

## Implementación de un ip core de una ALU de 4 bits con VHDL

Una ALU es un circuito digital que se utiliza para realizar operaciones aritméticas y lógicas. Es uno de los componentes esenciales en los microcontroladores. Es capaz de realizar las siguientes operaciones:

- Operaciones Aritméticas: suma, resta, multiplicación y división.
- Operaciones Lógicas: AND (Y), OR (O), NOT (NO) y XOR (O exclusivo).

El uso de FPGAs permite diseñar ALUs personalizadas que se ajustan a las necesidades específicas de una aplicación. Esto se hace utilizando lenguajes de descripción de hardware como VHDL. En la figura 1 se muestra la representación de la ALU de 4 bits que tiene como entrada dos parámetros (A y B) y como salida se obtiene el resultado (Result) de la operación que se le indicó realizar.

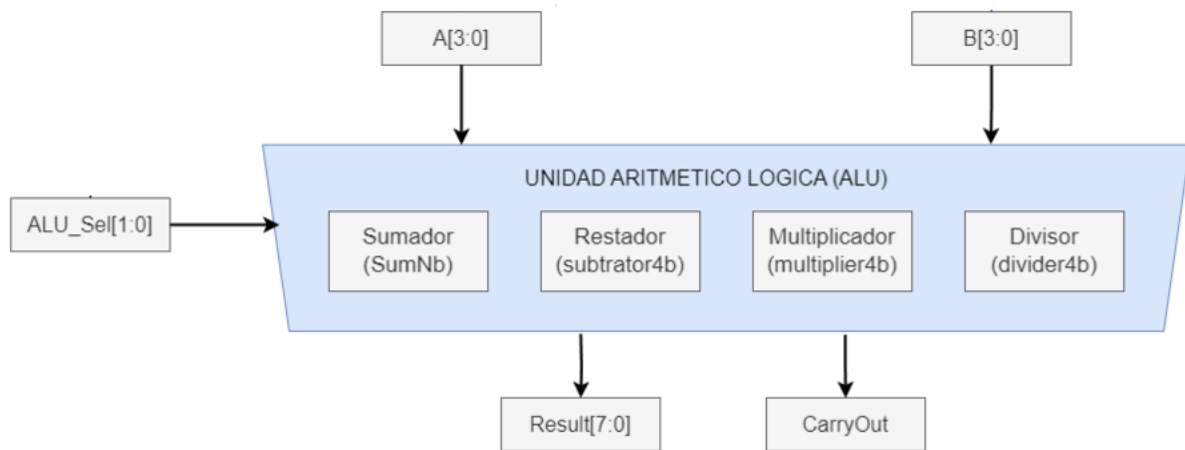


Figura 1. Diagrama de bloques ALU de 4 bits.

Para seleccionar qué operación aritmética se quiere realizar se utiliza un número de operación (ALU\_Sel).

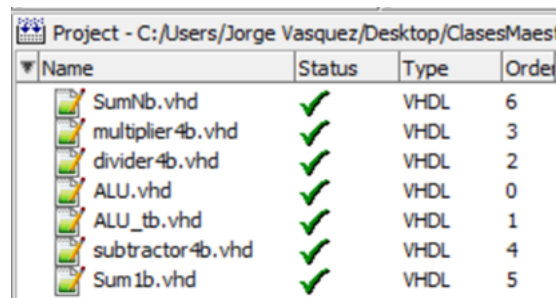
Tabla 1. Operaciones que realiza la ALU de 4 bits.

ALU_Sel	Result
0 0	A + B
0 1	A - B
1 0	A * B
1 1	A / B

La salida CarryOut se utiliza para la operación de la suma con el valor de 1 o 0 según sea el caso.

### Bloque principal de la ALU de 4 bits:

A continuación se muestra el test bench de la ALU de 4 bits.



Project - C:/Users/Jorge Vasquez/Desktop/ClasesMaesi

Name	Status	Type	Order
SumNb.vhd	✓	VHDL	6
multiplier4b.vhd	✓	VHDL	3
divider4b.vhd	✓	VHDL	2
ALU.vhd	✓	VHDL	0
ALU_tb.vhd	✓	VHDL	1
subtractor4b.vhd	✓	VHDL	4
Sum1b.vhd	✓	VHDL	5

Figura 2. Test bench de la ALU de 4 bits.

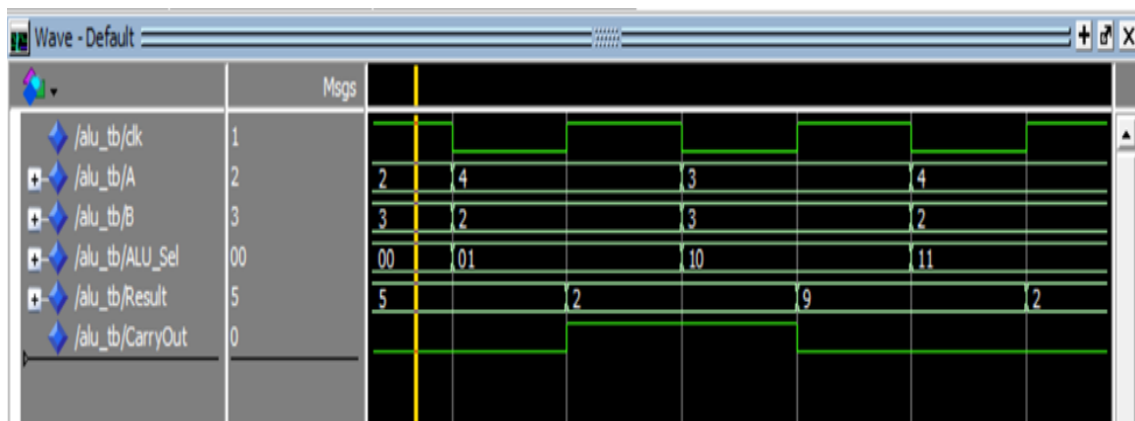


Figura 3. Simulación de la ALU de 4 bits.

### Ip core ALU:

En la figura 4 se muestra la conectividad entre el PS y la PL del sistema desarrollado. La **ALU IP** recibe/envía datos desde/hacia el micro de la FPGA. Dicho funcionamiento ha sido implementado a través de un código C.

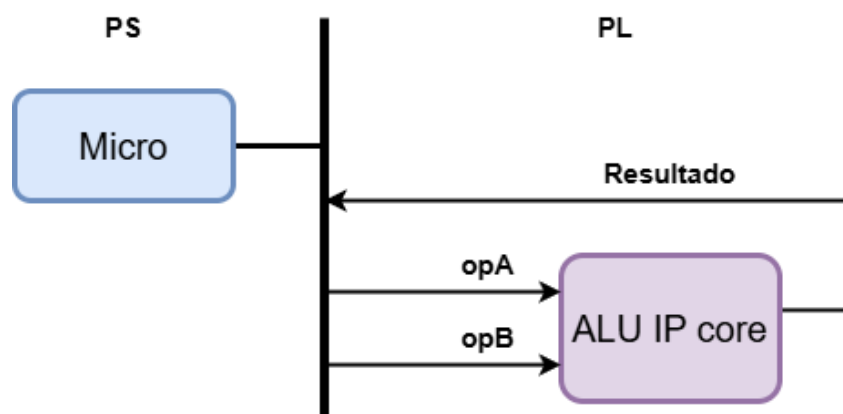
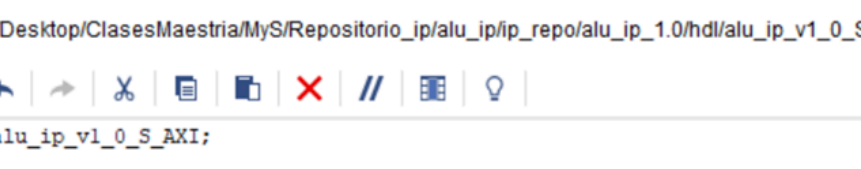


Figura 4. PS y PL del sistema a implementar.


Jorge Vasquez



The screenshot shows a VHDL code editor with the following content:

```
Project Summary x Package IP - alu_ip x alu_ip_v1_0.vhd x alu_ip_v1_0_S_AXI.vhd * x ? □ □  
c:/Users/jfvas/Desktop/ClasesMaestria/MyS/Repositorio_ip/alu_ip/ip_repo/alu_ip_1.0/hdl/alu_ip_v1_0_S_AXI.vhd x  
84 end alu_ip_v1_0_S_AXI;  
85  
86 architecture arch_imp of alu_ip_v1_0_S_AXI is  
87  
88 --Declaracion del componente ALU  
89 component ALU is  
90     port(  
91         clk : in std_logic; -- Reloj para la ALU  
92         A : in std_logic_vector(3 downto 0); -- Entrada A de 4 bits  
93         B : in std_logic_vector(3 downto 0); -- Entrada B de 4 bits  
94         ALU_Sel : in std_logic_vector(1 downto 0); -- Selector de operación  
95         Result : out std_logic_vector(7 downto 0); -- Resultado de la operación  
96         CarryOut : out std_logic -- Salida de acarreo o préstamo  
97     );  
98 end component;
```

Asimismo se tuvo que hacer la declaración de las señales que serán utilizadas para conectar la salida del core ALU.



```
123  ---- Number of Slave Registers 5
124  signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
125  signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
126  signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
127  signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
128  signal slv_reg4 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
129  signal slv_reg_rden : std_logic;
130  signal slv_reg_wren : std_logic;
131  signal reg_data_out :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
132  signal byte_index   : integer;
133  signal aw_en        : std_logic;
134
135  --Agrego las senales
136  signal sal_resul :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
137  signal sal_cout  :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
```

Figura 6. Agregando las señales.

## Microarquitecturas y Softcores

Jorge Vasquez

En la parte descriptiva de la arquitectura se instancia el componente ALU como se muestra en la figura 7.

```
416
417      -- Add user logic here
418
419      -- Instancia del componente ALU
420
421      alu_inst: ALU
422      port map(
423          clk => S_AXI_ACLK,
424          A => slv_reg0(3 downto 0),
425          B => slv_reg1(3 downto 0),
426          ALU_Sel => slv_reg2(1 downto 0),
427          Result => sal_resul(7 downto 0),
428          CarryOut => CarryOut_temp
429      );
430
431      -- Asignar CarryOut_temp a sal_cout
432      process(CarryOut_temp)
433      begin
434          sal_cout <= (others => '0'); -- Esto inicializa todos bits en '0'
435          sal_cout(0) <= CarryOut_temp; -- Asigna el bit de acarreo al bit menos significativo
436      end process;
437      -- User logic ends
438
439  end arch_imp;
440
```

Figura 7. Instancia del componente ALU

También se tuvo que modificar slv\_reg3 por sal\_resul en el proceso encargado de la lectura de los registros.

```
process (slv_reg0, slv_reg1, slv_reg2, sal_resul, sal_cout, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
        when b"000" =>
            reg_data_out <= slv_reg0;
        when b"001" =>
            reg_data_out <= slv_reg1;
        when b"010" =>
            reg_data_out <= slv_reg2;
        when b"011" =>
            reg_data_out <= sal_resul;
        when b"100" =>
            reg_data_out <= sal_cout;
        when others =>
            reg_data_out <= (others => '0');
    end case;
end process;
```

Figura 8. Modificación del código para poder leer la salida de la ALU.

Sistema completo:

La figura 9 muestra el esquema general del sistema que se ha implementado.

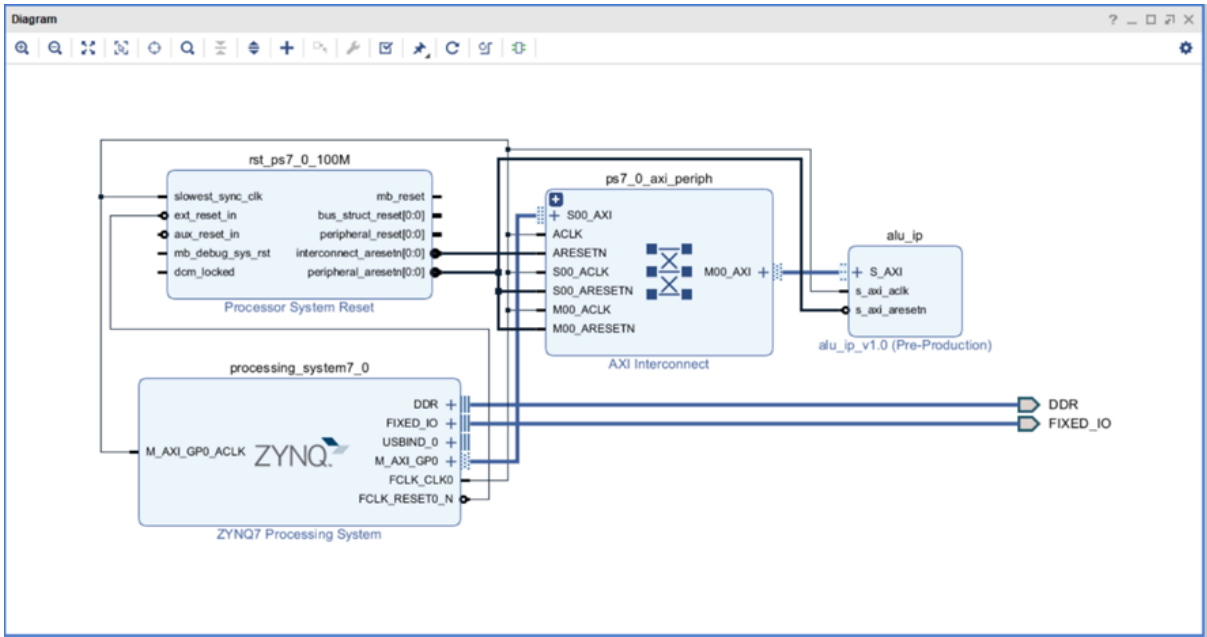


Figura 9. Sistema que incluye el ip core personalizado.

El sistema está formado por el microcontrolador, el sistema de reseteo, la matriz de interconexión y el ip core de la ALU de 4 bits que está conectado por el bus AXI al microcontrolador.

Tabla de uso de recursos de la FPGA:

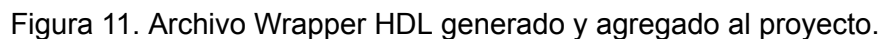
En las siguiente figura se muestra la tabla de uso de recursos del proyecto en el software vivado.

Utilization		Post-Synthesis   Post-Implementation		
		Graph   Table		
Resource	Utilization	Available	Utilization %	
LUT	478	17600	2.72	
LUTRAM	60	6000	1.00	
FF	666	35200	1.89	
BUFG	1	32	3.13	

Figura 10. Tabla de uso de recursos post-implementation.

Jorge Vasquez

Por último se generan las salidas de IP Integrator, el HDL top-level, y se ejecuta el SDK exportando el hardware.



El software desarrollado permite al microcontrolador realizar la escritura de los operadores “A” y “B”, así como la lectura del resultado que se obtiene del IP core de la ALU.

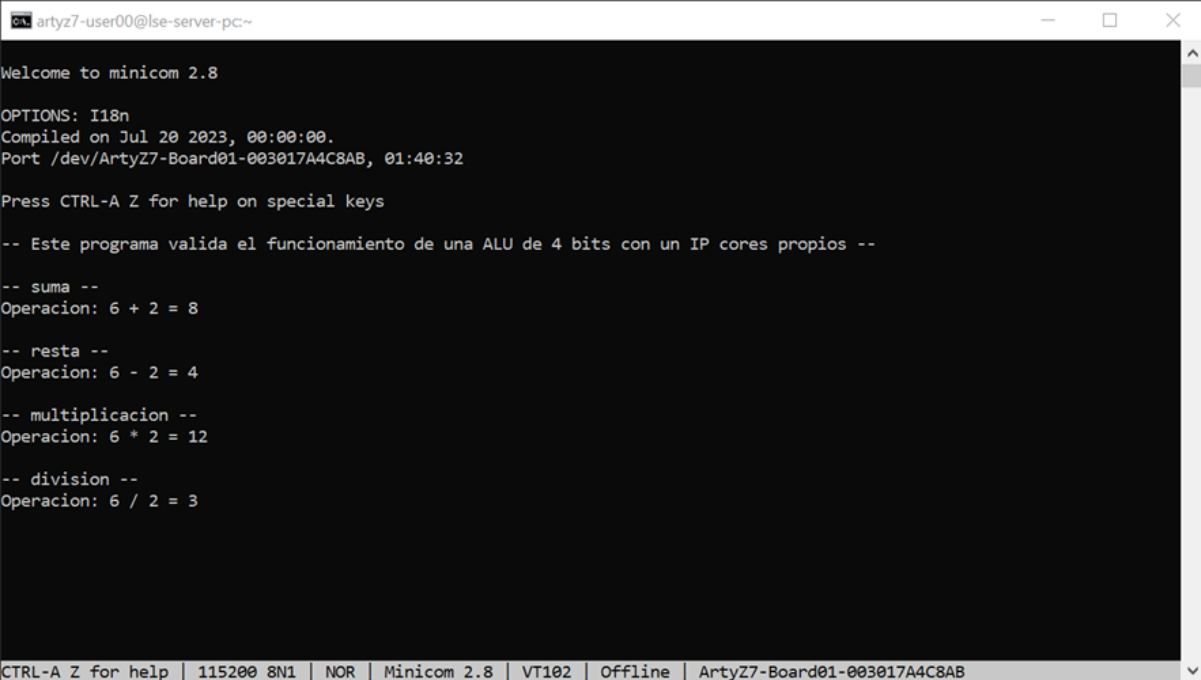


## Microarquitecturas y Softcores

Jorge Vasquez

### Resultado final:

Para realizar la prueba del sistema implementado, se hardcodean dos números y se verifica el resultado a través de la terminal.



```
artyz7-user00@lse-server-pc:~  
Welcome to minicom 2.8  
OPTIONS: I18n  
Compiled on Jul 20 2023, 00:00:00.  
Port /dev/ArtyZ7-Board01-003017A4C8AB, 01:40:32  
Press CTRL-A Z for help on special keys  
  
-- Este programa valida el funcionamiento de una ALU de 4 bits con un IP cores propios --  
  
-- suma --  
Operacion: 6 + 2 = 8  
  
-- resta --  
Operacion: 6 - 2 = 4  
  
-- multiplicacion --  
Operacion: 6 * 2 = 12  
  
-- division --  
Operacion: 6 / 2 = 3  
  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.8 | VT102 | Offline | ArtyZ7-Board01-003017A4C8AB
```

Figura 13. Resultado de la ejecución del código en el procesador.