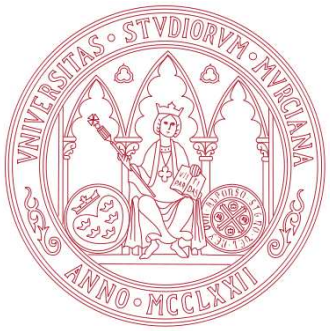

Universidad de Murcia

Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Fundamentos de Computadores

Tema 3: Sistemas Digitales: Circuitos Combinacionales

Boletines de prácticas

CURSO 2020 / 21

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Índice general

Boletines de prácticas

B3.1. Boletín 1: Diseño y simulación de un sumador de n bits

B3.1.1. Objetivos

Los objetivos de esta sesión son que el alumno comprenda todas las etapas necesarias para diseñar un circuito combinacional sencillo de ejemplo a partir de una especificación funcional determinada. En particular, se abordará el diseño incremental de un sencillo circuito capaz de sumar dos números expresados como sendas secuencias de bits interpretadas en binario natural. Finalmente, y una vez diseñado sobre el papel, se utilizará el simulador TkGate para comprobar el correcto funcionamiento de dicho circuito.

B3.1.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura previa del boletín por parte del alumno.
2. Diseño de un circuito semisumador de 1 bit.
3. Diseño de un circuito sumador completo de 1 bit.
4. Diseño de un circuito sumador de n bits.
5. Simulación del circuito diseñado sobre TkGate para comprobar su funcionamiento.

B3.1.3. Diseño modular de un circuito sumador de n bits

En esta sección se describe un primer diseño digital combinacional de carácter eminentemente práctico, el cual abordaremos de forma gradual. El objetivo es crear un circuito sumador binario de n bits, capaz de sumar dos ristras de n bits que representan a sendos números codificados en binario natural. Dicho circuito, además, podrá ser considerado también como capaz de restar puesto que, como sabemos desde el tema 2, realizando la suma aritmética de dos ristras arbitrarias de bits convencionales, si éstas se interpretan en C2 (para representar enteros positivos o negativos, en lugar de interpretarlas en binario natural), el resultado será entonces también correcto al ser interpretado en ese mismo C2, independientemente de que éste pueda ser positivo o negativo.

La forma más inmediata de plantearnos la implementación de un sumador con lógica digital es reproducir las mismas operaciones que realizamos "a mano" a la hora de calcular una suma en binario. El algoritmo que utilizaremos, pues, se basa en las operaciones que hemos de afrontar al realizar la suma con "lápiz y papel". Puesto que este procedimiento se va repitiendo bit a bit, realizaremos el diseño de forma modular, diseñando primero un circuito elemental capaz de sumar dos bits de entrada, junto con un posible acarreo que podría venir de la suma anterior. Esto es, implementaremos primero un circuito que realice la suma de dos dígitos binarios, correspondientes a dos dígitos del mismo peso de los números a sumar, siendo en todo momento conscientes de que al resultado de esta suma habrá que añadir también el posible acarreo generado (o *propagado*) por la suma de los 2 dígitos de peso inmediatamente anterior.

B3.1.3.1. Circuito semisumador de 1 bit

Comenzaremos nuestro diseño incremental, pues, diseñando primero un sencillo circuito capaz de sumar simplemente dos bits. Dicho circuito elemental será utilizado posteriormente de forma repetida para sumar cada pareja de bits del mismo peso en las ristras de entrada. Dado que la suma aritmética de dos bits puede generar un acarreo, necesitamos ser capaces no sólo de generar el bit de suma, sino también de detectar la posible generación de un acarreo.

Aparece de esta forma el circuito llamado *semisumador*, que podría implementarse de la manera indicada en la figura I.1.

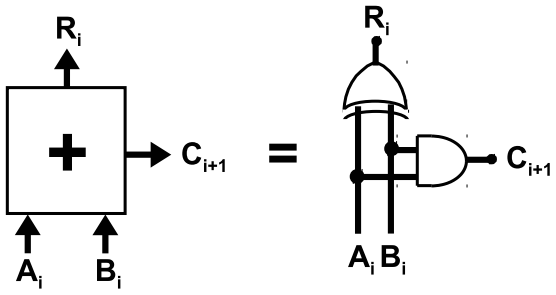

Figura I.1: Circuito semisumador.

En dicho circuito observamos a simple vista que *a)* el resultado de la suma de dos bits coincide con el *or exclusivo (XOR)* de los dos bits de entrada, es decir, el bit de resultado R_i será uno siempre que uno, y solo uno, de los bits de entrada A_i y B_i sea uno; y *b)* que se producirá un acarreo de salida C_{i+1} sólo en el caso de que ambos bits de entrada A_i y B_i sean uno, es decir, dicho bit de acarreo coincide con el *and lógico (AND)* de dichos bits.

B3.1.3.2. Circuito sumador completo de 1 bit usando dos semisumadores

Si usamos ahora el anterior semisumador para, además, sumar el resultado de esa suma con el acarreo entrante (acarreo que se formó en pesos anteriores) y para detectar la posibilidad de que se genere un acarreo de salida porque entra un acarreo de entrada C_i y el resultado de la suma de los dos dígitos A_i y B_i es 1, el circuito sumador completo de dos bits más un posible acarreo de entrada sería el indicado en la figura I.2.

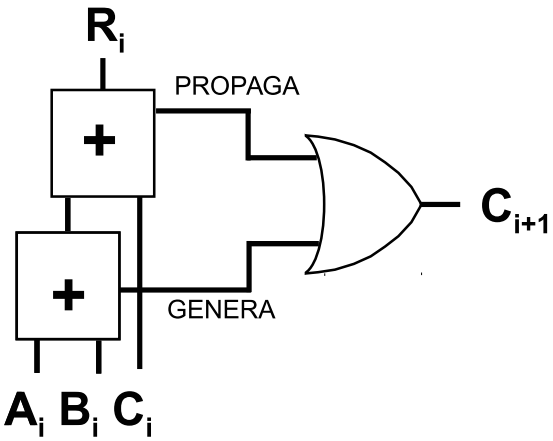

Figura I.2: Circuito sumador completo.

El lector ya habrá intuído que estas operaciones (sumar dos bits del mismo peso, sumarles el posible acarreo entrante, y detectar si se genera el acarreo saliente o se propaga el que entra) son operaciones que cuando realizamos la suma con "lápiz y papel" se repiten para cada peso de bits de nuestro par de números binarios. Por tanto podríamos repetir estos circuitos de la figura anterior tantas veces como número máximo de dígitos tengan las palabras a sumar. El diseño, pues, puede realizarse claramente de forma modular. ¿Cómo? Concatenando los módulos sumadores de 1 bit en serie de la manera indicada en la figura I.3.

Es lo que en sistemas digitales se conoce como *sumador serie*, *sumador en cascada* o *sumador con propagación de acarreo*. Por supuesto, el diseño abordado aquí, aunque funciona perfectamente, es algo ingenuo, sobre todo de cara al retardo que puede llevar asociado este tipo de conexión en cascada. Piénsese en el retardo asociado al sumar, por ejemplo, dos números de 64 bits, en el que para que la salida de cada sumador de un bit se estabilice con los

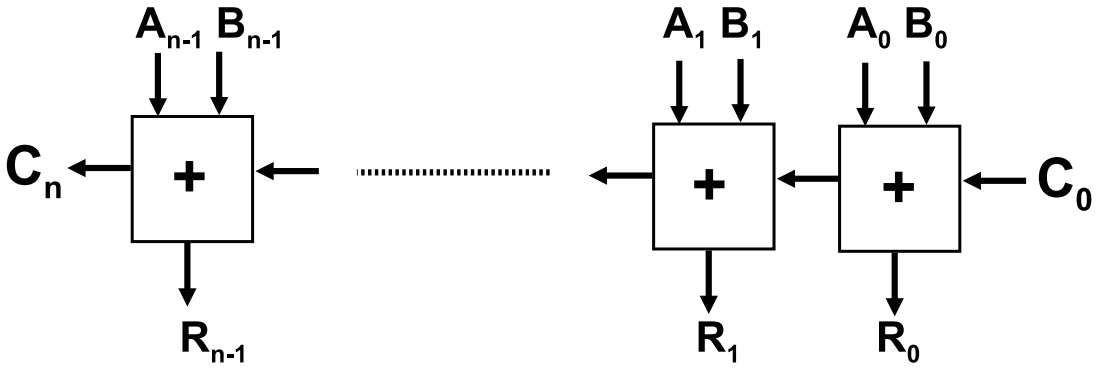

Figura I.3: Circuito sumador serie para n bits.

valores de salida correctos, tiene que estar estabilizada la entrada del acarreo anterior, lo que lleva a la acumulación de los retardos de los 64 sumadores individuales. Para solucionar este problema, los sumadores reales implementados en los procesadores actuales se apoyan en diseños más eficientes, en cuanto que necesitan menos tiempo para producir la suma. Son los llamados *sumadores con acarreo adelantado* (*Carry Lookahead Adder*) que, sin embargo, no estudiaremos aquí por ser notablemente más complejos.

B3.1.3.3. Diseños alternativos para un semisumador de 1 bit

También es posible apoyarnos en módulos individuales para la suma de cada bit ligeramente distintos. Por ejemplo, en la figura I.4 se muestran unos cuantos diseños alternativos para semisumadores, siguiendo siempre con la misma filosofía de "lápiz y papel". Cada cual tiene sus ventajas o limitaciones diseño. Por ejemplo, el último de los diseños, que utiliza sólo puertas NAND, es particularmente sencillo de implementar al nivel de transistores, aunque por motivos que en principio quedan fuera del alcance de los objetivos de esta asignatura.

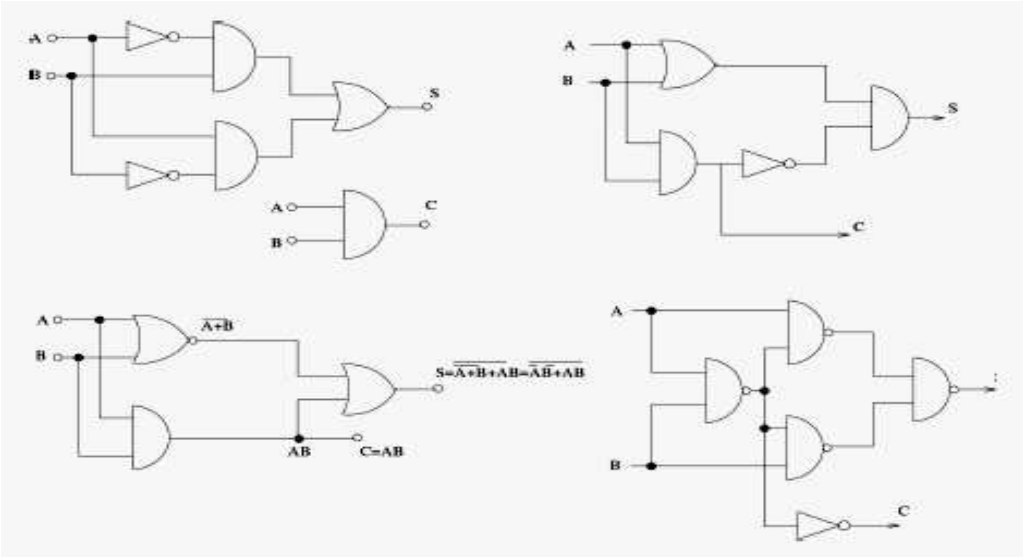

Figura I.4: Circuitos semisumadores alternativos.

B3.1.3.4. Diseño directo de un sumador de 1 bit

Otra opción alternativa al uso de semisumadores es abordar la implementación del sumador completo directamente, partiendo simplemente de la tabla de verdad de la suma de los dos bits de entrada, A_i y B_i, más el acarreo de

entrada, C_i, para obtener dos funciones lógicas como salida, la de los bits resultado R_i y el bit de acarreo C_{i+1}. La tabla de verdad asociada será entonces la indicada en la figura I.5.

A_i	B_i	C_i	R_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figura I.5: Tabla de verdad para un sumador completo de 1 bit.

Si pasamos esta tabla de verdad a sendos mapas de Karnaugh (uno para cada función) nos quedan los mapas indicados en la figura I.6:

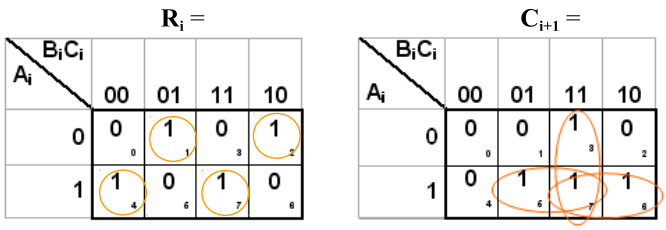

Figura I.6: Mapas de Karnaugh correspondientes a la tabla de verdad de la figura I.5.

Son todos implicantes primos esenciales, por lo que las funciones ya simplificadas, expresadas como suma de productos, quedarían así:

$$R_i = \overline{A}_i \cdot \overline{B}_i \cdot C_i + \overline{A}_i \cdot B_i \cdot \overline{C}_i + A_i \cdot B_i \cdot C_i + A_i \cdot \overline{B}_i \cdot \overline{C}_i.$$

$$C_{i+1} = A_i \cdot B_i + A_i \cdot C_i + B_i \cdot C_i$$

A la hora de implementarlo con puertas lógicas los correspondientes circuitos combinacionales quedarían como se indica en la figura I.7

B3.1.4. Simulación del circuito

A continuación probaremos en el entorno de diseño y simulación de circuitos digitales TkGate del diseño anteriormente realizado, capaz de sumar dos ristras de, en este caso concreto, 4 bits. Para ello, seguiremos los siguientes pasos:

1. Descargaremos del aula virtual el circuito `sumador.v`, y lo cargaremos en el entorno de simulación (Figura I.8).

2. Podemos comenzar analizando la estructura modular de dicho circuito. Si en la parte izquierda (vista *Árbol*) pulsamos en la etiqueta *sumador4bits*, se mostrará el contenido interior de dicho circuito (Figura I.9). En dicho esquema podemos comprobar la composición en cascada de los cuatro circuitos elementales sumadores de 1 bit. Se aprecian también claramente todas las entradas del módulo (bits a_0-a_3 y b_0-b_3), así como las salidas c_0-c_3 y el correspondiente acarreo de salida c_{out}. Obsérvese que, adicionalmente, el acarreo de entrada cin del primer módulo se encuentra conectado directamente a tierra, puesto que el acarreo de entrada al bit de menos peso deberá ser siempre cero.

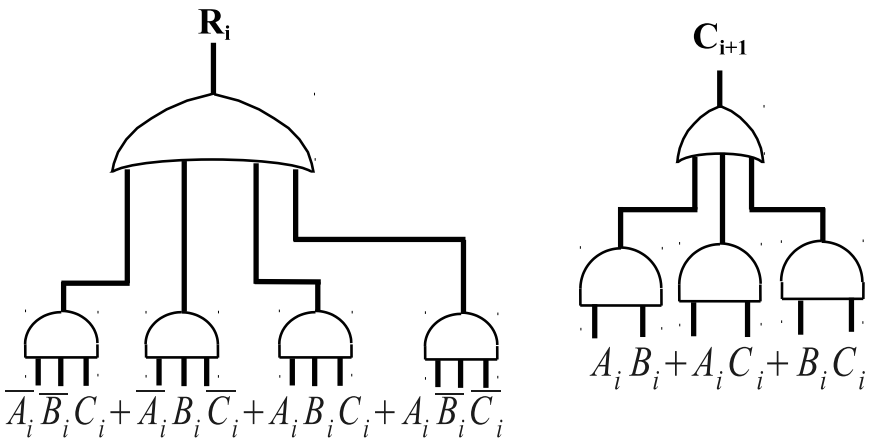

Figura I.7: Circuitos correspondientes a las salidas del sumador completo de 1 bit.

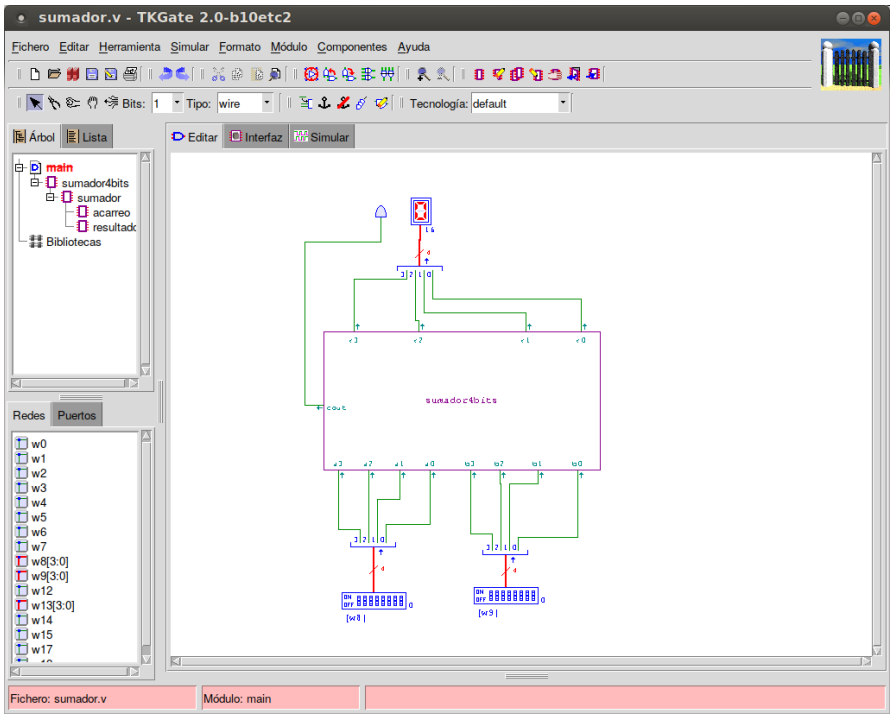

Figura I.8: TkGate mostrando el archivo `sumador.v` recién cargado.

3. Continuando con el análisis de módulos del circuito, pulsamos ahora sobre la etiqueta *sumador* de la vista de *Árbol*. Accedemos con ello al circuito sumador elemental de 1 bit (Figura I.10). De nuevo, en él se observan claramente sus tres bits de entrada (bits a y b, con los bits a sumar en una posición dada, y c, correspondiente al acarreo de entrada), así como sus dos bits de salida (el bit de resultado r y el acarreo de salida producido en esta posición, c_i). Así mismo, se observa que, internamente, este circuito está simplemente formado por dos módulos completamente independientes, correspondientes a los circuitos *acarreo* y *resultado*.

4. Para terminar la inspección del circuito, examinaremos finalmente estos circuitos *acarreo* y *resultado* (Figuras I.11 y I.12). Vemos que en su interior se encuentran ya los diseños a nivel de puertas lógicas elementales que desarrollamos sobre el papel en la subsección *"Diseño directo de un sumador de 1 bit"* anterior.

5. Volviendo al circuito global, *main* (Figura I.8), podemos comprobar que sus dos entradas de cuatro bits se encuentran conectadas a sendos dispositivos de entrada que nos permitirán codificar otras tantas ristras de 4 bits

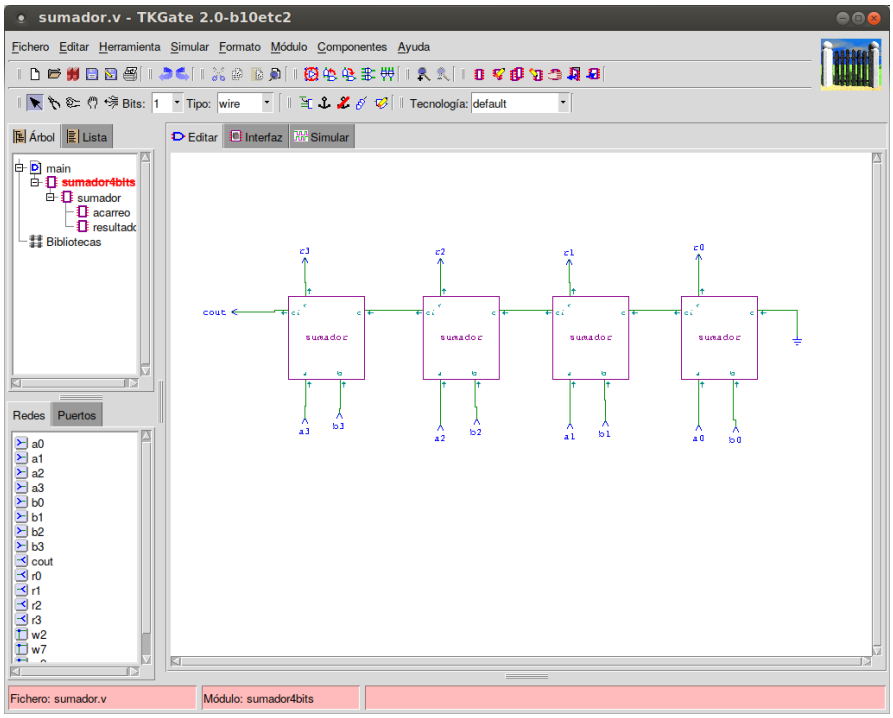

Figura I.9: Circuito *sumador4bits*.

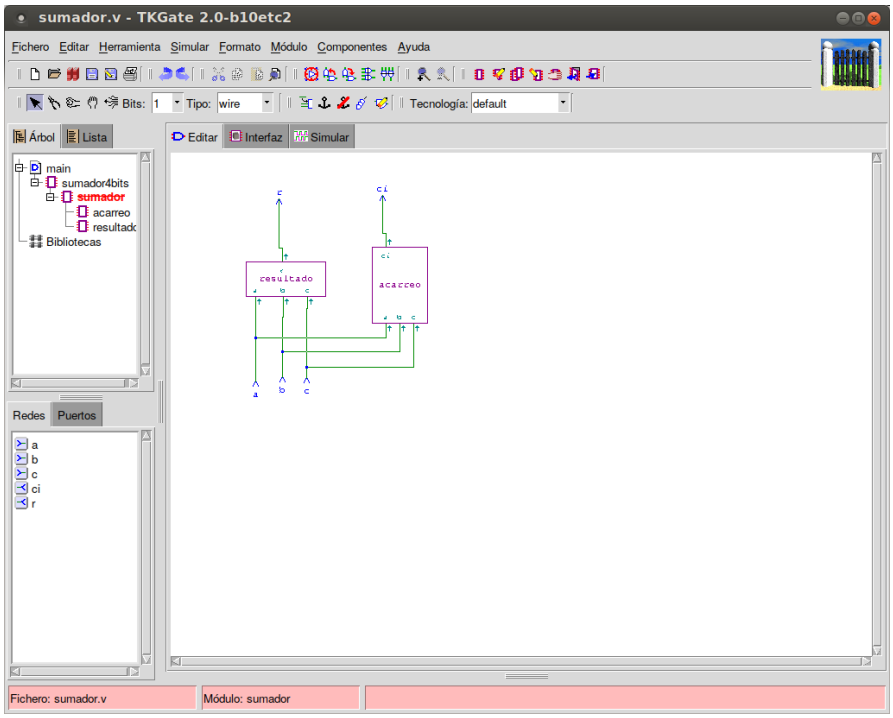

Figura I.10: Circuito *sumador*.

en forma de dígito hexadecimal (0-F, para las ristras de bits 0000 a 1111, respectivamente). De igual forma, los 4 bits de salida se encuentran conectados a un *display* de 7 segmentos, que mostrará el resultado igualmente en hexadecimal. Además, el acarreo de salida está igualmente conectado a un LED de salida, para poder observar fácilmente si éste se ha activado o no. Dicho LED, pues, deberá encenderse únicamente cuando el resultado de la suma sea mayor que 15.

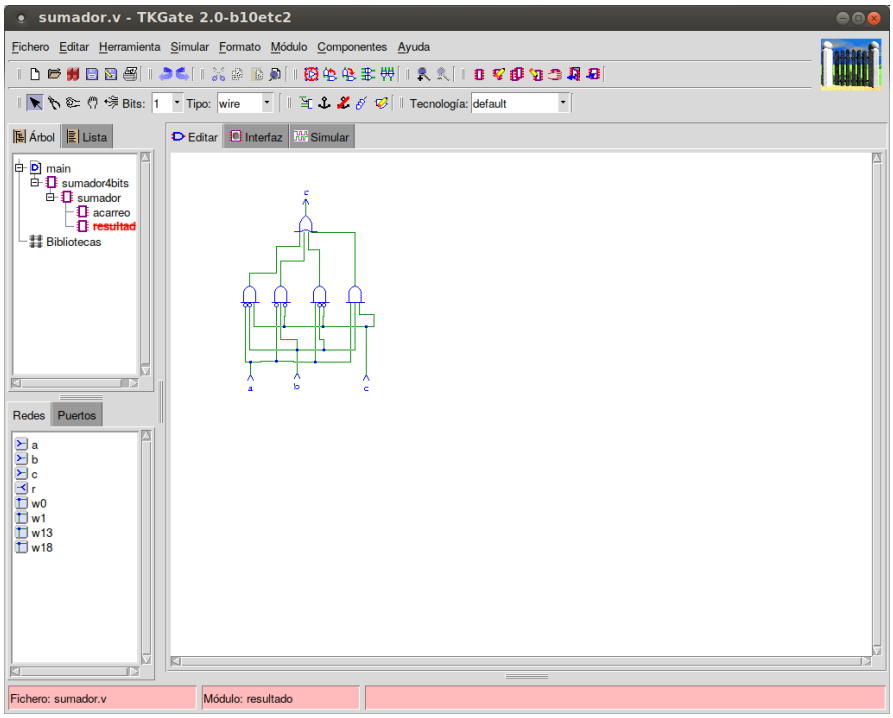

Figura I.11: Circuito *resultado*.

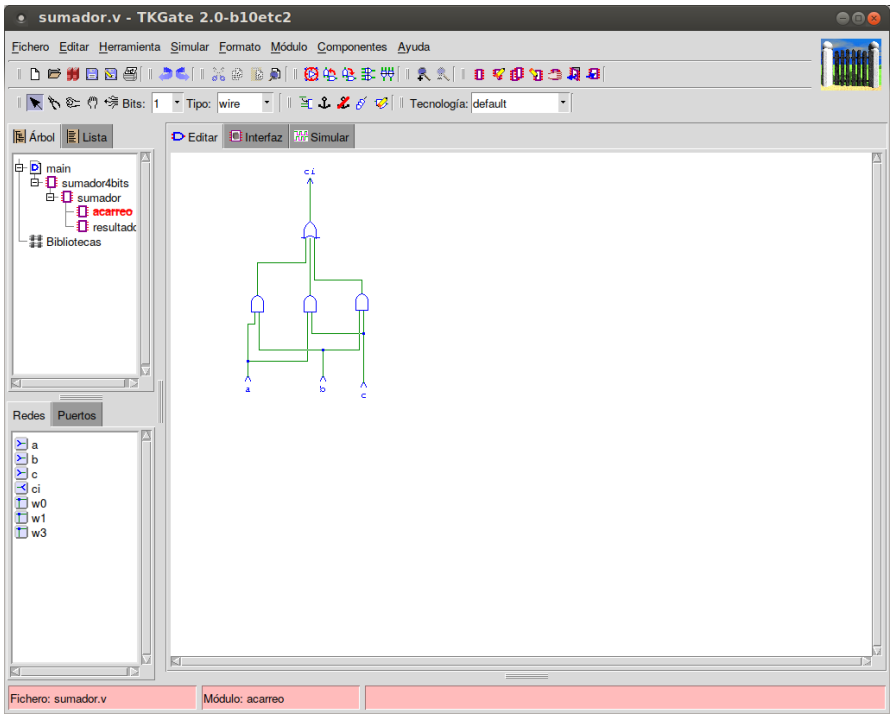

Figura I.12: Circuito *acarreo*.

6. Llegados a este punto podemos finalmente pasar a simular el circuito. Para ello clicamos en la pestaña de simulación, pulsamos el icono “play”, y hacemos clic derecho sobre los interruptores DIP para obtener una ventana de diálogo que nos permitirá gobernarlos. Cada vez que modificamos sus valores y pulsamos en aplicar se para la simulación al alcanzar la estabilidad las salidas, pudiendo pasar a comprobar los resultados. Si vamos probando entonces diferentes combinaciones de las entradas, podremos comprobar en todo momento la corrección de

las salidas obtenidas, lo que corrobora el correcto diseño del circuito.

B3.2. Boletín 2: Diseño y simulación de un decodificador para un display de 7 segmentos

B3.2.1. Objetivos

El objetivo principal de esta sesión es que el alumno sea capaz de crear y simular distintos circuitos sencillos a partir de sus funciones lógicas expresadas como tablas de verdad, para implementarlos después mediante distintas técnicas. En concreto, las técnicas de implementación que usaremos serán la *suma de productos*, el *producto de sumas*, mediante el *uso único de puertas NAND* y *NOR*, respectivamente, y finalmente mediante la ayuda de *multiplexores*. El diseño final servirá para implementar un decodificador BCD para usar con un display exhibidor de números de 7 segmentos luminosos. Al igual que en la sesión anterior, finalmente simularemos mediante TkGate todos los circuitos diseñados trabajando de forma conjunta, verificando así la correcta implementación de éstos.

B3.2.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura previa del boletín por parte del alumno.
2. Diseño del bit de salida a mediante la técnica de suma de productos.
3. Diseño del bit de salida b mediante la técnica de producto de sumas.
4. Diseño del bit de salida c usando sólo puertas NAND.
5. Diseño del bit de salida d usando sólo puertas NOR.
6. Diseño del bit de salida e usando un multiplexor de 3 entradas de control (más un inversor).
7. Simulación del circuito completo diseñado sobre TkGate para comprobar su funcionamiento.

B3.2.3. Diseño de un exhibidor BCD de 7 segmentos

Se desea implementar un decodificador para un "display" de 7 segmentos (ver figura I.13) capaz de representar los dígitos decimales (del 0 al 9), el signo menos (-) y un signo de error (la letra E mayúscula). Tal decodificador tiene como entrada 4 bits que son usados para codificar el carácter decimal (BCD, o *Binary Coded Decimal*; se corresponde con los valores 0 al 9 expresados en binario natural de 4 bits, es decir, desde 0000 hasta 1001, ambos inclusive), el signo de error (que asociaremos con la entrada de bits 1110) y el signo menos (entrada 1111). Las siete salidas de dicho decodificador, que alimentan cada uno de los segmentos del display, serán funciones lógicas de las anteriores cuatro entradas. El funcionamiento deseado queda completamente especificado en la tabla contenida en la figura I.14.

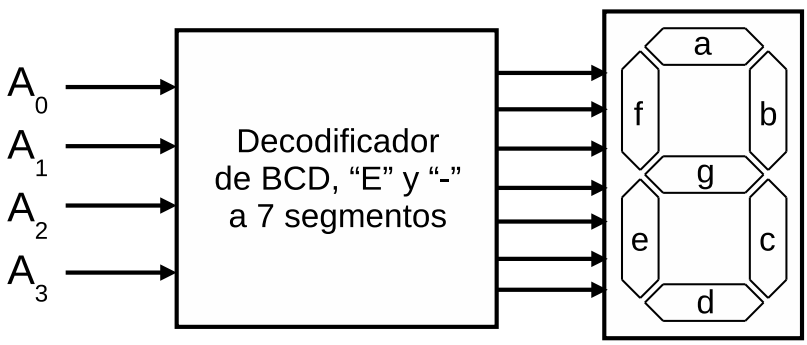

Figura I.13: Exhibidor BCD de 7 segmentos.

Nº	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g	**Carácter exhibido**
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	
A (Indet.)	1	0	1	0	-	-	-	-	-	-	-	
B (Indet.)	1	0	1	1	-	-	-	-	-	-	-	
C (Indet.)	1	1	0	0	-	-	-	-	-	-	-	
D (Indet.)	1	1	0	1	-	-	-	-	-	-	-	
E (Error)	1	1	1	0	1	0	0	1	1	1	1	
F (Menos)	1	1	1	1	0	0	0	0	0	0	1	

Figura I.14: Tabla de verdad del exhibidor BCD de 7 segmentos.

En la tabla se ve que los valores hexadecimales de entrada A, B, C y D (binario 1010, 1011, 1100 y 1101) no son utilizados para exhibir ningún carácter, por lo que podemos esperar que tales conjuntos de entradas no sean aceptados. Así pues, los podremos utilizar como indeterminaciones a la hora de minimizar las funciones lógicas correspondientes. A pesar de que las siete funciones (cada una de las funciones que encienden o apagan un LED del display) comparten las mismas variables de entrada (se trata en realidad de una multifunción cuyas técnicas de minimización escapan de los objetivos de este curso) nosotros vamos a implementar esas siete funciones por separado y con métodos de implementación diversos.

Para todos y cada uno de los siete segmentos indicados en los apartados siguientes se realizará un módulo independiente. En el último apartado se combinará el uso de estos siete módulos con displays y/o LEDs para probar la implementación completa en el simulador TkGate.

B3.2.3.1. Implementación del segmento a

Se implementará como una suma de productos convencional (forma normal disyuntiva). Se trata de una implementación típica, similar a la realizada en los circuitos del sumador de 1 bit realizados en los ejercicios de la sesión anterior. El circuito resultante, pues, tendrá un primer nivel de puertas AND seguido de un segundo nivel OR que combinará todas las salidas de las puertas AND anteriores.

B3.2.3.2. Implementación del segmento b

Se implementará como un producto de sumas (forma normal conjuntiva). Para ello se crea el mapa de Karnaugh al igual que cualquier función, pero a la hora de buscar implicantes nos fijamos en ceros en vez de los unos (maxitérminos). Hallaremos entonces grupos implicantes primos esenciales (de ceros en lugar de unos), haremos la cobertura mínima (de nuevo de ceros en vez de unos) y crearemos los productos de maxitérminos, teniendo la precaución de quedarnos con las variables del mapa de Karnaugh negadas, en el momento de simplificar. A la hora de implementarlo con puertas lógicas, los maxitérminos serán las variables negadas entrando a una puerta OR y la salida de todas las puertas OR serán las entradas de una AND.

B3.2.3.3. Implementación del segmento c

Implementarlo sólo con sólo puertas NAND. A partir de una minimización tipo suma de productos (forma disyuntiva) basta con llevar las variables de los minitérminos que hemos escogido en nuestra minimización a un primer nivel de puertas NAND, y las salidas de éstas llevarlas a otra puerta NAND con tantas entradas como implicantes hayamos escogido en nuestra minimización. Hay que tener cuidado con aquellos términos que impliquen a una sola variable, pues éstos se llevarán directamente a la puerta NAND del segundo nivel pero negada (con un inversor = una puerta NAND de dos entradas unidas entre sí).

B3.2.3.4. Implementación del segmento d

Implementarlo sólo con sólo puertas NOR. Si la implementación con sólo puertas NAND se hace de forma similar a la forma normal disyuntiva, la implementación con sólo puertas NOR recuerda a la implementación normal conjuntiva (segmento b). El proceso de minimización es igual al referido en el segmento b, pero a la hora de implementarlo con puertas lógicas se sustituyen directamente los dos niveles de puertas por puertas NOR. Hay que tener también cuidado con aquellos términos que impliquen a una sola variable, pues éstos se llevarán directamente a la puerta NOR del segundo nivel pero negada.

B3.2.3.5. Implementación del segmento e

Se implementará obligatoriamente con la ayuda de un multiplexor 8 a 1. El uso de multiplexores en la implementación de funciones lógicas como módulo lógico universal, deriva del hecho de que las entradas de control de un MUX seleccionan las entradas de datos de éste de manera que proporcionan una manera de generar los minitérminos de las variables que conectemos en las entradas de control. De esa manera, en nuestro caso, si usáramos un MUX 16 a 1 (con 4 entradas de control) bastaría con conectar los valores que en la tabla de verdad hacen 1 nuestra función a las entradas de datos (las variables a las 4 entradas de control). En nuestro caso hemos de conectar a la entrada de datos una "función residuo" (calculada a partir de las variables que no hayamos escogido en las entradas de control). Éste puede ser un buen momento para repasar la teoría de clase y ejercitarnos con un caso real.

B3.2.3.6. Implementación de los segmentos f y g

La implementación de estos dos bits NO la haremos en la clase de prácticas. En lugar de ello, y meramente a título ilustrativo, se proporcionan sendas implementaciones de estos bits usando un par de técnicas alternativas, en concreto, usando un decodificador 3 a 8 en el primer caso, y un demultiplexor de 3 a 8 en el segundo. Se trata de dos técnicas que NO vemos en clase y que, por tanto, no se exigirán en los exámenes. No obstante, puede resultar instructivo echarle un vistazo a sus respectivas implementaciones. Se puede comprobar así como, usando una técnica similar a la de los multiplexores, los decodificadores también nos pueden servir como módulos lógicos universales en la implementación de funciones. Los decodificadores, tal y como estudiamos en la teoría, son capaces de activar una y sólo una de sus 2^n salidas, dependiendo del valor de sus n entradas. De igual forma, y aunque no los vimos tampoco en las clases de teoría, un demultiplexor tiene una función muy parecida a la de un decodificador como módulo lógico universal, ya que si por su entrada de datos introducimos un 1 lógico (*Vdd* en TkGate), las entradas de control conectadas a las variables hacen que cada salida del demultiplexor sea un minitérmino de la función.

B3.2.4. Simulación del circuito

Para ello descargaremos del aula virtual el circuito `deco7seg.v`, y lo cargaremos en el entorno de simulación (Figura I.15).

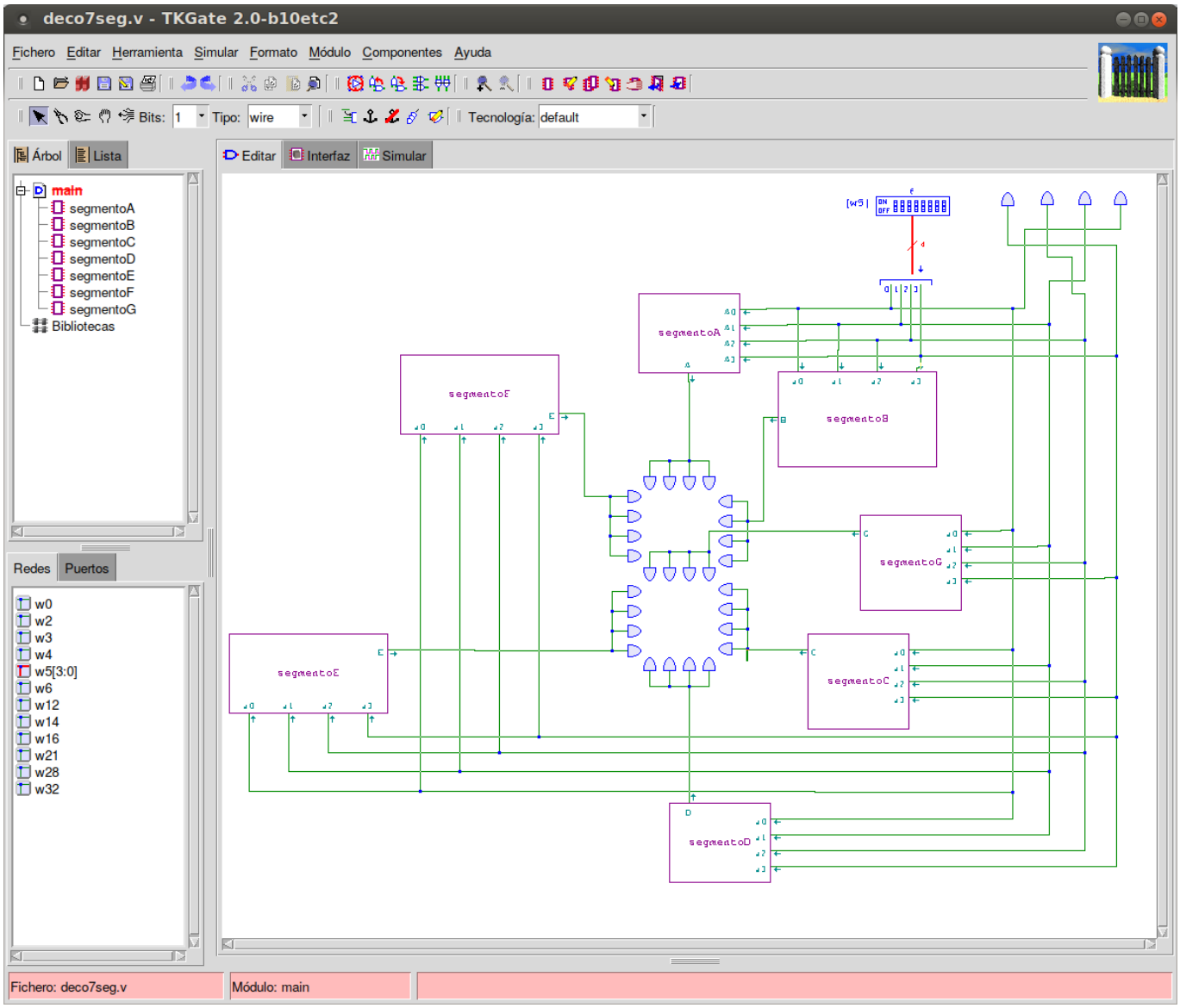

Figura I.15: Exhibidor BCD de 7 segmentos implementado en TkGate).

Al igual que hicimos con el diseño modular del sumador, conviene ahora echar un vistazo a la implementación interna de los segmentos a-f, seleccionando los respectivos circuitos *segmentoA*-*segmentoF* en la correspondiente vista de *Árbol*.

Finalmente, podemos simular el circuito para comprobar su correcto funcionamiento (pestaña *Simular*, y pulsando a continuación el icono *Play*). Al igual que hicimos la semana pasada con el circuito sumador, podemos interactuar con el interruptor DIP para ir introduciendo distintos los dígitos hexadecimales 0-F, correspondientes a las ristras de entrada 0000-1111, y comprobando el correcto funcionamiento del circuito en todos los casos.