PROYECTO ARQUITECTURA DE COMPUTADORAS

ETAPA 2

14/05/2024

Integrante 1: [Giacomodonato Giulia - 142049]
Integrante 2: [Kreczmer Tomás - 140009]
Logisim-Evolution: [3.8.0]

Índice

Introducción

Este proyecto de Arquitectura de Computadoras se centra en la implementación de un circuito controlador de estados basado en el diagrama proporcionado. La implementación se realizó utilizando la herramienta Logisim-Evolution.

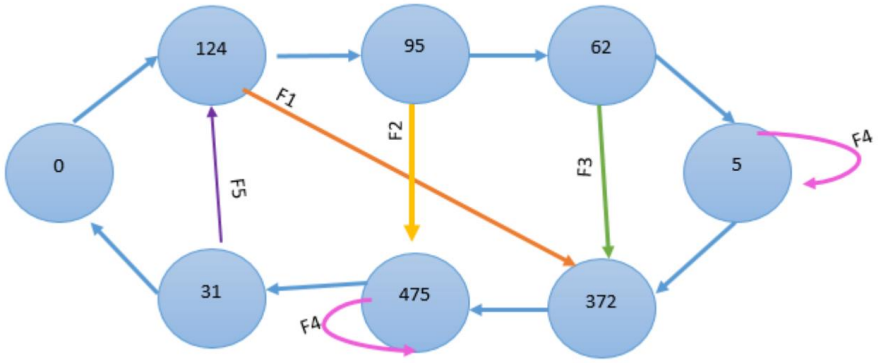


Figura 1: Grafo de estados

La idea general que utilizamos para resolver el problema fue utilizar un contador de 9 bits, separarlos en la entrada y en la salida, determinar en qué estado está y a cuál estado debe saltar para luego conformar el número siguiente bit por bit según corresponda.

Durante todo el proyecto, intentamos mantener la organización y claridad en el diseño para garantizar el correcto funcionamiento del circuito y facilitar su comprensión.

Etapa 1

Descripción del circuito

Funciones

Minimizamos las funciones mediante el método de Karnaugh y agrupamos los mintérminos adyacentes que forman implicantes primos esenciales y no esenciales. Dos términos productos están en casillero adyacentes si difieren en exactamente un literal, que aparece complementado en uno y sin complementar en el otro (código Gray).

Tanto como para F1, F2 y F3 utilizamos como líneas de selección a las entradas A y B para los multiplexores.

- F1(A, B, C, D, E) = Σ (3, 11, 12, 15, 16, 20, 21, 23, 24, 28), Σ_{op} (0)

AB\CDE	000	001	011	010	110	111	101	100
00	*		1					
01			1			1		1
11	1							1
10	1					1	1	1

Las funciones residuo que se generan son las siguientes:
00: C'DE
01: C'DE + CDE + CD'E'
11: D'E'
10: CE + D'E'

- F2(A, B, C, D, E) = Σ (0, 4, 10, 11, 15, 16, 19, 20, 23, 27, 31), Σ_{op} (2, 5, 13)

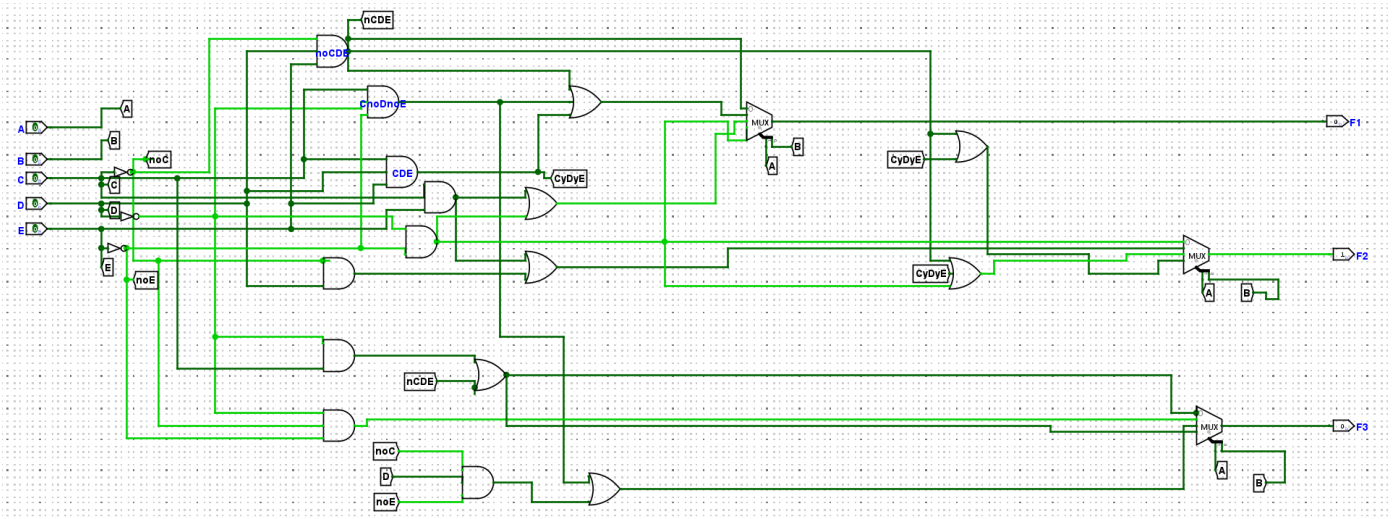
AB\CDE	000	001	011	010	110	111	101	100
00	1			*			*	1
01			1	1		1	*	
11			1			1		
10	1		1			1		1

Las funciones residuo que se generan son las siguientes:
00: D'E'
01: C'D + CE
11: C'DE + CDE
10: D'E' + C'DE + CDE

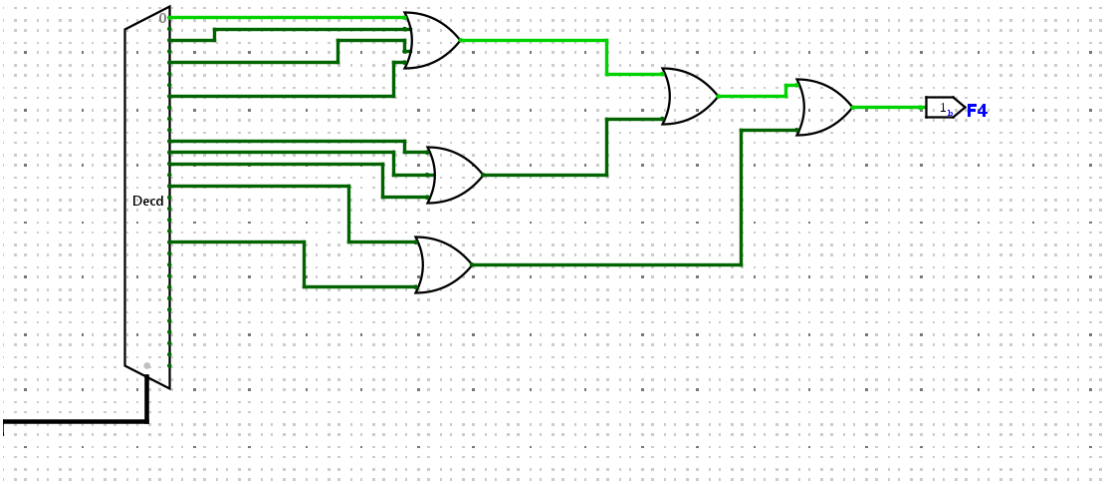
- F3(A, B, C, D, E) = Σ (3, 4, 5, 8, 18, 20, 27, 28, 29), Σ_{op} (31)

AB\CDE	000	001	011	010	110	111	101	100
00			1				1	1
01	1							
11			1			*	1	1
10				1				1

Las funciones residuo que se generan son las siguientes:
00: CD' + C'DE
01: C'D'E'
11: CD' + C'DE
10: C'DE' + CD'E'



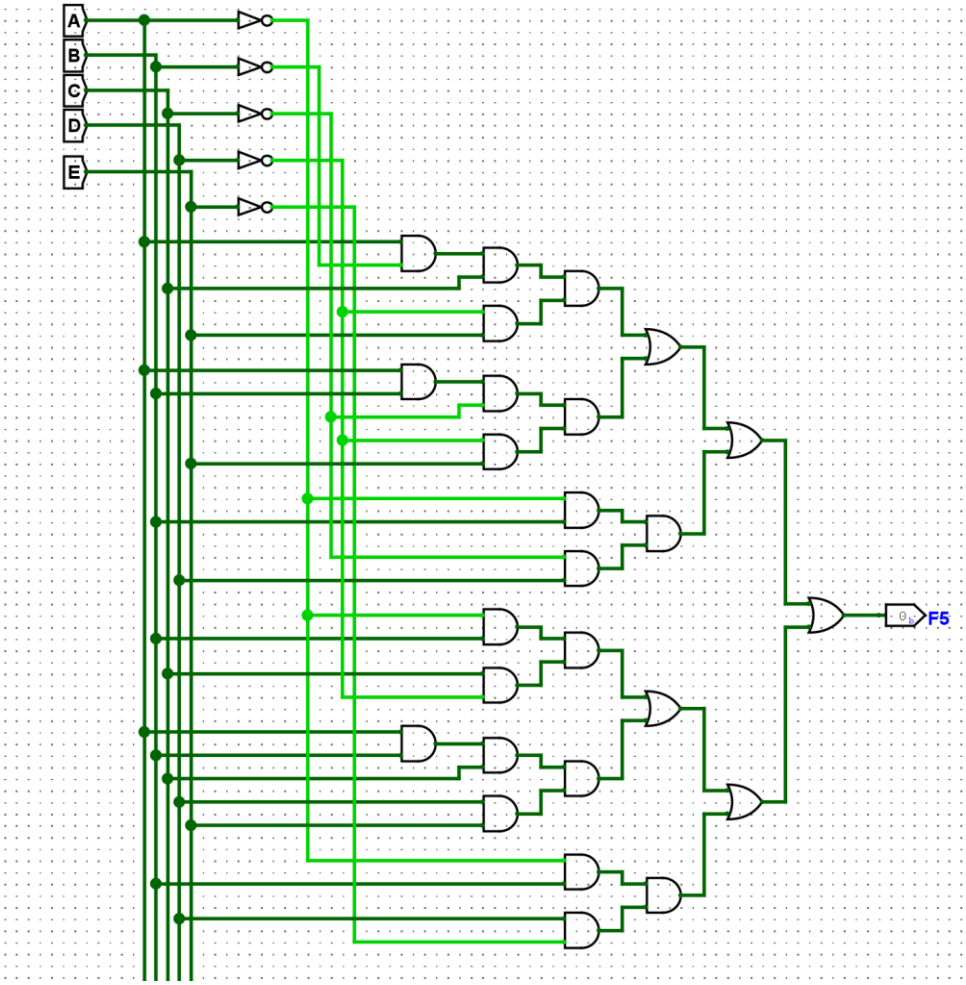
- F4(A, B, C, D, E) = Σ (0,2, 4, 7, 11, 12, 13, 15, 20), Σ_{op} (14)
Para computar F4 ingresamos A, B, C, D, y E al decodificador, elegimos las salidas que representan los números en los que F4 debe devolver 1 según su descripción, y las unimos con compuertas OR.



- F5(A, B, C, D, E) = Σ (10, 11, 12, 13, 14, 21, 23, 25, 31)
  Para F5 no tenemos líneas de selección ya que debemos utilizar compuertas.

AB\CDE	000	001	011	010	110	111	101	100
00								
01			1	1	1		1	1
11		1				1		
10							1	

Luego, obtenemos F5 = AB'CD'E + ABC'D'E + A'BC'D + A'BCD' + ABCDE + A'BDE' y escribimos la función mediante compuertas.



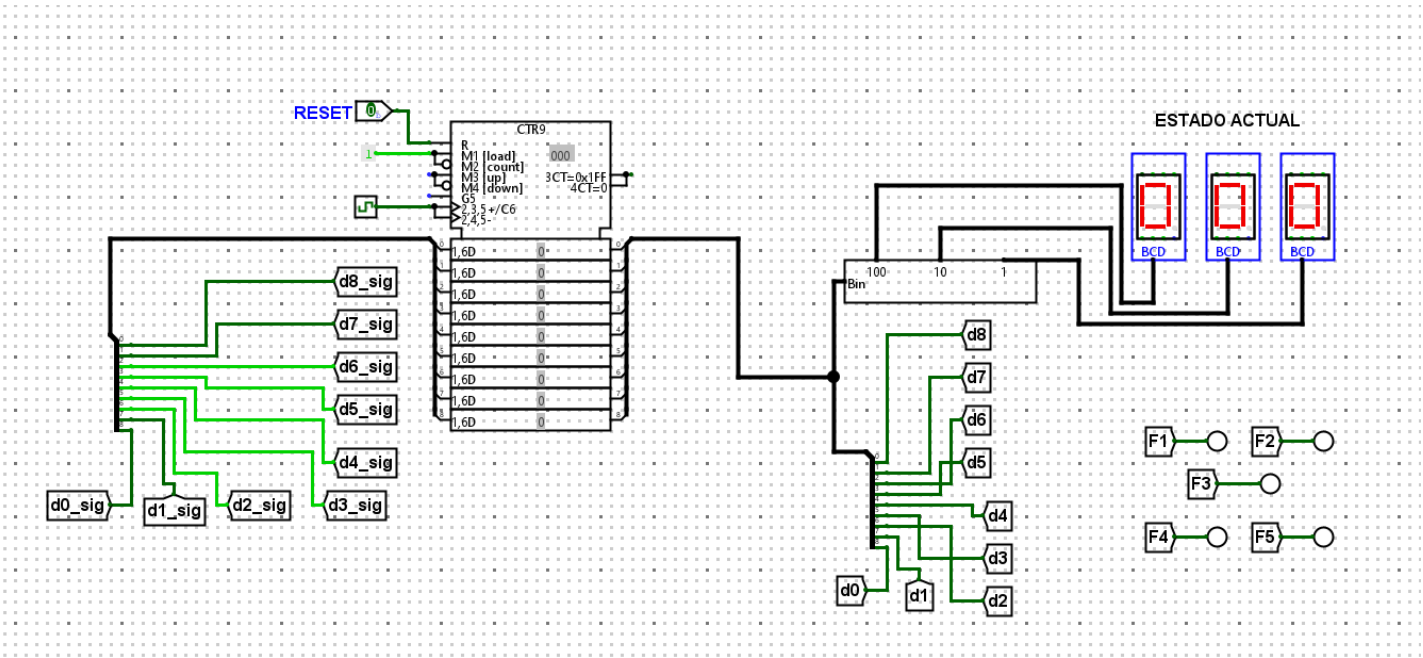
Estados

Los cambios de estados del circuito quedan definidos mediante la siguiente tabla realizada a partir del diagrama de estados proporcionado, que contiene el estado actual, los valores de las diferentes funciones y el estado siguiente. El contador trabajará con números de 9 bits.

Estado actual d0d1d2d3d4d5d6d7d8	**F1**	**F2**	**F3**	**F4**	**F5**	**Estado siguiente (d0d1dd2d3d4d5d6d7d8)+**
000000000 (0)	*	*	*	*	*	001111100 (124)
001111100 (124)	0	*	*	*	*	001011111 (95)
001111100 (124)	1	*	*	*	*	101110100 (372)
001011111 (95)	*	0	*	*	*	000111110 (62)
001011111 (95)	*	1	*	*	*	111011011 (475)
000111110 (62)	*	*	0	*	*	000000101 (5)
000111110 (62)	*	*	1	*	*	101110100 (372)
000000101 (5)	*	*	*	0	*	101110100 (372)
000000101 (5)	*	*	*	1	*	000000101 (5)
101110100 (372)	*	*	*	*	*	111011011 (475)
111011011 (475)	*	*	*	0	*	000011111 (31)
111011011 (475)	*	*	*	1	*	111011011 (475)
000011111 (31)	*	*	*	*	0	000000000 (0)
000011111 (31)	*	*	*	*	1	001111100 (124)

El contador recibe como entradas los 9 bits del número que representa el estado, el reloj, y siempre está en modo load para poder realizar los saltos necesarios. Las salidas son los 9 bits del estado actual, que se muestran en el display de 7 segmentos primero habiendo convertido el número de binario a decimal utilizando el conversor que provee Logisim.

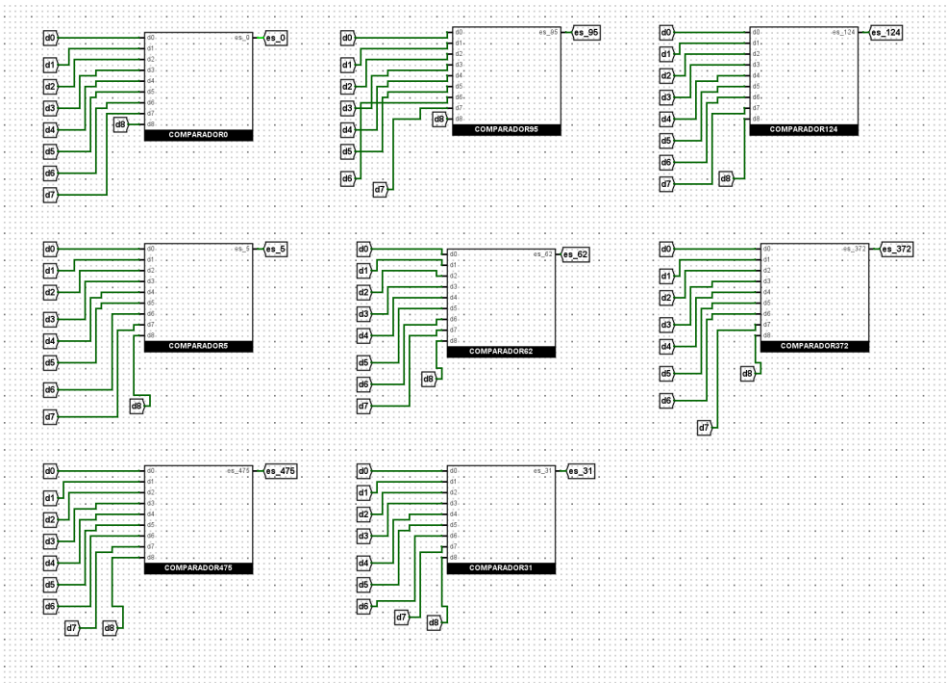
En todos los circuitos, para mantener la claridad y evitar errores, utilizamos etiquetas y túneles en todas las entradas y salidas.



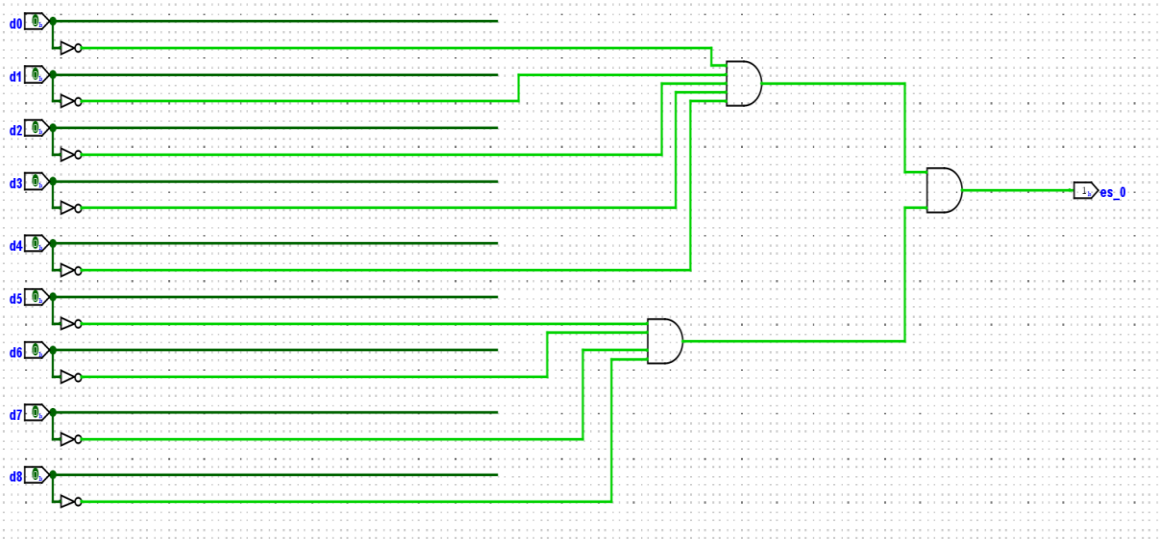
Comparadores

Para saber cuál es el estado actual y a qué estado debemos saltar, implementamos 8 comparadores que representan los diferentes 8 estados del circuito. Cada uno retorna 0 o 1 dependiendo de la salida del contador.

Por ejemplo, si el estado actual es el 0 el circuito COMPARADOR0 devuelve 1 y todos los demás devuelven 0.

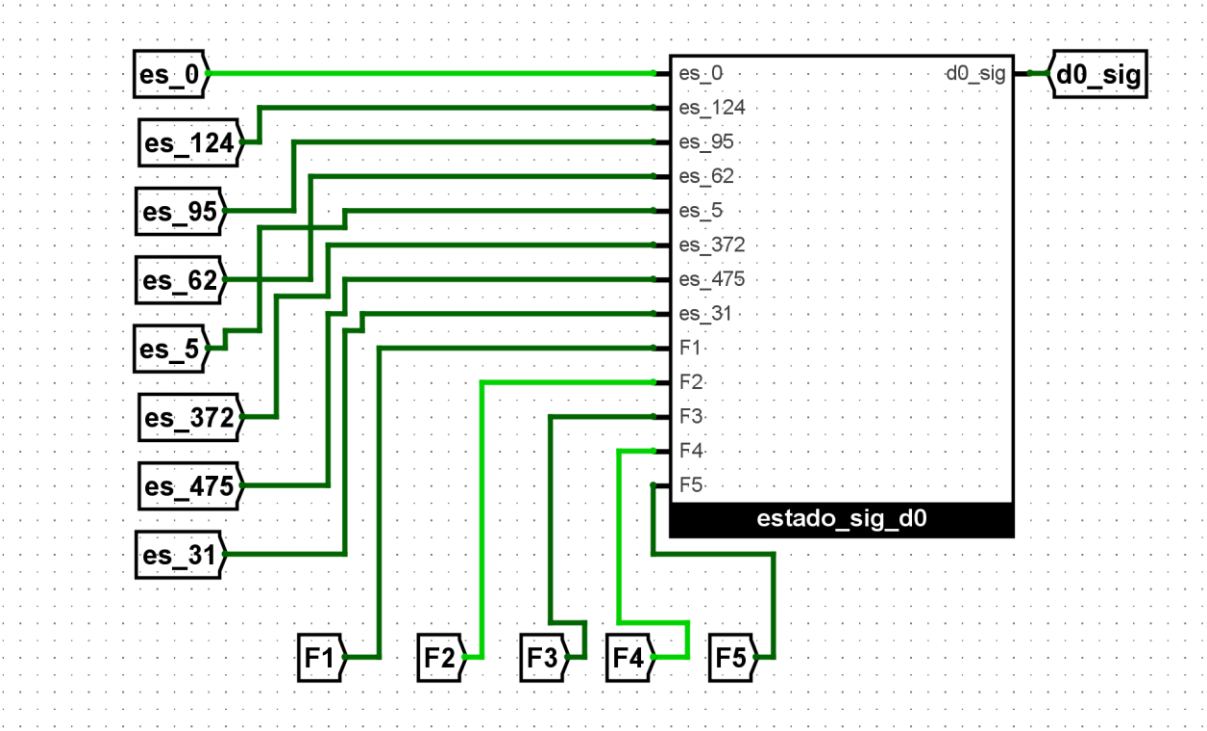


Dentro de cada uno, analizamos bit por bit si es el número que queremos y realizamos el cómputo mediante compuertas.



Cálculo del estado siguiente

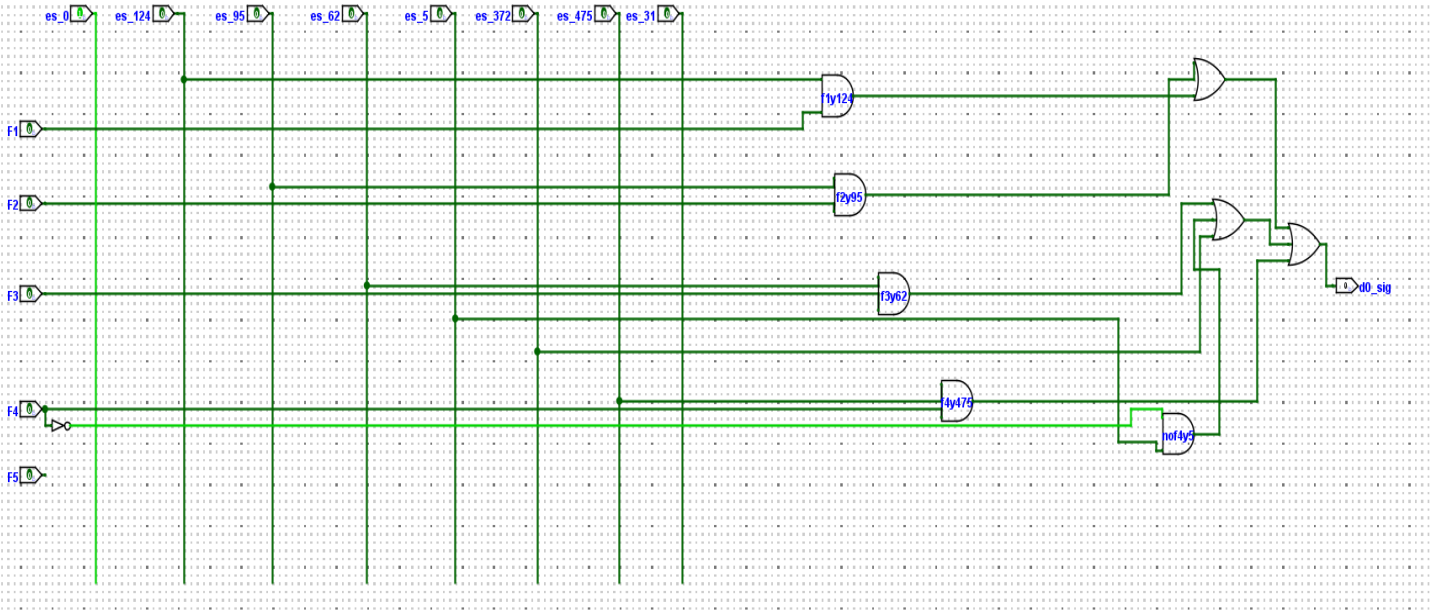
Una vez calculado el estado actual y para computar el estado siguiente, implementamos 9 circuitos donde cada uno calcula 1 bit para conformar el número al que se debe saltar.



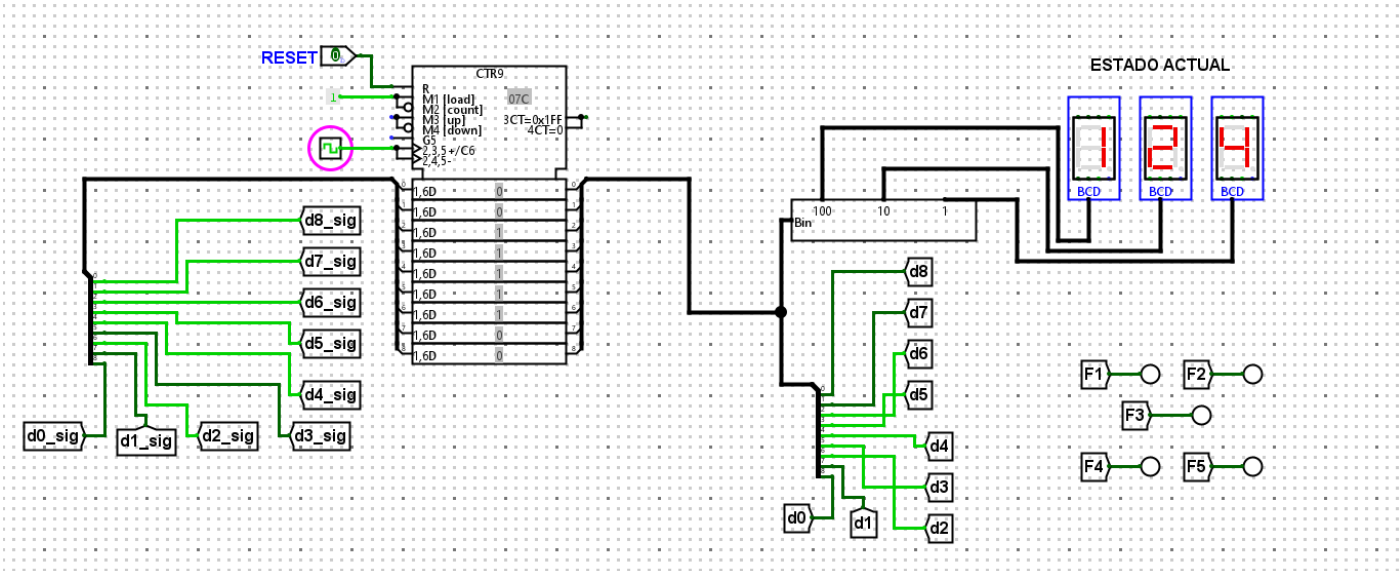
Cada uno recibe como entrada las salidas de los 8 comparadores, los estados de las 5 funciones y devuelve 1 bit. Para realizarlo, utilizamos la tabla anteriormente mencionada, y para cada bit creamos una tabla reducida que define en qué momentos ese bit es 1. Por ejemplo, para el bit d0:

Estado actual d0d1d2d3d4d5d6d7d8	F1	F2	F3	F4	F5	(do)+
001111100 (124)	1	*	*	*	*	1
001011111 (95)	*	1	*	*	*	1
000111110 (62)	*	*	1	*	*	1
00000101 (5)	*	*	*	0	*	1
101110100 (372)	*	*	*	*	*	1
111011011 (475)	*	*	*	1	*	1

Luego, calculamos el resultado mediante compuertas.



Cada una de estas salidas son ingresadas al contador para que cuando ocurra el siguiente pulso, pueda saltar al estado esperado.

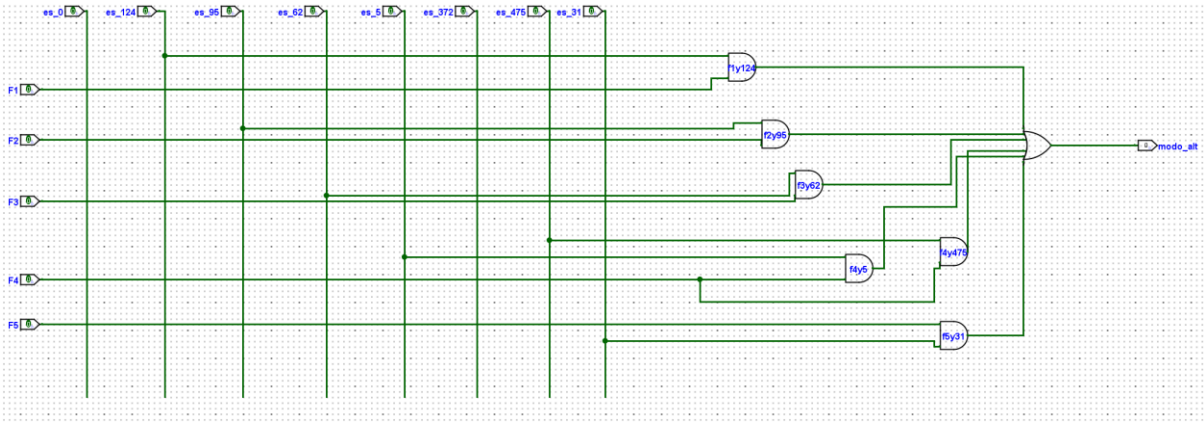


Etapa 2

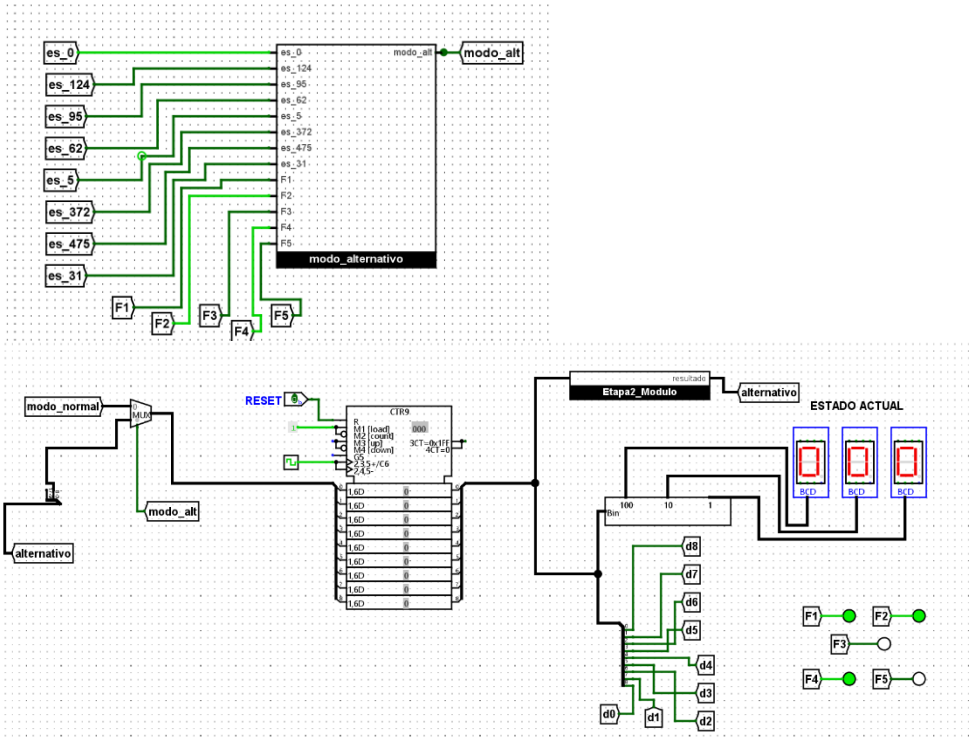
Descripción del circuito

Para esta etapa, dividimos el modo normal del modo alternativo. Manteniendo el cálculo del modo normal como en la etapa 1 (bit por bit según en qué estado está el contador y a cuál queremos ir), y eliminando los casos del modo alternativo para poder calcularlo mediante el multiplicador.

Creamos un circuito que devuelve verdadero cuando el modo alternativo está activo, teniendo en cuenta el grafo de estados.



La salida de este circuito entra como línea de selección a un MUX que elige entre el modo alternativo y el normal según cuál esté activo.

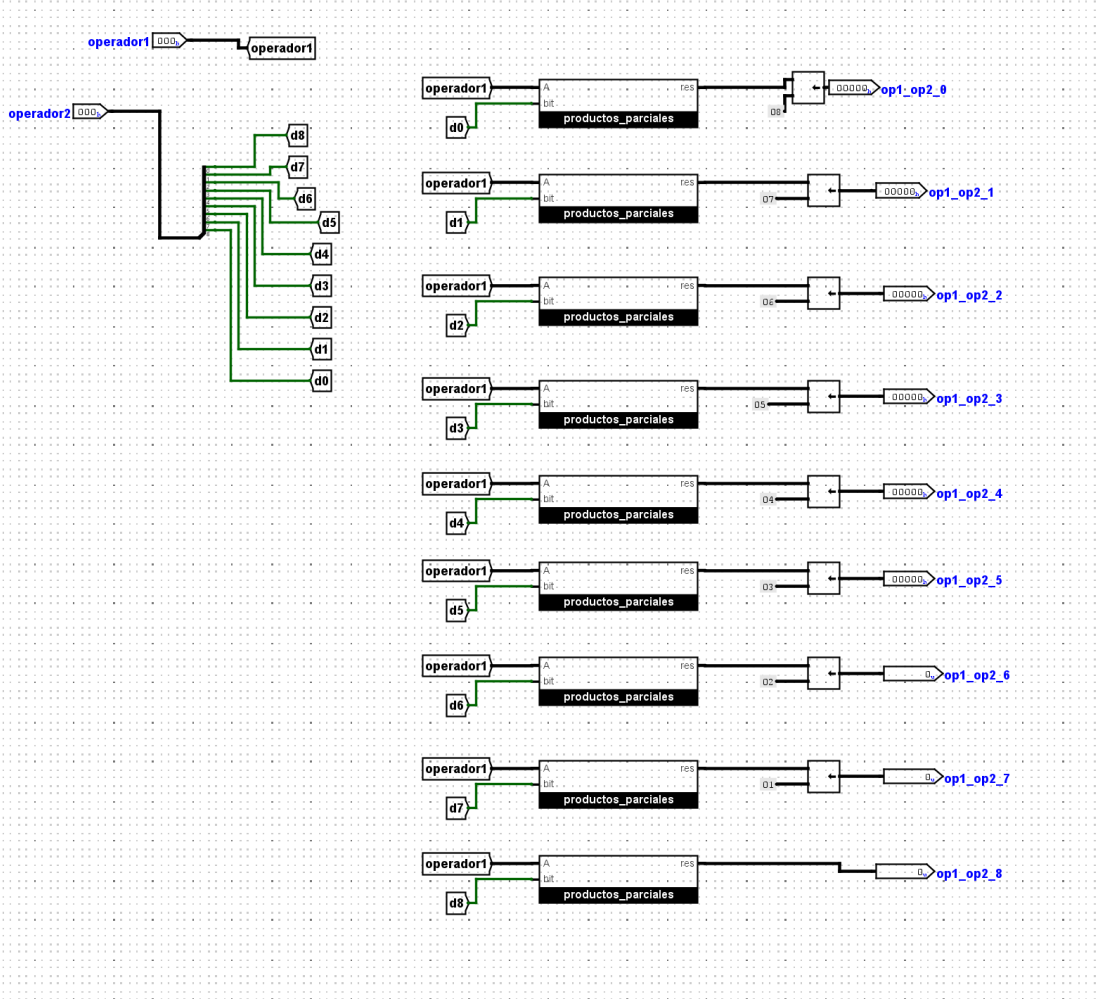


Multiplicador

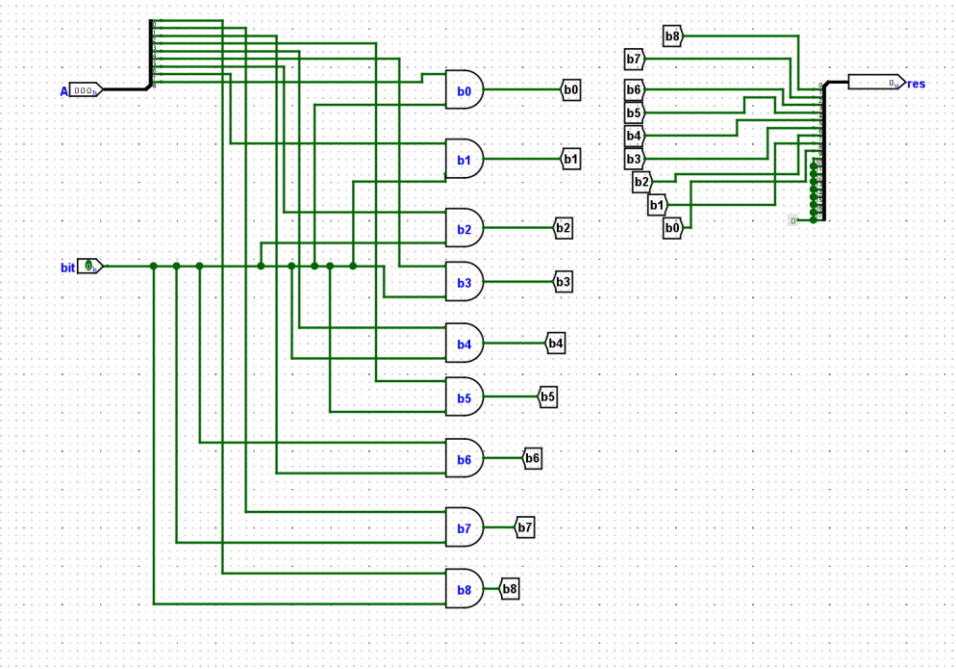
Para implementar el multiplicador, utilizamos los módulos brindados por la cátedra, donde según cuál es el estado actual, se calcula cuál es el factor por el que se debe multiplicar al estado para obtener el siguiente.

Dentro del multiplicador, implementamos un Árbol de CSA en cascada, primero obteniendo los productos parciales de la multiplicación que queremos calcular.

Teniendo los dos operandos, dividimos los bits del segundo y multiplicamos el primer operando por ese bit, aplicando los desplazamientos correspondientes al resultado para así obtener productos parciales de 18 bits.



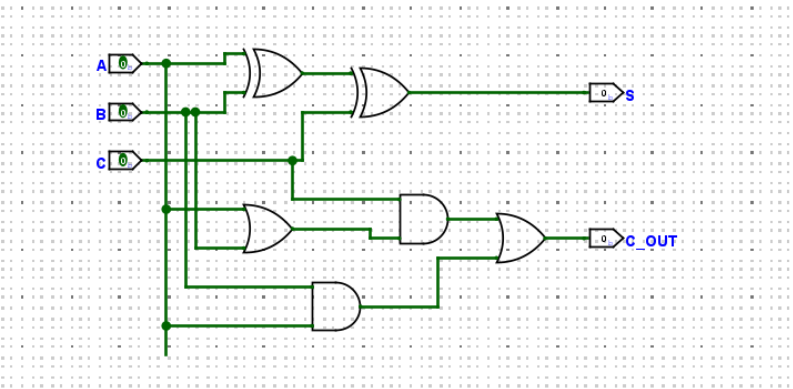
Dentro de los productos parciales, para calcular cada dígito del resultado hacemos un AND con el bit por el que estamos multiplicando el número. Los 9 bits del resultado son los 9 bits menos significativos de los 18 que necesitamos devolver, entonces completamos con ceros.



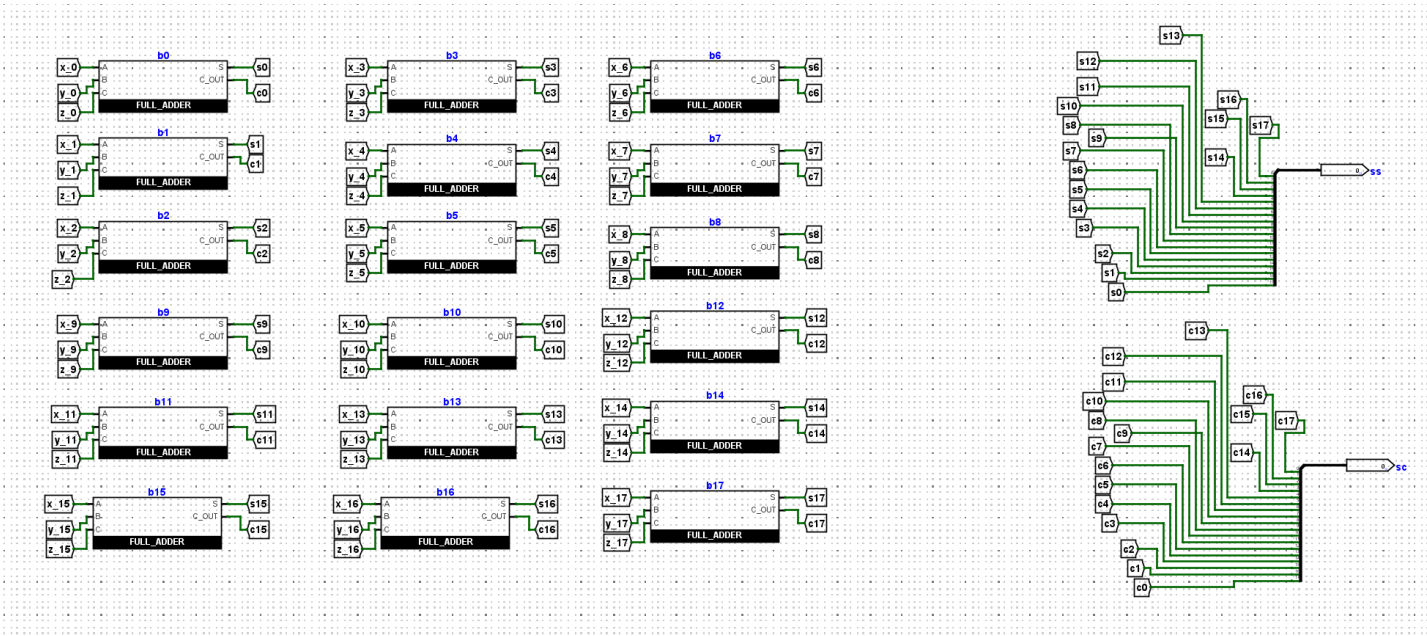
CSA

Una vez obtenidos todos los productos parciales, los vamos ingresando en el Árbol de CSA en casada, donde cada CSA recibe 3 números de 18 bits cada uno y devuelve una semisuma y un saemicarry ambos de 18 bits.

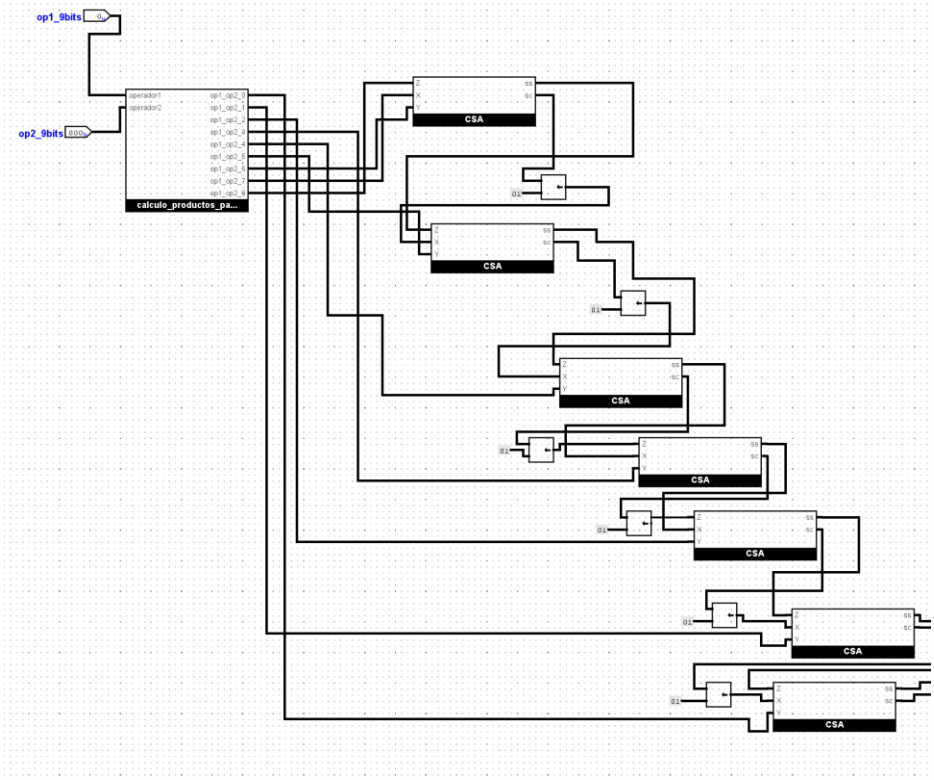
Cada CSA contiene 18 full adders que reciben 3 bits y devuelven suma y el carry de cada grupo.



Con el resultado de cada uno de los full adders, armamos la semisuma y el semicarry total de los 3 números de 18 bits que queríamos sumar.



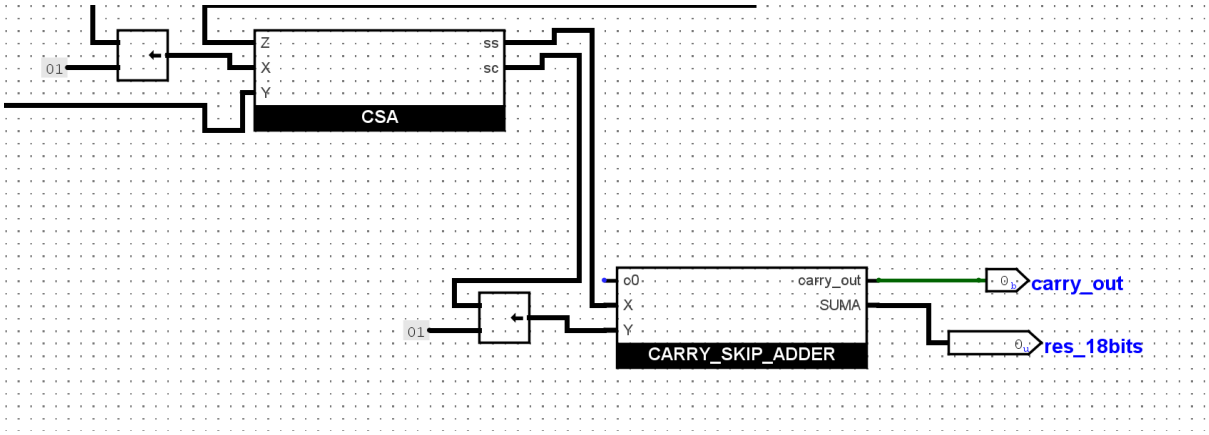
Luego, en cada CSA vamos ingresando también los resultados del CSA anterior, desplazando el semicarry una posición.



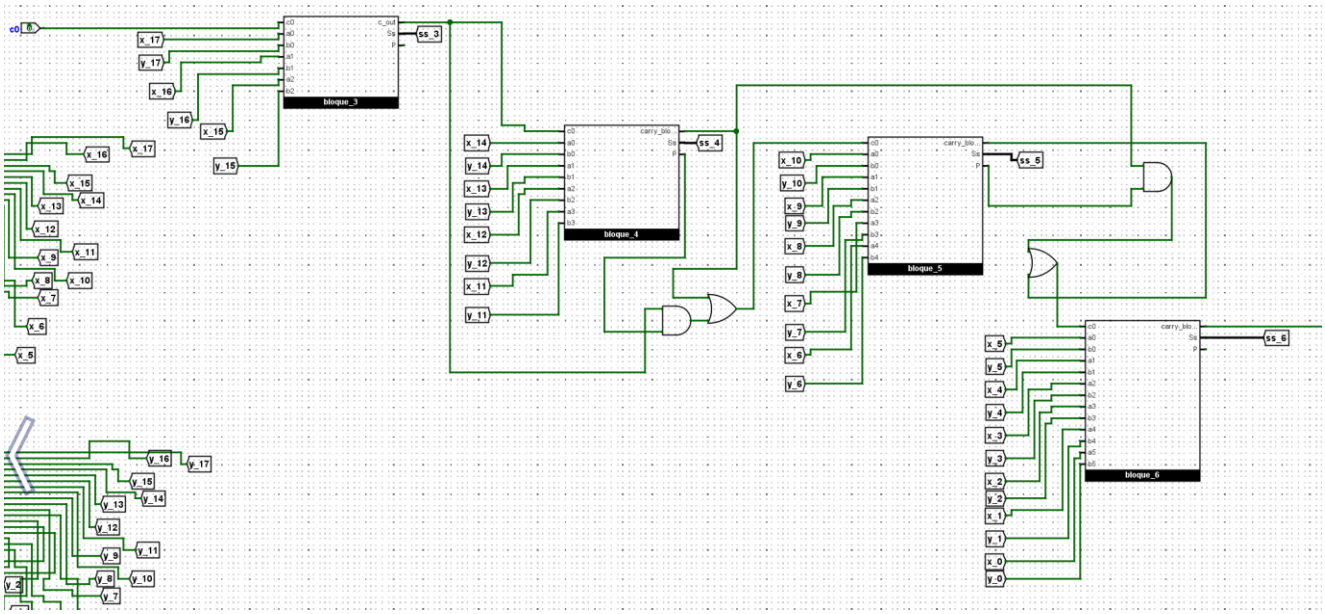
Carry Skip Adder

Finalmente, para obtener el resultado de la multiplicación debemos sumar los últimos semicarry y semisuma, para esto usamos el sumador paralelo Carry Skip Adder.

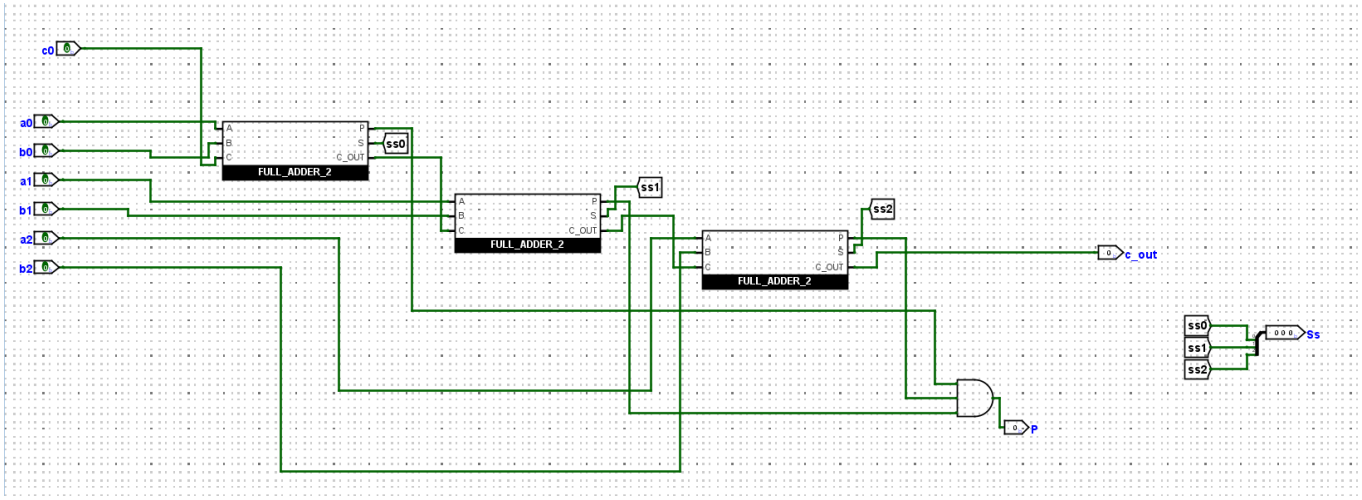
El Carry Skip Adder tiene como entradas dos números de 18 bits y devuelve su suma en 18 bits.



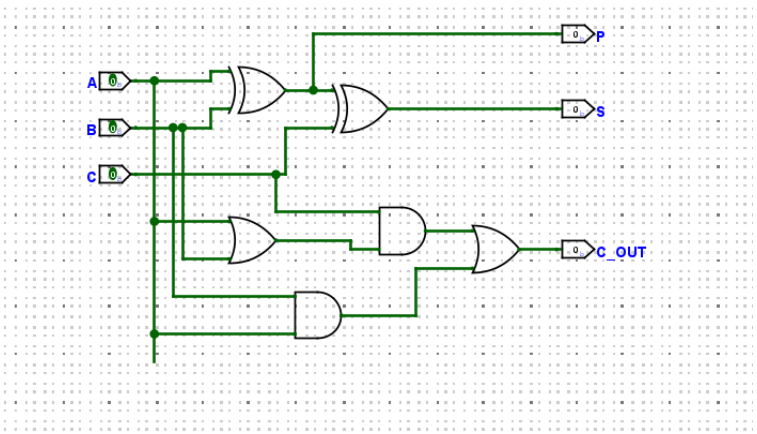
Para implementarlo, dividimos los full adders en bloques de 3-4-5-6.



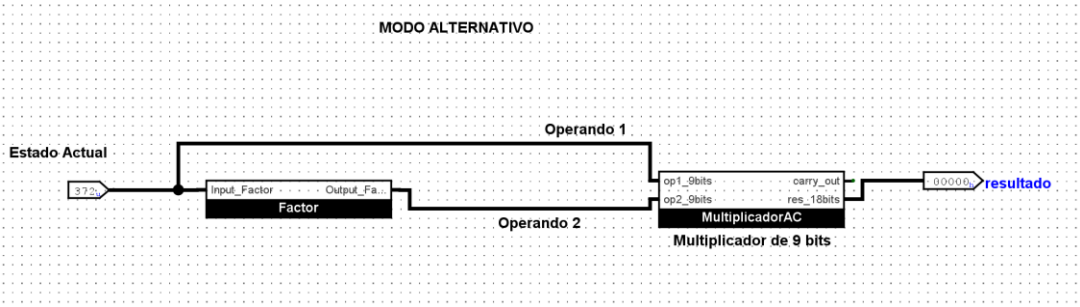
Dentro de cada bloque los full adders van conectados en ripple y devuelven la suma, el P grupal y el carry-out.



Para esto creamos un full adder adicional que devuelve el P del grupo.



Con este cálculo ya podemos devolver el resultado final de la multiplicación y obtenemos el próximo estado al que debemos saltar si el circuito está en modo alternativo.



El resultado del multiplicador son 18 bits, pero en este caso sólo utilizamos los 9 menos significativos para ingresarlos al contador.

Conclusiones

Este proyecto de Arquitectura de Computadoras nos permitió aplicar los conceptos en el diseño de circuitos. A través de la utilización de la herramienta Logisim-Evolution, nos enfocamos en mantener la claridad y la eficiencia en nuestro trabajo.

La incorporación del multiplicador en el modo alternativo en la segunda etapa del proyecto nos desafió a adaptar nuestro diseño y encontrar otra solución. Utilizamos un enfoque modular y aprovechamos lo anteriormente resuelto para poder obtener los resultados.