UNIVERSIDAD NACIONAL DE TUCUMÁN

FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍA

ELECTRÓNICA II

Actividad 7: Unidad Aritmética Lógica (ALU)

Autor: Chaves, Gonzalo Efraín - Estudiante de Ing. Electrónica

Resumen

En esta actividad se diseñó y verificó una Unidad Aritmética Lógica (ALU) implementada en VHDL, capaz de realizar diversas operaciones aritméticas y lógicas seleccionadas mediante un multiplexor. Entre las funciones implementadas se incluyen suma, resta, desplazamientos lógicos y aritméticos, comparaciones con y sin signo, y operaciones lógicas bit a bit (AND, OR, XOR).

El diseño se realizó de manera modular, incorporando submódulos como un detector de flanco ascendente y un decodificador hexadecimal a siete segmentos, lo que permitió una estructura más clara y reutilizable.

La verificación se efectuó mediante un banco de pruebas automático y posteriormente sobre una FPGA, confirmando el correcto funcionamiento de todas las operaciones. Los resultados demostraron el desempeño esperado del sistema y resaltaron la importancia de la simulación previa y del diseño modular para obtener un hardware confiable y verificable.

Introducción

Presenta las especificaciones de los componentes a desarrollar.

Esta actividad consiste en diseñar la descripción de hardware y prueba mediante un banco de pruebas, una Unidad Aritmética Lógica (ALU), la cual pueda realizar distintas operaciones lógicas y aritméticas de acuerdo a un multiplexor, de acuerdo a una tabla de Funciones ALU(TABLA 1.1).

TABLA 1.1 - Tabla de funciones implementadas por la ALU

sel_fn	Función(A, B) = Y =
0000	A + B (Suma)
0001	A - B (Resta)
001-	A << B (Desplazamiento a la izquierda Lógico)
010-	A < B (A menos que B) (Complemento a 2)
011-	A < B (A menos que B) (Binario Natural)
100-	A $\oplus$ B (XOR bit a bit)
1010	A >> B (Desplazamiento Lógico a la derecha)
1011	A >> B (Desplazamiento aritmético a la derecha)
110-	A OR B (OR bit a bit)
111-	A AND B (AND bit a bit)

En la entrada se deberá poder:

- Ingresar el valor de A
- Ingresar el valor de B
- Seleccionar la función que se desea realizar
- Cargar dicho resultado en A

Las entradas serán de una vector de 8 switches o interruptores, como se muestra en la figura 1.2.

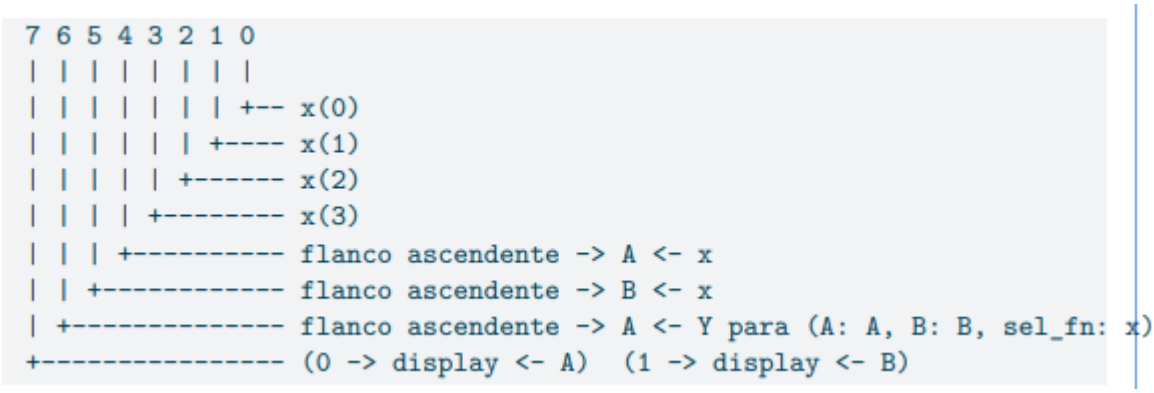

Figura 1.2 - Codificación de la entrada de los interruptores o switches

Para cargar un valor en un registro (A o B), se establece el valor deseado en X(3 a 0) y se activa momentáneamente el switch de carga correspondiente, X(4) para A o X(5) para B. De manera similar, para seleccionar una función se utiliza X(3 a 0), y para cargar el resultado (Y) en A se activa X(6).
Es decir X(3 a 0) sirve para ingresar la función y también los valores de A y B.

Además se deberá visualizar las entradas de la operación (A y B) de acuerdo a un selector.

La salida es hexadecimal a un display de 7 segmentos. Y en caso de que el resultado de la operación sea igual a “0“, se deberá mostrar en el punto del display(encendido si es cero, apagado si no lo es) antes de cargar dicho valor a A. Esto permite saber de antemano el valor del resultado(si es cero o no).

El diseño será cargado a una FPGA para poder comprobar su funcionamiento, por lo que deberá ser sincrónico con el reloj de la misma.

Materiales y Métodos

Para el diseño y la verificación de la ALU, se comenzó desarrollando un programa en C que simulaba la ALU según la Tabla 1.1. Esto permitió generar un archivo de texto patrón con valores aleatorios de entrada (A, B, y sel_fn) y las correspondientes salidas esperadas (Y y Z), fundamental para el banco de pruebas automático en VHDL..

Una vez generado nuestro archivo de texto, se realizó en banco de pruebas a partir de él una actividad anterior, el cúal deberá leer los archivos del texto patrón, y comprobar con los resultados de la ALU descrita en VHDL. De este modo se puede comprobar rápidamente el correcto funcionamiento Aritmético Lógico de la ALU realizada, y encontrar fallos en la descripción antes de continuar con el desarrollo.

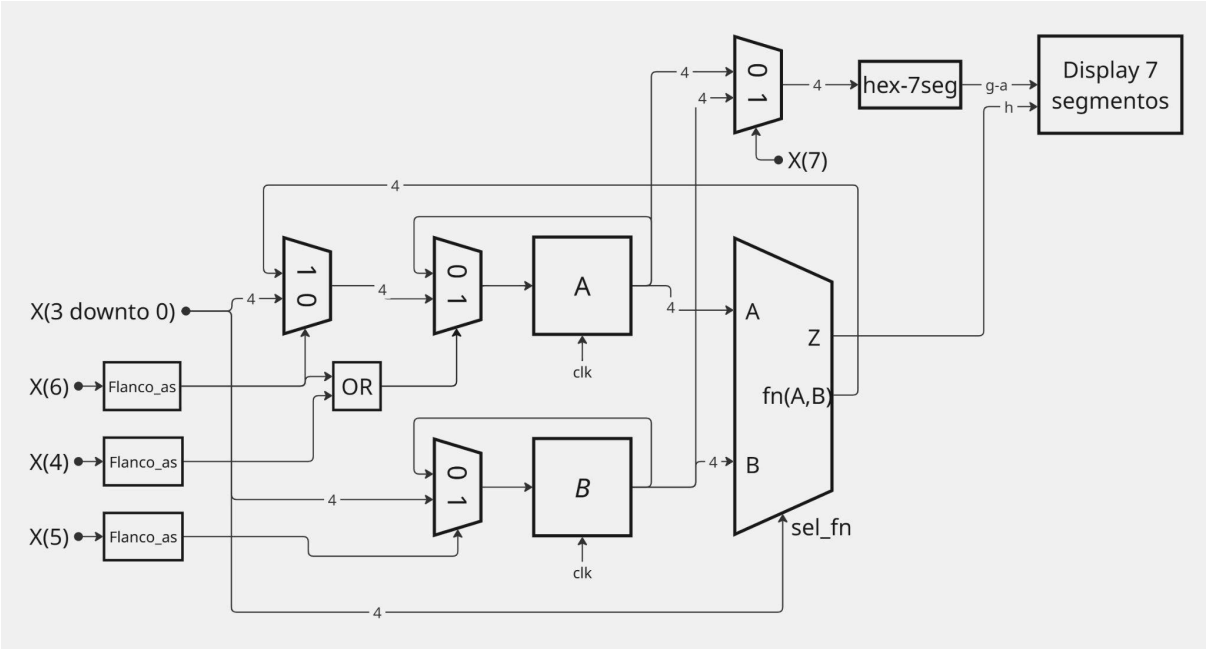

Figura 2.1 - Diagrama de bloques del controlador de la ALU

Se desarrollaron distintos módulos o Máquinas de Estado esclavas para poder hacer el proceso más claro y entendible a la hora del desarrollo.

El primer módulo a realizar fue Detector de Flanco Ascendente, el cual trabaja del siguiente modo:

Tiene una entrada de reloj y una señal(que puede ser ser asincrónica al reloj) y su salida es un pulso sincrónico al reloj a partir de la señal.

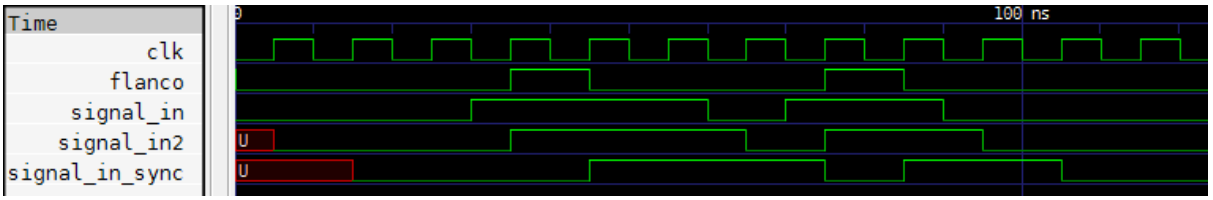

Figura 2.2 - Banco de Pruebas del Detector de Flanco Ascendente

En la figura 2.2 se muestra como una señal(signal_in) ingresa a la máquina, donde se guarda en el siguiente ciclo de reloj (en signal_in2), para luego hacer otra señal en el siguiente ciclo de reloj(signal_in_sync), estas dos últimas son comparadas para disparar el flanco.
El flanco de salida tiene la siguiente lógica:

flanco <= ‘1’ when (signal_in AND not signal_in_sync) else ‘0’;

De este modo el flanco es un pulso que dura un ciclo de reloj(cuando hay un cambio determinado en las dos señales que compara).

Dado que la implementación se realiza sobre una FPGA utilizando switches físicos, la asincronía y el posible rebote de estos interruptores frente a la alta frecuencia del reloj (12 MHz) podría generar múltiples ingresos de datos. Por ello, el Detector de Flanco Ascendente es un módulo muy importante, ya que convierte la activación asincrónica del switch en un pulso sincrónico de un solo ciclo de reloj, garantizando que cada evento de carga sea procesado una sola vez.

Otro módulo utilizado, es el decodificador hexadecimal a 7 segmentos, este módulo fué reutilizado de actividades anteriores.
El mismo consiste en pasar de un vector de 4 dígitos binarios(hexadecimal), a la codificación de 7 segmentos para mejor interpretación de la salida a un display de 7 segmentos.
La codificación es la siguiente:

“gfedcba”
"0111111" - - - - - - - - "0000", - - 0
"0000110" - - - - - - - - "0001", - - 1
"1011011" - - - - - - - - "0010", - - 2
"1001111" - - - - - - - - "0011", - - 3
"1100110" - - - - - - - - "0100", - - 4
"1101101" - - - - - - - - "0101", - - 5
"1111101" - - - - - - - - "0110", - - 6
"0000111" - - - - - - - - "0111", - - 7
"1111111" - - - - - - - - "1000", - - 8
"1101111" - - - - - - - - "1001", - - 9
"1110111" - - - - - - - - "1010", - - A
"1111100" - - - - - - - - "1011", - - b
"0111001" - - - - - - - - "1100", - - C
"1011110" - - - - - - - - "1101", - - d
"1111001" - - - - - - - - "1110", - - E
"1110001" - - - - - - - - “1111”, - - F

La Máquina Principal es la que se muestra en la Figura 1.1 donde la misma implementa el detector de flanco ascendente para algunas entradas, y el decodificador de hexa a 7 segmentos para la salida.
También notar que el Switch X(7) solo controla un multiplexor a la salida, con él podemos decidir que valor ver en el display, el valor actual de A, o el valor de B

La lógica que maneja la máquina principal no es nada compleja a que algunas tareas sencillas pero repetitivas fueron asignadas a otras máquinas por lo que la misma se ve bastante simple y entendible

Resultados

De la implementación se puede concluir que se hizo todo correctamente, luego de terminada la descripción de hardware y verificado que el mismo no tenga problemas de compilación, se lo cargó a la FPGA para probar su funcionamiento.

El mismo tuvo algunos problemas al comienzo pues a la hora de realizar la lógica de final no se la hizo correctamente. Luego de corregir los errores se cargó nuevamente el código, donde esta vez funcionó todo correctamente.

Luego de algunas pruebas verificando que todas las funciones de la ALU respondieron correctamente se dio por concluida la actividad.
Esta última prueba no haría falta, pues al momento de comprobar el funcionamiento de la ALU con el banco de pruebas automático, y no detectar fallos ya se sabía de antemano que la misma funcionaba correctamente.
Pero la prueba ayudó a entender la interfaz utilizada por la máquina, además de reinterpretar los cálculos que la misma hacía antes de obtener los resultados.

Algunos cálculos realizados con la ALU

- Se ingresó 5=’0101’ a A y 3=’0011’ a B, y luego se realizó una suma obteniendo 8=’1000’ en A;
- Con el valor A=8='1000' y un desplazamiento a la izquierda de 3 lugares, se obtuvo nuevamente 8='1000'. Luego, se aplicó un desplazamiento a la derecha lógico de 3 lugares sobre A=8='1000', obteniendo 1='0001'. Al realizar un desplazamiento a la izquierda de 3 lugares sobre A=1='0001', se volvió a 8='1000';
- Luego, haciendo un desplazamiento a la derecha aritmético sobre A=8='1000' se obtuvo F='1111', lo cual demuestra la propagación del bit de signo por el desplazamiento aritmético;
- Se probó la función XOR bit a bit con los valores A=F='1111' y B=3='0011', obteniendo C='1100'. Al cargar este resultado en A (A='1100') y mantener B='0011', una segunda operación XOR arrojó F='1111', confirmando el correcto funcionamiento de las operaciones lógicas bit a bit;
- Se cambió a B por 1=’0001’, y selecciono la suma sin cargar en A el resultado, esto activó el punto del 7 segmentos, demostrando que el resultado de la suma es 0=’0000’;
- Para la resta, se colocó un 6=’0110’ en A y en 13=D=’1101’ en B, obteniendo en el display un 9=‘1001’ o -7 en complemento a 2, recordar que el display de 7 segmentos utilizado no tiene interpretación de numeros negativos, por lo que queda para el usuario;
- Se continuó probando las demás funciones(menor con y sin signo, AND, OR) obteniendo los valores esperados.

De estas pruebas se confirmó el correcto funcionamiento de toda la ALU, desde la lógica misma hasta su interfaz para interpretar y mostrar los resultados obtenidos, obvio con la limitación que nos brinda un display de 7 segmentos.
Todo vale de la interpretación del usuario sobre los resultados que nos brinda el mismo, como por ejemplo al operar con restas, con sumas y su desborde, con las funciones lógicas bit a bit(XOR, OR y AND).

Conclusiones

En conclusión, la ALU desarrollada cumplió con todas las especificaciones, realizando correctamente las operaciones aritméticas y lógicas requeridas. El enfoque modular permite

simplificar el desarrollo y facilitar la comprensión del sistema, mediante la utilización de submódulos específicos para tareas recurrentes.

Las pruebas en simulación y en hardware demostraron la correcta funcionalidad del diseño y la utilidad del detector de flanco ascendente para el manejo de señales asincrónicas. Esta experiencia permitió afianzar los conocimientos sobre diseño digital y sobre la relación entre la descripción en VHDL y su comportamiento a través de una FPGA.