

Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

IE0424 - Laboratorio de Circuitos Digitales

I ciclo 2025

Reporte de Laboratorio #2

Gabriel Siles Chaves C17530

Jorge Loría Chaves C04406

Grupo 01

Profesor:

Marco Villalta Fallas

Índice

1. Resumen	2
2. Práctica 5: Creación del proyecto en Vivado [1]	2
3. Práctica 6: Introducción a entrada/salida	2
3.1. Ejercicio 3:	2
3.1.1. Descripción del problema [1]	2
3.1.2. Desarrollo de la solución	3
3.1.3. Diseño Final	7
3.2. Ejercicio 5:	8
3.2.1. Descripción del problema [2]	8
3.2.2. Desarrollo de la solución	8
3.2.3. Diseño Final	8
4. Práctica 8 RVfpga: Temporizadores	9
4.1. Ejercicio 2:	9
4.1.1. Descripción del problema [3]	9
4.1.2. Desarrollo de la solución	10
4.1.3. Diseño Final	16
4.2. Ejercicio 3:	17
4.2.1. Descripción del problema [3]	17
4.2.2. Desarrollo de la solución	17
4.2.3. Diseño Final	19
5. Problemas y complicaciones	23
6. Conclusiones y recomendaciones	23

1. Resumen

En este laboratorio, se utilizó la tarjeta Nexys A7 para modificar la descripción del hardware y hacer uso de diferentes periféricos como los botones, el temporizador y el LED tricolor. Se generó un *Bitstream* utilizando el software Vivado, lo que permitió a la placa reconocer y operar estos periféricos. La práctica incluyó la programación en C y en ensamblador para interactuar con los módulos mencionados. Los resultados se reflejan en el código y los algoritmos implementados, los cuales están disponibles en el siguiente enlace de GitHub: <https://git.ucr.ac.cr/GABRIEL.SILES/ie0424-gj>.

2. Práctica 5: Creación del proyecto en Vivado [1]

Por medio de las instrucciones de la guía 5 se realizó la creación del proyecto de Vivado, siguiendo los pasos de la práctica, con el objetivo de cargar el documento *RVfpgaNexys* en la Nexys A7 y poder modificar los periféricos que se desean utilizar.

Al finalizar con los pasos descritos en la guía, se genera el bitstream, lo que toma varios minutos, y se obtienen varios resultados y archivos. Cuando el proceso de generación de bitstream se completa, se obtiene el archivo final que confirma que todo se ha realizado con éxito.

3. Práctica 6: Introducción a entrada/salida

3.1. Ejercicio 3:

3.1.1. Descripción del problema [1]

Expandir el diseño *RVfpgaNexys* para que sea compatible con los cinco botones físicos disponibles en la placa. Los cinco botones se nombran según su ubicación: arriba, abajo, izquierda, derecha y centro, correspondientes a las etiquetas BTNU, BTND, BTNL, BTNR, y BTNC.

Instrucciones:

1. Dado que el tamaño máximo del módulo GPIO que estamos utilizando (*gpio_top*) es de 32, lo que corresponde al número de pines de E/S disponibles (16 LEDs + 16 Interruptores), es necesario incluir una nueva instancia del módulo GPIO en el diseño *SweRVolfX*, así como agregar 5 nuevos búferes de tres estados y todas las señales necesarias.
2. Utiliza las direcciones que comienzan en 0x80001800 (disponibles) para mapear los registros expuestos por el nuevo controlador GPIO. Ten en cuenta que debes modificar el multiplexor (Figura 9) para incluir el nuevo periférico.
3. También es necesario modificar el archivo de restricciones considerando que los cinco botones están conectados a los siguientes pines de la FPGA:
 - BTNC está conectado al PIN N17
 - BTNU está conectado al PIN M18
 - BTNL está conectado al PIN P17
 - BTNR está conectado al PIN M17
 - BTND está conectado al PIN P18

3.1.2. Desarrollo de la solución

Para lograr el correcto funcionamiento de los botones, necesitamos realizar las conexiones en los diferentes módulos para instanciar otro modulo GPIO. Se inicia con el módulo **rvpfganexys.sv** donde se toman las señales de cada uno de los botones.

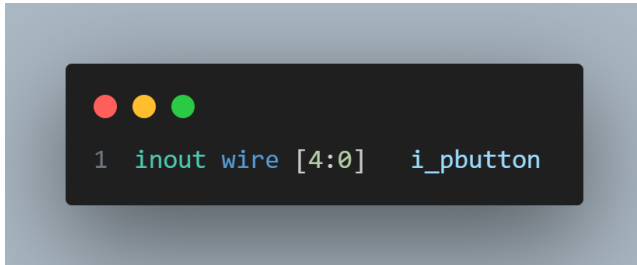


Figura 1: Se declara el wire como un inout

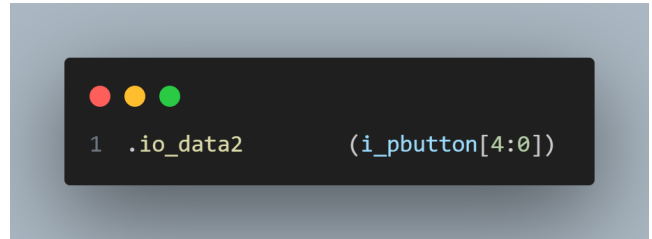


Figura 2: Se instancia el wire en el swervolfcore

Luego, se sigue con la modificación del archivo **swervolf_core.v** donde declaramos el modulo de GPIO2 para los botones, debido a que son 5 entradas se deben declarar únicamente 5 módulos bidireccionales y luego se realizan las conexiones en el wishbone.



Figura 3: Declaración de io_data2

```

1 // GPIO2 - Leds and Switches
2 wire [31:0] en_gpio2;
3 wire      gpio2_irq;
4 wire [31:0] i_gpio2;
5 wire [31:0] o_gpio2;
6
7 bidirec gpio2_0 (.oe(en_gpio2[0] ), .inp(o_gpio2[0] ), .outp(i_gpio2[0] ), .bidir(io_data2[0] ));
8 bidirec gpio2_1 (.oe(en_gpio2[1] ), .inp(o_gpio2[1] ), .outp(i_gpio2[1] ), .bidir(io_data2[1] ));
9 bidirec gpio2_2 (.oe(en_gpio2[2] ), .inp(o_gpio2[2] ), .outp(i_gpio2[2] ), .bidir(io_data2[2] ));
10 bidirec gpio2_3 (.oe(en_gpio2[3] ), .inp(o_gpio2[3] ), .outp(i_gpio2[3] ), .bidir(io_data2[3] ));
11 bidirec gpio2_4 (.oe(en_gpio2[4] ), .inp(o_gpio2[4] ), .outp(i_gpio2[4] ), .bidir(io_data2[4] ));
12
13 gpio_top gpio2_module(
14     .wb_clk_i      (clk),
15     .wb_rst_i      (wb_rst),
16     .wb_cyc_i      (wb_m2s_gpio2_cyc),
17     .wb_adr_i      ({2'b0,wb_m2s_gpio2_adr[5:2],2'b0}),
18     .wb_dat_i      (wb_m2s_gpio2_dat),
19     .wb_sel_i      (4'b1111),
20     .wb_we_i       (wb_m2s_gpio2_we),
21     .wb_stb_i      (wb_m2s_gpio2_stb),
22     .wb_dat_o      (wb_s2m_gpio2_dat),
23     .wb_ack_o      (wb_s2m_gpio2_ack),
24     .wb_err_o      (wb_s2m_gpio2_err),
25     .wb_inta_o     (gpio2_irq),
26     // External GPIO Interface
27     .ext_pad_i     (i_gpio2[4:0]),
28     .ext_pad_o     (o_gpio2[4:0]),
29     .ext_padoe_o   (en_gpio2));
30

```

Figura 4: Declaración del módulo GPIO 2

Ahora se procede a realizar los cambios necesarios en el archivo **wb_intercon.v** y en el **wb_intercon.vh**, lo que se hace en este archivo principalmente es realizar las declaraciones de las conexiones utilizadas en el archivo de **swervolf_core.v** y se llevan al *wb_mux* en donde se añade un nuevo periférico para tener en total 8. Seguidamente, se coloca una nueva instancia de la GPIO en la dirección de memoria indicada en el enunciado del ejercicio de 0x00001800 y en donde se realizan todas las conexiones con las entradas y salidas del bus wishbone.

```

1 // GPIO2
2     output [31:0] wb_gpio2_adr_o,
3     output [31:0] wb_gpio2_dat_o,
4     output [3:0] wb_gpio2_sel_o,
5     output      wb_gpio2_we_o,
6     output      wb_gpio2_cyc_o,
7     output      wb_gpio2_stb_o,
8     output [2:0] wb_gpio2_cti_o,
9     output [1:0] wb_gpio2_bte_o,
10    input  [31:0] wb_gpio2_dat_i,
11    input      wb_gpio2_ack_i,
12    input      wb_gpio2_err_i,
13    input      wb_gpio2_rty_i,

```

Figura 5: Declaración de los inputs y outputs del wb_intercon.v

```

1
2 wb_mux
3
4 #(.num_slaves (8),
5   .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00001800, 32'h00002000}),
6   .MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00}))
7 wb_mux_io
8 (.wb_clk_i (wb_clk_i),
9  .wb_rst_i (wb_rst_i),
10 .wbm_adr_i (wb_io_adr_i),
11 .wbm_dat_i (wb_io_dat_i),
12 .wbm_sel_i (wb_io_sel_i),
13 .wbm_we_i (wb_io_we_i),
14 .wbm_cyc_i (wb_io_cyc_i),
15 .wbm_stb_i (wb_io_stb_i),
16 .wbm_cti_i (wb_io_cti_i),
17 .wbm_bte_i (wb_io_bte_i),
18 .wbm_dat_o (wb_io_dat_o),
19 .wbm_ack_o (wb_io_ack_o),
20 .wbm_err_o (wb_io_err_o),
21 .wbm_rty_o (wb_io_rty_o),
22 .wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_gpio2_adr_o, wb_uart_adr_o}),
23 .wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_gpio2_dat_o, wb_uart_dat_o}),
24 .wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_gpio2_sel_o, wb_uart_sel_o}),
25 .wbs_we_o ({wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_gpio2_we_o, wb_uart_we_o}),
26 .wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_gpio2_cyc_o, wb_uart_cyc_o}),
27 .wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_gpio2_stb_o, wb_uart_stb_o}),
28 .wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_gpio2_cti_o, wb_uart_cti_o}),
29 .wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_gpio2_bte_o, wb_uart_bte_o}),
30 .wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_gpio2_dat_i, wb_uart_dat_i}),
31 .wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_gpio2_ack_i, wb_uart_ack_i}),
32 .wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_gpio2_err_i, wb_uart_err_i}),
33 .wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_gpio2_rty_i, wb_uart_rty_i}));

```

Figura 6: Modificación del MUX

```
1 // GPIO2
2 wire [31:0] wb_m2s_gpio2_adr;
3 wire [31:0] wb_m2s_gpio2_dat;
4 wire [3:0] wb_m2s_gpio2_sel;
5 wire      wb_m2s_gpio2_we;
6 wire      wb_m2s_gpio2_cyc;
7 wire      wb_m2s_gpio2_stb;
8 wire [2:0] wb_m2s_gpio2_cti;
9 wire [1:0] wb_m2s_gpio2_bte;
10 wire [31:0] wb_s2m_gpio2_dat;
11 wire      wb_s2m_gpio2_ack;
12 wire      wb_s2m_gpio2_err;
13 wire      wb_s2m_gpio2_rty;
14
```

Figura 7: Declaración de las variables en **wb_intercon.vh**

```
1 // GPIO2
2 .wb_gpio2_adr_o      (wb_m2s_gpio2_adr),
3 .wb_gpio2_dat_o      (wb_m2s_gpio2_dat),
4 .wb_gpio2_sel_o      (wb_m2s_gpio2_sel),
5 .wb_gpio2_we_o       (wb_m2s_gpio2_we),
6 .wb_gpio2_cyc_o      (wb_m2s_gpio2_cyc),
7 .wb_gpio2_stb_o      (wb_m2s_gpio2_stb),
8 .wb_gpio2_cti_o      (wb_m2s_gpio2_cti),
9 .wb_gpio2_bte_o      (wb_m2s_gpio2_bte),
10 .wb_gpio2_dat_i      (wb_s2m_gpio2_dat),
11 .wb_gpio2_ack_i      (wb_s2m_gpio2_ack),
12 .wb_gpio2_err_i      (wb_s2m_gpio2_err),
13 .wb_gpio2_rty_i      (wb_s2m_gpio2_rty),
```

Figura 8: Se instancia en **wb_intercon.vh**

Finalmente, se edita el documento de **rvfpganexys.xdc** donde se reciben las señales de los botones presionados y son asignados en la salida de **rvfpgagnexys.sv**. De acuerdo con el enunciado del ejercicio se realiza el mapeado de los pins a cada uno de los botones.

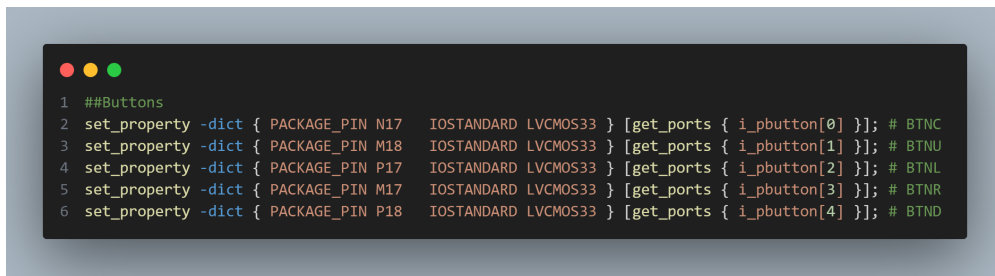


Figura 9: Cambios en el archivo rvfpganexys.xdc

3.1.3. Diseño Final

Una vez realizado las modificaciones al editar el core para incluir la GPIO2, incorporar los botones de diseño superior, asignaciones de pines físicos, interconexión del bus Wishbone y el mapeo en memoria, se realiza la síntesis y la implementación del diseño RTL en Vivado obteniendo así el Bitstream necesario para poder cargarlo a la FPGA. En la Figura 10 se puede visualizar la creación exitosa del Bitstream el cual se puede encontrar en la carpeta .runs de la implementación. Al igual en la Figura 11 se observa el dispositivo creado con todas las conexiones mostrando en las ventanas disponibles características como Potencia, Timing, DRC, metodología y otros detalles del diseño físico generado.

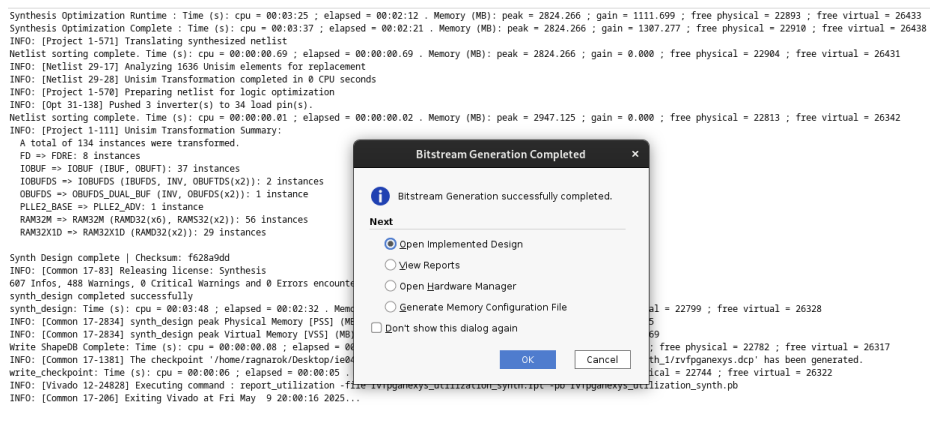


Figura 10: Generación exitosa del Bitstream



Figura 11

3.2. Ejercicio 5:

3.2.1. Descripción del problema [2]

Escribir un programa en ensamblador RISC-V y un programa en C que muestre una cuenta binaria que incremente de manera continua en los LEDs, comenzando desde el valor 1. Incluir un bucle vacío para esperar entre la visualización de cada valor incrementado, de manera que los valores sean observables por el ojo humano.

Leer el estado de BTNC a través del periférico OpenCores implementado en el Ejercicio 3 y utilizarlo para cambiar la velocidad del conteo. Además, leer el estado de BTNU a través del periférico y utilizarlo para reiniciar el conteo cada vez que se presione el botón.

3.2.2. Desarrollo de la solución

Con base al diagrama de flujo que se puede visualizar en la Figura 12, donde se realizó el programa en C para utilizar los botones BTNC y BTNU, manejando la direcciones de los GPIO correspondientes y poder habilitar los 5 componentes. Mientras que el programa realiza el conteo se habilita la opción de aumentar la velocidad o reiniciar el conteo.

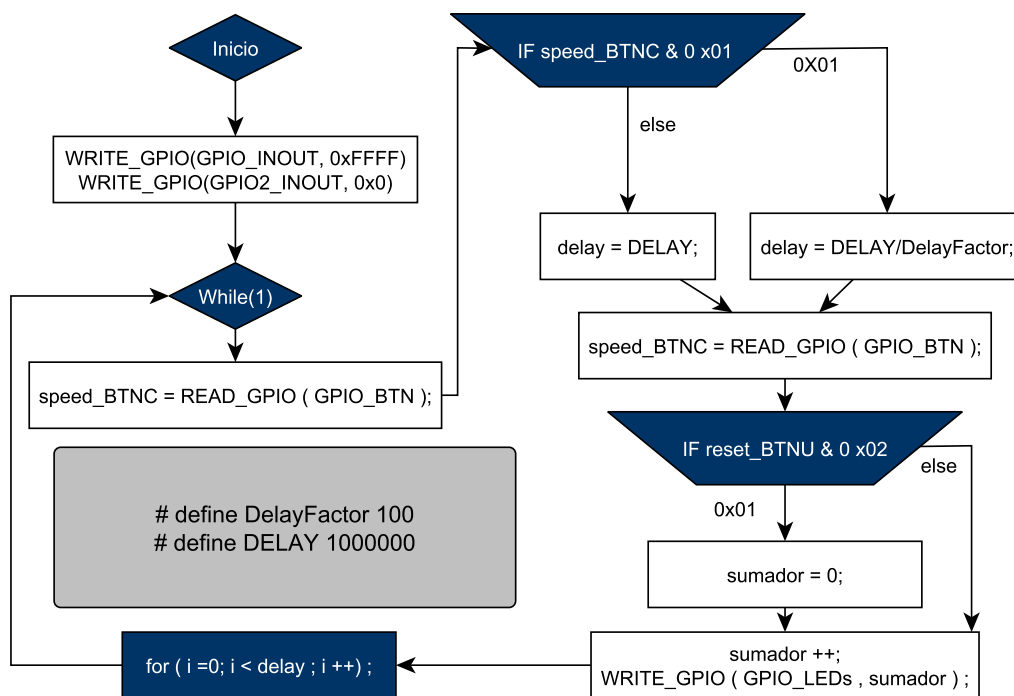


Figura 12: Diagrama de Flujo para ejercicio 5

3.2.3. Diseño Final

Con base en el diagrama realizado para la solución del ejercicio se tiene el siguiente código.

```
1 #define GPIO_SWs      0x80001400
2 #define GPIO_LEDs     0x80001404
3 #define GPIO_INOUT    0x80001408
4 #define GPIO_BTN      0x80001800
5 #define GPIO2_INOUT   0x80001808
```

```

6
7 #define DelayFactor 100
8 #define DELAY 1000000
9
10 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
11 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
12
13 int main ( void )
14 {
15     int i, speed_BTNC, reset_BTNU, delay;
16     int sumador = 0;
17
18     // Enable OE
19     WRITE_GPIO(GPIO_INOUT, 0xFFFF);
20     WRITE_GPIO(GPIO2_INOUT, 0x0);
21
22     while (1) {
23         speed_BTNC = READ_GPIO(GPIO_BTN);
24         if ((speed_BTNC & 0x01) == 0x1) {
25             delay = DELAY/DelayFactor;
26         }
27         else delay = DELAY;
28
29         reset_BTNU = READ_GPIO(GPIO_BTN);
30         if ((reset_BTNU & 0x2) == 0x2) {
31             sumador = 0;
32         }
33
34         sumador++;
35         WRITE_GPIO(GPIO_LEDS, sumador);
36
37         for (i=0; i<delay; i++);
38     }
39
40     return(0);
41 }

```

Listing 1: Código que controla los LEDs

Se puede visualizar la demostración en el siguiente video: <https://youtu.be/PushButtons>

4. Práctica 8 RVfpga: Temporizadores

4.1. Ejercicio 2:

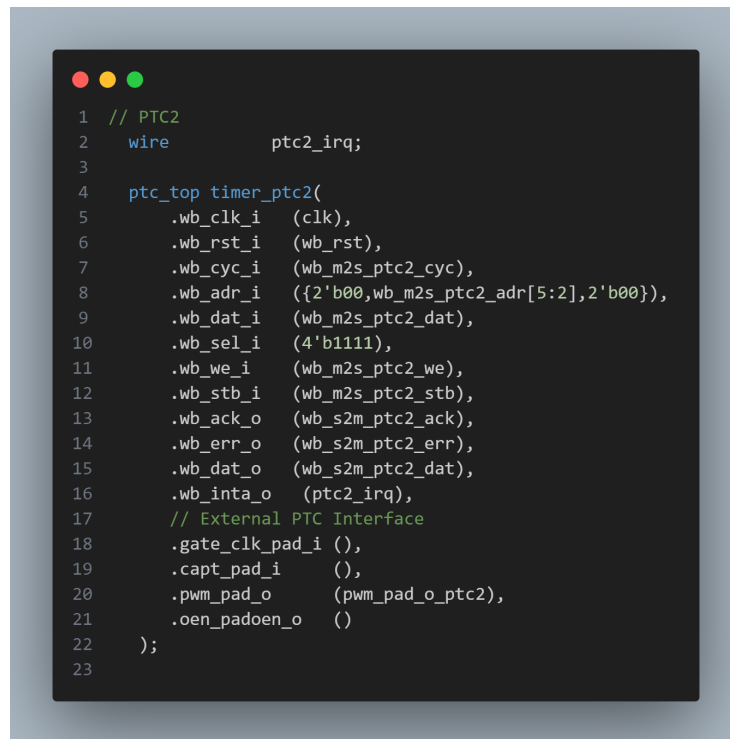
4.1.1. Descripción del problema [3]

El módulo PTC incluye una salida para generar señales PWM (Modulación por Ancho de Pulso). Esta técnica permite controlar la potencia eléctrica variando el ancho de los pulsos de una señal cuadrada. La variación del ancho de pulso está directamente relacionada con la variación en el brillo de un LED cuando este es controlado utilizando PWM. La placa RVfpgaNexys cuenta con dos LEDs tricolores. El objetivo de este ejercicio es ampliar las capacidades de la placa para poder controlar de manera independiente cada uno de los colores de uno de los LEDs, así como ajustar su brillo. Tomando como base el funcionamiento del módulo PTC y lo aprendido en las

prácticas anteriores, se busca desarrollar un control sobre el brillo y el color visible del LED utilizando los temporizadores y los módulos correspondientes, permitiendo modificar los colores y sus intensidades a través de señales PWM. Además, se hará uso de los interruptores disponibles en la tarjeta para gestionar la configuración del brillo y los colores de los LEDs.

4.1.2. Desarrollo de la solución

Teniendo de referencia los archivos editados en el ejercicio 3 del laboratorio 06, se inicia modificando el archivo `swervolf_core.v` instanciando tres veces el modulo `ptc_top`:

A screenshot of a code editor with a dark background and light-colored text. The code is Verilog, showing the instantiation of the `ptc_top` module. The code is numbered from 1 to 23. It includes a `wire` declaration for `ptc2_irq`, followed by the `ptc_top` module instantiation with various input and output signals. The signals include clock, reset, cycle, address, data, select, write enable, strobe, acknowledge, error, data output, interrupt acknowledge, and external interface signals like gate clock pad, capture pad, PWM pad, and open pad.

```
1 // PTC2
2 wire      ptc2_irq;
3
4 ptc_top timer_ptc2(
5     .wb_clk_i   (clk),
6     .wb_rst_i   (wb_rst),
7     .wb_cyc_i   (wb_m2s_ptc2_cyc),
8     .wb_adr_i   ({2'b00,wb_m2s_ptc2_adr[5:2],2'b00}),
9     .wb_dat_i   (wb_m2s_ptc2_dat),
10    .wb_sel_i   (4'b1111),
11    .wb_we_i    (wb_m2s_ptc2_we),
12    .wb_stb_i   (wb_m2s_ptc2_stb),
13    .wb_ack_o   (wb_s2m_ptc2_ack),
14    .wb_err_o   (wb_s2m_ptc2_err),
15    .wb_dat_o   (wb_s2m_ptc2_dat),
16    .wb_inta_o  (ptc2_irq),
17    // External PTC Interface
18    .gate_clk_pad_i (),
19    .capt_pad_i    (),
20    .pwm_pad_o     (pwm_pad_o_ptc2),
21    .oen_padoen_o  ()
22 );
23
```

Figura 13: PTC 2

```

1 // PTC2
2 wire          ptc2_irq;
3
4 ptc_top timer_ptc2(
5     .wb_clk_i   (clk),
6     .wb_rst_i   (wb_rst),
7     .wb_cyc_i   (wb_m2s_ptc2_cyc),
8     .wb_adr_i   ({2'b00,wb_m2s_ptc2_adr[5:2],2'b00}),
9     .wb_dat_i   (wb_m2s_ptc2_dat),
10    .wb_sel_i   (4'b1111),
11    .wb_we_i    (wb_m2s_ptc2_we),
12    .wb_stb_i   (wb_m2s_ptc2_stb),
13    .wb_ack_o   (wb_s2m_ptc2_ack),
14    .wb_err_o   (wb_s2m_ptc2_err),
15    .wb_dat_o   (wb_s2m_ptc2_dat),
16    .wb_inta_o  (ptc2_irq),
17    // External PTC Interface
18    .gate_clk_pad_i (),
19    .capt_pad_i   (),
20    .pwm_pad_o    (pwm_pad_o_ptc2),
21    .oen_padoen_o ()
22 );
23

```

Figura 14: PTC 3

```

1  // PTC4
2  wire          ptc4_irq;
3
4  ptc_top timer_ptc4(
5      .wb_clk_i    (clk),
6      .wb_rst_i    (wb_rst),
7      .wb_cyc_i    (wb_m2s_ptc4_cyc),
8      .wb_adr_i    ({2'b00,wb_m2s_ptc4_adr[5:2],2'b00}),
9      .wb_dat_i    (wb_m2s_ptc4_dat),
10     .wb_sel_i    (4'b1111),
11     .wb_we_i    (wb_m2s_ptc4_we),
12     .wb_stb_i    (wb_m2s_ptc4_stb),
13     .wb_ack_o    (wb_s2m_ptc4_ack),
14     .wb_err_o    (wb_s2m_ptc4_err),
15     .wb_dat_o    (wb_s2m_ptc4_dat),
16     .wb_inta_o    (ptc4_irq),
17     // External PTC Interface
18     .gate_clk_pad_i (),
19     .capt_pad_i    (),
20     .pwm_pad_o    (pwm_pad_o_ptc4),
21     .oen_padoen_o  ()
22 );

```

Figura 15: PTC 4

Seguidamente, se meodifica el archivo de **rvfpganexys.sv** agregando las instancias que se realizaron en el core:

```

1  .pwm_pad_o_ptc2 (pwm_pad_o_ptc2),
2  .pwm_pad_o_ptc3 (pwm_pad_o_ptc3),
3  .pwm_pad_o_ptc4 (pwm_pad_o_ptc4),

```

Figura 16: Instancia de los PWM

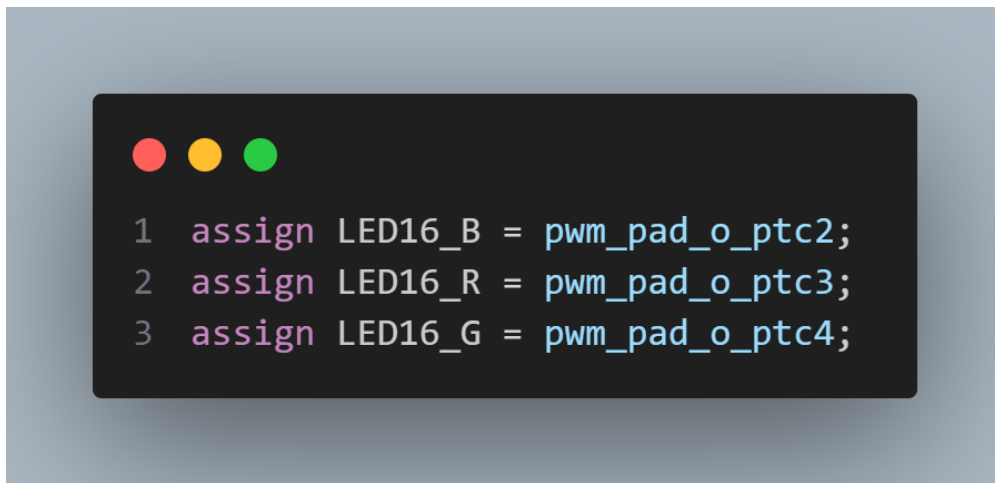


Figura 17: Asignación de los valores en el **rvfpganexys.sv**

Una vez que los nuevos módulos PTC están conectados al módulo de la Nexys, también se incorporan en los archivos **wb_intercon.v** y **wb_intercon.vh**.

```

1  // PTC2
2      output [31:0] wb_ptc2_adr_o,
3      output [31:0] wb_ptc2_dat_o,
4      output [3:0]  wb_ptc2_sel_o,
5      output        wb_ptc2_we_o,
6      output        wb_ptc2_cyc_o,
7      output        wb_ptc2_stb_o,
8      output [2:0]  wb_ptc2_cti_o,
9      output [1:0]  wb_ptc2_bte_o,
10     input  [31:0] wb_ptc2_dat_i,
11     input        wb_ptc2_ack_i,
12     input        wb_ptc2_err_i,
13     input        wb_ptc2_rty_i,
14
15 // PTC3
16     output [31:0] wb_ptc3_adr_o,
17     output [31:0] wb_ptc3_dat_o,
18     output [3:0]  wb_ptc3_sel_o,
19     output        wb_ptc3_we_o,
20     output        wb_ptc3_cyc_o,
21     output        wb_ptc3_stb_o,
22     output [2:0]  wb_ptc3_cti_o,
23     output [1:0]  wb_ptc3_bte_o,
24     input  [31:0] wb_ptc3_dat_i,
25     input        wb_ptc3_ack_i,
26     input        wb_ptc3_err_i,
27     input        wb_ptc3_rty_i,
28
29 // PTC4
30     output [31:0] wb_ptc4_adr_o,
31     output [31:0] wb_ptc4_dat_o,
32     output [3:0]  wb_ptc4_sel_o,
33     output        wb_ptc4_we_o,
34     output        wb_ptc4_cyc_o,
35     output        wb_ptc4_stb_o,
36     output [2:0]  wb_ptc4_cti_o,
37     output [1:0]  wb_ptc4_bte_o,
38     input  [31:0] wb_ptc4_dat_i,
39     input        wb_ptc4_ack_i,
40     input        wb_ptc4_err_i,
41     input        wb_ptc4_rty_i,

```

Figura 18: Declaración de las variables `wb_intercon.v`

```

1 wb_mux
2 #(.num_slaves (11),
3 .MATCH_ADDR ((32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001240, 32'h00001280, 32'h000012c0, 32'h00001400, 32'h00001800, 32'h00002000)),
4 .MATCH_MASK ((32'hffff0000, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00)))
5 wb_mux_io
6 (.wb_clk_i (wb_clk_i),
7 .wb_rst_i (wb_rst_i),
8 .wb_addr_i (wb_io_addr_i),
9 .wb_dat_i (wb_io_dat_i),
10 .wb_sel_i (wb_io_sel_i),
11 .wb_we_i (wb_io_we_i),
12 .wb_cyc_i (wb_io_cyc_i),
13 .wb_stb_i (wb_io_stb_i),
14 .wb_cti_i (wb_io_cti_i),
15 .wb_bte_i (wb_io_bte_i),
16 .wb_dat_o (wb_io_dat_o),
17 .wb_ack_o (wb_io_ack_o),
18 .wb_err_o (wb_io_err_o),
19 .wb_rty_o (wb_io_rty_o),
20 .wb_sys_adr_o (wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_ptc2_adr_o, wb_ptc3_adr_o, wb_ptc4_adr_o, wb_gpio_adr_o, wb_gpio2_adr_o, wb_uart_adr_o)),
21 .wb_sys_dat_o (wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_ptc2_dat_o, wb_ptc3_dat_o, wb_ptc4_dat_o, wb_gpio_dat_o, wb_gpio2_dat_o, wb_uart_dat_o)),
22 .wb_sys_sel_o (wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_ptc2_sel_o, wb_ptc3_sel_o, wb_ptc4_sel_o, wb_gpio_sel_o, wb_gpio2_sel_o, wb_uart_sel_o)),
23 .wb_sys_we_o (wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_ptc2_we_o, wb_ptc3_we_o, wb_ptc4_we_o, wb_gpio_we_o, wb_gpio2_we_o, wb_uart_we_o)),
24 .wb_sys_cyc_o (wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_ptc2_cyc_o, wb_ptc3_cyc_o, wb_ptc4_cyc_o, wb_gpio_cyc_o, wb_gpio2_cyc_o, wb_uart_cyc_o)),
25 .wb_sys_stb_o (wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_ptc2_stb_o, wb_ptc3_stb_o, wb_ptc4_stb_o, wb_gpio_stb_o, wb_gpio2_stb_o, wb_uart_stb_o)),
26 .wb_sys_cti_o (wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_ptc2_cti_o, wb_ptc3_cti_o, wb_ptc4_cti_o, wb_gpio_cti_o, wb_gpio2_cti_o, wb_uart_cti_o)),
27 .wb_sys_bte_o (wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_ptc2_bte_o, wb_ptc3_bte_o, wb_ptc4_bte_o, wb_gpio_bte_o, wb_gpio2_bte_o, wb_uart_bte_o)),
28 .wb_sys_dat_i (wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_ptc2_dat_i, wb_ptc3_dat_i, wb_ptc4_dat_i, wb_gpio_dat_i, wb_gpio2_dat_i, wb_uart_dat_i)),
29 .wb_sys_ack_i (wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_ptc2_ack_i, wb_ptc3_ack_i, wb_ptc4_ack_i, wb_gpio_ack_i, wb_gpio2_ack_i, wb_uart_ack_i)),
30 .wb_sys_err_i (wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_ptc2_err_i, wb_ptc3_err_i, wb_ptc4_err_i, wb_gpio_err_i, wb_gpio2_err_i, wb_uart_err_i)),
31 .wb_sys_rty_i (wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_ptc2_rty_i, wb_ptc3_rty_i, wb_ptc4_rty_i, wb_gpio_rty_i, wb_gpio2_rty_i, wb_uart_rty_i));
32

```

Figura 19: Modificaciones en el mux

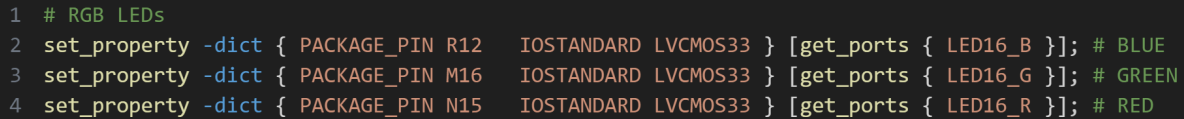
```

1 // PTC2
2 .wb_ptc2_adr_o (wb_m2s_ptc2_adr),
3 .wb_ptc2_dat_o (wb_m2s_ptc2_dat),
4 .wb_ptc2_sel_o (wb_m2s_ptc2_sel),
5 .wb_ptc2_we_o (wb_m2s_ptc2_we),
6 .wb_ptc2_cyc_o (wb_m2s_ptc2_cyc),
7 .wb_ptc2_stb_o (wb_m2s_ptc2_stb),
8 .wb_ptc2_cti_o (wb_m2s_ptc2_cti),
9 .wb_ptc2_bte_o (wb_m2s_ptc2_bte),
10 .wb_ptc2_dat_i (wb_s2m_ptc2_dat),
11 .wb_ptc2_ack_i (wb_s2m_ptc2_ack),
12 .wb_ptc2_err_i (wb_s2m_ptc2_err),
13 .wb_ptc2_rty_i (wb_s2m_ptc2_rty),
14
15 // PTC3
16 .wb_ptc3_adr_o (wb_m2s_ptc3_adr),
17 .wb_ptc3_dat_o (wb_m2s_ptc3_dat),
18 .wb_ptc3_sel_o (wb_m2s_ptc3_sel),
19 .wb_ptc3_we_o (wb_m2s_ptc3_we),
20 .wb_ptc3_cyc_o (wb_m2s_ptc3_cyc),
21 .wb_ptc3_stb_o (wb_m2s_ptc3_stb),
22 .wb_ptc3_cti_o (wb_m2s_ptc3_cti),
23 .wb_ptc3_bte_o (wb_m2s_ptc3_bte),
24 .wb_ptc3_dat_i (wb_s2m_ptc3_dat),
25 .wb_ptc3_ack_i (wb_s2m_ptc3_ack),
26 .wb_ptc3_err_i (wb_s2m_ptc3_err),
27 .wb_ptc3_rty_i (wb_s2m_ptc3_rty),
28
29 // PTC4
30 .wb_ptc4_adr_o (wb_m2s_ptc4_adr),
31 .wb_ptc4_dat_o (wb_m2s_ptc4_dat),
32 .wb_ptc4_sel_o (wb_m2s_ptc4_sel),
33 .wb_ptc4_we_o (wb_m2s_ptc4_we),
34 .wb_ptc4_cyc_o (wb_m2s_ptc4_cyc),
35 .wb_ptc4_stb_o (wb_m2s_ptc4_stb),
36 .wb_ptc4_cti_o (wb_m2s_ptc4_cti),
37 .wb_ptc4_bte_o (wb_m2s_ptc4_bte),
38 .wb_ptc4_dat_i (wb_s2m_ptc4_dat),
39 .wb_ptc4_ack_i (wb_s2m_ptc4_ack),
40 .wb_ptc4_err_i (wb_s2m_ptc4_err),
41 .wb_ptc4_rty_i (wb_s2m_ptc4_rty),

```

Figura 20: Instancia de los PTC

Finalmente, se propone la conexión con pines de la tarjeta en el archivo `rvfpganexys.xdc`. A continuación, se presenta los cambios realizados para cada color.



```
1 # RGB LEDs
2 set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }]; # BLUE
3 set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }]; # GREEN
4 set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED16_R }]; # RED
```

Figura 21: Asignación de pines a cada color de LED

4.1.3. Diseño Final

- **Asignación de pines físicos (XDC):**

En el archivo `rvfpganexys.xdc`, se asignaron tres pines físicos del FPGA a los colores del LED tricolor (LED16_R, LED16_G, LED16_B), utilizando el estándar de voltaje LVCMOS33. Esta configuración permite controlar cada canal de color mediante una señal lógica.

- **Definición de salidas PWM en el diseño superior:**

En `rvfpganexys.sv`, se definieron las salidas `pwm_pad_o_ptc2`, `ptc3` y `ptc4` que corresponden a las salidas PWM de los temporizadores PTC2, PTC3 y PTC4.

- **Exposición de señales en el núcleo:**

En `swerwolf_core.v`, se declararon las señales `pwm_pad_o_ptc2`, `ptc3` y `ptc4` como salidas del módulo superior. Estas señales provienen de los módulos `ptc_top` encargados de generar las señales PWM.

- **Instanciación de módulos de temporizador:**

En el mismo archivo, se instanciaron tres módulos `ptc_top` correspondientes a los temporizadores PTC2, PTC3 y PTC4. Cada instancia se conecta a su respectiva interfaz del bus Wishbone y genera su propia salida PWM, que luego se enruta al LED.

- **Interconexión Wishbone:**

En `wb_intercon.vh` y `wb_intercon.v`, se añadieron las señales del bus Wishbone necesarias para los tres temporizadores. Esto incluye las señales de dirección, datos, control y respuesta, permitiendo así que el procesador acceda a los registros de cada temporizador mediante memoria mapeada.

- **Síntesis**

Finalmente, en Vivado se genera el Bitstream del diseño para implementarlo y ejecutarlo en la FPGA donde tiene habilitado todos los cambios para la utilización los switches para manejar los colores y sus diferentes intensidades con los parámetros establecidos para cada canal de color.

4.2. Ejercicio 3:

4.2.1. Descripción del problema [3]

En este ejercicio, se debe implementar un programa que utilice el nuevo periférico para controlar el LED tricolor, utilizando el valor proporcionado por los 16 interruptores disponibles en la placa. Los interruptores se usarán de la siguiente manera:

- Los 5 interruptores más a la derecha se utilizarán para ajustar el ciclo de trabajo del color azul del LED.
- Los siguientes 5 interruptores se utilizarán para ajustar el ciclo de trabajo del color verde.
- Los siguientes 5 interruptores se utilizarán para ajustar el ciclo de trabajo del color rojo.
- El interruptor más a la izquierda no se utilizará.

El programa debe ser implementado de la siguiente manera:

- **Parte a:** Escribir el programa en ensamblador RISC-V. El programa debe leer los valores de los interruptores y controlar el ciclo de trabajo (duty cycle) de cada color del LED tricolor de acuerdo con los valores de los interruptores correspondientes.
- **Parte b:** Escribir el programa en C. El programa en C debe realizar la misma función que el programa en ensamblador RISC-V, es decir, leer los interruptores y ajustar el ciclo de trabajo de los colores azul, verde y rojo del LED tricolor.

4.2.2. Desarrollo de la solución

Para el desarrollo de la solución se basó al diagrama de flujo para el desarrollo de ambos códigos, para ensamblador y en C. En la Figura 22 se puede observar en general los pasos para habilitar los switches y definir los parámetros. Para los dos códigos se realizó dos códigos diferentes pero con una misma guía de habilitar y manejar los parámetros.

Código en C

- Como función principal por medio del CTRL se activa PWM para cada color, definiendo un LRC como un máximo ciclo de trabajo inicial.
- En el bucle principal se extraen 5 bits y dependiendo al color se desplaza, por ejemplo el Rojo son 0x1F, para el Verde lo mismo pero se desplaza 10 y el azul se desplaza 5.
- La función ActiveWorkCycle cuenta los bits en 1 dentro de los 5 bits de todos los switches.
- La función WorkCycle dependiendo el número de bits que están habilitados calculados anteriormente se reduce el valor de PWM en porcentajes diferentes.

Código en RISCv

- Se habilita la escritura a todos los pines del GPIO utilizando la dirección `GPIO_INOUT` con el valor `0xFFFF`.
- En el bucle principal, se leen los switches desde la dirección `GPIO_SWs` y se desplazan 16 bits para obtener los 15 más significativos, que se usan como entradas para configurar el PWM de los colores.
- Para cada color (Rojo, Azul, Verde) se realiza lo siguiente:
 - Se fija el valor de `HRC` en `0xF`, que representa el inicio del ciclo PWM.
 - Se extraen 5 bits correspondientes a cada color:
 - Rojo: `t0 & 0x1F`
 - Azul: se hace `srl t0, t0, 5` y luego `t0 & 0x1F`
 - Verde: se vuelve a hacer `srl t0, t0, 5` y `t0 & 0x1F`
 - Se suma `0xF` al número de bits activos para formar el valor de `LRC`, que determina el ciclo de trabajo (duty cycle).
 - Se escribe el valor `0x80` en `CTRL` para preparar el PWM, seguido de `0x9` para activarlo.
- Se incluye una pausa (delay) con un contador simple que decrementa hasta cero antes de repetir el ciclo.
- El programa es un bucle infinito que constantemente actualiza los valores de PWM dependiendo de cuántos switches están activos por color.

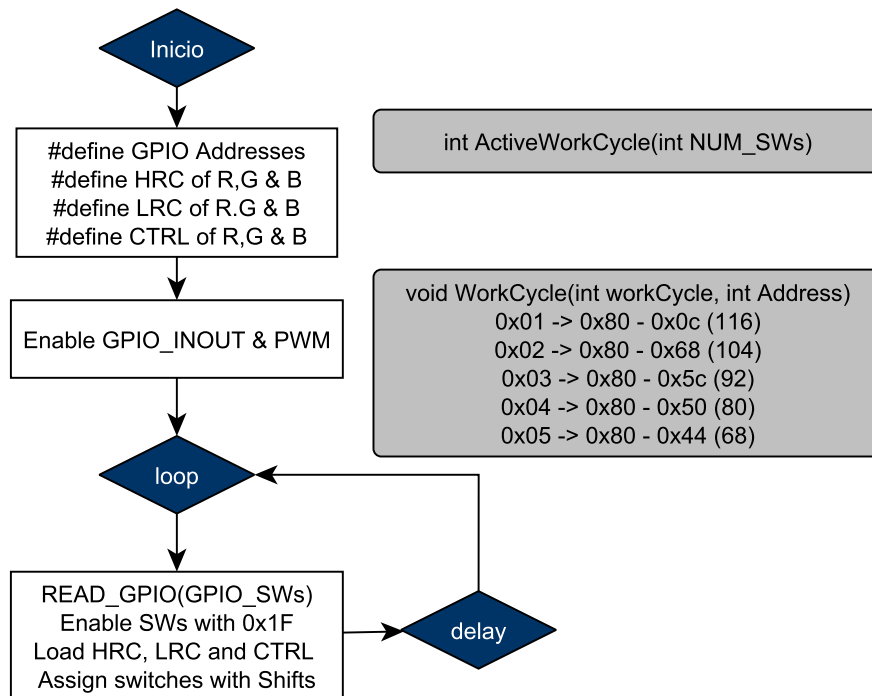


Figura 22: Diagrama de Flujo para ejercicio 3

4.2.3. Diseño Final

Finalmente, con base al diagrama anteriormente enseñado se realizó el siguiente código en ensamblador

```
1  #define GPIO_SWs      0x80001400
2  #define GPIO_LEDs     0x80001404
3  #define GPIO_INOUT    0x80001408
4
5  #define LED_R         0x80001240
6  #define HRC_R         0x80001244
7  #define LRC_R         0x80001248
8  #define CTRL_R        0x8000124c
9
10 #define LED_B         0x80001280
11 #define HRC_B         0x80001284
12 #define LRC_B         0x80001288
13 #define CTRL_B        0x8000128c
14
15 #define LED_G         0x800012c0
16 #define HRC_G         0x800012c4
17 #define LRC_G         0x800012c8
18 #define CTRL_G        0x800012cc
19
20
21 .globl main
22 main:
23
24 # Enable
25 li t0, 0xFFFF
26 li t1, GPIO_INOUT
27 sw t0, 0(t1)
28
29 next:
30
31     li a1, GPIO_SWs
32     lw t0, 0(a1)
33
34     # RED
35     li a0, GPIO_LEDs
36     srl t0, t0, 16
37     sw t0, 0(a0)
38
39     li t1, HRC_R
40     li t2, 0xF
41     sw t2, 0(t1)
42
43     and t2, t0, 0x1F
44     add t2, t2, 0xF
45     li t1, LRC_R
46     sw t2, 0(t1)
47
48     li t2, 0x80
49     li t1, CTRL_R
50     sw t2, 0(t1)
51     li t2, 0x9
52     li t1, CTRL_R
```

```

53     sw t2, 0(t1)
54
55     # BLUE
56     li t1, HRC_B
57     li t2, 0xF
58     sw t2, 0(t1)
59
60     srl t0, t0, 5
61     and t2, t0, 0x1F
62     add t2, t2, 0xF
63     li t1, LRC_B
64     sw t2, 0(t1)
65
66     li t2, 0x80
67     li t1, CTRL_B
68     sw t2, 0(t1)
69     li t2, 0x9
70     li t1, CTRL_B
71     sw t2, 0(t1)
72
73     # GREEN
74     li t1, HRC_G
75     li t2, 0xF
76     sw t2, 0(t1)
77
78     srl t0, t0, 5
79     and t2, t0, 0x1F
80     add t2, t2, 0xF
81     li t1, LRC_G
82     sw t2, 0(t1)
83
84     li t2, 0x80
85     li t1, CTRL_G
86     sw t2, 0(t1)
87     li t2, 0x9
88     li t1, CTRL_G
89     sw t2, 0(t1)
90
91     li t3, 10000
92
93     delay:
94         add t3, t3, -1
95         bge t3, zero, delay
96
97     beq zero, zero, next
98
99 .end

```

Listing 2: Código para control de LEDs y switches en ensamblador

De igual forma, se realizó la implementación del código en C con base en el diagrama realizado.

```

1 // PTC BLUE
2 #define LED_B      0x80001280
3 #define LRC_B      0x80001288
4 #define HRC_B      0x80001284
5 #define CTRL_B     0x8000128C

```

```

6
7 // PTC GREEN
8 #define LED_G      0x800012C0
9 #define LRC_G      0x800012C8
10 #define HRC_G      0x800012C4
11 #define CTRL_G     0x800012CC
12
13 // PTC RED
14 #define LED_R      0x80001240
15 #define LRC_R      0x80001248
16 #define HRC_R      0x80001244
17 #define CTRL_R     0x8000124C
18
19 // SWITCHES
20 #define GPIO_SWs    0x80001400
21 #define GPIO_INOUT  0x80001408
22
23
24 #define REAS_GPIO(dir) (*(volatile unsigned *)dir)
25 #define WRITE_GPIO(dir, value) {(*(volatile unsigned *)dir) = (value);}
26
27 // PARAMETROS
28 #define PORCENTAJE  0x0C
29
30 int main (void){
31
32     WRITE_GPIO(CTRL_R, 1);
33     WRITE_GPIO(LRC_R, 0x80);
34
35     WRITE_GPIO(CTRL_G, 1);
36     WRITE_GPIO(LRC_G, 0x80);
37
38     WRITE_GPIO(CTRL_B, 1);
39     WRITE_GPIO(LRC_B, 0x80);
40
41     WRITE_GPIO(GPIO_INOUT, 0xFFFF);
42
43     unsigned int SWs, RED_SW, GREEN_SW, BLUE_SW;
44     unsigned int TEMP_SW = 1;
45
46     while (1){
47         SWs = REAS_GPIO(GPIO_SWs);
48         SWs = SWs >> 16;
49
50         if (TEMP_SW != SWs) {
51
52             RED_SW = SWs & 0x1F;
53             RED_SW = ActiveWorkCycle(RED_SW);
54             WorkCycle(RED_SW, HRC_R);
55
56             GREEN_SW = SWs >> 10;
57             GREEN_SW = GREEN_SW & 0x1F;
58             GREEN_SW = ActiveWorkCycle(GREEN_SW);
59             WorkCycle(GREEN_SW, HRC_G);
60
61             BLUE_SW = SWs >> 5;

```

```

62     BLUE_SW = BLUE_SW & 0x1F;
63     BLUE_SW = ActiveWorkCycle(BLUE_SW);
64     WorkCycle(BLUE_SW, HRC_B);
65
66     TEMP_SW = SWs;
67 }
68 }
69 }
70
71
72 int ActiveWorkCycle(int NUM_SWs){
73     int CountSW = 0;
74     int i;
75
76     for (i = 0; i < 5; i++){
77
78         if (NUM_SWs & 0x1){
79             CountSW++;
80         }
81
82         NUM_SWs = NUM_SWs >> 1;
83     }
84     return CountSW;
85 }
86
87
88 void WorkCycle(int workCycle, int Address){
89
90     switch (workCycle){
91         case 0x1:
92             WRITE_GPIO(Address, 0x80 - PORCENTAJE);
93             break;
94         case 0x2:
95             WRITE_GPIO(Address, 0x80 - 2*PORCENTAJE);
96             break;
97         case 0x3:
98             WRITE_GPIO(Address, 0x80 - 3*PORCENTAJE);
99             break;
100        case 0x4:
101            WRITE_GPIO(Address, 0x80 - 4*PORCENTAJE);
102            break;
103        case 0x5:
104            WRITE_GPIO(Address, 0x80 - 5*PORCENTAJE);
105            break;
106        default:
107            WRITE_GPIO(Address, 0x80);
108            break;
109    }
110 }

```

Listing 3: Código para control de LEDs y switches en C

Se puede visualizar la demostración en el siguiente video: <https://youtu.be/RGBSwitches>

5. Problemas y complicaciones

- Se tuvo un problema inicial con la detección de botones, se habían realizado todos los cambios descritos para el ejercicio 3 del laboratorio 6. En una revisión exhaustiva notamos que una variable del botón tenía un nombre en uno de los archivos lo que hacía que el programa no reconociera todos los botones.
- Se tuvo problemas con el git, y no dejaba subir los archivos al repositorio. Por lo que se tuvo que avanzar de forma local, los errores proporcionados de acuerdo con la documentación del git de la UCR indicaban que sucedían al tener un doble factor de autenticación. Pero al revisar el repositorio no se contaba con esa opción activada. La solución fue crear un token que nos permitió subir los archivos al repositorio.
- Se tuvo que volver a realizar el ejercicio 2 del laboratorio 08 debido a que hubo un problema con git y al subir el ejercicio 3 borró los cambios realizados anteriormente y los archivos como **rvfpganexys.sv** no tenían las modificaciones realizadas para los LEDs.

6. Conclusiones y recomendaciones

- Se trabajó con vivado conociendo el software más de cerca permitiendo la realización de la síntesis e implementación de un diseño RTL para la FPGA implementando cambios en la microarquitectura Wishbone del SoC SweRVolfX.
- La integración de un segundo módulo GPIO2 en el diseño permitió ampliar la capacidad de entrada del sistema, habilitando la lectura de los cinco botones físicos de la placa Nexys A7 de forma independiente al resto de los periféricos.
- La correcta integración de los temporizadores PWM con los LEDs tricolores requiere una sincronización precisa entre hardware y software, especialmente en la asignación de direcciones de memoria, enrutamiento de señales en el diseño superior y interconexion al multiplexor.
- Es altamente recomendable mantener una nomenclatura rigurosa y consistente al nombrar señales, cables y direcciones en el entorno de diseño digital. La más mínima diferencia, como un espacio no intencional o una coma mal colocada, puede generar errores difíciles de rastrear que afectan directamente la funcionalidad del diseño. Además, es fundamental verificar cada conexión lógica tanto en el código como en el archivo de restricciones (.xdc) antes de proceder a la síntesis. Una organización clara y un control de versiones adecuado pueden facilitar significativamente el proceso de depuración y verificación del sistema.

Referencias

- [1] Imagination University Programme, *RVfpga Lab 5: Creating a Vivado Project*, ver. 2.2, Available from the Imagination University Programme, Imagination Technologies, 2022.
- [2] Imagination University Programme, *RVfpga Lab 6: Introduction to I/O*, ver. 2.2, Available from the Imagination University Programme, Imagination Technologies, 2022.
- [3] Imagination University Programme, *RVfpga Lab 8: Timers*, ver. 2.2, Available from the Imagination University Programme, Imagination Technologies, 2022.