Trabajo práctico N°1: ALU

Felipe Pillichody; Sol A. Nou

Facultad de Ciencias Exactas, Físicas y Naturales

Arquitectura de computadoras

24 de Septiembre, 2025

felipe.pillichody@mi.unc.edu.ar

sol.nou@mi.unc.edu.ar

OBJETIVO

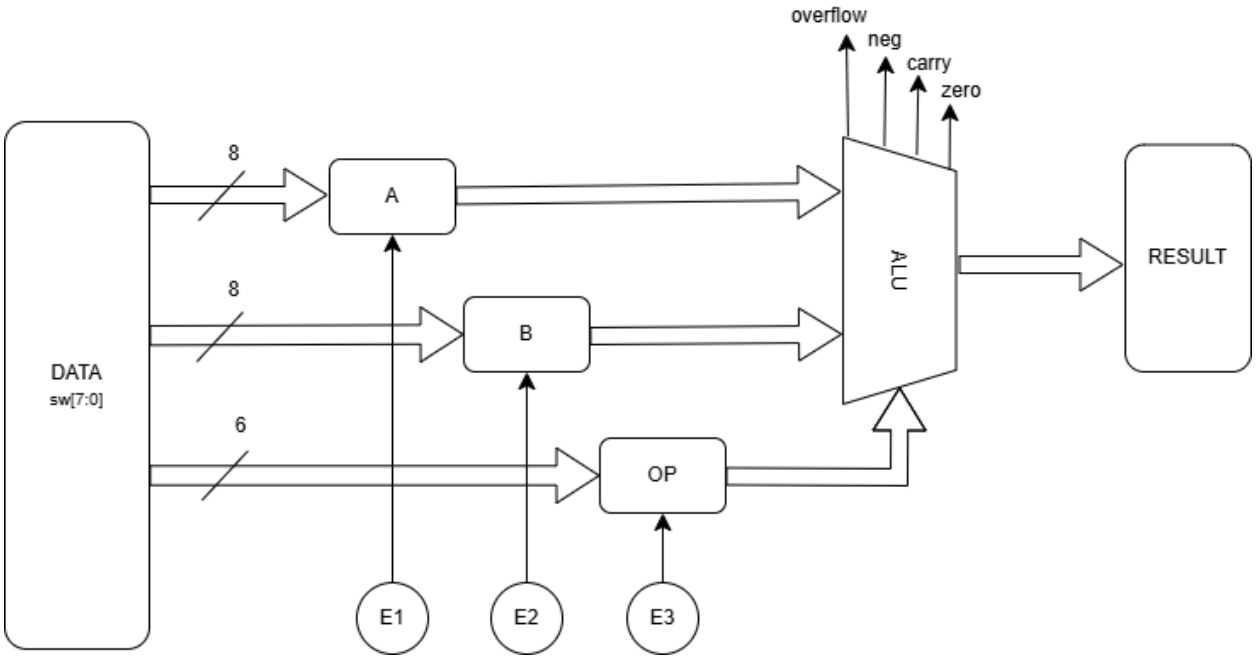
El objetivo del trabajo es implementar en FPGA una Unidad Aritmética Lógica (ALU) que sea capaz de realizar una serie de operaciones sobre dos operandos. Se busca que el tamaño del bus de datos sea parametrizable, con el objetivo de que sea escalable para próximos proyectos, permitiendo ajustar el diseño a diferentes anchos de palabra, manteniendo la estructura base de la ALU.

Además, se busca validar el diseño mediante un *testbench* que genere entradas arbitrarias y realice el chequeo automático de los resultados obtenidos. La simulación incluirá análisis de tiempos para evaluar el comportamiento del circuito en condiciones de operación, verificando tanto la funcionalidad lógica como el cumplimiento de los requerimientos temporales del diseño.

DESARROLLO

Se crea una ALU modular para lograr un diseño simple y adaptable a diferentes configuraciones de bus de datos, que permita implementar operaciones aritméticas y lógicas.

En el siguiente diagrama se muestra el diseño propuesto, donde 8 switches denominados DATA conforman la entrada principal. Mediante los botones de enable E1, E2 y E3, el valor seleccionado en DATA se almacena en el registro A (8 bits), en el registro B (8 bits) o en el registro OP (6 bits menos significativos de DATA). Los registros A y B funcionan como entradas de la ALU, mientras que OP actúa como señal de control que determina la operación a ejecutar. El resultado de la ALU se representa en 8 LEDs de salida, y adicionalmente se utilizan 4 LEDs para indicar el estado de las banderas carry, zero, overflow y negative.



Las operaciones de la ALU son las siguientes:

Operaciones	Codigo
ADD	100000
SUB	100010
AND	100100
OR	100101
XOR	100110
SRA	000011
SRL	000010
NOR	100111

Se desarrollaron tres módulos:

- **myALU** implementa la lógica funcional: contiene registros síncronos A y B (8 bits) y Op (6 bits) que se cargan en flanco de reloj cuando las señales e1, e2 o e3 están activas y se resetean con un reset que también es activo por flanco de subida. La operación se ejecuta en lógica combinacional según el Op seleccionando, mostrando en la salida de LEDS el result (8 bits) y las banderas zero, neg, carry y overflow.
- **myALU_top_basys3**, siendo la interfaz con la placa Basys3: recibe el reloj de 100 MHz, los switches y botones, instancia tres convertidores de pulsos para los botones de enable, conecta la ALU y asigna la salida (resultado y flags) a los LEDs. Los botones btnL, btnR y btnC se convierten en pulsos de un ciclo mediante las instancias de btn_onepulse y estos pulsos se usan como señales de habilitación e1, e2 y e3 para almacenar el valor presente en sw[7:0] en los registros A, B u Op dentro de la ALU. Además, se instancia btnU que actúa como reset síncrono.
- **btn_onepulse** se encarga de transformar la señal de un botón en un pulso limpio de un solo ciclo de reloj. Para lograrlo, primero sincroniza la señal del botón al reloj del sistema mediante dos registros y luego detecta el flanco de subida. De esta forma, aunque el botón permanezca presionado varios ciclos, la salida genera solo un pulso breve que puede usarse de manera segura para habilitar cargas en otros módulos.

MODULO MYALU

En el diseño se utilizan registros, para generar latch y guardar las señales en el tiempo, como los operandos A y B, la operación Op. Estos valores se cargan de manera secuencial dentro de un bloque always @(posedge clk), lo que significa que cambian únicamente en los flancos de subida del reloj o al recibir un reset. De esta forma, la ALU puede almacenar y mantener estables los datos hasta la siguiente operación. El reset hace que los valores de los operandos A y B vuelvan a cero y pone la operación ADD por defecto.

Por otro lado, se usan wires cuando no es necesario almacenar información, sino simplemente conectar señales o describir relaciones combinacionales, como en el caso de las banderas zero y neg, o la señal shamt que toma los tres bits menos significativos de B para definir el desplazamiento en las operaciones SRA, SRL.

Los parámetros permiten definir los códigos de operación que identifican qué función debe ejecutar la ALU, por ejemplo ADD, SUB, AND, OR, etc. Esto evita usar números mágicos en el código y facilita la lectura y modificación posterior.

En otras palabras, se utilizan dos bloques always: uno secuencial, dependiente del reloj, para cargar los registros A, B y Op con los datos ingresados, y otro combinacional (always @*) para describir la lógica de la ALU, donde el resultado depende únicamente de las entradas actuales. Esta separación asegura que el diseño sea modular: primero se almacenan los valores en registros sincronizados al reloj y luego se procesan las operaciones de forma combinacional.

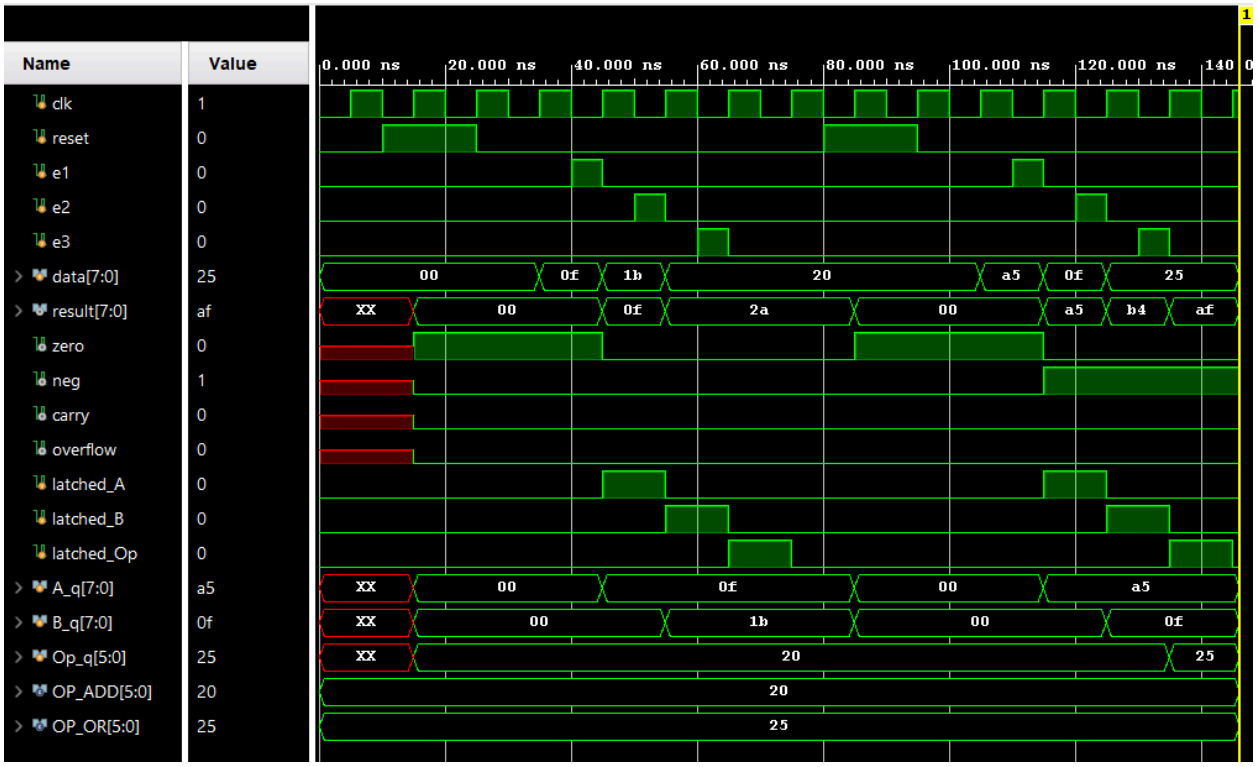
TESTBENCH

Para comprobar el correcto funcionamiento del diseño, se implementó un testbench en Verilog que permite simular el comportamiento de la ALU frente a distintas condiciones de entrada. El objetivo principal fue verificar la lógica de las operaciones implementadas y observar el comportamiento temporal del sistema. La simulación se realizó aplicando operaciones simples como ADD y OR, analizando cómo las señales de control y los registros internos responden a cada caso.

Algunos elementos y partes del test a destacar son:

- El testbench utiliza una señal de reloj de 100 MHz, para que en cada flanco ascendente se realizan todas las capturas de datos.
- La señal reset, activa en alto, reinicia el sistema y devuelve los registros internos (A, B y Op) a sus valores por defecto.

- Las señales e1, e2 y e3 son pulsos de un solo ciclo que indican cuándo deben capturarse los valores de entrada
    - e1 carga el operando A
    - e2 el operando B
    - e3 el código de operación.
- Los valores almacenados en los registros internos (A_q, B_q y Op_q) son los que realmente intervienen en el cálculo, mientras que la entrada data puede cambiar sin afectar el resultado hasta que se active un nuevo pulso de captura.
- La señal result muestra el resultado combinacional de la operación correspondiente a los valores latcheados.



Podemos ver en la imagen el comportamiento del test. Vemos como los valores de A, B y OP arrancan siendo indefinidos hasta que se aplica el reset, inicializandolos en 0 para A y B y en ADD (20) para el OP. Luego, al activar las señales de captura e1, e2 y e3, los valores de data se transfieren correctamente a los registros internos. Por ejemplo, cuando se cargan los operandos A = 0F, B = 1B y el código de operación Op = 100000 (ADD), el resultado obtenido es 2A, coincidiendo con lo esperado. Tras un nuevo reinicio, se realiza la segunda operación con A = A5, B = 0F y Op = 100101 (OR), generando un resultado de AF, también correcto según lo esperado.

Además, puede notarse que los pulsos de captura (e1, e2, e3) ocurren de manera secuencial y sincronizados con el flanco ascendente del reloj, asegurando que cada valor queda almacenado exactamente un ciclo después del pulso. En cambio, el retardo entre la carga del dato y la aparición del resultado en la señal result es prácticamente inmediato tras el flanco activo, lo que confirma que la salida es combinacional respecto a los registros latcheados. Así validamos su funcionamiento lógico como su respuesta temporal dentro de los márgenes esperados.

CONCLUSION

A lo largo del trabajo práctico se diseñó, implementó y verificó una Unidad Aritmético-Lógica (ALU) parametrizable en lenguaje Verilog, integrando conceptos fundamentales de arquitectura de procesadores y lógica secuencial. El proceso incluyó el desarrollo de un banco de pruebas capaz de estimular el módulo con diferentes operaciones y validar sus resultados, haciendo énfasis en el comportamiento sincrónico de los registros internos y en la correcta generación de salidas combinacionales.

Las simulaciones realizadas confirmaron que el circuito responde de manera adecuada ante las señales de control y que el resultado siempre refleja las operaciones ejecutadas sobre los valores efectivamente almacenados. Esto permitió comprobar el correcto funcionamiento del diseño y consolidar los conocimientos teóricos mediante su aplicación práctica.

También se implementó el diseño correctamente en la placa Basys 3.