_1} + E_2) + \overline{E_2} \cdot E_1$$

y puesto que $\overline{E_1} \cdot E_1 = 0$ y cualquier cosa multiplicada por 0 es 0, al final nos queda:

$$G = \overline{E_2} \cdot E_1$$

3.7.3. Método de Karnaugh

En este apartado veremos ***un método para obtener la función más simplificada a partir de una tabla de verdad***.

Vamos a ir poco a poco, viendo los fundamentos de este método. Supongamos que tenemos una función *F(A,B,C)* de tres variables, cuya tabla de verdad es:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Si la desarrollamos por la primera forma canónica obtenemos:

$$F = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

Veremos como aplicando el método de Karnaugh podemos simplificar esta función.Vamos a organizar esta misma tabla de la siguiente manera:

A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	1	1

Observamos lo siguiente:

- En total hay 8 casillas, cada una correspondiente a una fila de la tabla de verdad
- En cada casilla está colocado el valor de la función F, correspondiente a esa entrada. En la tabla de verdad hay dos filas en las que F=0 y 6 filas en las que F=1. En el nuevo diagrama hay dos casillas con ’0’ y 6 con ’1’.
- Hay dos filas, en al primera fila están todos los valores de F correspondientes a A=0, y en la segunda correspondientes a A=1.
- Hay 4 columnas, y el número que está en la parte superior de cada una de ellas nos indica los valores de las variables B y C en esa columna.
- Dada una casilla cualquiera, mirando el número situado en la misma fila, a la izquierda del todo nos informa del valor de la variable A y los dos valores superiores, en la misma

columna, nos dan los valores de B y C. Así por ejemplo, si tomamos como referencia la casilla que está en la esquina inferior derecha, se corresponde con el valor que toma F cuando A=1, B=1 y C=0.

- Entre dos casillas adyacentes cualesquiera, sólo varía una variable de entrada, quedando las otras dos con los mismos valores. Por ejemplo, si estamos en la casilla inferior derecha, en la que A=1, B=1 y C=0. Si vamos a la casilla que está a su izquierda obtenemos un valor de las variables de: A=1, B=1, C=1. Si lo comparamos los valores de las variables correspondientes a la casilla anterior, vemos que sólo ha cambiado una de las tres variables, la C. Lo mismo ocurre si nos desplazamos a cualquier otra casilla adyacente.

Ahora vamos a ver una propiedad "mágica" de esta tabla. Si obtenemos la primera forma canónica, obtenemos una función con 6 términos. Vamos a fijarnos sólo en los términos que obtenemos si desarrollamos sólo dos casillas adyacentes, como por ejemplos las marcadas en gris en la figura:

BC / A	00	01	11	10
0	0	0	1	1
1	1	1	1	1

Los valores de las variables en estas casillas son: A=1, B=1, C=1 y A=1, B=1, C=0. Si obtenemos los términos de la primera forma canónica y los sumamos:

$$A \cdot B \cdot C + A \cdot B \cdot \overline{C} = A \cdot B \cdot (C + \overline{C}) = A \cdot B$$

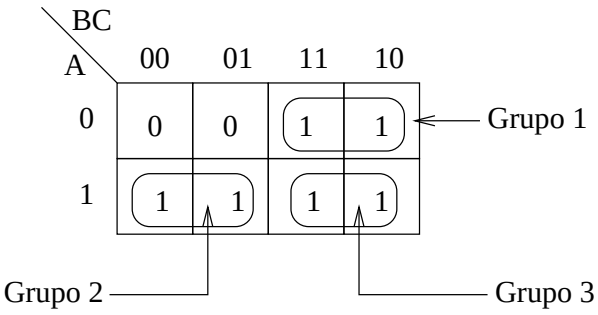
¡¡Se nos han simplificado!! Es decir, por el hecho de agrupar los términos obtenidos de estas dos casillas y sumarlos, se han simplificado. Y esto es debido a la propiedad antes comentada de que entre dos casillas adyacentes sólo varía una de las variables, de manera que podemos sacar factor común. Estos dos términos son los sumandos 5 y 6 de la primera forma canónica obtenida anteriormente, que al sumarlos y aplicar aglunas propiedades se han simplificado.

Si nos fijamos en estas dos casillas adyacentes, la variable C, que es la única que varía de una a otra, ha desaparecido en la suma. De esta manera podemos afirmar lo siguiente:

Si tomamos dos casillas adyacentes cuyo valor es '1' y desarrollamos por la primera forma canónica, desaparecerá una de las variables. Sólo permanecen las variables que no cambian

de una casilla a otra.

De esta manera, vamos a ver qué pasa si tomamos los siguientes grupos:



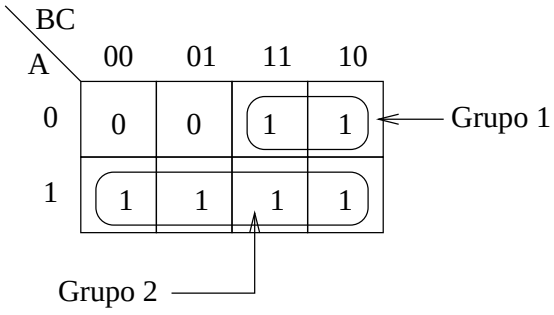
y sumamos los términos de estos grupos:

- **Grupo 1**: $\overline{A} \cdot B \cdot C + \overline{A} \cdot B \cdot \overline{C} = \overline{A} \cdot B \cdot (C + \overline{C}) = \overline{A} \cdot B$
- **Grupo 2**: $A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C = A \cdot \overline{B} \cdot (\overline{C} + C) = A \cdot \overline{B}$
- **Grupo 3**: El que teníamos antes: $A \cdot B$

Por tanto, la función F también la podemos expresar como suma de estos grupos:

$$F = \overline{A} \cdot B + A \cdot \overline{B} + A \cdot B$$

¡¡Y está más simplificada que la forma canónica!! Pero...¿Se puede simplificar más? ¡Si!. Inicialmente la función F tenía 6 sumandos, puesto que tenía 6 unos. Al hacer 3 grupos, ahora tiene 3 sumandos. ¿Podemos reducir el número de grupos? Si, vamos a ver qué pasa si tomamos los siguientes grupos:



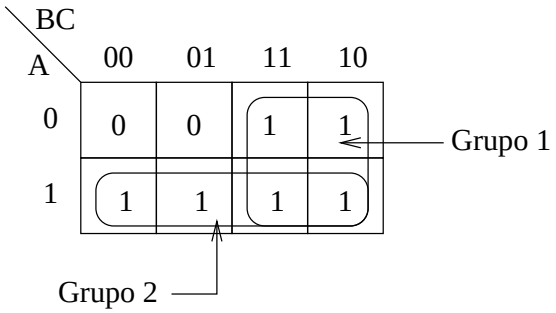
Ahora sólo hay 2 grupos. El nuevo grupo 2 está constituido por 4 casillas en las que F=1. La expresión de este grupo se obtiene sumando las expresiones de estas 4 casillas. Las nuevas expresiones de los grupos quedarían:

- **Grupo 1**: Igual que antes: $\overline{A} \cdot B$
- **Grupo 2**: $A \cdot \overline{B} + A \cdot B = A \cdot (\overline{B} + B) = A$

La nueva función F que obtenemos es:

$$F = \overline{A} \cdot B + A$$

¡¡Que está más simplificada que la anterior!! Pero... ¿Es la más simplificada? No, todavía podemos simplificarla más. ¿Por qué no podemos tomar 2 grupos de 4 casillas adyacentes?. Tomemos los grupos siguientes:



Las nuevas expresiones de los grupos son:

- **Grupo 1**: $\overline{A} \cdot B + A \cdot B = B \cdot (\overline{A} + A) = B$
- **Grupo 2**: Igual que antes: A

Por tanto, la nueva función F simplificada es:

$$F = A + B$$

¡¡¡Esta función está simplificada al máximo!!!

Criterio de máxima simplificación:

Para obtener una función que no se puede simplificar más hay que tomar el menor número de grupos con el mayor número de '1' en cada grupo.

Hay que tener en cuenta que los grupos cd unos que se tomen sólo pueden tener un tamaño de 1, 2, 4, 8, 16,... (es decir, sólo potencias de dos). Esa es la razón por la que en el ejemplo anterior los grupos que se han tomado son de tamaño 4 (y no se han tomado de tamaño 3).

Fijémonos en todas las funciones que hemos obtenido anteriormente:

- $F_C = \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + A.B.C$ (**CANONICA**)
- $F_1 = A.\overline{B} + A.B + \overline{A}.B$ (**3 grupos de 2 1's por grupo**)
- $F_2 = A + \overline{A}.B$ (**1 grupo de 4 1's y 1 grupo de 2 1's**)
- $F_3 = A + B$ (**2 grupos de 4 1's**)

¡¡Todas son funciones booleanas equivalentes!! (Porque tienen la misma tabla de verdad). ¡¡Pero es la función F_3 la que usamos!! ¡¡Sómos Ingenieros y queremos optimizar al máximo!!!

Ejemplo

Veamos con un ejemplo cómo podemos aplicar directamente el criterio para obtener una función simplificada. Dada la siguiente tabla de verdad, obtener la expresión de F más simplificada posible:

A	**B**	**C**	**F**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Colocamos la tabla de verdad como un diagrama de Karnaugh y hacer tres grupos de dos unos:

A \ BC	00	01	11	10
0	0	1	1	1
1	0	0	1	0

La función F la obtenemos sumando las expresiones de los tres grupos, siendo cada uno de ellos el producto de las dos variables booleanas que permanecen sin cambios dentro de cada grupo:

$$F = \overline{A} \cdot C + \overline{A} \cdot B + B \cdot C$$

Como hemos aplicado correctamente **el criterio de máxima simplificación**, tenemos la certeza absoluta de que esta es la expresión más simplificada posible para la función F.

A la hora de formar los grupos hay que tener en cuenta que las casillas situadas más a la derecha de la tabla son adyacentes a las que están más a la izquierda. Veamos un ejemplo:

Ejemplo:

Simplificar la siguiente función, utilizando el método de Karnaugh:

A	**B**	**C**	**F**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Lo representamos en un diagrama de Karnaugh y tomamos el siguiente grupo:

BC / A	00	01	11	10
0	1	0	0	1
1	1	0	0	1

con el que obtenemos la siguiente función simplificada:

$$F = \overline{C}$$

Funciones de 4 variables

¿Y qué ocurre si tenemos una función de 4 variables? La idea es la misma pero tendremos una tabla más grande. El criterio de máxima simplificación es el mismo: hacer el menor número posible de grupos con el máximo número de ’1’s. Veamos un ejemplo:

Ejemplo:

Dada la siguiente tabla de verdad, obtener la expresión de F más simplificada posible:

A	**B**	**C**	**D**	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Lo primero que hacemos es pasarlo a un diagrama de Karnaugh, de la siguiente manera (cuidado de no confundirse!!):

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	1
10	1	0	0	1

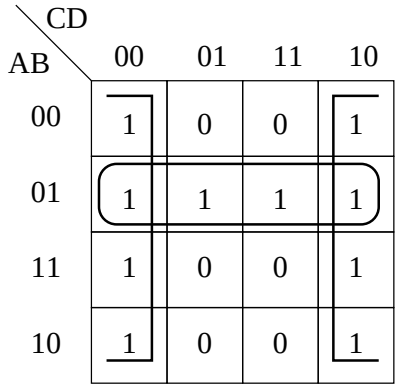
Vemos que ahora en la izquierda de la tabla están los valores de las variables A y B y en la parte superior los valores de C y D. Lo siguiente es agrupar los ’1’s. Vamos a hacer primero los siguientes grupos:

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	1
10	1	0	0	1

La expresión que obtenemos es:

$$F = \overline{C} \cdot \overline{D} + C \cdot \overline{D} + \overline{A} \cdot B$$

Sin embargo, ¿es esta la función más simplificada? O lo que es lo mismo, podemos hacer menos grupos de ’1’s. La respuesta es sí, porque no olvidemos que las casillas de la derecha son adyacentes a las de la izquierda de la tabla, por lo que podemos hacer sólo dos grupos:



Un grupo es de 8 unos y el otro de 4. Obtenemos la siguiente función:

$$F = \overline{D} + \overline{A} \cdot B$$

Esta sí es la más simplificada.

Ejercicios:

Hacer el ejercicio 9.

3.8. La operación ⊕

Hay un operación que en electrónica digital se utiliza mucho, llamada XOR y que se denota por el símbolo ⊕. Esta operación la podemos definir mediante una tabla de verdad:

A	**B**	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Fijándonos en esta tabla podemos ver lo que hace: esta operación devuelve '0' cuando los dos bits sobre los que operan son iguales, y '1' cuando con distintos. Tanto esta operación como su negada, $\overline{A \oplus B}$, las utilizaremos mucho, por ello vamos a ver cómo las podemos definir a partir de las operaciones + y ·, y ver algunas de sus propiedades.

Partiremos de la tabla de verdad, en la que además representaremos la operación negada:

A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Vamos a obtener las dos formas canónicas de ambas funciones. Estas expresiones las utilizaremos bastante:

$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B} = (A + B) \cdot (\overline{A} + \overline{B})$$

$$\overline{A \oplus B} = \overline{A} \cdot \overline{B} + A \cdot B = (A + \overline{B}) \cdot (\overline{A} + B)$$

Y la siguiente propiedad también es muy interesante:

$$\overline{A \oplus B} = \overline{A} \oplus B = A \oplus \overline{B}$$

Ejercicios:

Hacer el ejercicio 10.

3.9. Resumen

En este capítulo se han presentado las **herramientas matemáticas** que nos servirán para analizar y diseñar *circuitos digitales*. Trabajaremos con **dígitos binarios** o **bits** que pueden estar en dos estados '0' ó '1', sobre los que se definen las **operaciones + y** $\cdot$, del **Algebra de Boole**, y que no hay confundir con las operaciones de suma y producto a las que estamos acostumbrados.

Hemos vista una serie de propiedades y teoremas que nos permiten trabajar con *expresiones booleanas* y con los que es necesario practicar, haciendo los ejercicios indicados.

También hemos visto el concepto de **función boolena** y cómo podemos representar cualquier función de este tipo mediante **tablas de verdad** o mediante expresiones booleanas. También hemos visto cómo es posible obtener una *tabla de verdad* a partir de una *expresión booleana* y cómo obtener una e*xpresión booleana* a partir de la *tabla de verdad*.

Dada una *tabla de verdad*, existen multitud de *expresiones booleanas*, todas ellas equivalentes, que se pueden obtener. Sin embargo, hemos visto cómo es inmediato obtener **la primera**

y segunda forma canónica. Sin embargo, las funciones así obtenidas no tienen porqué ser las más simplificadas posibles. Para simplificar una función podemos utilizar las propiedades del *Algebra de Boole*, o también podemos utilizar el **método de Karnaugh**, que si lo aplicamos correctamente, conseguiremos obtener **la función más simplificada posible**.

Finalmente hemos visto una nueva operación, $\oplus$, que se define a partir de las operaciones + y $\cdot$, y que es conveniente que conozcamos puesto que la usaremos bastante.

Para repasar con todos estos conceptos se recomienda hacer todos los ejercicios y los problemas de los apartados 3.10

3.10. Ejercicios

Ejercicio 1:

Realizar las siguientes operaciones:

1. 1 + 0 =
2. 1 + 1 =
3. 1·0 =
4. 1·1 =
5. A+0 =
6. A+1=
7. A·1=
8. A·0=
9. A+A=
10. A.A=
11. A+$\overline{A}$=
12. A·$\overline{A}$=
13. A+AB =

14. A(A+B) =

15. A+AB+B =

Ejercicio 2:

Aplicar las leyes de Morgan en los siguientes casos:

1. $\overline{A(B+C)}$=

2. $\overline{\overline{AB+CD}\cdot E}$=

3. $\overline{(AB+CD)\cdot E}$=

Ejercicio 3:

Obtener el valor de las siguientes funciones booleanas, en todos los casos.

1. $F = A + B$

2. $F = A + \overline{B}$

3. F=$\overline{A}\cdot B + C$

Ejercicio 4:

Dadas las siguientes funciones booleanas, obtener su correspondiente tabla de verdad

1. $F = A + \overline{B}$

2. $G = A \cdot B + \overline{A} \cdot B$

3. $H = X \cdot Y \cdot \overline{Z} + \overline{X} \cdot \overline{Y} \cdot Z$

4. $S = E_3E_2E_1E_0 + E_3\overline{E_2}$

Ejercicio 5:

Desarrollar las siguientes tablas de verdad por la primera forma canónica:

1. Tabla 1:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

2. Tabla 2:

A	**B**	**C**	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Ejercicio 6:

Dadas las siguientes funciones, indicar si se encuentra expresadas en la primera forma canónica, y si es así, obtener la tabla de verdad

1. $F = \overline{A} \cdot B + A \cdot B$
2. $F = A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$
3. $F = E_2 \cdot E_1 \cdot E_0 + \overline{E_2} \cdot E_1 \cdot E_0 + E_1$
4. $F = E_2 \cdot E_1 \cdot E_0 + \overline{E_2} \cdot E_1 \cdot E_0 + \overline{E_2} \cdot \overline{E_1} \cdot E_0$

Ejercicio 7:

Desarrollar las siguientes tablas de verdad por la segunda forma canónica:

1. Tabla 1:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

2. Tabla 2:

A	**B**	**C**	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Ejercicio 8:

Dadas las siguientes funciones, indicar si se encuentra expresadas en la primera forma canónica o en la segunda. En caso de que así sea, obtener la tabla de verdad.

1. $F = (A + B) \cdot (\overline{A} + \overline{B})$
2. $F = \overline{A} \cdot B + \overline{B} \cdot A$
3. $F = (E_2 + \overline{E_1} + E_0) \cdot (\overline{E_2} + \overline{E_1} + E_0) \cdot (E_2 + E_1 + E_0)$
4. $F = E_2 \cdot \overline{E_1} \cdot E_0 + \overline{E_2} \cdot \overline{E_1} \cdot E_0 + E_2 \cdot E_1 \cdot E_0$
5. $F = (A \cdot B \cdot C) + (A + B + C)$

Ejercicio 9:

Obtener las expresiones más simplificadas a partir de las tablas de verdad:

- Tabla 1:

A	**B**	**C**	**D**	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

- Tabla 2:

E_3	E_2	E_1	E_0	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Ejercicio 10:

Operar con las siguientes expresiones obteniendo la mayor cantidad posible de operaciones $\oplus$

1. $A \cdot \overline{B} + \overline{A} \cdot B =$
2. $A \cdot B + \overline{A} \cdot \overline{B} =$
3. $(A \cdot \overline{B} + \overline{A} \cdot B) \cdot \overline{C} + \overline{(A\overline{B} + \overline{A} \cdot B)} \cdot C =$
4. $\overline{A} \cdot B + \overline{A} \oplus B + \overline{A \oplus \overline{B}} + A \cdot \overline{B} =$

Ejercicio 11:

Dejar las siguientes expresiones en forma de sumas de productos:

1. (x + y + z)($\overline{x}$+z) =

2. $\overline{(\overline{x}+y+z)}.(\overline{y}+z) =$
3. $\overline{x\overline{y}z}.\overline{\overline{x}yz} =$

Ejercicio 12:

Simplificar la función $F = A \cdot B + \overline{B}$ de las siguientes maneras:

1. Obteniendo la tabla de verdad y aplicando Karnaugh
2. Aplicando las propiedades del Algebra de Boole

Capítulo 4

CIRCUITOS COMBINACIONALES

4.1. Introducción

Después de introducir y trabajar con el **Algebra de Boole**, vamos a volver a los **circuitos digitales**. Recordemos que son circuitos electrónicos que trabajan con **números**, y que con la tecnología con la que están realizados, estos números están representados en **binario**. En la figura 4.1 se muestra el esquema general de un circuito digital, que tiene m bits de entrada y n bits de salida.

Si tomamos un circuito genérico y miramos en su interior, podemos ver que está constituido por otros circuitos más simples, interconecados entre sí. En la figura 4.2 hay un ejemplo de un circuito con 4 bits de entrada y 3 de salida, constituido por otros dos circuitos más simples e interconectados entre ellos.

Estos subcircuitos se pueden clasificar en dos tipos:

- **Circuitos combinacionales**
- **Circuitos secuenciales**

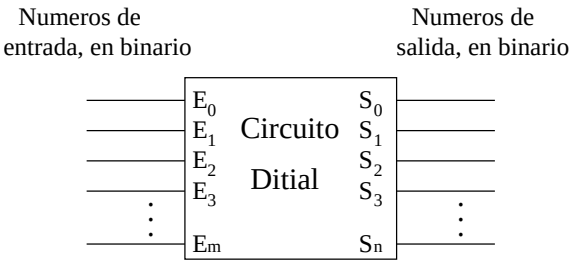


Figura 4.1: Un circuito digital, con m bits de entrada y n de salida

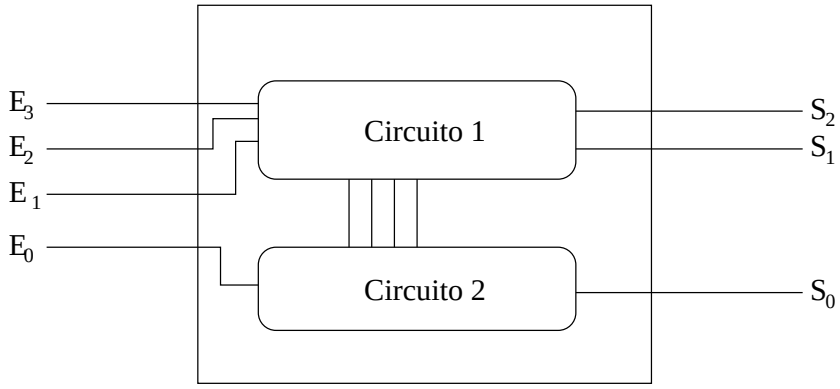


Figura 4.2: Un circuito digital constituido por otros dos circuitos interconectados

Así, podemos decir que todo circuito digital genérico tendrá **una parte combinacional** y otra **parte secuencial**. En este capítulo nos centraremos en los **circuitos combinacionales**, que no tienen parte secuencial. **Estos circuitos se caracterizan porque NO almacenan información**. Las salidas están relacionadas con las entradas a través de una **función booleana**, como las vistas en el capítulo 3. Como veremos más adelante, los circuitos secuenciales son capaces de "recordar" números que han recibido anteriormente.

En un circuito combinacional, las salidas dependen directamente del valor de las entradas, y no pueden por tanto almacenar ningún tipo de información, sólo realizan transformaciones en las entradas. Estos circuitos quedan caracterizados mediante funciones booleanas.

Cada bit de salida de un circuito combinacional, se obtiene mediante una función booleana aplicado a las variables de entrada. Así, si un circuito tiene n salidas, necesitaremos n funciones booleanas para caracterizarlo.

En la figura 4.3 vemos un circuito combinacional que tiene 3 entradas: A,B y C, y dos salidas F, G, que son dos funciones booleanas que dependen de las variables de entrada: F(A,B,C) y G(A,B,C). Por ejemplo, estas funciones podrían tener una pinta así:

$$F = A \cdot B + C$$

$$G = \overline{A \cdot B \cdot C}$$

En este capítulo estudiaremos las **puertas lógicas**, que son los elementos que usamos para

Figura 4.3: Un circuito combinacional de 3 entradas y 2 salidas

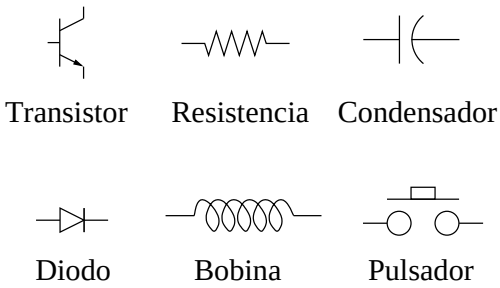


Figura 4.4: Algunos símbolos empleados en la electrónica analógica

construir estos circuitos, y cómo las *funciones booleanas* las podemos realizar mediante puertas lógicas, lo que se denomina **implementación de funciones booleanas**.

4.2. Puertas lógicas

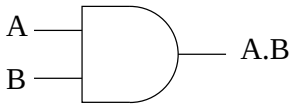
En todas las ingenierías se utilizan *planos* que describen los diseños. En ellos aparecen dibujos, letras y símbolos. Mediante estos planos o esquemas, el Ingeniero representa el diseño que tiene en la cabeza y que quiere construir.

En *electrónica analógica* se utilizan distintos símbolos para representar los diferentes componentes: Resistencias, condensadores, diodos, transistores... Algunos de estos símbolos se pueden ver en la figura 4.4.

En electrónica digital se utilizan otros símbolos, los de las puertas lógicas, para representar las manipulaciones con los bits.

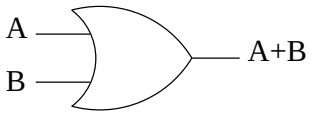
4.2.1. Puertas básicas

- **Puerta AND**



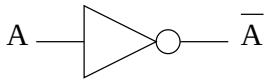
Esta puerta implementa la operación ·del Algebra de Boole. La que se muestra en esta figura tiene dos entradas, sin embargo puede tener más. Lo mismo ocurre con el resto de puertas lógicas que veremos a continuación.

- **Puerta OR**



Implementa la operación + del Algebra de Boole. Puede tener también mas de dos entradas.

- **Puerta NOT** (Inversor)

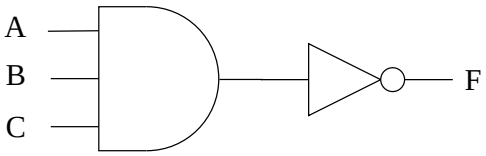


Tiene sólo una entrada y realiza la operación de negación lógica. Esta puerta se conoce normalmente con el nombre de **inversor**.

Sólo con estos tres tipos de puertas se pueden implementar cualquier función booleana.

Ejemplo:

Analizar el siguiente circuito y obtener la expresión booleana de la salida:

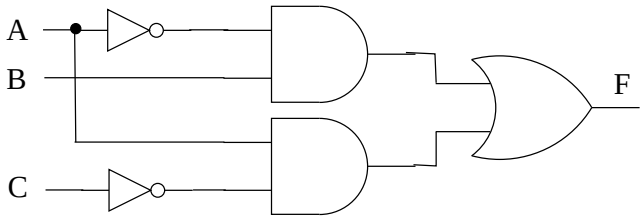


El circuito está constituido por dos puertas, una **AND de tres entradas** y **un inversor**. A la salida de la puerta AND se tiene el producto de las tres variables de entrada $A \cdot B \cdot C$ y al atravesar el inversor se obtiene la expresión final de F, que es:

$$F = \overline{A \cdot B \cdot C}$$

Ejemplo:

Obtener la expresión booleana de salida del siguiente circuito:



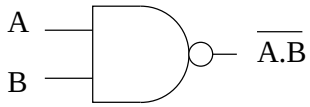
El circuito está constituido por dos puertas AND, dos inversores y una puerta OR. La expresión de F es:

$$F = \overline{A} \cdot B + A \cdot \overline{C}$$

4.2.2. Otras puertas

Con las puertas básicas podemos implementar cualquier función booleana. Sin embargo existen otras puertas que se utilizan mucho en electrónica digital.

- **Puerta NAND**



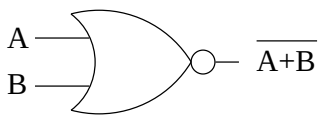
El nombre viene de la abreviación de NOT-AND, y la operación que realiza es la negación de un producto. Aplicando las leyes de DeMorgan vemos que la expresión a su salida es:

$$F = \overline{A \cdot B} = \overline{A} + \overline{B}$$

Esta puerta también puede tener más de dos entradas.

Las puertas NAND tienen una característica muy importante y es que sólo con ellas se puede implementar cualquier función booleana. Sólo hay que aplicar las propiedades del Algebra de Boole a cualquier expresión booleana para dejarla de forma que sólo existan este tipo de operaciones, como veremos en el apartado 4.3.3

- **Puerta NOR**

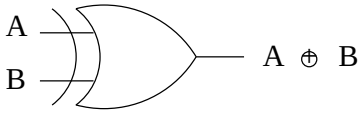


Es una puerta OR negada (NOT-OR). Aplicando las leyes de DeMorgan:

$$F = \overline{A + B} = \overline{A} \cdot \overline{B}$$

Lo mismo que con las puertas NAND, con las puertas NOR se puede implementar cualquier función booleana (ver apartado 4.3.4)

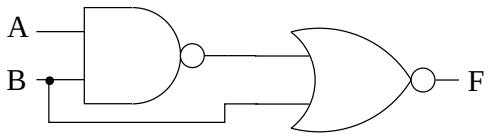
- **Puerta XOR**



Es la puerta que implementa la operación $\oplus$, definida en el apartado 3.8

Ejemplo:

Analizar el siguiente circuito y obtener la expresión booleana de la salida:



A la salida de la puerta NAND tenemos la expresión: $\overline{A \cdot B}$, que se introduce en una de las entradas de la puerta NOR, y por la otra B. El resultado es:

$$F = \overline{\overline{A \cdot B} + B}$$

y aplicando las leyes de DeMorgan nos queda:

$$F = \overline{\overline{A \cdot B}} \cdot \overline{B} = A \cdot B \cdot \overline{B} = A \cdot 0 = 0$$

Es decir, que es un circuito nulo. Con independencia de lo que se introduzca por las entradas, a su salida siempre se obtendrá '0'.

Ejercicios

Hacer el ejercicio 1.

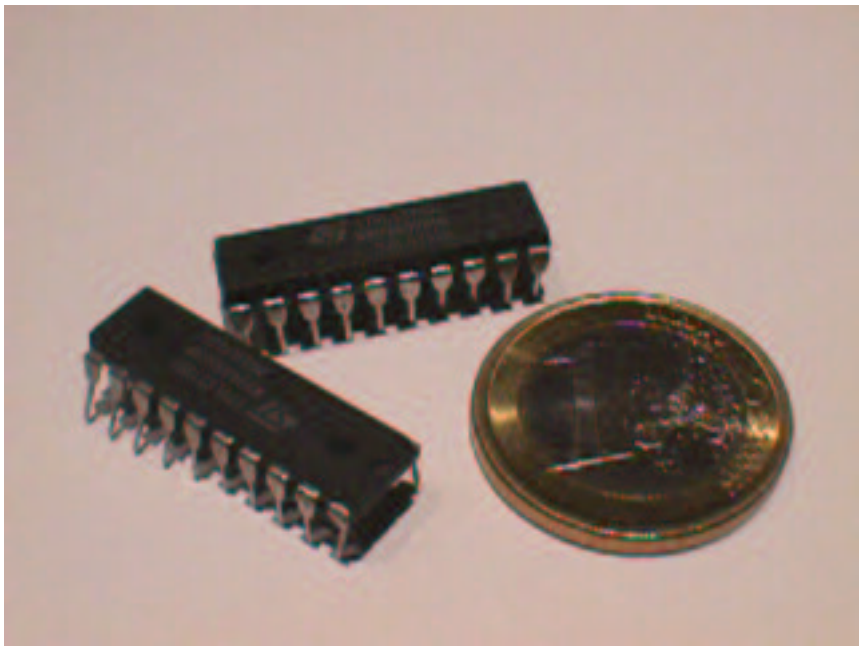


Figura 4.5: Dos circuitos integrados, junto a una moneda de 1 euro

4.2.3. Circuitos integrados

¿Y si ahora queremos construir un circuito? ¿Cómo lo implementamos físicamente? Las **puertas lógicas** se encuentra encapsuladas dentro de **circuitos integrados** o también conocidos como **chips**. En la figura 4.5 se muestra una foto de dos de ellos, junto a una moneda de 1 euro para apreciar su tamaño. Más coloquialmente, entre los alumnos, reciben el nombre de “cucarachas”, porque son negros y tienen patas.

Hay una familia de circuitos integrados, **74XX**, que está estandarizada de manera que se ha definido la información que entra o sale por cada una de las patas. Así pueden existir multitud de fabricantes, pero todos respectando el mismo estándar. En la figura 4.6 se muestra un esquema del integrado 7402, que contiene en su interior 4 puertas NOR de dos entradas.

Por las patas denominadas VCC y GND se introduce la alimentación del chip, que normalmente será de 5v, aunque esto depende de la tecnología empleada. Por el resto de patas entra o sale información binaria codificada según la tecnología empleda. Por ejemplo se puede asociar 5v al dígito ’1’ y 0v al dígito ’0’.

A la hora de **fabricar un diseño**, estos chips se insertan en una placa y se interconectan las patas con el resto de chips o partes de nuestro circuito. La interconexión se realiza por medio de cables. Cuando se realiza una **placa profesional**, las interconexiones entre los chips son **pistas de cobre** en la superficie de la placa. Estas placas reciben el nombre de **placas de circuito**

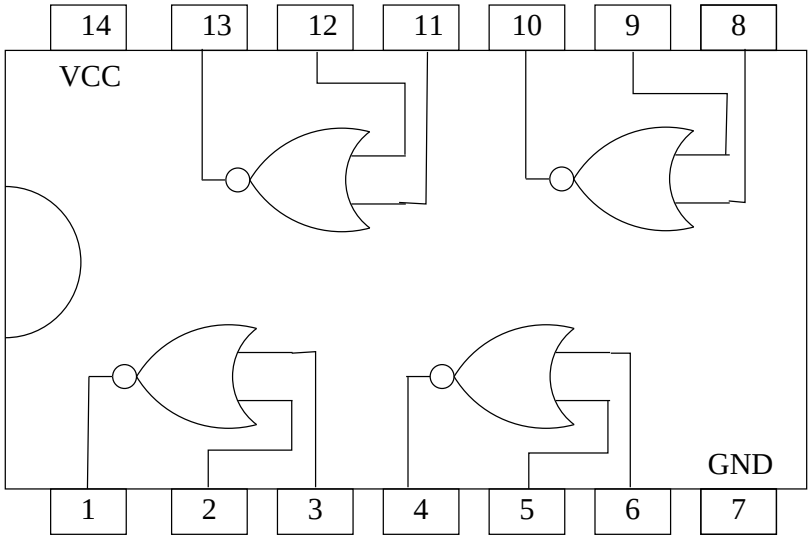


Figura 4.6: Esquema del integrado 7402

impreso, o por sus siglas en inglés **PCB** (printed circuito Board). En la figura 4.7 se muestra la parte inferior de una de estas placas. Por los agujeros se introducen las patas de los componentes y luego se sueldan. Los distintos agujeros están interconectados por pistas de cobre. Además existe una capa de un barniz verde para que las pistas no estén "al aire" y se puedan producir cortocircuitos.

4.2.4. Otras tecnologías

La electrónica ha avanzado muchísimo y en los chips en los que antes sólo se podían integrar una pocas puertas lógicas, ahora se pueden integrar muchísimas más. De esta manera, los chips tradicionalmente se han clasificado según el número de puertas que pueden integrar. Así tenemos la siguiente **clasificación de chips**:

- **SSI** (Small Scale Integration). Chips con menos de 12 puertas
- **MSI** (Medium Scale Integration). Entre 12 y 100 puertas.
- **LSI** (Large Scale Integration). Entre 100 y 10.000 puertas.
- **VLSI** (Very Large Scale Integration). Más de 10.000 puertas

Los VLSI se corresponden con los **microprocesadores** y los **microcontroladores**. Muchos diseños que antes se realizaban sólo con electrónica digital, ahora es más sencillo y barato hacerlos

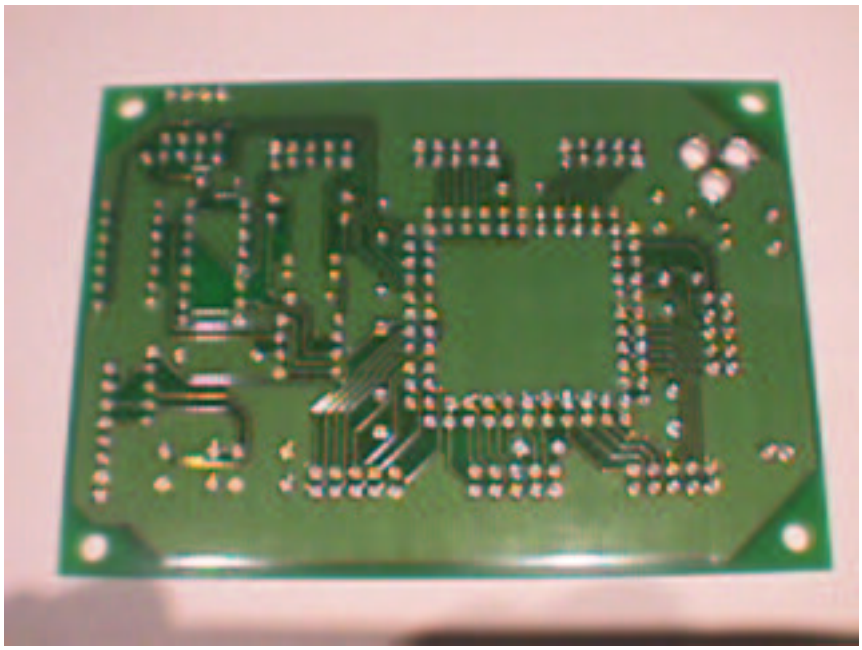


Figura 4.7: Una placa de circuito impreso (PCB) vista desde abajo

con un microprocesador o microcontrolador y **programarlos**. Es decir, hacer **software** en vez de **hardware**.

Sin embargo, existen otras manera de implementar circuitos digitales sin utilizar los chips tradicionales, es decir, sin tener que recurrir a los chips de la familia 74XX. Esta nueva forma de diseñar se denomina **lógica programable**. Existen unos **circuitos integrados genéricos** (PALs,GALs, CPLDs,FPGAS), que contienen en su interior muchas puertas lógicas y otros componentes. El diseñador especifica los circuitos digitales que quiere diseñar utilizando un **lenguaje de descripción hardware** (Como por ejemplo el VHDL). Un herramienta software, conocida como **sintetizador**, convierte esta descripción en un formato que indica cómo se deben interconectar los diferentes elementos de este chip genérico. El chip "se configura" (es decir, realiza conexiones entre sus elementos internos) según se indica en el fichero sintetizado, de manera que **¡¡¡¡nuestra descripción del hardware se ha convertido en un circuito que hace lo que hemos indicado!!!!**

¡¡¡Con esta técnica se pueden diseñar desde circuitos simples hasta microprocesadores!!! El hardware está siguiendo la misma tendencia que el software. Los diseñadores de ahora utilizan sus propios "lenguajes de programación" para especificar el hardware que están diseñando.

En esta asignatura se intenta **dar una visión lo más independiente posible de la tecnología**. De manera que bien se diseñe con puertas lógicas, o bien se utilice un lenguaje de descripción

hardware, los conocimientos aquí adquiridos sirvan para ambos casos.

4.3. Diseño de circuitos combinacionales

4.3.1. El proceso de diseño

En **Ingeniería** se entiende por **diseñar** el proceso por el cual se obtiene el objeto pedido a partir de unas **especificaciones iniciales**. Cuando diseñamos circuitos combinaciones, estamos haciendo lo mismo. Partimos de unas especificaciones iniciales y obtenemos un **esquema**, o **plano**, que indica qué puertas básicas u otros elementos hay que utilizar así como la interconexión que hay entre ellos.

Los pasos que seguiremos para el diseño son los siguientes:

1. **Estudio de las especificaciones iniciales**, para entender realmente **qué** es lo que hay que diseñar. Este punto puede parecer una trivialidad, sobre todo en el entorno académico donde las especificaciones son muy claras. Sin embargo, en la realidad, es muy difícil llegar a comprender o entender qué es lo que hay que diseñar.

2. **Obtención de las tablas de verdad y expresiones booleanas necesarias**. En el entorno académico este suele ser el punto de partida. Nos describen qué función es la que se quiere implementer y lo hacemos.

3. **Simplificación de las funciones booleanas**. ¡¡¡Este punto es importantísimo!!! No basta con implementar una función y ya está. ¡¡Somos ingenieros!!. **Hay que implementar la mejor función**, de manera que obtengamos el mejor diseño posible, reduciendo el número de puertas lógicas empleadas, el número de circuitos integrados o minimizando el retraso entre la entrada y la salida.

4. **Implementación de las funciones booleanas utilizando puertas lógicas**. Aquí podemos tener restricciones, como veremos. Puede ser que por especificaciones del diseño sólo se dispongan de puertas tipo NAND. O puede ser que sólo podamos utilizar puertas lógicas con el mínimo número de entradas. En ese caso habrá que tomar la función más simplificada y modificarla para adaptarla a este tipo de puertas. El resultado de esto es la obtención de un esquema o plano del circuito.

5. **Construcción**. El último paso es llevar ese plano o circuito a la realidad, construyendo

físicamente el diseño. Esto se estudia en el laboratorio de esta asignatura, utilizando tecnología TTL.

En este apartado veremos el punto 4, es decir, veremos cómo a partir de una función (que ya está simplificada) podemos obtener el circuito correspondiente, o cómo la podemos modificar para utilizar un tipo determinado de puertas lógicas. **Esto se denomina implementar una función**.

4.3.2. Implementación de funciones con cualquier tipo de puertas

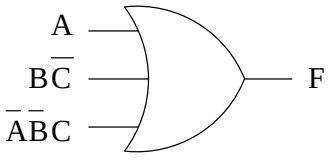
El proceso es muy sencillo. Sólo hay que tomar la función que queremos implementar e ir sustituyendo las operaciones del Algebra de Boole por sus correspondientes puertas lógicas. Y como siempre, lo mejor es ver un ejemplo.

Ejemplo 1:

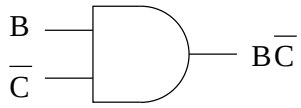
Implementar la siguiente función, utilizando cualquier tipo de puertas lógicas:

$$F = A + B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C$$

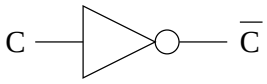
Se trata de implementar un circuito que tiene tres bits de entrada: A, B y C y como salida se quiere obtener la función F indicada. Se puede realizar de muchas formas, pero vamos a ir poco a poco. Primero nos fijamos que no tenemos ninguna restricción. Es decir, en el enunciado nos permiten utilizar cualquier tipo de puerta lógica, y con cualquier número de entradas. Tampoco vamos a simplificar la función, porque lo que queremos es ver cómo implementarla, aunque ya hemos visto que siempre hay que simplificar!!! (y de hecho, esta función se puede simplificar más, ¿como?, se deja como ejercicio). Vemos que en la función hay tres términos que van sumados: A, $B \cdot \overline{C}$, y $\overline{A} \cdot \overline{B} \cdot C$. La puerta lógica que representa la suma es la OR, por lo que podemos escribir:



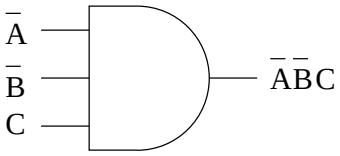
Ahora el problema es más sencillo. Hay que obtener esos tres términos independientemente. Uno ya lo tenemos, que es A (es directamente una de las entradas). El término $B \cdot \overline{C}$ es el producto de B y $\overline{C}$, y lo podemos obtener con una puerta AND así:



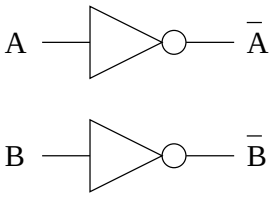
El término $\overline{C}$ lo obtenemos directamente a partir de un inversor:



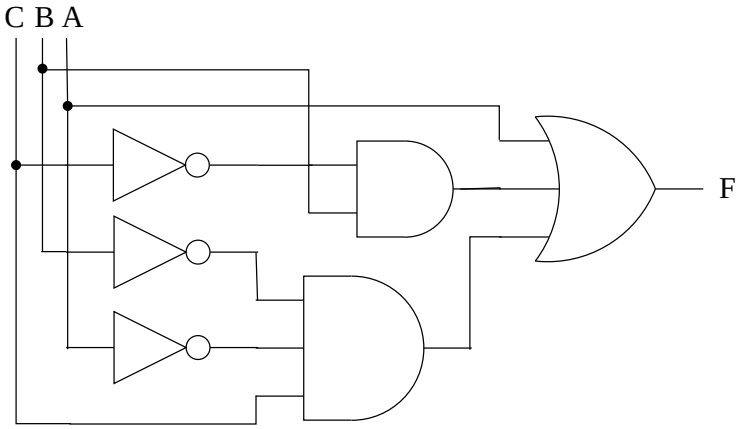
Para obtener el término $\overline{A} \cdot \overline{B} \cdot C$, que es el último que nos falta, nos fijamos que es un producto de tres elementos, por lo que usaremos una puerta AND de tres entradas:



y finalmente para obtener $\overline{A}$ y $\overline{B}$ usamos un par de inversores:



y ahora unimos todas las pierzas para obtener el circuito final:



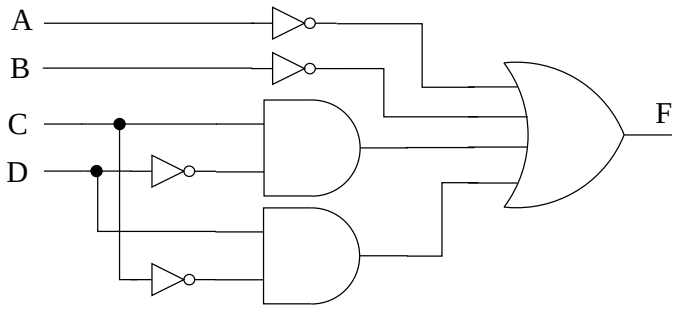
Ejemplo 2:

Implementar la siguiente función, utilizando el menor número posible de puertas lógicas de cualquier tipo. La función está simplificada al máximo.

$$F = \overline{A} + \overline{B} + C \cdot \overline{D} + \overline{C} \cdot D$$

En este caso nos dicen que la función está simplificada al máximo, por lo que no hay que hacer. ¡¡¡Pero es una pregunta que siempre nos tendremos que hacer!! ¿Está simplificada al máximo?. También nos introducen una restricción: usar el menor número posible de puertas lógicas.

Lo primero que se nos puede ocurrir es utilizar el método del ejemplo anterior, sustituyendo las operaciones del Algebra de Boole por puertas lógicas. El circuito que obtenemos es el siguiente:



Hemos utilizo las siguientes puertas lógicas:

- 4 inversores
- 2 puertas AND de dos entradas
- 1 puerta OR de cuatro entradas

La única restricción que nos han impuesto es utilizar el menor número posible de puertas lógicas... ¿Podemos implementar este circuito con menos puertas?. Echemos un vistazo la función F. Teniendo en cuenta que existen otras puertas, como las NAND, XOR, etc... vamos a realizar las siguientes operaciones:

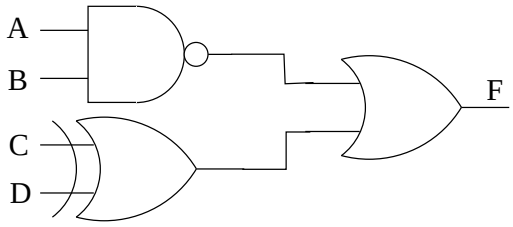
$$\overline{A} + \overline{B} = \overline{A \cdot B}$$

$$C \cdot \overline{D} + \overline{C} \cdot D = C \oplus D$$

La expresión de F que nos queda es la siguiente:

$$F = \overline{A \cdot B} + C \oplus D$$

y si ahora implementamos el circuito:



¡¡Sólo hemos utilizado 3 puertas!!. Una puerta NAND, una XOR y una OR, todas de dos entradas.

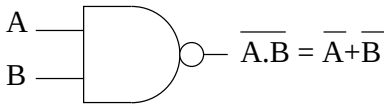
Ejercicios:

Hacer el ejercicio 2

4.3.3. Implementación de funciones con puertas NAND

Sólo con las puertas NAND es posible implementar cualquier función boolena. Para ello habrá que hacer transformaciones en la función original para obtener otra función equivalente pero que se pueda obtener sólo con puertas NAND. Para ver cómo podemos hacer eso, implementaremos las puertas NOT, AND, OR y XOR usando sólo puertas NAND.

Para refrescar ideas, a continuación se muestra una puerta NAND de dos entradas y las formas de expresar el resultado:

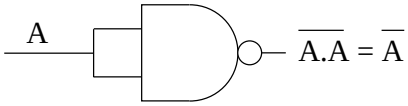


Implementación de una puerta NOT

Si introducimos la misma variable booleana por las dos entradas de una NAND obtendremos lo siguiente:

$$\overline{A \cdot A} = \overline{A}$$

Gráficamente:



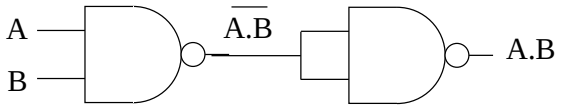
Tenemos un circuito por el que si introducimos una variable A, obtenemos a la salida su complementario $\overline{A}$, es decir, se comporta exactamente igual que un inversor.

Implementación de una puerta AND

Tenemos que diseñar un circuito con puertas NAND que implemente la función $F = A \cdot B$. Lo que haremos será aplicar propiedades del Algebra de Boole a esta función hasta dejarla de forma que la podamos implementar directamente con puertas NAND. Podemos hacer lo siguiente:

$$F = A \cdot B = \overline{\overline{A.B}}$$

La expresión $\overline{A \cdot B}$ se implementa con una puerta NAND y la expresión $\overline{\overline{A \cdot B}}$será por tanto la negación de la NAND. Como ya sabemos como negar utilizando una puerta NAND, el circuito resultante es:

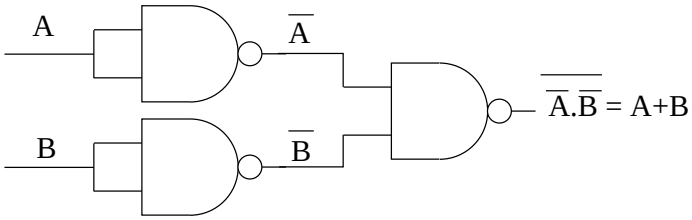


Implementación de una puerta OR

La función que queremos implementar con puertas NAND es: $F = A + B$. Aplicando propiedades del Algebra de Boole, esta expresión la convertimos en la siguiente:

$$F = A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$$

que es el negado de un producto de dos términos, es decir, es una puerta NAND aplicada a $\overline{A}$ y $\overline{B}$:

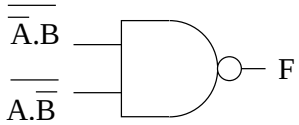


Implementación de una puerta XOR

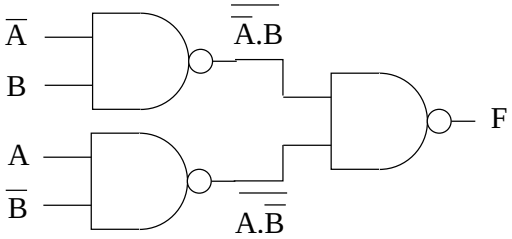
La función a implementar con puertas NAND es: $F = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$. Podemos modificarla de la siguiente manera:

$$F = \overline{A} \cdot B + A \cdot \overline{B} = \overline{\overline{\overline{A} \cdot B}} + \overline{\overline{A \cdot \overline{B}}} = \overline{\left(\overline{\overline{A} \cdot B}\right)} + \overline{\left(\overline{A \cdot \overline{B}}\right)}$$

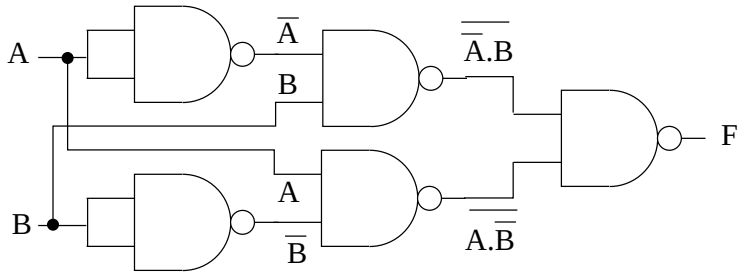
No nos dejemos asustar por aparente complejidad de esta expresión. Fijémonos en que la expresión es la suma de dos términos negados, es decir, que tiene la forma de: $\overline{Algo} + \overline{Algo}$. ¡¡Y esto es una puerta NAND!!, que lo podemos poner de la siguienet manera:



El término $\overline{\overline{A} \cdot B}$tiene también la forma de una puerta NAND, puesto que es del tipo $\overline{Algo \cdot Algo}$. Y lo mismo le ocurre al término $\overline{A \cdot \overline{B}}$. El circuito nos queda así:



Y finalmente hay que obtener $\overline{A}$ y $\overline{B}$ utilizando inversores con puertas NAND:



Ya tenemos implementada la función XOR sólo con puertasn NAND.

Ejemplo 1:

Implementar la siguiente función utilizando únicamente puertas NAND. La función está simplificada al máximo:

$$F = \overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C}$$

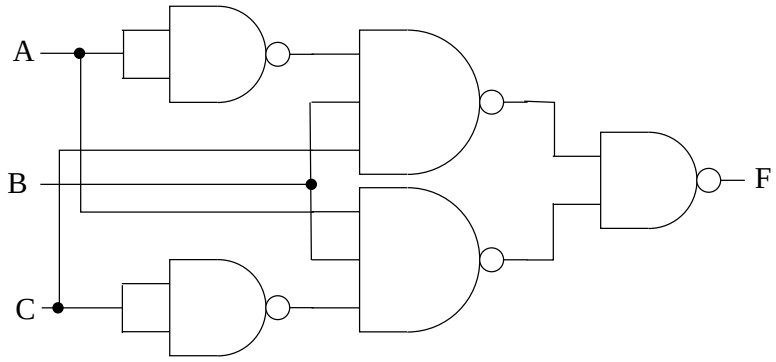
Tendremos que aplicar la propiedades del Algebra de Boole para dejar esta expresión de forma que la podamos implementar con puertas NAND. Como el enunciado no nos pone ninguna restricción, podremos usar puertas NAND con el número de entradas que queramos. Una puerta NAND de tres entradas puede realizar las siguientes operaciones:

$$\overline{Algo \cdot Algo \cdot Algo} = \overline{Algo} + \overline{Algo} + \overline{Algo}$$

Si aplicamos una doble negación a F y luego aplicamos sucesivamente las leyes de DeMorgan (o el teorema de Shannon):

$$F = \overline{\overline{\overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C}}} = \overline{(A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + C)}$$

Esta función es inmediata implementarla con puertas NAND:



Ejemplo 2:

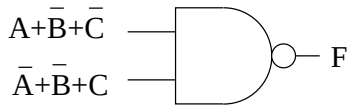
Implementar la siguiente función utilizando sólo puertas NAND de 2 entradas:

$$F = \overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C}$$

Es la misma función que la del apartado anterior, sin embargo, ahora tenemos la restricción de que sólo podemos usar puertas NAND de dos entradas. Si hacemos la misma transformación que antes, obtenemos:

$$F = \overline{\overline{\overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C}}} = \overline{(A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + C)}$$

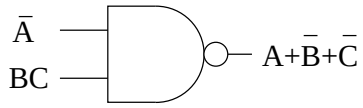
que tiene la forma $\overline{Algo \cdot Algo}$ y que se implementa fácilmente con una NAND de dos entradas:



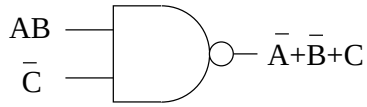
El problema ahora es cómo implementar los términos $A + \overline{B} + \overline{C}$ y $\overline{A} + \overline{B} + C$. Vamos con el primero de ellos. Se puede escribir también de la siguiente forma (aplicando el “truco” de la doble negación):

$$A + \overline{B} + \overline{C} = \overline{\overline{A} \cdot (B \cdot C)}$$

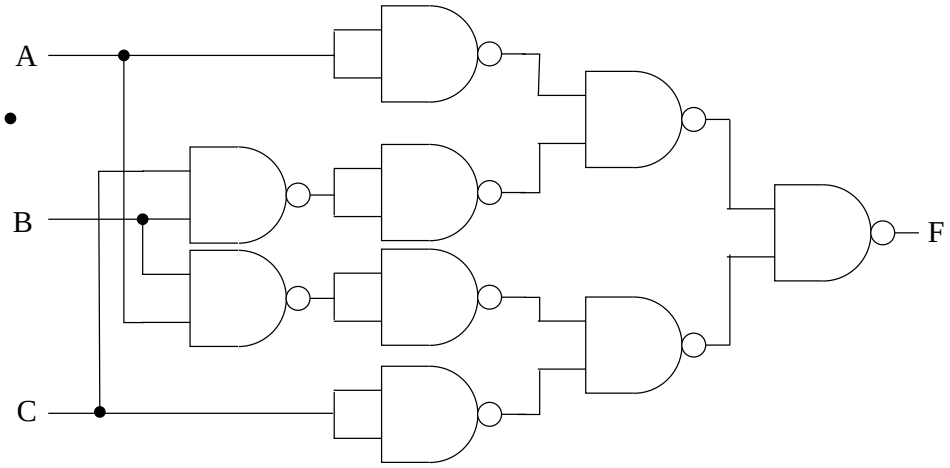
que se implementa de la siguiente forma:



El otro término lo podemos implementar de forma similar:



y ahora juntando todas las piezas e implementando lo que falta:

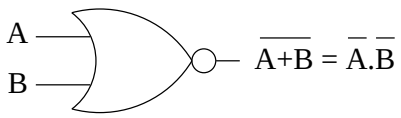


Ejercicios:

Hacer el ejercicio x

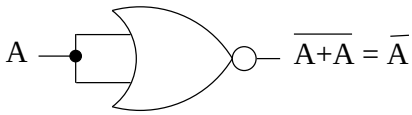
4.3.4. Implementación de funciones con puertas NOR

Lo mismo que con las puertas NAND, con las puertas NOR se puede implementar cualquier función booleana. Vamos a ver cómo se pueden implementar el resto de puertas lógicas. Recordemos que las expresiones a las salidas de las puertas NOR son:



Implementación de una puerta NOT

Se hace de la misma manera que con las puertas NAND. Si introducimos la misma variable por las dos entradas, obtenemos la variable negada:

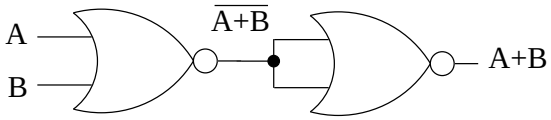


Implementación de una puerta OR

La función a implementar es: $F = A + B$. Esta expresión la podemos poner de la siguente manera:

$$F = A + B = \overline{\left(\overline{A + B}\right)}$$

es decir, que podemos utilizar una puerta NOR y luego un inversor, que ya sabemos cómo implementarlo con puertas NOR. Lo que nos queda es:

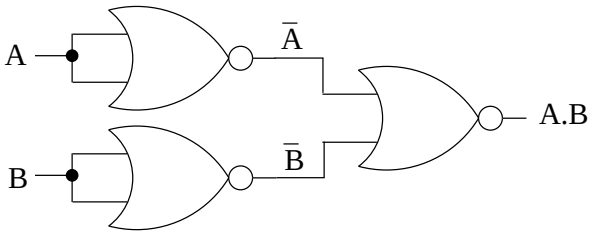


Implementación de una puerta AND

La función a implementar es: $F = A \cdot B$. Podemos realizar las siguientes modificaciones para que pueda ser implementada con puertas NOR:

$$F = A \cdot B = \overline{\overline{A}} \cdot \overline{\overline{B}} = \overline{\left(\overline{A}\right)} \cdot \overline{\left(\overline{B}\right)}$$

Y el circuito quedaría así:

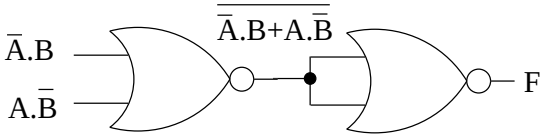


Implementación de una puerta XOR

La función a implementar es: $F = \overline{A} \cdot B + A \cdot \overline{B}$. Haciendo las siguientes modificaciones:

$$F = \overline{A} \cdot B + A \cdot \overline{B} = \overline{\overline{\overline{A} \cdot B + A \cdot \overline{B}}} = \overline{\left(\overline{\left(\overline{A} \cdot B\right) + \left(A \cdot \overline{B}\right)}\right)}$$

y de la misma manera que hemos hecho con las puertas NAND, vamos a ir implementando esta función poco a poco. Primero vemos que hay una puerta NOR cuyas entradas son $\overline{A} \cdot B$ y $A \cdot \overline{B}$, y que está negada:

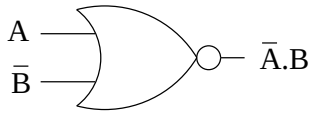


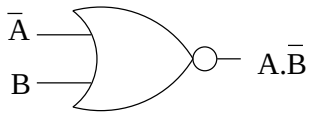
A continuación implementamos $\overline{A} \cdot B$ y $A \cdot \overline{B}$, teniendo en cuanta que los podemos reescribir de esta forma:

$$\overline{A} \cdot B = \overline{(A)} \cdot \overline{\left(\overline{B}\right)}$$

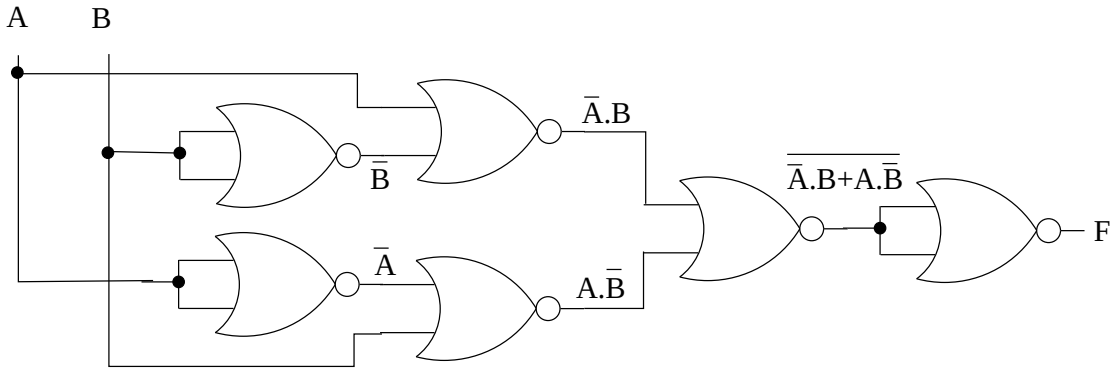
$$A \cdot \overline{B} = \overline{\left(\overline{A}\right)} \cdot \overline{(B)}$$

Gráficamente:





Uniendo “todas las piezas”, el circuito final que nos queda es:



Hemos implementado la puerta XOR sólo con puertas NOR.

Ejercicios:

Hacer el ejercicio x

4.4. Aplicación: Diseño de un controlador para un robot seguidor de línea

4.4.1. Introducción

En este apartado **diseñaremos un circuito digital que gobierne el comportamiento de un robot seguidor de línea**. El objetivo es que el alumno vea cómo todo lo aprendido hasta ahora se puede aplicar, y obtener también algo de intuición sobre el tipo de circuitos digitales que se pueden diseñar.

Este apartado es opcional. El lector no interesado puede saltar directamente al apartado 4.6. Sin embargo los alumnos inquietos pueden utilizarlo de base para introducirse en el mundo de la robótica y de la electrónica digital práctica, para ver cómo se puede hacer un proyecto real.

Obviamente no construiremos el robot entero, esto nos llevaría más tiempo :-). Partiremos de un robot ya existente, que tiene una estructura mecánica hecha con piezas de Lego, dos motores, dos sensores para detectar el color negro sobre un fondo plano y la electrónica necesaria para controlar los motores y leer los sensores. Este robot se comercializa bajo el nombre de Tritt. Sin embargo utiliza un microcontrolador 6811 para implementar el “cerebro”. Nosotros diseñaremos nuestro propio cerebro digital, para que el robot siga una línea negra. En la figura 4.8 se muestra el microbot Tritt, junto a un disquete, para hacerse una idea de las dimensiones que tiene.

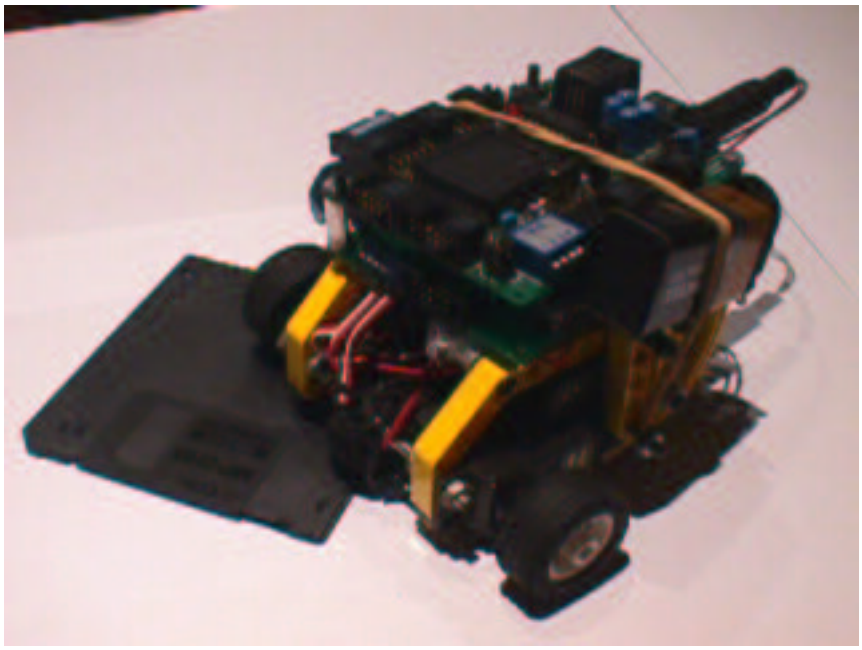


Figura 4.8: El microbot Tritt

En la figura 4.9 se muestra el mismo microbot Tritt pero sin la tarjeta CT6811 que lleva el microcontrolador 6811. En vez de ella diseñaremos nuestro propio “cerebro digital”.

4.4.2. Especificaciones

Las especificaciones son:

- **Objetivo**: Diseñar un circuito digital, capaz gobernar un microbot, haciendo que éste siga una línea negra pintada sobre un fondo blanco**.**
- **Sensores**: El microbot está dotado de dos sensores digitales capacez de diferenciar el color negro del blanco. La salida de estos sensores es ’0’ cuando leen blanco y ’1’ cuando leen negro. Denominaremos a este bit como **C**:

Sensor	**C**
Color Blanco	0
Color Negro	1

- **Motores**: Dos motores de corriente continua que son controlados cada uno mediante dos bits, denominados **S** y **P**, descritos mediante la siguiente tabla de verdad:

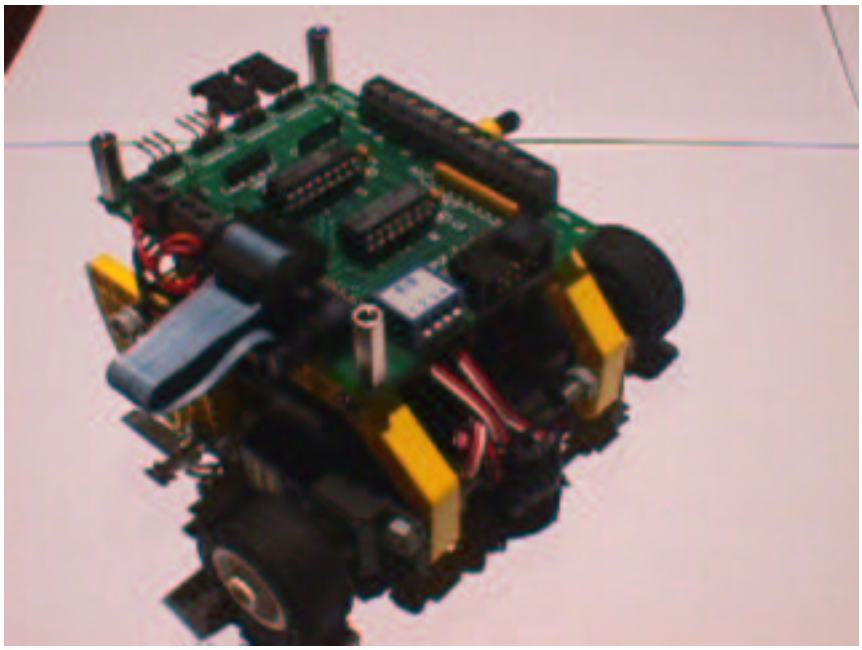
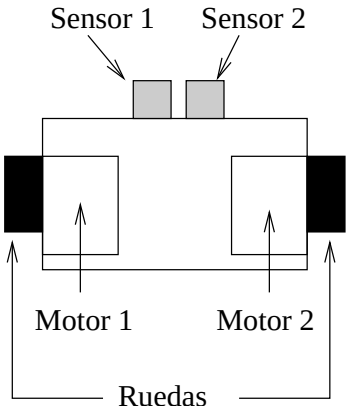


Figura 4.9: Microbot Tritt sin la tarjeta CT6811

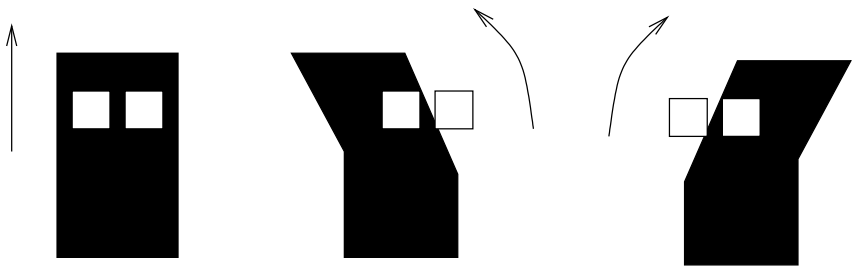
P	S	Motor
0	0	Parado
0	1	Parado
1	0	Giro derecha
1	1	Giro izquierda

El **bit P** es el bit de 'Power'. Indica si el motor está conectado o no. El **bit S** es el del sentido de giro. Según su valor el motor girará a la derecha o a la izquierda (siempre que el motor esté activado, con P=1).

- **El robot**: El esquema del robot es el siguiente (visto desde arriba):

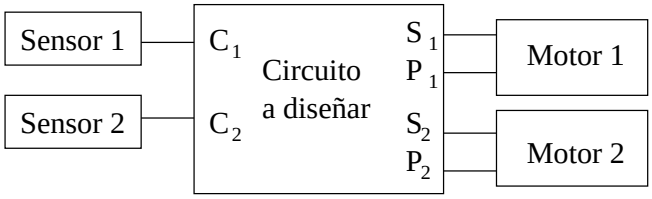


- **Algoritmo**: El algoritmo para seguir la línea negra es muy sencillo. Mientras los dos sensores detecten negro, el robot deberá avanzar. Cuando el sensor de la derecha detecte blanco y el de la izquierda negro, el robot girará a la izquierda y cuando ocurra el caso contrario girará a la derecha. Si ambos sensores leen blanco permanecerá parado. Esto se esquematiza en la siguiente figura:



4.4.3. Diagrama de bloques

Como primera fase del diseño tenemos que entender qué es lo que se nos está pidiendo y determinar el aspecto que tiene el circuito que hay que realizar. El circuito tendrá dos entradas provenientes de los sensores, C_1 y C_2, y cuatro salidas, dos para cada motor: S_1, P_1, S_2 y P_2:



4.4.4. Tabla de verdad

Ahora hay que definir el comportamiento del circuito, utilizando una tabla de verdad. Este comportamiento nos lo da el algoritmo de seguir la línea. La tabla de verdad es la siguiente:

C_1	C_2	S_1	P_1	S_2	P_2
0	0	x	0	x	0
0	1	0	1	1	1
1	0	1	1	0	1
1	1	0	1	0	1

Con una 'x' se han marcado las casillas de la tabla de verdad que es indiferente su valor. Según nos convenga puede valer '0' ó '1'.

4.4.5. Ecuaciones booleanas del circuito

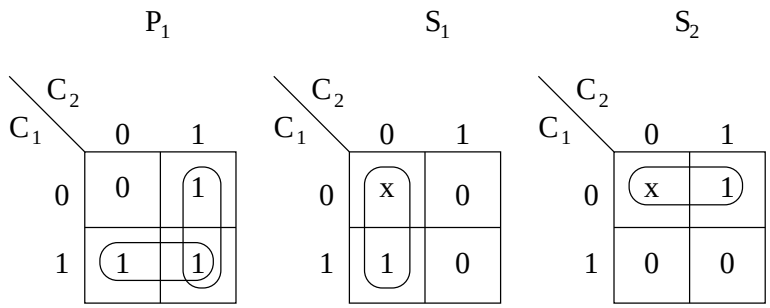
Puesto que el circuito sólo tiene 2 variables de entrada, es inmediato obtener las expresiones de S_1, P_1, S_2 y P_2.

$$S_1 = \overline{C_2}$$

$$S_2 = \overline{C_1}$$

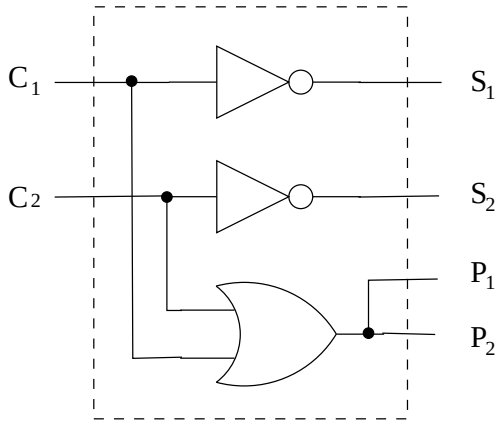
$$P_1 = P_2 = C_1 + C_2$$

También se podría haber hecho Karnaugh:

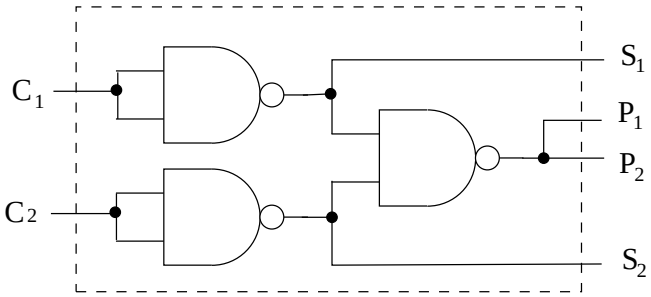


4.4.6. Implementación del circuito

El circuito, implementado con puertas lógicas básicas es el siguiente:



Si lo construimos utilizando puertas TTL, necesitamos dos integrados, uno para los inversores y otro para la puerta OR. Si en vez de ello lo implementamos sólo con puertas NAND, el circuito es el siguiente:



Tiene también 3 puertas, pero ahora sólo es necesario un sólo circuito integrado.

4.5. Análisis de circuitos combinacionales

Por análisis entendemos lo contrario de diseño. Al diseñar partimos de unas especificaciones, obtenemos una tabla de verdad o una función booleana, la simplificamos y la implementamos con puertas lógicas. En el análisis partimos de un circuito y tendremos que obtener bien la tabla de verdad, bien la expresión booleana, lo que nos permitirá analizar si el circuito era el más óptimo o nos permitirá hacer una re-implementación de dicho circuito utilizando otra tecnología.

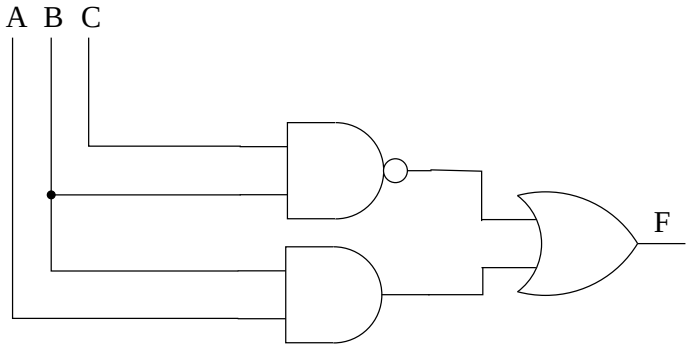
Si el circuito tiene pocas entradas, cuatro o menos, lo mejor es hacer la tabla de verdad. Para realizarla tomaremos puntos intermedios en el circuito, que incluiremos también en la propia

tabla. Iremos rellenando el valor de estos puntos intermedios hasta obtener el valor de la función.

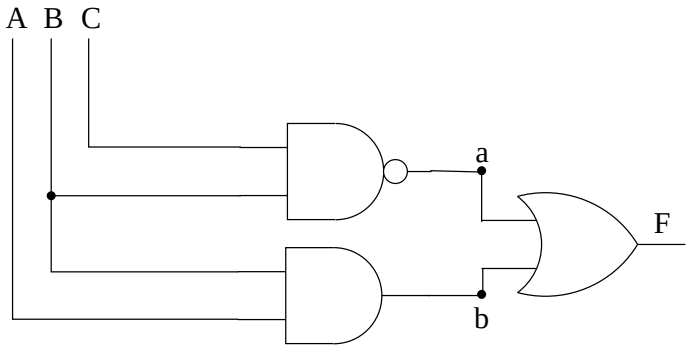
Y como siempre, lo mejor es ver ejemplos.

Ejemplo 1:

Obtener la tabla de verdad del siguiente circuito:



El problema se puede hacer de varias maneras. Y ese suele ser uno de los problemas. ¿Qué camino escojo para obtener la tabla de verdad?. Por un lado podemos obtener la expresión de F, pasando las puertas lógicas a operandos del Algebra de Boole y luego obtener la tabla de verdad. O podemos obtener directamente la tabla de verdad. Sea cual sea el camino elegido, lo primero que haremos será tomar **puntos intermedios:** seleccionamos las salidas de las puertas lógicas y les asignamos una variable boolena:



En este circuito hemos tomado dos puntos intermedios, el **a** y el **b**. Si decidimos obtener F usando el Algebra de Boole, la expresión que obtenemos es:

$$F = a + b = \overline{B \cdot C} + A \cdot B = \overline{B} + \overline{C} + A \cdot B$$

Y ahora la representaríamos en una tabla de verdad. Sin embargo, suele ser más sencillo

obtener la tabla de verdad directamente del diseño y luego aplicar karnaugh para obtener la expresión más simplificada de F, si fuese necesario. En la tabla de verdad dibujaremos nuevas columnas en las que aparecen los puntos intermedios, que nos permitirán ir anotando los cálculos intermedios para obtener F más fácilmente. La tabla de verdad sin rellenar es:

A	B	C	$a = \overline{B \cdot C}$	$b = B \cdot A$	$F = a + b$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Y ahora vamos columna por columna, rellenando la información. Comenzaremos por la columna **a**. Hay que hacer la NAND de B y C. Para no confundirnos, nos dibujamos la tabla NAND para dos variables:

A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

y nos fijamos en que sólo vale '0' cuando ambas variables son 1. Recorremos las filas de B y C buscando el caso en el que B=1 y C=1, y anotamos un '0'. Para el resto de casos a='1'. Nos queda lo siguiente:

A	**B**	**C**	$a = \overline{B \cdot C}$	$b = B \cdot A$	$F = a + b$
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	1		
1	1	1	**0**		

Se ha marcado con “negrita” los dos casos en los que B=1 y C=1. Para el resto de casos “no hemos tenido que pensar”, se puede rellenar de forma directa. Este método nos permite obtener las tablas de verdad de una manera muy rápida y cometiendo muy pocos errores.

Contiemos con la siguiente columna. En este caso hay que rellenar una columna con el producto entre B y A. Nuevamente nos fijamos en la tabla de la operación AND y vemos que el resultado sólo vale ’1’ cuando B=1 y A=1. Para el resto de casos se tendrá ’0’:

A	**B**	**C**	$a = \overline{B \cdot C}$	$b = B \cdot A$	$F = a + b$
0	0	0	1	0	
0	0	1	1	0	
0	1	0	1	0	
0	1	1	**0**	0	
1	0	0	1	0	
1	0	1	1	0	
1	1	0	1	0	
1	1	1	**0**	**1**	

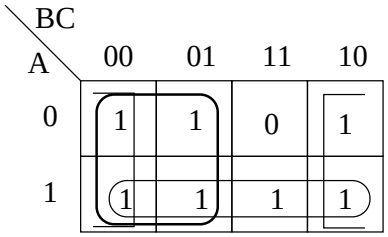
Y por último ya podemos obtener el valor de F, aplicando una operación OR a la columna **a** con la **b**. Por la definición de la operación OR (mirando su tabla), sabemos que sólo vale 0 cuando ambos operandos son ’0’. Buscamos ese caso en la tabla y en el resto de filas ponemos un ’1’. La tabla final es:

A	**B**	**C**	$a = \overline{B \cdot C}$	$b = B \cdot A$	$F = a + b$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	**0**	0	**0**
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	**0**	**1**	1

Aunque no los pide el enunciado del ejercicio, vamos a obtener la expresión más simplificada de F, usando Karnagh, y la vamos a comparar con la expresión F que antes obtuvimos. El diagrama de Karnaugh es muy sencillo de obtener a partir de la tabla de verdad, puesto que sólo un ’0’. Pintamos este ’0’ en su casilla correspondiente (A=0, B=1 y C=1) y el resto de casillas valdrán ’1’:

BC / A	00	01	11	10
0	1	1	0	1
1	1	1	1	1

Podemos hacer los siguientes grupos:



De los que obtenemos la expresión más simplificada de F:

$$F = A + \overline{B} + \overline{C}$$

Vemos que está más simplificada que la expresión inicial que obtuvimos aplicando el Algebra de Boole.

4.6. Resumen

Todo **circuito digital** está constituido en su interior por **circuitos combinacionales** y/o **circuitos secuenciales**. Estos últimos son capaces de almacenar información. En este capítulo hemos trabajado con *circuitos combinaciones*, en los que sus salidas dependen directamente de las entradas, y no son capaces de almacenar información ni recordar cuáles fueron las entradas anteriores.

Para la construcción de los circuitos combinacionales, se emplean las **puertas lógicas**, que permiten realizar electrónicamente las operaciones del **Algebra de Boole**. Las puertas lógicas básicas con **AND**, **OR** y **NOT**, pero también existen otras puertas lógicas que se usan mucho: **NAND**, **NOR** y **XOR**. Cualquier circuito combinacional se puede construir a partir de las puertas básicas, combinándolas adecuadamente. Sin embargo, también es posible implementar circuitos utilizando sólo **puertas NAND**, o sólo **puertas NOR**.

Las **puertas lógicas** se encuentran encapsuladas en un **circuito integrado**. Esto se denomina

tecnología TTL. También es posible utilizar otras tecnologías para la construcción de circuitos digitales, como son los dispositivos lógicos programables o las FPGA's.

El **diseño** de un **circuito combinacional** es sencillo. A partir de unas **especificaciones** se obtiene la **tabla de verdad** de las salidas del circuito, y utilizando el método de simplificación de **Karnaugh** obtendremos la **función más simplificada**. Las funciones así obtenidas se podrán implementar de diversas maneras, entre las que hemos visto, su implementación usando puertas básicas, sólo puertas NAND, o sólo puertas NOR.

Como ejemplo práctico, hemos diseñado un circuito combinacional que actúa de "cerebro" de un Microbot, controlándolo de manera que siga una línea negra sobre un fondo blanco.

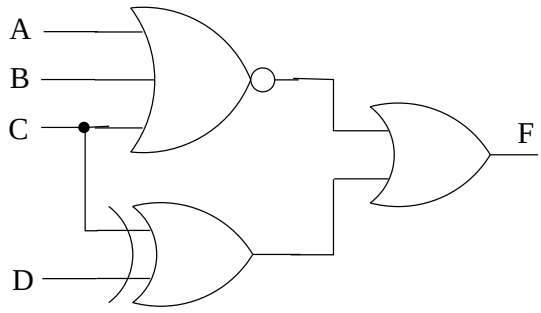
Finalmente hemos visto cómo se **analizan** los circuitos, obteniendo sus tablas de verdad o ecuaciones booleanas a partir de las puertas lógicas.

4.7. Ejercicios

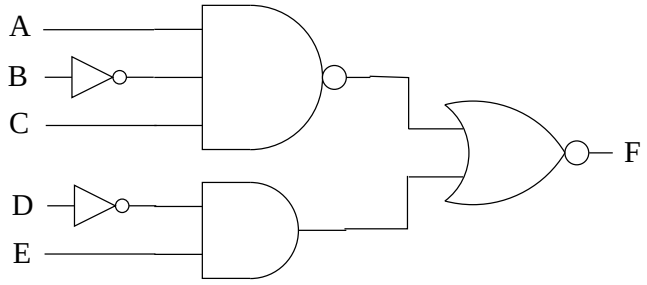
Ejercicio 1:

Obtener las expresiones booleanas de las salidas de los siguientes circuitos (no hay que simplificar ni operar estas expresiones):

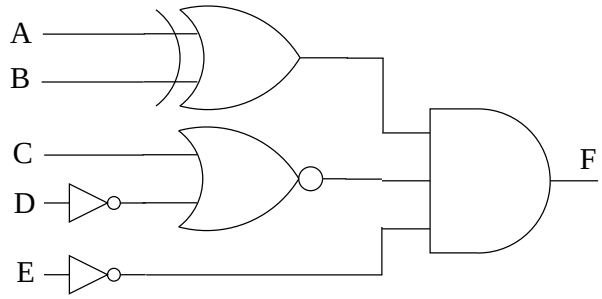
- **Circuito 1**:



- **Circuito 2**:



- **Circuito 3**:



Ejercicio 2:

Implementar las siguientes función, utilizando cualquier tipo de puertas lógicas, sabiendo que todas las funciones están simplificadas al máximo.

1. $F = A \cdot B + \overline{B} \cdot \overline{C}$

Ejercicio 2:

Implementar sólo con puertas NAND

Ejercicio 3:

Implementar sólo con puertas NOR

Ejercicio x:

Dada la función $F = A \cdot B + A \cdot \overline{C}$:

1. Implementar con cualquier tipo de puertas lógicas

2. Implementar sólo con puertas NAND
3. Implementar sólo con puertas NOR
4. Aplicar la propiedad distributiva e implementar con cualquier tipo de puertas lógicas
5. ¿En qué circuito se utilizan el menor número de puertas?

Capítulo 5

CIRCUITOS MSI (1): Multiplexores y demultiplexores

5.1. Introducción

Los **circutios MSI** son los que están constituidos por un número de puertas lógicas comprendidos entre 12 y 100 (ver apartado 4.2.4). En este capítulo veremos una serie de **circuitos combinaciones** que se utilizan mucho en electrónica digital y que son la base para la creación de diseños más complejos. Aunque se pueden diseñar a partir de puertas lógicas, estos circuitos se pueden tratar como "componentes", asignándoles un símbolo, o utilizando una cierta nomenclatura.

Los circuitos que veremos son los siguientes:

- **Multiplexores** y **demultiplexores**
- **Codificadores** y **decodificadores**
- **Comparadores**

Lo más importante es comprender para qué sirven, cómo funcionan y que bits de entrada y salida utilizan. Estos circuitos los podríamos diseñar perfectamente nosotros, puesto que se trata de *circuitos combinacionales* y por tanto podemos aplicar todo lo aprendido en el capítulo 4.

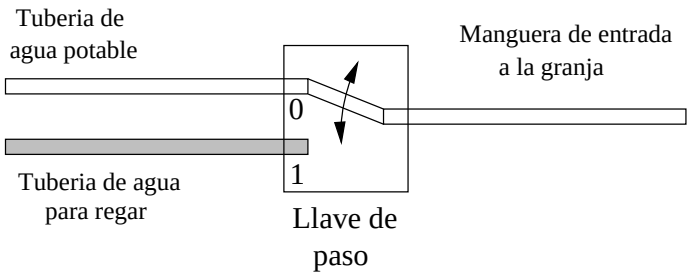


Figura 5.1: Simitud entre un multiplexor y un sistema de agua de una granja

5.2. Multiplexores

5.2.1. Conceptos

Un **Multiplexor** es un **circuito combinacional** al que entran varios canales de datos, **y sólo uno de ellos**, el que hallamos seleccionado, es el que **aparece por la salida**. Es decir, que es un circuito que nos permite **SELECCIONAR** que datos pasan a través de dicho componente.

Vamos a ver un ejemplo NO electrónico. Imaginemos que hay dos tuberías (*canales de datos*) por el que circulan distintos fluidos (*datos*). Una transporta agua para regar y la otra agua potable. Estas tuberías llegan a una granja, en la cual hay una única manguera por la que va a salir el agua (bien potable o bien para regar), según lo que seleccione el granjero posicionando la **llave de paso** en una u otra posición. En la figura 5.1 se muestra un esquema. Las posiciones son la 0 para el agua potable y 1 para el agua de regar.

Moviendo la llave de paso, el granjero puede seleccionar si lo que quiere que salga por la manguera es agua potable, para dar de beber al ganado, o agua para regar los cultivos. Según cómo se posicione esta llave de paso, en la posición 0 ó en la 1, seleccionamos una tubería u otra.

Pero ¿por qué sólo dos tuberías?. Porque es un ejemplo. A la granja podrían llegar 4 tuberías. En este caso el granjero tendría una llave de paso con 4 posiciones, como se muestra en la figura 5.2. Esta llave se podría poner en 4 posiciones distintas para dar paso a la tubería 0, 1, 2 ó 3. Obsérvese que sólo pasa una de las tuberías en cada momento, ¡y sólo una!. Hasta que el granjero no vuelva a cambiar la llave de paso no se seleccionará otra tubería.

Con este ejemplo es muy fácil entender la idea de **multiplexor**. Es como una **llave de paso**, que sólo conecta uno de los canales de datos de entrada con el canal de datos de salida.

Ahora en vez de en tuberías, podemos pensar en canales de datos, y tener un esquema como el que se muestra en la figura 5.3, en la que hay 4 canales de datos, y sólo uno de ellos es seleccionado por el multiplexor para llegar a la salida . En general, en un multiplexor tenemos dos tipos de entradas:

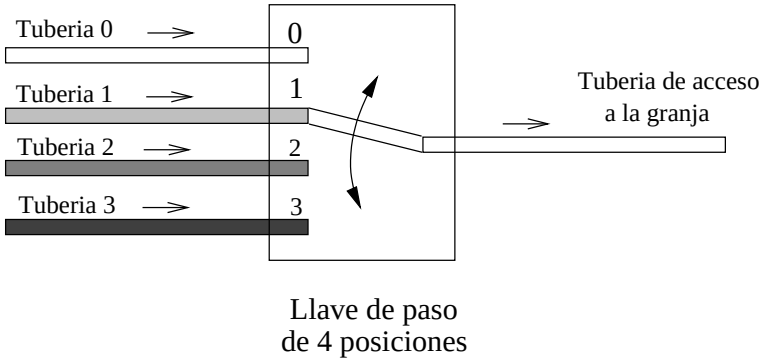


Figura 5.2: Sistema de agua de 4 tuberías

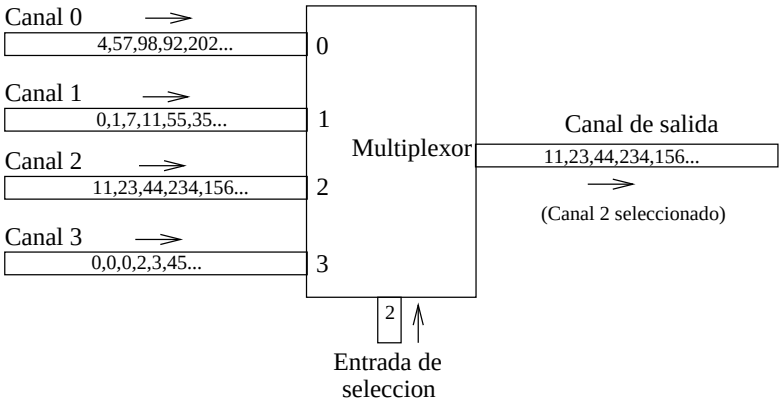


Figura 5.3: Un multiplexor que selecciona entre 4 canales de datos

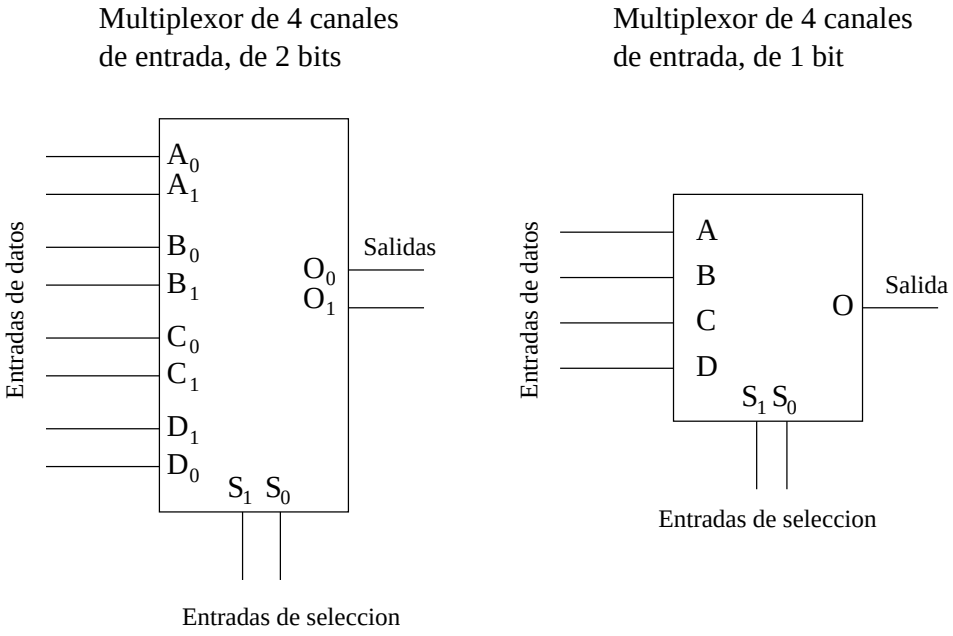


Figura 5.4: Dos multiplexores de 4 canales de entrada

- **Entradas de datos**: (Las tuberías en el ejemplo).
- **Entrada de selección**: Indica cuál de las entradas se ha seleccionado (posición de la llave de paso).

5.2.2. Multiplexores y bits

Hemos visto cómo a un **multiplexor** le llegan **números** por distintas entradas y según el número que le llegue por la **entrada de selección**, lo manda por la salida o no. **¡¡Números!!** Recordemos que los circuitos digitales sólo trabajan con números.

Pero estos números, vimos que siempre vendrán **expresados en binario** y por tanto se podrán expresar mediante **bits**. ¿Cuantos bits? Depende de lo grande que sean los números con los que se quiere trabajar.

En el interior de los microprocesadores es muy normal encontrar multiplexores de 8 bits, que tienen varias entradas de datos de 8 bits. Pero se puede trabajar con multiplexores que tengan 4 bits por cada entrada, o incluso 2, o incluso 1bit. En la figura 5.4 se muestran dos multiplexores que tienen 4 entradas de datos. Por ello la entrada de selección tiene dos bits (para poder seleccionar entre los cuatro canales posibles). Sin embargo, en uno las entradas de datos son de 2 bits y en el otro de 1 bit.

Mirando el número de salidas, podemos conocer el tamaño de los canales de entrada.

Así en los dos multiplexores de la figura 5.4, vemos que el de la izquierda tiene 2 bits de salida, por tanto sus canales de entrada son de 2 bits. El de la derecha tiene 1 bit de salida, por tanto los canales de 1 bit.

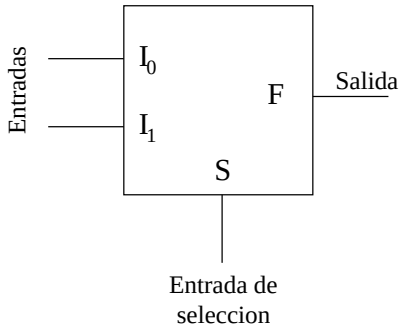
Los multiplexores en lo que principalmente nos centraremos son los que tienen canales de 1 bit. A partir de ellos podremos construir multiplexores mayores, bien con un mayor número de canales de entrada o bien con un mayor número de bits por cada canal.

5.2.3. Multiplexores de 1 bit y sus expresiones booleanas

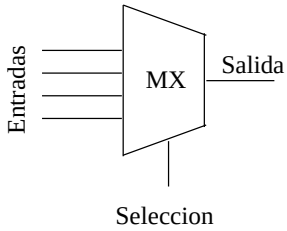
Llamaremos así a los **multiplexores que tienen canales de entrada de 1 bit**, y por tanto **sólo tienen un bit de salida**. Estudiaremos estos multiplexores, comenzando por el más simple de todos, el que sólo tienen una entrada de selección.

Multiplexores con una entrada de selección

El multiplexor más simple es el que sólo tiene una **entrada de selección**, **S**, que permite seleccionar entre dos entradas de datos, según que $S = 0$ ó $S = 1$. Su aspecto es el siguiente:



NOTA: *En esta asignatura representaremos los multiplexores de igual que cualquier otro circuito, mediante una "caja" que tiene unas entradas y unas salidas. No obstante, el símbolo normalmente emlpleado es el siguiente:*



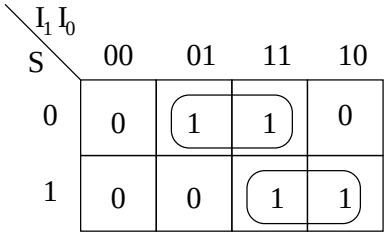
¿Cómo podemos expresar la función de salida F, usando el Algebra de Boole?. Existe una manera muy sencilla y que ya conocemos: hacer la tabla de verdad y obtener la función más simplificada.

Construyamos la tabla de verdad. Lo primero que nos preguntamos es, ¿Cuantas entradas tengo en este circuito?. En total hay tres entradas. Dos son de datos: I_1, I_0 y una es de selección: S. La tabla de verdad tendrá en total $2^3 = 8$ filas. Para construir esta tabla de verdad sólo hay que entender el funcionamiento del multiplexor e ir caso por caso rellenando la tabla. Por ejemplo, ¿qué ocurre si $S = 1$, $I_1 = 0$ y $I_0 = 1$?. Aplicamos la definición de multiplexor. Puesto que $S = 0$, se está seleccionando la entrada de datos 0, es decir, la entrada I_0. Por tanto, lo que entre por la entrada I_1 será ignorado por el multiplexor. Si la entrada seleccionada es la I_0, la salida tendrá su mismo valor. Y puesto que $I_0 = 1,$ entonces $F = 1$. Si hacemos lo mismo para todos los casos, tendremos la siguiente tabla de verdad:

S	I_1	I_0	F
0	0	**0**	**0**
0	0	**1**	**1**
0	1	**0**	**0**
0	1	**1**	**1**
1	**0**	0	**0**
1	**0**	1	**0**
1	**1**	0	**1**
1	**1**	1	**1**

La tabla se ha dividido en dos bloques, uno en el que $S = 0$ y otro en el que $S = 1$. En el primer bloque, se selecciona I_0 que aparecerá en la salida. Se ha puesto en negrita todos los valores de I_0 para que se vea que son los mismos que hay a la salida. En el bloque inferior, lo que se selecciona es I_1 y es lo que se obtiene por la salida.

Apliquemos el **método de Karnaugh** para obtener la expresión **más simplificada de F**. El diagrama que se obtiene es el siguiente: (Se aconseja al lector que lo haga por su propia cuenta, sin mirar los apuntes, así le sirve además para practicar :-)



Obtenemos la siguiente expresión:

$$F = \overline{S} \cdot I_0 + S \cdot I_1 \tag{5.1}$$

Y si ahora "escuchamos" lo que la ecuación nos dice, veremos que tiene mucho sentido:

"Si S=0, $F = I_0$ y si S=1, $F = I_1$"

¡¡Es justo la definición de un multiplexor!! La salida toma el valor de una de las entradas, según el valor que tome la entrada de selección.

En realidad, el multiplexor lo podríamos haber descrito de una manera más sencilla, y podríamos haber obtenido la ecuación de otra forma. Veamos cómo.

La función F que describe el comportamiento de un multiplexor con una única entrada de selección, la podemos describir mediante la siguiente tabla:

S	F
0	I_0
1	I_1

que lo que nos viene a decir es lo mismo que su ecuación: cuando S=0, por la salida del multiplexor aparece el valor I_0 y cuando S=1, aparece el valor I_1. Estamos considerando las variables I_0 e I_1 como parámetros y NO como variables de entrada del circuito y por tanto estamos considerando como si la función F sólo dependiese de la variable S, es decir, tenemos la función F(S). **¿Cómo podemos obtener la ecuación del multiplexor a partir de esta tabla?**: aplicando el **teorema de expansión**, que vimos en el apartado 3.4 obtenemos los siguiente:

$$F(S) = S \cdot F(1) + \overline{S} \cdot F(0)$$

y F(1) es la salida del multiplexor cuando S=1, es decir, que $F(1) = I_1$ y F(0) es la salida cuando S=0, $F(0) = I_0$. La ecuación del multiplexor es la siguiente:

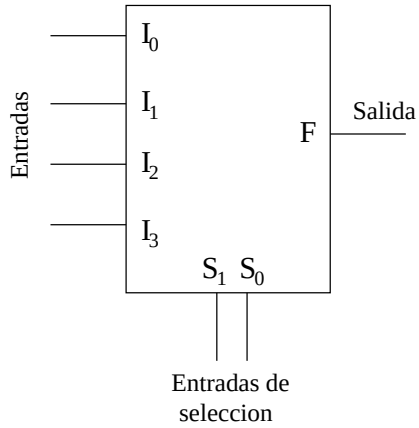
$$F(S) = S \cdot I_1 + \overline{S} \cdot I_0$$

¡¡Que es la misma ecuación que habíamos obtenido por Karnaugh!!

No se asuste el lector por los desarrollos teóricos. Lo importante es comprender cómo funcionan este tipo de multiplexores y cuál es la ecuación que los describe, independientemente de cómo la hallamos obtenido. Aquí, hemos obtenido la ecuación por dos métodos diferentes. Veremos que con los multiplexores de dos entradas de selección sólo lo podremos hacer por el segundo método.

Multiplexores con dos entradas de selección.

El siguiente multiplexor en complejidad es el que tenga **2 entradas de selección**, por lo que se **podrá seleccionar hasta 4 entradas posibles**. Habrá por tanto **4 entradas de datos**. El circuito es como el siguiente:



Hay 4 entradas de datos y 2 entradas de selección, en total 6 entradas. Ahora hacemos lo mismo que antes, construimos la tabla de verdad y aplicamos Karnaugh... pero.... ¿6 variables? **¡¡Hay que hacer una tabla que tenga $2^6 = 64$ filas!! ¡¡Y luego aplicar Karnaugh a una función de 6 variables!!!**

Vemos que este método, aunque fácil, require muchas operaciones. ¡¡Es un método ideal para que lo haga un ordenador!!. Nosotros obtendremos sus ecuaciones de otra manera diferente.

Vamos a describir este multiplexor mediante la siguiente tabla:

S_1	S_0	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

que lo que nos está expresando es que la salida del multiplexor valdrá I_0,I_1, I_2 o I_3 según el valor que tomen las variables de entrada S_1 y S_0. Estamos considerando que la función F sólo depende de estas dos variables: $F(S_1, S_0)$ y que I_0,I_1, I_2 e I_3 son parámetros, es decir, valores constantes que pueden valer '0' ó '1'.

Si aplicamos **el teorema de expansión** a la función $F(S_1, S_0)$, desarrollándola por S_1, obtenemos lo siguiente:

$$F(S_1, S_0) = S_1 \cdot F(1, S_0) + \overline{S_1} \cdot F(0, S_0)$$

Y si ahora aplicamos nuevamente el **teorema de expansión** a las funciones $F(1, S_0)$ y $F(0, S_0)$, desarrollándolas por la variable S_0, tenemos lo siguiente:

$$F(1, S_0) = S_0 \cdot F(1, 1) + \overline{S_0} \cdot F(1, 0)$$

$$F(0, S_0) = S_0 \cdot F(0, 1) + \overline{S_0} \cdot F(0, 0)$$

Y ahora, si lo juntamos todo en una única expresión, tenemos:

$$F(S_1, S_0) = S_1 \cdot F(1, S_0) + \overline{S_1} \cdot F(0, S_0) =$$

$$S_1 \cdot \Big[S_0 \cdot F(1, 1) + \overline{S_0} \cdot F(1, 0)\Big] + \overline{S_1} \cdot \Big[S_0 \cdot F(0, 1) + \overline{S_0} \cdot F(0, 0)\Big] =$$

$$S_1 S_0 F(1, 1) + S_1 \overline{S_0} F(1, 0) + \overline{S_1} S_0 F(0, 1) + \overline{S_1 S_0} F(0, 0)$$

¿Cuando vale F(0,0)?, es decir, ¿cuál es la salida del multiplexor cuando $S_0 = 0$ y $S_1 = 0$?. Por la definición de multiplexor, la salida será lo que venga por el canal 0, que es I_0. De la misma manera obtenemos que $F(0, 1) = I_1$, $F(1, 0) = I_2$, $F(1, 1) = I_3$. Sustituyendo estos valores en la ecuación anterior y reordenándola un poco tenemos la **expresión final para un multiplexor de dos entradas de selección**:

$$F = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3 \tag{5.2}$$

Olvidémonos ahora de cómo hemos obtenido esa ecuación. Lo importante es entenderla y

saber utilizarla. Vamos a comprobar si efectivamente esta ecuación describe el funcionamiento de un multiplexor de 2 entradas de selección y 4 entradas de datos.

Si $S_1 = 0$ y $S_0 = 0$, sabemos por el comportamiento de un multiplexor que se seleccionará la entrada I_0 para que aparezca por la salida. Vamos a comprobarlo. En la ecuación del multiplexor sustituimos S_1 por 0 y S_0 por 1. Obtenemos:

$$F = \overline{0} \cdot \overline{0} \cdot I_0 + \overline{0} \cdot 0 \cdot I_1 + 0 \cdot \overline{0} \cdot I_2 + 0 \cdot 0 \cdot I_3 =$$

$$= 1 \cdot 1 \cdot I_0 + 1 \cdot 0 \cdot I_1 + 0 \cdot 1 \cdot I_2 + 0 \cdot 0 \cdot I_3 =$$

$$= I_0$$

Se deja como ejercicio el que se compruebe la ecuación para el resto de valores de las entradas de selección.

Multiplexor con cualquier número de entradas de selección

Si ahora tenemos un multiplexor con 3 entradas de selección, que me permitirá seleccionar entre 8 entradas de datos, la ecuación que lo describe es la generalización de la ecuación 5.2. En total habrá 8 sumandos y en cada uno de ellos se encontrarán las variables S_2, S_1, y S_0, además de los correspondientes parámetros I_0,I_1, ..., I_7.

La ecuación será:

$$F = \overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot I_1 + \overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot I_2 + \overline{S_2} \cdot S_1 \cdot S_0 \cdot I_3 +$$

$$+ S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot I_4 + S_2 \cdot \overline{S_1} \cdot S_0 \cdot I_5 + S_2 \cdot S_1 \cdot \overline{S_0} \cdot I_6 + S_2 \cdot S_1 \cdot S_0 \cdot I_7$$

Y lo mismo podemos hacer para cualquier multiplexor con un número de entradas de selección mayor, lo que ocurre que la ecuación tendrá muchos más términos.

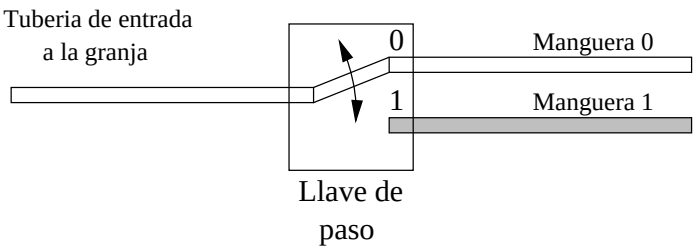


Figura 5.5: Similitud entre un demultiplexor y un sistema de agua de una granja

5.3. Demultiplexores

5.3.1. Conceptos

El concepto de demultiplexor es similar al de multiplexor, viendo las entradas de datos como salidas y la salida como entradas. En un multiplexor hay varias entradas de datos, y sólo una de ellas se saca por el canal de salida. En los demultiplexores hay un único canal de entrada que se saca por una de las múltiples salidas (y sólo por una!!!).

Si utilizamos el símil de la granja y las tuberías, podemos imaginar el siguiente escenario. Supongamos que ahora a la granja le llega una única tubería con agua, pero en el interior de la granja hay varias mangueras, cada una para limpiar una zona del establo o dar de beber a los animales de esa zona. Cómo sólo hay un granjero, sólo podrá usar una de las mangueras cada vez (el granjero no podrá usar a la vez dos mangueras, porque están en sitios diferentes!!).

Para seleccionar qué manguera quiere usar en cada momento, hay una llave de paso, de manera que si la sitúa en una posición, el agua que viene por la entrada saldrá por la manguera 0, mientras que si la sitúa en la otra posición, el agua saldrá por la manguera 1 (ver figura 5.5)

De la misma manera que en los multiplexores puede haber varias entradas, en los demultiplexores puede haber varias salidas. Por ejemplo en la figura 5.6 se muestra el mismo sistema de tuberías de la granja, pero ahora hay 4 mangueras, para llegar a 4 zonas distintas de la granja. Ahora el granjero tendrá que posicionar la llave de paso en una de las 4 posiciones posibles, para que el agua salga por la manguera seleccionada.

Ya comprendemos cómo funcionan los demultiplexores. Si lo aplicamos al mundo de la electrónica, en vez de tuberías tendremos canales de datos. Habrá un único canal de entrada, por el que llegarán números, que saldrán sólo por uno de los canales de salida, el que tengamos seleccionado, como se muestra en la figura 5.7.

En general en un demultiplexor tendremos:

- **Una entrada de datos**

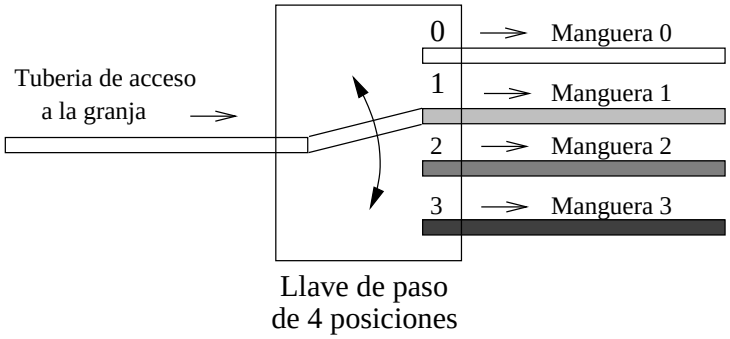


Figura 5.6: Sistema de agua de 4 mangueras

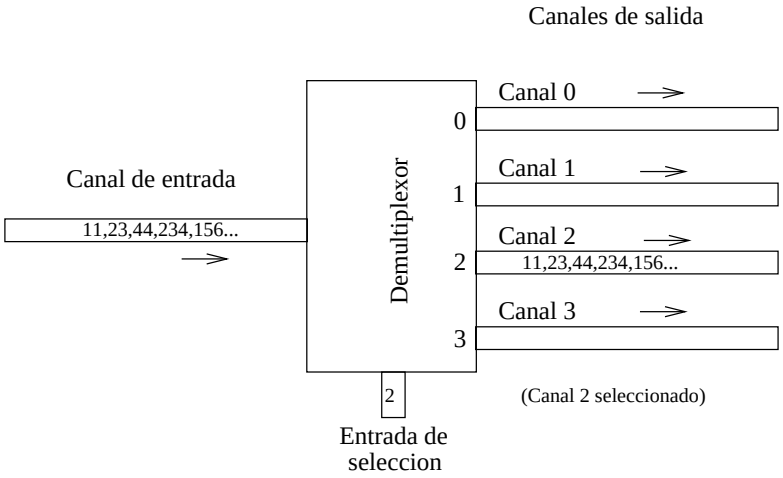


Figura 5.7: Un demultiplexor que selecciona entre 4 canales de datos

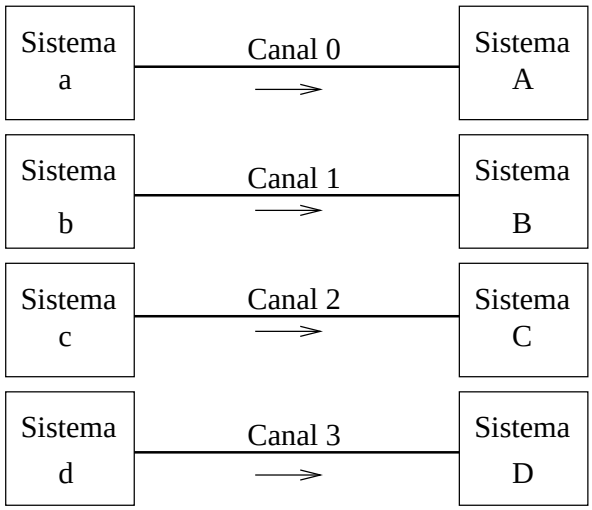


Figura 5.8: Una alternativa para comunicar sistemas

- **Una entrada de selección**: que indica a cuál de las salidas se manda la entrada
- **Varios canales de datos de salida**. Sólo estará activo el que se haya seleccionado.

5.3.2. Juntando multiplexores y demultiplexores

Vamos a ver una aplicación típica de los multiplexores y los demultiplexores. Imaginemos que tenemos 4 sistemas, que los llamaremos **a**,**b**,**c** y **d**, y que necesitan enviar información a otros 4 dispositivos **A**,**B**,**C** y **D**. La comunicación es uno a uno, es decir, el sistema **a** sólo envía información al sistema **A**, el **b** al **B**, el **c** al **C** y el **d** al **D**.

¿Qué alternativas hay para que se produzca este envío de datos? Una posibilidad es obvia, y es la que se muestra en la figura 5.10. Directamente se tiran cables para establecer los canales de comunicación.

Pero esta no es la única solución. Puede ser que podamos tirar los 4 cables, porque sean muy caros o porque sólo haya un único cable que comunique ambas parte, y será necesario llevar por ese cable todas las comunicaciones.

La solución se muestra en la figura 5.9. Vemos que los sismteas **a**, **b**, **c** y **d** se conectan a un multiplexor. Un circuito de control, conectado a las entradas de selección de este multiplexor, selecciona periódicamente los diferentes sistemas, enviando por la salida el canal correspondiente. Podemos ver que a la salida del multiplexor se encuentra la información enviada por los 4 sistemas. Se dice que esta información está **multiplexada en el tiempo**. Al final de esta línea hay un demultiplexor que realiza la función inversa. Un circuito de control selecciona periódicamente

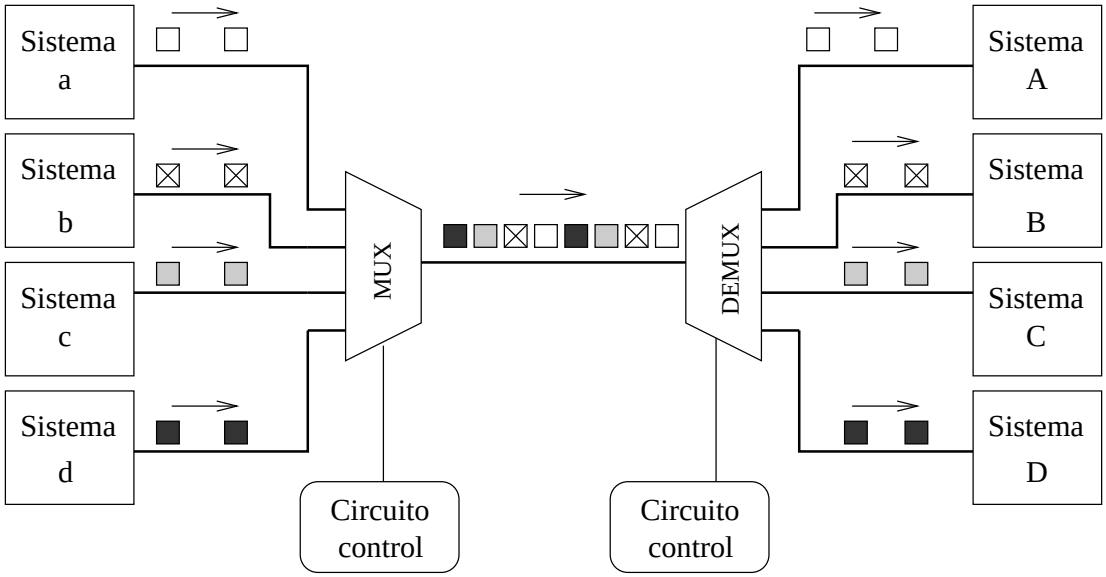


Figura 5.9: Uso de un multiplexor y demultiplexor para transmisión de datos por un único cable

por qué salidas debe salir la información que llega por la entrada.

Lo que hemos conseguido es que toda la información enviada por un sistema, llega a su homólogo en el extremo anterior, pero sólo hemos utilizado un único canal de datos.

5.3.3. Demultiplexores y bits

Un demultiplexor, como cualquier otro circuito digital trabaja sólo con **números**. Pero estos números vendrán expresados en **binario**, por lo que los **canales de datos de entrada y salida**, y **la entrada de selección** vendrán e**xpresados en binario** y tendrán un número determinado de **bits**.

Una vez más nos hacemos la pregunta, ¿Cuantos bits?. Depende. Depende de la aplicación que estemos diseñando o con la que estemos trabajando. En la figura 5.10 se muestran dos demultiplexores de 4 canales, por lo que tendrán 2 bits para la entrada de selección. El de la izquierda tiene canales de 2 bits y el de la derecha de 1 bit.

Los multiplexores que vamos a estudiar son lo que tienen canales de 1 bit. A partir de ellos podremos construir multiplexores con un mayor número de bits por canal.

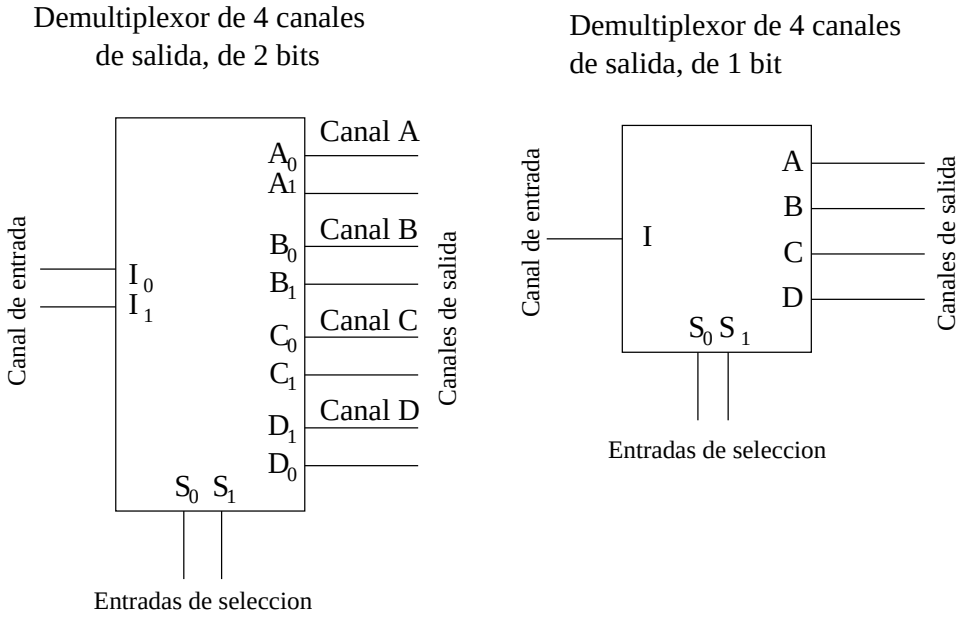
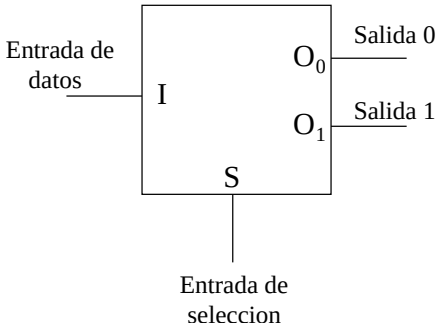


Figura 5.10: Dos demultiplexores de 4 canales de salida

5.3.4. Demultiplexores de 1 bit y sus expresiones booleanas

Demultiplexor de una entrada de selección

El demultiplexor más simple es el que tiene una entrada de selección, una entrada de datos y dos salidas. Según el valor de la entrada de selección, la entrada de datos se sacará por la salida O_0 o por la O_1:



Nos hacemos la misma pregunta que en el caso de los multiplesore: ¿Cómo podemos expresar las funciones de salida usando el Algebra de Boole?. Podemos escribir la tabla de verdad y obtener las expresiones más simplificadas. Para tener la tabla aplicamos la definición de demultiplexor y vamos comprobando caso por caso qué valores aparecen en las salidas. Por ejemplo, si S=1 e I=1, se estará seleccionando la salida O_1, y por ella saldrá el valor de I, que es 1. La salida

O_0 no estará seleccionada y tendrá el valor 0.

S	I	O_1	O_0
0	**0**	0	**0**
0	**1**	0	**1**
1	**0**	**0**	0
1	**1**	**1**	0

Para obtener las expresiones de O_0 y O_1 no hace falta aplicar Karnaugh puesto que cada salida sólo toma el valor '1' para un caso y '0' para todos los restantes. Desarrollando por la primera forma canónica:

$$O_1 = S \cdot I$$

$$O_0 = \overline{S} \cdot I$$

Y podemos comprobar que si hemos seleccionado la salida 0 ($S = 0$), entonces $O_0 = I$ y $O_1 = 0$, y si hemos seleccionado la salida 1 ($S = 1$), $O_0 = 0$ y $O_1 = I$.

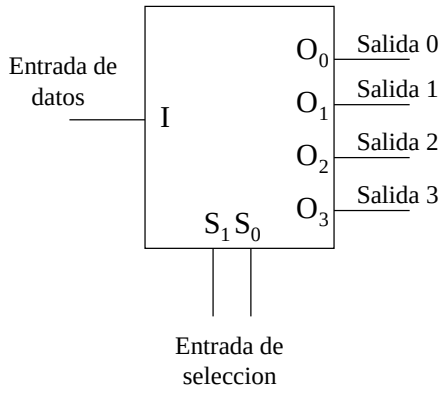
De la misma manera que hicimos con los multiplexores, podemos considerar que las funciones O_0 y O_1 sólo dependen de la entrada de Selección (S), tomando la entrada I como un parámetro. Así podemos describir este demultiplexor mediante la siguiente tabla:

S	O_1	O_0
0	I	0
1	0	I

Esta descripción será la que empleemos, ya que es más compacta.

Demultiplexor de dos entradas de selección

Este demultiplexor tiene dos entradas de selección y cuatro salidas:



La tabla de verdad “abreviada” la podemos expresar así:

S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

La entrada I se saca por la salida indicada en las entradas de selección. Las ecuaciones de las funciones de salida son:

$$O_0 = \overline{S_1} \cdot \overline{S_0} \cdot I$$

$$O_1 = \overline{S_1} \cdot S_0 \cdot I$$

$$O_2 = S_1 \cdot \overline{S_0} \cdot I$$

$$O_3 = S_1 \cdot S_0 \cdot I$$

Si analizamos la ecuación de O_0 lo que nos dice es lo siguiente: “$O_0 = I$ sólo cuando $S_1 = 0$ y $S_0 = 0$”. Para el resto de valores que pueden tomar las entradas de selección S_1 y S_0, O_0 siempre será 0.

Demultiplexor con cualquier número de entradas de selección

Para demultiplexores con mayor número de entradas de selección, las ecuaciones serán similares. Por ejemplo, en el caso de un demultiplexor que tenga tres entradas de selección: S_2, S_1 y S_0, y que por tanto tendrá 8 salidas, la ecuación para la salida O_5 será:

$$O_5 = S_2 \cdot \overline{S_1} \cdot S_0 \cdot I$$

y la ecuación de la salida O_0 será:

$$O_0 = \overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot I$$

Se deja como ejercicio al lector el que obtenga el resto de ecuaciones de salida.

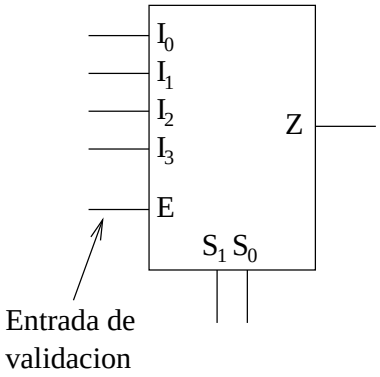
5.4. Multiplexores con entrada de validación (ENABLE)

Los multiplexores, y en general la mayoría de circuitos MSI, disponen de una **entrada adicional**, llamada **entrada de validación** (en inglés **Enable**). Esta entrada funciona como un interruptor de encendido/apagado para el circuito MSI. Si la **entrada de validación está activada**, **el circuito funcionará normalmente**. Pero si esta está desactivada, el circuito sacará el valor '0' por todas sus salidas, independientemente de lo que llegue por sus entradas. Se dice que está deshabilitado (no está en funcionamiento).

Las entradas de validación se les suele llamar E (del inglés Enable) y pueden ser de dos tipos: activas a nivel alto ó activas a nivel bajo.

5.4.1. Entrada de validación activa a nivel alto

Si esta entrada se encuentra a '1' (E=1) el multiplexor funciona normalmente (está conectado). Si se encuenrta a '0' (E=0) entonces su salida será '0' (estará desconectado). A continuación se muestra un multiplexor de 4 entradas de datos, 2 entradas de selección y una entrada de validación activa a nivel alto:



La tabla de verdad es la siguiente:

E	S_1	S_0	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

Sólo en los casos en los que E=1, el multiplexor se comporta como tal. Cuando E=0, la salida Z siempre está a '0'. Esta tabla de verdad se suele escribir de una manera más abreviada de la siguiente manera:

E	S_1	S_0	Z
0	**x**	**x**	**0**
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

Con las 'x' de la primera fila se indica que cuando E=0, independientemente de los valores que tengan las entradas S_1 y S_0 la salida siempre tendrá el valor '0'.

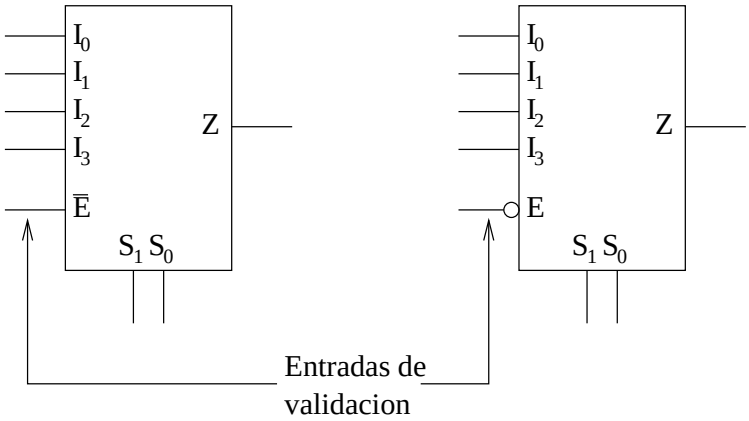
¿Y cual sería la nueva ecuación de este multiplexor? La misma que antes pero ahora multiplicada por E:

$$Z = (\overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3) \cdot E$$

Si E=0, entonces Z=0. El multiplexor está **deshabilitado**.

5.4.2. Entrada de validación activa a nivel bajo

Otros fabricamentes de circuitos integrados utilizan una **entrada de validación activa a nivel bajo**, que es justamente la inversa de la enterior. Se suele denotar mediante $\overline{E}$. Cuando la entrada E está a '0' el multiplexor funciona normalmente, y cuando está a '1' está desconectado. En la siguiente figura se muestran dos multiplexores de 4 entradas, dos entradas de selección y una entrada de validación activa a nivel bajo. Ambos multiplexores son iguales, pero se han utilizado notaciones distintas. En el de la izquierda se utiliza $\overline{E}$ y en el de la derecha E pero con un pequeño círculo en la entrada:



La tabla de verdad es la siguiente:

E	S_1	S_0	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	**x**	**x**	**0**

Y la nueva ecuación es:

$$Z = (\overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3) \cdot \overline{E}$$

Cuando E=1, $\overline{E} = 0$ y entonces Z=0, con lo que el multiplexor se encuentra **deshabilitado**.

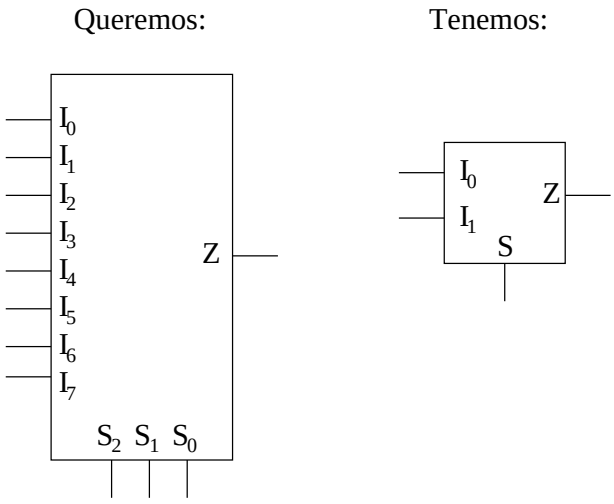
5.5. Extensión de multiplexores

La idea es poder conseguir tener **multiplexores más grandes a partir de otros más pequeños**. Y esto es necesario porque en nuestros diseños podemos necesitar unos multiplexores grandes, sin embargo en el mercado nos encontramos con multiplexores menores. Tenemos que saber cómo construir los multiplexores que necesitamos para nuestra aplicación a partir de los multiplexores que encontramos en el mercado.

La extensión puede ser bien **aumentando el número de entradas**, bien **aumentando el número de bits por cada canal de datos** o bien ambos a la vez.

5.5.1. Aumento del número de entradas

La solución es **conectarlos en cascada**. Lo mejor es verlo con un ejemplo. Imaginemos que necesitamos una multiplexor de 8 canales, pero sólo disponemos de varios de 2 canales:

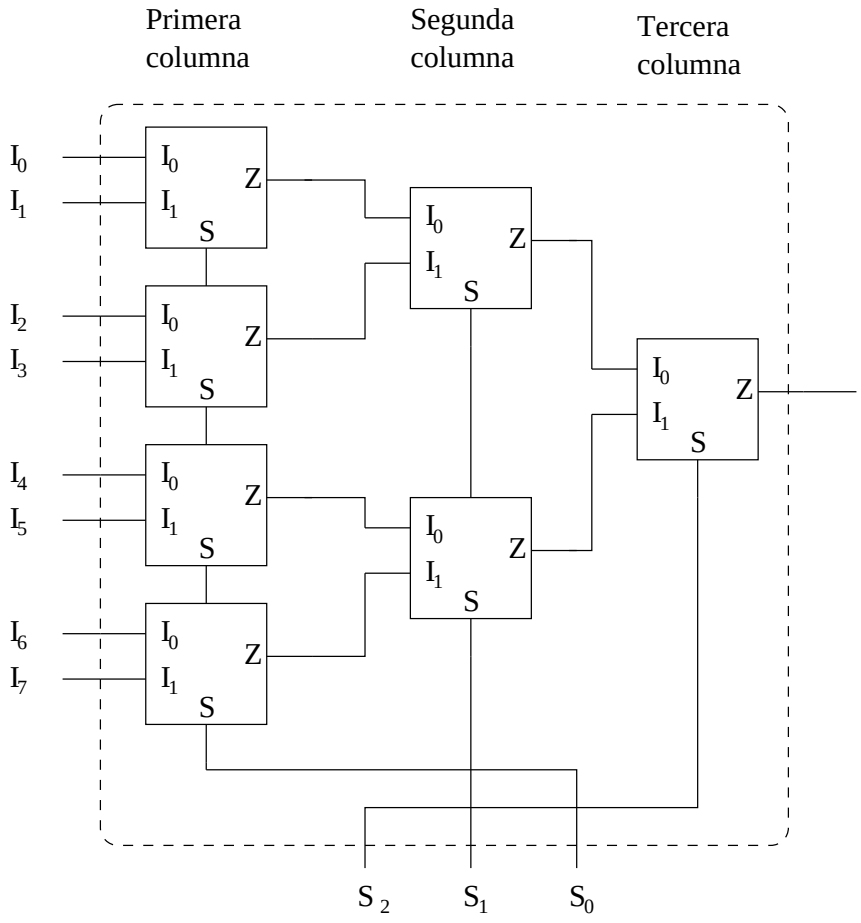


La solución es **conectarlos en cascada**. Primero colocamos una columna de 4 multiplexores de dos entradas, para tener en total 8 entradas. Todas las entradas de selección de esta primera columna se unen. Por comodidad en el dibujo, esto se representa mediante una línea vertical que une la salida S de un multiplexor con el de abajo.

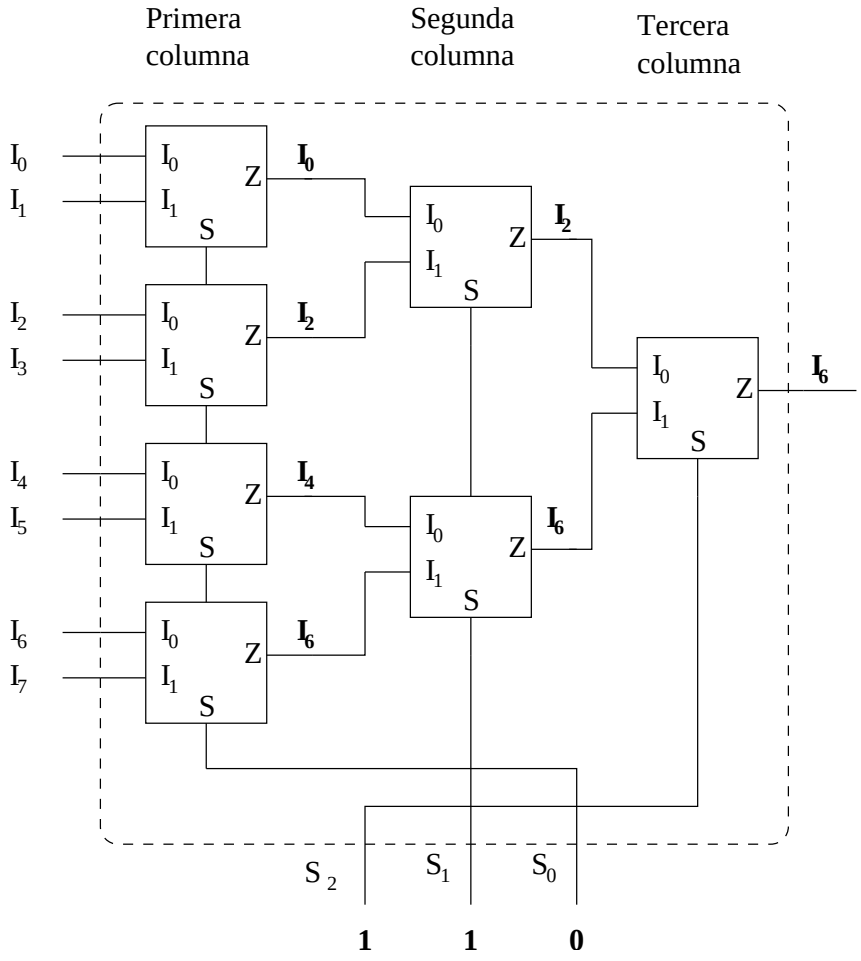
A continuación colocamos una segunda columna de 2 multiplexores de 2 entradas, también con sus entradas de selección unidas. Finalmente colocamos una última columna con un único multiplexor de 2 entradas.

Colocados de esta manera, conseguimos tener un multiplexor de 8 entradas y tres entradas de selección. La única consideración que hay que tener en cuenta es que la entrada de selección de

los multiplexores de la primera columna tiene peso 0, la segunda peso 1 y la última peso 2:



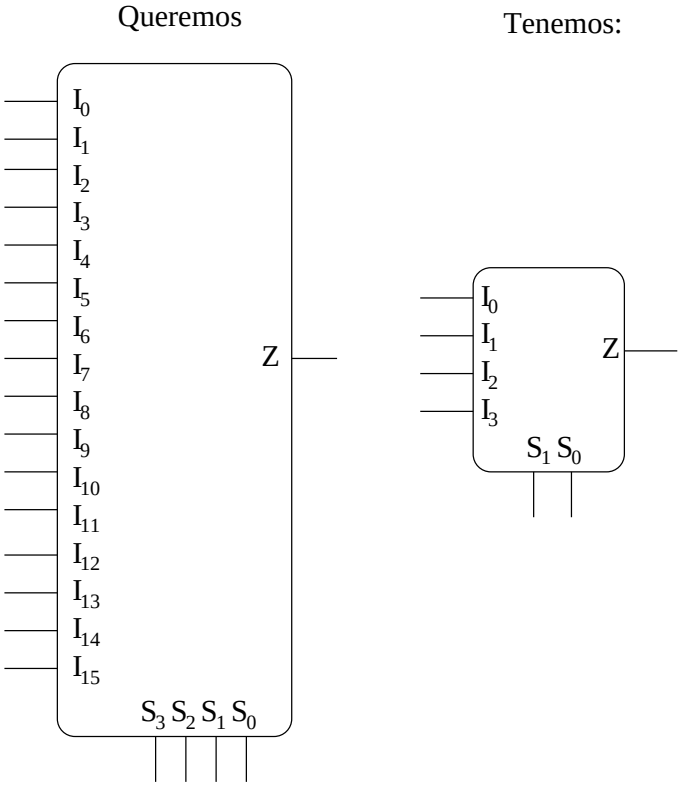
¡¡Vamos a comprobarlo!! (Siempre que se hace un diseño hay que comprobar si es correcto). Vamos a comprobar qué ocurre si seleccionamos el canal 6. Introducimos en binario el número 6 por las entradas de selección: $S_2 = 1$, $S_1 = 1$ y $S_0 = 0$. Por la entrada S de los multiplexores de la primera columna se introduce un '0', por lo que estos multiplexores sacan por sus salidas lo que hay en sus entradas I_1: I_0, I_2, I_4 e I_6. Por la entrada de selección de los multiplexores de la segunda columna se introduce un '1' por lo que están seleccionando su canal I_1. A la salida de estos multiplexores se tendrá: I_2 e I_6. Finalmente, el multiplexor de la última columna está seleccionando su entrada I_1, por lo que la salida final es I_6 (Recordar la idea de multiplexor como una llave de paso que conecta tuberías de agua):



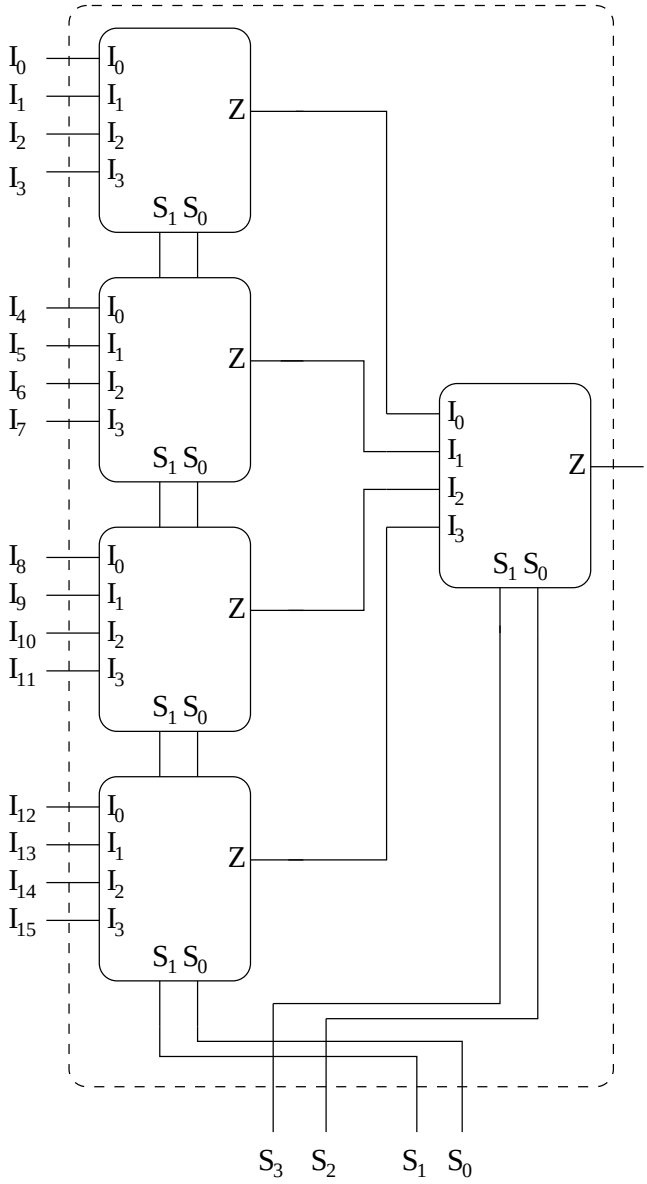
Ejemplo:

Construir un multiplexor de 16 entradas usando multiplexores de 4.

En este caso lo que *queremos* y lo que *tenemos* es lo siguiente:



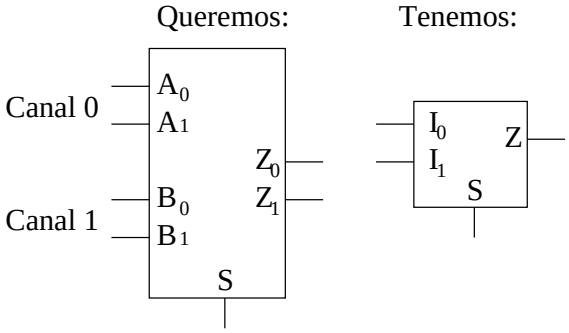
Los conectamos en cascada, para lo cual necesitamos una primera columna de 4 multiplexores de 4 entradas, con entradas S_0 de todos ellos unidos, así como las S_1. En la segunda fila hay un único multiplexor de 4 entradas:



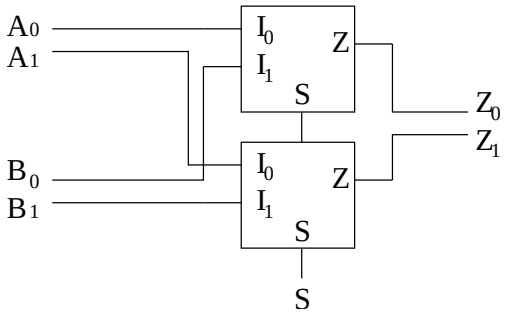
Se deja como ejercicio la comprobación de este diseño.

5.5.2. Aumento del número de bits por canal

Para conseguir esto hay que **conectarlos en paralelo**. Imaginemos que tenemos queremos construir un multiplexor de dos canales de entrada, cada uno de ellos de 2 bits, y para ello disponemos de multiplexores de 2 canales de un bit:



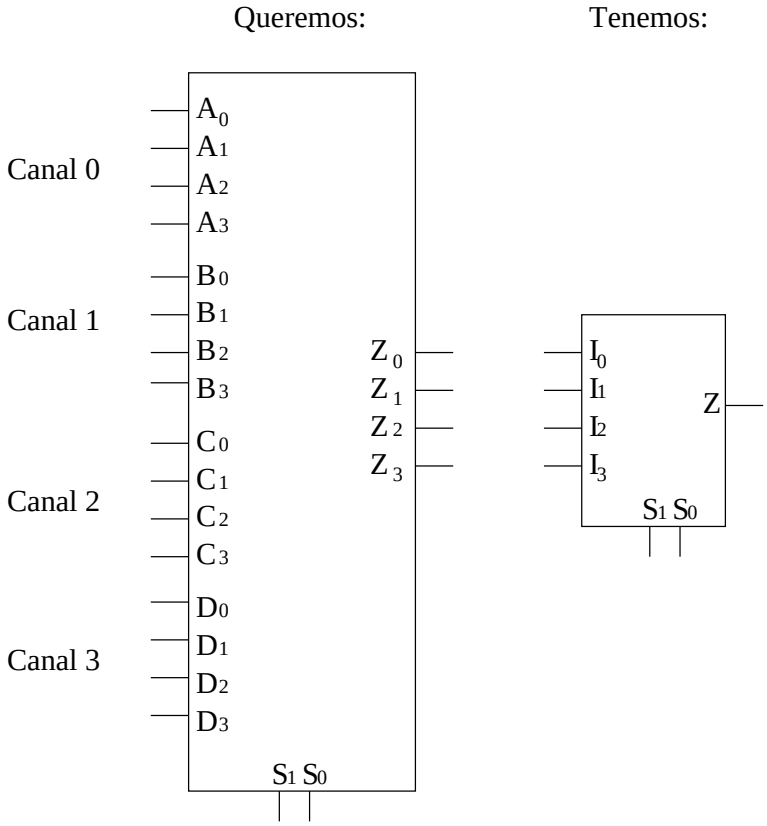
Utilizaremos dos multiplexores de lo que tenemos, uno por cada bit que tengamos en el nuevo canal de salida. Como los canales en el nuevo multiplexor son de 2 bits, necesitaremos 2 multiplexores de canales de 1 bit. Uno de estos multiplexores será al que vayan los bits de menos peso de los canales de entrada y el otro los de mayor peso. Las entradas de selección de ambos están unidas:



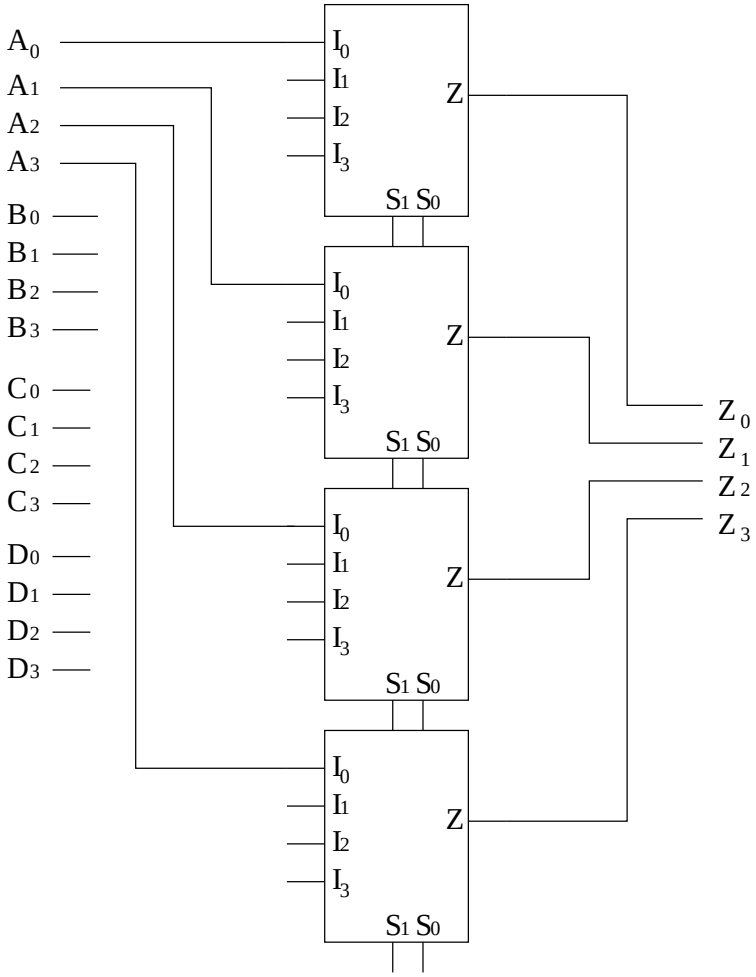
Si con en este nuevo multiplexor hacemos S=0, las salidas serán $Z_0 = A_0$ y $Z_1 = A_1$. Y si hacemos S=1, entonces obtenemos $Z_0 = B_0$ y $Z_1 = B_1$. ¡¡Es lo que andábamos buscando!!. Por la salida obtenemos bien el número que viene por el canal 0 (A_1A_0) ó bien el número que viene por el canal 1(B_1B_0).

Ejemplo:

Construir un muliplexor de 4 canales de 4 bits, usando multiplexores de 4 entradas de 1 bit.



Ahora necesitaremos 4 multiplexores de los que tenemos, a cada uno de los cuales les llegan los bits del mismo peso de los diferentes canales. Por el primer multiplexor entran los bits de menor peso (A_0, B_0, C_0 y D_0) y por el último los de mayor (A_3, B_3, C_3 y D_3). En el dibujo no se muestran todas las conexiones para no complicarlo:



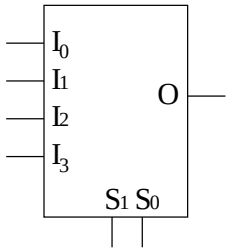
5.6. Implementación de funciones con MX's

Utilizando multiplexores es posible implementar funciones booleanas. En general, ***cualquier función de n variables se puede implementar utilizando un multiplexor de n-1 entradas de selección***.

Por ejemplo, dada la función:

$$F = \overline{x} \cdot y \cdot z + x \cdot \overline{y} + \overline{x} \cdot \overline{y} \cdot \overline{z}$$

que tiene 3 variables, se puede implementar utilizando un multiplexor de 2 entradas de control, como el mostrado a continuación:



Existen dos maneras de hacerlo. Una es emplear el algebra de boole y la ecuación de este tipo de multiplexores. Por lo general este método es más complicado. La otra es utilizar un método basado en la tabla de verdad.

5.6.1. Método basado en el Algebra de Boole

La ecuación de un multiplexor de 2 entradas de control y 4 entradas es la siguiente:

$$O = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$

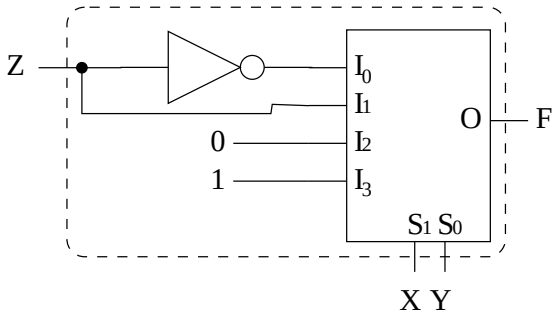
La ecuación de la función que queremos implementar la podemos expresar de la siguiente forma:

$$F = \overline{x} \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot y \cdot z + x \cdot \overline{y} \cdot 1 + x \cdot y \cdot 0$$

¡¡Que es muy parecida a Z!!. Si igualamos términos, obtenemos que por las entradas del multiplexor hay que introducir:

- $I_0 = \overline{z}$
- $I_1 = z$
- $I_2 = 1$
- $I_3 = 0$
- $S_0 = y$
- $S_1 = x$

La función se implementa así:



Vamos a comprobarlo. Para ello sustituimos en la ecuación del multiplexor los valores que estamos introduciendo por las entradas:

$$O = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3 = \overline{x} \cdot \overline{y} \cdot I_0 + \overline{x} \cdot y \cdot I_1 + x \cdot \overline{y} \cdot I_2 + x \cdot y \cdot I_3 =$$

$$= \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y} \cdot 0 + x \cdot y \cdot 1 = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot y = F$$

5.6.2. Método basado en la tabla de verdad

Este método se basa en lo mismo, pero se usan las tablas de verdad en vez de utilizar las ecuaciones del multiplexor, por ello es más sencillo e intuitivo. Además tiene otra ventaja: es un método mecánico, siempre se hace igual sea cual sea la función (Aunque como se verá en los ejercicios algunas funciones se pueden implementar de manera más fácil si utilizamos la entrada de validación).

Vamos a realizar este ejemplo con la función anterior. Seguimos los siguientes pasos:

1. Construimos la tabla de verdad de la función F a implementar.

X	**Y**	**Z**	**O**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

2. Dividimos la tabla en tantos grupos como canales de entrada halla. En este caso hay 4 entradas, por lo que hacemos 4 grupos. Las variables de mayor peso se introducen directamente por las entradas de selección S_1 y S_0:

X	Y	Z	O
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

   Las variables X e Y son las que se han introducido por las entradas de selección ($S_1 = x$, $S_0 = y$). Vemos que hay 4 grupos de filas. El primer grupo se corresponde con la entrada I_0, el siguiente por la I_1, el siguiente por la I_2 y el último por la I_3.

3. El valor a introducir por las entradas I_0, I_1, I_2, e I_3 lo obtenemos mirando las columnas de la derecha (la columna de Z y de O).

   En el primer grupo, cuando Z=0, O=1 y cuando Z=1, O=0, por tanto $O = \overline{Z}$. Esa será la salida cuando se seleccione el canal 0, por tanto por su entrada habrá que introducir lo mismo: $I_0 = \overline{Z}$.

   Ahora nos fijamos en el siguiente grupo, correspondiente a I_1. En este caso, cuando Z=0, O=0 y cuando Z=1, O=1, por lo que deducimos que $O = I_1 = Z$.

   Vamos a por el tercer grupo. Si Z=0, O=0 y si Z=0, también O=0. Independientemente del valor de Z, la salida vale 0: $I_2 = 0$.

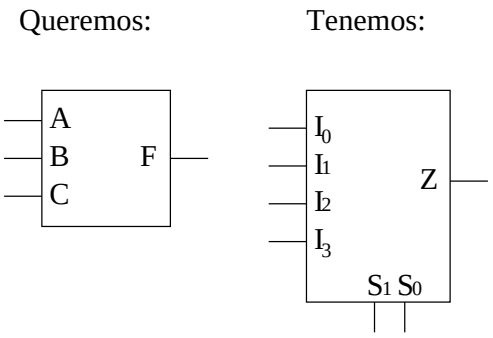
   Y para el último grupo ocurre que si Z=0, O=1, y si Z=1, O=1. Deducimos que $I_3 = 1$.

Si ahora hacemos la conexiones obtenemos el mismo circuito que en el caso anterior.

Ejemplo:

Implementar la función $F = A \cdot B + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C$ utilizando un multiplexor, sin entrada de validación.

Utilizaremos el método basado en las tablas de verdad. Lo que queremos implementar es un circuito que tiene 3 entradas y una salida. Como tienen 3 variables de entrada, en general necesitaremos un multiplexor de 2 entradas de control:



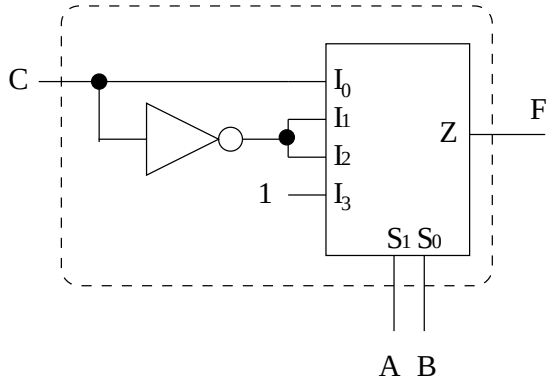
Ahora vamos siguiente los pasos del método. Primero construimos la tabla de verdad a partir de F:

A	**B**	**C**	**F**
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	1
1	1	1	1

Las entradas A y B las conectamos directamente a S_1 y S_0 respectivamente. Fijándonos en las columnas de C y F, deducimos las siguientes conexiones:

- $I_0 = C$
- $I_1 = \overline{C}$
- $I_2 = \overline{C}$
- $I_3 = 1$

El circuito final es el siguiente:



5.6.3. Implementación de funciones con multiplexores con entrada de validación

Para implementar funciones también se puede usar la entrada de validación. En este caso no todas las funciones se pueden implementar con este tipo de multiplexores. La entrada de validación la usamos como si fuese una entrada más.

Ejemplo

Implementar la siguiente función utilizando un multiplexor

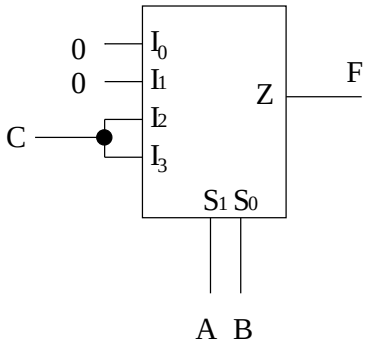
$$F = A \cdot \overline{B} \cdot C + A \cdot B \cdot C$$

Primero utilizaremos un multiplexor sin entrada de validación, utilizando el metodo de las tablas de verdad. Como la función tiene 3 variables, necesitamos un multiplexor de 2 entradas de control.

La tabla de verdad de esta función es:

A	**B**	**C**	**F**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Las entradas A y B se conectan directamente a las entradas S_1 y S_0. Los valores que se introducen por las entradas son: $I_0 = 0$, $I_1 = 0$, $I_2 = I_3 = C$. El circuito es el siguiente:



¿Se podría implementar esta función con un multiplexor con entrada de validación?. Si nos fijamos en la función F vemos que podemos sacar factor común en A:

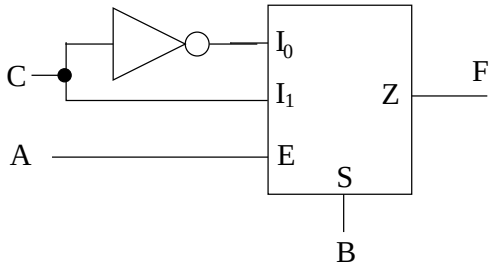
$$F = A \cdot \overline{B} \cdot C + A \cdot B \cdot C = A \cdot (\overline{B} \cdot \overline{C} + B \cdot C)$$

¡¡ Y esa es la ecuación de un multiplexor de una entrada de control y una entrada de validación!!. Si A=0, entonces F=0, y si A=1, se comporta como un multiplexo. Por tanto introducimos A directamente por la entrada de validación y para el resto necesitamos un multiplexor de 1 entrada de selección. Y como la ecuación es tan sencilla, no hace falta ni siquiera hacer el método de las tablas de verdad, fijándonos en su ecuación es suficiente.

La ecuación de un multiplexor con una entrada de selección es:

$$F = \overline{S} \cdot I_0 + S \cdot I_1$$

Si introducimos B por S, $\overline{C}$ por I_0 y C por I_1 ya lo tenemos:



5.7. Resumen

En este capítulo hemos visto los **multiplexores** y los **demultiplexores**, constituidos internamente por puertas lógicas. Los multiplexores nos permiten **seleccionar** entre uno de varios canales de entrada (tuberías) para sacarlo por la salida. Por ello disponen de unas **entradas de datos** (por donde entra el “agua”), **unas entradas de selección** (Llaves de paso) y un **canal de salida**. Estos canales de datos pueden ser de varios bits, sin embargo, en este capítulo nos hemos centrado en los multiplexores que tienen canales de datos de 1 bits, puesto que a partir de ellos podemos construir multiplexores con canales de datos de mayor cantidad de bit, así como multiplexores que tienen mayor cantidad de canales de entrada.

También hemos visto los **demultiplexores**, que realizan la función inversa. Un canal de entrada (tubería) se puede conectar a una de las diferentes salidas, según el valor introducido por las entradas de selección (llaves de paso).

Los multiplexores pueden tener opcionalmente una **entrada de validación**, que puede ser activa a nivel alto o a nivel bajo y actúa como una especie de interruptor que permite que el multiplexor funcione o no. Si está activada, el multiplexor funciona normalmente. Si la entrada de validación está desactivada, por la salida del multiplexor siempre hay un ’0’.

Por último hemos visto que con un multiplexor también se pueden **implementar funciones lógicas**, y es otra alternativa que tenemos además de las puertas lógicas. Metiante el **método de las tablas de verdad**, podemos saber facilmente qué variables hay que conectar a las entradas del multiplexor.

5.8. Ejercicios

Capítulo 6

Codificadores, decodificadores y comparadores

6.1. Introducción

En este capítulo veremos otros circuitos MSI: codificadores, decodificares y comparadores.

6.2. Codificadores

6.2.1. Conceptos

Los codificadores nos permiten **"compactar"** la información, **generando un código de salida a partir de la información de entrada**. Y como siempre, lo mejor es verlo con un ejemplo. Imaginemos que estamos diseñando un circuito digital que se encuentra en el interior de una *cadena de música*. Este circuito controlará la cadena, haciendo que funcione correctamente.

Una de las cosas que hará este circuito de control será activar la radio, el CD, la cinta o el Disco según el botón que haya pulsado el usuario. Imaginemos que tenemos 4 botones en la cadena, de manera que cuando no están pulsados, generan un '0' y cuando se pulsan un '1' (Botones digitales). Los podríamos conectar directamente a nuestro circuito de control la cadena de música, como se muestra en la figura 6.1.

Sin embargo, a la hora de diseñar el circuito de control, nos resultaría más sencillo que cada botón tuviese asociado un número. Como en total hay 4 botones, necesitaríamos 2 bits para identificarlos. Para conseguir esta asociación utilizamos un codificador, que a partir del botón que se haya pulsado nos devolverá su número asociado:

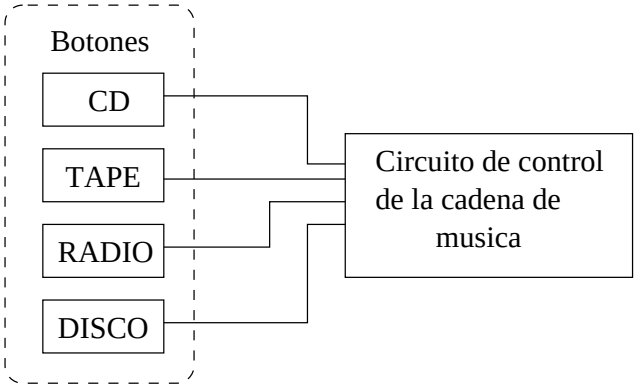
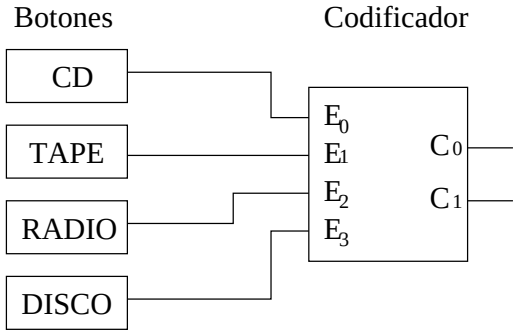


Figura 6.1: Circuito de control de una cadena de música, y 4 botones de selección de lo que se quiere escuchar



Fijémonos en las entradas del codificador, que están conectadas a los botones. **En cada momento, sólo habrá un botón apretado**, puesto que sólo podemos escuchar una de las cuatro cosas. Bien estaremos escuchando el CD, bien la cinta, bien la radio o bien un disco, pero no puede haber más de un botón pulsado[1]. Tal y como hemos hecho las conexiones al codificador, el **CD tiene asociado el número 0**, **la cinta el 1**, **la radio el 2** y **el disco el 3** (Este número depende de la entrada del codificador a la que lo hayamos conectado). **A la salida del codificador obtendremos el número del botón apretado**. La tabla de verdad será así:

E_3	E_2	E_1	E_0	C_1	C_0	**Botón**
0	0	0	1	0	0	**CD**
0	0	1	0	0	1	**TAPE**
0	1	0	0	1	0	**RADIO**
1	0	0	0	1	1	**DISCO**

El circuito de control de la cadena ahora sólo tendrá 2 bits de entrada para determinar el

[1] De hecho, en la cadena de música que tengo en casa, que es un poco antigua, cuando aprietas uno de los botones el otro "salta", y deja de estar apretado.

botón que se ha pulsado. Antes necesitábamos 4 entradas. El codificador que hemos usado tiene 4 entradas y 2 salidas, por lo que se llama **codificador de 4 a 2**. Existen codificadores de mayor número de entradas, como el que vamos a ver en el siguiente ejemplo.

Imaginemos que ahora queremos hacer un circuito para monitorizar la situación de un tren en una vía. En una zona determinada, la vía está dividida en 8 tramos. En cada uno de ellos existe un sensor que indica si el tren se encuentra en ese tramo (el sensor devuelve 1) o fuera de él (valor 0). Se ve claramente que cuando uno de los sensores esté activado, porque que el tren se encuentre en ese tramo, el resto de sensores devolverán un '0' (No detectan al tren).

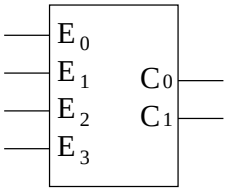
Si conectamos todas las entradas de los sensores a un **codificador de 8 a 3**, lo que tendremos es que a la salida del codificador saldrá un número que indica el tramo en el que se encuentra el tren. El circuito de control que conectemos a las salidas de este codificador sólo necesita 3 bits de entrada para conocer el tramo en el que está el tren, y no es necesario 8 bits. ¡¡Su diseño será más simple!!. La tabla de verdad es:

E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	C_2	C_1	C_0	**Tramo**
0	0	0	0	0	0	0	1	0	0	0	**0**
0	0	0	0	0	0	1	0	0	0	1	**1**
0	0	0	0	0	1	0	0	0	1	0	**2**
0	0	0	0	1	0	0	0	0	1	1	**3**
0	0	0	1	0	0	0	0	1	0	0	**4**
0	0	1	0	0	0	0	0	1	0	1	**5**
0	1	0	0	0	0	0	0	1	1	0	**6**
1	0	0	0	0	0	0	0	1	1	1	**7**

6.2.2. Ecuaciones

A continuación deduciremos las ecuaciones de un codificador de 4 a 2, y luego utilizaremos un método rápido para obtener las ecuaciones de un codificador de 8 a 3.

El codificador de 4 a 2 que emplearemos es el siguiente:



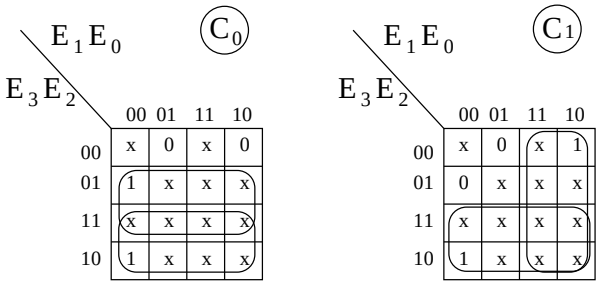
Las ecuaciones las obtenemos siguiendo el mismo método de siempre: primero obtendremos

la tabla de verdad completa y aplicaremos el método de Karnaugh. Con ello obtendremos las ecuaciones más simplificadas para las salidas C_1 y C_0.

Al hacer la tabla de verdad, hay que tener en cuenta que muchas de las entradas NO SE PUEDEN PRODUCIR. En las entradas de un decodificador, una y sólo una de las entradas estará activa en cada momento. Utilizaremos esto para simplificar las ecuaciones. Se ha utilizado una X para indicar que esa salida nunca se producirá:

E_3	E_2	E_1	E_0	C_1	C_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	x	x
0	1	0	0	1	0
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	x	x
1	0	0	0	1	1
1	0	0	1	x	x
1	0	1	0	x	x
1	0	1	1	x	x
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x

C_1 y C_0 siempre valen 'x' excepto para 4 filas. Los mapas de Karnaugh que obtenemos son:



Las casillas que tienen el valor 'x' podemos asignarles el valor que más nos convenga, de forma que obtengamos la expresión más simplificada. Las ecuaciones de un decodificador de 4 a 2 son:

$$C_0 = E_2 + E_3$$

$$C_1 = E_1 + E_3$$

La manera “rápida” de obtenerlas es mirando la tabla simplificada, como la que se muestra en el ejemplo de la cadena de música. Sólo hay que fijarse en los ’1’ de las funciones de salida (como si estuviésemos desarrollando por la primera forma canónica) y escribir la variable de entrada que vale ’1’. Habrá tantos sumandos como ’1’ en la función de salida.

Las ecuaciones para un codificador de 8 a 3, utilizando el método rápido, son:

$$C_0 = E_1 + E_2 + E_5 + E_7$$

$$C_1 = E_2 + E_3 + E_6 + E_7$$

$$C_2 = E_4 + E_5 + E_6 + E_7$$

6.3. Decodificadores

6.3.1. Conceptos

Un decodificador es un circuito integrado por el que se introduce un número y se activa una y sólo una de las salidas, permaneciendo el resto desactivadas. Y como siempre, lo mejor es verlo con un ejemplo sencillo. Imaginemos que queremos realizar un circuito de control para un semáforo. El semáforo puede estar verde, amarillo, rojo o averiado. En el caso de estar averiado, se activará una luz interna “azul”, para que el técnico sepa que lo tiene que reparar. A cada una de estas luces les vamos a asociar un número. Así el rojo será el 0, el amarillo el 1, el verde el 2 y el azul (averiado) el 3 (Ver figura 6.2).

Para controlar este semáforo podemos hacer un circuito que tenga 4 salidas, una para una de las luces. Cuando una de estas salidas esté a ’1’, la luz correspondiente estará encendida. Sin embargo, ocurre que NO PUEDE HABER DOS O MAS LUCES ENCENDIDAS A LA VEZ. Por ejemplo, no puede estar la luz roja y la verde encendidas a la vez!!!!.

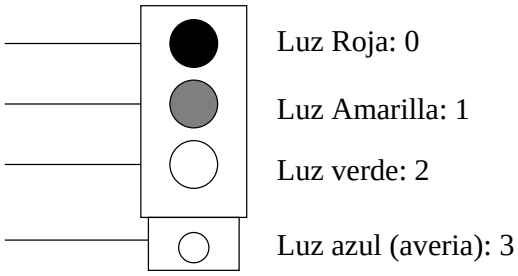


Figura 6.2: El semáforo que se quiere controlar

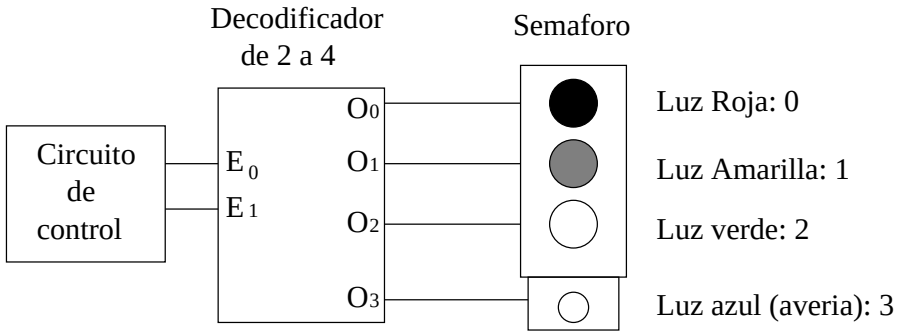


Figura 6.3: Circuito de control del semáforo, usando un decodificador de 2 a 4

Si utilizamos un decodificador de 2 a 4, conseguiremos controlar el semáforo asegurándonos que sólo estará activa una luz en cada momento. Además, el circuito de control que diseñemos sólo tienen que tener 2 salidas. El nuevo esquema se muestra en la figura 6.3.

El funcionamiento es muy sencillo. Si el circuito de control envía el número 2 ($E_1 = 1$, $E_0 = 0$), se encenderá la luz verde (que tiene asociado el número 2) y sólo la luz verde!!!. Un decodificador activa sólo una de las salidas, la salida que tiene un número igual al que se ha introducido por la entrada. En el ejemplo del semáforo, si el circuito de control envía el número 3, se activa la salida O_3 y se encenderá la luz azul (y sólo esa!!).

A la hora de diseñar el circuito de control, sólo hay que tener en cuenta que cada luz del semáforo está conectada a una salida del decodificador y que por tanto tiene asociado un número diferente.

6.3.2. Tablas de verdad y Ecuaciones

Decodificador de 2 a 4

Comenzaremos por el decodificador más sencillo, uno que tiene 2 entradas y 4 salidas, como se muestra en la figura 6.4.

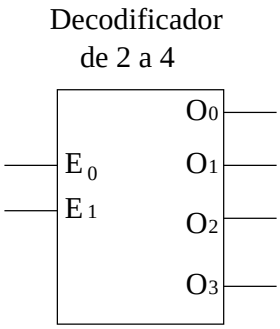


Figura 6.4: Un decodificador de 2 a 4

La tabla de verdad es la siguiente:

E_1	E_0	O_3	O_2	O_1	O_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

Y las ecuaciones las podemos obtener desarrollando por la primera forma canónica. Puesto que por cada función de salida sólo hay un '1', no se podrá simplificar (No hace falta que hagamos Karnaugh):

$$O_0 = \overline{\mathrm{E}_1} \cdot \overline{E_0}$$

$$O_1 = \overline{E_1} \cdot E_0$$

$$O_2 = E_1 \cdot \overline{E_0}$$

$$O_3 = E_1 \cdot E_0$$

La tabla de verdad la podemos expresar de forma abreviada de la siguiente manera, indicando la salida que se activa y sabiendo que las demás permanecerán desactivadas.

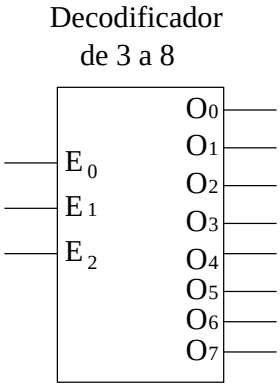


Figura 6.5: Un decodificador de 3 a 8

E_1	E_0	**Salida Activa**
0	0	O_0
0	1	O_1
1	0	O_2
1	1	O_3

Decodificicador de 3 a 8

Tiene 3 entradas y 8 salidas, como se muestra en la figura 6.5.

La tabla de verdad abreviada es la siguiente:

E_2	E_1	E_0	**Salida Activa**
0	0	0	O_0
0	0	1	O_1
0	1	0	O_2
0	1	1	O_3
1	0	0	O_4
1	0	1	O_5
1	1	0	O_6
1	1	1	O_7

Y las ecuaciones son: $O_0 = \overline{E_2} \cdot \overline{E_1} \cdot \overline{E_0}$, $O_1 = \overline{E_2} \cdot \overline{E_1} \cdot E_0$, ... ,$O_7 = E_2 \cdot E_1 \cdot E_0$.

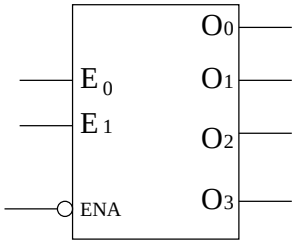


Figura 6.6: Un decodificador de 2 a 4, con entrada de validación activa a nivel bajo

6.3.3. Entradas de validación

Lo mismo que ocurría con los multiplexores y demultiplexores, existe una **entrada de validación** opcional. Si esta entrada está activada, el decodificador funciona normalmente, pero si está desactivada, sus salidas siempre estarán a '0'. Existen dos tipos de entrada de validación, las activas a nivel alto y las activas a nivel bajo.

En la figura 6.6 se muestra un decodificador de 2 a 4 con entrada de validación activa a nivel bajo, por lo el decodificador funcionará siempre que esta entrada esté a '0' y todas sus salidas permanecerán desactivadas cuando la entrada de validación esté a '1'.

Las ecuaciones de este decodificador irán multiplicadas por $\overline{ENA}$, siendo ENA la entrada de validación:

$$O_0 = \overline{\mathrm{E}_1} \cdot \overline{E_0} \cdot \overline{ENA}$$

$$O_1 = \overline{E_1} \cdot E_0 \cdot \overline{ENA}$$

$$O_2 = E_1 \cdot \overline{E_0} \cdot \overline{ENA}$$

$$O_3 = E_1 \cdot E_0 \cdot \overline{ENA}$$

Cuando por la entrada se introduce un '1' ($ENA = 1$), todas las salidas irán multiplicadas por $\overline{ENA}$, que vale '0' y todas ellas valdrán '0'. Si se introduce un '1', las ecuaciones serán las de un decodificador de 2 a 4.

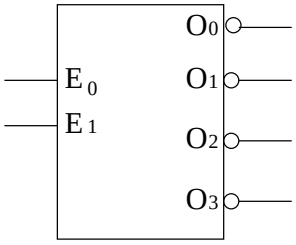


Figura 6.7: Un decodificador de 2 a 4 con salidas activas a nivel bajo

6.3.4. Tipos de decodificadores según sus salidas

Las salidas de los decodificadores pueden ser activas a nivel alto o a nivel bajo. Así, tendremos dos tipos: los **decodificadores con salidas activas a nivel alto** y los **decodificadores con salidas activas a nivel bajo**. Todos los que hemos visto hasta ahora son decodificadores activos a nivel alto, lo que quiere decir que si una salida está activa por ella sale un '1', y si está desactivada un '0'. Sin embargo, en los decodificadores con salidas activas a nivel bajo ocurre justo lo contrario.

En la figura 6.7 se muestra un decodificador de 2 a 4 con salidas a activas a nivel bajo.

La tabla de verdad completa es la siguiente:

E_1	E_0	$\overline{O_3}$	$\overline{O_2}$	$\overline{O_1}$	$\overline{O_0}$
0	0	1	1	1	**0**
0	1	1	1	**0**	1
1	0	1	**0**	1	1
1	1	**0**	1	1	1

6.4. Aplicaciones de los decodificadores

Además del uso normal de los decodificadores, como parte de nuestros diseños, existen otras aplicaciones que veremos a continuación.

6.4.1. Como Demultiplexor

Si examinamos las tablas de verdad, observamos que realmnte un decodificador con una entrada de validación se comporta como un demultiplexor. De hecho no existen circuitos integrados con demultiplexores, sino que se usan decodificadores. Imaginemos que necesitamos utilizar un demultiplexor de dos entradas de selección, como el mostrado en la figura XX.

Capítulo 7

CIRCUITOS ARITMETICOS

7.1. Introducción

7.2. Circuitos sumadores

7.2.1. Sumadores de números de 1 bit

Semisumador

Sumador total

7.2.2. Sumadores de números de más de 1 bit

Conexión de sumadores totales

Cuadruple sumador total

7.3. Circuitos restadores

7.3.1. Restador en ca1

7.3.2. Restador en ca2

7.4. Sumador/restador

7.4.1. En ca1

7.4.2. En ca2

7.5. Aplicación de los sumadores: transcodificadores

7.6. Resumen

7.7. Ejercicios

Capítulo 8

BIESTABLES

Capítulo 9

REGISTROS

Capítulo 10

CONTADORES

Capítulo 11

AUTOMATAS FINITOS

Capítulo 12

Solución a los ejercicios propuestos

12.1. Sistemas de representación

1. Pasar los siguientes números a decimal

   a) $347_8 = 3 \cdot 8^2 + 4 \cdot 8^1 + 7 \cdot 8^0 = 192 + 32 + 7 =$**231**

   b) $2201_3 = 2 \cdot 3^3 + 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 54$ + 18 + 1 = **73**

   c) $AF2_{16} = A \cdot 16^2 + F \cdot 16^1 + 2 \cdot 16^0 = 2560$ + 240 + 2 = **2802**

   d) $10111_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16$ + 4 + 2 + 1 = **23**

2. Pasar de binario a hexadecimal

   a) 0101101011111011 = 0101-1010-1111-1011 = 5-A-F-B = **5AFB**

   b) 10010001110000101 = 1-0010-0011-1000-0101 = 1-2-3-8-5 = **12385**

   c) 1111000011110000 = 1111-0000-1111-0000 = F-0-F-0 = **F0F0**

   d) 0101010110101010 = 0101-0101-1010-1010 = 5-5-A-A = **55AA**

3. Pasar de hexadecimal a binario

   a) FFFF = F-F-F-F = 1111-1111-1111-1111 = **1111111111111111**

   b) 01AC = 0-1-A-C = 0000-0001-1010-1100 = **0000000110101100**

   c) 55AA = 5-5-A-A = 0101-0101-1010-1010 = **0101010110101010**

   d) 3210 = 3-2-1-0 = 0011-0010-0001-0000 = **0011001000010000**

12.2. Algebra de Boole

Ejercicio 1:

En algunos ejercicios se explica entre llaves ({}) los pasos que se han seguido.

Realizar las siguientes operaciones:

1. 1 + 0 = **1** (Por la *definición* del operador booleano +)
2. 1 + 1 = **1** (Por la *definición* del operador booleano +)
3. 1.0 = **0** (Por la *definición* del operador booleano .)
4. 1·1 = **1** (Por la *definición* del operador booleano .)
5. A+0 =**A** (0 es el *elemento neutro* de la operación booleana +)
6. A+1= **1** (Por la *definición* del operador booleano +)
7. A·1= **A** (1 es el *elemento neutro* de la operación booleana .)
8. A·0= **0** (Por la definición del operador booleano .)
9. A+A= **A** (Propiedad de *Idempotencia* de la operación booleana +)
10. A·A= **A** (Propiedad de *Idempotencia* de la operación booleana .)
11. A+$\overline{A}$= **1** (*Elemento inverso*)
12. A·$\overline{A}$= **0** (*Elemento inverso*)
13. A+AB = {*Sacando factor común A*} = A(1+B) = {*B+1=1*} = **A** (También se conoce como *ley de absorción*).
14. A(A+B) = {*Propiedad distributica*} = AA + AB = {*AA=A*} = A + AB = { *Por el resultado anterior* } = **A**. (También se conoce como *ley de absorción*).
15. A+AB+B = {*Sacando factor común en A*} = A(1+B) + B = {1+B=1} = **A + B**. También se podría haber aplicado a la expresión inicial la ley de absorción: A+AB = A.

Ejercicio 2:

Aplicar las leyes de Morgan en los siguientes casos:

1. $\overline{A(B+C)}$= {Aplicando Morgan a ambos términos del producto} = $\overline{A}+\overline{B+C}$ = {Aplicando Morgan al segundo sumando} = $\overline{A}+\overline{B}\cdot\overline{C}$
2. $\overline{\overline{AB+CD}\cdot E}$= $\overline{\overline{AB+CD}}+\overline{E}$ = {$\overline{\overline{AB+CD}}=AB+CD$} = AB+CD+$\overline{E}$ (también se puede aplicar Morgan primero al termino $\overline{AB+CD}$, pero hay que dar más pasos para llegar al final).
3. $\overline{(AB+CD)\cdot E}$= $\overline{AB+CD}+\overline{E}$ = $\overline{AB}\cdot\overline{CD}+\overline{E}$= **(**$\overline{A}+\overline{B}$**)**·**(**$\overline{C}+\overline{D}$**)**+$\overline{E}$

Ejercicio 3:

Obtener el valor de las siguientes funciones booleanas, en todos los casos.

1. $F=A+B$

   Como hay 4 variables, tenemos 4 casos posibles:

   a) A=0, B=0 $\Rightarrow F(0,0)=0+0=0$

   b) A=0, B=1 $\Rightarrow F(0,1)=0+1=1$

   c) A=1, B=0 $\Rightarrow F(1,0)=1+0=1$

   d) A=1, B=1 $\Rightarrow F(1,1)=1+1=1$

2. $F=A+\overline{B}$

   También hay 4 casos posibles:

   a) A=0, B=0 $\Rightarrow F(0,0)=0+\overline{0}=0+1=1$

   b) A=0, B=1 $\Rightarrow F(0,1)=0+\overline{1}=0+0=0$

   c) A=1, B=0 $\Rightarrow F(1,0)=1+\overline{0}=1+1=1$

   d) A=1, B=1 $\Rightarrow F(1,1)=1+\overline{1}=1+0=1$

3. F=$\overline{A}\cdot B+C$

   Tenemos todos los siguientes casos:

a) A=0,B=0, C=0 $\Rightarrow F(0,0,0) = \overline{0} \cdot 0 + 0 = 1 \cdot 0 = 0$

b) A=0, B=0, C=1 $\Rightarrow F(0,0,1) = \overline{0} \cdot 0 + 1 = 1 \cdot 0 + 1 = 0 + 1 = 1$

c) A=0, B=1, C=0 $\Rightarrow F(0,1,0) = \overline{0} \cdot 1 + 0 = 1 \cdot 1 + 0 = 1$

d) A=0, B=1, C=1 $\Rightarrow F(0,1,1) = \overline{0} \cdot 1 + 1 = 1 \cdot 1 + 1 = 1$

e) A=1, B=0, C=0 $\Rightarrow F(1,0,0) = \overline{1} \cdot 0 + 0 = 0 \cdot 0 = 0$

f) A=1, B=0, C=1 $\Rightarrow F(1,0,1) = \overline{1} \cdot 0 + 1 = 0 \cdot 0 + 1 = 1$

g) A=1, B=1, C=0 $\Rightarrow F(1,1,0) = \overline{1} \cdot 1 + 0 = 0 \cdot 1 = 0$

h) A=1, B=1, C=1 $\Rightarrow F(1,1,1) = \overline{1} \cdot 1 + 1 = 0 \cdot 1 + 1 = 0 + 1 = 1$

Ejercicio 4:

Dadas las siguientes funciones booleanas, obtener su correspondiente tabla de verdad.

Para resolver este tipo de ejercicios resulta cómo colocar nuevas columnas con resultados intermedios.

1. $F = A + \overline{B}$

   Función de 2 variables. La tabla tiene 4 filas.

A	B	$\overline{B}$	F
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

2. $G = A \cdot B + \overline{A} \cdot B$

   Función de 2 variables. La tabla tiene 4 filas.

A	B	$A \cdot B$	$\overline{A}$	$\overline{A} \cdot B$	F
0	0	0	1	0	0
0	1	0	1	1	1
1	0	0	0	0	0
1	1	1	0	0	1

3. $H = X \cdot Y \cdot \overline{Z} + \overline{X} \cdot \overline{Y} \cdot Z$

   Función de 3 variables. La tabla tiene 8 filas

X	Y	Z	$\overline{X}$	$\overline{Y}$	$\overline{Z}$	$X \cdot Y \cdot \overline{Z}$	$\overline{X} \cdot \overline{Y} \cdot Z$	F
0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0	0
1	1	0	0	0	1	1	0	1
1	1	1	0	0	0	0	0	0

4. $S = E_3E_2E_1E_0 + E_3\overline{E_2}$

   Función de 4 variables. La tabla tiene 16 filas

E_3	E_2	E_1	E_0	$\overline{E_2}$	$E_3E_2E_1E_0$	$E_3\overline{E_2}$	S
0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	1	0	1	1
1	0	1	1	1	0	1	1
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	1	0	0

Ejercicio 5:

Desarrollar las siguientes tablas de verdad por la primera forma canónica:

1. Tabla 1:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

Como la función tiene dos “unos”, será la suma de dos términos:

$$F = \overline{A} \cdot B + A \cdot B$$

2. Tabla 2:

A	**B**	**C**	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

La función tiene tres “unos”, será la suma de tres términos:

$$F = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C}$$

Ejercicio 6:

Dadas las siguientes funciones, indicar si se encuentra expresadas en la primera forma canónica, y si es así, obtener la tabla de verdad

1. $F = \overline{A} \cdot B + A \cdot B$

   Sí se encuentra en la primera forma canónica, puesto que es una suma de productos, y en cada sumando se encuentran todas las variables. En la tabla de verdad F valdrá ’1’ cuando A=0, B=1 y A=1, B=1:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

2. $F = A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$

   También está en la primera forma canónica. En este caso la función es de tres variables:

A	**B**	**C**	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

3. $F = E_2 \cdot E_1 \cdot E_0 + \overline{E_2} \cdot E_1 \cdot E_0 + E_1$

   Esta función NO está en la primera forma canónica. En el último sumando no aparecen todas las variables.

4. $F = E_2 \cdot E_1 \cdot E_0 + \overline{E_2} \cdot E_1 \cdot E_0 + \overline{E_2} \cdot \overline{E_1} \cdot E_0$

   Esta sí lo está:

E_2	E_1	E_0	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Ejercicio 7:

Desarrollar las siguientes tablas de verdad por la segunda forma canónica:

1. Tabla 1:

A	**B**	F
0	0	0
0	1	1
1	0	0
1	1	1

   Nos fijamos en las filas en las que F=0 y obtenemos el producto de sumas, con el criterio de que si una variable está a 1 usaremos su negada y que si está a '0' usaremos esa misma variable:

$$F = (A + B) \cdot (\overline{A} + B)$$

2. Tabla 2:

A	**B**	**C**	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

En este caso la función es de tres variables y hay cuatro filas en las que F=0, por tanto tendrá cuatro términos que van multiplicados:

$$F = (A + \overline{B} + C) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{B} + \overline{C})$$

Ejercicio 8:

Dadas las siguientes funciones, indicar si se encuentra expresadas en la primera forma canónica o en la segunda. En caso de que así sea, obtener la tabla de verdad.

1. $F = (A + B) \cdot (\overline{A} + \overline{B})$

   Está en la segunda forma canónica, puesto que es un producto de sumas, y en todos los términos se encuentran las dos variables. Para construir la tabla de verdad tenemos que tener en cuenta que en las filas en las que A=0, B=0 y A=1,B=1 la función vale '0', y para el resto de filas vale '1':

A	**B**	F
0	0	0
0	1	1
1	0	1
1	1	0

2. $F = \overline{A} \cdot B + \overline{B} \cdot A$

   Se encuentra en la primera forma canónica puesto que es una suma de productos y en cada una de las sumas se encuentran las dos variables. En la tabla de verdad, en las filas en las que A=0,B=1 y A=1,B=0 la función vale '1', y en el resto de filas valdrá '0'

A	**B**	F
0	0	0
0	1	1
1	0	1
1	1	0

3. $F = (E_2 + \overline{E_1} + E_0) \cdot (\overline{E_2} + \overline{E_1} + E_0) \cdot (E_2 + E_1 + E_0)$

   Se encuentra en la segunda forma canónica. La tabla de verdad es:

E_2	E_1	E_0	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

4. $F = E_2 \cdot \overline{E_1} \cdot E_0 + \overline{E_2} \cdot \overline{E_1} \cdot E_0 + E_2 \cdot E_1 \cdot E_0$

   Está en la primera forma canónica. La tabla de verdad es:

E_2	E_1	E_0	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

5. $F = (A \cdot B \cdot C) + (A + B + C)$

   NO está en ninguna forma canónica.

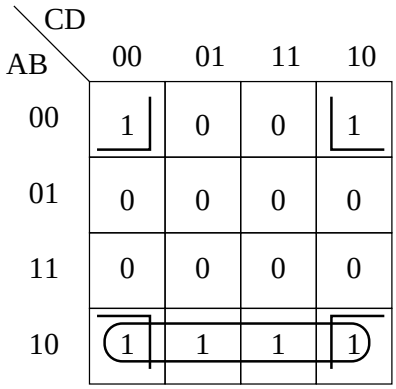
Ejercicio 9:

Obtener las expresiones más simplificadas a partir de las tablas de verdad:

1. Tabla 1:

A	**B**	**C**	**D**	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

El diagrama de Karnaugh es:



y la función es:

$$F = \overline{B} \cdot \overline{D} + A \cdot \overline{B}$$

2. Tabla 2:

E_3	E_2	E_1	E_0	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

El diagrama de Karnaugh es:

E_3E_2 \ E_1E_0	00	01	11	10
00	1	0	0	0
01	1	0	1	1
11	0	1	0	0
10	0	0	0	0

y la expresión de F:

$$F = \overline{E_3} \cdot \overline{E_1} \cdot \overline{E_0} + \overline{E_3} \cdot E_2 \cdot E_1 + E_3 \cdot E_2 \cdot \overline{E_1} \cdot E_0$$

Ejercicio 10:

Operar con las siguientes expresiones obteniendo la mayor cantidad posible de operaciones $\oplus$

1. $A \cdot \overline{B} + \overline{A} \cdot B$ = {Definición de la opereración $\oplus$} = $A \oplus B$
2. $A \cdot B + \overline{A} \cdot \overline{B}$= {Definición de la operacion XOR negada} = $\overline{A \oplus B}$
3. $(A \cdot \overline{B} + \overline{A} \cdot B) \cdot \overline{C} + \overline{(A\overline{B} + \overline{A} \cdot B)} \cdot C$={Aplicando la definición de la operación A$\oplus$B} = (A$\oplus$B).$\overline{C}$ + $\overline{(A \oplus B)}$.C = {Aplicando nuevamente la definición de la operación $\oplus$} = **A $\oplus$B$\oplus$C**
4. $\overline{A} \cdot B + \overline{A} \oplus B + \overline{A \oplus \overline{B}} + A \cdot \overline{B}$ ={Aplicando la propiedad $\overline{A \oplus B} = A \oplus \overline{B}$}=$\overline{A} \cdot B + \overline{A} \oplus B + A \oplus \overline{\overline{B}} + A \cdot \overline{B}$ ={Como $\overline{\overline{B}} = B$} = $\overline{A} \cdot B + \overline{A} \oplus B + A \oplus B + A \cdot \overline{B}$={Aplicando $\overline{A \oplus B} = \overline{A} \oplus B$}=$\overline{A} \cdot B + \overline{A \oplus B} + A \oplus B + A \cdot \overline{B}$={Aplicando que $\overline{A \oplus B} + A \oplus B = 1$}=$\overline{A} \cdot B + A \cdot \overline{B}$={Por la definición de $\oplus$}=$A \oplus B$

Ejercicio 11:

Dejar las siguientes expresiones en forma de sumas de productos:

1. (x + y + z)($\overline{x}$+z) = {Aplicando propiedad distributiva} = x$\overline{x} + y\overline{x} + z\overline{x} + xz + yz + zz$ = $\left\{ \begin{array}{c} zz = z \\ x\overline{x} = 0 \end{array} \right\}$ = y$\overline{x}$+$z\overline{x}$+xz+yz+z = {Sacando factor común en z} = y$\overline{x}$+$z(\overline{x}+x+y+1)$ = {Algo + 1 = 1} = **y**$\overline{x} + z$
2. $\overline{(\overline{x} + y + z)}.(\overline{y} + z)$ ={Aplicando Morgan al primer término} = $\overline{\overline{x}}.\overline{y}.\overline{z}.(\overline{y} + z)$ = {$\overline{\overline{x}} = x$} = x.$\overline{y}.\overline{z}.(\overline{y} + z)$ = {Prop. distributiva} = x.$\overline{y}.\overline{z}.\overline{y} + x.\overline{y}.\overline{z}z$ = $\left\{ \begin{array}{c} \overline{z}.z = 0 \\ \overline{y}.\overline{y} = \overline{y} \end{array} \right\}$ = **x.**$\overline{y}.\overline{z}$
3. $\overline{x\overline{y}z}.\overline{\overline{x}yz}$ ={Aplicando Morgan en términos $\overline{x.\overline{y}.z}$, $\overline{\overline{x}.y.z}$} = $(\overline{x} + \overline{\overline{y}} + \overline{z}).(\overline{\overline{x}} + \overline{y} + \overline{z})$ = $\left\{ \begin{array}{c} \overline{\overline{x}} = x \\ \overline{\overline{y}} = y \end{array} \right\}$ = $(\overline{x}+y+\overline{z}).(x+\overline{y}+\overline{z})$ = {Prop. distributiva} = $\overline{x}x + yx + \overline{z}.x + \overline{x}.\overline{y} + y\overline{y} + \overline{z}.\overline{z}$ = $\left\{ \begin{array}{c} \overline{x}.x = 0 \\ \overline{z}.\overline{z} = \overline{z} \\ y.\overline{y} = 0 \end{array} \right\}$ = yx + $\overline{z}x + \overline{x}.\overline{y} + \overline{z}$ = {Ley absorción: $\overline{z}x + \overline{z} = \overline{z}$} = **yx + $\overline{z}$+$\overline{x}.\overline{y}$**

Bibliografía

[1] Licencia GFDL. http://www.gnu.org/copyleft/fdl.es.html

[2] Web de la UPSAM. http://www.upsam.com/