UNIVERSIDAD SERGIO ARBOLEDA

DISEÑO E IMPLEMENTACIÓN DE UN SUMADOR Y RESTADOR DE 4 BITS USANDO COMPUERTAS AND, OR Y NOT

LENGUAJES DE PROGRAMACIÓN Y TRANSDUCCIÓN

DOCENTE

JOAQUIN SANCHEZ CIFUENTES

PRESENTADO POR

ANGEL ROBLES

1. Introducción

En este trabajo se presenta el diseño e implementación de un sumador y restador de 4 bits utilizando exclusivamente compuertas lógicas **AND, OR y NOT**

Para validar el funcionamiento del diseño lógico, se utilizó el software de simulación Proteus, y posteriormente se realizó una implementación equivalente en el lenguaje de programación **Python**, respetando las mismas operaciones lógicas utilizadas en el circuito

2. Objetivos

2.1 Objetivo General

Diseñar e implementar un sumador y restador de 4 bits utilizando únicamente compuertas AND, OR y NOT, validando su funcionamiento mediante simulación y programación

2.2 Objetivos Específicos

- Comprender el funcionamiento de las compuertas lógicas básicas
- Construir un sumador completo a partir de lógica elemental
- Implementar la operación de resta mediante complemento a dos
- Simular el circuito en Proteus
- Replicar el comportamiento lógico en Python

3. Fundamentos

3.1 Sistema binario

El sistema binario es un sistema de numeración que utiliza únicamente dos valores: 0 y 1. En los sistemas digitales, estos valores representan estados lógicos, donde 0 corresponde a nivel bajo y 1 a nivel alto

Un número de 4 bits puede representar valores desde 0000 hasta 1111 en binario, equivalentes a 0–15 en decimal

3.2 Compuertas Logicas

Compuerta NOT

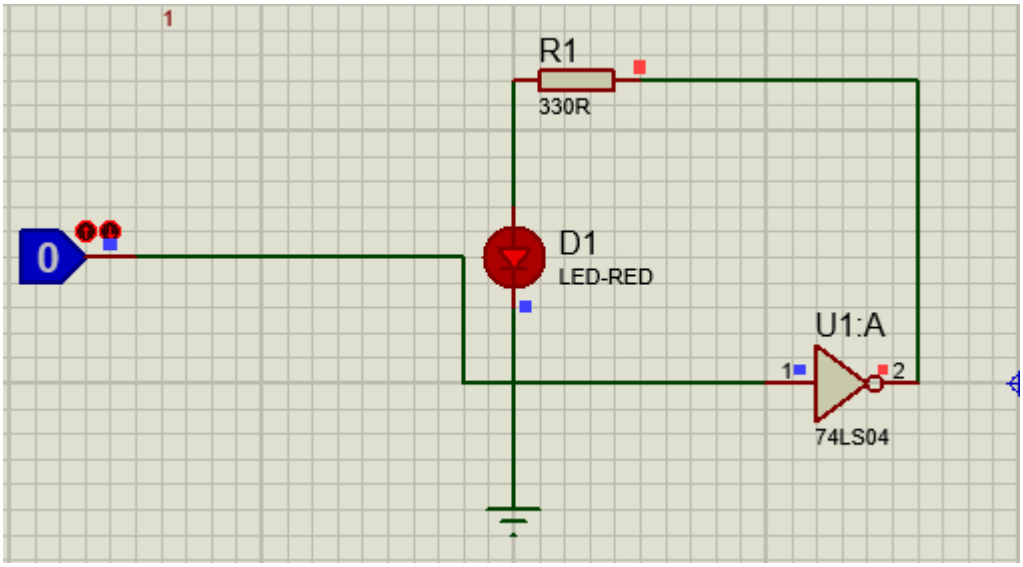


Figura 1 – Diseño Compuerta NOT (Proteus) LED ENCENDIDO

La compuerta NOT invierte el valor de la entrada. Si la entrada es 1, la salida será 0, y viceversa

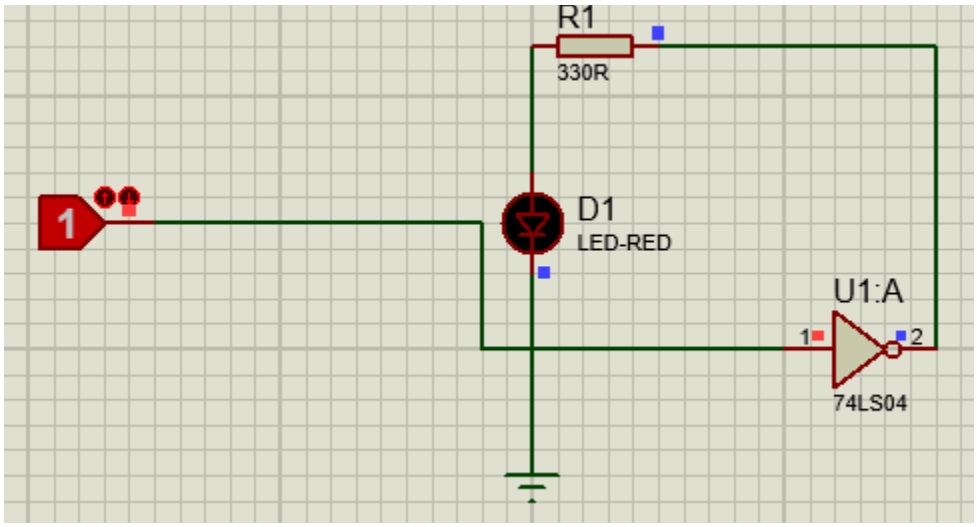


Figura 2 – Diseño Compuerta NOT (Proteus) LED APAGADO

Entrada	***Salida***
0	1
1	0

Compuerta AND

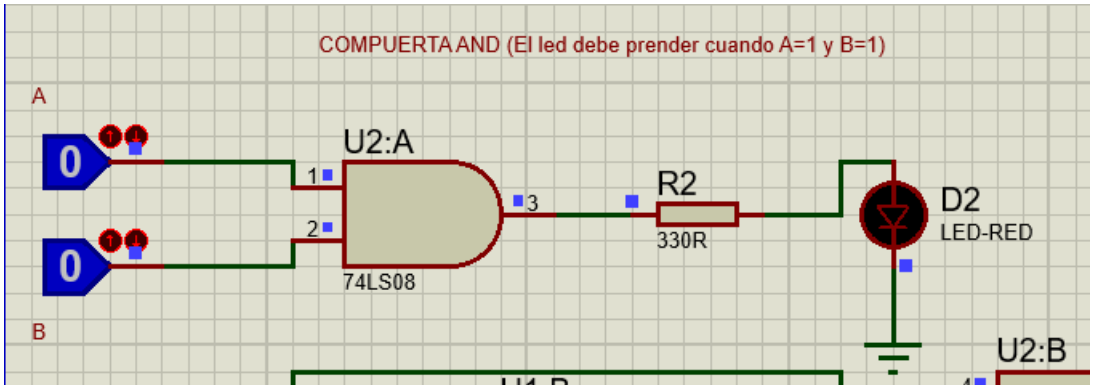


Figura 3 – Diseño Compuerta AND (Proteus) LED APAGADO (0-0)

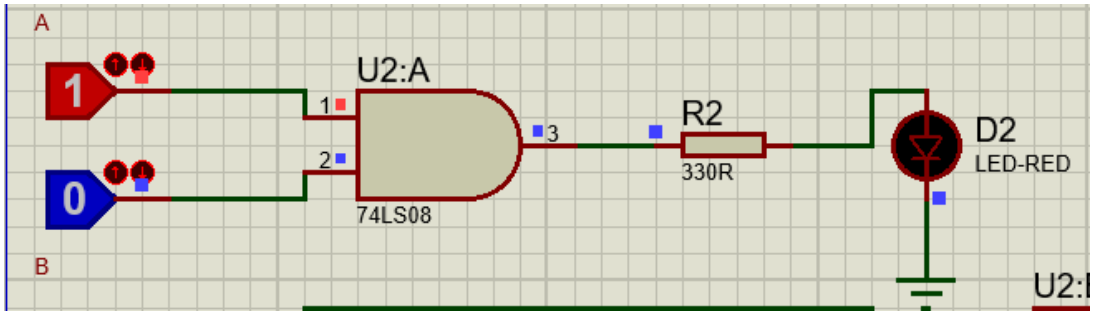


Figura 4 – Diseño Compuerta AND (Proteus) LED APAGADO (1-0)

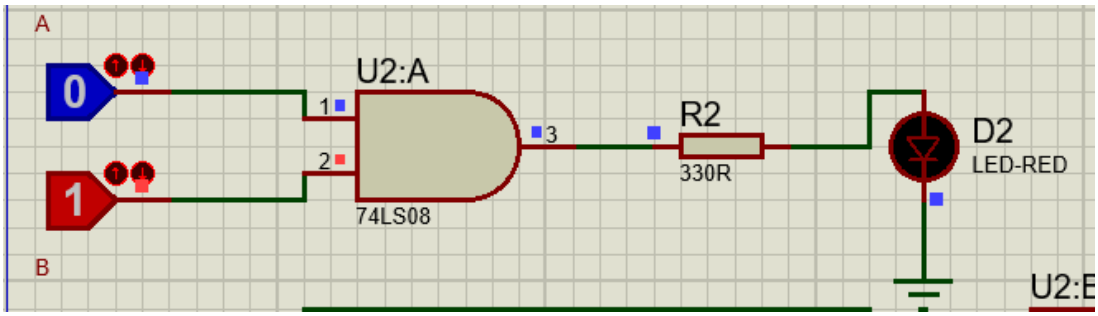


Figura 5 – Diseño Compuerta AND (Proteus) LED APAGADO (0-1)

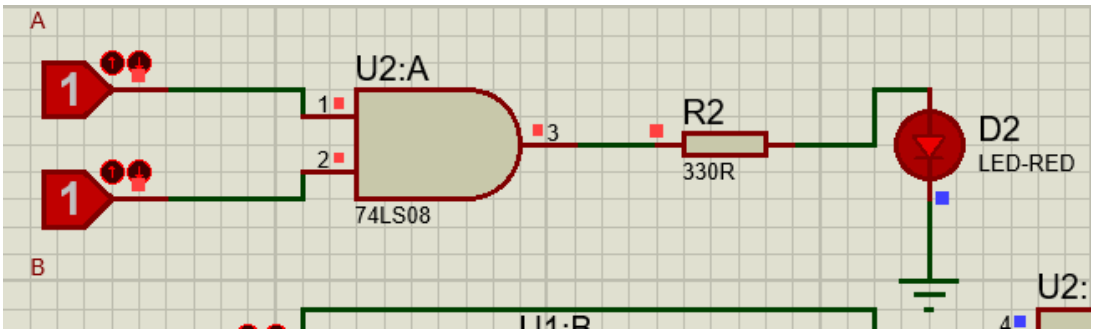


Figura 6 – Diseño Compuerta AND (Proteus) LED ENCENDIDO (1-1)

La compuerta AND produce una salida en 1 únicamente cuando todas sus entradas están en 1

A	*B*	*A AND B*
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Compuerta OR

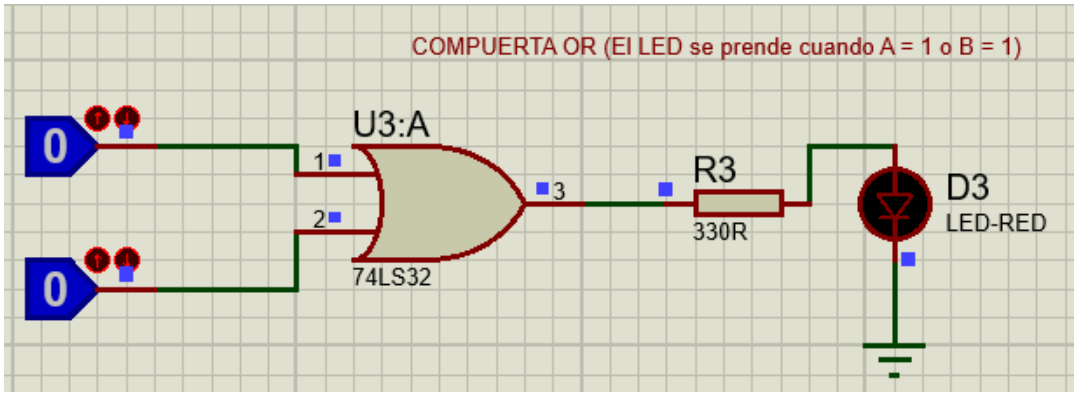


Figura 7 – Diseño Compuerta OR (Proteus) LED APAGADO (0-0)

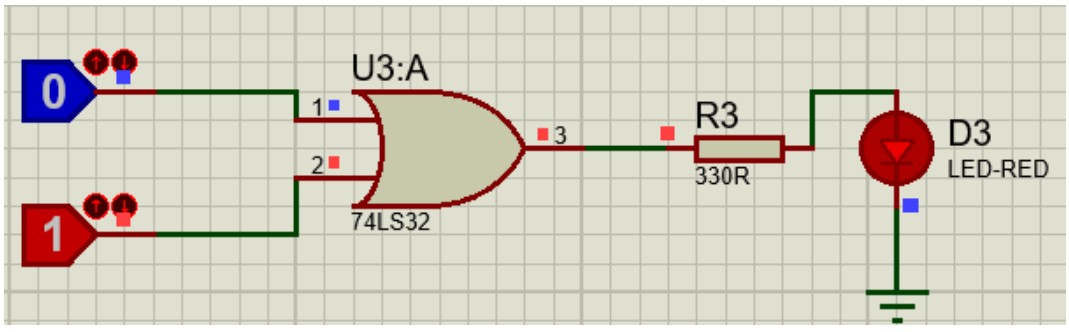


Figura 8 – Diseño Compuerta OR (Proteus) LED ENCENDIDO (0-1)

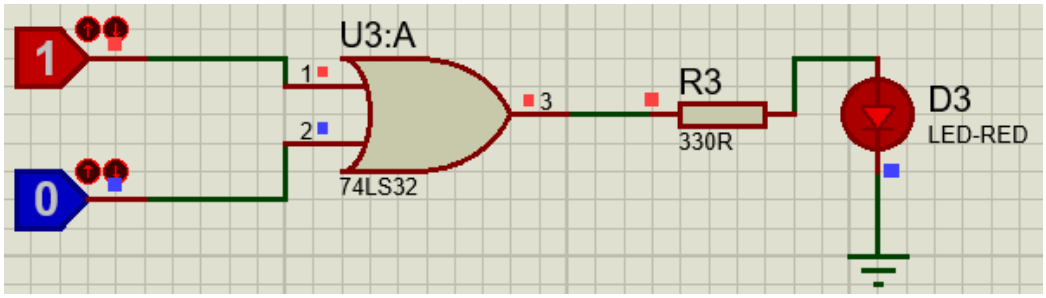


Figura 9 – Diseño Compuerta OR (Proteus) LED ENCENDIDO (1-0)

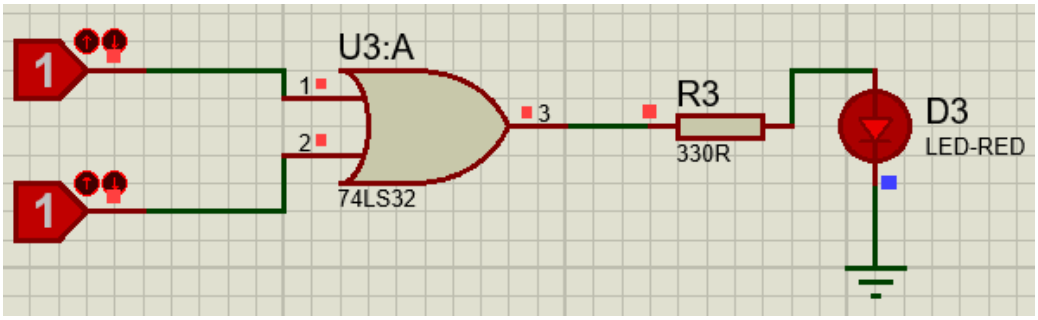


Figura 10 – Diseño Compuerta OR (Proteus) LED ENCENDIDO (1-1)

La compuerta OR produce una salida en 1 cuando al menos una de sus entradas es 1

A	***B***	***A OR B***
0	0	0
0	1	1
1	0	1
1	1	1

Compuerta XOR (Usando AND, OR y NOT)

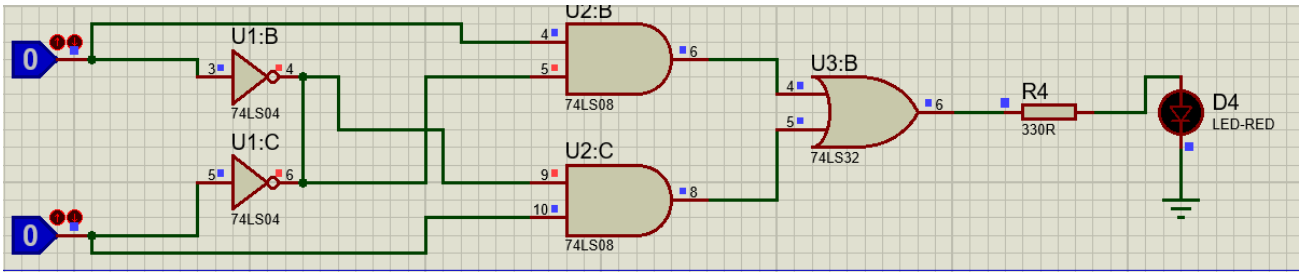


Figura 11 – Diseño Compuerta XOR (Proteus) LED APAGADO (0-0)

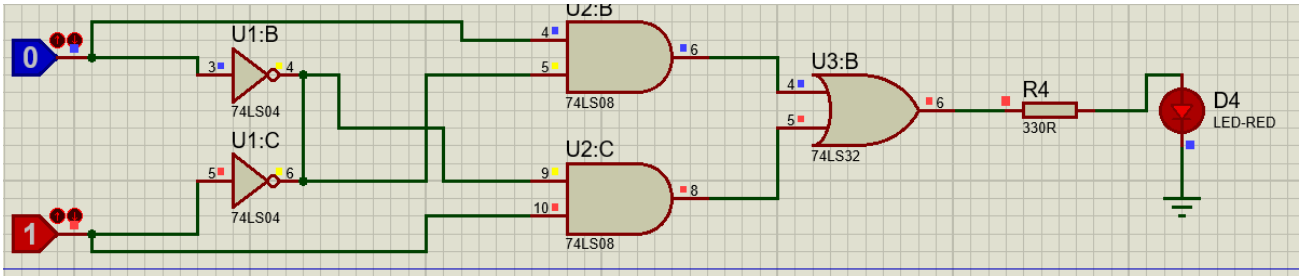


Figura 12 – Diseño Compuerta XOR (Proteus) LED ENCENDIDO (0-1)

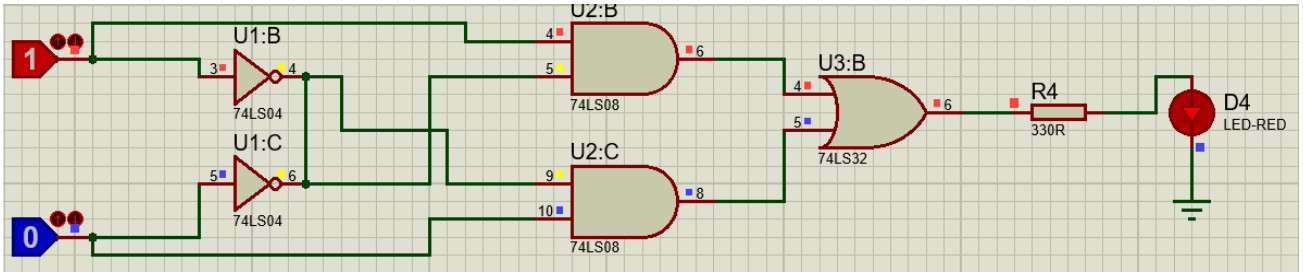


Figura 13 – Diseño Compuerta XOR (Proteus) LED ENCENDIDO (1-0)

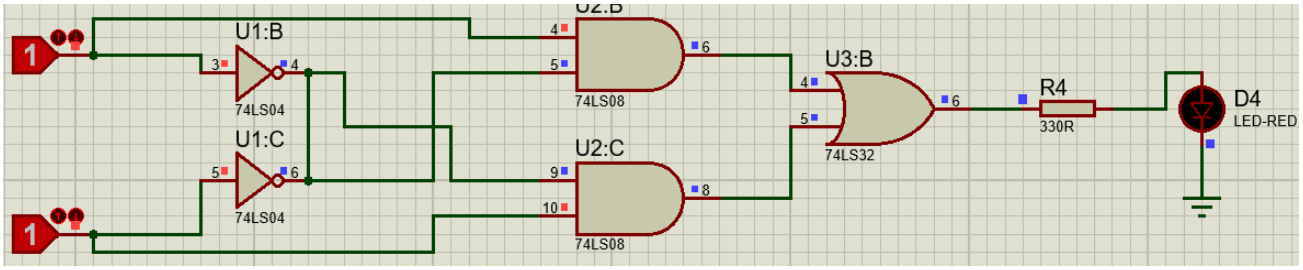


Figura 14 – Diseño Compuerta XOR (Proteus) LED APAGADO (1-1)

A	***B***	***XOR***
0	0	0
0	1	1
1	0	1
1	1	0

Dado que la compuerta XOR no estaba permitida directamente, se implementó utilizando compuertas básicas según la siguiente expresión lógica:

$$XOR\ (A, B) = (A\ AND\ \neg B)\ OR\ (\neg A\ AND\ B)$$

Sumador Completo

Es un circuito combinacional que permite realizar la suma de tres bits: dos bits de datos **(A y B)** y un bit adicional denominado acarreo de entrada **(C_n),** el cual proviene de una operación anterior.
Como resultado, el circuito genera dos salidas: el bit de suma **(S)** y el acarreo de salida **(C_{n+1})**

A	B	Acarreo de Entrada	Suma	Acarreo de Salida
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Suma

$$\boldsymbol{S = A \oplus B \oplus C}$$

Acarreo

$$Acarreo\ de\ Salida = (A \times B) + Acarreo\ de\ Entrada \times (A \oplus B))$$

Armar Sumador Completo (PASO A PASO)

1. **XOR entre A y B**

$$X1 = A \oplus B$$

2. **XOR con el acarreo**

$$S = X1 \oplus Cin$$

3. **AND entre A y B**

$$C1 = A \cdot B$$

4. **AND entre Acarreo de Entrada y X1**

$$C2 = Cin \cdot X1$$

5. **OR final**

$$Acarreo\ de\ Salida = C1 + C2$$

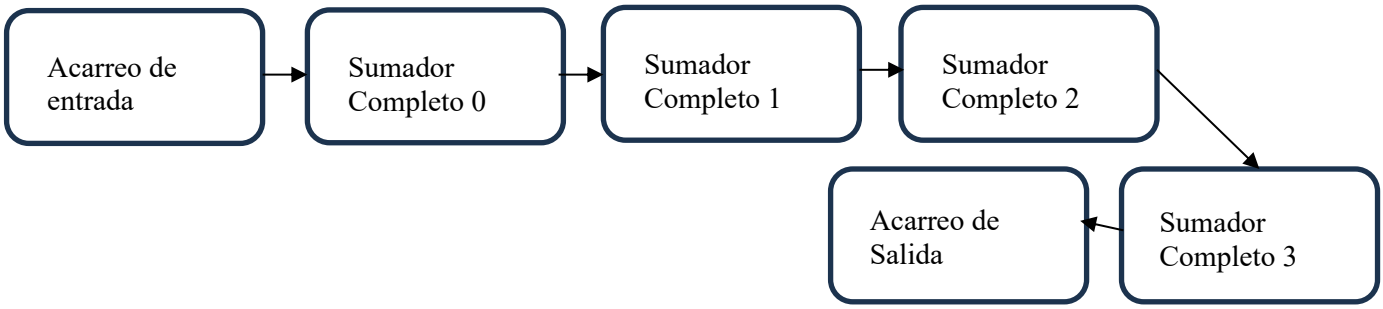
Extensión a 4 BITS

La extensión se realiza mediante la conexión en cascada de cuatro sumadores completos idénticos

El primer sumador completo **(FA0)** recibe los bits menos significativos $\boldsymbol{A_0}$**y** $\boldsymbol{B_0}$, y su entrada de acarreo $\boldsymbol{Cin}$ se fija en **0**

El acarreo de salida generado por cada sumador se conecta **directamente** como acarreo de entrada del siguiente sumador

Este proceso se repite hasta el cuarto sumador, que genera el bit más significativo de la suma y el acarreo final



Entradas	**Salidas**
$\boldsymbol{A = A_3\, A_2\, A_1\, A_0}$	**S = S₃ S₂ S₁ S₀**
B = B₃ B₂ B₁ B₀	**Acarreo Final**

Resta

Para comprobar el correcto funcionamiento de la operación de resta, se realizaron pruebas directas utilizando diferentes combinaciones de entrada. La resta se implementa activando la señal de control MODE = 1, lo que permite invertir el operando B y sumar una unidad automáticamente

1. Se realiza la operación 5 – 3

SEÑAL	**VALOR**
A	0101
B	0011
MODE	1

S = 0101 – 0011 = 0010

2. Se realiza la operación 3 - 5

SEÑAL	VALOR
A	0011
B	0101
MODE	1

S = 0011 – 0101 = 1110

Se realiza la operación 6 – 0

SEÑAL	VALOR
A	0110
B	0000
MODE	1

S = 0110

4. Implementación en Python

Con el fin de replicar el comportamiento del circuito digital, se implementó el sumador y restador de 4 bits en el lenguaje Python, utilizando funciones que simulan las compuertas lógicas AND, OR y NOT.

```python
def NOT(a):
    return 1 if a == 0 else 0

def AND(a, b):
    return 1 if a == 1 and b == 1 else 0

def OR(a, b):
    return 1 if a == 1 or b == 1 else 0
```

Figura 15 – Implementación Funciones Lógicas Basicas

```python
def XOR(a, b):
    return OR(AND(a, NOT(b)), AND(NOT(a), b))
```

Figura 16 – Implementación XOR con NOT, AND y OR

```python
def SumadorCompleto (A, B, Cin):
    X1 = XOR(A, B)
    S = XOR(X1, Cin)

    C1 = AND(A, B)
    C2 = AND(Cin, X1)
    Cout = OR(C1, C2)

    return S, Cout
```

Figura 17 – Sumador Completo de 1 Bit

Este bloque permite sumar dos bits y un acarreo de entrada, generando una suma y un acarreo de salida.

```python
def Sumador4Bits (A, B, MODE):
    # MODE = 0  suma
    # MODE = 1  resta

    result = []
    carry = MODE  # acarreo inicial

    for i in range(4):
        B_mod = XOR(B[i], MODE)  # invierte B si es resta
        s, carry = SumadorCompleto(A[i], B_mod, carry)
        result.append(s)

    return result, carry
```

Figura 18 – Sumador y Restador de 4 Bits

Pruebas

1. **Suma**

```
49   A = [1, 0, 1, 0]  # 0101 = 5
50   B = [1, 1, 0, 0]  # 0011 = 3
51
52   resultado, carry = Sumador4Bits(A, B, 0)
53
54   print("Suma A + B:", resultado[::-1], "Carry:", carry)
```

Figura 19 – Suma 5 + 3

```
c:/Users/angel/OneDrive - Universic
Suma A + B: [1, 0, 0, 0] Carry: 0
```

Figura 20 – Suma Correctamente = 8

2. **Resta**

```
resultado, carry = Sumador4Bits(A, B, 1)

print("Resta A - B:", resultado[::-1], "Carry:", carry)
```

Figura 21 – Resta 5 – 3

```
Resta A - B: [0, 0, 1, 0] Carry: 1
PS C:\Users\angel\OneDrive - Univers
```

Figura 22 – Resta Correctamente = 2

3. **Resta con resultado negativo**

```
62   A = [1, 1, 0, 0]  # 0011 = 3
63   B = [1, 0, 1, 0]  # 0101 = 5
64
65   resultado, carry = Sumador4Bits(A, B, 1)
66
67   print("Resta A - B:", resultado[::-1], "Carry:", carry)
68
```

Figura 23 – Resta negativa 3 – 5

```
Resta A - B: [1, 1, 1, 0] Carry: 0
PS C:\Users\angel\OneDrive - Universi
```

Figura 24 – Resta negativa correctamente = -2

Los resultados obtenidos en Python coinciden con los resultados observados en la simulación realizada en Proteus, validando el diseño lógico del sistema.