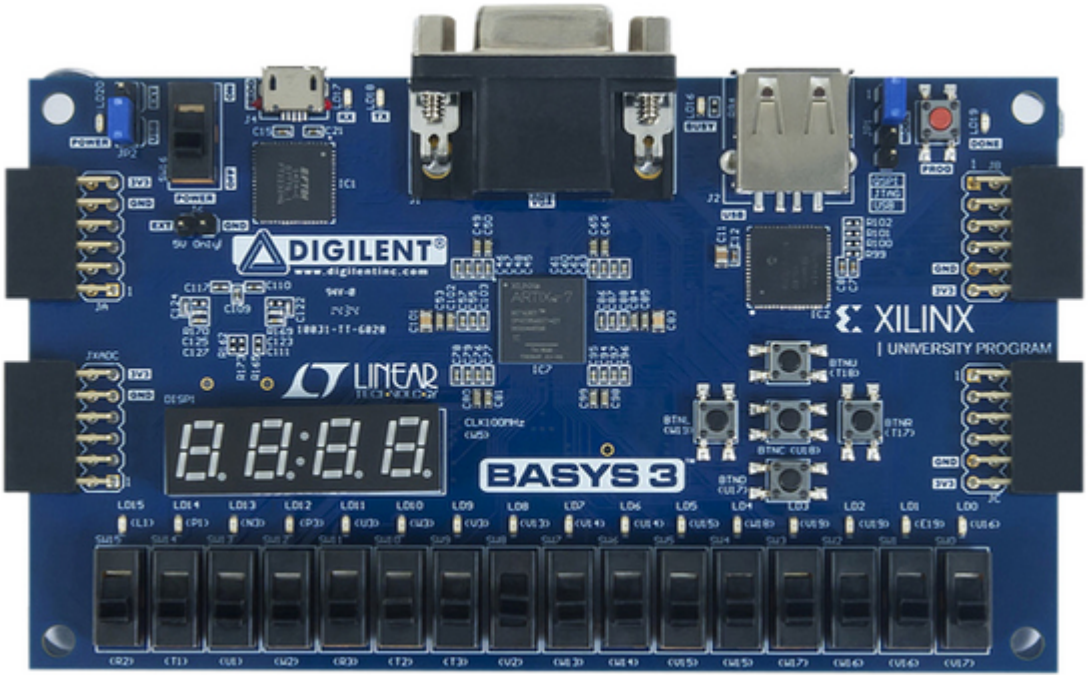
UNC Universidad Nacional de Córdoba

FCEFyN Facultad de Ciencias Exactas Físicas y Naturales



Trabajo Práctico Nº 1

ALU: Arithmetic and Logic Unit

Docentes:

- Ing. Santiago Rodriguez
- Ing. Martín Pereyra

Autores:

- Federico Joaquín Coronati
- Francisco Adrián De la Cruz

Arquitectura de Computadoras - Año 2021

Índice

Objetivo

Este trabajo práctico de laboratorio consiste en implementar una **Unidad Aritmética y Lógica** (ALU por sus siglas en inglés) para **FPGA**, parametrizable para poder ser utilizada en posteriores trabajos. Debe contar con un **Testbench** para probar el diseño y simularse en **Vivado**.

Las operaciones que debe poder realizar este módulo son las siguientes:

1) **ADD:** Adición (Código 100000)
2) **SUB:** Sustracción (Código 100010)
3) **AND** Lógico (Código 100100)
4) **OR** Lógico (Código 100101)
5) **XOR** Lógico (Código 100110)
6) **SRA:** Desplazamiento a la Derecha (Código 000011)
7) **SRL:** Desplazamiento a la Izquierda (Código 000010)
8) **NOR** Lógico (Código 100111)

El trabajo servirá como primera aproximación a la descripción de hardware, y probablemente no se llegue a implementar de manera física en una placa FPGA real debido a las limitaciones de la pandemia Covid19.

Desarrollo

Como las placas **FPGA** (Field Programmable Gate Array) típicamente no disponen de grandes cantidades de interruptores, será necesario contar con botones que guarden el estado actual de los interruptores (Switches) en determinados registros, para poder almacenar los valores con los que luego se va a operar. A continuación se presenta la figura 1, en la que se observa la configuración mencionada.

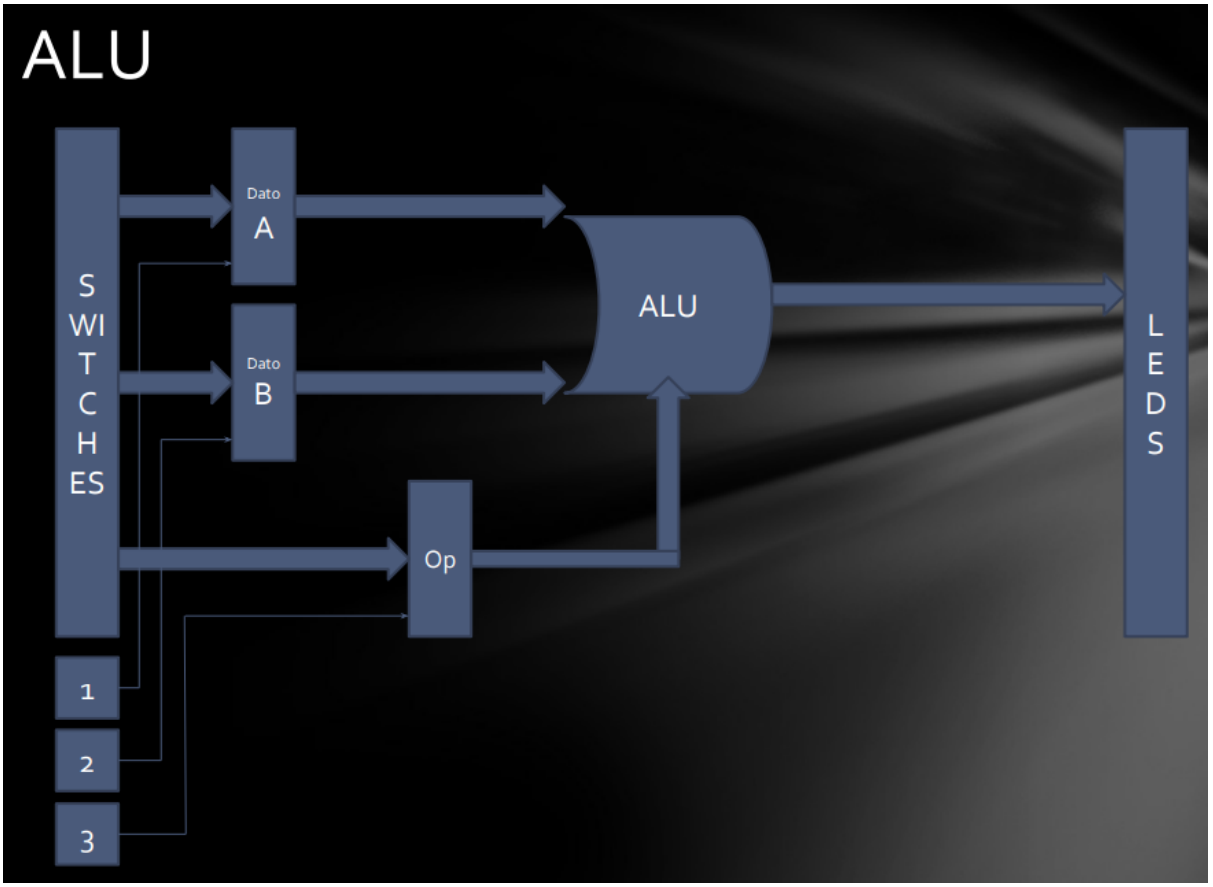


Figura 1: Disposición de Switches, Botones y Registros para la ALU

Los **módulos** que componen el sistema son los siguientes:

1) **ALU.v:** Módulo de operaciones aritméticas y lógicas, simplemente opera con sus entradas y presenta el resultado a la salida.
2) **tb_ALU.v:** Un testbench inicial propuesto para brindarle entradas al módulo ALU.v, no es empleado al final del trabajo pero sirve para una prueba preliminar.
3) **top_ALU.v:** Módulo top o general que se encarga de presentar los registros, y los comportamientos adecuados para que los botones introduzcan información en los registros correspondientes.
4) **tb_top.v:** El testbench general diseñado para probar todo el diseño por completo, instanciando el módulo ALU y generando valores aleatorios para los operandos y operaciones, permitiendo visualizar los resultados en la consola una vez ejecutados.

A continuación se procede a describir en detalle cada uno de los módulos antes listados, se sugiere acompañar la lectura con los códigos de lenguaje Verilog suministrados al final del presente documento para un mejor entendimiento.

Módulo 1) ALU.v

El núcleo de todo el sistema es el módulo ALU.v, en el cual se reciben las entradas de los registros A, B, y OP de la figura 1 (externos al módulo ALU), además se declara un reg para representar la salida del módulo. Es importante destacar que este módulo no tiene clock ya que se trata de un módulo completamente combinacional que ante una entrada presenta una salida sin almacenar ningún valor. El único motivo por el que la salida es de tipo reg es porque la herramienta necesita presentar la salida con una variable tipo reg en lugar de wire, de lo contrario se presenta una advertencia de generación de un latch provisorio para presentar el la salida del módulo.

El comportamiento de este módulo es muy sencillo, simplemente ante cada flanco positivo del clock verifica el código de operación en OP, y realiza la operación correspondiente para presentarla a la salida. Cada operación se identifica a través de un código binario, como se detalló al inicio de este informe. Finalmente asigna a la única salida el valor que se calculó.

Desde el punto de vista del circuito físico, este módulo se representa a través del diagrama que se muestra en la figura 2, en la cual se aprecia que el componente principal es un **multiplexor** que en función de su selector (OP) elige una de las 8 entradas correspondientes a cada una de las operaciones disponibles.

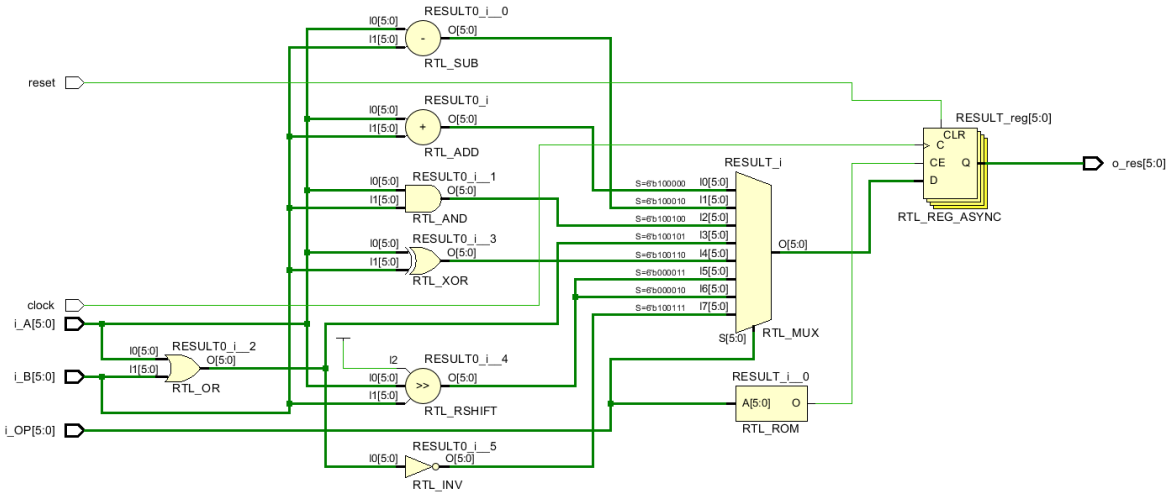


Figura 2: Diagrama de circuito del módulo ALU.v

Módulo 2) tb_ALU.v

No hay mucho para detallar de este módulo, porque es un módulo creado a modo de prueba y aprendizaje preliminar. Sencillamente este primer testbench brinda entradas con valores razonables para simular e instanciar el módulo ALU.v, sin tener que preocuparse por la forma en que el usuario ingresa esos valores.

Módulo 3) top_ALU.v

Este módulo general se ocupa de gestionar la interacción del usuario con la unidad aritmética y lógica. Se ocupa de gestionar los registros A, B y OP, para asignarle el valor actual de los interruptores (o switches) cuando el usuario presiona alguno de los 3 botones disponibles. Por supuesto se deben limpiar los valores de los registros en caso de que se pulse reset. Este módulo si posee señales de clock y de reset ya que se trata de un módulo principalmente secuencial.

Por último, se crea una instancia del módulo ALU.v pero esta vez se le suministra como valores de entrada, los registros mencionados con sus valores correspondientes, para que ALU se encargue de realizar las operaciones. Una vez la instancia de ALU termina de operar según el tiempo de su modo fundamental, se dispone a la salida del mismo el resultado de la operación para mostrarlo por los LEDs de la placa FPGA si se estuviera trabajando de manera física.

Módulo 4) tb_top

El testbench general se encarga de gestionar el sistema en su conjunto, en primer lugar con un arreglo OPs que contiene todos los códigos binarios de las operaciones, para así poder comparar con el ingresado al registro OP y efectuar la operación correcta.

Se indica que el clock alterna cada 1 unidad de tiempo, que por defecto son nanosegundos pero puede modificarse con la directiva ´timescale.

Además, la directiva **$random** proporciona un valor numérico aleatorio de 32 bits, que es truncado a la cantidad de bits correspondientes para cada uno de los registros con los que se trabaja. Los operandos y las operaciones se conforman de 6 bits, mientras que el registro NUM se usa para escribir aleatoriamente el estado de los switches, como lo solicita el enunciado.

También se genera un valor aleatorio de 3 bits en el registro AUX, que se usará como índice para recuperar un código del arreglo de operaciones antes mencionado. Como son 8 operaciones las que debe realizar el sistema, con 3 bits es suficiente para expresarlas todas.

Las directivas **$display** se usan para mostrar un valor por consola durante la simulación, para poder chequear que los resultados sean los esperados.

La figura 3 ilustra un diagrama general del módulo top, mostrando sus registros y la interacción de los mismos con el módulo ALU (de color celeste en la figura). Se observan también, que las entradas y salidas de la placa FPGA serán los interruptores y botones como entradas y los LEDs como salidas, representando de forma binaria el resultado de la operación actual.

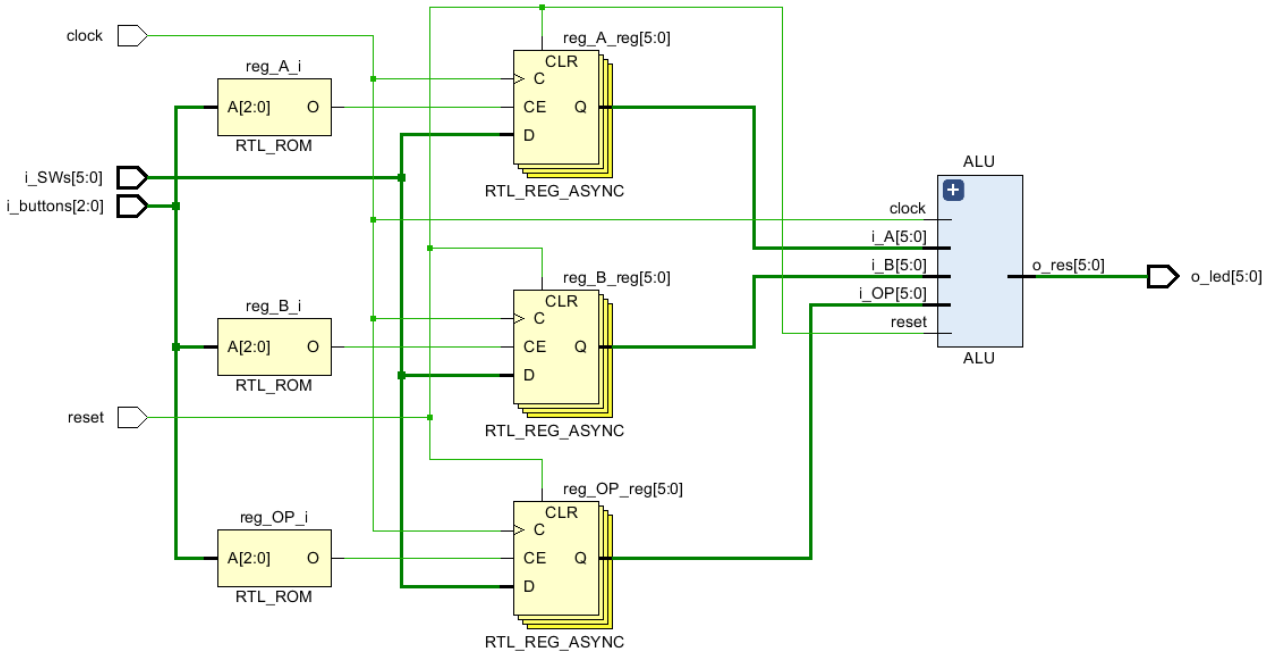


Figura 3: Diagrama general de todo el sistema

Análisis Temporal de las señales

Una vez descritos los módulos de trabajo, haremos un sencillo análisis de tiempo de las señales involucradas en el sistema. La herramienta de testbench nos permite ingresar un retardo temporal ante cada una de las instrucciones ejecutadas, de forma de poder simular una actuación real de un operario accionando los interruptores y botones.

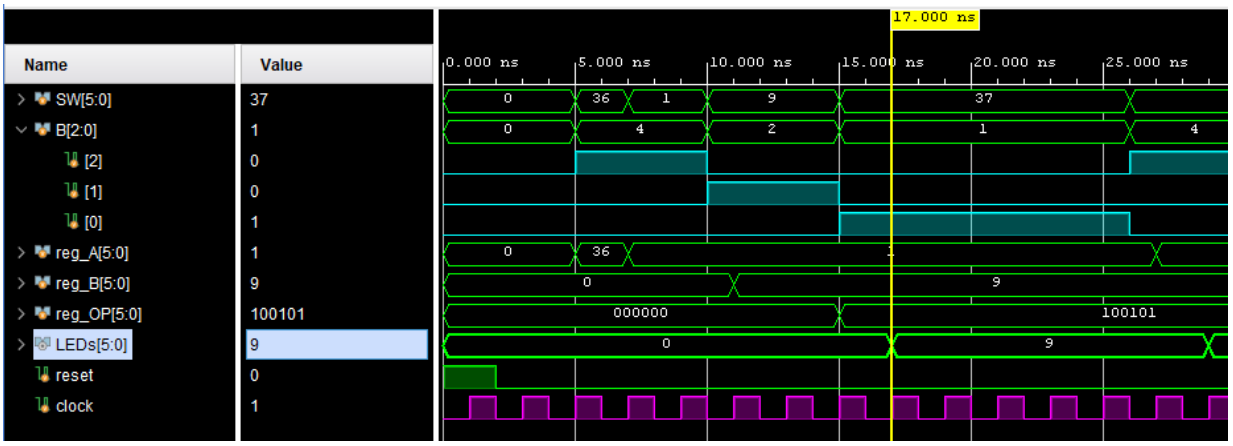


Figura 4: Señales del sistema en función del tiempo

La figura 4 muestra las principales **señales** de interés en un esquema temporal, en donde se quiere destacar:

- **Clock:** En color magenta se observa la señal de reloj o clock.
- **Botones:** En color turquesa se ven los 3 botones (B0, B1 y B2), siendo el estado alto (1 lógico) el estado de activación de los botones que da el accionamiento a guardar el estado de los switches en los registros, A, B y OP respectivamente a los botones presionados.
- **Reg A y B:** Operandos representados en valor decimal.
- **Reg OP:** Operación representada en valor binario.
- **SW:** El valor de los switches en cada momento.
- **LEDs:** El valor representado en los LEDs de salida, si bien se presenta en decimal, los leds físicamente deben ser interpretados en binario.

El instante de tiempo señalado por el cursor amarillo representa el momento en que se obtiene el resultado de una operación, en concreto un OR Lógico, siendo el resultado 9, que inmediatamente se escribe en los LEDs de salida. Cabe destacar que si bien el botón de operar se presiona un ciclo de reloj antes, el bloque always le da accionar a la operación una vez que se ingresa nuevamente el mismo bloque always pero con el registro de operación con el código de la operación OR en este caso.

Conclusiones

Como conclusión se expresa que **los resultados son correctos** con cualquiera de las operaciones aritméticas y lógicas, sin embargo para aquellas operaciones que arrojen un resultado que no pueda ser representado con los bits establecidos, se tendrá un **overflow** y por lo tanto el resultado será el módulo correspondiente. En nuestra aplicación se trabajó con 8 bits pero al ser parametrizable esto puede modificarse fácilmente en el futuro.

Con respecto al **consumo de potencia** y **porcentaje de utilización** de los recursos disponibles de la placa, se muestra el resultado hipotético de la síntesis, teniendo en cuenta que estos valores son de simulación y en ningún caso se implementó en una placa real.

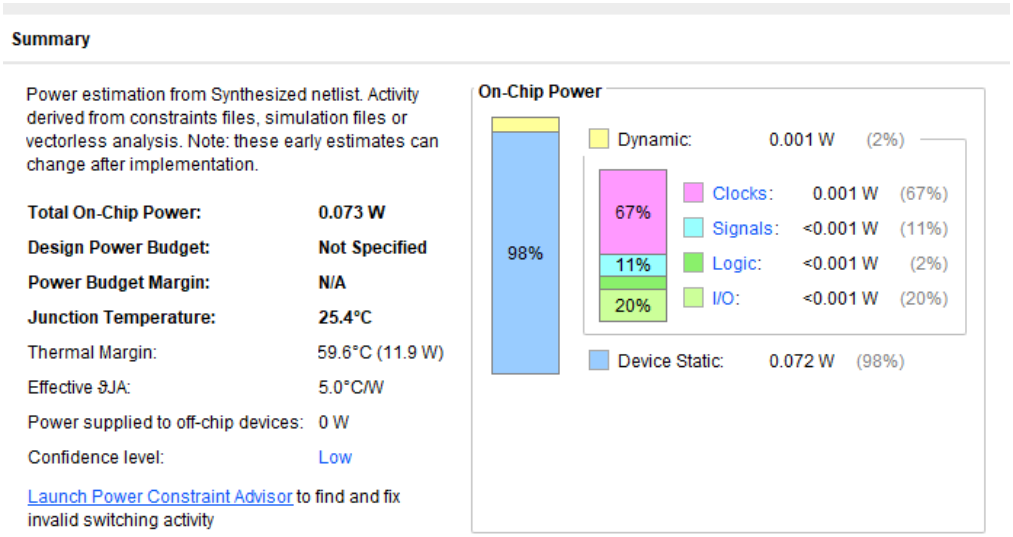


Figura 5: Datos de Consumo Hipotéticos

Referencias

Repositorio de Github: https://github.com/FranciscoCross/Arquitectura-TP1

Tutorial Vivado Simulations and Debugging: Vivado Design Suite Tutorial: Logic Simulation