Sistemas Digitales

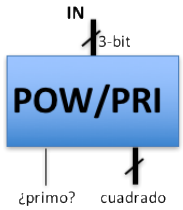
Ejercicios de Examen, Circuitos Combinacionales

Para solucionar cada ejercicio, abre un nuevo proyecto en LogiSim. Dentro de cada proyecto, vete creando los circuitos que marque el enunciado.

Ejercicio 1. Final 2016-2017 (Proyecto Ejercicio1.circ)

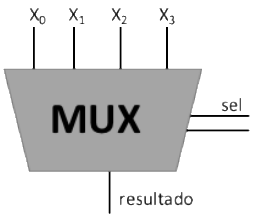
Realiza las siguientes implementaciones en *LogiSim*:

1. Construye, mediante suma de *minterms*, un circuito capaz de detectar números primos, siendo su entrada números codificados en binario natural con 3 bits. Anota la tabla de verdad en la hoja del examen y guarda dicho circuito en un fichero de nombre ***primos***.
2. Construye, mediante una *ROM*, un circuito capaz de calcular el cuadrado (X^2) de un número codificado en Complemento a 2 con 3 bits. Anota la tabla de verdad en la hoja del examen. Guarda el circuito con la rom como **potencia**.
3. Crea un circuito con una estructura como la de la figura inferior. Para cada valor de entrada de 3 bits la cápsula calculará de forma simultánea si se trata de un número primo y su cuadrado. El vector de 3 bits será interpretado como un número codificado en Binario natural para el cálculo de número primo o en Complemento a 2 para el cuadrado. Guarda el circuito con el nombre **POW-PRI**.



Ejercicio 2. Parcial 2018-2019 (Proyecto Ejercicio2.circ)

Utilizando multiplexores con un único bit de selección (2 entradas), crea un circuito multiplexor de 4 entradas, con un esquema similar al mostrado en la figura inferior. Utilizando pines de entrada y salida de 4 bits implementa un circuito para comprobar el funcionamiento de tu multiplexor. Guarda el circuito como **Mux4in**.



Ejercicio 3. Parcial 2018-2019

Como se muestra en la *Figura 1*, la cápsula CA se concatena para crear un circuito con múltiples bits de salida. Encontrar una fórmula que exprese el tiempo de propagación en función del número de bits de salida. Utiliza la tabla de retardos de la cápsula proporcionada en la *Figura 2*.

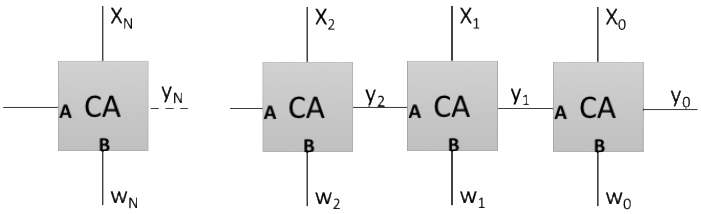


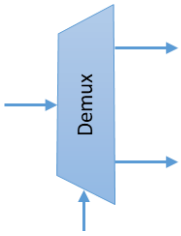
Figura 1: Circuito del ejercicio 24

out\in	X	Y
A	60	20
B	100	40

Figura 2: Retardos de la capsula CA

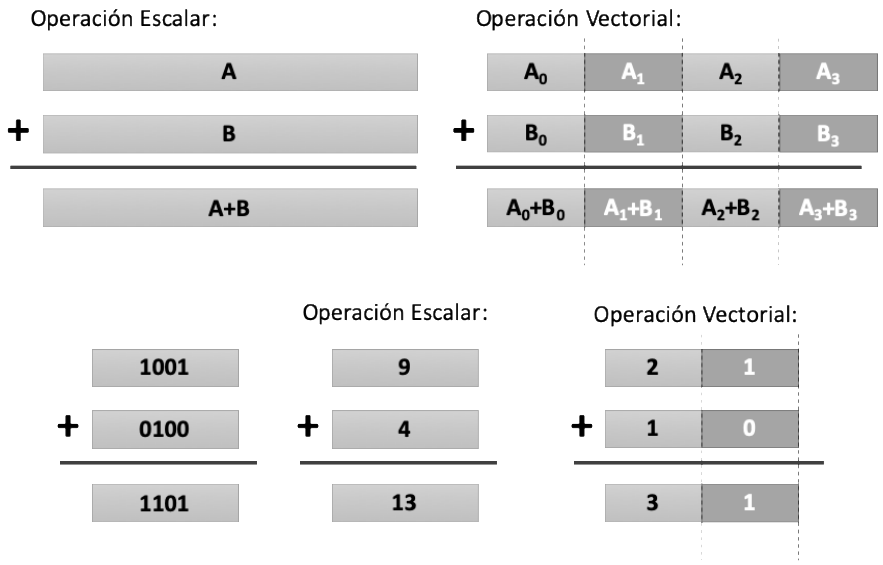
Ejercicio 4. Final 2018-2019 (Proyecto Ejercicio4.circ)

Mediante el método de síntesis en suma de *minterms*, implementa un *Demultiplexor* con una entrada y dos salidas como el de la figura inferior. En la hoja del examen, anota la tabla de verdad de dicho circuito y guarda tu implementación con el nombre **Demux.**



Ejercicio 5. Septiembre 2018-2019 (Proyecto Ejercicio5.circ)

Los procesadores comerciales actuales implementan, en su ALU, Unidades Funcionales para realizar operaciones vectoriales. Para ello, los registros que almacenan los datos se pueden interpretar de diferente manera, dependiendo del tipo de operación a realizar. Si la operación es escalar, el registro almacena un único dato, mientras que, si la operación es vectorial, el contenido del registro se interpreta como un vector con N valores (La Figura inferior muestra un ejemplo para vectores de 4 componentes).



El objetivo de este ejercicio es hacer uso de los sumadores de la librería de logisim para crear un circuito que opere de ambas maneras. A través de una señal de control llamada E-V, indicaremos el tipo de operación a realizar. Si E-V=1 la operación es escalar y el Sumador funciona de forma convencional. En caso contrario, los valores de entrada se interpretarán como vectores de 2 elementos (4 bits cada elemento) ($[A_0,A_1]$).

Lleva a cabo las siguientes tareas para implementar el circuito solicitado:

a) (2p) Realiza los cambios necesarios al sumador proporcionado para que sea capaz de realizar ambos tipos de operación, escalar y vectorial (vector de 2 elementos). Guarda tu circuito como ***sumadorVec***.

b) (2p) Mediante una ROM, implementa el circuito detector de *overflow* **si interpretamos los números como binario natural**. En el caso de una operación vectorial, existe *overflow* en la operación si se produce *overflow* en cualquiera de las dos sumas del vector. Guarda el circuito de comprobación como **overflow**. Conecta la ROM al sumador modificado y guarda dicho circuito como ***sumadorVecFinal***.

Ejercicio 6. Parcial 2019-2020 (Proyecto Ejercicio6.circ)

Las redes neuronales se construyen a partir de un elemento básico conocido como neurona LTU. Dicha neurona genera un valor de salida a partir de una función de activación. En este ejercicio vamos a implementar una función de activación denominada *Leaky ReLU*, cuyo funcionamiento se describe a continuación:

$$Y = \begin{cases} X \; si \; X > 0 \\ \dfrac{X}{2} \; si \; X \leq 0 \end{cases}$$

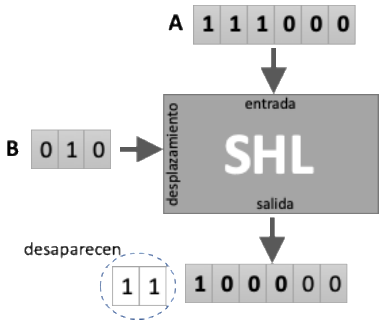
Para nuestro ejercicio, supondremos que tanto la entrada X como la salida Y, son números enteros de 4 bits representados en Ca2 (redondea al valor entero inferior en aquellos casos que sea necesario). Se pide:

a) Obtén la tabla de verdad del circuito que implemente la función anterior. Anota dicha tabla en tu hoja de examen.

b) Implementa dicho circuito con una ROM. Comprueba el funcionamiento correcto de tu circuito, conectando pines de entrada y salida de 4 bits. Guarda el circuito con el nombre **relu**.

Ejercicio 7. Parcial 2019-2020 (Proyecto Ejercicio7.circ)

En este ejercicio vamos a implementar una operación presente en la mayoría de CPUs actuales, la función de desplazamiento lógico (*SHL ó Shift Lógico*). Esta operación consiste en desplazar los bits de un operando (A) tantas posiciones a izquierda o derecha como indique un segundo operando (B). La dirección del desplazamiento vendrá dada por el signo de B (izquierda si es positivo, derecha si es negativo). Una vez hecho el desplazamiento, se rellena con ceros independientemente de la dirección, tal y como se puede observar en el ejemplo.



Se pide:

a) Obtén la tabla de verdad de un circuito SHL que desplace un operando A de 3 bits, tanto como indique un operador B de 2 bits en Ca2. La salida del circuito tendrá también 3 bits. Anota dicha tabla en tu hoja de examen.

b) Implementa dicho circuito con una ROM. Comprueba el funcionamiento correcto de tu ROM haciendo uso de pines de entrada y salida de 4 bits. Realiza un shift de 1 posición a la derecha del valor de entrada 111. Guarda el circuito con el nombre **shift.**

Ejercicio 8. Final 2019-2020 (Proyecto Ejercicio8.circ)

El Código binario reflejado (RBC en inglés), utilizado en tareas de corrección de errores en sistemas de comunicación es un tipo de codificación de números naturales en el que dos valores consecutivos solamente pueden diferir en uno de sus dígitos en su representación binaria (por ejemplo 3 y 4, consecutivos, no se pueden representar como 011 y 100 pues difieren en tres bits). En este ejercicio deberás implementar un circuito combinacional capaz de convertir un número natural de 3 bits a su formato en código RBC. Para ello, lleva a cabo las siguientes tareas:

a. Propón una implementación de código RBC para números naturales representables con 3 bits.

b. Escribe en la hoja de examen la tabla de verdad del circuito conversor de binario natural a código RBC.

c. Lleva a cabo el proceso de síntesis para dicha tabla de verdad utilizando el Decodificador y puertas OR proporcionadas en la librería de LogiSim. Guarda tu solución como **circuitoRBC**.

Ejercicio 9. Final 2019-2020 (Proyecto Ejercicio9.circ)

La implementación de sumador que hemos visto en la Práctica 2, conocida como "Ripple-Carry adder", tiene como ventaja su simplicidad de implementación, pero un problema con el retardo de peor caso (camino crítico del circuito). Con el fin de reducir el tiempo de propagación vamos a hacer una implementación más eficiente, usando sumadores más pequeños y un poco de "inteligencia". Dividiremos la suma de 8 bits en dos sumas de 4 bits. Para los 4 bits menos significativos llevaremos a cabo la suma de manera convencional. En el caso de los 4 bits más significativos, haremos dos operaciones de manera simultánea, una asumiendo un Carry-in de 0 y otra con un Carry-in de 1. De esta forma, el Carry-out de la suma de los bits menos significativos nos indicará cuál de las dos operaciones (Carry-in=0 o Carry-in=1) debemos utilizar como resultado para los bits más significativos.

a. Haciendo uso de sumadores y multiplexores de 4 bits proporcionados en la librería de logisim, lleva a cabo una implementación del circuito sumador de 8 bits con el formato descrito en este enunciado. Debe tener pines de entrada y salida de 8 bits, conectados con los sumadores a través de separadores. Guarda tu circuito en la librería del examen con el nombre **SumadorOpt**.

b. Calcula cuál ha sido la mejora de rendimiento (reducción del camino crítico) conseguida con tu implementación frente al Ripple-Carry adder visto en las prácticas. Asume que los retardos de las puertas son los siguiente: AND=OR=NOT=20u.t., XOR=50u.t..

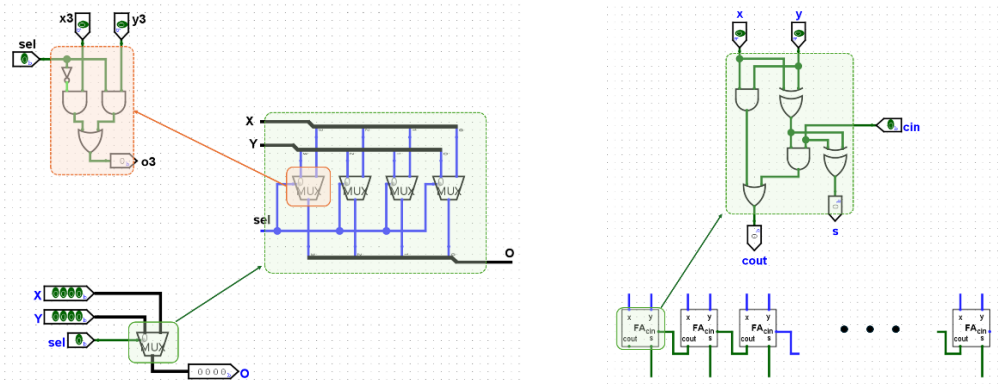


Figura: Ripple-Carry adder y Multiplexor.

Ejercicio 10. Septiembre 2019-2020 (Proyecto Ejercicio10.circ)

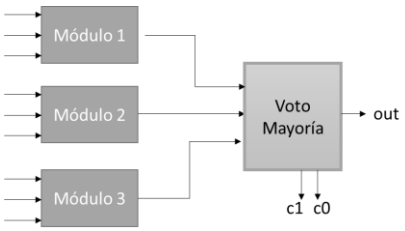
Vamos a implementar un circuito combinacional para optimizar la organización del transporte escolar en un colegio. Se trata de un colegio con tres clases, de 20, 16 y 10 alumnos, que cuenta con tres vehículos, un autobús de 30 plazas, un microbús de 10 y un coche de 6. El objetivo es crear un circuito que reciba como entrada qué clases van a salir de excursión y retorne como salida qué vehículos se deben utilizar para minimizar el consumo en gasolina (a menor capacidad del vehículo menor consumo de combustible, el autobús consume más que microbús y coche juntos). Para ello, lleva a cabo las siguientes tareas:

a. Propón una tabla de verdad que describa el problema (escríbela en la hoja del examen).

b. Lleva a cabo el proceso de síntesis para dicha tabla de verdad utilizando suma de minterms. Determina el camino crítico del circuito resultante, teniendo en cuenta los siguientes retardos: AND,OR=30u.t., NOT=10u.t. Guarda tu circuito solución como **Transporte**.

Ejercicio 11. Parcial 2020-2021 (Proyecto Ejercicio11.circ)

La tolerancia a fallos se define como la capacidad de un sistema para seguir funcionando aunque alguno de sus componentes falle (piensa en la importancia que pueden tener esto en los sistemas electrónicos de, por ejemplo, un avión o un satélite). Uno de los mecanismos más sencillos de tolerancia se conoce como "*triple modular redundancy*" (redundancia triple) y consiste en replicar tres veces un circuito y procesar el resultado a través de un sistema de "voto por mayoría". El esquema es sencillo, como puedes ver en la figura inferior. Tres circuitos realizan una misma operación, y sus salidas binarias se introducen a un circuito que genera una única salida binaria igual a la mayoría de los valores que ha observado en las entradas (por ejemplo, si dos módulos generan un 1 y el otro un 0, *out* debería ser 1). De esta forma, si uno de los tres circuitos falla, los otros dos forzarán a que el valor de salida sea el correcto.



c1	c0	Fallo
0	1	Módulo 1
1	0	Módulo 2
1	1	Módulo 3
0	0	Ninguno

El objetivo del ejercicio es que implementes el circuito de votación de este sistema, teniendo en cuenta que la salida *out* proporciona el valor correcto (valor que gana la votación) y las dos salidas restantes identifican cuál de los módulos previos produjo el fallo, con la codificación de la tabla de la figura. Para ello se pide:

a. Obtén la tabla de verdad del circuito a implementar. Anota dicha tabla en tu hoja de examen (así como las explicaciones que consideres necesarias).
b. Implementa dicho circuito como suma de minterms. Guarda dicho circuito con el nombre **Votador**.

Ejercicio 12. Parcial 2020-2021 (Proyecto Ejercicio12.circ)

Vamos a implementar un sumador capaz de proporcionar el resultado con un retardo menor en algunos casos especiales. Gracias a tus conocimientos de álgebra de Boole sabes que A+0=A, por lo que, si uno de los sumandos es igual a cero el resultado de la suma es el otro sumando. Vamos a implementar un circuito que aproveche esto, cuyo funcionamiento se describiría así:

$$W = \begin{cases} X + Y \; si \; X \neq 0 \\ Y \; si \; X = 0 \end{cases}$$

Para nuestro ejercicio, supondremos que tanto las entradas X,Y como la salida W, son números enteros de 4 bits representados en Ca2. Se pide:

a. Implementa, mediante una ROM, un circuito que tome como entrada un entero de 4 bits y cuya salida sea un bit indicando si es igual o distinto de cero. Guarda dicho circuito, con pines de entrada y salida de 4 bits como **ROMCMP**.
b. Haciendo uso de la ROM del apartado anterior, del sumador y del multiplexor proporcionados en la librería, implementa el circuito para la fórmula anterior y guárdalo como **OptAdder** con dos buses de entrada y uno de salida. Comprueba su funcionamiento**.**

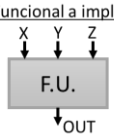
Ejercicio 13. Final 2020-2021 (Proyecto Ejercicio13.circ)

Tras analizar en detalle el código de todas las aplicaciones que se ejecutan en nuestro procesador, hemos detectado una secuencia de instrucciones que se repite con frecuencia, cuyo pseudocódigo se describe en la figura inferior. En dicha estructura, la operación para calcular el valor de salida cambia en función del valor de una de las entradas. Con el fin de optimizar el rendimiento de nuestro procesador, vamos a crear una unidad funcional específica que lleve a cabo las operaciones de dicha estructura, de tal forma que podamos expresarlo como una única instrucción. Para nuestra unidad funcional, supondremos que tanto las entradas como la salida son números enteros de 4 bits representados en Ca2.

Pseudocódigo

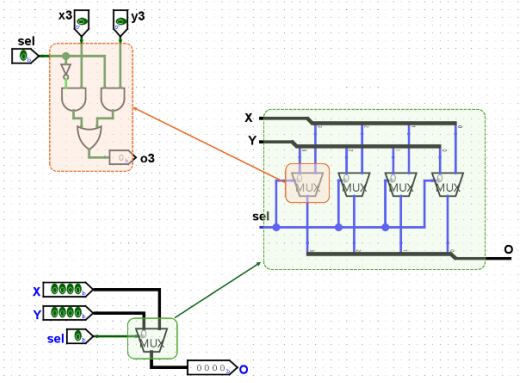
```
IF (Z%2==0) THEN    //si Z es par
    OUT=X+Y
ELSE                //si Z es impar
    OUT=X-Y
```

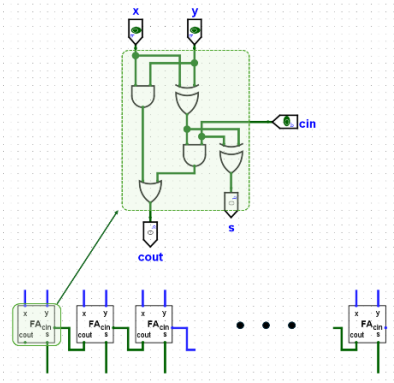
Unidad Funcional a implementar



Lleva a cabo las siguientes tareas:

a. Haciendo uso de los componentes proporcionados en la librería (Excepto el restador, debes buscar una forma alternativa de hacer la resta), implementa el circuito solicitado para la unidad funcional y guárdalo como **CondAdder**, con los pines de entrada y salida de 4 bits necesarios para hacer las comprobaciones de funcionamiento.
b. Con las implementación de Sumador y Multiplexor de las figuras inferiores, determina el camino crítico, así como el tiempo de propagación del circuito que has creado. **RETARDOS**: AND,OR = 20 u.t, NOT = 10 u.t., XOR = 60 u.t.





Ejercicio 14. Final extraordinario 2020-2021 (Proyecto Ejercicio14.circ)

El Ca2 es el sistema de representación visto en clase para números enteros. Existen representaciones alternativas tales como la "codificación en zig-zag", que se utilizan en la actualidad por sus propiedades (Por si tienes interés tras el examen, esta codificación es apropiada para optimizar el espacio de almacenamiento de números enteros a través de un mecanismo conocido como "Variable Length Encoding"). En dicha codificación los números positivos y negativos se mapean de manera alternativa sobre valores positivos, en una especie de zig-zag. Así, el -1 se codifica como 1, el 1 como 2, el -2 como 3 etc. (puedes ver la tabla anterior para comprender el funcionamiento.

Valor original	Codificación en zig-zag
0	0
-1	1
1	2
-2	3
2	4
…	…

En este ejercicio vamos a diseñar un circuito conversor de enteros de 4 bits en Ca2 a zig-zag. Para ello, lleva a cabo las siguientes tareas:

a. Determina la tabla de verdad del circuito conversor y lleva a cabo una implementación utilizando una ROM. Comprueba el funcionamiento utilizando pines de 4 bits para entrada y salida. Genera el valor de entrada -3 y guarda tu circuito como **zigzag**.
b. En Ca2 el bit más significativo indicaba el signo del número entero representado. Implementa un circuito capaz de determinar el signo para la representación zig-zag, guarda dicho circuito como **signozz** con la detección de signo de un -3 y explica en la hoja del examen los pasos que has dado.

Ejercicio 15. Parcial 2021-2022 (Proyecto Ejercicio15.circ)

Vamos a asumir que el procesador en el trabajamos implementa actualmente la operación de multiplicación de enteros con un circuito combinacional que presenta un retardo extremadamente elevado en su camino crítico. Para intentar reducir este tiempo de propagación y mejorar el rendimiento de nuestra CPU, vamos a proponer un circuito que trate de manera especial aquellas multiplicaciones que en realidad no necesitan llevar a cabo la operación para conocer el resultado (por ejemplo, las multiplicaciones por cero o por uno).El pseudocódigo que define el modo de operación del circuito multiplicador propuesto se muestra en la figura inferior (A y B son los operandos, entrada de nuestro CLC. W es el resultado, salida del CLC).

Parte 1. De cara a la implementación, nuestro primer paso será diseñar un circuito combinacional que detecte qué tipo de valor es el operando A (**de los 4 tipos posibles**). Asumiendo que A representa un valor entero de 4 bits, escribe en la hoja del examen la tabla de verdad del circuito, codificando los tipos de dato posibles para A con el mínimo número de bits (añade las explicaciones que consideres necesarias). Implementa dicho circuito mediante una ROM, utiliza pines de entrada y salida de 4 bits para comprobar tu resultado, guardando el circuito resultante como **AType**.

Justifica si tu asignación de tipos es la óptima para una síntesis en suma de minterms (mínimo número de puertas). Si consideras que no lo es, indica cómo cambiarías la asignación y la tabla de verdad para reducir el número de puertas. No es necesario hacer la implementación en LogiSim.

Parte 2 Comprobarás que uno de los casos particulares que se contempla es el cambio de signo de B (si A=-1 entonces W=-B), así que vamos a implementar el circuito que lo lleve a cabo. Lo haremos mediante el mecanismo visto en clase, consistente en hacer primero el Complemento a 1 del valor de entrada y sumar 1 a continuación. Aunque disponemos en la librería de un Sumador, vamos a intentar optimizarlo sabiendo que uno de los sumandos siempre será uno. Sabiendo que la implementación de cada Full Adder del sumador se corresponde con el circuito de la figura inferior, intenta optimizar dicho circuito para nuestro caso concreto (un sumando es siempre 1). Crea un circuito de nombre **CambioSigno** con un bus de entrada y uno de salida que contenga tu circuito optimizado.

Con tu implementación optimizada, ¿Qué ahorro de tiempo consigues para este componente si el bus de entrada es de 4 bits? (Comparado con el sumador completo). Busca una fórmula que exprese dicho ahorro en función del número de bits del bus de entrada. **RETARDOS**: AND = 30 u.t., OR = 20 u.t., NOT = 10 u.t., XOR = 60 u.t.

Parte 3 Haciendo uso de los componentes proporcionados en la librería y los que has ido creando[1], propón una implementación de CLC para el pseudocódigo del enunciado. Comprueba su funcionamiento conectando pines de entrada y salida de 4 bits. Fija los valores de entrada (A,B) a 2 y -3, comprueba el resultado y guarda dicho circuito como **E40Final**

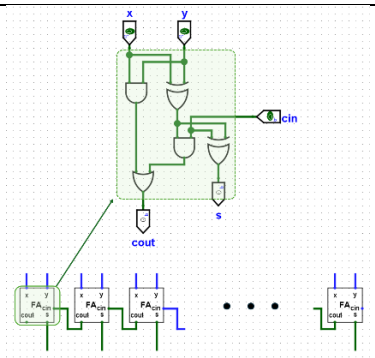
Operandos: A, B (4 bits Ca2) Resultado: W (4 bits Ca2) si A=0 entonces W=A si A=1 entonces W=B si A=-1 entonces W=-B en otro caso W=AxB	

Figura 1. Pseudocódigo de operación e implementación interna del sumador

[1] Si en algún apartado no has podido encontrar la solución, puedes crear una cápsula vacía para usar en este ejercicio.

Ejercicio 16. Final 2021-22 (Proyecto Ejercicio 16.circ)

En los primeros medios de almacenamiento de tipo magnético el coste de escritura de un bit a 0 era mayor que a 1. Por esta razón se ideaban mecanismos para minimizar el número de ceros a escribir. Uno de los mecanismos más sencillo consistía en almacenar un valor o su complemento a 1, en función de cual tuviera un número menor de ceros (si el número de ceros y unos es el mismo, el valor se deja como está). El objetivo de este ejercicio consistirá en proponer un circuito combinacional que decida, para valores de 4 bits, la representación que minimice el número de ceros con el mecanismo descrito. La salida de dicho circuito consistirá en el dato de 4 bits que se almacenará y un bit adicional que indica si se almacena en su formato original o como complemento a 1 (ver Figura 1). Lleva a cabo las siguientes tareas:

a. Escribe en la hoja del examen la tabla de verdad del circuito propuesto (CLC ZeroMin en la figura).
b. Implementa el circuito mediante una ROM. Crea un circuito para comprobar el funcionamiento de dicha ROM con pines y buses de entrada y salida de 4 bits para los datos y un cable de salida que indique si el valor se ha cambiado o no. Guarda tu circuito como **ZeroMin**

El siguiente paso consistirá en implementar una Unidad Funcional de suma para valores almacenados en dicho formato.

c. Haciendo uso de los circuitos sumados, multiplexor y las puertas lógicas que necesites, implementa un circuito que tenga como entrada dos datos almacenados en formato ZeroMin (5 bits cada uno) y retorne el resultado de su suma con su representación convencional (4 bits, Ca2). Guarda dicho circuito como **SumadorZeroMin**
d. A la vista de la implementación de los circuitos de Suma y Multiplexor de las Figuras 2 y 3, determina el camino crítico y tiempo de propagación del circuito completo que has implementado en el apartado anterior (not: 10u.t; and,or:30u.t; xor:60u.t).

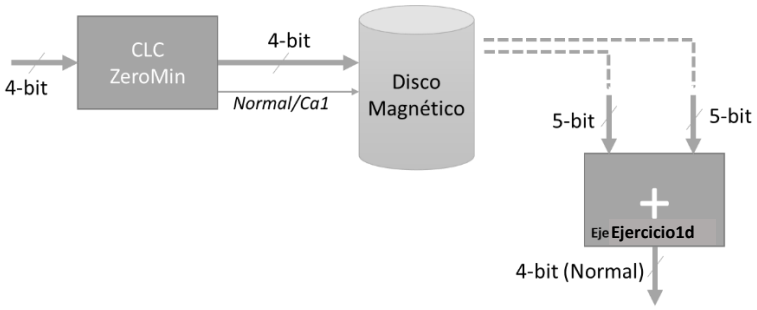


Figura 1. Esquema del circuito SumaZeroMin.

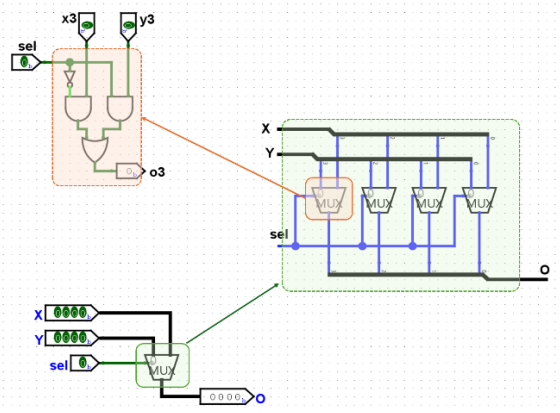


Figura 2. Implementación de Multiplexor

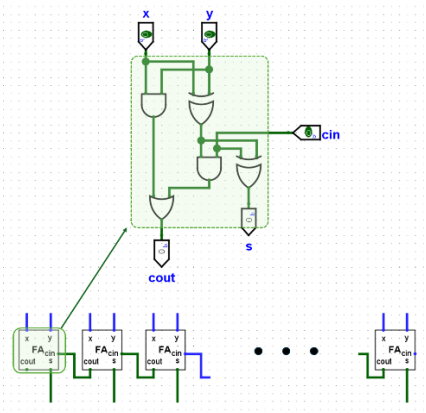


Figura 3. Implementación de Sumador

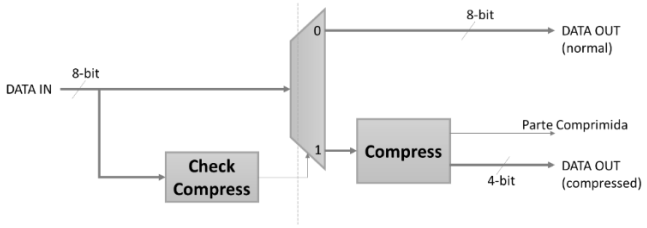
Ejercicio 17. Final extraordinario 2021-2022 (Proyecto Ejercicio17.circ)

Vamos a implementar un circuito que permita la compresión de datos bajo ciertas condiciones. El mecanismo de compresión funciona de la siguiente manera: Se divide el dato de entrada (N bits) en dos partes iguales y se comprueba por separado si cada parte es igual a cero. Si alguna parte (o las dos) es igual a cero, en vez de almacenar el dato completo almacenamos solo la parte distinta de cero, y utilizamos un bit adicional para indicar qué parte hemos eliminado por ser igual a cero (bits más o menos significativos).

El esquema de la implementación se muestra en la Figura inferior, organizado en dos partes. La primera parte se encarga de determinar si un valor de entrada es comprimible o no. El objetivo es proponer un circuito combinacional que realice esta labor. La entrada a dicho circuito será un valor de 8 bits. El circuito dividirá el dato, comprobará si alguna de sus partes es cero, y generará un bit de salida indicando si el dato es comprimible. Si solamente puedes hacer uso de los componentes de la carpeta Puertas de Logisim, esboza en la hoja del examen un esquema del circuito que vas a implementar[2] y explica su funcionamiento. Lleva a cabo dicha implementación, usando un bus de entrada para el dato y un cable de salida que indique si el valor se puede comprimir o no, guarda tu circuito como **CheckCompress.**

La segunda parte del circuito es la que lleva a cabo la compresión cuando sea pertinente. Como puedes observar, un demultiplexor gobernado por el circuito CheckCompress dirige el dato hacia el circuito de compresión. Haciendo uso de los componentes proporcionados en la librería, implementa el circuito de compresión. Dicho circuito tiene un bus de entrada (8 bits) para el dato, un bus de salida para el valor comprimido (4 bits) y un bit adicional que indica qué parte del dato se elimina. Guarda tu circuito como **Compress**.

Finalmente, Usando pines de entrada y salida, las cápsulas CheckCompress y Compress que has creado y el decodificador de la librería de LogiSim, implementa un circuito de comprobación completo de nombre **FullCompressor** (siguiendo el esquema de la figura inferior) y prueba su funcionamiento con el valor de entrada 0x0A.



Ejercicio 18. Parcial 2022-2023 (Proyecto Ejercicio18.circ)

Un circuito "Encoder" presenta en la salida el código en binario natural de la correspondiente entrada activa. Es decir, lleva a cabo la operación inversa a la de un Decodificador como los vistos en clase (Decoder: f(x) = y <-> Encoder: $f^{-1}(y) = x$). En un Decodificador, solo una de las salidas toma valor 1, por lo que no se dan todas las combinaciones de valores (Por ejemplo, la salida 1010 no es posible). Por eso es necesario definir el comportamiento del Codificador para estas combinaciones. En nuestro caso vamos a considerar que las salidas toman valor cero para todas aquellas combinaciones de entrada que no se corresponden con la salida de un Decodificador (Entrada 1010 ->salida todo ceros).

1. Dada la descripción de su funcionamiento, obtén la Tabla de verdad de un circuito Encoder de 4 entradas.
2. Lleva a cabo una implementación de tu circuito mediante una ROM, guardándola con el nombre **Encoder4input**. Recuerda añadir a tu circuito pines de entrada y salida para hacer las comprobaciones de funcionamiento pertinentes.

[2] Ten en cuenta que con 8 bits de entrada utilizar los métodos de síntesis que conoces no parece adecuado. Piensa, teniendo en cuenta lo que debe hacer y los componentes disponibles, cómo llegar a una solución más razonable.

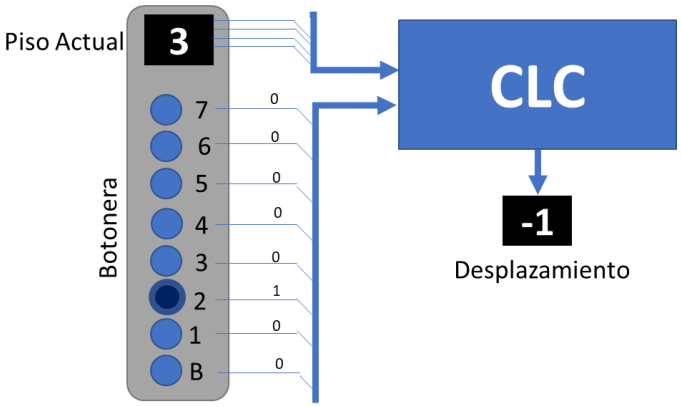
Ejercicio 19. Parcial 2022-23 (Proyecto Ejercicio19.circ)

Vamos a llevar a cabo la implementación de un circuito combinacional que forma parte del mecanismo de funcionamiento de un ascensor. Dicho circuito, cuyo esquema puedes ver en la figura inferior, funciona de la siguiente forma:

- La botonera tiene un cable por cada piso, y cuando se pulsa el botón para ir a una planta, ese cable tomar valor 1 (el resto de cables toman valor 0).
- En la otra entrada de nuestro CLC se proporciona información sobre la planta en la que nos encontramos actualmente, codificada como un número de 4 bits.
- La salida que proporciona el CLC es un número de 4 bits en Ca2 que indica al motor del ascensor cuántas plantas se debe desplazar (magnitud) y en qué dirección (signo, + indica hacia arriba, - indica hacia abajo).

Lleva a cabo las siguientes tareas:

1. Haciendo uso de los siguientes componentes: Puertas lógicas, Multiplexor, Codificador de prioridad y sumador, propón una implementación del CLC indicado. Incluye en tu hoja del examen todas aquellas explicaciones que consideres necesarias. Añade pines de entrada y salida para llevar a cabo las comprobaciones de funcionamiento y guarda tu circuito solución como **Ascensor**.



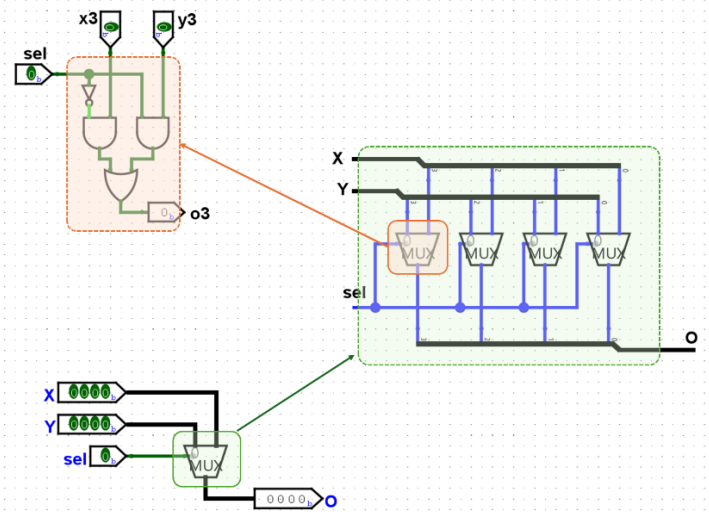
Ejercicio 20. Final 2022-2023 (Proyecto: Ejercicio20.circ)

Queremos conectar dos sistemas electrónicos que trabajan con representaciones distintas para números enteros. Para ello vamos a implementar un circuito convertidor que acepta números de 4 bits en Signo-Magnitud y los convierte a Ca2. Comienza llevando a cabo las siguientes tareas:

1. Obtén la tabla de verdad del circuito a implementar, anótala en tu hoja de examen.
2. Lleva a cabo una primera implementación del circuito convertidor mediante síntesis con ROM, guardándola en tu proyecto con el nombre **Converter4bit**.

A continuación, vamos a intentar optimizar el circuito inicial. Si te fijas en la T.V. de tu convertidor, en algunas filas el valor de salida coincide con el de la entrada. En dichos casos nos podríamos ahorrar el proceso de conversión, y el valor de salida se obtendría en un tiempo más corto. Implementa un nuevo convertidor siguiendo estos pasos:

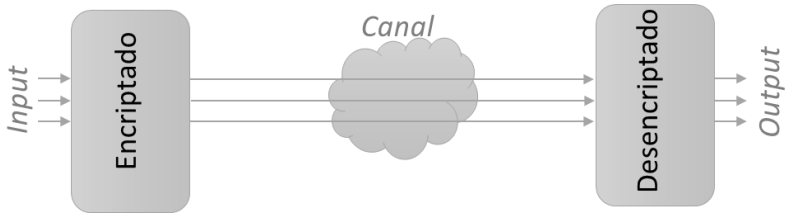
3. Haciendo uso del Multiplexor (2 entradas) y la ROM del apartado anterior, implementa un circuito en el que solo los números que cambian tengan que atravesar el convertidor. Añade pines de entrada y salida para tus comprobaciones y guarda dicho circuito en tu proyecto como **ConverterV2**.
4. Si el 80% de los valores a la entrada del convertidor son números que no necesitan un cambio, y asumiendo una implementación de multiplexor como la mostrada en la figura inferior, ¿Cuál será el tiempo promedio que se ahorra utilizando la versión optimizada de tu convertidor? Asume los siguientes tiempos de propagación: ROM-100u.t. XOR-50u.t. AND,OR-20u.t. NOT-10u.t.



Ejercicio 21. Parcial 2023-2024 (Proyecto: Ejercicio21.circ)

Tu empresa quiere proteger un canal de comunicación por el que se envían números en Ca2 con un esquema de encriptado/desencriptado como el de la figura inferior. Tu cometido será diseñar los dos módulos de acuerdo con las especificaciones proporcionadas. El algoritmo de encriptado funciona de la siguiente forma: Si el valor de entrada es par, los bits se rotan una posición hacia la derecha, en caso contrario dos posiciones hacia la izquierda (Ejemplo: El valor ABCD rotado 2 posiciones a la derecha se convierte en CDAB). Finalmente, al valor rotado se le suma un 1.

1. Obtén la tabla de verdad del circuito de cifrado para entradas de 3 bits, y lleva a cabo una implementación mediante ROM, añade pines de entrada y salida y guarda tu circuito con el nombre **Encriptado**.
2. Obtén la tabla de verdad del circuito de desencriptado e implementa el circuito mediante síntesis con decodificador y Puertas OR. Añade pines de entrada y salida y guarda tu circuito con el nombre **Desencriptado**..



Ejercicio 22. Parcial 2023-2024 (Proyecto: Ejercicio22.circ)

Trabajas en una empresa que se dedica a la fabricación de procesadores de propósito específico. Uno de vuestros clientes os solicita que sus procesadores incluyan una unidad funcional con las especificaciones de la figura inferior (ABS es la operación "valor absoluto"). Tu cometido será implementar dicho componente llevando a cabo las siguientes tareas:

Haciendo uso de los componentes de tu librería, propón una implementación del CLC indicado. Ten en cuenta que los sumadores tienen una salida que indica si ha habido desbordamiento en operación Ca2. La salida "Ov" del circuito debe tomar valor 1 si ha habido overflow en cualquiera de las operaciones realizadas. Incluye en tu hoja del examen las explicaciones que consideres necesarias. Guarda tu circuito solución como **AbsFu** con pines de entrada y de salida de 4 bits.