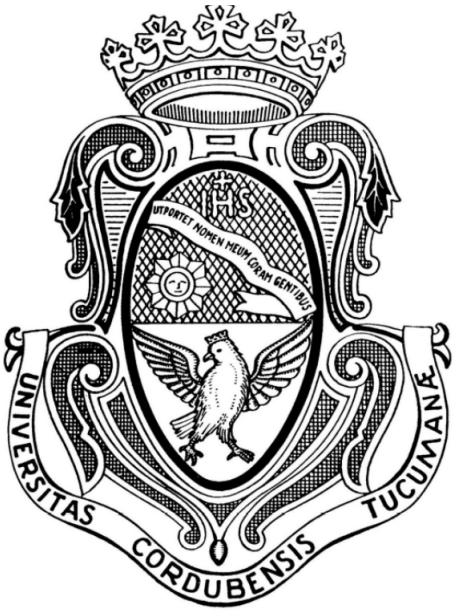
Universidad Nacional de Córdoba



Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Arquitectura de Computadoras

Trabajo Práctico 1: ALU

Profesor Titular: -
Profesor Adjunto: -

Integrantes:
Pallardó Agustín - apallardo@mi.unc.edu.ar
Trachta Agustín

15 de septiembre de 2025

Índice

1. Introducción

En este informe se documenta el desarrollo de una Unidad Aritmético-Lógica (ALU) implementada en una FPGA Basys 3. La ALU constituye un bloque fundamental en el diseño de procesadores y sistemas digitales, ya que permite realizar operaciones aritméticas, lógicas y de desplazamiento sobre datos binarios.

El objetivo principal de este trabajo práctico es adquirir experiencia en la descripción de hardware mediante Verilog, la validación con bancos de prueba, la simulación usando la herramienta Vivado, y la implementación física real.

Las herramientas utilizadas para el desarrollo fueron:

- **Lenguaje Verilog** para la descripción del hardware.
- **Vivado** como entorno de síntesis, simulación e implementación.
- **FPGA Basys 3** como plataforma de ejecución del diseño.

2. Objetivos

2.1. Objetivo general

Implementar y validar una Unidad Aritmético-Lógica (ALU) en una FPGA Basys 3.

2.2. Objetivos específicos

- Implementar una ALU parametrizable en Verilog, de modo que el ancho del bus de datos pueda ajustarse según futuras necesidades.
- Validar la implementación mediante un banco de pruebas (*testbench*) que genere entradas asignadas y contemple un mecanismo de validación.
- Integrar la ALU con los periféricos de la Basys 3 (switches, botones, LEDs y display de 7 segmentos) para verificar el correcto funcionamiento en hardware real.

3. Marco teórico

3.1. La ALU en el contexto de un procesador

La Unidad Aritmético-Lógica (ALU) es el bloque funcional encargado de ejecutar operaciones aritméticas, lógicas y de desplazamiento sobre operandos binarios. En un procesador, la ALU recibe los datos desde el banco de registros y ejecuta la operación seleccionada por la unidad de control. Su salida alimenta de vuelta el banco de registros y/o el camino de datos (bus) para su posterior uso. En arquitecturas *RISC*, la ALU suele implementar un conjunto mínimo pero completo de operaciones combinacionales de un ciclo, mientras que operaciones más complejas (multiplicación, división) pueden resolverse por hardware dedicado o microcódigo.

3.2. Representación de datos: complemento a dos y `signed/unsigned`

En hardware digital, los enteros con signo suelen representarse en *complemento a dos*. Para un ancho de palabra N, el rango representable es:

$$[-2^{N-1},\ 2^{N-1}-1]$$

En Verilog, declarar un bus como `signed` hace que las operaciones aritméticas y de desplazamiento aritmético consideren el bit más significativo (MSB) como bit de signo. Para operaciones lógicas (`&`, `|`, `^`) el signo no afecta el resultado, pero sí es relevante en sumas/restas y corrimientos aritméticos.

3.3. Operaciones implementadas

La ALU de este trabajo implementa las siguientes operaciones (las etiquetas son códigos de 6 bits que representan la operación seleccionada por `i_op`):

Código	Mnemónico	Descripción
100000	ADD	Suma de `i_data_a` e `i_data_b` (enteros con signo).
100010	SUB	Resta `i_data_a - i_data_b` (con signo).
100100	AND	AND bit a bit.
100101	OR	OR bit a bit.
100110	XOR	XOR bit a bit.
000011	SRA	Corrimiento a derecha **aritmético**: preserva el bit de signo.
000010	SRL	Corrimiento a derecha **lógico**: ingresa ceros por la izquierda.
100111	NOR	NOR bit a bit: `~(A | B)`.

3.4. Corrimientos: aritmético vs. lógico

- **SRA** (Arithmetic Right Shift) preserva el bit de signo: si el operando es negativo, se rellenan unos por la izquierda. Esto aproxima una división entre 2 para enteros

con signo (redondeo hacia $-\infty$ en representación a dos).

- **SRL** (Logical Right Shift) inserta ceros por la izquierda, apropiado para datos sin signo o para manipulación de campos de bits.

Al parametrizar el ancho de dato a N bits, el corrimiento máximo significativo es $N-1$. Es buena práctica enmascarar la magnitud de corrimiento con $\log_2(N)$ bits para evitar comportamientos dependientes de la herramienta:

$$\texttt{shift\_amt} = \texttt{i\_data\_b[}\lceil \log_2(N) \rceil\texttt{-1:0]}$$

3.5. Parametrización por ancho de datos

La ALU se parametriza mediante `NB_DATA` (ancho de datos) y `NB_OP` (ancho del código de operación). Esto permite reutilizar el diseño para 8, 16 o más bits cambiando sólo parámetros de síntesis, sin modificar el cuerpo del módulo.

3.6. Señales de estado (flags) y saturación (consideraciones)

En esta práctica no se implementan *flags* (Carry, Zero, Overflow, Negative) ni saturación; el resultado aritmético se trunca a `NB_DATA` bits (comportamiento de *wrap-around*). En extensiones futuras se pueden agregar:

- **Zero**: `o_result == 0`
- **Negative**: `o_result[NB_DATA-1]`
- **Carry/Overflow**: a partir del bit extra en la suma/resta.
- **Saturación**: limitar el resultado al máximo/mínimo representable.

3.7. Multiplexado de display de 7 segmentos

La Basys 3 posee 4 dígitos de 7 segmentos con *ánodo común*, controlados por líneas `o_an` (bajo activo) y segmentos `o_seg` (bajo activo). Se multiplexan los dígitos activando uno por vez a alta velocidad. En este diseño:

- Se usa un contador de 16 bits (`div`) a 100 MHz.
- La selección `sel = div[15:14]` genera 4 estados, uno por dígito.
- La tasa de avance de estados es $\frac{100\,\text{MHz}}{2^{14}} \approx 6103\,\text{Hz}$; el refresco por dígito es $\frac{6103}{4} \approx 1526\,\text{Hz}$, suficiente para evitar parpadeo.
- Se muestra el **resultado** de la ALU en hexadecimal (2 dígitos activos; los superiores en cero).

4. Desarrollo

4.1. Descripción general de la arquitectura

La arquitectura implementada se compone de cinco bloques principales:

1. **ALU**: bloque combinacional parametrizable que ejecuta las operaciones definidas por `i_op`.
2. **TOP**: bloque secuencial que registra operandos A/B y el código de operación a partir de los switches y botones de la Basys 3; además enruta resultados a LEDs y display.
3. **Hex_to_sseg**: decodifica un nibble hexadecimal (0–F) a segmentos (bajo activo).
4. **SevenSeg_hex**: multiplexa cuatro dígitos, selecciona el nibble a mostrar y controla `o_an`/`o_seg`.
5. **Testbench**: banco de prueba autocontenible que aplica estímulos representativos y observa salidas.

4.2. ALU: consideraciones de diseño

Interfaz y parámetros. La ALU recibe dos operandos `signed` de `NB_DATA` bits y un código de operación de `NB_OP` bits. La salida `o_result` es `signed` y del mismo ancho que los operandos.

Ruta de datos interna. Se utiliza un registro interno `r_result` de `NB_DATA+1` bits para computar resultados de suma/resta con un bit extra (posible acarreo/overflow). Finalmente se asigna a `o_result` truncando a `NB_DATA` bits (sin saturación).

Operaciones de corrimiento.

- **SRA**: `i_data_a >>> i_data_b`. Al ser `i_data_a` `signed`, el operador `>>>` replica el bit de signo.
- **SRL**: se fuerza lógico enmascarando el corrimiento: `$unsigned(i_data_a)>> i_data_b [$clog2(NB_DATA)-1:0]`.

Recomendación de robustez: también enmascarar el corrimiento en SRA con `i_data_b[$clog2(NB_DATA)-1:0]` para hacer explícito el límite del ancho, evitando dependencias de implementación.

Estilo de codificación. El bloque es combinacional (`always @(*)`) con asignaciones bloqueantes para la lógica interna; las señales de salida se asignan por `assign`. Este estilo evita *latches* y facilita el mapeo a LUTs/sumadores del FPGA.

4.3. TOP: captura de entradas y ruteo de salidas

Captura de operandos y operación.

- **Reset**: síncrono con `i_clk`; limpia registros `data_a`, `data_b`, `op`.
- **Botones**: al detectar `i_btn_a`, `i_btn_b`, `i_btn_op` se capturan respectivamente `i_sw_data` (operandos) y `i_sw_data[NB_OP-1:0]` (operación).
- **Nota**: en hardware real, los botones *rebotan*. Este diseño funcional no incluye *debounce*; para robustez en placa se recomienda añadir sincronización doble y filtro (p.ej., contador temporal).

Visualización en LEDs y display.

- `o_led_now` refleja en tiempo real `i_sw_data` (útil para cargar operandos/códigos).
- `o_led_res` muestra el resultado de la ALU.
- **Display 7-seg**: `disp_val = {8'h00, alu_result[7:4], alu_result[3:0]}` muestra el resultado en dos dígitos hex; los dos dígitos altos quedan en cero. El punto decimal (`o_dp`) permanece apagado.

4.4. Módulos de visualización

`hex_to_sseg`**.** Tabla de verdad que mapea un nibble (0–F) a segmentos **bajo activo**. La Basys 3 utiliza ánodos comunes por dígito y cátodos compartidos para segmentos, por eso la codificación de 0 es `7'b1000000` (todos los segmentos activos menos el punto).

`sevenseg_hex`**.**

- **Divisor**: contador de 16 bits a 100 MHz. Con `sel = div[15:14]` se recorre cada dígito a ∼6.1 kHz y cada dígito se refresca a ∼1.5 kHz (sin parpadeo).
- **Selección de nibble**: según `sel` se elige el nibble correspondiente de `i_value`.
- **Anodos**: `o_an` es bajo activo; se activa sólo el dígito seleccionado.

4.5. Banco de pruebas (*testbench*)

El `testbench` (`tb_top_alu`) instancia el `top`, genera reloj a 100 MHz (`always #5 clk = ~clk`) y define **tareas** para cargar operandos y operación:

- `load_A(v)`: coloca `v` en `sw`, pulsa `btn_a`.
- `load_B(v)`: análogo para `btn_b`.
- `load_OP(v)`: carga `v` (6 bits) en `sw` justificado a la derecha, pulsa `btn_op`.

Secuencia de prueba incluida:

1. Reset.
2. $A = -5$ (`0xFB`), $B = 3$.
3. **ADD** $\Rightarrow$ `0xFE` (-2).
4. **SUB** $\Rightarrow$ `0xF8` (-8).
5. Cargar $B = 1$ para corrimientos; **SRA**$(A, 1) \Rightarrow$ `0xFD` (-3).
6. **SRL**$(A, 1) \Rightarrow$ `0x7D` (125).
7. **NOR**$(A, 1) \Rightarrow$ `0x04`*.

* NOR($0xFB$, $0x01$): $\sim (1111\,1011 \mid 0000\,0001) = 0000\,0100$.

4.6. Integración con la placa Basys 3 (resumen)

La integración física se realiza mediante el archivo de *constraints* (`.xdc`) asignando:

- **Reloj 100 MHz**: `i_clk` (W5).
- **Switches**: `i_sw_data[7:0]`.
- **Botones**: `i_btn_a`, `i_btn_b`, `i_btn_op`.
- **LEDs**: `o_led_now[7:0]`, `o_led_res[7:0]`.
- **7 segmentos**: `o_seg[6:0]`, `o_an[3:0]` (bajo activo).

Nota: En el repositorio se incluye el `.xdc` correspondiente a la Basys 3 con las líneas relevantes descomentadas y renombradas.