Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2463 Sistema electrónicos programables

Ayudantia 03 2S23

Logica Combinacional y Secuencial

Ayudante: Cristobal Vasquez cristobal.vasquez@uc.cl
Prof. Dr.-Ing. Félix Rojas - felix.rojas@uc.cl

---

1. Objetivo de la Ayudantía

- Ejercitar los conceptos y uso de lógica secuencial y combinacional en VHDL.
- Comprender los componentes principales de un procesador y su función.
- Ejercitar el uso de perféricos (IP Cores) en un *Block Design* en Vivado para el desarrollo de un procesador simple.
- Familiarizarse con el uso y desarrollo de *testbenches* para verificar el funcionamiento de módulos.

2. Actividades Previas a la Ayudantía

En el video de la ayudantía se explican y/o desarollan de forma breve códigos para 5 módulos: RAM, ALU, Control Unit, Memoria de instrucciones y Memoria de programa, los cuales se empaquetan como IP Cores. Finalmente se construye un Block Design en Vivado con los módulos nombrados para replicar la arquitectura de un procesador simple, y tras la generación del bitstream se carga el diseño a la Zybo.

- Incluir en el *IP Repository* los IP Cores asociados a RAM, ALU, Control Unit, Memoria de instrucciones y Memoria de programa. Esto se logra añadiendo el *path* relativo a la carpeta donde se encuentran los IP Cores.
- Crear un *Block Design* donde se conectarán los módulos RAM, ALU, Control Unit, Memoria de instrucciones y Memoria de programa de acuerdo a lo visto en el video de la ayudantía.

- Incluir archivo *Constraints* asociado a la tarjeta y editarlo de acuerdo al *Block Design* desarrollado.
- Generar el *HDL Wrapper* del *Block Design* y luego el *bitstream*, para cargarlo a su tarjeta Zybo y probar su funcionamiento.

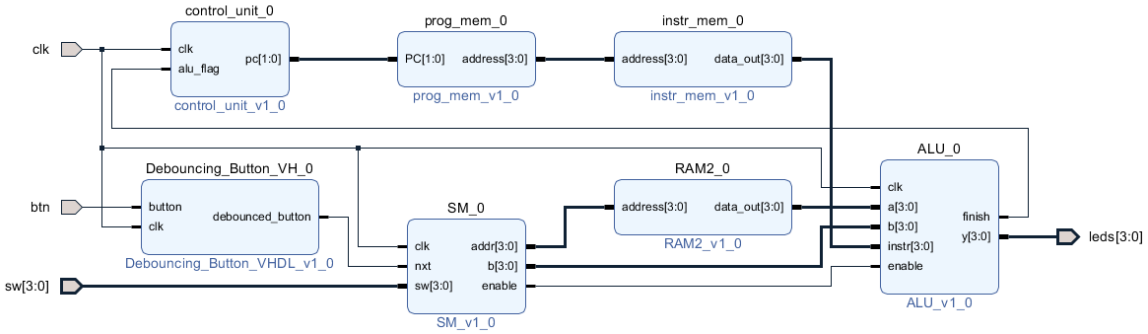

Figura 1: *Block Design* en Vivado del proyecto completo

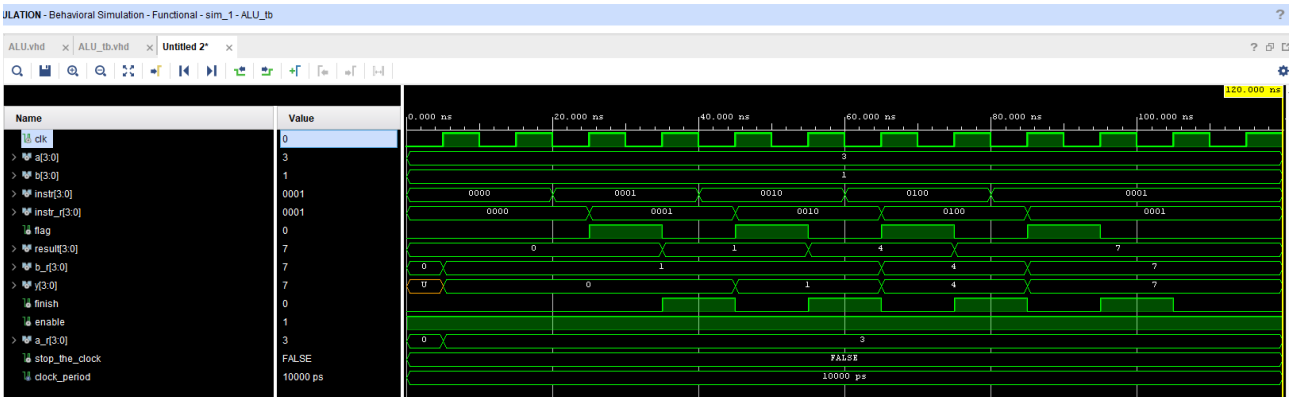

Figura 2: *Testbench* en Vivado del funcionamiento de la ALU

Código	Operación
0000	y <= a
0001	y <= b
0010	y <= a+1
0011	y <= b-1
0100	y <= a+b
0101	y <= a-b
0110	y <= a*b
0111	y <= a/b
1000	y <=NOT a
1001	y <= a AND b
1010	y <= a OR b
1011	y <= a NAND b
1100	y <= a NOR b
1101	y <= a XOR b
1110	y <= sl a
1111	y <= rr a

Figura 3: Tabla instrucciones ALU

3. Actividades Durante la Ayudantía

El ejercicio propuesto para esta ayudantía consiste en editar el módulo *control unit* para generar mediante una combinación de lógica secuencial y combinacional una interacción en la cual mediante los botones, se pueda modificar la frecuencia en la que se muestran los leds con las operaciones de la ALU. A continuación se presentará una serie de pasos a seguir para el desarrollo de esta actividad.

- Crear las entradas btn1, btn2 y btn3 en el bloque de control unit.
- La señal de *clk counter*, elimínela de las señales, e implémentela como variable dentro del primer process. Defínala como integer (sin rango) y no la inicialice en ningún valor.
- Crear señales intermedias de *btn1 anterior*, *btn2 anterior*, *btn3 anterior*. Y crear señales que sirvan de flag para identificar cuando se prende un botón. Ej: flagbtn1, flagbtn2, flagbtn3. Luego, crear señal *std logic vector* que contenga las señales de flag. Ej: si flagbtn1 = 0, flagbtn2 = 1, flagbtn3 = 0, entonces vector = (010).

```
signal btn1_anterior : std_logic := '0';
signal flag_1 : std_logic := '0';
signal btn2_anterior : std_logic := '0';
signal flag_2 : std_logic := '0';
signal btn3_anterior : std_logic := '0';
signal flag_3 : std_logic := '0';

signal vector: std_logic_vector(2 downto 0) := (flag_1, flag_2, flag_3);
```

Figura 4: Imágen de referencia señales

- Crear un *process* que identifique cuando se presiona un botón, y al reconocerlo, este suba la flag respectiva al botón presionado, y baje las demas. Ej: Se presiona el btn1, se levanta la flagbtn1, y se baja la flagbtn2, y flagbtn3. Al final de este process, también se debe asignar los valores de *btn1 anterior, btn2 anterior, btn3 anterior* a sus respectivos botónes.

  Hint: Guíese según el checkeo de la señal ***"alu flag"*** *que también se trata en este IPcore*

- Crear un *process* que actualice el valor del *std logic vector* que contiene los valores de las flag. Esto se debe actualizar en cada flanco de subida de clock.
- Finalmente, fuera de cualquier process, utilice lógica combinacional para actualizar el valor de *max count* y le asigne un valor arbitrario para cada estado del *std logic vector* (100, 010, 001). Ej: Para 100, *max count* = 125000000. Para 010 *max count* = 250000000, etc.

- Re-empaquete el IPcore, actualícelo en el *Block Diagram*, agregue un Debouncer a cada btn, y agregue los botones a las constraints.

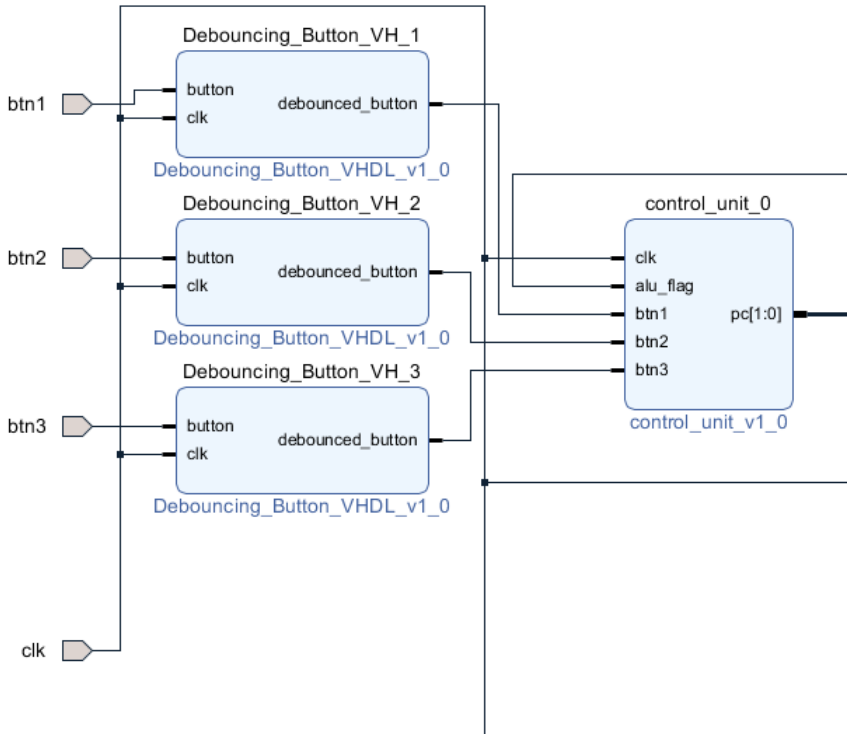

Figura 5: Diagrama de bloques de referencia

- Corra el bitstream y pruébelo en la tarjeta.