Versión 261C.01

Carrera: INGENIERÍA EN INFORMÁTICA

Asignatura: 3631 - Fundamentos de sistemas embebidos

Tema: Álgebra de Boole, circuitos combinatorios y aritmética binaria

Unidad: 2

Objetivo: Comprender el diseño de circuitos combinatorios, lógica de dos niveles, simplificaciones y circuitos típicos de una ALU.

Competencias a desarrollar:

- Concepción, diseño y desarrollo de proyectos de ingeniería en informática.
- Gestión, planificación, ejecución y control de proyectos de ingeniería en informática.
- Utilización de técnicas y herramientas de aplicación en la ingeniería en informática.
- Generación de desarrollos tecnológicos y/o innovaciones tecnológicas.
- Desarrollo de una actitud profesional emprendedora.
- Aprendizaje continuo
- Actuación profesional ética y responsable.
- Comunicación efectiva.
- Desempeño en equipos de trabajo.
- Identificación, formulación y resolución de problemas de ingeniería en informática

Descripción de la actividad:

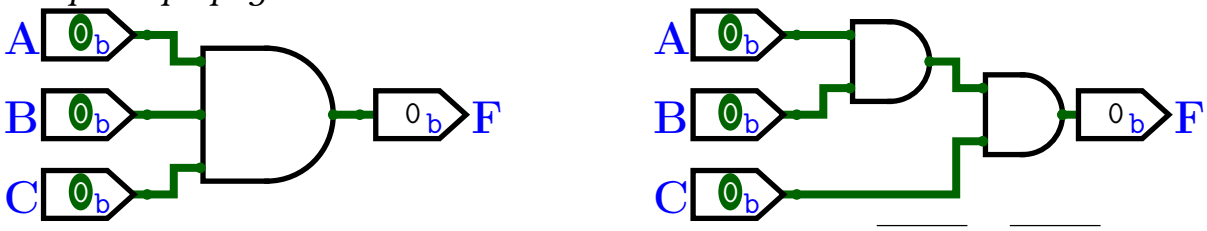
1. Tiempo estimado de resolución: 2 semana
2. Metodología: Ejercicios verificados en simuladores
3. Forma de entrega: No obligatoria
4. Metodología de corrección y feedback al alumno: Presencial y por Miel.

D- Lógica de dos niveles

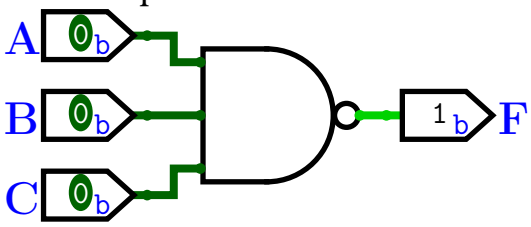
D.1 Utilizando los postulados del álgebra de Boole simplifique las siguientes funciones

- $F(A,B) = A + A.B$
- $F(A,B) = A + \overline{A}.B$
- $F(A,B,C) = (A+B).(A+C)$

D.2 El producto lógico (AND) es asociativo, es decir que $A.B.C = (A.B).C = A.(B.C)$, por ende implementar una compuerta AND de 3 entradas puede lograrse con dos compuertas AND de 2 entradas de la siguiente forma.*Nota: esto aumenta el tiempo de propagación*



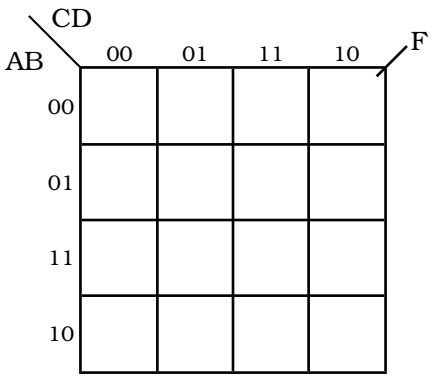
La operación NAND **NO** es asociativa. Es decir, $\overline{A.B.C} \neq \overline{\overline{A.B}.C} \neq \overline{A.\overline{B.C}}$. Recordando que NAND es una AND negada, podemos entonces plantear la equivalencia $\overline{A.B.C} = \overline{\overline{\overline{A.B}}.C}$.
Implemente en logisim-evolution una compuerta NAND de 3 entradas utilizando solo compuertas NAND de 2 entradas.

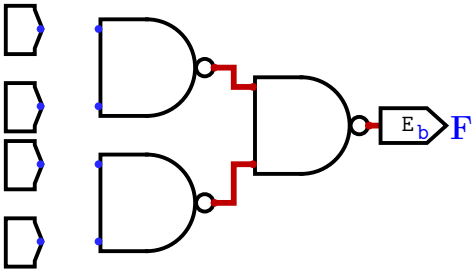


D.3 Dada la siguiente tabla de verdad para la función $F(A,B,C,D)$

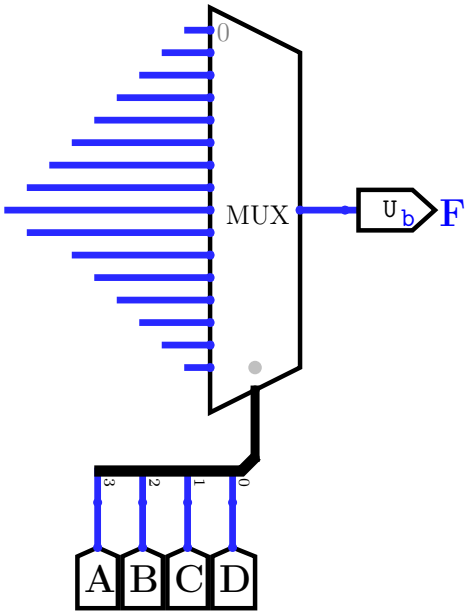
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

- Escriba la función F en primera forma canónica (suma de productos).$\mathbf{F(A,B,C,D)} =$
- Escriba la función F en segunda forma canónica (producto de sumas). $\mathbf{F(A,B,C,D)} =$
- Simplifique F (minitérminos) utilizando el mapa de Karnaugh y luego implemente el circuito simplificado utilizando solo compuertas NAND.





- Implemente el circuito utilizando un MUX de 4 entradas de selección.

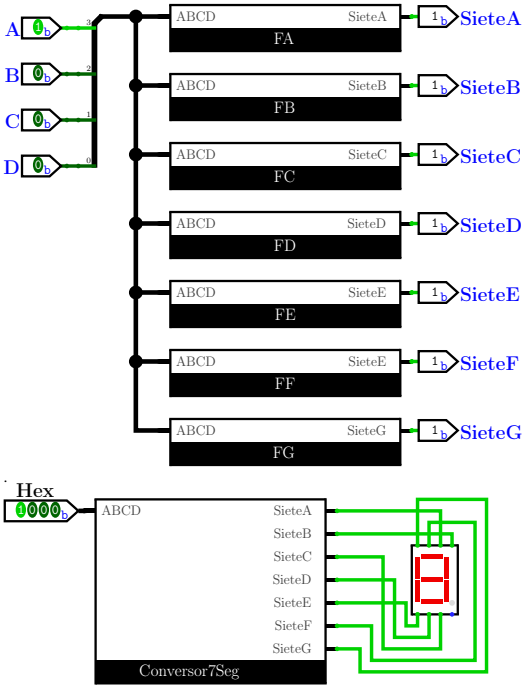


D.4 Escriba las siguientes funciones en forma primera forma canónica:

- $F(A,B,C,D) = \overline{A}.B + A.B.\overline{D}$
- $F(A,B,C,D) = B.\overline{D} + \overline{A}.B.D$

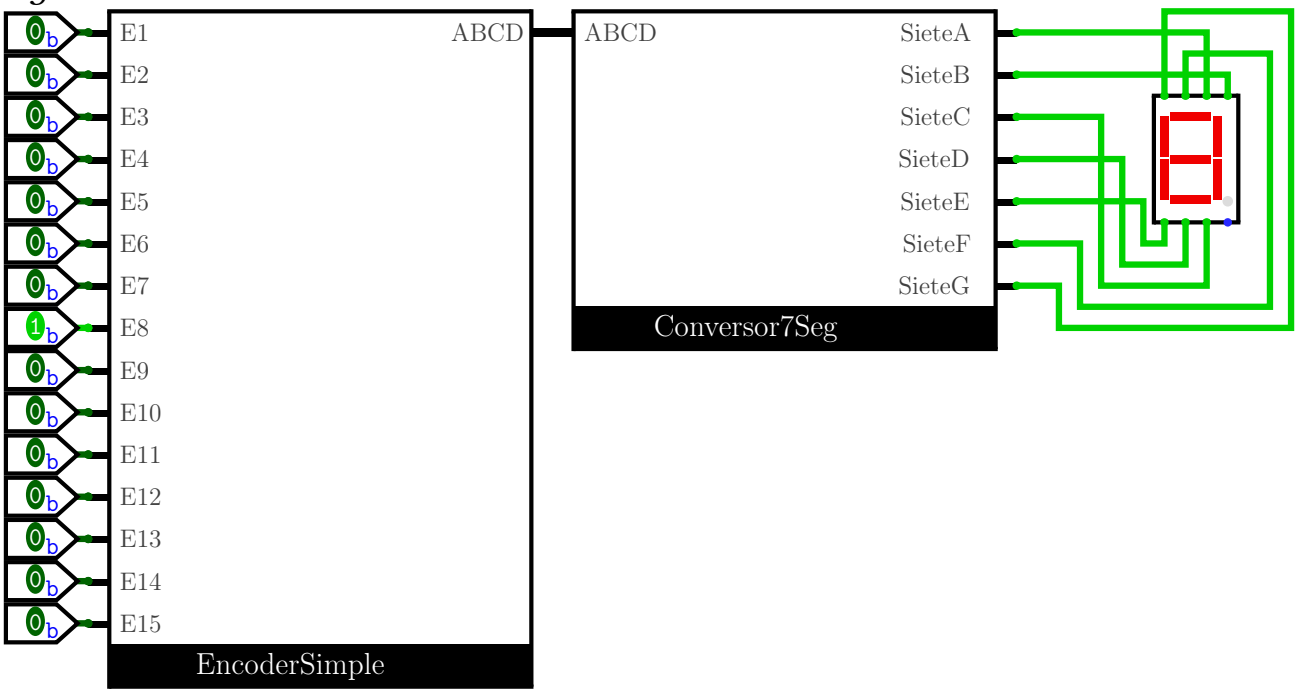
D.5 Dadas 4 variables de entrada (A,B,C,D), utilizando logisim-evolution, haga la implementación de 7 funciones de salida (FA,FB,FC,FD,FE,FF,FG). Utilice en cada caso la forma simplificada mas conveniente (suma de productos o producto de sumas). Cada función F_x debe tener una entrada de 4 bits llamada $ABCD$ y una salida $SieteX$ de un bit como se ve en la imagen. Utilice separadores (splitters) para juntar o separar conjuntos de bits.

A	B	C	D	FA	FB	FC	FD	FE	FF	FG
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	0	0	1	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

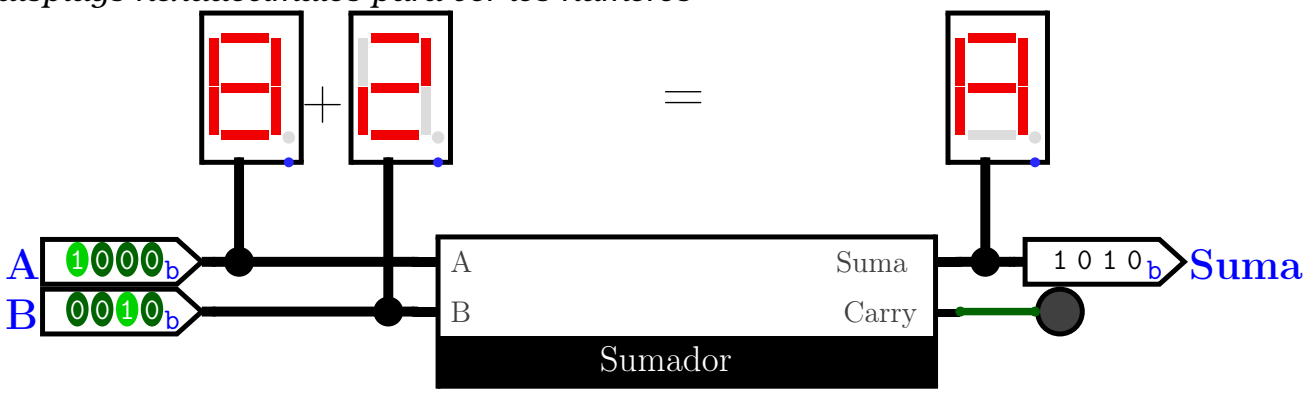


D.6 Implemente un nuevo circuito llamado EncoderSimple. El mismo se comporta como un codificador simple con 15 entradas (de un bit). Posee una salida de 4 bits que indican en binario el valor de la entrada que vale uno. Al ser un

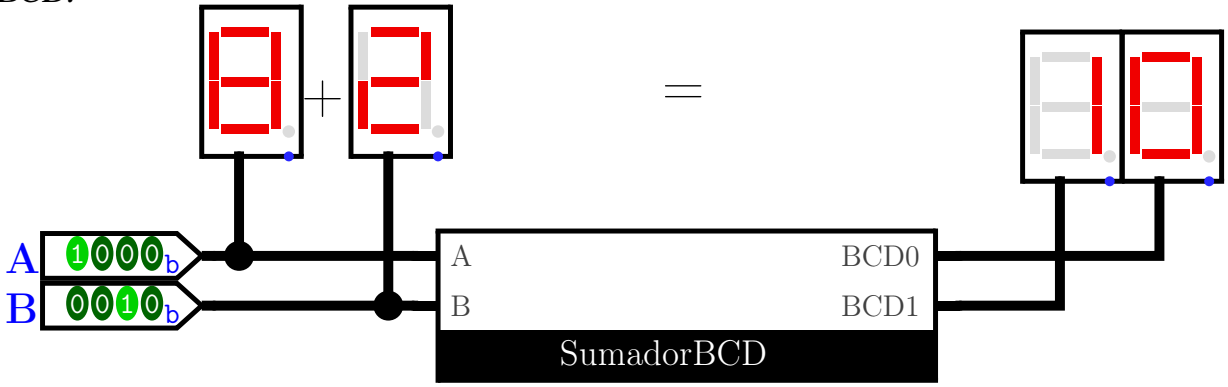
codificador simple solo una entrada a la vez puede valer uno. *Nota: En la imagen se ve pulsado E8, lo que genera una salida 1000 que equivale a 8 en el siete segmentos*



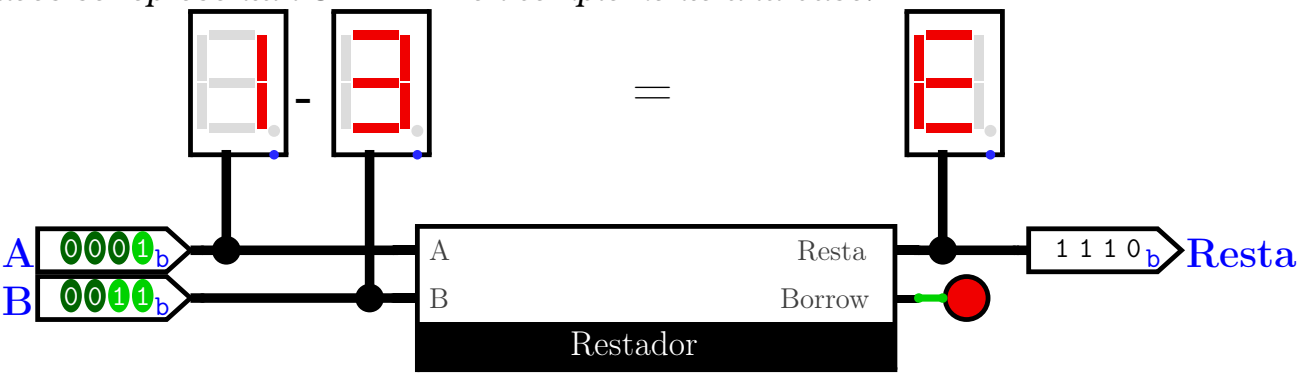
D.7 Implementar en logisim-evolution un sumador de dos números de 4 bits. Como entrada tiene los números A ($A_3A_2A_1A_0$) y B ($B_3B_2B_1B_0$). Como salida un número de 4 bits llamado Salida y un numero de un bit llamado Carry. *Nota: utilice displays hexadecimales para ver los números*



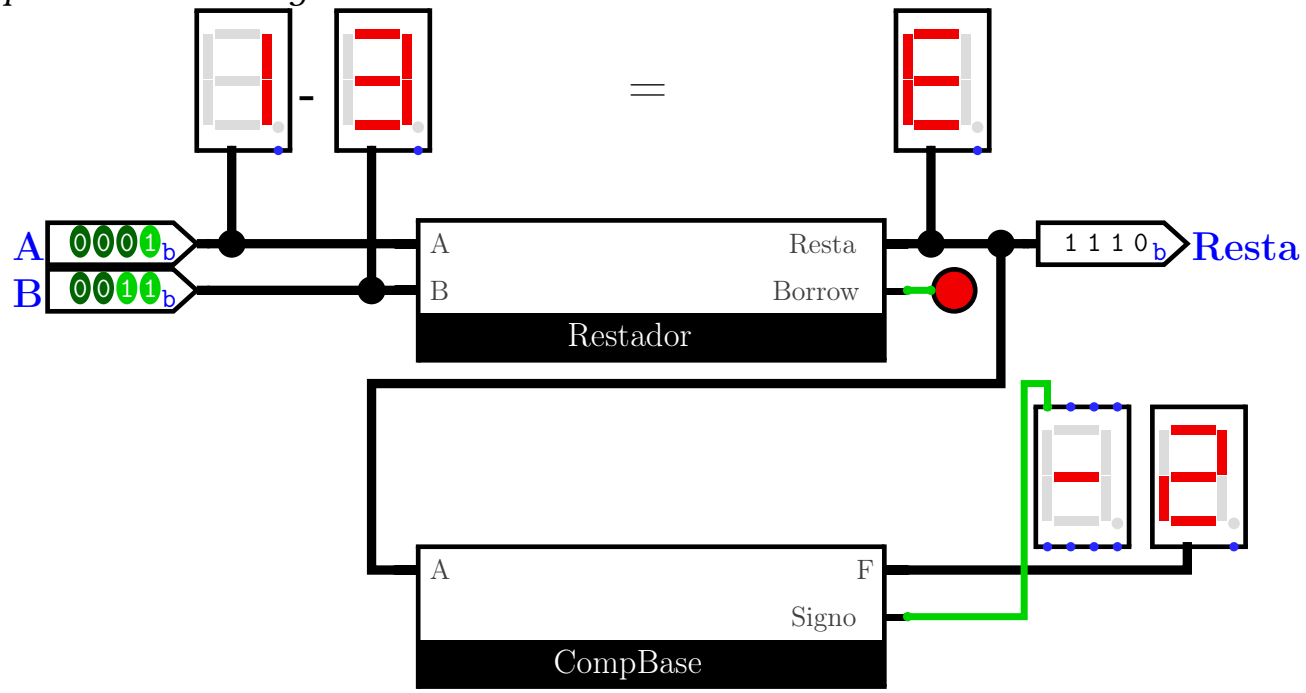
D.8 Implementar en logisim-evolution un sumador de dos números BCD. Como entrada tiene los números A ($A_3A_2A_1A_0$) y B ($B_3B_2B_1B_0$). Como salida posee dos dígitos BCD. *Nota: pruebe las combinaciones que generan un uno en el dígito más significativo, como por ejemplo 9+9=18. Solo considere dígitos válidos en BCD.*



D.9 Implementar en logisim-evolution un restador binario de dos números 4 bits. Como entrada tiene el minuendo A ($A_3A_2A_1A_0$) y el sustraendo B ($B_3B_2B_1B_0$). Como salida posee el resultado de la resta y un bit de Borrow. *Nota: los negativos se representan **SIEMPRE** en complemento a la base.*

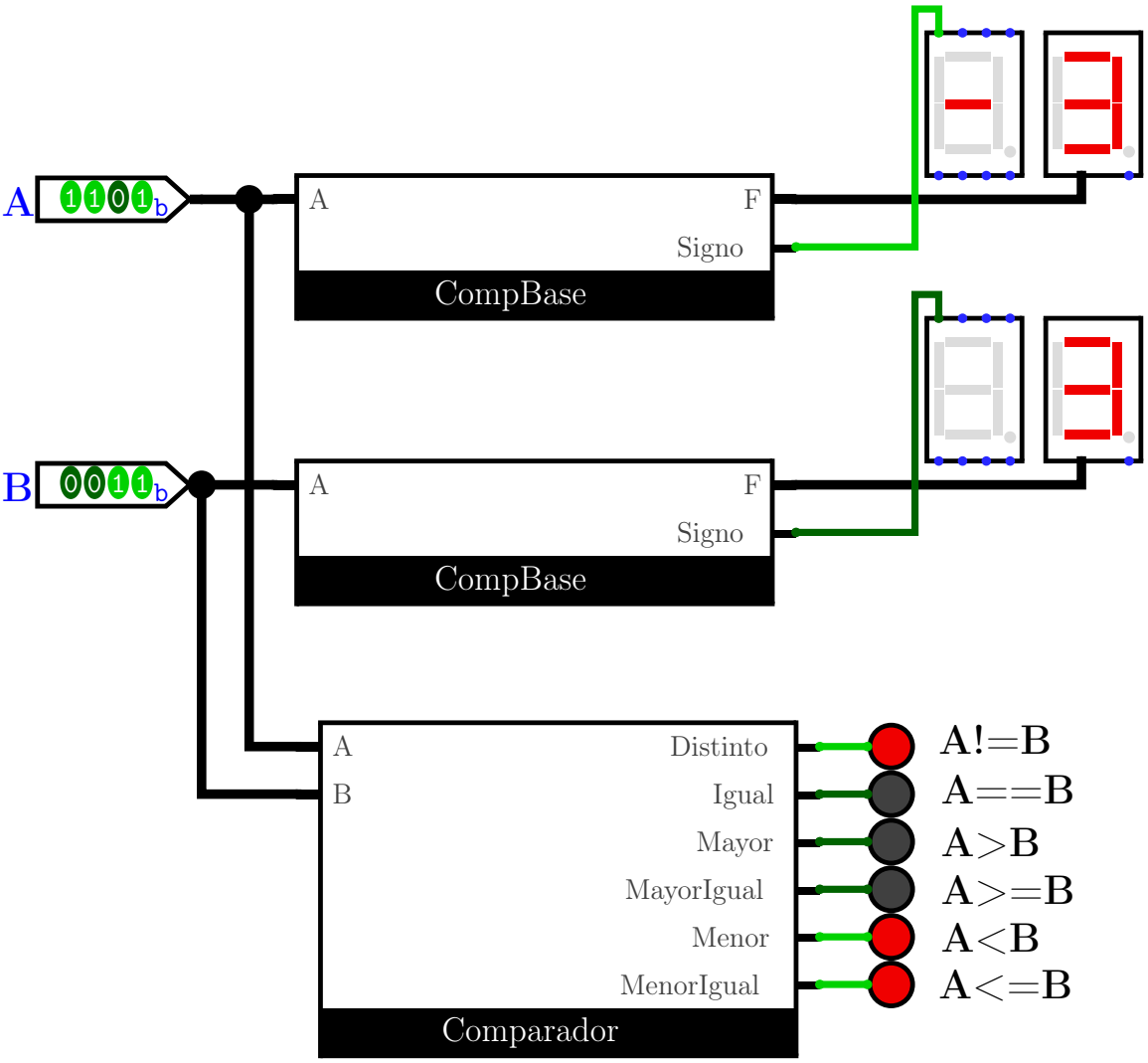


D.10 Implementar en logisim-evolution un circuito que genere el complemento a la base de un número de 4 bits **si y SOLO si** el número es negativo (o sea su bit más significativo esta en uno). En dicho caso debe generar un uno en la salida de signo. *Nota: utilice un display hexadecimal para el número y un display siete segmentos para el signo en caso de ser negativo. En el caso que el número sea positivo o cero el signo no se enciende.*



D.11 Dados un número de 4 bits minuendo A ($A_3A_2A_1A_0$) y el sustraendo B ($B_3B_2B_1B_0$), ambos representan los negativos en complemento a la base, realice un circuito que compare ambos números y que tenga 6 leds con la siguiente leyenda:

- LED A=B se enciende si A=B
- LED A<B se enciende si A<B
- LED A>=B se enciende si A>=B
- LED A > B se enciende si A > B
- LED A <= B se enciende si A <= B
- LED A != B se enciende si A != B



Recuerde sistemas de numeración. Luego de hacer A-B se sabe que:

- A=B si A-B= 0
- A!=B si A-B != 0
- A < B si SignoResultado XOR Overflow = 1
- A >=B si SignoResultado XNOR Overflow = 1
- Overflow se produce en la resta cuando el signo del minuendo y el sustraendo es distinto y el resultado tiene el signo opuesto al minuendo.

D.12 Implemente un circuito en logisim-evolution llamado DesplazaDerecha. El mismo posee una entrada de 4 bits llamada Dato y otra entrada de 2 bits llamada Cantidad. La salida del circuito son 4 bits llamados Salida. La siguiente tabla describe el funcionamiento del circuito que desplaza a la derecha el Dato según el valor de Cantidad. *Nota: Tenga en cuenta que existen 4 valores de salida posible, que son 4 formas distintas de tomar el dato y completar con ceros a la izquierda. Utilice un MUX para seleccionar la forma correcta.*

Dato	Cantidad	Salida
abcd	00	abcd
abcd	01	0abc
abcd	10	00ab
abcd	11	000a

D.13 Implemente un circuito en logisim-evolution llamado DesplazaDerechaAritmetico. El mismo posee una entrada de 4 bits llamada Dato y otra entrada de 2 bits llamada Cantidad. La salida del circuito son 4 bits llamados Salida. La siguiente tabla describe el funcionamiento del circuito que desplaza a la derecha el Dato según el valor de Cantidad. *Nota: Tenga en cuenta que el valor del bit 'a' de la entrada se repite a diferencia del circuito anterior donde se completaba con ceros*

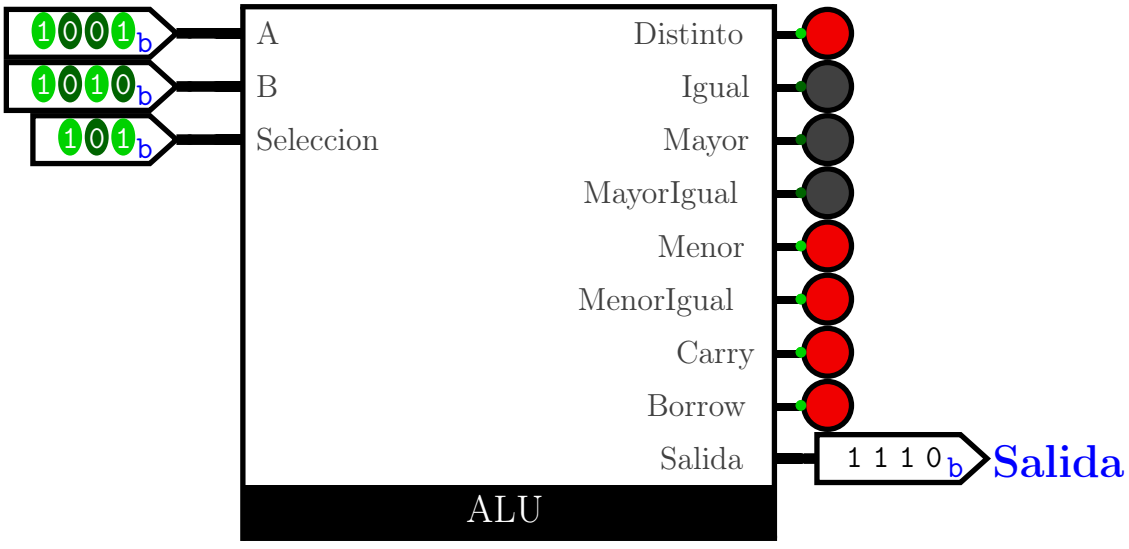
Dato	Cantidad	Salida
abcd	00	abcd
abcd	01	aabc
abcd	10	aaab
abcd	11	aaaa

D.14 Implemente un circuito en logisim-evolution llamado DesplazaIzquierda. *Nota: es similar a los anteriores solo que desplaza el número a la izquierda completando con ceros a la derecha*

Dato	Cantidad	Salida
abcd	00	abcd
abcd	01	bcd0
abcd	10	cd00
abcd	11	d000

D.15 Implemente en logisim-evolution un circuito llamado ALU. El mismo tiene como entrada un número A de 4 bits, un número B de 4 bits, y una entrada de 3 bits llamada Selección. La salida de la ALU es un número de 4 bits llamado Salida , los 6 leds del punto **D.11**, un led de Carry y un led de Borrow. El valor de salida se define dependiendo de los valores de selección:

- 000 → Salida = A - B
- 001 → Salida = A + B
- 010 → Salida = A en complemento a la base
- 011 → Salida = B en complemento a la base
- 100 → Salida = A desplazado derecha usando B1B0 como desplazamiento
- 101 → Salida = A desplazado aritmético usando B1B0 como desplazamiento
- 110 → Salida = A desplazado izquierda usando B1B0 como desplazamiento
- 111 → Salida = A AND B (bitwise A4 AND B4, A3 AND B3, etc).



D.16 Dados dos números, A (A1,A0) y B (B1, B0) de dos bits, se sabe que A representa números sin signo mientras que B representa números signados en complemento a la base. Escriba la tabla de verdad de un sumador A+B. Tenga en cuenta que A puede valer 0,1,2 o 3, mientras que B puede valer -2, -1,0,1. Elija la cantidad de bits de resultado acorde considerando que tiene que soportar sumas como 0 + (-2) , 3 + 1, etc. Implemente el circuito de la forma que crea más conveniente para cada bit del resultado (sumas de productos o producto de sumas) utilizando siempre un único tipo de compuertas. El circuito debe estar implementado con lógica de dos niveles (no encadenado sumadores).

D.17 Dados los números del punto anterior, implemente un circuito multiplicador de AxB. Tenga en cuenta que debe soportar resultados como 3 x -2 por ende elija la cantidad de bits de resultado acorde a estos valores. Comience por la tabla de verdad y luego haga la implementación en logisim-evolution