DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2024)

Actividad 2

Requisitos

- Esta actividad es en **pareja** y se debe realizar de manera **presencial**.
- Su desarrollo es en **laboratorio** y deberá ser terminada dentro del horario.
- Deberá ser realizada en VHDL.

Objetivos

- Implementar la lógica de ALU que usará con su grupo de proyecto.
- Leer tablas de valores de un componente y poder implementarlos y/o usarlos.
- Profundizar las nociones de componentes que sirven como entradas y salidas.
- Construir una calculadora con la capacidad de acumular sus resultados.

Materiales

- Computador con Vivado WebPack instalado.
- Proyecto base de Vivado, con las señales definidas y *constraints* configurados.
- Archivos .vhd con componentes para la actividad.
- Una placa Basys3.

Evaluación

La actividad será evaluada como lograda o no lograda. Se considera lograda cuando su descripción e implementación del *hardware* concuerda al menos con la funcionalidad descrita en los diagramas. Se podrá retirar tan pronto como finalice la actividad.

Descripción

La lógica de la ALU de su proyecto requiere, de acuerdo a su tabla de valores, poder controlar el flujo de los datos para cada operación, tanto para el resultado como para las distintas *flags*. Para seleccionar el flujo de datos, podemos usar componentes como *muxers*, *demuxers*, *encoders* y *decoders*. Estos pueden expresar en VHDL con una instancia de una o más expresiones con **`with/select`**:

```
-- Muxer
 with mux_select select
    mux_out <= mux_in_1 when "00",
               mux_in_2 when "01",
               mux_in_3 when "10",
               mux_in_4 when "11";
-- Demuxer
 with demux_select select
    demux_out_1 <= demux_in when "00",
                   '0'      when others;
 with demux_select select
    demux_out_2 <= demux_in when "01",
                   '0'      when others;
 with demux_select select
    demux_out_3 <= demux_in when "10",
                   '0'      when others;
 with demux_select select
    demux_out_4 <= demux_in when "11",
                   '0'      when others;
-- Decoder
 with decoder_in select
    decoder_out <= "0001" when "00",
                   "0010" when "01",
                   "0100" when "10",
                   "1000" when "11";
-- Encoder
 with endecoder_in select
    encoder_out <= "00" when "0001",
                   "01" when "0010",
                   "10" when "0100",
                   "11" when "1000";
                   "00" when others;
```

Antes de construir una calculadora que pueda acumular su resultado con esta ALU, es necesario extender sus conocimientos de componentes de entradas y salidas:

Entradas y salidas:

Display

Las salidas de un circuito que representan un número binario usando LEDs no escalan bien. Una alternativa es ordenar los LEDs para que puedan formar figuras que tengan un significado para nosotros, por ejemplo, ordenar 7 LEDs para que puedan representar los segmentos de un dígito hexadecimal arbitrario. Esto mejora la legibilidad de la información, pero aumenta las señales necesarias. Sin embargo, ya que la vista tiene persistencia, podríamos rotar rápidamente entre distintos dígitos, entregándonos la ilusión de que funcionan simultáneamente y, con eso, efectivamente usar menos señales. Este componente lo llamaremos *display controller* y para poder hacer la rotación, recibirá una señal de un *clock* con una frecuencia conocida. A continuación, se detallará el comportamiento de dicho componente:

- `Display_Controller.vhd`

  Este módulo recibe cuatro vectores y un valor lógico: `dis_a[3:0]`, `dis_b[3:0]`, `dis_c[3:0]`, `dis_d[3:0]` y `clk`. Entrega como salida los vectores `seg[7:0]` y `an[3:0]`.

  Su función es decodificar los valores que quieres entregar en los *display* de 7 segmentos correspondientes:

  Para que este módulo funcione, debe conectar la señal de entrada de la placa Basys3 `clk` a la señal de entrada `clk` de este módulo.

  La salida de este módulo genera las señales de salida de la placa Basys3: `seg[7:0]` y `an[3:0]` para encender el *display*.

Botones

Por otra parte, las entradas de un circuito pueden no ser lo suficientemente estables. Por ejemplo, al apretar un botón, la señal tiende a rebotar antes de estabilizarse en su nuevo valor.

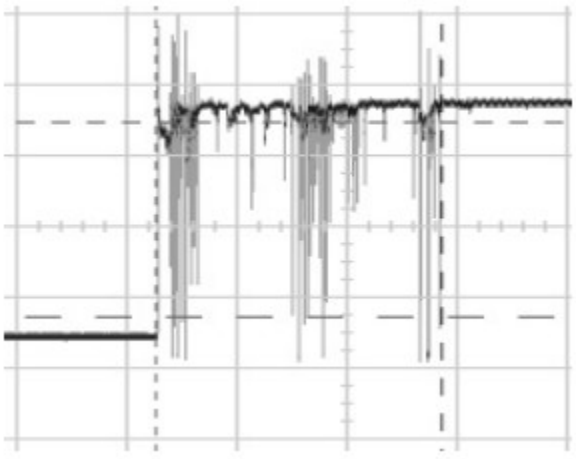

Figura 1: Señal al pulsar un botón

Una solución es tener un circuito que no transfiera la señal hasta que esta sea estable por un periodo de tiempo. A este componente lo llamaremos *debouncer* y, para permitirle medir el tiempo, le daremos una señal de un *clock* con una frecuencia conocida.

De este modo, y usando algunos *muxers*, podemos elegir mostrar los operandos o el resultado en los *display* de 7 segmentos. Para almacenar los operandos, podemos usar registros/contadores de uso general y, por medio un *muxer*, elegir si cargarlos con el resultado de la ALU o la entrada de los interruptores. Como tenemos señales de los botones con anti-rebote, podemos usarlas para sincronizar la carga de cada registro a un botón.

Tareas

- **Copiar e importar** los componentes al proyecto base (Cápsula).
- Incorporar el Sumador de 8-bits a la ALU para realizar las operaciones aritméticas (Cápsula).
- Completar las operaciones lógicas y las *flags* según la tabla de valores de la ALU.
- Modificar la asignación de señales entre los componentes en Basys3 para que concuerde con el diagrama.
- Probar la implementación de su ALU cargando el binario en la FPGA.
- Guardar lo realizado en su correo, ya que **utilizará este trabajo más adelante**.

Tablas de Valores

A continuación, se mostrará la tabla de verdad de los componentes de esta actividad.

- `ALU`

Entradas			Salidas			
a(7:0)	b(7:0)	sop(2:0)	result(7:0)	c	z	n
*	*	add	a + b	result < a + b	result = 0	0
*	*	sub	a - b	a >= b	result = 0	a < b
*	*	and	a and b	0	result = 0	0
*	*	or	a or b	0	result = 0	0
*	*	xor	a xor b	0	result = 0	0
*	*	not	not a	0	result = 0	0
*	*	shr	`0'& a(7 downto 1)	a(0)	result = 0	0
*	*	shl	a(6 downto 0) & `0'	a(7)	result = 0	0

Figura 2: Tabla de verdad de la ALU.

- `Reg`

Entradas						Salidas
`clock`	`clear`	`load`	`up`	`down`	`datain(7:0)`	`dataout(7:0)`
*	1	*	*	*	*	"00000000"
Flanco Subida	0	1	*	*	*	`datain`
Flanco Subida	0	0	1	*	*	`dataout + 1`
Flanco Subida	0	0	0	1	*	`dataout - 1`
*	*	*	*	*	*	`dataout`

Figura 3: Tabla de verdad de `Reg`.

Diagramas

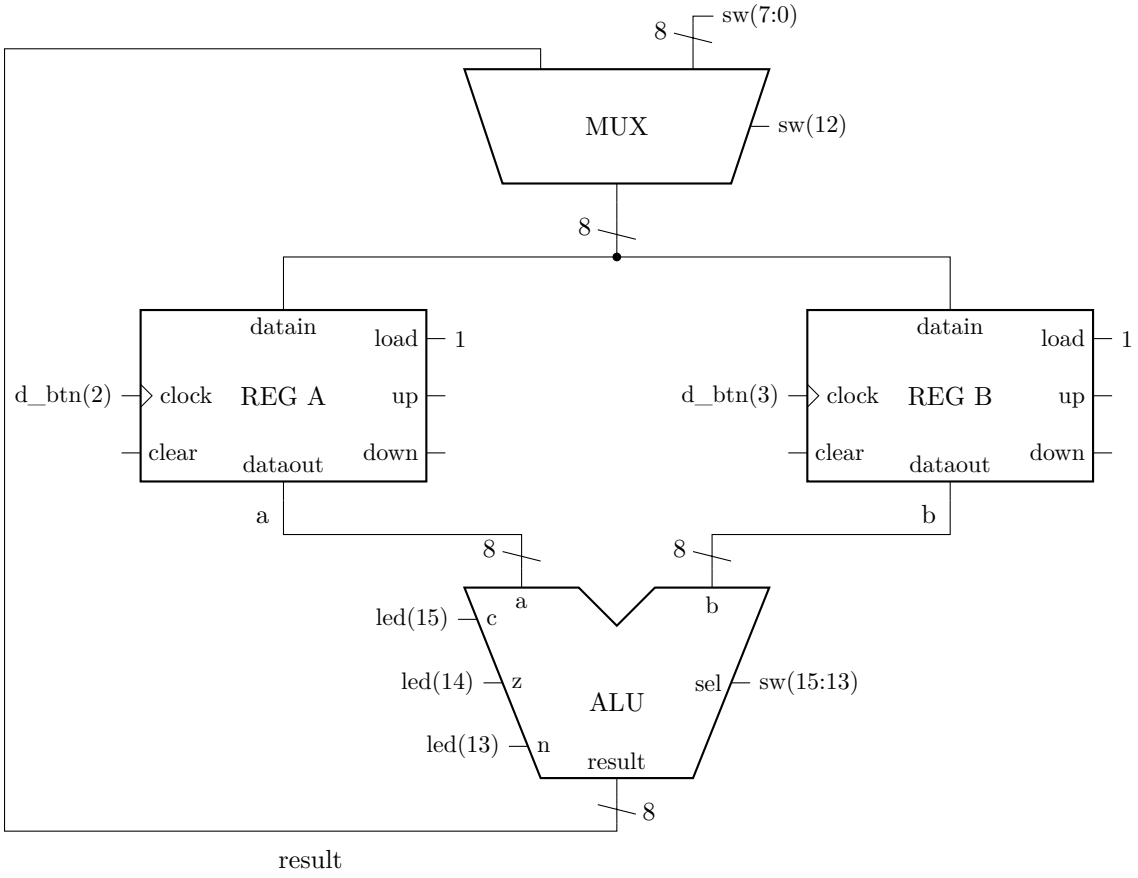

Figura 4: Diagrama de la calculadora con acumulación de resultado.

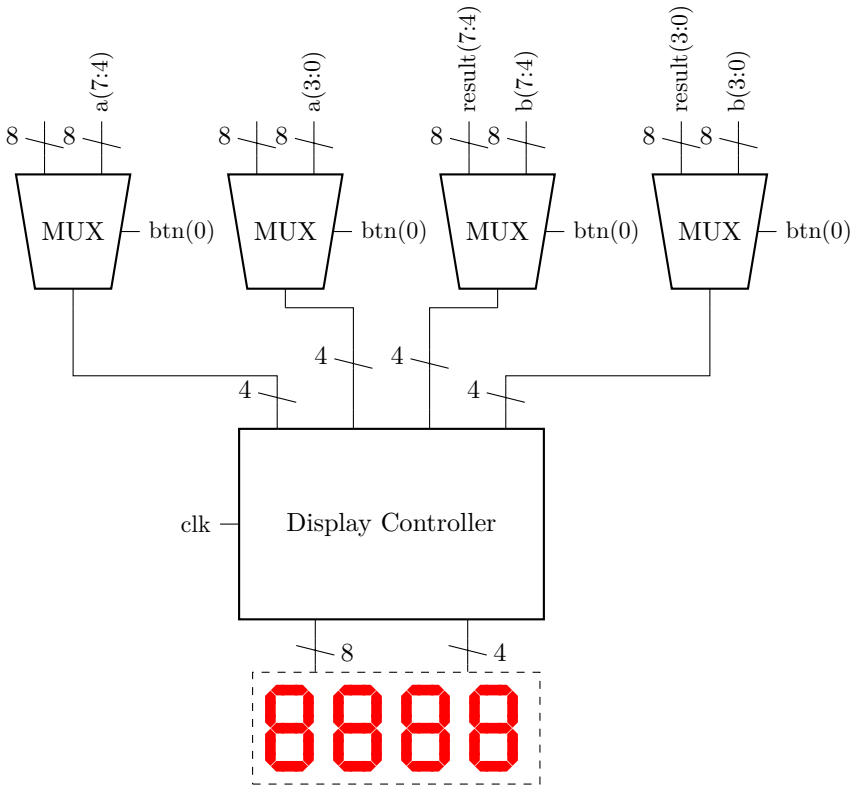

Figura 5: Diagrama de las conexiones de los *display* de 7 segmentos.