UNIVERSIDAD SERGIO ARBOLEDA

Sumador 4bits

DOCENTE

JOAQUIN SANCHEZ CIFUENTES

PRESENTADO POR

JULIAN CAMILO CARO HERRERA

INTRODUCCION

En el presente informe se describe el diseño, implementación y verificación de un circuito digital capaz de realizar operaciones de suma y resta binaria, basado en la interconexión de sumadores completos construidos únicamente con compuertas lógicas básicas (AND, OR y NOT).

Este tipo de circuitos constituye un bloque fundamental en sistemas digitales como la Unidad Aritmético-Lógica (ALU) de los procesadores, donde se requiere realizar operaciones aritméticas elementales de forma confiable y eficiente. El objetivo principal de esta actividad es comprender el funcionamiento interno de un sumador completo, extenderlo a un sistema de varios bits y adaptarlo para que pueda realizar tanto suma como resta mediante una señal de control.

DISEÑO

1. Diseño y explicación funcional del circuito

El diseño del sumador–restador se desarrolló a partir de un enfoque **modular y funcional**, donde cada parte del circuito cumple un propósito específico dentro de la operación de suma y resta. En lugar de analizar compuertas de forma aislada, el diseño se explica mediante **bloques funcionales**, lo que permite comprender de manera clara el flujo de la información y las decisiones de diseño tomadas.

1.1 Señal de control de modo (M)

La señal **M** es el elemento central que define el comportamiento del circuito:

- **M = 0**: el circuito realiza una suma binaria
- **M = 1**: el circuito realiza una resta binaria

Esta señal cumple dos funciones simultáneas:

1. Controla la modificación del operando B
2. Actúa como el carry de entrada inicial (CI) del primer sumador

Gracias a esta doble función, el circuito puede implementar la resta mediante el método de complemento a dos.

Resumiendo, M es el valor que se decide poner al logicstate para que realice una suma o una resta en caso de ser 1

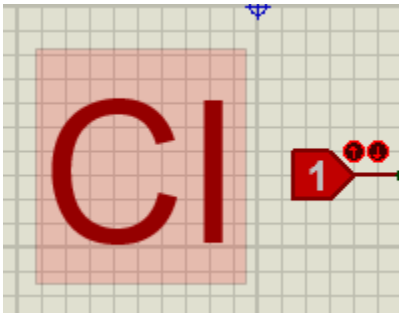

Figura 1 (Ubicación de la señal de control M)

Después de la descripción anterior de como funcionaba M, en la figura 1 se puede apreciar como se encuentra en el diseño

1.2 Bloque de modificación del operando B

Para implementar la resta binaria es necesario trabajar con el complemento a dos del operando B.

Cuando **M = 0**, el valor de B no se modifica y el circuito opera como un sumador convencional. Cuando **M = 1**, cada bit de B se invierte, preparando el operando para la operación de resta.

La operación XOR se implementa exclusivamente mediante compuertas AND, OR y NOT, cumpliendo con las restricciones del enunciado. Este bloque permite reutilizar el mismo sumador completo tanto para suma como para resta.

1.3 Generación de la señal intermedia X

Una vez obtenido el operando modificado B', se genera una señal intermedia.

Esta señal representa una **suma parcial** entre los operandos principales, independiente del carry de entrada. La introducción de la señal X permite dividir el proceso de suma en dos etapas claramente diferenciadas:

1. Combinación de los operandos A y B'
2. Incorporación del acarreo

Esta separación simplifica el diseño, mejora su claridad y facilita el análisis del comportamiento interno del circuito.

1.4 Incorporación del carry de entrada (CI)

El carry de entrada se combina con la señal X para generar el bit de suma final S:

En el primer bit del circuito, el carry de entrada se define como CI = M. Esto permite que:

- En suma (M = 0), el carry inicial sea cero
- En resta (M = 1), se sume la unidad necesaria para completar el complemento a dos

En los bits siguientes, el carry de entrada proviene directamente del carry de salida del bit anterior, garantizando la correcta propagación del acarreo a lo largo del circuito.

1.5 Generación del carry de salida (COUT)

El carry de salida se genera cuando al menos dos de las tres entradas del sumador (A, B' y CI) se encuentran en nivel lógico alto. Esta condición se implementa mediante la expresión lógica:

$$COUT = (A \cdot B') + (A \cdot CI) + (B' \cdot CI)$$

El carry de salida se conecta al siguiente bloque como carry de entrada, formando una estructura en cascada que permite operar con múltiples bits.

1.6 Integración del sumador–restador de varios bits

Un **sumador completo** es un circuito combinacional que suma tres bits de entrada:

- A: bit del primer operando
- B: bit del segundo operando
- CI (Carry In): acarreo de entrada

Y produce dos salidas:

- S (Suma)
- COUT (Carry Out): acarreo de salida

El sistema completo se construyó conectando varios sumadores completos de 1 bit en cascada, formando un **sumador–restador de varios bits**. Cada bloque procesa un bit de los operandos y propaga el carry al siguiente bloque.

Este enfoque modular permite:

- Escalar fácilmente el diseño a más bits
- Analizar cada etapa de forma independiente
- Garantizar un comportamiento coherente tanto en suma como en resta

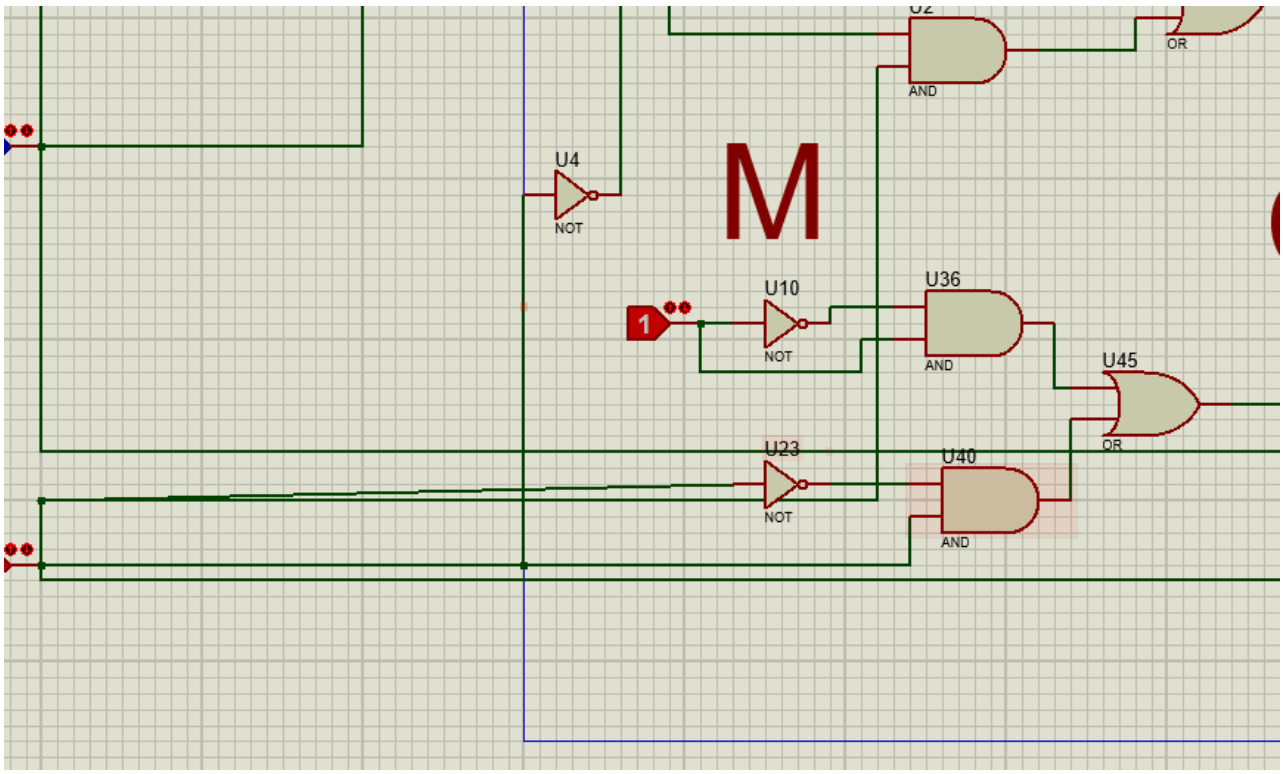

Figura 2 (restador de 1 bit)

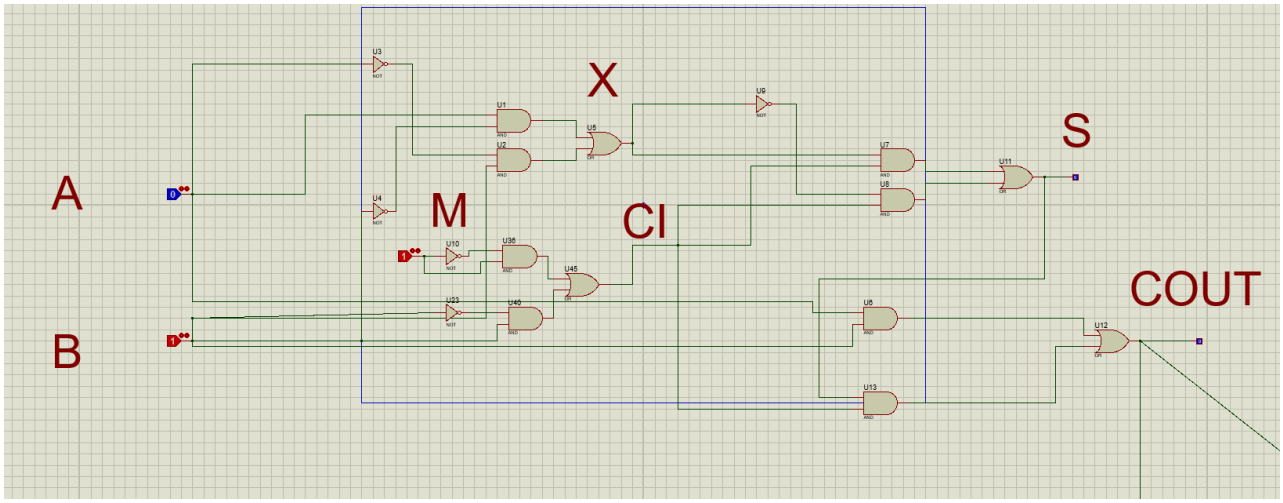

Figura 3 (sumador de 1 bit)

En las figuras anteriores se pueden observar cómo están diseñados el sumador y restador de 1 bit, teniendo en cuenta que el logicstate (M) es el que decidirá qué operación se realiza en este caso la figura 2 realiza resta mientras que la figura 3 hace suma

1.7 Diseño del sumador–restador

Para adaptar el sumador completo a un sumador–restador:

- Se añadió la señal de control M.
- Cada bit B pasa por una operación XOR con M.
- M se conecta como CI en el primer sumador.

Este diseño permite realizar suma y resta sin modificar la estructura base del sumador completo.

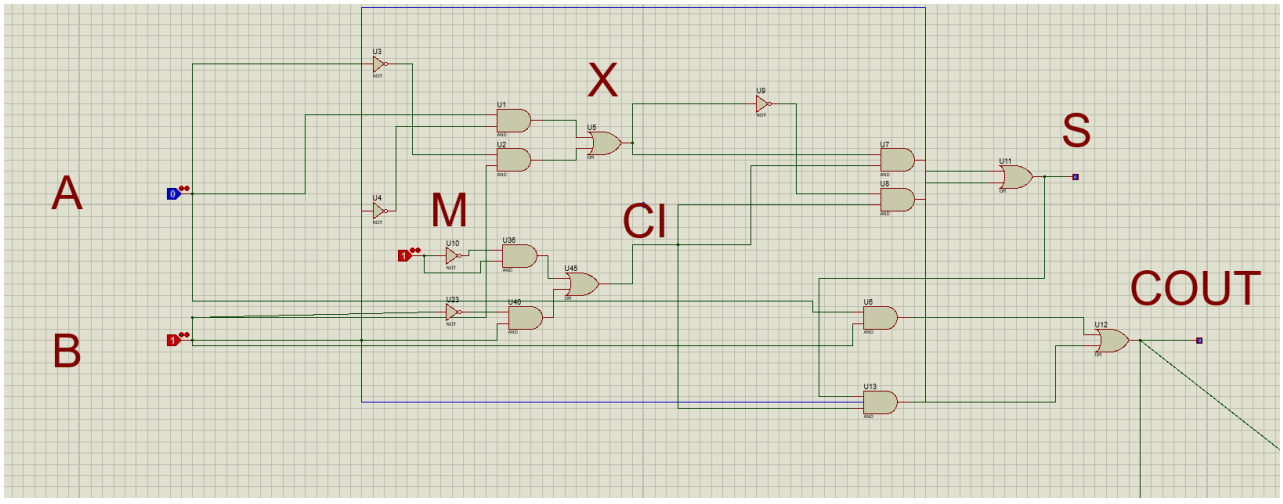

Figura 4 (sumador y restador de 1 bit)

1.8 Sumador–restador de varios bits

El sistema final se construyó conectando varios sumadores completos en cascada:

- El COUT de cada sumador se conecta al CI del siguiente.
- Cada etapa maneja un bit del operando.

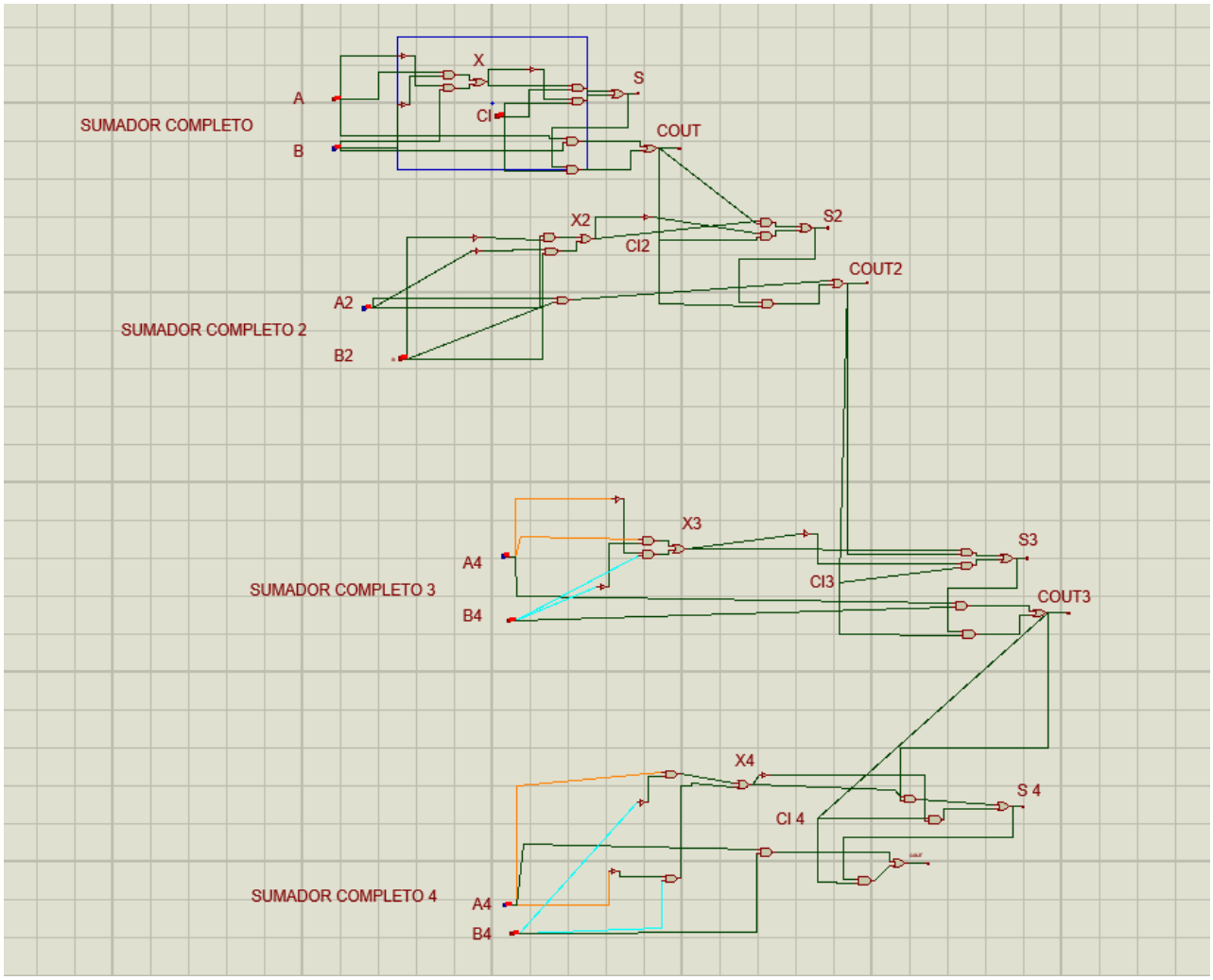

Figura 5 (sumador y restador de 4 bit)

2. Pruebas y Resultados

2.1 Pruebas de suma

Se realizaron pruebas con M = 0, verificando que el circuito realiza correctamente la suma binaria para diferentes combinaciones de A y B.

Ejemplo:

- A = 0101
- B = 0011
- Resultado = 1000

El circuito produjo el resultado esperado.

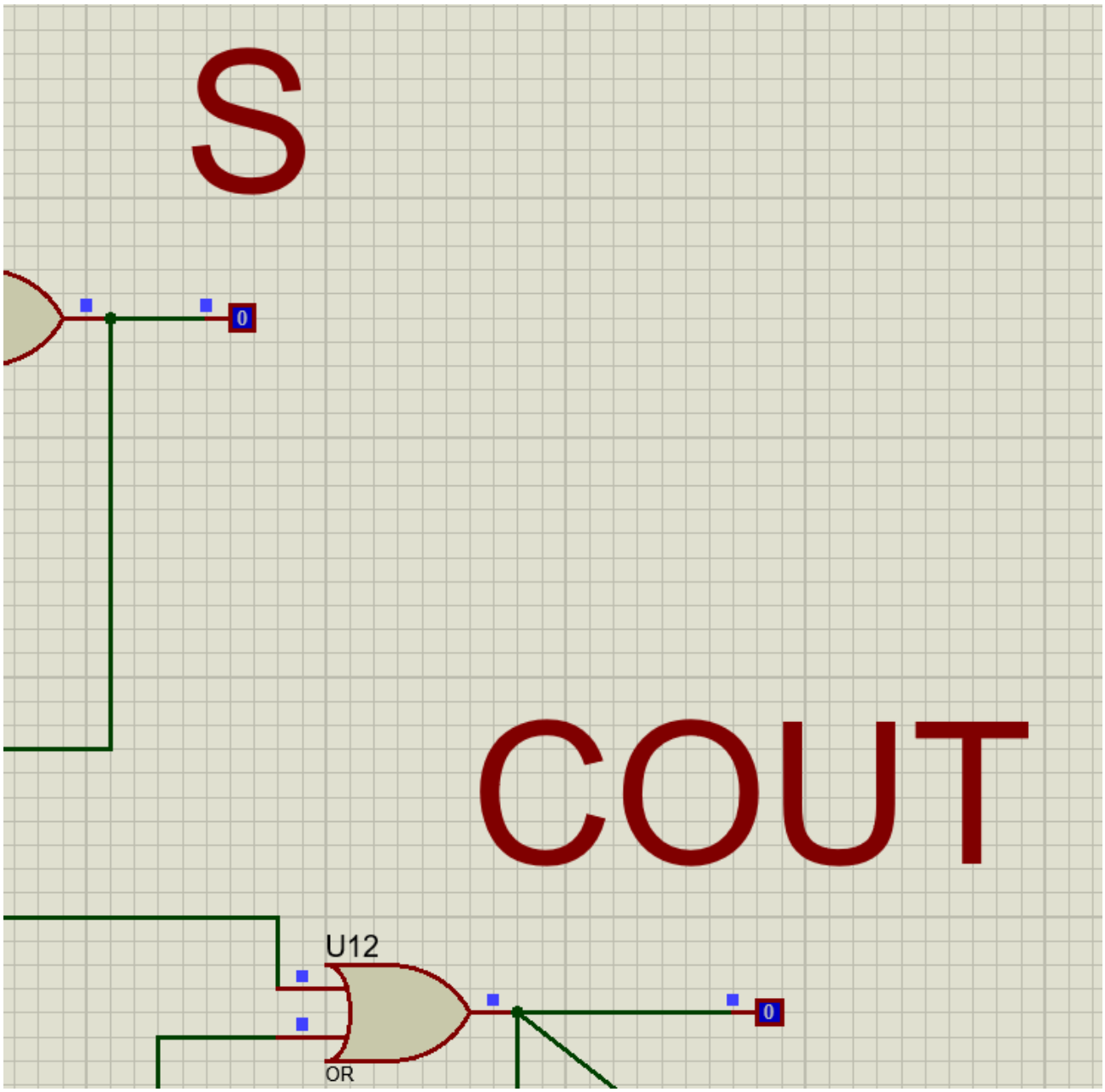

Figura 6 (Resultado bit 1)

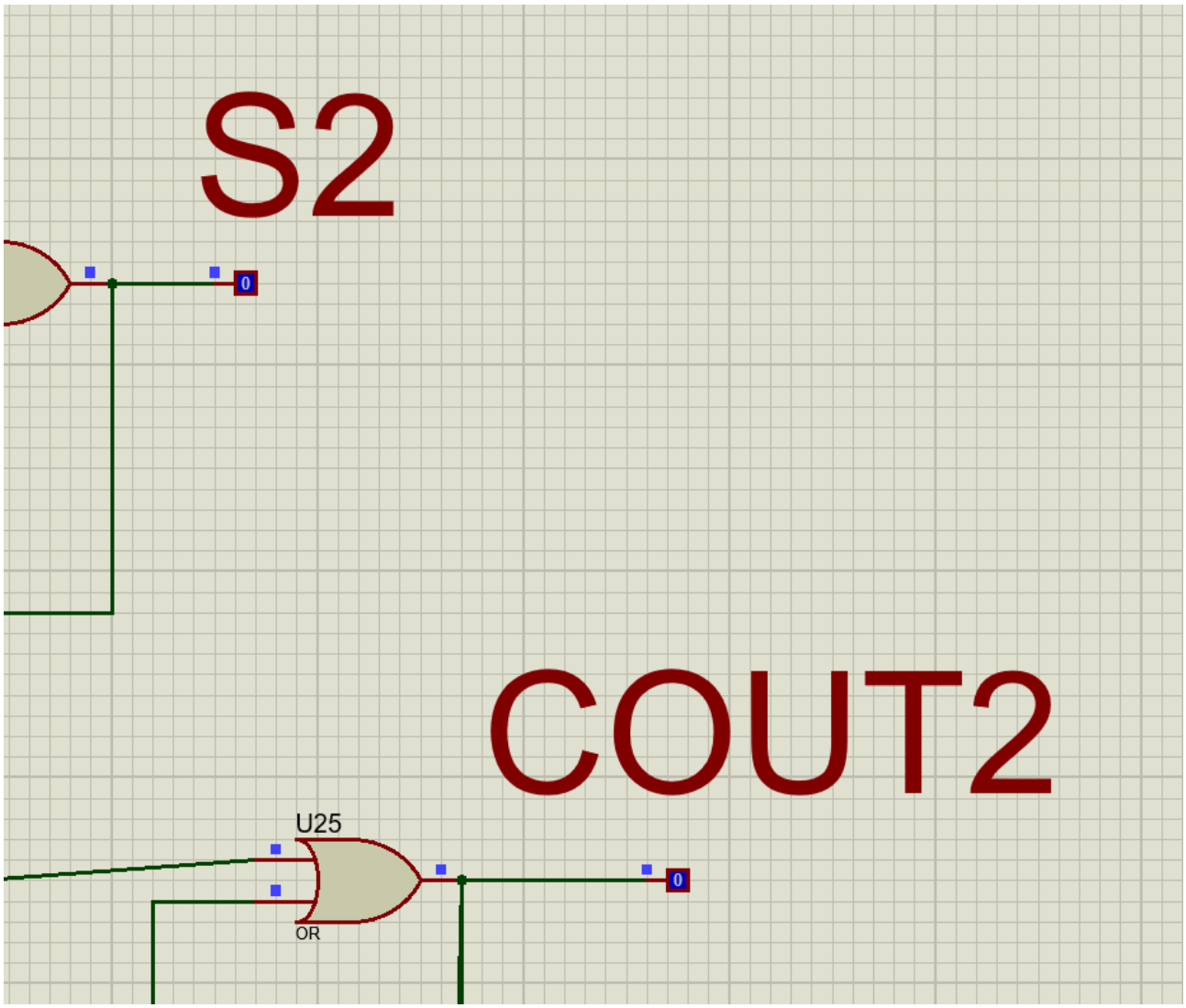

Figura 7(Resultado bit 2)

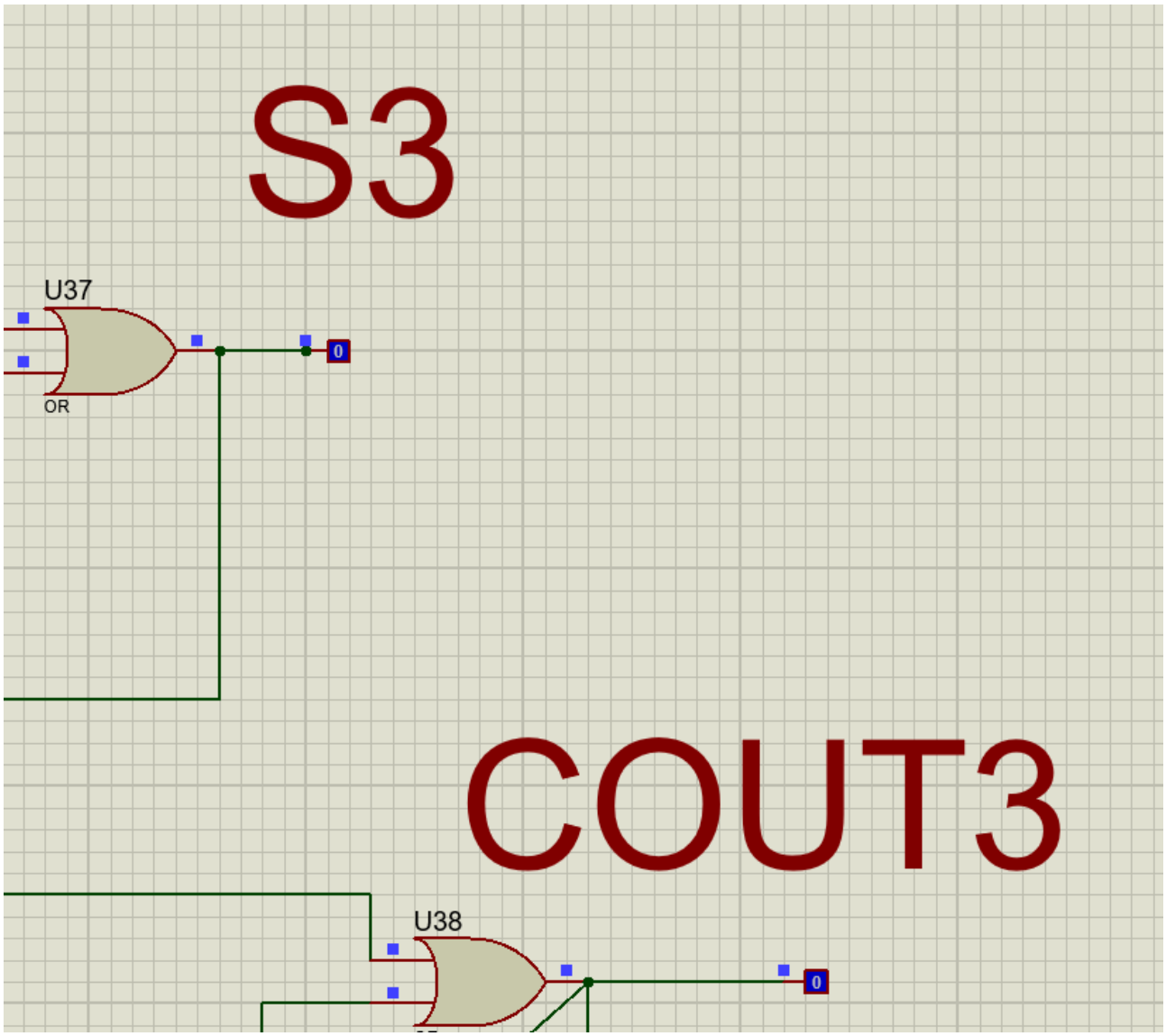

Figura 8 (resultado bit 3)

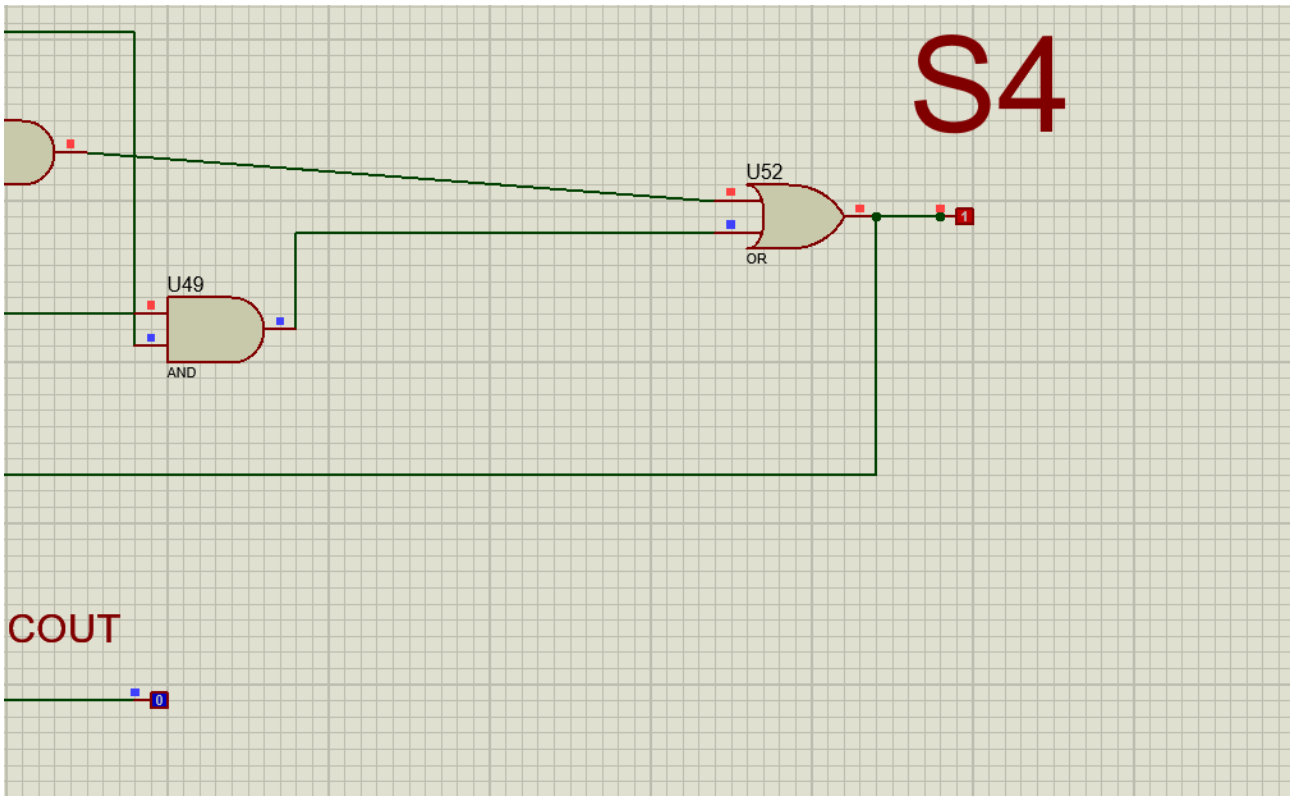

Figura 9(Resultado bit 4)

- A = 0101
- B = 0011
- Resultado = 1000

El circuito produjo el resultado esperado.

2.2 Pruebas de resta

Se realizaron pruebas con M = 1, verificando que el circuito realiza correctamente la resta binaria utilizando complemento a dos.

Ejemplo:

- A = 0101
- B = 0011
- Resultado = 0010

El circuito produjo el resultado correcto.

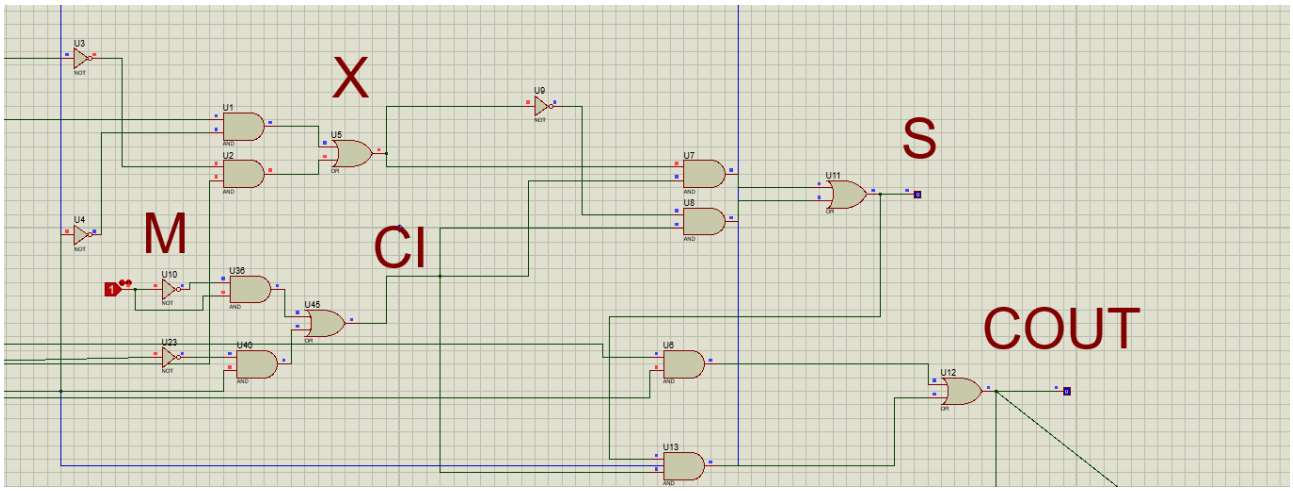

Figura 10(Resultado primer bit con M como 1(Resta) da 0)

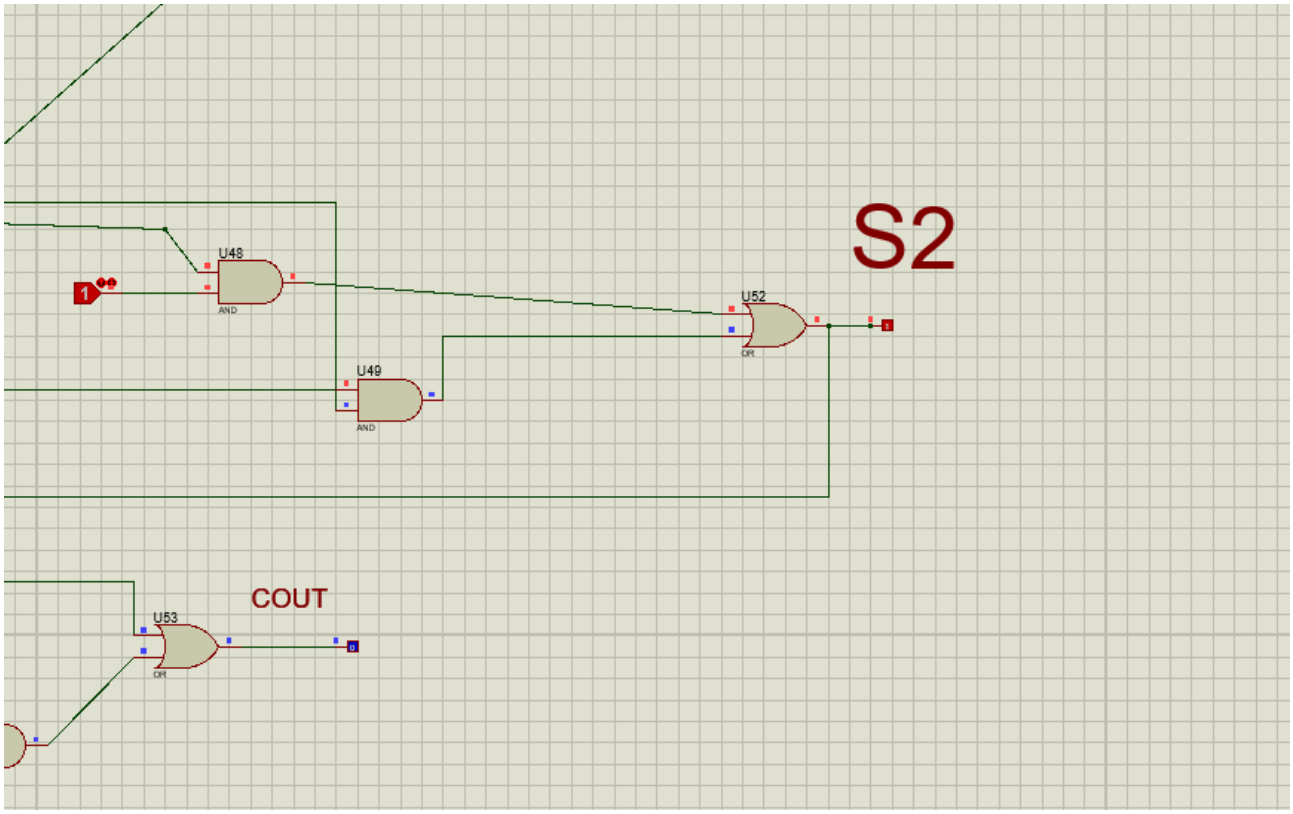

Figura 11(Resultado segundo bit)

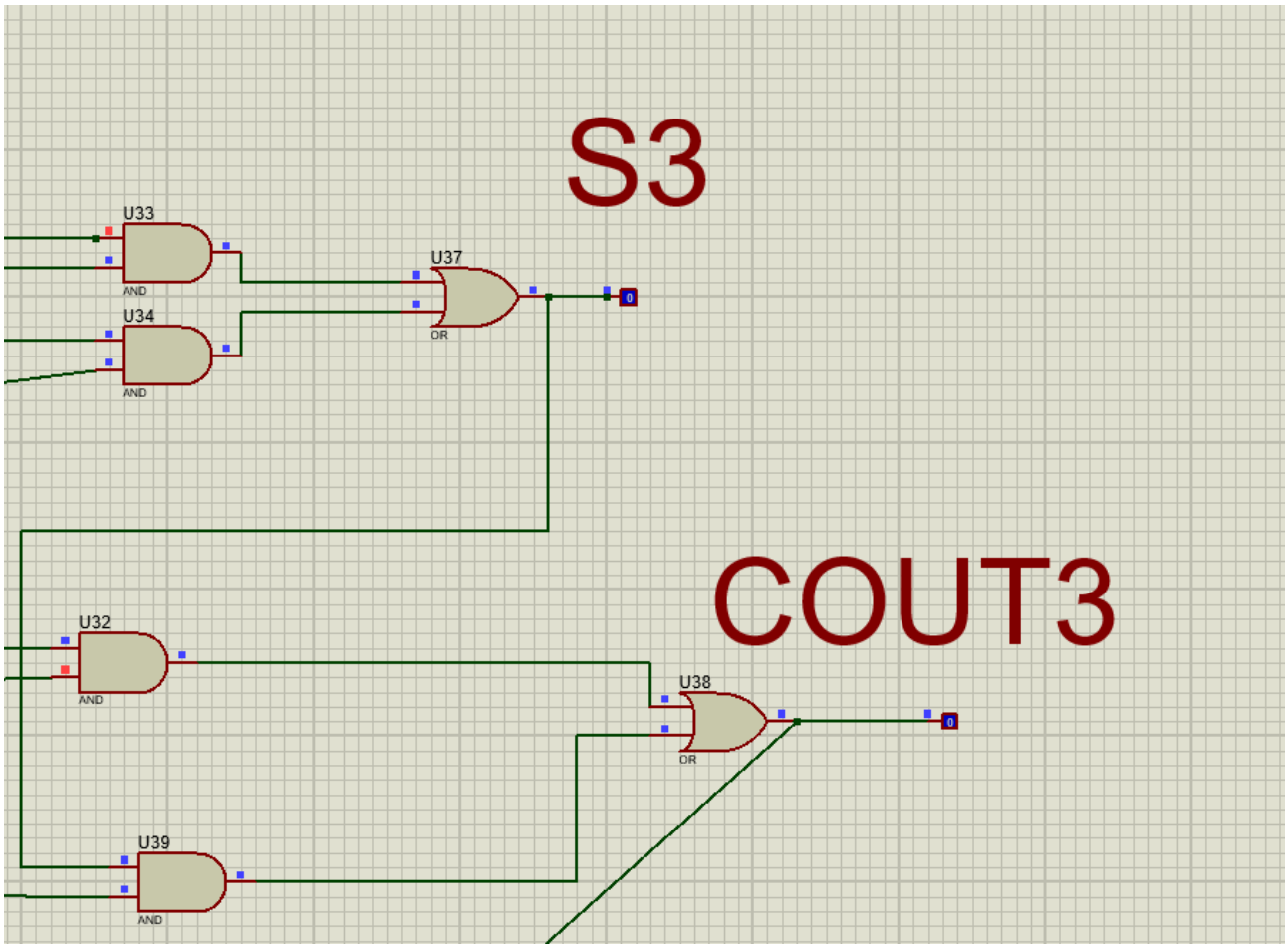

Figura 12(Resultado tercer bit)

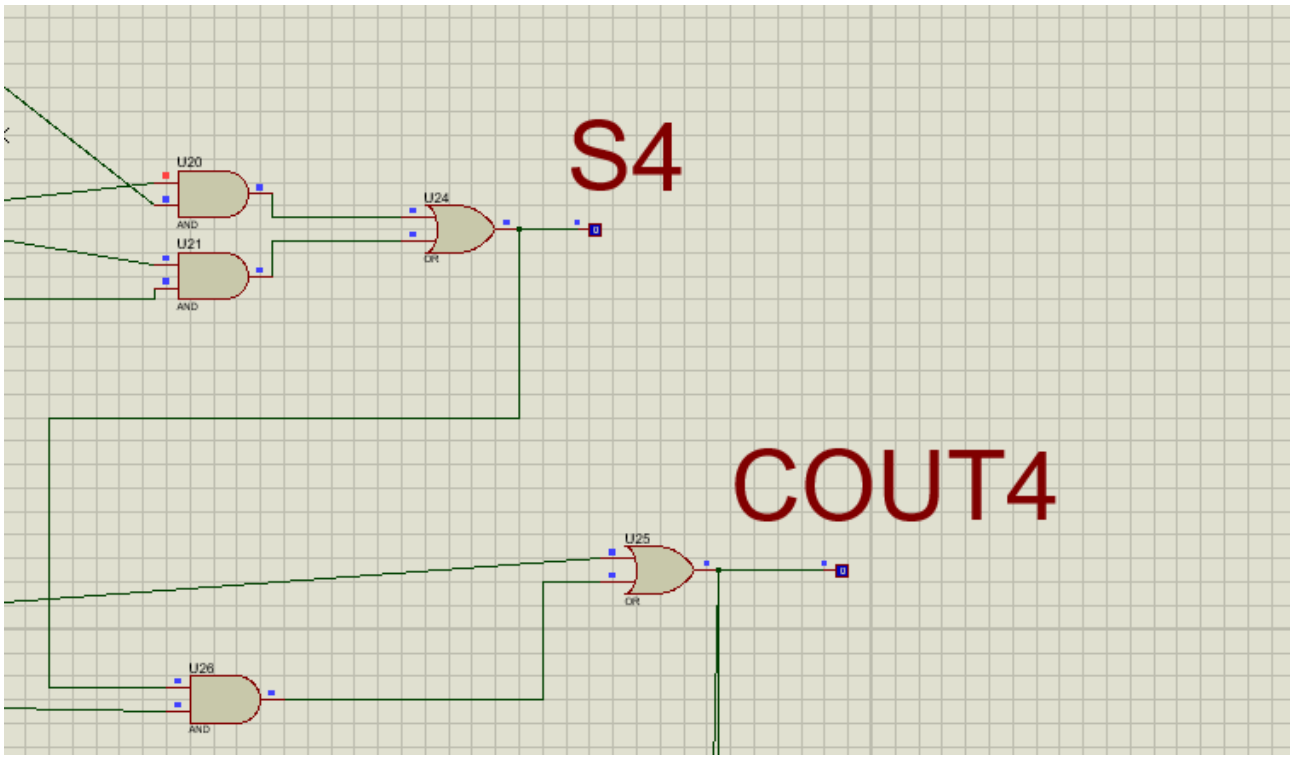

Figura 13(Resultado cuarto bit)

- A = 0101
- B = 0011
- Resultado = 0010

El circuito produjo el resultado correcto.

3. Implementación del sumador–restador binario

La implementación del sistema se realizó en el lenguaje **Python**, con el objetivo de reproducir el comportamiento lógico del circuito diseñado en Proteus. El código respeta estrictamente la restricción del enunciado, utilizando únicamente operaciones lógicas equivalentes a las compuertas **AND, OR y NOT**, sin emplear operadores aritméticos ni compuertas XOR directas.

El diseño implementado en software replica la estructura del circuito físico, incluyendo la señal de control **M**, la propagación del acarreo y el uso del complemento a dos para la operación de resta.

3.1 Implementación de las compuertas lógicas básicas

En primer lugar, se definieron funciones que representan las compuertas lógicas básicas AND, OR y NOT. Estas funciones constituyen la base de todo el sistema, ya que a partir de ellas se construyen operaciones más complejas como la XOR y el sumador completo.

Cada función recibe entradas binarias (0 o 1) y devuelve el resultado lógico correspondiente, simulando el comportamiento de las compuertas físicas utilizadas en el diseño del circuito.

```python
# COMPUERTAS LOGICAS BASICAS
def AND(a, b):
    return a & b

def OR(a, b):
    return a | b

def NOT(a):
    return 1 - a
```

Figura 14(definición de las compuertas)

3.2 Construcción de la compuerta XOR mediante compuertas básicas

Debido a que no se permite el uso directo de compuertas XOR, esta operación se implementó utilizando la identidad lógica:

A⊕B=(A·¬B)+(¬A·B)A \oplus B = (A \cdot \neg B) + (\neg A \cdot B)A⊕B=(A·¬B)+(¬A·B)

Esta implementación es equivalente a la utilizada en el diseño del circuito y permite obtener una señal de suma parcial sin violar las restricciones del enunciado. La compuerta XOR construida de esta forma se utiliza tanto para la generación de la suma como para la modificación del operando B en la operación de resta.

```python
# XOR construido con compuertas basicas
def XOR(a, b):
    return OR(AND(a, NOT(b)), AND(NOT(a), b))
```

Figura 15(construcción del XOR con las compuertas basicas)

3.3 Implementación del sumador completo de 1 bit

El sumador completo de 1 bit se implementó siguiendo el mismo enfoque del circuito físico. Primero se genera una señal intermedia **X**, que corresponde a la suma parcial entre las entradas A y B. Posteriormente, esta señal se combina con el carry de entrada **CI** para obtener el bit de suma final **S**.

El carry de salida **COUT** se genera cuando al menos dos de las tres entradas (A, B y CI) se encuentran en nivel lógico alto, replicando el comportamiento clásico de un sumador completo.

Este bloque constituye la unidad fundamental del sistema y es reutilizado múltiples veces en el diseño del sumador–restador de varios bits.

```python
# SUMADOR COMPLETO DE 1 BIT
def full_adder(a, b, ci):
    x = XOR(a, b)
    s = XOR(x, ci)

    c1 = AND(a, b)
    c2 = AND(ci, x)
    cout = OR(c1, c2)

    return s, cout
```

Figura 16(Sumador completo de 1 bit)

3.4 Implementación del modo suma–resta mediante la señal M

Para permitir que el circuito realice tanto suma como resta, se incorporó una señal de control **M**, la cual cumple una doble función dentro del sistema:

- Cuando **M = 0**, el circuito opera como un sumador binario convencional.
- Cuando **M = 1**, el circuito realiza la resta binaria mediante el método de complemento a dos.

En la implementación, la señal M se utiliza para modificar el operando B mediante una operación XOR, obteniendo así el valor **B' = B ⊕ M**. Además, M se conecta directamente como el carry de entrada inicial del primer sumador completo, lo que permite sumar la unidad necesaria para completar el complemento a dos durante la resta.

Este enfoque permite reutilizar el mismo hardware (y código) para ambas operaciones sin duplicar bloques funcionales.

```python
# SUMADOR-RESTADOR DE 4 BITS
def add_sub_4bits(A, B, M):
    S = [0, 0, 0, 0]
    ci = M  # carry inicial = M (clave para la resta)

    for i in range(4):
        b_mod = XOR(B[i], M)  # B' = B XOR M
        S[i], ci = full_adder(A[i], b_mod, ci)

    return S, ci
```

Figura 17(Sumador-Restador completo de 1 bit)

3.5 Implementación del sumador–restador de 4 bits

El sistema completo se construyó conectando cuatro sumadores completos de 1 bit en cascada, formando un sumador–restador de 4 bits. El carry de salida de cada etapa se conecta al carry de entrada de la siguiente, garantizando la correcta propagación del acarreo entre bits.

El carry de entrada del primer sumador se define como **CI = M**, mientras que los siguientes carries dependen del resultado del bit anterior. Este tipo de arquitectura corresponde a un **Ripple Carry Adder**, ampliamente utilizado en sistemas digitales por su simplicidad y claridad de funcionamiento.

```python
# SUMADOR–RESTADOR DE 4 BITS
def add_sub_4bits(A, B, M):
    S = [0, 0, 0, 0]
    ci = M  # carry inicial = M (clave para la resta)

    for i in range(4):
        b_mod = XOR(B[i], M)  # B' = B XOR M
        S[i], ci = full_adder(A[i], b_mod, ci)

    return S, ci
```

Figura 18(Sumador-Restador completo de 1 bit)

3.6 Relación entre el diseño físico y la implementación en software

La implementación en Python reproduce fielmente el comportamiento del circuito diseñado en Proteus. Cada bloque lógico del circuito tiene su equivalente directo en el código, lo que permite validar el diseño tanto a nivel visual como funcional.

Esta correspondencia entre hardware y software facilita la verificación del sistema y refuerza la comprensión del funcionamiento interno del sumador–restador binario.

4. Pruebas y resultados

Las pruebas del circuito se realizaron mediante simulación en Proteus, verificando el correcto funcionamiento tanto en modo suma como en modo resta. Para evitar redundancias, se seleccionaron casos representativos que validan el comportamiento general del sistema.

4.1 Pruebas de suma (M = 0)

Con la señal de control M en nivel lógico bajo, el circuito opera como un sumador binario convencional.

Caso de prueba:

- A = 0101
- B = 0011
- M = 0

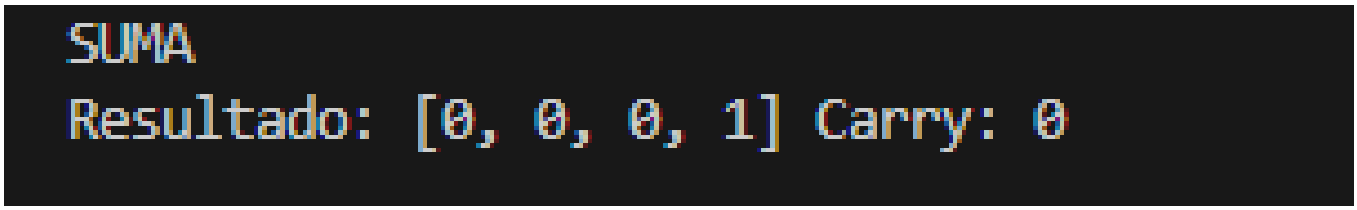

Figura 19(Resultado Suma)

Resultado esperado:

- S = 1000

La simulación arrojó el resultado correcto, confirmando que el circuito realiza adecuadamente la suma binaria y la propagación del acarreo entre etapas.

4.2 Pruebas de resta (M = 1)

Al establecer M = 1, el circuito invierte el operando B y añade un carry inicial, implementando la resta mediante complemento a dos.

Caso de prueba:

- A = 0101
- B = 0011
- M = 1

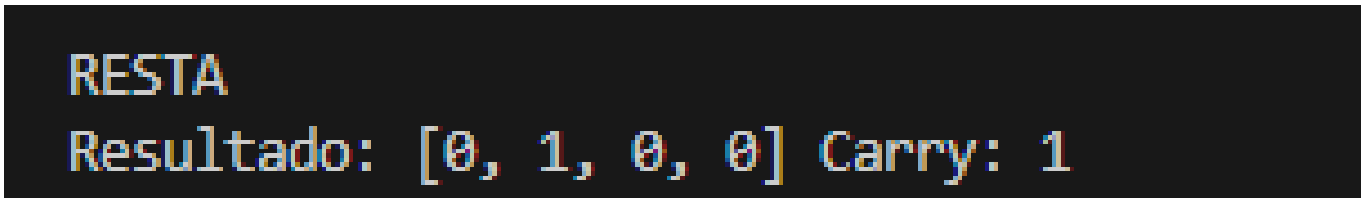

Figura 20(Resultado Resta)

Resultado esperado:

- S = 0010

La simulación confirmó que el circuito realiza correctamente la operación A − B, validando el diseño del bloque de control y la correcta conexión del carry inicial.

5.3 Observaciones

Durante el proceso de pruebas se identificaron errores iniciales relacionados con la conexión de la señal CI y la lógica de control M. Estas fallas fueron corregidas ajustando la conexión directa de M al carry de entrada del primer sumador, lo que permitió obtener resultados consistentes en todas las pruebas realizadas.