Dispositivos y Sistemas Programables Avanzados

Autores:

Antonio Calomardre
Jordi Zaragoza

Índice:

Práctica 1: Barra de Leds

Antonio Calomardre
Jordi Zaragoza

1. Introducción

Esta práctica pretende dar una visión de cómo crear los principales módulos VHDL de un proyecto, tales como un **Control** (diagrama de estados pasado a VHDL), un **Divisor**, un **Contador**, un **Decoder** y sus respectivas simulaciones temporales.

Se han adjuntado todos los ficheros código VHDL para esta práctica, para que puedan ser consultados en caso de duda en su programación. Se recomienda hacer el menor uso posible de ellos, de esta forma se adquirirán unos mayores conocimientos para las prácticas sucesivas.

2. Descripción del diseño

Esta primera práctica consiste en realizar un encendido ascendente y descendente progresivo de la barra de les leds. Los diferentes módulos de los que constará el proyecto a diseñar son los siguientes:

Módulo de control. Esta parte será creada con la herramienta *State Diagram*, para la creación de nuestro sistema de control mediante un diagrama de estados. Este módulo debe permitir al usuario el control del sistema mediante dos pulsadores *"reset and start*"

Módulo Contador. El contador deberá ser ascendente y descendente de tal forma que se pueda conseguir la iluminación de cada uno de los leds de forma progresiva y en ambos sentidos.

Módulo Decoder. El decodificador realiza la tarea de pasar el valor del contador a su correspondiente combinación binaria para que se encienda el led correspondientes en cada momento.

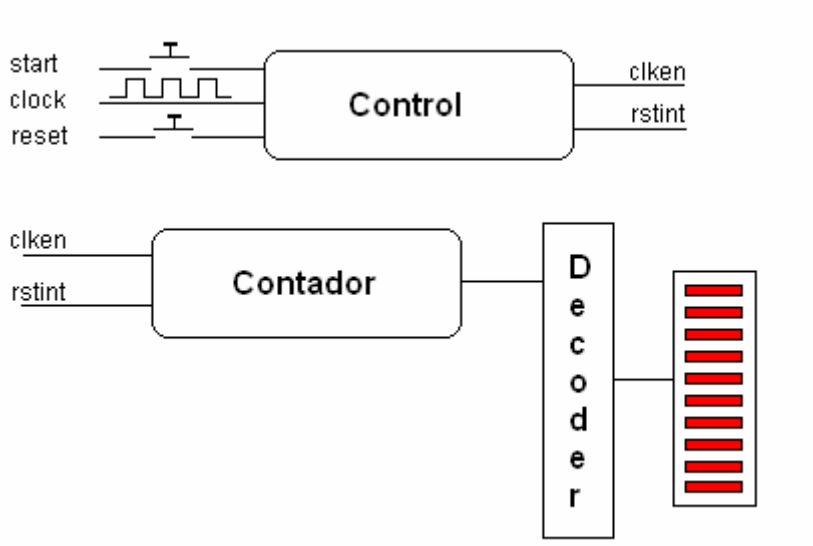

Figura 1: Diagrama de bloques del sistema a implementar

3. Procedimiento a Seguir

Para poder empezar con el primer proyecto a diseñar, se recomienda antes hacer una lectura del manual de prácticas.

3.1 Módulo de control

Primero será necesario crear un proyecto y el primer modulo VHDL. Seguiremos los pasos indicados en el tutorial de prácticas (pag 7) con la pequeña modificación que el nombre de nuestro primer módulo VHDL será **Contador.** Para poder hacer uso de los ficheros adjuntos del programa Contador se recomienda utilizar las mismas variables de entrada y salida de nuestro módulo:

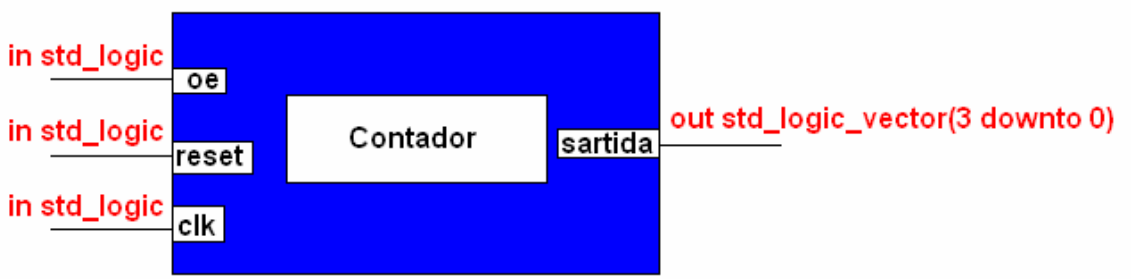

Figura 2: Módulo Contador

Se puede apreciar que las dos entradas **oe** (habilitación del dispositivo), **reset** (puesta a cero del contador) y **clk** (clock) son entradas del tipo *std_logic*, por su carácter '1' o '0'. La salida en cambio es del tipo std_logic_vector (3 downto 0). En este último caso la salida debe ser un vector de cuatro bits, para poder asociar su valor a los diez Leds de nuestra placa.

A partir de aquí se debe proceder a la implementación del código VHDL para que el contador realice un conteo ascendente y descendente.

Una ver implementado el código pertinente, se pasara a la verificación de este mediante *Synthesize* en la ventana *Process for Current Source*.

Siendo nuestro código VHDL correcto, se realizará su simulación mediante **Modelsim** (ap.5.4 del tutorial).

Podemos apreciar el código de verificación VHDL en la figura 3 y en la figura 4 el resultado de la simulación.

Para lanzar la simulación seleccionar Contador_sim (nombre creado para el código de verificación) *Simulate Behavioral VHDL Modal* en la ventana *Process for Current Source.*

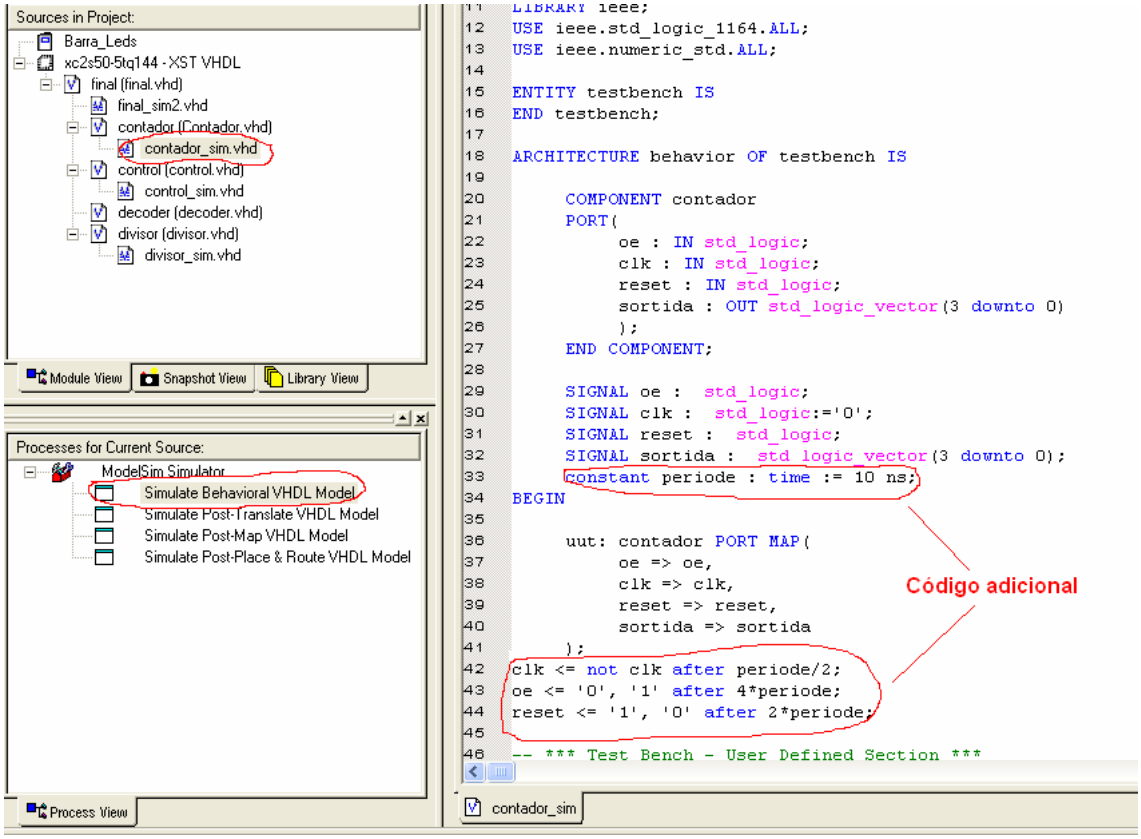

Figura 3: Código verificación VHDL

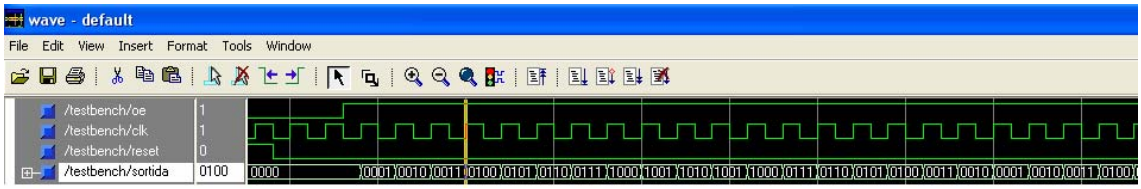

Figura 4: Simulación

3.2 Módulo Decoder

El proceso de código del módulo Decoder es el mismo que el anterior. Este módulo debe decodificar la salida binaria del módulo Contador y adaptarla al encendido y apagado de la barra de leds de nuestra placa.

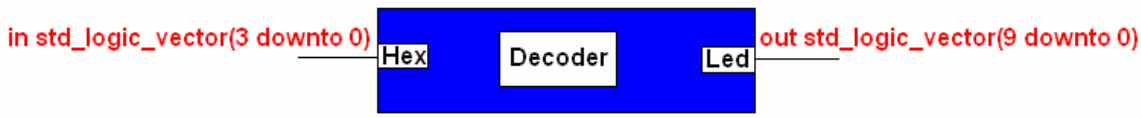

Figura 5: Módulo Decoder

A continuación se muestra el código de este módulo:

```
--pasar de HEX a la barra de leds
--    HEX:    in     STD_LOGIC_VECTOR (3 downto 0);
--    LED:    out    STD_LOGIC_VECTOR (9 downto 0);
--
--
--          Barra de Leds
--    |-----------------------|
--    | | | | | | | | | | | |
--    | 0 1 2 3 4 5 6 7 8 9 10|
--    |-----------------------|

entity Decoder is
    Port (
          HEX:    in     STD_LOGIC_VECTOR (3 downto 0);
          LED:    out    STD_LOGIC_VECTOR (9 downto 0)
     );
end Decoder;

architecture Behavioral of Decoder is

begin

    with HEX SELect
   LED<= "0000000001" when "0001",    --1
         "0000000010" when "0010",    --2
         "0000000100" when "0011",    --3
         "0000001000" when "0100",    --4
         "0000010000" when "0101",    --5
         "0000100000" when "0110",    --6
         "0001000000" when "0111",    --7
         "0010000000" when "1000",    --8
         "0100000000" when "1001",    --9
         "1000000000" when "1010",    --10
         "0000000000" when others;    --0
 end Behavioral;
```

Figura 6: Código VHDL

3.4 Módulo Control

Esta parte será creada con la herramienta *State Diagram*, para la creación de nuestro sistema de control mediante un diagrama de estados. Este módulo debe permitir al usuario el control del sistema mediante dos pulsadores *"reset and start"*(pag 25 tutorial).

En al figura 7 podemos apreciar el diagrama de estados y el módulo VHDL.

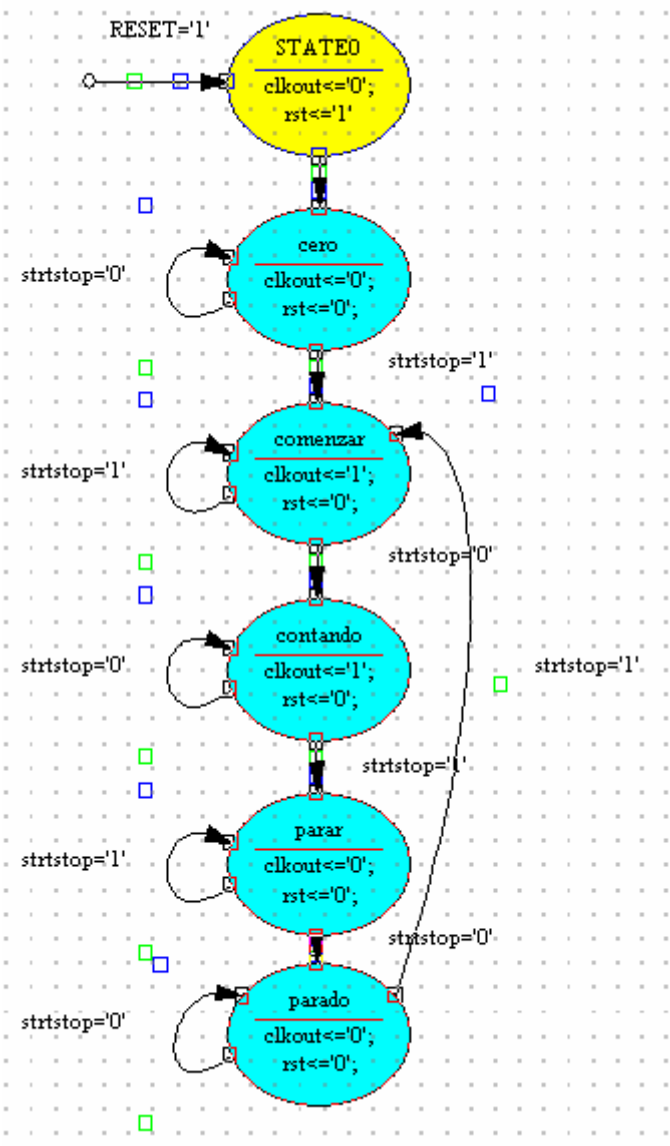

Figura 7: Diagrama de estados

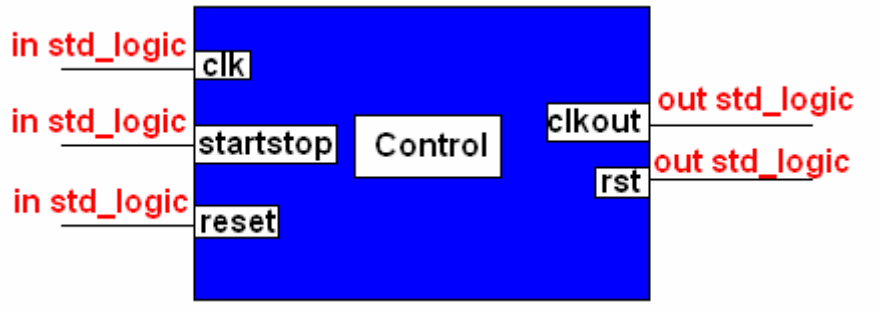

Figura 8: Módulo Control

3.5 Módulo Divisor

Para poder ver el desplazamiento de los leds, será necesario reducir la frecuencia de reloj de trabajo. Teniendo en cuenta que por software podremos dividir la frecuencia de 100Mhz inicial por 2000 (según apartado 5), seguirá siendo necesario el módulo Divisor para dividirla por 5000 más.

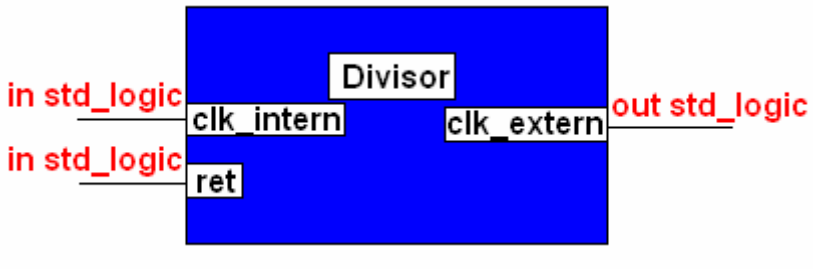

Figura 9: Módulo Control

3.6 Módulo Final

Para poder interconectar los diferentes módulos, es creado este último para poder enlazarlos todos, en este módulo se realiza una llamada a los distintos módulos ya creados. Si consultamos el fichero Final.VHDL podemos apreciar como se realizan estas llamadas para incorporar los componentes previamente creados.

Por ejemplo para la incorporación del módulo contador:

- Seleccionar Contador
- *Ejecutar View VHDL Instantation Template* en la ventana *Process for Current Source.*
- El código generado es el siguiente:

```
COMPONENT contador
PORT(
      oe : IN std_logic;
      clk : IN std_logic;
      reset : IN std_logic;
      sortida : OUT std_logic_vector(3 downto 0)
      );
END COMPONENT;

Inst_contador: contador PORT MAP(
      oe => ,
      clk => ,
      reset => ,
      sortida =>
);
```

- Mirar en el fichero Final.VHDL su ubicación y realizar el mismo proceso para los restantes módulos.

4. Implementación del diseño

Una vez ya simulados y comprobado el buen funcionamiento de los bloques anteriores, se pasara a la asignación de los pines E/S. Un método avanzado seria tal y como se describe en la pag.37 del tutorial. Existe una forma de asignación más restrictiva, pero muy fácil y rápida:

- Seleccionar el módulo Final, entrar en el menú *Project* y crear *New Source*
- Seleccinar *Implementation Constraints File* y crearemos el fichero *.ucf con el nombre Asignación. Nos aparece la ventana siguiente, donde asignaremos los pines.

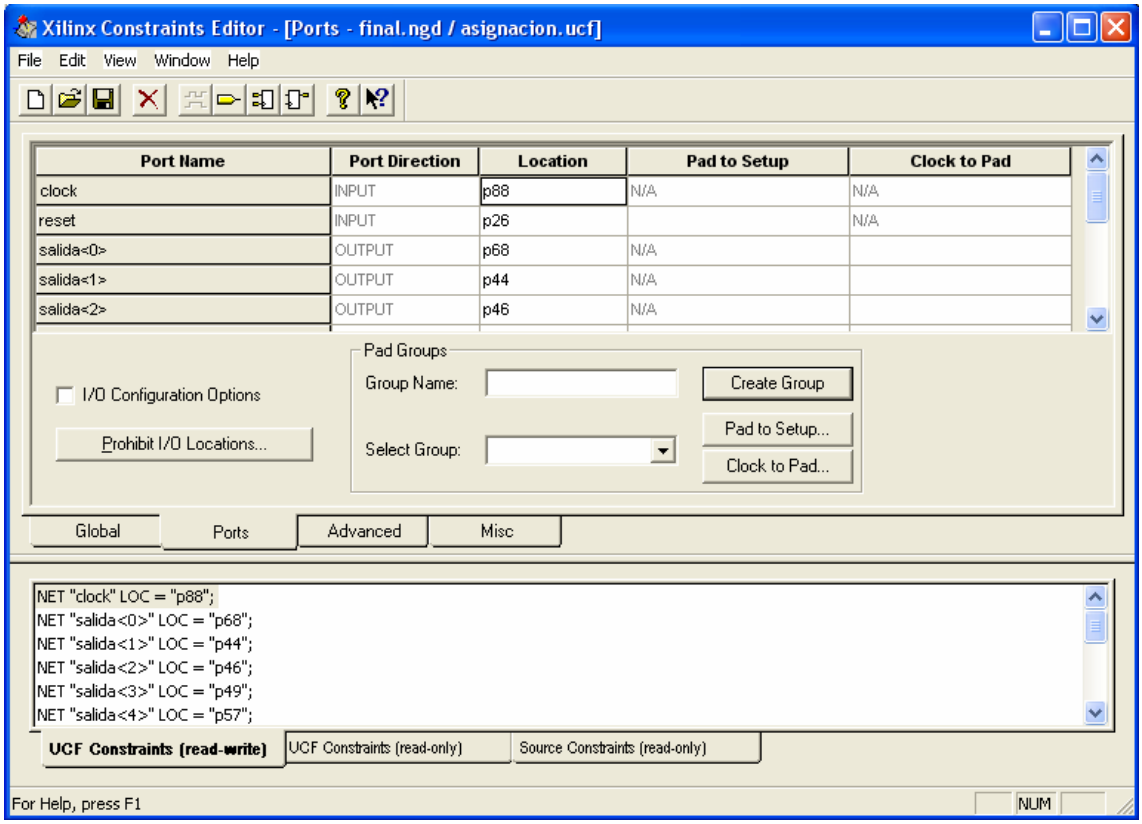

Figura 10: Asignación de los pines

A partir de este punto ya podemos crear el fichero *.bit para poderlo volcar en nuestra placa (pag 47 tutorial).

5. Programación de la frecuencia de reloj y volcado del programa

Una vez generado el archivo *final.bit* solo queda su volcado a la FPGA, pero ante se debe realizar la programación de la frecuencia de reloj. Para la generación de estas dos tareas se ara uso del programa *Xstools* subministrado por la corporación *Xess* (http://www.xess.com).

5.1 Oscilador programable.

Este periférico es de vital importancia en la mayoría de las aplicaciones, ya que controla el sincronismo de ejecución.

Se trata de un oscilador programable con una frecuencia máxima base de 100 MHz del fabricante Dallas Semiconductor. El modelo en cuestión es el DS1075 y permite programar vía software diferentes frecuencias de oscilación, tales como 100MHz, 50Mhz, 33.3Mhz, 25Mhz,... 48.7KHz a través de un divisor.

La frecuencia base y el valor del divisor son almacenados en una memoria EEPROM siendo modificable por el usuario.

Una característica añadida es la posibilidad de utilizar un cristal externo en aplicaciones de gran precisión realizando unos ajustes mínimos en el conjunto hardware. Una ver realizados estos ajustes el reloj externo sustituye al oscilador programable disponible en el equipo de pruebas.

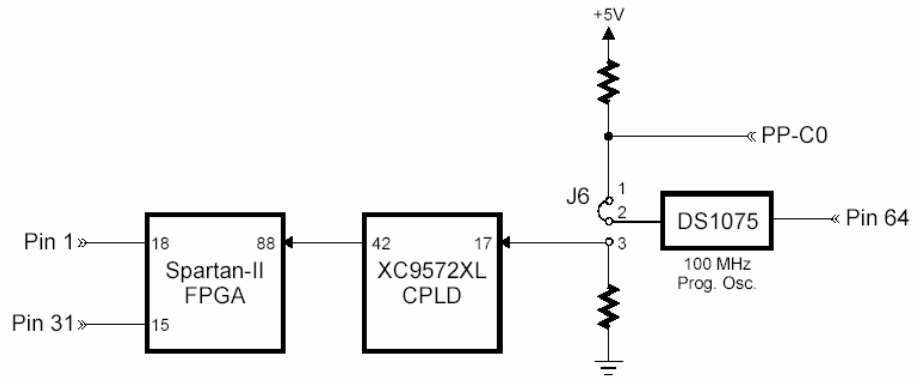

Figura 5.1 *Esquema conexión del oscilador programable.*

5.2. Software de programación “Xstools”.

Este software consta de un conjunto de programas sirven para volcar el código VHDL a la FPGA, testar la placa y programar la frecuencia de reloj.

Un programa disponible es el de Testar la conexión PC a Placa XSA, que deberemos asegurarnos que funciona correctamente, sino no podremos realizar ningún tipo de conexión a la placa, como por ejemplo volcar el programa a la FPGA.

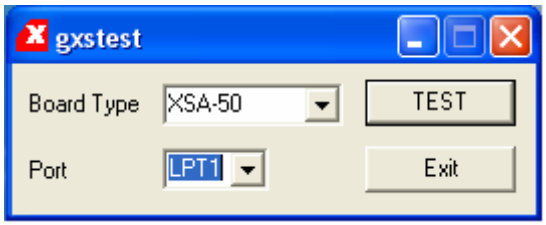

Figura 5.1 *Programa de Testeo PC a FPGA.*

Como podemos ver en la figura anterior, realizar un TEST es muy fácil solo debemos poner el tipo de placa que tenemos y el puerto de transmisión.

En el Xstools, se encuentra un programa para variar la frecuencia del reloj programable de la placa, como ya se ha comentado también se puede poner un reloj externo, pero si utilizamos el de la placa podemos realizar una serie de divisiones a partir de la frecuencia máxima que es 100Mhz.

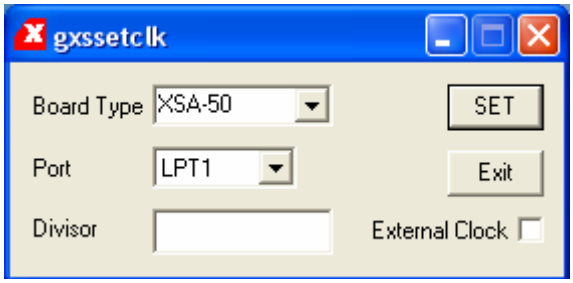

Figura 5.2 *Programa para variar la frecuencia de reloj.*

Si nos fijamos en la figura debemos especificar que tipo de placa es, el puerto de transmisión y finalmente el número por el cual dividimos los 100 Mhz (en este caso 2.000). También hay la opción como se puede ver de reloj externo.

Finalmente existe el programa para volcar el código VHDL compilado (.bit) a la FPGA, como veremos a continuación es muy fácil de utilizar:

Primero nos tenemos que asegurar que la conexión es correcta y luego haber elegido la frecuencia de reloj adecuada. Una vez hemos hecho los pasos anteriores solo debemos cargar en el programa el archivo ***. bit** que deseamos y automáticamente se volcará en la FPGA.

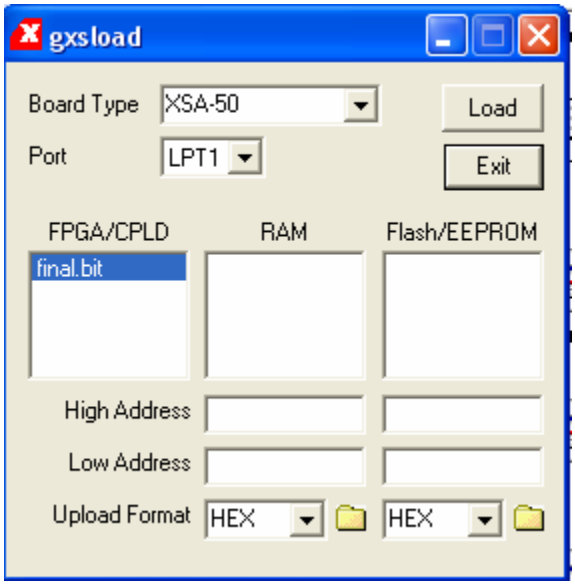

Figura 5.3 *Programa para volcar el código a la FPGA*

Práctica 2: Loopback

Antonio Calomardre
Jordi Zaragoza

1. Introducción

La FPGA por sí sola no puede obtener audio del exterior y tampoco sacarlo posteriormente, para ello necesitamos un hardware y software adicional.

Se pretende hacer pasar una señal de audio por la FPGA, dicha señal audible procede de nuestro reproductor de CD o en nuestro caso de un ordenador donde a través de su tarjeta de sonido podremos adquirir la señal. Para hacer pasar dicha señal audible por nuestro dispositivo programable, será necesario un hardware adicional, tal y como es un conversor de audio (AK4520A).

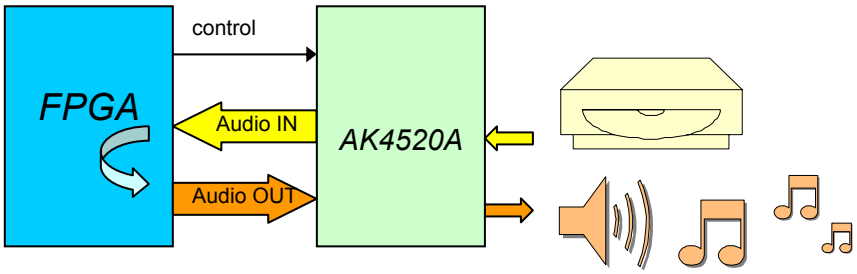

Figura 1.Diagrama de bloques del sistema

Esta práctica consiste en controlar a partir de la FPGA el convertidor de audio y adquirir una señal audible por el AK4520A, pasando por la FPGA y volviendo a salir por el AK4520A.

2. Convertidor de audio digital.

Este periférico consiste en un conversor analógico/digital y digital/analógico con características orientadas a aplicaciones con señales de audio. Dispone de una entrada y una salida de dos canales, izquierda y derecha, con una resolución de 20 bits. La conexión de audio se realiza a través de sendos conectores tipo jack.

La señal se envía a la FPGA en formato serie sincronizada con una señal master generado por el dispositivo lógico programable.

La conexión con el sistema lógico programable principal, se muestra en la siguiente figura siguiente.

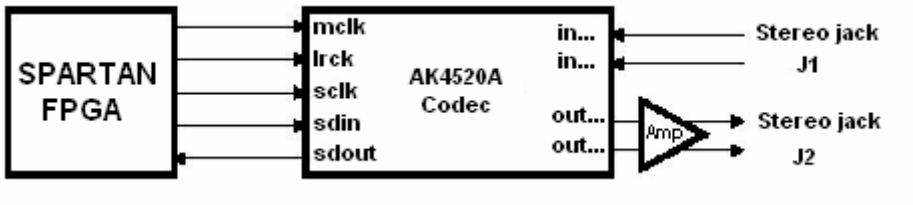

Figura 2.Esquema de conexión del convertidor de audio digital.

Del convertidor de audio podemos apreciar en la figura 3 su estructura interna, como sus entradas y salida disponibles. Sus principales bloques a destacar son:

- □ ΔΣ Stereo ADC
- □ ΔΣ Stereo DAC
- □ Frecuencia de muestreo 16kHz a 54kHz
- □ Master Clock: 256fs or 384fs
- □ 2.7 a 3.6V o 4.5 a 5.5V de alimentación

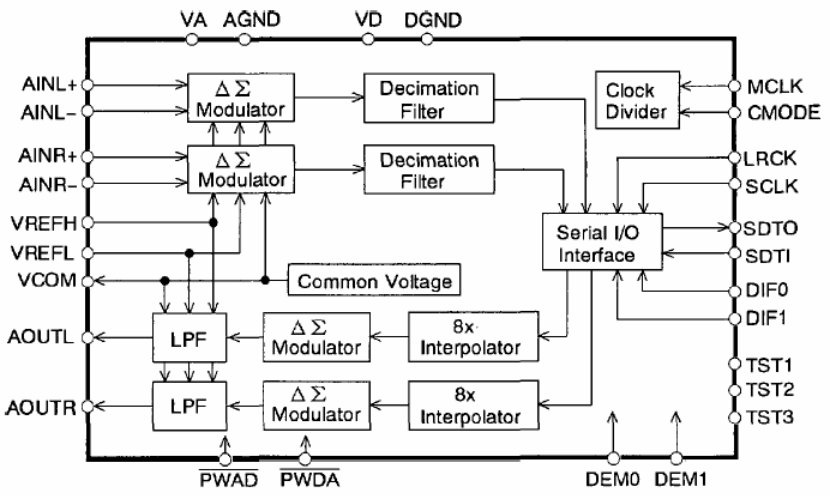

Figura 3. Estructura interna del AK4520A

Para más información consultar el data sheet del AK4520A.

3. Control de datos

3.1 Sistema de *clock input*

Nosotros queremos una frecuencia de muestreo de 48.1 kHz, en la placa esta configurada con CMODE = "L", por lo tanto el reloj del sistema será de 12.2880 MHz. La frecuencia de reloj se obtendrá mediante el oscilador de la placa.

Podemos ver a continuación una tabla donde se refleja la señal de reloj que se le debe aplicar al conversor según como este configurado y según la frecuencia de muestreo que deseemos.

fs	MCLK		SCLK	
	256fs CMODE="L"	384fs CMODE="H"	64fs	32fs
32.0kHz	8.1920MHz	12.2880MHz	2.048MHz	1.0240MHz
44.1kHz	11.2896MHz	16.9344MHz	2.822MHz	1.4112MHz
48.0kHz	12.2880MHz	18.4320MHz	3.072MHz	1.5360MHz

Tabla 1. Ejemplos de clock de sistema

Para obtener una frecuencia de muestreo de 48 KHz, se aplicará una frecuencia de reloj de 12.2880 MHz donde la señal MCLK = reloj sistema y SCLK = ¼ * MCLK (esta ultima se obtendrá por software).

3.2 Formato de interfaz serie de datos

Los datos que adquiere nuestro conversor son adquiridos en formato serie por los pines in/out de SDTI/SDTO y controlados por las entradas de SCLK y LRCK. Hay diferentes modos de transmisión serie de los datos, tal y como muestra la Tabla 2. Estos modos se pueden seleccionar mediante los pines DIF0 y DIF1 (por defecto en modo 2).

Mode	DIF1	DIF0	SDTO(ADC)	SDTI(DAC)	L/R	SCLK
0	0	0	20bit, MSB justified	16bit, LSB justified	H/L	≥32fs
1	0	1	20bit, MSB justified	20bit, LSB justified	H/L	≥40fs
2	1	0	20bit, MSB justified	20bit, MSB justified	H/L	≥40fs
3	1	1	IIS(I2S)	IIS(I2S)	L/H	32fs or ≥40fs

Tabla 2. Modos de comunicación serie de datos

En la figura 4 podemos apreciar la evolución de los datos de entrada y salida (SDTI y SDTO), como también las de control (LRCK y SCLK) que establecen una relación con la de MCLK.

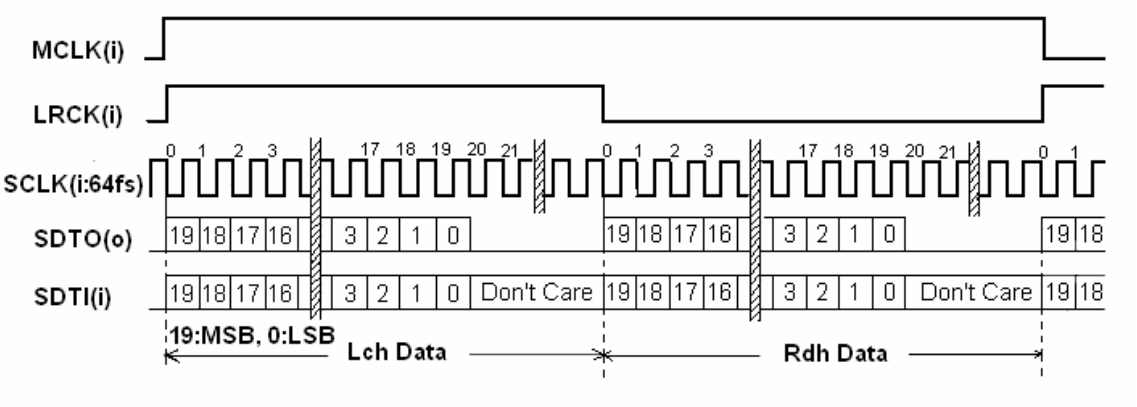

Figura 3.1 Temporización en Modo 2

4. Asignación de pines

Por último, se deberán asignar los pines correspondientes a las señales de control y audio entre la FPGA y nuestro codificador/decodificador de audio (AK4520A).En la figura 5 se detallan los pines de asignación correspondientes a las principales señales a utilizar en el diseño de esta práctica.

	I/O Name	I/O Direction	Loc
	sdout	Input	p76
	sdin	Output	p74
	sclk	Output	p75
	mclk	Output	p77
	lrck	Output	p59
	clk	Input	p88

Figura 4.1 Asignación de pines