Laboratorio de Sistemas Empotrados

Juan Carlos Fabero
Hortensia Mecha
José Manuel Mendías
Carlos González
Juan Antonio Clemente

12 de febrero de 2015

Índice general

1. Sistema básico **1**

2. Teclado PS2 **5**
2.1. Introducción . . . 5
2.2. Conexión al sistema . . . 5
2.3. Control software . . . 7

3. Teclado Matricial o Keypad **9**
3.1. Introducción . . . 9
3.2. Conexión al sistema . . . 10
3.3. Control software . . . 12

4. Matriz de Puntos o Banner **15**
4.1. Introducción . . . 15
4.2. Conexión al sistema . . . 16
4.3. Control software . . . 18

5. Control de un monitor VGA **19**
5.1. Introducción . . . 19
5.2. Monitor VGA . . . 19
5.3. Adición del periférico al sistema . . . 21
5.3.1. Señales de entrada/salida y conexión con MicroBlaze . . . 21
5.3.2. Cómo importar el periférico en EDK . . . 22
5.3.3. Control del periférico . . . 23

6. Control de una pantala LCD **25**
6.1. Introducción . . . 25
6.2. Pantalla LCD . . . 25
6.3. Adición del periférico al sistema . . . 26
6.3.1. Señales de entrada/salida y conexión con MicroBlaze . . . 26
6.3.2. Cómo importar el periférico en EDK . . . 28

6.3.3. Control del periférico 29

7. Control de un zumbador **31**
7.1. Introducción 31
7.2. Zumbador 31
7.3. Adición del periférico al sistema 32
7.3.1. Señales de entrada/salida y conexión con MicroBlaze 32
7.3.2. Cómo importar el periférico en EDK 33
7.3.3. Control del periférico 33

8. Control de un altavoz **35**
8.1. Introducción 35
8.2. Altavoz 35
8.3. Adición del periférico al sistema 37
8.3.1. Señales de entrada/salida y conexión con MicroBlaze 37
8.3.2. Cómo importar el periférico en EDK 38
8.3.3. Control del periférico 39

9. Control del encedido de un LED RGB a través de un PWM **43**
9.1. PWM 43
9.2. LED RGB 43
9.3. Descripción hardware del periférico *LED_RGB* 44
9.4. Software de control del periférico *LED_RGB* 45
9.5. Ficheros fuente 46

10. Control de un par de dispositivos emisor-receptor de infrarrojos **47**
10.1. Dispositivo emisor de infrarrojos 47
10.2. Dispositivo receptor de infrarrojos 48
10.3. Descripcion hardware del periférico *ER_Infrared* 49
10.4. Software de control del periférico *ER_Infrared* 49
10.5. Ficheros fuente 50

11. Control de un motor paso a paso **51**
11.1. Introducción 51
11.2. Motor paso a paso 51
11.2.1. Motor paso a paso bipolar 52
11.2.2. Motor paso a paso unipolar 52
11.3. Descripción *hardware* del periférico `motorstep` 53
11.4. Control software del periférico `motorstep` 54
11.5. Ficheros fuente 55

12.Conversión analógico/digital **57**
12.1. Fotoresistencias LDR . . . 57
12.2. Sensor de temperatura LM35 . . . 58
12.3. Sensor de flexión . . . 58
12.4. Sensor de fuerza resistivo circular . . . 59
12.5. Conversor analógico-digital ADC0808 . . . 61
12.6. Control software . . . 62
12.7. Ficheros fuente . . . 63
12.8. Sensor de flexión . . . 63
12.9. Sensor de fuerza resistivo circular . . . 64

13.Control de un bus de comunicaciones I2C **67**
13.1. Introducción . . . 67
13.2. Bus de comunicaciones I2C . . . 67
13.3. Implementación hardware del controlador I2C . . . 69
13.3.1. Introducción . . . 69
13.3.2. Modo de funcionamiento . . . 69
13.4. Adición del periférico al sistema . . . 71
13.4.1. Señales de entrada/salida y conexión con MicroBlaze . . . 71
13.4.2. Cómo importar el periférico en EDK . . . 73
13.4.3. Control del periférico . . . 74

14.Dispositivos I2C básicos **77**
14.1. Expansor I2C PCF8574 . . . 77
14.1.1. Funcionamiento . . . 77
14.2. Conversor analógico/digital y digital/analógico PCF8591 . . . 79
14.2.1. Conversión Analógica/Digital . . . 79
14.2.2. Conversión Digital/Analógica . . . 80

15.Acelerómetro **83**

16.Magnetómetro **85**
16.0.3. LSM303D . . . 85

Práctica 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 + controladores de memoria.
3. Controlador RS232 conectado al puerto serie.
4. GPIO conectados a los *switches*.
5. GPIO conectados a los leds.
6. Placa de expansión con diversos periféricos.

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 1.3

El fichero `ucf` debe tener las restricciones necesarias para conectar el reloj, el reset, los leds, 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 a continuación:

```
# Reloj
Net sys_clk_pin LOC=P8;
#Reset
Net sys_rst_pin LOC=E11;
## System level constraints (reloj a 50MHz)
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 20000 ps;
#### Module RS232 constraints
Net RX_pin LOC=G5;
Net TX_pin LOC=J2;
```

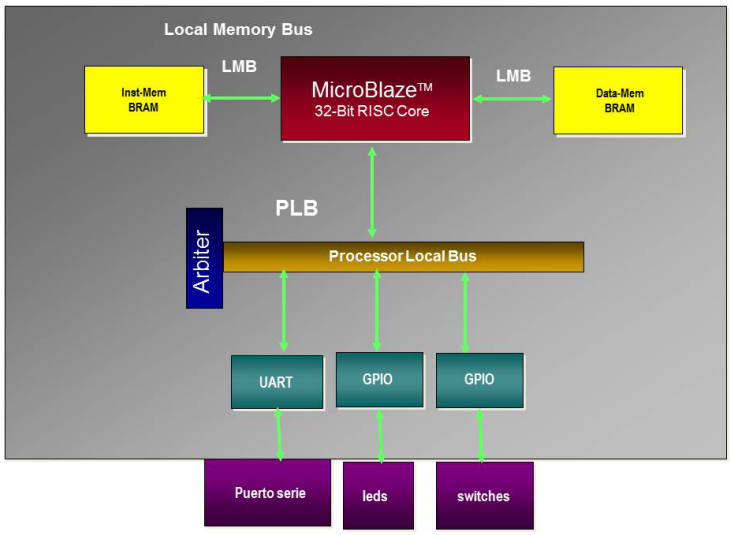


FIGURA 1.1: Sistema básico utilizado en las prácticas

```
## 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 superior
NET switches_pin<0> LOC=K4;
NET switches_pin<1> LOC=K3;
NET switches_pin<2> LOC=K2;
NET switches_pin<3> LOC=J4;
```

Una vez generada la `netlist` se exporta el diseño a SDK y se genera el programa que se ejecutará en la 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 (`Gpio_LEDs`), y dos constantes que son las direcciones de memoria que ocupan los *switches* (`XPAR_SWITCHES_DEVICE_ID`) y los leds (`XPAR_LEDS_DEVICE_ID`).

```
/* The driver instance for GPIO Device configured as Salida */
XGpio Gpio_LEDs;
```

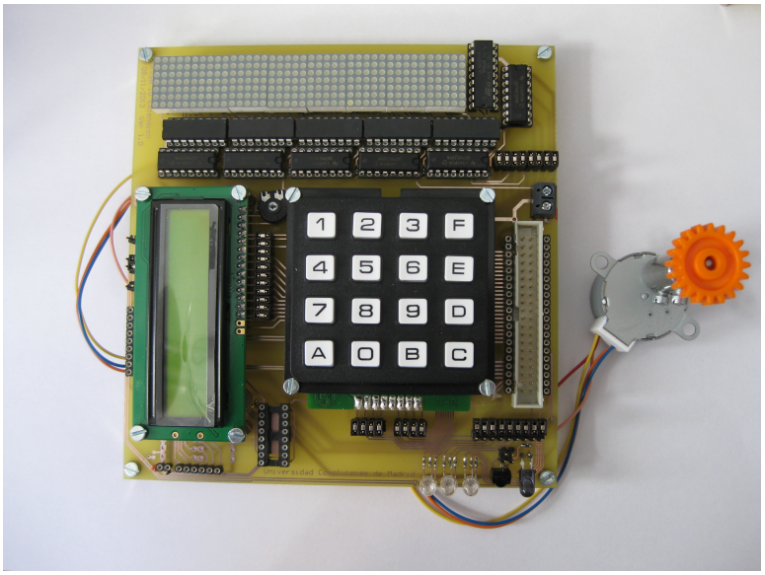


FIGURA 1.2: Placa de expansión

```
/* The driver instance for GPIO Device configured as Entrada */
XGpio GpioSwitches;

// Configuración de la GPIO para los LEDs de la placa extendida
/*Obtiene el puntero a la estructura */
XGpio_Initialize(&Gpio_LEDs, XPAR_LEDS_DEVICE_ID);
/*Coloca la dirección de salida */
XGpio_SetDataDirection (&Gpio_LEDs, 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, 0xFF);
```

Para leer de los *switches* usaremos la función `XGpio_DiscreteRead`:

```
DataRead = XGpio_DiscreteRead(&GpioSwitches, XPAR_SWITCHES_DEVICE_ID);
```

Para escribir en los leds usaremos la instrucción `XGpio_DiscreteWrite`:

```
XGpio_DiscreteWrite(&Gpio_LEDs, 1, DataRead);
```

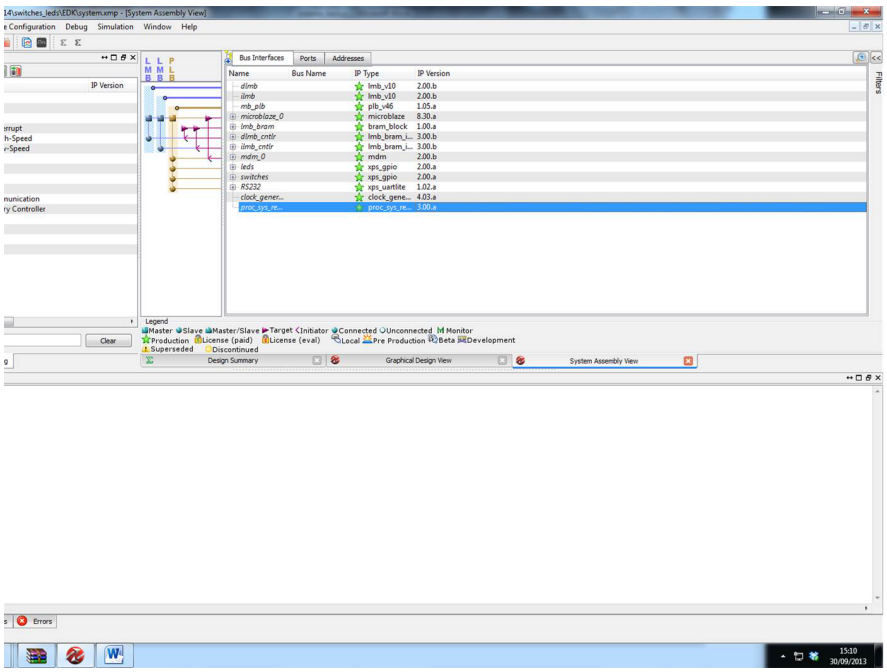


FIGURA 1.3: Entorno EDK

Como 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");
```

Práctica 2

Teclado PS2

2.1. Introducción

Nombre del periférico: `teclado1`.

Funcionamiento: El interfaz teclado PS2 nos permite conectar un teclado estándar al conector PS2 de la placa de FPGAs. La comunicación es de tipo serie con 2 líneas, una de reloj y otra de datos, que sigue el esquema de la figura 2.1:

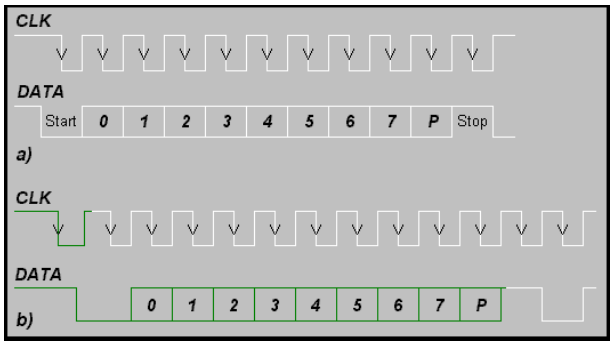


Figura 2.1: Protocolo PS2

2.2. Conexión al sistema

En `user_logic` añadir:

```
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
    -- 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
```

Las líneas `ps2CLK` y `ps2Data` son las líneas que se conectan directamente al conector PS2, y dependiendo de la placa tendrán una situación diferente. Es necesario definirlas como pines externos (dentro de `user_logic` y de `teclado1`) 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 la siguiente:

```
Net ps2clk_pin LOC=B16;
Net ps2Data_pin LOC=E13;
```

Este periférico consta de un único registro `slv_reg0` donde se carga el valor correspondiente a la última tecla pulsada.

La entidad que se instancia desde el interfaz `user_logic` es la siguiente:

```
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;
```

donde `rst` y `clk` son las líneas globales de reset y reloj respectivamente; `ps2CLK` y `ps2Data` son 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`), y que el dato se ha cargado en el registro de datos (`ackData`). En `user_logic`, cada vez que se activa `ldData`, el valor `Datap` se carga en `slv_reg0(24 to 31)` y se activa la señal `ackData`. Esto se realiza mediante el siguiente código:

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

Dentro de la entidad `keyboard` hay otra entidad llamada `ps2KeyboardInterface` donde se realiza el control del protocolo PS2. Esta entidad está definida en el archivo `ps2_kdb.vhd`.

2.3. Control software

En el fichero `*.c` debe definirse la dirección de memoria correspondiente al teclado PS2, que en el ejemplo que proporcionamos es la `0xC3800000`.

```
#define BASE_ADDRESS_PS2 0xC3800000
```

El código para leer una tecla es el siguiente:

```
baseaddr = BASE_ADDRESS_PS2;
Reg32Value = TECLADO_mReadSlaveReg0(baseaddr, 0);
```

Un ejemplo de programa que pide, a través del terminal, que se pulse una tecla y la muestra en el mismo 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);
}
```

Práctica 3

Teclado Matricial o Keypad

3.1. Introducción

Nombre del periférico: `keypad`.

Funcionamiento: Se trata de una matriz de 4 filas por 4 columnas que se conecta a la FPGA a través de 8 líneas. Se suelen conectar las columnas (o filas) a alimentación a través de resistencias de *pull-up*. Los *switches* están conectados a las mismas líneas que las filas. 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. Pasa lo mismo cuando se pulsa un *switch*. Mediante un sistema de *polling* se determina cuál es la tecla presionada. El resultado se envía a través de la entrada `tecla`.

Los conectores son (de izda. a dcha.):

- alimentación
- tierra
- código de retorno (4 bits) correspondiente a las 4 filas
- código de *scan* (5 bits) 4 columnas más *switches*
- `AND` del código de *scan*

El interfaz *hardware* dispone de un registro `slv_reg0` donde se almacena la última tecla pulsada. Cuando se lee este registro (desde microblaze) se borra su contenido. El interfaz (`user_logic.vhd`) se comunica:

1. Por un lado con el procesador microblaze que lee el registro. Si su contenido es distinto de 0, es que se ha pulsado una tecla. Las líneas de este interfaz son `tecla`, `switch`.
2. Con el periférico, mediante 9 líneas, 4 de salida (`S`) conectadas a las filas, y 5 de entrada (`R`) conectadas a las columnas y a los 4 *switches*.

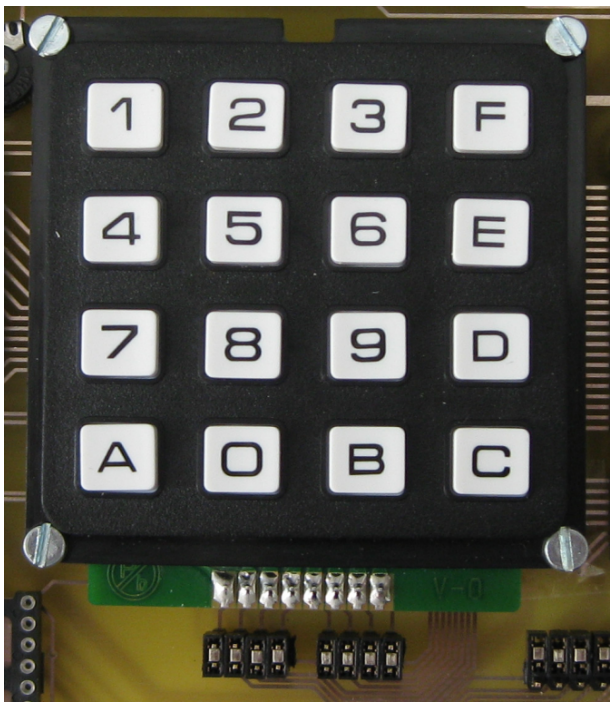


FIGURA 3.1: Teclado matricial 4×4

```
S : out  STD_LOGIC_VECTOR (3 downto 0);
R : in   STD_LOGIC_VECTOR (5 downto 0);
```

3.2. Conexión al sistema

En `user_logic` (y en el `top`) añadir:

```
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 (5 downto 0);

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

Es necesario definirlas como pines externos (dentro de `user_logic` y de `keypad`) 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 la siguiente:

```
net keypad_0_S_pin<0> loc=P12;
net keypad_0_S_pin<0> iostandard=LVCMOS25;
net keypad_0_S_pin<1> loc=J1;
net keypad_0_S_pin<1> iostandard=LVCMOS25;
net keypad_0_S_pin<2> loc=H1;
net keypad_0_S_pin<2> iostandard=LVCMOS25;
net keypad_0_S_pin<3> loc=H3;
net keypad_0_S_pin<3> iostandard=LVCMOS25;

net keypad_0_R_pin<0> loc=G2;
net keypad_0_R_pin<0> iostandard=LVCMOS25;
net keypad_0_R_pin<1> loc=K15;
net keypad_0_R_pin<1> iostandard=LVCMOS25;
net keypad_0_R_pin<2> loc=K16;
net keypad_0_R_pin<2> iostandard=LVCMOS25;
net keypad_0_R_pin<3> loc=F15;
net keypad_0_R_pin<3> iostandard=LVCMOS25;
net keypad_0_R_pin<4> loc=E2;
net keypad_0_R_pin<4> iostandard=LVCMOS25;
net keypad_0_R_pin<4> PULLUP;
net keypad_0_R_pin<5> loc=E1;
net keypad_0_R_pin<5> iostandard=LVCMOS25;
net keypad_0_R_pin<5> PULLUP;
```

La entidad que se instancia desde el interfaz `user_logic` es la siguiente:

```
  component teclaDetect is
    Port ( reloj : in  STD_LOGIC;
```

```
            reset : in  STD_LOGIC;
            S : out  STD_LOGIC_VECTOR (3 downto 0);
            R : in   STD_LOGIC_VECTOR (5 downto 0);
            KeyCode : out  STD_LOGIC_VECTOR (7 downto 0);
            keyPressed: out std_logic;
            swPressed: out std_logic);
end component;
```

Cada vez que se pulsa una tecla (o *switch*), se activa la señal `keyPressed` (`swPressed`) 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`:

```
  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
    elsif (swPressed = '1') then
      slv_reg0(4 to 7) <= KeyCode(3 downto 0); -- El botón pulsado en bits 4-7
    end if;
  end if;
```

Además cada vez que se lee el registro desde microblaze se debería borrar su contenido escribiendo en el mismo un 0.

3.3. Control software

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

```
#define BASE_ADDRESS_KEYPAD 0xC9600000
```

El código para escribir un dato en el registro es:

```
baseaddr = BASE_ADDRESS_KEYPAD;
KEYPAD_mWriteReg(baseaddr, 0,Dato);
```

El código para leer una tecla es el siguiente:

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

Un ejemplo de programa que lee una tecla y la muestra por el hypertérminal es:

```
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;
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);
  /* Se escribe un 0 para borrar la última tecla leída */
  KEYPAD_mWriteReg(baseaddr, 0,0);
}
```

Práctica 4

Matriz de Puntos o Banner

4.1. Introducción

Nombre del periférico: `banner`.

Funcionamiento: la placa forma 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.

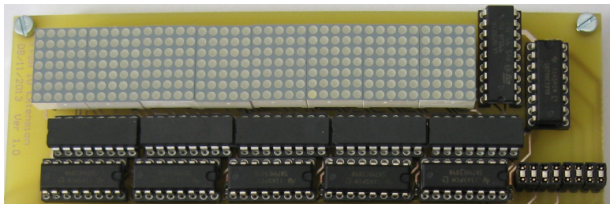


Figura 4.1: Matriz de puntos usada en el laboratorio

Los conectores son (de izda. a dcha.):

- tierra
- alimentación
- entrada serie columnas
- reloj columnas
- entrada serie filas
- reloj filas
- reset
- salida serie columnas (no lo usamos)

- salida serie filas (no lo usamos)

El interfaz hardware dispone de una memoria para almacenar el contenido que se quiere escribir en la matriz. La memoria está formada por 7 filas, y 8 columnas de 5 bits cada una. El interfaz (bannerDesp.vhd) se comunica:

1. Por un lado con el procesador microblaze que actualiza el contenido de la memoria. Las líneas de este interfaz son `fila`, `columna`, `dato` y `load`. Cada vez que `load` se activa, se carga el contenido de `dato` en la posición de memoria correspondiente.

2. Con el periférico, realizando un barrido de todos los pixeles 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).

4.2. Conexión al sistema

En `user_logic` (y en el `top`) añadir:

```
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
);
```

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` del periférico. Es necesario definirlas como pines externos (dentro de `user_logic` y de `banner`) 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 la siguiente:

```
net banner_0_col_serial_out_pin loc=L5;
net banner_0_col_serial_out_pin iostandard=LVCMOS25;
net banner_0_col_clk_pin loc=N2;
net banner_0_col_clk_pin iostandard=LVCMOS25;
net banner_0_row_serial_out_pin loc=M3;
net banner_0_row_serial_out_pin   iostandard=LVCMOS25;
net banner_0_row_clk_pin loc=N1;
net banner_0_row_clk_pin    iostandard=LVCMOS25;
net banner_0_reset_out_pin loc=T13;
net banner_0_reset_out_pin       iostandard=LVCMOS25;
```

La comunicación con el sistema se realiza a través de una `fifo` de escritura (microblaze escribe en la `fifo`). Cada vez que se detecta una escritura en la `fifo`, se lee el dato y se activa la señal de `load`. La estructura de la información es la siguiente:

- La `fila` es el byte menos significativo (bits 5-7): `fila(2 downto 0) <= WFIFO2IP_Data(5 to 7);`
- La `columna` son los bytes 2 y 3 (bits 13-15): `columna(2 downto 0) <= WFIFO2IP_Data(13 to 15);`
- El `dato` son los bytes 4 y 5 (bits 19 al 23): `dato(4 downto 0) <= WFIFO2IP_Data(19 to 23);`

La entidad que se instancia desde el interfaz `user_logic` es la siguiente:

```
 mybanner: bannerDesp port map (
      reset_in       => Bus2IP_Reset,
      clock          => Bus2IP_Clk,
      col_serial_out => col_serial_out,
      col_clk        => col_clk,
      row_serial_out => row_serial_out,
      row_clk        => row_clk,
      reset_out      => reset_out,
      fila           => fila,
      columna        => columna,
      dato           => dato,
      load           => load
);
```

4.3. Control software

En el fichero `*.c` debe definirse la dirección de memoria correspondiente al `banner`, que en el ejemplo que proporcionamos es la `0xC3800000`.

```
#define BANNER_BASE_ADDR 0xC5800000
```

El código para escribir un dato en la FIFO es:

```
baseaddr = BANNER_BASE_ADDR 0xC5800000;
BANNER_mWriteToFIFO(baseaddr, 0, Data);
```

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` a escribir mediante el siguiente comando:

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

Práctica 5

Control de un monitor VGA

En esta práctica aprenderemos a utilizar 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.

5.1. Introducción

El objetivo de esta práctica es diseñar el controlador de un monitor VGA. Dicho controlador debe implementar la funcionalidad básica que nos sirva para utilizar un monitor VGA, tal como una función que nos sirva para colorear los distintos píxeles de nuestra pantalla.

Durante la realización de la práctica aprenderemos el fucionamiento de un monitor VGA, la adición de nuevos periféricos al sistema y el paradigma de diseño hardware/software para realizar un controlador.

5.2. Monitor VGA

Una pantalla de tubos de rayos catódicos (CRT) está formado por: un tubo en forma piramidal 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. Cuando alcanza el final de la fila, apaga

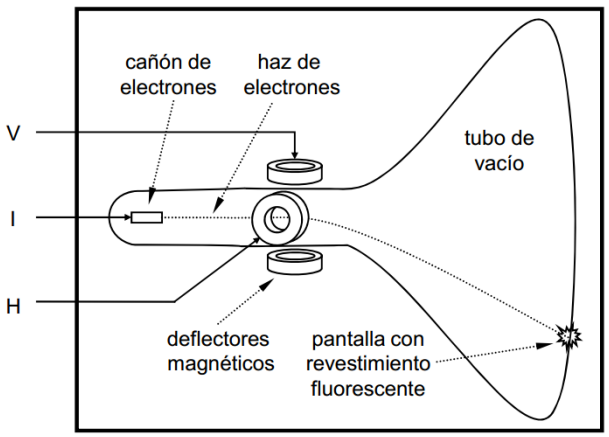


FIGURA 5.1: Diseño de un monitor de tubo de rayos catódicos

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.

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). Sincronización vertical: que marca el comienzo y final de una imagen completa (cuadro o *frame*). Intensidad: que indica la intensidad del haz de electrones.

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

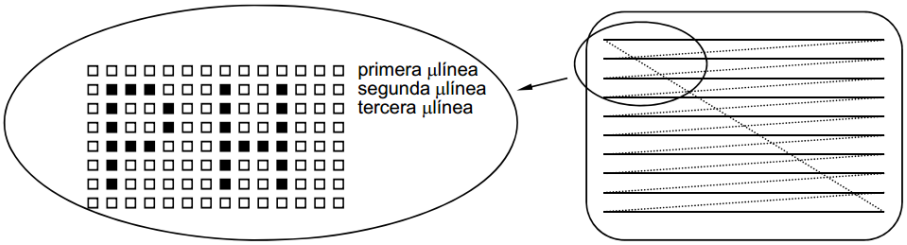


FIGURA 5.2: Ejemplo de barrido de pantalla

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 a 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.

5.3. Adición del periférico al sistema

5.3.1. Señales de entrada/salida y conexión con MicroBlaze

Debajo podemos ver la cabecera de nuestro periférico y sus señales de entrada/salida.

```
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
    Bus2IP_Clk                  : in  std_logic;
    Bus2IP_Reset                : in  std_logic;
    Bus2IP_Data                 : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                   : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                 : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                : out std_logic;
    IP2Bus_WrAck                : out std_logic;
    IP2Bus_Error                : out std_logic;
    IP2WFIFO_RdReq              : out std_logic;
```

```
    WFIFO2IP_Data                : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    WFIFO2IP_RdAck               : in  std_logic;
    WFIFO2IP_AlmostEmpty         : in  std_logic;
    WFIFO2IP_Empty               : in  std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
```

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

- `hsyncb` (1 bit): señal de sincronismo horizontal.
- `vsyncb` (1 bit): señal de sincronismo vertical.
- `rgb` (9 bits): señal que indica el color.

El resto de señales corresponden a las señales del bus PLB utilizado para interconectar MicroBlaze con el controlador hardware.

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.

5.3.2. Cómo importar el periférico en EDK

Para incorporar el periférico al sistema descomprima el fichero '`VGA_core.zip`' en el directorio raíz del proyecto básico de EDK (es decir, a la misma altura que el fichero '`system.xmp`').

Ahora debemos importar dicho periférico desde EDK. Pulsamos sobre '**Hardware**' ->'**Create or Import Peripheral**'. En ese momento nos aparecerán una serie de ventanas de diálogo. A continuación, se va a exponer la lista de pasos necesarios para importar dicho periférico a través de las 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**'.
3. A continuación pulsamos '**Next**'.
4. En '**Name**' escribimos 'pantalla' y pulsamos '**Next**'.
5. A continuación pulsamos '**Next**'.
6. 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`'). A continuación, pulsamos '**Next**'.

7. Pulsamos '**Add Files**', seleccionamos todos los ficheros que estaban en el fichero comprimido y pulsamos '**Next**'.
8. Seleccionamos '**PLBV46 Slave (SPLB)**' y pulsamos '**Next**'.
9. Pulsamos '**Next**'.
10. En '**Parameter determine high address**' seleccionamos '**C_HIGHADDR**' y pulsamos '**Next**'.
11. Deseleccionamos '**Select and configure interrup(s)**' y pulsamos '**Next**'.
12. Pulsamos '**Next**'.
13. Pulsamos '**Next**'.
14. 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`'.

5.3.3. Control del periférico

Para llevar a cabo un control adecuado de la pantalla VGA (de 120 x 159 píxeles) se ha desarrollado el *driver* que implementa las siguientes funciones:

```
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)(fila*(256)+columna)));
}
```

```
void PANTALLA_inicializa(){

    int i;
    int j;

    PANTALLA_mResetWriteFIFO( XPAR_PANTALLA_0_BASEADDR );

    for (i=0; i < 120; i++){
        for (j=0; j < 159; j++){
            while (PANTALLA_mWriteFIFOFull(XPAR_PANTALLA_0_BASEADDR)){}
            PANTALLA_pintaPixel(i, j, ROJO + VERDE + AZUL);
        }
```

```
    }
}
```

La función '`PANTALLA_inicializa`' borra toda la pantalla dejándola en blanco. La función '`PANTALLA_pintaPixel`' es útil para ordenar el coloreado de un píxel indicándole su posición (fila y columna) y el color que queremos que tome.

Para establecer un color determinado debe seguirse el siguiente 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 siguientes colores:

```
#define ROJO                    0xE0000000
#define VERDE                   0x1C000000
#define VERDE_OSCURO            0x0C000000
#define AZUL                    0x03800000
```

El ejemplo de uso que se muestra debajo, 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 < 120; i++){
        for (j=0; j < 159; 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;
}
```

Práctica 6

Control de una pantala 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.

6.1. Introducción

El objetivo de esta práctica es diseñar el controlador de una pantalla LCD. Dicho controlador debe implementar las funcionalidades básicas que nos sirvan para utilizar un LCD, tales como una función que nos sirva para inicializar la pantalla, situarnos en las distintas posiciones de la pantalla y ordenar la escritura de un carácter.

Durante la realización de la práctica aprenderemos el fucionamiento de una pantalla LCD, la adición de nuevos periféricos al sistema y el paradigma de diseño hardware/software para realizar un controlador.

6.2. Pantalla LCD

Un LCD (ver figura 6.1) 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

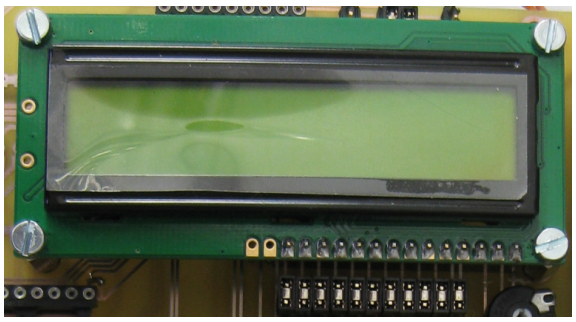


FIGURA 6.1: Ejemplo de una pantalla LCD real

usuario del patrón de hasta 8 nuevos caracteres. 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/ea079/LCDDisplay.pdf`). Para la implementación del controlador hardware se ha seguido el diagrama de estados que aparece en el mismo.

6.3. Adición del periférico al sistema

6.3.1. Señales de entrada/salida y conexión con MicroBlaze

A continuación podemos ver la cabecera de nuestro periférico y sus señales de entrada/salida.

```
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
    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); --data signals for lcd
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                  : in  std_logic;
    Bus2IP_Reset                : in  std_logic;
    Bus2IP_Data                 : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                   : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                 : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                : out std_logic;
    IP2Bus_WrAck                : out std_logic;
    IP2Bus_Error                : out std_logic;
    IP2WFIFO_RdReq              : out std_logic;
    WFIFO2IP_Data               : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    WFIFO2IP_RdAck              : in  std_logic;
    WFIFO2IP_AlmostEmpty        : in  std_logic;
    WFIFO2IP_Empty              : in  std_logic
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
```

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.

El resto de señales corresponden a las señales del bus PLB utilizado para interconectar MicroBlaze con el controlador hardware.

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.

6.3.2. Cómo importar el periférico en EDK

Para incorporar el periférico al sistema descomprima el fichero '`LCD_core.zip`' en el directorio raíz del proyecto básico de EDK (es decir, a la misma altura que el fichero '`system.xmp`').

Ahora debemos importar dicho periférico desde EDK. Pulsamos sobre '**Hardware**' ->'**Create or Import Peripheral**'. En ese momento nos aparecerán una serie de ventanas de diálogo. A continuación, se va a exponer la lista de pasos necesarios para importar dicho periférico a través de las 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**'.
3. A continuación pulsamos '**Next**'.
4. En '**Name**' escribimos '`vga`' y pulsamos '**Next**'.
5. A continuación pulsamos '**Next**'.
6. 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`'). A continuación, pulsamos '**Next**'.
7. Pulsamos '**Add Files**', seleccionamos todos los ficheros que estaban en el fichero comprimido y pulsamos '**Next**'.
8. Seleccionamos '**PLBV46 Slave (SPLB)**' y pulsamos '**Next**'.
9. Pulsamos '**Next**'.
10. En '**Parameter determine high address**' seleccionamos '**C_HIGHADDR**' y pulsamos '**Next**'.
11. Deseleccionamos '**Select and configure interrup(s)**' y pulsamos '**Next**'.
12. Pulsamos '**Next**'.
13. Pulsamos '**Next**'.
14. 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`'.

6.3.3. Control del periférico

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

```
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);
}
```

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

```
#define CLEAR_DISPLAY_CMD               0x00000001
#define RETURN_HOME_CMD                 0x00000002
#define WRITE_CMD                       0x00000200
#define FIRST_ROW                       0x00000080
#define SECOND_ROW                      0x000000C0
```

A continuación, explicaremos cada uno de ellos: '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 caracter de la primera fila y 'SECOND_ROW' direcciona el cursor en el primer caracter de la segunda fila.

El ejemplo de uso que se muestra debajo, da como resultado la escritura en la primera fila de los caracteres 'HOLA MUNDO! :)' y de los caracteres 'abcdefg 0123456' en la segunda fila. El comando 'WRITE_CMD' ordena la escritura de un caracter, pero no establece qué caracter debe escribir (los dos bytes menos significativos valen cero). Por ello, para indicar la escritura de un caracter se lo sumanos al comando (OR lógica).

```
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;
}
```

Práctica 7

Control de un zumbador

En esta práctica aprenderemos a utilizar 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.

7.1. Introducción

El objetivo de esta práctica es diseñar el driver de un zumbador. Dicho driver debe implementar las funcionalidades básicas que nos sirvan para utilizar un zumbador, tales como una función para indicarle que suene y otra para silenciarlo.

Durante la realización de la práctica aprenderemos el funcionamiento de un zumbador, la adición de nuevos periféricos al sistema y la utilización de las entradas/salidas de propósito general para escritura.

7.2. Zumbador

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 piezo eléctrico. 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 (ver figura 7.1(a)), como veréis 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 7.1(b).

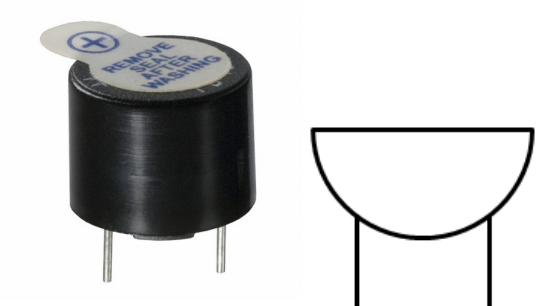


FIGURA 7.1: Ejemplo de zumbador real (a) y símbolo electrónico (b)

Su funcionamiento es muy sencillo y vamos a explicarlo a través del circuito de la figura 7.2(a). 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.

Finalmente, la figura 7.2(b) muestra el circuito realizado 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`).

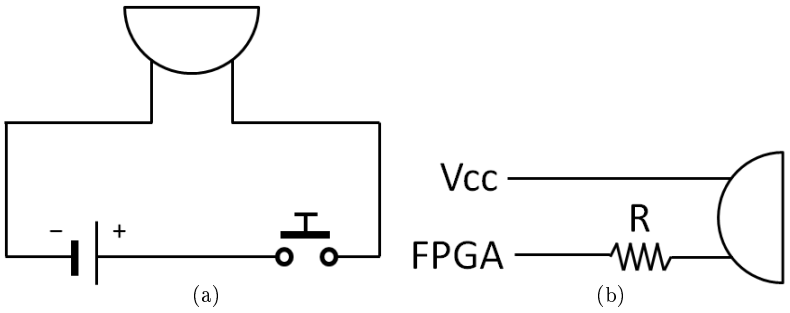


FIGURA 7.2: Ejemplo de uso de un zumbador en un circuito sencillo (a) y conexionado en la placa real (b)

7.3. Adición del periférico al sistema

7.3.1. Señales de entrada/salida y conexión con MicroBlaze

Para esta práctica se va a disponer de una única línea de salida que será controlada por el MicroBlaze a través de una `GPIO`.

7.3.2. Cómo importar el periférico en EDK

En la pestaña '`IP Catalog`' debajo de '`General Purpose IO`' aparecerá '`XPS General Purpose IO`'. Para añadirlo al sistema, lo seleccionamos y lo arrastramos a la pestaña '`Bus Interfaces`' y lo renombramos como '`zumbador`'. Conectamos el periférico al bus PLB y en la pestaña '`Ports`' configuramos su puerto como externo. Finalmente, añadimos la asignación del puerto externo a los pines de la FPGA en el fichero '`*.ucf`'.

7.3.3. Control del periférico

Para llevar a cabo un control adecuado del zumbador se ha desarrolado el driver que implementa las siguientes 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, 0x00);
                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);
}
```

La función '`ZUMBADOR_inicializa`' configura el `GPIO` asociado al zumbador como salida. La función '`ZUMBADOR_calla`' silencia el zumbador. La función '`ZUMBADOR_suena`' hace que el zumbador emita sonido en la frecuencia preestablecida en su fabricación.

El ejemplo de uso que se muestra debajo 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 < 0x00070000; i++){} // retardo

                ZUMBADOR_calla(&Gpio_zumbador);
                xil_printf("Calla\r\n");
                for(i = 0; i < 0x00070000; i++){} // retardo
        }

    return 0;
}
```

Práctica 8

Control de un altavoz

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

8.1. Introducción

El objetivo de esta práctica es diseñar el controlador de un altavoz. Dicho controlador debe implementar las funcionalidades básicas que nos sirvan para utilizar un altavoz, tales como una función para indicarle que suene en una determinada frecuencia y otra para silenciarlo.

Durante la realización de la práctica aprenderemos el funcionamiento de un altavoz y la generación de notas musicales, la adición de nuevos periféricos al sistema y el paradigma de diseño hardware/software para realizar un controlador.

8.2. Altavoz

Un altavoz (ver figura 8.1(a)) 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 dento 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 8.1(b).

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 8.2 muestra las frecuencias que corresponden a cada una de las notas musicales según la octava.

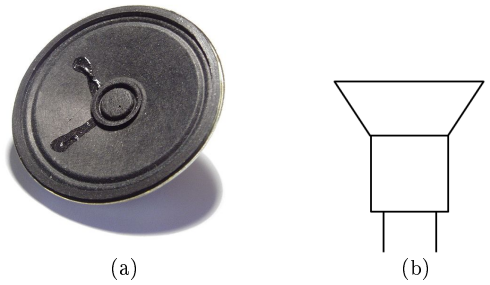


FIGURA 8.1: Ejemplo de un altavoz real (a) y símbolo electrónico (b)

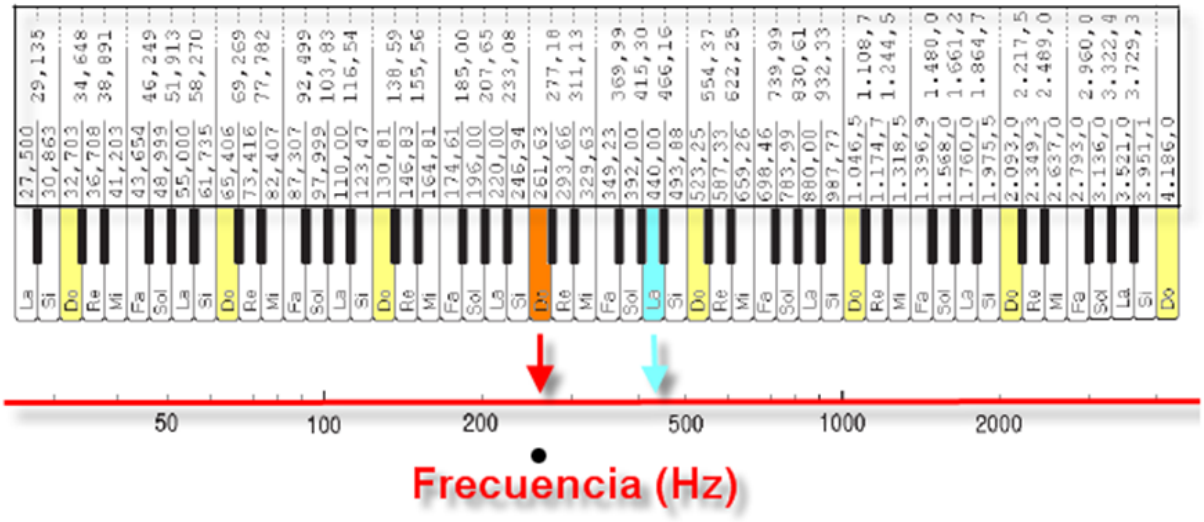


FIGURA 8.2: Frecuencia de las distintas notas musicales

Su funcionamiento es muy sencillo y vamos a explicarlo a través del circuito de la figura 8.3(a). 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 8.3(b) muestra el circuito realizado en las placas del laboratorio. En nuestro caso concreto, dejamos fijo el valor de polariación `Vcc`, por lo que si queremos que suene deberemos generar una frecuencia audible en el pin que sale de la FPGA.

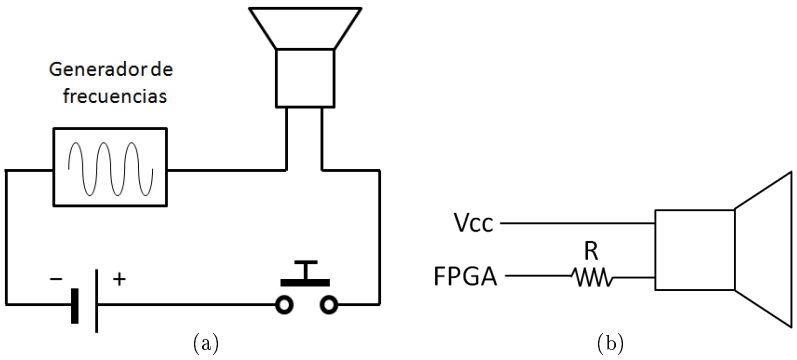


FIGURA 8.3: Ejemplo de uso de un altavoz en un circuito sencillo (a) y conexionado en la placa real (b)

8.3. Adición del periférico al sistema

8.3.1. Señales de entrada/salida y conexión con MicroBlaze

Debajo podemos ver la cabecera de nuestro periférico y sus señales de entrada/salida.

```
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 ----------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                  : in  std_logic;
    Bus2IP_Reset                : in  std_logic;
```

```
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic;
    IP2WFIFO_RdReq                 : out std_logic;
    WFIFO2IP_Data                  : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    WFIFO2IP_RdAck                 : in  std_logic;
    WFIFO2IP_AlmostEmpty           : in  std_logic;
    WFIFO2IP_Empty                 : in  std_logic
    -- DO NOT EDIT ABOVE THIS LINE --------------------
  );
```

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

- `sonido` (1 bit): señal de salida con la frecuencia que se desee.

El resto de señales corresponden a las señales del bus PLB utilizado para interconectar MicroBlaze con el controlador hardware.

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.

8.3.2. Cómo importar el periférico en EDK

Para incorporar el periférico al sistema descomprima el fichero '`ALTAVOZ_core.zip`' en el directorio raíz del proyecto básico de EDK (es decir, a la misma altura que el fichero '`system.xmp`').

Ahora debemos importar dicho periférico desde EDK. Pulsamos sobre '`Hardware`' ->'`Create or Import Peripheral`'. En ese momento nos aparecerán una serie de ventanas de diálogo. A continuación, se va a exponer la lista de pasos necesarios para importar dicho periférico a través de las 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**'.
3. A continuación pulsamos '**Next**'.
4. En '**Name**' escribimos '`altavoz`' y pulsamos '**Next**'.

5. A continuación pulsamos '**Next**'.
6. 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`'). A continuación, pulsamos '**Next**'.
7. Pulsamos '**Add Files**', seleccionamos todos los ficheros que estaban en el fichero comprimido y pulsamos '**Next**'.
8. Seleccionamos '**PLBV46 Slave (SPLB)**' y pulsamos '**Next**'.
9. Pulsamos '**Next**'.
10. En '**Parameter determine high address**' seleccionamos '**C_HIGHADDR**' y pulsamos '**Next**'.
11. Deseleccionamos '**Select and configure interrupt(s)**' y pulsamos '**Next**'.
12. Pulsamos '**Next**'.
13. Pulsamos '**Next**'.
14. 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`'.

8.3.3. Control del periférico

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

```
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);
}
```

La función '`my_delay`' es útil para establecer un tiempo de espera antes de ejecutar la siguiente instrucción de código. La función '`ALTAVOZ_calla`' silencia el altavoz. La función '`ALTAVOZ_suena`' se utiliza para fijar la frecuencia de la señal de salida indicándoles qué nota y en qué octava queremos que suene. Para indicar la nota concreta que debe sonar se definen las siguientes notas y octava:

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

#define OCTAVA          3
```

El ejemplo de uso que se muestra debajo se produce una escala musical ascende y tras un silencio, la misma escala pero de forma descencente.

```
int main()
{
    print("Práctica altavoz\n\r");


    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");
    ALTAVOZ_suena(FA, OCTAVA);
    my_delay(300); print("FA\n\r");
    ALTAVOZ_suena(SOL, OCTAVA);
    my_delay(300); print("SL\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");
    ALTAVOZ_suena(DO, (OCTAVA-1));
```

```
    my_delay(300); print("DO\n\r");
    ALTAVOZ_suena(SI, OCTAVA);
    my_delay(300); print("SI\n\r");
    ALTAVOZ_suena(LA, OCTAVA);
    my_delay(300); print("LA\n\r");
    ALTAVOZ_suena(SOL, OCTAVA);
    my_delay(300); print("SL\n\r");
    ALTAVOZ_suena(FA, OCTAVA);
    my_delay(300); print("FA\n\r");
    ALTAVOZ_suena(MI, OCTAVA);
    my_delay(300); print("MI\n\r");
    ALTAVOZ_suena(RE, OCTAVA);
    my_delay(300); print("RE\n\r");
    ALTAVOZ_suena(DO, OCTAVA);
    my_delay(300); print("DO\n\r");
    ALTAVOZ_calla();

    return 0;
}
```

Práctica 9

Control del encedido de un LED RGB a través de un PWM

En esta práctica aprenderemos en qué consiste un modulador de anchos de pulso (o PWM, de sus siglas en inglés *Pulse Width Modulation*). Este tipo de circuitos es muy utilizado para el control de diversos sistemas digitales: servo-motores, elementos termo-eléctricos, LED... En esta práctica utilizaremos un PWM para controlar la intensidad de encendido de las 3 componentes lumínicas de un LED RGB (Rojo (R), Verde (G) y Azul (B)).

9.1. PWM

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 perído de oscilación de la señal ésta permanece a 0 y qué porcentaje permanece a 1.

A este porcentaje se le denomina comúnmente *duty cycle*, el cual 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 9.1.

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. Esto es lo que haremos en esta práctica.

9.2. LED RGB

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

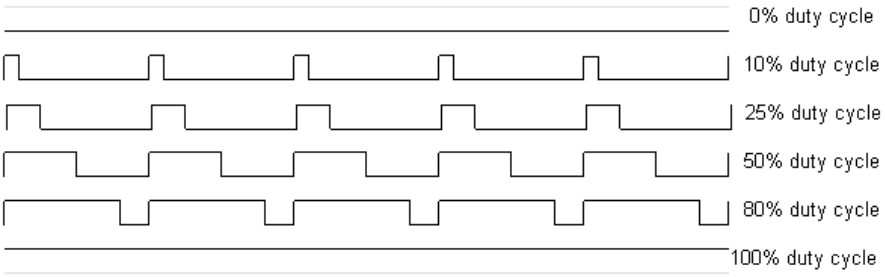
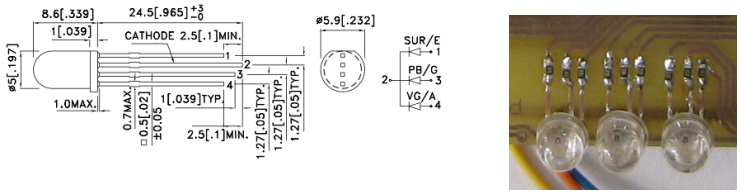


FIGURA 9.1: Funcionamiento de un modulador de ancho de pulsos

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 9.2(a). Estas 3 patillas únicamente admiten 2 valores de entrada: encendido ('1') y apagado ('0').



(a) (b)

FIGURA 9.2: Esquema de los LED RGB que se utilizarán en esta práctica y su disposición en la placa de expansión

9.3. Descripción hardware del periférico *LED_RGB*

Para el desarrollo de esta práctica, se proporciona un periférico de EDK que servirá para controlar el encendido del color rojo del LED RGB. Se puede encontrar en el directorio `<dir_proyecto>/pcores/` con el nombre `led_RGB_v1_00_a`. Como en otras prácticas, el código fuente del periférico se proporciona a través de dos archivos fuente `.vhd`, en este caso: `led_RGB.vhd` y `user_logic.vhd`. Este último fichero contiene la implementación personalizada de este periférico.

En la `architecture` del periférico hay 3 registros de 8 bits accesibles por el usuario: `reg0`, `reg1` y `reg2`. Escribiendo en ellos un valor comprendido entre 0 y 255, estos registros determinan la intensidad de encendido de cada una de las componentes lumínicas del LED RGB (0: mínima intensidad; 255: máxima intensidad). La escritura en estos registros se realiza en el proceso VHDL `SLAVE_REG_WRITE_PROC` que se encuentra en la arquitectura de la entidad `user_logic`.

Dicha arquitectura también contiene un proceso llamado `PWM1` que implementa el PWM, el cual se muestra en el siguiente fragmento de código. 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 `output_red` es 1, en caso contrario, `output_red` será 0. De este modo, cuanto más alto sea el valor de referencia, mayor tiempo estará esta señal a 1.

```
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 = 2n-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;
```

Finalmente, dicho periférico contiene, además de la interfaz de comunicación con el bus PLB, un puerto de salida llamado `red`, de tipo std_logic, la cual está conectada a la salida del módulo comparador descrito anteriormente.

9.4. Software de control del periférico *LED_RGB*

En el proyecto software que se proporciona como parte del proyecto base de esta práctica, se incluye un fichero `main.c` con un código de ejemplo de uso del periférico `led_RGB`.

La función `main()` contiene un pequeño menú para que el usuario seleccione la intensidad de encendido de cada uno de los colores del LED a través de una variable de tipo `u8` (`unsigned` de 8 bits). 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_0_BASEADDR` es la dirección base de acceso al periférico `led_RGB`, 0 es un valor constante con el que se debe invocar esta función, e `intensidadR` es el valor que se escribirá en el registro.

Análogamente para el registro 1, las funciones `LED_RGB_mWriteSlaveReg1` y `LED_RGB_mWriteSlaveReg2` escriben valores en los registros accesibles por el usuario 1 y 2, respectivamente.

9.5. Ficheros fuente

En esta práctica se proporcionan los siguientes ficheros fuente:

- `Practica_PWM_LED_RGB.rar`, el cual contiene comprimido el proyecto EDK que servirá de base para el desarrollo del resto de la práctica.

Práctica 10

Control de un par de dispositivos emisor-receptor de infrarrojos

En esta práctica aprenderemos a controlar un par de dispositivos emisor-receptor de infrarrojos. Así, aprenderemos a enviar un código a través del emisor de infrarrojos y a capturarlo con el receptor.

10.1. Dispositivo emisor de infrarrojos

El dispositivo emisor de infrarrojos utilizado en esta práctica 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 10.1).

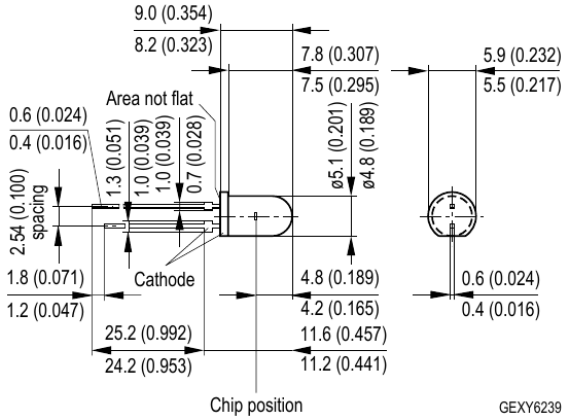


Figura 10.1: Modelo de chip emisor de infrarrojos utilizado en esta práctica

10.2. Dispositivo receptor de infrarrojos

El dispositivo receptor de infrarrojos que utilizaremos en esta práctica (figura 10.2(a)) 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.

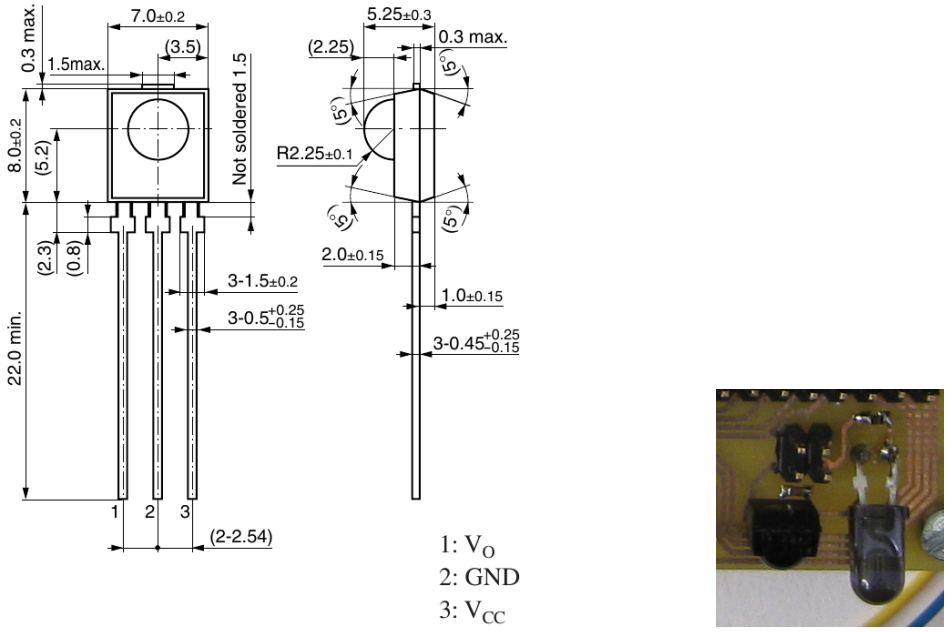


FIGURA 10.2: Modelo de chip receptor de infrarrojos utilizado en esta práctica y la disposición del emisor-receptor en la placa de expansión

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 trabajan con una frecuencia portadora de 38 KHz (figura 10.2(b)).

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 10.3 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.

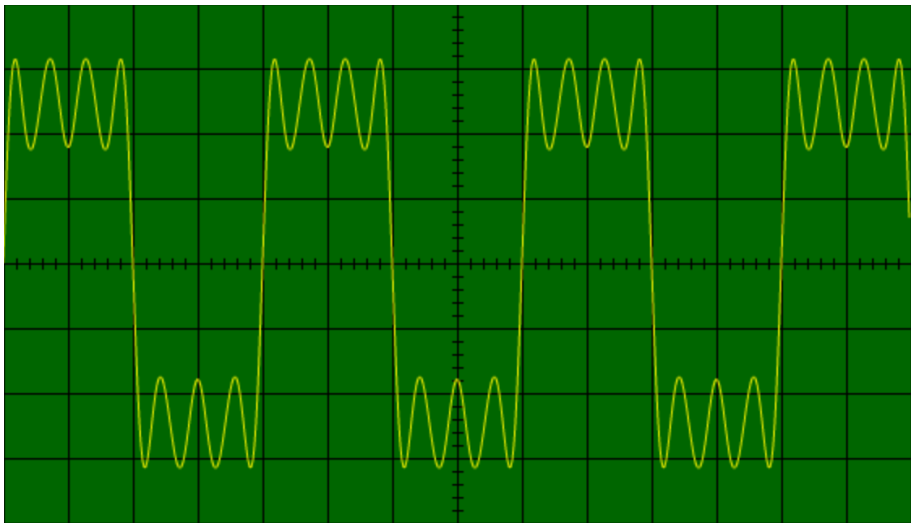


FIGURA 10.3: Ejemplo de una portadora de 38 KHz modulada con una onda de 9 KHz

10.3. Descripcion hardware del periférico *ER_Infrared*

Para el desarrollo de esta práctica, se proporciona un periférico de EDK que servirá para controlar el envío y recepción de señales infrarrojas con nuestros dispositivos emisor y receptor. Se puede encontrar en el directorio `<dir_proyecto>/pcores/` con el nombre `ER_Infrared_v1_00_a`. Como en otras prácticas, el código fuente del periférico se proporciona a través de dos archivos fuente `.vhd`, en este caso: `ER_Infrared.vhd` y `user_logic.vhd`. Este último fichero contiene la implementación personalizada de este periférico.

En la `architecture` del periférico hay 1 registro de 8 bits accesible por el usuario: `slv_reg0`. Escribiendo en él un valor distinto de 0, el periférico comenzará a enviar una señal de frecuencia 1 Hz con una frecuencia portadora de 38 KHz a través de su salida `output`. 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).

10.4. Software de control del periférico *ER_Infrared*

En el proyecto software que se proporciona como parte del proyecto base de esta práctica, se incluye un fichero `main.c` con un código de ejemplo de uso del periférico `ER_Infrared`.

La función `main()` contiene un pequeño menú para que el usuario determine si se desea activar o desactivar la emisión de infrarrojos. Para ello, se utiliza una variable de tipo `u8` (`unsigned` de 8 bits). Así, se escribe un valor distinto de 0 (en este caso, `0xFF`) en el registro `slv_reg0` invocando a la función:

```
ER_INFRARED_mWriteSlaveReg0 (XPAR_ER_INFRARED_0_BASEADDR, 0, 0xFF);
```

donde `XPAR_ER_INFRARED_0_BASEADDR` es la dirección base de acceso al periférico `ER_Infrared`, 0 es un valor constante con el que se debe invocar esta función, 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.

10.5. Ficheros fuente

En esta práctica se proporcionan los siguientes ficheros fuente:

- `Practica_ER_Infrared.rar`, el cual contiene comprimido el proyecto EDK que servirá de base para el desarrollo del resto de la práctica.

Práctica 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.

11.1. Introducción

El objetivo de esta práctica es diseñar el sistema de control de 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.

Durante la realización del proyecto aprenderemos el funcionamiento de diversos dispositivos, como un motor paso a paso, un conversor analógico-digital y sensores de luz.

11.2. Motor paso a paso

Estos motores 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 las figuras 11.1(a) hasta la 11.1(d).

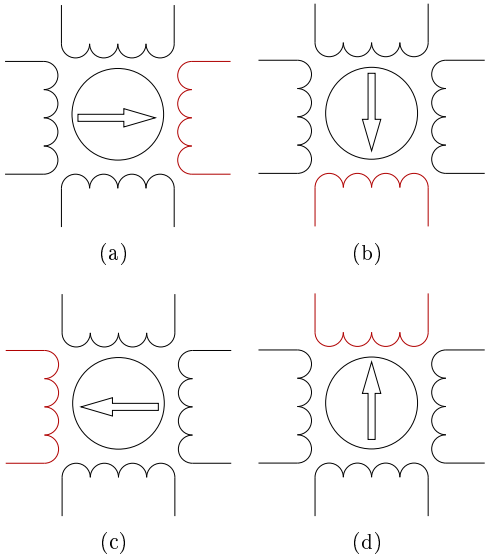


FIGURA 11.1: Funcionamiento de un motor paso a paso

11.2.1. Motor paso a paso bipolar

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 11.2. Los diodos sirven de protección frente a la fuerza contraelectromotriz que genera la bobina al cambiar su polaridad.

Por ejemplo, activando simultáneamente los transistores `T1` y `T4`, la intensidad circula por la bobina `L1` en el sentido indicado en la figura 11.3(a), mientras que activando `T3` y `T2` circula como en 11.3(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.

Existen circuitos integrados especializados que constituyen un puente H, como por ejemplo el L293B.

11.2.2. Motor paso a paso unipolar

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 11.4 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'

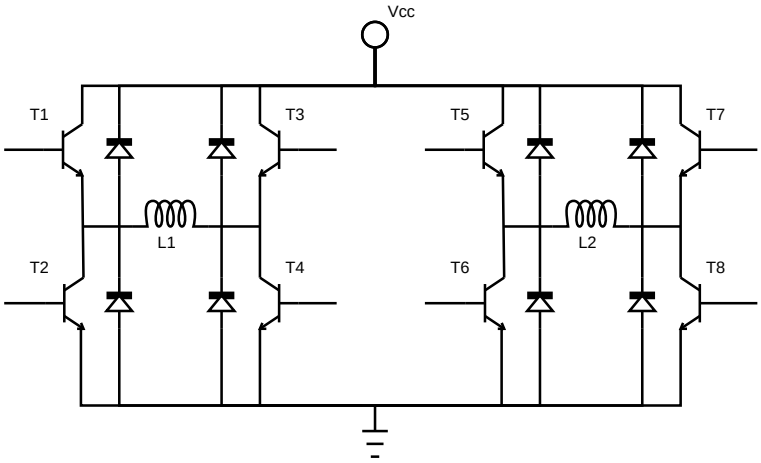


FIGURA 11.2: Configuración del puente-H

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.

11.3. Descripción *hardware* del periférico `motorstep`

En nuestra práctica emplearemos un motor unipolar. El módulo VHDL que implementa el control de dicho motor se puede encontrar en el directorio `<dir_proyecto>/pcores/`. Lo hemos llamado `motorstep` y tiene la siguiente definición:

```
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;
```

Pasaremos a describir cada una de las señales:

- 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.
- `rst` es la señal global de *reset*.

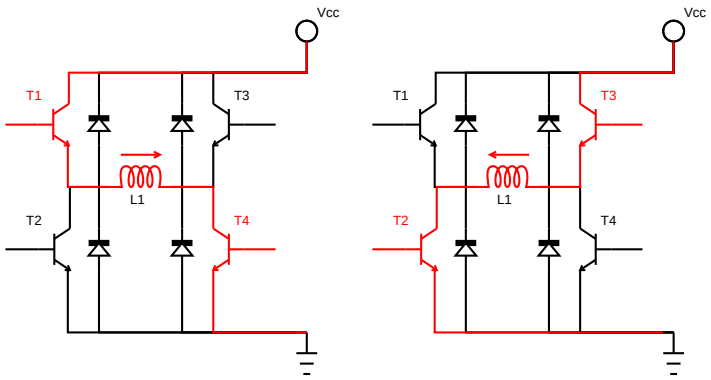


FIGURA 11.3: Funcionamiento del puente-H: activando simultáneamente `T1` y `T4` (a), la intensidad a través de la bobina `L1` circula en sentido contrario que al activar `T2` y `T3` (b)

- La señal `dir` define el sentido de rotación del motor.
- Cuando la señal `stop` vale 1, el motor está parado; 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. De esta manera se consigue mayor precisión en la posición del motor, a cambio de perder algo de fuerza.
- Las señales `motor` son las líneas de salida de control del motor. Atacan directamente al *driver* del motor.
- En `step` tenemos disponible la codificación del paso del motor actual. Al utilizar `halfstep`, el número de pasos es 8.

11.4. Control software del periférico `motorstep`

En el proyecto *software* que se proporciona como parte del proyecto base de esta práctica se incluye un fichero `testperiph.c` con el código de ejemplo de uso del periférico `motorstep`.

La función `main()` hace girar el motor primero en un sentido y luego en sentido contrario escribiendo en el registro de control `motor_ctl`. Este registro tiene la siguiente estructura:

```
| d | s | h | nnnn |
```

donde `d` indica la dirección de giro, `s` es la señal de `stop`, `h` vale 1 si estamos utilizando la opción de medios pasos, y `nnnn` representa el número de pasos que queremos que gire el motor.

Así, para que el motor comience a girar (`s=0`) en sentido horario (`d=1`), pasos completos (`h=0`), y avance 15 pasos (`nnnn=15`), debemos hacer:

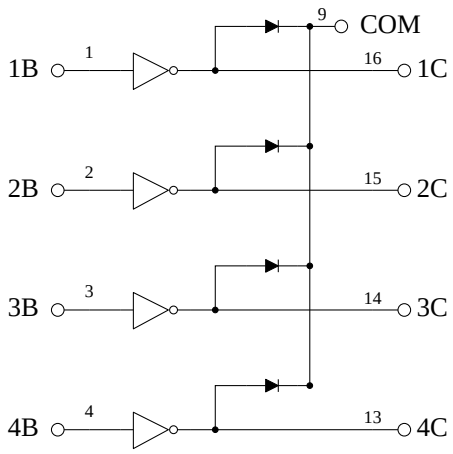


FIGURA 11.4: Esquema del *driver* de motor ULN2003

```
Data=0x9E000000;
MOTOR_HW_mWriteReg(MOTOR_HW_BASEaddress, 0, Data);
```

En cada paso, el controlador del motor va decrementando el contador `nnnn`. Cuando llega a 0, se detiene el motor y el bit `s` se pone al valor '1'. Por tanto, leyendo el mismo registro `motor_ctl` e inspeccionando el valor del bit `s`, podemos saber si el motor está girando o no:

```
Data=MOTOR_HW_mReadReg(MOTOR_HW_BASEaddress, 0);
while (!(Data & 0x40000000))
{
  Data=MOTOR_HW_mReadReg(MOTOR_HW_BASEaddress, 0);
}
```

A través de la consola serie muestra el contenido del registro de sólo lectura `motor_step`, donde el controlador del motor va escribiendo el paso actual del motor, desde 0 hasta 7. Para leer el registro `motor_step`:

```
Data=MOTOR_HW_mReadReg(MOTOR_HW_BASEaddress, 4);
```

11.5. Ficheros fuente

En esta práctica se proporcionan los siguientes ficheros fuente:

- `practica_motor.rar`, que contiene comprimido el proyecto EDK que servirá de base para el desarrollo del resto de la práctica.

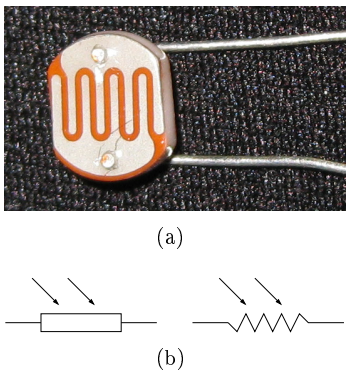
Práctica 12

Conversión analógico/digital

En esta práctica aprenderemos a manejar un conversor analógico-digital que nos permitirá leer la luz incidente en dos sensores LDR.

12.1. Fotoresistencias 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 (12.1(a)). 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 12.1(b).



(a)

(b)

FIGURA 12.1: Ejemplo de fotoresistencia LDR (a) y símbolos electrónicos (b)

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 nuestro proyecto esto 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 fotoresistencias dispuestas como en la figura 12.2. 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.

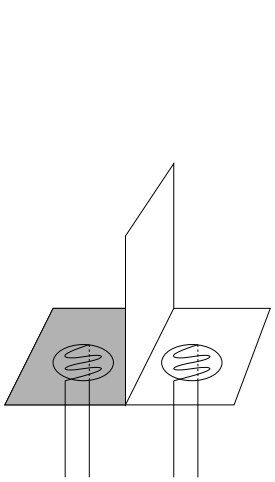


FIGURA 12.2: Disposición física de las LDR

12.2. Sensor de temperatura LM35

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/°C. Sin necesidad de ningún componente externo adicional, proporciona una precisión típica de 1/4°C. Por ejemplo, para una temperatura ambiente de 23,5°C, la salida del sensor es de 0,235V.

En la figura 12.3(a) se puede ver el esquema de conexión y el patillaje del LM35.

12.3. Sensor de flexión

En esta sección se describen las características y el funcionamiento del sensor de flexión incluido en la placa de prototipado. Dicho sensor (Figura 12.10(a)) es un dispositivo de 56 mm de longitud activa cuya resistencia varía a medida que se flexiona. Es muy importante no doblar el dispositivo fuera del rango utilizable (ver Figura 12.10(b)), y siempre fuera de la zona de los pines o conexión.

Cuando el sensor no se encuentra doblado, su resistencia es de 25KΩ. Sin embargo, cuando éste se dobla, tiene una resistencia dentro del rango [45KΩ–125KΩ], tal y como se refleja en la figura 12.11.

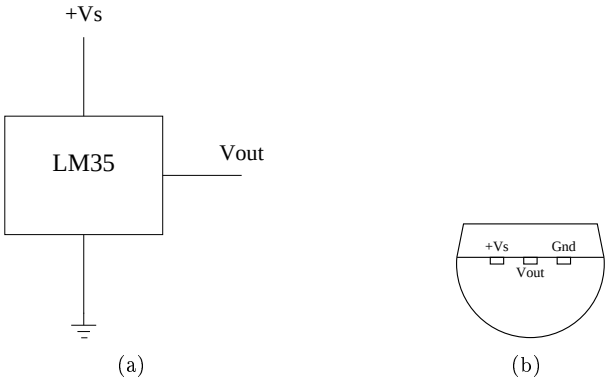


FIGURA 12.3: Sensor de temperatura LM35. Aplicación típica (a) y patillaje del encapsulado plástico TO-92 (b)

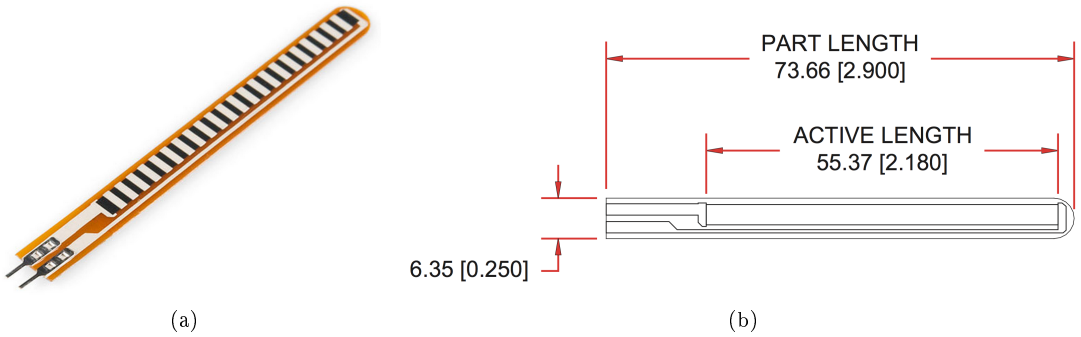


FIGURA 12.4: Sensor de flexión incluido en la placa de prototipado (a) y su zona utilizable (b)

Este sensor ha sido incluido en la placa de prototipado siguiendo el esquema de la figura 12.12. De esta manera, el potencial eléctrico en el pin `ADC` de la placa disminuye a medida que aumenta la flexión del sensor (y en consecuencia, su resistencia).

12.4. Sensor de fuerza resistivo circular

En esta sección se describen las características y el funcionamiento del sensor de fuerza incluido en la placa de prototipado. Dicho sensor (figura 12.13(a)) es un dispositivo de 54 mm de longitud, que incluye un área sensible circular de 18,3 mm de diámetro (figura 12.13(b)) sobre la cual se puede aplicar una fuerza dentro del rango 0 a 10 Kg. A mayor fuerza aplicada, menor será su resistencia. Estos sensores son, en general, fáciles de instalar y de utilizar, pero no son muy precisos. Sin embargo, son más que suficientes para realizar las prácticas que se proponen en esta asignatura.

Cuando el sensor no recibe ninguna fuerza, su resistencia es de aproximadamente 1MΩ. Sin embargo, a medida que éste recibe una fuerza en su zona sensible, su resistencia disminuye, tal y como se indica en la figura 12.14(a) (nótese que la gráfica es doblemente logarítmica).

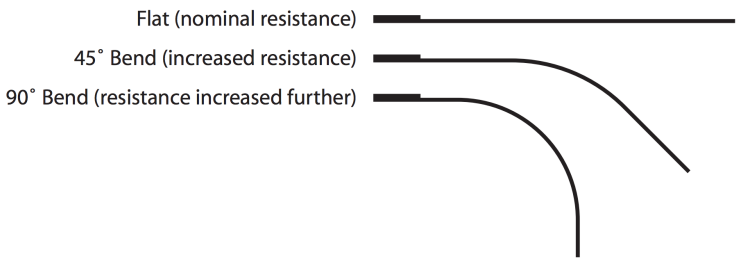
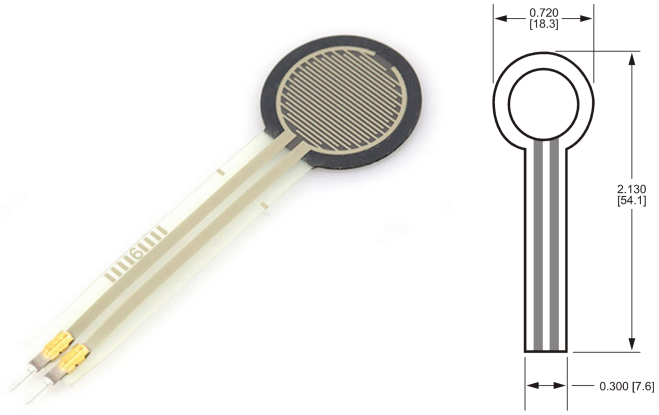


FIGURA 12.5: Variación de la resistencia del sensor de flexión a medida que éste se dobla

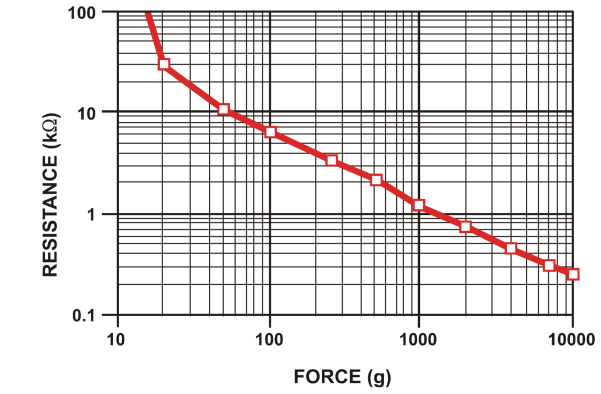
FIGURA 12.6: Integración del sensor de flexión en la placa de prototipado



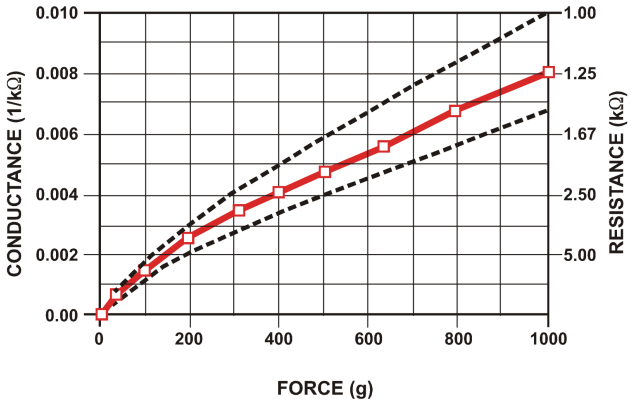
(a) (b)

FIGURA 12.7: Sensor de fuerza incluido en la placa de prototipado (a) y sus dimensiones (b)

Como se puede observar, existe una fuerza de aproximadamente 20 gramos que "activa" el dispositivo y que hace que su resistencia disminuya de más de 100KΩ a aproximadamente 10KΩ. A partir de ese punto, la resistencia del dispositivo disminuye a medida que aumenta la presión, hasta que ésta alcanza los 10000 gramos, punto en el que la respuesta del sensor satura. La figura 12.14(b) muestra en detalle la respuesta de este sensor para fuerzas pequeñas. En este caso, la respuesta se expresa en términos de conductancia y de resistencia (que son magnitudes inversas la una respecto a la otra). Nótese que, a medida que disminuye la presión (para presiones menores de 100 gramos), la resistencia aumenta muy rápidamente, al aproximarse el valor de la conductancia a 0.



(a)



(b)

FIGURA 12.8: Variación de la resistencia del sensor de fuerza incluido en la placa de prototipado para diferentes rangos de fuerza aplicados en su zona sensible. Respuesta del sensor de fuerza para fuerzas en el rango 0 – 10.000 gramos (a). Respuesta del sensor de fuerza para fuerzas en el rango 0 – 1.000 gramos (b)

Este sensor ha sido incluido en la placa de prototipado siguiendo el esquema de la figura 12.15. De esta manera, el potencial eléctrico en el pin `ADC` de la placa aumenta a medida que se ejerce más presión en el sensor (y en consecuencia, disminuye su resistencia).

12.5. Conversor analógico-digital ADC0808

El CI ADC0808 de *National Semiconductor* 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. La lectura del resultado de la conversión se efectúa a través de sus 8 líneas de datos, que tienen capacidad triestado.

La conversión se realiza por aproximaciones sucesivas y mediante una red 256R.

FIGURA 12.9: Integración del sensor de fuerza en la placa de prototipado

El proceso de conversión y la lectura de los datos se realiza 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 `adc0808` que se proporciona en los archivos fuente de esta práctica, y cuya *entity* se puede ver a continuación:

```
entity adc0808 is
    Port ( clk        : in  std_logic;
           rst        : in  std_logic;
           addr_in    : in  std_logic_vector (2 downto 0);
           start      : 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);
           data_out   : out std_logic_vector (7 downto 0);
           data_valid : out std_logic);
end adc0808;
```

12.6. Control software

En el archivo `conversorAD/SDK/peripheral_tests_0/src/testperiph.c` se proporciona un ejemplo del control del conversor ADC.

```
/*
 * Se arranca una conversión en dirección 1
 */
Data=0x40000000;
while (1) {
  xil_printf(" Arranca conversion en direccion %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) {
    xil_printf("   - read %x from register 0 word 0\n\r", Datoleido);
    Datoleido = CONVERSOR_mReadSlaveReg0(baseaddr, 0);
  }

  xil_printf("   - read %x from register 0 word 0\n\r", Datoleido);

  Datoleido = CONVERSOR_mReadSlaveReg1(baseaddr, 0);
  xil_printf(" Valor de la conversión = %x from register 1 word 0\n\r", Datoleido);
}
```

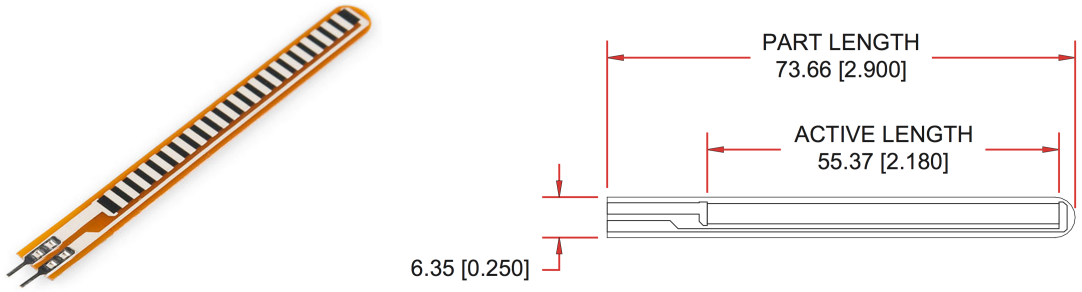
12.7. Ficheros fuente

En esta práctica se proporcionan los siguientes ficheros fuente:

- `practica_conversor.rar`, que contiene comprimido el proyecto EDK que servirá de base para el desarrollo del resto de la práctica.

12.8. Sensor de flexión

En esta sección se describen las características y el funcionamiento del sensor de flexión incluido en la placa de prototipado. Dicho sensor (Figura 12.10(a)) es un dispositivo de 56 mm de longitud activa cuya resistencia varía a medida que se flexiona. Es muy importante no doblar el dispositivo fuera del rango utilizable (ver Figura 12.10(b)), y siempre fuera de la zona de los pines o conexión.



(a) (b)

FIGURA 12.10: Sensor de flexión incluido en la placa de prototipado (a) y su zona utilizable (b)

Cuando el sensor no se encuentra doblado, su resistencia es de 25KΩ. Sin embargo, cuando éste se dobla, tiene una resistencia dentro del rango [45KΩ–125KΩ], tal y como se refleja en la figura 12.11.

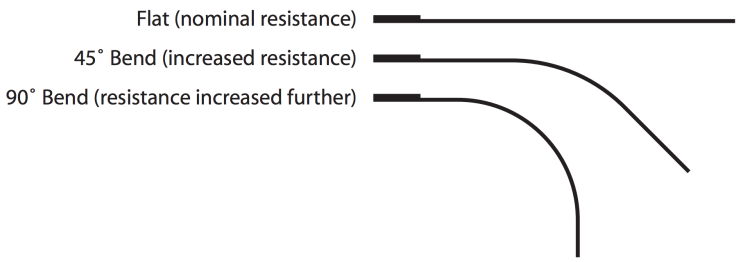


FIGURA 12.11: Variación de la resistencia del sensor de flexión a medida que éste se dobla

Este sensor ha sido incluido en la placa de prototipado siguiendo el esquema de la figura 12.12. De esta manera, el potencial eléctrico en el pin `ADC` de la placa disminuye a medida que aumenta la flexión del sensor (y en consecuencia, su resistencia).

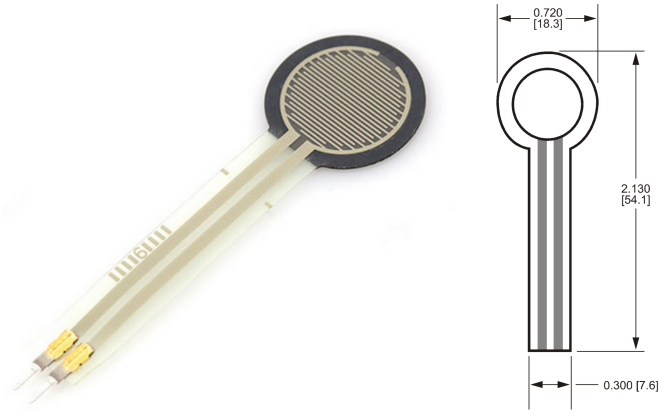
FIGURA 12.12: Integración del sensor de flexión en la placa de prototipado

12.9. Sensor de fuerza resistivo circular

En esta sección se describen las características y el funcionamiento del sensor de fuerza incluido en la placa de prototipado. Dicho sensor (figura 12.13(a)) es un dispositivo de 54 mm de longitud, que incluye un área sensible circular de 18,3 mm de diámetro (figura 12.13(b)) sobre la cual se puede aplicar una fuerza dentro del rango 0 a 10 Kg. A mayor fuerza aplicada, menor será su resistencia. Estos sensores son, en general, fáciles de instalar y de utilizar, pero no son muy precisos. Sin embargo, son más que suficientes para realizar las prácticas que se proponen en esta asignatura.

Cuando el sensor no recibe ninguna fuerza, su resistencia es de aproximadamente 1MΩ. Sin embargo, a medida que éste recibe una fuerza en su zona sensible, su resistencia disminuye, tal y como se indica en la figura 12.14(a) (nótese que la gráfica es doblemente logarítmica).

Como se puede observar, existe una fuerza de aproximadamente 20 gramos que "activa" el dispositivo y que hace que su resistencia disminuya de más de 100KΩ a aproximadamente 10KΩ. A partir de ese punto, la resistencia del dispositivo disminuye a medida que aumenta la presión, hasta que ésta alcanza los 10000 gramos, punto en el que la respuesta del sensor satura. La figura 12.14(b) muestra en detalle la respuesta de este sensor para fuerzas pequeñas. En este caso, la respuesta se expresa en términos de conductancia y de resistencia (que son magnitudes inversas la una respecto a la otra). Nótese que, a

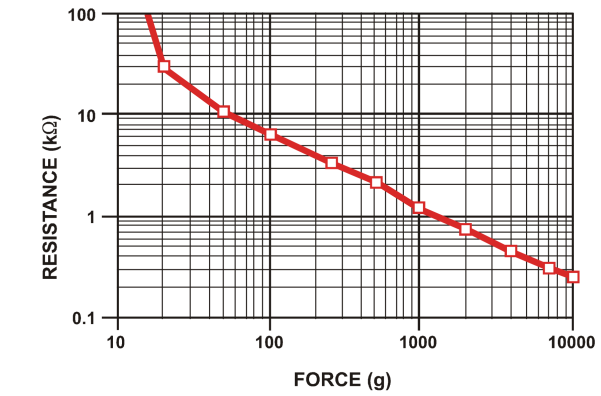


(a) (b)

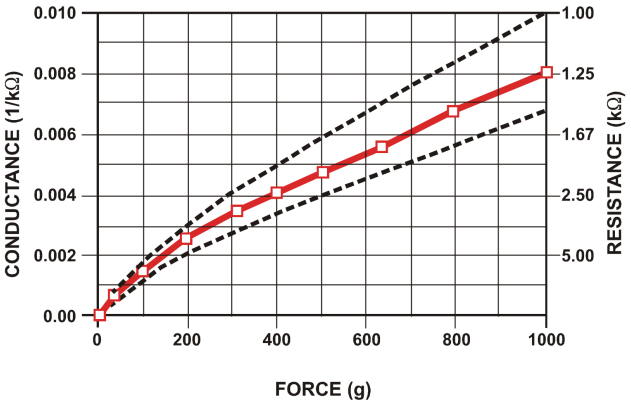
FIGURA 12.13: Sensor de fuerza incluido en la placa de prototipado (a) y sus dimensiones (b)

medida que disminuye la presión (para presiones menores de 100 gramos), la resistencia aumenta muy rápidamente, al aproximarse el valor de la conductancia a 0.

Este sensor ha sido incluido en la placa de prototipado siguiendo el esquema de la figura 12.15. De esta manera, el potencial eléctrico en el pin `ADC` de la placa aumenta a medida que se ejerce más presión en el sensor (y en consecuencia, disminuye su resistencia).



(a)



(b)

FIGURA 12.14: Variación de la resistencia del sensor de fuerza incluido en la placa de prototipado para diferentes rangos de fuerza aplicados en su zona sensible. Respuesta del sensor de fuerza para fuerzas en el rango 0 – 10.000 gramos (a). Respuesta del sensor de fuerza para fuerzas en el rango 0 – 1.000 gramos (b)

FIGURA 12.15: Integración del sensor de fuerza en la placa de prototipado

Práctica 13

Control de un bus de comunicaciones I2C

En esta práctica aprenderemos a utilizar un bus de comunicaciones I2C. I2C es el acrónimo de la expresión inglesa "Inter-Integrated Circuit", es decir, Inter-Circuitos Integrados. Se utiliza principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados.

13.1. Introducción

El objetivo de esta práctica 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).

Durante la realización de la práctica aprenderemos el fucionamiento de un bus I2C, la adición de nuevos periféricos al sistema y el paradigma de diseño hardware/software para realizar un controlador.

13.2. Bus de comunicaciones I2C

I2C es 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 es 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 si que normalmente residen en un mismo circuito impreso.

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 se la pueden ir pasando los dispositivos que tengan esa capacidad. Esta característica hace que al bus I2C se le denomine bus multimaestro.

Las transacciones en el bus I2C tienen este formato:

```
| start | A7 A6 A5 A4 A3 A2 A1 R/W | ACK | ... DATA ... | ACK | stop | idle |
```

- El bus esta 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 leido/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.

13.3. Implementación hardware del controlador I2C

13.3.1. Introducción

En este apartado se detalla un componente I2C maestro para un único bus maestro, escrito en VHDL para su uso en FPGAs. El componente lee y escribe a través del "user logic" mediante una interfaz paralela. La Figura 13.1 ilustra un ejemplo típico de I2C maestro integrado en un sistema.

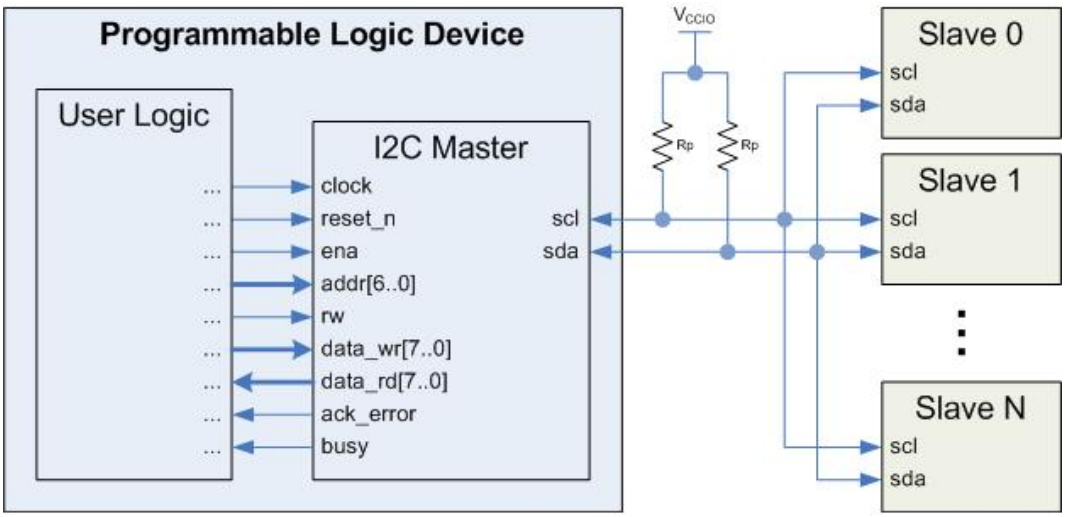


FIGURA 13.1: Ejemplo de implementación I2C

13.3.2. Modo de funcionamiento

El I2C maestro utiliza la máquina de estado se muestra en la Figura 13.2 para implementar el protocolo I2C. Tras el encendido, el componente entra inmediatamente en el estado "ready". Se espera en este estado

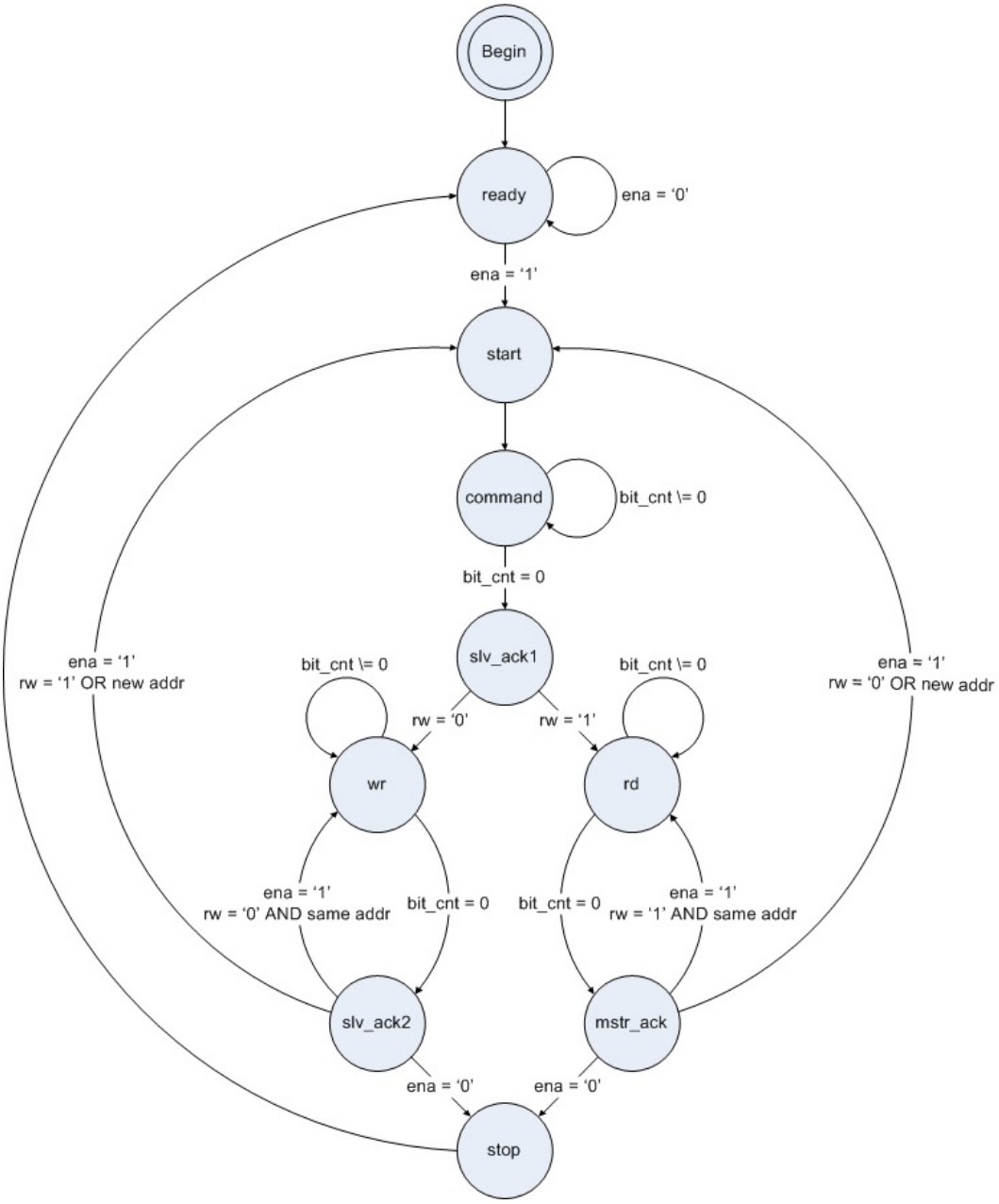


FIGURA 13.2: Máquina de estados del I2C maestro

hasta que la señal *ena* se activa y se registra el nuevo comando. 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. El estado "slv_ack1" a continuación, 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 ("mstr_ack" estado) 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) entonces, el maestro 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 (stop estado) y vuelve al estado "ready".

La temporización para la máquina del estados se calcula de la siguiente manera. El componente obtiene el reloj del 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.

13.4. Adición del periférico al sistema

13.4.1. Señales de entrada/salida y conexión con MicroBlaze

Debajo podemos ver la cabecera de nuestro periférico y sus señales de entrada/salida.

```
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 ---------------

    -- 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
    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
    Bus2IP_Clk                  : in  std_logic;
    Bus2IP_Reset                : in  std_logic;
    Bus2IP_Data                 : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                   : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                 : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                : out std_logic;
    IP2Bus_WrAck                : out std_logic;
    IP2Bus_Error                : out std_logic;
    IP2WFIFO_RdReq              : out std_logic;
    WFIFO2IP_Data               : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    WFIFO2IP_RdAck              : in  std_logic;
    WFIFO2IP_AlmostEmpty        : in  std_logic;
    WFIFO2IP_Empty              : in  std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
```

A continuación, explicamos las señales de salida referentes al control del bus de comunicaciones I2C y su significado. Destacar que para poder incorporar un periférico con señales de entrada/salida, EDK obliga a que se definan desde el user_logic las 3 señales que controlaran el IOBUF.

- 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.

El resto de señales corresponden a las señales del bus PLB utilizado para interconectar MicroBlaze con el controlador hardware.

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.

13.4.2. Cómo importar el periférico en EDK

Para incorporar el periférico al sistema descomprima el fichero 'I2C.zip' en el directorio raíz del proyecto básico de EDK (es decir, a la misma altura que el fichero 'system.xmp').

Ahora debemos importar dicho periférico desde EDK. Pulsamos sobre '**Hardware**' ->'**Create or Import Peripheral**'. En ese momento nos aparecerán una serie de ventanas de diálogo. A continuación, se va a exponer la lista de pasos necesarios para importar dicho periférico a través de las 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**'.
3. A continuación pulsamos '**Next**'.
4. En '**Name**' escribimos 'pantalla' y pulsamos '**Next**'.
5. A continuación pulsamos '**Next**'.
6. 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'). A continuación, pulsamos '**Next**'.
7. Pulsamos '**Add Files**', seleccionamos todos los ficheros que estaban en el fichero comprimido y pulsamos '**Next**'.
8. Seleccionamos '**PLBV46 Slave (SPLB)**' y pulsamos '**Next**'.
9. Pulsamos '**Next**'.
10. En '**Parameter determine high address**' seleccionamos '**C_HIGHADDR**' y pulsamos '**Next**'.
11. Deseleccionamos '**Select and configure interrup(s)**' y pulsamos '**Next**'.
12. Pulsamos '**Next**'.
13. Pulsamos '**Next**'.
14. 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'.

13.4.3. Control del periférico

Para llevar a cabo un control adecuado del bus de comunicaciones I2C se ha desarrolado el driver que implementa las siguientes funciones:

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

```
Xuint32 I2C_lectura(Xuint32 direccion){
   //LECTURA I2C
   while (I2C_mWriteFIFOFull(BASE_ADDRESS_I2C) == TRUE){}
   I2C_mWriteToFIFO(BASE_ADDRESS_I2C, 0, (Xuint32) ((direccion << 1)
                                                    + LECTURA_I2C_CMD));
   while (I2C_mReadFIFOEmpty(BASE_ADDRESS_I2C) == TRUE){}
   return I2C_mReadFromFIFO(BASE_ADDRESS_I2C, 0);
}
```

La función 'I2C_escritura' establece la configuración *dato* del periférico *direccion*. La función 'I2C_lectura' se utiliza para leer la configuración del periférico *direccion*.

El ejemplo de uso que se muestra debajo, lee la configuración establecida en los switches, escribe el dato en un expansor, lee la configración 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, 0x00);
    XGpio_SetDataDirection(&GpioSwitches, 1, 0xFF);

    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;
}
```

Práctica 14

Dispositivos I2C básicos

En esta práctica aprenderemos a utiliar los dos dispositivos I2C básicos incorporados en la placa de periféricos: un expansor de entrada/salida (PCF8574) y un conversor analógico/digital y digital/analógico (PCF8591).

14.1. Expansor I2C PCF8574

El integrado `PCF8574` 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 14.1 puede verse su diagrama de bloques. Además de las señales `SCL` y `SDA` del bus I2C, se tienen los pines `A0`, `A1` y `A2` para fijar la dirección del dispositivo. De esta manera podrían instalarse hasta 8 dispositivos `PCF8574` en el mismo bus. La dirección en el bus I2C es `0 1 0 0 A2 A1 A0 R/W`. En nuestro caso, las 3 señales de dirección están fijadas a 0, por lo que el expansor es seleccionable en las direcciones `0x40` (escritura) y `0x41` (lectura).

14.1.1. Funcionamiento

El `PCF8574` 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 | 0 1 0 0 0 0 0 0 | ACK | P7 P6 P5 P4 P3 P2 P1 P0 | ACK | stop | idle |
```

donde `P7..P0` 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`).

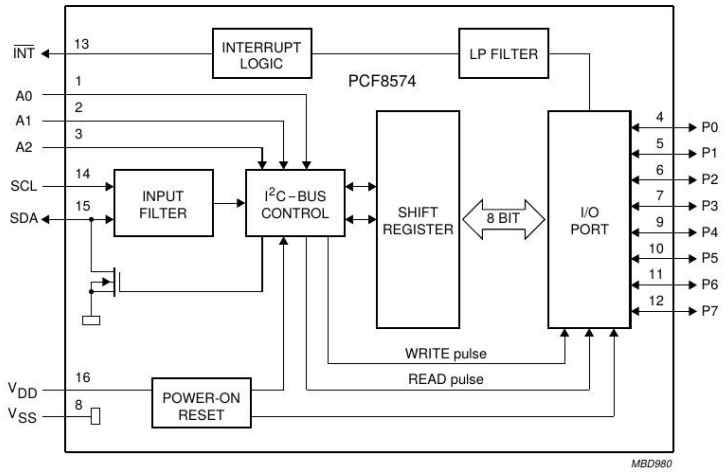


FIGURA 14.1: Expansor de entrada salida PCF8574

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

```
| start | 0 1 0 0 0 0 0 1 | ACK | P7 P6 P5 P4 P3 P2 P1 P0 | ACK | stop | idle |
```

Ahora, los valores obtenidos en `P7..P0` 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..P0` como entradas (para leer, por ejemplo, el estado de cuatro pulsadores o interruptores). Entonces deberíamos hacer lo siguiente:

1. Escribir el estado inicial de los leds (pines de salida, `L3..L0`) y el valor '1' en los pines de entrada:

   ```
   | start | 0 1 0 0 0 0 0 0 | ACK | L3 L2 L1 L0 1 1 1 1 | ACK | stop | idle |
   ```

2. Leer el estado de los interruptores:

   ```
   | start | 0 1 0 0 0 0 0 0 | ACK | X X X X P3 P2 P1 P0 | ACK | stop | idle |
   ```

3. Descartar, mediante máscara, los bits más significativos 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.

14.2. Conversor analógico/digital y digital/analógico PCF8591

El integrado `PCF8591` es un dispositivo que incluye cuatro entradas al 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 14.2 puede verse su diagrama de bloques. Además de las señales `SCL` y `SDA` del bus I2C, se tienen los pines `A0`, `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 0 0 1 A2 A1 A0 R/W`. En nuestro caso, las 3 señales de dirección están fijadas a 0, por lo que el conversor es seleccionable en las direcciones `0x90` (escritura) y `0x91` (lectura).

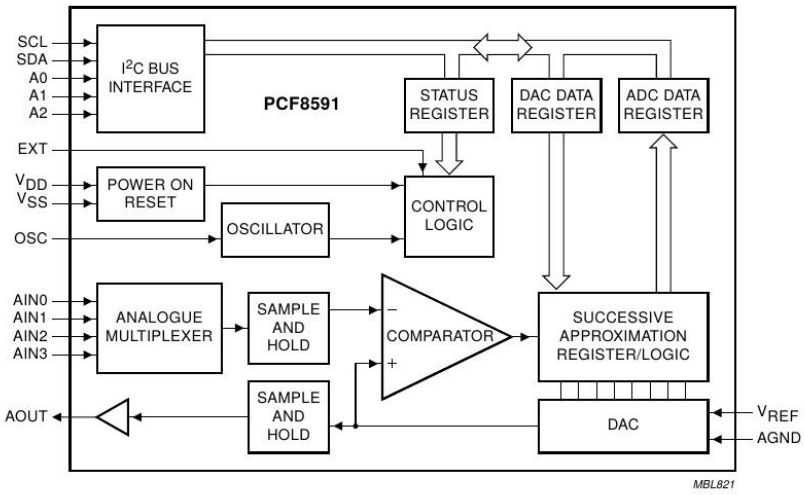


Figura 14.2: Expansor de entrada salida PCF8591

14.2.1. Conversión Analógica/Digital

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 | Ai0 |`

con el siguiente significado:

- `AoE` (*Analog output Enable*): cuando vale '1', se activa la salida del conversor digital/analógico.
- `AiP` (*Analog input Programming*): programa el modo de funcionamiento de las entradas analógicas, según la tabla 14.1.
- `AI` (*Auto Increment*): si vale 1, se incrementa de manera autómatica el canal de entrada para lecturas sucesivas.

- `Ai` (*Analog input*): selección del canal de entrada analógica.

AiP	Canal 0	Canal 1	Canal 2	Canal 3	Observaciones
00	AIN0	AIN1	AIN2	AIN3	4 entradas analógicas simples
01	AIN0 - AIN3	AIN1 - AIN3	AIN2 - AIN3	X	3 entradas diferenciales
10	AIN0	AIN1	AIN2 - AIN3	X	2 entradas simples y 1 diferencial
11	AIN0 - AIN1	AIN2 - AIN3	X	X	2 entradas diferenciales

Tabla 14.1: 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:

1. 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 | 1 0 0 1 0 0 0 0 | ACK | 0 0 0 0 0 0 0 0 | ACK | stop | idle |`

2. Leer el valor de la conversión:

   `| start | 1 0 0 1 0 0 0 1 | ACK | V V V V V V V V | 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 `0x80`. 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... Los sensores resistivos (LDR, NTC, flexión y fuerza) deben conectarse por medio de un divisor de tensión, como el mostrado en la figura 14.3.

14.2.2. Conversión Digital/Analógica

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 | 1 0 0 1 0 0 0 0 | ACK | 0 1 P P 0 A I I | ACK | V V V V V V V V | ACK | stop | idle |`

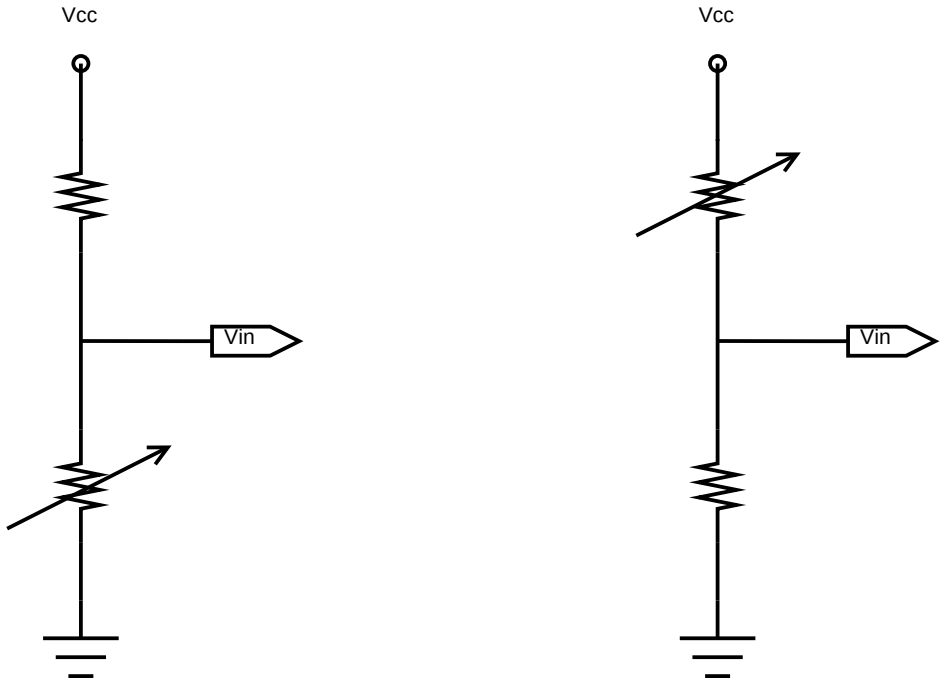


FIGURA 14.3: Divisor de tensión. (a) directo: a mayor resistencia, mayor tensión en `Vin`. (b) inverso: a mayor resistencia, menor tensión en `Vin`

Práctica 15

Acelerómetro

Un acelerómetro es un tipo de sensor capaz de medir la aceleración que sufre el dispositivo en un eje determinado. Pueden utilizarse, por ejemplo, en los dispositivos *airbag* de los automóviles y activar la protección al detectar una brusca deceleración provocada por un impacto. Si unimos tres acelerómetros, cada uno orientado en uno de los tres ejes espaciales, podremos determinar con precisión la posición del dispositivo. Esto es muy útil, por ejemplo, para los mandos de videoconsolas.

Utilizaremos el acelerómetro junto con el conversor Analógico/Digital estudiado previamente. Disponemos de un acelerómetro basado en el MMA7361LC, que nos proporciona una señal analógica por cada uno de los 3 ejes, X, Y, Z, con una sensibilidad de 800mV/g. Además dispone de una salida digital para detección de la condición de caída libre cuando se detecta 0g en los 3 ejes simultáneamente.

El principio de funcionamiento de este acelerómetro está basado en una microcelda que consta de tres placas conductoras (véase la figura 15.1). De estas tres placas, las dos de los extremos están fijas, mientras que la del centro es móvil y se puede mover solidariamente con una pequeña masa. Esta celda se puede modelar como dos pequeños condensadores variables, cuya capacidad varía en función de la posición relativa de la placa central. Cuando se aplica una aceleración, digamos, hacia la derecha, la placa central se acerca, por inercia, a la situada a la izquierda. Entonces, la capacidad del condensador formado entre las placas izquierda y central aumenta, mientras que la del formado entre las placas central y derecha disminuye. Midiendo la diferencia entre ambas capacidades, podemos saber la aceleración que sufre el dispositivo a lo largo de ese eje.

La tensión de alimentación del MMA7361LC es de 3,3V, por lo que para 0g proporciona una tensión de 1,65V. Se ha dispuesto para un rango de 1,5g, por lo que la sensibilidad es de 800mV/g. Se deben tener en cuenta estos datos a la hora de interpretar correctamente la salida del conversor analógico/digital.

En la figura 15.2 puede verse la placa que contiene el acelerómetro. Las señales disponibles son las que se indican en la tabla 15.1

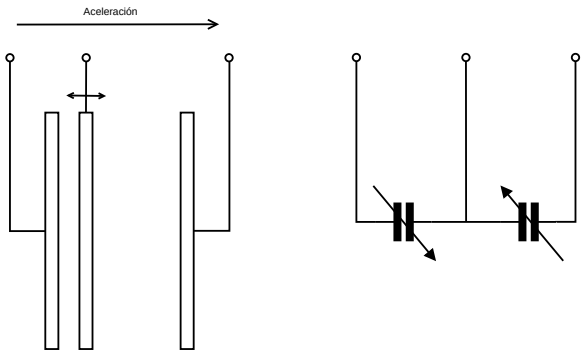


FIGURA 15.1: Celda básica del acelerómetro y su modelo electrónico simplificado

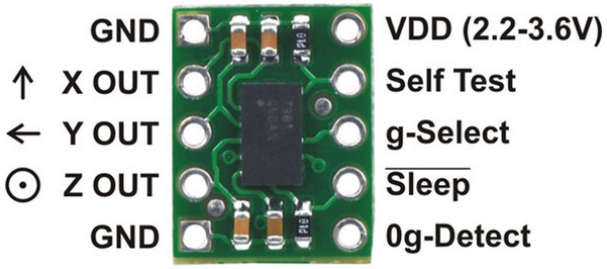


FIGURA 15.2: Placa del acelerómetro

Nombre	Sentido	Función
`Vdd`	–	Tensión de alimentación (2,2–3,6V)
`GND`	–	Tierra
`X`	out	Salida analógica del eje `X`
`Y`	out	Salida analógica del eje `Y`
`Z`	out	Salida analógica del eje `Z`
`STest`	in	Self Test
`SLP`	in	Sleep
`0G`	out	Detección de caída libre

TABLA 15.1: Señales del `MMA7361LC`

Práctica 16

Magnetómetro

Utilizaremos un módulo MinIMU-9v3 de Pololu[1], que integra un giróscopo, un acelerómetro, un magnetómetro y un termómetro, todo ello accesible mediante el bus I2C. El giróscopo está contenido en el chip L3GD20H, mientras que el magnetómetro, el acelerómetro y el termómetro están contenidos en el chip LSM303D. Ambos chips están incluidos en la misma placa, que puede verse en la figura 16.1(a), conectados según el esquema de la figura 16.1(b).

16.0.3. LSM303D

Para utilizar los sensores ubicados en el chip `LSM303D` es necesario configurarlos previamente escribiendo en los registros de control, accesibles mediante el bus I2C. A continuación se muestra la secuencia típica de activación. Para más información, consultar la hoja técnica adjunta.

CTRL2 $\leftarrow 0x00$
CTRL1 $\leftarrow 0x57$
CTRL5 $\leftarrow 0xE4$
CTRL6 $\leftarrow 0x00$
CTRL7 $\leftarrow 0x00$

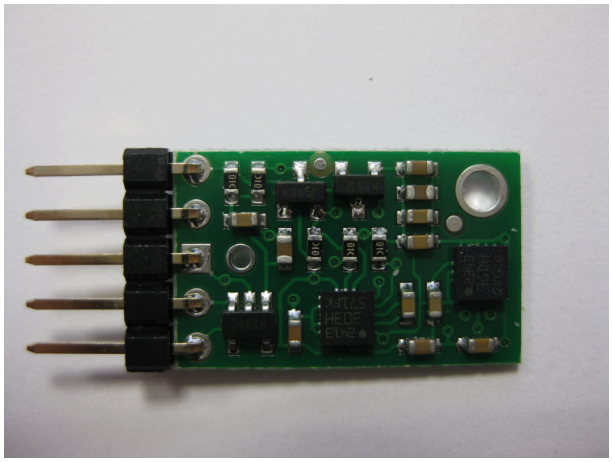
Para escribir un registro del LSM3003D, se debe seguir la siguiente secuencia I2C:

```
| start | 0x3A | <num_reg.>| <valor>| stop | idle |
```

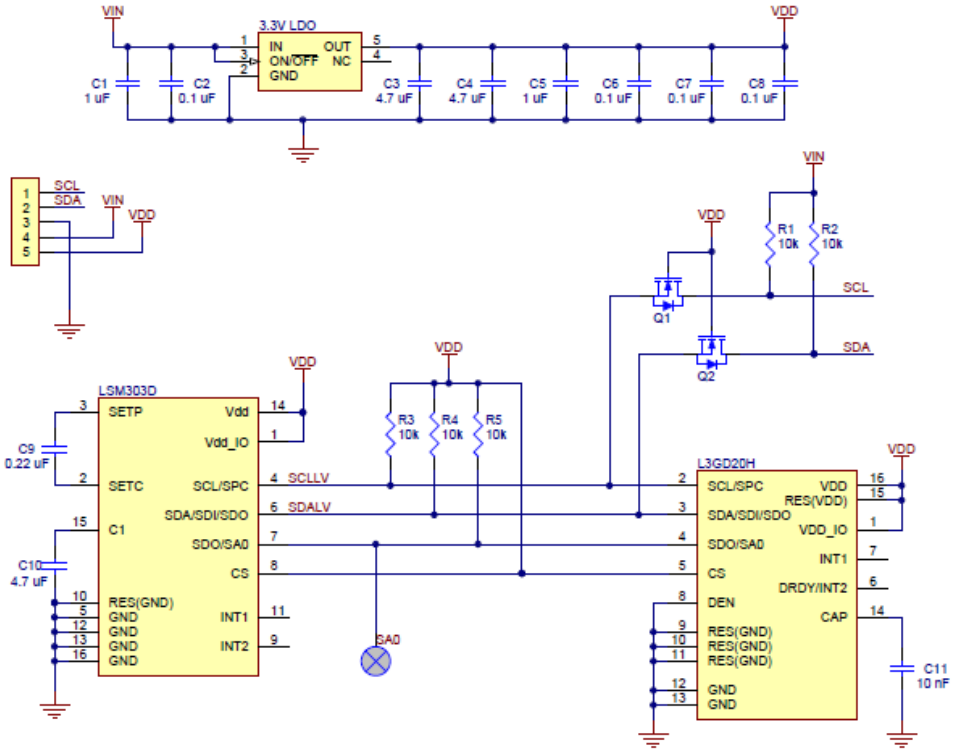
Para leer un registro, primero debe realizarse una escritura con la dirección del registro, seguida de una lectura, según la siguiente secuencia:

```
| start | 0x3A | <num_reg.>| start | 0x3B | <lectura>| stop | idle |
```

[1] http://www.pololu.com



(a)



(b)

FIGURA 16.1: Placa de sensores MinIMU-9v3 (a) y su esquema electrónico (b)