Calculadora

Trabajo Práctico Final MyS-CESE 10 Cohorte

Alberto Yapura

Introducción

El presente trabajo práctico corresponde al desarrollo de una calculadora que permite realizar cuatro funciones básicas, que son la de sumar, restar, multiplicar y dividir números enteros utilizando VHDL como lenguaje de programación.

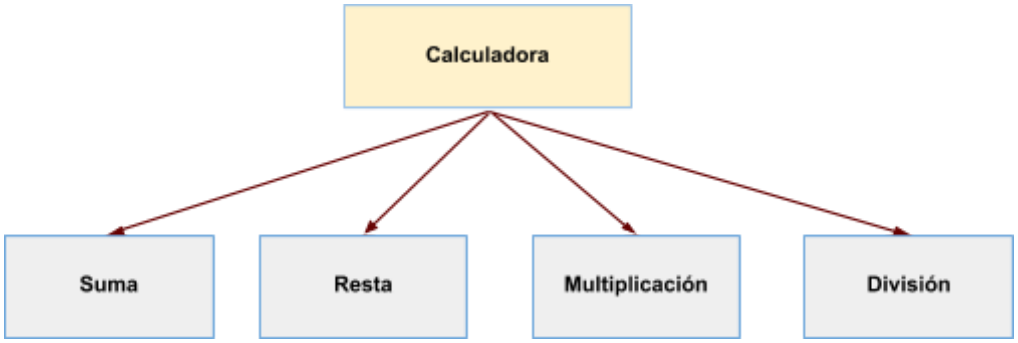
Se utilizó el componente desarrollado (Calculadora) para crear una IP mediante el IP Packager de Vivado, seguidamente se creó un proyecto que conecta la IP generada y una IP que permite controlar los led de la placa Arty Z7-10 (para visualizar la operación realizada) al sistema de procesamiento ZYNQ7 utilizando el bus AXI4 Lite, se desarrolló una interfaz con un menú en lenguaje C que permite a un usuario ingresar la operación a realizar y mostrar el resultado del mismo

Módulo Calculadora

Este módulo se desarrolló en VHDL, el mismo se compone de un módulo Sumador/Restador (desarrollado para el TP de CLP), Multiplicación, División.

El módulo Sumador/Restador permite realizar la suma/resta de número enteros con signo, característica no implementada en los módulos División y Multiplicación que solo admite número enteros sin signo.

El módulo Calculadora de acuerdo a la operación ingresada enviará los parámetros al módulo correspondiente para que realice la operación y retorna el resultado.



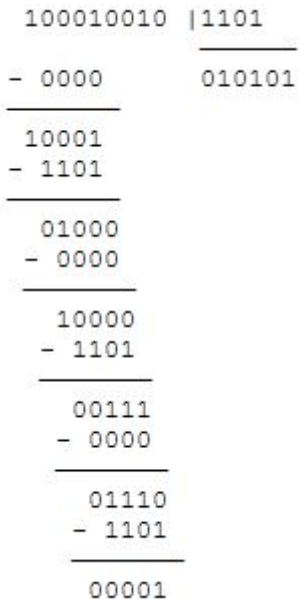
Los circuitos que realizan la multiplicación y la división son análogos, dado que el producto se puede realizar por sumas sucesivas y el cociente se puede realizar

mediante restas sucesivas lo que permitió la reutilización del módulo sumador_1b utilizado también en el módulo Sumador/Restador.

La división en binario es similar a la decimal, la única diferencia es que, a la hora de hacer las restas, dentro de la división, estas deben ser realizadas en binario.

Ejemplo

Dividir 100010010 (274) entre 1101 (13):

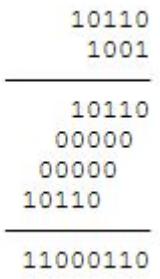


Se obtiene como cociente 010101 (21) y como resto 00001(1).

El algoritmo del producto en binario es igual que en números decimales; aunque se lleva a cabo con más sencillez, ya que el 0 multiplicado por cualquier número da 0, y el 1 es el elemento neutro del producto, el resultado de la multiplicación tendrá tantos bits como la suma del número de bits de los productos.

Ejemplo

Multiplicar 10110 (22) con 1001 (9):



Se obtiene como resultado 1100 0110 (198).

Archivos

El módulo Calculadora se compone de los siguientes archivos:

Componentes

- Calculadora.vhd
- Division.vhd
- Multiplicacion.vhd
- Sumador1b.vhd
- SumadorRestador.vhd
- Procesar.vhd

Simulación y pruebas

- tb_calculadora.vhd

Simulación

Para realizar las simulaciones, síntesis e implementación se utilizó el software Vivado 2018.1. Se simularon los componentes *Multiplicacion.vhd*, *Division.vhd*, *SumadorRestador.vhd* y el *Calculadora.vhd* como módulo principal.

Se muestra a continuación las capturas de pantallas de la simulación:

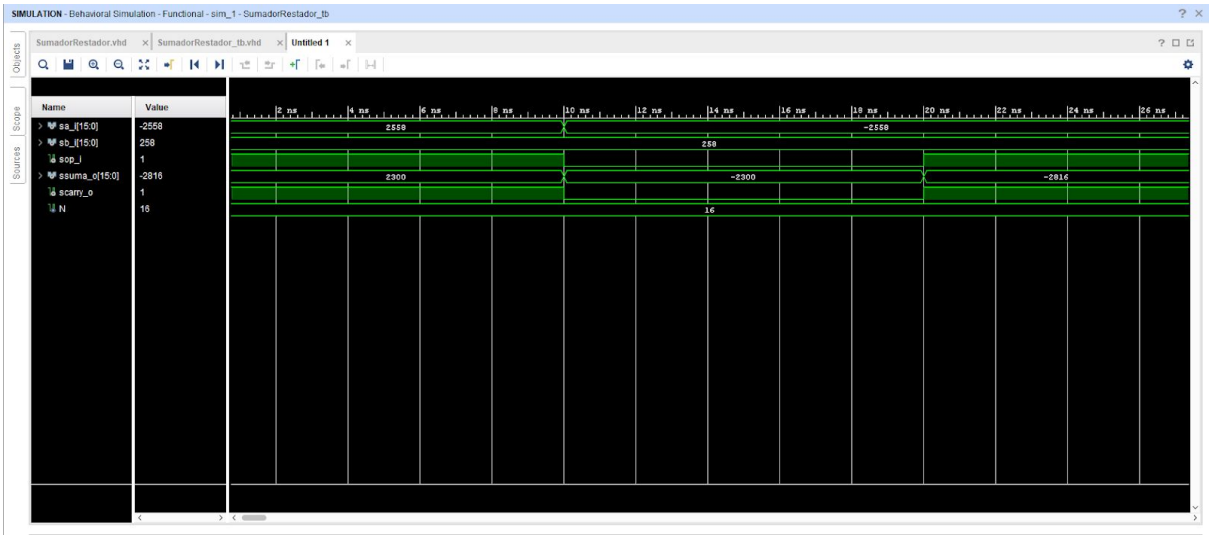


Figura 3. Simulación componente Sumador/Restador

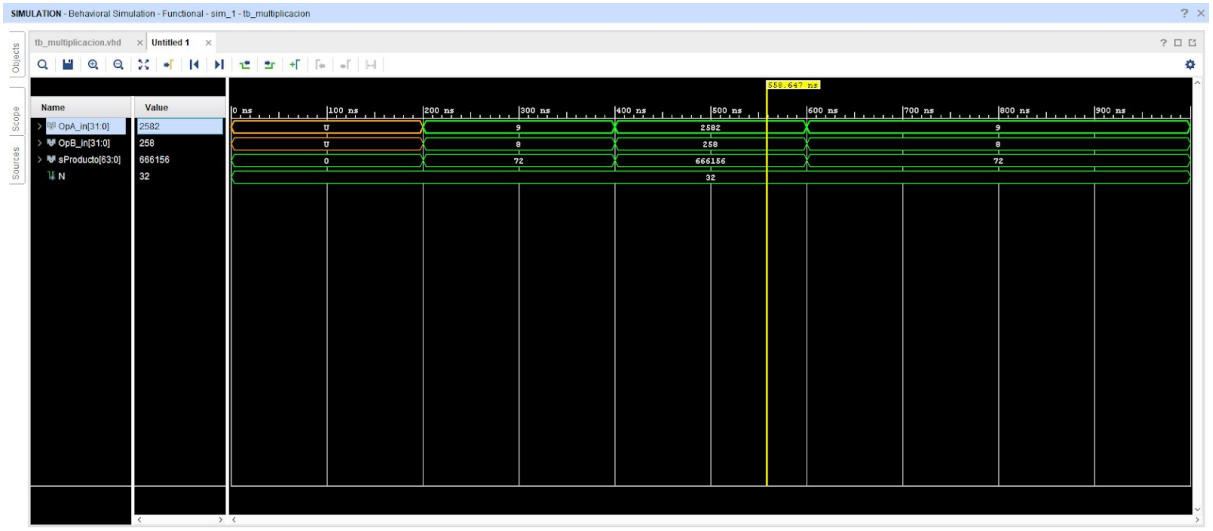


Figura 4. Simulación componente Multiplicación

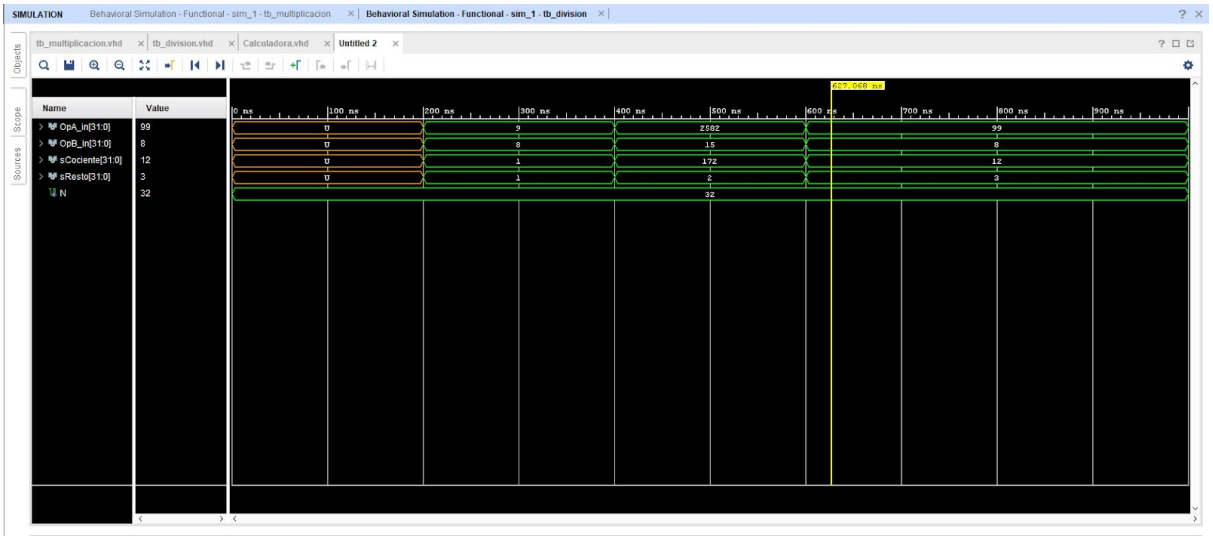


Figura 5. Simulación componente División

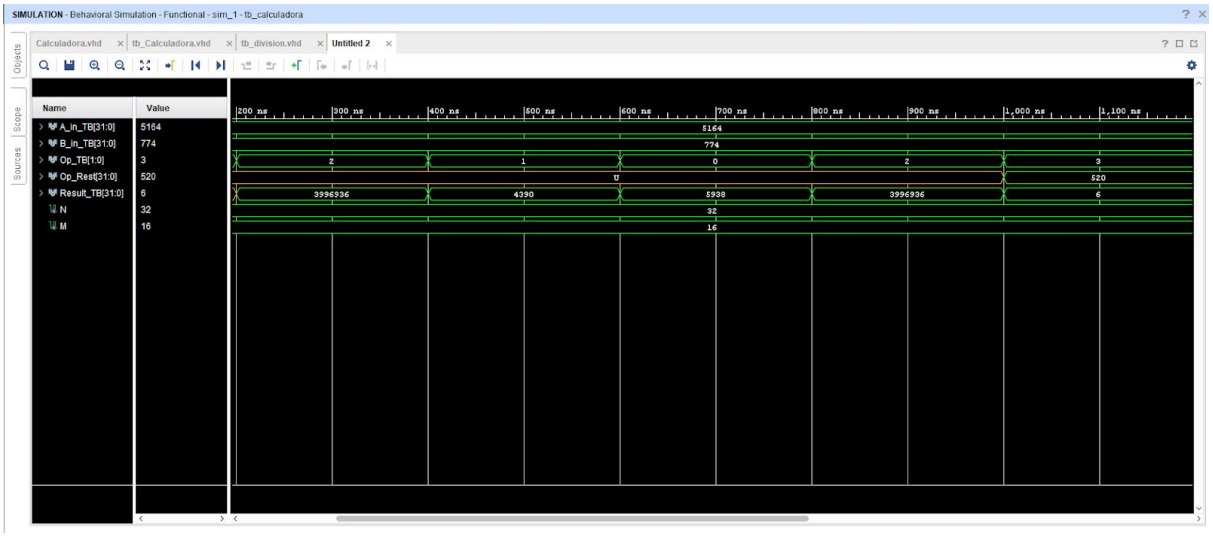


Figura 6. Simulación componente principal - Calculadora

Síntesis e implementación

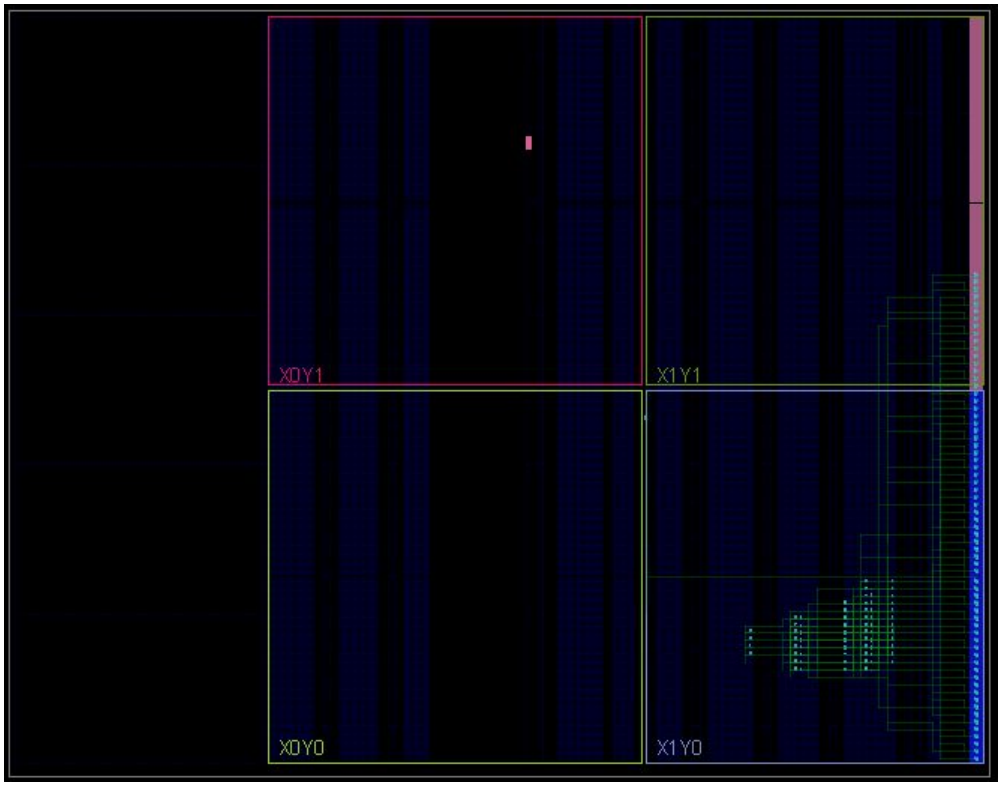


Figura 7. Síntesis del dispositivo

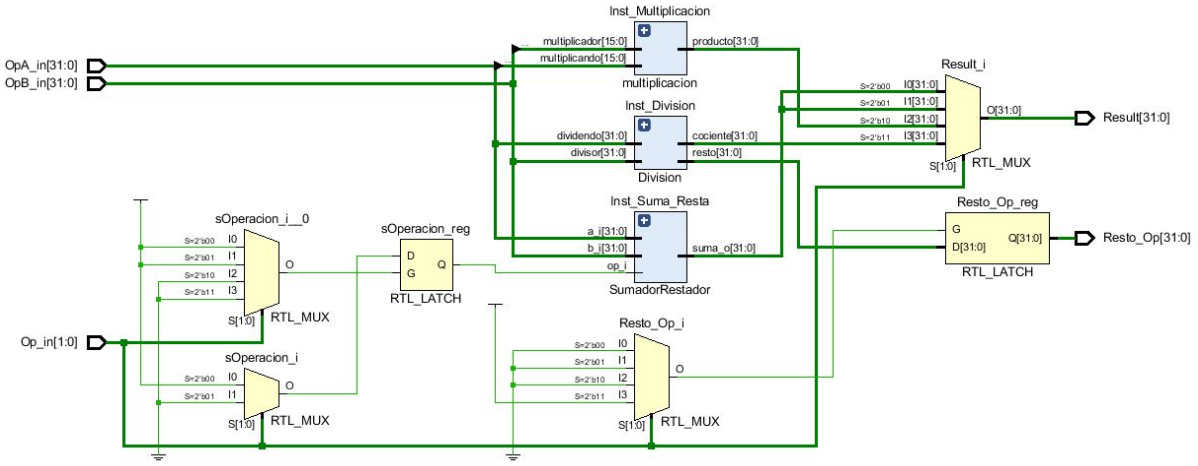


Figura 8. Esquemático del dispositivo

Resource	Utilization	Available	Utilization %
LUT	389	17600	2.21
FF	17	35200	0.05
IO	66	100	66.00

Figura 9. Tabla de utilización

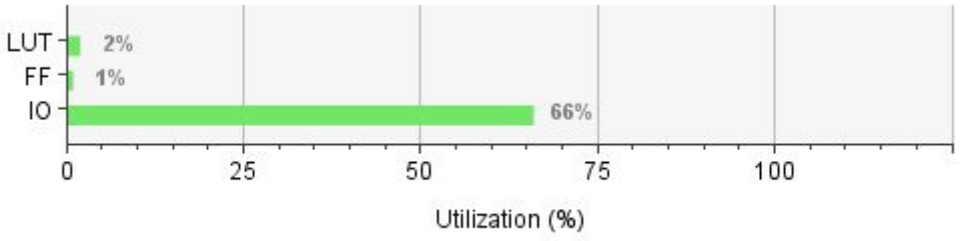


Figura 10. Gráfico de utilización

Como puede observarse en el módulo Calculadora implementada en FPGA, el consumo de recursos es en gran medida proporcional al número de bit utilizados (para la generación de estos reportes se utilizaron 16 bit para la suma/resta y división y 8 bit para la multiplicación) y al número de iteraciones.

IP Calculadora

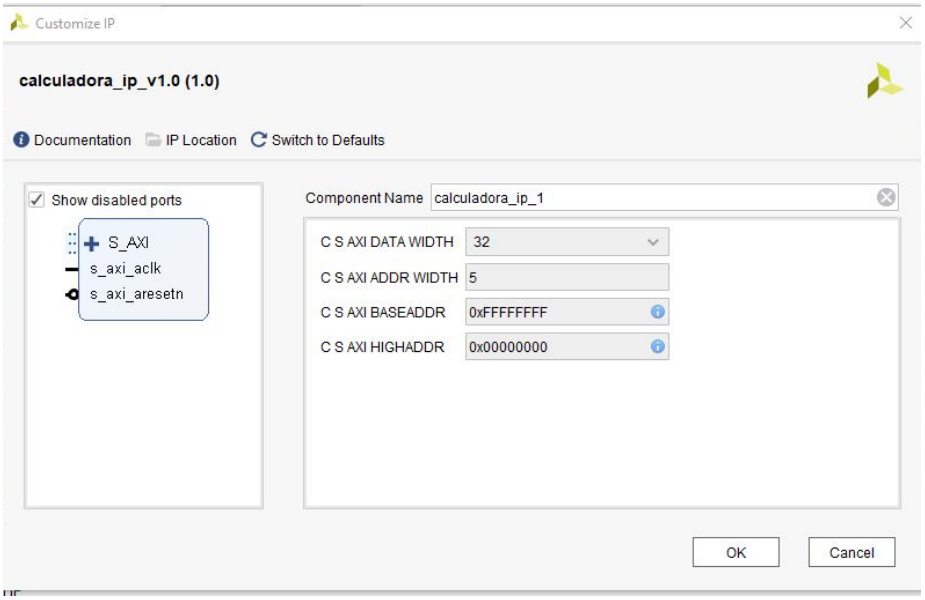


Figura 11. Calculadora IP

Para la generación de la IP calculadora se utilizaron 5 registros, 3 correspondientes a las entradas del operador A, operador B y la operación a realizar y 2 que se utilizan para obtener el resultado y el resto en caso de la división.

```
377  process (slv_reg0, slv_reg1, slv_reg2, resto, resultado, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
378  variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
379  begin
380      -- Address decoding for reading registers
381      loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
382      case loc_addr is
383        when b"000" =>
384          reg_data_out <= slv_reg0;
385        when b"001" =>
386          reg_data_out <= slv_reg1;
387        when b"010" =>
388          reg_data_out <= slv_reg2;
389        when b"011" =>
390          reg_data_out <= resto;
391        when b"100" =>
392          reg_data_out <= resultado;
393        when others =>
394          reg_data_out  <= (others => '0');
395      end case;
396  end process;
```

```
416
417      -- Add user logic here
418      inst_Calc : Calculadora
419        port map (
420              OpA_in   => slv_reg0,
421              OpB_in   => slv_reg1,
422              Op_in    => slv_reg2(1 downto 0),
423              Resto_Op => resto,
424              Result   => resultado
425        );
426      -- User logic ends
427
428  end arch_imp;
```

Figura 12. Configuración VHDL

Proyecto Calculadora

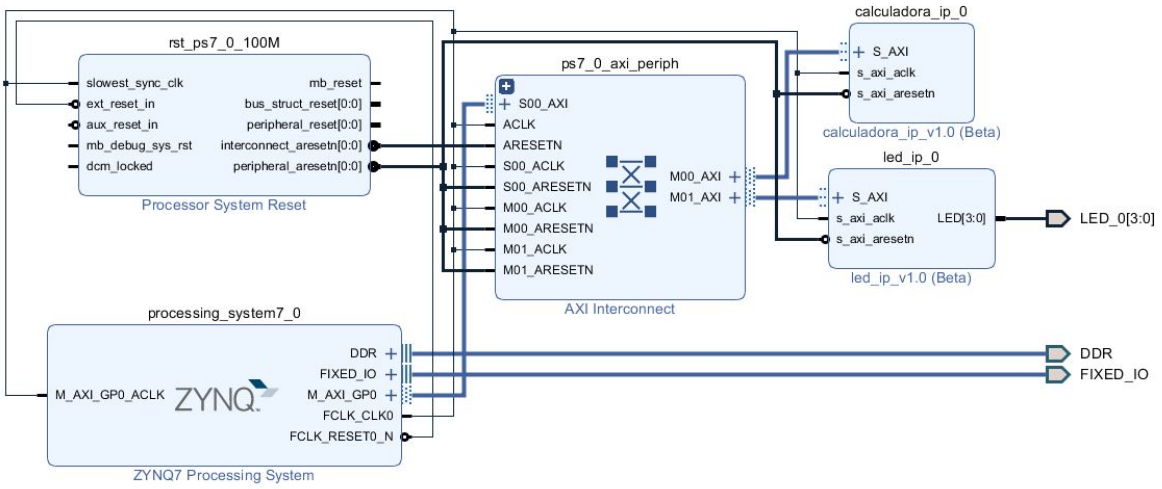


Figura 13. Diseño de Bloques

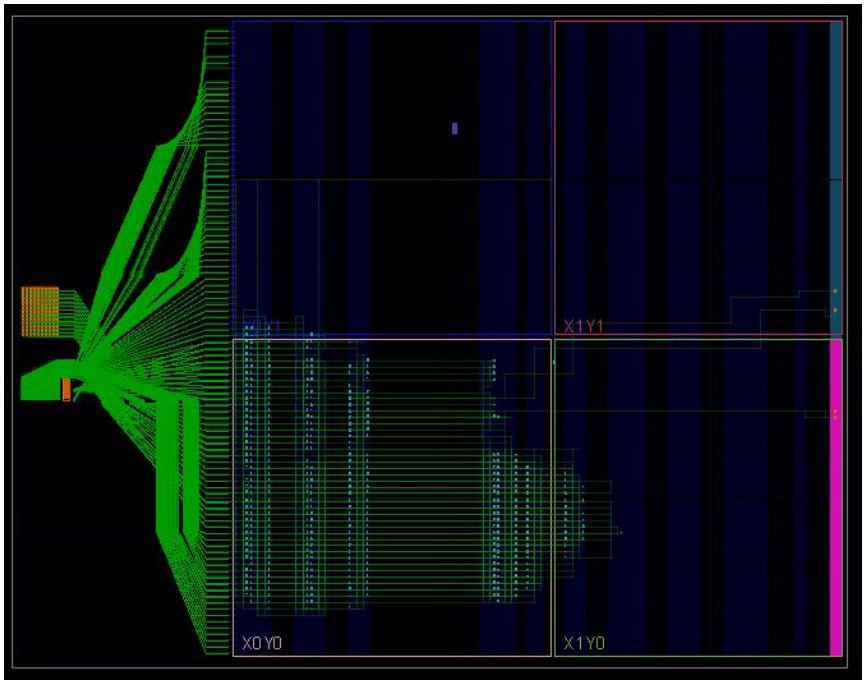


Figura 14. Proyecto sintetizado

Resource	Utilization	Available	Utilization %
LUT	2383	17600	13.54
LUTRAM	62	6000	1.03
FF	1030	35200	2.93
IO	4	100	4.00

Figura 15. Tabla de utilización

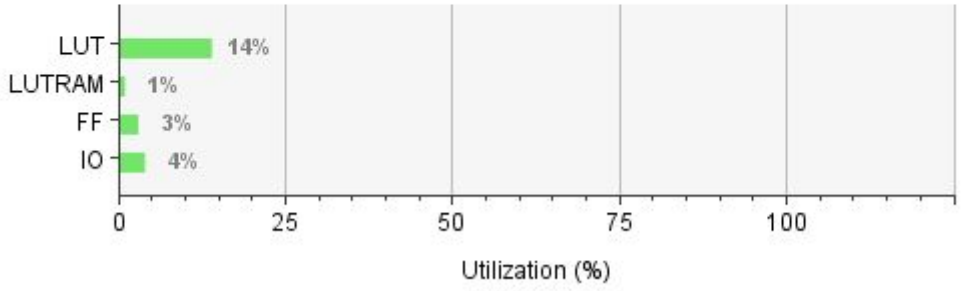


Figura 16. Gráfico de utilización

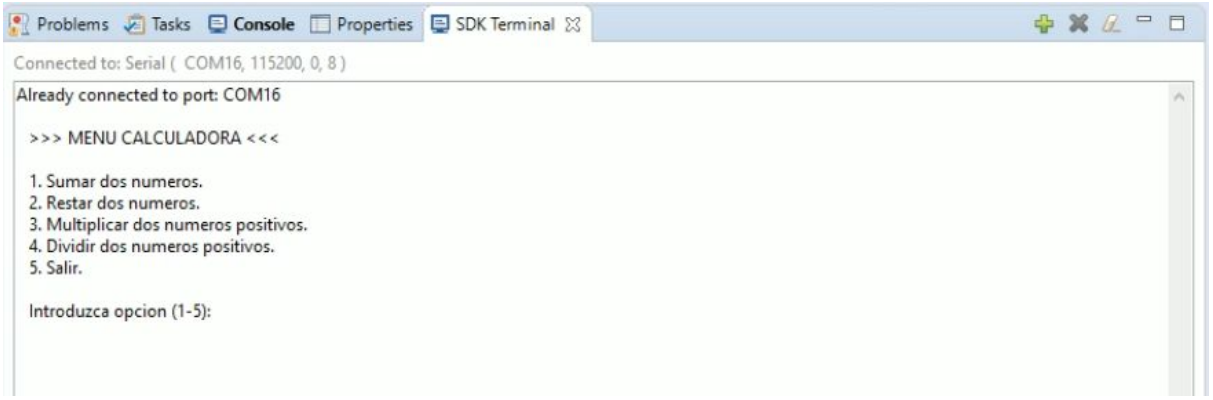


Figura 17. Menú de la Calculadora

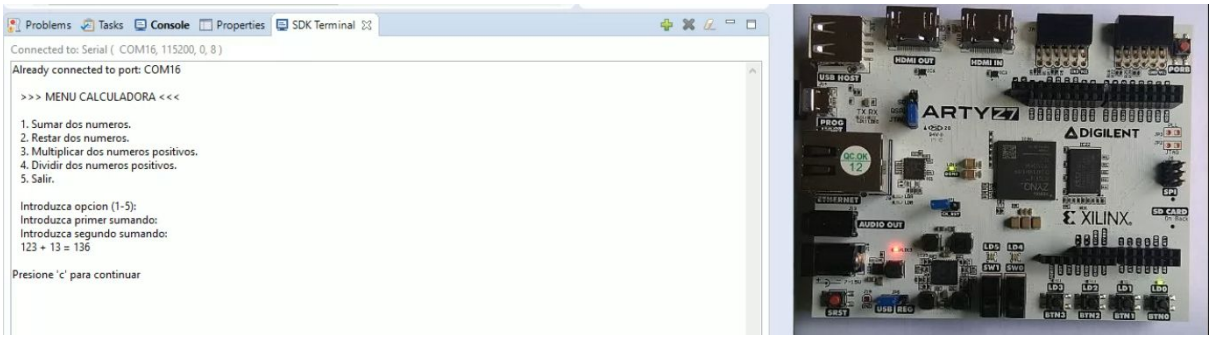


Figura 18. Relación Led con la operación

Código C

Para obtener los parámetros de entrada ingresados por el usuario, se decidió utilizar la función C **`scanf`**, a modo de interfaz sencilla con el usuario se desarrolló un menú con las operaciones permitidas, los resultados de las operaciones serán visualizadas en la terminal serie y la operación realizada será a través de un LED tal como se muestra en la figura 18.

```
#include "xparameters.h"
#include "calculadora_ip.h"
#include "led_ip.h"
#include "xil_io.h"
//================================================

int main (void)
{
        char opcion;
        char continuar;
        int OpA, OpB, resto;
        int  result;
        do
        {
                 xil_printf( "\n\n   >>> MENU CALCULADORA <<<" );
                 xil_printf( "\n\n   1. Sumar dos numeros.", 163 );
                 xil_printf( "\n   2. Restar dos numeros.", 163 );
                 xil_printf( "\n   3. Multiplicar dos numeros positivos.", 163 );
                 xil_printf( "\n   4. Dividir dos numeros positivos.", 163 );
                 xil_printf( "\n   5. Salir.\n" );
                 LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, 0);
        /* Filtramos la opción elegida por el usuario */
                 xil_printf( "\n   Introduzca opcion (1-5): ", 162 );
        do
        {
                scanf( "%c", &opcion);

        } while ( opcion < '1' || opcion > '5' );
        /* La opción sólo puede ser '1', '2', '3', '4' o '5' */

        switch ( opcion )
        {
                        /* Opción 1: Sumar */
                case '1': xil_printf( "\n   Introduzca primer sumando: " );
                                LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, 1);
                        scanf( "%d", &OpA);
                        xil_printf( "\n   Introduzca segundo sumando: " );
                        scanf( "%d", &OpB);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG0_OFFSET, OpA);

                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG1_OFFSET, OpB);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG2_OFFSET, 0);
                        result = CALCULADORA_IP_mReadReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG4_OFFSET);
                        xil_printf( "\n   %d + %d = %d\n", OpA, OpB, result);
                        break;
                        /* Opción 2: Restar */
                case '2': xil_printf( "\n   Introduzca minuendo: " );
```

```
                        LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, 2);
                scanf( "%d", &OpA);
                xil_printf( "\n   Introduzca sustraendo: " );
                scanf( "%d", &OpB);
                CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                CALCULADORA_IP_S_AXI_SLV_REG0_OFFSET, OpA);
                CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                CALCULADORA_IP_S_AXI_SLV_REG1_OFFSET, OpB);
                CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                CALCULADORA_IP_S_AXI_SLV_REG2_OFFSET, 1);
                result = CALCULADORA_IP_mReadReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                CALCULADORA_IP_S_AXI_SLV_REG4_OFFSET);
                xil_printf( "\n   %d - %d = %d\n", OpA, OpB, result );
                break;

                /* Opción 3: Multiplicar */
        case '3': xil_printf( "\n   Introduzca primer operando: " );
                        LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, 4);
                scanf( "%d", &OpA);
                xil_printf( "\n   Introduzca segundo operando: " );
                scanf( "%d", &OpB);
                if (( OpA >= 0 )&&( OpB >= 0 )){
                         CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG0_OFFSET, OpA);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG1_OFFSET, OpB);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG2_OFFSET, 2);
                        result =
                        CALCULADORA_IP_mReadReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG4_OFFSET);
                        xil_printf( "\n   %d * %d = %d\n", OpA, OpB, result );
                }else{
                        xil_printf( "\n   ERROR: los numeros ingresados deben ser
                                        positivos.\n" );
                        xil_printf( "\n   %d div %d = ERROR\n", OpA, OpB);
                                          }
                break;

                /* Opción 4: División entera */
        case '4': xil_printf( "\n   Introduzca dividendo: " );
                                LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, 8);
                scanf( "%d", &OpA);
                xil_printf( "\n   Introduzca divisor: " );
                scanf( "%d", &OpB);
                if (( OpB != 0 )&&( OpA >= 0 )&&( OpB >= 0 )){
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG0_OFFSET, OpA);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG1_OFFSET, OpB);
                        CALCULADORA_IP_mWriteReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG2_OFFSET, 3);
                        resto =
                        CALCULADORA_IP_mReadReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG3_OFFSET);
                        result =
                        CALCULADORA_IP_mReadReg(XPAR_CALCULADORA_IP_0_S_AXI_BASEADDR,
                        CALCULADORA_IP_S_AXI_SLV_REG4_OFFSET);
                        xil_printf( "\n   %d div %d = %d resto %d\n", OpA, OpB,
                        result, resto);
                }else{
```

```
                                    if ( OpB != 0 ){
                                            xil_printf( "\n   ERROR: No se puede dividir entre
                                                        cero.\n" );
                                            xil_printf( "\n   %d div %d = ERROR\n", OpA, OpB);
                                    }
                                    if (( OpA >= 0 )&&( OpB >= 0 )){
                                            xil_printf( "\n   ERROR: los numeros ingresados deben
                                                        ser positivos.\n" );
                                            xil_printf( "\n   %d div %d = ERROR\n", OpA, OpB);
                                    }
                            }

         }
        xil_printf( "\nPresione 'c' para continuar", 162 );
        do{
                            scanf( "%c", &continuar);
        } while(continuar!='c');
        } while ( opcion != '5' );

        return 0;

}
```

Figura 19. Código C