Laboratorio 2: Lógica Combinacional y Aritmética

Calderón. B Válerin
Tecnológico de Costa Rica
Estudiante de Ingeniería en Computadores
Email: 2020033313@estudiantec.cr

Luna.A Mauricio
Tecnológico de Costa Rica
Estudiante de Ingeniería en Computadores
Email: mluna@estudiantec.cr

Marin. M Kendall
Tecnológico de Costa Rica
Estudiante de Ingeniería en Computadores
Email: kendallmarin@estudiantec.cr

RESUMEN

Este informe presenta el desarrollo del Laboratorio 2: Lógica combinacional y aritmética, el cual aborda el diseño e implementación de una Unidad Lógico-Aritmética (ALU) basada en lógica combinacional, utilizando un lenguaje de descripción de hardware. La ALU es un componente clave en los procesadores, encargado de ejecutar operaciones aritméticas y lógicas. Se exploran conceptos fundamentales como los tiempos de propagación y contaminación, la ruta crítica y su influencia en la frecuencia máxima de operación. Además, se desarrollan ejercicios prácticos para diseñar, verificar e implementar una ALU parametrizable en una FPGA, enfatizando la optimización del rendimiento y el uso eficiente de los recursos en circuitos digitales.

PALABRAS CLAVE

Lógica combinacional, Unidad Lógico-Aritmética (ALU), Lenguaje de descripción de hardware, Tiempo de propagación, Tiempo contaminación, Ruta crítica, FPGA.

I. INTRODUCCIÓN

En los sistemas digitales modernos, la lógica combinacional juega un papel muy importante en el procesamiento de datos, ya que permite la implementación de circuitos cuya salida depende exclusivamente de las combinaciones de entrada en un instante dado, sin necesidad de sincronización con señales de reloj [1]. Esta característica es crucial en dispositivos de alto rendimiento, como los microprocesadores, donde la velocidad de ejecución de las operaciones aritméticas y lógicas es determinante para su eficiencia global [4].

Uno de los componentes fundamentales en los procesadores es la Unidad Lógico-Aritmética (ALU), encargada de realizar las operaciones matemáticas y lógicas esenciales para la ejecución de programas. La ALU debe ser capaz de operar con alta velocidad y eficiencia, minimizando los retardos de propagación y asegurando una adecuada administración de los recursos del hardware [2]. Su diseño e implementación dependen del uso adecuado de circuitos combinacionales, los cuales permiten realizar operaciones básicas como sumas, restas, desplazamientos y operaciones lógicas sin la intervención de señales de control temporizadas [6].

Además del diseño funcional de la ALU, es fundamental comprender los aspectos relacionados con los tiempos de propagación y contaminación, que influyen en el rendimiento del sistema. Estos tiempos determinan la rapidez con la que una señal puede propagarse a través del circuito y afectar su salida, lo que impacta directamente en la ruta crítica, es decir, el camino más largo que una señal debe recorrer en un circuito digital. La optimización de la ruta crítica es clave para aumentar la frecuencia de operación del sistema y mejorar su desempeño global [5].

En este laboratorio, se llevará a cabo el diseño e implementación de una ALU parametrizable de N bits, utilizando un lenguaje de descripción de hardware. Se abordarán los principios de diseño de circuitos combinacionales y se explorarán aspectos relacionados con los tiempos de propagación y la ruta crítica en sistemas digitales. Adicionalmente, se realizarán simulaciones y pruebas en una FPGA para validar el correcto funcionamiento de la ALU y evaluar su desempeño en diferentes configuraciones.

II. Investigación

Para este laboratorio se van a contestar una serie de preguntas.

A. Investigue sobre el funcionamiento general de una ALU. Muestra tablas de verdad y diagramas de circuitos lógicos y aritméticos simples (sumas, restas, operaciones lógicas, etc.). Incluya una descripción de las banderas de estado de una ALU, por ejemplo las de la arquitectura ARMv4..

La ALU recibe operandos desde registros o memoria y aplica operaciones definidas por el conjunto de instrucciones del procesador. Sus principales funciones incluyen:

- Operaciones aritméticas: suma, resta, incremento, decremento.
- Operaciones lógicas: AND, OR, XOR, NOT.
- Comparaciones: igualdad, menor que, mayor que.
- Manipulación de bits: desplazamientos y rotaciones.

1) Tablas de Verdad y Diagramas de Circuitos: Las operaciones lógicas más comunes en una ALU pueden representarse mediante tablas de verdad: Suma

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

Table I
TABLA DE VERDAD PARA LA OPERACIÓN SUMA

Resta

A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

Table II
TABLA DE VERDAD PARA LA OPERACIÓN RESTA

Multiplicación

A	B	A $\times B$
0	0	0
0	1	0
1	0	0
1	1	1

Table III
TABLA DE VERDAD PARA LA OPERACIÓN MULTIPLICACIÓN

División

A	B	A $\div B$
0	0	No definido
0	1	0
1	0	No definido
1	1	1

Table IV
TABLA DE VERDAD PARA LA OPERACIÓN DIVISIÓN

AND

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

Table V
TABLA DE VERDAD PARA LA OPERACIÓN AND

OR

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

Table VI
TABLA DE VERDAD PARA LA OPERACIÓN OR

XOR

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

Table VII
TABLA DE VERDAD PARA LA OPERACIÓN XOR

NOT

A	NOT
0	1
1	0

Table VIII
TABLA DE VERDAD PARA LA OPERACIÓN NOT

Operaciones Aritméticas en la ALU
Sumador Completo (Full Adder)

A	B	Cin	S	Cout
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

Table IX
TABLA DE VERDAD PARA UN SUMADOR COMPLETO

Restador Completo (Full Subtractor)

A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table X
TABLA DE VERDAD PARA UN RESTADOR COMPLETO

2) Banderas de Estado en ARMv4: La arquitectura ARMv4 utiliza un conjunto de banderas en el registro de estado para indicar el resultado de las operaciones de la ALU. Las banderas principales son:

- **N (Negative)**: Se establece en 1 si el resultado de la operación es negativo.
- **Z (Zero)**: Se establece en 1 si el resultado de la operación es cero.
- **C (Carry)**: Indica la presencia de un acarreo en operaciones aritméticas.
- **V (Overflow)**: Se activa cuando ocurre un desbordamiento aritmético.

Estas banderas son utilizadas para la toma de decisiones en la ejecución de instrucciones condicionales.

B. Explique los conceptos de tiempos de propagación y tiempos de contaminación, en circuitos combinacionales.

El tiempo de propagación es el tiempo máximo que tarda una salida en alcanzar su valor final después de que una entrada cambia. Este tiempo depende de la ruta crítica del circuito, que es el camino más largo que una señal debe recorrer desde una entrada hasta una salida. A medida que la señal atraviesa diferentes compuertas lógicas, cada una introduce un pequeño retardo, y la suma de estos retardos a lo largo de la ruta crítica determina el valor de tpd[2].Por ejemplo, en un circuito con varias compuertas AND y OR, el tiempo de propagación sería la suma de los retardos introducidos por cada una de ellas.

Por otro lado, el tiempo de contaminación es el tiempo mínimo desde que una entrada cambia hasta que la salida comienza a modificarse. Este tiempo está asociado con la ruta más corta del circuito, es decir, el camino más rápido que la señal puede tomar desde una entrada hasta una salida. La salida puede empezar a cambiar después de tcd aunque aún no haya alcanzado su valor final. Este tiempo es importante para evitar que se produzcan cambios no deseados en la salida antes de que el circuito se estabilice.[2]

Estos tiempos pueden variar debido a varios factores, como las diferencias en los retardos al cambiar de bajo a alto y de alto a bajo, las múltiples rutas que una señal puede tomar en circuitos con varias entradas y salidas, y el impacto de la temperatura en la velocidad de los circuitos (que tienden a ralentizarse cuando se calientan y acelerarse cuando están fríos). En términos prácticos, tpd y tcd pueden calcularse sumando los tiempos de retardo de cada elemento en la ruta correspondiente.

C. Investigue sobre la ruta crítica y cómo esta afecta en el diseño de sistemas digitales más complejos, por ejemplo un procesador con pipeline. Investigue su relación con la frecuencia máxima de operación de un circuito

La ruta crítica en sistemas digitales representa el camino más largo de retardo entre elementos secuenciales, lo que determina el tiempo mínimo de ciclo de reloj [7].. En diseños más complejos, como procesadores con pipeline, la ruta crítica influye directamente en el rendimiento, ya que un retardo mayor en esta ruta puede limitar la velocidad a la que se pueden completar las instrucciones.

En procesadores con pipeline, la segmentación de la ejecución de instrucciones busca mejorar el rendimiento al permitir que múltiples instrucciones se procesen simultáneamente. Sin embargo, la ruta crítica impone restricciones en la frecuencia de operación, ya que cualquier retardo en los caminos combinacionales entre registros puede requerir una reducción en la frecuencia de reloj para garantizar un funcionamiento correcto [3].

La relación entre la ruta crítica y la frecuencia máxima de operación de un circuito radica en el hecho de que la frecuencia está inversamente relacionada con el retardo de propagación en la ruta crítica. Cuanto mayor sea el retardo acumulado en la ruta crítica, menor será la frecuencia máxima a la que puede operar el circuito sin incurrir en errores de sincronización [7].. En consecuencia, optimizar la ruta crítica mediante técnicas como la reducción de carga capacitiva, la optimización de puertas lógicas y el uso de arquitecturas eficientes es fundamental para aumentar el rendimiento de los sistemas digitales avanzados.

III. Desarrollo

En esta sección se detallan los ejercicios realizados durante el laboratorio, los cuales involucraron el diseño e implementación de circuitos digitales utilizando lenguajes de descripción de hardware (HDL), específicamente SystemVerilog. Cada ejercicio abordó distintos aspectos del diseño digital, desde la creación de una ALU parametrizable hasta la evaluación del impacto del tamaño de la lógica combinacional en la frecuencia de operación del sistema.

A continuación, se presenta una descripción detallada de cada uno de los ejercicios realizados, incluyendo el proceso de diseño, implementación en hardware y los resultados obtenidos.

A. Problema 1.

Se deberá realizar una calculadora parametrizable, con el fin de que pueda ejecutar las operaciones de suma, resta, multiplicación, división, módulo, and, or, xor, shift left y shift right. Se debe de implementar las banderas de estados de la ALU: Negativo (N), Cero (Z), Acarreo (C)y Desbordamiento (V).

1) Diseñe la ALU con un modelo de estructura, utilizando SystemVerilog. Para la suma,resta y otra operación arimética elegida por el grupo, parta de los circuitos básicos (no se puede utilizar el operador directamente). Las demás operaciones listadas pueden utilizar el operador del HDL. Muestre los diagramas de bloques, tablas y cualquier otro elemento para describir el diseño implementado. : La ALU ha sido diseñada con las siguientes operaciones:

- **Operaciones Aritméticas:** Suma, Resta, Multiplicación, División, Módulo.
- **Operaciones Lógicas:** AND, OR, XOR.
- **Operaciones de Desplazamiento de Bits:** Shift Left, Shift Right.

Para permitir flexibilidad, el ancho de los operandos está parametrizado por `WIDTH`. Este parámetro permite que la ALU se reutilice en diferentes contextos sin cambiar el diseño central. Para la implementación, se diseñaron módulos auxiliares para la suma, resta y multiplicación, los cuales fueron instanciados dentro de la ALU principal. Las demás operaciones se implementaron mediante asignaciones directas utilizando operadores de SystemVerilog.La siguiente tabla resume las operaciones asociadas a cada opcode:

Opcode	Operación	Descripción
4'b0000	Suma	Se realiza utilizando el módulo `Adder`.
4'b0001	Resta	Se realiza utilizando el módulo `Subtractor`.
4'b0010	Multiplicación	Se realiza utilizando el módulo `Multiplier`.
4'b0011	División	Se realiza utilizando el operador `/`, con verificación de división por cero.
4'b0100	Módulo	Se realiza utilizando el operador `%`, con verificación de división por cero.
4'b0101	AND	Se realiza utilizando el operador AND a nivel de bits: `A & B`.
4'b0110	OR	Se realiza utilizando el operador OR a nivel de bits: `A | B`.
4'b0111	XOR	Se realiza utilizando el operador XOR a nivel de bits: `A ^ B`.
4'b1000	Shift Left	Se realiza utilizando el operador de desplazamiento a la izquierda: `A << B`.
4'b1001	Shift Right	Se realiza utilizando el operador de desplazamiento a la derecha: `A >> B`.
default	N/A	Se establece el resultado a cero.

Dado que la ALU debía ser flexible en cuanto al tamaño de los operandos, se utilizó un parámetro (WIDTH) que permite definir el ancho de los datos, asegurando así su reutilización en distintos contextos sin modificar la estructura del código.

Se incluyeron además los registros de estado o "flags" (N, Z, C, V) para indicar condiciones especiales en los resultados, como signo negativo, resultado cero, acarreo y desbordamiento. La lógica para determinar estos flags se diseñó con base en las condiciones típicas de las operaciones aritméticas.

- **N (Flag Negativo):** Este flag se establece si el bit más significativo (MSB) del resultado es 1, lo que indica un resultado negativo.
- **Z (Flag Cero):** Este flag se establece si el resultado es igual a cero.
- **C (Flag de Acarreo):** Este flag depende de la operación:
  - Para la suma, se establece con `addResult[WIDTH]` (acarreo de la suma).
  - Para la resta, se establece con `subResult[WIDTH]` (acarreo de la resta).
  - Para el desplazamiento a la izquierda, se establece con `A[WIDTH-1]` (acarreo del desplazamiento a la izquierda).
  - Para el desplazamiento a la derecha, se establece con `A[0]` (acarreo del desplazamiento a la derecha).
- **V (Flag de Desbordamiento):** Este flag se establece en los siguientes casos:
  - En la suma, si los operandos tienen el mismo signo y el resultado tiene un signo diferente.
  - En la resta, si los operandos tienen signos opuestos y el resultado tiene un signo diferente del primer operando.
  - En la multiplicación, si la parte superior de `mulResult` es diferente de cero.
  - En la división, si ocurre una división por cero.

2) Realice un testbech de auto-chequeo, en SystemVerilog, en el que se muestre de manera simple el funcionamiento de la Calculadora en 4 bits. Muestre el resultado de la prueba para al menos 2 valores diferentes para cada operación.: Se realizó un testbench para comprobar el funcionamiento de la ALU de 4 bits, utilizando varias combinaciones de entradas para cada operación aritmética y lógica. En este testbench, se inicializan las entradas A y B, y se cambian en diferentes combinaciones. También se monitorean las salidas result, N, Z, C, y V, que representan el resultado de la operación y las banderas correspondientes a las condiciones de la ALU. A continuación, se detallan las operaciones probadas:

- **Suma (opcode 0000):** Se probaron dos combinaciones de las entradas A y B con el opcode de suma. Se verificó si el resultado y las banderas (C, Z, N, V) eran correctos para cada operación.
- **Resta (opcode 0001):** Se probaron dos combinaciones de las entradas A y B con el opcode de resta. Se verificó el comportamiento esperado de la ALU con respecto a la operación de resta y las banderas.
- **Multiplicación (opcode 0010):** Se probaron dos combinaciones para la multiplicación y se verificó si el resultado y las banderas se comportaban correctamente.
- **División (opcode 0011):** Se probaron dos combinaciones para la división y se verificó si el resultado de la operación era correcto, especialmente en el caso de divisores distintos de cero.
- **Módulo (opcode 0100):** Se probaron dos combinaciones de las entradas para la operación de módulo y se verificaron los resultados obtenidos.
- **Operaciones Lógicas (AND, OR, XOR):** Se probaron combinaciones de entradas con los opcodes correspondientes (0101 para AND, 0110 para OR, 0111 para XOR) y se verificaron los resultados de cada operación.
- **Shift Left (opcode 1000) y Shift Right (opcode 1001):** Se probaron combinaciones de entradas con los opcodes correspondientes para los desplazamientos de bits a la izquierda y a la derecha, y se monitorearon los resultados y las banderas.

El testbench muestra los resultados de las operaciones en la consola para cada combinación de

entradas. Cada operación se ejecuta durante 10 unidades de tiempo, y los resultados, junto con las banderas de la ALU, se muestran para asegurar que el comportamiento es el esperado. Una vez completada la simulación, se observó el comportamiento correcto de la ALU para las diversas operaciones y las banderas asociadas. Se utilizó ModelSim para ejecutar la simulación y verificar los resultados de cada operación.

3) Implemente la calculadora con parámetro de 4 bits, en la FPGA (configuración). Utilice los switches y botones como entradas y el display de 7 segmentos para visualizar el funcionamiento de la misma. Cuando se configuren los operandos, se deberá accionar un mecanismo para que ejecute la operación deseada. La manera en que se elija la operación será creatividad de cada grupo.: Para la calculadora, la lógica de funcionamiento se basa en varios componentes clave que interactúan entre sí para realizar operaciones aritméticas, lógicas y de desplazamiento.

Primero, se asignan los switches para ingresar los operandos. Los switches $SW[3:0]$ y $SW[7:4]$ se asignan a los operandos A y B, respectivamente. Esto permite que se ingresen dos números de 4 bits utilizando los switches de la FPGA. Luego, los botones $KEY[0:3]$ se utilizan para seleccionar la operación que se desea realizar. Dependiendo de los valores de los switches $SW[8]$ y $SW[9]$, se determina qué tipo de operación se llevará a cabo, ya sea una operación aritmética (como suma, resta, multiplicación o división), lógica (como módulo, AND, OR o XOR), o de desplazamiento (shift left o shift right).

La unidad encargada de realizar las operaciones es la ALU, esta recibe los operandos y el código de operación determinado previamente. La ALU ejecuta la operación seleccionada y genera un resultado, junto con varios flags que indican el estado de la operación. Estos flags se asignan a los LEDs de la FPGA para dar una retroalimentación visual sobre el estado de la operación realizada.

Finalmente, el resultado de la operación se muestra en los displays de 7 segmentos. El valor binario del resultado se convierte en un formato que puede ser visualizado en los displays de 7 segmentos para mostrar el número correspondiente. Esta conversión es realizada por un módulo que mapea los valores binarios de 4 bits a los patrones de segmentos apropiados en el display.

B. Problema 2.

Un asunto importante, en sistemas combinacionales complejos, es la ruta cítica en tiempos de propagación, dato que finalmente determina la frecuencia de operación de un sistema digital.Un circuito simple para la medición de esta frecuencia se muestra en la figura 2, en la que se utilizan dos registros de carga paralela entre la l´ogica combinacional, para determinar la frecuencia máxima a la que puede operar un reloj. Para el circuito anterior y utilizando el diseño de la ALU

1) Diseñe el diagrama de la figura 2 con la ALU para 2, 4, 8, 16 y 32 bits. Determine en cada caso el uso de los recursos de la FGPA en términos de celdas básicas y otras métricas que consideren importantes. Utilice la herramienta TimeQuest de Altera para determinar la frecuencia máxima de operación para cada ALU (distintos tamaños). Realice una tabla y gráfico con estos datos donde se muestre el comportamiento. Discuta sobre el efecto de aumentar el tamaño de la lógica combinacional en la frecuencia de operación del sistema: Para este problema se utilizó la Alu creada para el problema anterior, la cual fue ideada con el fin de que fuera flexible y eficiente y así pudiera adaptarse a diferentes tamaños de datos (2, 4, 8, 16 y 32 bits). Para lograr esto, se utilizó un parámetro WIDTH que permitiera configurar el tamaño de los operandos y resultados, asegurando que el diseño fuera reutilizable para distintos escenarios de procesamiento.

En primer lugar, se definió el módulo `alu_pipeline` con los parámetros esenciales como `clk`, `reset`, `opcode`, `A`, `B`, `result_out`, `N`, `Z`, `C`, y `V`. El parámetro WIDTH es clave, ya que determina el tamaño de los operandos y resultados.

De esta forma, la ALU es adaptable y puede ser escalada según las necesidades de la aplicación.

Para manejar las entradas de la ALU, se implementó un bloque `always_ff` que actualiza los registros `A_reg` y `B_reg` en cada flanco positivo del reloj o cuando se activa el reset. Esto asegura la sincronización correcta de las entradas con el ciclo de reloj, lo cual es fundamental para mantener la integridad de los datos en un diseño basado en pipeline.

La operación de la ALU se realiza mediante un bloque combinacional donde se utilizan las entradas sincronizadas `A_reg` y `B_reg`, junto con un código de operación `opcode`. En el diseño actual, la ALU está configurada para realizar solo una operación de suma, indicada por el `opcode` `4'b0000`, aunque el código está diseñado para ser fácilmente ampliable a otras operaciones con la modificación del valor de opcode. La ALU genera el resultado de la operación junto con los flags correspondientes.

Para la salida, se empleó otro bloque `always_ff` que permite almacenar los resultados de la ALU en el registro `result_reg`, sincronizándose con el flanco negativo del reloj. Esto asegura que los resultados se mantengan estables hasta el siguiente ciclo de reloj, lo que es crucial para la correcta propagación de datos en un sistema basado en pipeline.

Con este diseño, la ALU es capaz de manejar diferentes tamaños de datos de manera flexible. Al incrementar el valor del parámetro, el código se adapta automáticamente para manejar operandos más grandes sin la necesidad de realizar cambios estructurales importantes. Esto optimiza el uso de recursos y permite la expansión del sistema para aplicaciones más complejas.

En cuanto al análisis de rendimiento, el incremento en el tamaño de la ALU (de 2 a 32 bits) afecta directamente el uso de recursos de la FPGA, como las LUTs y flip-flops. A medida que el tamaño de los datos crece, también lo hace la complejidad de la lógica combinacional, lo que incrementa el número de LUTs y flip-flops necesarios. Esto, a su vez, puede impactar en la frecuencia máxima de operación, ya que las señales deben recorrer rutas más largas, lo que aumenta la latencia y reduce la velocidad máxima alcanzable por la ALU.

Finalmente, el análisis de la frecuencia máxima de operación y el uso de recursos se realizó utilizando la herramienta TimeQuest de Altera. Esta herramienta permitió evaluar el impacto del tamaño de la ALU en la frecuencia máxima, proporcionando una visión clara de cómo la complejidad del diseño afecta el rendimiento. Con los datos obtenidos, se pudieron generar tablas y gráficos que ilustran la relación entre el tamaño de la ALU, el uso de recursos y la frecuencia máxima de operación. Esto facilitó la comprensión de cómo el aumento en el tamaño de los datos influye negativamente en la frecuencia máxima, debido a la mayor complejidad de la lógica combinacional y las rutas críticas más largas.

IV. Resultados

A. *Problema 1.*

Durante la simulación del testbench para la ALU, los resultados obtenidos fueron consistentes con lo esperado para cada una de las operaciones realizadas. Se probaron diversas combinaciones de entradas, cubriendo las operaciones aritméticas y lógicas principales. En general, los resultados de las salidas, junto con las banderas de estado, indicaron que la ALU se comporta correctamente y realiza las operaciones de acuerdo con las expectativas.

```
VSIM 15> run -all
# Test de ALU - Auto-chequeo
# SUMA: A = 0011, B = 0001, Resultado = 0100, C = 0, Z = 0, N = 0, V = 0
# SUMA: A = 1111, B = 0001, Resultado = 0000, C = 1, Z = 1, N = 0, V = 0
# RESTA: A = 1001, B = 0010, Resultado = 0111, C = 1, Z = 0, N = 0, V = 1
# RESTA: A = 0010, B = 0010, Resultado = 0000, C = 1, Z = 1, N = 0, V = 0
# RESTA NEGATIVA: A = 0010, B = 1010, Resultado = 1000, C = 0, Z = 0, N = 1, V = 1
# MULTIPLICACION: A = 0010, B = 0011, Resultado = 0110, C = 0, Z = 0, N = 0, V = 0
# DIVISION: A = 1000, B = 0010, Resultado = 0100, C = 0, Z = 0, N = 0, V = 0
# AND: A = 1100, B = 1010, Resultado = 1000
# OR: A = 1100, B = 1010, Resultado = 1110
# XOR: A = 1100, B = 1010, Resultado = 0110
# SHIFT LEFT: A = 0011, B = 0001, Resultado = 0110, C = 0, Z = 0, N = 0
# SHIFT RIGHT: A = 1000, B = 0001, Resultado = 0100, C = 0, Z = 0, N = 0
# SHIFT LEFT (C=1): A = 1001, B = 0001, Resultado = 0010, C = 1, Z = 0, N = 0
# SHIFT RIGHT (C=1): A = 1001, B = 0001, Resultado = 0100, C = 1, Z = 0, N = 0
# Test completado.
# ** Note: $finish    : ALU_tb.sv(83)
#    Time: 140 ns  Iteration: 0  Instance: /ALU_tb
```

Figure 1. Simulación del testbench en ModelSim.

Tambien se realizan pruebas en la FPGA, para corroborar que todas las operaciones esten funcionando correctamente, a continuación se muestran los resultados de las puebas realizadas.

1) Suma: Para probar los resultados, realiza la suma de 1+1 en el FPGA. La cual se espera que tenga como resultado 10.

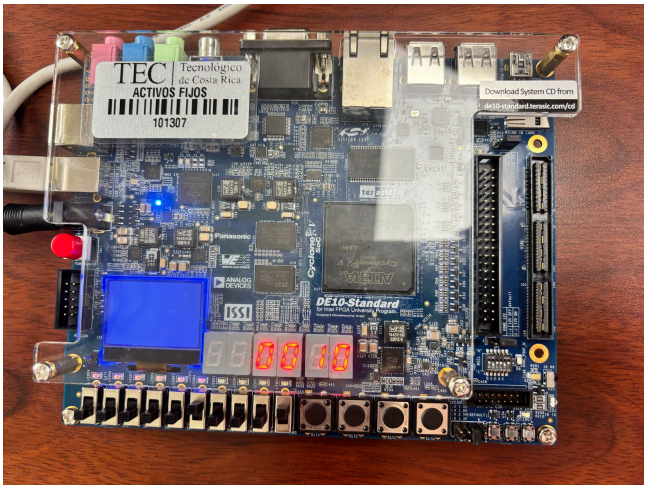

Figure 2. Suma en FPGA.

2) Resta: Para probar los resultados, realiza la resta de 11-1 en el FPGA. La cual se espera que tenga como resultado 10.

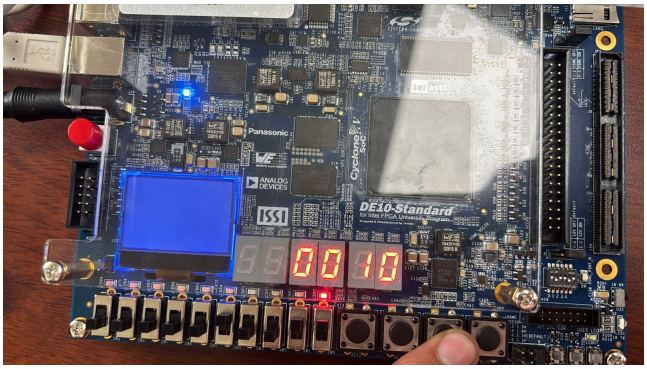

Figure 3. Resta en FPGA.

3) Multiplicación: Para probar los resultados, realiza la multiplicación de 11*1 en el FPGA. La cual se espera que tenga como resultado 11.

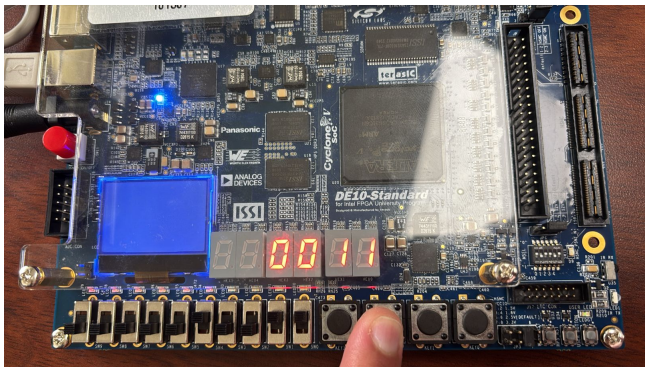

Figure 4. Multiplicación en FPGA.

4) División: Para probar los resultados, realiza la división de 11÷1 en el FPGA. La cual se espera que tenga como resultado 11.

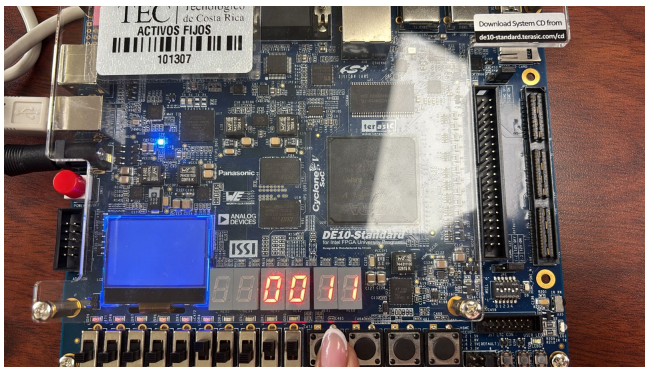

Figure 5. División en FPGA.

5) AND: Para probar los resultados, realiza la operación AND de 11&1 en el FPGA. La cual se espera que tenga como resultado 01

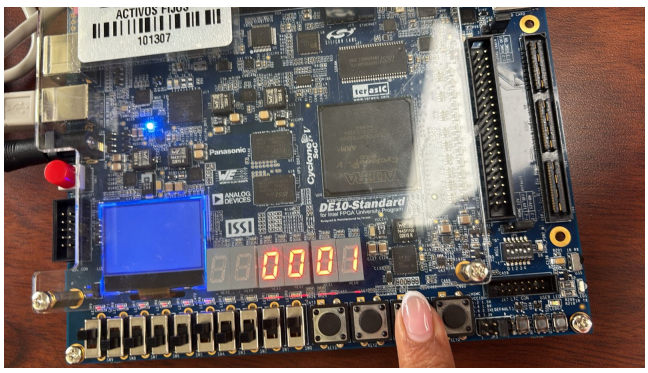

Figure 6. Operación AND en FPGA.

6) OR: Para probar los resultados, realiza la operación OR de 11 | 1 en el FPGA. La cual se espera que tenga como resultado 11.

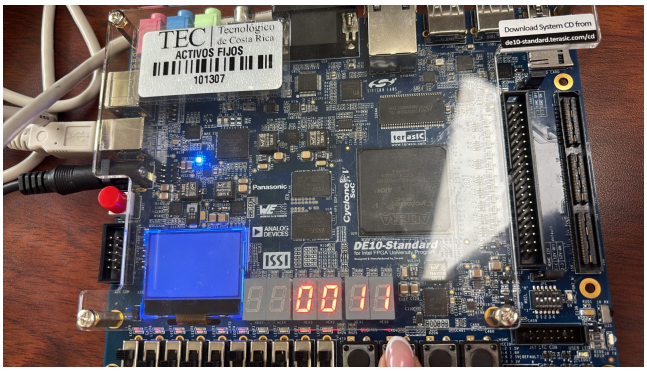

Figure 7. Operación OR en FPGA.

7) XOR: Para probar los resultados, realiza la operación XOR de $11 \oplus 1$ en el FPGA. La cual se espera que tenga como resultado 10.

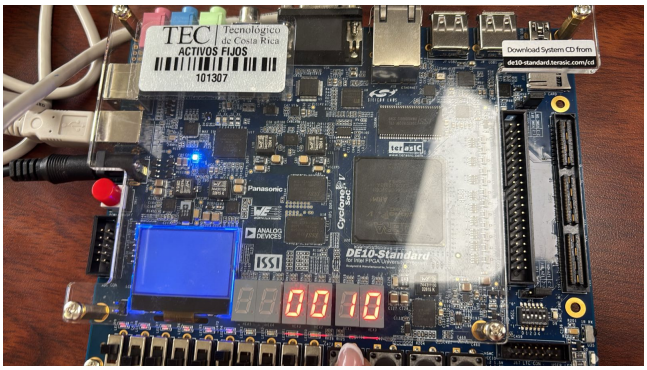

Figure 8. Operación XOR en FPGA.

8) Shift Left: Para probar los resultados, realiza la operación Shift Left de $11 << 1$ en el FPGA. La cual se espera que tenga como resultado 110.

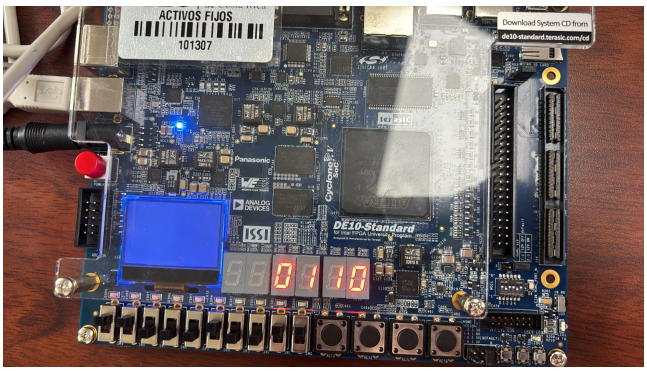

Figure 9. Shift Left en FPGA.

9) Shift Right: Para probar los resultados, realiza la operación Shift Right de $11 >> 1$ en el FPGA. La cual se espera que tenga como resultado 001.

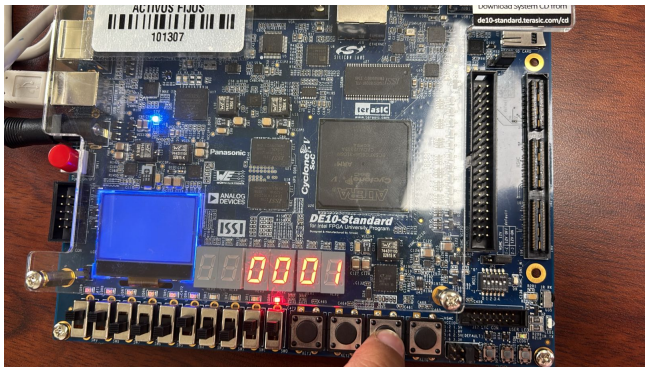

Figure 10. Shift Right en FPGA.

B. Problema 2.

Se presentan los resultados derivados del análisis del comportamiento de la ALU utilizando diferentes tamaños de operando (2, 4, 8, 16 y 32 bits), con el objetivo de evaluar la frecuencia máxima de operación de cada configuración. Para ello, se empleó la herramienta TimeQuest de Altera, que permitió calcular los tiempos de propagación y las frecuencias máximas de operación correspondientes a cada caso. Además, se incluyen los datos sobre el uso de recursos de la FPGA, especificando la cantidad de celdas básicas utilizadas, junto con otras métricas clave relacionadas con el rendimiento del sistema.

Tamaño (bits)	Tiempo de propagación (ns)	Frecuencia Máxima (MHz)
2	4.970 ns	201.2 MHz
4	5.564 ns	179.7 MHz
8	6.142 ns	162.8 MHz
16	7.343 ns	136.2 MHz
32	9.013 ns	110.9 MHz

Figure 11. Tiempo de propagación y frecuencia máxima.

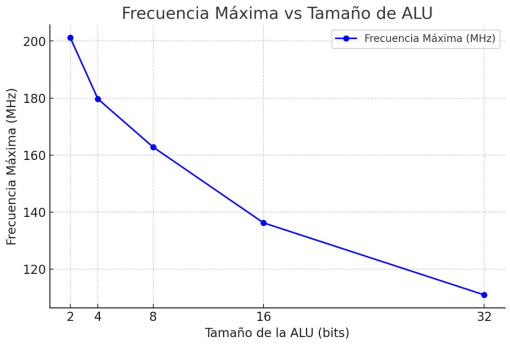

Figure 12. Gráfica de frecuencia.

En la imagen anterior se puede observar la frecuencia máxima en función del tamaño de la ALU, la cual está mostrando una tendencia decreciente

Tamaño (bits)	ALMs Utilizados	Registros	Pines	Frecuencia Máxima (MHz)
2	3 ALMs	6	16	201.2 MHz
4	6 ALMs	12	22	179.7 MHz
8	13 ALMs	24	34	162.8 MHz
16	25 ALMs	48	58	136.2 MHz
32	51 ALMs	96	106	110.9 MHz

Figure 13. Recursos utilizados.

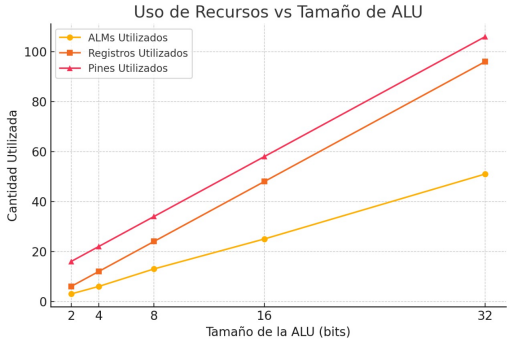

Figure 14. Gráfica de los recursos utilizados.

V. Análisis

A. Problema 1

Analizando los resultados obtenidos durante las simulaciones y las pruebas en la FPGA se demuestra que la ALU está funcionando correctamente y conforme a lo esperado. Los resultados obtenidos en la simulación en ModelSim y en las pruebas realizadas en la FPGA coinciden con los resultados teóricos establecidos para cada operación aritmética y lógica.

En primer lugar, las operaciones aritméticas de suma, resta, multiplicación y división fueron verificadas exitosamente tanto en simulación como en hardware. Los resultados de la suma, resta, división y multiplicación en la FPGA coincidieron con los valores esperados, lo que indica una correcta implementación del diseño. Las banderas de estado, también se comportaron adecuadamente, reflejando los resultados correctos de las operaciones.

Las operaciones lógicas AND, OR y XOR, también se comportaron de acuerdo con las expectativas. Los resultados de estas operaciones tanto en la simulación como en la FPGA fueron los correctos, y las salidas producidas reflejaron correctamente las reglas lógicas correspondientes.

Las operaciones de desplazamiento shift left y shift right fueron igualmente evaluadas en ambas plataformas, mostrando resultados consistentes con los valores esperados. Las banderas de estado también indicaron correctamente los resultados de estas operaciones, lo que confirma la correcta funcionalidad de la ALU en estos casos.

B. Problema 2

Una observación importante de los resultados es la disminución en la frecuencia máxima de operación conforme aumenta el tamaño de la ALU. Este comportamiento se puede explicar a través de los siguientes factores:

- Mayor lógica combinacional y aumento del tiempo de propagación: Esto debido a que al incrementar el número de bits en la ALU, también se incrementa la cantidad de lógica combinacional necesaria para realizar las operaciones aritméticas y lógicas. Esto significa que la ALU debe procesar más señales a través de un mayor número de puertas lógicas y registros, lo que aumenta el tiempo de propagación de la señal en el sistema. La propagación más lenta de las señales reduce la frecuencia máxima a la que el sistema puede operar. En los resultados obtenidos en la figura 3, se puede observar cómo la frecuencia máxima disminuye conforme el tamaño de la ALU aumenta.
- Ruta crítica más larga: Al esta ser el camino más largo que las señales deben seguir dentro del sistema, genera un aumento en el tiempo de propagación. Un mayor tiempo de propagación resulta en una menor frecuencia máxima posible de operación. La relación entre la longitud de la ruta crítica y la frecuencia máxima es directa, es decir, cuanto más largo es el recorrido de las señales, menor será la frecuencia en la que el sistema puede operar de manera estable. Conforme aumenta el tamaño de

la ALU, también aumenta el uso de recursos de la FPGA, lo que se refleja en un mayor número de ALMs utilizadas. Esto es esperado, ya que una ALU más grande requiere más flip-flops, celdas lógicas, y más interconexiones dentro de la FPGA. Los recursos necesarios para implementar la ALU crecen proporcionalmente con el número de bits, lo cual es evidente en los gráficos de uso de recursos

que muestran un aumento continuo en la cantidad de celdas necesarias a medida que se incrementa el tamaño de la ALU.

VI. Conclusiones

A. Problema 1.

Los resultados obtenidos durante las pruebas del testbench y las pruebas realizadas en la FPGA indican que la ALU está funcionando correctamente, tal como se esperaba. Las operaciones aritméticas como la suma, resta, multiplicación y división fueron verificadas exitosamente tanto en simulación como en hardware, mostrando que el diseño de la ALU ha sido implementado correctamente. Los resultados de estas operaciones coinciden con los valores teóricos establecidos, lo que confirma que la ALU realiza las operaciones de manera precisa y confiable. Asimismo, las banderas de estado se comportaron adecuadamente, reflejando los resultados correctos para cada operación. Las operaciones lógicas (AND, OR y XOR), así como las de desplazamiento (shift left y shift right), también mostraron los resultados esperados en ambas plataformas, garantizando que la ALU funciona de manera coherente con las reglas lógicas y aritméticas previstas.

B. Problema 2.

Se observó una disminución en la frecuencia máxima de operación a medida que aumentaba el tamaño de la ALU. Este comportamiento se explica por el incremento en la lógica combinacional y el aumento del tiempo de propagación de las señales, debido a que una ALU más grande requiere procesar más señales a través de un mayor número de puertas lógicas y registros. Este aumento en el tiempo de propagación ralentiza el sistema, reduciendo la frecuencia máxima a la que puede operar de manera estable. Además, se identificó que la ruta crítica del diseño se alarga con el incremento del tamaño de la ALU, lo que también contribuye a un mayor tiempo de propagación. Estos factores combinados resultan en una frecuencia máxima de operación más baja en las ALUs de mayor tamaño. Además, se observó que el uso de recursos de la FPGA, como ALMs, aumenta conforme el tamaño de la ALU crece. Esto es esperado, ya que una ALU de mayor tamaño requiere más flip-flops, celdas lógicas y más interconexiones, lo que incrementa el uso de recursos de la FPGA proporcionalmente al número de bits de la ALU.

En conclusión, el desarrollo de este laboratorio permitió aplicar los conceptos fundamentales de lógica combinacional y aritmética mediante el diseño e implementación de una Unidad Lógico-Aritmética (ALU). A través del uso de SystemVerilog, se logró modelar circuitos digitales estructurales y verificar su correcto funcionamiento mediante simulaciones en ModelSim, lo que permitió detectar y corregir errores antes de su implementación en FPGA. Además, la síntesis en Quartus Prime facilitó la conversión del diseño en una representación física dentro del hardware, evaluando el desempeño real del sistema. Finalmente, el análisis de tiempos de propagación y la ruta crítica evidenció la importancia de estos factores en el rendimiento de circuitos digitales complejos, resaltando cómo influyen en la frecuencia máxima de operación.

VII. Referencias

References

[1] Brown, S., & Vranesic, Z. (2019). *Fundamentals of Digital Logic with Verilog Design*. McGraw-Hill.

[2] Harris, D., & Harris, S. (2021). *Digital Design and Computer Architecture*. Morgan Kaufmann.

[3] Hennessy, J. L., & Patterson, D. A. (2017). *Computer Architecture: A Quantitative Approach (6th ed.)*. Morgan Kaufmann.

[4] Mano, M. M., & Ciletti, M. D. (2017). *Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog*. Pearson.

[5] Smith, D. J. (2020). *High-Speed Digital Design: A Handbook of Black Magic*. Prentice Hall.

[6] Wakerly, J. F. (2018). *Digital Design: Principles and Practices*. Pearson.

[7] Weste, N. H. E., & Harris, D. M. (2010). *CMOS VLSI Design: A Circuits and Systems Perspective (4th ed.)*. Addison-Wesley.