INFORME
LABORATORIO No.1
Implementación de sistemas combinacionales en VHDL
Diseño de una ALU

Nelson Antonio Becerra Carrillo
nelsonabc25@hotmail.com

Jaime Alberto López Rincón
jaimealopezr@yahoo.com

Universidad del Quindío
Programa de Ingeniería Electrónica
Facultad de Ingenierías
Armenia, Colombia
Septiembre de 2004

OBJETIVOS

- Apropiarse de las técnicas de programación en lenguajes de descripción en hardware para la definición de circuitos lógicos combinatorios.

- Desarrollar un programa en VHDL que permita realizar diferentes operaciones matemáticas y lógicas que son características de una ALU (Unidad Aritmética Lógica).

- Familiarizarse con el manejo de las herramientas de desarrollo de CPLD/FPGAs de Xilinx.

- Determinar la técnica más optimizada para implementación de circuitos lógicos combinatorios.

DESCRIPCIÓN DEL PROBLEMA

En este laboratorio se pretende implementar una Unidad Aritmético Lógico de 8 bits. La unidad de contar con las siguientes operaciones:

- Suma,
- Resta,
- Complemento a 1,
- Complemento a 2,
- rotación a la izquierda,
- rotación a la derecha,
- desplazamiento a la derecha,
- desplazamiento a la izquierda,
- and,
- nand,
- or,
- nor,
- xor,
- xnor,
- comparación.

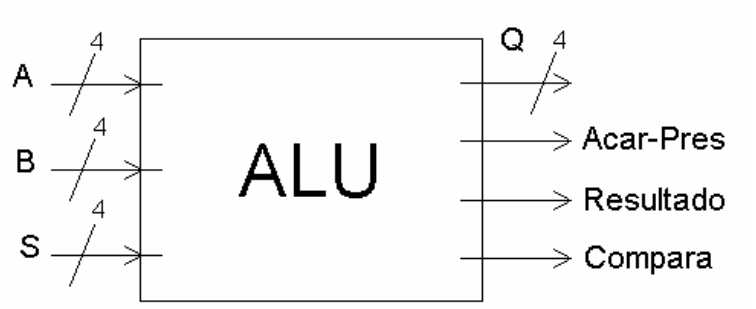
Así mismo, debe poseer las siguientes salidas:

- acarreo/prestamo,
- resultado = 0/comparación A = B verdadera
- resultado negativo/comparación A < B verdadera
- resultado positivo/comparación A > B verdadera

Para probar el desempeño de la implementación proponer diferentes esquemas de programación del sistema en notación de flujo de datos y comportamental, con el fin de verificar en el Fitting Report el porcentaje de bloques lógicos configurables empleados.

DESCRIPCIÓN DE LA SOLUCIÓN

Para realizar la implementación de la ALU propondremos una entidad como la que se muestra a continuación.



Los vectores A y B serán las entradas, S será el seleccionador para la operación que se desee realizar, el vector Q será la salida y Aca_Pres, Resultado y Compara serán las banderas que indican el resultado de una determinada operación.

Para llevar a cabo las Instrucciones, proponemos el siguiente orden en la entrada de cuatros bits S :

S3	S2	S1	S0	Instrucción
0	0	0	0	Suma A + B
0	0	0	1	Resta A –B
0	0	1	0	Complemento a 1
0	0	1	1	Complemento a 2
0	1	0	0	Rotar a la izquierda
0	1	0	1	Rotar a la derecha
0	1	1	0	Des. a la izquierda
0	1	1	1	Des. a la derecha
1	0	0	0	A and B
1	0	0	1	A nand B
1	0	1	0	A or B
1	0	1	1	A nor B
1	1	0	0	A xor B
1	1	0	1	A xnor B
1	1	1	0	Compara A y B
1	1	1	1	No hace nada

OBSERVACIONES

- La elaboración del algoritmo de la ALU, no trajo mayores dificultades, la implementación de esta fue hecha por dos métodos: funcional y flujo de datos, para luego determinar cual de los dos algoritmos consumía menos recursos para así lograr quemar el programa en la memoria.

- Después de corregir el algoritmo, procedimos a la simulación, confirmando de esta forma el buen funcionamiento de todas las operaciones que nos características de la ALU, así como del buen funcionamiento de todas la banderas.

- Al intentar quemar la ALU se presentaron varias dificultades:
    - El algoritmo implementado consumía muchos recursos, por esta razón inicialmente no se pudo quemar, para solucionar esto disminuimos de bits de las entradas y la salida a la mitad.

    - Luego al intentar quemar nuevamente el programa, no obtuvimos el voltaje deseado en la PAL, esto fue debido a que el regulador de voltaje estaba en mal estado. Al intentar subir el voltaje incrementamos el voltaje de la fuente y por esta razón se quemo uno de los condensadores. Luego, cambiando estos dos dispositivos, logramos quemar satisfactoriamente la ALU.

    - Después de quemar la ALU y polarizarla con 5V, se quemo debido a que el voltaje de polarización sólo era de 3.5V.

Después de superar todo estos inconvenientes quemamos nuevamente la ALU verificando su buen funcionamiento.

CONCLUSIONES

- La elaboración de la ALU nos permite realizar operaciones aritméticas y lógicas simples de una forma rápida y confiable, de ahí la importancia de saber programar para un ingeniero.

- Lenguajes de programación como VHDL o cualquier otro nos ofrecen posibilidades infinitas de aplicación, basta con un poco de imaginación para implementar programas que nos permitan realizar operaciones complicadas o cualquier otra tarea.

- El consumo de recursos en un programa es muy importante, debido a que si este es muy grande y la capacidad de la memoria es reducida, tendremos problemas al querer quemar el programa. Por consiguiente nuestro objetivo como ingenieros es lograr la mayor eficiencia, consumiendo la menor cantidad de recursos.

ANEXOS

A continuación anexamos el código del programa utilizado para implementar la ALU.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ALU is port(
            A: in unsigned (3 downto 0);
            B: in unsigned (3 downto 0);
            S: in unsigned (3 downto 0);
            Q: buffer unsigned (3 downto 0);
            Aca_pres:  out std_ulogic;
            Resultado: out std_ulogic;
            Compara: out std_ulogic
);
end ALU;

architecture flujo_ALU of ALU is
      signal  A1: unsigned (4 downto 0);
      signal  B1: unsigned (4 downto 0);
      signal  R: unsigned (4 downto 0);
begin
      with S select
            Q <=  ( A + B )         when "0000",
                  ( A - B )         when "0001",
                  ( not A )         when "0010",
                  ( not A ) + "0001" when "0011",
                  ( A rol 1 )       when "0100",
                  ( A ror 1 )       when "0101",
                  ( A sll 1 )       when "0110",
                  ( A srl 1 )       when "0111",
                  ( A and B )       when "1000",
                  ( A nand B )      when "1001",
                  ( A or B )        when "1010",
                  ( A nor B )       when "1011",
                  ( A xor B )       when "1100",
                  ( A xnor B )      when "1101",
                  "0000"            when others;

Resultado <= '1' when ( ( ( Q="0000" ) and ( S/="1110" ) ) or
                  ( ( S="1110" ) and ( A=B ) ) ) else '0';
Compara <= '1' when ( ( S="1110" ) and ( A>B ) ) else '0';
A1(4) <= '0' ;
A1( 3 downto 0 ) <= A;
B1(4) <= '0' ;
B1( 3 downto 0 ) <= B;
R <= ( A1 + B1 ) when ( S="0000" ) else
     ( A1 - B1 ) when ( S="0001" ) else
     "00000";
Aca_pres <= R(4) when ( S="0000" ) or ( S="0001" ) else '0' ;
end flujo_ALU;
```

Puesto que se va usar operaciones matemáticas, la definición del programa comienza definiendo la librería `IEEE.numeric_std.all` que es la que permite esta clase de operaciones.

El programa comienza definiendo la entidad en la cual se declaran las diferentes entradas (vectores A ,B y S) y salidas de la ALU (vector Q y banderas). Es destacable el hecho de que el vector Q se define como `buffer` porque siendo un bus de salida se utiliza en algunos condicionales. También hay que notar que los demás vectores junto con las banderas se declaran como `unsigned` y `std_ulogic` (enteros sin signo) para permitir el desarrollo de las operaciones aritméticas.

Luego al comienzo de la arquitectura se definen las señales internas que ayudan a realizar las operaciones requeridas en el problema. Hay que notar que cada uno de los vectores que son señales (A1, B1 y R) son de 5 bits (un bit más que los demás vectores declarados en la entidad). Este detalle permitirá calcular de una manera rápida y sencilla el acarreo y préstamo en las operaciones aritméticas.

En la arquitectura se selecciona la instrucción a realizarse de acuerdo al dato ingresado en el vector S (seleccionador), según la secuencia `with S select.` Allí se elige la operación a realizarse y se asigna el valor de la salida (Q).

La bandera `Resultado` se pone en uno cuando:

- La salida es “0000” y no está comparando.
- Cuando está comparando y A=B.

Esto se logra con el siguiente código:

```
Resultado <= '1' when ( ( ( Q="0000" ) and ( S/="1110" ) ) or
                    ( ( S="1110" ) and ( A=B ) ) ) else '0';
```

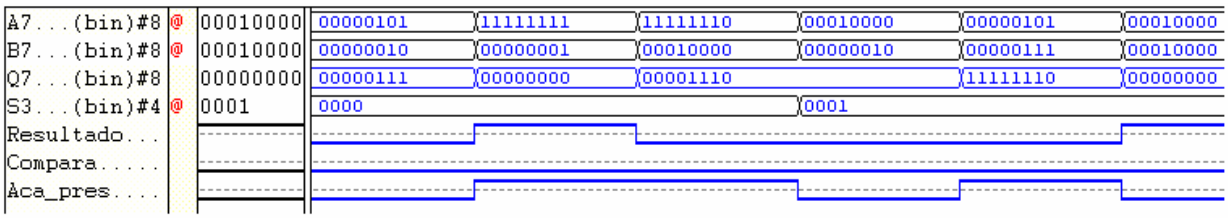
La bandera `Compara` se colocará en uno solo cuando se está comparando las entradas y A es mayor que B. Esto se logra por medio del siguiente código:

```
Compara <= '1' when ( ( S="1110" ) and ( A>B ) ) else '0';
```

Para calcular el acarreo y préstamo se utiliza la ayuda de las señales declaradas en la parte inicial de la arquitectura. Primero se pone el bit más significativo de cada una a cero. Luego se copian en los cuatro bits restantes los valores de las entradas A y B respectivamente. Después se realiza la suma de las señales de 5 bits (A1 y B1), guardando el resultado en R (otra señal de 5 bits). Por último la bandera del acarreo (`Aca_pres`) tomará el valor del bit más significativo del resultado (R(4)). Cabe anotar que la bandera `Aca_pres` sólo toma su valor si se está realizando una suma o una resta. Esto se logra por medio del código:

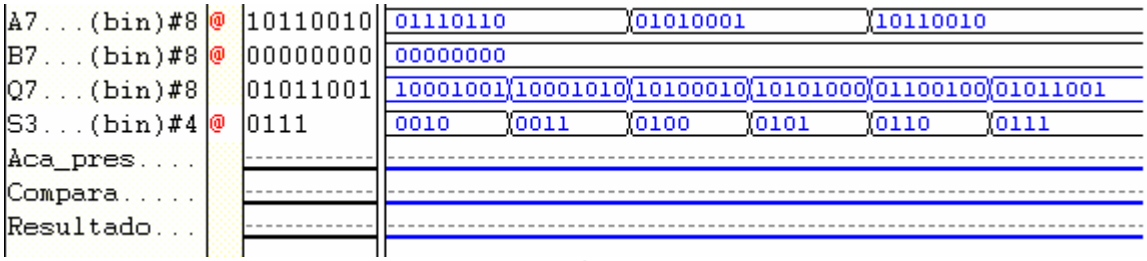
```
Aca_pres <= R(4) when ( S="0000" ) or ( S="0001" ) else '0' ;
```

Para comprobar el comportamiento de la ALU, simulamos el programa obteniendo los siguientes resultados.



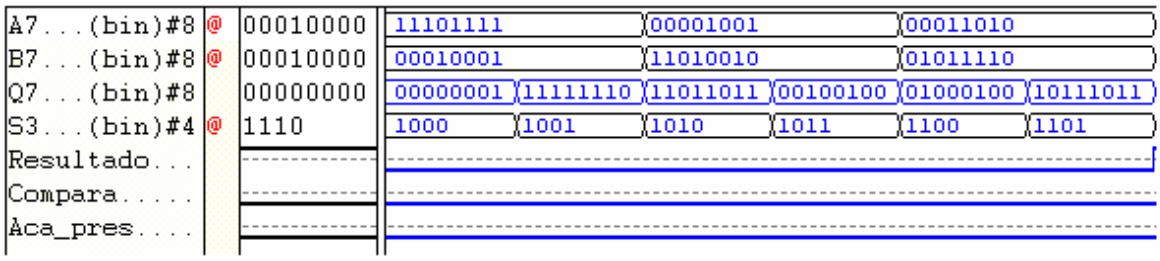
Simulación Parte I

En la primera parte de la simulación, se observan las instrucciones de suma (S="0000") y resta (S="0001"). En la salida (Q) se observa la ejecución adecuada de esta instrucciones. También es notable que la bandera `Resultado` se pone a uno cuando la salida es cero, y que la bandera Aca_pres se pone a uno cuando una de las operaciones aritméticas requiere más de 8 bits (FF) para almacenar el resultado.



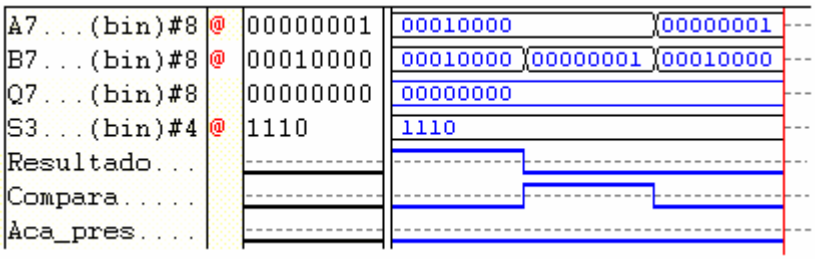
Simulación Parte II

En la segunda parte de la simulación se prueban las instrucciones 2 (Complemento a 1), 3 (Complemento a 2), 4 (Rotación a la izquierda), 5 (Rot. a la derecha), 6 (Desplazamiento a la izquierda), y 7 (Des. a la derecha). Es de notar que en las operaciones de rotación ninguno de los bits se pierde, mientras que en las operaciones de desplazamiento, uno de los bits (El más o menos significativo) se vuelve cero.



Simulación Parte III

En la tercera parte de la simulación se prueban las instrucciones 8 (A and B), 9 (A nand B), 10 (A or B), 11 (A nor B), 12 (A xor B), y 13 (A xnor B). En la simulación se observa que cada una de ellas se comporta adecuadamente.



Simulación Parte IV

En la cuarta parte de la simulación se prueba la instrucción 14 (A comparación B). En esta instrucción cuando las entradas son iguales la bandera `Resultado` se pone a uno; cuando A es mayor a B la bandera `Compara` se pone a uno y cuando A es menor que B esta última se pone a cero.