UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

ARQUITECTURA DE COMPUTADORAS

TRABAJO PRÁCTICO N° 1

Apellido	Nombre	Matrícula
Guglielmotti	Bruno	43474558
Rodríguez	Franco Aníbal	42994188

Índice

1.Introducción

En este laboratorio se hará el desarrollo de un código en Verilog con el fin de sintetizar una FPGA Diligent Basys3 para que implemente una Unidad Aritmética Lógica (ALU) permitiendo introducir los dos operandos y el operador mediante el uso de los switches y botones, y obtener el resultado de las operaciones a través de los leds.

2.Desarrollo

Para el correcto funcionamiento del programa y para una interacción más sencilla, se implementó una interfaz de control, luego la ALU en sí y el TOP para interconectar la interfaz con la ALU.

Por otro lado, como partes necesarias, se implementó un testbench sobre el TOP y el constraint.

2.1.Interfaz

El módulo denominado *interface* proporciona una estructura que permite la interacción con el FPGA a través de una serie de entradas y salidas que corresponden a operandos, operadores, y señales de control. A continuación, se detalla la estructura y funcionamiento de esta interfaz.

2.1.1.Parámetros

El módulo contiene dos parámetros principales:

- *NB_OP*: Define el tamaño (en bits) del operador, en este caso 6 bits.
- *NB_DATA*: Define el tamaño (en bits) de los operandos, configurado para 8 bits.

2.1.2.Entradas

Las entradas están compuestas por señales que provienen de interruptores, botones y del reloj del sistema:

- *i_switches*: Un bus de 8 bits que se utiliza para proporcionar los valores de los operandos y/o del operador.
- *i_btn_set_operand1*: Botón que indica cuándo los interruptores (*i_switches*) deben asignar su valor al primer operando.
- *i_btn_set_operand2*: Botón que asigna el valor de los interruptores al segundo operando.
- *i_btn_set_operator*: Botón que asigna los 6 bits menos significativos de los interruptores al operador.
- *i_clk*: Señal de reloj que sincroniza las operaciones.

- *i_reset*: Señal de reset para restablecer los valores de los operandos y del operador a cero.

2.1.3.Salidas

- *o_operand1*: Primer operando, de 8 bits, utilizado como entrada en la ALU.
- *o_operand2*: Segundo operando, también de 8 bits.
- *o_operator*: Operador de 6 bits que se selecciona a partir de los bits menos significativos de los interruptores.

2.1.4.Asignación de los Operandos

Si se pulsa el botón correspondiente a i*_btn_set_operand1*, el valor actual de los interruptores se asigna al registro *operand1_reg*.

De manera similar, al pulsar el botón *i_btn_set_operand2*, el valor de los interruptores se asigna a *operand2_reg*.

2.1.5.Asignación del Operador

El operador toma los bits más bajos de los interruptores (definidos por NB_OP) cuando se pulsa *i_btn_set_operator*, y se almacena en *operator_reg*.

2.1.6.Sincronización con el reloj y reset

Todas las operaciones son controladas por el flanco positivo del reloj (posedge *i_clk*).

Cuando la señal de *i_reset* está activa, los registros de los operandos y del operador se inicializan a cero, garantizando un estado conocido al inicio de la operación o tras un reinicio del sistema.

2.1.7.Asignación de las Salidas

Finalmente, los registros internos (*operand1_reg*, *operand2_reg*, *operator_reg*) se asignan a las salidas correspondientes del módulo (*o_operand1*, *o_operand2*, *o_operator*), de manera que los datos procesados internamente son accesibles para la ALU.

2.2.ALU (Unidad Aritmética Lógica)

En este diseño, la ALU opera con operandos de 8 bits y tiene un conjunto de operaciones controladas por un código de operación de 6 bits.

2.2.1. Parámetros

El módulo tiene tres parámetros configurables:

- NB_OP: Número de bits del código de operación (*i_opcode*), definido como 6 bits.
- NB_DATA: Tamaño de los operandos, que en este caso es de 8 bits.
- NB_OUT: Tamaño de la salida, que es de 16 bits para permitir operaciones aritméticas que puedan resultar en un valor mayor que el tamaño de los operandos.

2.2.2. Entradas

El módulo recibe las siguientes entradas:

- *i_operand1*: Primer operando, de 8 bits, que puede ser positivo o negativo, ya que está definido como un número con signo.
- *i_operand2*: Segundo operando, también de 8 bits y con signo.
- *i_opcode*: Código de operación de 6 bits, que define la operación que se realizará sobre los operandos.

2.2.3. Salida

- *o_result*: Resultado de la operación, es una señal de salida de 16 bits con signo, que contiene el valor calculado según la operación seleccionada.

2.2.4. Estructura Interna y Funcionamiento

La ALU implementa varias operaciones básicas tanto aritméticas como lógicas, dependiendo del valor del código de operación (*i_opcode*). Estas operaciones se definen dentro de un bloque always @(*), lo que significa que el resultado se actualiza de forma continua y combinacional, sin necesidad de estar sincronizado con el reloj.

2.2.5. Operaciones Soportadas

Las siguientes operaciones se realizan dependiendo del valor de *i_opcode*:

- **Suma (ADD)**: Cuando *i_opcode* es 6'b100000, la ALU suma los operandos (*i_operand1* + *i_operand2*).
- **Resta (SUB)**: Para 6'b100010, la ALU resta el segundo operando del primero (*i_operand1* - *i_operand2*).
- **AND**: Para 6'b100100, realiza una operación bit a bit AND entre los operandos (*i_operand1* & *i_operand2*).
- **OR**: Para 6'b100101, realiza una operación bit a bit OR (*i_operand1* | *i_operand2*).

- **XOR**: Para 6'b100110, ejecuta una operación bit a bit XOR (*i_operand1* ^ *i_operand2*).
- **Shift Arithmetic Right (SRA)**: Para 6'b000011, realiza un desplazamiento aritmético a la derecha sobre el primer operando por el número de posiciones indicado por el segundo operando (*i_operand1* >>> *i_operand2*).
- **Shift Logical Right (SRL)**: Para 6'b000010, realiza un desplazamiento lógico a la derecha sobre el primer operando (*i_operand1* >> *i_operand2*).
- **NOR**: Cuando i_opcode es 6'b100111, realiza una operación bit a bit NOR (~(*i_operand1* | *i_operand2*)).

2.2.5.1.Caso por defecto

Si el código de operación no coincide con ninguno de los valores especificados, la salida por defecto de la ALU es cero (*o_result* = 8'b00000000), garantizando un comportamiento controlado incluso para códigos no válidos.

2.3.Top

El módulo *top_alu_interface* actúa como la unidad superior del sistema, integrando la interfaz de usuario (controlada mediante interruptores y botones) con la Unidad Aritmético-Lógica (ALU), y mostrando el resultado en una matriz de LEDs. Este diseño conecta todos los componentes necesarios para realizar operaciones aritméticas y lógicas, utilizando operandos introducidos por el usuario a través de los interruptores.

2.3.1.Parámetros

El módulo superior hereda los parámetros principales del sistema:

- **NB_OP**: Define el número de bits del código de operación, que en este caso es de 6 bits.
- **NB_DATA**: Define el tamaño de los operandos (8 bits en este caso).
- **NB_OUT**: Tamaño de la salida, que es de 16 bits para manejar los posibles resultados de las operaciones aritméticas y lógicas.

2.3.2.Entradas

El sistema recibe varias entradas del usuario:

- *i_switches*: Un bus de 8 bits para introducir los operandos o el operador a través de interruptores.
- *i_btn_set_operand1*: Botón para asignar el valor de los interruptores al primer operando.
- *i_btn_set_operand2*: Botón para asignar el valor de los interruptores al segundo operando.
- *i_btn_set_operator*: Botón para asignar el valor de los interruptores al código de operación.
- *i_clk*: Señal de reloj que sincroniza el funcionamiento del sistema.

- *i_reset*: Señal de reset para reiniciar el sistema y establecer los valores iniciales.

2.3.3. Salidas

- *o_leds*: Señal de salida que conecta el resultado de la operación de la ALU a los LEDs, permitiendo una visualización directa del resultado.

2.3.4. Estructura Interna y Funcionamiento

El módulo superior interconecta dos módulos esenciales:

- **Interfaz**: El módulo de interface toma las entradas del usuario (interruptores y botones) y asigna los valores a los operandos y al código de operación, que luego son enviados a la ALU para realizar la operación correspondiente.

- **ALU**: El módulo de la ALU toma los operandos generados por la interfaz y, según el código de operación recibido, realiza la operación aritmética o lógica. El resultado de esta operación se envía a los LEDs.

2.3.5. Conexiones Internas

- **Operandos y Operador**: Los operandos (o_*operand1* y o_*operand2*) y el operador (o_*operator*) generados por el módulo de la interfaz se conectan directamente a las entradas correspondientes de la ALU.

- **Resultado de la ALU**: La salida de la ALU (*o_result*) se conecta a la salida del sistema (*o_leds*), de modo que el resultado de las operaciones se refleja en los LEDs.

2.3.6. Instanciación de Módulos

- **Interfaz**: El módulo interface se instancia bajo el nombre *u_interface*. Este módulo toma las entradas del usuario, realiza la lógica de selección de operandos y operador, y las asigna a los correspondientes cables internos (*operand1*, *operand2*, *operator*).

```
interface #(
    .NB_OP(NB_OP),
    .NB_DATA(NB_DATA)
) u_interface (
    .i_switches(i_switches),
    .i_btn_set_operand1(i_btn_set_operand1),
    .i_btn_set_operand2(i_btn_set_operand2),
    .i_btn_set_operator(i_btn_set_operator),
    .i_clk(i_clk),
    .i_reset(i_reset),
    .o_operand1(operand1),
```

```
    .o_operand2(operand2),
    .o_operator(operator)
);
```

- **ALU**: El módulo ALU se instancia como *u_alu*. Este módulo toma los operandos y el código de operación proporcionados por la interfaz, realiza la operación correspondiente y entrega el resultado en el cable *result*.

```
ALU #(
       .NB_OP(NB_OP),
       .NB_DATA(NB_DATA),
       .NB_OUT(NB_OUT)
   ) u_alu (
       .i_operand1(operand1),
       .i_operand2(operand2),
       .i_opcode(operator),
       .o_result(result)
   );
```

2.3.7.Asignación de salidas

El resultado generado por la ALU, que está disponible en *result*, se asigna directamente a los LEDs. Esto permite al usuario visualizar el resultado de la operación en tiempo real.

```
assign o_leds = result;
```

2.4.Testbench

El testbench para el módulo *top_alu_interface* se diseñó para verificar el correcto funcionamiento del sistema que integra la interfaz de usuario y la Unidad Aritmético-Lógica (ALU). A través de diferentes casos de prueba, se evalúa cómo el módulo reacciona al cambiar los operandos y el código de operación. A continuación se ofrece una descripción general de su estructura y funcionamiento:

2.4.1.Parámetros y Señales

Se definen los parámetros *NB_OP*, *NB_DATA*, y *NB_OUT* para especificar el tamaño del código de operación, operandos y salida, respectivamente. Las señales *switches*, *btn_set_operand1*, *btn_set_operand2*, *btn_set_operator*, *clk* y *reset* son las entradas que controlan la interfaz y la ALU. La salida del sistema se conecta a *leds*.

2.4.2.Instancia del Módulo

El testbench instancia el módulo *top_alu_interface*, llamado dut (Device Under Test), el cual recibe las señales de entrada generadas por el testbench.

2.4.3. Generación del Reloj (Clock)

Un proceso initial genera la señal de reloj (clk) que alterna cada 50 unidades de tiempo. Esto simula el reloj del sistema que sincroniza las operaciones.

2.4.4. Inicialización y Reset

Al inicio de la simulación, todas las señales se inicializan a cero. Luego se aplica una señal de reset (reset) para reiniciar el sistema y garantizar que comienza desde un estado conocido.

2.4.5. Secuencia de Prueba

El testbench realiza las siguientes operaciones de prueba:

- **Asignación de Operandos y Operador**: Se establece el primer operando a través de los interruptores y se activa el botón correspondiente para cargar el valor en el sistema. El proceso se repite para el segundo operando y el código de operación, que corresponde a una suma (6'b100000).
- **Verificación del Resultado**: Después de asignar los operandos y el operador, el testbench espera para verificar el resultado. Posteriormente, se cambia el valor del primer operando y se comprueba que el resultado también cambia, validando que el sistema reacciona correctamente a los cambios en los datos de entrada.

2.4.6. Monitorización

Un bloque initial con un comando *$monitor* se utiliza para observar los valores de los operandos, el código de operación y el resultado en tiempo real durante la simulación, proporcionando una visión detallada del comportamiento del sistema.