Práctica 3

Simón Sánchez Rúa

Juan Camilo Arteaga Ibarra

Grupo 8

José Edinson Aedo Cobo

Universidad de Antioquia

Facultad de ingeniería

Medellín

2022

Objetivos:

- Entender y manejar el software Vivado para el diseño de circuitos digitales mediante el uso de VHDL.
- Afianzar el conocimiento obtenido durante las sesiones teóricas sobre el uso y el funcionamiento de diferentes dispositivos electrónicos como los Flip Flops, las ROMs, los Mux, decodificadores y displays siete segmentos.
- Reforzar el manejo del reloj de la FPGA u otros dispositivos electrónicos, para la sincronización de señales de activación de procesos.

Materiales:

- Software Vivado.
- FPGAS.

Descripción:

Realizar el código del circuito de planteado en la guía de la práctica de laboratorio. Teniendo en cuenta las ROMs y las operaciones de la ALU propuestas. Además, haciendo uso del clock de la FPGA, desarrollar un divisor para el reloj, permitiendo obtener pulsos con menor frecuencia para la activación de los componentes electrónicos planteados para el diseño.

Práctica No. 3: diseño e implementación de circuitos combinacionales modulares.

Objetivos:

- Diseñar circuitos con circuitos combinacionales y registradores constituidos de múltiples módulos (Mux, memorias ROM, decodificadores, ALU, etc.).
- Adquirir habilidades en el modelado genérico (parametrizables) con VHDL de circuitos digitales.
- Adquirir habilidades en el diseño del Test Bench con VHDL.
- Utilizar los displays de 7 segmentos para desplegar información.

DESCRIPCIÓN

En este laboratorio se suministrará a cada Equipo un problema que deberá implementar en la FPGA (Xilinx Artix-7 XC7A35T-ICPG236C) de la tarjeta Basys3. El diseño debe implementarse de forma modular y debe ser parametrizado (use generics) donde sea posible. Se pueden utilizar en la implementación las diferentes estrategias de modelado (comportamental, flujo de datos, estructural). Los Test Bench para la simulación del circuito deben permitir una verificación completa del mismo Para esto debe generarse los vectores de prueba y realizarse la verificación automática mediante un módulo verificador.

Procedimiento:

1. Construya un esquema de su circuito y un plan para implementar cada módulo de su circuito: Mux, decodificadores, ALU, memorias ROM, etc. (qué instrucciones VHDL va a usar para cada módulo).

2. Use donde sea posible la instrucción “generic” en la entidad.

3. Planee una estrategia para automatizar las pruebas por simulación. Debe considerar el tamaño de puertos de entrada y salida para definir el número de vectores de prueba necesarios, que permitan verificar completamente su circuito.

4. Modele su circuito con VHDL, además el Test Bench. Organice el código por módulos. Introduzca comentarios en el código para que sea fácilmente entendible. En cada sección de código explique cómo funciona.

5. Realice una simulación comportamental.

6. Sintetice el circuito y realice una simulación de timing. Analice los reportes y los esquemáticos generados. Establezca el tiempo de atraso total de su circuito.

7. SI DISPONE DE UNA TARJETA DE DESARROLLO, Implemente el circuito en la tarjeta. Realice las pruebas del circuito en la tarjeta. Especifique el conjunto de vectores de prueba que va a usar.

INFORME:

1. La práctica tiene una duración de 2 semanas.

2. Cada Grupo debe mostrar funcionando el circuito en el laboratorio a nivel de simulación.

3. Se debe entregar un informe detallado con los resultados obtenidos, incluyendo las gráficas con las simulaciones y los informes suministrados por la herramienta. Anexe los archivos VHDL de cada módulo.

4. Siga las pautas dadas para la realización y el envío del informe, de acuerdo con las indicaciones publicadas en Classroom.

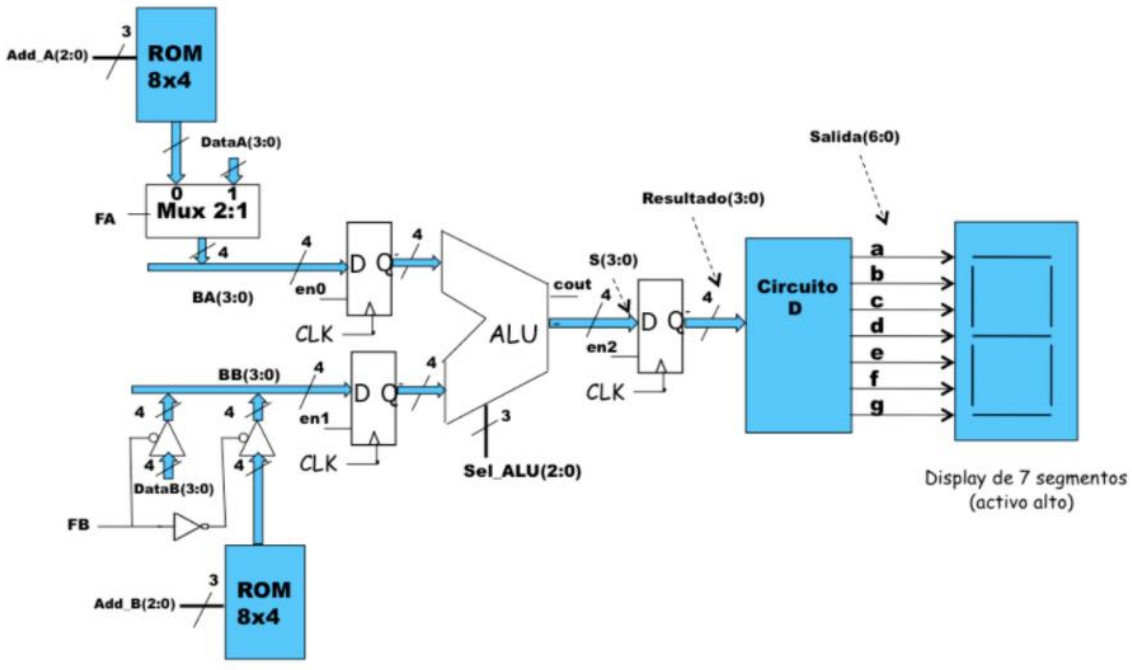

Figure 1: circuito a diseñar.

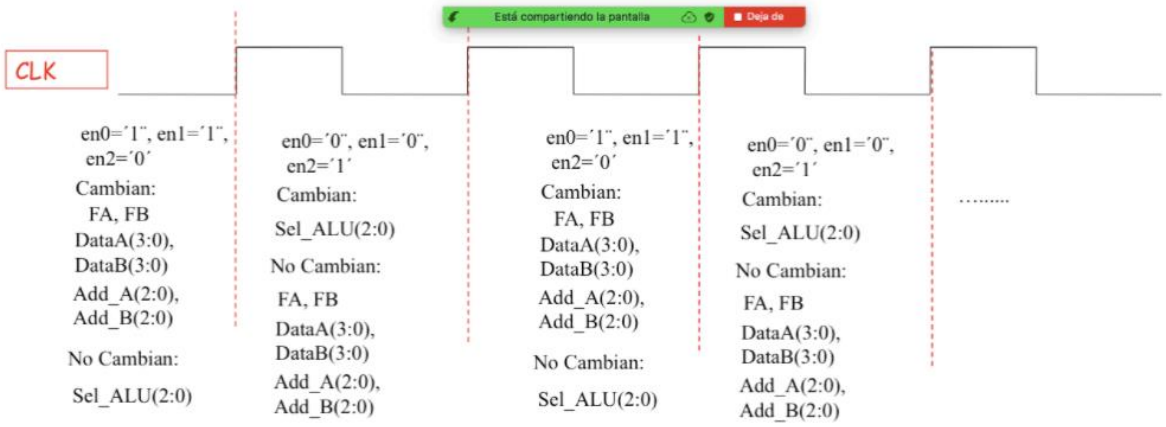

Figure 2: CLK planteado para la implementación.

GRUPO	Sel_ALU	FUNCIONES
8	000	s <= A – B (0 si A< B)
	001	s <= A nor B
	010	s <= A + 2
	011	s <= A xnor B
	100	s <= B
	101	s <= A * 2
	110	s <= B + A
	111	s <= A and B

Figure 3: operaciones planteadas de la ALU.

Grupo	ADDRESS	ROM_A	ROM_B
8	000	X7	X5
	001	X6	X7
	010	X5	X3
	011	X3	X8
	100	X1	X2
	101	XF	X9
	110	XD	X1
	111	XA	XA

Figure 4: valores de las ROMs planteados.

Desarrollo:

Diseño principal.

Para el código en VHDL se hace uso del software vivado, creando un proyecto con 2 archivos .vhd inicialmente, uno de diseño y simulación y otro únicamente de simulación, siendo este último el Test Bench.

Para el archivo de diseño se añaden las librerías necesarias y se define la entidad, teniendo en cuenta las señales de entrada y las de salida deseadas, las cuales se pueden visualizar en la figura 1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.all;

use STD.textio.all;
use IEEE.std_logic_textio.all;

library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity Circuit is
  Port (
        Add_A : in STD_LOGIC_VECTOR(2 downto 0);
        Add_B : in STD_LOGIC_VECTOR(2 downto 0);
        DataA : in STD_LOGIC_VECTOR(3 downto 0);
        DataB : in STD_LOGIC_VECTOR(3 downto 0);
        FA : in STD_LOGIC;
        FB : in STD_LOGIC;
        en : in STD_LOGIC_VECTOR(0 to 2);
        Sel_ALU : in STD_LOGIC_VECTOR(2 downto 0);
        Salida : out STD_LOGIC_VECTOR(6 downto 0);
        clk : in STD_LOGIC;
        carry : out STD_LOGIC
      );
end Circuit;
```

Figure 5: entidad del diseño del circuito.

Luego, se añaden dos archivos .vhd, los cuales van a tener como objetivo definir la componente ROM, tal como se presentó durante las sesiones teóricas. Para ello, se debe tener a consideración la figura 4.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity ROMa is
    Port (
        addA : in std_logic_vector(2 downto 0);
        outA : out STD_LOGIC_VECTOR (3 downto 0)
    );
end ROMa;

architecture Behavioral of ROMa is
    type rom_type is array (0 to 7) of std_logic_vector (3 downto 0);
    constant ROM : rom_type :=(x"7",x"6",x"5",x"3",x"1",x"F",x"D",x"A");

    -- inicio de la arquitectura
    Begin
        outA <= ROM(conv_integer(addA));
end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity ROMb is
    Port (
        addB : in std_logic_vector(2 downto 0);
        outB : out STD_LOGIC_VECTOR (3 downto 0));
end ROMb;

architecture Behavioral of ROMb is
    type rom_type is array (0 to 7) of std_logic_vector (3 downto 0);
    constant ROM : rom_type :=(x"5",x"7",x"3",x"8",x"2",x"9",x"1",x"A");

    -- inicio de la arquitectura
    Begin
        outB <= ROM(conv_integer(addB));
end Behavioral;
```

Figure 6: diseño de las ROMs.

Ahora se definen las componentes a utilizar y las señales auxiliares dentro de la arquitectura:

```vhdl
architecture Behavioral of Circuit is

    --CLOCK
    signal clk1 : std_logic := '1';
    --Counter
    signal aux : integer := 0;

    --ROMs outputs
    signal outROMA : std_logic_vector(3 downto 0);
    signal outROMB : std_logic_vector(3 downto 0);

    --Processes outputs
    signal BA : STD_LOGIC_VECTOR(3 downto 0);
    signal BB : STD_LOGIC_VECTOR(3 downto 0);
    signal s : STD_LOGIC_VECTOR(4 downto 0); --Pos 4, carry out
    signal Resultado : STD_LOGIC_VECTOR(3 downto 0);

    --Signals for Flip Flops
    signal QA, QB : STD_LOGIC_VECTOR(4 downto 0) := "00000";
    signal QC : STD_LOGIC_VECTOR(3 downto 0):= "0000";

        --ROMs
        component ROMa port
            ( addA : in std_logic_vector(2 downto 0);
              outA : out std_logic_vector(3 downto 0)
              );
        end component;

        component ROMb port
            ( addB : in std_logic_vector(2 downto 0);
              outB : out std_logic_vector(3 downto 0)
              );
        end component;
```

Figure 7: señales auxiliares y componentes.

También, se diseña el divisor de pulsos del reloj, para obtener la frecuencia deseada para el diseño codificado del circuito.

```
--CLOCK
CLK_DIV1 : process(clk)
    begin
        if aux = 0 then
            clk1 <= '0';
            aux <= aux + 1;
        end if;

        if (clk' event and clk='1') then
            if(aux = 10000000) then
                aux <= 1;
                clk1 <= not(clk1);
            else
                aux <= aux + 1;
            end if;
        end if;
end process;
```

Figure 8: reloj a utilizar.

Por último, se procede a diseñar cada uno de los componentes y procesos que se deben realizar para cumplir con la implementación en código del circuito planteado. Para observar más a detalle revisar el archivo Circuit.vhd en la carpeta:

Digitales1/Practica 3/Practica3/Practica3.srcs/sources_1/new/Circuit.vhd

Test Bench.

Al ser un archive puro de simulación la entidad se es vacía y dentro de la arquitectura se define un componente referente al circuito principal:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.all;

use STD.textio.all;
use IEEE.std_logic_textio.all;

library UNISIM;
use UNISIM.VCOMPONENTS.all;

Entity Circuit_tb Is
end Circuit_tb;

Architecture Behavioral of Circuit_tb Is
    Component Circuit
        port (
            Add_A : in STD_LOGIC_VECTOR(2 downto 0);
            Add_B : in STD_LOGIC_VECTOR(2 downto 0);
            DataA : in STD_LOGIC_VECTOR(3 downto 0);
            DataB : in STD_LOGIC_VECTOR(3 downto 0);
            FA : in STD_LOGIC;
            FB : in STD_LOGIC;
            en : in STD_LOGIC_VECTOR(0 to 2);
            Sel_ALU : in STD_LOGIC_VECTOR(2 downto 0);
            Salida : out STD_LOGIC_VECTOR(6 downto 0);
            clk : in STD_LOGIC;
            carry : out STD_LOGIC
            );
    end Component;
```

Figure 9: entidad y componente Circuit.

Luego se describen las señales auxiliares y los procedimientos de los elementos que componen el diseño total del circuito, para ello ver:

C:/Users/juanc/Documents/GitHub/Digitales1/Practica 3/Practica3/Practica3.srcs/sim_1/new/Circuit_tb.vhd

A continuación, se definen el port map y un process que defina un reloj que sea igual al suministrado de la FPGA para realizar un divisor para las señales enable de la simulación.

```
begin
    uut : Circuit port map(
        Add_A => add_a,
        Add_B => add_b,
        DataA => dataa,
        DataB => datab,
        FA => fa,
        FB => fb,
        en => ena,
        Sel_ALU => sel_ALU,
        Salida => salida,
        clk => CLK,
        carry => Carry
        --clockprobe => clkprobe,
        --outALU => outAL,
        --outFlipFlop1 => outFlip1,
        --outFlipFlop2 => outFlip2,
        --outROM1 => outROMA1

    );

    process
        begin
            CLK <= '0';
            wait for 5ns;
            CLK <= '1';
            wait for 5ns;
    end process;
```

```
process
    begin

        ena(2) <= '1';
        clkpr <= '0';

        wait for 95ms;
        ena(0) <= '1';
        ena(1) <= '1';

        wait for 5ms;
        clkpr <= '1';

        wait for 5ms;
        ena(2) <= '0';

        wait for 95ms;
        clkpr <= '0';

        wait for 95ms;
        ena(2) <= '1';

        wait for 5ms;
        clkpr <= '1';

        wait for 5ms;
        ena(0) <= '0';
        ena(1) <= '0';

        wait for 95ms;
end process;
```

Figure 10: port map, reloj y divisor del reloj.

Una vez codificado esto, se procede a definir más señales auxiliares que sean de ayuda durante el proceso principal en la obtención de las respuestas esperadas, haciendo uso de ciclos for en los que cambian de valores las diferentes entradas dependiendo de los valores de las señales FA y FB, ya que estas son las que indican qué dato de entrada será el elegido durante el procesamiento y las operaciones algebraicas implementadas.

```
begin
    write(s, string'(" "));write (s, add_a);write(s, string'(" "));write (s, add_b);write(s, string'(" "));
    writeline(output, s);
    for auxa in 0 to 1 loop
        if auxa = 0 then
            fa <= '0';
        else
            fa <= '1';
        end if;
        for auxb in 0 to 1 loop
            if auxb = 0 then
                fb <= '0';
            else
                fb <= '1';
            end if;
            write(s, string'(" FA: "));write (s, fa);write(s, string'(" FB: "));write (s, fb);
            writeline (output, s);
            if fa = '0' and fb = '0' then
                for count_adda in 0 to 7 loop
                    for count_datab in 0 to 15 loop
                        wait for 100ms;

                        RomA(add_a, outRomA);
                        RomB(add_b, outRomB);
                        MUX(dataa, outRomA, fa, upper);
                        Buffers(datab, outRomB, fb, lower);
                        FLIPFLOP(ena(0), upper, flipflop1); --First Flip-Flop
                        FLIPFLOP(ena(1), lower, flipflop2); --Second Flip-Flop
```

Figure 11: ejemplo 1 del proceso principal.

```
        for count_dataSel in 0 to 7 loop

            if count_dataSel = 0 then
                wait for 200ms;
            else
                wait for 400ms;
            end if;

              write(s, string'("Enable A: "));write (s, ena(2));
              writeline(output, s);
              write(s, string'("Enable D: "));write (s, ena(2));
              writeline(output, s);
            ALU(flipflop1, flipflop2, sel_ALU, outputALU, outCarry);
            --Carry <= outCarry;
            FLIPFLOP(ena(2), outputALU, flipflop3); --Third FLip-Flop
            Decoder(flipflop3, output7); --Last operation

            write(s, string'(" "));write (s, add_a);write(s, string'(" "));write (s, add_b),
            writeline(output, s);
            write(s, string'("Address A: "));write (s, add_a);write(s, string'(" ROM A: ")),
            writeline(output, s);
            write(s, string'(" Expected Out Decoder: "));write (s, output7);write(s, string
            writeline (output, s);
            write(s, string'(" Expected ALU: "));write (s, outputALU);--write(s, string'(" ,
            writeline (output, s);
            write(s, string'(" sel ALU: "));write (s, sel_ALU);
            writeline (output, s);
            sel_ALU <= sel_ALU + 1; --Change the Sel Alu value

        end loop;
        wait for 100ms;
        sel_ALU <= "000";
        datab <= datab + 1;
```

Figure 12: ejemplo 2 del proceso principal.

```
            end loop;
            wait for 100ms;
            sel_ALU <= "000";
            datab <= datab + 1;

        end loop;

        datab <= "0000";
        add_a <= add_a + 1;

    end loop;

    add_a <= "000";
```

Figure 13: ejemplo 3 del proceso principal.

Para ver el código completo dirigirse a la carpeta mencionada anteriormente.

C:/Users/juanc/Documents/GitHub/Digitales1/Practica 3/Practica3/Practica3.srcs/sim_1/new/Circuit_tb.vhd

Constraint.

En el archivo constraint se definen los puertos a utilizar según los datos que se digitaron en la entidad del archivo de diseño.

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {Add_A[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Add_A[0]}]
set_property PACKAGE_PIN V16 [get_ports {Add_A[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Add_A[1]}]
set_property PACKAGE_PIN W16 [get_ports {Add_A[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Add_A[2]}]
set_property PACKAGE_PIN W17 [get_ports {Add_B[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Add_B[0]}]
set_property PACKAGE_PIN W15 [get_ports {Add_B[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Add_B[1]}]
```

Figure 14: ejemplo 1 constraint.

Ya que se debe observar una salida en el display siete segmentos, se define la utilización de los pines según lo trabajado en ocasiones anteriores.

```
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Salida[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[6]}]
set_property PACKAGE_PIN W6 [get_ports {Salida[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[5]}]
set_property PACKAGE_PIN U8 [get_ports {Salida[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[4]}]
set_property PACKAGE_PIN V8 [get_ports {Salida[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[3]}]
set_property PACKAGE_PIN U5 [get_ports {Salida[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[2]}]
set_property PACKAGE_PIN V5 [get_ports {Salida[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[1]}]
set_property PACKAGE_PIN U7 [get_ports {Salida[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Salida[0]}]
```

Figure 15: ejemplo 2 constraint.

Y debido a que la cantidad de Switches que provee la FPGA no son suficientes, se debe hacer uso de los Pmod para la entrada de más datos con un circuito diseñado aparte.

```
##Pmod Header JA
##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {Sel_ALU[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Sel_ALU[0]}]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {Sel_ALU[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Sel_ALU[1]}]
##Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {Sel_ALU[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Sel_ALU[2]}]
```

Figure 16: ejemplo 3 constraint.

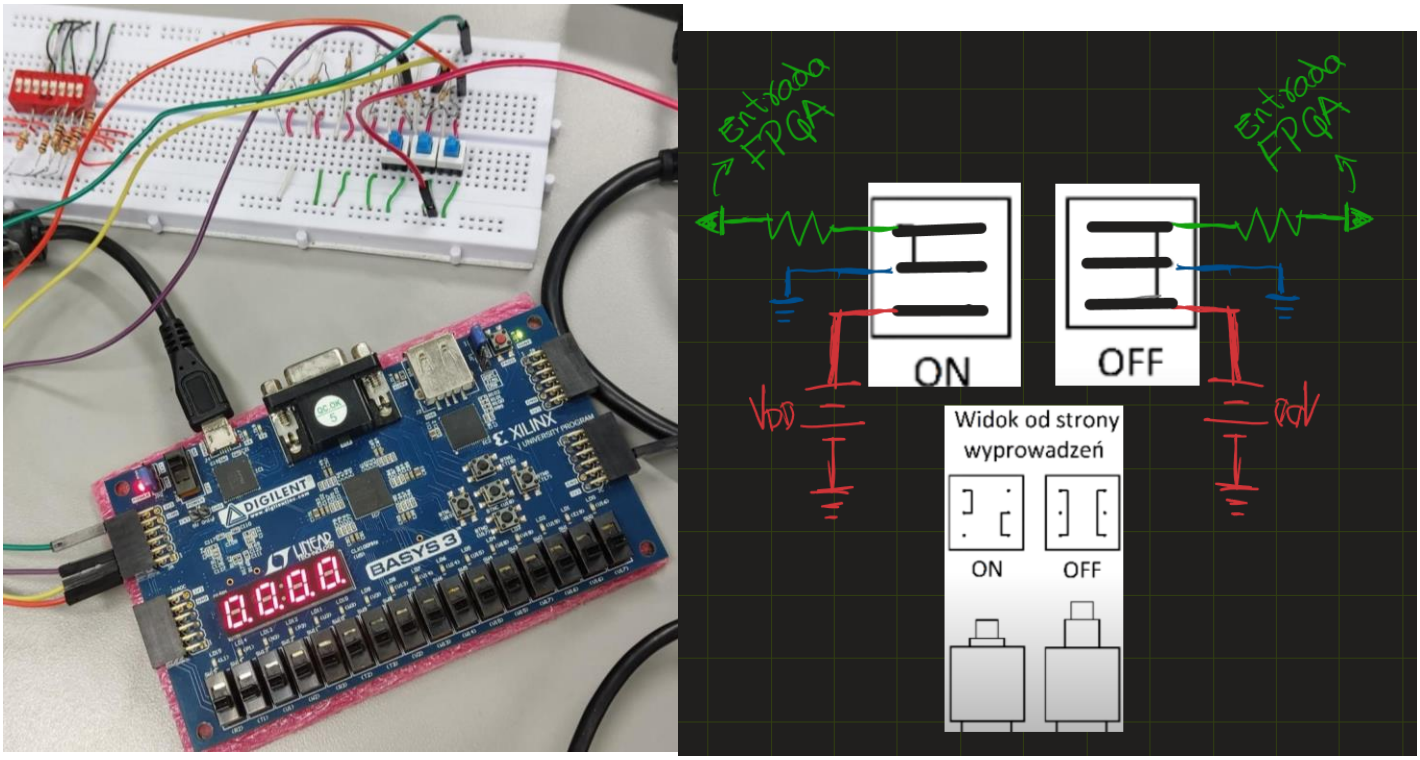

Figure 17: circuito utilizado para los Switches faltantes.

Para ver el archivo constraint (.xdc) completo dirigirse a:

C:/Users/juanc/Documents/GitHub/Digitales1/Practica 3/Basys3_Master.xdc

Resultados.

Después de que el código del diseño esté completo se ejecuta el análisis RTL y se obtiene el siguiente diseño:

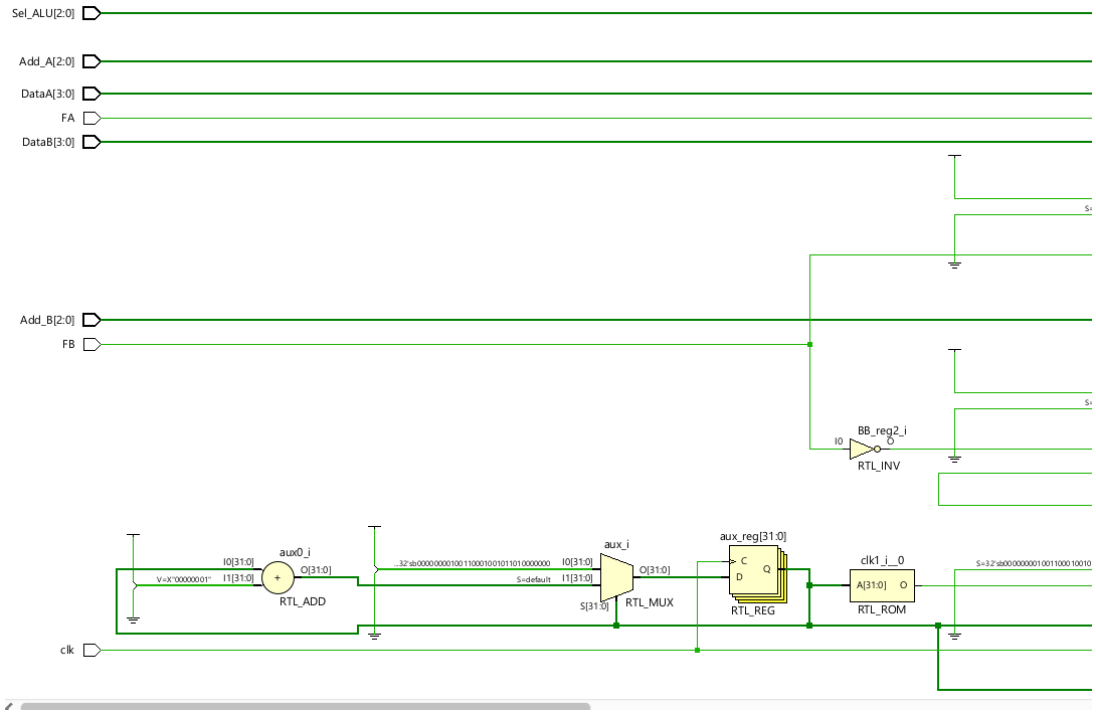

Figure 18: parte 1 del diseño obtenido.

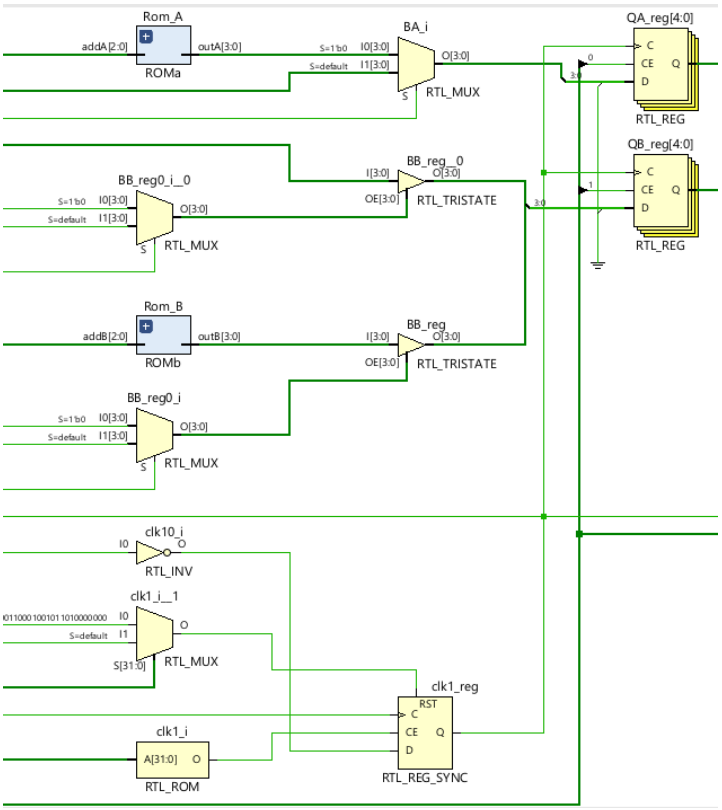

Figure 19: parte 2 del diseño obtenido.

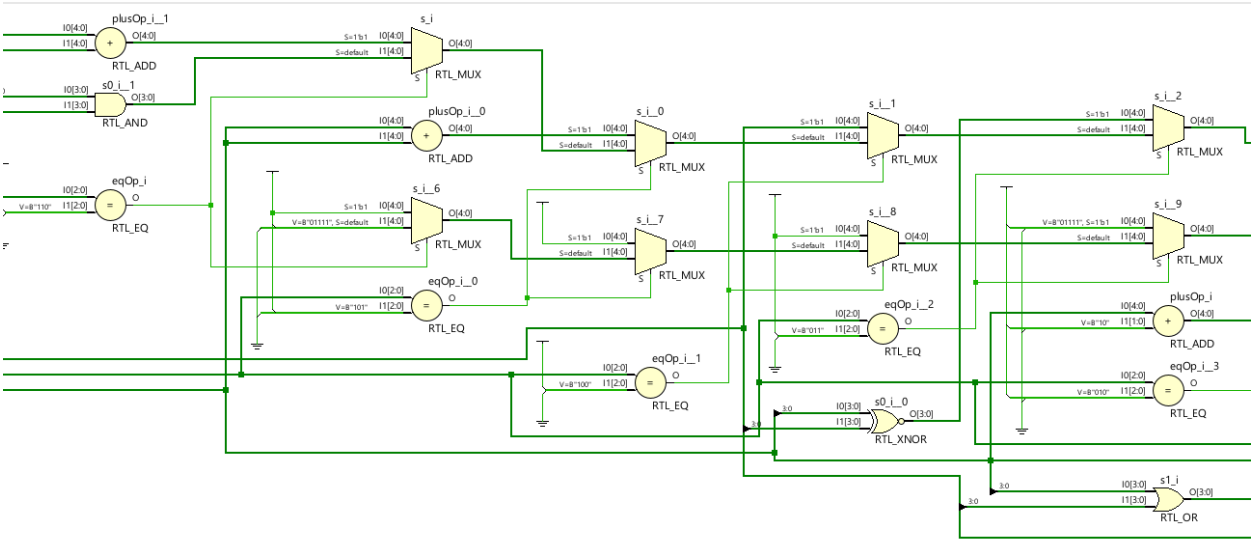

Figure 20: parte 3 del diseño obtenido.

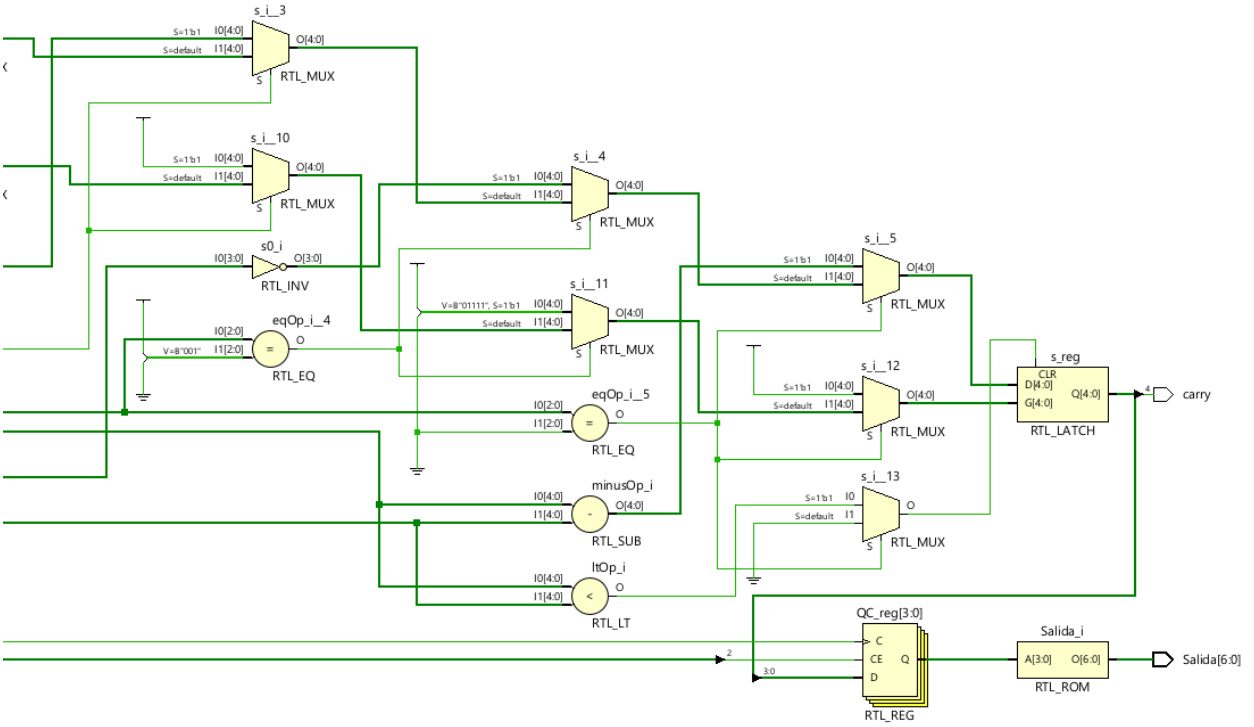

Figure 21: parte 4 del diseño obtenido.

Es posible observar en cada una de las imágenes del diseño generado, los elementos planteados en la figura 1, además de unos cuantos más por la implementación usada, los cuales no se tenían en cuenta en el diseño planteado si no en las instrucciones de codificación para el proyecto.

Por otro lado, en cuanto a la respuesta de la simulación por parte del Test Bench, debido al tiempo usado para el divisor del reloj, toma mucho tiempo obtener la simulación de cada una de las respuestas posibles en el proceso. Aun así, es posible dar evidencia de que la respuesta esperada y la respuesta real son iguales.

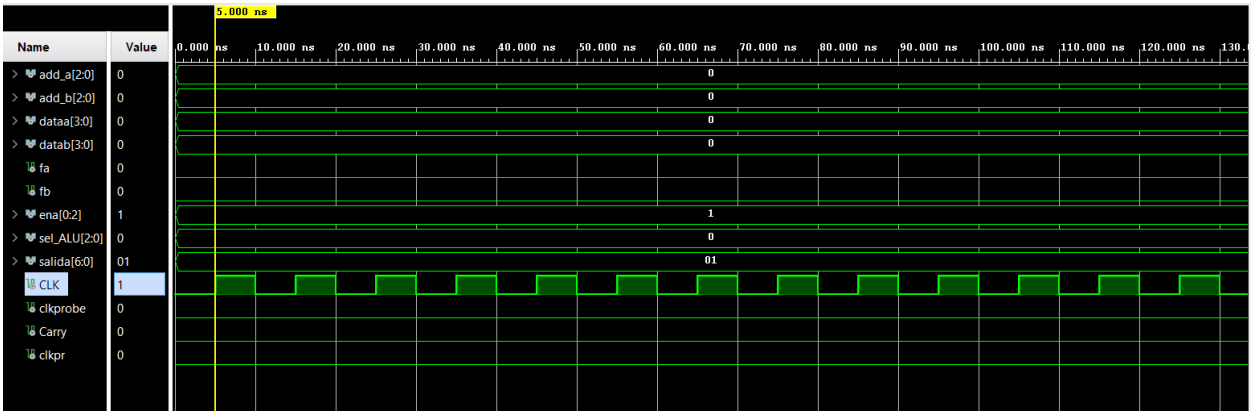

Figure 22: señal del reloj de la FPGA.

Es posible ver que se cumple con que las especificaciones de la FPGA que habla sobre la frecuencia del reloj, la cual es de 100MHz. Por ellos es que se realiza el divisor, el cual, para este caso, genera una señal con una frecuencia de 5Hz.

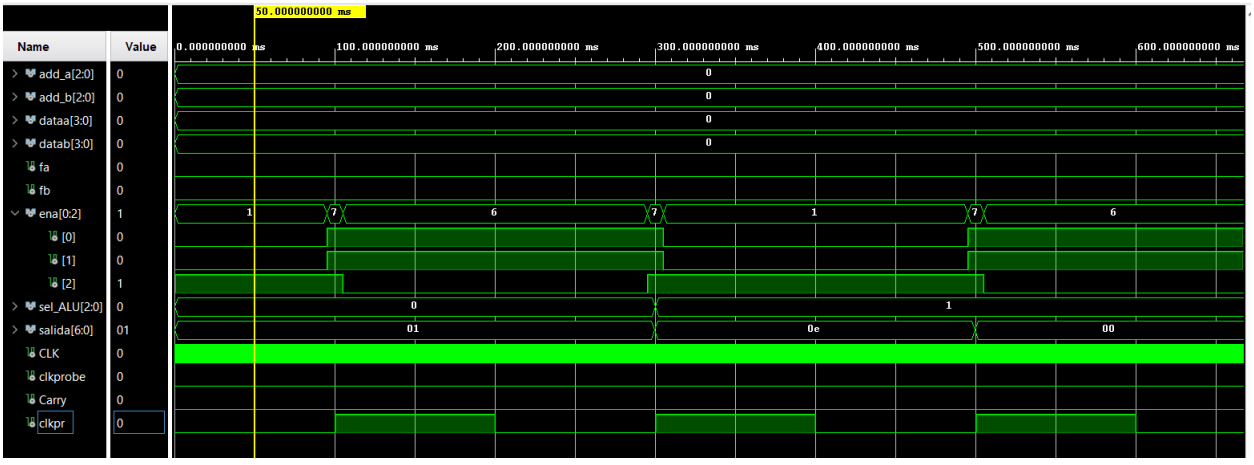

Figure 23: señal obtenida a partir del divisor.

$$f = \frac{1}{T} \rightarrow T = \frac{1}{f}$$

Además, es posible observar que en cada flanco de subida las señales de enable cambian de estado, aunque tienen un pequeño adelanto y atraso para evitar inconvenientes durante la simulación.

En cuanto a la consola, se imprime:

```
 000 000 0000 0000 0 0 001 000 UUUUUUU
 FA: 0 FB: 0
 000 000 0000 0000 0 0 111 000 0001110
Address A: 000 ROM A: 0111 Data B: 0000 FLIP FLOP A: 0111 FLIP FLOP B: 0000 FLIP FLOP C: 0111
 Expected Out Decoder: 0001110 Actual Out Decoder: 0001110 Expected Out Carry: 0 Actual Out Carry: 0
 Expected ALU: 0111
 sel ALU: 000
 000 000 0000 0000 0 0 111 001 0000000
Address A: 000 ROM A: 0111 Data B: 0000 FLIP FLOP A: 0111 FLIP FLOP B: 0000 FLIP FLOP C: 1000
 Expected Out Decoder: 0000000 Actual Out Decoder: 0000000 Expected Out Carry: 0 Actual Out Carry: 0
 Expected ALU: 1000
 sel ALU: 001
 000 000 0000 0000 0 0 111 010 0000100
Address A: 000 ROM A: 0111 Data B: 0000 FLIP FLOP A: 0111 FLIP FLOP B: 0000 FLIP FLOP C: 1001
 Expected Out Decoder: 0000100 Actual Out Decoder: 0000100 Expected Out Carry: 0 Actual Out Carry: 0
 Expected ALU: 1001
 sel ALU: 010
 000 000 0000 0000 0 0 111 011 0000000
Address A: 000 ROM A: 0111 Data B: 0000 FLIP FLOP A: 0111 FLIP FLOP B: 0000 FLIP FLOP C: 1000
 Expected Out Decoder: 0000000 Actual Out Decoder: 0000000 Expected Out Carry: 0 Actual Out Carry: 0
 Expected ALU: 1000
 sel ALU: 011
 000 000 0000 0000 0 0 111 100 0000001
Address A: 000 ROM A: 0111 Data B: 0000 FLIP FLOP A: 0111 FLIP FLOP B: 0000 FLIP FLOP C: 0000
 Expected Out Decoder: 0000001 Actual Out Decoder: 0000001 Expected Out Carry: 0 Actual Out Carry: 0
 Expected ALU: 0000
 sel ALU: 100
```

Figure 24: impresión en la consola.

La consola muestra los datos de entrada y algunos de los datos que pasan en los filtros del procesamiento, además de que imprime la respuesta esperada y la obtenida del decodificador y del carry, los cuales se muestran mediante un display siete segmentos y un LED respectivamente. En la siguiente figura es notable que en el caso en el que todos los datos de entrada son 0, se obtiene la respuesta esperada, la cual es 7.

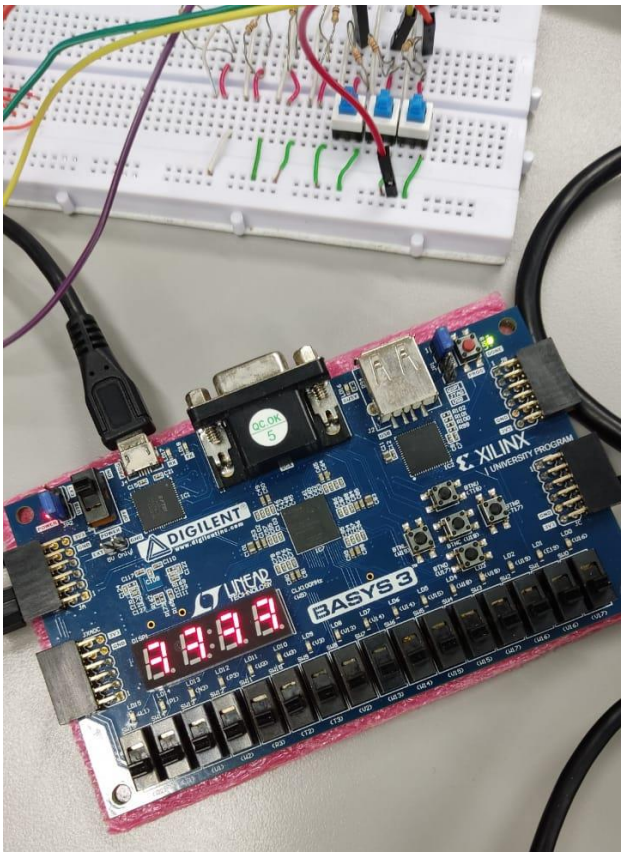

Figure 25: demostración de la implementación.

- Los Switches de la FPGA son las entradas Add_A, Add_B, DataA, DataB, FA y FB.
- 3 de los 5 botones observables son las señales enable o en.
- Los Switches integrados por medio del circuito externo son los seleccionadores de las operaciones de la ALU, es decir, el vector Sel_ALU de la figura 1.

Conclusiones:

- El uso de los registros, aunque no es simple en su totalidad, evidencia su utilidad para evitar el cruce de datos viejos y nuevos que permiten poder registrar resultados de un diseño que se esté trabajando.

- Al usar memorias ROM se facilita la obtención de información con más bits, sin la necesidad de usar entradas con mayor cantidad de datos, optimizando el espacio para el diseño de circuitos más eficientes.

- La FPGA es una herramienta de gran utilidad que permite el diseño de múltiples circuitos con diferentes funciones, permitiendo obtener respuestas sin diseñar un circuito físico para cada una.

- El uso de señales periódicas es fundamental para la manipulación de la información sin pérdidas ni alteraciones, ya que delimitan tiempos específicos en los que se debe realizar cada proceso de manera eficaz y efectiva.

- El lenguaje VHDL permite descripciones detalladas en las que se puede procesar cadenas de datos de múltiples maneras, dando una gran variedad en las formas de abordar diferentes soluciones.