BENEMERITA UNIVERSIDAD AUTONOMA DE PUEBLA
Facultad de Ciencias de la Electrónica
Reporte de la Práctica n°4: Controlador de una Matriz del led por puerto serial a través de FPGA..
Asignatura: Sistemas Empotrados
Catedrático: Carlos García Lucero
Juan Hadad Aguilar Romero e-mail: juanelo@gmail.com
Abel Alejandro Rubín Alvarado e-mail: arubinbw@hotmail.com
Idvard Francisco Martínez Lugao e-mail: idvgao@hotmail.com
Oscar Cuapio López e-mail: cuapio@hotmail.com
Eduardo Pastor Torres e-mail: edypastor@gmail.com

Resumen— Para la elaboración de esta práctica se hizo uso de cinco Softwares (Xilinx EDK, Xilinx SDK, MicroBlazeINTESC, INTEGRA y LabView) los cuales se requirieren para programar, diseñar y crear el punto bit. La tarjeta que se usó es un dispositivo lógico programable (FPGA) en el cual se mostraba en alguna de sus salidas, como una Matriz de led de 8x8, que cambia de imagen de acuerdo a lo enviado desde labview con palabras de 32bits para crear un video, esto a través de una interfaz gráfica que se realizó el software LabView.
.

Palabras clave— *FPGA, LED , Labview Programación..*

INTRODUCCION

La finalidad en la elaboración de la práctica número tres, es conocer (además de saber), la estructura y manera en que operan los Softwares de LabView y Xilinx (XPS y XSDK), desde la creación de la arquitectura y asignación de puertos, así como la programación en lenguaje C, e incluso, la asignación de valores en registros reservados.

Posteriormente se cargaron los archivos adecuados en el software MicroBlazeINTESC para la creación del punto bit, éste se cargó en el programa INTEGRA, para así programar la tarjeta FPGA de nombre (AVANXE) diseñada por la empresa INTESC. Una vez programada la FPGA, a través de una interfaz gráfica con el software LabView, se controla el envió de las imágenes con su respectivo tiempo para así cambiar las imágenes a una velocidad de 100ms para que esta misma se vea como video.

En este trabajo se dará a conocer el código del programa elaborado en lenguaje C y los resultados que se observaron en el LED RGB conectado a una de las salidas de la tarjeta FPGA.

I. Diodo LED

El LED es un tipo especial de diodo, que trabaja como un diodo común, pero que al ser atravesado por la corriente eléctrica, emite luz. Existen diodos LED de varios colores que dependen del material con el cual fueron construidos. Hay de color rojo, verde, amarillo, ámbar, infrarrojo, entre otros.

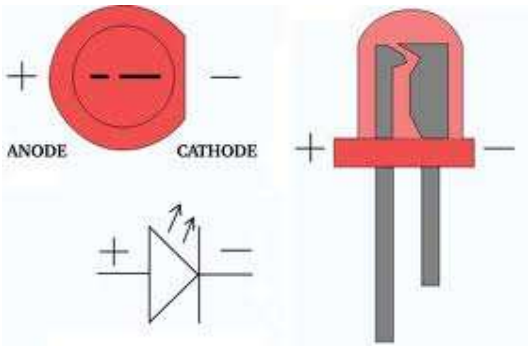

Figura 1. Diodo tipo LED

II. MATRIZ DE LEDS

Una matriz de led es un arreglo de leds por columna, debido a que si se conectara la matriz y tuviera un habilitador por LED se perdería mucho tanto en potencia como en recursos.

La técnica que se utiliza para realizar estos carteles es el multiplexado ya que es el método que nos permite encender mayor cantidad de leds con una cantidad de pines

del microcontrolador razonable, ante todo esto hay que decir que según sea el tamaño del cartel que se quiera construir (tanto en números de letras como en tamaño de las mismas) va a depender de que tan sofisticado debe ser el microcontrolador (PIC) a utilizar, esto a su vez va a incidir en el costo monetario claro está.

Figura2. Matriz de LEDs 8x8.

La matriz está compuesta por una serie de filas y columnas la intersección entre ambas contiene un led, para que este encienda, tiene que recibir simultáneamente un 0 en la fila y un 1 en la columna, cuando se da esta condición la electrónica del circuito se encarga de encender el led correspondiente.
La forma de generar un mensaje sobre el cartel es relativamente más sencilla si procedemos a aplicar el siguiente algoritmo:
1) Apagar todas las filas
2) Escribir los valores correspondientes a la primer fila en el que el registro de desplazamiento, teniendo en cuenta que el primer digito binario colocado corresponde al último en poner al de la primer columna.
3) Encender la primera fila, esperar un tiempo y volver a apagarla.
4) Repetir los pasos 2 y 3 para las filas restantes.
El tiempo de la demora debe ser tal que permita una visualización correcta, sin molestos parpadeos y con los leds brillantes. Hay que tener en cuenta que si utilizamos tiempos mayores para el encendido de cada fila, el brillo de los leds será mayor, pero también aumentara el parpadeo.

La forma de transformar este algoritmo en un programa funcional depende de cada programador, y puede ser más o menos complejo según se permitan diferentes tipos de caracteres, animaciones etc.
Un punto a tener en cuenta es el brillo del led por lo que la intensidad del mismo dependerá de que tipo de led se utilice. En caso de un típico cartel de 7 filas, a pesar de que las veremos encendidas al mismo tiempo, cada led estará solamente encendido la séptima parte del tiempo, por lo que su brillo será siete veces inferior al normal, y nuestro cartel apenas será visible.
Por suerte esto tiene solución, dados que los tiempos que permanecerá encendido cada led no superara los 20 o 30 milisegundos, podremos hacerles circular una corriente mayor a la nominal sin que lleguen a dañarse, con lo que brillaran mucho más intensamente, dando como resultado un cartel perfectamente visible.

III. Desarrollo

La matriz está compuesta por una serie de filas y Primero se realizó la configuración inicial en el software XPS (nombre de la carpeta, nomenclatura de la FPGA, arquitectura, asignación de puertos y exportación del diseño) a partir de la arquitectura creada que se utilizó en la práctica número uno y dos; lo que se agregó fue el corrimiento y estado de los registros para poder prender cada columna de la matriz de leds.

Después se realizó el diseño del programa en código C en el software XSDK (creación de carpeta, selección de código, incluir las librerías adecuadas que se usaran y que han sido previamente exportadas con el anterior programa y por último creación y compilación del proyecto). El código de la práctica se podrá observar en la parte “Código de programa”.

3.1 Parámetros a considerar

Se hace el análisis de la documentación existente en el EDK de Xilinx, para ello recurrimos al documento referente de configuración de hardware, éste documento es llamado “xps_timer.pdf” y es

localizado en la ruta C:\Xilinx\14.6\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\xps_timer_v1_01_b\doc

3.1.1 Control PWM

En modo PWM, se usan dos timers para producir una salida tipo PWM con una cierta frecuencia y ciclo de trabajo, el TIMER0 configura periodo y TIMER1 configura el tiempo en alto para el PMW (Ciclo de trabajo).

3.1.2 Configuración del ciclo de trabajo y periodo

Acorde a la documentación de Xilinx, tenemos que hacer los cálculos de valores para el periodo y ciclo de trabajo, de ello tenemos las siguientes ecuaciones:

a) Cuando se configura contador ascendente (UDT = '0'):

PWM_PERIODO = (MAX_COUNT - TLR0 + 2) x PLB_CLOCK_PERIOD
PWM_Tiempo_alto = (MAX_COUNT - TLR1 + 2) x PLB_CLOCK_PERIOD

b) Cuando se configura contador descendente (UDT = '1'):

PWM_PERIODO = (TLR0 + 2) x PLB_CLOCK_PERIOD
PWM_Tiempo_alto = (TLR1 + 2) x PLB_CLOCK_PERIOD

¡NOTA IMPORTANTE!: Para esta práctica hemos optado por el conteo descendente del contador debido al poco conocimiento de ciertos parámetros, sin embargo, se hicieron unas pruebas dentro de laboratorio para calcular otros, uno de ellos fue el periodo de reloj PLB, mismo que se manejó "prueba y error del valor", al final se concluye que la frecuencia del reloj PLB es 66.66MHZ, como referencia de aproximación tenemos tano los datos capturados en el osciloscopio (frecuencia final del PWM) como un foro de ayuda que habla de este tema (consultar referencias).

3.1.3 Escritura en los registros

Para hacer la configuración del PWM, debemos hacer la escritura de modo explícito en los registros del módulo, para ello debemos consultar dos registros esenciales.

TCSR_x: O registro de control. Dentro de este registro se almacena los bits de control del módulo, esto es, si queremos "que funcione como timer", "como PWM", "como interrupción", entre otros, además de otros valores, por ejemplo: para el PWM debemos configurar si el contador será de tipo ascendente o descendente, si debe limpiar algunos registros de memoria/control, etc. Cabe destacar que dentro de este registro, tenemos los bits de 0 a 20 como bits reservados (funciones desconocidas por el usuario), el resto pertenece a nuestra configuración.

Para nuestro caso, manejaremos los siguientes bits de control:

Bit 21: 1	Habilita todos los timers
Bit 22: 1	Habilita PWM
Bit 23: 0	Interrupción del Timer0 no cambia, cambio manual
Bit 24: 0	Deshabilita Timer0
Bit 25: 0	Deshabilita señal de interrupción
Bit 26: 0	Sin carga al Timer0
Bit 27: 0	Mantiene valor de Timer0
Bit 28: 0	Deshabilita interrupciones externas
Bit 29: 1	Habilita señal externa (PWM)
Bit 30: 1	Contador de Timer descendiente
Bit 31: 0	Modo de Timer es generación

El valor hexadecimal a manejar es 0X00000606.

Figura 3. Distribución de los bits de control para el registro TCSR_x

Dentro de estos registros se hace el almacenamiento de información referente al ciclo de trabajo (TLR1) y periodo (TLR0), para un módulo con configuración de registro de tamaño x, se hace el llenado del registro con x bits, en nuestro caso usamos los 32 bits.

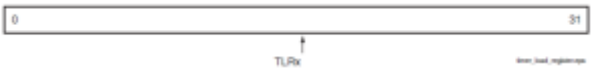

Figura 4. Registro TLR_x para almacenar datos

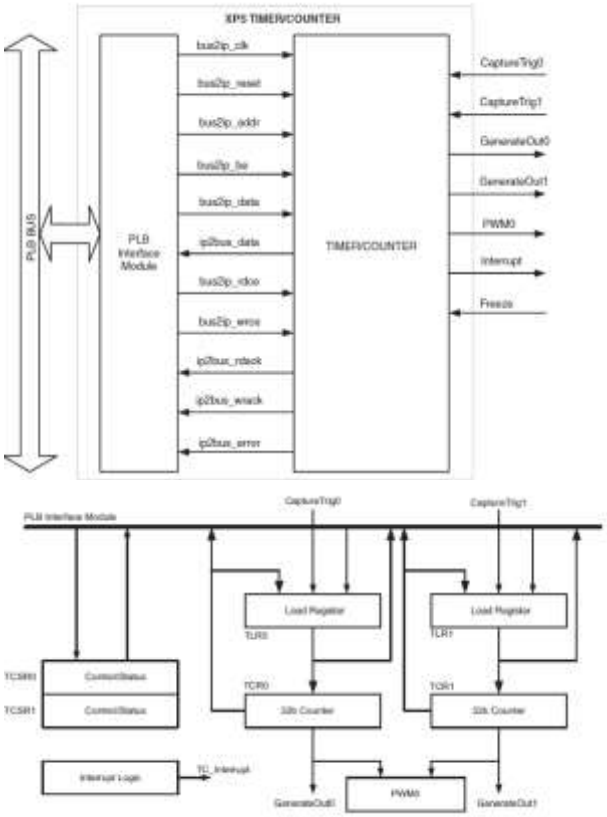

Figura 5. Módulo que caracteriza el TIMER

IV. Arquitectura del módulo timer

Básicamente tenemos un módulo que hace una comunicación con el bus de datos, éste emite la información que requieres los diversos registros de entrada/salida para configuración de timers, a continuación se presenta un diagrama referente a su comportamiento, vemos que posee diversas configuración, para nuestra práctica optamos por el desarrollo tipo PWM

Una vez diseñada la arquitectura, la asignación de puertos, y el software en código C, se procedió a cargar los puntos .bmm (carpeta con el diseño de la arquitectura importada por el XPS), .bit (carpeta creada con el nuevo proyecto por el XSDK), .elf (carpeta creada por el software XSDK). Con todos los archivos anteriores cargados, con el software MicroBlazeINTESC, se creó el .bit para así programar la tarjeta FPGA.
Con el .bit creado con el anterior programa se cargó en el software INTEGRA, para que una vez conectada la tarjeta AVANXE, se programe con ese .bit creado anteriormente.

Para finalizar, con el software LabView, se realizó la interfaz gráfica, con la cual el usuario controló el cambio del color del LED RGB conectado a la salida de la tarjeta AVANXE. En la revisión de los resultados, se pudo observar como utilizando el cuadro de Color Box en la interfaz de LabView (imagen 1) , se podía cambiar el color del LED, esto enviándole diferentes anchos de pulso para controlar la intensidad de voltaje.

V. Matriz_leds(vhdl)

En matriz de led se agregó:

```
Library
plbv46_slave_single_v1_01_a;
use
plbv46_slave_single_v1_01_a.plbv46
_slave_single;

library matriz_leds_v1_00_a;

use
matriz_leds_v1_00_a.user_logic;
```

Sirve para declarar las librerías que harán conexión con el user logic

```
--    C_BASEADDR                    -
- PLBv46 slave: base address
--    C_HIGHADDR                    -
- PLBv46 slave: high address
```

Estos son para las columnas y renglones

```
C_BASEADDR       : std_logic_vector     := X"FFFFFFFF";
C_HIGHADDR       : std_logic_vector     := X"00000000";
C_SPLB_AWIDTH    : integer               := 32;
C_SPLB_DWIDTH    : integer              :=128;
C_SPLB_NUM_MASTERS  : integer              := 8;
C_SPLB_MID_WIDTH   : integer              := 3;
C_SPLB_NATIVE_DWIDTH      : integer              := 32;
C_SPLB_P2P       : integer              := 0;
C_SPLB_SUPPORT_BURSTS     : integer              := 0;
C_SPLB_SMALLEST_MASTER  : integer              := 32;
C_SPLB_CLK_PERIOD_PS   : integer              := 10000;
C_INCLUDE_DPHASE_TIMER     : integer              := 1;
C_FAMILY            : string               := "virtex6"
```

Los baseaddres son los encargados del estado, AWIDTH sirve para el tamaño dela palabra, DWIDTH para la memoria , NUM_MASTERS es para el tamaño del dato, MID_WIDTH es para la cantidad de estados, CLK_PERIOD_PS este es para usar el divisor de frecuencia y nos sirve para poder saber el tiempo que se debe de poner para que se visible nuestro dato y por ultimo FAMILY es para conocer la familia del controlador que se está ocupando.

VI. User logic(VHDL)

En este apartado se agrego:

```
if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
    if Bus2IP_Reset = '1' then
     slv_reg0 <= (others => '0');
     slv_reg1 <= (others => '0');
     slv_reg2 <= (others => '0');
    else
case slv_reg_write_sel is
      when "100" =>
       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
        if ( Bus2IP_BE(byte_index) = '1' ) then
         slv_reg0(byte_index*8      to      byte_index*8+7)      <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
        end if;
       end loop;
      when "010" =>
       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
        if ( Bus2IP_BE(byte_index) = '1' ) then
         slv_reg1(byte_index*8      to      byte_index*8+7)      <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
        end if;
       end loop;
      when "001" =>
       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
        if ( Bus2IP_BE(byte_index) = '1' ) then
         slv_reg2(byte_index*8      to      byte_index*8+7)      <=
Bus2IP_Data(byte_index*8 to byte_index*8+7);
        end if;
       end loop;
      when others => null;
     end case;
    end if;
  end if;

  case slv_reg_read_sel is

when "100" => slv_ip2bus_data <= slv_reg0;
 when "010" => slv_ip2bus_data <= slv_reg1;
when "001" => slv_ip2bus_data <= slv_reg2;
 when others => slv_ip2bus_data <= (others => '0');
title('Codificación Bipolar RZ')
```

Que son tres registros los cuales los dos primeros son para cada uno de los registros de estado y el tercero solamente se puso por si se llegaba a necesitar para agregar un extra.

VII. Código de Programa

Con ayuda de un compañero de la facultad, Noé Mendoza Morales, hemos logrado el manejo adecuado del PWM para nuestro propósito.
Finalmente tenemos el siguiente algoritmo que se ha trabajado desde la práctica uno y actualmente lo estamos adaptando según los requisitos de práctica.

```
if Bus2IP_Clk'event#include<xparameters.h>
#include<xgpio.h>
#include<xintc.h>
#include<xil_exception.h>
#include<xuartlite.h>
#include <xtmrctr_l.h>
#include
"C:\Repositorios_empotrados\MyProcessorIPLib/drivers/display_7seg
_v1_00_a/src/display_7seg.h"
#include
"C:\Repositorios_empotrados\MyProcessorIPLib/drivers/display_7seg
_v1_00_a/src/display_7seg.c"
#include
"C:\Repositorios_empotrados\MyProcessorIPLib\drivers\matriz_leds
_v1_00_a\src/matriz_leds.c"
#include
"C:\Repositorios_empotrados\MyProcessorIPLib\drivers\matriz_leds
_v1_00_a\src/matriz_leds.h"


XIntc INT;
XUartLite UART;

u8 BUFF[22], RGB[6];
u8 BUFF1[22], RGB1[6],RGB0[6];
u32 reg1,reg0;
XGpio sw, led;
int lec,estado;

void funcion_irq(void *callback){
                u32 i;
                u8 V,j = 0;
  XUartLite_DisableInterrupt(&UART);
  for(i=0;i<0x2BFFC;i++){
  }
  XUartLite_Recv(&UART, BUFF, 22);

//String-Decimal
  for (V = 0 ; V <= 30 ; V++)
  {
    if(BUFF[V] == 0x20){
      V++;
      j++;
    }
    RGB[j] = (BUFF[V] & 0x0F)  + RGB[j]*10;

  }
                reg1=RGB[0];
                reg1=reg1 << 8;
                reg1=reg1+RGB[1];
                reg1=reg1 << 8;
                reg1=reg1+RGB[2];
                reg1=reg1 << 8;
                reg1=reg1+RGB[3];
                if (estado==1)
                {
MATRIZ_LEDS_mWriteSlaveReg1(0xC2400000,1,reg1);
                        estado=0;
                }
                else
                {

MATRIZ_LEDS_mWriteSlaveReg0(0xC2400000,1,reg1);
                        estado=1;
    }
    RGB_PWM();


  lec=XGpio_DiscreteRead(&sw,1);
  XGpio_DiscreteWrite(&led,1,RGB[lec]);

  RGB[0]=RGB[1]=RGB[2]=RGB[3]=RGB[4]=0x00;
  XUartLite_ResetFifos(&UART);
  XUartLite_EnableInterrupt(&UART);
}

void main()
{
 XUartLite_Initialize(&UART, XPAR_UARTLITE_1_DEVICE_ID);
XIntc_Initialize(&INT, XPAR_INTC_0_UARTLITE_1_VEC_ID);
 XIntc_Connect(&INT,
XPAR_INTC_0_UARTLITE_1_VEC_ID,(Xil_ExceptionHandler)funcion_ir
q, &UART);
XGpio_Initialize(&led , XPAR_LEDS_DEVICE_ID);
XGpio_Initialize(&sw , XPAR_DIP_SWITCHES_DEVICE_ID);
XUartLite_EnableInterrupt(&UART);
  XIntc_Enable(&INT, XPAR_INTC_0_UARTLITE_1_VEC_ID);
microblaze_enable_interrupts();

//pwm COLORES(rgb)

XTmrCtr_WriteReg(XPAR_TMRCTR_0_BASEADDR,0,XTC_TLR_OFFSET
,0X0002FDEE);    // Configuración del ciclo de trabajo.

XTmrCtr_WriteReg(XPAR_TMRCTR_0_BASEADDR,1,XTC_TLR_OFFSET
,0xFFFFFFFF);

XTmrCtr_WriteReg(XPAR_TMRCTR_0_BASEADDR,0,XTC_TCSR_OFFSE
T,0X00000606);    // Se pone a '0' MDT1; ENALL, GENT1 Y PWMB0 A
'1';UDT1 Contador desc.

XTmrCtr_WriteReg(XPAR_TMRCTR_0_BASEADDR,1,XTC_TCSR_OFFSE
T,0X00000606);


XTmrCtr_WriteReg(XPAR_TMRCTR_1_BASEADDR,0,XTC_TLR_OFFSET
,0X0002FDEE);

XTmrCtr_WriteReg(XPAR_TMRCTR_1_BASEADDR,1,XTC_TLR_OFFSET
,0xFFFFFFFF);

XTmrCtr_WriteReg(XPAR_TMRCTR_1_BASEADDR,0,XTC_TCSR_OFFSE
T,0X00000606);

XTmrCtr_WriteReg(XPAR_TMRCTR_1_BASEADDR,1,XTC_TCSR_OFFSE
T,0X00000606);


XTmrCtr_WriteReg(XPAR_TMRCTR_2_BASEADDR,0,XTC_TLR_OFFSET
,0X0002FDEE);

XTmrCtr_WriteReg(XPAR_TMRCTR_2_BASEADDR,1,XTC_TLR_OFFSET
,0xffffffff);

XTmrCtr_WriteReg(XPAR_TMRCTR_2_BASEADDR,0,XTC_TCSR_OFFSE
T,0X00000606);

XTmrCtr_WriteReg(XPAR_TMRCTR_2_BASEADDR,1,XTC_TCSR_OFFSE
T,0X00000606);
                estado=0;

MATRIZ_LEDS_mWriteSlaveReg0(0xC2400000,1,0xFFFFFFFF);//(0xC24
00000,1,reg1);
```

```
  while(1){


  }

}
void RGB_PWM(){
  XTmrCtr_WriteReg(XPAR_TMRCTR_0_BASEADDR,1,
XTC_TLR_OFFSET, RGB[2]*100);
  XTmrCtr_WriteReg(XPAR_TMRCTR_1_BASEADDR,1,
XTC_TLR_OFFSET, RGB[0]*100);
  XTmrCtr_WriteReg(XPAR_TMRCTR_2_BASEADDR,1,
XTC_TLR_OFFSET, RGB[1]*100);
```

VIII. Explicación código programa

En la primera parte se declaran las librerías a utilizar, en las cuales incluyen el Gpio, y el UART RS232. Dentro del primer void (para la función de la interrupción) al inicio se convierte la cadena de caracteres, recibida a través del UART, en valores decimales u8 y se obtiene su complemento; después se realiza la conversión de los valores String a decimal. Después se obtiene el complemento de los decimales, se habilitan los Fifos.

En el void main se inicializa el Uart, se habilita la conexión en C por software, se habilita la interrupción del Uart y se inician las interrupciones del Xintc.

Después se inicia con cada uno de los tres PWM para cada uno de los colores (Rojo, Verde y Azul), para lo cual se inicia con la configuración del ciclo de trabajo, para después poner a '0' MDT1; ENALL, GENT1 Y PWMB0 A '1';UDT1 Contador descendiente; esto se realizó para los tres PWM implementados en el programa

Al final en el void RGB_PWM se multiplica por 100 para que el ciclo de trabajo sea desde cero hasta el número multiplicado. Por lo tanto es el mismo programa de la mini practica solo se anexo lo que se encuentra en color rojo y esto sirve si estado es igual a cero entonces prende las primeras 4 columnas y si estado es igual a uno encienden las últimas cuatro columnas para poder ocupar los 64 en dos palabras de 32 bits y estas divididas en cuatro y por tanto quedan de 8 bits,y el reg lo que hace es un corrimiento de los 8 bits sigue la próxima columna y lee los 8 bits y por la velocidad mayor a 60hz pinta en la matriz el dibujo en los 64 bits, y con los 100ms que se pusieron para mandar a pintar 10 imágenes por segundo lo que hace que sea posible ver la animación.

Y la matriz de led mWriteSlave es en donde hace referencia a la escritura dependiendo del estado del reg.

IX. Interfaz en LabView

Figura 6. Esquema grafico de Labview.

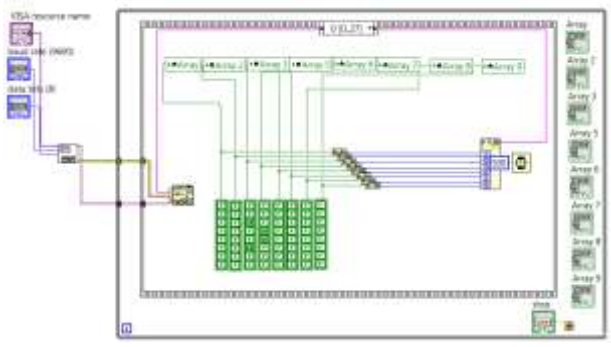

Figura 7. Esquemático de Labview.

Después se inicia con cada uno de los tres PWM para cada uno de los colores (Rojo, Verde y Azul), Para la interfaz en LabView lo que se hizo fue insertar las secuencias de imágenes para poder crear el mini video, como se observa en las figuras tiene la velocidad de sincronización de 9600

baudios, la entrada de 8 bits y el visa resource name es para escoger el puerto usb para realizar la conexión rs232 por medio del uart.

Dentro del while se observa la matriz sonde se habilita o deshabilitan los leds con datos booleanos de cada arreglo en forma secuencial de ahí labview hace una conversión de arreglo a número y de número a string y se concatena a un tiempo que se puede elegir en nuestro caso usamos 500ms para las pruebas y para la demostración de aplico a 10ms.

X. Resultados

Afortunadamente fue lo esperado solo por el tiempo de implementación nos vimos en la necesidad de pintar para las imágenes la primera mitad del led y luego la segunda mitad.

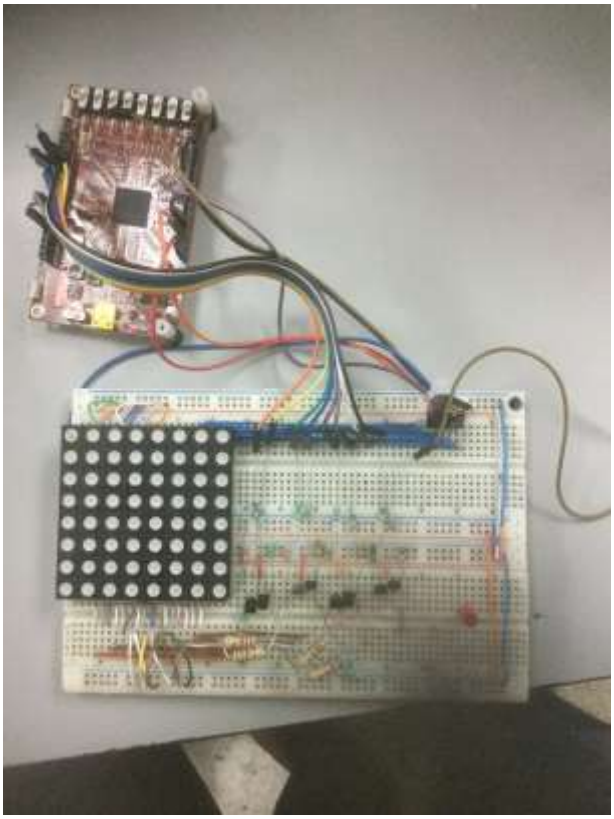

Figura8. Circuito armado con la Fpga el protoboard y la matriz de led.

XI. Conclusiones

Para cargar correctamente los archivos y generar el .bit, es necesario verificar antes que se asignó el espacio correcto para poder usar los recursos necesarios, ya que en este caso generará el error de falta de espacio, otro punto importante es que se debe de asegurar que los puertos fueron asignados y conectados correctamente a la arquitectura. Además de esto para la interfaz entre el equipo y la tarjeta FPGA se debe de conocer que funciones del software se usaran, ya que en este caso se utilizó LabView y se tuvieron dificultades para asignar las funciones se debe considerar la velocidad, el tamaño y tiempo para la sincronización de lo clk.

XII. BIBLIOGRAFIA Y REFERENCIAS

Led: http://unicrom.com/Tut_diodo_led.asp

Led RGB: http://www.forosdeelectronica.com/f27/led-rgb-12001/

PWM: http://www.ibertronica.es/blog/refrigeracion/funcion-pwm

Colores: http://www.aulaclic.es/photoshop-cs5/a_1_8_1.htm

PLB_CLOCK_TIME: https://forums.xilinx.com/t5/Spartan-Family-FPGAs/Problem-in-generating-PWM-output-using-XPS-TIMER-Module/td-p/19034

Matriz_de_led: http://www.monografias.com/trabajos101/matrices-leds/matrices-leds.shtml

Estructura de código C: Noé Mendoza Morales, estudiante de la carrera de electrónica en la facultad de Ciencias de la electrónica, BUAP, Matrícula 2010XXXXX.