CAMPUS QUERÉTARO

TE2002B.1 DISEÑO CON LÓGICA PROGRAMABLE

(GRUPO 1)

"Evidencia 1 Reto. Diseño e implementación de una ALU de 4 bits en Intel Quartus Prime"

Evidencia presentada por la estudiante:

A01706095 - Naomi Estefanía Nieto Vega

Profesores:

Jesús Esteban Cienfuegos Zurita

Rick Leigh Swenson Durie

Fecha de entrega:

Sábado 20 de febrero de 2021

Evidencia 1. Diseño de una Unidad Aritmética Lógica (ALU) de 4 bits

I. Introducción

En esta evidencia se desarrollará una Unidad Aritmética Lógica (ALU) de 4-bits considerando números con signo, en ella se presentarán las siguientes operaciones que se muestran en la Tabla 1.

Tabla 1. Operaciones aritméticas y lógicas de la ALU desarrollada.

Instrucción	Código	Descripción
Add rd, rs, op2	“000”	La suma de rs más op2 almacenada en rd.
Addc rd, rs, op2, cin	“001”	La suma de rs más op2 más cin, almacenada en rd.
Sub rd, rs, op2	“010”	La resta de rs menos op2 y almacenada en rd.
Subc rd, rs, op2, cin	“011”	La resta de rs menos op2 menos cin, guardada en rd.
AND rd, rs, op2	“100”	El AND de rs y op2 almacenado en rd.
OR rd, rs, op2	“101”	El OR de rs y op2 almacenado en rd.
XOR rd, rs, op2	“110”	El XOR de rs y op2 guardado en rd.
MASK rd, rs, op2	“111”	El AND de rs con el NOT de op2 guardado en rd.

II. Desarrollo

Para el desarrollo de esta evidencia se planteó primeramente la lógica a utilizar para poder realizar una ALU que permitiera las operaciones antes mencionadas incluyendo los números con signo.

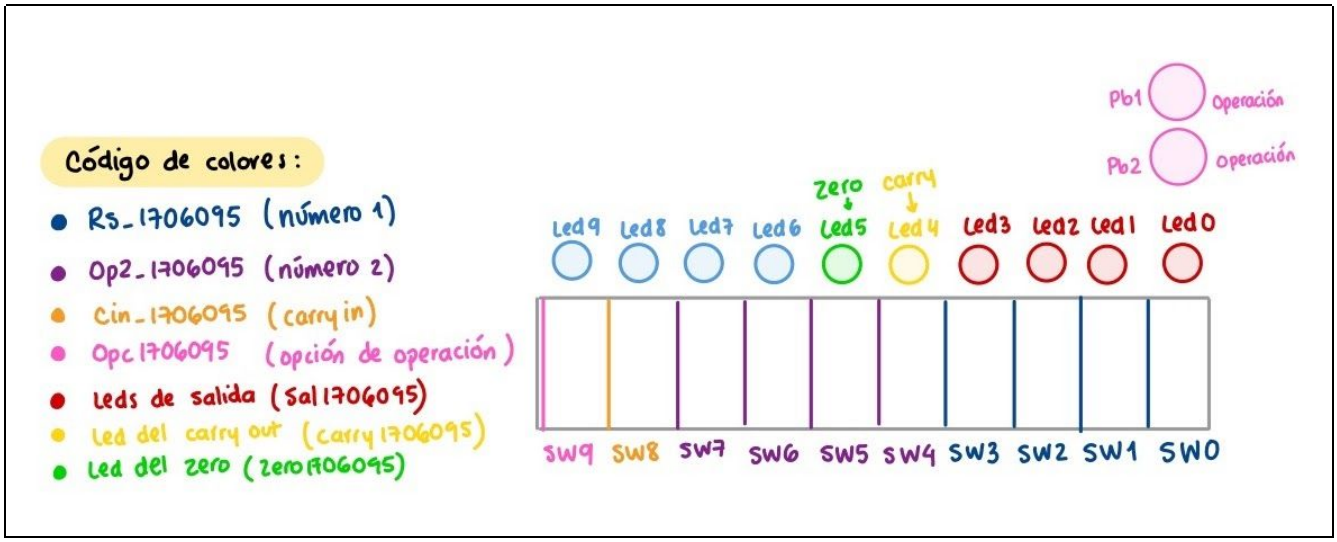
A. Diseño y modelado

En la figura 1 podemos observar un esquema de cómo fueron utilizados cada uno de los switches y push buttons disponibles en el FPGA y los LEDs que son utilizados para mostrar la salida, en este caso los LEDs del 6 al 9 no fueron utilizados por lo que siempre permanecen encendidos en un tono un poco menos brillante. Por otra parte, para introducir el opcode o el código correspondiente a la operación aritmética o lógica que va a realizar la ALU utilicé el push button 0 y 1, y el switch 9. Para introducir el valor del source register o “rs_1706095” que equivale al número 1 a insertar los switches que se modifican son los del 0 al 3; para el número 2 o “op2_1706095” utilicé los switches del 4 al 7. En ellos es posible introducir un número binario de 4 bits que puede ser del -7 al 8.

Para las operaciones que utilizan “carry in” o “Cin_1706095” es necesario levantar el switch 8, de esta forma se le indica que será utilizado.

Por último para mostrar el número de salida se utilizan los LEDs del 0 al 3, y el LED4 y 5 se activan para mostrar el carry y el zero respectivamente. (Ver figura 1).

Figura 1. Ilustración esquemática del uso del FPGA DE10-Lite de Intel®



B. Código fuente en VHDL

A continuación se muestra el código fuente utilizado para el desarrollo de esta ALU, este código lo que realiza brevemente es: primero se importan las librerías necesarias para los cálculos, una de ellas muy importante es la librería “IEEE.std_logic_signed.all” ya que con esta es posible utilizar los números con signo. Posteriormente se crea la entidad con las variables que se van a utilizar. En la arquitectura se crean tres señales ya que con ellas se va a trabajar para realizar la suma con signo. La primera señal es la que nos servirá para almacenar el resultado, nos manda la salida para guardarlo y manipularlo, después la segunda señal sirve para guardar el resultado cuando se trata de una suma de números positivos y la tercera señal para cuando es una suma de números negativos.

Posteriormente en el caso 1, se le asigna a la señal 1 la suma positiva, y para esta únicamente seleccionamos los primeros 3 bits, por eso se le agrega un ‘0’.
Después dentro del ciclo if lo que se hace es evaluar si será necesario utilizar un bit extra para cuando la suma se trata de dos números positivos, por lo que sí si lo es se prende el carry out asignando los 3 primeros bits de la señal 1.

Después en caso de que no haya caído en el primer if, evalúa el segundo que es para cuando tenemos número negativo y negativo y ahí evalúa nuevamente otro if, en el cual se evalúa lo mismo que el primer if pero ahora considerando los negativos, y cómo son negativos en este sí es importante considerar los 4 bits, entonces se evalúan los casos posibles en los que la suma se trate de ambos números positivos, o ambos negativos o mezclados que serían positivo y negativo.

Entonces lo que se hace con ese if indexado es evaluar con las señales y dependiendo de lo que regrese la señal decidir si se prenderá el carry out o no.

Esta misma secuencia aplica para la suma con carry, la única diferencia es que ahora se le suma el "carry in" al final. Y se replica para la resta únicamente cambiando los signos y agregando un +1 ya que para este caso a nuestro segundo número le afecta un signo "negativo" y por ello es que se convierte a complementos a dos.

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- TE2002B. Diseño con lógica programable
-- Autor(a): A01706095 Naomi Estefanía Nieto Vega
-- Fecha: 17 de febrero 2021
-- Evidencia 1. Implementación y diseño de una ALU de 4-bits
-- Instrucciones: Add, Addc, Sub, Subc, And, Or, Xor, Mask
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_arith.all;
Use IEEE.numeric_std.all;
Use IEEE.std_logic_signed.all;

Entity Evidencia1_1706095 is
Port (
              Rs_1706095, Op2_1706095            : in std_logic_vector(3 downto 0);
              Opc1706095                         : in std_logic_vector(2 downto 0);
              Cin_1706095                        : in std_logic;
              Sal1706095                         : out std_logic_vector(3 downto 0);
              Carry1706095                       : out std_logic;
              Zero1706095                        : out std_logic);
End Evidencia1_1706095;

Architecture a of Evidencia1_1706095 is
       Signal Rd_1706095                         :      std_logic_vector(3 downto 0);
       Signal senal1_1706095                     :      std_logic_vector(3 downto 0);
       Signal senal2_1706095                     :      std_logic_vector(4 downto 0);

Begin
       ALU_1706095 : Process(Rs_1706095, Op2_1706095, Opc1706095)
       Begin
```

```
            Case (Opc1706095) is

      When "000" => --Add source register and op2, result in destination register

            Rd_1706095 <= Rs_1706095 + Op2_1706095;

      senal1_1706095 <= ('0' & Rs_1706095 (2 downto 0)) + ('0' & Op2_1706095(2 downto 0));
      senal2_1706095 <= ('0' & Rs_1706095(3 downto 0)) + ('0' & Op2_1706095(3 downto 0));

                  If (Rs_1706095(3) = '0' and Op2_1706095(3) = '0') then
                        carry1706095 <= senal1_1706095(3);

                  Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '1' ) then
                        If (senal2_1706095(4) = '1' and senal2_1706095(3) = '1') then
                              Carry1706095 <= '0';
                        Else Carry1706095 <= '1';

                        End If;
                  Else Carry1706095 <= '0'; --numeros 7 a -8
                  End If;

      When "001" => -- Addc rd, rs, op2   // Add rs and op2 with carry, result in rd

            Rd_1706095 <= Rs_1706095 + Op2_1706095 + Cin_1706095;

      senal1_1706095 <= ('0' & Rs_1706095 (2 downto 0)) + ('0' & Op2_1706095(2 downto 0)) +
Cin_1706095;
      senal2_1706095 <= ('0' & Rs_1706095(3 downto 0)) + ('0' & Op2_1706095(3 downto 0)) +
Cin_1706095;

                  If (Rs_1706095(3) = '0' and Op2_1706095(3) = '0') then
                        Carry1706095 <= senal1_1706095(3);

                  Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '1' ) then --neg y neg
                        If (senal2_1706095(4) = '1' and senal2_1706095(3) = '1') then
                              Carry1706095 <= '0';
                        Else Carry1706095 <= '1';

                        End if;
                  Else Carry1706095 <= '0'; --numeros 7 a -8
                  End if;

      When "010" => -- Sub rd, rs, op2    // Subtract op2 from rs, result in rd

            Rd_1706095 <= Rs_1706095 - Op2_1706095;

    senal1_1706095 <= ('0' & Rs_1706095 (2 downto 0)) + ('0' & NOT(Op2_1706095(2 downto 0))+1);
    senal2_1706095 <= ('0' & Rs_1706095(3 downto 0)) + ('0' & NOT(Op2_1706095(3 downto 0))+1);

                  If (Rs_1706095(3) = '0' and Op2_1706095(3) = '0') then
                        carry1706095 <= '0';

                  Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '1' ) then
                        carry1706095 <= '0';
```

```
                    Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '0' ) then
                           If (senal2_1706095(4) = '1' and senal2_1706095(3) = '1') then
                                  Carry1706095 <= '0';
                           Else Carry1706095 <= '1';

                           End if;
                    Else Carry1706095 <= senal1_1706095(3); --numeros 7 a -8
                    End if;

      When "011" => -- Substract op2 from rs with carryin(Cin), result in rd

             Rd_1706095 <= Rs_1706095 - Op2_1706095 - Cin_1706095;

      senal1_1706095 <= ('0' & Rs_1706095 (2 downto 0)) + ('0' & NOT(Op2_1706095(2 downto 0))+1)
- Cin_1706095;
      senal2_1706095 <= ('0' & Rs_1706095(3 downto 0)) + ('0' & NOT(Op2_1706095(3 downto 0))+1)
- Cin_1706095;

                    If (Rs_1706095(3) = '0' and Op2_1706095(3) = '0') then --positivo y neg
                           carry1706095 <= '0';

                    Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '1' ) then --neg y positivo
                           carry1706095 <= '0';

                    Elsif (Rs_1706095(3) = '1' and Op2_1706095(3) = '0' ) then
                           If (senal2_1706095(4) = '1' and senal2_1706095(3) = '1') then
                                  Carry1706095 <= '0';
                           Else Carry1706095 <= '1';

                           End if;

                    Else Carry1706095 <= senal1_1706095(3); --numeros 7 a -8
                    End if;

      When "100" => -- Logical AND of rs and op2, result in rd

             Rd_1706095 <= Rs_1706095 AND Op2_1706095;
      When "101" => -- OR rd, rs, op2     // Logical OR of rs and op2, result in rd

             Rd_1706095 <= Rs_1706095 OR Op2_1706095;

      When "110" => -- XOR rd, rs, op2    // Logical XOR of rs and op2, result in rd

             Rd_1706095 <= Rs_1706095 XOR Op2_1706095;

      When "111" => -- MASK rd, rs, op2   // Logical AND of rs and NOT op2, result in rd

             Rd_1706095 <= Rs_1706095 AND (NOT Op2_1706095);

      When others => Rd_1706095 <= "0000";
                    Zero1706095 <= '0';

      End Case;
```

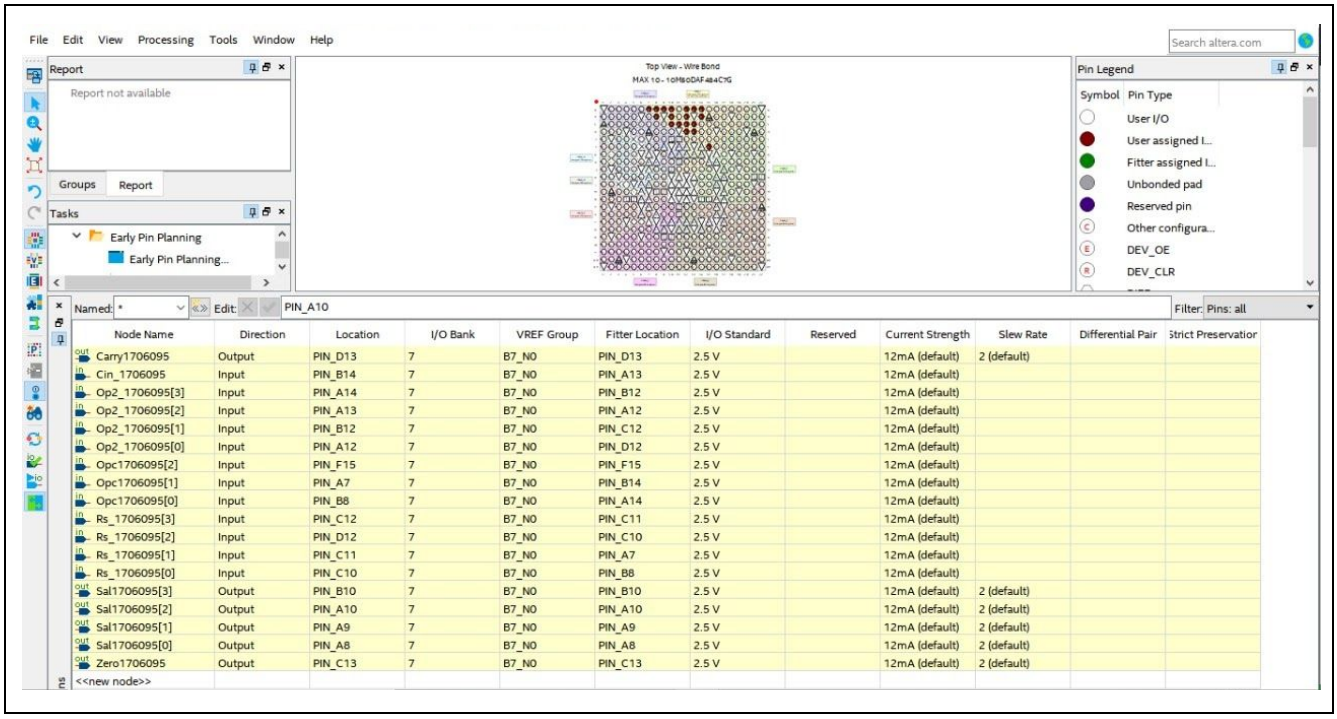
```
        If Rd_1706095 = "0000" then
                Zero1706095 <= '1';
        Else
                Zero1706095 <= '0';
        End If;
        Sal1706095 <= Rd_1706095;
        End Process ALU_1706095;
End a;
```

A continuación se adjunta un link con dirección hacia la carpeta en drive en la que están todos los archivos del código fuente: [Evidencia1_TE2002B_A01706095]

C. Simulación en Quartus Prime

Para la simulación en Quartus lo que hice primero fue asignar las entradas y salidas en el Pin Planner (ver figura 2) y después compilar para verificar que no existieran errores, posteriormente abrí la pestaña de "Programmer" para enviar el código a la tarjeta.

Figura 2. Asignación de pines en el Pin Planner de Intel® Quartus Prime



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
Carry1706095	Output	PIN_D13	7	B7_N0	PIN_D13	2.5 V		12mA (default)	2 (default)		
Cin_1706095	Input	PIN_B14	7	B7_N0	PIN_A13	2.5 V		12mA (default)			
Op2_1706095[3]	Input	PIN_A14	7	B7_N0	PIN_B12	2.5 V		12mA (default)			
Op2_1706095[2]	Input	PIN_A13	7	B7_N0	PIN_A12	2.5 V		12mA (default)			
Op2_1706095[1]	Input	PIN_B12	7	B7_N0	PIN_C12	2.5 V		12mA (default)			
Op2_1706095[0]	Input	PIN_A12	7	B7_N0	PIN_D12	2.5 V		12mA (default)			
Opc1706095[2]	Input	PIN_F15	7	B7_N0	PIN_F15	2.5 V		12mA (default)			
Opc1706095[1]	Input	PIN_A7	7	B7_N0	PIN_B14	2.5 V		12mA (default)			
Opc1706095[0]	Input	PIN_B8	7	B7_N0	PIN_A14	2.5 V		12mA (default)			
Rs_1706095[3]	Input	PIN_C12	7	B7_N0	PIN_C11	2.5 V		12mA (default)			
Rs_1706095[2]	Input	PIN_D12	7	B7_N0	PIN_C10	2.5 V		12mA (default)			
Rs_1706095[1]	Input	PIN_C11	7	B7_N0	PIN_A7	2.5 V		12mA (default)			
Rs_1706095[0]	Input	PIN_C10	7	B7_N0	PIN_B8	2.5 V		12mA (default)			
Sal1706095[3]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA (default)	2 (default)		
Sal1706095[2]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA (default)	2 (default)		
Sal1706095[1]	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA (default)	2 (default)		
Sal1706095[0]	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA (default)	2 (default)		
Zero1706095	Output	PIN_C13	7	B7_N0	PIN_C13	2.5 V		12mA (default)	2 (default)		
<<new node>>											

En la tabla 2 se muestra la información más completa de la variable, su dirección (input o output) y en donde se ubica dentro del pin planner y físicamente en el FPGA.

Tabla 2. Ubicación de los nodos de entrada y salida en la tarjeta. [1]

Nodo	Dirección	Ubicación	Ubicación física

Rs[0]	INPUT	PIN_C10	Switch 0
Rs[1]	INPUT	PIN_C11	Switch 1
Rs[2]	INPUT	PIN_D12	Switch 2
Rs[3]	INPUT	PIN_C12	Switch 3
Op2[0]	INPUT	PIN_A12	Switch 4
Op2[1]	INPUT	PIN_B12	Switch 5
Op2[2]	INPUT	PIN_A13	Switch 6
Op2[3]	INPUT	PIN_A14	Switch 7
Opc[0]	INPUT	PIN_B8	Push_button[0]
Opc[1]	INPUT	PIN_A7	Push_button[1]
Opc[2]	INPUT	PIN_F15	Switch 9
Cin	INPUT	PIN_B14	Switch 6
Carry	OUTPUT	PIN_D13	LED4
Zero	OUTPUT	PIN_C13	LED5
Sal[0]	OUTPUT	PIN_A8	LED0
Sal[1]	OUTPUT	PIN_A9	LED1
Sal[2]	OUTPUT	PIN_A10	LED2
Sal[3]	OUTPUT	PIN_B10	LED3

D. Simulación en el FPGA DE-10 Lite

a. Casos de prueba

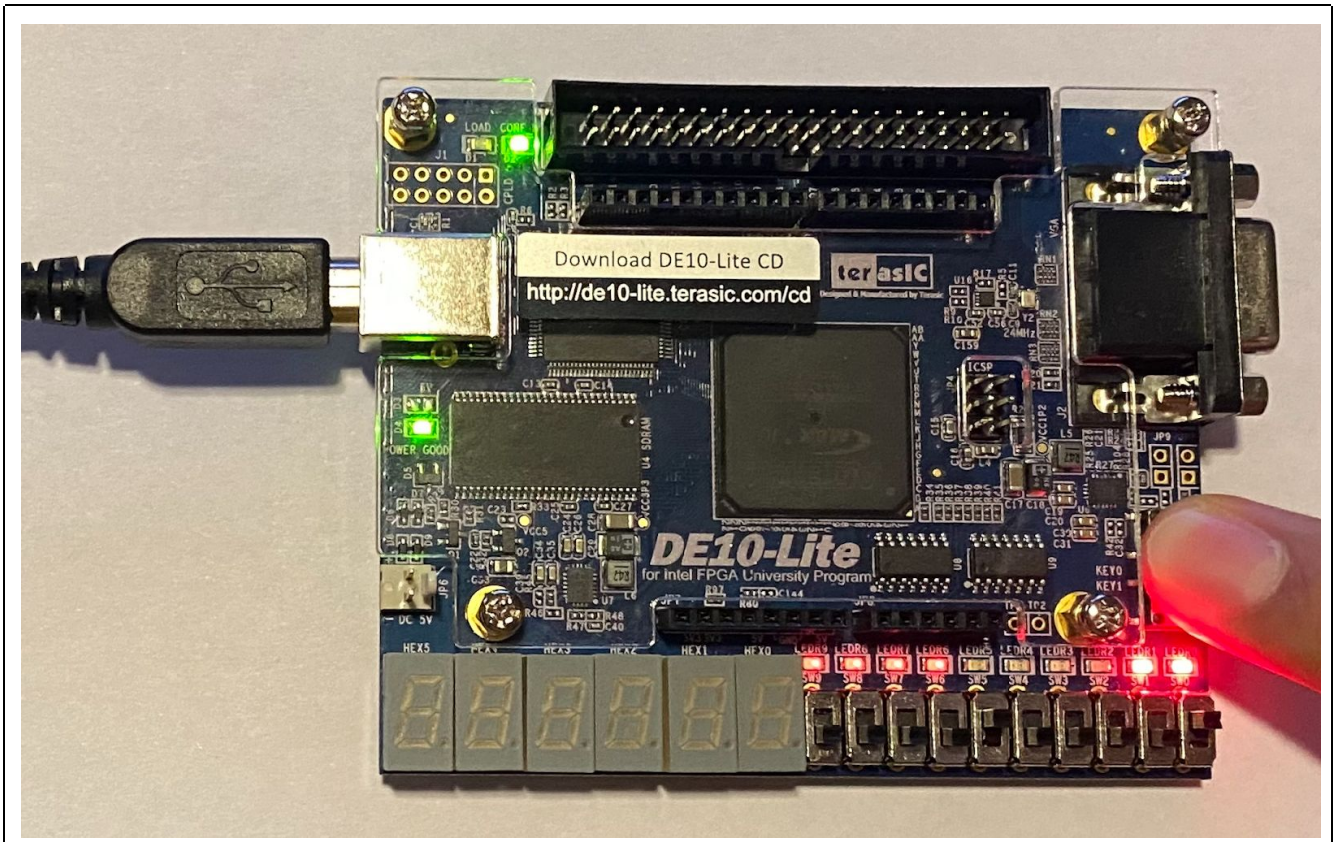
Para comprobar el correcto funcionamiento del programa una vez que compiló correctamente en la computadora y asigne los pines como se mostró anteriormente (ver Tabla 2) comencé a ingresar valores directamente en la placa, los cuales se muestran a continuación.

Para primer caso de prueba lo que hice fue primero introducir el código de la suma normal que en este caso es “000” por lo que se deben presionar los dos push buttons y dejar en 0 el switch 9, de esta forma le decimos a nuestra tarjeta que la operación que vamos a realizar será una suma.

Posteriormente lo que hice fue sumar dos números (seleccionados por mi aleatoriamente), y así probar que funciona correctamente la suma normal. Los números

que seleccioné fueron el 1 (0001) y el 2 (0010) para sumarse, esto debería dar un 3 (0011) como resultado y como se puede observar en la figura 3, se encienden los LEDs 0 y 1 dando como resultado un “0011” qué es el resultado esperado.

Figura 3. Resultado del primer caso de prueba (suma normal).



Para el segundo caso fue una suma con carry in (Cin), para este caso es importante notar que estamos seleccionando la opción “001” presionando ambos push buttons y levantando el switch 9. Asimismo debe estar levantado el switch 8 que pertenece al carry. Posteriormente seleccioné otros dos números aleatoriamente para proceder a hacer la suma, para esto el primer número fue el 0001 más el 0010 y dió 0011. (como se puede observar en la figura 4).

Repetí el procedimiento anterior para el caso de la resta normal fue el 0101 menos el 0100 que debería dar 0001. (ver figura 5). Para el siguiente caso los números usados fueron el 0011 menos el 0001 y debería dar 0010 (ver figura 6). En el siguiente caso los números fueron 0000 AND 0000 lo que debería dar como resultado 0000 (ver figura 7). Para el OR fueron 0001 y 0001 lo que debería dar 0001 (ver figura 8). En el caso del XOR los números fueron 0001 XOR 0000 = 0001 (ver figura 9). Y por último el MASK que es A

and (not B) usé los números 0001 y 0000 lo que dió como resultado el 0001, se puede comprobar que son correctos (ver figura 10).

Figura 4. Caso de prueba 2 (suma con carry in).

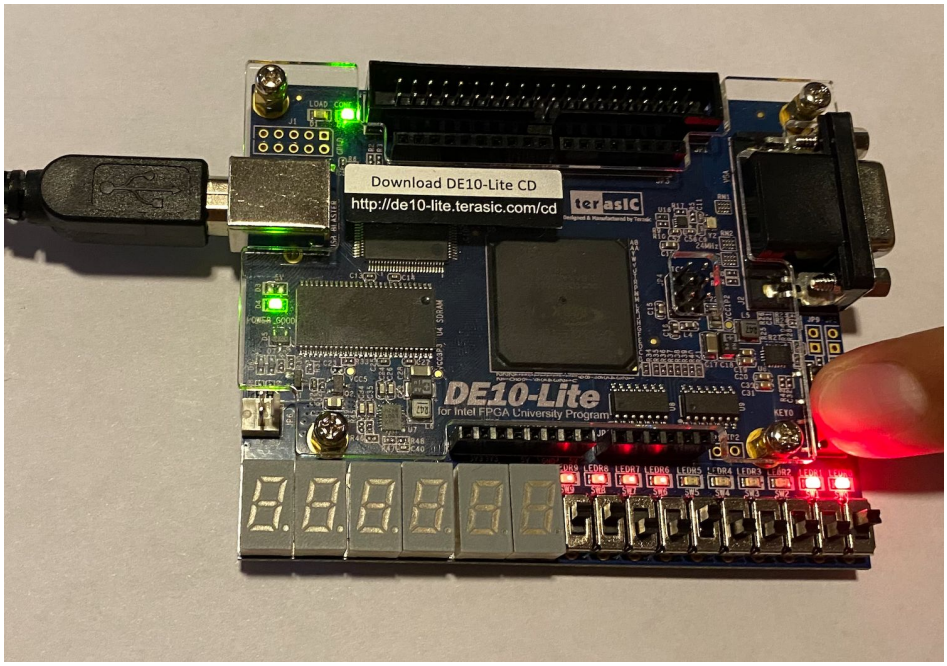


Figura 5. Caso de prueba 3 (resta normal).

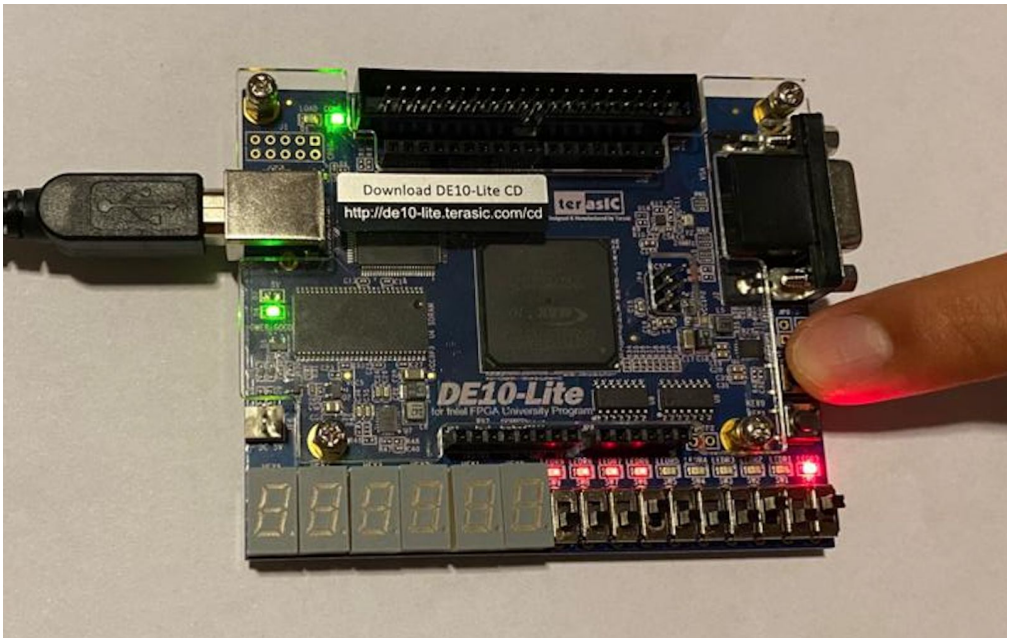


Figura 6. Caso de prueba 4 (resta con carry in).

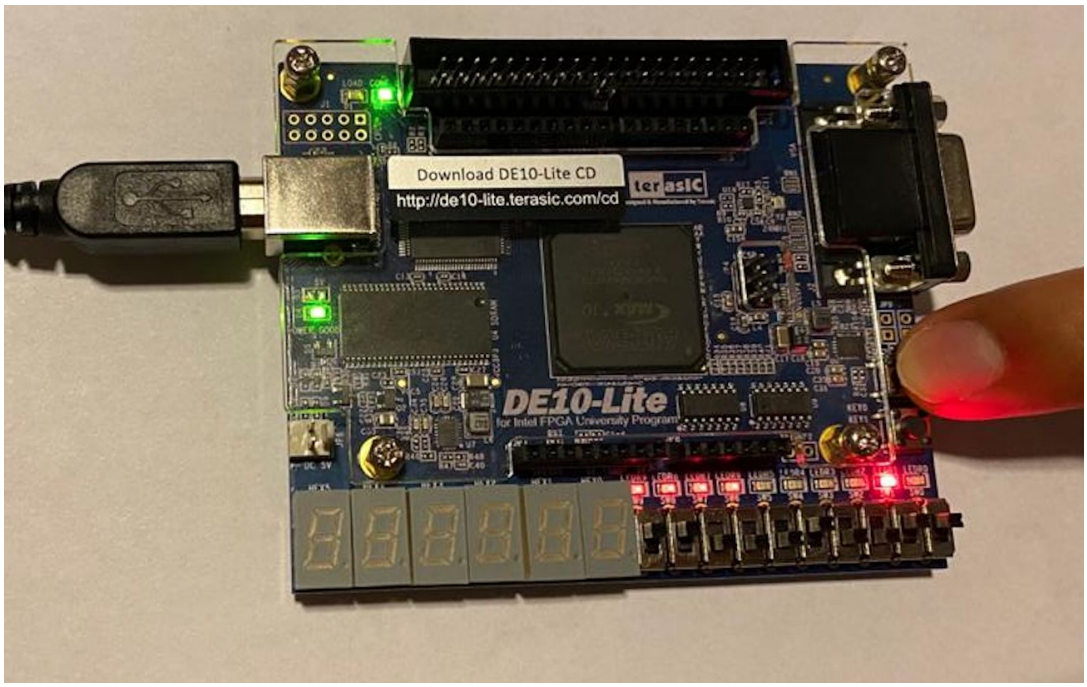


Figura 7. Caso de prueba 5 (operador lógico AND).

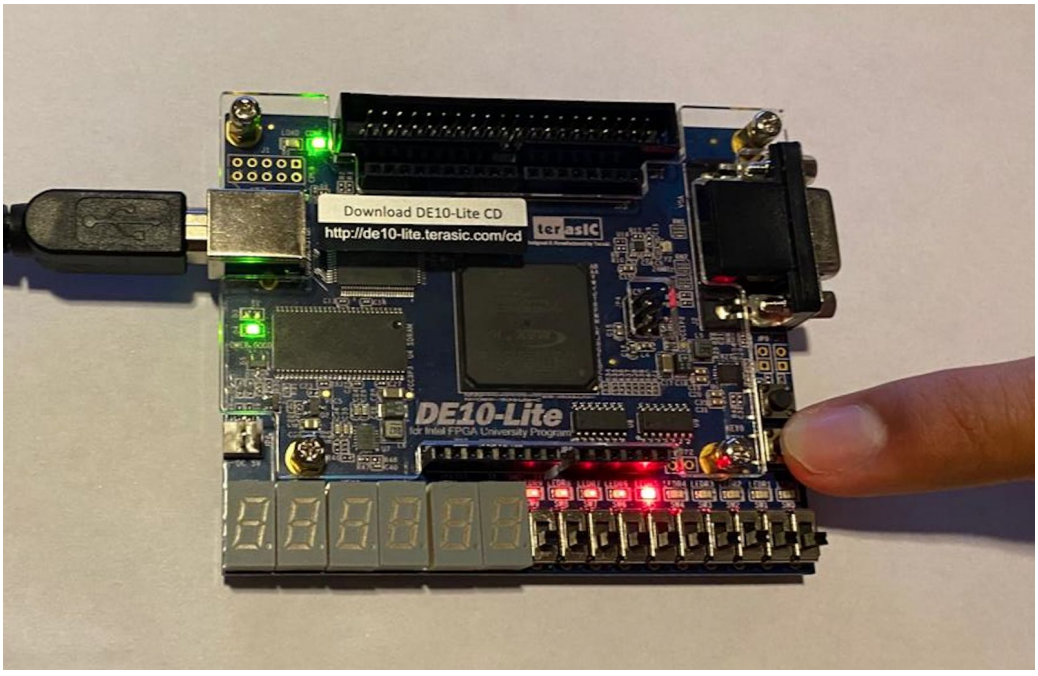


Figura 8. Caso de prueba 6 (operador lógico OR).

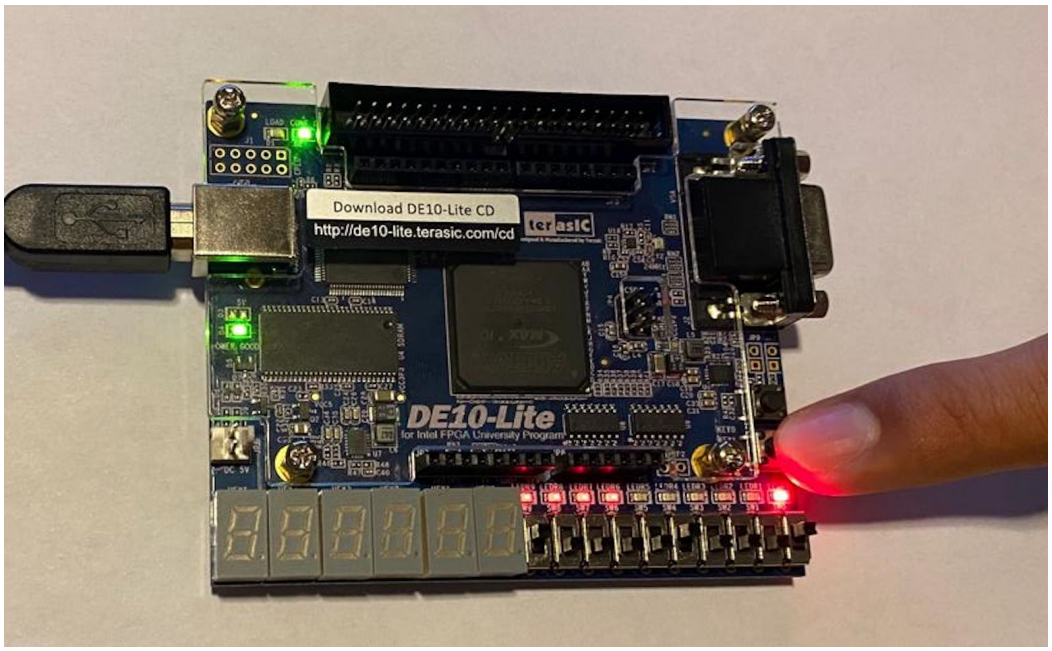


Figura 9. Caso de prueba 7 (operador lógico XOR).

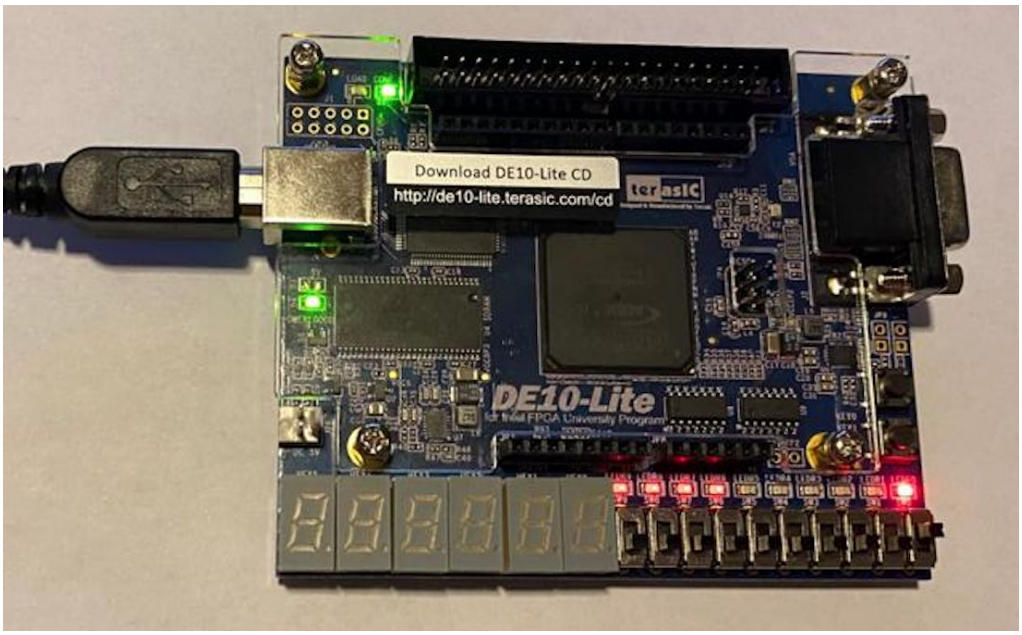
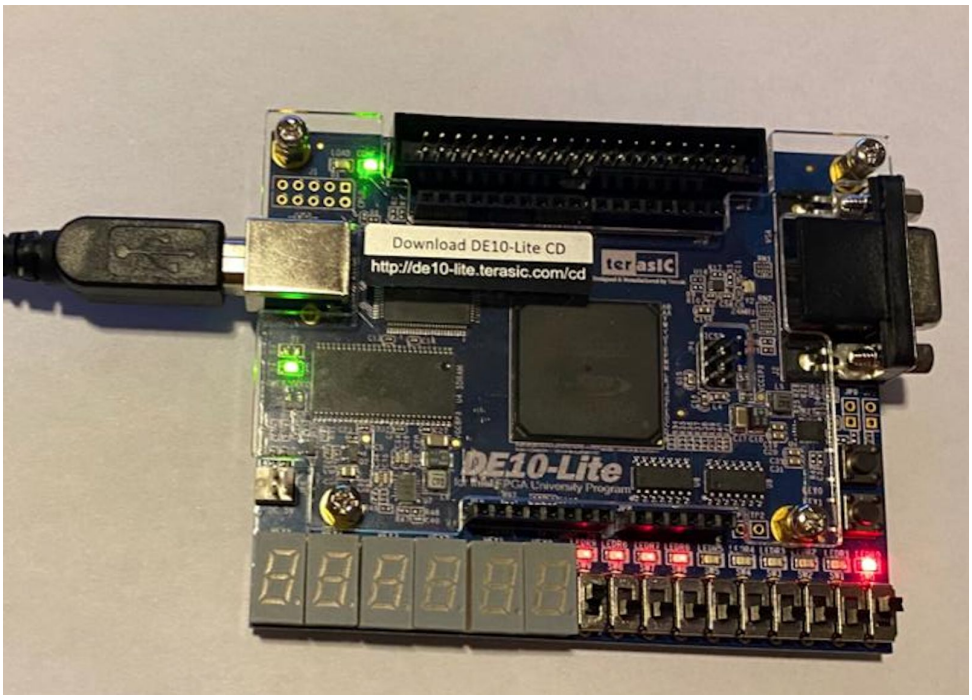


Figura 10. Caso de prueba 8 (MASK).



III. Conclusiones

En conclusión, gracias al desarrollo de esta práctica pude comprender mejor el funcionamiento de una Unidad Aritmética y Lógica, que es muy importante ya que es uno de los circuitos básicos de las computadoras y que gracias a su existencia es que pueden realizar una infinidad de operaciones en determinado tiempo. Además pude profundizar y reforzar más los conocimientos adquiridos sobre la programación en VHDL utilizando Quartus Prime.

IV. Referencias

[1] Intel.com. 2016. *Intel DE10-Lite User Manual*. [online] Available at: <https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-2912030810549-de10-lite-user-manual.pdf> [Accessed 21 February 2021].