Universidad Nacional de Córdoba.

Facultad de Ciencias Exactas, Físicas y Naturales.

ARQUITECTURA DE COMPUTADORAS

Trabajo Práctico Nº 1

ALU

Alumnos

- Perez, Esteban Andres ... 39026980
- Carrizo, Facundo ... 39419552

Docentes

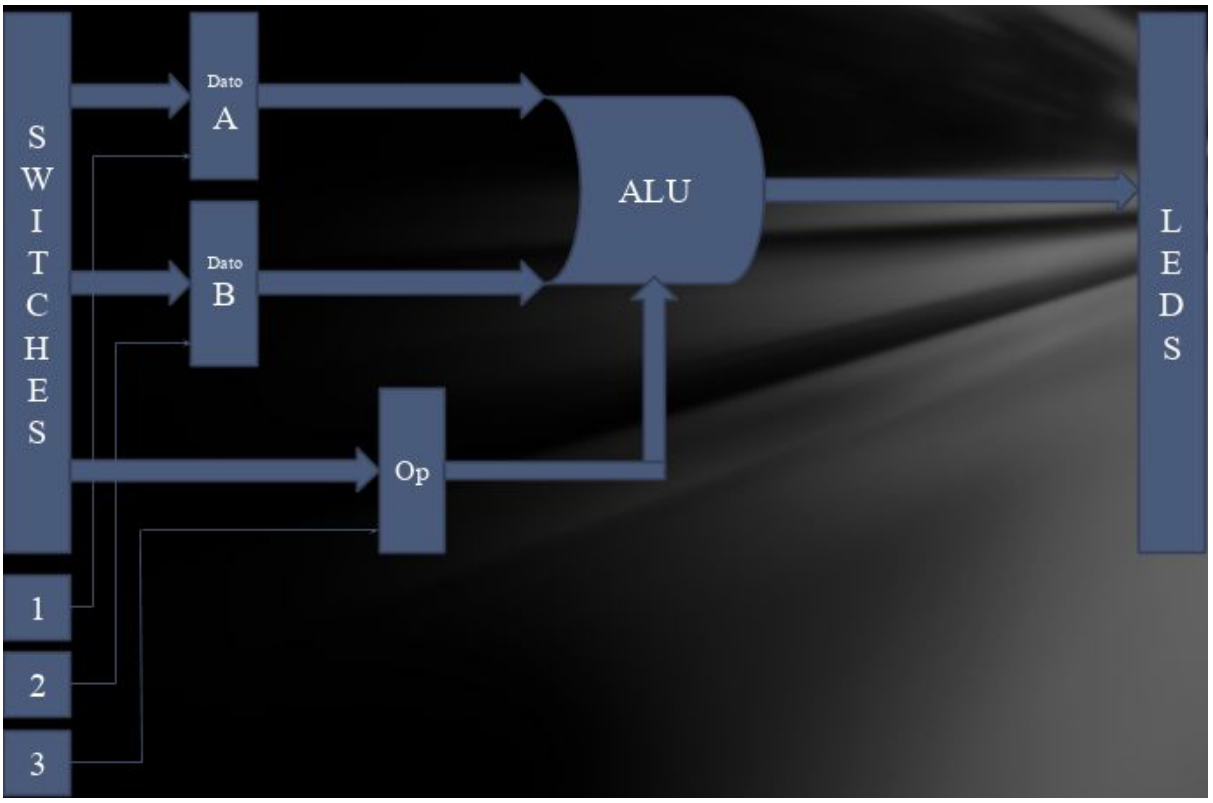
- Rodriguez, Santiago
- Pereyra, Martin

INTRODUCCIÓN

El presente informe se realiza con el objetivo de describir en detalle la implementación de una unidad de lógica aritmética ,con bus de datos parametrizable, mediante lenguaje de descripción de hardware, en lenguaje verilog, para luego poder ser utilizado en la placa FPGA,Basys 3.

Además de la ALU ,el sistema deberá implementarse de tal manera que permita validar el funcionamiento de la misma, desde la FPGA,mediante el uso de los switches y botones (1,2,3) para el ingreso de datos y de diodos leds para la salida del resultado,junto con los respectivos testbenches que validen el sistema completo .

En la siguiente figura se muestra la arquitectura del sistema a implementar



Objetivos

La ALU deberá soportar las operaciones de suma,resta,and,or,or exclusiva,or negado,desplazamiento lógico izquierdo y desplazamiento aritmético derecho, según los siguientes códigos de operación.

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

DESARROLLO

El sistema consta de dos módulos principales,el módulo ALU para la implementación de la unidad de aritmética lógica,y el módulo **TOP Alu** el cual tiene como finalidad sincronizar las entradas con el clock de la Fpga y proveer de una lógica que permita el cargado de datos a la unidad aritmética, para así luego poder presentar el resultado mediante los leds de la placa y el otro módulo llamado **Alu**.

A continuación se muestran los diagramas de bloque de los módulos mencionados

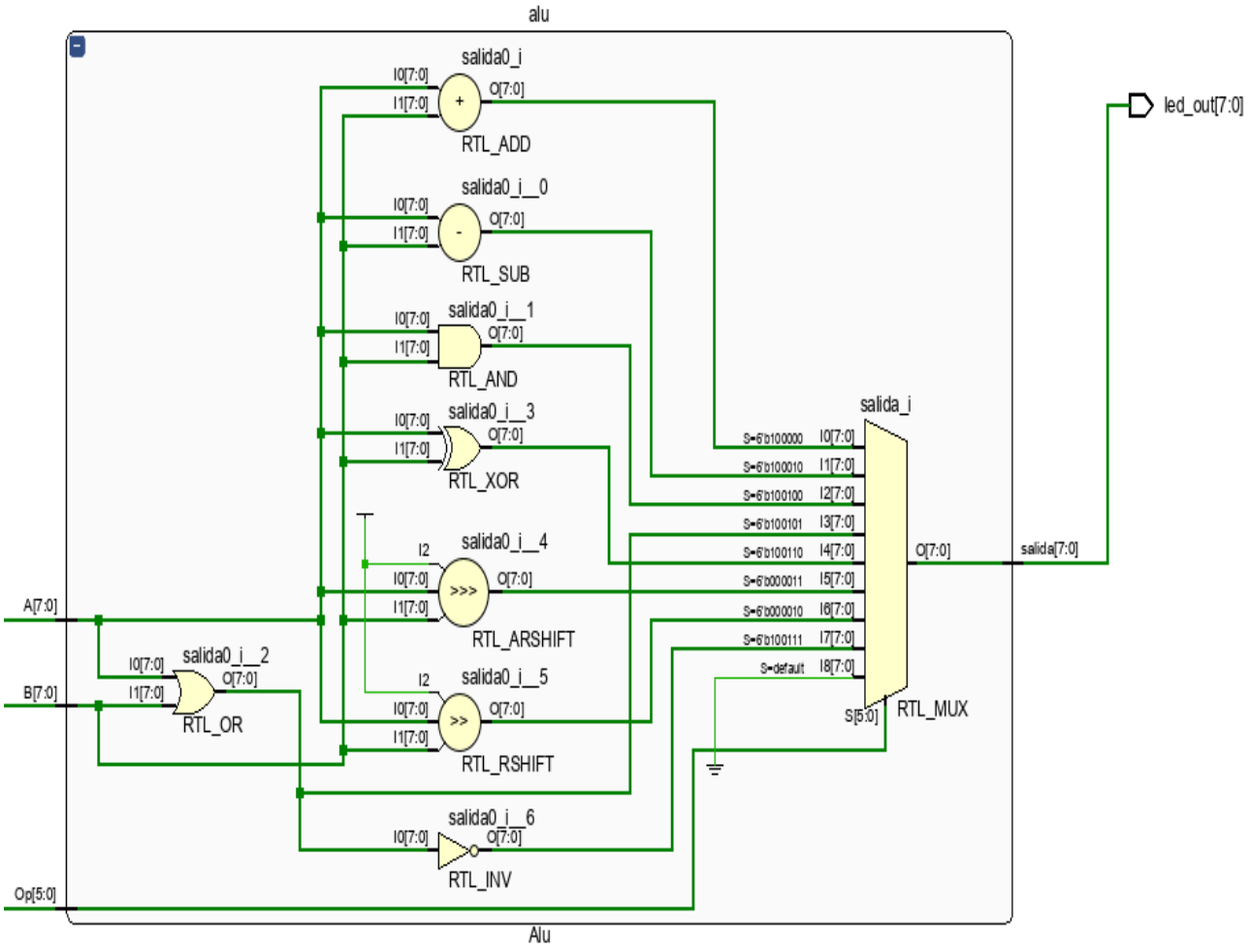


Figura 1: Modulo Alu

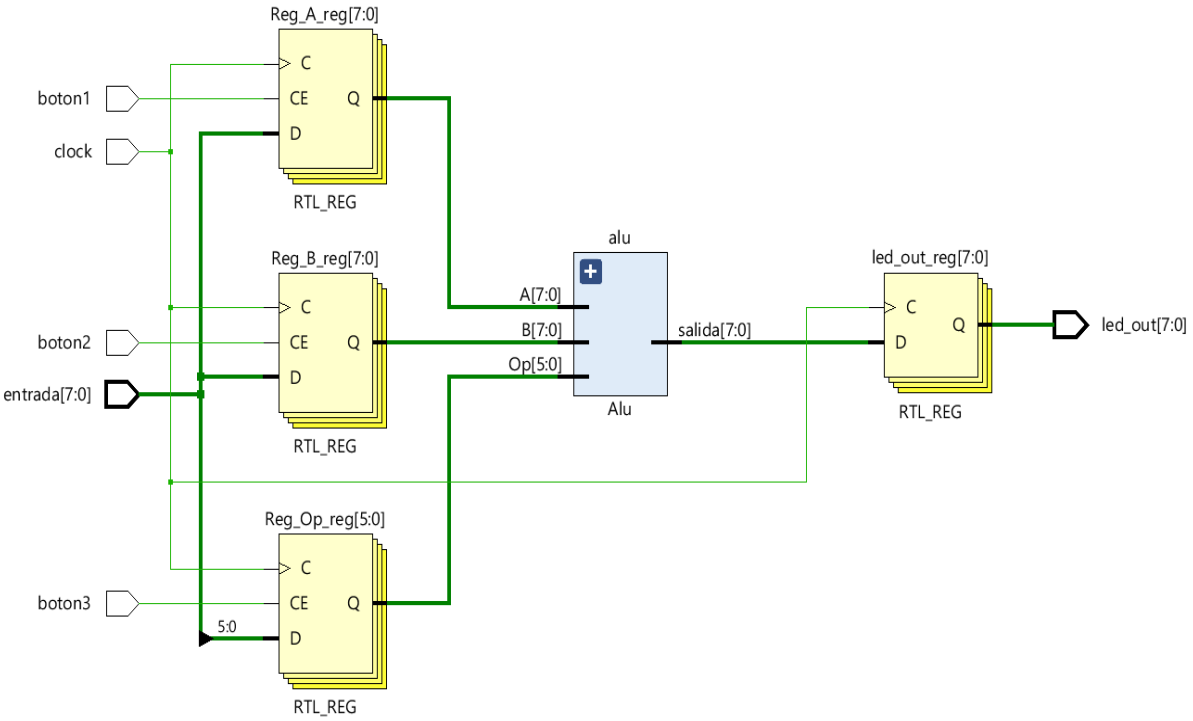


Figura 2: Módulo TOP

Para configurar el archivo constrain se utilizó la imagen 3 en el cual nos indica la posición de los switch y de los botones que están implementados en la FPGA (basys 3) así procedemos a interconectarlos con el módulo TOP de la Alu. Este archivo se encuentra adjunto en la carpeta del trabajo práctico.

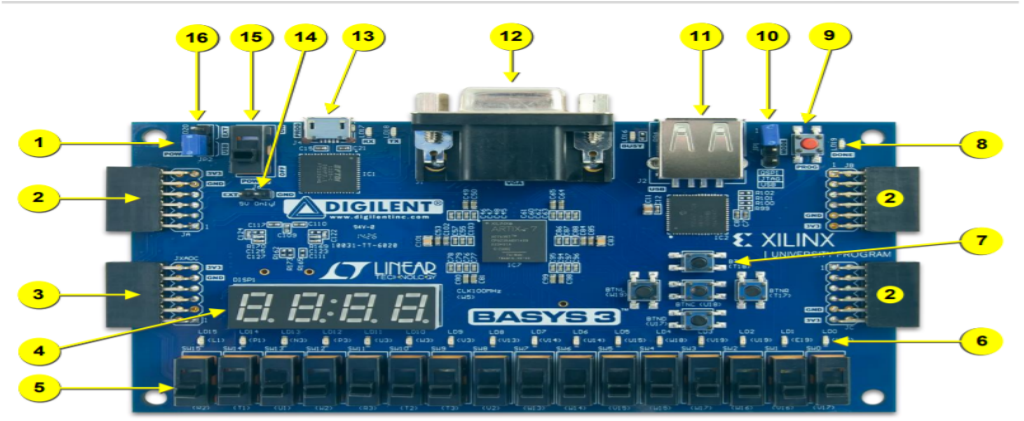


Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Table 1. Basys 3 Callouts and component descriptions.

Figura 3: basys3

TEST BENCH

Para la realización de los test bench se usaron las siguientes funciones:

- $fopen : esta función nos permite abrir un archivo y crearlo con permiso de escritura y tiene como valor de retorno un file descriptor que va a ser usado para escribir en él.
- $fdisplay: permite escribir en el archivo abierto anteriormente con formato.
- $random: genera números aleatorios de 32 bits.
- $monitor: monitorea los parámetros e imprime en pantalla cada vez que ocurre un cambio de los argumentos.
- $signed: castea el valor signado ya que desde verilog lee siempre los valores sin signo.

Además se implementó una task del sistema llamada assert, esta tarea tiene como utilidad verificar los resultados de la ALU.

Prototipo

```
task <task_name>;
    <I/O declarations>
    <variable and event declarations>
    begin // if more than one statement needed
     <statement(s)>
    end   // if begin used!
endtask
```

Implementación

```
task assert;
input  signed [BUS_LENGTH:0] A, B;
begin
   if (A == B)
        $fdisplay (fd,"TEST PASSED: value1 %d %b es igual a value2 %d %b \n", A, B, A, B);
   else
        $fdisplay (fd, "TEST FAILED: value1 %d %b es distinto a value2 %d %b \n", A, B, A, B);
end
endtask
```

SIMULACIÓN POST-IMPLEMENTACIÓN CON TIMING

Una vez descritos los módulos TOP y Alu, haremos un análisis profundo del delay de cada señal involucrada en el sistema.
Los testbench nos permite ingresar un retardo temporal ante cada una de los datos ingresados por el usuario y ver en pantalla o en un archivo los resultados de las operaciones que realizó la Alu.

A continuación se mostrará capturas de pantalla de la simulación donde se presencia un delay ante cada carga de valores en los registros involucrados y además un delay considerable entre que el dato está a la salida de la alu y su valor sincronizado al clock para guardarlo en un registro.

Señales involucradas en la simulación (waveform)

- btn_A: representa el botón que permite la carga del dato A.
- btn_B: representa el botón que permite la carga del dato B.
- btn_Op: representa el botón que permite la carga de la operación a realizarse en la Alu.
- datos: datos random generados en el switch.
- Reg_A, Reg_B: datos cargados una vez que se ha apretado el botón correspondiente.
- Alu_out: resultado a la salida de la Alu.
- Resultado, led_out: resultado de la Alu sincronizado con el clock para el guardado en un registro.

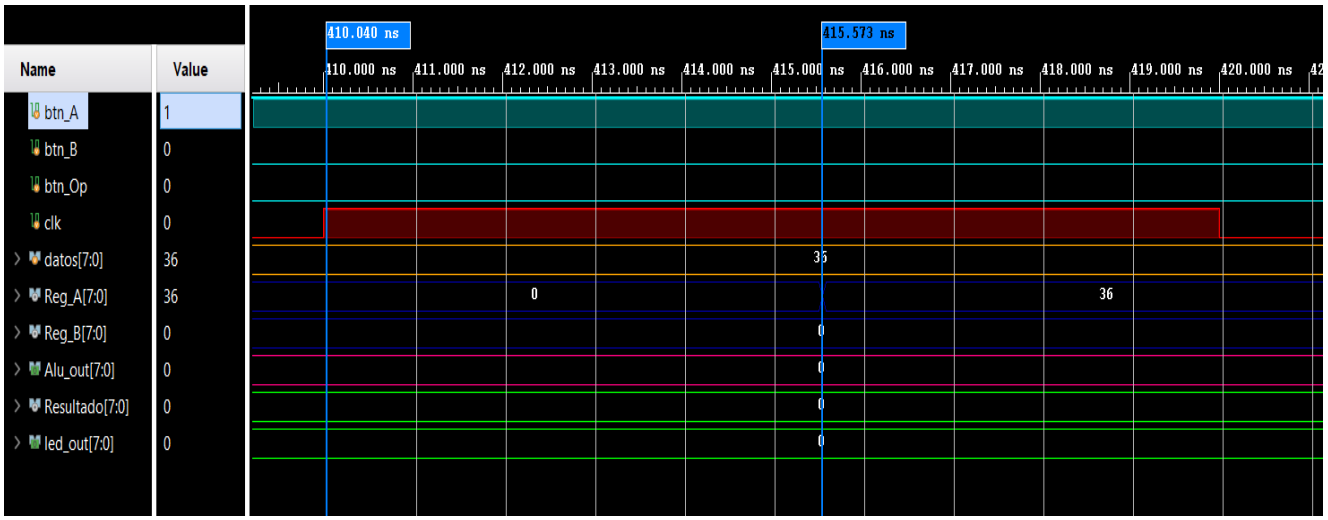


Figura 4: delay carga switch registro top ALU

En la imagen anterior el delay que hay entre que sucede el clock y se carga el valor del switch en el registro del módulo TOP es aproximadamente 5ns

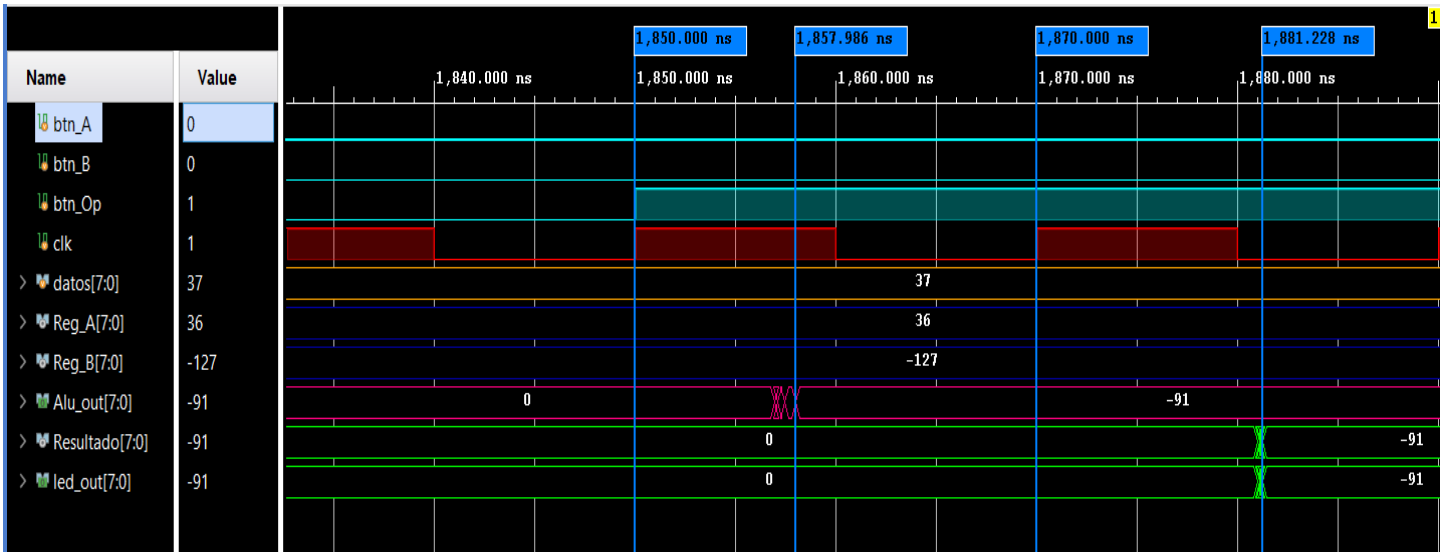


Figura 5: delay operación y resultado.

Analizando el resultado de la simulación con timing post-implementación se puede apreciar que desde que se carga el valor de la operación tarda en realizarse la suma aproximadamente 8ns.

Otro punto importante analizar, es sincronizar el resultado con el clock, por lo cual el resultado debería estar a la salida en un registro en el siguiente flanco positivo de clock, pero se logra mostrar que no es instantáneo debido a un delay de 11 ns.

Resultados de ejecución

```
Dato del switch:   30...00011110
Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: ADD : 100000
TEST PASSED: value1  -91 110100101 es igual a value2  -91 110100101

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: SUB : 100010
TEST PASSED: value1  -93 110100011 es igual a value2  -93 110100011

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: AND : 100100
TEST PASSED: value1    0 000000000 es igual a value2    0 000000000

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: OR : 100101
TEST PASSED: value1  -91 110100101 es igual a value2  -91 110100101

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: XOR : 100110
TEST PASSED: value1  -91 110100101 es igual a value2  -91 110100101

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: SRA : 000011
TEST PASSED: value1    0 000000000 es igual a value2    0 000000000

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: SRL : 000010
TEST PASSED: value1    0 000000000 es igual a value2    0 000000000

Dato_A:   36, Dato_B: -127
Dato_A: 00100100, Dato_B: 10000001
Operacion: NOR : 100111
TEST PASSED: value1   90 001011010 es igual a value2   90 001011010
```