.UBAfiuba
FACULTAD DE INGENIERÍA

Carrera de Especialización en Sistemas Embebidos

Unidad Aritmético-Lógica (ALU)

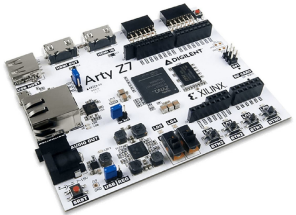
Autor: Mg. Luis Alberto Gómez Parada
Frof: Nicolas
correo: lgomez@patagones.cl

.UBAfiuba FACULTAD DE INGENIERÍA

Contenido

Objetivo del Proyecto

Implementar una Unidad Aritmético-Lógica (ALU) en FPGA como parte del trabajo práctico final del curso de Circuitos Lógicos Programables.



Aspectos Clave del Proyecto

- Diseño e implementación de una ALU de 4 bits
- Uso de VHDL como lenguaje de descripción de hardware
- Implementación en FPGA Xilinx ARTY-Z7 10
- Aplicación práctica de conceptos teóricos

¿Qué es una ALU?

Definición

ALU significa **A**rithmetic **L**ogic **U**nit (Unidad Aritmético-Lógica)

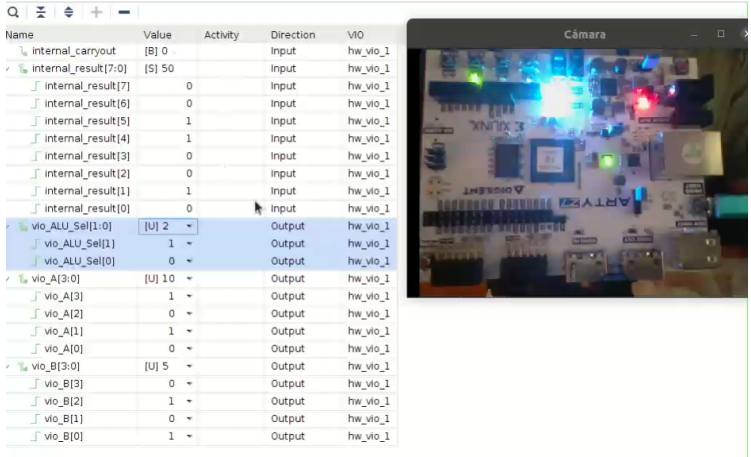
- Sistema dedicado dentro de una unidad de procesamiento
- Realiza operaciones aritméticas y lógicas específicas:
  - Aritméticas: suma, resta, multiplicación, división
  - Lógicas: AND, OR, NOT, XOR
- Componente fundamental en el diseño de procesadores
- Actúa como el "motor de cálculo" de un procesador
- Ejecuta operaciones bajo el control de la unidad de control del procesador

Características clave

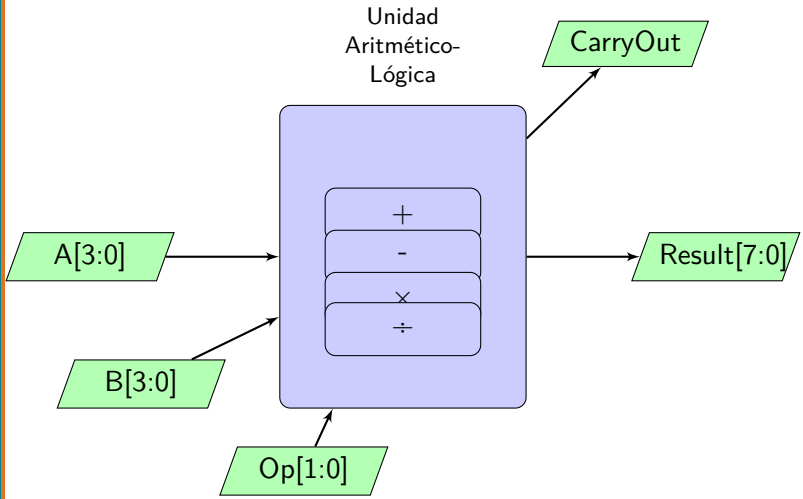
- Sistema dedicado: Diseñado para tareas específicas
- Alta velocidad: Optimizado para operaciones frecuentes
- Eficiencia: Realiza cálculos complejos con recursos mínimos

Características técnicas

- Ancho de bits de entrada: 4 bits
- Ancho de bits de salida: 8 bits (para acomodar la multiplicación)
- Frecuencia de operación: 125 MHz (periodo de 8 ns)
- Uso de Virtual I/O (VIO) para control y monitoreo



Descripción general de la ALU implementada



Definición VHDL de la ALU

Entidad ALU

```
1   entity ALU is
2   port(
3     clk : in std_logic;
4     A : in std_logic_vector(3 downto 0);
5     B : in std_logic_vector(3 downto 0);
6     ALU_Sel : in std_logic_vector(1 downto 0);
7     Result : out std_logic_vector(7 downto 0);
8     CarryOut : out std_logic
9   );
10  end entity ALU;
```

Arquitectura ALU

```
1   architecture Behavioral of ALU is
2   -- Senales internas
3     signal Sum: std_logic_vector(3 downto 0);
4     signal Difference: std_logic_vector(3 downto 0);
5     signal Product: std_logic_vector(7 downto 0);
6     signal Quotient: std_logic_vector(3 downto 0);
7     signal Remainder: std_logic_vector(3 downto 0);
8     signal Carry: std_logic;
9     signal Borrow: std_logic;
```

Descripción general de la ALU implementada (2/2)

Operaciones soportadas

Aritméticas:

- Suma: A + B
- Resta: A - B
- Multiplicación: A * B
- División: A / B

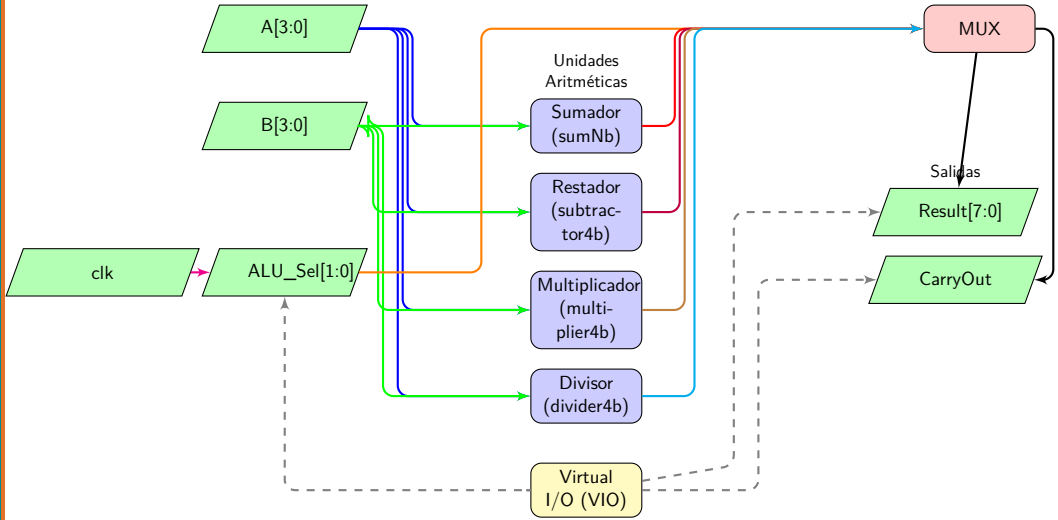
Características:

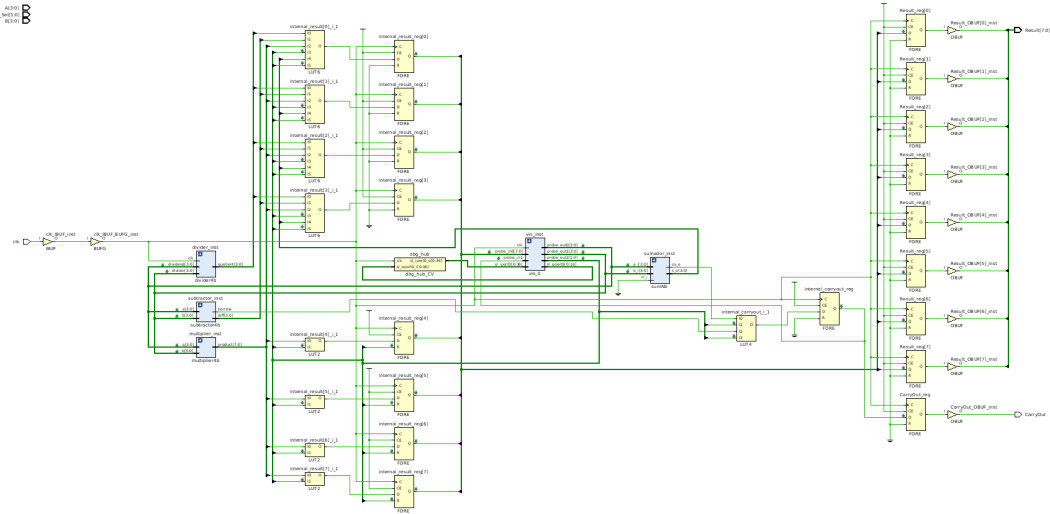
- Manejo de overflow en suma/resta
- Producto de 8 bits en multiplicación
- Manejo de división por cero

Ejemplo de funcionamiento

- Entrada A: 1010 (10 en decimal)
- Entrada B: 0011 (3 en decimal)
- Operación: Multiplicación (10 * 3)
- Resultado: 00011110 (30 en decimal)

Diagrama de bloques de la ALU





Operaciones de la ALU

1. **Suma (ALU_Sel = 00)**
   - Método: Sumador Ripple Carry
   - Operación: A + B
   - Salida: Result[7:0], CarryOut
2. **Resta (ALU_Sel = 01)**
   - Método: Complemento a 2 y suma
   - Operación: A - B
   - Salida: Result[7:0], Borrow (CarryOut invertido)
3. **Multiplicación (ALU_Sel = 10)**
   - Método: Suma y desplazamiento
   - Operación: A * B
   - Salida: Result[7:0] (producto de 8 bits)
4. **División (ALU_Sel = 11)**
   - Método: Resta sucesiva
   - Operación: A / B
   - Salidas:
     - Result[7:4]: Cociente
     - Result[3:0]: Resto

- Método: Suma con Propagación de Acarreo
- Entradas: A[3:0], B[3:0]
- Componente: Sumador (sumNb)
- Resultado: Sum[3:0] + Carry

Características del Ripple Carry:

- Suma secuencial bit a bit, de derecha a izquierda
- Cada etapa depende del acarreo de la etapa anterior
- Simple de implementar, pero puede ser lento para muchos bits

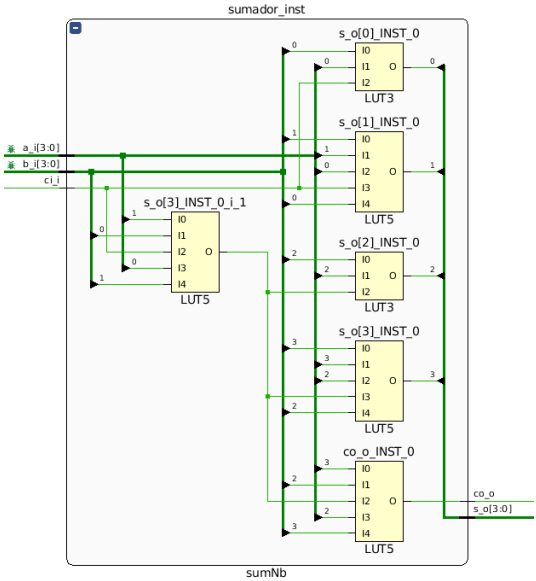
Ejemplo:

- A = 1101 (13 en decimal)
- B = 0110 (6 en decimal)
- Suma: 1101 + 0110 = 10011
- Resultado: 0011 (3), Carry = 1

Nota

La suma con propagación de acarreo es eficiente para ALUs pequeñas, pero para implementaciones más grandes se pueden considerar métodos más rápidos como la

Esquemático Sumador



Esquemático Subtractor

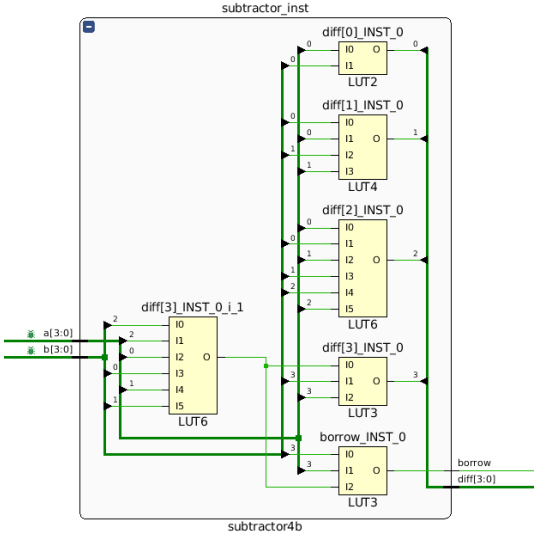


Figure:

Esquemático Multiplicación

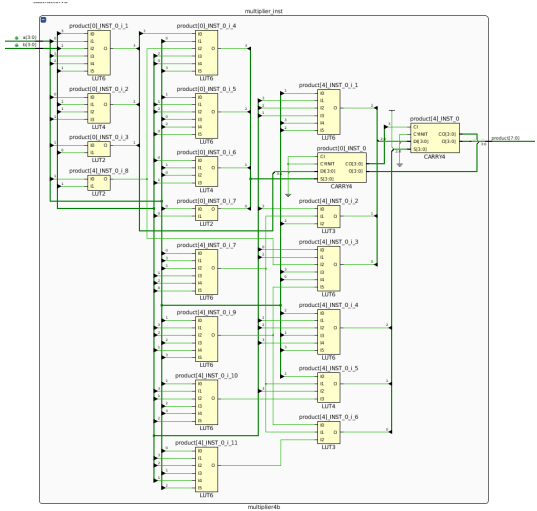
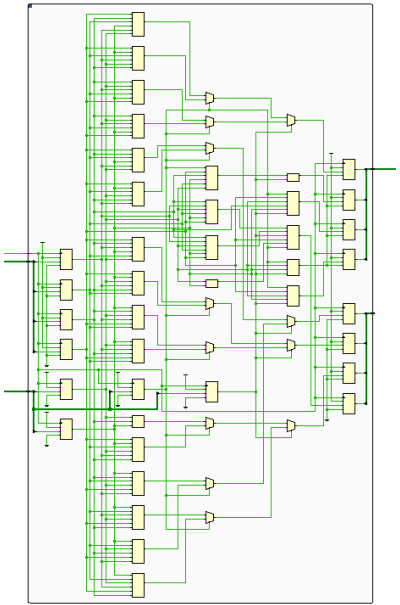


Figure:

Esquemático División



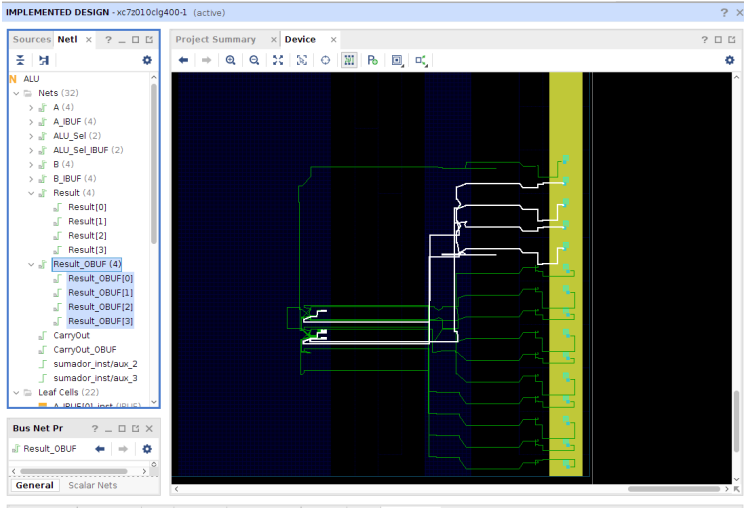

IMPLEMENTED DESIGN - xc7z010clg400-1 (active)
Sources
Netl
ALU
Nets (32)
A (4)
A_IBUF (4)
ALU_Sel (2)
ALU_Sel_IBUF (2)
B (4)
B_IBUF (4)
Result (4)
Result[0]
Result[1]
Result[2]
Result[3]
Result_OBUF (4)
Result_OBUF[0]
Result_OBUF[1]
Result_OBUF[2]
Result_OBUF[3]
CarryOut
CarryOut_OBUF
sumador_inst/aux_2
sumador_inst/aux_3
Leaf Cells (22)
Bus Net Pr
Result_OBUF
General
Scalar Nets
Project Summary
Device

Figure:

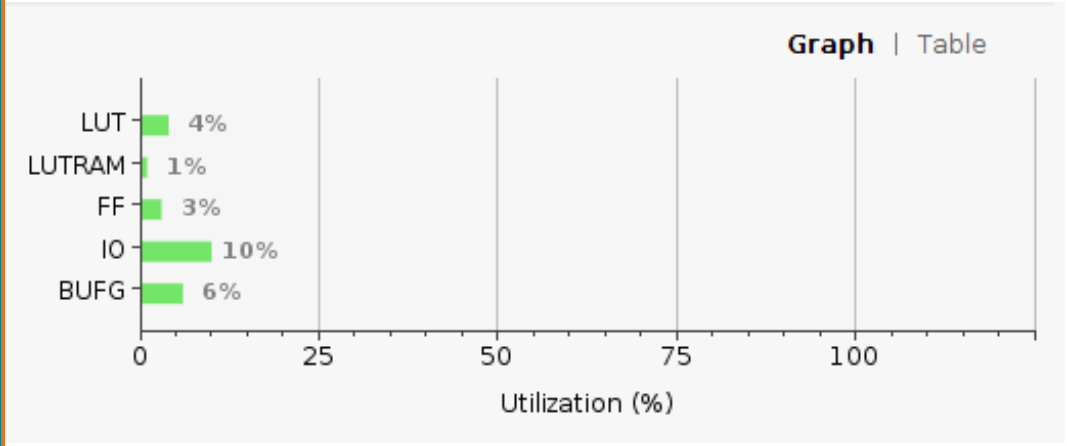
Utilización de recursos de la FPGA

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	658	17600	3.74
LUTRAM	24	6000	0.40
FF	1077	35200	3.06
IO	10	100	10.00
BUFG	2	32	6.25

Figure:


Graph | Table
LUT
4%
LUTRAM
1%
FF
3%
IO
10%
BUFG
6%
0
25
50
75
100
Utilization (%)

Simulaciones y resultados

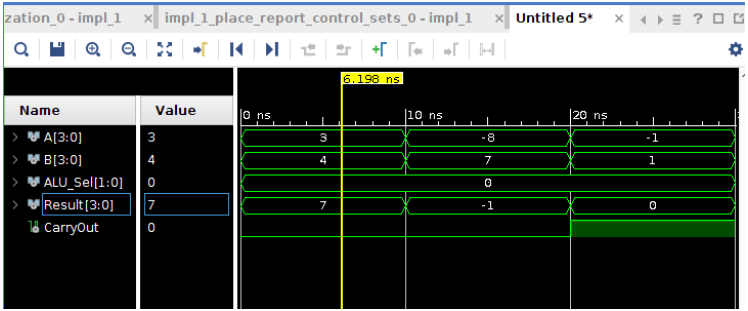


Figure: Simulación Suma

Desafíos y soluciones

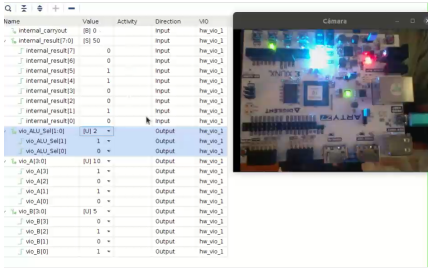
- Principales desafíos enfrentados durante el desarrollo
  - implementar método para entrada y salida de variables (VIO)
  - definición de variables adecuadas
  - encontrar métodos adecuados para las operaciones

Conclusiones y trabajo futuro

- Resumen de los logros principales
- Cumplimiento de los objetivos del proyecto
- Posibles mejoras o expansiones futuras
  - implementar parte lógica de al ALU
  - generar una interfaz de entrada y salida de datos real
  - mejorar algoritmos

Demostración y Recursos Adicionales

Video del Proyecto



https://www.youtube.com/watch?v=fYx1muBo78U

- Demostración de la ALU
- Explicación detallada del diseño
- Resultados y análisis

Código Fuente en GitHub

https://github.com/cpatagon/ALU_vivado

- Código VHDL completo
- Archivos de proyecto Vivado
- Documentación adicional

Preguntas y respuestas

¿Preguntas?