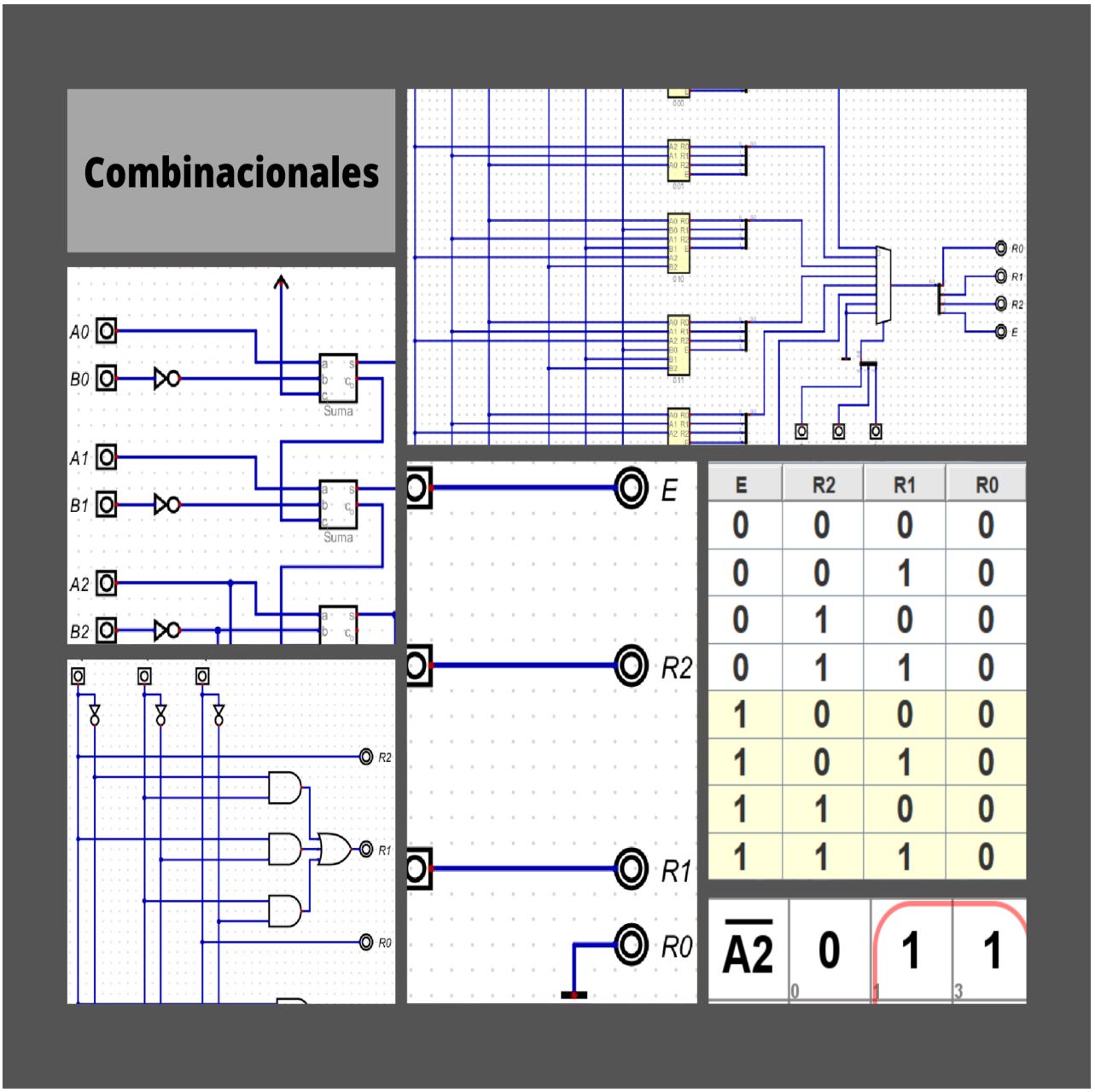
Práctica combinacionales



Hugo Serra Ruiz (43222816C)
Jose Benito Santaner Castro (45187027P)

2. Introducción

Práctica que consiste en la creación de una Unidad Aritmética-Lógica (ALU). Consta de dos operadores de entrada, cada uno de ellos de 3 bits. (A2, A1, A0 y B2, B1, B0), y 6 operaciones controladas por el código de control C (C2, C1, C0). La salida proporcionada tiene 3 bits de resultado (R2, R1, R0), y a parte un bit extra para diferentes casos especiales, definidos según cada operación. La siguiente tabla proporciona las diferentes acciones de la ALU.

C2 C1 C0	Operaciones	Aclaraciones
0 0 0	Transformación de SM a C2	La salida R es la representación en C2 del valor de entrada en SM. Salida auxiliar E=1 si A no es representable en C2. E=0 en caso contrario.
0 0 1	Transformación de C2 a SM	La salida R es la representación en SM del valor de entrada en C2. Salida auxiliar E=1 si A no es representable en SM. E=0 en caso contrario.
0 1 0	A - B	Los dos valores de la operación están representados en C2. La salida auxiliar E=1 si se produce overflow. E=0 si no se produce.
0 1 1	Comparador	Los dos valores para comparar están representados en C2. La salida será R=001 en el caso de que $a>b$; R=010 en el caso que $a=b$; R=100 en el caso que $a< b$.
1 0 0	A x 2	El resultado y la entrada A, están representados en BNSS. El bit E, será utilizado como si fuera el cuarto bit de salida, es decir, R3.
1 0 1	¿Es o no número primo?	La salida R=111 si A es un número primo. Y será R=000 en el caso de no ser primo.

En caso de no ser necesaria la salida adicional *E*, está valdrá 0 por defecto.

3. Descripción

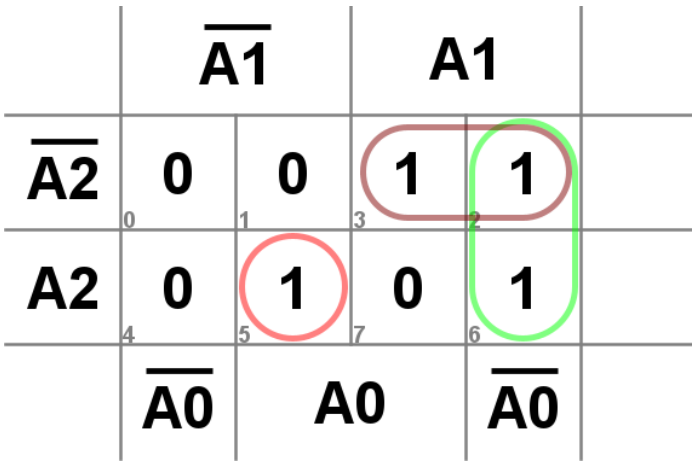
Operación 0 0 0

Creación de un circuito que lleve a cabo la transformación de SM a C2. Dada una entrada *A* de 3 bits en signo magnitud, se hace la conversión a complemento a 2 y da la salida con 3 bits y uno auxiliar (*E*). El auxiliar indica, en caso de ser 1, que el número introducido no es representable, y no se deben tener en cuenta las salidas de *R*.

Solución: la tabla de verdad que obtuvimos fué la siguiente:

A2	A1	A0	R2	R1	R0	E
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	1	1	1	0
1	1	0	1	1	0	0
1	1	1	1	0	1	0

De aquí, rápidamente notamos que la salida *R0* se corresponde con la entrada *A0* respectivamente. Por tanto, la función mínima a implementar es *R0=A0*. La entrada R2 podemos ver que vale uno cuando el valor A2 sea positivo y también uno de A1 o A2, al menos Para la salida *R1* no pudimos extraer directamente su función mínima, así que la convertimos en la siguiente mapa de Karnaugh:



De dónde obtuvimos la función mínima *R1=A0A1'A2 + A0'A1 + A1A2'*.

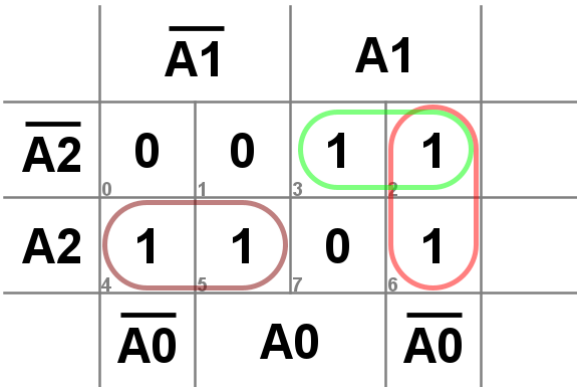
Operación 0 0 1

Creación de un circuito que lleve a cabo la transformación de C2 a SM. Dada una entrada *A* de 3 bits en complemento a 2, se hace la conversión a signo magnitud y da la salida con 3 bits y uno auxiliar (*E*). El auxiliar indica, en caso de ser 1, que el número introducido no es representable.

Solución: la tabla de verdad que obtuvimos fué la siguiente:

A2	A1	A0	R2	R1	R0	E
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	1	1	0
1	0	0	1	1	0	1
1	0	1	1	1	1	0
1	1	0	1	1	0	0
1	1	1	1	0	1	0

De aquí, rápidamente notamos que las salidas *R2* y *R0* se corresponden con las entradas *A2* y *A0* respectivamente. Por tanto, las funciones mínimas a implementar son *R2=A2* y *R0=A0*. Para la salida *E*, sólo tenemos un caso donde valga 1, quedándonos su función mínima como *E=A2A1'A0'*. Para la salida *R1* no pudimos extraer directamente su función mínima, así que la convertimos en la siguiente mapa de Karnaugh:



De dónde obtuvimos la función mínima *R1=A0'A1 + A1A2' + A1'A2*.

Operación 0 1 0

Creación de un circuito que lleva a cabo la operación *A* - *B*. Se piden dos entradas (*A* y *B*) de 3 bits en complemento a 2 y calcula la resta. La salida que mostrará será de 3 bits y uno auxiliar (*E*). Si en el resultado hay overflow, esta salida *E* será 1. Y si por lo contrario no se produce overflow, saldrá el valor 0.

Solución: una resta en C2 de A-B es lo mismo que A + (-B) como sí fuera BNSS. Para cambiar un signo de un número, lo que debemos hacer es cambiar cada uno de sus bits y sumar uno. Luego, lo que hicimos fué sumar bit a bit, siendo el C_{out} de uno el C_{in} del siguiente, utilizando la operación aritmética de suma ya incluida en *Digital*.

Para determinar sí la operación tiene o no overflow, debemos distinguir 2 casos. El primero es sí los números a sumar son de distinto signo, dónde nunca puede haber overflow. En caso de ser del mismo

signo, debemos comparar este con el signo obtenido de la operación, sí mantienen el mismo signo no hay overflow, pero sí cuando son distintos.

Operación 0 1 1

Creación de un circuito comparador de dos números representados en C2. Se mostrará la salida según las siguientes comparaciones: $R = 0\ 0\ 1$ si $A > B$, $R = 0\ 1\ 0$ si $A = B$ y $R = 1\ 0\ 0$ si $A < B$.

Solución: en el programa usado para el desarrollo de los circuitos, *Digital*, ya viene incluido un módulo comparador, con opción de que compare 2 números en complemento a 2, y activando una de sus 3 salidas, dependiendo de sí un número *a* és mayor, igual o menor a otro. Con esto, lo único que debemos hacer es introducir los valores de *a* en la entrada *A* y *B* en la entrada *b*, y conectar las salidas de *a>b* con R1, *a=* con *R2* y *a<b* con *R2*, además de conectar la salida *E* a tierra.

Operación 1 0 0

Creación de un circuito que realiza la operación A x 2. Se pide una entrada de 3 bits en BNSS, realiza la operación y muestra la salida con un bit auxiliar, para representar un cuarto bit de salida. Por tanto la salida tendrá 4 bits y estará representada en BNSS.

Solución: la tabla de verdad que obtuvimos fué la siguiente:

A2	A1	A0	E	R2	R1	R0
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	1	1	0
1	0	0	1	0	0	0
1	0	1	1	0	1	0
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Vemos que la salida R0 siempre vale 0, puesto que el primer bit de un número en binario sólo puede valer 1 o 0, y el resto todos números pares, múltiplos de 2. Entonces, que el primer bit valga 1 o 0 nos indica sí el número es impar o par respectivamente, pero como todos los número obtenidos en la operación son múltiplos de 2, R0 siempre valdrá 0. Para el resto de casos vimos una relación muy curiosa, las salidas E, R2 y R1 se corresponden directamente con los valores de entrada A2, A1 y A0 respectivamente.

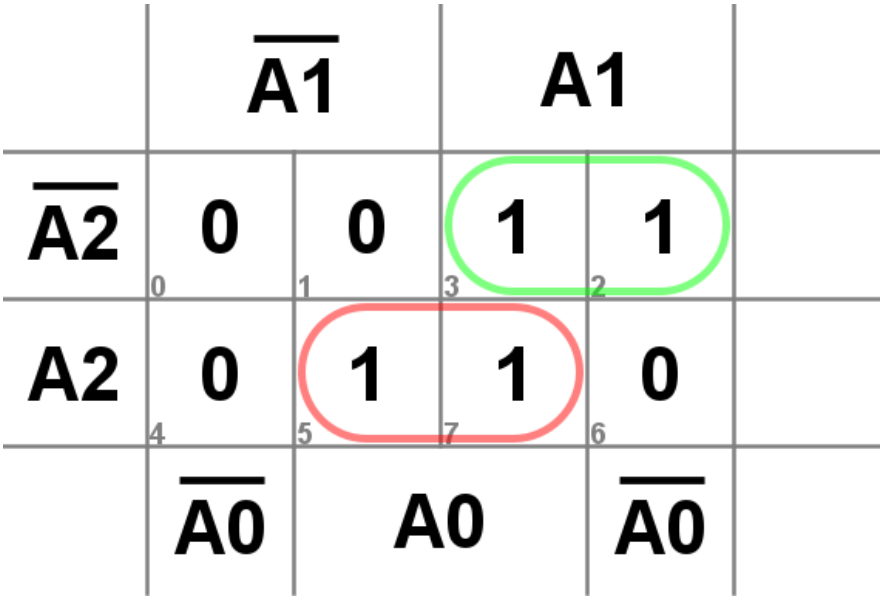
Operación 1 0 1

Creación de un circuito que dice si un número de entrada, representado en BNSS es o no primo. Mostrará una salida de R = 1 1 1, en caso de que la entrada sea prima y de R = 0 0 0 en caso de que no sea primo.

Solución: Observando la tabla de verdad

A2	A1	A0	R2	R1	R0	E
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	1	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	1	1	1	1	0

Vemos que R2, R1 y R0 son siempre iguales ya que si es 1, el número es primo y si es 0 no es un número primo el introducido. Entonces observamos el mapa de Karnaugh, obtuvimos que podemos sacar la siguiente función mínima. La función mínima que obtuvimos fué R0=R1=R2=A0A2 + A1A2'

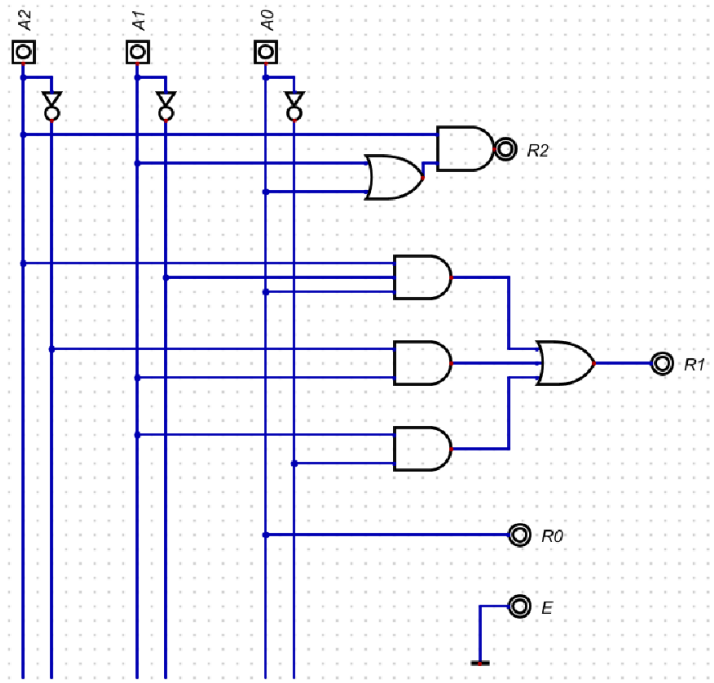


Circuito ALU

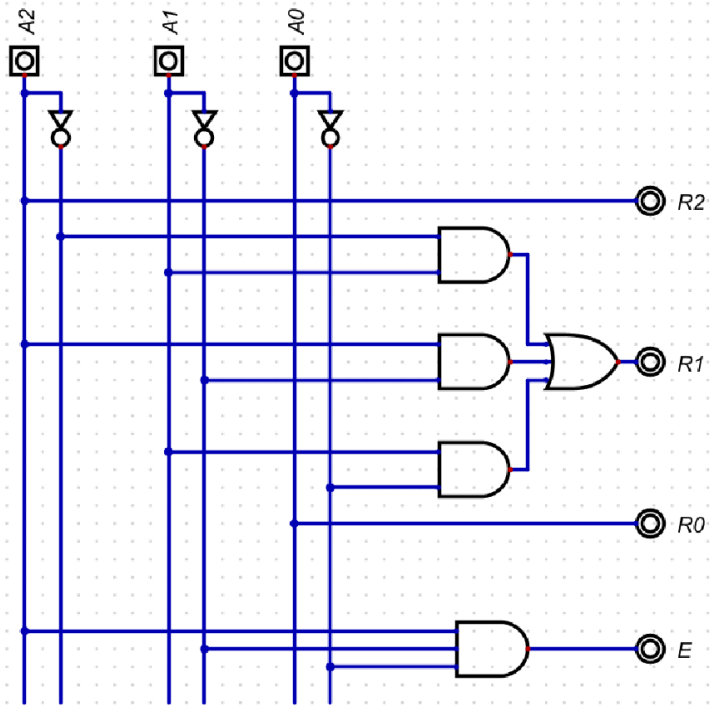
Creación de un circuito ALU (Unidad Aritmética-Lógica) que contiene las 6 diferentes operaciones. Este circuito está formado por las 6 entradas de los números *A* y *B*, conectados todos a las diferentes operaciones descritas con anterioridad. Los resultados de estas están conectados a un Multiplexor de 3 bits de selección, que se corresponden a los bits de *C*. Al usar 3 bits de selección, el multiplexor debe recibir 8 entradas, de las cuales 6 se corresponden con los resultados de las operaciones y los 2 restantes están conectados a tierra. Con esto, al elegir una operación que no exista (110 y 111), la ALU devolverá en todos los valores de *R* y *E* 0.

4. Circuitos

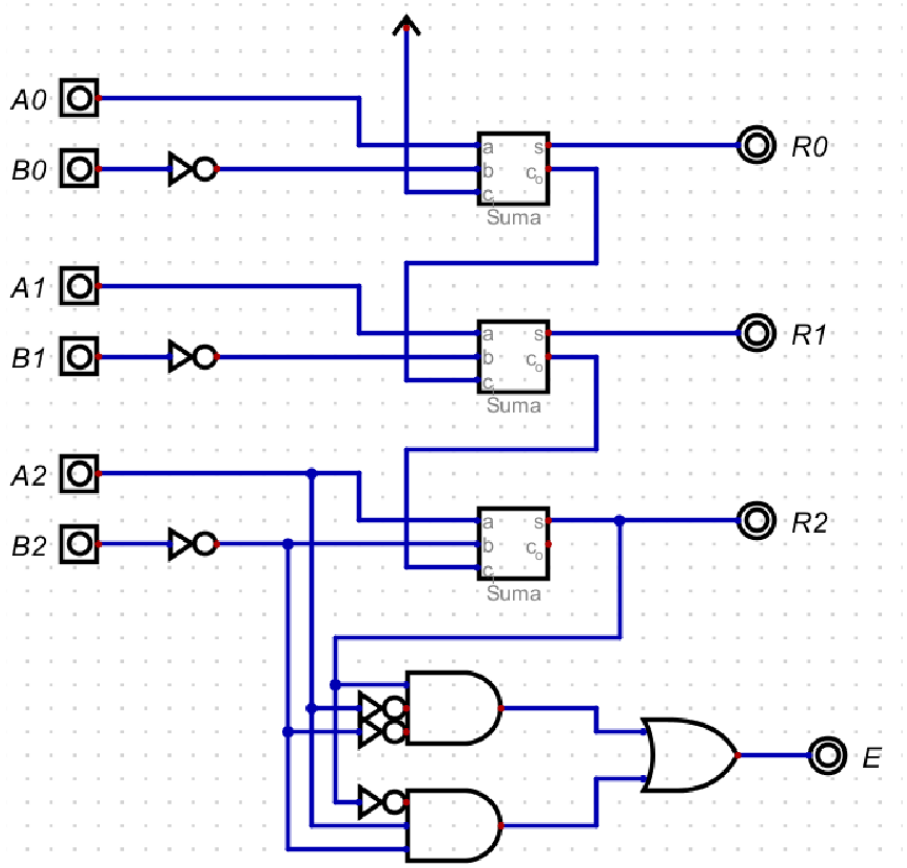
Operación 0 0 0



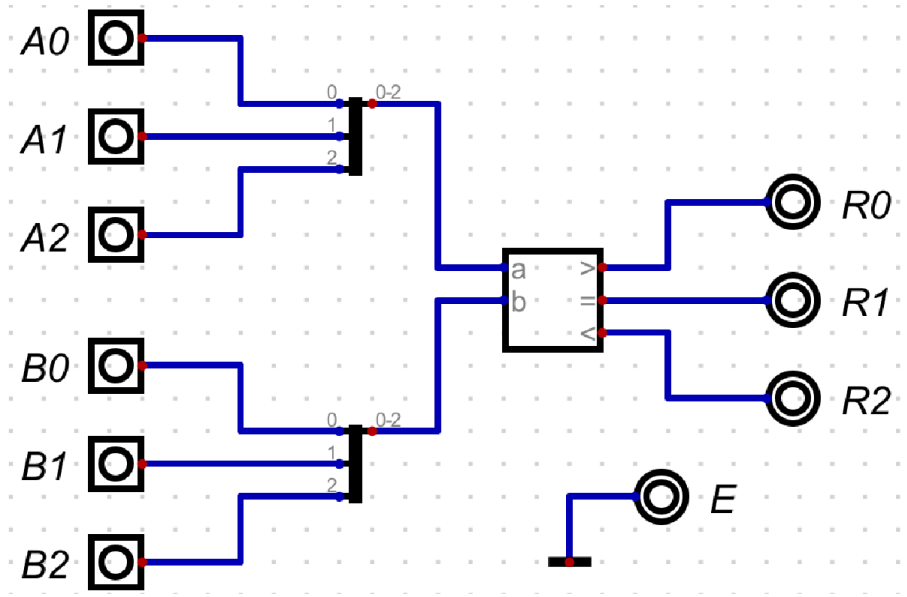
Operación 0 0 1



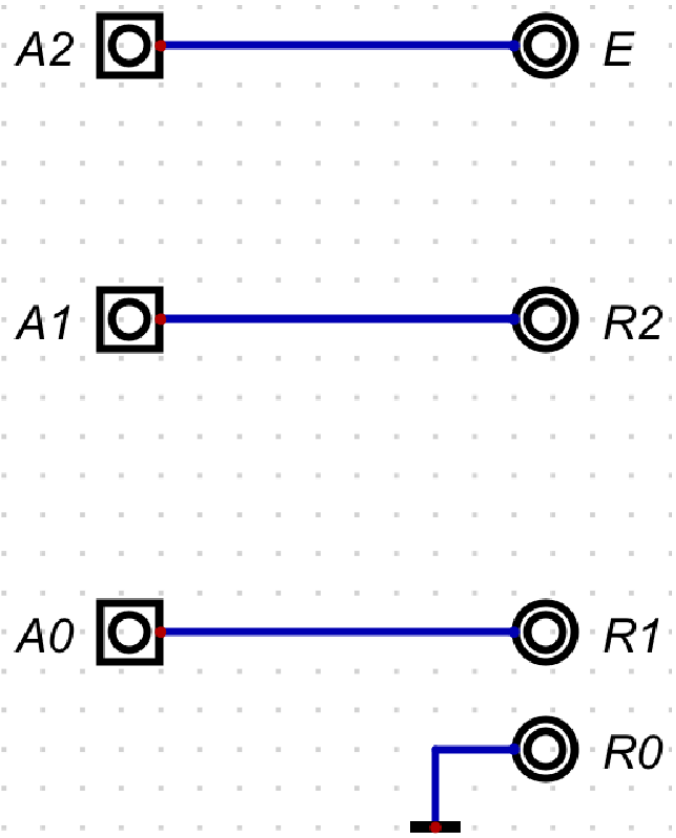
Operación 0 1 0



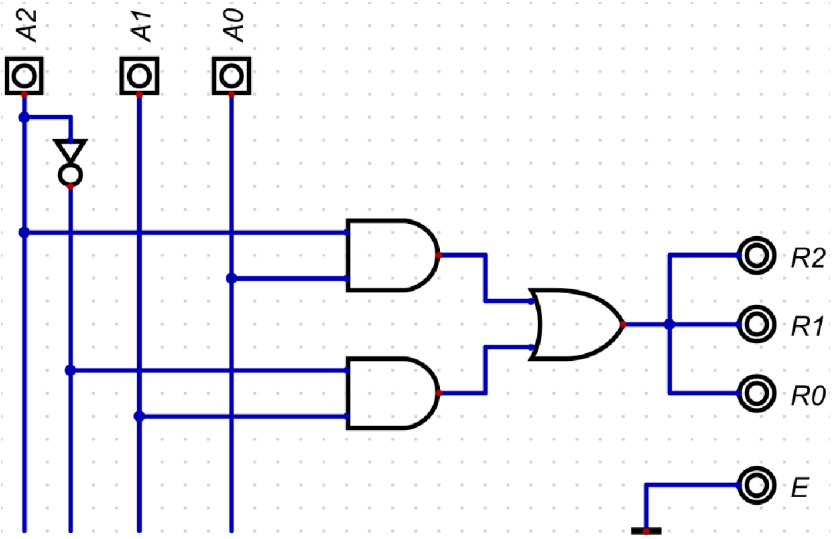
Operación 0 1 1



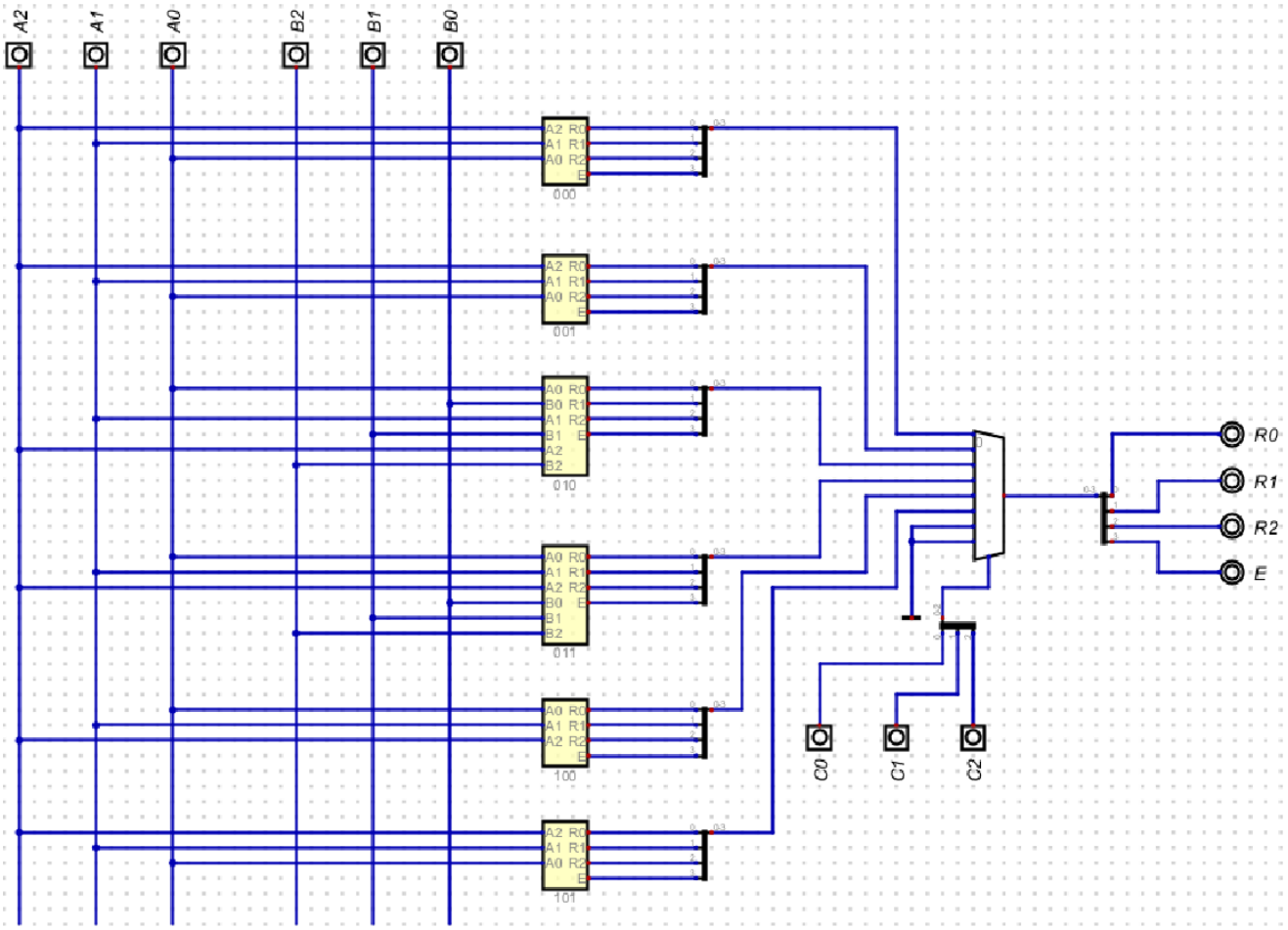
Operación 1 0 0



Operación 1 0 1



Circuito ALU



5. Juego de pruebas

El propio programa Digital incluye una función de comprobar circuitos. En este, deberemos incluir todas las entradas posibles y las salidas que nosotros esperamos que salgan. Como podemos comprobar en las siguientes imágenes, no nos da error en ningún caso, indicando que cada uno de los circuitos es correcto. Las operaciones 010 y 011 en cambio, requieren analizar muchas combinaciones posibles, e implementar ambas tablas de verdad no nos parece un método óptimo. En estos, comprobaremos unos cuantos casos para obtener todas las salidas posibles.

Operación 0 0 0

SM - C2 pasado

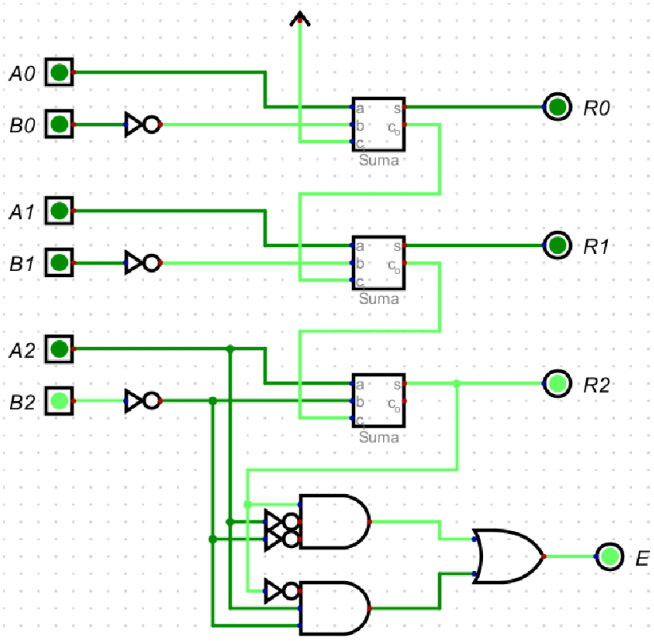
	A2	A1	A0	E	R2	R1	R0
L2	1	1	1	0	1	0	1
L3	1	1	0	0	1	1	0
L4	1	0	1	0	1	1	1
L5	1	0	0	0	0	0	0
L6	0	0	0	0	0	0	0
L7	0	0	1	0	0	0	1
L8	0	1	0	0	0	1	0
L9	0	1	1	0	0	1	1

Operación 0 0 1

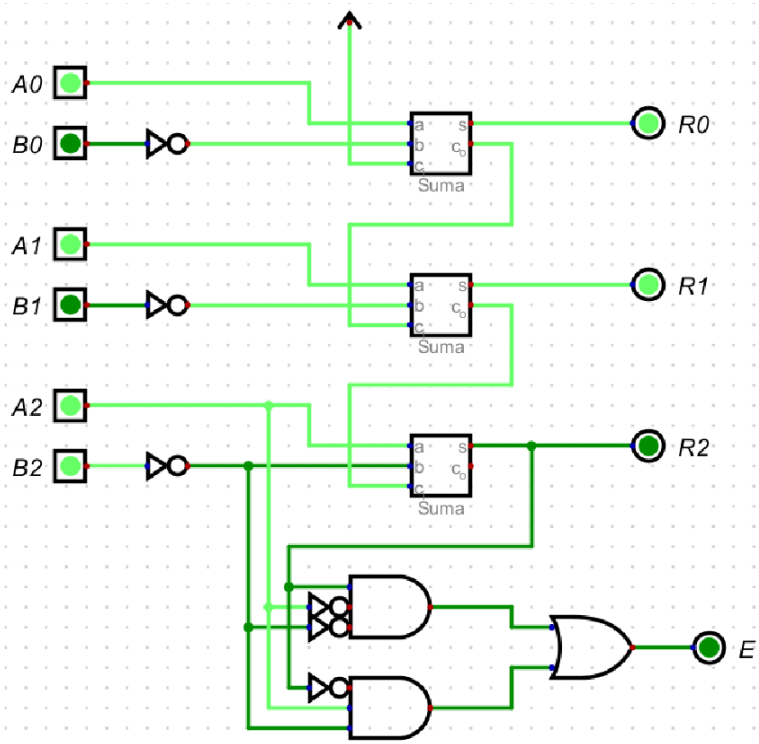
C2 - SM pasado

	A2	A1	A0	E	R2	R1	R0
L2	1	0	0	1	1	1	0
L3	1	0	1	0	1	1	1
L4	1	1	0	0	1	1	0
L5	1	1	1	0	1	0	1
L6	0	0	0	0	0	0	0
L7	0	0	1	0	0	0	1
L8	0	1	0	0	0	1	0
L9	0	1	1	0	0	1	1

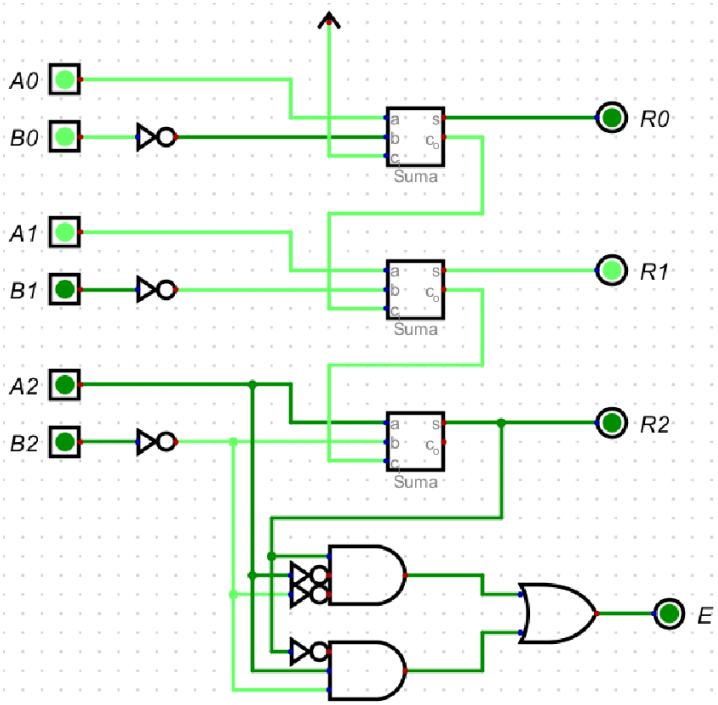
Operación 0 1 0



Para la operación 0-(-4) vemos que obtenemos overflow. Al realizar la operación, debería quedarnos 4, pero este número no es representable en complemento a 2 con sólo 3 bits. El rango de número posibles va desde -4 hasta el 3.



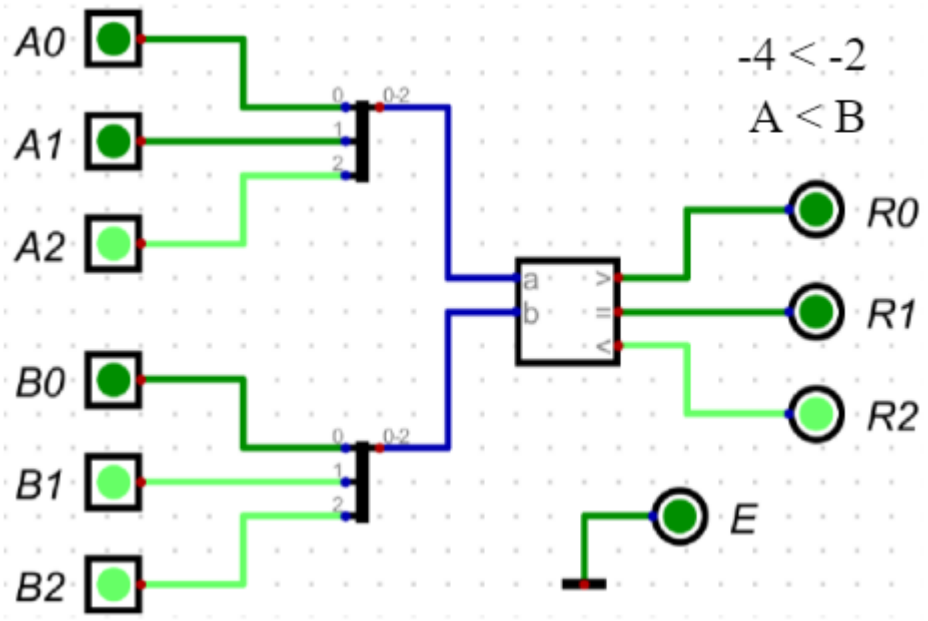
En este caso, la operación es -1-(-4), que es igual a 3, como bien aparece en las salidas de *R*. Al ser un número representable en complemento a 2 con los 3 bits que disponemos, vemos que no hay overflow.

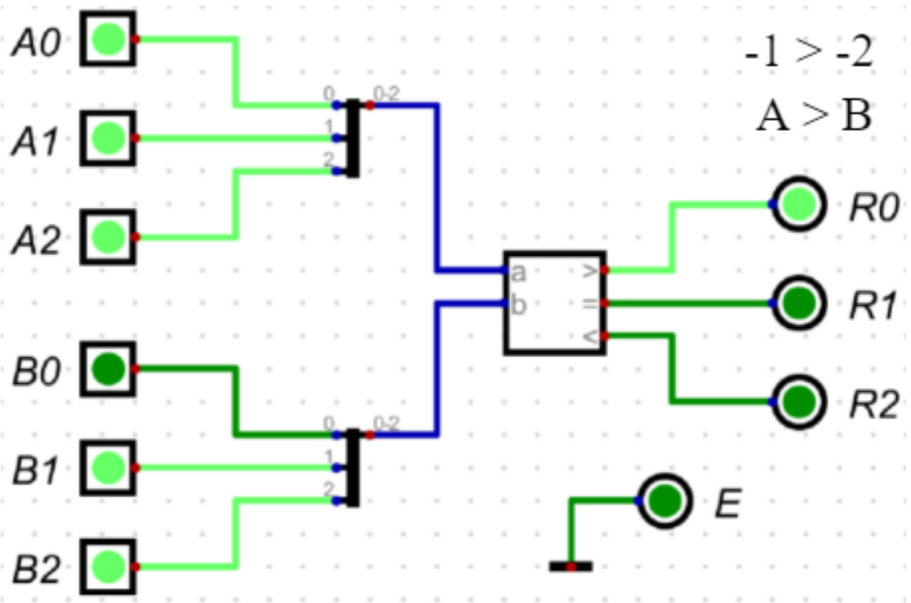


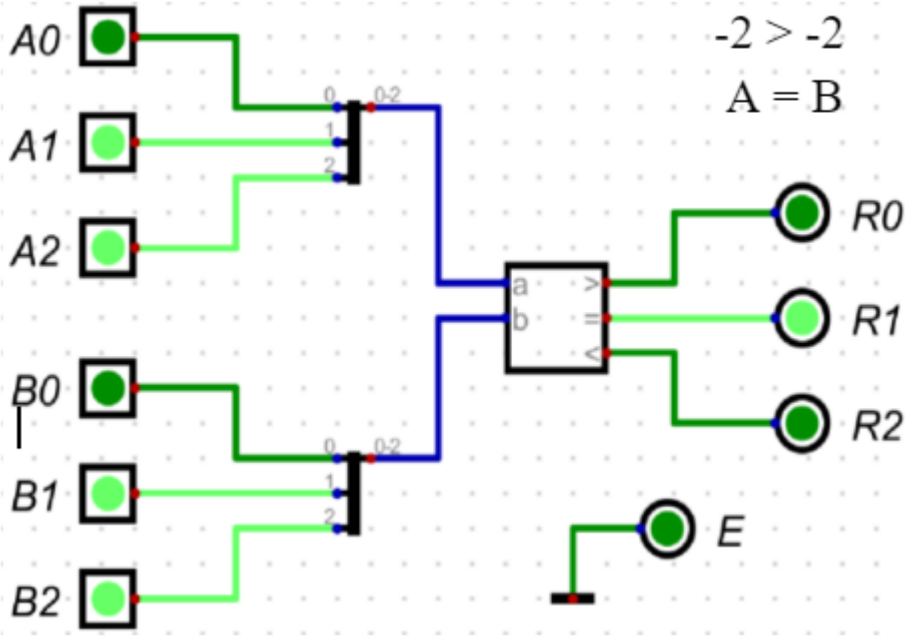
En este caso, la operación es 3-1, que es igual a 2, como bien aparece en las salidas de *R*. Al ser un número representable en complemento a 2 con los 3 bits que disponemos, podemos comprobar que efectivamente no hay overflow.

Operación 0 1 1

En esta operación elegiremos un caso dónde *A*<*B*, otro dónde *A*=*B* y por último uno dónde *A*>*B*.




A0
A1
A2
B0
B1
B2
0
1
2
0-2
a
b
>
=
<
R0
R1
R2
E
-1 > -2
A > B


A0
A1
A2
B0
B1
B2
0
1
2
0-2
a
b
>
=
<
R0
R1
R2
E
-2 > -2
A = B

Operación 1 0 0

A x 2 pasado

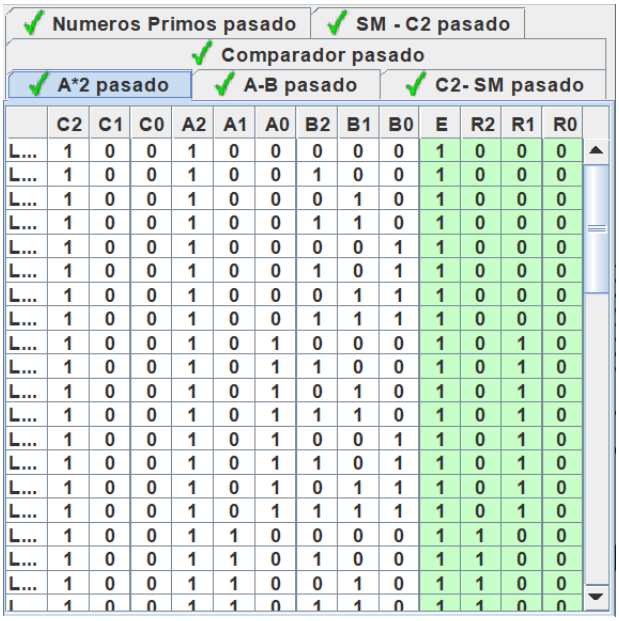
	A2	A1	A0	E	R2	R1	R0
L2	1	0	0	1	0	0	0
L3	1	0	1	1	0	1	0
L4	1	1	0	1	1	0	0
L5	1	1	1	1	1	1	0
L6	0	0	0	0	0	0	0
L7	0	0	1	0	0	1	0
L8	0	1	0	0	1	0	0
L9	0	1	1	0	1	1	0

Operación 1 0 1

NUMEROS PRIMOS pasado

	A2	A1	A0	E	R2	R1	R0
L2	0	0	0	0	0	0	0
L3	0	0	1	0	0	0	0
L4	0	1	0	0	1	1	1
L5	0	1	1	0	1	1	1
L6	1	0	0	0	0	0	0
L7	1	0	1	0	1	1	1
L8	1	1	0	0	0	0	0
L9	1	1	1	0	1	1	1

Circuito ALU



	C2	C1	C0	A2	A1	A0	B2	B1	B0	E	R2	R1	R0
L...	1	0	0	1	0	0	0	0	0	1	0	0	0
L...	1	0	0	1	0	0	1	0	0	1	0	0	0
L...	1	0	0	1	0	0	0	1	0	1	0	0	0
L...	1	0	0	1	0	0	1	1	0	1	0	0	0
L...	1	0	0	1	0	0	0	0	1	1	0	0	0
L...	1	0	0	1	0	0	1	0	1	1	0	0	0
L...	1	0	0	1	0	0	0	1	1	1	0	0	0
L...	1	0	0	1	0	0	1	1	1	1	0	0	0
L...	1	0	0	1	0	1	0	0	0	1	0	1	0
L...	1	0	0	1	0	1	1	0	0	1	0	1	0
L...	1	0	0	1	0	1	0	1	0	1	0	1	0
L...	1	0	0	1	0	1	1	1	0	1	0	1	0
L...	1	0	0	1	0	1	0	0	1	1	0	1	0
L...	1	0	0	1	0	1	1	0	1	1	0	1	0
L...	1	0	0	1	0	1	0	1	1	1	0	1	0
L...	1	0	0	1	0	1	1	1	1	1	0	1	0
L...	1	0	0	1	1	0	0	0	0	1	1	0	0
L...	1	0	0	1	1	0	1	0	0	1	1	0	0
L...	1	0	0	1	1	0	0	1	0	1	1	0	0
L...	1	0	0	1	1	0	1	1	0	1	1	0	0

6. Conclusiones

Con este trabajo, lo que más hemos aprendido es analizar los resultados que obtenemos. Aquellas operaciones dónde hemos usado tablas de verdad no nos ha sido necesario calcular las funciones mínimas de cada salida ya sea con mapas de Karnaugh o simplificando algebraicamente, ahorrando mucho tiempo y esfuerzo en el desarrollo.

También, al usar varios componentes nuevos no usados en clase como el comparador, hemos tenido que buscar información externa en internet, ayudándonos así a adquirir habilidades de búsqueda y comprobación de la información obtenida.

Uno de los mayores problemas que tuvimos fué a la hora de hacer la ALU usando el resto de circuitos como incrustaciones. Ya que las entradas y salidas no se correspondían con el orden que nosotros queríamos, dándonos resultados incorrectos, hasta que descubrimos una configuración de *Digital* para poder reordenar todo. De esto nos llevamos las habilidades para explorar e indagar más en las posibilidades de los programas que usamos

Otro problema al construir la ALU fué la unión de cables en uno sólo y, sobre todo, la división posterior a estos. El orden de las cosas volvió a tener mucha importancia en esto y para solucionarlo tuvimos que aprender a ser tan claros como podamos, para no confundirnos a la hora de conectar ningún cable.