Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería

CIRCUITOS DIGITALES AVANZADOS

Practica 8
Unidad Aritmética y Lógica en Dispositivos Programables

Docente: Lara Camacho Evangelina

Alumnos:
Gómez Cárdenas Emmanuel Alberto **1261509**

Contenido

Objetivo

Simular una Unidad Aritmética y Lógica en dispositivos programables FPGA.

Equipo

Computadora con el IDE Xilinx Vivado u otro software para desarrollo de código para FPGAs.

Fundamento teórico

Una Unidad Aritmética y Lógica (ALU) es un circuito combinacional que realiza operaciones aritméticas y lógicas sobre números enteros, denominados operandos. La ALU es uno de los componentes básicos de los procesadores.

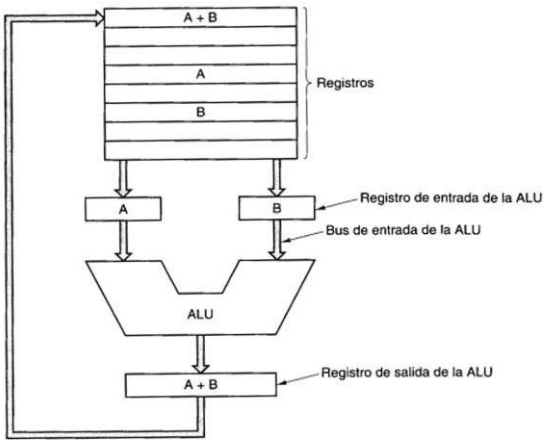

Figura 1. ALU.

Una ALU recibe las siguientes entradas:

- Operandos.
- Acarreo de entrada.
- Función. Este es un código que permite seleccionar la operación que va a realizar la ALU. La cantidad de bits de función determina el número máximo de operaciones diferentes que la ALU puede realizar. Si los bits de función son N, la

ALU puede realizar máximo 2N operaciones diferentes.

Una ALU produce las siguientes salidas:

- Resultado de la operación.
- Bits de estado del resultado. También denominados banderas, estos bits son señalesindividuales del estado del resultado de la última operación realizada por la ALU. Algunos de ellos son: acarreo de salida, sobreflujo, signo, cero y paridad. Cuando la condición asociada a la bandera ocurre, el bit de bandera se activa. Por ejemplo, siexiste un sobreflujo después de una operación aritmética, la bandera de sobreflujo se activa, de lo contrario se desactiva.

Desarrollo

- Revise los siguientes videos sobre los componentes de un sistema computarizado.Favor de revisarlos en el orden indicado:
  1. **ALU**: https://youtu.be/1l5ZMmrOfnA
  2. **Registros y memoria**: https://youtu.be/fpnE6UAfbtU
  3. **CPU**: https://youtu.be/FZGugFqdr60

- Cree en Vivado un nuevo proyecto de código VHDL llamado ALU donde implemente una ALU para operandos de 32 bits.

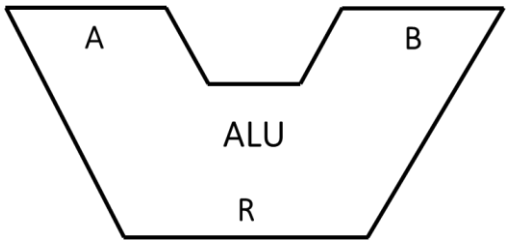

Figura 2. Símbolo de la ALU.La ALU debe tener las siguientes funciones:

Función	Operación	Descripción
0	AND A,B	$R = A$ A B
1	OR A,B	$R = A \vee B$
2	XOR A,B	$R = A \oplus B$
3	NOT A	$R = \overline{A}$
4	ADD A,B	$R = A + B$
5	ADC A,B	$R = A + B + Cin$
6	SUB B,A	$R = B - A$
7	SBB B,A	$R = B - A - Cin$
8	INC A	$R = A + 1$
9	DEC A	$R = A - 1$
A	SHL A	**Corrimiento a la izquierda de A**. Desplaza los bits de **A** una posición a la izquierda y coloca un 0 en la posiciónmenos significativa. El anterior bit más significativo se descarta de **A** y se coloca en **Cout**. *Ejemplo*: Sea **A** = 1010 1101. Se realiza **SHL A**.El resultado es: **A** = 0101 1010. **Cout** = 1.

B	SHR A	**Corrimiento a la derecha de A**. Desplaza los bits de **A** una posición a la derecha y coloca un 0 en la posiciónmás significativa. El anterior bit menos significativo se descarta de **A** y se coloca en **Cout**. *Ejemplo*: Sea **A** = 1010 1101. Se realiza **SHR A**.El resultado es:**A** = 0101 0110. **Cout** = 1.
C	ROL A	**Rotación a la izquierda de A**. Desplaza los bits de **A** una posición a la izquierda y coloca el bit más significativo enla posición menos significativa. El bit más significativo se copia a **Cout**. Puede considerarse como un *corrimiento circular a la izquierda*, donde en lugar de insertar siempre un cero, seinserta el bit **más significativo**. *Ejemplo*: Sea **A** = 1010 1101. Se realiza **ROL A**.El resultado es: **A** = 0101 1011. **Cout** = 1.
D	ROR A	**Rotación a la derecha de A**. Desplaza los bits de **A** una posición a la derecha y coloca el bit menos significativoen la posición más significativa. El bit menos significativose copia a **Cout**. Puede considerarse como un *corrimiento circular a la derecha*, donde en lugar de insertar siempre un cero, seinserta el bit **menos significativo**. *Ejemplo*: Sea **A** = 1010 1101. Se realiza **ROR A**.El resultado es: **A** = 1101 0110. **Cout** = 1.

Tabla 1. Funciones de la ALU.La ALU debe contar con los siguientes bits de estado:

Cin: Acarreo de entrada.
Cout: Acarreo de salida.
Z: Cero. Se activa si el resultado es cero.
Ov: Sobreflujo
P: Paridad. Se activa si el resultado tiene un número par de bits activos.

Después de realizar una operación aritmética, los bits **Cout**, **Z**, **Ov** y **P** son actualizados para reflejar el estado de **R**.
Después de realizar una operación lógica, **Z** y **P** son actualizados para reflejar elestado de **R**. Los bits **Cout** y **Ov** son puestos en cero.

ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU is
  generic (
    constant N: natural := 1;  -- number of shited or rotated bits
    constant nBits: integer := 32
    );
    Port (
    A, B    : in   STD_LOGIC_VECTOR(nbits-1 downto 0);  -- 2 inputs 32-bit
    ALU_Opc     : in   STD_LOGIC_VECTOR(3 downto 0);  -- 1 input 4-bit for selecting function
    ALU_Out : out  STD_LOGIC_VECTOR(nBits-1 downto 0); -- 1 output 32-bit
    Cout    : out  std_logic;        -- Carry Out flag
    Cin     : in   std_logic;
    Parity  : out  std_logic;
    Overflow: out  std_logic;
    Zero    : out  std_logic
    );
end ALU;

architecture Behavioral of ALU is

signal tmp : std_logic_vector (nBits downto 0);
signal temp2 : std_logic_vector (nBits-1 downto 0);
signal temp3 : std_logic_vector (nBits-1 downto 0);
signal temp4 : std_logic_vector (nBits downto 0);

begin
   process(A,B,ALU_Opc, Cin)
 begin
  case(ALU_Opc) is
  when "0000" => -- Logical and
   ALU_Out <= A and B;
  when "0001" => -- Logical or
   ALU_Out <= A or B;
  when "0010" => -- Logical xor
   ALU_Out <= A xor B;
  when "0011" => -- Logical not
   ALU_Out <= not A;
  when "0100" => -- Addition
   ALU_Out <= A + B ;
  when "0101" => -- Addition with carry
   ALU_Out <= A + B + Cin;
  when "0110" => -- Subtraction
   ALU_Out <= A - B - Cin ;
  when "0111" => -- Subtraction with carry
   ALU_Out <= A - B  ;
  when "1000" => -- Inc A
   ALU_Out <= A + 1 ;
  when "1001" => -- Dec A
   ALU_Out <= A - 1 ;
  when "1010" => -- Logical shift left
   ALU_Out <= std_logic_vector(unsigned(A) sll N);
  when "1011" => -- Logical shift right
   ALU_Out <= std_logic_vector(unsigned(A) srl N);
  when "1100" => --  Rotate left
   ALU_Out <= std_logic_vector(unsigned(A) rol N);
  when "1101" => -- Rotate right
   ALU_Out <= std_logic_vector(unsigned(A) ror N);
  when others => ALU_Out <= A + B ;
  end case;
 end process;

 tmp <= ('0' & A) + ('0' & B);
 Cout <= tmp(8); -- Cout flag
 Overflow <= tmp(8); -- Ov flag

end Behavioral;
```

ALU_tb

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY tb_ALU IS
END tb_ALU;

ARCHITECTURE behavior OF tb_ALU IS
    constant nBits: integer := 32;
    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ALU

    PORT(
        A       : IN   std_logic_vector(nBits-1 downto 0);
        B       : IN   std_logic_vector(nBits-1 downto 0);
        ALU_Opc : IN   std_logic_vector(3 downto 0);
        ALU_Out : out  std_logic_vector(nBits-1 downto 0);
        Cout    : out  std_logic;
        Cin     : in   std_logic;
        Parity  : out  std_logic;
        Overflow: out  std_logic;
        Zero    : out  std_logic
        );
    END COMPONENT;
    --Inputs
    signal A       : std_logic_vector(nBits-1 downto 0) := (others => '0');
    signal B       : std_logic_vector(nBits-1 downto 0) := (others => '0');
    signal ALU_Opc : std_logic_vector(3 downto 0)       := (others => '0');
    signal Cin     : std_logic;
     --Outputs
    signal ALU_Out : std_logic_vector(nBits-1 downto 0);
    signal Cout    : std_logic;
    signal Parity  : std_logic;
    signal Overflow: std_logic;
    signal Zero    : std_logic;
    signal i       : integer;
BEGIN

 -- Instantiate the Unit Under Test (UUT)
   uut: ALU PORT MAP (
        A       => A,
        B       => B,
        ALU_Opc => ALU_Opc,
        ALU_Out => ALU_Out,
        Cout    => Cout,
        Cin     => Cin,
        Parity  => Parity,
        Overflow => Overflow,
        Zero    => Zero
      );

   -- Stimulus process
   stim_proc: process
   begin
   A <= x"12345678";
   B <= x"FEDCBA98";
   ALU_Opc <= x"0";
   Cin <= '1';
   Zero <= '0';
   Parity <= '0';
   Overflow <= '0';
   wait for 100 ns;
   for i in 0 to 15 loop
    ALU_Opc <= ALU_Opc + x"1";
    if ALU_Out = x"0" then
       Zero <= '1';
    else
       Zero <= '0';
    end if;
    wait for 50 ns;
   end loop;
   A <= x"55555555";
   B <= x"AAAAAAAA";
   wait;
   end process;
END;
```

CONCLUSIONES

La ALU es de los componentes más importantes de un procesador ya que realiza todas las operaciones aritméticas y lógicas, después de esta práctica logramos aplicar nuestros conocimientos sobre la ALU y lo que hemos aprendido sobre VHDL.