PRÁCTICA 2

Implementación de funciones lógicas. Sumador binario combinacional.

Objetivos

Después de realizar esta práctica, el alumno deberá:

1) Saber implementar cualquier función lógica en suma de minterms, usando los siguientes dispositivos: 1)puertas básicas *Not, And, Or*, 2)decodificador y puertas *Or*, 3)*ROM*.
2) Conocer la funcionalidad de un *Full-adder* y saberlo implementar usando puertas *Xor* y Or .
3) Saber implementar y comprender el funcionamiento de un sumador binario combinacional con propagación del acarreo.
4) Saber obtener el tiempo de propagación del sumador, para cualquier número de bits.

Desarrollo 1

El *Full-adder* es un circuito combinacional con tres entradas (*x*, *y*, *z*) y dos salidas (*c*, *s*). El vector de salida *W* = (*c*, *s*) codifica en binario el número de entradas con valor igual a 1, esto es:

$$W_u = x + y + z = c.2^1 + s.2^0$$

Completa la siguiente tabla de verdad del Full-adder:

x	***y***	***z***	***c***	***s***
0	*0*	*0*		
0	*0*	*1*		
0	*1*	*0*		
0	*1*	*1*		
1	*0*	*0*		
1	*0*	*1*		
1	*1*	*0*		
1	*1*	*1*		

Escribe la expresión lógica en suma de *minterms* de cada una de las dos salidas del *Full-adder*

***c* =**

***s* =**

Construye en *Logic-works* el circuito que implementa las dos expresiones anteriores, utilizando las puertas *Not, And-4, Or-4* (las encontrarás en la librería de la práctica2: *Lib-SD-2*). A continuación comprueba el funcionamiento lógico del circuito.

Desarrollo 2

Construye en *Logic-works* el circuito que implementa las dos expresiones anteriores, utilizando un decodificador y puertas *Or* (en la librería encontrarás el decodificador necesario, en este caso de tres entradas: analiza su funcionamiento). A continuación comprueba el funcionamiento lógico del circuito.

Desarrollo 3

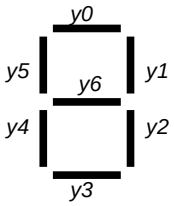
Construye en *Logic-works* el circuito que implementa las dos expresiones anteriores, utilizando una *ROM*. Para crear la *ROM* hay que seguir los siguientes pasos:

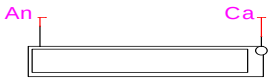
1. Pinchar en  de la barra superior de herramientas y seleccionar *PROM* (ROM programable)
2. Especificar el número de lineas de dirección (entradas del decodificador interno de la *PROM*), y el número de bits por palabra (nº de salidas de la *PROM*, es decir, nº de funciones a implementar)
3. Seleccionar *enter hex data manually*
4. Entrar los datos del contenido de la *PROM* en hexadecimal. Lo mejor es escribir este contenido previamente en un fichero de texto y cortar y pegar en la ventana de datos manual del simulador (para guardar una copia del contenido de la *PROM*, porque una vez creada ya no podemos modificarla en caso de error)
5. Dar nombre al dispositivo y salvarle en la librería correspondiente (*Lib-SD-2*).

A continuación comprueba el funcionamiento lógico del circuito.

Desarrollo 4

Construye en *Logic-works* un sistema para representar un dígito en decimal. El sistema consta de 4 señales digitales de entrada (para representar en binario números del 0 al 9) y 7 funciones de salida *y0, y1, y2, y3, y4, y5, y6*. Estas señales codifican los 10 posibles dígitos en decimal.
Cada una de las funciones está asociada a un segmento de un visualizador de 7 segmentos como el que se muestra en la figura. Cuando *yi* vale 1, el segmento correspondiente se ilumina. En cada instante, el visualizador ha de formar el dígito que codifiquen las 7 señales





Cada segmento del visualizador se implementa con el dispositivo *LED-Black* , y funciona de la siguiente manera: cuando la entrada *Ca* es 1 el led está apagado, independientemente de la otra entrada *An*, y cuando *Ca* es 0 el led se enciende para *An* = 1.

1. Encuentra la tabla de verdad de las señales *yi*
2. Implementa el circuito en una ROM
3. Conecta adecuadamente las salidas de la ROM a los leds (en lugar de crear cables, se recomienda conexión por nombre, para evita que los cables pasen por encima de los leds)

Desarrollo 5

El objetivo es implementar un sumador de dos números naturales, *X* e *Y*, representados en binario con 4 bits cada uno: *X = x3 x2 x1 x0* e *Y = y3 y2 y1 y0*. Como *X* e *Y* pueden valer cada uno $0 \le X, Y \le 2^4-1$, la suma *W = X + Y* puede valer $0 \le W \le 2^5-1$. Esto significa que necesitamos 5 bits para representar el resultado. Por tanto, el circuito necesario consta de ocho entradas y cinco salidas:

¿Cúantas filas tiene la tabla de verdad que especifica el funcionamiento del sumador binario de dos números de 4 bits?

¿Por qué no es viable la implementación en suma de minterms del circuito anterior?

Un método más sencillo que la suma de minterms para implementar el sumador es especificar las salidas (el resultado de la suma) mediante el siguiente algoritmo (suma en binario con propagación de acarreo):

$c_0 = 0$

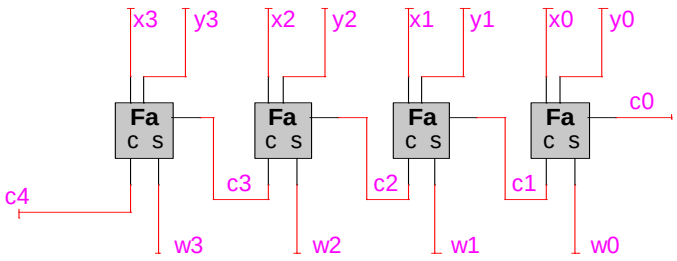
para k = 0 **hasta** $n-1$ **hacer**

(c_{k+1}, w_k) = **Full-adder** (x_k, y_k, c_k)

fin_para

$w_n = c_n$

La implementación combinacional de un algoritmo iterativo como el anterior se puede obtener desenrollando totalmente el bucle (en nuestro caso, para *n* = 4). Entonces, necesitamos 4 dispositivos *Full-adder* y conectarlos entre sí como indica el algoritmo desenrollado. En el circuito *Sum-4* asociado a la práctica encontramos la solución:



Observa la implementación interna de los dipositivos *Full-adder* (Fa) del circuito Sum-4 : no es el diseño en suma de minterms que vimos anteriormente. Se trata de otra implementación que utiliza dispositivos *Half-adder,* construidos con puertas *Or* y *Xor* (3 niveles de encapsulado):

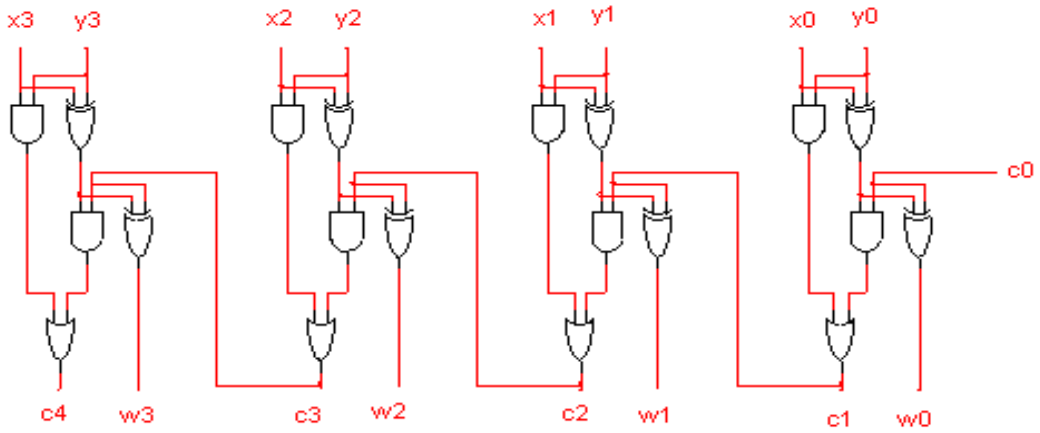
¿Con cuantas puertas básicas (*And, Or, Not*) está construido cada *Full-Adder*?

Comprueba el funcionamiento lógico del sumador, siguiendo estos pasos:

1. Conecta dos teclados hexadecimales (*HexKeyboard*) para las entradas *X* e *Y*. Primero hay que comprobar el orden de los bits de los teclados, por ejemplo conectando un *BinaryProbe* y pinchando valores del teclado
2. Conecta un visor hexadecimal (*HexDisplay*) en la salida *W* .
3. Conecta un *BinarySwitch* en el carry inicial (c0) con el valor 0
4. Conecta un *BinaryProbe* en el carry final (c4)
5. Da valores a las entradas y comprueba si las salidas son las esperadas

Desarrollo 6

Análisis temporal del sumador: Estudiaremos en este apartado el retardo del dispositivo implementado en el desarrollo 5. Utiliza el esquema facilitado a continuación, en el que se muestran las puertas lógicas que forman cada full-adder, para responder a las preguntas que se plantean.



¿Cuál es el camino crítico del sumador? Lista la secuencia de puertas por las que pasa dicho camino crítico.

¿Cuál será el retardo (tiempo de propagación) del sumador? Utiliza los retardos de cada puerta vistos en prácticas anteriores y escribe también el valor como suma de los retardos de cada Full Adder.

Encuentra una fórmula que exprese el tiempo de propagación en función del número de bits de cada sumando.

Desarrollo 7

En el análisis planteado en el Desarrollo 5 la entrada c0 tenía siempre el valor lógico 0. Con esta configuración, ¿Será posible encontrar con el cronograma algún caso en el que podamos visualizar el retardo calculado en el desarrollo anterior?

¿Podrías definir cuál sería en este caso el nuevo camino crítico y su retardo?

El nuevo valor de retardo será ahora representable en el cronograma. Completa la siguiente tabla, de manera que el retardo conseguido en las señales c2, c3, w3 sea el máximo posible, visualizando en el cronograma los ciclos de retardo calculados previamente. Dos pistas:

1. Para que exista un cambio de señal a lo largo del camino crítico debe haber carry en los Full adders intermedios.
2. La puerta XOR no tiene un retardo fijo para todas las combinaciones de entrada, recupera la tabla de retardos que hiciste en la práctica 1 para completar este apartado.

Valor en t-dt								***Valor en t+dt***								***Retardo***			
x3	***x2***	***x1***	***x0***		***y3***	***y2***	***y1***	***y0***	***x3***	***x2***	***x1***	***x0***		***y3***	***y2***	***y1***	***y0***	***c1***	***c2***	***c3***	***w3***
0	*0*	*1*	*0*		*0*	*0*	*0*	*0*													
1	*0*	*0*	*1*		*0*	*1*	*1*	*1*													