Laboratorio de Sistemas Empotrados

Juan Carlos Fabero

Hortensia Mecha

José Manuel Mendías

Carlos González

Juan Antonio Clemente

03 de noviembre de 2017

Contenido

1 1. Sistema básico

El sistema básico, que vamos a utilizar en el resto de las prácticas, estará formado por los siguientes elementos:

1. Core procesador *Microblaze*
2. Memorias de datos e instrucciones y controladores de memoria
3. Controlador RS232 conectado al puerto serie
4. GPIO conectados a los switches
5. GPIO conectados a los leds.

Este sistema lo vamos a implementar sobre la placa basada en FPGA disponible en el laboratorio. Se trata de una placa con una Spartan 3 (XC3s1000) y una placa de expansión 3.0 como se muestra en la *Figura 1*.

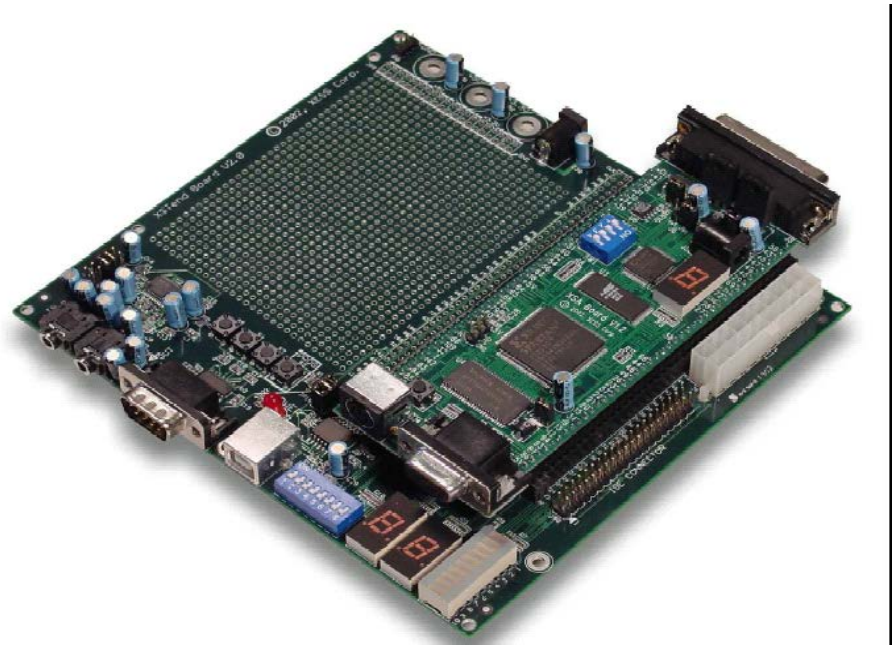


Figura 1 Placa prototipado basada en Spartan 3(XC3S1000) con placa de extensión 3.0

Además, utilizaremos la placa de expansión con diversos periféricos que se muestran en la *Figura **2***.

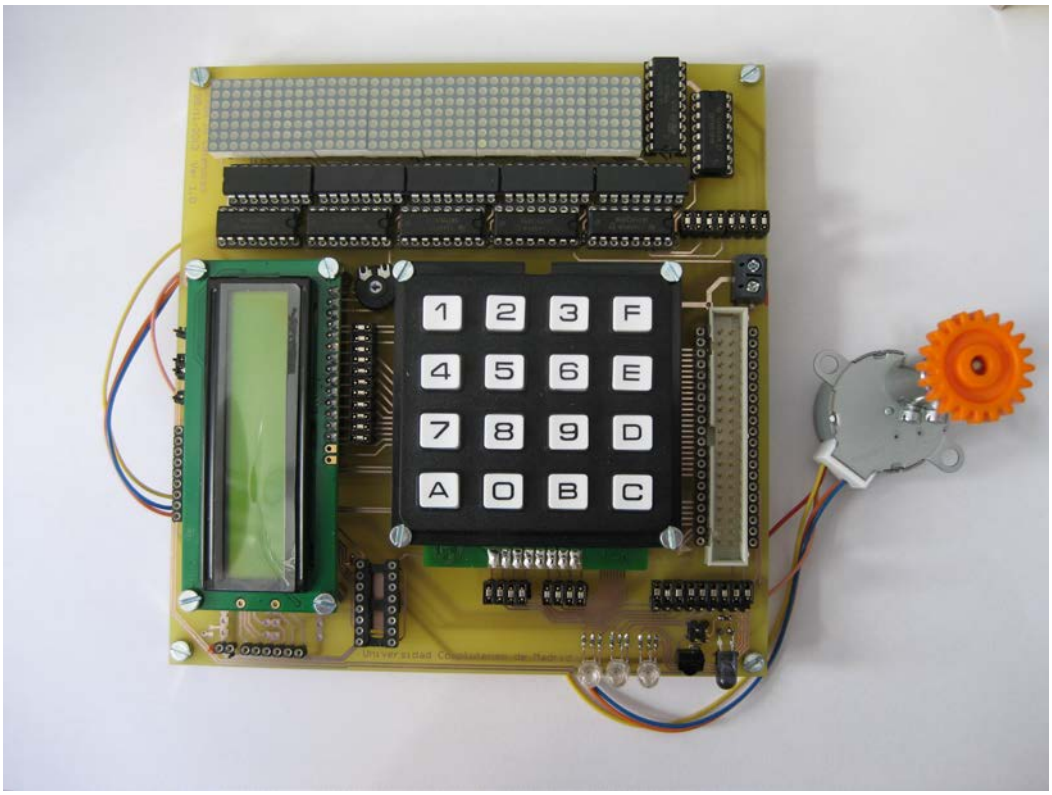


Figura 2. Placa de expansión

Dentro de la herramienta EDK que vamos a utilizar para implementar el sistema hardware, todos los módulos del sistema aparecen como IPcores dentro de la pestaña IPcatalog. La conexión de los distintos elementos debe quedar como en la *Figura **3***.

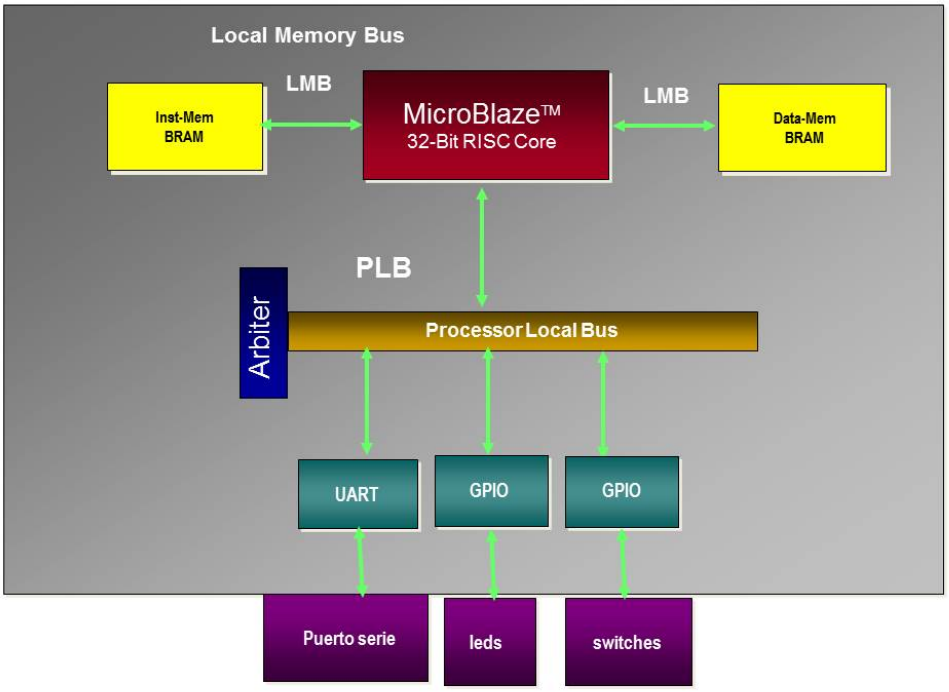


Figura 3. Sistema básico

En el entorno de EDK el diseño quedará como se muestra en la *Figura 4*

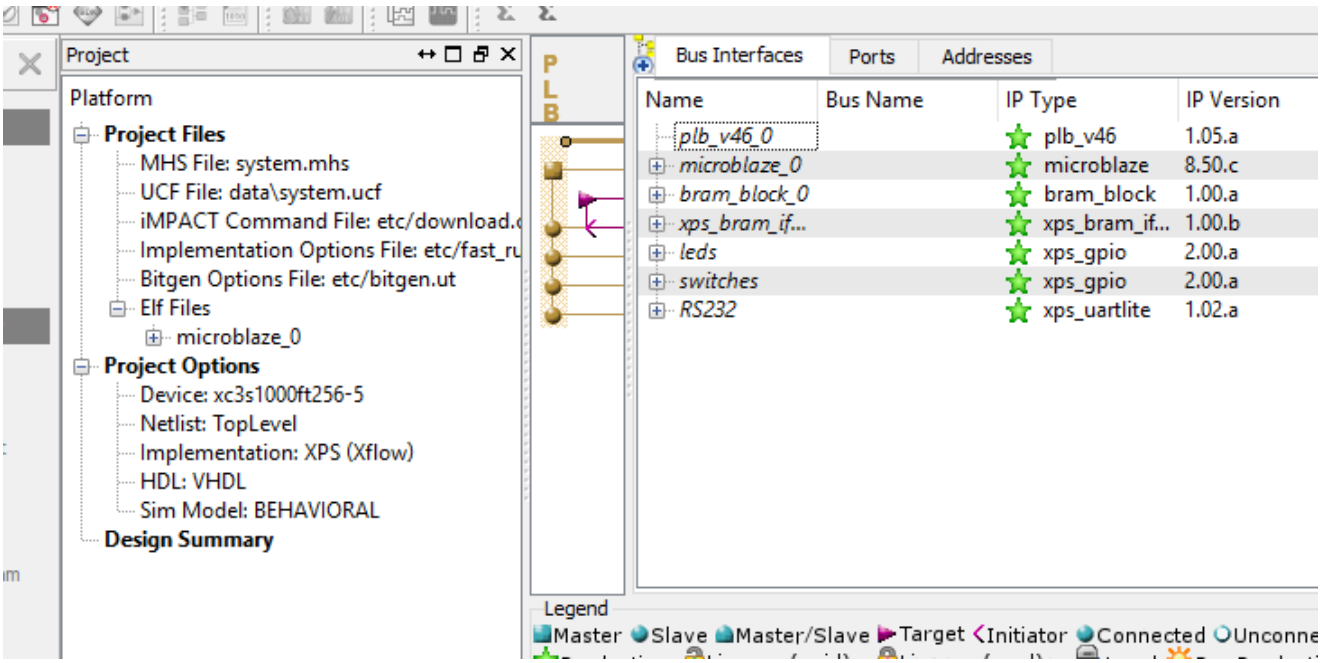


Figura 4. Entorno EDK

El fichero de restricciones (User Constraints File - '**.ucf*') *debe* tener las restricciones necesarias para conectar el reloj, el reset, los leds y los switches y las líneas rx y tx del protocolo RS232. Un ejemplo para las placas XSA3S1000 con la extensión XST 3. 0 basadas en Spartan 3 se muestra el *Cuadro **1***.

```
# Reloj  y restricciones del sistema ( reloj a 50MHz )
Net clk_pin LOC=P8;
Net clk_pin TNM NET = clk_pin ;
TIMESPEC TS clk_pin = PERIOD clk_pin 20000 ps ;

#Reset  (sw 1 de la placa base)
Net  rst_pin LOC= K4;
Net rst_pin  TIG;

#### Module RS232 constraints
Net RX pin LOC=G5 ;
Net TX pin LOC=J2 ;

## GPIO LEDs ###############
# barra de leds placa extendida
NET leds pin <0 > LOC=L5;
NET leds pin <1 > LOC=N2;
NET leds pin <2 > LOC=M3;
NET leds pin <3 > LOC=N1;

## GPIO SWITCHEs ###################################
#switches placa extendida
NET switches_pin<0> LOC=P12;
NET switches_pin<1> LOC=J1;
NET switches_pin<2> LOC=H1;
NET switches_pin<3> LOC=H3;
```

*Cuadro 1. Fichero *.ucf del sistema básico*

Desde el entorno de EDK, se genera la netlist y se exporta el diseño a SDK donde podemos generar el programa que se ejecutará en *Microblaze*. Podemos usar las funciones de acceso a los periféricos que se generan automáticamente para todos los que aparecen en nuestro sistema; por ejemplo, para los GPIO están definidas en gpio.c, y para la UART en xuartlite.c. Simplemente se trata de lecturas y escrituras en posiciones de memoria.

Necesitamos al menos 2 variables: una para acceder a los switches (*GpioSwitches*) y otra para escribir en los leds (*GpioLEDs*), y dos constantes que son las direcciones de memoria que ocupan los switches (XPAR_SWITCHES_ DEVICE_ID) y los leds (XPAR_LEDS_DEVICE_ID). En el *Cuadro **2*** se muestra un ejemplo que lee los switches y escribe en los leds.

```
/ * Instancia del driver para configurar como salida un dispositivo GPIO * /
XGpio GpioLEDs ;

/* Instancia del driver para configurar como entrada un dispositivo GPIO */
XGpio GpioSwitches ;

/* Configuración de la GPIO para los LEDs de la placa extendida (Obtiene el puntero a la estructura) */
XGpio_Initialize (& GpioLEDs , XPAR_LEDS_DEVICE_ID );

/* Coloca la dirección de salida */
XGpio_SetDataDirection (& GpioLEDs , 1 , 0x0);

/* Configuración de la GPIO para los Switches (obtiene el puntero a la estructura) */
XGpio_Initialize (& GpioSwitches , XPAR_SWITCHES_DEVICE_ID );

/* Coloca la dirección de entrada */
XGpio_SetDataDirection (& GpioSwitches , 1 , 0 xF F );

/* Para leer de los switches usaremos la función XGpio DiscreteRead*/
DataRead = XGpio_DiscreteRead (& GpioSwitches , 1);

/* Para escribir en los leds usaremos la instrucción XGpio DiscreteWrite*/
XGpio DiscreteWrite (& GpioLEDs , 1 , DataRead );
```

Cuadro 2. Ejemplo de uso de los GPIO

Como durante la creación del sistema se ha definido que la entrada/salida se realiza a través de la UART, para escribir en el terminal usamos la función *print*; por ejemplo*: print*(" ---Test para switches y leds ---\n\r");

1.1. Adición de un nuevo periférico

La adición de un nuevo periférico viene explicada en las transparencias de clase en el tema "*Adición de un periférico a un SoC*".

1.2. Adición de un periférico previamente creado

Para incorporar un periférico creado previamente al sistema hay que descomprimir el fichero *'core.zip*' correspondiente en el directorio raíz del proyecto básico de EDK (es decir, a la misma altura que el fichero *'system.xmp'*).
A continuación, hay que importar dicho periférico desde EDK. Para ello pulsamos sobre

'Hardware' ->'Create or Import Peripheral'

En ese momento nos aparecerán una serie de ventanas de diálogo.

1. En la primera ventana que nos aparecerá pulsamos '*Next*' .
2. En la siguiente ventana seleccionamos *'Import existing peripheral'* y pulsamos '*Next*' 2 veces.
3. En '*Name*' escribimos el nombre del periférico, por ejemplo ps2, vga, o el que vayamos a utilizar, y pulsamos '*Nex*t' .
4. A continuación, pulsamos 'Next'.
5. Seleccionamos *'Use existing Peripheral Analysis Order file (*.pao)*', pulsamos en '*Browse*' y buscamos el fichero '*.pao' dentro de la carpeta descomprimida (dentro del directorio 'data').
6. A continuación, pulsamos '*Next'* y pulsamos *'Add Files'*, seleccionamos todos los ficheros que estaban en el fichero comprimido y pulsamos '*Nex*t'.
7. Seleccionamos 'PLBV46 Slave (SPLB)' y pulsamos '*Nex*t' 2 veces.
8. En *'Parameter determine high address*' seleccionamos 'C HIGHADDR' y pulsamos '**Next**'
9. Deseleccionamos *'Select and configure interrup(s)'* y pulsamos '*Next*' 3 veces y para terminar pulsamos en '*Finish*'

Si todo ha salido bien, debería aparecernos el core en la pestaña *'IP Catalog'* debajo de *'Project Local PCores*'. Para añadirlo al sistema, lo seleccionamos y lo arrastramos a la pestaña *'Bus Interfaces'*. Conectamos el periférico al bus PLB y en la pestaña '*Ports*' configuramos sus puertos como externos. Finalmente, añadimos la asignación de los puertos externos a los pines de la FPGA en el fichero '*.*ucf*'.

2. Teclado PS2

El interfaz teclado PS2 nos permite conectar un teclado estándar al conector PS2 de la placa de FPGA. La comunicación es de tipo serie con 2 líneas, una de reloj y otra de datos, que sigue el esquema de la *Figura 5*. Las líneas CLK y Data son las líneas que físicamente se conectan *al* conector PS2, y dependiendo de la placa tendrán una situación en un pin diferente, que habrá que indicar en el fichero '**.ucf*'.

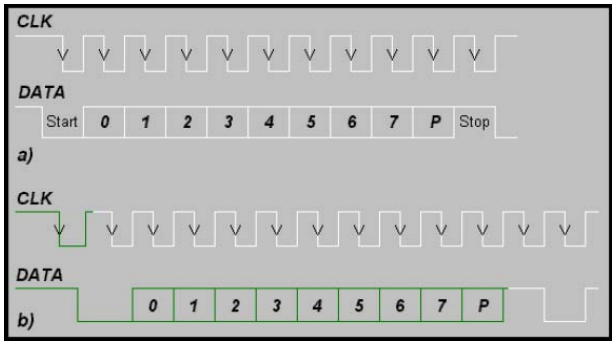


Figura 5. Protocolo PS2

La entidad del control del teclado se debe instanciar desde el interfaz *user_logic*, se denomina *keyboard*, y se define en el fichero keyboard.vhd. Esta entidad instancia a su vez a la entidad *ps2KeyboardInterface*, definida en el archivo ps2_kdb.vhd, donde realmente se realiza el control del protocolo PS2.

```
ENTITY keyboard IS
PORT (
rst : in std logic ;
clk : in std logic ;
ps2Clk : in std logic ;
ps2Data : in std logic ;
Datap : out std logic vector (7 downto 0);
ldData : out std logic ;
ackData : in std logic
); END keyboard ;
```

Cuadro 3. Entity del teclado PS2

El interfaz de la entidad *keyboard* se muestra en el *Cuadro **3***, donde *rst* y *clk* son las líneas globales de reset y reloj respectivamente; *ps2CLK* y *ps2Data* son las líneas CLK y Data del protocolo, es decir las líneas externas de la comunicación PS2; *Datap* es el dato de 8 bits correspondiente a la última tecla pulsada, y *ldData* y *ackData* son las líneas de handshake que indican que hay un nuevo dato (*ldData*=1), y que el dato se ha cargado en el registro de datos (*ackData*=1). En *user_logic*, cada vez que se activa *ldData*, el valor *Datap* se debe almacenar, y activar la señal *ackData*

2.1. Conexión al sistema

En user_logic añadir los puertos de entrada salida correspondientes a CLK y Data, que como hemos visto en el ***Cuadro 3***, se llaman *ps2Clk y ps2Data*. El resultado se muestra en el *Cuadro **4***.

```
generic (
--ADD USER GENERICS B ELOW THIS LINE ----------
---USER generics added here
-- ADD USER GENERICS ABOVE THIS LINE ----------
-- DO NOT EDIT BELOW THIS LINE ----------
-- Bus protocol parameters , do not add to or delete
C_SLV_DWIDTH : integer := 32;
C_NUM_REG : integer := 1
-- DO NOT EDIT ABOVE THIS LINE ----------
);
port
(
-- ADD USER PORTS BELOW THIS LINE ----------
--USER ports added here
-- ADD USER PORTS ABOVE THIS LINE ----------
ps2Clk : IN std_logic ;
ps2Data : IN std_logic ;
-- DO NOT EDIT BELOW THIS LINE ----------
-- Bus protocol ports , do not add to or delete
```

Cuadro 4. Adición puertos de E/S del teclado PS2 en usr_logic

Es necesario definir dichas señales externas como pines externos también en el módulo top del periférico y dar sus restricciones correspondientes en el fichero '**.ucf*' . En el caso de la placa que vamos a utilizar en el laboratorio su situación es el pin B6 para el reloj y el pin E13 para los datos. Por tanto, al fichero '**.ucf*' hay que añadir las dos líneas que se muestran el en *Cuadro **5***.

```
Net ps2clk pin LOC = B16 ;
Net ps2Data pin LOC =E13;
```

*Cuadro 5. Restricciones del teclado PS2 en fichero *.ucf*

Para la comunicación con *Microblaze*, este periférico consta de un único registro *slv_reg0,* donde se debe cargar el valor correspondiente a la última tecla pulsada. Cuando se activa la señal *ldData*, el valor *Datap* se debe almacenar en *slv_reg0*(24 to 31) y activar *ackData*. Esto se realiza mediante el código mostrado en el *Cuadro **6***.

```
if (ldData = '1') then
slv_reg0 (24 to 31 ) <= Datap ;
ackData <= '1';
else ackData <= '0';
end if;
```

Cuadro 6. Actualización del registro de datos en el periférico PS2

2.2. Control software

En el fichero *.c debe definirse la dirección de memoria correspondiente al teclado PS2, que en el ejemplo que se proporciona en el Campus Virtual es la 0xC3800000.

```
# define BASE_ADDRESS_PS2 0 xC3800000
```

La función para leer una tecla en este ejemplo, que viene definida en el directorio driver correspondiente al periférico, es *TECLADO_ mReadSlaveReg0* (baseaddr , 0). Por tanto, para leer una tecla debemos usar el siguiente segmento de código:

```
baseaddr = BASE_ADDRESS_PS2 ;
Reg32Value = TECLADO_ mReadSlaveReg0 ( baseaddr , 0);
```

Un ejemplo de programa que pide que se pulse una tecla y la muestra por pantalla es:

```
xil_printf("Pulse una tecla cualquiera \n\r");
Reg32Value = TECLADO_ mReadSlaveReg0 ( baseaddr , 0);
TeclaOld = Reg32Value ;
while (1) {
if ( Reg32Value ! = TeclaOld )
        xil_printf(" -Se ha leido %d del registro 0 del teclado \n\r", Reg32Value );
 TeclaOld = Reg32Value ;
 Reg32Value = TECLADO mReadSlaveReg0 ( baseaddr , 0);
}
```

3. Teclado Matricial o Keypad

El teclado matricial (*Figura **6***) es una matriz de 4 filas por 4 columnas de teclas conectadas como se muestra en la *Figura **7***. En la tarjeta donde se monta se suelen conectar las columnas (o filas) a alimentación a través de resistencias de pull-up. Por tanto, si no hay ninguna tecla pulsada, por las columnas se recibe 1. Para el control del teclado, por las filas se envían ceros. Cuando se pulsa una tecla se produce la continuidad eléctrica fila-columna, recibiéndose por la columna correspondiente un 0 en lugar de un 1. En controlador del teclado, mediante un sistema de *polling* determina cuál es la tecla presionada.

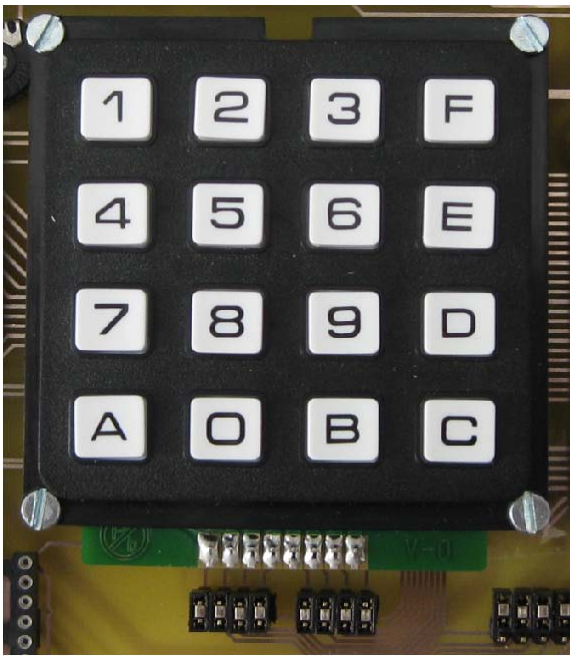


Figura 6. Teclado matricial 4 × 4

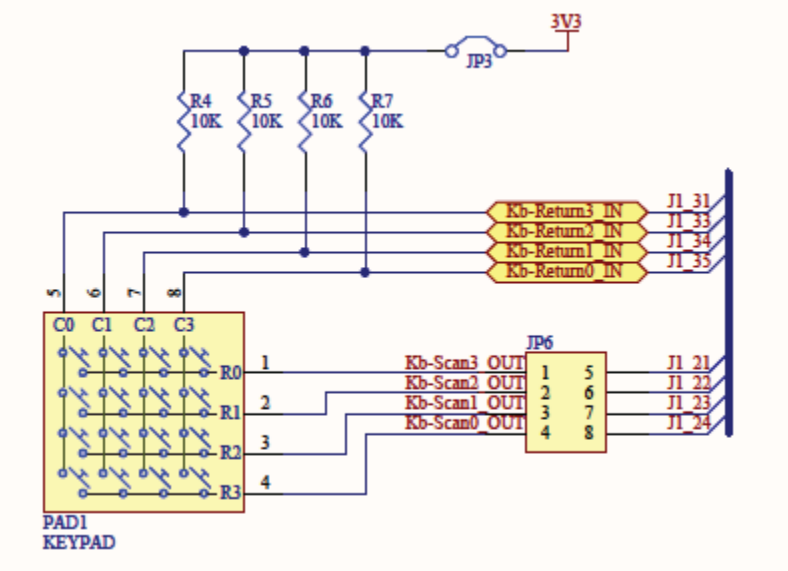


Figura 7. Conexión del teclado matricial

La entidad del control del teclado matricial, que se debe instanciar desde el interfaz *user_logic*, se denomina *teclaDetect*, y se define en el fichero teclaDetect.vhd. El interfaz de dicha entidad se muestra en el *Cuadro **7***. Los puertos S (*Scan*) y R (*Return*) son las señales correspondientes a las filas y las columnas. *KeyCode* es el código de la tecla pulsada y *keyPressed* se pone a 1 cuando se ha reconocido la pulsación de una tecla.

```
entity teclaDetect is
   Port (
         reloj : in  STD_LOGIC;
          reset : in  STD_LOGIC;
         S : out  STD_LOGIC_VECTOR (3 downto 0);
         R : in  STD_LOGIC_VECTOR (3 downto 0);
          KeyCode   :   out     STD_LOGIC_VECTOR   (3
downto 0);
          keyPressed: out std_logic
);
end teclaDetect;
```

Cuadro 7. Entity del teclado matricial

El interfaz hardware en *user_logic* dispone de un registro *slv_reg0* donde se almacena la última tecla pulsada en los bits del 0 al 3.

3.1. Conexión al sistema

En user_logic y en el top del periférico hay que añadir los puertos de entrada salida correspondientes a las filas y columnas, que como hemos visto en el ***Cuadro 7***, se llaman *S y R.* El resultado se muestra en el *Cuadro **8***.

Además, hay que dar sus restricciones correspondientes en el fichero '**.ucf*' de la ubicación de los puertos correspondientes en el top del sistema. En el caso de la placa que vamos a utilizar en el laboratorio su situación se muestra en el *Cuadro **9***. El nombre de los dos puertos externos se muestra en el archivo top del sistema y es S_pin y R_pin respectivamente.

```
generic (
--ADD USER GENERICS BELOW THIS LINE ---------------
--USER generics added here
--ADD USER GENERICS ABOVE THIS LINE ---------------
--DO NOT EDIT BELOW THIS LINE ----------------------
-Bus protocol parameters , do not add to or delete
C SLV DWIDTH : integer := 32; C NUM REG : integer := 1
--DO NOT EDIT ABOVE THIS LINE ----------------------
);
port (
--ADD USER PORTS BELOW THIS LINE -----------------
---USER ports added here
S : out STD LOGIC VECTOR (3 downto 0);
R : in STD LOGIC VECTOR (3 downto 0);

--DO NOT EDIT BELOW THIS LINE -------------------
---Bus protocol ports , do not add to or delete
);
```

Cuadro 8. Adición puertos de E/S del teclado matricial en usr_logic

```
net S_pin<3> loc=F2;
net S_pin<3> iostandard=LVCMOS25;
net S_pin<2> loc=G4;
net S_pin<2> iostandard=LVCMOS25;
net S_pin<1> loc=G3;
net S_pin<1> iostandard=LVCMOS25;
net S_pin<0> loc=G1;
net S_pin<0> iostandard=LVCMOS25;
net R_pin<3> loc=J14;
net R_pin<3> iostandard=LVCMOS25;
net R_pin<2> loc=H14;
net R_pin<2> iostandard=LVCMOS25;
net R_pin<1> loc=M4;
net R_pin<1> iostandard=LVCMOS25;
net R_pin<0> loc=P1;
net R_pin<0> iostandard=LVCMOS25;
```

*Cuadro 9. Restricciones del teclado matricial en fichero *.ucf*

Para la comunicación con *Microblaze*, este periférico utiliza un único registro *slv_reg0*, donde se debe cargar el valor correspondiente a la última tecla pulsada. Cada vez que se pulsa una tecla, se activa la señal *keyPressed* y el valor de la tecla se envía por *KeyCode*. En *user_logic* se añade el hw necesario para cargar dicho valor en *slv_reg0*. Esto se realiza mediante el código mostrado en el *Cuadro **10***.
Es aconsejable que cada vez que se lee el registro de datos desde *Microblaze* se borre su contenido escribiendo en el mismo un 0

```
if ( slv reg write sel = "1") then
        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 ;
 elsif ( keyPressed = '1') then
        slv_reg0 (0 to 3) <= KeyCode (7 downto 4);  --La tecla pulsada en bits 0 -3
end if;
```

Cuadro 10. Actualización del registro de datos en el teclado matricial

3.2. Control software

En el fichero *.c debe definirse la dirección de memoria correspondiente al keypad, que en el ejemplo que se proporciona en el laboratorio es la 0xC9600000

```
# define BASE ADDRESS KEYPAD 0xC9600000
```

Para el ejemplo que se proporciona en el laboratorio, la función para escribir un dato en el registro de datos, que viene definida en el directorio de drivers del periférico es *KEYPAD_mWriteReg*, cuya llamada se hace de la siguiente forma:

```
baseaddr = BASE ADDRESS KEYPAD ;
KEYPAD_mWriteReg ( baseaddr , 0, Dato );
```

El código para leer una tecla debe utilizar la función correspondiente de lectura del registro, que para el ejemplo del laboratorio es *KEYPAD_mReadReg* (baseaddr , 0). Un ejemplo sería:

```
Reg32Value = KEYPAD_mReadReg ( baseaddr , 0);
```

En el *Cuadro **11*** se muestra un ejemplo de programa que lee una tecla y la muestra por el *hypertermina.*

```
print(" ---Test para el keypad ---\n\r");
baseaddr = BASE ADDRESS KEYPAD ;
xil printf(" Pulse una tecla cualquiera \n\r");
Reg32Value = KEYPAD_mReadReg ( baseaddr , 0);
xil printf (" Se ha leido %d del registro 0 del teclado \n\r" , Reg32Value );
 KEYPAD_mWriteReg ( baseaddr , 0 ,0);
TeclaOld = Reg32Value ; /* Se escribe un 0 para borrar la última tecla leída */
while (1) {
        if ( Reg32Value != TeclaOld )  {
                xil printf (" Se ha leido %d del registro 0 del teclado \n\r" , Reg32Value );
                TeclaOld = Reg32Value ;
        }
Reg32Value = KEYPAD_mReadReg ( baseaddr , 0);
KEYPAD_mWriteReg ( baseaddr , 0 ,0);  /* Se escribe un 0 para borrar la última tecla leída */
}
```

Cuadro 11. Ejemplo de uso del teclado matricial

4. Matriz de Puntos o Banner

Este dispositivo, mostrado en la *Figura **8***, es una matriz 40x7 de leds multiplexada (lógica directa para filas e inversa para columnas), accesible en serie a través de 2 registros de desplazamiento de 40 y 7 bits respectivamente.

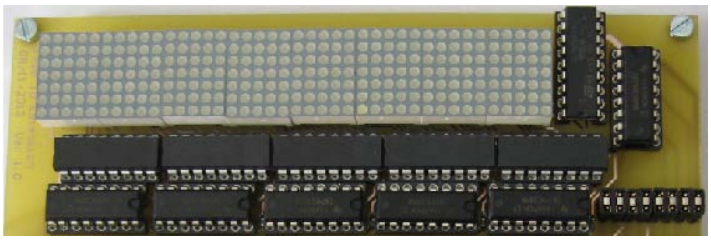


Figura 8. Matriz de puntos usada en el laboratorio

El esquema de conexión a la placa que utilizamos en el laboratorio se muestra en la *Figura **9***. Obsérvese que se trata de 8 módulos, cada uno de ellos formado por una matriz de 8 filas * 5 columnas.

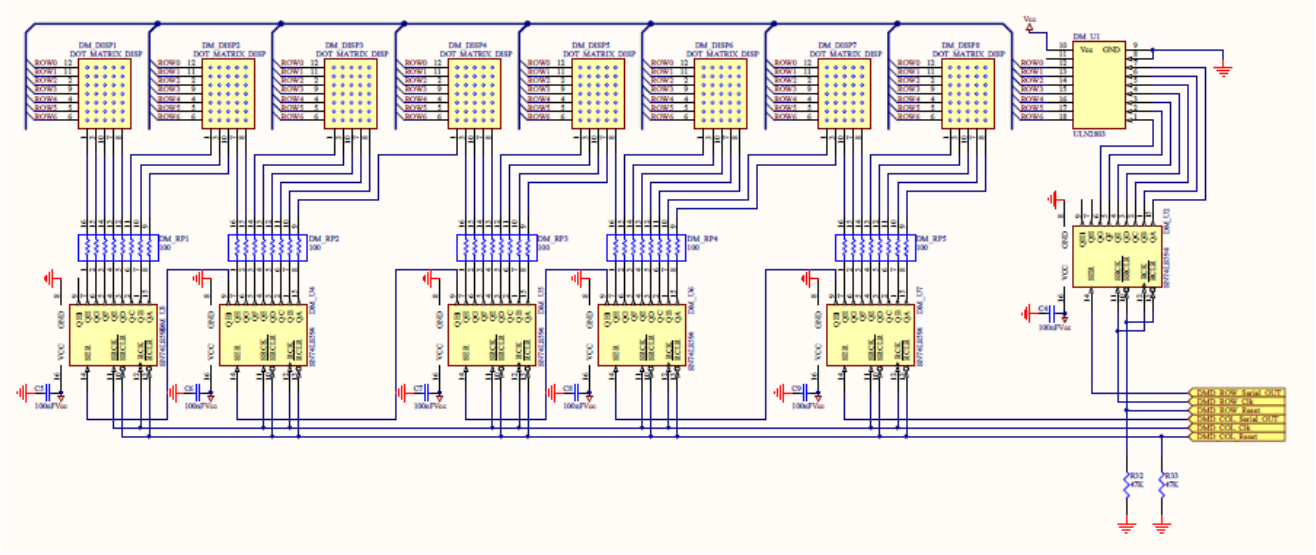


Figura 9. Conexión de la matriz de puntos a la placa del laboratorio.

La entidad correspondiente al controlador de este periférico se llama *bannerDesp*, se muestra en el *Cuadro 12* y se encuentra en el fichero bannerDesp.vhd. Para su utilización debe instanciarse dentro del interfaz user_logic. El controlador se encarga de realizar un barrido de todos los píxeles de la matriz, con una persistencia de 70µs (modificable por hw). Para cada fila se mandan los valores de los 40 bits con un desplazamiento de 3 píxeles por segundo (modificable por hw).

```
entity bannerDesp is
port
(
reset_in: in std_logic;        -- reset
clock: in std_logic; --
col_serial_out: out std_logic;
col_clk: out std_logic;
row_serial_out: out std_logic;
row_clk: out std_logic;
reset_out, reset_out2: out std_logic;
fila: in std_logic_vector (2 downto 0);
columna: in std_logic_vector ( 2 downto 0);
dato: in std_logic_vector (4 downto 0);
load: in std_logic

);
end bannerDesp;
```

Cuadro 12. Entity de la matriz de puntos

Las líneas *col_serial_out*, *col_clk*, *row_serial_out*, *row_clk* y *reset_out* son las correspondientes a entrada_serie_columnas, reloj_columnas, entrada_serie_filas, reloj_filas y reset (en la placa antigua hay 2 reset, pero en la nueva sólo uno) del periférico, y que están conectadas físicamente a los pines correspondientes de la FPGA. Las entradas *fila*, *columna*, *dato* y *load* sirven para actualizar la memoria interna a la entidad BannerDesp, que almacena lo que se quiere mostrar en la matriz de puntos, según veremos a continuación. La memoria está formada por 7 filas, y 8 columnas de 5 bits cada una. Cada elemento de dicha memoria corresponde realmente a una fila de uno de los módulos mostrados en la ***Figura 9***.

El interfaz hardware en user_logic dispone de una FIFO donde *Microblaze* escribe lo que se quiere almacenar en cada fila/columna de dicha memoria según el siguiente formato:

- La fila se encuentra en el byte menos *significativo* (bits 5-7):
  fila(2 downto 0) <= *WFIFO2IP_Data*(5 to 7);
- La columna en el byte 1 (bits 13-15):
  columna(2 downto 0) <= *WFIFO2IP_Data*(13 to 15);
- El dato se encuentra en el byte 2 (bits 19 al 23):
  dato(4 downto 0) <= *WFIFO2IP_Data*(19 to 23);

Cada vez que *Microblaze* escribe en la FIFO, en *usr_logic* se lee dicho valor, se activa la señal *load* y se manda al controlador del periférico *BannerDesp* el contenido del *dato* que se ha de modificar y en qué posición de memoria, a través de las entradas *dato*, *fila* y *columna*.

.

4.1. Conexión al sistema

En user_logic y en el top del periférico hay que añadir los puertos de entrada/salida correspondientes a las salidas externas del banner, que como hemos visto en el *Cuadro 12* son *col_serial_out, col_clk, row_serial_out, row_clk, reset_out, y reset_out2* (esta última sólo para las placas antiguas). El resultado se muestra en el *Cuadro 13*.

```
generic (
--ADD USER GENERICS BELOW THIS LINE ---------------
--USER generics added here
--ADD USER GENERICS ABOVE THIS LINE ---------------
--DO NOT EDIT BELOW THIS LINE ---------------------
--Bus protocol parameters , do not add to or delete
C SLV DWIDTH : integer := 32;
C NUM REG : integer := 1
--DO NOT EDIT ABOVE THIS LINE ---------------------
);
port (
--ADD USER PORTS BELOW THIS LINE -----------------
--USER ports added here
col_serial_out : out std logic ;
col_clk : out std logic ;
row_serial_out : out std logic ;
row_clk : out std logic ;
reset_out : out std logic ;
--DO NOT EDIT BELOW THIS LINE ------------------
--Bus protocol ports , do not add to or delete
);
```

Cuadro 13. Adición puertos de E/S de la matriz de puntos en usr_logic

Por otro lado, hay que dar las restricciones correspondientes de la ubicación de los puertos en el fichero '**.ucf*' del top del sistema. En el caso de la placa que vamos a utilizar en el laboratorio su situación se muestra en el *Cuadro 14*. El nombre de los puertos externos en el archivo top del sistema dependerá de la elección que haya realizado el diseñador. Lo que se muestra en dicho cuadro es el diseño proporcionado en el laboratorio.

```
net col_serial_out_pin loc=L5;
net col_serial_out_pin iostandard = LVCMOS25 ;
net col_clk_pin loc =N2;
net col_clk_pin iostandard = LVCMOS25 ;
net row_serial_out_pin loc=M3;
net row_serial_out_pin iostandard = LVCMOS25 ;
net row_clk_pin loc =N1;
net row_clk_pin iostandard = LVCMOS25 ;
net reset_out_pin loc = T13 ;
net reset_out_pin iostandard = LVCMOS25 ;
```

Cuadro 14. Restricciones de la matriz de puntos en fichero *.ucf

4.2. Control software

En el fichero *.c debe definirse la dirección de memoria correspondiente al banner, que en el ejemplo que proporcionamos es la 0xC5800000.

```
# define BANNER BASE ADDR 0 xC5800000.
```

La función para escribir un dato en la FIFO es BANNER_mWriteToFIFO(baseaddr,0,Data). Un ejemplo de utilización es:

```
baseaddr = BANNER_BASE_ADDR_0;
BANNER_mWriteToFIFO ( baseaddr , 0, Data );
```

Antes de escribir, hay que comprobar siempre que la FIFO no esté llena mediante la función BANNER_mWriteFIFOFull (baseaddr).
Para generar el dato a enviar, hay que concatenar los valores de fila, columna y dato según las especificaciones dadas en *usr_logic*. Para ello puede utilizarse la siguiente instrucción:

```
Data =(( fila & 0 xff ) << (31 -7)) +  (( columna & 0 xff ) << (31 -15)) + (( dato & 0 xff ) << (31 -23));
```

5. Control de un monitor VGA

VGA es el acrónimo de la expresión inglesa “Video Graphics Array”, es decir, matriz gráfica de vídeo. Se trata de un estándar de presentación de imágenes en pantalla desarrollado por IBM.
Una pantalla de tubos de rayos catódicos (CRT) está formado por un tubo en forma piramidal (*Figura **10***) cuya base está recubierta de un material fluorescente, un filamento que produce un haz de electrones (y varios para pantallas en color) y un par de bobinas deflectoras perpendiculares que permiten modificar la trayectoria del haz de electrones El choque del haz de electrones con el material fluorescente hace que éste se ilumine El tipo de material fluorescente determina el color de la luz. La densidad del haz de electrones determina la intensidad de la luz. La desviación introducida por las bobinas determina el lugar de impacto del haz. El tamaño del punto de impacto determina la resolución de la pantalla.
Existen dos métodos para generar imágenes sobre la pantalla: CRT de barrido (raster scan), donde el haz barre la superficie fluorescente de una forma sistemática modulando la intensidad del haz de acuerdo a la información por representar, y CRT vectoriales en el que se manipula el haz para formar directamente los dibujos.
En los CRT de barrido, el haz de electrones recorre la pantalla completa comenzando por la esquina superior izquierda y recorre horizontalmente una fila de píxeles (*Figura **11***). Cuando alcanza el final de la fila, se apaga momentáneamente el cañón y se coloca al comienzo de la siguiente fila. Cuando todas las filas han sido recorridas y se ha alcanzado la esquina inferior derecha, se apaga el cañón y se retorna al comienzo.

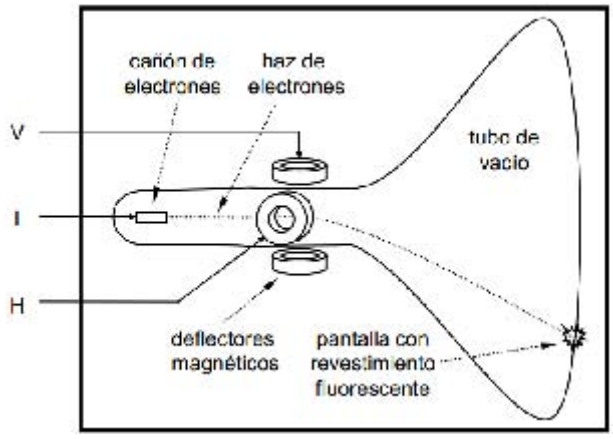


Figura 10. Diseño de un monitor de tubo de rayos catódicos

Para controlar el barrido, el interfaz envía a la pantalla tres señales:

- **Sincronización horizontal**: que marca el comienzo y final de una fila de píxeles (µlínea). Es la señal H mostrada en la ***Figura 10***.
- **Sincronización vertical**: que marca el comienzo y final de una imagen completa (cuadro o frame). Es la señal V mostrada en la *Figura **10***.

- **Intensidad**: que indica la intensidad del haz de electrones. Es la señal I mostrada en la *Figura **10***.

La información que se ha de representar se almacena en la memoria de refresco.

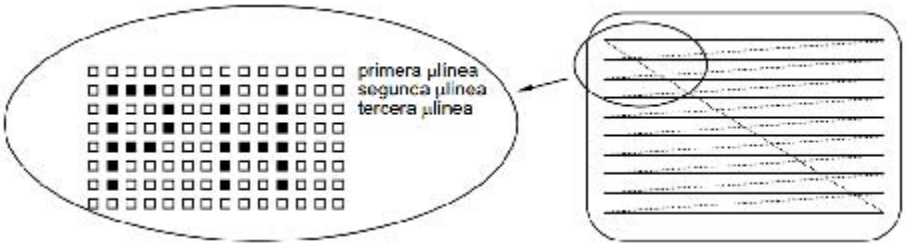


Figura 11. Ejemplo de barrido de pantalla

En un monitor en color (*RGB*) la pantalla se cubre con tríos de puntos de fósforo de colores diferentes (rojo, verde, azul) colocados muy próximos. Cada punto del trío puede ser estimulado por un cañón diferente. Para evitar que la dispersión de los haces pueda ocasionar que un instante se ilumine más de un trío, los haces se pasan a través de una máscara. La intensidad relativa de los diferentes haces determina el color del trío.

Para controlar el barrido, el interfaz envía a la pantalla cinco señales:

- Sincronización horizontal y sincronización vertical: que controlan la trayectoria de los tres haces (que recorren acompasadamente la pantalla).
- Intensidad roja, intensidad verde e intensidad azul: que indican por separado la intensidad de cada uno de los haces de electrones (valores digitales que pasan a través de conversores digitales/analógicos).

Los interfaces para monitores en color pueden requerir memorias de refresco enormes y altas velocidades de transferencia. Por ello se suelen utilizar paletas que reducen el número de colores que pueden mostrarse simultáneamente a un subconjunto de los colores que permite la pantalla. Los colores que se almacenan en la memoria de refresco se traducen a colores reales a través de una memoria de paleta.

La entidad correspondiente al controlador de este periférico se llama *vgacore*, se muestra en el *Cuadro 15* y se encuentra en el fichero vgacore.vhd. Las señales de salida referentes al control del monitor VGA físico son *hsync*b (1 bit) que es la señal de sincronismo horizontal, *vsync*b (1 bit), que es la señal de sincronismo vertical y *rgb* (9 bits) que es la señal que indica el color.

El resto de señales (load, color y rectángulo) corresponden a las señales del bus PLB utilizado para interconectar *Microblaze* con el controlador hardware y que explicaremos posteriormente.

El controlador se encarga de realizar un barrido de todos los píxeles de la pantalla (de 290*490 píxeles), con el sincronismo requerido por el monitor y explicado en clase. Dentro de dicho controlador, y para las prácticas que vamos a realizar en el laboratorio, se ha definido una memoria que almacena el contenido que se quiere mostrar en la pantalla. Dicha memoria es un array de 128 rectángulos (16 filas * 8 columnas), cada uno definido por 9 bits, 3 para cada color. Cada rectángulo tiene una dimensión de 16 filas* 8 columnas de píxeles, y todos los píxeles de un mismo rectángulo se pintarán del mismo color, almacenado en la memoria ram_type. Por tanto, sólo se van a utilizar de la pantalla 64*256 píxeles.

```
Type ram_type is array (0 to 127) of std_logic_vector(8 downto 0);
```

La comunicación entre *Microblaze* y el controlador hardware se realiza a través de una FIFO en la que *Microblaze* escribirá los comandos que deberán ser ejecutados secuencialmente por el controlador. Este tipo de esquema de conexión se utiliza frecuentemente en aquellos sistemas en los que la velocidad del

procesador es superior a la velocidad de respuesta del periférico, para evitar que el procesador pare su ejecución en espera de que termine el comando anterior

El formato de los datos en la FIFO es el siguiente:

- El número de rectángulo en el byte más *significativo* (bits 31-25):
  rectangulo(6 downto 0) <= *WFIFO2IP_Data*(31 downto 25);
- El color en los bits 8-0:
  color(8 downto 0) <= *WFIFO2IP_Data*(8 downto 0);

```
entity vgacore is
        port
        (
                reset: in std_logic;            -- reset
                clock: in std_logic; -- 12,5 MHz
                hsyncb: out std_logic;      -- horizontal (line) sync
                vsyncb: out std_logic;      -- vertical (frame) sync
                rgb: out std_logic_vector(8 downto 0);        -- red,green,blue colors
                load: in std_logic;
                color: in std_logic_vector(8 downto 0); -- color
                rectangulo: in std_logic_vector(6 downto 0); -- rectangulo      );
end vgacore;
```

Cuadro 15. Entity del controlador de vga

Cada vez que *Microblaze* escribe en la FIFO, en *usr_logic* se lee dicho valor, se activa la señal load y se manda al controlador de la vga (vgacore) el número de rectángulo que se va a modificar y el nuevo color, a través de las entradas color y rectángulo.

5.1. Conexión al sistema

```
entity user_logic is generic (
--ADD USER GENERICS BELOW THIS LINE ---------------
-USER generics added here --ADD USER GENERICS ABOVE THIS LINE ----------¬
--DO NOT EDIT BELOW THIS LINE --------------------
--Bus protocol parameters , do not add to or delete
C SLV DWIDTH : integer := 32;
C NUM REG : integer := 1
--DO NOT EDIT ABOVE THIS LINE ----------------¬
);
port (
--ADD USER PORTS BELOW THIS LINE -----------------
--USER ports added here
hsyncb : out std logic ;
vsyncb : out std logic ; --vertical ( frame ) sync
rgb : out std logic vector (8 downto 0); --red ,green ,blue colors
--Bus protocol ports , do not add to or delete
---
--DO NOT EDIT ABOVE THIS LINE ----------------¬
);
```

Cuadro 16. Adición de puertos de E/S al controlador de vga en usr_logic

En user_logic y en el top del periférico hay que añadir los puertos de entrada/salida correspondientes a las salidas externas del controlador de vga, que como hemos visto en el Cuadro 15 son *hsync*, *vsync* y *rgb*. El resultado se muestra en el *Cuadro 16*.
Por otro lado, hay que dar las restricciones correspondientes de la ubicación de los puertos en el fichero '**.ucf*' del top del sistema. En el caso de la placa que vamos a utilizar en el laboratorio su situación se muestra en el *Cuadro 17*. El nombre de los puertos externos en el archivo top del sistema dependerá de la elección que haya realizado el diseñador. Lo que se muestra en dicho cuadro es el diseño proporcionado en el laboratorio.

```
##                                VGA
###################################
Net hsyncb_pin LOC=B7;
Net vsyncb_pin LOC=D8;
Net rgb_pin<0> LOC=C9;
Net rgb_pin<1> LOC=E7;
Net rgb_pin<2> LOC=D5;
Net rgb_pin<3> LOC=A8;
Net rgb_pin<4> LOC=A5;
Net rgb_pin<5> LOC=C3;
Net rgb_pin<6> LOC=C8;
Net rgb_pin<7> LOC=D6;
Net rgb_pin<8> LOC=B1;
```

Cuadro 17. Restricciones del controlador de pantalla

5.2. Control software

Para llevar a cabo un control adecuado de la pantalla VGA se ha desarrollado el driver que implementa varias funciones capaces de modificar un total de 16*8 rectángulos de 16*8 píxeles cada uno. Para establecer un color determinado debe seguirse el formato 'rrrgggbbb' donde los tres primeros bits representan la intensidad de color rojo (red), lo tres siguientes la intensidad de color verde (green) y los tres últimos la intensidad de color azul (blue). A modo de ejemplo se han definido los colores que se muestran en el *Cuadro 18*.

```
#define    ROJO       0xE0000000
#define    VERDE      0x1C000000
#define    VERDE      OSCURO    0x0C000000
#define    AZUL       0x03800000
#define nfilas                  16
#define ncolumnas               8
```

Cuadro 18. Definición de colores para controlador VGA

La función *'PANTALLA_inicializa'* borra toda la pantalla dejándola en blanco. Dicho color se consigue sumando los colores ROJO+VERDE+AZUL.

```
void PANTALLA_inicializa (){
int i; int j;
PANTALLA_mResetWriteFIFO ( XPAR PANTALLA 0 BASEADDR );
for (i=0; i < nfilas*ncolumnas; i++){
        while (PANTALLA_mWriteFIFOFull(BASE_ADDRESS_VGA) == TRUE){}
        PANTALLA_mWriteToFIFO(BASE_ADDRESS_VGA, 0, VERDE + AZUL + ROJO + i);
}
}
```

Recordemos que los comandos que se mandan a través de la FIFO tienen el siguiente formato de 32 bits

- 7 LSB → Codifican el rectángulo a colorear
- 9 MSB → Codifican el color para el rectángulo

El resto de bits no se utilizan.

Para determinar la posición de un rectángulo situado en (fila,columna) realizamos la siguiente operación:

posicion=columna *nfilas+ fila; /* nfilas=16 */

Si queremos pintarlo de color rojo usamos

color=ROJO;

Teniendo en cuenta que en la definición de colores del *Cuadro 18* los datos correspondientes al color ya se han desplazado a los bits del 0 al 8, el dato a escribir vendría dado por:

Data= color| (posicion& 0x7f) ;

Si no se ha realizado el desplazamiento habría que hacer:

Data=((color & 0x1ff) << (31-9)) | (posicion& 0x7f) ;

Y la escritura en la FIFO sería:

PANTALLA_mWriteToFIFO(baseaddr, 0, Data);

La función *'PANTALLA_pintaPixel'* es útil para realizar el coloreado de un rectángulo indicándole su posición (fila y columna) y el color (de la paleta definida) que queremos que tome.

```
void PANTALLA_pintaPixel ( int fila , int columna , Xuint32 color ){
        while ( PANTALLA_mWriteFIFOFull ( XPAR PANTALLA 0 BASEADDR ) ) {}
        PANTALLA_mWriteToFIFO ( XPAR PANTALLA 0 BASEADDR , 0, color + (( Xuint32 )( columna * nfilas + fila ))); }
```

El ejemplo de uso que se muestra a continuación da como resultado una pantalla con cuadros alternos de colores verde claro y verde oscuro.

```
int     main ()
{ int i; int j;
PANTALLA_inicializa (); // pintamos la pantalla de blanco
// Pintamos cuadros verdes y verdes oscuros alternos
for     (i =0; i < ncolumnas; i++){
        for     (j =0; j < nfilas; j++){
                while ( PANTALLA_mWriteFIFOFull ( XPAR PANTALLA 0 BASEADDR )){}
                 if (((i+j) %2) == 0)
                        PANTALLA_pintaPixel(i, j, VERDE);
                else
                        PANTALLA_pintaPixel(i, j, VERDE OSCURO ); } }
return 0;
 }
```

6. Control de una pantalla LCD

En esta práctica aprenderemos a utilizar una pantalla LCD Un LCD (Liquid Crystal Display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora A menudo se utiliza en dispositivos electrónicos, ya que utiliza cantidades muy pequeñas de energía eléctrica

Un LCD (ver *Figura 12*) es un dispositivo de bajo coste capaz de mostrar texto y/o gráficos. Pueden ser clasificados según su funcionamiento en LCD de reflexión y en LCD de absorción, y según su capacidad de mostrar información en 7-segmentos, matrices de puntos alfanuméricos o matrices de puntos gráficos.

Un LCD incorpora: el propio display de cristal líquido, una memoria de patrones que almacena el mapa de bits de los caracteres imprimibles, una memoria de refresco y un controlador que simplifica al máximo la circuitería de interfaz y que genera las señales eléctricas necesarias para el refresco de la información

Un LCD de matriz de puntos alfanuméricos típico está formado por 2 filas de 16 caracteres cada una y cada carácter se muestra sobre una matriz de 5x8 puntos. Internamente almacena el patrón de 208 caracteres diferentes (seleccionables por su código ASCII) y también permite la definición por el usuario del patrón de hasta 8 nuevos caracteres.

Figura 12. Ejemplo de una pantalla LCD real

Además, este dispositivo tiene facilidades para el manejo del cursor y la inicialización del LCD. Para comunicarse utiliza un protocolo tipo *strobe* con señales de selección de operación (lectura/escritura), de selección de registro (instrucción-estado/datos) y paralela de datos (8 bits).

Para profundizar más en la inicialización y los distintos modos de funcionamiento, así como en la temporización necesaria para las lecturas y escrituras, podemos consultar el datasheet de la pantalla LCD utilizada (http://www.dca.fee.unicamp.br/-gudwin/ftp/eaO79/LCDDisplay.pdf). Para la implementación del controlador hardware se ha seguido el diagrama de estados que aparece en el mismo.

La entidad correspondiente al controlador de este periférico se llama lcd_controller, se muestra en el *Cuadro 19* y se encuentra en el fichero lcd_controller.vhd.

A continuación, explicamos las señales de salida referentes al control del display LCD físico y su significado:

- *rw* (1 bit): señal de selección de lectura o escritura
- rs (1 bit): señal de selección de setup o de datos
- e (1 bit): señal de enable
- lcd data (8 bits): señales de datos

```
ENTITY lcd_controller IS
  PORT(
    clk        : IN    STD_LOGIC;  --system clock
    reset      : IN    STD_LOGIC;  --active high reinitializes lcd
    lcd_enable : IN    STD_LOGIC;  --latches data into lcd controller
    lcd_bus    : IN    STD_LOGIC_VECTOR(9 DOWNTO 0);  --data and control signals
    busy       : OUT   STD_LOGIC := '1';  --lcd controller busy/idle feedback
    rw, rs, e  : OUT   STD_LOGIC;  --read/write, setup/data, and enable for lcd
    lcd_data   : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0)); --data signals for lcd
END lcd_controller;
```

Cuadro 19. Entity del controlador de LCD

El resto de señales corresponden a las señales para comunicar *Microblaze* con el controlador hardware. Dicha comunicación se realiza a través de una FIFO en la que *Microblaze* escribirá los comandos que deberán ser ejecutados secuencialmente por el controlador. Este tipo de esquema de conexión se utiliza frecuentemente en aquellos sistemas en los que la velocidad del procesador es superior a la velocidad de respuesta del periférico, para evitar que el procesador pare su ejecución en espera de que termine el comando anterior.

Dentro de *usr_logic*, cada vez que se detecta que se ha escrito un nuevo dato en la FIFO, se activa la señal lcd_enable y se pasa el dato por lcd_bus.

El dato en lcd_bus tiene el siguiente formato:

- lcd_bus(9) es la señal rs de selección de setup o datos
- lcd_bus(8) es la señal de lectura/escritura
- lcd_bus(7 DOWNTO 0) son los 8 bits de datos.

Estos datos se pasan por la FIFO en los bits más significativos, es decir,

```
lcd_bus(0) <= WFIFO2IP_Data(31);
lcd_bus(1) <= WFIFO2IP_Data(30);
lcd_bus(2) <= WFIFO2IP_Data(29);
lcd_bus(3) <= WFIFO2IP_Data(28);
lcd_bus(4) <= WFIFO2IP_Data(27);
lcd_bus(5) <= WFIFO2IP_Data(26);
lcd_bus(6) <= WFIFO2IP_Data(25);
lcd_bus(7) <= WFIFO2IP_Data(24);
lcd_bus(8) <= WFIFO2IP_Data(23);
lcd_bus(9) <= WFIFO2IP_Data(22);
```

6.1. Conexión al sistema

En user_logic y en el top del periférico hay que añadir los puertos de entrada/salida correspondientes a las salidas externas del controlador de lcd que como hemos visto en el Cuadro 19 son *rw*, *rs*, y *lcd_data*. El resultado se muestra en el *Cuadro 20*.

Por otro lado, hay que dar las restricciones correspondientes de la ubicación de los puertos en el fichero '**.ucf*' del top del sistema. En el caso de la placa que vamos a utilizar en el laboratorio su situación se muestra en *Cuadro 21*. El nombre de los puertos externos en el archivo top del sistema dependerá de la

elección que haya realizado el diseñador. Lo que se muestra en dicho cuadro es el diseño proporcionado en el laboratorio.

```
port
(
 -- ADD USER PORTS BELOW THIS LINE ------------------
 --USER ports added here
        rw                      : OUT   STD_LOGIC;  --read/write for lcd
        rs                      : OUT   STD_LOGIC;  --setup/data for lcd
        e                       : OUT   STD_LOGIC;  --enable for lcd
        lcd_data                : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0);

 -- ADD USER PORTS ABOVE THIS LINE ------------------

 -- DO NOT EDIT BELOW THIS LINE ---------------------
 -- Bus protocol ports, do not add to or delete
```

Cuadro 20. Adición de puertos del LCD en usr_logic

```
#NET lcd_rs LOC=P12;
#NET lcd_rw LOC=J1;  #Este no se usa en la placa nueva (sólo se escribe)
#NET lcd_e LOC=H1;
#NET lcd_data<0> LOC=H3;
#NET lcd_data<1> LOC=G2;
#NET lcd_data<2> LOC=K15;
#NET lcd_data<3> LOC=K16;
#NET lcd_data<4> LOC=F15;
#NET lcd_data<5> LOC=E2;
#NET lcd_data<6> LOC=E1;
#NET lcd_data<7> LOC=F3;
```

Cuadro 21. Restricciones del lcd en *.ucf

6.2. Control software

Para llevar a cabo un control adecuado de la pantalla LCD se ha desarrollado el driver que implementa las siguientes funciones:

- Inicializar: La función 'LCD_inicializa' resetea el display borrando todos los posibles caracteres que pudiese tener y sitúa el cursor de escritura en el primer carácter de la primera fila.
- Escribir caracteres y mandar comandos: La función 'LCD_enviarCMD', una vez inicializada la pantalla, es útil para ordenar la escritura de un carácter en la posición del cursor o para direccionar la posición del mismo. Para llevar a cabo el control del display se definen los siguientes comandos:

```
void LCD_inicializa () {
LCD_mResetWriteFIFO ( XPAR LCD 0 BASEADDR );
 LCD_enviarCMD ( CLEAR DISPLAY CMD );
LCD_enviarCMD ( RETURN HOME CMD ); // moverse al comienzo de la pantalla LCD
enviarCMD ( WRITE CMD ); // primera escritura
}

void LCD_enviarCMD ( Xuint32 cmd ){
// Comprobamos que la FIFO no esté llena
while ( LCD_mWriteFIFOFull ( XPAR LCD 0 BASEADDR )){}
// Escribimos el comando en la FIFO
LCD_mWriteToFIFO ( XPAR LCD 0 BASEADDR , 0, cmd ); }
```

'CLEAR_DISPLAY_CMD': borra el display,
'RETURN_HOME_CMD': hace que el cursor se mueva al comienzo de la pantalla,
'WRITE_CMD': ordena la escritura de un caracter,
'FIRST_ROW ': direcciona el cursor en el primer carácter de la primera fila
'SECOND_ROW': direcciona el cursor en el primer carácter de la segunda fila

El ejemplo de uso que se muestra a continuación, da como resultado la escritura en la primera fila de los caracteres ¡HOLA MUNDO! y de los caracteres 'abcdefg O123456 'en la segunda fila. El comando 'WRITE_CMD' ordena la escritura de un carácter, pero no establece qué carácter debe escribir (los dos bytes menos *significativo*s valen cero). Por ello, para indicar la escritura de un carácter se lo sumamos al comando (OR lógica).

```
# define CLEAR_DISPLAY_CMD 0 x00000001
# define RETURN_HOME_CMD 0 x00000002
# define WRITE_CMD 0 x00000200
# define FIRST_ROW 0 x00000080
# define SECOND_ROW 0 x000000C0
int main ()
{
xil_printf (" Practica LCD\r\n");
LCD_inicializa ();
LCD_enviarCMD ( WRITE_CMD + 'H' );
LCD_enviarCMD ( WRITE_CMD + 'O' );
LCD_enviarCMD ( WRITE_CMD + 'L' );
LCD_enviarCMD ( WRITE_CMD + 'A' );
LCD_enviarCMD ( WRITE_CMD + ' ' );
LCD_enviarCMD ( WRITE_CMD + 'M' );
LCD_enviarCMD ( WRITE_CMD + 'U' );
LCD_enviarCMD ( WRITE_CMD + 'N' );
LCD_enviarCMD ( WRITE_CMD + 'D' );
LCD_enviarCMD ( WRITE_CMD + 'O' );
LCD_enviarCMD ( WRITE_CMD + '!' );
LCD_enviarCMD ( WRITE_CMD + ' ' );
LCD_enviarCMD ( WRITE_CMD + ':' );
LCD_enviarCMD ( WRITE_CMD + ')' );
LCD_enviarCMD ( SECOND_ROW ); // cambio fila
LCD_enviarCMD ( WRITE_CMD + 'a' );
LCD_enviarCMD ( WRITE_CMD + 'b' );
LCD_enviarCMD ( WRITE_CMD + 'c' );
LCD_enviarCMD ( WRITE_CMD + 'd' );
LCD_enviarCMD ( WRITE_CMD + 'e' );
LCD_enviarCMD ( WRITE_CMD + 'f' );
LCD_enviarCMD ( WRITE_CMD + 'g' );
LCD_enviarCMD ( WRITE_CMD + ' ' );
LCD_enviarCMD ( WRITE_CMD + ' ' );
LCD_enviarCMD ( WRITE_CMD + '0' );
LCD_enviarCMD ( WRITE_CMD + '1' );
LCD_enviarCMD ( WRITE_CMD + '2' );
```

```
LCD_enviarCMD ( WRITE_CMD + '3' );
LCD_enviarCMD ( WRITE_CMD + '4' );
LCD_enviarCMD ( WRITE_CMD + '5' );
LCD_enviarCMD ( WRITE_CMD + '6' );
return 0;
}
```

7. Control de un zumbador

Un zumbador es un transductor electroacústico que produce un sonido o zumbido continuo de un mismo tono. Sirve como mecanismo de señalización o aviso y son utilizados en múltiples sistemas como en automóviles o en electrodomésticos.
El driver de un zumbador debe implementar las funcionalidades básicas que nos sirvan para utilizar un zumbador, tales como una función para indicarle que suene y otra para silenciarlo. Desde el punto de vista electrónico, el zumbador (buzzer o piezo speaker en inglés), es un elemento capaz de transformar la electricidad en sonido. El corazón de los buzzer piezoeléctricos es un simple disco piezoeléctrico, que consiste de una placa cerámica con una capa metálica. Si el disco es controlado por un circuito oscilante externo se habla de un transductor piezoeléctrico. Si el circuito oscilador está incluido en la carcasa, se le denomina zumbador piezoeléctrico (que es el que utilizaremos en la práctica).
Los generadores de sonidos piezoeléctricos son dispositivos aptos para el diseño de alarmas y controles acústicos de estrecho rango de frecuencia, por ejemplo, en aparatos domésticos y de medicina
Primero vamos a fijarnos en el zumbador que se muestra en la *Figura 13*. Tiene dos conectores que revelan que los piezos tienen polaridad, y la pegatina superior (también suele indicarse con un grabado en la carcasa de recubrimiento) indica precisamente cómo conectar nuestro dispositivo a la placa Su símbolo electrónico se presenta en la *Figura 14*.

Figura 13. Ejemplo de zumbador real y símbolo electrónico

Figura 14. Símbolo del zumbador

Su funcionamiento se puede entender a través del circuito de la *Figura 15*. En realidad, un zumbador es un pequeño altavoz conectado a un oscilador de frecuencia fija, por lo que cada vez que se pulse el interruptor, se cerrará el circuito y el zumbador quedará polarizado (con la ayuda de una batería) lo que producirá sonido en esa determinada frecuencia. Mientras no se pulse el interruptor, el circuito permanecerá abierto y no se producirá sonido

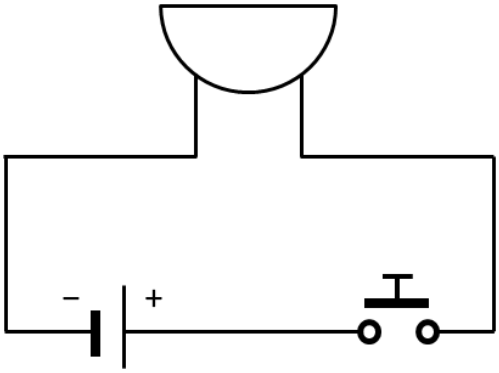


Figura 15. Uso de un zumbador en un circuito sencillo

La *Figura 16* muestra el circuito implementado en las placas del laboratorio. En nuestro caso concreto, dejamos fijo el valor de polarización Vcc, por lo que si queremos que suene deberemos poner el pin que sale de la FPGA con un valor de '0' lógico (GND) y si queremos que se silencie pondremos un valor de '1' lógico (Vcc).

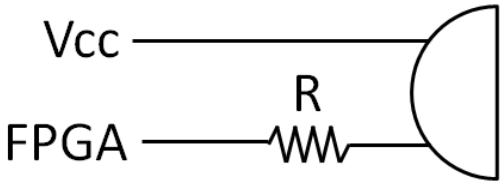


Figura 16. Conexión del zumbador a la placa del laboratorio

7.1. Conexión al sistema

Dada la sencillez del circuito del zumbador, lo único que necesitamos es un puerto de salida que se ponga a ‘0’ cuando queramos que el zumbador suene. Para ello vamos a utilizar un GPIO cualquiera. La salida del puerto correspondiente debe conectarse mediante un cable a la entrada del zumbador. En el fichero de restricciones habrá que indicar el nombre *del* puerto asignado a la salida del zumbador.

7.2. Control software

Para llevar a cabo un control adecuado del zumbador se ha desarrollado el driver que implementa las siguientes funciones:

- 'ZUMBADOR_inicializa' : Esta función configura el GPIO asociado al zumbador como salida.
- 'ZUMBADOR_calla': Esta función silencia el zumbador escribiendo un ‘1’ en el GPIO correspondiente.
- 'ZUMBADOR_suena' hace que el zumbador emita sonido en la frecuencia preestablecida en su fabricación, escribiendo un ‘0’ en el GPIO correspondiente.

A continuación, se muestran estas 3 funciones:

```
void ZUMBADOR_inicializa ( XGpio * Gpio zumbador , Xuint32 zumbador )
{ Xuint32 status ;
// Configuración de la GPIO para el zumbador de la placa de expansión
status = XGpio Initialize ( Gpio zumbador , zumbador );
if ( status !=  XST_SUCCESS )
```

```
xil_printf("Error en la inicializacion\r\n");
else {
XGpio_SetDataDirection ( Gpio zumbador , 1, 0 x00 );
Xil_printf("Inicializado con exito\r\n");
} }

void ZUMBADOR_calla ( XGpio * Gpio zumbador ) {
XGpio DiscreteWrite(Gpio zumbador , 1, (u8)1);
}
void ZUMBADOR_suena ( XGpio * Gpio zumbador ) {
 XGpio DiscreteWrite(Gpio zumbador , 1, (u8)0);
 }
```

El ejemplo de uso que se muestra a continuación produce un sonido intermitente.

```
int main (void) {
xil_printf("Practica zumbador\r\n");
unsigned int i;
// La instancia del zumbador usada para la comunicación
// con el zumbador físico
XGpio Gpio zumbador ;
// Los Device IDs del zumbador para la comunicación
// con el zumbador físico
Xuint32 zumbador = XPAR_ZUMBADOR_DEVICE_ID ;
ZUMBADOR_inicializa (& Gpio zumbador , zumbador );
while (1){
            ZUMBADOR_suena (& Gpio zumbador );
           xil_printf("Suena\r\n");
           for (i = 0; i < 0 x00070000 ; i ++){} // retardo
           ZUMBADOR_calla (& Gpio zumbador );
           xil_printf("Calla\r\n");
           for (i = 0; i < 0 x00070000 ; i ++){} // retardo
}
return 0; }
```

8. Control de un altavoz

Un altavoz es un transductor electroacústico utilizado para la reproducción de sonido. La transducción sigue un doble procedimiento: eléctrico-mecánicoacústico. En la primera etapa convierte las ondas eléctricas en energía mecánica, y en la segunda convierte la energía mecánica en ondas de frecuencia acústica.

El controlador de un altavoz debe implementar las funcionalidades básicas que nos sirvan para utilizar un altavoz, tales como una función para indicarle que suene en una determinada frecuencia (por ejemplo, la correspondiente a notas musicales) y otra para silenciarlo.

Un altavoz (ver *Figura 17*) es un transductor capaz de generar una onda sonora análoga (en frecuencia y amplitud) a una señal eléctrica dada. Se compone de una membrana elástica unida a una bobina móvil que se monta dentro del campo magnético de un imán permanente. La fuerza de atracción entre la bobina y el imán es función de la intensidad y el sentido de la corriente que circule por la bobina. Cambios en la corriente, provocan movimientos en la bobina que se traducen en vibraciones en la membrana. Si estas vibraciones tienen la frecuencia adecuada, se escucha un sonido. Su símbolo electrónico se presenta en la *Figura 18*.

Figura 17. Altavoz

Dado que un sistema digital puede generar una señal digital periódica a través de uno de sus pines, puede producir sonidos si dicho pin se conecta al altavoz. Si se desea generar un sonido complejo, se debe generar por separado cada uno de sus armónicos y sumarlos para generar una única señal. La *Figura 19* muestra las frecuencias que corresponden a cada una de las notas musicales según la octava.

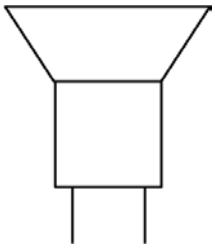


Figura 18. Símbolo del altavoz

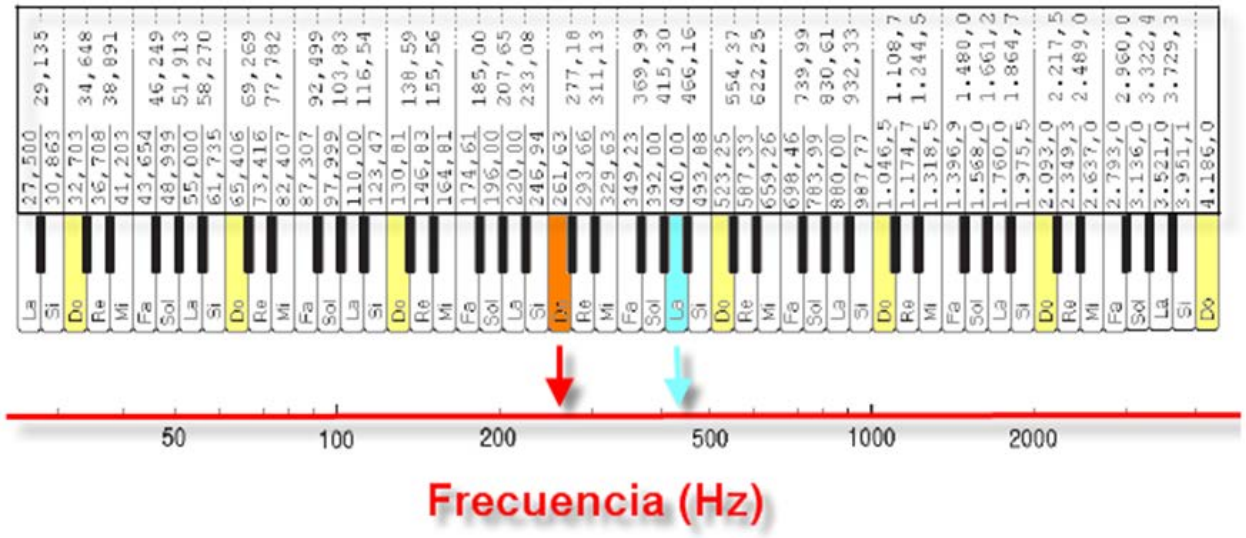


Figura 19. Frecuencia de distintas notas musicales

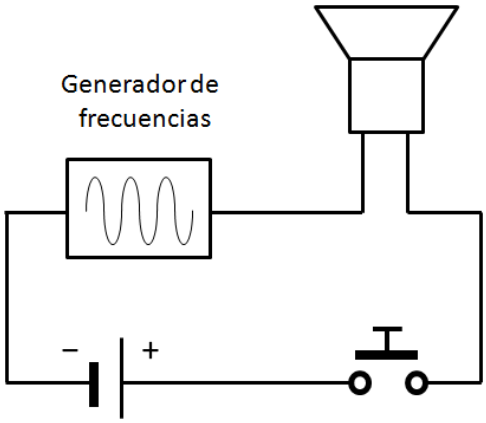


Figura 20. Ejemplo de uso de un altavoz en un circuito sencillo

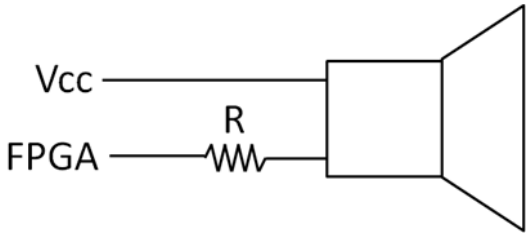


Figura 21. Conexionado del altavoz a la placa de FPGA

Su funcionamiento es muy sencillo y se puede explicar a través del circuito de la *Figura 20*. Cada vez que se pulse el interruptor, se cerrará el circuito y el generador de frecuencias quedará polarizado (con la ayuda de una batería) lo que producirá una determinada frecuencia y que el altavoz produzca la nota correspondiente a dicha frecuencia. Mientras no se pulse el interruptor, el circuito permanecerá abierto y no se producirá sonido.

Finalmente, la *Figura 21* muestra el circuito implementado en las placas del laboratorio. En nuestro caso concreto, dejamos fijo el valor de polarización Vcc, por lo que si queremos que suene deberemos generar una frecuencia audible en el pin que sale de la FPGA.

El proceso encargado de generar una onda con una frecuencia fijada en el registro *slv_reg0* se muestra en *Cuadro 22*, y debe incluirse en *usr_logic*.

```
--------------------------
-- GENERADOR DEL SONIDO --
--------------------------
altavoz_p: PROCESS(Bus2IP_Clk, Bus2IP_Reset)
BEGIN
  IF (Bus2IP_Reset='1')THEN
          q <= '0';
         cs <= (OTHERS=>'0');
  ELSIF (Bus2IP_Clk'event and Bus2IP_Clk='1') THEN
                    IF (slv_write_ack = '1') THEN
                              q <= '0';
                              cs <= (OTHERS=>'0');
                    ELSIF (slv_reg0 = cs) THEN
                              cs <= (OTHERS=>'0');
                              q <= not q;
                    ELSE
                              cs <= cs + 1;
                              q <= q;
                    END IF;
   END IF;
  END PROCESS;
sonido <= q;
```

Cuadro 22. Generación de señal de frecuencia fija

8.1. Conexión al sistema

Como se muestra en el *Cuadro 23*, tanto en *usr_logic* como en el top del controlador es necesario añadir el puerto correspondiente a la salida de sonido, que en el ejemplo del laboratorio hemos llamado *sonido*. Por dicha salida hay que generar una señal de la frecuencia deseada.

La comunicación entre *Microblaze* y el controlador hardware se realiza a través del registro *slv_reg0*, en la que *Microblaze* escribirá el valor en ciclos del sistema del semiperiodo de la onda que debe generar el controlador. Finalmente, añadimos la asignación de los puertos externos a los pines de la FPGA en el fichero '‘*.*ucf*". En principio puede usarse cualquier puerto, que físicamente deberá conectarse al altavoz.

```
entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                : integer             := 32;
    C_NUM_REG                   : integer             := 1
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
            sonido                          : out  std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
```

Cuadro 23. Puertos de usr_logic para el controlador de altavoz

8.2 Control software

Para llevar a cabo un control adecuado del altavoz se ha desarrollado el driver que implementa las siguientes funciones:

- ***My_delay***: genera un retardo. Es útil para establecer un tiempo de espera antes de ejecutar la siguiente instrucción de código.
- ***ALTAVOZ_calla***: silencia el altavoz.
- ***ALTAVOZ_suena***: se utiliza para fijar la frecuencia (en realidad el número de ciclos del reloj del sistema del semiperiodo de la onda) de la señal de salida, escribiendo en el registro del controlador la nota despalzada a la izda tantas posiciones como valor tenga la octava que queremos que suene.

  mWriteSlaveReg0 (XPAR_ALTAVOZ_0 _BASEADDR , 0, nota << octava)

Para indicar la nota concreta que debe sonar se definen las notas y octava del *Cuadro 24*.

```
# define SILENCIO 0 x00000000
# define DO 0 x00000BAA
# define RE 0 x00000A64
# define MI 0 x00000942
# define FA 0 x000008BD
# define SOL 0 x000007C9
# define LA 0 x000006F0
# define SI 0 x0000062E
# define OCTAVA 3
```

Cuadro 24. Definición del semiperiodo y escala de algunas notas musicales

A continuación se presenta el código de las tres funciones definidas:

```
void my_delay ( int delay ){
        int i, j;
        for (i =0; i<delay; i=i+1)
        for (j =0; j <500; j=j +1){} }

void ALTAVOZ_calla(){ ALTAVOZ mWriteSlaveReg0 ( XPAR_ALTAVOZ_0_BASEADDR , 0, SILENCIO ); }

void ALTAVOZ_suena ( Xuint32 nota , Xuint32 octava ) {
        ALTAVOZ mWriteSlaveReg0 ( XPAR_ALTAVOZ_0_BASEADDR , 0, nota << octava );
 }
```

El ejemplo de uso que se muestra a continuación produce una escala musical ascendente y un silencio.

```
int main () {
ALTAVOZ_suena (DO , OCTAVA );
my delay (300);  print("DO\n\r");
ALTAVOZ_suena (RE , OCTAVA );
my delay (300);  print("RE\n\r");
ALTAVOZ_suena (MI , OCTAVA );
my delay (300);  print("MI\n\r"); A
LTAVOZ suena (FA , OCTAVA );
my delay (300);  print("FA\n\r");
ALTAVOZ suena(SOL, OCTAVA);
my delay (300);  print("SOL\n\r");
ALTAVOZ_suena (LA , OCTAVA );
my delay (300); print("LA\n\r");
ALTAVOZ_suena (SI , OCTAVA );
my delay (300); print("SI\n\r");
ALTAVOZ suena(DO, (OCTAVA -1));
my delay (300);  print("DO\n\r");
ALTAVOZ_calla();
my delay (300);  print(" --\n\r");
return 0; }
```

9. Control de un LED *RGB* a través de un PWM

Un modulador de anchos de pulso (o PWM, de sus siglas en inglés *Pulse Width Modulation*) es un tipo de circuito muy utilizado para el control de diversos sistemas digitales: servo-motores, elementos termo-eléctricos, LED. En particular es muy útil para controlar la intensidad de encendido de las 3 componentes lumínicas de un LED *RGB* (Rojo (R), Verde (G) y Azul (B)).
Pulse Width Modulation es una técnica de modulación de ondas cuadradas que determina la anchura de los pulsos transmitidos por éstas. En otras palabras, permite decidir qué porcentaje del período de oscilación de la señal, permanece a 0 y qué porcentaje permanece a 1. A este porcentaje se le denomina comúnmente *duty cycle*, y es un valor comprendido entre 0 y 1. Así, si el *duty cycle* es 0,5, la señal permanece a 1 el 50% del tiempo de ciclo y a 0 el 50% del tiempo restante. Por el contrario, si fuese 0,25, permanecería a 1 el 25 % del tiempo de ciclo y a 0 el 75 % del tiempo restante. Esto se muestra gráficamente en la *Figura 23*.

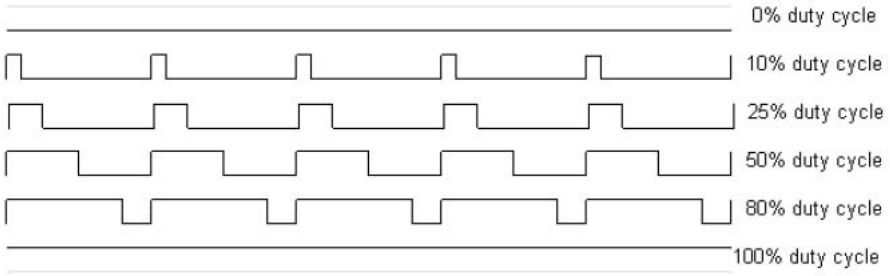


Figura 23. Funcionamiento de un modulador de ancho de pulsos

Una señal cuadrada modulada a través de un PWM se puede utilizar para controlar la intensidad de encendido de un LED detectada por el ojo humano. Si la frecuencia de esta señal es lo suficientemente alta, el ojo no detectará que el LED realmente está parpadeando, con lo que únicamente apreciará que éste se enciende con más o menos intensidad.

Un LED *RGB* es un tipo de LED capaz de iluminarse en las 3 componentes lumínicas consideradas básicas: rojo, verde y azul. Los LED que utilizaremos en esta práctica tienen 4 patillas: una de ellas es el cátodo común (que debe conectarse a tierra), mientras que las 3 restantes determinan el encendido de las 3 componentes lumínicas del LED, tal y como se muestra en la *Figura 24*. Estas 3 patillas únicamente admiten 2 valores de entrada: encendido ('1') y apagado ('0').

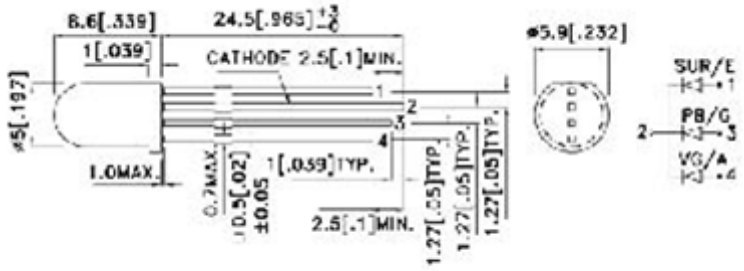


Figura 24. Esquema de un led RGB

En la *Figura 25* se muestran los Led *RGB* de la placa de expansión del laboratorio.

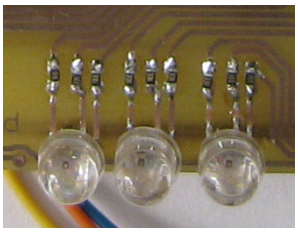


Figura 25. Led RGB de la placa del laboratorio

La comunicación entre *Microblaze* y el controlador del led *RGB* se debe hacer a través de 3 registros de 8 bits accesibles por el usuario: *slv_reg0*, *slv_reg1* y *slv_reg2*, uno para cada color. En dichos registros *Microblaze* escribe el valor de la intensidad para cada color, entre 0 y 255 (0: mínima intensidad; 255: máxima intensidad). En el *Cuadro 25* se proporciona el código vhdl para generar una onda cuadrada con un periodo de 256 ciclos (*n*=8) que está a 1 el número de ciclos indicado en el registro *slv_reg0*, que en este ejemplo es el registro de intensidad del color rojo.

```
ref_red <= unsigned (slv_reg0);

        PWM1: process (Bus2IP_Reset, Bus2IP_Clk) begin
                if (Bus2IP_Reset = '1') then
                        cnt_red <= (others => '0');
                elsif rising_edge(Bus2IP_Clk) then
                        if (cnt_red = 2**n-1) then
                                cnt_red <= (others => '0');
                        else
                                cnt_red <= cnt_red + 1;
                        end if;
                end if;
        end process;

        output_red <= '1' when (cnt_red < ref_red) else '0';
        red <= output_red;
```

Cuadro 25. PWM para un led RGB

Su implementación es muy sencilla: consiste en un contador módulo 256, y un módulo que compara la salida del contador con el valor de referencia *ref_red*, el cual está guardado en el registro *slv_reg0* (registro para el encendido del color rojo). Si el valor del contador es menor que el valor de referencia, la salida *red* es 1, en caso contrario, *red* será 0. De este modo, cuanto más alto sea el valor de referencia, mayor tiempo estará esta señal a 1

9.1. Conexión al sistema

En el fichero *usr_logic* y el del top del periférico hay que añadir los 3 puertos de salida (ver *Cuadro 26*), uno para cada color (*red, green y blue*). Además en *usr_logic* hay que añadir 3 *process* similares al del *Cuadro 25*.

```
port
(
  -- ADD USER PORTS BELOW THIS LINE ------------------
  --USER ports added here
  -- ADD USER PORTS ABOVE THIS LINE ------------------
          red: out std_logic;
          green: out std_logic;
          blue: out std_logic;


  -- DO NOT EDIT BELOW THIS LINE ---------------------
  -- Bus protocol ports, do not add to or delete
```

Cuadro 26. Puertos del controlador RGB

Por último en el fichero de restricciones '**.ucf*' hay que añadir los 3 puertos que para la placa de expansión del laboratorio se muestran en el *Cuadro 27*.

```
####LEDS COLORES

#NET RED_pin LOC=P12;
#NET GREEN_pin LOC=H1;
#NET BLUE_pin LOC=J1;

#NET USER1_RED_pin LOC=E15;
#NET USER1_GREEN_pin LOC=D16;
#NET USER1_BLUE_pin LOC=J16;

#NET USER2_RED_pin LOC=J14;
#NET USER2_GREEN_pin LOC=N15;
#NET USER2_BLUE_pin LOC=G16;
```

Cuadro 27. Fichero *.ucf para el controlador de led RGB

9.2. Control software

El control del periférico a través de Microblaze es tan sencillo como escribir en cada uno de los 3 registros de control el valor de la intensidad correspondiente a los 3 colores. Por ejemplo, para el color rojo, este valor se escribirá en el registro *slv_reg0* invocando a la función:

LED_RGB_mWriteSlaveReg0 (*XPAR_LED_RGB_0_BASEADDR* , 0, intensidadR);

donde XPAR_LED_*RGB*_O_BASEADDR es la dirección base de acceso al periférico e *intensidadR* es el valor de la intensidad que se escribirá en el registro. El nombre de esta función dependerá del nombre que se haya dado al periférico y puede consultarse en el fichero *.h generado al crear dicho periférico. Análogamente, las funciones *LED_RGB_mWriteSlaveReg1* y *LED_RGB mWriteSlaveReg2* escriben valores en los registros accesibles por el usuario 1 y 2, respectivamente.

10. Control de un par de dispositivos emisor-receptor de infrarrojos

El dispositivo emisor de infrarrojos utilizado en la tarjeta de expansión del laboratorio es un tipo especial de LED que emite luz infrarroja no visible directamente a través del ojo humano. Tiene dos patillas: una de tierra y otra que se utiliza para encenderlo ('1') y apagarlo ('0') (*Figura 26*).

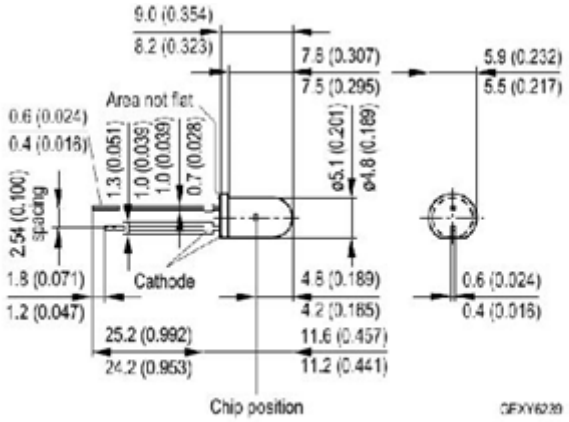


Figura 26. Modelo de chip emisor de infrarrojos de la placa de expansión

El dispositivo receptor de infrarrojos que utilizaremos en esta práctica (*Figura 27*) es el "ojo artificial" que utilizaremos para capturar los códigos enviados por el emisor. Tal y como se aprecia en la figura, tiene 3 patillas: la número 1 proporciona la señal infrarroja recibida por el dispositivo, la número 2 se debe conectar a tierra, y finalmente la número 3 se debe conectar a alimentación. En este caso, podemos alimentar el dispositivo con 5V o con 3,3V. La FPGA Spartan 3 de laboratorio proporciona ambas tensiones. Puede observarse el modelo de emisor/receptor utilizado en dicha placa en la *Figura 28* .

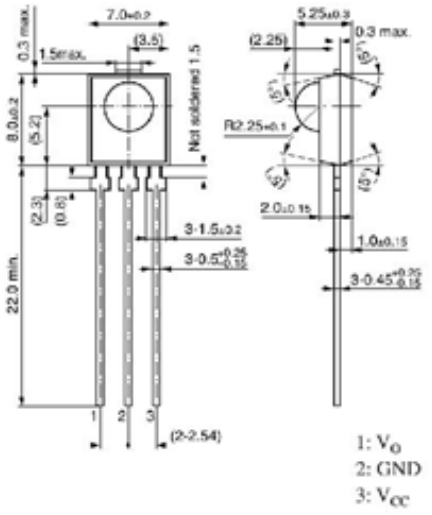


Figura 27. Esquema del dispositivo receptor de infrarrojos

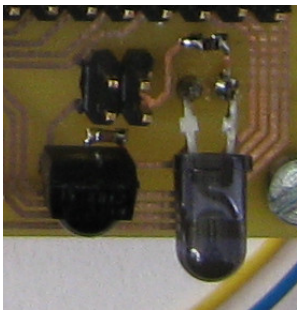


Figura 28. Emisor/receptor de infrarrojos de la placa de expansión

Es muy importante tener en cuenta que este tipo de dispositivos no son capaces de capturar cualquier señal de infrarrojos, sino sólo aquéllas que contengan una determinada frecuencia portadora. En este caso, el par emisor-receptor de infrarrojos que utilizaremos trabaja con una frecuencia portadora de 38 KHz (*Figura 29*).

En general, la frecuencia portadora de una señal es una frecuencia mucho más alta que la de la propia señal que sirve para modular ésta y enviarla. Tal tipo de modulación de señal se utiliza para desplazar su contenido espectral en frecuencia, ocupando así un cierto ancho de banda alrededor de la frecuencia de la onda portadora. Esto permite enviar varias ondas con diferentes portadoras simultáneamente (multiplexadas) y así utilizar más eficientemente el espectro disponible de frecuencias. La *Figura 29* ilustra esta idea con un ejemplo. En este caso, la onda que se muestra tiene una frecuencia de 9 KHz, y se envía modulada con una frecuencia portadora de 38 KHz (la cual es 4 veces mayor). Por tanto, esta onda será la suma entre la portadora de 38 KHz y la frecuencia original de 9 KHz de la señal.

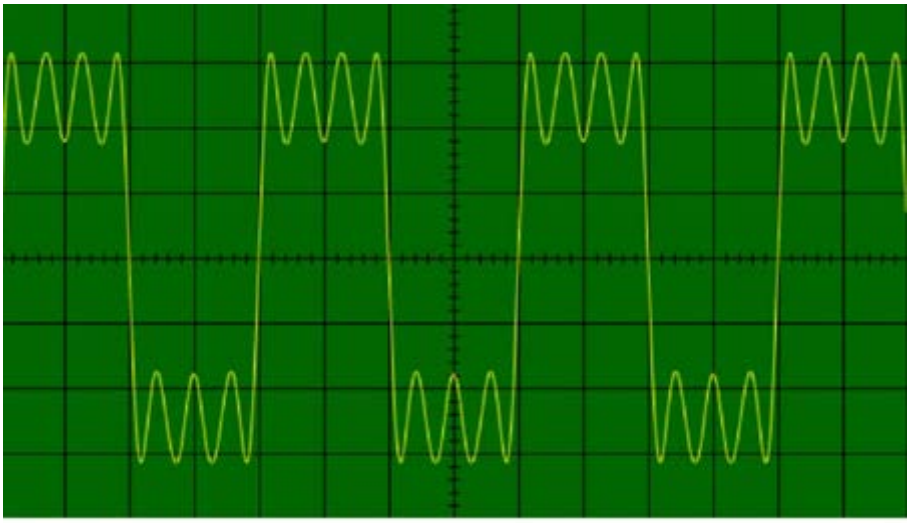


Figura 29. Portadora de 38Khz y onda 9KHz

Para la generación de una señal de 38KHz se utilizará el código que se presenta en el *Cuadro 28*.

```
-- Divisor de frecuencias que obtiene en Reloj_38KHz una señal de reloj de 38 KHz (frecuencia portadora)
 divisor_38KHz:
 PROCESS( Bus2IP_Reset, Bus2IP_Clk )
 BEGIN
  IF (Bus2IP_Reset='1') THEN
   cuenta1<= (OTHERS=>'0');
  ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
                IF (cuenta1="00000000000000000010100100011") THEN -- 100 MHz / 38 KHz / 2 = 1315
                        Reloj_38KHz <= not Reloj_38KHz;
                         cuenta1<= (OTHERS=>'0');
                 ELSE
                        cuenta1 <= cuenta1 + '1';
   END IF;
  END IF;
 END PROCESS divisor_38KHz;
```

Cuadro 28. Generación de onda de 38KHz

Para generar la frecuencia de 1Hz se utilizará el código del *Cuadro 29*.

```
-- Divisor de frecuencias que obtiene en Reloj_1Hz una señal de reloj de 1 Hz (frecuencia de la señal)
 divisor_1Hz:
 PROCESS ( Bus2IP_Reset, Bus2IP_Clk )
 BEGIN
  IF (Bus2IP_Reset='1') THEN
   cuenta2<= (OTHERS=>'0');
  ELSIF (Bus2IP_Clk'EVENT AND Bus2IP_Clk='1') THEN
   IF (cuenta2 = "01011111010111100001000000000") THEN  -- 1 Hz
        Reloj_1Hz <= not Reloj_1Hz;
        cuenta2<= (OTHERS=>'0');
   ELSE
        cuenta2 <= cuenta2 + '1';
   END IF;
  END IF;
 END PROCESS divisor_1Hz;
```

Cuadro 29. Generación de onda de 1Hz

La comunicación entre Microblaze y el controlador del periférico se realiza a través de un registro de 8 bits (o de 32 según el diseño) accesible por el usuario: *slv_reg0*. Escribiendo en él un valor distinto de 0, el periférico comenzará a enviar por el emisor de infrarrojos una señal de frecuencia 1 Hz con una frecuencia portadora de 38 KHz a través de su salida output. Para ello utilizaremos el código que se presenta en el *Cuadro 30*.

```
  active_ER <= '0' when slv_reg0 = ( OTHERS=>'0') else '1';
 code_infrared <= Reloj_38KHz AND Reloj_1Hz AND active_ER;
```

Cuadro 30. Generación de la señal del emisor de infrarrojos

10.1. Conexión al sistema

Para el desarrollo de esta práctica, se proporciona el código para controlar el envío de señales infrarrojas con nuestro dispositivo emisor directamente en el archivo *usr_logic*.vhd. Será necesario añadir el puerto de salida (del emisor) en el top del sistema. Además, hay que conectar la entrada del receptor a un led utilizando un GPIO.
En el fichero "*.ucf" hay que colocar la situación de dicho puerto, que para la placa de expansión del laboratorio es:

#NET code_infrared_pin LOC=D16;
#NET in_infrared_pin LOC=J14;

Leyendo esta salida con el receptor y conectando la patilla de recepción a un LED visible, veremos que éste parpadea a una frecuencia de 1 Hz (el LED está 0,5 segundos apagado y 0,5 segundos encendido). En la placa de expansión del laboratorio esta conexión ya está hecha, pues el receptor está conectado a la patilla D16, que a su vez está conectada a la patilla *green* del led de colores 1.

10.2. Control software

El control software del controlador de infrarrojos se realiza a través del registro *slv_reg0*. Escribiendo en él un valor distinto de 0 el periférico comienza a emitir. La función que podemos utilizar es:

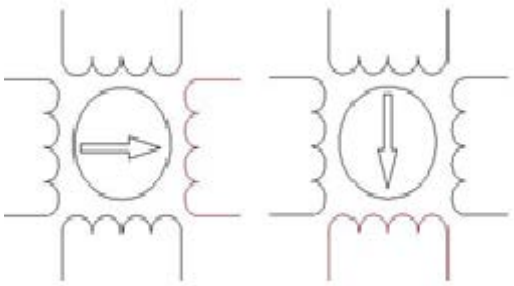
ER_INFRARED_mWriteSlaveReg0 (XPAR_ER_INFRARED_0_BASEADDR, 0,OxFF)

donde XPAR_ER_INFRARED_0_BASEADDR es la dirección base de acceso al periférico ER_Infrared, y 0xFF es el valor que se escribirá en el registro para indicar a este periférico que el LED infrarrojo debe comenzar a enviar datos. Para apagarlo debe escribirse un cero. El nombre de la función puede variar dependiendo de cómo se haya llamado al periférico. Puede consultarse en el fichero *.h correspondiente.

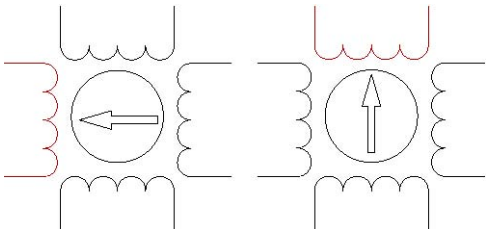
11. Control de un motor paso a paso

En esta práctica aprenderemos a controlar un motor de los llamados *"paso a paso"*. Este tipo de motores son muy usados en robótica. Este motor podría utilizarse, por ejemplo, para controlar un panel solar. Dicho sistema debe mantener el panel orientado en todo momento hacia el sol, de manera que la luz incida de forma perpendicular al panel y maximizar así la producción de energía. Para realizar este proyecto habría que utilizar, además del motor paso a paso, un conversor analógico-digital y sensores de luz, que veremos en los siguientes capítulos.

Los motores "*paso a paso*" son bastante diferentes a los motores de corriente continua. Están formados por un rotor, que generalmente es un imán permanente, y un estator, formado por varias bobinas. Cuando la corriente circula por una de las bobinas del estator, se crea un campo magnético que hace girar el imán del rotor hasta alinearse con aquél. Activando las bobinas del estator en la secuencia adecuada, puede conseguirse que el rotor gire en cada momento un ángulo determinado.
Una característica que hace a este tipo de motores muy adecuado para nuestro proyecto es que, si se mantiene la alimentación en una de las bobinas, el rotor permanece anclado en su posición y ejerce una fuerza que impide que pueda rotar libremente. De esta forma podemos mantener fácilmente nuestro panel orientado en la posición deseada.
La secuencia de activación de las bobinas depende de la construcción del motor y la debe proporcionar el fabricante en la hoja de características. Además, dicha secuencia depende también de si el motor es bipolar o unipolar. Podemos ver un ejemplo en la *Figura 30*, desde la (a) hasta la (d).



(a) (b)



(c) (d)

Figura 30. Funcionamiento de un motor paso a paso

Un motor paso a paso bipolar consta de dos bobinas en el estator que irán adoptando distinta polaridad para provocar cada paso de avance del motor. La dificultad de control de este tipo de motores radica en que cada bobina debe invertir su polaridad según el paso en el que nos encontremos. Para ello se necesita un circuito electrónico denominado puente H' y cuya configuración se puede ver en la *Figura 31*. Los

diodos sirven de protección frente a la fuerza contra electromotriz que genera la bobina al cambiar su polaridad.

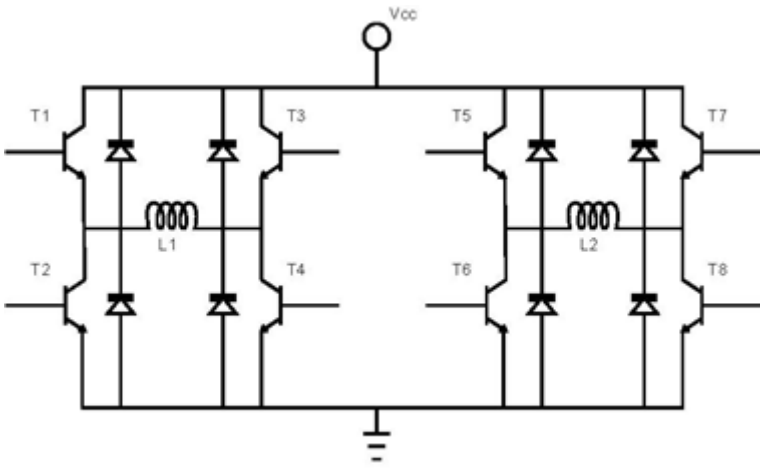
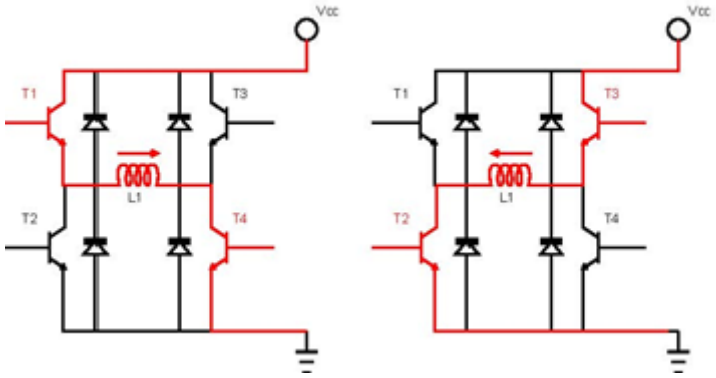


Figura 31. Configuración del puente-H

Por ejemplo, activando simultáneamente los transistores T1 y T4, la intensidad circula por la bobina L1 en el sentido indicado en la *Figura 32* (a), mientras que activando T3 y T2 circula como en *Figura 32* (b), consiguiendo la polaridad inversa del campo magnético. Es importante asegurar que nunca se activen simultáneamente, por ejemplo, los transistores T1 y T2, pues en ese caso tendríamos un camino directo desde Vcc hacia tierra y la elevada intensidad que circularía destruiría ambos transistores. Resumiendo, según se muestra en la *Figura 32*, activando simultáneamente T1 y T4 (a), la intensidad a través de la bobina L1 circula en sentido contrario que al activar T2 y T3 (b).



a) ***b)***

Figura 32. Funcionamiento del puente-H

Existen circuitos integrados especializados que constituyen un puente H, como por ejemplo el L293B. Un motor paso a paso unipolar dispone de un número mayor número de bobinas, pero cada una de ellas funciona siempre con la misma polaridad, por lo que el circuito de control es mucho más sencillo. Se suele utilizar un driver que proporcione la corriente necesaria para el funcionamiento del motor, como el ULN2003 que, además, contiene ya integrados los diodos de protección. En la *Figura 33* se puede ver el esquema de parte del integrado ULN2003. El terminal marcado como COM se conecta a Vcc del motor. Las señales de control se conectan a los terminales 1..4B, y las bobinas del motor a 1..4C. Escribiendo un '1' en el terminal 1B, el terminal 1C se conecta a tierra, con lo que la bobina 1 del motor queda polarizada. Lo mismo con los demás terminales.

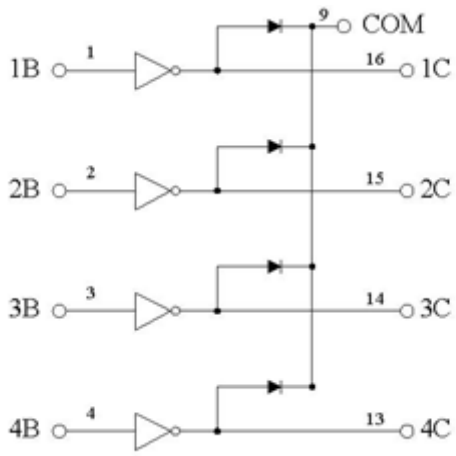


Figura 33. Esquema del driver de motor ULN2003

En la placa de expansión disponible en los laboratorios emplearemos un motor unipolar.
La entidad correspondiente al controlador de este periférico, que se debe instanciar desde el interfaz user_logic, se llama *motorstep*, se muestra en el *Cuadro 31* y se encuentra en el fichero motorstep.vhd.

```
entity motorstep is
Port (
clk : in std logic ;
rst : in std logic ;
dir : in std logic ;
stop : in std logic ;
halfstep : in std logic ;
motor: out std logic vector (3 downto 0);
step : out std logic vector (2 downto 0)
);
end motorstep ;
```

Cuadro 31. Entidad del controlador de motor paso a paso

La señal *clk* es el reloj para la máquina de estados del controlador. Cuanto mayor sea su frecuencia, más rápido avanzará el motor. Sin embargo, su frecuencia máxima viene dada por la capacidad de rotación del motor y la especifica el fabricante en la hoja de características. En nuestro caso hemos fijado un máximo de 10Hz y se proporciona el fichero 100_Kcounter.vhd donde se encuentra la entidad *one_hundred_K_counter* que genera dicho reloj a partir de un reloj de 50MHz. Será necesario instanciarla también dentro de user_logic, y conectar la salida del reloj generado a la entrada de reloj de motorstep. La señal rst es la señal global de reset. La señal dir define el sentido de rotación del motor. Cuando la señal *stop* vale 1, el motor está parado y cuando vale 0, el motor gira en el sentido marcado por *dir*. Con la señal *halfstep* podemos forzar al motor a avanzar en medios pasos. Al utilizar *halfstep*, el número de pasos por vuelta es 8. De esta manera se consigue mayor precisión en la posición del motor, a cambio de perder algo de fuerza. La señal *motor*, de 4 bits, se corresponde con las líneas de salida de control del motor, que atacan directamente al driver del motor.
Para la comunicación con Microblaze, este periférico consta de dos registros, uno de control motor_ctl (*slv_reg0*) y otro de estado motor_step (*slv_reg1*). La información del registro de control es la siguiente:

- La dirección de giro se encuentra en *motor_ctl*(0);
- La señal de stop se encuentra en *motor_ctl*(1);
- La señal de *halfstep* se encuentra en *motor_ctl*(2);
- El número de pasos a girar está en *motor_ctl*(4 to 7).

11.1. Conexión al sistema

Dentro de *user_logic* y del fichero top del periférico hay que añadir los puertos correspondientes a la salida del driver del motor, como se muestra en el Cuadro 32.

```
entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                : integer             := 32;
    C_NUM_REG                   : integer             := 2
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    -- ADD USER PORTS ABOVE THIS LINE ------------------
        control_motor: out std_logic_vector (3 downto 0);
    -- DO NOT EDIT BELOW THIS LINE ----------------------
```

Cuadro 32. Adición de puertos del motor en user_logic

Además, hay que añadir en el fichero *.ucf los nombres de los puertos según aparecen en el top del sistema. Un ejemplo se muestra en el *Cuadro 33*.

```
## IO Devices constraints

###### Motor
#
NET control_motor<0> LOC=L15;
NET control_motor<1> LOC=L5;
NET control_motor<2> LOC=N2;
NET control  motor<3> LOC=M3;
```

Cuadro 33. Restricciones en el *.ucf para el motor

Hay que instanciar la entity del contador `one_hundred_K_counter` para generar una frecuencia adecuada al motor.

Hay que instanciar la entity del motor y conectar adecuadamente las entradas y salidas. La entrada de reloj es la salida del contador. La entrada rst, el reset global. Las entradas dir, stop y halfstep se pasan a

través del registro de control (*slv_reg0* bits 0, 1 y 2). La salida *motor* es externa y la salida step hay que almacenarla en el registro de estado (*slv_reg1*).
Además, es necesario implementar el hardware necesario para que cuando el número de pasos dados sea igual al dado en el registro de control (bits 4 al 7) el motor se pare (se modifique el registro de control para que la señal stop se active). Se aconseja implementar un contador contador_step sobre los bits de dicho registro, de tal forma que cada vez que el contador `one_hundred_K_counter` active su salida ce_10 se decremente dicho contador, y cuando llegue a 0 se escriba un 0 en el bit 1 del registro de control *motor_ctl*, que corresponde a la señal de stop.

11.2. Control software

En el fichero *.c debe definirse la dirección de memoria correspondiente al motor, que en el ejemplo que se proporciona en el Campus Virtual es:

```
#define MOTOR_HW_BASEaddress (0xc0a00000)
```

La función para hacer girar al motor, que viene definida en el directorio driver correspondiente al periférico, es MOTOR_HW_mWriteSlaveReg0 (MOTOR_HW_BASEaddress, 0, *Data*) donde *Data* debe almacenar la información de control según se ha especificado en el apartado anterior:

- La dirección de giro se encuentra en *slv_reg0*(0);
- La señal de stop se encuentra en *slv_reg0* (1);
- La señal de *halfstep* se encuentra en *slv_reg0* (2);
- El número de pasos a girar está en *slv_reg0* (4 to 7).

Así, para que el motor comience a girar (stop=0) en sentido horario (dir=1), pasos completos (halfstep=0), y avance 15 pasos (nnnn=15), debemos hacer:

Data =0 x9E000000 ;
MOTOR_HW_mWriteReg (MOTOR_HW, BASEaddress , 0, Data);

En cada paso, el controlador del motor va decrementando un contador de pasos. Cuando llega a 0, se detiene el motor y el bit stop se pone al valor '1'. *L*eyendo el mismo registro de control e inspeccionando el valor del bit stop, podemos saber si el motor está girando o no:

Data = *MOTOR_HW_mReadReg* (MOTOR_HW, BASEaddress , 0);
while (!(Data & 0 x40000000)) {
Data = *MOTOR_HW_mReadReg* (MOTOR_HW, BASEaddress , 0); }

A través de la consola serie se muestra el contenido del registro estado (*slv_reg1*), donde el controlador del motor va escribiendo el paso actual del motor, desde 0 hasta 7. Para leer dicho registro se utiliza la función:
Data = *MOTOR_HW_mReadReg* (MOTOR_HW BASEaddress , 4);

12. Conversor analógico/digital

En este capítulo vamos a estudiar cómo utilizar un conversor analógico /digital para conectar diferentes dispositivos analógicos como sensores de temperatura, sensores de luminosidad, etc. a nuestro sistema básico. En particular, en la placa de expansión del laboratorio se encuentra el circuito integrado ADC0808, de National Semiconductor, que es un conversor analógico/digital con ocho canales analógicos multiplexados y resolución de 8 bits por canal. Se presenta en encapsulado DIP de 28 patas. Contiene las siguientes señales:

- *conv_clk* : entrada de reloj a 640 Khz
- *conv_oe:* permite volcar el contenido de la lectura en la salida *conv_data.*
- *Conv_addr:* codifica el número de canal sobre la que se va a realizar la conversión.
- *conv_star:* arranca una conversión por el canal determinado por *conv_addr.*
- *Conv_eoc:* determina el final de una conversión.
- *Conv_ale:* address latch enable
- *conv_data:* datos de salida

La lectura del resultado de la conversión se efectúa a través de sus 8 líneas de datos (*conv_data*), que tienen capacidad triestado. La conversión se realiza por aproximaciones sucesivas y mediante una red 256R.
El proceso de conversión y la lectura de los datos se realizan mediante las líneas de control y de datos. Es necesario activar las líneas de control en la forma especificada por el fabricante. De ello se encarga el módulo *adcO8O8* que se proporciona en los archivos fuente en el Campus Virtual, y cuya entity se puede ver en el *Cuadro 34.*

```
entity     adc0808 is
Port (
clk : in std logic ;
rst : in std logic ;
conv_clk : out std logic ;
conv_eoc : in std logic ;
conv_ale : out std logic ;
conv_start : out std logic ;
conv_addr : out std logic VECTOR (2 downto 0);
conv_oe : out std logic ;
conv_data : in std logic vector (7 downto 0);
addr_in : in std logic vector (2 downto 0);
start : in std logic ;
 data_out : out std logic vector (7 downto 0);
data_valid : out std logic );
end adc0808 ;
```

Cuadro 34. Entity del conversor A/D

Puede comprobarse que en la entity se encuentran definidas todas las entradas y salidas del CI ADC0808. Además de dichas señales, del reloj a 640K (*clk*) y del reset global (rst) existen cuatro señales de comunicación entre *usr_logic* y el driver adcO8O8 que son las siguientes:
Addr_in: dirección del canal por el que se quiere realizar la conversión, que genera *usr_logic*;

*Star*t: Señal de comienzo de conversión, que genera *usr_logic*;
Data_out: resultado de la conversión;
Data_valid: indica que el dato proporcionado de *data_out* es válido.
La comunicación entre *microblaze* y *usr_logic* se realiza a través de dos registros *slv_reg0* y *slv_reg1*.
Cada vez que se escribe en el *slv_reg0* se arranca una conversión por el canal dado por los bits 24 a 26, es decir:

```
addr_in <= slv_reg0(24 to 26);
```

Cuando se termina la conversion se pone a 1 el bit 23 de slv_reg0, es decir:

```
slv_reg0(23) <=data_valid;
```

El resultado de la conversión se proporciona en slv_reg1(24 to 31).

```
slv_reg1(24 to 31) <= data_out;
```

12.1 Conexión al sistema

Para conectar el driver del circuito ADC0808 hay que instanciar dentro de usr_logic tanto la entity adcO8O8 como la del divisor de 640K. Además hay que añadir en *usr_logic* y en el top del driver los puertos de entrada salida externos, según se muestra en el *Cuadro 35*.

```
entity user_logic is
generic
(
-- ADD USER GENERICS BELOW THIS LINE ---------------
--USER generics added here
-- ADD USER GENERICS ABOVE THIS LINE ---------------

-- DO NOT EDIT BELOW THIS LINE ----------------------
-- Bus protocol parameters, do not add to or delete
C_SLV_DWIDTH                : integer             := 32;
C_NUM_REG                   : integer             := 2
-- DO NOT EDIT ABOVE THIS LINE ----------------------
);
port
(
-- ADD USER PORTS BELOW THIS LINE ------------------
--USER ports added here
-- ADD USER PORTS ABOVE THIS LINE ------------------
conv_clk : out std_logic;
conv_eoc : in  STD_LOGIC;
conv_ale : out  STD_LOGIC;
conv_start : out  STD_LOGIC;
conv_addr : out  STD_LOGIC_VECTOR (2 downto 0);
conv_oe : out  STD_LOGIC;
conv_data : in std_logic_vector (7 downto 0);
-- DO NOT EDIT BELOW THIS LINE ----------------------
```

Cuadro 35. Adición de puertos en usr_logic del conversor A/D

Además hay que añadir la situación de los puertos correspondientes en la placa del laboratorio en el fichero *.ucf. El resultado se muestra en el *Cuadro 36*

```
NET conv_addr_pin<0> LOC= G1;
NET conv_addr_pin<1> LOC= H4;
NET conv_addr_pin<2> LOC= P2;
NET conv_ale_pin LOC= R1;
NET conv_clk_pin LOC = F2;
NET conv_eoc_pin LOC=N15;
NET conv_oe_pin LOC=G3;
NET conv_start_pin LOC = G4;
NET conv_data_pin<7>  LOC = J14;
NET conv_data_pin<6>  LOC = H14 ;
NET conv_data_pin<5>  LOC = M4;
NET conv_data_pin<4>  LOC = P1;
NET conv_data_pin<3>  LOC = N3;
NET conv_data_pin<2>  LOC = M15;
NET conv_data_pin<1>  LOC = H13;
NET conv_data_pin<0> LOC=G16;
```

Cuadro 36. fichero *.ucf del conversor A/D

Cada vez que se escriba en el registro *slv_reg0* hay que activar la señal *start.*
Cada vez que se activa *Data_valid* hay que almacenar el resultado de la conversión en el registro *slv_reg1*.

12.2 Control software

Para arrancar una conversión hay que escribir la dirección del canal en el registro *slv_reg0*. Para ello utilizaremos la función *CONVERSOR_mWriteSlaveReg0 (baseaddr , 0, Data)*. El resultado se proporciona en el registro *slv_reg1*. Para leerlo utilizaremos la función *CONVERSOR_mReadSlaveReg1 (baseaddr , 0).* A continuación, se muestra un ejemplo de código:

```
/*
* Se arranca una conversión en canal 1
*/
 Data =0 x40000000 ;
Canal=1;
Data =      Canal<<(31-26);
 while (1) {
xil_printf(" Arranca conversion en dirección %x 0 word 0\n\r", Data);
CONVERSOR_mWriteSlaveReg0 ( baseaddr , 0, Data );
Datoleido = CONVERSOR_mReadSlaveReg0 ( baseaddr , 0);
/* Esperamos fin conversión ( bit 23=1) */
while ( Datoleido == Data ) {
Datoleido = CONVERSOR_mReadSlaveReg0 ( baseaddr , 0);
}
xil_printf(" -read %x from register 0 word 0\n\r", Datoleido );
/* Leemos el resultado de la conversión en SlaveReg1 */
Datoleido = CONVERSOR_mReadSlaveReg1 ( baseaddr , 0);
xil_printf(" Valor de la conversión = %x from register 1 word 0\n\r", Datoleido ); }
```

13 Dispositivo analógicos

13.1 Sensor LDR

Un sensor LDR (Light Dependent Resistor) es una resistencia cuyo valor depende de la luz incidente. Se fabrican generalmente a base de sulfuro de cadmio, material semiconductor que tiene la propiedad de disminuir su resistencia cuando inciden fotones sobre él (ver Figura 34). La resistencia de una LDR puede variar desde varios megaohmios en oscuridad total hasta unos pocos centenares de ohmios con iluminación solar directa. Su símbolo electrónico se presenta en la Figura 35

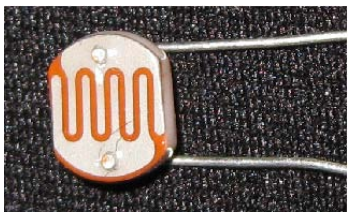


Figura 34. Ejemplo de fotoresistencia LDR

Figura 35. Símbolos electrónicos de la LDR

La respuesta de una LDR suele ser mucho más lenta (del orden de algunos ms) que la de otros dispositivos fotoelectrónicos, como por ejemplo los fotodiodos, por lo que no son adecuadas para la detección de variaciones rápidas en la luminosidad. Sin embargo, para los proyectos que se van a realizar en el laboratorio representa una ventaja, pues las variaciones que se esperan son lentas, y la LDR eliminará variaciones espúreas como, por ejemplo, el paso de la sombra de un pájaro sobre el sensor.
Para detectar la orientación adecuada hacia el Sol emplearemos dos foto resistencias dispuestas como en la Figura 36. La lámina que separa ambas LDR proyectará una sombra sobre una de ellas a no ser que se encuentre perfectamente alineada en la dirección al Sol. Comparando la resistencia de ambas y, por tanto, la luz incidente en cada una, podremos saber en qué dirección hay que mover el panel para que se alinee correctamente con el Sol.

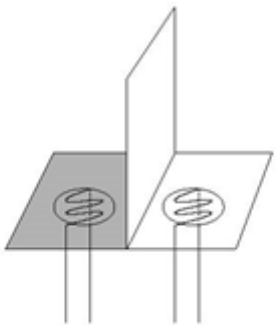


Figura 36. Disposición física de las LDR

En la placa del laboratorio existe una LDR que deberá conectarse mediante un cable al conversor analógico/digital visto en el capítulo anterior. Si se añade otro dispositivo LDR y se monta como se muestra en el esquema de la Figura 36 y utilizando el motor visto en el capítulo 11, podemos construir un sistema que se oriente automáticamente hacia la luz.

13.2 Sensor de temperatura

El LM35 es un circuito integrado que contiene un sensor de temperatura de precisión. Su tensión de salida depende de manera lineal de la temperatura, a razón de 10mV/ºC. Sin necesidad de ningún componente externo adicional, proporciona una precisión típica de 1/4ºC. Por ejemplo, para una temperatura ambiente de 23,5ºC, la salida del sensor es de 0,235V.
En la figura Figura 37 se puede ver el esquema de conexión y el patillaje del LM35 del encapsulado. plástico TO-92.

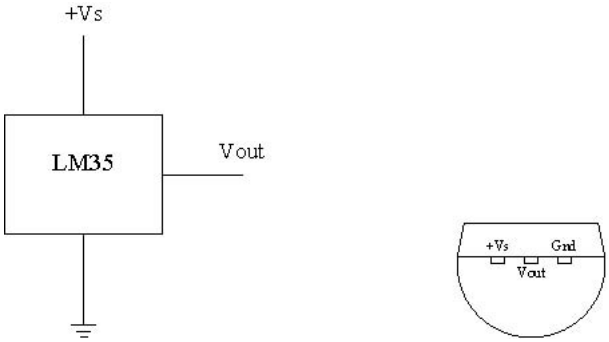


Figura 37. Sensor de temperatura LM35

En la placa de laboratorio existe un LM35 que deberá conectarse mediante un cable al conversor A/D visto en el capítulo anterior.

14 Control de un bus de comunicaciones I2C

I2C es el acrónimo de la expresión inglesa “Inter-Integrated Circui”', es decir, Inter-Circuitos Integrados. Se utiliza principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados. Se trata de un bus de comunicaciones en serie. La versión 1.0 data del año 1992 y la versión 2.1 del año 2000. Su diseñador es Philips. La velocidad máxima de funcionamiento de 100 kbit/s en el modo estándar, aunque también permite velocidades de 3-4 Mbit/s. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados (Embedded Systems) y generalizando más para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso.

El objetivo de este capítulo es diseñar el controlador de un bus de comunicaciones I2C. Dicho controlador debe implementar la funcionalidad básica que nos sirva para utilizar un bus de comunicaciones I2C, tales como una función que nos sirva para realizar la lectura (conocer el estado de un periférico) y otra para realizar la escritura (establecer su configuración).

La principal característica de I2C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, pero esta sólo es la referencia (masa). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa, esta tercera línea no suele ser necesaria.

Las líneas se llaman:

- SDA: datos
- SCL: reloj
- GND: tierra

Las dos primeras líneas son drenador abierto, por lo que necesitan resistencias de pull-up.

Los dispositivos conectados al bus I2C tienen una dirección única para cada uno. También pueden ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo. Esta característica hace que al bus I2C se le denomine bus multimaestro.

Las transacciones en el bus I2C tienen este formato:

|start | A7A6A5A4A3A2A1R/W | ACK | ...DATA... | ACK | stop | idle |

El bus está libre cuando SDA y SCL están en estado lógico alto.

En estado bus libre, cualquier dispositivo puede ocupar el bus I2C como maestro.

El maestro comienza la comunicación enviando un patrón llamado “*start condition*”. Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción

El maestro se dirige al dispositivo con el que quiere hablar, enviando un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada (L/E), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo).

La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit R/W.

El esclavo responde enviando un bit de ACK que le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse Seguidamente comienza el intercambio de información entre los dispositivos.
El maestro envía la dirección del registro interno del dispositivo que se desea leer o escribir.
El esclavo responde con otro bit de ACK.
Ahora el maestro puede empezar a leer o escribir bytes de datos. Todos los bytes de datos deben constar de 8 bits, el número máximo de bytes que pueden ser enviados en una transmisión no está restringido, siendo el esclavo quien fija esta cantidad de acuerdo a sus características.
Cada byte leído/escrito por el maestro debe ser obligatoriamente reconocido por un bit de ACK por el dispositivo maestro/esclavo. Se repiten los 2 pasos anteriores hasta finalizar la comunicación entre maestro y esclavo.
Aun cuando el maestro siempre controla el estado de la línea del reloj, un esclavo de baja velocidad o que deba detener la transferencia de datos mientras efectúa otra función, puede forzar la línea SCL a nivel bajo. Esto hace que el maestro entre en un estado de espera, durante el cual, no transmite información esperando a que el esclavo esté listo para continuar la transferencia en el punto donde había sido detenida.
Cuando la comunicación finaliza, el maestro transmite una "*stop condition*" para dejar libre el bus. Después de la "*stop condition*" es obligatorio para el bus estar idle durante unos microsegundos.

14.1 Implementación hardware del controlador I2C

En este apartado se detalla un componente I2C maestro para un único bus maestro, escrito en VHDL para su uso en FPGA. El componente lee y escribe a través del "*user_logic*" mediante una interfaz paralela. La Figura 38 ilustra un ejemplo típico de I2C maestro integrado en un sistema.

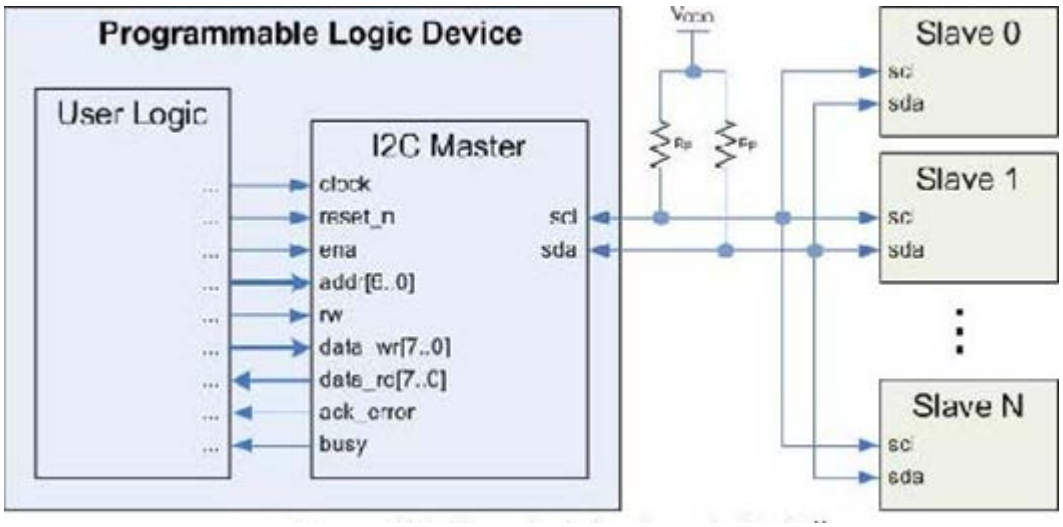


Figura 38. Ejemplo de implementación I2C.

El I2C maestro utiliza la máquina de estado se muestra en *Figura 39* para implementar el protocolo I2C. Tras el encendido, el componente entra inmediatamente en el estado "*ready*". Se espera en este estado hasta que la señal *ena* se activa y se registra el nuevo comando.

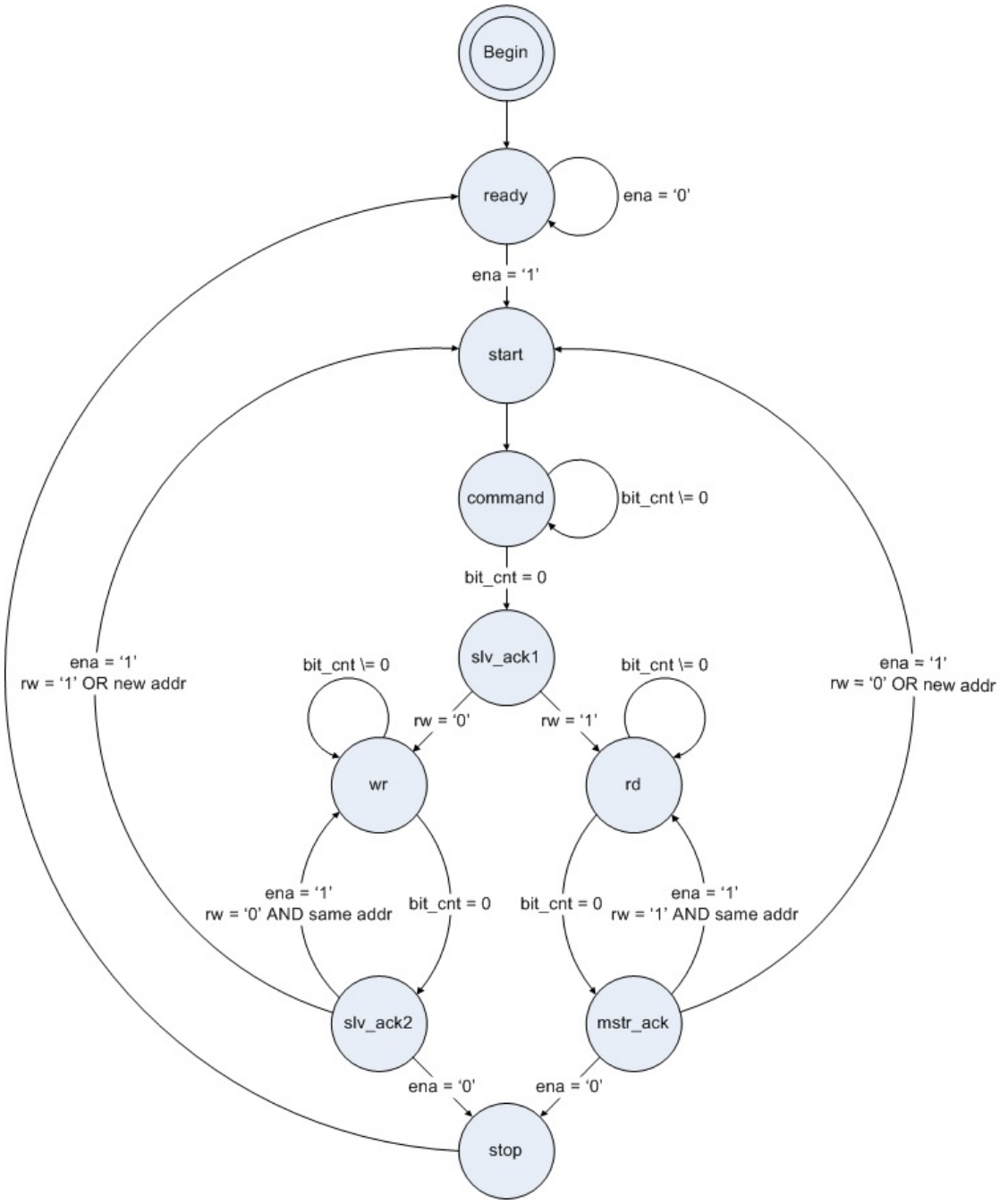


Figura 39. Máquina de estados del I2C maestro

El estado "*start*" genera la condición de arranque en el bus I2C, y el estado "*command*" comunica la dirección y el comando *rw* al bus. A continuación, se pasa al estado "*slv_ack1*" que captura y verifica el ACK del esclavo. Dependiendo del comando *rw*, el componente procede entonces a escribir datos al esclavo (estado "*rw*") o recibir datos desde el esclavo (estado "*rd*"). Una vez terminado, el maestro captura y verifica la respuesta del esclavo (estado "slv_ack2") si es una escritura o emite su propia respuesta (estado "*mstr_ack*") si es una lectura. Si la señal *ena* se activa para otro comando, el maestro continúa inmediatamente con otra escritura (estado "*wr*") o lectura (estado "*rd*") si el comando es el mismo que el comando anterior. Si es diferente a la orden anterior (es decir, una lectura después de una escritura o una escritura después de una lectura o una nueva dirección para el esclavo) , el maestro

```
ENTITY i2c_master IS
  GENERIC(
    input_clk : INTEGER := 50_000_000; --input clock speed from user logic in Hz
    bus_clk   : INTEGER := 100_000);   --speed the i2c bus (scl) will run at in Hz
  PORT(
    clk       : IN     STD_LOGIC;                    --system clock
    reset     : IN     STD_LOGIC;                    --active low reset
    ena       : IN     STD_LOGIC;                    --latch in command
    addr      : IN     STD_LOGIC_VECTOR(6 DOWNTO 0); --address of target slave
    rw        : IN     STD_LOGIC;                    --'0' is write, '1' is read
    data_wr   : IN     STD_LOGIC_VECTOR(7 DOWNTO 0); --data to write to slave
    busy      : OUT    STD_LOGIC;                    --indicates transaction in progress
    data_rd   : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0); --data read from slave
    ack_error : BUFFER STD_LOGIC;                    --flag if improper acknowledge from slave
--    sda       : INOUT  STD_LOGIC;                    --serial data output of i2c bus
    sda_I     : IN   STD_LOGIC;                    --serial data input of i2c bus
    sda_O     : OUT  STD_LOGIC;                    --serial data output of i2c bus
    sda_T     : OUT  STD_LOGIC;                    --serial data three state of i2c bus
--    scl       : INOUT  STD_LOGIC);                   --serial clock output of i2c bus
    scl_I     : IN   STD_LOGIC;                    --serial data input of i2c bus
    scl_O     : OUT  STD_LOGIC;                    --serial data output of i2c bus
    scl_T     : OUT  STD_LOGIC);                   --serial data three state of i2c bus
END i2c_master;
```

Cuadro 37. Entity del controlador I2C

realiza un nuevo comienzo (estado *"start")* según la especificación I2C. Una vez que el maestro termine la lectura o escritura y la señal *ena* no se active para un nuevo comando, el maestro genera la condición de parada (estado *"stop")* y vuelve al estado *"ready"*.

La temporización para la máquina de estados se calcula de la siguiente manera. El componente obtiene el reloj *scl* a partir de dos parámetros genéricos declarados en la entidad, *input_clk* y *bus_clk*. El parámetro *input_clk* debe estar ajustado a la frecuencia de reloj de entrada del sistema *clk* en Hz. La configuración por defecto en el código es de 50 MHz. El parámetro *bus_clk* debe ajustarse a la frecuencia deseada para señal de reloj de scl. La configuración por defecto es de 400 kHz, lo que corresponde a la tasa de bits más rápida en la especificación I2C.

En el *Cuadro 37* se muestra el interfaz de la entity correspondiente al controlador I2C. Destacar que para poder incorporar un periférico con señales de entrada/salida, EDK obliga a que se definan las 3 señales que controlaran el IOBUF (entrada, salida y triestado). Por tanto, para *sda* y *scl* tendremos:

- sda_I (1 bit): entrada de la señal de datos
- sda_O (1 bit): salida de la señal de datos
- sda_T (1 bit): triestado de la señal de datos
- scl_I (1 bit): entrada de la señal de reloj
- scl_O (1 bit): salida de la señal de reloj
- scl_T (1 bit): triestado de la señal de reloj

Además, para la comunicación entre el controlador y *usr_logic* tenemos las siguientes E/S:

- ena: valida la dirección del dispositivo
- addr: dirección del dispositivo
- *rw*: '0' para escritura y '1' para lectura
- data_wr: dato para escribir en dispositivo
- busy: indica que el driver está ocupado

- data_rd: dato leído por el dispositivo
- ack_error: indica que no se ha podido leer correctamente

La comunicación entre *Microblaze* y el controlador hardware se realiza a través de una FIFO en la que *Microblaze* escribirá los comandos que deberán ser ejecutados secuencialmente por el controlador. En la FIFO se escribe en los 16 bits menos significativos (16-31) la cadena valor+dirección+r/w donde:

- Valor: dato de 8 bits que se quiere escribir en una operación de escritura
- Dirección: dirección de 7 bits del dispositivo sobre el que se va a realizar la operación
- r/w: 1 bit para indicar lectura (0) o escritura (1). En el driver del controlador estos valores se invierten.

14.2 Conexión al sistema

Dentro de *user_logic* y del fichero top del periférico hay que añadir los puertos correspondientes a la entrada/salida controlador del I2C, como se muestra en el *Cuadro 38*.

```
entity user_logic is
generic (
--ADD USER GENERICS BELOW THIS LINE ----------------USER generics added here
input_clk : INTEGER := 50000000;
--input clock speed from user_logic in Hz
 bus clk : INTEGER := 400000;
--speed the i2c bus ( scl ) will run at in Hz
--ADD USER GENERICS ABOVE THIS LINE ----------
(
--ADD USER PORTS BELOW THIS LINE -----------------
--USER ports added here
sda_I : IN STD LOGIC ; --serial data input of i2c bus
sda_O : OUT STD LOGIC ; --serial data output of i2c bus
sda_T : OUT STD LOGIC ; --serial data three state of i2c bus
scl_I : IN STD LOGIC ; --serial clock input of i2c bus
scl_O : OUT STD LOGIC ; --serial clock output of i2c bus
scl_T : OUT STD LOGIC ; --serial clock three state of i2c bus
--Bus protocol ports , do not add to or delete
…);
```

Cuadro 38. Definición de puertos en usr_logic del controlador I2C

Además hay que añadir la posición en la placa del laboratorio de los puertos de E/S correspondientes al I2C en el fichero *.ucf, según se muestra en el *Cuadro 39*.

```
#### I2C
Net i2c_scl LOC=F5;
Net i2c_sda LOC=D2;
```

Cuadro 39. Restricciones de los puertos para el controlador I2C en el fichero *.ucf

Dentro de *usr_logic* hay que implementar el hw necesario para leer de la FIFO los datos y enviárselos al driver. Por tanto, deberemos hacer las siguientes asignaciones:

```
i2c_addr(0) <= WFIFO2IP_Data_REG(30);
i2c_addr(1) <= WFIFO2IP_Data_REG(29);
i2c_addr(2) <= WFIFO2IP_Data_REG(28);
i2c_addr(3) <= WFIFO2IP_Data_REG(27);
i2c_addr(4) <= WFIFO2IP_Data_REG(26);
i2c_addr(5) <= WFIFO2IP_Data_REG(25);
i2c_addr(6) <= WFIFO2IP_Data_REG(24);

i2c_data_wr(0) <= WFIFO2IP_Data_REG(23);
i2c_data_wr(1) <= WFIFO2IP_Data_REG(22);
i2c_data_wr(2) <= WFIFO2IP_Data_REG(21);
i2c_data_wr(3) <= WFIFO2IP_Data_REG(20);
i2c_data_wr(4) <= WFIFO2IP_Data_REG(19);
i2c_data_wr(5) <= WFIFO2IP_Data_REG(18);
i2c_data_wr(6) <= WFIFO2IP_Data_REG(17);
i2c_data_wr(7) <= WFIFO2IP_Data_REG(16);
lecturaEscritura <= WFIFO2IP_Data_REG(31);

IP2RFIFO_Data(31) <= i2c_data_rd(0);
IP2RFIFO_Data(30) <= i2c_data_rd(1);
IP2RFIFO_Data(29) <= i2c_data_rd(2);
IP2RFIFO_Data(28) <= i2c_data_rd(3);
IP2RFIFO_Data(27) <= i2c_data_rd(4);
IP2RFIFO_Data(26) <= i2c_data_rd(5);
IP2RFIFO_Data(25) <= i2c_data_rd(6);
IP2RFIFO_Data(24) <= i2c_data_rd(7);
```

Cada vez que se envíe un comando nuevo a la FIFO hay que activar la señal *ena*, que valida la dirección del dispositivo, para comenzar la lectura/escritura correspondiente. Cuando se trate de una lectura, hay que esperar a que el dispositivo desactive *busy* para leer el dato y pasárselo a Microblaze a través de la FIFO. En el caso de múltiples lecturas o escrituras hay que implementar también el contador correspondiente.

14.3 Control software

Para realizar una lectura o escritura en un dispositivo determinado conectado al controlador I2C primero hay que mandar la orden correspondiente y luego escrituras o lecturas consecutivas.
Vamos a utilizar las funciones que nos proporciona SDK en el directorio del driver correspondiente. Para este caso en particular tenemos:
I2C_mWriteToFIFO (BASE_ADDRESS_I2C , 0, (Xuint32) ((value << 8) + (dirección << 0) + I2C_ CMD));
I2C_mReadFromFIFO (BASE_ADDRESS_I2C , 0)

El formato de una orden es una escritura en FIFO, como hemos mencionado anteriormente es
Valor(8bits)<<8+(dirección(7bits)<<0)+ I2C_ CMD
Donde I2C_ CMD es 0 para lectura y 1 para escritura
Hay que definir las siguientes constantes:

```
#define BASE_ADDRESS_I2C   0xC3000000 /* depende del diseño

#define LECTURA_I2C_CMD    0x00000000
#define ESCRITURA_I2C_CMD  0x00000001 /* por hw se cambia el valor */
#define DIR_EXPANSOR       0x00000072 /*depende del chip particular
#define DIR_MEMORIA        0x000000A4 /*depende del chip particular
```

Para llevar a cabo un control adecuado del bus de comunicaciones I2C se ha desarrollado el driver que implementa las funciones de escritura y lectura de 1 byte:

```
// ESCRITURA I2C 1 byte
void I2C_escritura ( Xuint32 dirección , Xuint32 dato ){
        while ( I2C_mWriteFIFOFull ( BASE_ADDRESS_I2C ) == TRUE ){}
        I2C_mWriteToFIFO(BASE_ADDRESS_I2C, 0, (Xuint32) ((dato << 8)) + (direccion <<0) + ESCRITURA_I2C_CMD);

}
```

Para realizar la escritura de varios bytes que pasaremos como un array en dato[] primero enviamos la orden de escritura y el número de bytes, y a continuación escribimos en la FIFO los datos de 4 en 4 bytes hasta llegar al número de bytes fijado. Utilizaremos la siguiente función:

```
        // ESCRITURA I2C num_datos

void I2C_escritura(Xuint32 dato[], Xuint32 direccion, Xuint32 num_datos){
        int i;
        while (I2C_mWriteFIFOFull(BASE_ADDRESS_I2C) == TRUE){}
        I2C_mWriteToFIFO(BASE_ADDRESS_I2C,0,(Xuint32)((num_datos<< 16) + (direccion <<0) + ESCRITURA_I2C_CMD));

        for (i=0; i< (num_datos/4 + ((num_datos%4)!=0)); i=i+1)
        {   while (I2C_mWriteFIFOFull(BASE_ADDRESS_I2C) == TRUE){}
                I2C_mWriteToFIFO(BASE_ADDRESS_I2C, 0, (Xuint32) dato[i]);
        }
        my_delay(20);
}


// LECTURA I2C
 Xuint32 I2C_lectura ( Xuint32 dirección ){
        while ( I2C_mWriteFIFOFull ( BASE_ADDRESS_I2C ) == TRUE ){}
        I2C_mWriteToFIFO ( BASE_ADDRESS_I2C , 0, ( Xuint32 ) (( dirección << 1) + LECTURA I2C CMD ));
         while ( I2C_mReadFIFOEmpty ( BASE_ADDRESS_I2C ) == TRUE ){}
        return I2C_mReadFromFIFO ( BASE_ADDRESS_I2C , 0);
}
```

El ejemplo que se muestra a continuación *lee* la configuración establecida en los switches, escribe el dato en el expansor cuya dirección es DIR_EXPANSOR = 0x00000072, luego lee la configuración de ese mismo expansor y escribe la lectura en los leds

```
int main (){
xil_printf (" Practica bus i2c \n\r" );
XGpio GpioLEDs ; XGpio GpioSwitches ; XStatus Status ;
Xuint32 value ;
Status = XGpio_Initialize (& GpioLEDs , XPAR_XPS_GPIO_LEDS_DEVICE_ID );
If (Status !=  XST_SUCCESS)
        { xil_printf("ERROR initializing GPIO\r\n");
        return  XST_FAILURE ; }
Status = XGpio Initialize (& GpioSwitches , XPAR_XPS_GPIO_SWITCHES_DEVICE_ID );
```

```
 if (Status !=  XST_SUCCESS)
              { xil_printf("ERROR initializing GPIO\r\n");
              return  XST_FAILURE ; }
XGpio SetDataDirection (& GpioLEDs , 1, 0 x00 );
XGpio SetDataDirection (& GpioSwitches , 1, 0 xFF );
I2C_mReset ( BASE_ADDRESS_I2C );
 I2C_mResetWriteFIFO ( BASE_ADDRESS_I2C );
 I2C_mResetReadFIFO ( BASE_ADDRESS_I2C );
while ( TRUE ){
 value = XGpio DiscreteRead ( & GpioSwitches , 1 );
xil_printf("Value read from the Switches: %x\r\n", value);
I2C_escritura ( DIR_EXPANSOR , value );
value = I2C_lectura ( DIR_EXPANSOR );
XGpio DiscreteWrite ( & GpioLEDs , 1, ( Xuint32 ) value );
xil_printf("Value written to the LEDs: %x\r\n\n", value);
my delay (5000); }
return 0; }
```

15 Dispositivos I2C básicos

En esta práctica aprenderemos a utilizar los dos dispositivos I2C básicos incorporados en la placa de periféricos: un expansor de entrada/salida (PCF8574A) y un conversor analógico/digital y digital/analógico (PCF8591)

15.1 Expansor I2C PCF8574A

El integrado PCF8574A es un dispositivo expansor de entrada/salida controlable mediante el bus I2C. Dispone de 8 pines cuasi-bidireccionales que se pueden emplear como entradas o como salidas. En la Figura 40 puede verse su diagrama de bloques. Además de las señales SCL y SDA del bus I2C, se tienen los pines AO, A1 y A2 para fijar la dirección del dispositivo. De esta manera podrían instalarse hasta 8 dispositivos PCF8574A en el mismo bus. La dirección en el bus I2C es O 1 1 1 A2 A1 AO R/W. En nuestro caso, las 3 señales de dirección están fijadas a 0 (a 2 para el segundo expansor), por lo que el expansor es seleccionable en las direcciónes Ox7O (escritura) y Ox71 (lectura)- Ox72 (escritura) y Ox73 (lectura) para el otro expansor.

El PCF8574A puede emplearse como expansor de entrada o de salida, y cada pin es utilizable de manera independiente. Cuando se escribe un dato en el expansor, el valor de cada bit escrito se refleja en el estado del pin correspondiente. Cuando se lee un dato desde el expansor, el resultado de cada bit leído es el estado del respectivo pin.
Si queremos emplear el dispositivo como expansor de salida, debemos realizar escrituras sobre el mismo. La secuencia en el bus I2C sería la siguiente:
| *start* | 0111 0000 | *ACK* | P7P6 P5 P4P3P2 P1 PO | *ACK* | *stop* | *idle* |

Donde P7..PO representan los valores que se quieren mostrar en cada pin de salida. Por ejemplo, escribiendo un '1' en la posición P5 provocará que el pin P5 se ponga en estado alto (Vcc), mientras que la escritura de un '0' lo pondrá en estado bajo (GND).

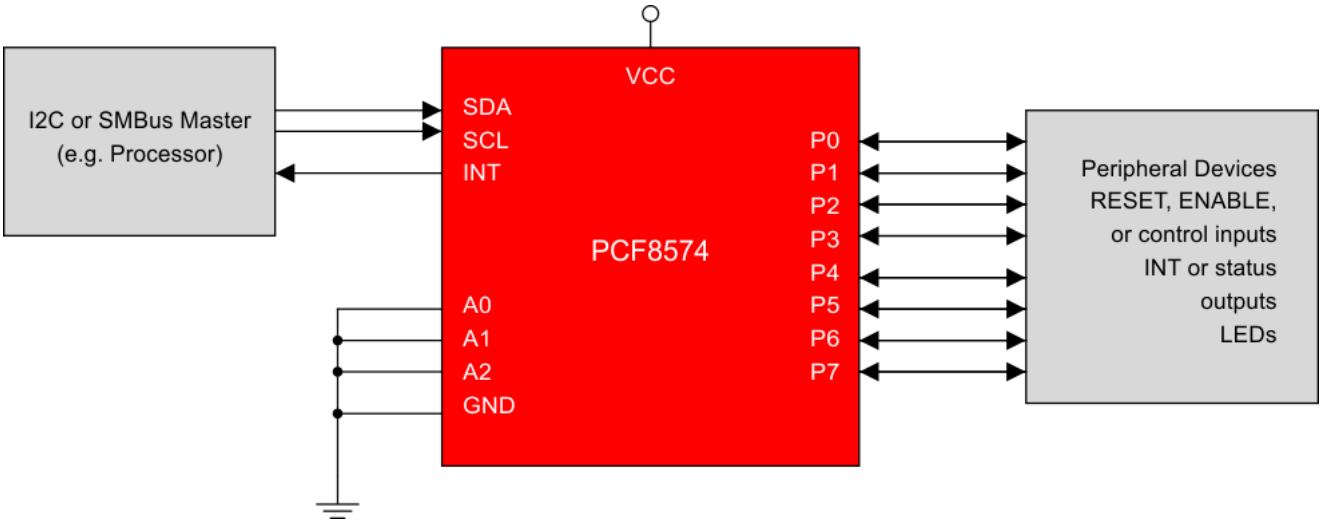


Figura 40. Expansor de entrada salida PCF8574

Para emplearlo como expansor de entrada, realizaremos lecturas. La secuencia en el bus I2C sería la siguiente:

| *start* | 0111 0001 | *ACK* | P7P6 P5 P4P3P2 P1 PO | *ACK* | stop | *idle* |
Ahora, los valores obtenidos en P7..PO reflejan el estado de los pines. Hay que tener en cuenta que, para emplear un pin como entrada, previamente es necesario escribir un '1' en dicho pin. Como cada pin puede usarse como entrada o como salida de manera independiente de los demás, supongamos que deseamos utilizar los pines P7..P4 como salidas (para controlar, por ejemplo, sendos leds) y los pines P3..PO como entradas (para leer, por ejemplo, el estado de cuatro pulsadores o interruptores). Entonces deberíamos hacer lo siguiente:

- Escribir el estado inicial de los leds (pines de salida, L3..LO) y el valor '1' en los pines de entrada:
  - | *start* | 0111 0000 | *ACK* | L3L2 L1 LO1 1 1 1| *ACK* | *stop* | *idle* |
- Leer el estado de los interruptores:
  - | start | 0111 0001 | ACK | X X X XP3 P2 P1PO | ACK | stop | idle |
- Descartar, mediante máscara, los bits más *significativo*s del valor leído.

El PCF8574 dispone, además, de una línea de interrupción, INT, activa a baja, que indica cuándo se ha producido un cambio en alguno de los pines de entrada. En nuestra placa, sin embargo, esta línea se ha dejado sin conectar.

15.2 Conversor analógico/digital y digital/analógico PCF8591

El integrado PCF8591 es un dispositivo que incluye cuatro entradas a un conversor analógico/digital para la adquisición de datos, y una salida para el conversor digital/analógico. Todas las conversiones se realizan con una profundidad de muestreo de 8 bits. En la Figura 41 puede verse su diagrama de bloques. Además de las señales SCL y SDA del bus I2C, se tienen los pines AO, A1 y A2 para fijar la dirección del dispositivo. De esta manera podrían instalarse hasta 8 dispositivos PCF8591 en el mismo bus. La dirección en el bus I2C es 1 O O 1 A2 A1 AO R/W. En nuestro caso, las 3 señales de dirección están fijadas a 0, por lo que el conversor es seleccionable en las direcciónes Ox9O (escritura) y Ox91 (lectura).

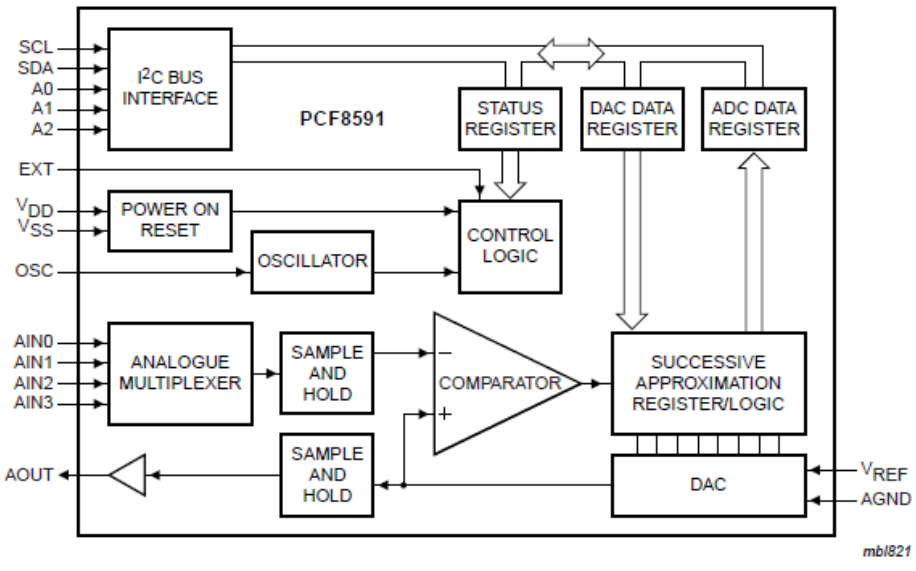


Figura 41. Esquema del PCF8591

Antes de realizar una conversión analógica/digital, es necesario configurar adecuadamente el integrado PCF8591. Para ello, se debe escribir el valor adecuado en su registro de control. Dicho registro, de 8 bits, tiene la siguiente estructura:
| 0 | *AoE* | *AiP1* | *AiP0* | 0 | *AI* | *Ai1* | *AiO* |
con el siguiente significado:
AoE (Analog output Enable): cuando vale '1', se activa la salida del conversor digital/analógico
AiP (Analog input Programming) (*AiP1AiP0*): programa el modo de funcionamiento de las entradas analógicas, según la tabla 15.1.
AI (Auto Increment): si vale 1, se incrementa de manera automática el canal de entrada para lecturas sucesivas
Ai (Analog input) (*Ai1Ai0*): selección del canal de entrada analógica.

	AiP	Canal 0	Canal 1	Canal 2	Canal 3	Observaciones	
	OO	AINO	AIN1	AIN2	AIN3	4 entradas analógicas simples	
	O1	AINO AIN3	AIN1 AIN3	AIN2 AIN3	X	3 entradas diferenciales	
	1O	AINO	AIN1	AIN2 AIN3	X	2 entradas simples y 1 diferencial	
	11	AINO AIN1	AIN2 AIN3	X	X	2 entradas diferenciales	

TABLA 15 .1: Configuración de las entradas analógicas en el PCF8591

Cuando se utiliza el modo de autoincremento, es conveniente activar también la salida analógica, para que así el oscilador interno esté funcionando de manera continua y evitar errores en las conversiones.
Al utilizar la conversión analógica/digital, el PCF8591 se direcciona para lectura a través del bus I2C. La secuencia sería la siguiente:

- Programar en el registro de control la configuración adecuada de las entradas y seleccionar el canal de entrada. Por ejemplo, para configurar las entradas como 4 entradas analógicas independientes, seleccionar el canal 0 y sin autoincremento, la secuencia que se debe enviar por el bus I2C es la siguiente:
  | *start* | 1001 0000 | ACK | 0000 0000 | ACK | *stop* | *idle* |
- Leer el valor de la conversión:
  | start | 1001 0001 | ACK | VVVV VVVV | ACK | *stop* | idle |

Hay que tener en cuenta que cuando se realiza una lectura, se está en realidad leyendo el valor de la conversión anterior. La primera vez que se programa el PCF8591, hay que hacer una lectura fantasma, que devolverá el valor Ox8O. Las siguientes lecturas contendrán valores válidos de la conversión.
A las entradas del conversor analógico/digital podemos conectar la salida de cualquiera de los sensores analógicos de los que disponemos, como la célula LDR, la resistencia NTC, el sensor de temperatura LM35, el sensor de flexión, el de fuerza, etc. Los sensores resistivos (LDR, NTC, flexión y fuerza) deben conectarse por medio de un divisor de tensión, como el mostrado *Figura 42*.
Para emplear la salida analógica del PCF8591 tan sólo es necesario poner a '1' el bit AoE del registro de control y escribir el valor de salida en el tercer byte que se envía por el bus I2C. La secuencia es la siguiente, donde V representa el valor deseado:

| *start* | 1001 0000 | *ACK* | 0 1 *P1 P0* 0 *Ai1Ai0* | ACK | VVVV VVVV | *ACK* | *stop* | *idle* |

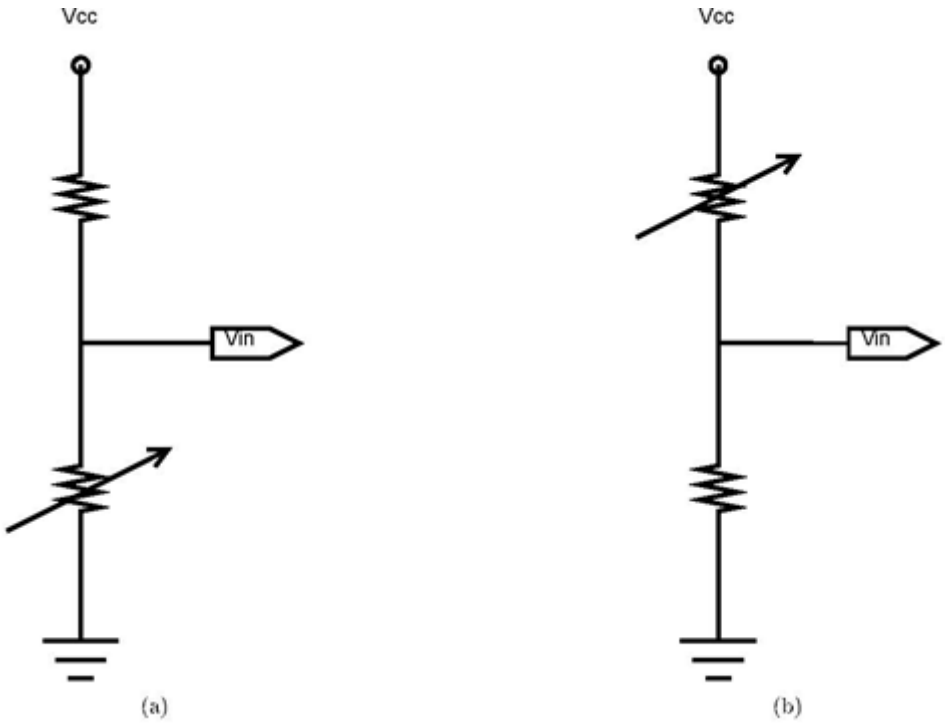


Figura 42. Divisor de tensión directo (a) o inverso (b)