APENDICE A

En este apéndice, se ha añadido todo el código fuente que se ha usado durante el desarrollo del proyecto. En los apartados A.1 y A.2 se encuentra el código en VHDL correspondiente al diseño hardware para controlar la matriz de LEDs. En los capítulos A.3, A.4 y A.5, se encuentra el código en C para las aplicaciones realizadas en Vitis para controlar los LEDS, el teclado PmodKYPD, y el juego, respectivamente.

A.1 LEDS_RGB_32X8.vhd:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    20:49:18 11/22/2022
-- Design Name:
-- Module Name:    led_rgb_8x8_02 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity led_rgb_8x8_02 is
    Port ( clk : in  STD_LOGIC;
           init : in  STD_LOGIC;
                    reset: in STD_LOGIC;
           fila0 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila1 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila2 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila3 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila4 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila5 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila6 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila7 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila8 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila9 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila10 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila11 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila12 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila13 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila14 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila15 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila16 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila17 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila18 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila19 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila20 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila21 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila22 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila23 : in  STD_LOGIC_VECTOR (31 downto 0);
```

```
            fila24 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila25 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila26 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila27 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila28 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila29 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila30 : in  STD_LOGIC_VECTOR (31 downto 0);
            fila31 : in  STD_LOGIC_VECTOR (31 downto 0);
            dato_rgb: out STD_LOGIC);
end led_rgb_8x8_02;

architecture Behavioral of led_rgb_8x8_02 is

constant t0h: integer:= 39;
constant t0l: integer:= 83;
constant t1h: integer:= 79;
constant t1l: integer:= 43;

signal color_act: std_logic_vector(23 downto 0); -- Color actual
signal cont_v: integer range 0 to 85; -- Contador de tiempo de '0' y '1'
signal cont_bits: integer range 0 to 23; -- Contador del numero de bit que vamos cargando en cada led // SE
MANTIENE EN ESTE PROGRAMA
signal cont_led:integer range 0 to 7; -- Contador que nos indica el numero de led que estamos cargando //HABRIA
QUE RECORTARLO HASTA 8
signal cont_reg:integer range 0 to 31; -- Contador que nos indica por el numero de registro que vamos cargando
signal color_act_4bits:std_logic_vector(3 downto 0); --
signal color_act_8bits:std_logic_vector(31 downto 0);

type statetype is (s0,s1,s2,s3,sf);
signal estado:statetype := s0;

-- COLORES

constant apagar: std_logic_vector(23 downto 0):=(others => '0'); -- 0000
constant red: std_logic_vector(23 downto 0):=("000011110000000000000000"); -- 0001
constant green: std_logic_vector(23 downto 0):=("000000000000011110000000"); -- 0010
constant blue: std_logic_vector(23 downto 0):=("000000000000000000001111"); -- 0011
constant magenta: std_logic_vector(23 downto 0):=("000011110000000000001111"); -- 0100
constant yellow: std_logic_vector(23 downto 0):=("000011110000111100000000"); -- 0101
constant cyan: std_logic_vector(23 downto 0):=("000000000000111100001111"); -- 0110
constant gray: std_logic_vector(23 downto 0) :=("000011110000011100000111"); -- 0100
constant white: std_logic_vector(23 downto 0):=("000011110000111100001111"); -- 1000


begin


-- Este programa trata de cargar los datos de los registros (in) en las salidas al pulsar init en la salida
-- Se iran cargando en la salida desde el registro 0 hasta el registro 31
-- Cada entrada de 31 bits contiene el color asignado a 8 leds, el cual esta codificado a traves de 4 bits de
color
-- Lo primero que tenemos que hacer antes de cargar el color de un led, es decodificar los 4 bits de color y
cargar en color_act, el
-- color que se va a cargar en ese led.
-- Una vez se carguen los 8 leds de una filaxx, pasaremos a la siguiente fila, y asi hasta llegar a la fila 31,
que es donde habremos
-- terminado nuestro trabajo.

-- Podriamos crear una matriz para cargar ahi el valor de cada led. (matriz 32x32).

-- En el otro programa que haremos, se asignaran unos switches como entrada, para ir cargando el color de los
leds que se va a mostrar.
-- Lo suyo seria asignar algun boton que no sea ni el de reset ni el init, para ir cargando los pertinentes
colores en cada registro.
-- A su vez, lo suyo seria ir iluminando leds de la placa, para ver por (ACABAR)

-- PARTE COMBINACIONAL

-- PASAR LAS ENTRADAS DIRECTAMENTE A LA ROM Y TRABAJAR DESDE LA ROM

-- DECODIFICADOR COLOR_ACT_4BITS => COLOR_ACT // TENER EN CUENTA QUE COLOR_ACT_4BITS SE ACTUALIZA SOLO AL
FINALIZAR EL PROCESS
-- POR TANTO, ANTES DE HACER UN CICLO HAY QUE CARGAR AQUI EL VALOR FUTURO DEL COLOR_ACT_4BITS

with color_act_4bits select
        color_act <= apagar when "0000",
        red when "0001",
        green when "0010",
        blue when "0011",
        magenta  when "0100",
        yellow when "0101",
        cyan when "0110",
        gray when "0111",
        white when "1000",
        apagar when others;


-- PARTE SECUENCIAL

process(clk, reset)
```

```
begin

        if reset = '1' then
                estado <= s0;
                cont_v <= 0;
                cont_bits <= 0;
                cont_led <= 0;
        elsif rising_edge(clk) then

                case estado is

                when s0 => if init = '1' then
-- En este estado inicializamos todos los contadores y comenzamos cargando los datos del primer registro
-- en nuestra ROM
                                        cont_bits <= 23;
                                        cont_led <= 0;
                                        cont_reg <= 0;
-- Cargamos el color del primer led en la variable color_act_4bits
                                        color_act_8bits <= fila0;
                                        color_act_4bits <= fila0(3 downto 0);
                                        estado <= s1;
-- Este valor se actualiza al finalizar el process
                                end if;

                  when s1 => dato_rgb <= '1';
-- Empezamos a enviar el '1' del bit de color correspondiente
                                    if color_act(cont_bits) = '0' then
                                        cont_v <= t0h;
                                    else
                                        cont_v <= t1h;
                                    end if;
                                    estado <= s2;

                  when s2 => cont_v <= cont_v - 1;
-- Acabamos de enviar el '1' del bit de color correspondiente
                                    dato_rgb <= '1';
                                    if cont_v = 0 then
-- Empezamos a enviar el '0' del bit de color correspondiente
                                        dato_rgb <= '0';
                                        if color_act(cont_bits) = '0' then
                                            cont_v <= t0l;
                                        else
                                            cont_v <= t1l;
                                        end if;
                                        estado <= s3;
                                    end if;

                when s3 => cont_v <= cont_v - 1;
-- Acabamos de enviar el '0' del bit de color correspondiente
                                dato_rgb <= '0';
                                if cont_v = 0 then
                                    if cont_bits = 0 then
-- Si hemos transmitido todos los bits pasariamos al siguiente led
                                        if cont_led = 7 then
-- Si hemos acabado todos los leds pasamos a la siguiente fila de la ROM
                                            if cont_reg = 31 then
-- Si ya hemos cargado todos pasamos al estado final
                                                cont_led <= 0;
                                                estado <= sf;
                                            else
-- Si no los hemos pasado todos, incrementamos el registro
                                                cont_reg <= cont_reg + 1;
                                                cont_led <= 0;
                                                cont_bits <= 23;
                                                if (cont_reg + 1) = 1 then
                                                        color_act_8bits <= fila1;
                                                        color_act_4bits <= fila1(31 downto 28);
                                                    elsif (cont_reg + 1) = 2 then
                                                        color_act_8bits <= fila2;
                                                        color_act_4bits <= fila2(3 downto 0);
                                                    elsif (cont_reg + 1) = 3 then
                                                        color_act_8bits <= fila3;
                                                        color_act_4bits <= fila3(31 downto 28);
                                                    elsif (cont_reg + 1) = 4 then
                                                        color_act_8bits <= fila4;
                                                        color_act_4bits <= fila4(3 downto 0);
                                                    elsif (cont_reg + 1) = 5 then
                                                        color_act_8bits <= fila5;
                                                        color_act_4bits <= fila5(31 downto 28);
                                                    elsif (cont_reg + 1) = 6 then
                                                        color_act_8bits <= fila6;
                                                        color_act_4bits <= fila6(3 downto 0);
                                                    elsif (cont_reg + 1) = 7 then
                                                        color_act_8bits <= fila7;
                                                        color_act_4bits <= fila7(31 downto 28);
                                                    elsif (cont_reg + 1) = 8 then
                                                        color_act_8bits <= fila8;
                                                        color_act_4bits <= fila8(3 downto 0);
                                                    elsif (cont_reg + 1) = 9 then
```

```
                    color_act_8bits <= fila9;
                    color_act_4bits <= fila9(31 downto 28);
                elsif (cont_reg + 1) = 10 then
                    color_act_8bits <= fila10;
                    color_act_4bits <= fila10(3 downto 0);
                elsif (cont_reg + 1) = 11 then
                    color_act_8bits <= fila11;
                    color_act_4bits <= fila11(31 downto 28);
                elsif (cont_reg + 1) = 12 then
                    color_act_8bits <= fila12;
                    color_act_4bits <= fila12(3 downto 0);
                elsif (cont_reg + 1) = 13 then
                    color_act_8bits <= fila13;
                    color_act_4bits <= fila13(31 downto 28);
                elsif (cont_reg + 1) = 14 then
                    color_act_8bits <= fila14;
                    color_act_4bits <= fila14(3 downto 0);
                elsif (cont_reg + 1) = 15 then
                    color_act_8bits <= fila15;
                    color_act_4bits <= fila15(31 downto 28);
                elsif (cont_reg + 1) = 16 then
                    color_act_8bits <= fila16;
                    color_act_4bits <= fila16(3 downto 0);
                elsif (cont_reg + 1) = 17 then
                    color_act_8bits <= fila17;
                    color_act_4bits <= fila17(31 downto 28);
                elsif (cont_reg + 1) = 18 then
                    color_act_8bits <= fila18;
                    color_act_4bits <= fila18(3 downto 0);
                elsif (cont_reg + 1) = 19 then
                    color_act_8bits <= fila19;
                    color_act_4bits <= fila19(31 downto 28);
                elsif (cont_reg + 1) = 20 then
                    color_act_8bits <= fila20;
                    color_act_4bits <= fila20(3 downto 0);
                elsif (cont_reg + 1) = 21 then
                    color_act_8bits <= fila21;
                    color_act_4bits <= fila21(31 downto 28);
                elsif (cont_reg + 1) = 22 then
                    color_act_8bits <= fila22;
                    color_act_4bits <= fila22(3 downto 0);
                elsif (cont_reg + 1) = 23 then
                    color_act_8bits <= fila23;
                    color_act_4bits <= fila23(31 downto 28);
                elsif (cont_reg + 1) = 24 then
                    color_act_8bits <= fila24;
                    color_act_4bits <= fila24(3 downto 0);
                elsif (cont_reg + 1) = 25 then
                    color_act_8bits <= fila25;
                    color_act_4bits <= fila25(31 downto 28);
                elsif (cont_reg + 1) = 26 then
                    color_act_8bits <= fila26;
                    color_act_4bits <= fila26(3 downto 0);
                elsif (cont_reg + 1) = 27 then
                    color_act_8bits <= fila27;
                    color_act_4bits <= fila27(31 downto 28);
                elsif (cont_reg + 1) = 28 then
                    color_act_8bits <= fila28;
                    color_act_4bits <= fila28(3 downto 0);
                elsif (cont_reg + 1) = 29 then
                    color_act_8bits <= fila29;
                    color_act_4bits <= fila29(31 downto 28);
                elsif (cont_reg + 1) = 30 then
                    color_act_8bits <= fila30;
                    color_act_4bits <= fila30(3 downto 0);
                elsif (cont_reg + 1) = 31 then
                    color_act_8bits <= fila31;
                    color_act_4bits <= fila31(31 downto 28);
                end if;
                    estado <= s1;
            end if;
        else
-- Si no hemos acabado todos los leds, pasamos al siguiente
            cont_led <= cont_led + 1;
            cont_bits <= 23;
-- antes de elegir la parte correspondiente del registro hay que elegir de que registro cargamos
            if cont_reg = 0 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
                    color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
```

```
                end if;

            elsif cont_reg = 2 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
                    color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
                end if;

            elsif cont_reg = 4 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
                    color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
end if;

            elsif cont_reg = 6 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
                    color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
                end if;

            elsif cont_reg = 8 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
        color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
                end if;

            elsif cont_reg = 10 then
                if (cont_led + 1) = 1 then
                    color_act_4bits <= color_act_8bits(7 downto 4);
                elsif (cont_led + 1) = 2 then
                    color_act_4bits <= color_act_8bits(11 downto 8);
                elsif (cont_led + 1) = 3 then
                    color_act_4bits <= color_act_8bits(15 downto 12);
                elsif (cont_led + 1) = 4 then
                    color_act_4bits <= color_act_8bits(19 downto 16);
                elsif (cont_led + 1) = 5 then
                    color_act_4bits <= color_act_8bits(23 downto 20);
                elsif (cont_led + 1) = 6 then
                    color_act_4bits <= color_act_8bits(27 downto 24);
                elsif (cont_led + 1) = 7 then
                    color_act_4bits <= color_act_8bits(31 downto 28);
                end if;

            elsif cont_reg = 12 then
                if (cont_led + 1) = 1 then
```

```
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 14 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 16 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 18 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 20 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 22 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
```

```
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 24 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 26 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 28 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    elsif cont_reg = 30 then
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(7 downto 4);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 6 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 7 then
            color_act_4bits <= color_act_8bits(31 downto 28);
        end if;

    else
        if (cont_led + 1) = 1 then
            color_act_4bits <= color_act_8bits(27 downto 24);
        elsif (cont_led + 1) = 2 then
            color_act_4bits <= color_act_8bits(23 downto 20);
        elsif (cont_led + 1) = 3 then
            color_act_4bits <= color_act_8bits(19 downto 16);
        elsif (cont_led + 1) = 4 then
            color_act_4bits <= color_act_8bits(15 downto 12);
        elsif (cont_led + 1) = 5 then
            color_act_4bits <= color_act_8bits(11 downto 8);
        elsif (cont_led + 1) = 6 then
```

```
                                                    color_act_4bits <= color_act_8bits(7 downto 4);
                                                elsif (cont_led + 1) = 7 then
                                                    color_act_4bits <= color_act_8bits(3 downto 0);
                                                end if;

                                            end if;
-- Si hubiese que poner un color diferente en el siguiente led, pasar por aqui
                                                estado <= s1;
                                            end if;
                                        else
--color_act <= SHL(color_act,'0');
                                            cont_bits <= cont_bits - 1; -- Si no hemos transmitido todos los
bits, transmitimos el siguiente de los que nos faltan
                                            estado <= s1;
                                        end if;
                                    end if;

                when sf => dato_rgb <= '0';
                                    if init <= '0' then
                                        estado <= s0;
                                    end if;

                end case;
        end if;
end process;

end Behavioral;
```

A.2 CONTROL_MAIN.vhd:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:03:14 12/10/2022
-- Design Name:
-- Module Name:    control_led_rgb_32x8 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CONTROL_MAIN is
    Port ( clk : in  STD_LOGIC;
                           reset : in STD_LOGIC;
           init_config : in  STD_LOGIC; -- Pulsador
-- Los 3 bits de configuracion de leds
           switches : in  STD_LOGIC_VECTOR (1 downto 0);
-- La salida a los leds se usará en la instanciacion 1
                           dato_rgb: out STD_LOGIC
                           );


end CONTROL_MAIN;

architecture Behavioral of CONTROL_MAIN is

type statetype is (s0,s1,s2,s3,s4,s5);
signal estado:statetype := s0;
signal permiso_leds: std_logic:='0'; -- Señal que dará permiso al programa de leds para comenzar su carga

signal fila0 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila1 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila2 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
```

```
signal fila3 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila4 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila5 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila6 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila7 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila8 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila9 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila10 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila11 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila12 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila13 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila14 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila15 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila16 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila17 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila18 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila19 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila20 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila21 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila22 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila23 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila24 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila25 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila26 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila27 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila28 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila29 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila30 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');
signal fila31 : STD_LOGIC_VECTOR (31 downto 0):=(others => '0');

-- DECLARACION DE COMPONENTE PARA CARGAR COLOR DE LEDS DESDE REGISTRO
COMPONENT LEDS_RGB_32X8

PORT ( clk : in  STD_LOGIC;
           init : in  STD_LOGIC;
                      reset: in STD_LOGIC;
           fila0 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila1 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila2 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila3 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila4 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila5 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila6 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila7 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila8 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila9 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila10 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila11 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila12 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila13 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila14 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila15 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila16 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila17 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila18 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila19 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila20 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila21 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila22 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila23 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila24 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila25 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila26 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila27 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila28 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila29 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila30 : in  STD_LOGIC_VECTOR (31 downto 0);
           fila31 : in  STD_LOGIC_VECTOR (31 downto 0);
                      dato_rgb: out STD_LOGIC);

END COMPONENT;


begin

-- En este programa hay que hacer un proceso que se inicie con el init_config y con el reset a 0
-- El reloj clk contabilizará los ciclos. Estaremos en el estado 0 antes de que se pulse init_config (1)
-- Una vez se haya pulsado init_config, pasaremos al estado que nos marquen los switches.

INSTANCIACION_LEDS : LEDS_RGB_32X8 PORT MAP(

    clk => clk,
    init => permiso_leds,
        reset => reset,
    fila0 => fila0,
    fila1 => fila1,
    fila2 => fila2,
    fila3 => fila3,
    fila4 => fila4,
```

```
    fila5 => fila5,
    fila6 => fila6,
    fila7 => fila7,
    fila8 => fila8,
    fila9 => fila9,
    fila10 => fila10,
    fila11 => fila11,
    fila12 => fila12,
    fila13 => fila13,
    fila14 => fila14,
    fila15 => fila15,
    fila16 => fila16,
    fila17 => fila17,
    fila18 => fila18,
    fila19 => fila19,
    fila20 => fila20,
    fila21 => fila21,
    fila22 => fila22,
    fila23 => fila23,
    fila24 => fila24,
    fila25 => fila25,
    fila26 => fila26,
    fila27 => fila27,
    fila28 => fila28,
    fila29 => fila29,
    fila30 => fila30,
    fila31 => fila31,
         dato_rgb => dato_rgb
);

-- PARTE SECUENCIAL

CARGA_DE_REGISTROS: process(clk, reset)--,confirm)

begin

    if reset = '1' then
        permiso_leds <= '0';
                estado <= s0;
       elsif rising_edge(clk) then

                case estado is

                        when s0 => -- Estado de espera hasta que se inicie el init_config
                                if init_config = '1' then

                                        permiso_leds <= '0';
                                        -- Tendremos 2 bits de
configuracion. Se harán 4 configuraciones.
                                        -- Comprobamos cual es nuestra
configuracion
                                        -- Inicializamos los registros
                                        if switches = "00" then
                                            estado <= s1;
                                        elsif switches = "01" then
                                            estado <= s2;
                                        elsif switches = "10" then
                                            estado <= s3;
                                        elsif switches = "11" then
                                            estado <= s4;
                                        end if;

                                end if;

                        when s1 => -- Configuracion 00 APAGADO

                                fila0 <= (others => '0');
                                fila1 <= (others => '0');
                                fila2 <= (others => '0');
                                fila3 <= (others => '0');
                                fila4 <= (others => '0');
                                fila5 <= (others => '0');
                                fila6 <= (others => '0');
                                fila7 <= (others => '0');
                                fila8 <= (others => '0');
                                fila9 <= (others => '0');
                                fila10 <= (others => '0');
                                fila11 <= (others => '0');
                                fila12 <= (others => '0');
                                fila13 <= (others => '0');
                                fila14 <= (others => '0');
                                fila15 <= (others => '0');
                                fila16 <= (others => '0');
                                fila17 <= (others => '0');
                                fila18 <= (others => '0');
                                fila19 <= (others => '0');
                                fila20 <= (others => '0');
                                fila21 <= (others => '0');
                                fila22 <= (others => '0');
                                fila23 <= (others => '0');
                                fila24 <= (others => '0');
```

```
                                    fila25 <= (others => '0');
                                    fila26 <= (others => '0');
                                    fila27 <= (others => '0');
                                    fila28 <= (others => '0');
                                    fila29 <= (others => '0');
                                    fila30 <= (others => '0');
                                    fila31 <= (others => '0');

                                    estado <= s5;

            when s2 => -- Configuracion 01 -- 32 filas de 8 leds

                                    fila0 <= X"11111111";
                                    fila1 <= (others => '0');
                                    fila2 <= X"11111111";
                                    fila3 <= (others => '0');
                                    fila4 <= X"11111111";
                                    fila5 <= (others => '0');
                                    fila6 <= X"11111111";
                                    fila7 <= (others => '0');
                                    fila8 <= X"11111111";
                                    fila9 <= (others => '0');
                                    fila10 <= X"11111111";
                                    fila11 <= (others => '0');
                                    fila12 <= X"11111111";
                                    fila13 <= (others => '0');
                                    fila14 <= X"11111111";
                                    fila15 <= (others => '0');
                                    fila16 <= X"11111111";
                                    fila17 <= (others => '0');
                                    fila18 <= X"11111111";
                                    fila19 <= (others => '0');
                                    fila20 <= X"11111111";
                                    fila21 <= (others => '0');
                                    fila22 <= X"11111111";
                                    fila23 <= (others => '0');
                                    fila24 <= X"11111111";
                                    fila25 <= (others => '0');
                                    fila26 <= X"11111111";
                                    fila27 <= (others => '0');
                                    fila28 <= X"11111111";
                                    fila29 <= (others => '0');
                                    fila30 <= X"11111111";
                                    fila31 <= (others => '0');

                                    estado <= s5;

            when s3 => -- Configuracion 10 -- 8 filas de 32 leds

                                    fila0 <= X"11111111";
                                    fila1 <= X"11111111";
                                    fila2 <= X"11111111";
                                    fila3 <= X"11111111";
                                    fila4 <= (others => '0');
                                    fila5 <= (others => '0');
                                    fila6 <= (others => '0');
                                    fila7 <= (others => '0');
                                    fila8 <= X"11111111";
                                    fila9 <= X"11111111";
                                    fila10 <= X"11111111";
                                    fila11 <= X"11111111";
                                    fila12 <= (others => '0');
                                    fila13 <= (others => '0');
                                    fila14 <= (others => '0');
                                    fila15 <= (others => '0');
                                    fila16 <= X"11111111";
                                    fila17 <= X"11111111";
                                    fila18 <= X"11111111";
                                    fila19 <= X"11111111";
                                    fila20 <= (others => '0');
                                    fila21 <= (others => '0');
                                    fila22 <= (others => '0');
                                    fila23 <= (others => '0');
                                    fila24 <= X"11111111";
                                    fila25 <= X"11111111";
                                    fila26 <= X"11111111";
                                    fila27 <= X"11111111";
                                    fila28 <= (others => '0');
                                    fila29 <= (others => '0');
                                    fila30 <= (others => '0');
                                    fila31 <= (others => '0');

                                    estado <= s5;

            when s4 => -- Configuracion 11 -- COMPROBACION DEL FINAL DE CADA FILA (MAPEO)

                                    fila0 <= X"01010101";
                                    fila1 <= X"01010101";
                                    fila2 <= X"01010101";
```

```
                                                    fila3 <= X"01010101";
                                                    fila4 <= X"01010101";
                                                    fila5 <= X"01010101";
                                                    fila6 <= X"01010101";
                                                    fila7 <= X"01010101";
                                                    fila8 <= X"01010101";
                                                    fila9 <= X"01010101";
                                                    fila10 <= X"01010101";
                                                    fila11 <= X"01010101";
                                                    fila12 <= X"01010101";
                                                    fila13 <= X"01010101";
                                                    fila14 <= X"01010101";
                                                    fila15 <= X"01010101";
                                                    fila16 <= X"01010101";
                                                    fila17 <= X"01010101";
                                                    fila18 <= X"01010101";
                                                    fila19 <= X"01010101";
                                                    fila20 <= X"01010101";
                                                    fila21 <= X"01010101";
                                                    fila22 <= X"01010101";
                                                    fila23 <= X"01010101";
                                                    fila24 <= X"01010101";
                                                    fila25 <= X"01010101";
                                                    fila26 <= X"01010101";
                                                    fila27 <= X"01010101";
                                                    fila28 <= X"01010101";
                                                    fila29 <= X"01010101";
                                                    fila30 <= X"01010101";
                                                    fila31 <= X"01010101";

                                                    estado <= s5;

                                when s5 => -- Estado final

                                                    permiso_leds <= '1';
                                                    if init_config = '0' then
                                                          estado <= s0;
                                                    end if;

                        end case;
                end if;

end process;

end Behavioral;
```

A.3 Fichero principal de la aplicación para control de LEDs en PYNQ: freertos_helloworld.c

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
//#include "xgpio.h"
#include "sleep.h"
#include "xil_io.h"
#include "driver_ip.h"

#define R 0x04 //0100
#define G 0x02 //0010
#define B 0x03 //0011

// Colores

#define apagar 0b0000 //0x0
#define rojo 0b0001 // 0x1
#define verde 0b0010 // 0x2
#define azul 0b0011 // 0x3
#define magenta 0b0100 // o 0x4
#define amarillo 0b0101 // o 0x5
#define cyan 0b0110 // o 0x6
#define gris 0b0100 // o 0x4
```

```
#define blanco 0b1000 // o 0x8

// Mascaras para el registro de control
#define reset_mask 0x0001
#define init_mask 0x0002

// Manejadores (handles)

XGpio gpio;
XGpio gpio_ip;

void driverInit(){//inicializar Bloque IP
	int status;
	status = XGpio_Initialize(&gpio_ip, XPAR_DEMO5_0_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Gpio Initialization failed\n\r");
	}
	else{
		print("Info: Gpio Initialization successful\n\r");
	}
}

void driverInit2(){//inicializar bloque GPIO_0 (botones y switch0)
	int status;
	status = XGpio_Initialize(&gpio, XPAR_AXI_GPIO_0_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Gpio Initialization failed\n\r");
	}
	else{
		print("Info: Gpio Initialization successful\n\r");
	}
}


void configGpio() {//configurar mapeo
	XGpio_SetDataDirection(&gpio, 1, 0); //Configurar botones como input
	XGpio_SetDataDirection(&gpio, 2, 0); //Configurar switch0 como input

	XGpio_DiscreteSet(&gpio, 1, 0);//Dar valor 0 a los botones inicialmente

}

void runProject(){

	int button;
	int switches;

	while(1){

		switches = XGpio_DiscreteRead(&gpio, 2); // Leemos el valor del switch
		switch (switches)
		{
		case 0:
			//Leemos los botones
			button = XGpio_DiscreteRead(&gpio, 1);
			switch (button)
			{
			case 1://0001 Config 1: Cargar todos los leds del mismo color
```

```
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Cargando config 1. (Init desactivado)
\n\r");
                        //Cargamos los registros a 0
                        Config_leds1(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Config 1 cargada. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                case 2://0010 Config 2: Cargar una fila si y la siguiente no

                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Cargando config 2. (Init desactivado)
\n\r");
                        //Cargamos los registros a 0
                        Config_leds2(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Config 2 cargada. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                case 4://0100 Leer registros

                        Leer_registros(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        break;

                case 8://1000 Apagar leds

                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Notificamos
                        xil_printf("\t Apagando leds... \n\r");
                        //Cargamos los registros a 0
                        Apagar_leds(XPAR_DEMO5_0_S00_AXI_BASEADDR,33);
                        //Notificamos avance
                        xil_printf("\t Registros a 0. \n\r");
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Leds apagados. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                default:
                        break;
                }

            case 1:
```

```
                    //Leemos los botones
                    button = XGpio_DiscreteRead(&gpio, 1);
                    switch (button)
                    {
                    case 1://0001 Config 3: Cargar led si y led no

                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          xil_printf("\t Cargando config 3. (Init desactivado)
\n\r");
                          //Cargamos los registros a 0
                          Config_leds3(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Config 3 cargada. (Init activado) \n\r");
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(2);
                          break;

                    case 2://0010 Config 4:

                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          xil_printf("\t Cargando config 4. (Init desactivado)
\n\r");
                          //Cargamos los registros a 0
                          Config_leds4(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Config 4 cargada. (Init activado) \n\r");
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(2);
                          break;

                    case 4://0100 Leer registros

                          Leer_registros(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          break;

                    case 8://1000 Apagar leds

                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Notificamos
                          xil_printf("\t Apagando leds... \n\r");
                          //Cargamos los registros a 0
                          Apagar_leds(XPAR_DEMO5_0_S00_AXI_BASEADDR,33);
                          //Notificamos avance
                          xil_printf("\t Registros a 0. \n\r");
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Leds apagados. (Init activado) \n\r");
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(2);
```

```
                        break;

                default:
                        break;
                }

        case 2:
                //Leemos los botones
                button = XGpio_DiscreteRead(&gpio, 1);
                switch (button)
                {
                case 1://0001 Config 5: Nombre con los leds

                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Cargando config 5. (Init desactivado)
\n\r");
                        //Cargamos los registros a 0
                        Config_leds5(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Config 5 cargada. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                case 2://0010 Config 6: Desplazar el contenido de cada fila a la
siguiente. Esto será un bucle. Se podrá salir de el con los switches
                        xil_printf("\t Entrando en config6. PONER SW EN OTRA
POSICION PARA SALIR \n\r");
                        while(switches == 2)
                        {
                                switches = XGpio_DiscreteRead(&gpio, 2);
                                //Desactivamos Init siempre al escribir datos en un
registro.
                                Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                                //Config6
                                Config_leds6(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                                xil_printf("\t ITERACION C6 \n\r");
                                //Activamos el init para que se comiencen a cargar
                                Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                                //Damos tiempo a que se carguen los leds, y se
llegue al ultimo estado del diseño (sf)
                                sleep(1);
                        }
                    xil_printf("\t Saliendo de config6. \n\r");
                        break;
                case 4://0100 Leer registros

                        Leer_registros(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        break;

                case 8://1000 Apagar leds

                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Notificamos
                        xil_printf("\t Apagando leds... \n\r");
```

```
                        //Cargamos los registros a 0
                        Apagar_leds(XPAR_DEMO5_0_S00_AXI_BASEADDR,33);
                        //Notificamos avance
                        xil_printf("\t Registros a 0. \n\r");
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Leds apagados. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                default:
                        break;
                }

            case 3:
                //Leemos los botones
                button = XGpio_DiscreteRead(&gpio, 1);
                switch (button)
                {
                case 1://0001 Config 7: Nombre con los leds
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        xil_printf("\t Cargando config 7. (Init desactivado)
\n\r");
                        //Cargamos los registros a 0
                        Config_leds7(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Config 7 cargada. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;
                case 2://0010 Config 8: Desplazar el contenido de cada fila a la
anterior. Esto será un bucle. Se podrá salir de el con los switches
                        xil_printf("\t Entrando en config8. PONER SW EN OTRA
POSICION PARA SALIR \n\r");
                        while(switches == 3)
                        {
                            switches = XGpio_DiscreteRead(&gpio, 2);
                            //Desactivamos Init siempre al escribir datos en un
registro.
                            Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                            //Config6
                            Config_leds8(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                            xil_printf("\t ITERACION C8 \n\r");
                            //Activamos el init para que se comiencen a cargar
                            Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                            //Damos tiempo a que se carguen los leds, y se llegue
al ultimo estado del diseño (sf)
                            sleep(1);
                        }
                        xil_printf("\t Saliendo de config8. \n\r");
                        break;
                case 4://0100 Leer registros

                        Leer_registros(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
```

```
                        break;

                case 8://1000 Apagar leds

                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Notificamos
                        xil_printf("\t Apagando leds... \n\r");
                        //Cargamos los registros a 0
                        Apagar_leds(XPAR_DEMO5_0_S00_AXI_BASEADDR,33);
                        //Notificamos avance
                        xil_printf("\t Registros a 0. \n\r");
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                    xil_printf("\t Leds apagados. (Init activado) \n\r");
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(2);
                        break;

                default:
                        break;
                }

        default:
                break;
        }
    }

}

int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    driverInit2();
    driverInit();
    configGpio();
    runProject();
    return 0;
}
```

A.3.1 Libreria creada (v1): driver_ip.c

```
/**************************** Include Files *********************************/
#include "xil_io.h"
#include "driver_ip.h"
#include "xil_printf.h"
#include "xgpio_i.h"

XDriver_Config XDriver_ConfigTable[XPAR_DEMO5_NUM_INSTANCES] =
{
        {
                XPAR_AXI_GPIO_0_DEVICE_ID,
                XPAR_AXI_GPIO_0_BASEADDR,
        }
```

```
};

/************************ Function Prototypes ****************************/

/****************** Cargar todos los registros a 0 ******************/


void Apagar_leds(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int offset = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	for(i = 0; i <= num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/

/************Leer registros y mostrarlos por el puerto serie*************/

void Leer_registros(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int x = 0;
	unsigned long int buffer;
	int offset_read = 0x00;
	num_reg = num_reg - 1;    // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.


//¿Puedo cambiar la i por la x?

	for(i = 0; i <= num_reg; i++)
	{
	buffer = Xil_In32((Base_ip_address + offset_read));
	offset_read = offset_read + 0x4;
    xil_printf("\t Registro %d: %x\n\r", x, buffer);
    x++;
	}
}

/*****************************************************************************/
// num_led: Insertar entero entre 1 y 256, para nuestro diseño.
/*****************************************************************************/
/*************************Cargar un solo led*************************/

void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color)

{
	int puntero_reg = 0;
	int puntero_led = 0;
	int total_leds = 0;
	unsigned long int copia_color = 0;
	unsigned long int color_led = 0;
```

```
	unsigned long int buffer;
	int i;
	int puntero = 0;
	//1- Calcular el numero total de leds. Cada registro tiene 8 leds.
	//En el parámetro num_reg hay que descontar un registro, ya que el ultimo registro es el de control
	//Y no corresponde a los leds.

	//Este numero (num_reg) lo usamos para comprobar que los parámetros que nos han dado son correctos.
	num_reg = num_reg - 1;
	total_leds = num_reg*8; //Multiplicamos el numero de registros por el numero de leds de cada registro; 8.
	//Comprobamos que el parámetro num_led esta dentro de rango
	if(num_led <= total_leds && num_led > 0)
	{
		//Parámetro correcto. Procedemos a cargar el led correspondiente

		//Creamos el puntero que apuntará al registro que contiene al led que queremos cargar
		//Las direcciones de memoria van de 4 en 4 para cada registro, y cada registro consta de 8 leds.
		//Basta con dividir el numero
		puntero_reg = num_led/8; // Ya tenemos el número del registro al que corresponde el led.
		puntero_led = num_led%8; //El resto nos indica en que posicion queda ese led dentro del registro correspondiente.

		//Como los leds estan conectados en zig zag, probablemente haya que alternar el orden del led, de izquierda a derecha
		//dependiendo de si el numero del registro es par o impar.

		// PENDIENTE


		//Una vez tenemos los dos punteros, solo faltaría cargar el color en la direccion asignada
		//Actualizamos el puntero con la dirección en hexadecimal
		if (puntero_led == 0)
		{
			puntero = Base_ip_address + (puntero_reg-1)*0x4;
		}
		else
		{
			puntero = Base_ip_address + (puntero_reg)*0x4;
		}
		//Una vez conocemos la direccion del registro, solo falta ajustar el color del led en una máscara adecuada
		//Con esa máscara podremos cargar el led que nos plazca sin alterar el resto
		//En nuestro caso se aplican dos mascaras, una para no altelar el contenido de los registros y borrar el del led
		//Otra para crear una variable que contenga el color (4 bits) en la posicion correcta dentro del registro

		copia_color = copia_color | color;
		for(i=0; i<7; i++)
		{
			copia_color = copia_color<<4 | color; //Creamos una variable que contenga el color repetido para todos los leds
		}
```

```
		xil_printf("\t Revisando puntero... \n\r");

		//Antes que nada leemos el registro en el que vamos a cargar los datos y lo guardamos en un buffer
		buffer = Xil_In32(puntero);

		//Comprobamos cual es el led que hay que cargar dentro del registro
		switch (puntero_led)
		{
			case 0:
				//Borramos el led que vamos a cargar en el buffer y dejamos el resto de leds con su valor.
				buffer = buffer & notled_8;
				//Aplicamos máscara para tener el color solo en la posición del led deseado
				color_led = copia_color & led_8;
				//Escribimos el dato del color dentro de los que ya habia cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
				break;
			case 1:
				//Borramos el led que vamos a cargar en el buffer y dejamos el resto de leds con su valor.
				buffer = buffer & notled_1;
				//Aplicamos máscara para tener el color solo en la posición del led deseado
				color_led = copia_color & led_1;
				//Escribimos el dato del color dentro de los que ya habia cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
				break;
			case 2:
				//Borramos el led que vamos a cargar en el buffer y dejamos el resto de leds con su valor.
				buffer = buffer & notled_2;
				//Aplicamos máscara para tener el color solo en la posición del led deseado
				color_led = copia_color & led_2;
				//Escribimos el dato del color dentro de los que ya habia cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
				break;
			case 3:
				//Borramos el led que vamos a cargar en el buffer y dejamos el resto de leds con su valor.
				buffer = buffer & notled_3;
				//Aplicamos máscara para tener el color solo en la posición del led deseado
				color_led = copia_color & led_3;
				//Escribimos el dato del color dentro de los que ya habia cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
```

```
                        break;
                case 4:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_4;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_4;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 5:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_5;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_5;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 6:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_6;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_6;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 7:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_7;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_7;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                default:
                        break;
        }
    }
    else
    {}
}
```

```
/****************************************************************/
/*******************************INIT ON*******************************/

void Activar_init(long unsigned int Base_ip_address, int num_reg)
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del
init
	//Y hacer una operacion OR a esa máscara
	buffer = buffer | init_mask_on;
	Xil_Out32(puntero, buffer);
}

/****************************************************************/
/*******************************RESET ON*******************************/

void Activar_reset(long unsigned int Base_ip_address, int num_reg) // El reset es
activo en baja
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga un 0 en la posicion del
reset y el resto a 1
	//Y hacer una operacion AND a esa máscara
	buffer = buffer | reset_mask_on;
	Xil_Out32(puntero, buffer);
}

/****************************************************************/
/*******************************INIT OFF*******************************/

void Desactivar_init(long unsigned int Base_ip_address, int num_reg)
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 0 en la posicion del
init
	//Y hacer una operacion AND a esa máscara
	buffer = buffer & init_mask_off;
	Xil_Out32(puntero, buffer);
}

/****************************************************************/
/*******************************RESET OFF*******************************/
```

```
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg) //El reset es activo en baja
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del init
	//Y hacer una operacion OR a esa máscara
	buffer = buffer & reset_mask_off;
	Xil_Out32(puntero, buffer);
}

/*****************************************************************************/
/****************Cargar todos los leds del mismo color*****************/
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}

	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/
/****************Cargar leds fila si y fila no*************************/
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = (num_reg - 1)/2; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}
```

```
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/
/****************** Cargar led si y led no ***************************/
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<3;j++)
	{
		color_reg = color_reg <<8;
		color_reg = color_reg | color;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/
/*************** Cargar led si y led no zig zag ***********************/
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j, paridad;
	int offset = 0;
	unsigned long int color_reg_impar = 0;
	unsigned long int color_reg_par = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg_impar = color_reg_impar + color;
	color_reg_par = color_reg_par + color;
	color_reg_par = color_reg_par <<4;
	for(j=0;j<3;j++)
	{
		color_reg_impar = color_reg_impar <<8;
		color_reg_impar = color_reg_impar | color;
	}
	for(j=0;j<3;j++)
```

```
	{
		color_reg_par = color_reg_par <<4;
		color_reg_par = color_reg_par | color;
		color_reg_par = color_reg_par <<4;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
		paridad = i%2;
		if(paridad == 0)// el numero de registro es par
		{
			Xil_Out32((Base_ip_address + offset), color_reg_par);
			offset = offset + 0x4;
		}
		else// el numero de registro es impar
		{
			Xil_Out32((Base_ip_address + offset), color_reg_impar);
			offset = offset + 0x4;
		}
	}
}

/*****************************************************************************/
/**************** Iluminar nombre con los leds *************************/
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 3 espacios
	for(i=0;i<3;i++)
	{
		Xil_Out32((Base_ip_address + offset), 0x00000000);
		offset = offset + 0x4;
	}
	//Letra C
	color_reg = color_mask & C1;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
```

```
//Letra A
color_reg = color_mask & A1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & A2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & A1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra R
color_reg = color_mask & R1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra L
color_reg = color_mask & L1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra O
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra S
color_reg = color_mask & S1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S2;
```

```
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S3;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Ultimos 3 espacios
	for(i=0;i<3;i++)
	{
		Xil_Out32((Base_ip_address + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

/****************************************************************************/
/************* Mover contenido de registros a la izquierda ***************/
void Config_leds6(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int offset = 0;
	unsigned long int buffer = 0;
	unsigned long int buffer_posterior = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-2" ya que el ultimo
registro se usará como registro de control
	//y no tendra INFO de color como el resto.
	//Y el anterior a ese se cargará fuera del bucle.
	//Leemos el primer dato
	buffer = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR);
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
		if(i == 31)
		{
			Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR, buffer); //Escribimos en
la siguiente dirección de registro
		}
		else
		{
			offset = offset + 0x4;
			buffer_posterior = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR +
offset); //Leemos el siguiente dato para que no se pierda
			Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR + offset, buffer);
//Escribimos en la siguiente dirección de registro
			buffer = buffer_posterior;
		}
	}
}

/****************************************************************************/
/*************** Iluminar nombre con los leds ************************/
void Config_leds7(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
```

```
color_mask = color_mask | color;
for(i=0;i<7;i++)
{
	color_mask = color_mask<<4;
	color_mask = color_mask | color;
}
//Primeros 3 espacios
for(i=0;i<3;i++)
{
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
}
//Letra S
color_reg = color_mask & S4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra O
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra L
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra R
color_reg = color_mask & R4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R3;
Xil_Out32((Base_ip_address + offset), color_reg);
```

```
	offset = offset + 0x4;
	color_reg = color_mask & R2;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & R1;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra A
	color_reg = color_mask & A1_3;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A1_3;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra C
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C1;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Ultimos 3 espacios
	for(i=0;i<3;i++)
	{
		Xil_Out32((Base_ip_address + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

/*****************************************************************************/
/************* Mover contenido de registros a la derecha ***************/
void Config_leds8(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int offset_inicial, offset = 0;
	unsigned long int buffer = 0;
	unsigned long int buffer_posterior = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-2" ya que el ultimo
registro se usará como registro de control
	offset_inicial = (num_reg-1)*0x4;
	offset=offset_inicial;
	//y no tendra INFO de color como el resto.
	//Y el anterior a ese se cargará fuera del bucle.
	//Leemos el primer dato
	buffer = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR + offset);
```

```
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
		if(i == 31)
		{
			Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR + offset_inicial,
buffer); //Escribimos en la siguiente dirección de registro
		}
		else
		{
			offset = offset - 0x4;
			buffer_posterior = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR +
offset); //Leemos el siguiente dato para que no se pierda
			Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR + offset, buffer);
//Escribimos en la siguiente dirección de registro
			buffer = buffer_posterior;
		}
	}
}

/*****************************************************************************/
/************* Imprimir caracteres en el centro de los leds ***************/
/*Los caracteres se encenderán en un rectangulo centrado en la matriz, de 6x4 leds*/
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra A
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
```

```
	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra B
	color_reg = color_mask & B1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra C
	color_reg = color_mask & C1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra D
```

```
	color_reg = color_mask & D1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & D3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 1
	color_reg = color_mask & uno_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & uno_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
```

```
	}
}

void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 2
	color_reg = color_mask & dos_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & dos_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
```

```
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 3
		color_reg = color_mask & tres_1;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & tres_2;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		color_reg = color_mask & tres_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & tres_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;

		//Ultimos 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}
}

void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
		for(i=0;i<7;i++)
		{
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 4
		color_reg = color_mask & cuatro_1;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cuatro_2;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		color_reg = color_mask & cuatro_3;
```

```
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 5
	color_reg = color_mask & cinco_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color)
{
```

```
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 6
	color_reg = color_mask & seis_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
```

```
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 7
		color_reg = color_mask & siete_1;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & siete_2;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		color_reg = color_mask & siete_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & siete_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;

		//Ultimos 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}
}

void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
		for(i=0;i<7;i++)
		{
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 8
		color_reg = color_mask & ocho_1_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & ocho_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		color_reg = color_mask & ocho_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & ocho_1_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
```

```
	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 9
	color_reg = color_mask & nueve_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 0
	color_reg = color_mask & cero_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cero_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & cero_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cero_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_asterisco(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Caracter asterisco
```

```
	color_reg = color_mask & asterisco_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & asterisco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & asterisco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & asterisco_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_almohadilla(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 13 espacios
	for(i=0;i<13;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Caracter almohadilla
	color_reg = color_mask & almohadilla_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & almohadilla_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	color_reg = color_mask & almohadilla_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & almohadilla_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & almohadilla_5;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & almohadilla_6;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
```

```
	color_reg = color_mask & almohadilla_7;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 12 espacios
	for(i=0;i<12;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}


/************************************************************************/
/********************** Inicializacion del driver ***********************/
int Driver_init(XDriver * InstancePtr, u16 DeviceId)
{
	XDriver_Config *ConfigPtr;

	/*
	 * Assert arguments
	 */
	Xil_AssertNonvoid(InstancePtr != NULL);

	/*
	 * Lookup configuration data in the device configuration table.
	 * Use this configuration info down below when initializing this
	 * driver.
	 */
	ConfigPtr = XDriver_LookupConfig(DeviceId);
	if (ConfigPtr == (XDriver_Config *) NULL) {
		InstancePtr->IsReady = 0;
		return (XST_DEVICE_NOT_FOUND);
	}

	return XDriver_CfgInitialize(InstancePtr, ConfigPtr,
				ConfigPtr->BaseAddress);
}

XDriver_Config *XDriver_LookupConfig(u16 DeviceId)
{
	XDriver_Config *CfgPtr = NULL;

	int Index;

	for (Index = 0; Index < XPAR_DEMO5_NUM_INSTANCES; Index++) {
		if (XDriver_ConfigTable[Index].DeviceId == DeviceId) {
			CfgPtr = &XDriver_ConfigTable[Index];
			break;
		}
	}

	return CfgPtr;
}

int XDriver_CfgInitialize(XDriver * InstancePtr, XDriver_Config * Config,
			UINTPTR EffectiveAddr)
{
	/* Assert arguments */
	Xil_AssertNonvoid(InstancePtr != NULL);
```

```
	/* Set some default values. */
	InstancePtr->BaseAddress = EffectiveAddr;
	/*
	 * Indicate the instance is now ready to use, initialized without error
	 */
	InstancePtr->IsReady = XIL_COMPONENT_IS_READY;
	return (XST_SUCCESS);
}
```

A.3.2 Librería creada (v1): driver_ip.h

```
//includes
#include "xil_io.h"
#include "xgpio.h"

/************************** Type Definitions ***************************/

/**
 * This typedef contains configuration information for the device.
 */
typedef struct {
	u16 DeviceId;			/**< Unique ID  of device */
	UINTPTR BaseAddress;		/**< Device base address */
} XDriver_Config;

/**
 * The XGpio driver instance data. The user is required to allocate a
 * variable of this type for every GPIO device in the system. A pointer
 * to a variable of this type is then passed to the driver API functions.
 */
typedef struct {
	UINTPTR BaseAddress;		/**< Device base address */
	u32 IsReady;		/**< Device is initialized and ready */
} XDriver;
/************************************************************************/

//Functions

//Initialize functions

int XDriver_init(XDriver *InstancePtr, u16 DeviceId);
XDriver_Config *XDriver_LookupConfig(u16 DeviceId);
int XDriver_CfgInitialize(XDriver *InstancePtr, XDriver_Config * Config, UINTPTR
EffectiveAddr);

// Estas funciones solo van a colocar los valores de los leds en las filas
correspondientes.
void Apagar_leds(long unsigned int Base_ip_address, int num_reg);
void Leer_registros(long unsigned int Base_ip_address, int num_reg);
void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color);
void Activar_init(long unsigned int Base_ip_address, int num_reg);
void Activar_reset(long unsigned int Base_ip_address, int num_reg);
void Desactivar_init(long unsigned int Base_ip_address, int num_reg);
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg);
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color);
```

```
void Config_leds6(long unsigned int Base_ip_address, int num_reg);
void Config_leds7(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds8(long unsigned int Base_ip_address, int num_reg);
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_asterisco(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_almohadilla(XDriver *Puntero_inst, int num_reg, short int color);

//int Config1_leds(uint_32 BASE_IP_ADDRESS);
//int Config2_leds(uint_32 BASE_IP_ADDRESS);
//int Config3_leds(uint_32 BASE_IP_ADDRESS);


//Mascaras para los leds:

//Estas máscaras se han definido con MSB como el led 8 (izquierda) y LSB el led 1
(derecha)

#define led_8 0xF0000000 //Led 8
#define led_7 0x0F000000 //Led 7
#define led_6 0x00F00000 //Led 6
#define led_5 0x000F0000 //Led 5
#define led_4 0x0000F000 //Led 4
#define led_3 0x00000F00 //Led 3
#define led_2 0x000000F0 //Led 2
#define led_1 0x0000000F //Led 1

//Mascaras inversas
#define notled_8 0x0FFFFFFF //Led 8
#define notled_7 0xF0FFFFFF //Led 7
#define notled_6 0xFF0FFFFF //Led 6
#define notled_5 0xFFF0FFFF //Led 5
#define notled_4 0xFFFF0FFF //Led 4
#define notled_3 0xFFFFF0FF //Led 3
#define notled_2 0xFFFFFF0F //Led 2
#define notled_1 0xFFFFFFF0 //Led 1

//Mascaras registro de control
#define init_mask_on      0x00000002
#define init_mask_off   0xFFFFFFFD
#define reset_mask_off    0xFFFFFFFE
#define reset_mask_on  0x00000001

//Mascaras para las letras
#define A1_3 0x0FFFFFF0
#define A2 0x0F00F000
#define B1 0x0FFFFFF0
#define B2_3 0x0F0FF0F0
```

```
#define B4 0x0FF00FF0
#define C1 0x0FFFFFF0
#define C2_3_4 0x0F0000F0
#define D1 0x0FFFFFF0
#define D2 0x0F0000F0
#define D3 0x00F00F00
#define D4 0x000FF000
#define R1 0x0FFFFFF0
#define R2 0x0F0FF000
#define R3 0x0F0F0F00
#define R4 0x0FFF00F0
#define L1 0x0FFFFFF0
#define L2_3 0x000000F0
#define O1_3 0x0FFFFFF0
#define O2 0x0F0000F0
#define S1 0x0FFF00F0
#define S2 0x0F0F00F0
#define S3 0x0F00F0F0
#define S4 0x0F00FFF0
#define uno_1 0x000F00F0
#define uno_2 0x00F000F0
#define uno_3 0x0FFFFFF0
#define uno_4 0x000000F0
#define dos_1 0x00F00FF0
#define dos_2 0x0F000FF0
#define dos_3 0x0F00F0F0
#define dos_4 0x00FF00F0
#define tres_1 0x00F00F00
#define tres_2 0x0F0000F0
#define tres_3 0x0F0F00F0
#define tres_4 0x00F0FF00
#define cuatro_1 0x000FFF00
#define cuatro_2 0x00F00F00
#define cuatro_3 0x0FFFFFF0
#define cuatro_4 0x00000F00
#define cinco_1 0x0FFF00F0
#define cinco_2_3 0x0F0F00F0
#define cinco_4 0x0F00FF00
#define seis_1 0x0FFFFFF0
#define seis_2_3 0x0F0F00F0
#define seis_4 0x0F0FFFF0
#define siete_1 0x0F000000
#define siete_2 0x0F00FFF0
#define siete_3 0x0F0F0000
#define siete_4 0x0FF00000
#define ocho_1_4 0x0FF00FF0
#define ocho_2_3 0x0F0FF0F0
#define nueve_1 0x0FFF0F00
#define nueve_2_3 0x0F0F00F0
#define nueve_4 0x0FFFFF00
#define cero_1_4 0x00FFFF00
#define cero_2_3 0x0F0000F0
#define asterisco_1_4 0x00F00F00
#define asterisco_2_3 0x000FF000
#define almohadilla_1 0x00000F00
#define almohadilla_2 0x00F00FF0
#define almohadilla_3 0x00FFFF00
#define almohadilla_4 0x0FF00FF0
#define almohadilla_5 0x00FFFF00
#define almohadilla_6 0x0FF00F00
#define almohadilla_7 0x00F00000
```

A.4 Fichero principal de la aplicación para control de PmodKYPD en PYNQ: helloworld.c

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "PmodKYPD.h"
#include "sleep.h"
#include "xil_io.h"
#include "driver_ip.h"


// keytable is determined as follows (indices shown in Keypad position below)
// 12 13 14 15
// 8  9  10 11
// 4  5  6  7
// 0  1  2  3
#define DEFAULT_KEYTABLE "0FED789C456B123A" // Son las filas de abajo a arriba de izq
a dcha

PmodKYPD myDevice;
XDriver gpio_ip;

void Driver_Demo5_Initialize(){//inicializar Bloque IP de la matriz de LEDS
	//En esta funcion podemos cerciorarnos de si el dispositivo se ha iniciado
correctamente o no.
	int status;
	//status = XGpio_Initialize(&gpio_ip, XPAR_DEMO5_0_DEVICE_ID);
	status = Driver_initialize(&gpio_ip, XPAR_DEMO5_0_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Demo_5 Initialization failed\n\r");
	}
	else{
		print("Info: Demo_5 Initialization successful\n\r");
	}

}

void KYPD_Initialize() {
   KYPD_begin(&myDevice, XPAR_PMODKYPD_0_AXI_LITE_GPIO_BASEADDR);
   KYPD_loadKeyTable(&myDevice, (u8*) DEFAULT_KEYTABLE);
//     PARA INDICAR QUE TODAS LAS SALIDAS SE COMPORTARAN COMO SALIDA DE ENTRADA DE
DATOS.
   Xil_Out32(myDevice.GPIO_addr, 0xF);
}

void Imprime_tecla(char key){
	switch(key)
		{
			case '0':
				xil_printf("/t Pulsada tecla 0 \n\r");
				//Desactivamos Init siempre al escribir datos en un
registro.
				Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
```

```
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Apagar_leds(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                case '1':
                        xil_printf("/t Pulsada tecla 1 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_1(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                case '2':
                        xil_printf("/t Pulsada tecla 2 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_2(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                case '3':
                        xil_printf("/t Pulsada tecla 3 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_3(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                case '4':
                        xil_printf("/t Pulsada tecla 4 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_4(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
```

```
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                    case '5':
                        xil_printf("/t Pulsada tecla 5 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_5(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                    case '6':
                        xil_printf("/t Pulsada tecla 6 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_6(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                    case '7':
                        xil_printf("/t Pulsada tecla 7 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_7(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                    case '8':
                        xil_printf("/t Pulsada tecla 8 \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_8(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
```

```
                    case '9':
                          xil_printf("/t Pulsada tecla 9 \n\r");
                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Imprimimos el caracter con los leds para verificar su
funcionamiento
                          Dibujar_9(&gpio_ip, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(1);
                          break;
                    case 'A':
                          xil_printf("/t Pulsada tecla A \n\r");
                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Imprimimos el caracter con los leds para verificar su
funcionamiento
                          Dibujar_A(&gpio_ip, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(1);
                          break;
                    case 'B':
                          xil_printf("/t Pulsada tecla B \n\r");
                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Imprimimos el caracter con los leds para verificar su
funcionamiento
                          Dibujar_B(&gpio_ip, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(1);
                          break;
                    case 'C':
                          xil_printf("/t Pulsada tecla C \n\r");
                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Imprimimos el caracter con los leds para verificar su
funcionamiento
                          Dibujar_C(&gpio_ip, 33, rojo);
                          //Activamos el init para que se comiencen a cargar
                          Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                          //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                          sleep(1);
                          break;
                    case 'D':
                          xil_printf("/t Pulsada tecla D \n\r");
                          //Desactivamos Init siempre al escribir datos en un
registro.
                          Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
```

```
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_D(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                  case 'E':
                        xil_printf("/t Pulsada tecla E \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_E(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                  case 'F':
                        xil_printf("/t Pulsada tecla F \n\r");
                        //Desactivamos Init siempre al escribir datos en un
registro.
                        Desactivar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Imprimimos el caracter con los leds para verificar su
funcionamiento
                        Dibujar_F(&gpio_ip, 33, rojo);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_DEMO5_0_S00_AXI_BASEADDR, 33);
                        //Damos tiempo a que se carguen los leds, y se llegue al
ultimo estado del diseño (sf)
                        sleep(1);
                        break;
                  default:
                        break;

            }
}


void runProject() {
   //Variables
   u16 keystate; //Estado de filas y columnas
   XStatus status, last_status = KYPD_NO_KEY; //Estado actual y anterior
   u8 key, last_key = 'x'; //Tecla actual y anterior

   // El valor inicial de la ultima tecla no puede ser contenida en la cadena de la
tabla de teclas cargada (x).

//   Xil_Out32(myDevice.GPIO_addr, 0xF);

   xil_printf("Pmod KYPD demo started. Press any key on the Keypad.\r\n");
   while (1) {
      // Captura el estado de cualquier tecla
      keystate = KYPD_getKeyStates(&myDevice);
```

```
      // Si solo se ha pulsado una tecla, determina cual es
      status = KYPD_getKeyPressed(&myDevice, keystate, &key);

      // Print key detect if a new key is pressed or if status has changed
      if (status == KYPD_SINGLE_KEY
            && (status != last_status || key != last_key)) {
         xil_printf("Key Pressed: %c\r\n", (char) key);
         last_key = key;
         Imprime_tecla(key);
      }
      else if (status == KYPD_MULTI_KEY && status != last_status)
         xil_printf("Error: Multiple keys pressed\r\n");

      last_status = status;

      usleep(1000);
   }
}

int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    Driver_Demo5_Initialize();
    KYPD_Initialize();

    runProject();
    return 0;
}
```

A.4.1 Libreria creada (v2): driver_ip.c

```
/**************************** Include Files *******************************/
#include "xil_io.h"
#include "driver_ip.h"
#include "xil_printf.h"

XDriver_Config XDriver_ConfigTable[XPAR_DEMO5_NUM_INSTANCES] =
{
      {
            XPAR_DEMO5_0_DEVICE_ID,
            XPAR_DEMO5_0_S00_AXI_BASEADDR,
      }
};

/************************** Function Prototypes ****************************/

/****************** Cargar todos los registros a 0 ********************/


void Apagar_leds(long unsigned int Base_ip_address, int num_reg)
{
      int i;
      int offset = 0;
      num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
```

```
	for(i = 0; i <= num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/****************************************************************/

/************Leer registros y mostrarlos por el puerto serie*************/

void Leer_registros(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int x = 0;
	unsigned long int buffer;
	int offset_read = 0x00;
	num_reg = num_reg - 1;    // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.


//¿Puedo cambiar la i por la x?

	for(i = 0; i <= num_reg; i++)
	{
	buffer = Xil_In32((Base_ip_address + offset_read));
	offset_read = offset_read + 0x4;
    xil_printf("\t Registro %d: %x\n\r", x, buffer);
    x++;
	}
}

/****************************************************************/
// num_led: Insertar entero entre 1 y 256, para nuestro diseño.
/****************************************************************/
/*************************Cargar un solo led*************************/

void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color)

{
	int puntero_reg = 0;
	int puntero_led = 0;
	int total_leds = 0;
	unsigned long int copia_color = 0;
	unsigned long int color_led = 0;
	unsigned long int buffer;
	int i;
	int puntero = 0;
	//1- Calcular el numero total de leds. Cada registro tiene 8 leds.
	//En el parámetro num_reg hay que descontar un registro, ya que el ultimo
registro es el de control
	//Y no corresponde a los leds.

	//Este numero (num_reg) lo usamos para comprobar que los parámetros que nos
han dado son correctos.
	num_reg = num_reg - 1;
	total_leds = num_reg*8; //Multiplicamos el numero de registros por el numero
de leds de cada registro; 8.
	//Comprobamos que el parámetro num_led esta dentro de rango
```

```
	if(num_led <= total_leds && num_led > 0)
	{
		//Parámetro correcto. Procedemos a cargar el led correspondiente

		//Creamos el puntero que apuntará al registro que contiene al led que queremos cargar
		//Las direcciones de memoria van de 4 en 4 para cada registro, y cada registro consta de 8 leds.
		//Basta con dividir el numero
		puntero_reg = num_led/8; // Ya tenemos el número del registro al que corresponde el led.
		puntero_led = num_led%8; //El resto nos indica en que posicion queda ese led dentro del registro correspondiente.

		//Como los leds estan conectados en zig zag, probablemente haya que alternar el orden del led, de izquierda a derecha
		//dependiendo de si el numero del registro es par o impar.

		// PENDIENTE


		//Una vez tenemos los dos punteros, solo faltaría cargar el color en la direccion asignada
		//Actualizamos el puntero con la dirección en hexadecimal
		if (puntero_led == 0)
		{
			puntero = Base_ip_address + (puntero_reg-1)*0x4;
		}
		else
		{
			puntero = Base_ip_address + (puntero_reg)*0x4;
		}
		//Una vez conocemos la direccion del registro, solo falta ajustar el color del led en una máscara adecuada
		//Con esa máscara podremos cargar el led que nos plazca sin alterar el resto
		//En nuestro caso se aplican dos mascaras, una para no altelar el contenido de los registros y borrar el del led
		//Otra para crear una variable que contenga el color (4 bits) en la posicion correcta dentro del registro

		copia_color = copia_color | color;
		for(i=0; i<7; i++)
		{
			copia_color = copia_color<<4 | color; //Creamos una variable que contenga el color repetido para todos los leds
		}
		xil_printf("\t Revisando puntero... \n\r");

		//Antes que nada leemos el registro en el que vamos a cargar los datos y lo guardamos en un buffer
		buffer = Xil_In32(puntero);

		//Comprobamos cual es el led que hay que cargar dentro del registro
		switch (puntero_led)
		{
			case 0:
				//Borramos el led que vamos a cargar en el buffer y dejamos el resto de leds con su valor.
				buffer = buffer & notled_8;
```

```
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_8;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                    case 1:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_1;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_1;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                    case 2:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_2;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_2;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                    case 3:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_3;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_3;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                    case 4:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_4;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_4;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
```

```
                        break;
                case 5:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_5;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_5;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 6:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_6;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_6;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 7:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_7;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_7;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                default:
                        break;
            }
        }
        else
        {}
}

/*********************************************************************/
/*******************************INIT ON*******************************/

void Activar_init(long unsigned int Base_ip_address, int num_reg)
{
        int puntero;
        unsigned long int buffer;
        //Primero hay que situar el puntero del registro de control (el ultimo de
todos)
        puntero = Base_ip_address + 0x4*(num_reg-1);
        //Ahora hay que sacar el contenido del registro al buffer
        buffer = Xil_In32(puntero);
```

```
	//Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del
init
	//Y hacer una operacion OR a esa máscara
	buffer = buffer | init_mask_on;
	Xil_Out32(puntero, buffer);
}

/*********************************************************************/
/*****************************RESET ON*****************************/

void Activar_reset(long unsigned int Base_ip_address, int num_reg) // El reset es
activo en baja
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga un 0 en la posicion del
reset y el resto a 1
	//Y hacer una operacion AND a esa máscara
	buffer = buffer & reset_mask_on;
	Xil_Out32(puntero, buffer);
}

/*********************************************************************/
/*****************************INIT OFF*****************************/

void Desactivar_init(long unsigned int Base_ip_address, int num_reg)
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 0 en la posicion del
init
	//Y hacer una operacion AND a esa máscara
	buffer = buffer & init_mask_off;
	Xil_Out32(puntero, buffer);
}

/*********************************************************************/
/*****************************RESET OFF*****************************/
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg) //El reset es
activo en baja
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del
init
```

```
	//Y hacer una operacion OR a esa máscara
	buffer = buffer | reset_mask_off;
	Xil_Out32(puntero, buffer);
}

/*****************************************************************/
/*****************Cargar todos los leds del mismo color*****************/
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}

	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************/
/*****************Cargar leds fila si y fila no*********************/
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = (num_reg - 1)/2; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/*****************************************************************/
```

```
/****************** Cargar led si y led no ****************************/
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<3;j++)
	{
		color_reg = color_reg <<8;
		color_reg = color_reg | color;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/
/***************** Cargar led si y led no zig zag ***********************/
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j, paridad;
	int offset = 0;
	unsigned long int color_reg_impar = 0;
	unsigned long int color_reg_par = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg_impar = color_reg_impar + color;
	color_reg_par = color_reg_par + color;
	color_reg_par = color_reg_par <<4;
	for(j=0;j<3;j++)
	{
		color_reg_impar = color_reg_impar <<8;
		color_reg_impar = color_reg_impar | color;
	}
	for(j=0;j<3;j++)
	{
		color_reg_par = color_reg_par <<4;
		color_reg_par = color_reg_par | color;
		color_reg_par = color_reg_par <<4;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
		paridad = i%2;
		if(paridad == 0)// el numero de registro es par
		{
```

```
                        Xil_Out32((Base_ip_address + offset), color_reg_par);
                        offset = offset + 0x4;
                }
                else// el numero de registro es impar
                {
                        Xil_Out32((Base_ip_address + offset), color_reg_impar);
                        offset = offset + 0x4;
                }
        }
}

/*****************************************************************************/
/**************** Iluminar nombre con los leds ************************/
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color)
{
        int i;
        int offset = 0;
        unsigned long int color_reg = 0;
        unsigned long int color_mask = 0;
        num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
        color_mask = color_mask | color;
        for(i=0;i<7;i++)
        {
                color_mask = color_mask<<4;
                color_mask = color_mask | color;
        }
        //Primeros 3 espacios
        for(i=0;i<3;i++)
        {
                Xil_Out32((Base_ip_address + offset), 0x00000000);
                offset = offset + 0x4;
        }
        //Letra C
        color_reg = color_mask & C1;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        color_reg = color_mask & C2_3_4;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        color_reg = color_mask & C2_3_4;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        color_reg = color_mask & C2_3_4;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        //Espacio
        Xil_Out32((Base_ip_address + offset), 0x00000000);
        offset = offset + 0x4;
        //Letra A
        color_reg = color_mask & A1_3;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        color_reg = color_mask & A2;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        color_reg = color_mask & A1_3;
        Xil_Out32((Base_ip_address + offset), color_reg);
        offset = offset + 0x4;
        //Espacio
        Xil_Out32((Base_ip_address + offset), 0x00000000);
```

```
offset = offset + 0x4;
//Letra R
color_reg = color_mask & R1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra L
color_reg = color_mask & L1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra O
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra S
color_reg = color_mask & S1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Ultimos 3 espacios
for(i=0;i<3;i++)
{
	Xil_Out32((Base_ip_address + offset), 0x00000000);
```

```
			offset = offset + 0x4;
		}
}

/*****************************************************************/
/************* Mover contenido de registros a la derecha ***************/
void Config_leds6(long unsigned int Base_ip_address, int num_reg)
{
		int i;
		int offset = 0;
		unsigned long int buffer = 0;
		unsigned long int buffer_posterior = 0;
		num_reg = num_reg - 1; // Se elige como limite "numreg-2" ya que el ultimo
registro se usará como registro de control
		//y no tendra INFO de color como el resto.
		//Y el anterior a ese se cargará fuera del bucle.
		//Leemos el primer dato
		buffer = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR);
		// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
		for(i = 0; i < num_reg; i++)
		{
			if(i == 31)
			{
				Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR, buffer); //Escribimos en
la siguiente dirección de registro
			}
			else
			{
				offset = offset + 0x4;
				buffer_posterior = Xil_In32(XPAR_DEMO5_0_S00_AXI_BASEADDR +
offset); //Leemos el siguiente dato para que no se pierda
				Xil_Out32(XPAR_DEMO5_0_S00_AXI_BASEADDR + offset, buffer);
//Escribimos en la siguiente dirección de registro
				buffer = buffer_posterior;
			}
		}
}

/*****************************************************************/
/************* Imprimir caracteres en el centro de los leds ***************/
/*Los caracteres se encenderán en un rectangulo centrado en la matriz, de 6x4 leds*/
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
		for(i=0;i<7;i++)
		{
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
```

```
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra A
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra B
	color_reg = color_mask & B4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B1;
```

```
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra C
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
```

```
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra D
	color_reg = color_mask & D4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
```

```
		offset = offset + 0x4;
	}

	//Numero 1
	color_reg = color_mask & uno_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 2
	color_reg = color_mask & dos_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
```

```
	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 3
	color_reg = color_mask & tres_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 4
	color_reg = color_mask & cuatro_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
```

```
	//Numero 5
	color_reg = color_mask & cinco_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 6
	color_reg = color_mask & seis_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
```

```
	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 7
	color_reg = color_mask & siete_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 8
	color_reg = color_mask & ocho_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
```

```
	//Numero 9
	color_reg = color_mask & nueve_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 0
	color_reg = color_mask & cero_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cero_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cero_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cero_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
```

```
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_E(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Caracter asterisco
	color_reg = color_mask & E_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_F(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 13 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Caracter almohadilla
	color_reg = color_mask & F_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 12 espacios
	for(i=0;i<12;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}


/****************************************************************************/
/********************** Inicializacion del driver ***********************/
int Driver_initialize(XDriver * InstancePtr, u16 DeviceId)
{
	XDriver_Config *ConfigPtr;

	/*
	 * Assert arguments
	 */
	Xil_AssertNonvoid(InstancePtr != NULL);

	/*
	 * Lookup configuration data in the device configuration table.
	 * Use this configuration info down below when initializing this
	 * driver.
	 */
	ConfigPtr = XDriver_LookupConfig(DeviceId);
	if (ConfigPtr == (XDriver_Config *) NULL) {
		InstancePtr->IsReady = 0;
		return (XST_DEVICE_NOT_FOUND);
	}

	return XDriver_CfgInitialize(InstancePtr, ConfigPtr,
```

```
                                ConfigPtr->BaseAddress);
}

XDriver_Config *XDriver_LookupConfig(u16 DeviceId)
{
	XDriver_Config *CfgPtr = NULL;

	int Index;

	for (Index = 0; Index < XPAR_DEMO5_NUM_INSTANCES; Index++) {
		if (XDriver_ConfigTable[Index].DeviceId == DeviceId) {
			CfgPtr = &XDriver_ConfigTable[Index];
			break;
		}
	}

	return CfgPtr;
}

int XDriver_CfgInitialize(XDriver * InstancePtr, XDriver_Config * Config,
			  UINTPTR EffectiveAddr)
{
	/* Assert arguments */
	Xil_AssertNonvoid(InstancePtr != NULL);

	/* Set some default values. */
	InstancePtr->BaseAddress = EffectiveAddr;
	/*
	 * Indicate the instance is now ready to use, initialized without error
	 */
	InstancePtr->IsReady = XIL_COMPONENT_IS_READY;
	return (XST_SUCCESS);
}
```

A.4.2 Libreria creada (v2): driver_ip.h

```
//includes
#include "xil_io.h"

/************************** Type Definitions ****************************/

/**
 * This typedef contains configuration information for the device.
 */
typedef struct {
	u16 DeviceId;		/**< Unique ID  of device */
	UINTPTR BaseAddress;	/**< Device base address */
} XDriver_Config;

/**
 * The XGpio driver instance data. The user is required to allocate a
 * variable of this type for every GPIO device in the system. A pointer
 * to a variable of this type is then passed to the driver API functions.
 */
typedef struct {
	UINTPTR BaseAddress;	/**< Device base address */
	u32 IsReady;		/**< Device is initialized and ready */
} XDriver;
/************************************************************************/
```

```
//Functions

//Initialize functions

int Driver_initialize(XDriver *InstancePtr, u16 DeviceId);
XDriver_Config *XDriver_LookupConfig(u16 DeviceId);
int XDriver_CfgInitialize(XDriver *InstancePtr, XDriver_Config * Config, UINTPTR
EffectiveAddr);

// Estas funciones solo van a colocar los valores de los leds en las filas
correspondientes.
void Apagar_leds(long unsigned int Base_ip_address, int num_reg);
void Leer_registros(long unsigned int Base_ip_address, int num_reg);
void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color);
void Activar_init(long unsigned int Base_ip_address, int num_reg);
void Activar_reset(long unsigned int Base_ip_address, int num_reg);
void Desactivar_init(long unsigned int Base_ip_address, int num_reg);
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg);
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds6(long unsigned int Base_ip_address, int num_reg);
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_E(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_F(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_pantalla_de_inicio(XDriver *Puntero_inst, int num_reg, short int color);

//Mascaras para los leds:

//Estas máscaras se han definido con MSB como el led 8 (izquierda) y LSB el led 1
(derecha)

#define led_8 0xF0000000 //Led 8
#define led_7 0x0F000000 //Led 7
#define led_6 0x00F00000 //Led 6
#define led_5 0x000F0000 //Led 5
#define led_4 0x0000F000 //Led 4
#define led_3 0x00000F00 //Led 3
#define led_2 0x000000F0 //Led 2
#define led_1 0x0000000F //Led 1

//Mascaras inversas
#define notled_8 0x0FFFFFFF //Led 8
#define notled_7 0xF0FFFFFF //Led 7
```

```
#define notled_6 0xFF0FFFFF //Led 6
#define notled_5 0xFFF0FFFF //Led 5
#define notled_4 0xFFFF0FFF //Led 4
#define notled_3 0xFFFFF0FF //Led 3
#define notled_2 0xFFFFFF0F //Led 2
#define notled_1 0xFFFFFFF0 //Led 1

//Mascaras registro de control
#define init_mask_on      0x00000002
#define init_mask_off   0xFFFFFFFD
#define reset_mask_on     0xFFFFFFFE
#define reset_mask_off  0x00000001

//Mascaras para las letras
#define A1_3 0x0FFFFFF0
#define A2 0x0F00F000

#define B1 0x0FFFFFF0
#define B2_3 0x0F0FF0F0
#define B4 0x0FF00FF0

#define C1 0x0FFFFFF0
#define C2_3_4 0x0F0000F0

#define D1 0x0FFFFFF0
#define D2 0x0F0000F0
#define D3 0x00F00F00
#define D4 0x000FF000

#define R1 0x0FFFFFF0
#define R2 0x0F0FF000
#define R3 0x0F0F0F00
#define R4 0x0FFF00F0

#define L1 0x0FFFFFF0
#define L2_3 0x000000F0

#define O1_3 0x0FFFFFF0
#define O2 0x0F0000F0

#define S1 0x0FFF00F0
#define S2 0x0F0F00F0
#define S3 0x0F00F0F0
#define S4 0x0F00FFF0

#define uno_1 0x000F00F0
#define uno_2 0x00F000F0
#define uno_3 0x0FFFFFF0
#define uno_4 0x000000F0

#define dos_1 0x00F00FF0
#define dos_2 0x0F000FF0
#define dos_3 0x0F00F0F0
#define dos_4 0x00FF00F0

#define tres_1 0x00F00F00
#define tres_2 0x0F0000F0
#define tres_3 0x0F0F00F0
#define tres_4 0x00F0FF00
```

```
#define cuatro_1 0x000FFF00
#define cuatro_2 0x00F00F00
#define cuatro_3 0x0FFFFFF0
#define cuatro_4 0x00000F00

#define cinco_1 0x0FFF00F0
#define cinco_2_3 0x0F0F00F0
#define cinco_4 0x0F00FF00

#define seis_1 0x0FFFFFF0
#define seis_2_3 0x0F0F00F0
#define seis_4 0x0F0FFFF0

#define siete_1 0x0F000000
#define siete_2 0x0F00FFF0
#define siete_3 0x0F0F0000
#define siete_4 0x0FF00000

#define ocho_1_4 0x0FF00FF0
#define ocho_2_3 0x0F0FF0F0

#define nueve_1 0x0FFF0F00
#define nueve_2_3 0x0F0F00F0
#define nueve_4 0x0FFFFF00

#define cero_1_4 0x00FFFF00
#define cero_2_3 0x0F0000F0

#define E_1 0x0FFFFFF0
#define E_2_3 0x0F0FF0F0
#define E_4 0x0F0000F0

#define F_1 0x0FFFFFF0
#define F_2_3 0x0F0F0000
#define F_4 0x0F000000

#define A_corta_1_4 0x00FFFFF0
#define A_corta_2_3 0x00F0F000

#define dos_puntos 0x000F0F00

#define E_corta_3 0x00F000F0
#define E_corta_2 0x00F0F0F0
#define E_corta_1 0x00FFFFF0

#define S_corta_3 0x00F0FFF0
#define S_corta_2 0x00F0F0F0
#define S_corta_1 0x00FFF0F0

#define L_corta_3 0x000000F0
#define L_corta_2 0x000000F0
#define L_corta_1 0x00FFFFF0

#define U_corta_1_3 0x00FFFFF0
#define U_corta_2 0x000000F0

#define P_corta_3 0x00FFF000
#define P_corta_2 0x00F0F000
#define P_corta_1 0x00FFFFF0
```

```
// Colores

#define apagar 0b0000 //0x0
#define rojo 0b0001 // 0x1
#define verde 0b0010 // 0x2
#define azul 0b0011 // 0x3
#define magenta 0b0100 // o 0x4
#define amarillo 0b0101 // o 0x5
#define cyan 0b0110 // o 0x6
#define gris 0b0100 // o 0x4
#define blanco 0b1000 // o 0x8
```

A.5 Fichero principal de la aplicación: freertos_helloworld.c

```
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include <stdlib.h>
#include <stdio.h>
#include "PmodKYPD.h"
//#include "sleep.h"
#include "xil_io.h"
#include "driver_ip.h"

#define DELAY_1_SECOND          1000UL
#define DELAY_0_9_SECOND        900UL
#define DEFAULT_KEYTABLE "0FED789C456B123A" // Son las filas de abajo a arriba de izq
a dcha
#define NUEVO_LED 0x10000000 //Color verde en el Bit MSB
#define incremento_caida 250
#define incremento_caida_2 50
/*-----------------------------------------------------------*/

/* Declaración de las tareas del teclado y de la maquina de estados. */
static void prvKYPDTask( void *pvParameters );
static void Maquina_de_Estados_Task( void *pvParameters );
/*-----------------------------------------------------------*/

/* Manejadores para las tareas. */
static TaskHandle_t xKYPDTask;
static TaskHandle_t xMaq_estadosTask;

/* Estructuras para las funciones que controlan los módulos IP creados en Vivado. */

PmodKYPD myDevice;
XDriver custom_ip;


/* Variables globales. */
int slicer_izq = 0;//Número entre 4 y 31.
int slicer_dcha = 0;//Número entre 0 y 27.
```

```
int estado = 0; //Variable de estado para la tarea Maquina de estados.
int x = 0; //
int y = 0; //
int siguiente_caida = 2; // Indica el ciclo de la proxima caida. Se inicializa en 2
int DELAY_FALL = 0; //Delay para la caida de los leds. Lo inicializamos a 0.
int delay = 3000; //
int Puntuacion = 0; //Puntuación del juego.
int colision = 0; //colision:
                                //0: no ha caido el led.
                                //1: caida y no colision con barra / Game over.
                                //2: caida y colision con barra.
_Bool Game_over = FALSE; //Flag para indicar el final del juego.
_Bool parpadeo = FALSE; //Flag para hacer un parpadeo al mostrar la puntuación.

/* Funciones. */

void Driver_LEDS_RGB_Initialize()//inicialización Bloque IP de la matriz de LEDS
{
	//En esta funcion podemos cerciorarnos de si el dispositivo se ha iniciado
correctamente o no.
	int status;
	status = Driver_initialize(&custom_ip, XPAR_LEDS_RGB_32X8_0_DEVICE_ID);
	if(status != XST_SUCCESS){
		print("Err: Demo_5 Initialization failed\n\r");
	}
	else{
		print("Info: Demo_5 Initialization successful\n\r");
	}

}

void KYPD_Initialize()//Inicialización del bloque de control para el PMODB para el
teclado.
{
   KYPD_begin(&myDevice, XPAR_PMODKYPD_0_AXI_LITE_GPIO_BASEADDR);
   KYPD_loadKeyTable(&myDevice, (u8*) DEFAULT_KEYTABLE);
//     PARA INDICAR QUE TODAS LAS SALIDAS SE COMPORTARAN COMO SALIDA DE ENTRADA DE
DATOS.
   Xil_Out32(myDevice.GPIO_addr, 0xF);
}

void Inicializacion_del_juego(XDriver *Puntero_inst)
{
	u32 Buffer[32] = {};//Inicializamos el buffer a 0 (IMPORTANTE)
	u32 Address;
	int offset_read = 0;
	int i;

	Address = Puntero_inst->BaseAddress;
	slicer_dcha=14;
	slicer_izq=17;

	//Desactivamos Init siempre al escribir datos en un registro.
	Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);

	//Esta barra (slicer) ocupará 4 leds. Y Se insertaran en el centro de la
matriz
	for(i=0;i<32;i++)
	{
		//Añadimos la ubicacion del slicer
		if(i == slicer_dcha)
```

```
		{
			Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro
			Buffer[i+1] = Buffer[i+1] | azul; //Activamos el slicer en este
registro
			Buffer[i+2] = Buffer[i+2] | azul; //Activamos el slicer en este
registro
			Buffer[i+3] = Buffer[i+3] | azul; //Activamos el slicer en este
registro	Hay que cambiar el 0xF por el color
		}
	}

	/*------CARGA-----*/
	offset_read = 0x0;
	for(i=0;i<32;i++)
	{
		Xil_Out32(Address + offset_read, Buffer[i]);
		offset_read = offset_read + 0x4;
	}

	//Activamos el init para que se comiencen a cargar
	Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
	/*----------------*/

}

void Desplazar_barra_dcha(XDriver *Puntero_inst)
{

		u32 Buffer[32]= {}; //Inicializamos el buffer a 0 (IMPORTANTE)
		u32 Address;
		int offset_read = 0;
		int i;

		Address = Puntero_inst->BaseAddress;
		//Actualizamos los parametros extremos de la barra
		if(slicer_dcha != 0)//Creamos este condicional para que no sobrepase el
extremo derecho la barra (slicer)
		{
			slicer_dcha = slicer_dcha - 1;
			slicer_izq = slicer_izq - 1;

		}

		//Desactivamos Init siempre al escribir datos en un registro.
		Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);

		//Leemos los registros y los vamos actualizando 1 a 1.
		for(i=0;i<32;i++)
		{
			Buffer[i] = Xil_In32(Address + offset_read);
			offset_read = offset_read + 0x4;

			//Añadimos la ubicacion del slicer.
			if(i == slicer_dcha)
			{
				Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro.
			}
			else if(i == slicer_dcha+1)
```

```
                {
                        Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro.
                }
                else if(i == slicer_dcha+2)
                {
                        Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro.
                }
                else if(i == slicer_izq)
                {
                        Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro.
                }
                //Al ser desplazamiento a la derecha, si cogemos el valor del
slicer_izq y le sumamos 1 tendremos la esquina anterior que habra que borrar.
                else if(i == slicer_izq + 1)
                {
                        Buffer[i] = Buffer[i] & 0xFFFFFFF0; //Desactivamos el
slicer en este registro.
                }
        }

        /*------CARGA-----*/
        offset_read = 0x0;
        for(i=0;i<32;i++)
        {
                Xil_Out32(Address + offset_read, Buffer[i]);
                offset_read = offset_read + 0x4;
        }

        //Activamos el init para que se comiencen a cargar
        Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
        /*----------------*/
}

void Desplazar_barra_izq(XDriver *Puntero_inst)
{

        u32 Buffer[32] = {}; //Inicializamos el buffer a 0 (IMPORTANTE)
        u32 Address;
        int offset_read = 0;
        int i;

        Address = Puntero_inst->BaseAddress;
        //Actualizamos los parametros extremos de la barra
        if(slicer_izq != 31)//Creamos este condicional para que no sobrepase el
extremo izquierdo la barra (slicer)
        {
                slicer_dcha = slicer_dcha + 1;
                slicer_izq = slicer_izq + 1;
        }

        //Desactivamos Init siempre al escribir datos en un registro.
        Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);

        for(i=0;i<32;i++)
        {
                Buffer[i] = Xil_In32(Address + offset_read);
                offset_read = offset_read + 0x4;
```

```
                    //Añadimos la ubicacion del slicer
                    if(i == slicer_dcha)
                    {
                          Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro
                    }
                    else if(i == slicer_dcha+1)
                    {
                          Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro
                    }
                    else if(i == slicer_dcha+2)
                    {
                          Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro
                    }
                    else if(i == slicer_izq)
                    {
                          Buffer[i] = Buffer[i] | azul; //Activamos el slicer en este
registro.
                    }
                    //Al ser desplazamiento a la izquierda, si cogemos el valor del
slicer_dcha y le restamos 1 tendremos la esquina anterior que habra que borrar.
                    else if(i == slicer_dcha - 1)
                    {
                          Buffer[i] = Buffer[i] & 0xFFFFFFF0; //Desactivamos el
slicer en este registro.
                    }

             }

             /*------CARGA-----*/
             offset_read = 0x0;
             for(i=0;i<32;i++)
             {
                    Xil_Out32(Address + offset_read, Buffer[i]);
                    offset_read = offset_read + 0x4;
             }

             //Activamos el init para que se comiencen a cargar
             Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
             /*----------------*/
}

int Generacion_bolas(void) //Funcion para generar un número aleatorio para introducir
una bola nueva.
{
      int bola;
      // Generamos un número aleatorio
      bola = rand() %32; //Establecemos el valor limite para el numero aleatorio.
      return bola;
}

void Led_fall(XDriver *Puntero_inst) //Desplazamiento vertical
{

      u32 Address = 0;
      u32 Buffer[32] = {};
      u32 bola_caida = 0; //Flag que nos indicará si una bola (led) ha llegado al
"suelo" en el ciclo de caida.
```

```
	int offset_read = 0x00;
	int i;
	int bola_nueva = 0; //Variable que usaremos para almacenar la posicion donde caerá la siguiente bola (led).
	_Bool introducir_bola = FALSE; //Flag que se usará para indicar cuando hay que introducir una bola nueva.

	//Desactivamos Init siempre al escribir datos en un registro.
	Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);

	//Lo que hay que hacer en cada ciclo que se ejecute es desplazar los bits de cada registro una posicion hacia abajo (en este caso sería
	//una posición a la derecha) (Hacia el bit LSB).

	//Bajada con introducción de BOLA NUEVA
	if(x == siguiente_caida)//Se introducirá una nueva bola cuando lo indique "siguiente_caida"
	{
		bola_nueva = Generacion_bolas();
		introducir_bola = TRUE;
		siguiente_caida = (rand() %4) + 2 + x;
	}

	/*---Descarga y actualización---*/
	Address = Puntero_inst->BaseAddress;
	for(i=0;i<32;i++)
	{
		Buffer[i] = Xil_In32(Address + offset_read);
		offset_read = offset_read + 0x4;

		//BAJADA
		Buffer[i] = Buffer[i] >> 4;
		bola_caida = Buffer[i] & 0xF; // Aplicamos máscara para ver si alguna bola ha llegado al suelo
		if(bola_caida == rojo)//Si una bola (led) ha llegado al suelo, evaluamos:
		{
			if(slicer_dcha == i)
			{
				colision = 2;
			}
			else if(slicer_dcha + 1 == i)
			{
				colision = 2;
			}
			else if(slicer_dcha + 2 == i)
			{
				colision = 2;
			}
			else if(slicer_dcha + 3 == i)
			{
				colision = 2;
			}
			else//Si no coincide con ninguna posicion del slicer -> Game Over
			{
				colision = 1;
			}
		}
		else//Si NO ha llegado al suelo, continuamos con la caida
		{
			colision = 0;
```

```
		}

		switch(colision)
		{
			case 0://Seguimos con el juego.

				//Añadimos la ubicacion del slicer

				if(i == slicer_dcha)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				else if(i == slicer_dcha+1)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				else if(i == slicer_dcha+2)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				else if(i == slicer_izq)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				if(i == bola_nueva)
				{
					if(introducir_bola == TRUE)
					{
						Buffer[i] = Buffer[i] | NUEVO_LED;
					}
					introducir_bola = FALSE;
				}

				break;
			case 1://La bola ha caido, Game over. Lo indicamos en la variable
"Game_over" y Esperamos a que termine el bucle for.
				Game_over = TRUE;
				break;
			case 2://La bola ha caido, pero ha colisionado, seguimos con el
juego.

				Puntuacion = Puntuacion + 1;

				if(i == slicer_dcha)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				else if(i == slicer_dcha+1)
				{
					Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
				}
				else if(i == slicer_dcha+2)
				{
```

```
                                Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
                        }
                        else if(i == slicer_izq)
                        {
                                Buffer[i] = Buffer[i] | azul; //Activamos el slicer
en este registro
                        }
                        if(i == bola_nueva)
                        {
                                if(introducir_bola == TRUE)
                                {
                                        Buffer[i] = Buffer[i] | NUEVO_LED;
                                }
                                introducir_bola = FALSE;
                        }
                        break;
                default:
                        break;
        }
        //Añadimos la ubicacion del slicer

    }
    /*------CARGA-----*/
    offset_read = 0x0;
    for(i=0;i<32;i++)
    {
        Xil_Out32(Address + offset_read, Buffer[i]);
        offset_read = offset_read + 0x4;
    }

    //Activamos el init para que se comiencen a cargar
    Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
    /*----------------*/

    //Vemos si continuamos jugando o no
    if(Game_over == TRUE)
    {
        estado = 3;
        Game_over = FALSE;
    }
    else
    {
        x = x + 1;

        if(x%5 == 0)//Cada 5 ciclos se acelerará la caida.
        {

                if(delay > DELAY_1_SECOND)//Habrá una aceleracion de 250 ms hasta
que nuestra velocidad de caida sea de 1 segundo.
                {
                        delay = delay - incremento_caida;
                }
                else if(delay <= DELAY_1_SECOND)//Una vez que nuestra velocidad
es de 1 segundo, ahora la aceleración será de 50 ms, hasta llegar a una velocidad de
200 ms
                //que será nuestra velocidad máxima.
                {
                        if(delay != DELAY_0_9_SECOND) //Velocidad limite de caida
                        {
                                delay = delay - incremento_caida_2;
```

```
                        }
                        else
                        {
                                delay = 2500;
                        }
                }
        }
        DELAY_FALL = pdMS_TO_TICKS(delay);
    }
}

/*-----------------------------------------------------------*/
int main( void )
{
    xil_printf( "Hello from Freertos example main\r\n" );

    Driver_LEDS_RGB_Initialize();

    KYPD_Initialize();

#if ( configSUPPORT_STATIC_ALLOCATION == 0 ) /* Normal or standard use case */
    //Creamos las 2 tareas.
    xTaskCreate( prvKYPDTask,
                        ( const char * ) "KYPD",
                        configMINIMAL_STACK_SIZE,
                        NULL,
                        tskIDLE_PRIORITY,
                        &xKYPDTask);

    xTaskCreate( Maquina_de_Estados_Task,
                        ( const char * ) "Maq_estados",
                        configMINIMAL_STACK_SIZE,
                        NULL,
                        tskIDLE_PRIORITY + 1,
                        &xMaq_estadosTask);

#else /* Use case where memories for tasks/queues/timers etc are provided statically
by the users */
    xil_printf( "Using static memory for tasks, queue and timer creations. \r\n"
);

#endif

    /* Arrancamos el planificador en tiempo real. */
    vTaskStartScheduler();

    /* If all is well, the scheduler will now be running, and the following line
    will never be reached.  If the following line does execute, then there was
    insufficient FreeRTOS heap memory available for the idle and/or timer tasks
    to be created.  See the memory management section on the FreeRTOS web site
    for more details. */

    print("Successfully ran Initialization\n\r");

    for( ;; );
}

/*-----------------------------------------------------------*/

/* Tareas. */
```

```
static void prvKYPDTask( void *pvParameters )
{
	//Variables
	u16 keystate; //Estado de filas y columnas
	XStatus status, last_status = KYPD_NO_KEY; //Estado actual y anterior
	u8 key, last_key = 'x'; //Tecla actual y anterior

	for( ;; )
	{
		// El valor inicial de la ultima tecla no puede ser contenida en la cadena de la tabla de teclas cargada (x).

		//  xil_printf("Pmod KYPD demo started. Press any key on the Keypad.\r\n");

			// Captura el estado de cualquier tecla
			keystate = KYPD_getKeyStates(&myDevice);

			// Si solo se ha pulsado una tecla, determina cual es
			status = KYPD_getKeyPressed(&myDevice, keystate, &key);

			//Detecta si se ha pulsado una nueva tecla o si el estado (status) ha cambiado
			if (status == KYPD_SINGLE_KEY
				&& (status != last_status || key != last_key))
			{
			   xil_printf("Key Pressed: %c\r\n", (char) key);
			   last_key = key;
			   switch(key)
			   {
				case 'A'://Tecla Init para nuestro juego
					if(estado == 0)
					{
						estado = 1;
					}
					//Si no estamos en el caso inicial, esta tecla no tendrá ningun uso.
					break;
				case 'D'://Tecla para terminar el juego. Pasariamos al estado de reposo (estado 0)
					if(estado != 0)
					{
						estado = 0; //Pasamos al estado inicial.
					}
					break;
				case '4'://Tecla para desplazar la barra un led a la derecha
					//Antes que nada comprobamos si estamos con el juego en marcha:
					if(estado == 2)
					{
						//Ahora tenemos que cambiar la posición de la barra sin alterar el resto de leds.
						Desplazar_barra_izq(&custom_ip);
					}
					break;
				case '6'://Tecla para desplazar la barra un led a la izquierda
					//Antes que nada comprobamos si estamos con el juego en marcha:
```

```
                                    if(estado == 2)
                                    {
                                          //Ahora tenemos que cambiar la posición de la
barra sin alterar el resto de leds.
                                          Desplazar_barra_dcha(&custom_ip);
                                    }
                                    break;
                              default:
                                    break;
                        }

                  }
                  else if (status == KYPD_MULTI_KEY && status != last_status)
                    {
                          xil_printf("Error: Multiple keys pressed\r\n");
                    }
                  last_status = status;

      }
}

static void Maquina_de_Estados_Task( void *pvParameters )
{
      //Variables
      int DELAY_PANTALLA_INICIO = 0;
      int DELAY_PANTALLA_PUNTOS = 0;
      DELAY_PANTALLA_INICIO = pdMS_TO_TICKS(1500);
      DELAY_PANTALLA_PUNTOS = pdMS_TO_TICKS(1000);

      for( ;; )
      {
            switch (estado)
            {
            case 0://Caso de reposo.

                  //Se mostrara "PULSA : A" con los leds.
                  //Una vez se haya pulsado A, el juego se inicializará.

                  //Desactivamos Init siempre al escribir datos en un registro.
                  Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                  Dibujar_pantalla_de_inicio(&custom_ip, blanco);
                  //Activamos el init para que se comiencen a cargar
                  Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                  //Pequeño delay para que se carguen bien los leds.
                  vTaskDelay(DELAY_PANTALLA_INICIO);
                  break;
            case 1://Inicialización del juego.

                  Puntuacion = 0;
                  Inicializacion_del_juego(&custom_ip);
                  //Una vez se haya inicializado ponemos en marcha el juego:
                  estado = 2;
                  break;
            case 2://Juego en marcha.

                  //Aqui le daremos marcha a la caida de los leds.
                  Led_fall(&custom_ip);
                  //Este será el delay de la caida de los leds, que ira
disminuyendo conforme mas lejos se llegue en el juego
                  vTaskDelay(DELAY_FALL);
```

```
                        break;
                case 3://Fin del juego, Limpiar la pantalla.Reiniciar variable de
velocidad de caida
                        x=0;
                        y=0;
                        delay = 3000;
                        siguiente_caida=0;
                        //Desactivamos Init siempre al escribir datos en un registro.
                        Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                        //Dibujar_pantalla_de_inicio(&custom_ip, 33, rojo);
                        Apagar_leds(&custom_ip, 33);
                        //Activamos el init para que se comiencen a cargar
                        Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                        //Pequeño delay para que se carguen bien los leds.
                        vTaskDelay(DELAY_PANTALLA_PUNTOS);
                        estado = 4;
                        break;
                case 4://Mostramos puntuacion parpadeando.

                        if(parpadeo == FALSE)
                        {
                                //Desactivamos Init siempre al escribir datos en un
registro.
                                Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                                Mostrar_puntuacion(&custom_ip, cyan, Puntuacion);
                                //Activamos el init para que se comiencen a cargar
                                Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                                //Pequeño delay para que se carguen bien los leds.
                                vTaskDelay(DELAY_PANTALLA_PUNTOS);
                                //Activamos flag
                                parpadeo = TRUE;
                        }
                        else
                        {
                                //Desactivamos Init siempre al escribir datos en un
registro.
                                Desactivar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                                Apagar_leds(&custom_ip, 15);//Son menos registros (15) ya
que solo queremos que parpadeen los numeros
                                //Activamos el init para que se comiencen a cargar
                                Activar_init(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, 33);
                                //Pequeño delay para que se carguen bien los leds.
                                vTaskDelay(DELAY_PANTALLA_PUNTOS);
                                //Desactivamos flag
                                parpadeo = FALSE;
                        }
                        break;
                default:
                        break;
                }

        }

}
```

A.5.1 Libreria creada (versión del juego): driver_ip.c

Solo se incluirán aquí aquellas funciones que se han usado en el diseño final del juego, las versiones v1 y v2 contienen el resto de funciones que se han creado durante el desarrollo del proyecto.

```
/**************************** Include Files *****************************/
#include "xil_io.h"
#include "driver_ip.h"
#include "xil_printf.h"

XDriver_Config XDriver_ConfigTable[XPAR_LEDS_RGB_32X8_NUM_INSTANCES] =
{
	{
	    XPAR_LEDS_RGB_32X8_0_DEVICE_ID,
	       XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR,
	}
};

/************************** Function Prototypes *****************************/

/****************** Cargar todos los registros a 0 ********************/


void Apagar_leds(XDriver *Puntero_inst, int num_reg)
{
	int i;
	int offset = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	for(i = 0; i <= num_reg; i++)
	{
	Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/

/************Leer registros y mostrarlos por el puerto serie*************/

void Leer_registros(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int x = 0;
	unsigned long int buffer;
	int offset_read = 0x00;
	num_reg = num_reg - 1;    // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.


//¿Puedo cambiar la i por la x?

	for(i = 0; i <= num_reg; i++)
	{
	buffer = Xil_In32((Base_ip_address + offset_read));
	offset_read = offset_read + 0x4;
    xil_printf("\t Registro %d: %x\n\r", x, buffer);
    x++;
	}
}

/*****************************************************************************/
// num_led: Insertar entero entre 1 y 256, para nuestro diseño.
```

```
/*****************************************************************/
/*************************Cargar un solo led**********************/

void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color)

{
	int puntero_reg = 0;
	int puntero_led = 0;
	int total_leds = 0;
	unsigned long int copia_color = 0;
	unsigned long int color_led = 0;
	unsigned long int buffer;
	int i;
	int puntero = 0;
	//1- Calcular el numero total de leds. Cada registro tiene 8 leds.
	//En el parámetro num_reg hay que descontar un registro, ya que el ultimo
registro es el de control
	//Y no corresponde a los leds.

	//Este numero (num_reg) lo usamos para comprobar que los parámetros que nos
han dado son correctos.
	num_reg = num_reg - 1;
	total_leds = num_reg*8; //Multiplicamos el numero de registros por el numero
de leds de cada registro; 8.
	//Comprobamos que el parámetro num_led esta dentro de rango
	if(num_led <= total_leds && num_led > 0)
	{
		//Parámetro correcto. Procedemos a cargar el led correspondiente

		//Creamos el puntero que apuntará al registro que contiene al led que
queremos cargar
		//Las direcciones de memoria van de 4 en 4 para cada registro, y cada
registro consta de 8 leds.
		//Basta con dividir el numero
		puntero_reg = num_led/8; // Ya tenemos el número del registro al que
corresponde el led.
		puntero_led = num_led%8; //El resto nos indica en que posicion queda ese
led dentro del registro correspondiente.

		//Como los leds estan conectados en zig zag, probablemente haya que
alternar el orden del led, de izquierda a derecha
		//dependiendo de si el numero del registro es par o impar.

		// PENDIENTE


		//Una vez tenemos los dos punteros, solo faltaría cargar el color en la
direccion asignada
		//Actualizamos el puntero con la dirección en hexadecimal
		if (puntero_led == 0)
		{
			puntero = Base_ip_address + (puntero_reg-1)*0x4;
		}
		else
		{
			puntero = Base_ip_address + (puntero_reg)*0x4;
		}
		//Una vez conocemos la direccion del registro, solo falta ajustar el
color del led en una máscara adecuada
```

```
		//Con esa máscara podremos cargar el led que nos plazca sin alterar el
resto
		//En nuestro caso se aplican dos mascaras, una para no altelar el
contenido de los registros y borrar el del led
		//Otra para crear una variable que contenga el color (4 bits) en la
posicion correcta dentro del registro

		copia_color = copia_color | color;
		for(i=0; i<7; i++)
		{
			copia_color = copia_color<<4 | color; //Creamos una variable que
contenga el color repetido para todos los leds
		}
		xil_printf("\t Revisando puntero... \n\r");

		//Antes que nada leemos el registro en el que vamos a cargar los datos y
lo guardamos en un buffer
		buffer = Xil_In32(puntero);

		//Comprobamos cual es el led que hay que cargar dentro del registro
		switch (puntero_led)
		{
			case 0:
				//Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
				buffer = buffer & notled_8;
				//Aplicamos máscara para tener el color solo en la posición
del led deseado
				color_led = copia_color & led_8;
				//Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
				break;
			case 1:
				//Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
				buffer = buffer & notled_1;
				//Aplicamos máscara para tener el color solo en la posición
del led deseado
				color_led = copia_color & led_1;
				//Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
				break;
			case 2:
				//Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
				buffer = buffer & notled_2;
				//Aplicamos máscara para tener el color solo en la posición
del led deseado
				color_led = copia_color & led_2;
				//Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
				color_led = color_led | buffer;
				// Cargamos el led indicado
				Xil_Out32(puntero, color_led);
```

```
                        break;
                case 3:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_3;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_3;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 4:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_4;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_4;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 5:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_5;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_5;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 6:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_6;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_6;
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                case 7:
                        //Borramos el led que vamos a cargar en el buffer y dejamos
el resto de leds con su valor.
                        buffer = buffer & notled_7;
                        //Aplicamos máscara para tener el color solo en la posición
del led deseado
                        color_led = copia_color & led_7;
```

```
                        //Escribimos el dato del color dentro de los que ya habia
cargados inicialmente en el buffer
                        color_led = color_led | buffer;
                        // Cargamos el led indicado
                        Xil_Out32(puntero, color_led);
                        break;
                  default:
                        break;
            }
      }
      else
      {}
}

/****************************************************************************/
/*******************************INIT ON*******************************/

void Activar_init(long unsigned int Base_ip_address, int num_reg)
{
      int puntero;
      unsigned long int buffer;
      //Primero hay que situar el puntero del registro de control (el ultimo de
todos)
      puntero = Base_ip_address + 0x4*(num_reg-1);
      //Ahora hay que sacar el contenido del registro al buffer
      buffer = Xil_In32(puntero);
      //Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del
init
      //Y hacer una operacion OR a esa máscara
      buffer = buffer | init_mask_on;
      Xil_Out32(puntero, buffer);
}

/****************************************************************************/
/*******************************RESET ON*******************************/

void Activar_reset(long unsigned int Base_ip_address, int num_reg) // El reset es
activo en baja
{
      int puntero;
      unsigned long int buffer;
      //Primero hay que situar el puntero del registro de control (el ultimo de
todos)
      puntero = Base_ip_address + 0x4*(num_reg-1);
      //Ahora hay que sacar el contenido del registro al buffer
      buffer = Xil_In32(puntero);
      //Ahora bastaría con aplicar una máscara que contenga un 0 en la posicion del
reset y el resto a 1
      //Y hacer una operacion AND a esa máscara
      buffer = buffer & reset_mask_on;
      Xil_Out32(puntero, buffer);
}

/****************************************************************************/
/*******************************INIT OFF*******************************/

void Desactivar_init(long unsigned int Base_ip_address, int num_reg)
{
      int puntero;
      unsigned long int buffer;
```

```
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 0 en la posicion del
init
	//Y hacer una operacion AND a esa máscara
	buffer = buffer & init_mask_off;
	Xil_Out32(puntero, buffer);
}

/*****************************************************************************/
/*****************************RESET OFF*****************************************/
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg) //El reset es
activo en baja
{
	int puntero;
	unsigned long int buffer;
	//Primero hay que situar el puntero del registro de control (el ultimo de
todos)
	puntero = Base_ip_address + 0x4*(num_reg-1);
	//Ahora hay que sacar el contenido del registro al buffer
	buffer = Xil_In32(puntero);
	//Ahora bastaría con aplicar una máscara que contenga el 1 en la posicion del
init
	//Y hacer una operacion OR a esa máscara
	buffer = buffer | reset_mask_off;
	Xil_Out32(puntero, buffer);
}

/*****************************************************************************/
/*****************Cargar todos los leds del mismo color*****************/
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta
llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}

	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************************/
/*****************Cargar leds fila si y fila no*************************/
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color)
{
```

```
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = (num_reg - 1)/2; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<7;j++)
	{
		color_reg = color_reg <<4;
		color_reg = color_reg | color;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	}
}

/*****************************************************************/
/***************** Cargar led si y led no ***********************/
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j;
	int offset = 0;
	unsigned long int color_reg = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta llenar los 32 bits del registro
	color_reg = color_reg + color;
	for(j=0;j<3;j++)
	{
		color_reg = color_reg <<8;
		color_reg = color_reg | color;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg es impar
	for(i = 0; i < num_reg; i++)
	{
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	}
}

/*****************************************************************/
/*************** Cargar led si y led no zig zag ******************/
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i, j, paridad;
	int offset = 0;
	unsigned long int color_reg_impar = 0;
	unsigned long int color_reg_par = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.

	//Copiamos los 4 bits del color a lo largo de la variable color_reg, hasta llenar los 32 bits del registro
	color_reg_impar = color_reg_impar + color;
	color_reg_par = color_reg_par + color;
	color_reg_par = color_reg_par <<4;
	for(j=0;j<3;j++)
	{
		color_reg_impar = color_reg_impar <<8;
		color_reg_impar = color_reg_impar | color;
	}
	for(j=0;j<3;j++)
	{
		color_reg_par = color_reg_par <<4;
		color_reg_par = color_reg_par | color;
		color_reg_par = color_reg_par <<4;
	}
	// Solo va a salir bien si el numero de registros que se introduce por num_reg es impar
	for(i = 0; i < num_reg; i++)
	{
		paridad = i%2;
		if(paridad == 0)// el numero de registro es par
		{
			Xil_Out32((Base_ip_address + offset), color_reg_par);
			offset = offset + 0x4;
		}
		else// el numero de registro es impar
		{
			Xil_Out32((Base_ip_address + offset), color_reg_impar);
			offset = offset + 0x4;
		}
	}
}

/*****************************************************************************/
/*************** Iluminar nombre con los leds ************************/
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 3 espacios
	for(i=0;i<3;i++)
	{
		Xil_Out32((Base_ip_address + offset), 0x00000000);
		offset = offset + 0x4;
	}
	//Letra C
	color_reg = color_mask & C1;
```

```
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & C2_3_4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & C2_3_4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & C2_3_4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra A
color_reg = color_mask & A1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & A2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & A1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra R
color_reg = color_mask & R1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R2;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & R4;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra L
color_reg = color_mask & L1;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & L2_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Base_ip_address + offset), 0x00000000);
offset = offset + 0x4;
//Letra O
color_reg = color_mask & O1_3;
Xil_Out32((Base_ip_address + offset), color_reg);
offset = offset + 0x4;
```

```
	color_reg = color_mask & O2;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & O1_3;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Base_ip_address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra S
	color_reg = color_mask & S1;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S2;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S3;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S4;
	Xil_Out32((Base_ip_address + offset), color_reg);
	offset = offset + 0x4;
	//Ultimos 3 espacios
	for(i=0;i<3;i++)
	{
		Xil_Out32((Base_ip_address + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

/*****************************************************************************/
/************* Mover contenido de registros a la derecha ***************/
void Config_leds6(long unsigned int Base_ip_address, int num_reg)
{
	int i;
	int offset = 0;
	unsigned long int buffer = 0;
	unsigned long int buffer_posterior = 0;
	num_reg = num_reg - 1; // Se elige como limite "numreg-2" ya que el ultimo
registro se usará como registro de control
	//y no tendra INFO de color como el resto.
	//Y el anterior a ese se cargará fuera del bucle.
	//Leemos el primer dato
	buffer = Xil_In32(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR);
	// Solo va a salir bien si el numero de registros que se introduce por num_reg
es impar
	for(i = 0; i < num_reg; i++)
	{
		if(i == 31)
		{
			Xil_Out32(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR, buffer);
//Escribimos en la siguiente dirección de registro
		}
		else
		{
			offset = offset + 0x4;
			buffer_posterior = Xil_In32(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR
+ offset); //Leemos el siguiente dato para que no se pierda
			Xil_Out32(XPAR_LEDS_RGB_32X8_0_S00_AXI_BASEADDR + offset,
buffer); //Escribimos en la siguiente dirección de registro
			buffer = buffer_posterior;
```

```
			}
		}
}

/*****************************************************************************/
/************* Imprimir caracteres en el centro de los leds ***************/
/*Los caracteres se encenderán en un rectangulo centrado en la matriz, de 6x4 leds*/
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra A
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A1_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
```

```
	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra B
	color_reg = color_mask & B4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & B1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra C
```

```
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C2_3_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & C1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Letra D
	color_reg = color_mask & D4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & D1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
```

```
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 1
	color_reg = color_mask & uno_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & uno_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
```

```
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 2
	color_reg = color_mask & dos_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & dos_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 3
	color_reg = color_mask & tres_4;
```

```
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & tres_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 4
	color_reg = color_mask & cuatro_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cuatro_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
```

```
			offset = offset + 0x4;
		}
}

void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
		for(i=0;i<7;i++)
		{
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 5
		color_reg = color_mask & cinco_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cinco_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cinco_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cinco_1;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;

		//Ultimos 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}
}

void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
```

```
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 6
	color_reg = color_mask & seis_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 7
	color_reg = color_mask & siete_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
```

```
	color_reg = color_mask & siete_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_2;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 8
	color_reg = color_mask & ocho_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_1_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
```

```
	}
}

void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Numero 9
	color_reg = color_mask & nueve_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & nueve_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
```

```
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Numero 0
		color_reg = color_mask & cero_1_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cero_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cero_2_3;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & cero_1_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;

		//Ultimos 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}
}

void Dibujar_E(XDriver *Puntero_inst, int num_reg, short int color)
{
		int i;
		int offset = 0;
		unsigned long int color_reg = 0;
		unsigned long int color_mask = 0;


		num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
		color_mask = color_mask | color;
		for(i=0;i<7;i++)
		{
			color_mask = color_mask<<4;
			color_mask = color_mask | color;
		}
		//Primeros 14 espacios
		for(i=0;i<14;i++)
		{
			Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
			offset = offset + 0x4;
		}

		//Caracter asterisco
		color_reg = color_mask & E_4;
		Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
		offset = offset + 0x4;
		color_reg = color_mask & E_2_3;
```

```
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 14 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}

void Dibujar_F(XDriver *Puntero_inst, int num_reg, short int color)
{
	int i;
	int offset = 0;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;


	num_reg = num_reg - 1; // Se elige como limite "numreg-1" ya que el ultimo
registro se usará como registro de control y no tendra INFO de color como el resto.
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Primeros 13 espacios
	for(i=0;i<14;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}

	//Caracter almohadilla
	color_reg = color_mask & F_4;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_2_3;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & F_1;
	Xil_Out32((Puntero_inst->BaseAddress + offset), color_reg);
	offset = offset + 0x4;

	//Ultimos 12 espacios
	for(i=0;i<12;i++)
	{
		Xil_Out32((Puntero_inst->BaseAddress + offset), 0x00000000);
		offset = offset + 0x4;
	}
}
```

```
/****************************************************************************/
/*********************** Inicializacion del driver **************************/
int Driver_initialize(XDriver * InstancePtr, u16 DeviceId)
{
	XDriver_Config *ConfigPtr;

	/*
	 * Assert arguments
	 */
	Xil_AssertNonvoid(InstancePtr != NULL);

	/*
	 * Lookup configuration data in the device configuration table.
	 * Use this configuration info down below when initializing this
	 * driver.
	 */
	ConfigPtr = XDriver_LookupConfig(DeviceId);
	if (ConfigPtr == (XDriver_Config *) NULL) {
		InstancePtr->IsReady = 0;
		return (XST_DEVICE_NOT_FOUND);
	}

	return XDriver_CfgInitialize(InstancePtr, ConfigPtr,
				ConfigPtr->BaseAddress);
}

XDriver_Config *XDriver_LookupConfig(u16 DeviceId)
{
	XDriver_Config *CfgPtr = NULL;

	int Index;

	for (Index = 0; Index < XPAR_LEDS_RGB_32X8_NUM_INSTANCES; Index++) {
		if (XDriver_ConfigTable[Index].DeviceId == DeviceId) {
			CfgPtr = &XDriver_ConfigTable[Index];
			break;
		}
	}

	return CfgPtr;
}

int XDriver_CfgInitialize(XDriver * InstancePtr, XDriver_Config * Config,
			UINTPTR EffectiveAddr)
{
	/* Assert arguments */
	Xil_AssertNonvoid(InstancePtr != NULL);

	/* Set some default values. */
	InstancePtr->BaseAddress = EffectiveAddr;
	/*
	 * Indicate the instance is now ready to use, initialized without error
	 */
	InstancePtr->IsReady = XIL_COMPONENT_IS_READY;
	return (XST_SUCCESS);
}

void Dibujar_pantalla_de_inicio(XDriver *Puntero_inst, short int color)
```

```
{
	int i;
	int offset = 0;
	int Address;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;

	Address = Puntero_inst->BaseAddress;
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Ultimos 2 espacios
	for(i=0;i<2;i++)
	{
		Xil_Out32((Address + offset), 0x00000000);
		offset = offset + 0x4;
	}
	//Letra A
	color_reg = color_mask & A_corta_1_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A_corta_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A_corta_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & A_corta_1_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio x2
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//" : "
	color_reg = color_mask & dos_puntos;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio x2
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra E
	color_reg = color_mask & E_corta_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_corta_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & E_corta_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra S
```

```
	color_reg = color_mask & S_corta_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S_corta_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & S_corta_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra L
	color_reg = color_mask & L_corta_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & L_corta_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & L_corta_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra U
	color_reg = color_mask & U_corta_1_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & U_corta_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & U_corta_1_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra P
	color_reg = color_mask & P_corta_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & P_corta_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & P_corta_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Ultimos 2 espacios
	for(i=0;i<2;i++)
	{
		Xil_Out32((Address + offset), 0x00000000);
		offset = offset + 0x4;
	}
}
//----------------------------------------------------------------------------//
void Mostrar_puntuacion(XDriver *Puntero_inst, short int color, int puntuacion)
{
	int i, j;
	int offset = 0;
```

```
	int Address;
	unsigned long int color_reg = 0;
	unsigned long int color_mask = 0;
	int centenas = 0;
	int decenas = 0;
	int unidades = 0;
	char puntuacion_separada[3] = {};//MSB serán las centenas y LSB seran las
unidades.

	Address = Puntero_inst->BaseAddress;
	color_mask = color_mask | color;
	for(i=0;i<7;i++)
	{
		color_mask = color_mask<<4;
		color_mask = color_mask | color;
	}
	//Aqui vamos a pasar el valor de la puntuación lo haremos desplazando la
variable puntuación.
	unidades = puntuacion%10;
	puntuacion = puntuacion / 10;
	decenas = puntuacion%10;
	puntuacion = puntuacion / 10;
	centenas = puntuacion;

	puntuacion_separada[0] = '0' + unidades;
	puntuacion_separada[1] = '0' + decenas;
	puntuacion_separada[2] = '0' + centenas;

	//Ultimo espacio
		Xil_Out32((Address + offset), 0x00000000);
		offset = offset + 0x4;
	//Ultimo numero (unidades)
	for(j = 0; j<3; j++)
	{
		switch(puntuacion_separada[j])
		{
			case '0':
				color_reg = color_mask & cero_1_4;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				color_reg = color_mask & cero_2_3;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				color_reg = color_mask & cero_2_3;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				color_reg = color_mask & cero_1_4;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				//Espacio
				Xil_Out32((Address + offset), 0x00000000);
				offset = offset + 0x4;
				break;
			case '1':
				color_reg = color_mask & uno_4;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				color_reg = color_mask & uno_3;
				Xil_Out32((Address + offset), color_reg);
				offset = offset + 0x4;
				color_reg = color_mask & uno_2;
```

```
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & uno_1;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            //Espacio
            Xil_Out32((Address + offset), 0x00000000);
            offset = offset + 0x4;
            break;
        case '2':
            color_reg = color_mask & dos_4;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & dos_3;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & dos_2;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & dos_1;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            //Espacio
            Xil_Out32((Address + offset), 0x00000000);
            offset = offset + 0x4;
            break;
        case '3':
            color_reg = color_mask & tres_4;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & tres_3;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & tres_2;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & tres_1;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            //Espacio
            Xil_Out32((Address + offset), 0x00000000);
            offset = offset + 0x4;
            break;
        case '4':
            color_reg = color_mask & cuatro_4;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & cuatro_3;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & cuatro_2;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            color_reg = color_mask & cuatro_1;
            Xil_Out32((Address + offset), color_reg);
            offset = offset + 0x4;
            //Espacio
            Xil_Out32((Address + offset), 0x00000000);
            offset = offset + 0x4;
            break;
```

```
case '5':
	color_reg = color_mask & cinco_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & cinco_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	break;
case '6':
	color_reg = color_mask & seis_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & seis_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	break;
case '7':
	color_reg = color_mask & siete_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & siete_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	break;
case '8':
	color_reg = color_mask & ocho_1_4;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & ocho_2_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
```

```
                        color_reg = color_mask & ocho_1_4;
                        Xil_Out32((Address + offset), color_reg);
                        offset = offset + 0x4;
                        //Espacio
                        Xil_Out32((Address + offset), 0x00000000);
                        offset = offset + 0x4;
                        break;
                case '9':
                        color_reg = color_mask & nueve_4;
                        Xil_Out32((Address + offset), color_reg);
                        offset = offset + 0x4;
                        color_reg = color_mask & nueve_2_3;
                        Xil_Out32((Address + offset), color_reg);
                        offset = offset + 0x4;
                        color_reg = color_mask & nueve_2_3;
                        Xil_Out32((Address + offset), color_reg);
                        offset = offset + 0x4;
                        color_reg = color_mask & nueve_1;
                        Xil_Out32((Address + offset), color_reg);
                        offset = offset + 0x4;
                        //Espacio
                        Xil_Out32((Address + offset), 0x00000000);
                        offset = offset + 0x4;
                        break;
                default:
                        break;
        }
}
//Espacio
Xil_Out32((Address + offset), 0x00000000);
offset = offset + 0x4;
//" : "
color_reg = color_mask & dos_puntos;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
//Espacio x 2
Xil_Out32((Address + offset), 0x00000000);
offset = offset + 0x4;
Xil_Out32((Address + offset), 0x00000000);
offset = offset + 0x4;
//Letra S
color_reg = color_mask & S_larga_3;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S_larga_2;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & S_larga_1;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
//Espacio
Xil_Out32((Address + offset), 0x00000000);
offset = offset + 0x4;
//Letra T
color_reg = color_mask & T_larga_1_3;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
color_reg = color_mask & T_larga_2;
Xil_Out32((Address + offset), color_reg);
offset = offset + 0x4;
```

```
	color_reg = color_mask & T_larga_1_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Espacio
	Xil_Out32((Address + offset), 0x00000000);
	offset = offset + 0x4;
	//Letra P
	color_reg = color_mask & P_larga_3;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & P_larga_2;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	color_reg = color_mask & P_larga_1;
	Xil_Out32((Address + offset), color_reg);
	offset = offset + 0x4;
	//Ultimos 2 espacios
	for(i=0;i<2;i++)
	{
		Xil_Out32((Address + offset), 0x00000000);
		offset = offset + 0x4;
	}
}
```

A.5.2 Librería creada (version del juego): driver_ip.h

Solo se incluirán aquí aquellas funciones que se han usado en el diseño final del juego, las versiones v1 y v2 contienen el resto de funciones que se han creado durante el desarrollo del proyecto.

```
//includes
#include "xil_io.h"

/************************** Type Definitions ***************************/

/**
 * This typedef contains configuration information for the device.
 */
typedef struct {
	u16 DeviceId;		/**< Unique ID  of device */
	UINTPTR BaseAddress;	/**< Device base address */
} XDriver_Config;

/**
 * The XGpio driver instance data. The user is required to allocate a
 * variable of this type for every GPIO device in the system. A pointer
 * to a variable of this type is then passed to the driver API functions.
 */
typedef struct {
	UINTPTR BaseAddress;	/**< Device base address */
	u32 IsReady;	/**< Device is initialized and ready */
} XDriver;
/************************************************************************/

//Functions

//Initialize functions

int Driver_initialize(XDriver *InstancePtr, u16 DeviceId);
XDriver_Config *XDriver_LookupConfig(u16 DeviceId);
```

```
int XDriver_CfgInitialize(XDriver *InstancePtr, XDriver_Config * Config, UINTPTR
EffectiveAddr);

// Estas funciones solo van a colocar los valores de los leds en las filas
correspondientes.
void Apagar_leds(XDriver *Puntero_inst, int num_reg);
void Leer_registros(long unsigned int Base_ip_address, int num_reg);
void Cargar_led(long unsigned int Base_ip_address, int num_reg, int num_led, unsigned
long int color);
void Activar_init(long unsigned int Base_ip_address, int num_reg);
void Activar_reset(long unsigned int Base_ip_address, int num_reg);
void Desactivar_init(long unsigned int Base_ip_address, int num_reg);
void Desactivar_reset(long unsigned int Base_ip_address, int num_reg);
void Config_leds1(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds2(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds3(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds4(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds5(long unsigned int Base_ip_address, int num_reg, short int color);
void Config_leds6(long unsigned int Base_ip_address, int num_reg);
void Dibujar_A(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_B(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_C(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_D(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_1(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_2(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_3(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_4(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_5(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_6(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_7(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_8(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_9(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_0(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_E(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_F(XDriver *Puntero_inst, int num_reg, short int color);
void Dibujar_pantalla_de_inicio(XDriver *Puntero_inst, short int color);
void Mostrar_puntuacion(XDriver *Puntero_inst, short int color, int puntuacion);

//Mascaras para los leds:

//Estas máscaras se han definido con MSB como el led 8 (izquierda) y LSB el led 1
(derecha)

#define led_8 0xF0000000 //Led 8
#define led_7 0x0F000000 //Led 7
#define led_6 0x00F00000 //Led 6
#define led_5 0x000F0000 //Led 5
#define led_4 0x0000F000 //Led 4
#define led_3 0x00000F00 //Led 3
#define led_2 0x000000F0 //Led 2
#define led_1 0x0000000F //Led 1

//Mascaras inversas
#define notled_8 0x0FFFFFFF //Led 8
#define notled_7 0xF0FFFFFF //Led 7
#define notled_6 0xFF0FFFFF //Led 6
#define notled_5 0xFFF0FFFF //Led 5
#define notled_4 0xFFFF0FFF //Led 4
#define notled_3 0xFFFFF0FF //Led 3
#define notled_2 0xFFFFFF0F //Led 2
```

```
#define notled_1 0xFFFFFFF0 //Led 1

//Mascaras registro de control
#define init_mask_on      0x00000002
#define init_mask_off   0xFFFFFFFD
#define reset_mask_on     0xFFFFFFFE
#define reset_mask_off  0x00000001

//Mascaras para las letras
#define A1_3 0x0FFFFFF0
#define A2 0x0F00F000

#define B1 0x0FFFFFF0
#define B2_3 0x0F0FF0F0
#define B4 0x0FF00FF0

#define C1 0x0FFFFFF0
#define C2_3_4 0x0F0000F0

#define D1 0x0FFFFFF0
#define D2 0x0F0000F0
#define D3 0x00F00F00
#define D4 0x000FF000

#define R1 0x0FFFFFF0
#define R2 0x0F0FF000
#define R3 0x0F0F0F00
#define R4 0x0FFF00F0

#define L1 0x0FFFFFF0
#define L2_3 0x000000F0

#define O1_3 0x0FFFFFF0
#define O2 0x0F0000F0

#define S1 0x0FFF00F0
#define S2 0x0F0F00F0
#define S3 0x0F00F0F0
#define S4 0x0F00FFF0

#define uno_1 0x000F00F0
#define uno_2 0x00F000F0
#define uno_3 0x0FFFFFF0
#define uno_4 0x000000F0

#define dos_1 0x00F00FF0
#define dos_2 0x0F000FF0
#define dos_3 0x0F00F0F0
#define dos_4 0x00FF00F0

#define tres_1 0x00F00F00
#define tres_2 0x0F0000F0
#define tres_3 0x0F0F00F0
#define tres_4 0x00F0FF00

#define cuatro_1 0x000FFF00
#define cuatro_2 0x00F00F00
#define cuatro_3 0x0FFFFFF0
#define cuatro_4 0x00000F00

#define cinco_1 0x0FFF00F0
```

```
#define cinco_2_3 0x0F0F00F0
#define cinco_4 0x0F00FF00

#define seis_1 0x0FFFFFF0
#define seis_2_3 0x0F0F00F0
#define seis_4 0x0F0FFFF0

#define siete_1 0x0F000000
#define siete_2 0x0F00FFF0
#define siete_3 0x0F0F0000
#define siete_4 0x0FF00000

#define ocho_1_4 0x0FF00FF0
#define ocho_2_3 0x0F0FF0F0

#define nueve_1 0x0FFF0F00
#define nueve_2_3 0x0F0F00F0
#define nueve_4 0x0FFFFF00

#define cero_1_4 0x00FFFF00
#define cero_2_3 0x0F0000F0

#define E_1 0x0FFFFFF0
#define E_2_3 0x0F0FF0F0
#define E_4 0x0F0000F0

#define F_1 0x0FFFFFF0
#define F_2_3 0x0F0F0000
#define F_4 0x0F000000

#define A_corta_1_4 0x00FFFFF0
#define A_corta_2_3 0x00F0F000

#define dos_puntos 0x000F0F00

#define E_corta_3 0x00F000F0
#define E_corta_2 0x00F0F0F0
#define E_corta_1 0x00FFFFF0

#define S_corta_3 0x00F0FFF0
#define S_corta_2 0x00F0F0F0
#define S_corta_1 0x00FFF0F0

#define L_corta_3 0x000000F0
#define L_corta_2 0x000000F0
#define L_corta_1 0x00FFFFF0

#define U_corta_1_3 0x00FFFFF0
#define U_corta_2 0x000000F0

#define P_corta_3 0x00FFF000
#define P_corta_2 0x00F0F000
#define P_corta_1 0x00FFFFF0

#define P_larga_3 0x0FFF0000
#define P_larga_2 0x0F0F0000
#define P_larga_1 0x0FFFFFF0

#define T_larga_1_3 0x0F000000
#define T_larga_2 0x0FFFFFF0
```

```
#define S_larga_3 0x0F0FFFF0
#define S_larga_2 0x0F0FF0F0
#define S_larga_1 0x0FFFF0F0



// Colores

#define apagar 0b0000 //0x0
#define rojo 0b0001 // 0x1
#define verde 0b0010 // 0x2
#define azul 0b0011 // 0x3
#define magenta 0b0100 // o 0x4
#define amarillo 0b0101 // o 0x5
#define cyan 0b0110 // o 0x6
#define gris 0b0100 // o 0x4
#define blanco 0b1000 // o 0x8
```