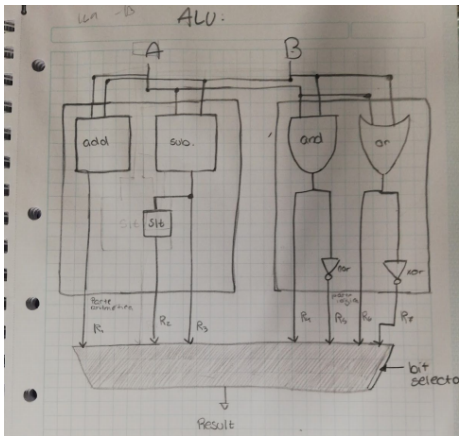
Lab04
Arquitectura de computadoras

Integrantes:
Claudia Noche
Mauricio Bernuy

Fecha: 2 de octubre 2019

1 ALU

En este trabajo se busca realizar un Arithmetic-Logic Unit o ALU en verilog. En la arquitectura Von Neumann, la ALU se encarga de realizar operaciones aritméticas en integars binarios.
En clase, se realizó un diagrama en bloques del ALU, el cual se muestra en la Figura 1. En la Figura 2, se encuetra la tabla que relaciona los códigos con las operaciones



(a) Diagrama de Bloques de un ALU

Figure 1

AluOp	Mnemonic	Result =	Description
0000	add	$A + B$	Addition
0010	sub	$A - B$	Subtraction
0100	and	$AandB$	Logical and
0101	or	$AorB$	Logical or
0110	xor	$AxorB$	Exclusive or
0111	nor	$AnorB$	Logical nor
1010	slt	$(A - B)[31]$	Set less than
Other	n.a.	Don't care	

(a) Tabla

Figure 2

2 Código

2.1 Variables

El código inicia con la implementaicón de un módulo llamado ALU, el cual inicializa las variables que se van a utilizar a lo largo del programa.

```
module ALU (clk, A, B, Opin, result, zero);
```

(a) Módulo principal

Figure 3

Se declara la valriable clk, de tipo input. Esta se usa más adelante en el código para simular un Clock. Luego, se definen los bits con los que se va a trabajar, llamados A y B respectivamente. Ambos son declarados con 32 bits de tamaño.

Como se ve en la figura 2, cada operación viene acompañada de un código de 4 dígitos o bits. Esto se ha tomado como una variable de tipo Input de 4 bits de tamaño, llamada Opin. Con el fin de poder implementar un Case de estos códigos, se requiere volver el valor ingresado en Opin en un registro, para lo cual se crea la variable de tipo Output llamada Opcode, la cual es declarada como registro y toma el valor que contiene Opin.

Se crea una variable de tipo Output que recibe el resultado, llamada Result, la cual también es declarada como registro de 32 bits.

Finalmente, se crea una variable zero, la cual va a representar un binario en 0.

```
input clk;

input[31:0] A;
input[31:0] B;
input[3:0] Opin;
output[31:0] result;
output zero;

reg[3:0] Opcode;
reg[31:0] result;
reg zero;
```

(a) Declaración de variables

Figure 4

2.2 Parámetros

Los parámetros son constantes locales que son definidas dentro de un módulo. Cada parámetro posee un nombre y un valor, el cual no se debe ver alterado por el código. Se definen los siguientes parametros en base a la tabla mostrada

en la figura 2. El parámetro "zeronum" va a ser utilizado en el case como caso "default".

```
// Logic params

parameter AND   = 4'b0100;
parameter OR    = 4'b0101;
parameter NOR   = 4'b0111;
parameter XOR   = 4'b0110;

// Arithmetic params

parameter add   = 4'b0000;
parameter sub   = 4'b0010;
parameter slt   = 4'b1010;

// Zero params
parameter zeronum = 32'b0;
```

(a) Declaración de parametros

Figure 5

2.3 Lógica del ALU

La sección de lógica del ALU se separa en dos partes principales. En la primera parte, mostrada en la Figura 6, se define el comportamiento de la salida Zero, la cual muestra un Logic-1 si es que todos los valores de result resultan ser cero. En cualquier otro caso la salida es de 0.

```
// ALU Logic
always @(posedge clk or negedge clk)
    begin
        if (result == zeronum)  zero = 1'b1;
        else                    zero = 1'b0;
    end
```

Figure 6

La segunda parte, Figura 7, muestra la implementación de un case sobre el Opcode y los parámetros mostrados en la sección previa. Previo a este case, se asigna el valor de la entrada Opin al registro Opcode. Si el Opcode ingresado es igual a uno de los valores de los parámetros, se ejecuta una operación con las variables A y B y se asigna el resultado a la variable result. Cabe resaltar que se tiene un caso "default", el cual es un bit de 0s.

```
always @(posedge clk)
    begin
        Opcode <= Opin;
        case (Opcode)
            AND:    result <= A & B;
            OR:     result <= A | B;
            NOR:    result <= A ~| B;
            XOR:    result <= A ^ B;
            add:    result <= A + B;
            sub:    result <= A - B;

            slt:    result = (A<B) ? 1'b1 : 1'b0;

            default: result <= zeronum;
        endcase
    end

endmodule
```

(a) Implementación del Case

Figure 7

3 Test bench

El test bench de este sistema se inicia declarando el módulo del código, el cual no inicializa ninguna variable. Se incluye el módulo anterior en la línea include "ALU.v", y se delcaran las variables del módulo a probar. El test bench es un código autómana, es decir, no requiere variables de tipo input o output, pues funciona dentro de si mismo. Por esta razón, las variables son declaradas como wires y registros, con sus tamaños respectivos.

```
`timescale 1s/1ns
`include "ALU.v"

module ALU_tb;
//var declaration
    reg clk;
    reg[31:0] A;
    reg[31:0] B;
    reg[3:0] Opin;

    wire[31:0] result;
    wire zero;
```

(a) Declaración de variables

Figure 8

Luego, se deben inicializar los parámetros utilizados en el módulo del ALU, pues al son variables locales.

```
// Logic params

parameter AND   = 4'b0100;
parameter OR    = 4'b0101;
parameter NOR   = 4'b0111;
parameter XOR   = 4'b0110;

// Arithmetic params

parameter add   = 4'b0000;
parameter sub   = 4'b0010;
parameter slt   = 4'b1010;
```

(a) Declaración de parámetros

Figure 9

En la sección siguiente, se inicia la formación de ondas. La instrucción "dumpfile" se encarga de mandar todos los cambios en las variables a un file llamado, en este caso, ALU.vcd. Con "dumpvars", parametrizada con 0 y el nombre del archivo de del test bench, manda todas las variables en el módulo a cualquier otro módulo que haya sido instanciado dentro de este. Esta práctica asegura el funcionamiento del código y crea un archivo VCD, el cual servirá para graficar el código. Asimismo, se inicializa el clock del sistema para que cree ondas de un intervalo de 1s.

```
//device under test
ALU dut(clk, A, B, Opin, result, zero);

initial begin
    $dumpfile("ALU.vcd");
    $dumpvars(0, ALU_tb);
//1s clock
    clk = 0;
    forever #0.5 clk=~clk;
end
```

Figure 10

Finalmente, se inicializan las variables A y B con dos números distintos y se prueban los casos del ALU con los parámetros definidos, con intervalos de tiempo entre ellos.

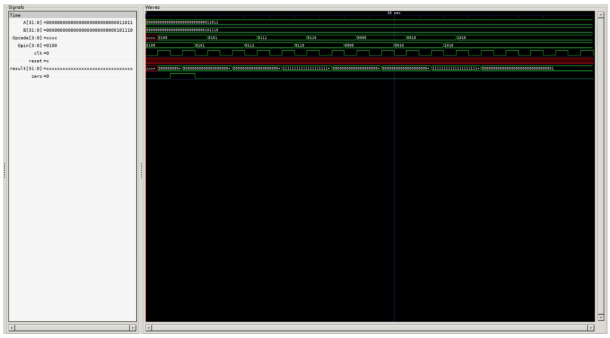
```
initial begin
//inputs
    A = 32'b00000000000000000000000000011011;
    B = 32'b00000000000000000000000000101110;
end

initial begin
//Op inputs
    Opin = AND; #2
    Opin = OR; #2
    Opin = NOR; #2
    Opin = XOR; #2
    Opin = add; #2
    Opin = sub; #2
    Opin = slt; #2
    #4
    $finish;
end

endmodule
```

(a) Prueba de códigos

Figure 11

4 VCD

Finalmente, se generó un VCD con el Test Bench para graficar las waveforms. El resutlado, graficado en Gtkwave, se muestra en la Figura 12.

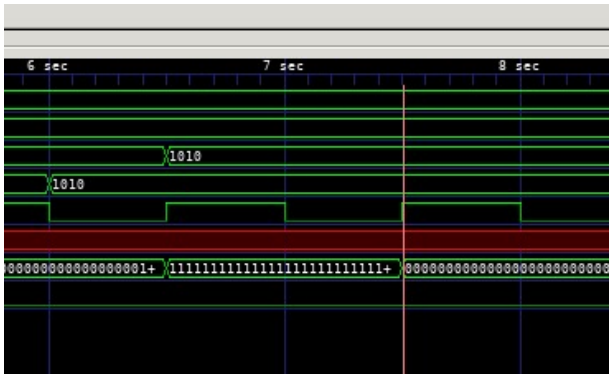


(a) Gráfica

Figure 12

5 Tiempo de Ejecución

En caso de que se puebe un nuevo Opin cada segundo, se tendría que considerar cuanto es el delay entre las operaciones sobre los registros. La asignación de Opin a Opcode demora un posedge, y la ejecucion del case demora otro posedge. Por lo tanto, debido a que el clock empieza en 0, este caso acabaría en 7.5s como se observa en la figura 13.



(a) Fin del VCD

Figure 13