Sistemas Informáticos
Curso 2005 - 2006

Diseño y fabricación de una placa didáctica basada en FPGA

Francisco Javier Arellano Mauleón
Francisco Calderón Setién
Sergio Somovilla de la Torre

Dirigido por:
José Manuel Mendías Cuadros
Dpto: Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

ÍNDICE DE IMÁGENES

RESUMEN DEL PROYECTO Y PALABRAS CLAVE

RESUMEN DEL PROYECTO

El objetivo de este proyecto es el desarrollo de una placa basada en FPGA que será utilizada por los alumnos de Segundo Curso en la asignatura Laboratorio de Tecnología de Computadores. El proceso se divide en tres fases principales: diseño del prototipo, implementación del circuito impreso y desarrollo y depuración del software de comunicación entre el PC y la placa.

ABSTRACT

The aim of this project is the development of an FPGA board for being used in a practical course on Computer Technologies by Second Year students. The process involves three major steps: prototype design, PCB physical implementation and PC-board communication software development and debugging.

PALABRAS CLAVE

FPGA, Spartan-II, USB, FT2232, placa, prototipado, didáctica, Xilinx, FTDI, PCB.

AUTORIZACIÓN

Los ponentes Francisco Javier Arellano Mauleón, con DNI 50544160, Francisco Calderón Setién, con DNI 50470372, y Sergio Somovilla de la Torre, con DNI 52877509, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos tanto la propia memoria, como el código, la documentación y el prototipo desarrollado.

Francisco Javier Arellano Mauleón

Francisco Calderón Setién

Sergio Somovilla de la Torre

En Madrid, a 4 de julio de 2006

INTRODUCCIÓN

El proyecto de la asignatura Sistemas Informáticos denominado "*Diseño y fabricación de una placa didáctica basada en FPGA*" consiste en la realización de un prototipo de placa de desarrollo de sistemas digitales basada en FPGA, con objeto de reemplazar las que en la actualidad se usan en el Laboratorio de Tecnología de Computadores.

Objetivos del proyecto

Durante el segundo cuatrimestre del Segundo Curso de Ingeniería Informática, en la asignatura Laboratorio de Tecnología de Computadores, los alumnos ponen en práctica sus conocimientos de diseño e implementación de sistemas digitales haciendo uso de un conjunto de placas de prototipado basadas en FPGA. Estas placas son básicamente un circuito impreso con una FPGA, una conexión al PC y circuitería adicional como bancos de displays, pulsadores, LEDs, puertos IDE, puertos VGA o altavoces, entre otros. La FPGA es un elemento de hardware dinámicamente reconfigurable y es en ella donde se cargan los diseños realizados mediante algún software de diseño asistido por computador, como Xilinx Foundations o Xilinx ISE. La conexión entre el PC y la FPGA permite el envío del diseño realizado a la FPGA en forma de archivo de configuración, y todos elementos adicionales (bancos, switches, puertos…) sirven para la comprobación de que el diseño es correcto y para la construcción de sistemas digitales más complejos.

Este proyecto pretende reemplazar las placas didácticas actuales por otras más adecuadas para los requerimientos de este laboratorio, sustituyendo los componentes principales de la placa por otros más modernos, añadiendo nuevas funcionalidades y eliminando aquellos módulos utilizados con poca frecuencia o que no tienen demasiado valor pedagógico.

Uno de los principales cambios en la placa didáctica que se va a diseñar con respecto a las que se utilizan en la actualidad en el Laboratorio de Tecnología de Computadores está en la interfaz entre el ordenador personal y la FPGA. Mientras que los modelos actuales se comunican con el PC a través del puerto paralelo LPT mediante un conector DB-25, la nueva placa lo hará a través de un puerto USB.

El motivo de esta modificación es evidente. El protocolo USB está desplazando a todos los demás, como el RS-232, el PS/2 o el LPT, en un gran número de periféricos y dispositivos hardware. Hoy en día, la gran mayoría de las impresoras, discos duros externos y módems, entre otros, se comunica con el PC por USB.

Las ventajas de este protocolo son numerosas. En primer lugar, es una solución flexible y de bajo coste que puede conseguir tasas de transferencia de hasta 12 Mb/s. Además, aporta una interfaz serie bidireccional isócrona de fácil uso e instalación dinámica plug-and-play. Por último, permite conectar simultáneamente a un mismo bus hasta 127 periféricos de naturaleza muy diversa.

En lo que al proyecto se refiere, una placa de prototipado con conexión USB evitará el uso de material obsoleto por parte de los alumnos del Laboratorio de Tecnología de Computadores (el protocolo USB es muy reciente y no parece que vaya a ser reemplazado en el futuro a corto plazo) y aportará mayor comodidad (por un lado, la configuración de la placa se realizará automáticamente al conectarla al PC; y por otro lado, el circuito se podrá alimentar a través del bus USB y se podrá prescindir de la voluminosa fuente de alimentación externa, si se desea). También constituirá una solución mucho más robusta y fiable que la actual a través del puerto paralelo LPT.

Otra modificación importante estará en el modelo de FPGA utilizado. En la mayor parte de las placas existentes actualmente, se usa una XC4010XL de Xilinx. Esta familia es bastante antigua y el fabricante ha ido paulatinamente reduciendo el soporte ofrecido para la misma. Hoy en día, las versiones más modernas del software de desarrollo de sistemas digitales (como

Xilinx ISE 6) no permiten la creación de archivos de configuración para FPGAs de esta familia. Por ello, se decidió reemplazar estos componentes por uno más moderno, el modelo XC2S100-5 de la familia Spartan-II de Xilinx.

Además de ser un modelo compatible con las últimas versiones de software de desarrollo de sistemas digitales, esta FPGA tiene una mayor capacidad, pudiendo así alojar diseños de mayores dimensiones, y posee hasta 92 pines de usuario, de forma que se puede conectar un mayor número de módulos externos.

Se incluirán además nuevos módulos de circuitería adicional para el desarrollo de sistemas digitales, o bien se aumentará el número de los ya existentes en las placas antiguas. Entre ellos, se incluirá un teclado matricial, entre 8 y 12 switches, unos 8 pulsadores, entre 10 y 12 LEDs, un zumbador y un altavoz, 8 displays 7-segmentos con refresco y un reloj programable mediante jumpers, entre otros.

También se ha decidido prescindir de algunos elementos presentes en las placas actuales, como la XS40, que posee un microcontrolador 8031 y una memoria SRAM. Ninguno de estos componentes estará presente en la nueva placa, ya que no se ajustan a los requerimientos de la asignatura Laboratorio de Tecnología de Computadores.

Junto con el diseño y fabricación de la placa de circuito impreso conteniendo estos dispositivos, se desarrollará un software de comunicación para que los alumnos de Segundo Curso puedan cargar adecuadamente sus diseños desde el PC a la FPGA.

Logros realizados

Aunque el objetivo inicial del proyecto incluía todas las fases del desarrollo del prototipo (diseño, fabricación del circuito impreso, montaje, desarrollo del software de programación y testeo), no se han podido cubrir todas con la suficiente profundidad y los esfuerzos se han centrado en el diseño e implementación de la parte mas novedosa del circuito, es decir, la interfaz entre el PC y la FPGA y en la realización del software de comunicación.

Se ha implementado, por tanto, un pequeño prototipo con el controlador USB, junto con su memoria EEPROM de configuración y su conector USB, que se comunica con una FPGA a través de 4 líneas de datos. La FPGA usada no ha sido la Spartan-II anteriormente mencionada, sino un modelo antiguo de la familia XC4010XL. El uso de este componente no trae mayores consecuencias, pues su protocolo de comunicación no tiene diferencias sustanciales con respecto a la Spartan-II. Se ha añadido además como circuitería externa un simple LED con una resistencia para probar un diseño de prueba que encienda el LED.

También se ha desarrollado el software de programación de la FPGA, que permite cargar un diseño en formato .rbt desde el PC, y el software de configuración de la EEPROM, que permite modificar datos del dispositivo USB como el número de serie, identificador del fabricante y del producto o protocolo. Estos programas han sido testados exhaustivamente y se ha comprobado su correcto funcionamiento cargando un diseño que permite encender o apagar el LED de los pines de usuario de la FPGA.

Adicionalmente, se ha realizado un diseño completo de los esquemáticos de toda la placa didáctica, incluyendo toda la circuitería adicional como puertos VGA, puertos IDE, pulsadores, switches, etc. A partir de estas capturas, se podrá generar un floorplanning de la placa, lo que constituye el punto de partida de la fabricación del circuito impreso.

Sin embargo, la fabricación y testeo del circuito impreso no se ha llegado a realizar por falta de tiempo. Para comprender esto, es necesario tener en cuenta las dimensiones de este proyecto, la naturaleza muy diversa de las tareas que supone (programación de alto nivel, programación de bajo nivel, diseño lógico, implementación de protocolos de comunicación, diseño y depuración de circuitos electrónicos y fabricación de PCBs) y la dificultad a la hora de introducirse en el dominio de todas las tecnologías implicadas.

En cualquier caso, siempre queda la posibilidad de plantear el proceso de fabricación del circuito impreso como un nuevo proyecto de Sistemas Informáticos para el próximo curso, partiendo de los logros obtenidos en el proyecto presente (la interfaz PC-FPGA y el software de comunicación han sido concluidos con éxito).

Estructura del documento

El presente documento está dividido en dos partes bien diferenciadas. En la primera se realiza una aproximación tecnológica de todos los dispositivos y software involucrados en la realización del proyecto. Se describen con gran detalle las características y funcionalidades de la FPGA, desde los modos de configuración hasta el patillaje, pasando por la estructura y formato de los archivos de configuración. A continuación se hace un análisis similar del controlador USB, mencionando algunos de los módulos funcionales que lo constituyen, las características de los modos de comunicación usados, el patillaje y las funciones típicas de la Interfaz de Programación de Aplicaciones asociada al controlador. Seguidamente, se resumen las funciones principales de los paquetes software que fueron necesarios, como Visual Studio .NET o Altium DXP, y de los dispositivos de medida usados, como el analizador lógico y los multímetros. Finalmente, se enumeran las características típicas de una placa de circuito impreso y se mencionan brevemente los pasos a seguir durante el proceso de diseño y de fabricación de un PCB.

En la segunda parte de la memoria, se describe todo el desarrollo del proyecto. Se explican todas las etapas en la realización del prototipo: montaje del módulo del controlador USB, programación y depuración del software de comunicación entre el PC y el controlador, análisis de las tramas enviadas a través del puerto, ensamblaje con el módulo de la FPGA y chequeo de la correcta configuración de la misma. Se incluye un manual de usuario del software para configurar la EEPROM y para cargar los diseños en la FPGA. Se añade al final de esta parte una serie de esquemáticos con una propuesta de circuitería adicional de usuario externa a la FPGA. También se describen en detalle todos los problemas encontrados durante el proyecto y cómo se abordaron.

Como apéndice, se muestra el código del software desarrollado y un glosario de términos.

APROXIMACIÓN TECNOLÓGICA

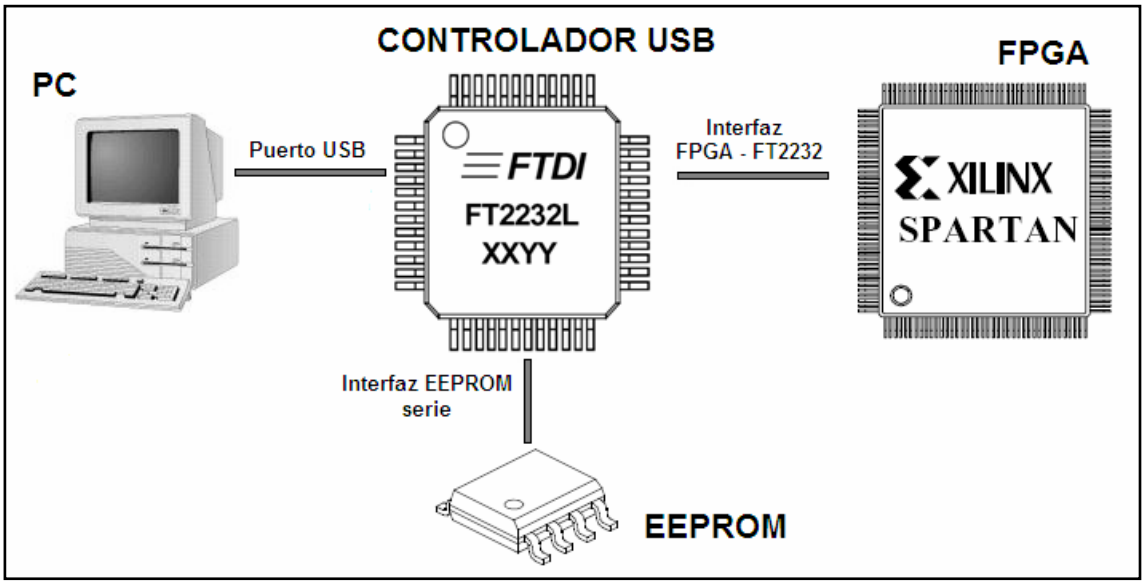
De forma global, se puede considerar que el sistema está formado por cuatro dispositivos que se comunican entre sí para lograr un único objetivo final: que el archivo de configuración de la FPGA generado en el PC se cargue correctamente en la misma. Los cuatro elementos implicados son los siguientes:

- **PC:** En él se crea el diseño desde la herramienta de diseño asistido por ordenador y en él estará instalado el software encargado de enviarlo a través del puerto USB hacia la FPGA.
- **Controlador USB:** Gestiona la comunicación entre el puerto USB del PC y la FPGA.
- **EEPROM:** Contiene la información necesaria para configurar adecuadamente el controlador USB.
- **FPGA:** Es el dispositivo de hardware dinámicamente reconfigurable donde se carga el diseño realizado.

Existen en realidad dos procesos de comunicación distintos. En el primero, se configura la EEPROM para establecer el modo de funcionamiento del controlador USB. El software de configuración de la EEPROM envía los datos de configuración a través del puerto USB, éstos se gestionan en el controlador USB, que los envía por la interfaz serie de la EEPROM. La EEPROM queda programada adecuadamente y el controlador se configura según la información contenida en ella. Si se desea conocer el contenido de la EEPROM desde el PC, los pasos son similares. Por la salida serie de la EEPROM, saldrán los datos hacia el controlador USB, que los enviará por el puerto USB al PC.

En el segundo proceso de comunicación, se genera un archivo de configuración .rbt en el PC y se envía gracias al software de carga del diseño en la FPGA. El archivo sale serializado del ordenador a través del puerto USB. El controlador USB, correctamente configurado gracias a la EEPROM, reenviará esta trama a la FPGA a través de su interfaz de entrada/salida. Esta interfaz está constituida por 8 líneas (algunas serán configuradas como salidas, otras como entradas), dedicándose al menos una para enviar los datos que vienen desde el USB en serie (archivo .rbt serializado) y otra para la señal de reloj que sincroniza los datos. Cuando la FPGA queda correctamente configurada, envía una señal de confirmación en el sentido contrario. La señal saldrá por una de las 8 líneas de la interfaz de E/S del controlador USB. Éste enviará esta señal al PC, que será recogida desde el software de carga, confirmándose que la FPGA se configuró adecuadamente.

Figura 1: Diagrama global de bloques e interfaces



Antes de exponer todo el desarrollo del proyecto, es necesario describir con el suficiente detalle el funcionamiento de los dos dispositivos principales de la placa: la FPGA donde se cargará el diseño de los alumnos y el controlador USB que permitirá la comunicación entre el PC y la FPGA. Se comentarán además los rasgos principales de los paquetes software utilizados y de los dispositivos de medidas necesarios para la depuración del prototipo realizado. Junto con esto, se mencionarán algunas de las características de la placa de circuito impreso que constituye el soporte físico y eléctrico de estos componentes, indicando el proceso de diseño y fabricación de la misma.

FPGA

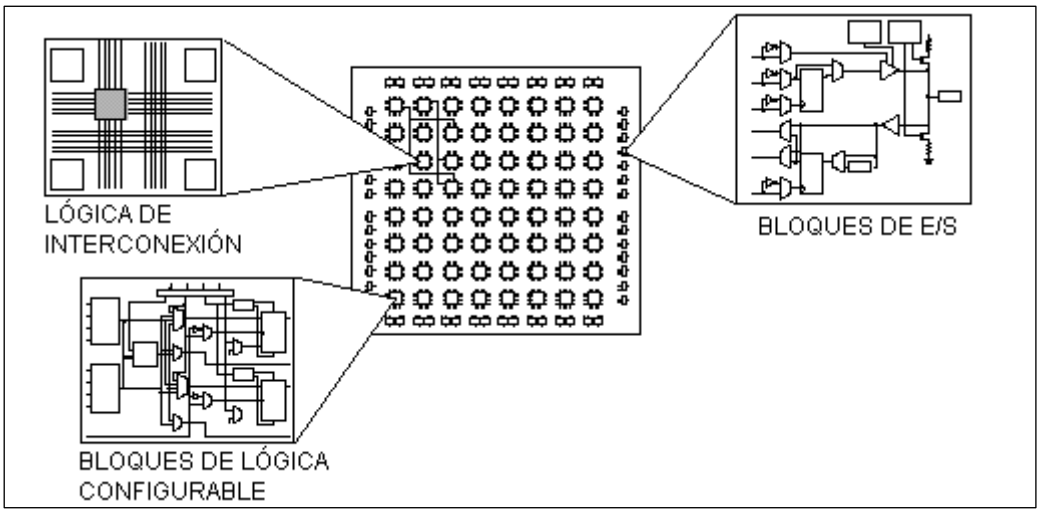
Generalidades

FPGA es el acrónimo de *Field-Programmable Gate Array*. Es un dispositivo hardware dinámicamente reconfigurable, donde se pueden programar infinidad de diseños digitales distintos.

Las FPGAs son una alternativa superior a las mask-programmed ASCIs (Application-Specific Integrated Circuit), con las cuales es imposible hacer actualizaciones de diseño sin reemplazamiento de hardware.

Tres son las principales partes de toda FPGA: los Bloques de Lógica Configurables (Configurables Logic Blocks, CLBs), los Bloques de Entrada/Salida (Input/Output Blocks, IOBs) y toda la lógica de interconexión de los elementos funcionales de la FPGA (figura 1). A continuación se tratará brevemente de cada parte.

Figura 2: Bloques funcionales principales de la FPGA



Cada CLB tiene un conjunto de celdas lógicas (Logic Cells, LC). Un LC contiene un generador de funciones implementado, a su vez, con un look-up table. Además de los cuatro LCs, cada CLB contiene lógica que combina los generadores de funciones para permitir funciones de mayor complejidad.

Los IOBs proporcionan interfaz entre los pines externos y la lógica interna. Cada IOB controla un pin, que puede ser configurado como entrada, como salida o como pin bidireccional. Además cada uno tiene tres registros que comparten la señal de reloj (CLK), pero con distintas señales de capacitación de reloj (CE).

Con la lógica de interconexión concluye la descripción de los elementos funcionales. Su principal característica es que las interconexiones están dirigidas, como el resto de los elementos lógicos, por los valores guardados en las celdas de memoria internas.

Una FPGA debe ser configurada para poder simular un diseño digital, donde configuración se define como el proceso mediante el cual el bitstream (flujo de unos y ceros) de un diseño se carga en la memoria de configuración de la FPGA. La parte central del proyecto consistió en realizar esta labor, en programar o configurar una FPGA mediante el envío de señales generadas por software.

La FPGA con la que se ha trabajado es la XC2S100-5 TQ144C de familia Spartan-II de Xilinx. La familia Spartan-II se caracteriza por cargar los datos de configuración en celdas internas de memoria estática. Los valores guardados en estas celdas determinan las funciones lógicas y las interconexiones implementadas en la FPGA. El hecho de guardar los valores dentro de estas celdas permite infinitas reprogramaciones del dispositivo.

El nombre de la FPGA, XC2S100-5 TQ144C, indica lo siguiente:

- **Tipo de dispositivo:** XC2S100, dispositivo 100 de la familia Spartan II.
- **Grado de velocidad:** 5, estándar.
- **Tipo de embalaje:** TQ, Plastic Thin with Quad Flat Pack.
- **Número de pines:** 144.
- **Rango de temperatura:** C, comercial (entre 0º y 85º).

Patillaje

Hay dos tipos de pines en los dispositivos Spartan-II: pines dedicados, que realizan sólo funciones específicas relacionadas con la configuración, y el resto de pines, que sirven como pines de entrada/salida de propósito general una vez el dispositivo está en modo usuario.

La siguiente tabla muestra los distintos tipos de pines que se pueden encontrar en una FPGA y su descripción.

Nombre	Dedicado	Dirección	Descripción
GCK0, GCK1, GCK2, GCK3	No	Entrada	Pines de entrada de reloj para conectar los Buffers de reloj globales. Estos pines pasan a ser pines de usuario tras la configuración.
M0, M1, M2	Sí	Entrada	Usados para especificar el modo de programación.
CCLK	Sí	Entrada o Salida	Pin de reloj durante la configuración. Es de entrada para los modos Slave-Parallel y Slave-Serial y de salida para el modo Master-Serial.
/PROGRAM	Sí	Entrada	Cuando se pone en baja inicializa la secuencia de configuración.
DONE	Sí	Bidireccional	Indica que la configuración ha terminado, y que la secuencia de Start-Up esta en proceso. Si se usa como salida debe ser colector abierto.
/INIT	No	Bidireccional (colector abierto)	Cuando esta en baja, indica que la memoria de configuración se está limpiando. Este pin se convierte en pin de E/S usuario después de la configuración.
BUSY/DOUT	No	Salida	En el modo Slave Parallel, BUSY controla la velocidad a la que los datos de configuración se cargan. Este pin se convierte en pin de E/S usuario después de la configuración, salvo si se mantiene el puerto Slave Parallel. En los modos serie, si se configura en cadena (daisy-chain) DOUT suministra los datos de configuración al siguiente dispositivo de la cadena.
D0/DIN, D1, D2, D3, D4, D5, D6, D7	No	Entrada o Salida	En el modo Slave Parallel, D0-D7 son lo pines de entrada de los datos de configuración. En los modos en serie, DIN es el pin de entrada única de datos Este pin se convierte en pin de E/S usuario después de la configuración.

WRITE	No	Entrada	En el modo Slave Parallel, es la señal Write Enable, activa en baja. Este pin se convierte en pin de E/S usuario después de la configuración, salvo si se mantiene el puerto Slave Parallel.
CS	No	Entrada	En el modo Slave Parallel es la señal Write Enable, activa en baja. Este pin se convierte en pin de E/S usuario después de la configuración, salvo si se mantiene el puerto Chip Select.
TDI, TDO, TMS, TCK	Sí	Mixta	Pines para el acceso al puerto Boundary Scan (IEEE 1149.1).
VCCINT	Sí	Entrada	Pines de suministro de potencia para el núcleo de lógica interna.
VCCO	Sí	Entrada	Pines de suministro de potencia para señales de salida
VREF	No	Entrada	Pines de entrada del voltaje umbral. Se convierte en pin de E/S usuario cuando dicho voltaje no es necesario.
GND	Sí	Entrada	Tierra
IRDY, TRDY	No	Dependiente de PCI core	Sólo se puede acceder a estas señales si usamos Xilinx PCI cores. Si no es así, estos pines se usan como pines de E/S usuario.

En total hay 144 pines en la XC2S100-5 TQ144C, cuyo desglose ve a continuación:

Tipo de pin	Número de pines
I/O	52
VCC	12
I/O, VREF	24
VCCINT	8
GND	16
GCK, GCK, GCK, GCK,	4
M0, M1, M2	3
CCLK	1
/PROGRAM	1
DONE	1
/INIT	1
BUSY/DOUT	1
D0/DIN, D1, D2, D3, D4, D5, D6, D7	8
/WRITE	1
/CS	1
TDI, TDO, TMS, TCK	4
IRDY, TRDY	4
No conectados	2

Modos de programación

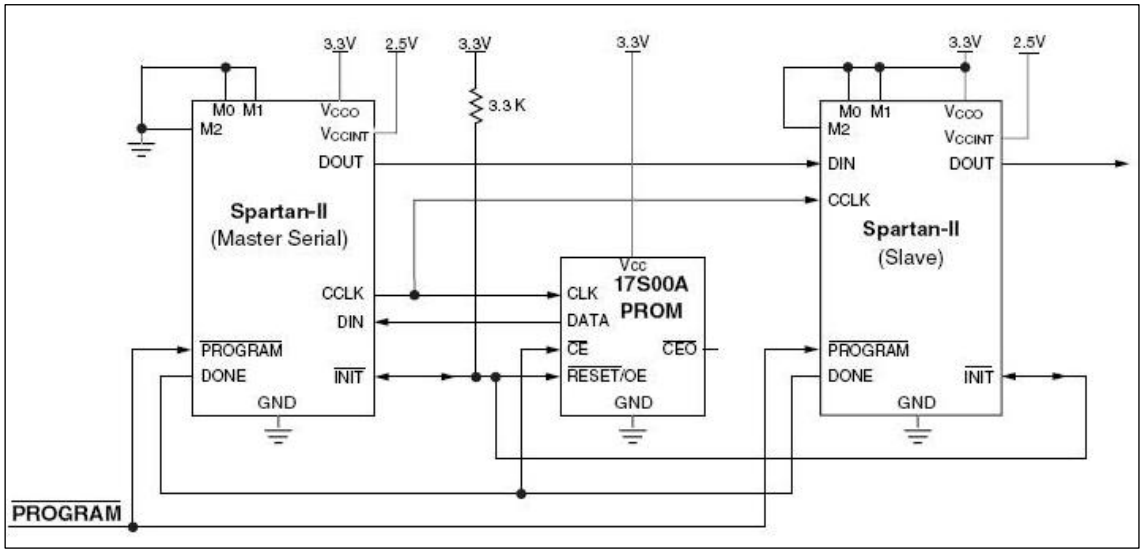
Las FPGAs de familia Spartan-II pueden ser configuradas en cuatro modos principalmente: Master Serial, Slave Serial, Slave Parallel y Boundary Scan, que se explican brevemente a continuación.

En el modo Master Serial, la FPGA controla el proceso de configuración, generando una señal de reloj (por el pin CCLK) como salida que es recibida por una memoria PROM externa, que contiene los datos de configuración. La PROM envía estos datos bit a bit (de ahí que éste sea un modo serie), a la patilla DIN de la FPGA en cada flanco de subida del reloj.

En el modo Slave Serial, la FPGA recibe pasivamente CCLK como entrada de un dispositivo externo, que puede ser un microprocesador, una segunda FPGA en modo Master o, lo que se ha realizado en este proyecto, utilizando un controlador, en este caso un controlador USB. Al igual que en el modo Master, en cada flanco de subida de reloj, se lee el contenido de su patilla DIN.

Existe la posibilidad de programar varias FPGAs a la vez usando el método daisy-chain (configuración en cadena), conectando, entre otras cosas, el pin DOUT de una FPGA, al DIN de la siguiente. Esto se realiza añadiendo al diseño del modo Master Serial, una o varias FPGAs en modo Slave Serial (figura 2). Después de que una FPGA carga un bit de configuración, su pin DOUT, tras el siguiente flanco de subida del reloj, presenta el dato de configuración para el próximo dispositivo en la cadena.

Figura 3: Diagrama del circuito de configuración Master/Slave



Por otro lado está el modo Slave Parallel, que es el método de programación más rápido, ya que los bits llegan de ocho en ocho por los pines D0:D7.

Por último, para finalizar la explicación de los modos de programación, queda el modo Boundary Scan, que se realiza por un puerto especial, el Test Access Port (TAP). Utiliza cuatro pines dedicados: TDI, TDO, TMS, TCK. Para configurar mediante TAP necesitamos una instrucción especial CFG_IN, que permite a los datos que entran por TDI, ser convertidos en paquetes de datos para el bus de configuración interno de la FPGA.

En cada uno de los cuatro modos se puede elegir entre tener resistencias de pull-up o no. Luego, en total, hay ocho modos diferentes de programación que se pueden seleccionar dando distintos valores a los pines M0, M1 y M2.

Modo de configuración	M0	M1	M2	Pull-ups
Master Serial	0	0	1	No
Slave Serial	1	1	1	No
Slave Parallel	0	1	1	No
Boundary Scan	1	0	1	No
Master Serial (con Pull ups)	0	0	0	Sí
Slave Serial (con Pull ups)	1	1	0	Sí
Slave Parallel (con Pull ups)	0	1	0	Sí
Boundary Scan (con Pull ups)	1	0	0	Sí

De los modos de programación disponibles, se eligió, para este proyecto, el modo Serial Slave. En serie, por su sencillez respecto al paralelo. Y en Slave, para no necesitar una PROM y por ser más sencillo el proceso. Se utilizó un controlador USB como generador de las señales de configuración de la FPGA (PROGRAM, CCLK, DIN). En la siguiente tabla se muestra una descripción de cada señal usada en la configuración.

Señal	Tipo	Dirección	Descripción
M0, M1, M2	Selección de modo	Entrada	Especifican el tipo de configuración: en este proyecto se va a trabajar en modo Slave Serial y con pull-ups, luego deben valer M0=1, M1=1, M2=0.
PROGRAM	Control	Entrada	Cuando se pone en baja inicia la secuencia de configuración, comenzando por la limpieza de memoria de configuración (mientras ésta se lleva a cabo, la señal INIT esta en baja).
CCLK	Reloj	Entrada	Puede ser entrada o salida. Para el modo Slave Serial es de entrada. Durante la configuración lee un dato de DIN en cada flanco de subida.
DIN	Datos	Entrada	En el modo Slave Serial es el único pin de entrada de datos. Se convierte en pin de I/O de usuario después de la configuración.
DONE	Estado	Salida	Cuando se pone en alta, indica que se ha completado la carga de las tramas de configuración, lo que significa que tras la secuencia de Start-Up, la FPGA pasará a modo usuario. Para su lectura se requiere una resistencia pull-up.

El proceso de Configuración

La configuración es el proceso mediante el cual el bitstream de un diseño se carga en la memoria de configuración de la FPGA.

El proceso de configuración se resume en la figura 4.

Éste se realiza en cuatro pasos:

- Limpieza de la memoria de configuración.
- Inicialización
- Carga de los data-frames
- Start-Up

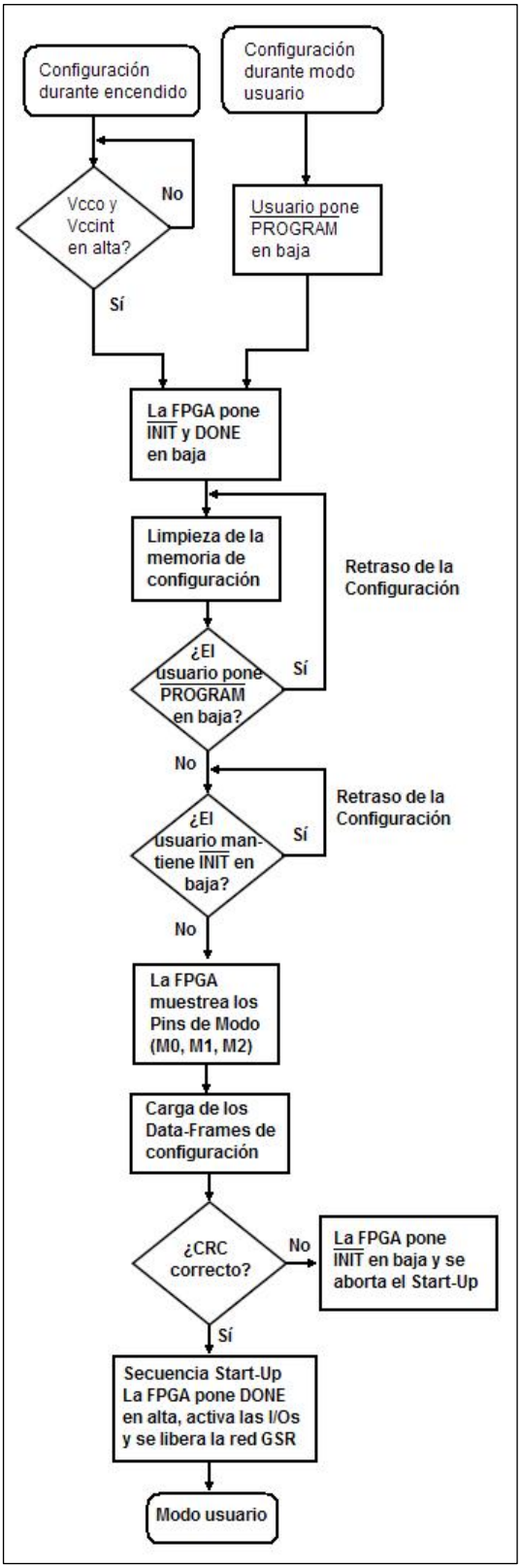
Tras el encendido, después de que Vcc alcance el umbral de 'Power-on-reset' la FPGA automáticamente comienza la limpieza de la memoria de configuración. Esto también sucede si aplicamos un pulso en baja a la señal /PROGRAM. Esto hace posible la reconfiguración de la FPGA sin tener que reiniciarla: si el dispositivo esta en modo usuario y se aplica un pulso en baja en /PROGRAM, el dispositivo automáticamente comienza el proceso de configuración poniendo la señal DONE en baja, entrando a continuación en la fase de limpieza de la memoria de configuración.

La documentación de la Spartan-II recomienda no mantener /PROGRAM en baja por más de 500 µs.

Una vez la memoria de configuración está limpia, lo cual se indica con una transición de baja a alta de la señal /INIT, comienza la inicialización, en la que el dispositivo muestrea las señales M0, M1 y M2 e identifica el modo en que se va realizar la configuración.

La fase de carga de los data-frames consiste en serializar el archivo de configuración o bitstream, que contiene las tramas de configuración o data-frames y enviárselos a la FPGA.

Figura 4: Diagrama de flujo del proceso de configuración



El chequeo de errores se realiza durante la carga del bitstream, el valor CRC que está dentro de cada data-frame se compara con un valor CRC calculado por la FPGA. Si ambos valores no coinciden, la FPGA pone /INIT en baja para indicar que ha ocurrido un error durante el envío de tramas y la configuración es abortada.

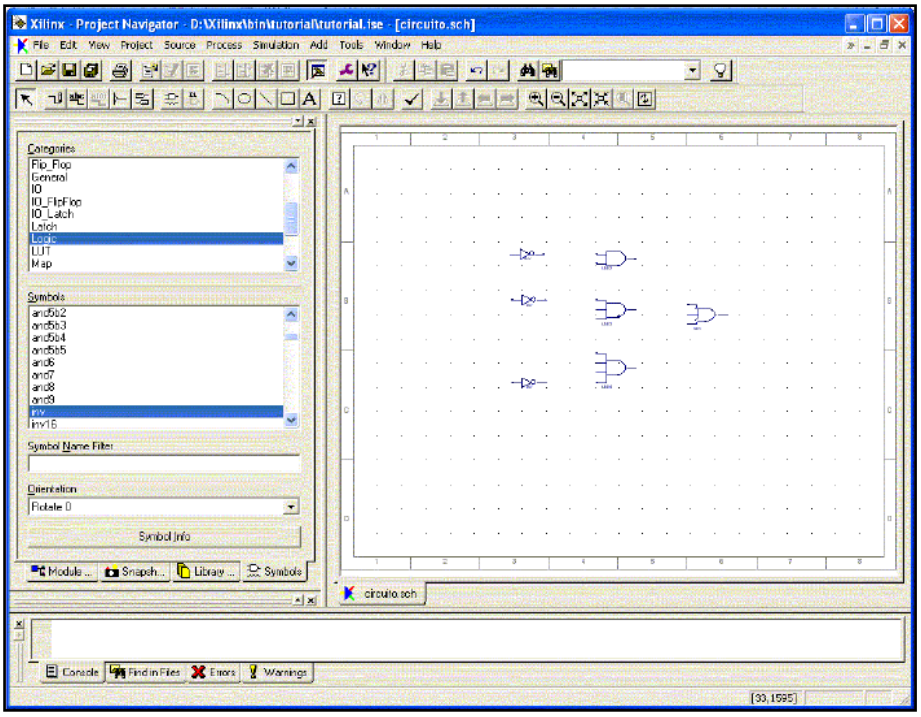
La secuencia de Start-Up comienza si no ha habido errores durante el chequeo CRC. Esta secuencia supervisa la transición de la FPGA de la fase de configuración a la fase de operación de usuario. Realiza cuatro operaciones, pero la única involucrada en el desarrollo del proyecto es la activación de la señal DONE.

Una vez finalizado el proceso de configuración la FPGA pasa a modo usuario.

Generación de configuraciones

La configuración a cargar en la FPGA se obtiene con algún software de desarrollo de Xilinx, por ejemplo el Xilinx ISE (mostrado en la figura 5) o el Xilinx Foundations (algo más antiguo). Estas herramientas pueden generar el archivo de configuración en varios formatos, pero el más manejable, aunque no el más compacto, es el formato de archivo rawbits, es decir, la codificación del bitstream en ASCII utilizando un byte (un carácter) por cada bit de configuración.

Figura 5: Software para la realización de diseños con esquemáticos Xilinx ISE



Formato	Extensión	Descripción
Rawbits	.RBT	El bitstream esta codificado en ASCII, un byte por cada bit de configuración.
Hexadecimal	.HEX	Cada grupo de cuatro bits de configuración consecutivos se representa como un dígito hexadecimal que, a su vez, es codificado en un byte ASCII.
Binario	.BIT	El bitstream esta codificado en binario, formado por un bit de configuración tras otro.

Una ventaja del archivo RBT es que puede verse su contenido en cualquier editor de texto (ver figura 6), lo cual nos permite ver su configuración interna. El archivo tiene dos partes claramente diferenciadas, la información del archivo de configuración y el bitstream que debe ser enviando a la FPGA. La primera parte da la siguiente información:

- Formato del archivo de configuración.
- Versión de sistema de desarrollo de Xilinx que se ha usado.
- El nombre del diseño.
- El dispositivo objetivo.
- Fecha en que se creó el archivo.
- El número de bits del archivo de configuración (para los dispositivos XC2S100 el tamaño es de 781.216 bits).

A continuación de esta información viene el bitstream que comienza con 40 bits de cabecera y justo después con primer data-frame. Todos los data-frames, comienzan con cero y terminan con cuatro bit CRC para el chequeo de errores. El bitstream termina con el código 0111111, seguido de los ocho bits de la secuencia Start-Up (todo unos).

Figura 6: Muestra de un archivo .RBT de Spartan

```
Xilinx ASCII Bitstream
Created by Bitstream M1.5
Design name: cntlogix.ncd
Architecture:spartan
Part: s30pq240
Date: Tue Sep 29 16:40:13 1998
Bits: 247968
11111111001000000011110010001001100111111
01010111111111111101111010111111101011111111
01011111110101111111010111111101011111111010
11111110101111111010111111101011111111010111
11110110111111110101111111010111111110101111
11010111111101011111110101111111010111111110
10111111101011111110101111111010111111110111
11110010
.
.
.
01111111111111101111111010111111101011111111
01011111110101111111010111111101011111111010
11111110101111111010111111101011111111010111
11110110111111110101111111010111111110101111
11010111111101011111110101111111010111111110
10111111101011111110101111111010111111110011
01110010011111111111111111
```

NOTA: Para cada modelo distinto de FPGA, el software de desarrollo de Xilinx genera un archivo RBT ligeramente distinto. La aplicación *Loader* desarrollada funciona siempre que la cabecera del archivo este compuesta por siete líneas.

CONTROLADOR USB

Como se ha comentado antes, una de las modificaciones más importantes introducidas en la nueva placa con respecto a las actuales está en la interfaz con el PC. La placa diseñada se comunicará con el ordenador a través del puerto USB. Esto permitirá usar un protocolo muy reciente, más fiable y robusto que el LPT, de instalación muy cómoda (la placa se configurará al conectar el USB) y con la posibilidad de prescindir de una fuente de alimentación externa.

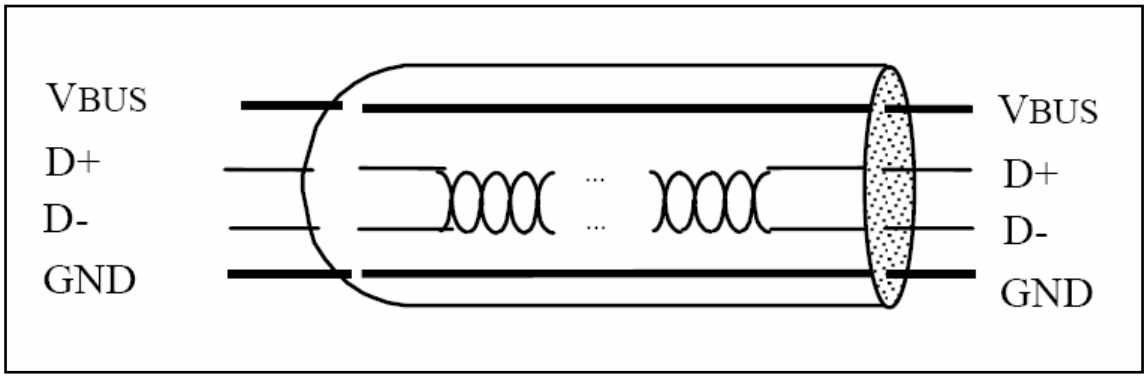
Protocolo USB

El protocolo USB (Universal Serial Bus) es un protocolo serie diseñado originariamente para PCs y Macintosh, pero su popularidad hizo que se comenzará a usar también en videoconsolas, cámaras fotográficas, reproductores de música o dispositivos de almacenamiento.

A nivel eléctrico, el cable USB transfiere la señal y la alimentación sobre 4 hilos. La alimentación se suministra por el primero (VBUS) con una tensión nominal de 5 V. El cuarto establece la masa (GND). La señal se transmite a través de un par trenzado con impedancia característica de 90 Ω. El reloj se incluye en el flujo de datos (la codificación es de tipo NRZI, existiendo un dispositivo que genera un bit de relleno que garantiza que la frecuencia de reloj permanezca constante) y cada paquete va precedido por un campo de sincronismo.

En cualquier caso, existen dos formas de suministrar la energía. El periférico puede obtener la energía del PC o host a través del bus (bus powered) o puede estar autoalimentado (self powered). En el primer caso, es el ordenador el que gestiona el consumo, teniendo capacidad de poner en reposo (suspend) o en marcha al periférico USB. En reposo, este reduce su consumo (si puede), quedándose la parte USB funcional (ideal para equipos portátiles).

Figura 7: Los cuatro hilos de un conector USB



Un sistema USB tiene un diseño asimétrico, que consiste en un solo servidor y múltiples dispositivos conectados en una estructura de árbol utilizando concentradores especiales. Se pueden conectar hasta 127 dispositivos a un solo servidor, pero la suma debe incluir a los concentradores también, así que el total de dispositivos realmente usables es algo menor. La ramificación máxima es de 5 niveles.

El USB es un bus basado en encuesta (polling), no hay interrupciones. Es un bus punto a punto: dado que el lugar de partida es el host (PC o hub), el destino es un periférico u otro hub. Los periféricos comparten la banda de paso del USB. El protocolo se basa en el llamado paso de testigo (token). El ordenador proporciona el testigo al periférico seleccionado y seguidamente, éste le devuelve el testigo en su respuesta. Este bus permite la conexión y la desconexión en cualquier momento sin necesidad de apagar el equipo.

El protocolo USB soporta tres tasas de transferencia: Low Speed (1.5 Mbit/s), Full Speed (12 Mbit/s) y Hi-Speed (480 Mbit/s). Los dispositivos con tasa de transferencia de 480 Mbit/s suelen denominarse como dispositivos USB 2.0, aunque no siempre se da el caso de que un dispositivo etiquetado como USB 2.0 alcance esa velocidad de transferencia (es el caso de controlador USB utilizado, que sólo llega a los 12 Mbits/s, pero se denomina "USB 2.0"). Para cada tasa de transferencia, existe un límite en el tamaño de información útil que se puede enviar en cada paquete, siendo de 64 bytes cuando se selecciona el modo Full Speed.

El protocolo USB es flexible, soporta una interfaz serie bidireccional isócrona y permite una cómoda instalación dinámica al conectar el dispositivo al PC.

Descripción del chip FT2232C

Existen infinidad de dispositivos controladores USB en el mercado. Son bastante populares los controladores de USB de Philips, como el ISP1362, o los de Cypress, como el CY7C68013. En este proyecto, se ha escogido el chip FT2232C de FTDI. Los motivos de esta elección son diversos. Por un lado, se trata de un componente barato y disponible en la mayor parte de los establecimientos especializados en la venta de productos electrónicos. Por otro lado, es uno de los controladores USB con mayor soporte en Internet por parte del fabricante (existe gran cantidad de documentación acerca del uso del chip y de sus drivers). Finalmente, se decidió usar este dispositivo al existir experiencias previas de su utilización en otros proyectos.

El FT2232C es un controlador USB de tercera generación, diseñado y fabricado por la compañía inglesa FTDI. En realidad, se dispone de dos controladores de propósito general (UART o FIFO) en un mismo chip, de forma que se pueden obtener dos canales de E/S de un único puerto USB, sin necesidad de usar hub alguno. Además de la posibilidad de configurar cada canal como interfaz UART o interfaz FIFO, existen diversas variantes de estos modos de comunicación, como el denominado Bit-Bang Mode (asíncrono y síncrono), el MPSSE (Multi-Protocol Serial Engine Interface), el modo de MCU Host Bus Emulation o el Fast Opto-Isolated Serial Interface.

Además de los dos canales y de los múltiples modos, el chip FT2232C tiene soporte para transferencias isócronas, como las de vídeo y audio de bajo ancho de banda, en las que importa más el cumplimiento de los plazos de una planificación que la integridad de los datos. Otra característica presente en este dispositivo es la opción de comunicación mediante el protocolo USB 2.0, versión con una tasa de transferencia mayor que la del USB 1.0 típico (hasta 12 Mb/s). También es de destacar la posibilidad de alimentar cada canal de E/S de forma independiente, con un nivel de señal entre los 3 y los 5 Voltios.

El dispositivo se encuentra disponible en encapsulados del tipo LQFP (Low Profile Quad Flat Pack) de 48 pines. Existe una versión libre de plomo, denominada FT2232L, que tiene un impacto medioambiental menor.

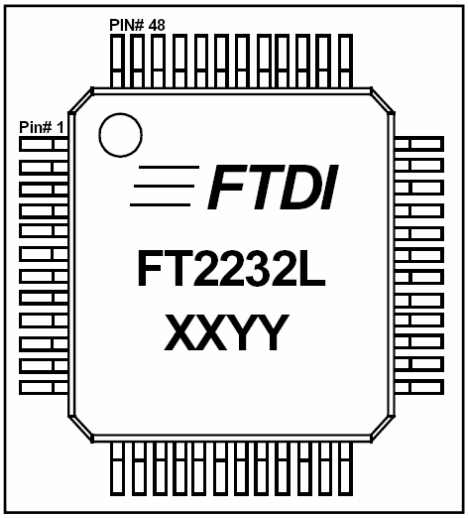


Figura 8: Chip FT2232L

Internamente, se distinguen varios módulos funcionales, entre los que destacan:

- ***Controlador UART / FIFO de propósito general:*** Se encargan de gestionar las transferencias entre los búferes TX y RX y los registros de transmisión y de recepción UART / FIFO. Existe uno por cada canal.

- ***Búfer de Transmisión TX:*** Los datos que vienen desde el puerto USB se guardan en este búfer de 128 bytes antes de ser enviados al registro de transmisión del Controlador UART / FIFO. Existe uno por cada canal.

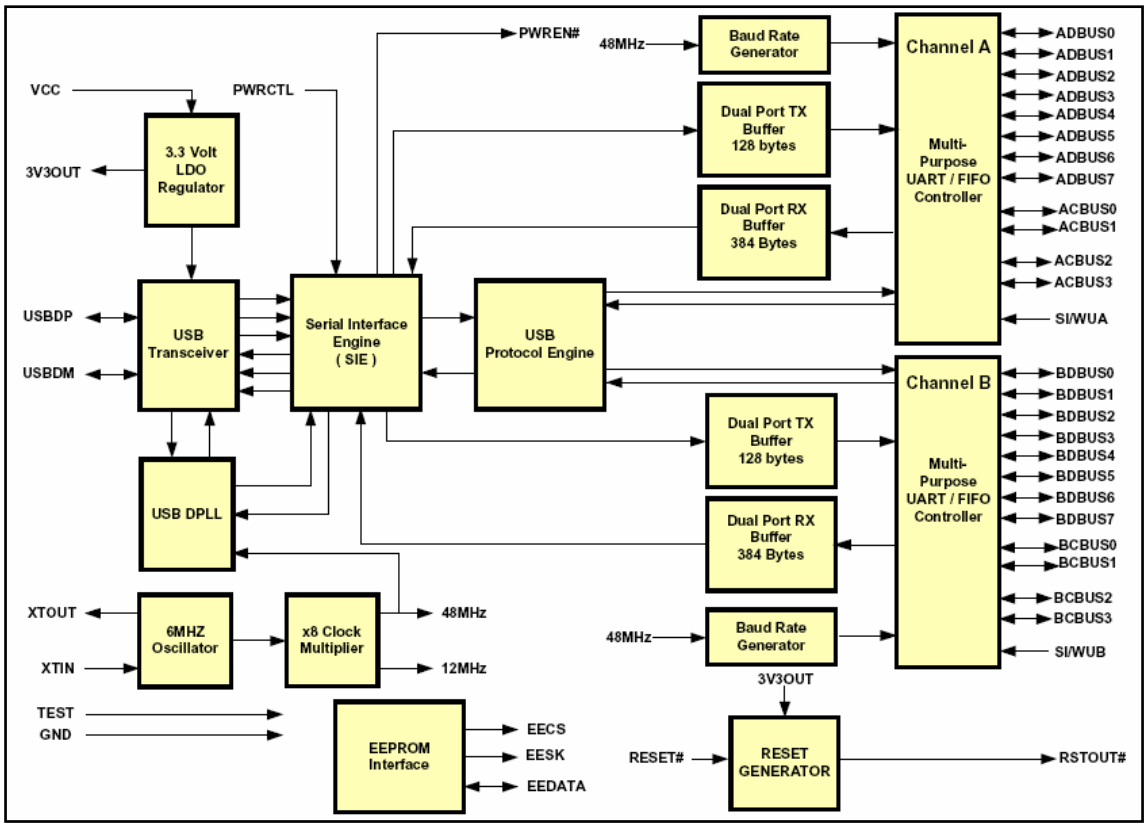
- ***Búfer de Recepción RX:*** Los datos que vienen desde el registro de recepción del Controlador UART / FIFO, se almacenan en este búfer de 384 bytes antes de ser enviados hacia el puerto USB en una petición de datos desde el host. Existe uno por cada canal.

- ***SIE (Serial Interface Engine):*** Es un conversor serie-paralelo y paralelo-serie de los datos que vienen del enlace USB, siguiendo la especificación del protocolo USB 2.0 (genera o elimina bits de relleno, añade o chequea códigos de control de errores, etc.).

- ***Oscilador de 6 MHz:*** Este módulo genera una señal de reloj de referencia de 6 MHz a partir de un cristal o resonador cerámico de 6 MHz externo al chip. De este modo, las señales de reloj destinadas a la FPGA tendrán esta frecuencia como máximo. La señal obtenida se hace pasar además por un multiplicador, consiguiendo así un reloj de 48 MHz para ser utilizado en otros bloques, como el generador de tasa de baudios para el modo UART o el módulo encargado de separar reloj y datos de la señal codificada en NRZI que viene del puerto USB.

- ***Interfaz EEPROM:*** Constituye una interfaz de tres pines con un memoria EEPROM externa de la clase 93C46, 93C56 ó 93C66, que servirá para guardar datos de configuración del dispositivo y establecer el modo de comunicación.

Figura 9: Bloques funcionales del dispositivo FT2232C



Patillaje

Este dispositivo cuenta con 48 pines que se pueden clasificar en los siguientes grupos:

- Interfaz E/S: 26 pines, 13 por cada canal.
- Interfaz eléctrica: 11 pines.
- Interfaz con el USB: 2 pines.
- Interfaz con la EEPROM: 3 pines.
- Interfaz con el oscilador externo: 2 pines.
- Señales de reseteo del dispositivo: 2 pines.
- Señal de modo suspendido (USB suspend): 1 pin.
- Señal de test: 1 pin.

Los 13 pines de cada canal dedicados a la entrada / salida se pueden dividir a su vez en 8 líneas de datos y 5 de control. El comportamiento y tipo de estos pines depende del protocolo de comunicación elegido, pero tienen un nombre genérico que se recoge en la siguiente tabla:

# Pin	Señal	Tipo	# Pin	Señal	Tipo
24	ADBUS0	E/S Datos	40	BDBUS0	E/S Datos
23	ADBUS1	E/S Datos	39	BDBUS1	E/S Datos
22	ADBUS2	E/S Datos	38	BDBUS2	E/S Datos
21	ADBUS3	E/S Datos	37	BDBUS3	E/S Datos
20	ADBUS4	E/S Datos	36	BDBUS4	E/S Datos
19	ADBUS5	E/S Datos	35	BDBUS5	E/S Datos
17	ADBUS6	E/S Datos	33	BDBUS6	E/S Datos
16	ADBUS7	E/S Datos	32	BDBUS7	E/S Datos
15	ACBUS0	E/S Control	30	BCBUS0	E/S Control
13	ACBUS1	E/S Control	29	BCBUS1	E/S Control
12	ACBUS2	E/S Control	28	BCBUS2	E/S Control
11	ACBUS3	E/S Control	27	BCBUS3	E/S Control
10	SI/WUA	E/S Control	26	SI/WUB	E/S Control

Las 11 patillas de la interfaz eléctrica son:

# Pin	Señal	Tipo	Descripción
3, 42	VCC	Potencia	Voltaje de alimentación (entre 4.35 V y 5.25 V). Alimenta el núcleo del chip, el regulador interno de nivel y los pines que no son de los Controladores UART / FIFO.
9, 18, 25, 34	GND	Potencia	Tierra.
14	VCCIOA	Potencia	Voltaje de alimentación (entre 3.0 V y 5.25 V) de los pines de E/S del canal A. Alimenta el Controlador UART / FIFO A.
31	VCCIOB	Potencia	Voltaje de alimentación (entre 3.0 V y 5.25 V) de los pines de E/S del canal B. Alimenta el Controlador UART / FIFO B.
6	3V3OUT	Salida	Salida de 3.3 V del regulador interno de nivel.
46	AVCC	Potencia	Alimentación analógica para el multiplicador de la frecuencia de reloj interna.
45	AGND	Potencia	Tierra analógica para el multiplicador de la frecuencia de reloj interna.

La interfaz con el USB está constituida por tan sólo dos patillas:

# Pin	Señal	Tipo	Descripción
7	USBDP	Entrada/Salida	Señal de datos del USB (positiva).
8	USBDM	Entrada/Salida	Señal de datos del USB (negativa).

La interfaz con el oscilador externo de 6 MHz, los pines de test y de modo suspendido y los de reset se muestran en la siguiente tabla:

# Pin	Señal	Tipo	Descripción
43	XTIN	Entrada	Entrada al módulo del oscilador de 6 MHz.
44	XTOUT	Salida	Salida al módulo del oscilador de 6 MHz.
4	RESET#	Entrada	Permite el reseteo del chip desde el exterior.
5	RSTOUT#	Salida	Salida del generador de reset interno.
41	PWREN#	Salida	Si está en alta, indica que el chip está en modo suspendido.
47	TEST	Entrada	Sitúa al dispositivo en el modo de testeo.

La interfaz con la EEPROM está constituida por 3 pines:

# Pin	Señal	Tipo	Descripción
1	EESK	Salida	Señal del reloj a la EEPROM.
2	EEDATA	Entrada/Salida	Datos hacia y desde la EEPROM.
48	EECS	Entrada/Salida	Chip Select de la EEPROM.

Descripción de protocolos

Las dos interfaces principales que pueden adoptar tanto el canal A como el B del dispositivo son la 232 UART (Universal Asynchronous Receiver Transmitter) y la 245 FIFO (First In, First Out). Cada una de estas interfaces se puede configurar a su vez en distintos modos de comunicación. De entre todos los existentes, se explicarán en detalle el modo FIFO y una de sus variantes, el BitBang Mode Asíncrono, ya que es éste último el que se ha adoptado para la comunicación entre el controlador USB y la FPGA.

Cuando los canales se configuran como FIFO, las líneas de entrada/salida toman el siguiente comportamiento:

# Pin	Señal	Tipo	Descripción
24 / 40	D0	Entrada/Salida	Bit 0 del bus de datos FIFO (canal A / B).
23 / 39	D1	Entrada/Salida	Bit 1 del bus de datos FIFO (canal A / B).
22 / 38	D2	Entrada/Salida	Bit 2 del bus de datos FIFO (canal A / B).
21 / 37	D3	Entrada/Salida	Bit 3 del bus de datos FIFO (canal A / B).
20 / 36	D4	Entrada/Salida	Bit 4 del bus de datos FIFO (canal A / B).
19 / 35	D5	Entrada/Salida	Bit 5 del bus de datos FIFO (canal A / B).
17 / 33	D6	Entrada/Salida	Bit 6 del bus de datos FIFO (canal A / B).
16 / 32	D7	Entrada/Salida	Bit 7 del bus de datos FIFO (canal A / B).

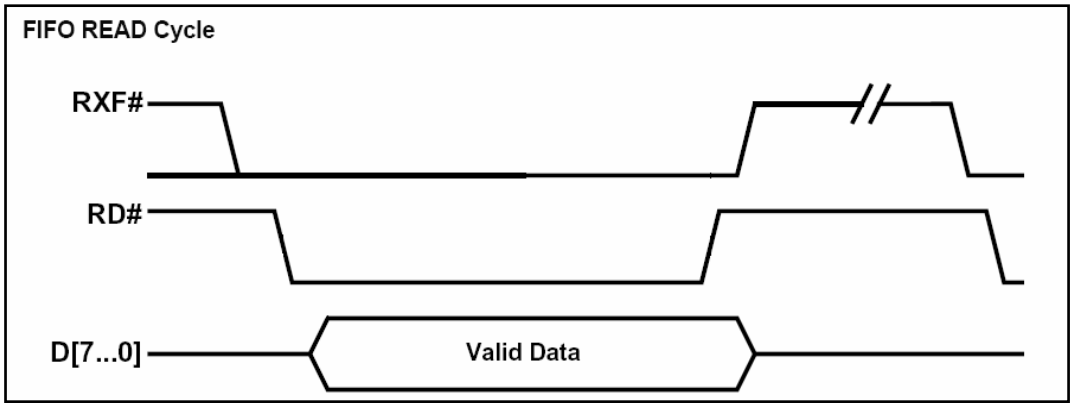
Las señales de control de la interfaz FIFO son las siguientes:

# Pin	Señal	Tipo	Descripción
12 / 28	RD#	Entrada	En baja, se leen los datos disponibles a la entrada de los pines D0..D7. Cuando esta señal pasa de baja a alta, se realiza un fetch del siguiente byte de datos del búffer de recepción del chip.
11 / 27	WR	Entrada	Cuando esta señal pasa de alta a baja, se escribe el byte contenido en el búffer de transmisión del chip a las líneas de datos D0..D7.
15 / 30	RXF#	Salida	En alta, indica que no se debe leer datos de la

			FIFO. En baja, indica que hay datos disponibles en la FIFO que se pueden leer poniendo RD# en baja y a continuación en alta.
13 / 29	TXE#	Salida	En alta indica que no se debe escribir datos desde la FIFO. En baja, indica que se pueden escribir datos poniendo WR en alta y seguidamente en baja.
10 / 26	SI/WU	Entrada	Si está en baja, todos los datos del búffer de transmisión del chip se envían en la siguiente llamada, independientemente del tamaño del paquete (permite tasas mayores de transferencia). Si el USB está en modo suspendido (PWREN# vale 1) y se ha habilitado el wakeup remoto en la EEPROM, el efecto de poner en baja este pin es distinto: se reanuda el bus USB, rehabilitando el host.

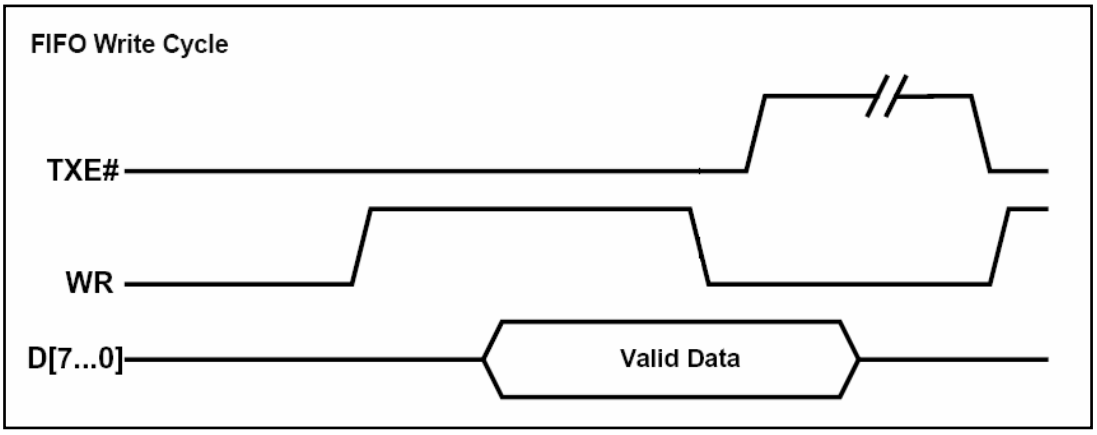
El proceso de lectura con este protocolo es el siguiente: en primer lugar, desde el dispositivo externo al controlador USB, que en este caso sería la FPGA, se lee el estado de la señal de salida RXF#. Si está en baja, significa que hay datos disponibles en las líneas de datos D0..D7. Por tanto, el dispositivo externo pondría en baja la señal de entrada RD#, de forma que se leerían los datos situados en las líneas de datos. A continuación, y respetando el ancho mínimo del pulso RD# (50 ns), la señal RD# volvería a estar en alta, y el byte se introduciría en el búffer de lectura del chip.

Figura 10: Ciclo de lectura en el modo FIFO



El proceso de escritura de datos es análogo. El dispositivo externo comprobaría que la señal TXE# está en baja. Si así fuera, significaría que hay datos disponibles en el búffer de transmisión para ser escritos, y pondría la señal WR en alta, y se escribirían los datos hacia las líneas D0..D7.

Figura 11: Ciclo de escritura en el modo FIFO



Este protocolo, por tanto, requiere circuitería externa para su control (el dispositivo origen y destino de los datos, es decir, la FPGA, debería ser el que generara las señales RD# y WR adecuadamente). Esto puede acarrear varias dificultades añadidas, debido a que los dos dispositivos (FPGA y controlador USB) deberían estar muy bien sincronizados.

Existen dos variantes de este interfaz, que son los modos Bit Bang Mode Asíncrono y Bit Bang Mode Síncrono, que no requieren circuitería externa adicional para gestionar el control del protocolo. En estos modos, las 8 líneas de datos de cada canal se convierten en sendos buses bidireccionales de 8 bits. Cada pin de datos se puede configurar como salida o como entrada. Las líneas de control, a excepción de la SI/WU, se convierten en salidas, tal y como muestra la siguiente tabla:

# Pin	Señal	Tipo
12 / 28	RD#	Salida
11 / 27	WR#	Salida
15 / 30	RD#	Salida
13 / 29	WR#	Salida
10 / 26	SI/WU	Entrada

De esta forma, con este modo, la FPGA no debe generar las señales WR y RD# para solicitar datos desde el USB o indicar que va a escribir en el mismo. Cuando hay un nuevo dato en un pin de datos configurado como entrada (es decir, cuando la FPGA escribe hacia el USB), este dato se recoge al acontecer un flanco del reloj interno del controlador USB. Si se escribe desde el controlador a la FPGA, el nuevo dato se situará en el pin de datos configurado como salida en un flanco de reloj. Si los datos no se modifican, los pines mantendrán el último valor escrito en los sucesivos ciclos. En definitiva, los datos se escriben y se leen en base a la señal de reloj interna del controlador USB, y no esperan que un dispositivo externo gestione los ciclos de lectura ni de escritura.

El Bit Bang Mode Síncrono tan solo se diferencia del Asíncrono en que los datos sólo salen del dispositivo justo después de que se escriba en él (es decir, si hay espacio para leer de los pines de entrada).

Para la comunicación entre USB y FPGA se eligió por tanto el modo Bit Bang Mode Asíncrono, ya que facilita enormemente la transmisión de las tramas de configuración desde el controlador USB a la FPGA (simplemente se configuran un conjunto de líneas del controlador como salidas y por ellas se envían en serie los bits de configuración con las señales de reloj y de control, sin necesidad de que la FPGA devuelva señal de confirmación para cada dato recibido). La comunicación desde la FPGA al USB sólo consiste en una señal indicando que todo el conjunto de las tramas de configuración se ha recibido correctamente (DONE) y para enviar esta señal también el modo Bit Bang es adecuado: la FPGA envía esta señal a un pin de entrada del USB y éste lo lee sin más complicación.

Como se mencionó inicialmente, existen muchos otros modos de comunicación disponibles en este dispositivo, además de los ya descritos. Son los siguientes:

- El modo MPSSE (Multi-Protocol Synchronous Serial Engine) se ha diseñado para que el dispositivo FT2232C pueda comunicarse eficientemente con protocolos síncronos serie, como JTAG o SPI Bus. También se puede usar para programar FPGAs basadas en memorias SRAM a través del puerto USB. En todo caso, MPSSE es tan flexible que puede configurarse para permitir la comunicación con cualquier protocolo síncrono serie, ya sea un protocolo estándar o un protocolo propietario.

- El Fast Opto-Isolated Serial Interface permite una comunicación con una tasa de transferencia máxima a través de sólo 4 cables de datos con aislantes ópticos, consiguiendo un aislamiento galvánico entre el controlador USB y el dispositivo externo.

- El MCU Host Bus Emulation Mode usa los dos canales del controlador USB para hacer que el chip se comporte como un bus host MCU del tipo 8048 / 8051. De esta forma,

los periféricos de estas familias se pueden conectar directamente por USB a través del controlador FT2232C.

- La interfaz UART (Universal Asynchronous Receiver-Transmitter) permite el uso del controlador USB con dispositivos que sigan protocolos serie como el RS-232, el RS-422 ó el RS-485, como es el caso de una gran cantidad de módems actuales.

Configuración por EEPROM

La presencia o no de una EEPROM externa al chip FT2232C juega un papel fundamental en el comportamiento de éste. Si no se conecta ninguna EEPROM (o la EEPROM está vacía), el controlador USB adoptará el protocolo por defecto, que es el 232 UART Mode. Los valores del VID (identificador del fabricante), del PID (identificador del producto), de la Descripción del Producto y del Valor Descriptor de Potencia serán los que vengan de fábrica. El dispositivo no tendrá número de serie alguno.

Si se quiere usar otro protocolo de comunicación diferente del 232 UART, es imprescindible conectar al chip FT2232C una EEPROM y especificar el protocolo deseado en el contenido de la memoria. La EEPROM debe ser del tipo 93C46, 93C56 ó 93C66, con una anchura de palabra de, cómo mínimo, 16 bits y capaz de leer datos a 1Mb con una alimentación de entre 4.35 y 5.25 V.

Para realizar esta conexión, el chip posee tres pines dedicados a la interfaz con la EEPROM (EESK, EEDATA y EECS), mencionados anteriormente.

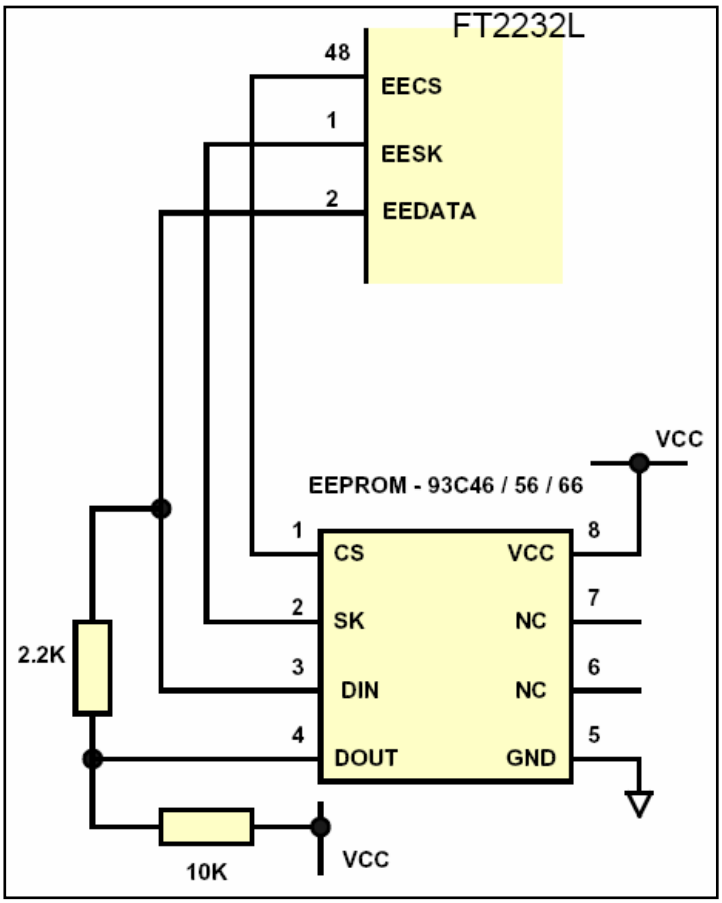


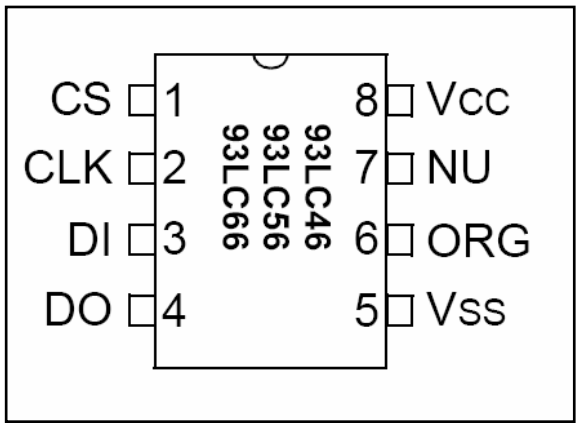
Figura 12: Conexión del FT2232L con la EEPROM

El pin EESK se conectará directamente al reloj de la memoria (SK), al igual que el pin EECS al chip select (CS) de la misma. El pin de datos EEDATA se conecta también de forma directa a la entrada de datos de la EEPROM (DIN), pero se interpone una resistencia de 2.2 KΩ entre este pin y la salida de datos de la memoria DOUT. Esta resistencia sirve para evitar la pérdida de datos cuando se escribe simultáneamente por el pin EEDATA y el pin DOUT. Cuando se le envía a la EEPROM un comando con éxito, ésta pone en baja la señal DOUT. Por tanto, también es necesario usar una resistencia de 10 KΩ entre este pin y la alimentación para mantener en alta el valor de DOUT cuando no se ha introducido un comando válido o para cuando no hay ninguna memoria conectada.

La información contenida en la EEPROM incluye, entre otros, los valores del PID, VID, número de serie, Descripción del Producto, Valor Descriptor de Potencia, modo de comunicación, versión del protocolo USB y forma de alimentación (a través del bus o autónoma).

La EEPROM escogida fue el modelo 93LC66 de Microchip. Se trata de una memoria EEPROM (Electrically Erasable PROM) de 4K. La anchura de palabra puede seleccionarse entre 8 bits o 16 bits, dependiendo de si la patilla ORG está unida a tierra o a Vcc, respectivamente (como para el correcto funcionamiento del FT2232C es imprescindible que la memoria tenga anchura de palabra de 16 bits, será necesario conectar el pin ORG a la alimentación).

Figura 13: Patillaje de la EEPROM 93LC66



El encapsulado elegido fue el SMD (Surfaced Mounted Device) del tipo SOIC de 8 pines. Además de los pines de tierra (Vss - 5) y de alimentación (Vcc - 8) y del pin ORG para seleccionar la organización de memoria (ORG - 6), se dispone de un pin para seleccionar el chip (CS - 1), otro para una señal de reloj (CLK - 2) que sincronice la memoria con un dispositivo maestro (el FT2232C), la entrada de datos serie (DI - 3) y la salida de datos (DO - 4), sincronizadas con la señal de reloj.

Interfaz de Programación de Aplicaciones

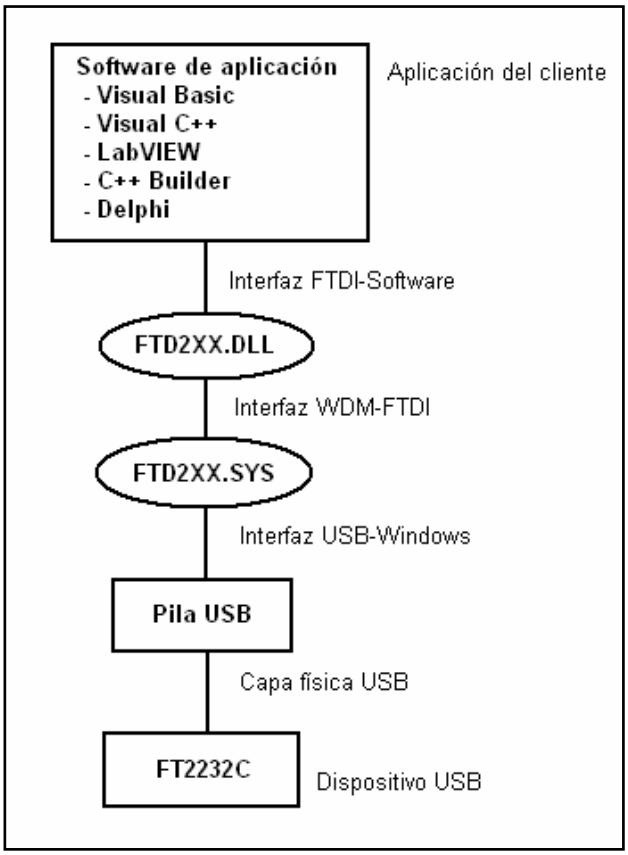
Existen dos alternativas a la hora de implementar el software para que los alumnos del Laboratorio de Tecnología de Computadores puedan cargar sus diseños en la FPGA. La primera es usar los drivers Virtual COM Port (VCP) típicos, con los que el dispositivo USB aparece como un puerto COM adicional. El software accedería al dispositivo USB de la misma forma en que lo haría con un puerto COM estándar. La segunda opción es usar los D2XX Direct Drivers ofrecidos por el fabricante FTDI.

Los D2XX Direct Drivers son una alternativa a los VCP que permiten la interacción entre el software y el controlador USB FT2232C usando una única DLL en lugar de un Virtual COM Port. A pesar de ser unos controladores más específicos (sólo sirven para los productos de la casa FTDI), su instalación es más rápida y sencilla que la de VCP. Tan sólo es necesario enchufar el dispositivo USB e instalar la biblioteca ftd2xx.dll. Otra ventaja de los drivers D2XX

es la presencia en la API de funciones específicas para la configuración del modo de comunicación del controlador USB y para el manejo de la memoria EEPROM asociada.

La arquitectura de los drivers D2XX consiste en un driver WDM de Windows que se comunica con el dispositivo a través de la pila USB y de una DLL que hace de interfaz entre el software de aplicación y el driver WDM. El software de aplicación puede estar escrito en Visual C++, C++ Builder, Delphi o VB, entre otros.

Figura 14: Arquitectura de la API D2XX



Entre las llamadas a la API dedicadas al manejo básico del controlador USB, las más importantes son aquellas que abren y cierran el dispositivo USB (FT_Open, FT_OpenEx y FT_Close), las que listan todos los dispositivos USB conectados al host (FT_ListDevices) y las que escriben y leen del dispositivo (FT_Write y FT_Read).

Seguidamente se detallan las funciones más usadas en el software de aplicación:

`FT_STATUS FT_OpenEx(PVOID arg1, DWORD flags, FT_HANDLE descriptor)`	
Cometido	Abre el dispositivo deseado y obtiene un descriptor del mismo.
Parámetros	`arg1` – Indica el número de serie, descripción o localización del dispositivo a abrir. `flags` – Determina si el dispositivo se abre por número de serie, por localización o por descripción, mediante los flags `FT_OPEN_BY_SERIAL_NUMBER`, `FT_OPEN_BY_LOCATION`, `FT_OPEN_BY_DESCRIPTION`, respectivamente. `descriptor` – Descriptor del dispositivo.
Valor de retorno	`FT_OK` si termina con éxito. Un código de error en caso contrario.

FT_STATUS FT_Close(FT_HANDLE descriptor)	
Cometido	Cierra el dispositivo con un descriptor dado.
Parámetros	descriptor – Descriptor del dispositivo.
Valor de retorno	FT_OK si termina con éxito. Un código de error en caso contrario.

FT_STATUS FT_ListDevices(PVOID arg1, PVOID arg2, DWORD flags)	
Cometido	Obtiene el número de dispositivos conectados e información relacionada.
Parámetros	arg1 – Depende del valor de flags. arg2 – Depende del valor de flags. flags – Si tiene el valor FT_LIST_NUMBER_ONLY, hace que la función guarde el número de dispositivos conectados en un DWORD apuntado por arg1. Si tiene los valores FT_LIST_BY_INDEX y FT_OPEN_BY_DESCRIPTION, almacena en la variable apuntada por arg2 la descripción del dispositivo cuyo índice se indica en arg1. Un uso similar se obtiene si se usan los flags FT_OPEN_BY_SERIAL_NUMBER o FT_OPEN_BY_LOCATION, guardándose en la posición de memoria apuntada por arg2 el valor del número de serie o del puerto del dispositivo, respectivamente.
Valor de retorno	FT_OK si termina con éxito. Un código de error en caso contrario.

FT_STATUS FT_Read(FT_HANDLE descriptor, LPVOID buffer, DWORD bytesALeer, LPWORD bytesLeidos)	
Cometido	Lee datos del dispositivo.
Parámetros	descriptor – Descriptor del dispositivo. buffer – Puntero al buffer que almacena los datos desde el dispositivo. bytesALeer – Número de bytes a leer del dispositivo. bytesLeidos – Puntero a un DWORD que recibe el número de bytes realmente leídos del dispositivo.
Valor de retorno	FT_OK si termina con éxito. FT_IO_ERROR en caso contrario.

FT_STATUS FT_Write(FT_HANDLE descriptor, LPVOID buffer, DWORD bytesAEscribir, LPWORD bytesEscritos)	
Cometido	Escribe datos en el dispositivo.
Parámetros	descriptor – Descriptor del dispositivo. buffer – Puntero al buffer que almacena los datos para ser escritos en el dispositivo. bytesAEscribir – Número de bytes a escribir en el dispositivo. bytesEscritos – Puntero a un DWORD que recibe el número de bytes realmente escritos en el dispositivo.
Valor de retorno	FT_OK si termina con éxito. Un código de error en caso contrario.

Entre las llamadas para la interfaz de programación de la EEPROM, destacan aquellas que se encargan de programarla (como FT_EE_Program o FT_EE_ProgramEx), las que se ocupan de leer datos de la memoria (FT_EE_Read, FT_EE_ReadEx o FT_ReadEE) o las que borran el contenido de la misma (FT_EraseEE).

A continuación se explican con más detalle las funciones más utilizadas en la aplicación desarrollada para la configuración de la EEPROM:

FT_STATUS FT_EraseEE(FT_HANDLE descriptor)	
Cometido	Borra el contenido de la EEPROM completamente.
Parámetros	`descriptor` – Descriptor del dispositivo.
Valor de retorno	`FT_OK` si termina con éxito. Un código de error en caso contrario.

FT_STATUS FT_EE_Program(FT_HANDLE descriptor, PFT_PROGRAM_DATA datos)	
Cometido	Programa la EEPROM.
Parámetros	`descriptor` – Descriptor del dispositivo. `datos` – Puntero a un *struct* con los datos a introducir en la memoria.
Valor de retorno	`FT_OK` si termina con éxito. Un código de error en caso contrario.

FT_STATUS FT_EE_Read(FT_HANDLE descriptor, PFT_PROGRAM_DATA datos)	
Cometido	Lee el contenido de la EEPROM.
Parámetros	`descriptor` – Descriptor del dispositivo. `datos` – Puntero a un *struct* para guardar los datos leídos de la memoria.
Valor de retorno	`FT_OK` si termina con éxito. Un código de error en caso contrario.

Por último, se ha usado una función de la extensión de la API que sirve para establecer el protocolo de comunicación usado por el chip (FT_SetBitMode):

FT_STATUS FT_SetBitMode(FT_HANDLE descriptor, UCHAR mask, UCHAR mode)	
Cometido	Establece el bit mode.
Parámetros	`descriptor` – Descriptor del dispositivo. `mask` – Establece qué pines son de salida (1) y cuáles son de entrada (0). `mode` – Valor del bit mode: 0 – Reset del bit mode. 1 – Bit Bang Mode Asíncrono. 2 – MPSSE. 3 – Bit Bang Mode Síncrono. 8 – Emulador de Bus de MCU como Host. 10 – Fast Opto-Isolated Serial Mode.
Valor de retorno	`FT_OK` si termina con éxito. Un código de error en caso contrario.

Paquetes software utilizados

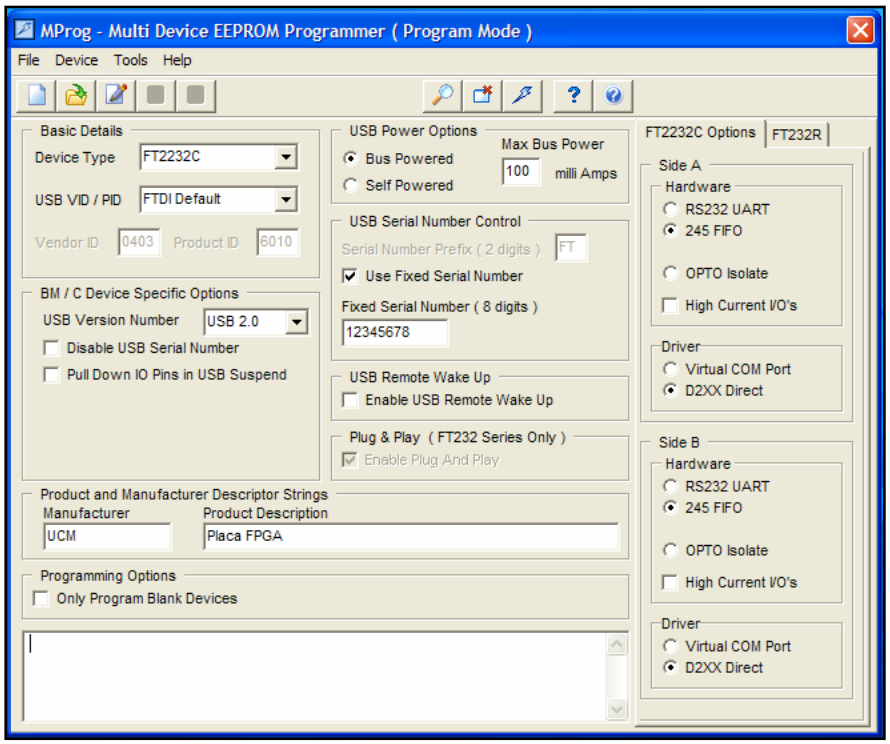
Para el desarrollo del software de comunicación con la FPGA a través del USB, se utilizó un entorno de programación muy popular (Visual Studio de Microsoft) y una aplicación específica para la programación de la EEPROM. Además fue necesario crear diseños de prueba con el paquete de Xilinx Foundations. Los esquemáticos y el diseño en PCB se realizaron con el software DXP 2004 de Altium.

MProg 2.8

MProg es una pequeña utilidad para programar la EEPROM usada por algunos dispositivos ofrecidos por el fabricante FTDI, entre los que se encuentra el controlador USB utilizado en el proyecto (el chip FT2232C).

Posee una interfaz muy intuitiva con la que se pueden modificar los datos programados en la EEPROM que determinan el comportamiento del controlador USB, tales como la interfaz (UART o FIFO), la alimentación (autónoma o a través del bus), los drivers de la API (D2XX o VCP), el número de serie, el identificador del fabricante y del producto, o la versión del protocolo USB usada. También tiene una opción para borrar todo el contenido de la memoria.

Figura 15: Entorno de MProg 2.8



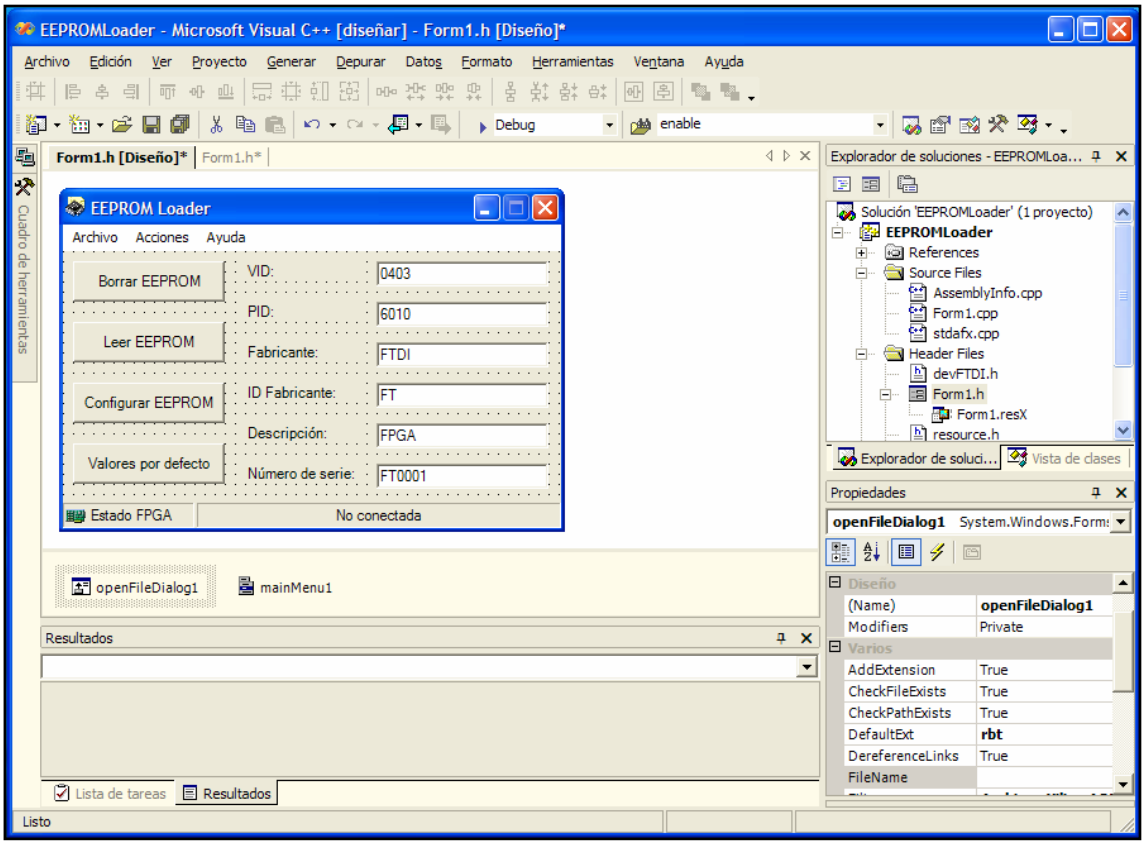
MProg fue utilizado básicamente durante el proceso de depuración de la aplicación encargada de configurar la EEPROM.

Microsoft Visual Studio .NET 2003

Visual Studio .NET es un conjunto de aplicaciones completo para la creación tanto de aplicaciones de escritorio como de aplicaciones Web y móviles. Permite el desarrollo de programas en varios lenguajes de programación, como Visual Basic, Visual C++, Visual J++, ASP o Visual C#. Todos estos lenguajes están integrados gracias a la presencia de la plataforma .NET. El Framework .NET es un entorno multilenguaje que además permite generar, implantar y ejecutar Servicios Web y aplicaciones XML.

Visual Studio tiene todas las funcionalidades típicas de cualquier entorno de desarrollo integrado, como un editor inteligente de código, exploradores de proyecto y de fuentes, asistentes para el diseño de interfaces gráficas y depurador avanzado.

Figura 16: Entorno de Visual Studio .NET 2003



El lenguaje utilizado para la implementación del software de comunicación fue Visual C++, por ser este el lenguaje con un mayor número de ejemplos de comunicación a través del chip FT2232C, y por poseer un alto grado de integración con la API D2XX y con las bibliotecas del sistema operativo. La práctica totalidad del proceso de desarrollo y depuración del software de comunicación se realizó desde Visual Studio.

A pesar de existir una versión más actual de este producto, Microsoft Visual Studio .NET 2005, se escogió la versión previa por ser ésta de la que se disponía en los laboratorios de la Facultad.

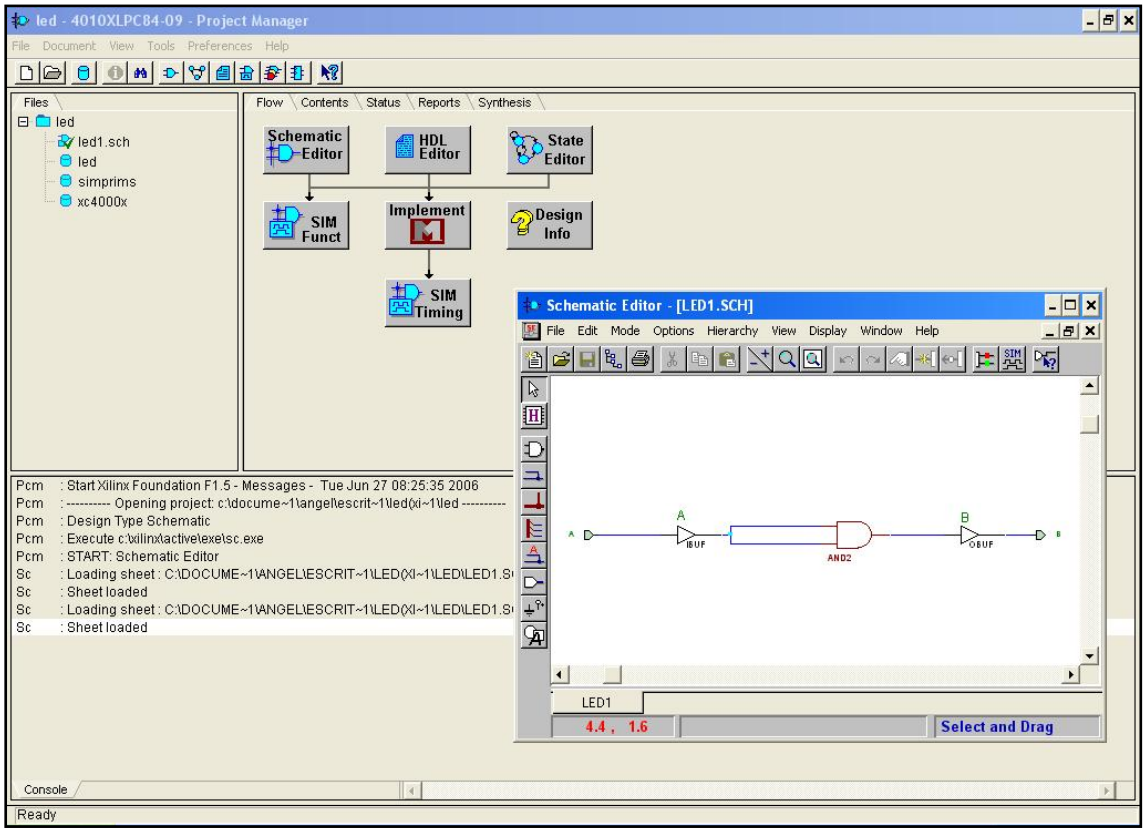
Xilinx Foundations F1.5

El sistema Xilinx Foundations F1.5 es una herramienta de desarrollo formada por un conjunto integrado de herramientas software y hardware para crear, simular e implementar diseños digitales en una FPGA o CPLD. Todas las herramientas usan una interfaz de usuario gráfica que permite usar todos los programas desde iconos, menús o barras de herramientas. También se dispone de ayuda en línea desde la mayoría de ventanas.

Todas las herramientas de Foundation Series involucradas en el proceso de diseño son administradas y supervisadas por el Project Manager, las cuales quedan integradas en un entorno unificado.

Este entorno incluye herramientas como un editor de esquemáticos, un editor de HDL, un editor de diagramas de estado, un simulador lógico, y herramientas de terceros.

Figura 17: Entorno de Xilinx Foundations F1.5

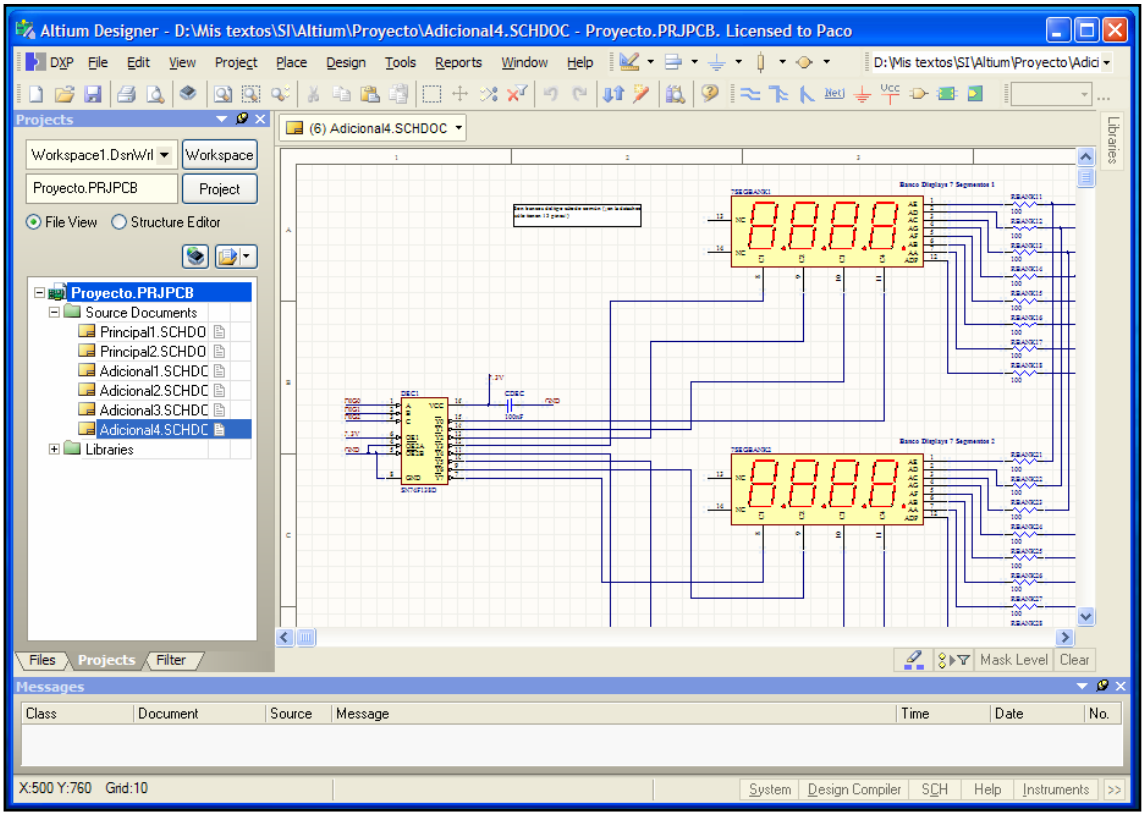


En el proyecto, este software se usó únicamente para la obtención de archivos .bit y .rbt con una configuración de la FPGA conteniendo un circuito básico de prueba. Hubo de usarse esta versión antigua del Xilinx Foundation debido a que las pruebas de la aplicación desarrollada se hicieron sobre una FPGA antigua.

Altium DXP 2004

Altium Designer es un completo entorno unificado de desarrollo de productos electrónicos que cubre todos los aspectos de cualquier proceso de diseño y fabricación de componentes. Entre sus muchas funcionalidades, se encuentran el diseño y captura de esquemáticos, el diseño físico de PCBs (Printed Circuit Board), el diseño de HW para FPGAs, la simulación de circuitos de señal mixta y el análisis de integridad de señal, entre otras.

Figura 18: Entorno de Altium DXP 2004



Altium DXP 2004 se ha utilizado en este proyecto para el diseño de la placa de prototipado basada en FPGA, obteniendo todos los esquemáticos del circuito, a partir de los cuales se realizará el rutado y obtención del floorplanning que permitirá en el futuro la implementación física del circuito en el Laboratorio de Circuitos Impresos.

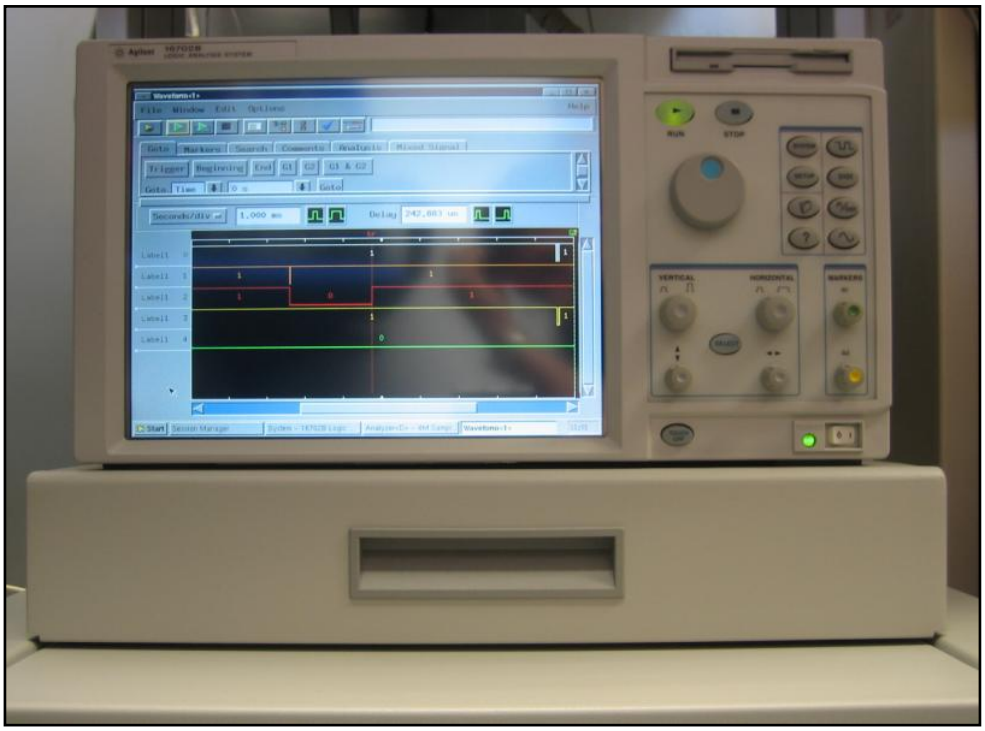
Dispositivos de medida

Principalmente, se utilizó un analizador lógico para la visualización y chequeo de las tramas enviadas desde el USB a la FPGA. También fue necesario usar un multímetro para comprobar niveles de señal y un soldador para añadir cables de depuración al prototipo.

Analizador lógico

Se hizo uso de un analizador lógico de la marca Agilent Technologies, modelo 16702B, que se muestra en la siguiente fotografía.

Figura 19: Analizador lógico



Un analizador lógico es un instrumento electrónico orientado a la verificación de circuitos digitales secuenciales. Es un dispositivo cuyo objetivo es visualizar un conjunto de valores digitales durante un periodo de tiempo de adquisición.

Por lo tanto el analizador lógico:

- Sólo adquiere muestras que tomen unos valores discretos.
- Adquiere varias muestras simultáneamente para poder observar un conjunto de líneas digitales (por ejemplo un bus).
- Las muestras pueden tomar diferentes valores a lo largo del tiempo de adquisición.
- Las muestras se almacenan en una memoria digital interna, llamada memoria de adquisición, para su posterior observación.

Un analizador lógico representa las señales de forma semejante a un osciloscopio: el eje horizontal representa el tiempo y el eje vertical el valor de la señal. Sin embargo, un osciloscopio representa señales analógicas que pueden tomar infinitos valores entre unos límites establecidos y que normalmente son periódicas. El número de señales a visualizar en un osciloscopio es reducido dependiendo del número de canales del equipo (de 1 hasta 4 normalmente). A diferencia del osciloscopio, que trata de representar las señales con gran resolución de voltaje y precisión temporal, los objetivos de los analizadores lógicos son los

siguientes:

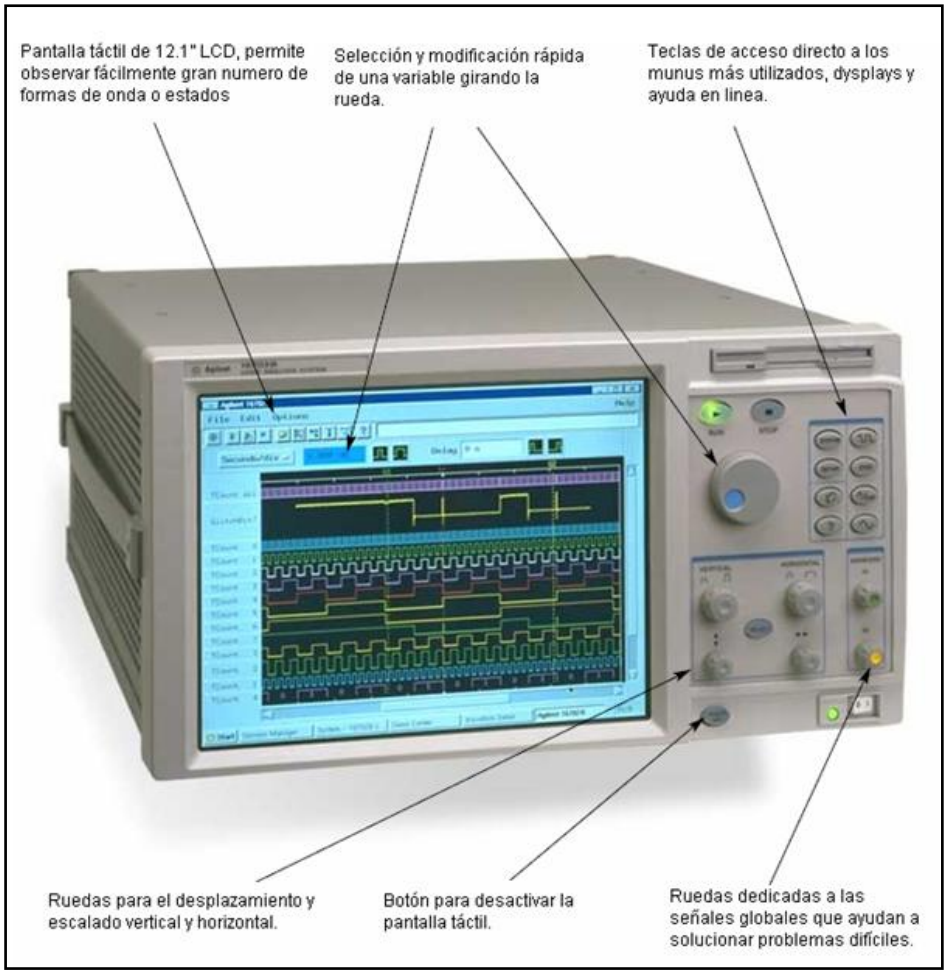
- Representar simultáneamente un gran número de señales (en general superior a 16).
- Visualizar las señales mediante el nivel lógico (“0”/“1”) que representan en el circuito y no mediante valores precisos de voltaje.
- Observar el estado de las señales entorno a la aparición en varias líneas de un determinado patrón de bits (condición de disparo o trigger).

Dado que el analizador lógico no observa señales periódicas y la memoria de adquisición es limitada, es necesario determinar el momento en que se desea realizar la adquisición. Esto se consigue mediante el establecimiento de una condición de disparo (trigger) que es la que determina cuando se comienza a guardar las muestras en la memoria de adquisición. La condición de disparo puede ser un patrón de bits determinado de las señales que se quieren visualizar o puede ser una señal de disparo externa. Cuando se utiliza una condición de disparo, el analizador lógico empieza a muestrear de forma continuada al recibir la orden de inicio y hasta que se produce la condición de disparo. Cuando se cumple la condición de disparo, las muestras se empiezan a guardar en la memoria (pre-trigger) o se guardan las últimas muestras (post-trigger). Al usuario se le muestran los datos almacenados en la memoria de adquisición que incluyen la condición de disparo.

Por ello, los analizadores lógicos resultan adecuados para observar relaciones temporales entre múltiples líneas de datos, esto los hace ideales para depurar los problemas hardware en sistemas digitales.

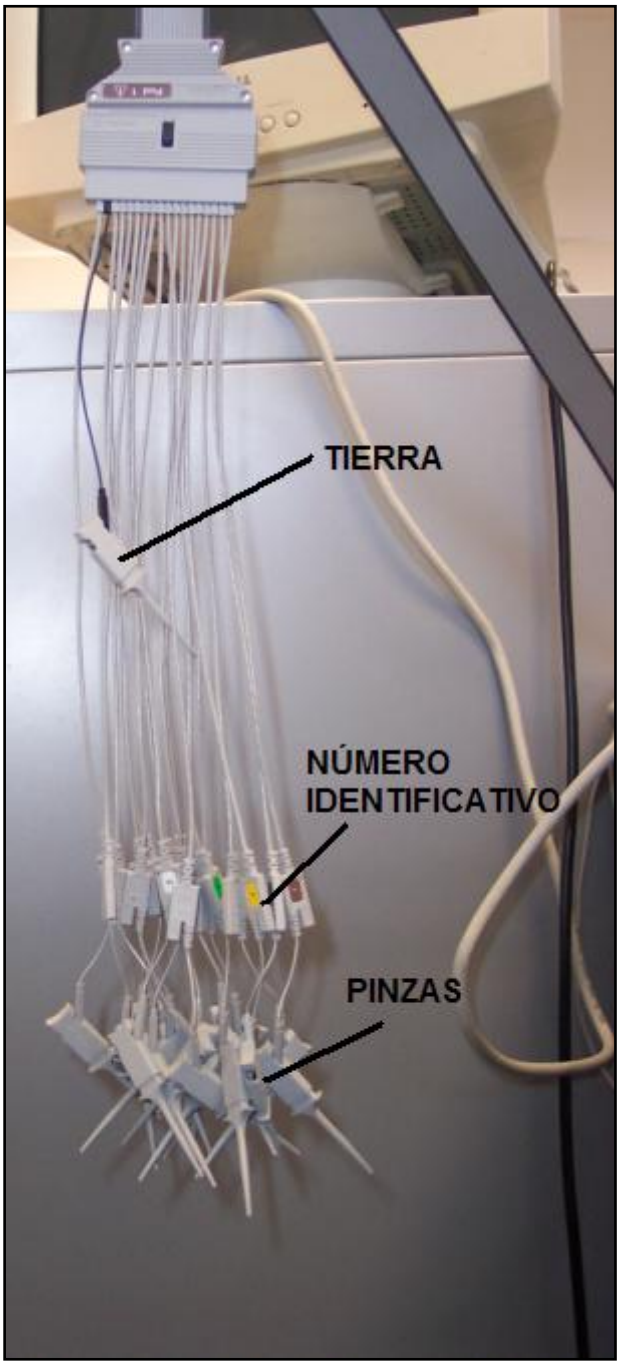
El analizador consta de varios elementos, los cuales se muestran en la siguiente imagen.

Figura 20: Descripción del analizador lógico



Para la captura de las señales dispone de unas canales, estos son los que se conectan al circuito digital. A continuación se muestra un bus con varios canales.

Figura 21: Bus con varios canales



Soldadores, Multímetros y otros

Se ha usado distintos dispositivos eléctricos para la depuración de los prototipos de trabajo: entre ellos, un multímetro y un soldador de estaño.

Un multímetro es un instrumento electrónico de medida que combina varias funciones en una sola unidad. Las más comunes son las de voltímetro (medida de la tensión eléctrica), amperímetro (medida de la intensidad eléctrica) y ohmetro (medida de resistencia eléctrica). A

veces, los multímetros también poseen un comprobador de continuidad, que emite un sonido cuando el circuito bajo prueba no está interrumpido.

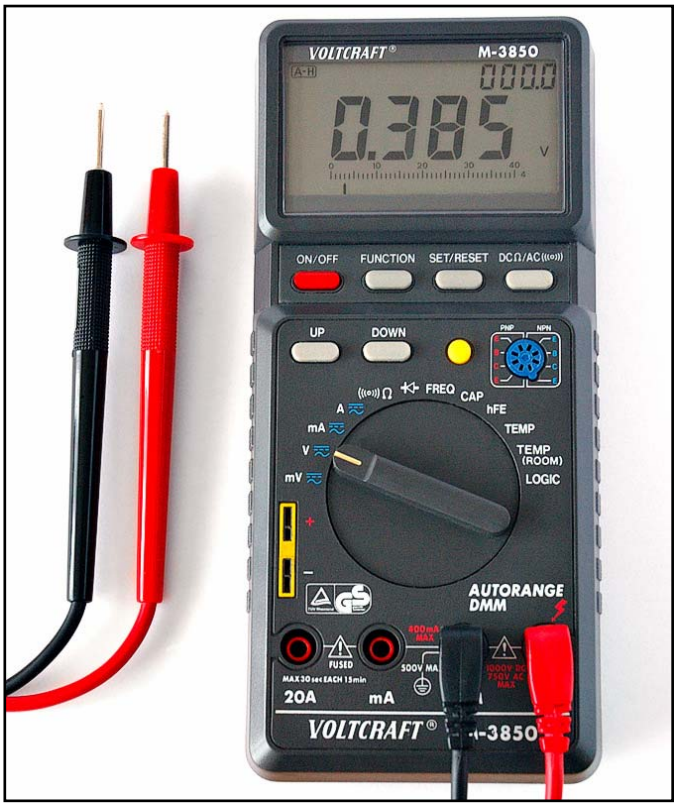


Figura 22: Multímetro

Para obtener uniones eléctricas de calidad cuando se monta un circuito eléctrico, es necesario recurrir a la soldadura. La soldadura ayuda a mantener a los componentes en su sitio, y asegura el contacto eléctrico. Los soldadores de lápiz tienen un mango aislante térmico, alineado con una resistencia eléctrica y una punta. La potencia ideal puede oscilar entre 20 y 40 W. La punta está formada por varias capas metálicas y debe siempre ser limpiada con cuidado para no deteriorarla. El material de aportación a emplear se trata de una aleación que contiene un 60% de estaño y un 40% de plomo, viene presentado en forma de carretes de hilo normalmente de 0,8 ó 1 mm de diámetro, y tiene en su alma una resina desoxidante que ayuda a limpiar los metales que se van a unir en el momento de realizarse la soldadura.

Figura 23: Soldador

PCB

Una placa de circuito impreso es una plancha de material rígido aislante, cubierta por unas pistas de cobre por una de sus caras o por ambas para servir como conexiones entre los distintos componentes que se montarán sobre ella. La materia prima consiste en esa plancha aislante, típicamente fibra de vidrio, cubierta completamente por una lámina de cobre. Dependiendo del tipo de placa, el cobre puede ir a su vez protegido por una capa de resina fotosensible.

Los circuitos impresos son robustos, baratos, y habitualmente de una fiabilidad elevada. Requieren de un esfuerzo mayor para el posicionamiento de los componentes, y tienen un coste inicial más alto que otras alternativas de montaje, como el montaje *punto a punto* (o *wire-wrap*), pero son mucho más baratos, rápidos y consistentes en producción en grandes cantidades.

La mayoría de los circuitos impresos están compuestos por entre una a dieciséis capas conductoras, separadas y soportadas por capas de material aislante (*sustrato*) laminadas (pegadas) entre sí. Las capas pueden conectarse a través de orificios, llamados vías.

El diseño del PCB se ha llevado a cabo en este proyecto mediante la utilidad Altium DXP 2004, que permite la captura de esquemáticos y la obtención del floorplanning a partir de éstos. Para la fabricación del prototipo, se dispone de un Laboratorio de Fabricación de Circuitos Impresos, con todas las herramientas y materiales necesarios.

Proceso de diseño

El proceso de diseño del PCB se puede dividir en varias etapas:

- Captura del esquemático: se genera un esquemático con todos los componentes que va a ser utilizados, especificando todas sus conexiones.
- Conversión del esquemático en lista de nodos (net list).
- Determinación de la posición de cada componente: basándose en los footprint de cada componente y colocándolos en un lugar apropiado.
- Rutado: fijar la ruta que seguirán las pistas hasta las patas de los footprints.

De estos pasos, el que presenta mayores problemas es el de rutado, ya que no se presta a una solución perfecta. Después de realizar un rutado automático, lo más habitual es que obtener una lista de nodos que no han podido ser rutados de manera adecuada y que por lo tanto deberán ser rutados manualmente.

Una vez finalizados todos estos pasos, el programa proporciona una mascara que deberá ser impresa sobre el fotolito. Como fotolito puede utilizarse papel cebolla o acetato, siendo preferible este ultimo.

Una vez finalizado el proceso de diseño, en el que además se puede incluir la serigrafía, se pasa al proceso de fabricación, aunque esta fase no ha sido posible llevarla a cabo.

Proceso de fabricación

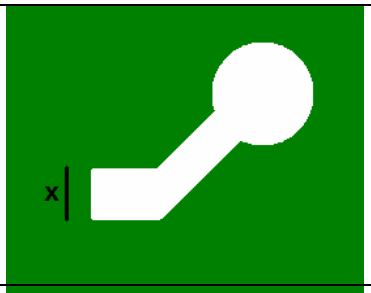
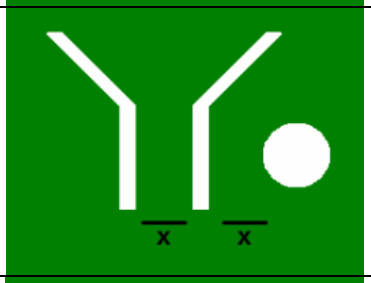
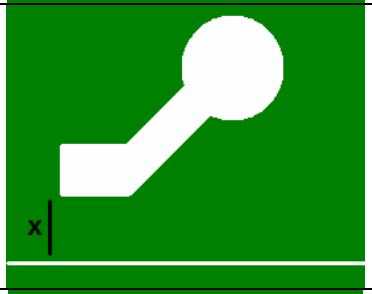
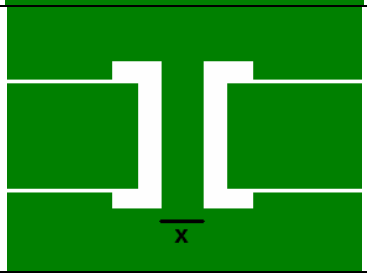
El proceso de fabricación consta de los siguientes pasos:

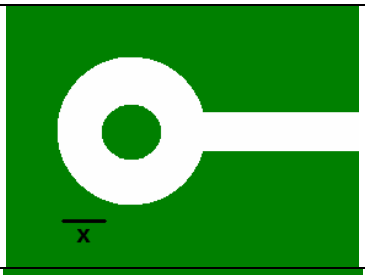
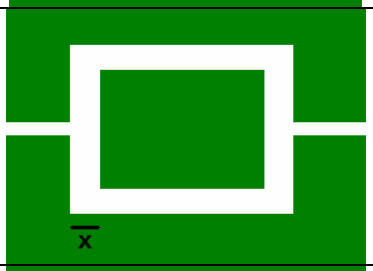
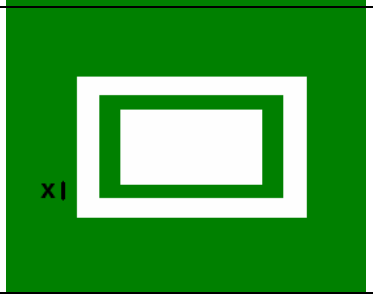
- Cortado: Se calcula aproximadamente el tamaño de placa que será necesario y se corta con una guillotina o una sierra.
- Insolado: Se retira la pegatina protectora y se introduce en la insoladora con la máscara. El tiempo de insolación suele oscilar entre 2 y 3 minutos.

- Revelado: Se introduce la placa insolada en la cubeta del revelador hasta que el cobre que debe ser eliminado quede completamente limpio.
- Atacado: Tras el revelado se lava la placa con agua, y se introduce en la cubeta del ácido hasta que desaparezca todo el cobre sobrante.
- Limpieza posterior: Tras el atacado se debe lavar y secar la placa, y volver a introducirla en la insoladora, aunque esta vez sin máscara. Una vez insolada de nuevo, se repetirá la etapa del revelado hasta que todo el cobre esté limpio de resina y se volverá a lavar con agua.
- Soldado: Se sueldan los componentes a la PCB.

Reglas de diseño

La maquinaria del fabricante con la que se va a realizar el PCB indica las siguientes restricciones de diseño:

Anchura de pistas y pads	0.3 mm / 12 mils	
Separación entre pistas y pads	0.3 mm / 12 mils	
Separación entre pistas y borde del PCB	0.25 mm / 10 mils	
Diámetro mínimo de orificio taladrado (a metalizar)	0.5 mm / 20 mils	

Corona (círculo metálico que rodea a los orificios)	0.22 mm / 8.6 mils ~0.3 mm / ~12 mils	
Grueso mínimo del marcaje de componentes	0.20 mm / 8 mils	
Separación mínima entre máscara y pads de cobre	0.15 mm / 6 mils	

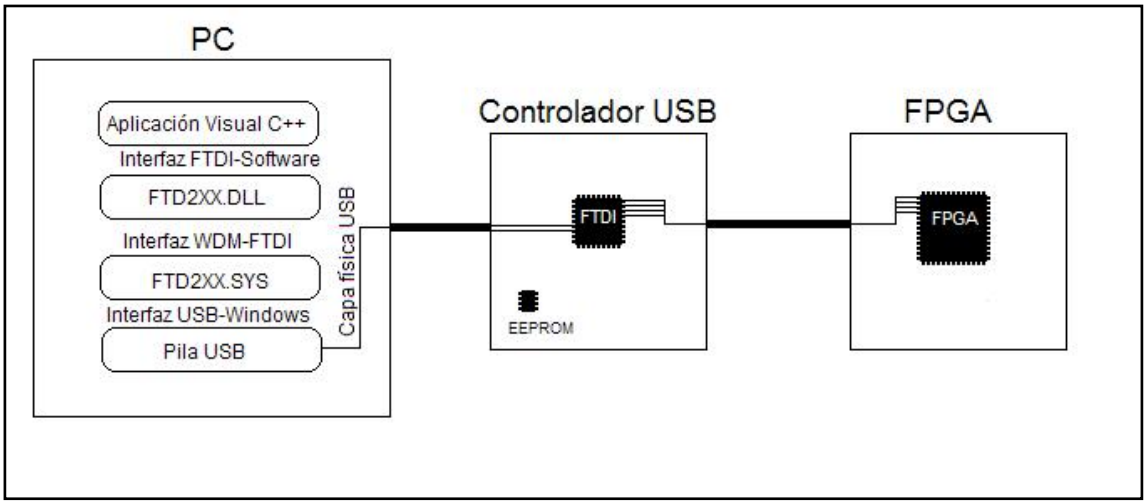
Además de estas restricciones, se seguirán las siguientes reglas de diseño:

- Debajo de un componente no se dibujará nada en ese plano (hay componentes que tienen la parte inferior metálica y podría haber cortocircuitos).
- Los agujeros para los tornillos del PCB serán de 3 mm de diámetro, dejando alrededor 6 mm para evitar cortocircuitos.
- Las pistas no forman ángulos rectos, en su lugar dibujarán codos de 45º.
- Las bifurcaciones en una pista sí serán ángulos rectos.
- Se usarán encapsulados del tipo SMD (Surface Mounted Device) para los diodos, resistencias, condensadores y otros elementos circuitales básicos.
- Los condensadores de desacoplo (que se colocan entre la alimentación y tierra y actúan como filtros paso-altas, eliminando así las fluctuaciones de alta frecuencia) se situarán en las cercanías de las líneas de alimentación que desacoplan.
- Todos los componentes importantes (FPGA, EEPROM, FT2232C, etc.) se sueldan en la cara superior del PCB.
- La circuitería adicional (resistencias, condensadores, etc.) se soldará, si es posible, en el dorso del circuito impreso.

DESARROLLO DEL PROYECTO

La primera fase del proyecto consistió en crear un prototipo de conexión entre el controlador USB y la FPGA. Este problema abarca, por un lado, un apartado de hardware, que incluye el diseño, fabricación y depuración de los prototipos del controlador USB y de la FPGA; y por otro lado, un apartado software, relacionado con el manejo del interfaz del chip USB para generar las señales que se enviarán a la FPGA.

Figura 24: **Diagrama PC-USB-FPGA**



Una vez encontrada una solución que fuera viable, se pudo comenzar con la última fase, consistente en el proceso de diseño de la placa basada en FPGA.

Prototipo Hardware

De los modos de programación disponibles, se eligió para este proyecto el modo Serial Slave, utilizando como generador de las señales de configuración de la FPGA el chip controlador USB, FT2232L de la compañía FTDI.

Inicialmente se hizo un prototipo (mostrado en la figura 25) sobre el que trabajar con los drivers del controlador USB. En esta primera fase se trabajó sin la FPGA. Se empleó únicamente dicho prototipo para comprobar la correcta generación de las señales que servirán de configuración para la FPGA. Para ver la forma de éstas, se utilizó el analizador lógico.

Una vez conseguida la generación correcta de las señales de configuración de la FPGA (una de las partes más difíciles del proyecto), se unió la placa de la figura 26 (provista de una FPGA) al prototipo anterior por medio de una serie de cables que partían desde la interfaz de E/S del controlador y llegaba a la interfaz de configuración de la FPGA.

Figura 25: **Fotografía del prototipo con el controlador USB**

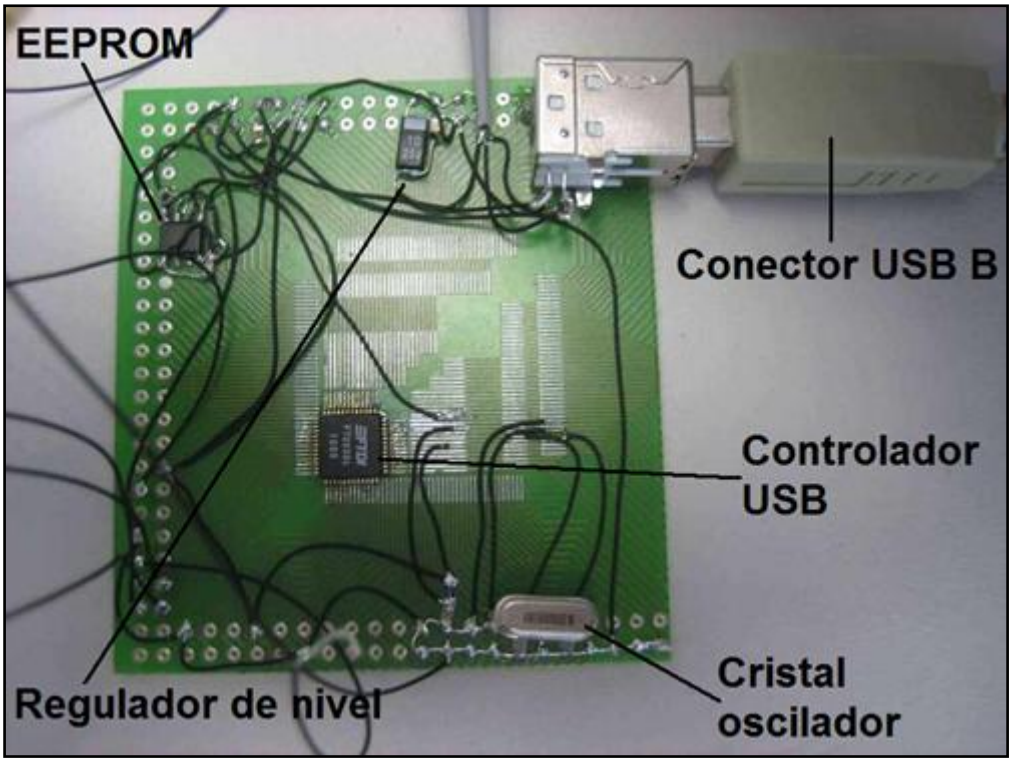
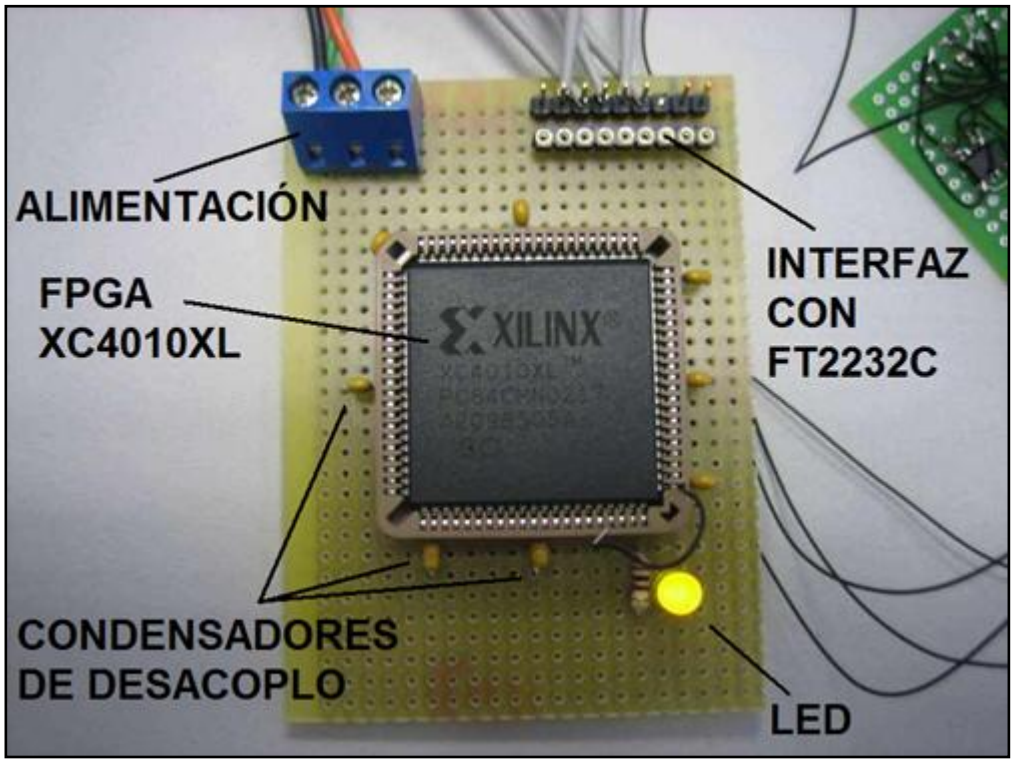


Figura 26: **Fotografía del prototipo con la FPGA**



Una vez unidas las dos placas, comenzaron las pruebas de funcionamiento, consistentes en la programación de la FPGA para poder encender un LED de usuario. Esto se explica más detenidamente en la sección “Ejemplo de funcionamiento”.

Conectividad

El chip USB genera un conjunto de señales (reloj, datos, etc.) que son enviadas a la FPGA. Dicho chip está conectado con la FPGA mediante 4 líneas, tal y como se muestra en la siguiente tabla:

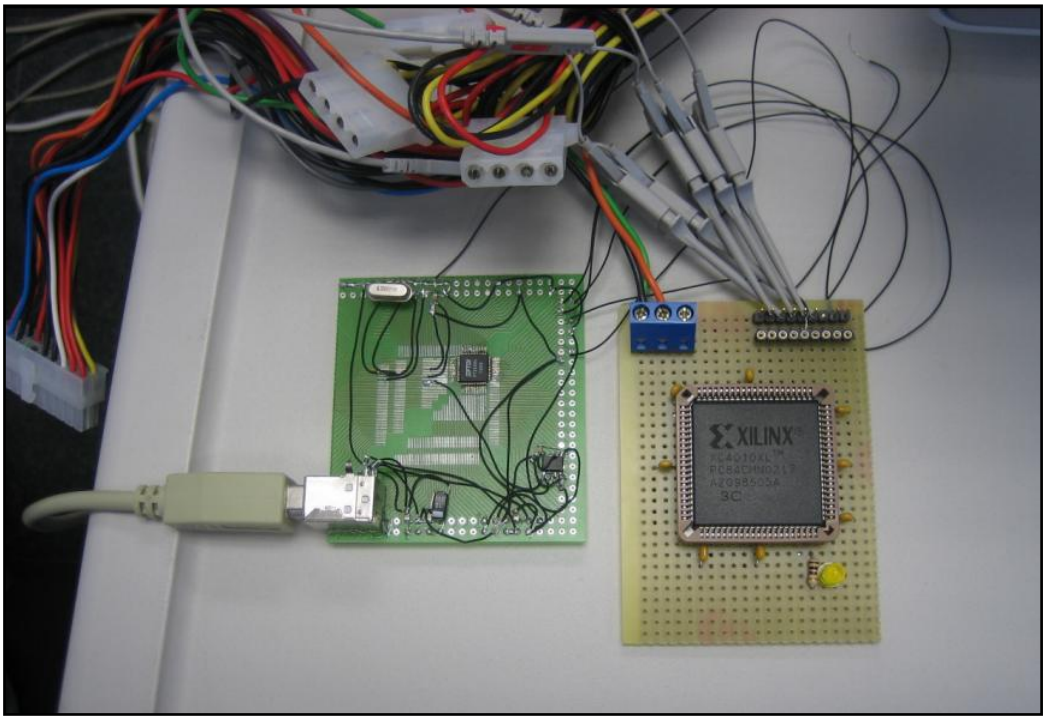
Señal	Patilla del chip FT2232L (Pin #)	Patilla (Pin #)
CCLK	ADBUS0 (24)	CCLK (37)
/PROGRAM	ADBUS1 (23)	/PROGRAM (69)
DIN	ADBUS3 (21)	DIN (39)
DONE	ADBUS4 (20)	DONE (72)

> **NOTA**: ADBUS2 está asociado en el software desarrollado a la señal /INIT y se podría haber usado para determinar cuándo la FPGA ha terminado la limpieza de la memoria de configuración (ésta pone /INIT en alta). Sin embargo, al ser necesaria una resistencia de pull-up en dicha patilla, esta condición no se comprueba y se usa un retardo lo suficientemente alto antes de pasar al envío de las tramas.

Así pues, se han empleado 4 pines de la interfaz de E/S del canal A del controlador USB. Las patillas que se usan del controlador USB no tienen por qué ser exclusivamente esas (24,24,21,20). Se podrían asignar otras en su lugar, realizando unas sencillas modificaciones en el software y cambiando las conexiones físicas.

El controlador USB tiene dos canales de entrada/salida, A y B, de ocho pines cada uno. Con la selección de patillas que hemos hecho, el canal B queda totalmente en desuso.

Figura 27: **Fotografía de la conexión del interfaz USB con el prototipo de FPGA**



Prototipo Software

Aplicaciones software implementadas

La implementación del software es una de las partes más importante del proyecto. Se han desarrollado dos aplicaciones:

- *EEPROM Loader*, cuya función es configurar la memoria EEPROM que acompaña al controlador USB.
- *Loader*, encargada de situar a la FPGA en el modo de configuración, enviarle el archivo con los datos de configuración y testear si la configuración se ha realizado correctamente.

La implementación de los métodos que aparecen en negrita a lo largo de esta sección se puede consultar en el apéndice del documento.

Características comunes a las dos aplicaciones

Las dos aplicaciones comparter, entre otras, las siguientes características:

- Funcionan bajo Windows XP.
- Requieren tener instalados los drivers del chip FT2232L.
- Utilizan la librería FTD2XX.lib que contiene las llamadas del interfaz D2XX, proporcionado por el fabricante del chip controlador USB.
- Ambos están implementados partiendo de un mismo esquema, por lo que tienen una estructura muy similar. Esta estructura es bastante sencilla y se basa en una gran clase principal que contiene todas las funciones que hacen llamadas al interfaz USB.

Antes de cada llamada a una función de la interfaz, ya sea para programar la EEPROM o para enviar las señales a la FPGA, es necesario abrir un puerto o canal del chip controlador. En la aplicación desarrollada, se usa siempre el canal A. Para realizar la configuración de la EEPROM no hay diferencia entre usar un canal u otro, pero no lo es para comunicarse con la FPGA, ya que ésta y el chip USB están conectados por el canal A.

El puerto se abre automáticamente al ejecutar el programa con una llamada a **`AbrirPuerto`** desde el método `Form1_Load`, siempre que la placa esté conectada correctamente al ordenador mediante el cable USB. El método `AbrirPuerto` realiza los siguientes pasos: primero hace una búsqueda de dispositivos con `FT_ListDevices`, si encuentra el dispositivo controlador USB, (encontrará como mínimo dos, ya que el chip FT2232L funciona como dos dispositivos USB, uno para cada canal) selecciona su canal A y a continuación lo abre con `FT_OpenEx`, guardándose el manejador del dispositivo en la variable global `handle`.

El puerto se cerrará automáticamente al cerrarse la aplicación. Esto se ha implementado usando una llamada al método **`CerrarPuerto`** (que hace una llamada a `FT_Close`) en `Form1_Closing`. Luego el usuario no debe preocuparse por el establecimiento y cierre de la conexión con el chip.

Todas las funciones que comienzan por el prefijo `FT_` pertenecen al interfaz USB D2XX y ya se han analizado en la aproximación tecnológica del controlador USB. Por último, es conveniente añadir que, tras cada llamada a una de estas funciones, siempre se comprueba si se han completado con éxito (devuelven el valor `FT_OK`) o no. Los valores que pueden devolver las funciones del interfaz se muestran en el siguiente extracto de código:

Extracto de código 1: Valores de retorno de las funciones del interfaz D2XX

```
enum {
        FT_OK,
        FT_INVALID_HANDLE,
        FT_DEVICE_NOT_FOUND,
        FT_DEVICE_NOT_OPENED,
        FT_IO_ERROR,
        FT_INSUFFICIENT_RESOURCES,
        FT_INVALID_PARAMETER,
        FT_INVALID_BAUD_RATE,

        FT_DEVICE_NOT_OPENED_FOR_ERASE,
        FT_DEVICE_NOT_OPENED_FOR_WRITE,
        FT_FAILED_TO_WRITE_DEVICE,
        FT_EEPROM_READ_FAILED,
        FT_EEPROM_WRITE_FAILED,
        FT_EEPROM_ERASE_FAILED,
        FT_EEPROM_NOT_PRESENT,
        FT_EEPROM_NOT_PROGRAMMED,
        FT_INVALID_ARGS,
        FT_NOT_SUPPORTED,
        FT_OTHER_ERROR
};
```

A continuación se explica de modo más detallado el funcionamiento de cada aplicación por separado.

EEPROM Loader

El núcleo de esta aplicación es el método **`ProgramarEEPROM`**, que utiliza una estructura de 58 campos, llamada `FT_PROGRAM_DATA` (cuyos campos vienen definidos por el fabricante), usada para programar una memoria EEPROM con la información necesaria para definir el modo en que posteriormente (mediante la aplicación *Loader*) se realizará la conexión con la FPGA. Esta estructura ha de ser pasada como parámetro a la función `FT_EE_PROGRAM` del interfaz USB.

La estructura `FT_PROGRAM_DATA` contiene, como principal información, el tipo de interfaz de comunicación que se usa en cada canal (Asynchronous Bit-Bang Mode, una variante del FIFO, en nuestro caso), la velocidad de transmisión e información personalizada (descripciones, número de serie, etc), que podrán ser definidas a mano rellenando unos campos habilitados para ello.

Extracto de código 2: Extracto de la estructura con información de programación de la EEPROM, FT_PROGRAM_DATA (El valor de los campos es el utilizado en la aplicación EEPROM Loader).

```
FT_PROGRAM_DATA ftData = {
      0x0,              // Header - must be 0x00000000
      0xffffffff,       // Header - must be 0xffffffff
      1,                // Header - FT_PROGRAM_DATA version
                        //          0 = original
                        //          1 = FT2232C extens
      0x0403,           // VID
      0x6010,           // PID
      "FTDI",           // Manufacturer string
```

```
      "FT",             // Manufacturer ID string
      "FPGA",           // description string
      "FT0001",         // serial number
      90,               // max current
      1,                // PNP
      0,                // Self Powered
      1,                // Remote wakeup
      //
      // Rev4 extensions
      //
      FALSE,            // Rev4
      FALSE,            // in endpoint type
      FALSE,            // out endpoint type
      FALSE,            // pulldown
      FALSE,            // serial number
      FALSE,            // USB Version select
      0,                // USB version
      //
      // FT2232C extensions
      //
      TRUE,             // non-zero if Rev5 chip, zero otherwise
      FALSE,            // TRUE if in endpoint is isochronous
      FALSE,            // TRUE if in endpoint is isochronous
      FALSE,            // TRUE if out endpoint is isochronous
      FALSE,            // TRUE if out endpoint is isochronous
      FALSE,            // TRUE if pull down enabled
      TRUE,             // TRUE if serial number to be used
      TRUE,             // TRUE if chip uses USBVersion
      0x0200,           // BCD (0x0200 => USB2)

      FALSE,            // TRUE if A interface is high current
      FALSE,            // TRUE if B interface is high current

      TRUE,             // TRUE if A interface is 245 FIFO
      FALSE,            // TRUE if A interface is 245 FIFO CPU target
      FALSE,            // TRUE if A interface is Fast serial
      FALSE,            // TRUE if A interface is to use VCP drivers

      TRUE,             // TRUE if B interface is 245 FIFO
      FALSE,            // TRUE if B interface is 245 FIFO CPU target
      FALSE,            // TRUE if B interface is Fast serial
      FALSE,            // TRUE if B interface is to use VCP drivers

      // Resto del archivo
      .....
};
```

Por defecto se programa la EEPROM con los campos Fabricante, ID del Fabricante, Descripción y Número de serie, con los valores que se observan en la figura anterior, pero esto se puede cambiar dentro de la aplicación; de hecho son los únicos campos que se permite variar para que la aplicación *Loader* pueda funcionar correctamente.

Los campos VID y PID deben ser los mostrados en la figura o de lo contrario los drivers suministrados por FTDI para su chip FT2232L no reconocerán el dispositivo.

Otro campo a destacar dentro de la sección `FT2232C extensions` (el chip FT2232C y el FT2232L es el mismo a diferencia del embalaje sin plomo que usa el nuestro), es que se ha programado el chip para que funcione como USB 2.0, ya que si se utiliza la configuración por defecto, la conexión sería USB 1.0.

El último aspecto importante es la programación como FIFO del interfaz del canal A, que es el que se utiliza para la conexión con la FPGA.

La aplicación *EEPROM Loader* además permite borrar la memoria EEPROM y leer sus datos de programación, si los tiene. Estas funciones son más simples que la programación. Para el borrado se ha implementado la función **`BorrarEEPROM`** que hace una llamada a `FT_EraseEE`, y para la lectura del contenido se codificó la función **`LeerEEPROM`**, que declara una variable `FT_PROGRAM_DATA`, y hace una llamada a `FT_EE_Read`, que vuelca la información actual de la EEPROM en dicha variable.

Loader

En base a los datos introducidos en la EEPROM con el anterior programa, se ha implementado un programa que envía las señales de configuración a la FPGA.

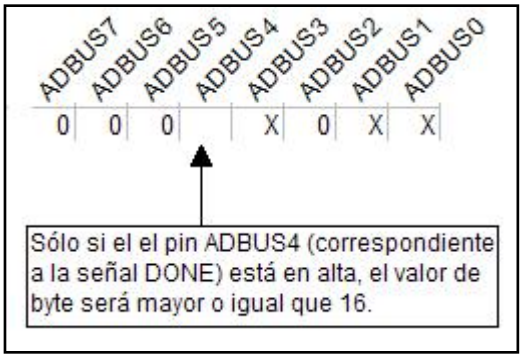
Omitiendo el proceso de selección y apertura del archivo RBT a enviar, lo cual es una práctica de programación común a muchos programas, estos son los pasos que realiza la aplicación *Loader*:

- Abre el canal A (al ejecutarse el programa, como ya se explicó antes).
- Establece el modo en que funciona el interfaz de dicho canal a *Asynchronous Bit Bang Mode*. Esto se hace con una llamada a `FT_SetBitMode` de la siguiente manera:

```
estado = FT_SetBitMode(handle,0xff,0x1);
```

  Donde `0xff`, que en binario corresponde a 1111 1111, establece los ocho pines del puerto como salida, (0 significa entrada y 1 salida). Y `0x1` establece el modo del interfaz a *Asynchronous Bit Bang Mode*.
- Tras esto, se envía una señal a la FPGA para que se ponga en modo de configuración (a la espera de la trama de configuración) con una llamada a **`ConfiguracionInicial`**. Esta señal consiste simplemente en aplicar un pulso en baja a la señal /PROGRAM, tras lo cual la FPGA comienza a limpiar su memoria de configuración.
- A continuación comienza el envío de las tramas del archivo RBT en el método **`SerializarArchivoYEnviar`**. La manera en que esto se hace se explica en la sección 'Coherencia de protocolos'.
- Para ver si la FPGA se ha configurado correctamente, se lee el estado de la señal DONE, cuyo valor en alta significa que la FPGA ha pasado a modo de operación de usuario. Esto se implementa en el método **`ComprobacionFinal`** en los siguientes pasos: primero se hace un borrado preventivo (con `FT_Purge`) del buffer de recepción del controlador USB; a continuación se lee 1 byte (8 bits correspondientes a las ocho patillas del canal A) y en este byte se comprueba si su valor es mayor o igual que 2^4 =16 (ya que el pin DONE entra por ADBUS4 y el resto de patillas desde ADBUS5 a ADBUS7 están al aire, luego devuelven cero). Si lo es, significa que la señal DONE está en alta. Esto se explica gráficamente en la figura 28.

Figura 28: **Modo de comprobación del estado de la señal DONE**



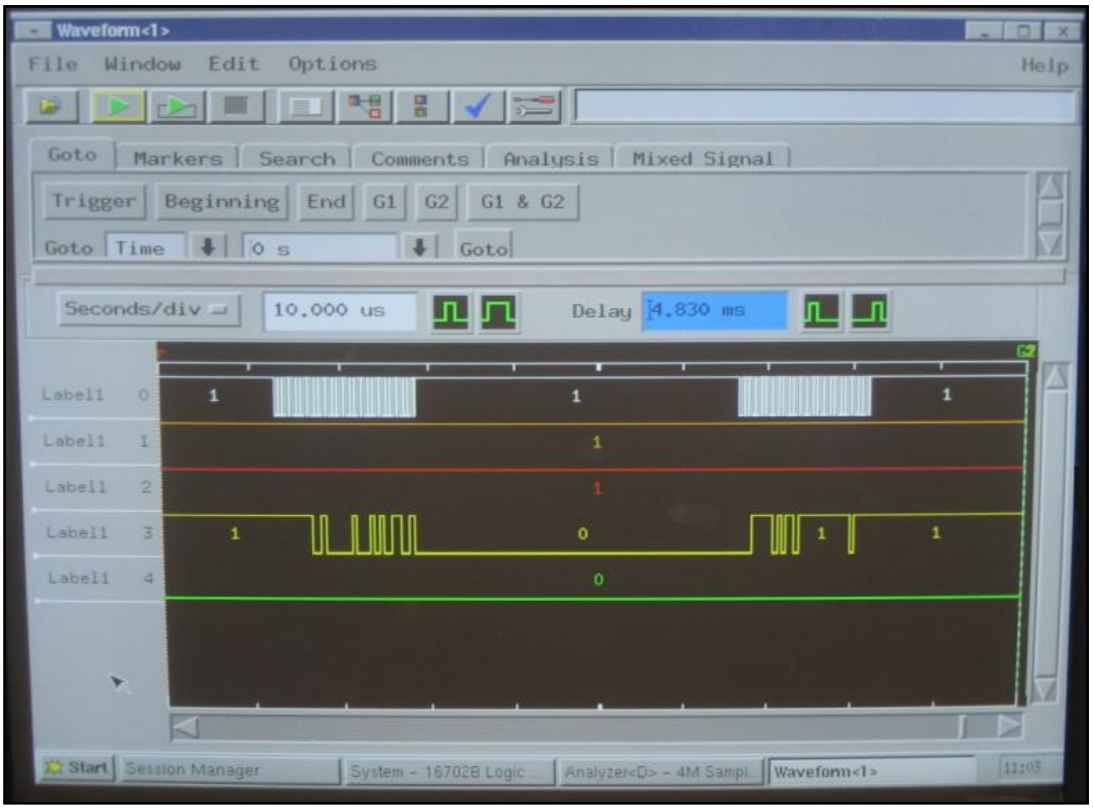
Los tres últimos pasos completan la configuración de la FPGA y se realizan dentro de un método llamado **`CargarFPGA`**, que es ejecutado cuando se presiona el botón correspondiente de la aplicación.

Tiempos

En nuestra aplicación utilizamos buffers de 128 bytes, que es el tamaño del buffer de transmisión del controlador FT2232L.

El protocolo USB como máximo envía paquetes con 64 bytes de información útil, luego al enviar nuestros buffers de 128 bytes al controlador USB (con `FT_WRITE`), éste se divide en dos paquetes de 64 bytes, comenzando uno, cierto tiempo, del orden de microsegundos (54 µs en nuestro prototipo -figura 29-).

Figura 29: **Envío de 128 bytes a través del chip USB**



La transmisión de cada paquete USB (64 bytes) a través de los pines dura 16 µs. Esto es debido a los dos siguientes motivos:

- En el protocolo *Asynchronous Bit-Bang Mode* cualquier dato escrito en el buffer de transmisión irá saliendo automáticamente por los pines con cada periodo del reloj.
- La frecuencia de reloj que nos da el controlador USB es de 4 Mhz (luego un ciclo dura 0'25 µs).

Luego los 64 bytes tardarán en enviarse 64 x 0'25 = 16 µs.

Coherencia de protocolos

El proyecto esta orientado a la FPGA XC2S100-5 TQ144C, pero las pruebas se han hecho en una de la familia XC4000XL, la XC4010XL, más antigua y, por lo tanto, más lenta. En esta sección se hablará de ambas FPGAs.

Se describirá inicialmente la forma en que se envía la trama de la configuración inicial, y a continuación, cómo se cumplen las restricciones necesarias durante el envío de las tramas de configuración, función llevada a cabo por el método `SerializarArchivoYEnviar`.

Dos factores previos a tener en cuenta son:

- Se envían las tramas con la función `FT_WRITE`, utilizando un buffer de 128 bytes, aprovechando así al máximo el buffer de transmisión del controlador USB.
- Cada byte contiene un bit para la señal de reloj, un bit para el /PROGRAM (que siempre vale uno salvo en la configuración inicial) y un bit para el DIN. El resto de bits son sobrantes y da igual su valor (en el programa siempre cero) ya que no llegan a la FPGA.

Trama de configuración inicial

El protocolo para poner a la FPGA en modo de configuración se basa simplemente en aplicar un pulso en baja a la señal /PROGRAM y asegurarse de que la FPGA haya terminado la limpieza de la memoria de configuración.

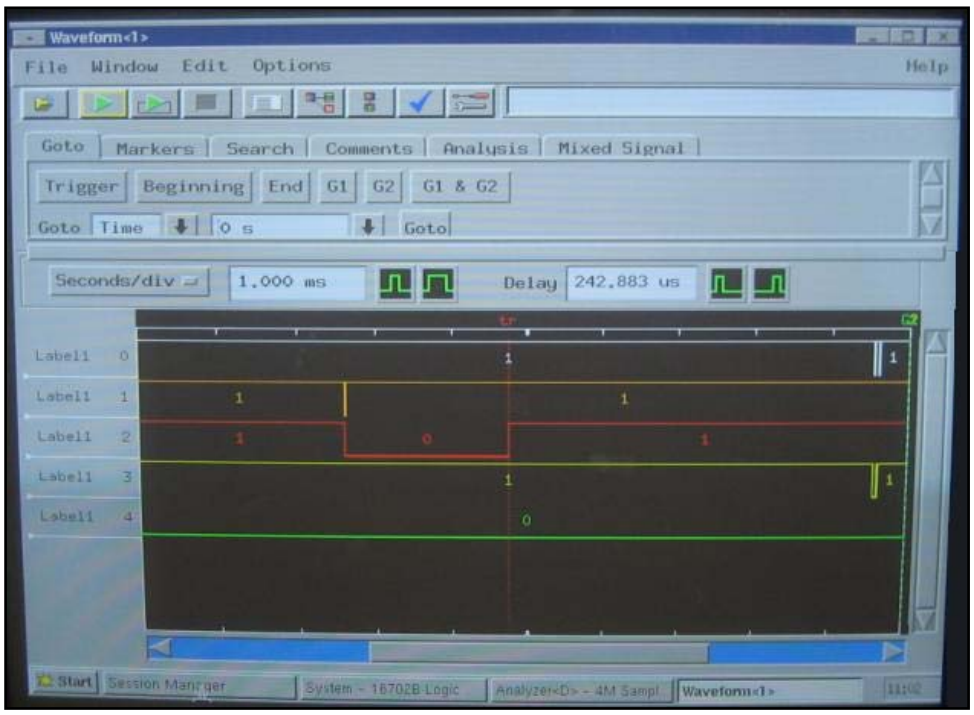
La documentación de la Spartan-II recomienda no mantener /PROGRAM en baja por más de 500 µs. Esto se cumple, pues el pulso generado dura 16 µs.

La memoria de configuración tarda en limpiarse, según el data-sheet de la Familia Spartan-II, un máximo de 2 ms. Pero en la XC4010XL, se ha comprobado empíricamente que el tiempo es mayor, aproximadamente 2.2 ms. Además, se debe esperar otros 4 µs después de que la señal /INIT se ponga en alta antes de comenzar a enviar las tramas de configuración. Si se asegura en la aplicación *Loader*, que se espera el tiempo necesario antes de enviar las tramas en la FPGA más antigua, el programa también servirá para la FPGA nueva.

Esto se cumple asegurando mediante software un retardo de 3 ms entre el pulso de /PROGRAM y el primer pulso del CCLK. Con esto concluye el protocolo para poner a la FPGA en modo de configuración.

Figura 30: **Analizador lógico durante la inicialización de la fase de configuración**

Señales en orden descendiente: CCLK: Blanco,
/PROGRAM: Naranja,
/INIT: Rojo,
DIN: Amarillo,
DONE: Verde.



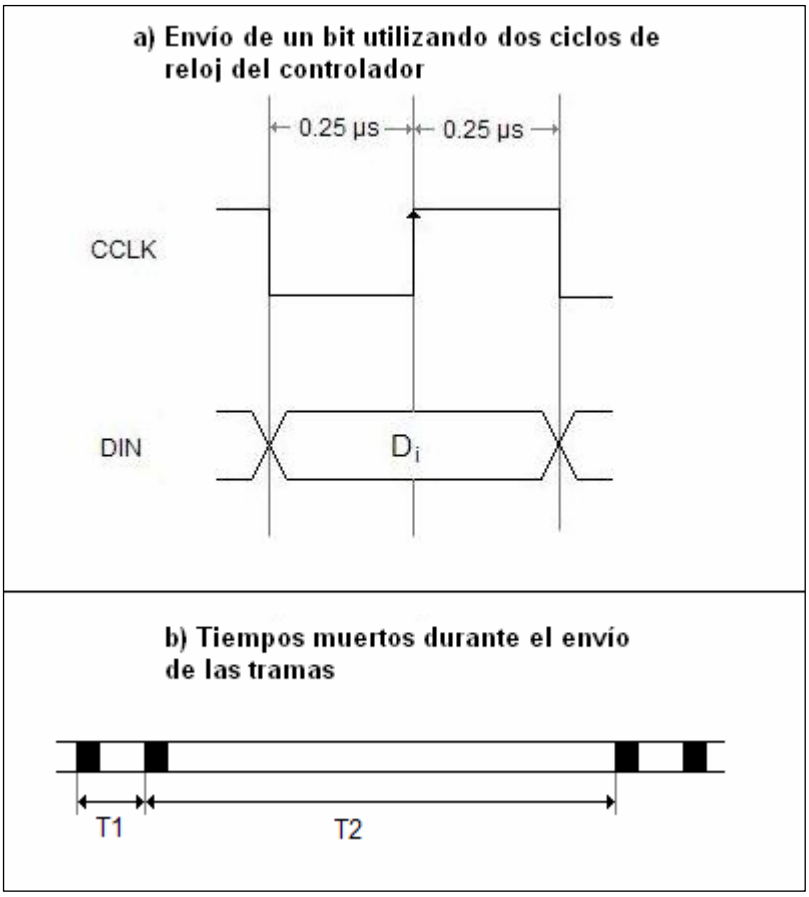
Envío de tramas de configuración

El envío de las tramas de configuración, implementado por la función `SerializarArchivoYEnviar`, tiene como principal dificultad las restricciones de reloj de la FPGA. Éstas, con las del modo Slave Serial, son:

- Para que la FPGA pueda leer un dato, éste tiene que estar en la patilla DIN un tiempo antes del flanco de subida de reloj; a este tiempo se les llama DIN hold y debe ser como mínimo de 5 ns.
- Otra restricción de la FPGA es que el tiempo 'High time', es decir el tiempo que el reloj esta en alta, debe ser mayor de 5 ns. Esta restricción de 5 ns también debe cumplirse para el tiempo 'Low time'. Y la suma de los dos, es decir, el ciclo de reloj, debe ser de algo superior a 15 ns (= 0'015 µs), ya que la frecuencia máxima que soporta es de 66 MHz.

Para enviar el bitstream a la FPGA, se lee el RTB de 64 en 64 bytes (recordando que un byte del RTB guarda un bit del bitstream). Pero, para enviarlo a la FPGA, cada bit debe "tener asociado" un flanco de subida del reloj. Esto se hace utilizando 2 bytes del buffer por cada byte leído del RTB, uno con el bit de reloj en baja y otro con dicho bit en alta. Luego para enviar 64 dígitos del bitstream se necesita un buffer 128 bytes. Esto se ve en la figura 31a.

Figura 31: **Modo de las señales generadas por nuestra aplicación**



Luego las dos restricciones anteriormente comentadas se respetan:

- El tiempo de ciclo es de 0'5 µs, luego la frecuencia que le llega la FPGA es de 2 MHz, mucho menor que los 66 MHz máximos permitidos.
- También respetamos la restricción del DIN hold mínimo de 5 ns, ya que el visualizado es de 0'25 µs.

Por último, es importante destacar que hay tiempos muertos en la aplicación, es decir, momentos durante el envío en los que no se envía nada (CCLK y DIN permanecen constantes). Estos tiempos tienen dos causas:

- Para comprender el tiempo muerto T1 que se muestra en la figura 31b, hay que tener en cuenta que se envían 128 bytes con cada llamada a FT_WRITE, pero que el paquete USB acepta como máximo 64 bytes, luego los datos llegarán en divididos en dos paquetes. Este retardo es del orden de microsegundos, 54 µs en las pruebas realizadas.
- El tiempo muerto T2 es debido a que se serializan los datos del RBT y se envían, y así sucesivamente. El retardo (del orden de milisegundos, 2 ms en el prototipo) depende de la velocidad de la computadora y de otros factores (sistema operativo, cantidad de programas en ejecución en ese instante…) entre cada envío de bytes.

Pero esto no es problema, ya que las FPGAs utilizadas no ponen ninguna limitación de frecuencia mínima.

Las restricciones se cumplen, luego el envío del bitstream es correcto, y por lo tanto al finalizar el bitstream la señal DONE pasa a alta (figura 33), indicando que la FPGA esta programada y que se pasa al modo de operación de usuario.

Figura 32: Analizador lógico durante el envío del bitstream

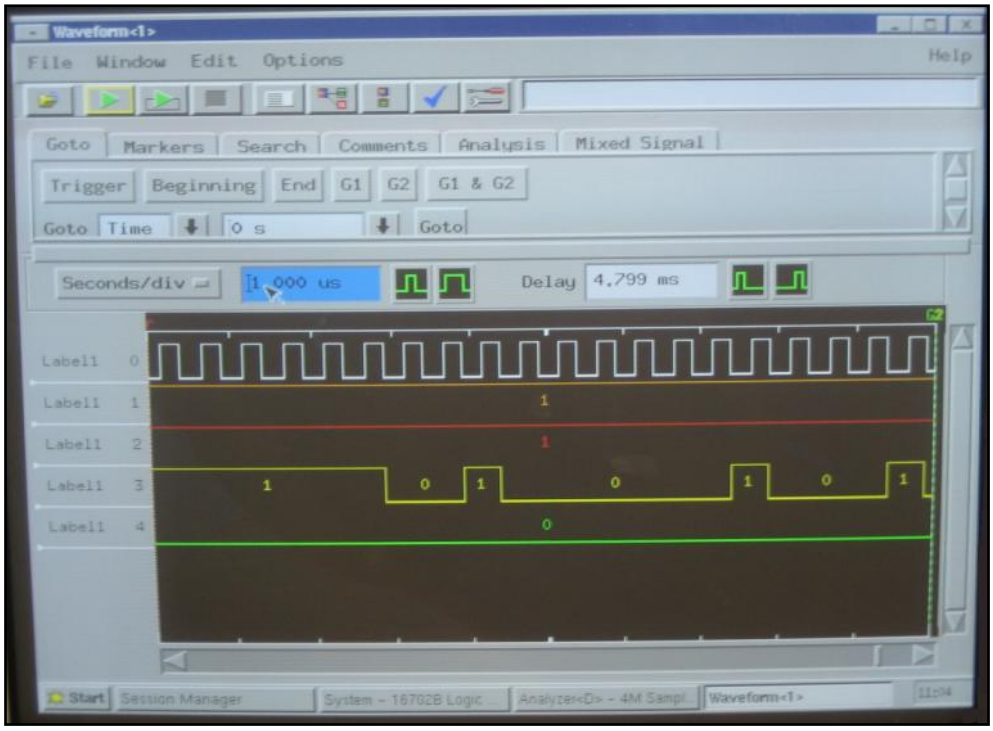
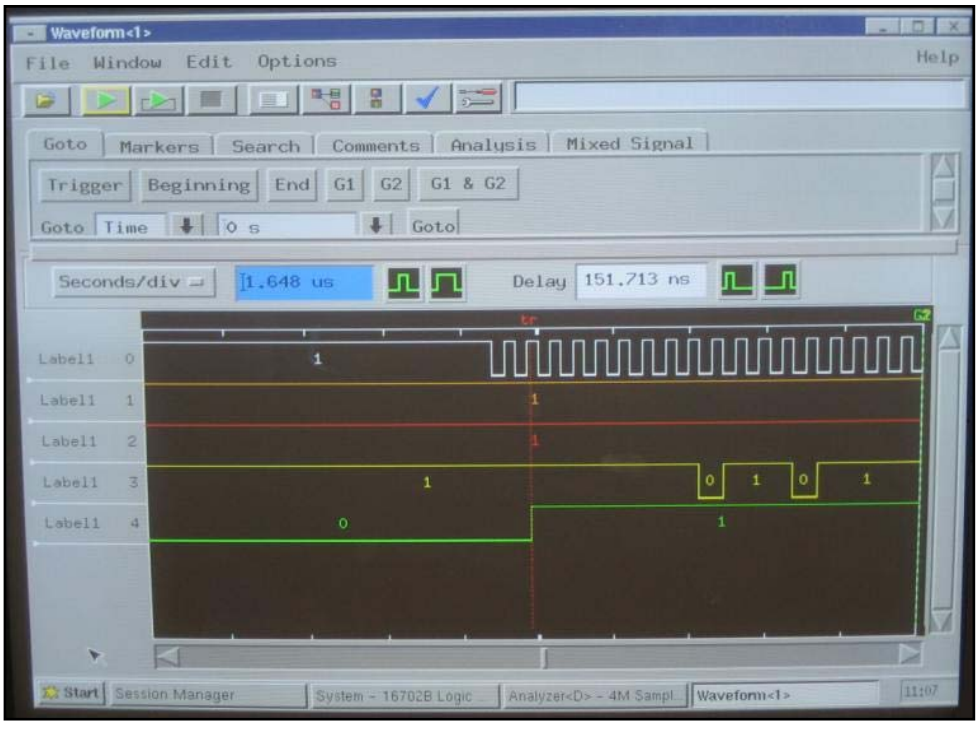


Figura 33: Puesta a uno de la señal DONE tras finalizar el envío de las tramas



Manual de las aplicaciones

Para utilizar los programas desarrollados para la configuración del chip controlador de USB y carga de la FPGA, primero es necesario cargar en el ordenador los drivers necesarios para su funcionamiento. Estos drivers son proporcionados junto con dichos programas.

El proceso para cargar los drivers es el mismo que para cualquier otro dispositivo que se conecta mediante el puerto USB por primera vez. Esto es, primero se conecta el dispositivo al puerto USB, a continuación el sistema operativo informará de que se ha conectado un nuevo dispositivo y, puesto que es la primera vez que se ha conectado, pedirá que el usuario le indique cuáles son sus drivers correspondientes. Se elige la opción de cargar el fichero y seleccionamos el fichero ftd2xx.dll que ha sido proporcionado.

Una vez cargados los drivers ya se pueden utilizar los programas de configuración y carga sin necesidad de añadir ningún otro fichero o librería.

Como se ha mencionado anteriormente, se proporcionan dos programas ejecutables, cada uno con una finalidad bien distinta y dirigida a dos tipos de usuarios distintos. El programa de configuración del chip controlador de USB, esta diseñado especialmente para el personal técnico encargado de los laboratorios, mientras que el programa de carga de archivos binarios a la FPGA, esta diseñado para que sea utilizado por los alumnos

En los puntos siguientes se detalla como utilizar cada uno de estos programas.

Manual del programa de configuración

El programa diseñado para el personal técnico de los laboratorios es EEPROM Loader. Este programa configura de manera apropiada el chip controlador de USB, que es el mecanismo mediante el cual se va a establecer la comunicación entre la placa FPGA y el ordenador. Esta configuración no es necesaria realizarla siempre, puesto que el chip controlador de USB dispone de una memoria EEPROM donde almacenar dicha configuración.

Al ejecutar el programa, se muestra una ventana con el siguiente aspecto.

Figura 34: EEPROM Loader

El programa consta de los cuatro botones siguientes: Borrar EEPROM, Leer EEPROM, Configurar EEPROM y Valores por defecto.

También tiene seis campos donde se recoge la información relevante que es leída o que va a ser cargada en la memoria EEPROM.

Por último se muestra el estado en que se encuentra la conexión con la FPGA.

A continuación se describe con más detalle la función de cada uno de los elementos que nos muestra el programa:

- **Botón Borrar EEPROM:** La función de este botón es la de borrar el contenido de la memoria EEPROM del chip controlador USB dejándola completamente vacía; de esta manera el chip utiliza la configuración que le viene cargada de fabrica, que se encuentra en el propio chip. Esta función de borrado de la EEPROM, se puede llevar a cabo también mediante el comando Borrar EEPROM del menú acciones.

- **Botón Leer EEPROM:** La función de este botón es leer la información contenida en la memoria EEPROM asociada al chip controlador de USB y mostrar la información relevante en los distintos campos que se muestran en el programa. Esta función de lectura de la EEPROM se puede llevar a cabo también mediante el comando Leer EEPROM del menú acciones.

- **Botón Configurar EEPROM:** La función de este botón es cargar la configuración del chip controlador de USB en la memoria EEPROM que tiene asociada. De esta información que se carga en la memoria EEPROM, únicamente se puede modificar la referente a los 4 campos editables que se muestran en el programa, ya que los otros dos campos no editables y el resto de la información cargada no debe ser modificada por el usuario de este programa. Esta función de configuración de la EEPROM se puede llevar a cabo también mediante el comando Configurar EEPROM del menú acciones.

- **Botón Valores por defecto:** La función de este botón es rellenar los campos editables con la información que tenía el chip controlador de USB originalmente.

- **Campos VID y PID:** Se corresponden con la información referida al identificador del vendedor y al identificador del producto respectivamente. Estos valores no son modificables, puesto que si fueran modificados, al conectar el dispositivo al ordenador, éste no lo reconocería como un chip controlador de USB, sino como otro dispositivo, y por tanto no se podría realizar ninguna acción sobre él.

- **Campos Fabricante e ID Fabricante:** contiene el nombre del fabricante y su identificador respectivamente. Esta información se puede modificar, puesto que no influye en el funcionamiento del chip controlador de USB.

- **Campos Descripción y Número de serie:** Contienen una descripción del dispositivo y su número de serie. Esta descripción puede ser modificada puesto que no es relevante para el funcionamiento del chip controlador de USB, sólo es información para el usuario.

- **Estado FPGA:** Informa del estado en que se encuentra la conexión con la FPGA, conectada o no conectada. Para poder utilizar el programa correctamente, éste debe indicar que la FPGA está conectada; de no ser así, se mostrará un mensaje informando de ello. El estado de conexión se puede actualizar mediante el comando Actualizar estado FPGA del menú acciones o pulsando sobre el icono de estado.

Los pasos a seguir para una configuración correcta del chip controlador de USB son los siguientes:

- Comprobar que el estado de la FPGA sea conectado.

Figura 35: EEPROM Loader conexión establecida

- Borrado de la memoria EEPROM.
- Rellenar los campos editables con la información deseada.

Figura 36: EEPROM Loader campos modificados

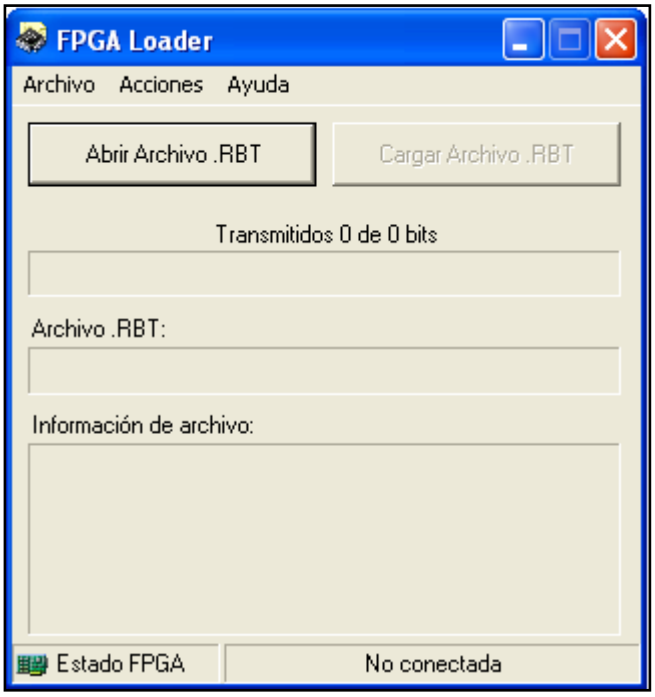
- Configurar EEPROM.
- Por ultimo, comprobar que se ha cargado la información haciendo una lectura de la memoria EEPROM.

Manual del programa de carga

El programa diseñado para que sea utilizado por los alumnos para cargar sus ficheros binarios a la FPGA es Loader.exe. El fin de este programa es que los alumnos puedan cargar sus diseños desarrollados con Xilinx Foundations en una FPGA mediante el envío a través del puesto USB del fichero binario *.RBT generado con dicha herramienta.

Si ejecutamos este programa se muestra una ventana con el siguiente aspecto.

Figura 37: FPGA Loader



El programa consta de dos botones Abrir Archivo .RBT y Cargar Archivo .RBT.

También se muestran varios campos de información como son Transmitidos, Archivo .RBT e Información de archivo.

Por ultimo, se muestra el estado en que se encuentra la conexión con la FPGA.

Se comenta a continuación la función de cada uno de los elementos que nos muestra el programa con más detalle.

- **Botón Abrir Archivo .RBT:** La función de este botón es la apertura del fichero .RBT, que ha generado la herramienta Xilinx Foundations, con el diseño del alumno. Para ello, al pulsar sobre el botón aparecerá una ventana donde se podrá elegir el fichero .RBT que se desea abrir. Esta función de apertura del fichero .RBT también se puede llevar a cabo mediante el comando Abrir archivo .RBT del menú Archivo.

- **Botón Cargar Archivo .RBT:** La función de este botón es la de enviar a través del puerto USB la información contenida en el fichero .RBT, para ser cargada en la FPGA. Una vez finalizado el proceso de envío de información a la FPGA, se muestra un mensaje indicando si la carga del programa ha sido correcta o no. Esta función de carga del fichero .RBT en la FPGA también se puede llevar a cabo mediante el comando Cargar Archivo .RBT del menú Acciones.

- **El campo de información Transmitidos:** Muestra los bits transmitidos durante el proceso de carga del fichero .RBT; también se muestra una barra de progreso de la carga de dicho fichero.

- **El campo de información Archivo .RBT:** Muestra la ruta de directorio donde se encuentra el fichero abierto actualmente. Así el alumno puede comprobar en todo momento qué fichero se ha abierto para ser cargado en la FPGA.

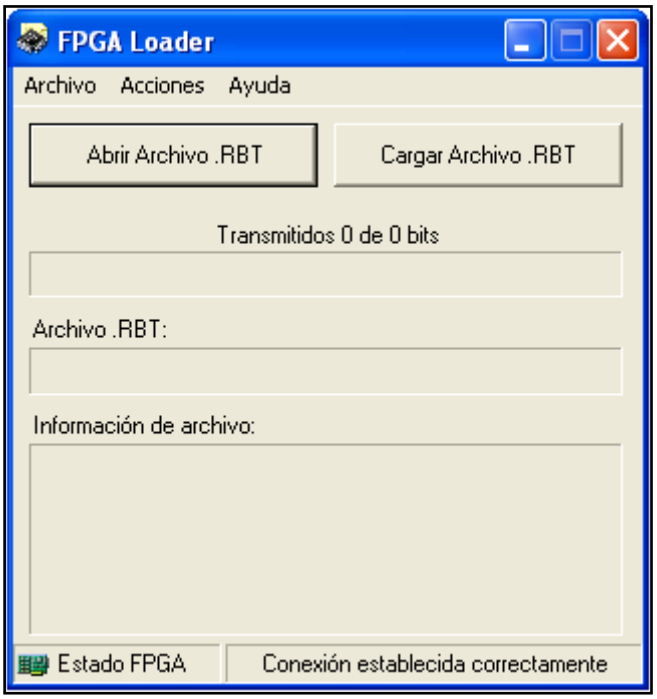
- **El campo Información de archivo:** Muestra la cabecera del fichero .RBT, donde aparece, entre otras cosas, cuál es el tipo del fichero, el nombre del diseño, la arquitectura de FPGA para la que ha sido diseñado, la fecha de creación y el número de bits del fichero.

- **Estado FPGA:** Informa del estado en que se encuentra la conexión con la FPGA, conectada o no conectada. Para poder utilizar el programa correctamente, éste debe indicar que la FPGA está conectada; de no ser así, se mostrará un mensaje informando de ello. El estado de conexión se puede actualizar mediante el comando Actualizar estado FPGA del menú Acciones o pulsando sobre el icono de estado.

Los pasos a seguir para realizar una carga correcta de un fichero .RBT son los siguientes:

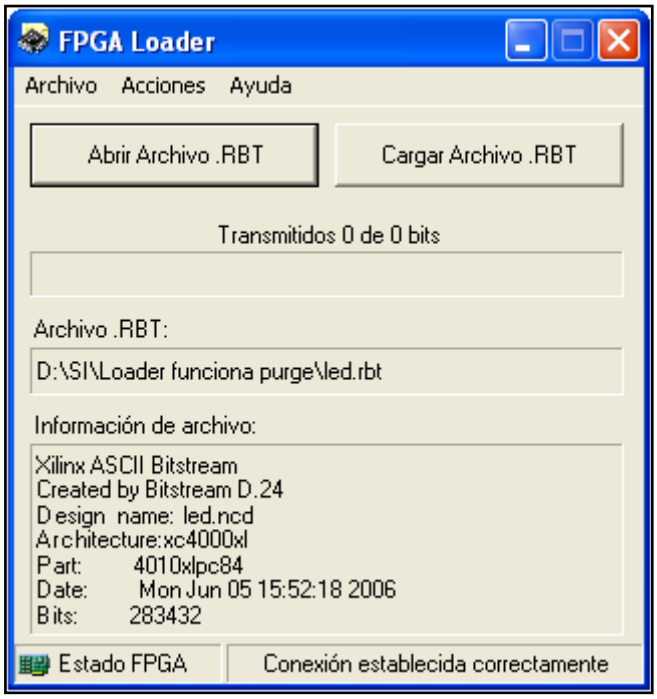
- Comprobar que el estado de la FPGA sea conectado.

Figura 38: FPGA Loader conexión establecida



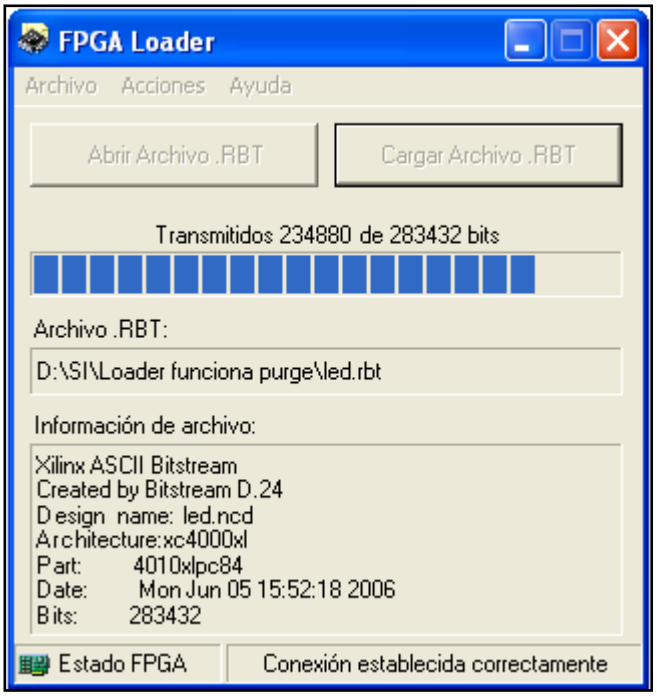
- Apertura del fichero .RBT.

Figura 39: FPGA Loader fichero abierto



- Carga del fichero .RBT en la FPGA.

Figura 40: FPGA Loader transmitiendo fichero



Una vez finalizada la carga del fichero .RBT en la FPGA, se mostrará un mensaje indicando si ha habido éxito o no en dicho proceso.

Ejemplo de funcionamiento

Se ha realizado una prueba del funcionamiento de la utilidad para configurar FPGAs sobre una FPGA de la familia XC4000XL, exactamente la XC4010XL-PC84CMN0217 de 84 pines, mostrada en la figura 41. Esta FPGA es más antigua que la que se iba a probar inicialmente; el cambio de FPGA pudo haber traído dos problemas concernientes al proceso de configuración: primero, que usara un protocolo de configuración distinto; y segundo, que, al ser una FPGA más antigua, no soportara la frecuencia del reloj generado para la señal CCLK.

La frecuencia máxima de reloj que se genera es de 2 MHz, lo cual no supone ningún problema para la FPGA XC2S100-5 TQ144C, cuya frecuencia máxima permitida es de 66 MHz. Tampoco lo es para la nueva FPGA, cuya frecuencia es de 15 MHz como máximo.

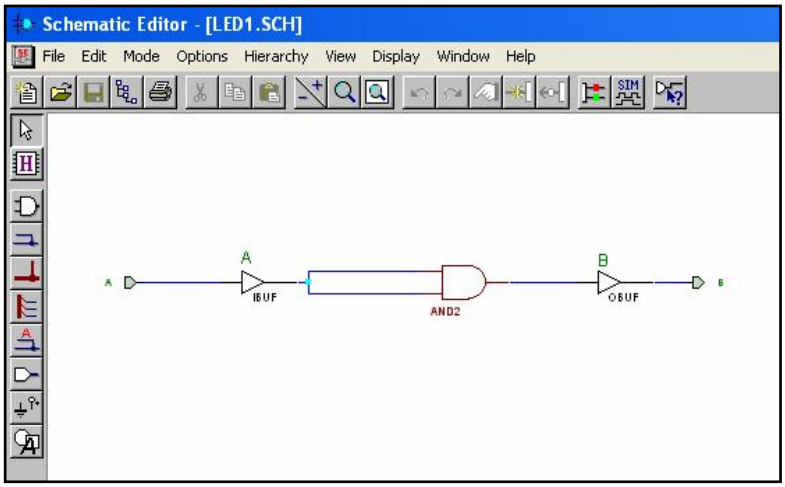
Con respecto al protocolo de configuración, tampoco implica ningún problema, ya que ambas FPGAs tienen protocolos de configuración compatibles.

Figura 41: FPGA XC4010XL-PC84CMN0217 con un LED en el pin 51

La prueba consiste en configurar la FPGA para que un LED conectado a la patilla 51, se encienda. Como se puede ver, es un ejemplo muy sencillo, pero no es necesario complicarlo más para comprobar si se ha conseguido programar la FPGA con éxito.

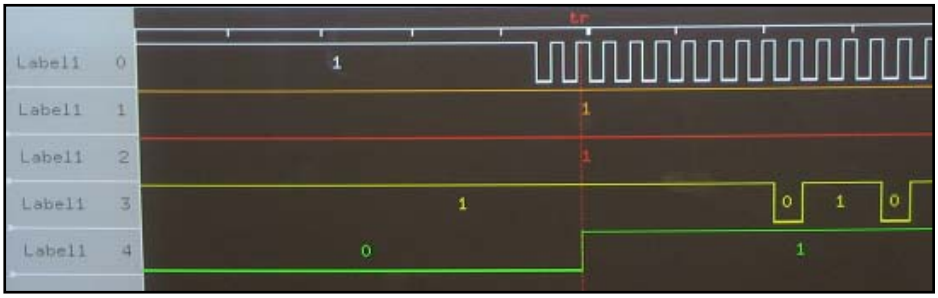
Inicialmente se iba a hacer un diseño sencillo, en el que simplemente se unía la patilla 51 a Vcc, pero este tipo de diseño tan poco útil no lo permite la herramienta Xilinx Foundations, así que finalmente se realizó un diseño con un switch que encendiera el LED, en el que se utilizó como switch la patilla 50 (figura 42).

Figura 42: Esquemático de la prueba de programación. El IBUF A corresponde a la patilla 50 y el OBUF B a la patilla 51



Al terminar la programación (figura 3), si se hace una conexión entre la patilla 50 y algún VCC, el LED se encenderá. La conexión se realizó uniendo el switch mediante un cable con el VCC de la patilla 54, ambas cosas se muestran en la figura 44.

Figura 43: **Fin de la programación con la puesta de la señal DONE (label 4) en alta**



Por último, añadir que para generar el archivo RBT de esta prueba se ha tenido que usar una versión antigua del Xilinx Foundations (en concreto la F1.5 del año 1998), debido a que las nuevas versiones (como Xilinx ISE 6) no traen las librerías de las FPGAs de la familia XC4000.

Figura 44: El LED encendido después de la programación de la FPGA

PCB

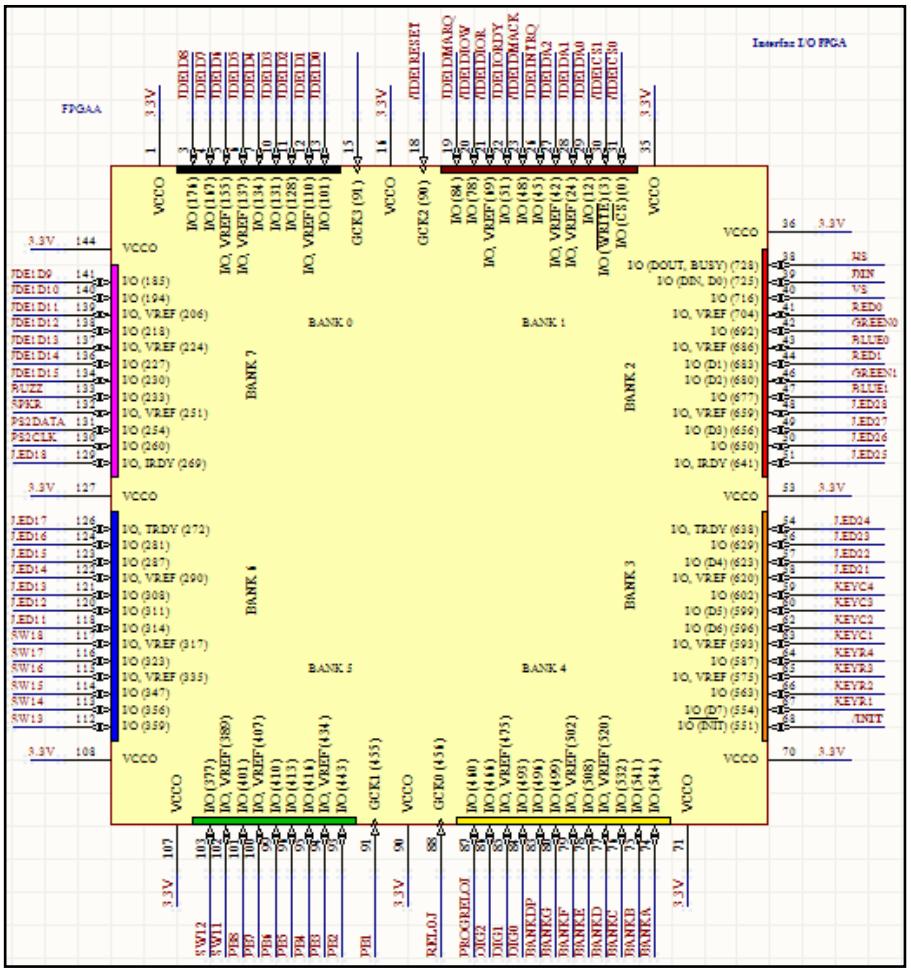
Se ha realizado una propuesta de diseño de la placa didáctica completa usando la herramienta DXP 2004 y siguiendo unas especificaciones dadas:

- La placa se podrá alimentar a través del bus USB o de una fuente de alimentación externa de 5 V, según la posición de un jumper.
- La placa contendrá 8 pulsadores (8 pines de usuario necesarios).
- La placa contendrá 2 bancos de 8 LEDs (16 pines de usuario necesarios).
- La placa contendrá 2 bancos de 8 switches deslizantes (16 pines de usuario necesarios).
- La placa contendrá 8 displays 7-segmentos (11 pines de usuario necesarios).
- La placa contendrá un teclado matricial de 4x4 (8 pines de usuario necesarios).
- La placa contendrá un zumbador y un speaker (2 pines de usuario necesarios).
- La placa contendrá un reloj programable mediante jumper (2 pines de usuario necesarios).
- La placa contendrá un puerto PS/2 (2 pines de usuario necesarios).
- La placa contendrá un puerto VGA (8 pines de usuario necesarios).
- La placa contendrá dos conectores IDE (28 pines de usuario necesarios).

Sin embargo, es imposible añadir tanta circuitería adicional de usuario a la FPGA sin que los componentes compartan pines. Se dispone de 96 pines de usuario, contando los 4 relojes de usuario (GCK1 – GCK4). Así que se decidió prescindir de los siguientes elementos:

- Segundo conector IDE.
- Segundo banco de 8 switches.

Figura 45: Esquemático de la interfaz de E/S de usuario de la Spartan-II

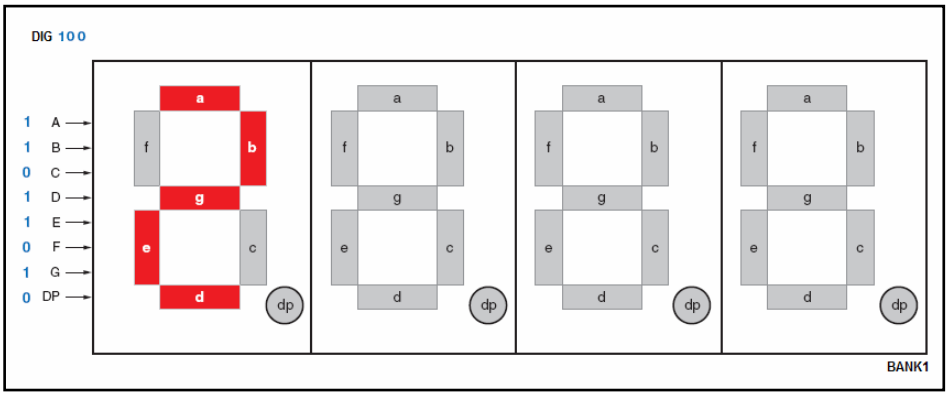


De esta forma, se ocuparían todos los pines disponibles en la FPGA, salvo 5. En el caso de no necesitar los pulsadores y el teclado matricial simultáneamente, se podrían usar únicamente 8 pines para un conjunto de 4 pulsadores y el teclado. De esta forma se dispondría de 4 pines libres más. Se puede obtener un pin libre más usando el speaker y el zumbador con el mismo pin y eligiendo cuál usar mediante un jumper. También, si el puerto VGA sólo usara un pin por cada color, se obtendrían 3 pines libres más. Se pueden, por tanto, conseguir un total de 13 pines libres, así que sería posible incluir el segundo banco de 8 switches, si se realizaran los cambios anteriores. El segundo conector IDE no se podrá añadir, salvo que se compartan 28 pines ya usados de la FPGA.

Banco de displays 7-segmentos

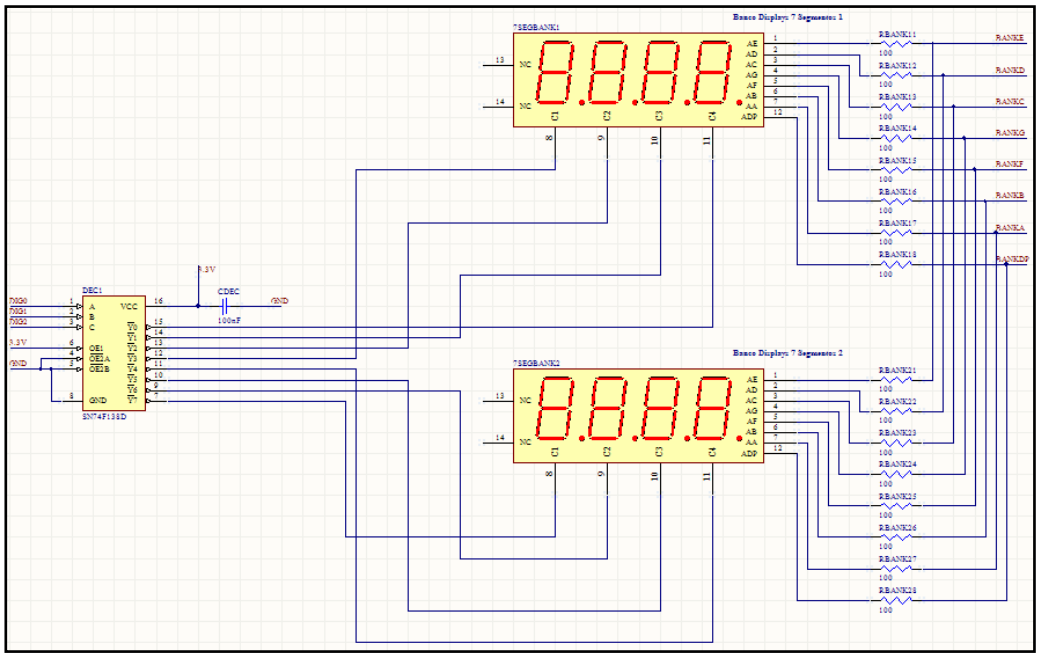
La placa de prototipado dispondrá de dos bancos de 4 displays 7-segmentos cada uno, de forma que se tienen un total de 8 displays. Los bancos serán de cátodo común (lógica directa). Cada dígito comparte ocho líneas de control para iluminar cada uno de los siete segmentos del dígito junto con el punto decimal. Estas líneas se gestionan desde ocho pines de usuario de la FPGA. Para seleccionar el dígito, existen otras 3 señales, que a través de un decodificador de 3 a 8 (modelo SN74F138D de Texas Instruments), permiten indicar qué dígito iluminar con un número binario de 3 bits (también en lógica directa).

Figura 46: Banco de displays 7-segmentos



El esquemático es el siguiente:

Figura 47: **Esquemático de los bancos de displays 7-segmentos**



La interconexión con la FPGA estará formada por 11 patillas:

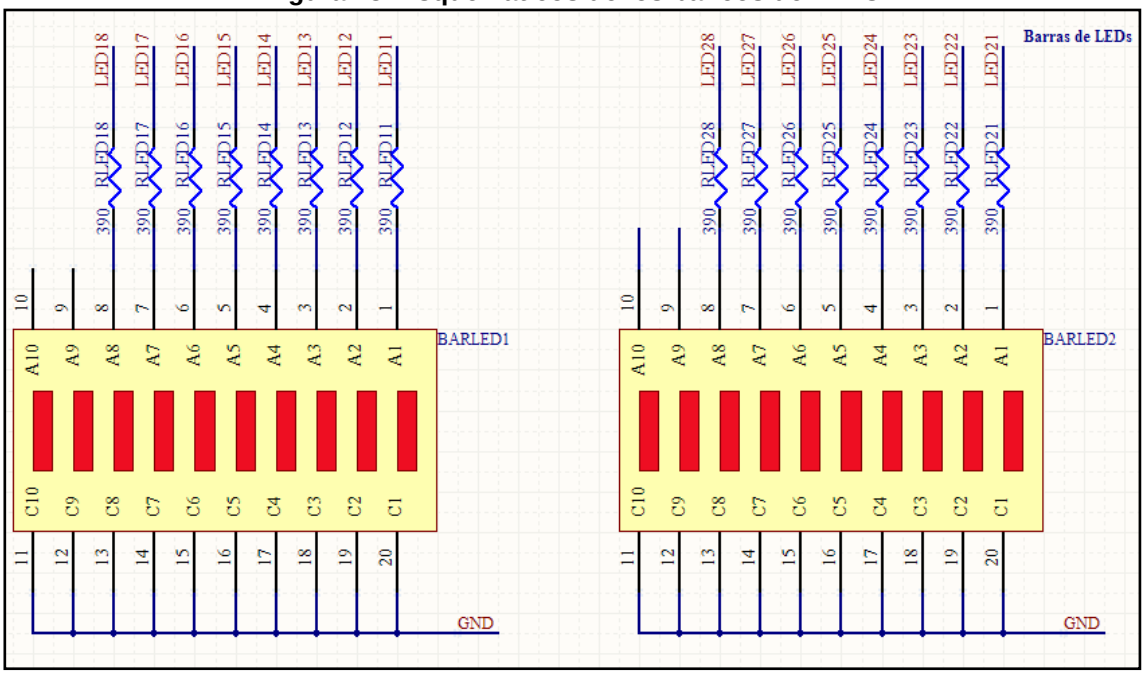
Señal	Patilla de la FPGA
Bit menos significativo de selección de display	DIG0 (84)
Segundo bit de selección de display	DIG1 (85)
Bit más significativo de selección de display	DIG2 (86)
Segmento A del display seleccionado	BANKA (74)
Segmento B del display seleccionado	BANKB (75)
Segmento C del display seleccionado	BANKC (76)
Segmento D del display seleccionado	BANKD (77)
Segmento E del display seleccionado	BANKE (78)
Segmento F del display seleccionado	BANKF (79)
Segmento G del display seleccionado	BANKG (80)
Punto decimal del display seleccionado	BANKDP (83)

Bancos de LEDs

Se conectarán dos bancos de 10 LEDs, utilizando sólo 8 de cada banco, para obtener un total de 16 LEDs de usuario en la placa de prototipado. Cada uno de estos LEDs se podrá iluminar individualmente gracias a los 16 pines de usuario de la FPGA asignados.

El esquema queda como sigue:

Figura 48: Esquemáticos de los bancos de LEDs



Y la interconexión con la FPGA es:

Señal	Patilla de la FPGA
Ánodo del LED 1 del banco 1 y ánodo del LED 1 del banco 2	LED11 (118) y LED21 (58)
Ánodo del LED 2 del banco 1 y ánodo del LED 2 del banco 2	LED11 (120) y LED21 (57)
Ánodo del LED 3 del banco 1 y ánodo del LED 3 del banco 2	LED11 (121) y LED21 (56)
Ánodo del LED 4 del banco 1 y ánodo del LED 4 del banco 2	LED11 (122) y LED21 (54)
Ánodo del LED 5 del banco 1 y ánodo del LED 5 del banco 2	LED11 (123) y LED21 (51)
Ánodo del LED 6 del banco 1 y ánodo del LED 6 del banco 2	LED11 (124) y LED21 (50)
Ánodo del LED 7 del banco 1 y ánodo del LED 7 del banco 2	LED11 (126) y LED21 (49)
Ánodo del LED 8 del banco 1 y ánodo del LED 8 del banco 2	LED11 (129) y LED21 (48)

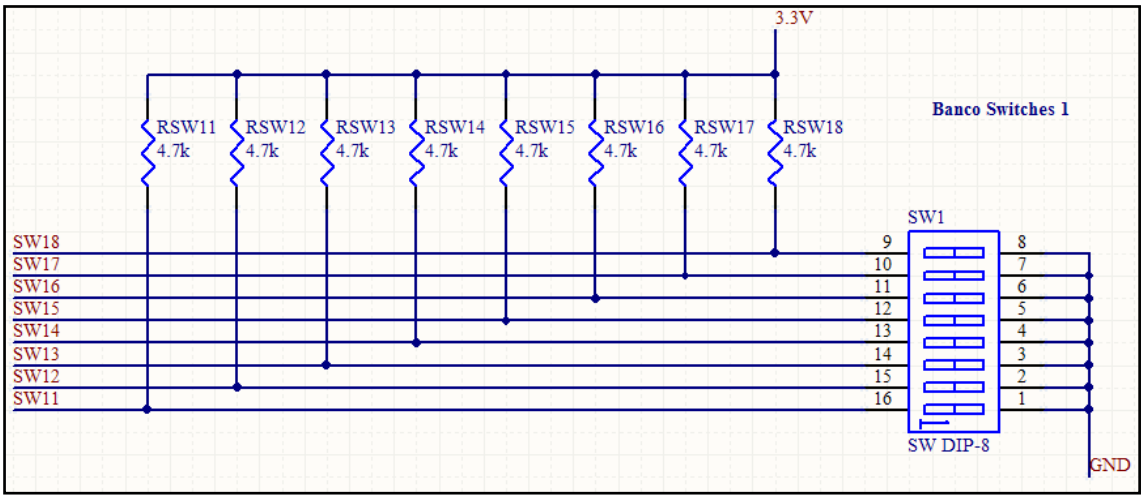
Banco de switches

Se ha añadido un banco de 8 switches deslizantes, que entran a la FPGA a través de 8 líneas de usuario. El patillaje viene resumido en la siguiente tabla:

Señal	Patilla de la FPGA
Switch 1	SW11 (102)
Switch 2	SW12 (103)
Switch 3	SW13 (112)
Switch 4	SW14 (113)
Switch 5	SW15 (114)
Switch 6	SW16 (115)
Switch 7	SW17 (116)
Switch 8	SW18 (117)

La captura de este módulo se recoge en la siguiente figura:

Figura 49: Esquemáticos del banco de switches



Pulsadores

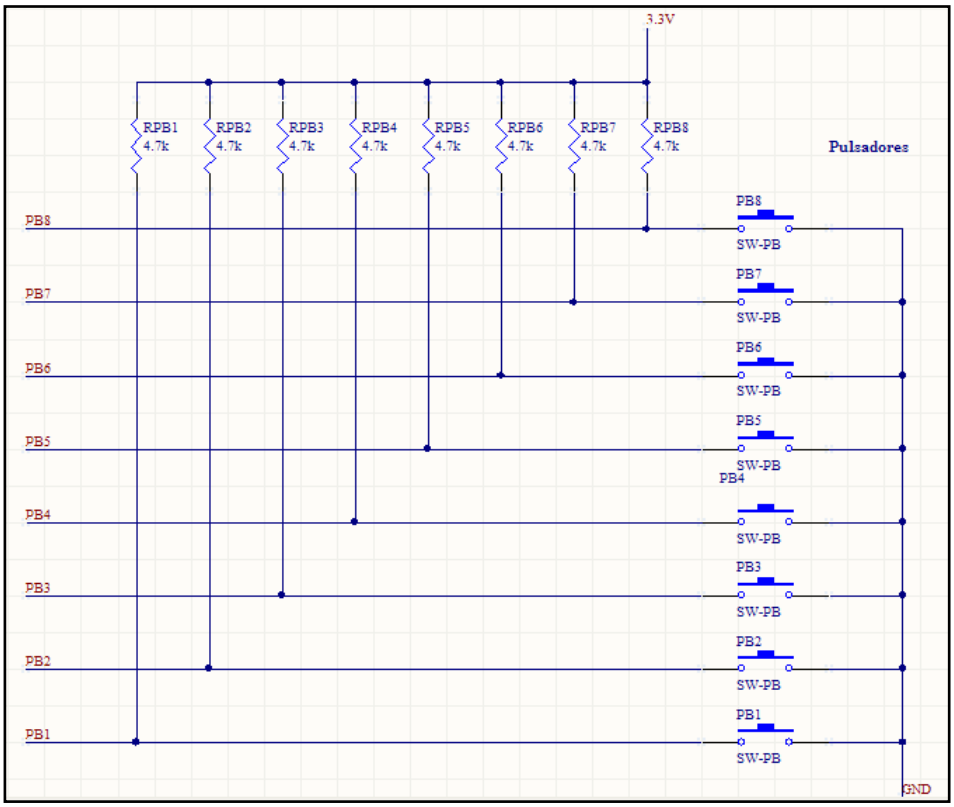
También se incluye en la placa de prototipado un conjunto de 8 botones pulsadores, conectados a 8 pines de usuario de la FPGA.

La conexión con la FPGA es la siguiente:

Señal	Patilla de la FPGA
Pulsador 1	PB1 (91)
Pulsador 2	PB2 (93)
Pulsador 3	PB3 (94)
Pulsador 4	PB4 (95)
Pulsador 5	PB5 (96)
Pulsador 6	PB6 (99)
Pulsador 7	PB7 (100)
Pulsador 8	PB8 (101)

Y el esquemático es:

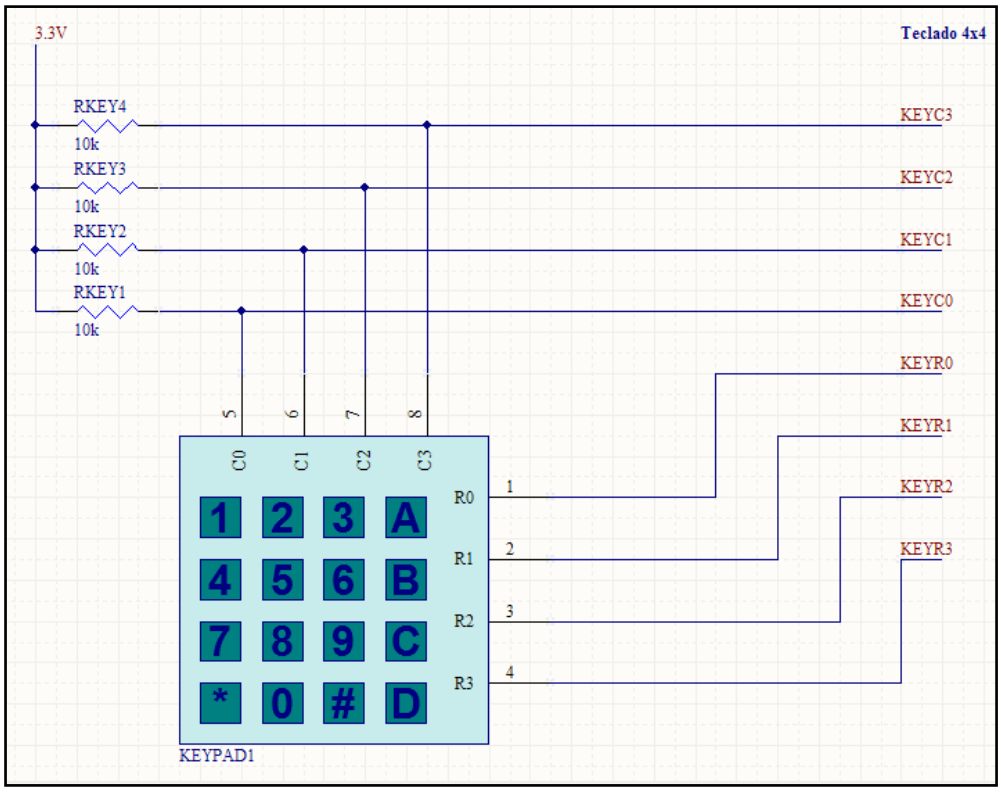
Figura 50: Esquemático de los pulsadores



Teclado matricial

La placa contará con un teclado matricial de 4x4, requiriendo 4 pines de la FPGA para el control de las filas y otros 4 para el de las columnas.

Figura 51: Esquemático del teclado matricial



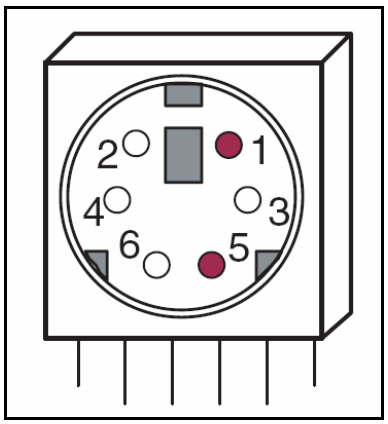
Las patillas usadas en la FPGA vienen definidas en la siguiente tabla:

Señal	Patilla de la FPGA
Columna 1 del teclado matricial	KEYC1 (63)
Columna 2 del teclado matricial	KEYC2 (62)
Columna 3 del teclado matricial	KEYC3 (60)
Columna 4 del teclado matricial	KEYC4 (59)
Fila 1 del teclado matricial	KEYR1 (67)
Fila 2 del teclado matricial	KEYR2 (66)
Fila 3 del teclado matricial	KEYR3 (65)
Fila 4 del teclado matricial	KEYR4 (64)

Puerto PS/2

La placa didáctica poseerá un puerto PS/2 para ratón y teclado. Para ello, se ha incluido en el diseño un conector mini-DIN de 6 pines, de los cuales sólo dos estarán conectados directamente a la FPGA, ya que tanto los ratones como los teclados de PC necesitan un bus serie de 2 líneas para comunicarse con el host (la FPGA en este caso). Por una de ellas irán los datos (DATA) y por la otra la señal de reloj de referencia (CLK). Los datos varían si la interfaz es con un ratón (estado y posición del mismo) o con un teclado (código de la tecla pulsada). En el caso del teclado, la comunicación es además bidireccional, para indicar si se deben iluminar o no los LEDs del teclado (Mayúsculas, Scroll Lock y Numérico).

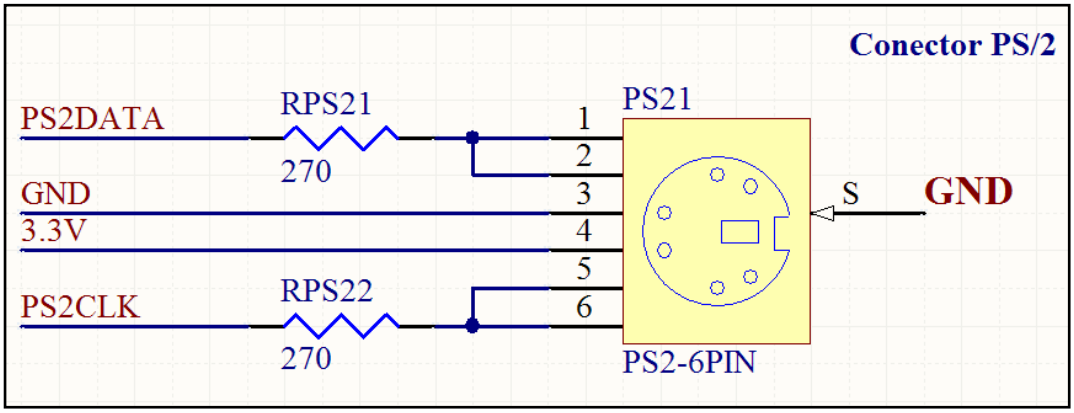
Figura 52: Conector PS/2



La mayoría de los teclados y ratones modernos funcionan tanto con un voltaje de 3.3 V como de 5 V. En esta placa, la alimentación del puerto PS/2 es de 3.3 V, con lo que cabe la posibilidad de que algunos modelos antiguos de teclado o ratón no sean compatibles.

El esquema de conexión del puerto PS/2 es el siguiente:

Figura 53: Esquemático del conector PS/2



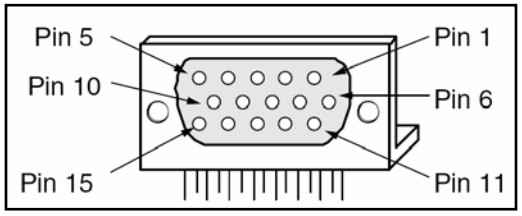
El patillaje viene recogido en la siguiente tabla:

Patilla del puerto PS/2	Señal	Patilla de la FPGA
1	DATA	PS2DATA (131)
2	Reservada	-
3	GND	-
4	Alimentación	-
5	CLK	PS2CLK (130)
6	Reservada	-

Puerto VGA

Se ha añadido a la placa un conector DB15 para permitir la conexión de monitores CRT y pantallas LCD a la misma, a través de un puerto VGA. El conector dispone de 15 pines, de los cuales 5 estarán conectados a la FPGA, para así poder controlar las cinco señales implicadas en la interfaz VGA: Rojo (R), Verde (G), Azul (B), Sincronización Horizontal (HS) y Sincronización Vertical (VS).

Figura 54: Conector DB15

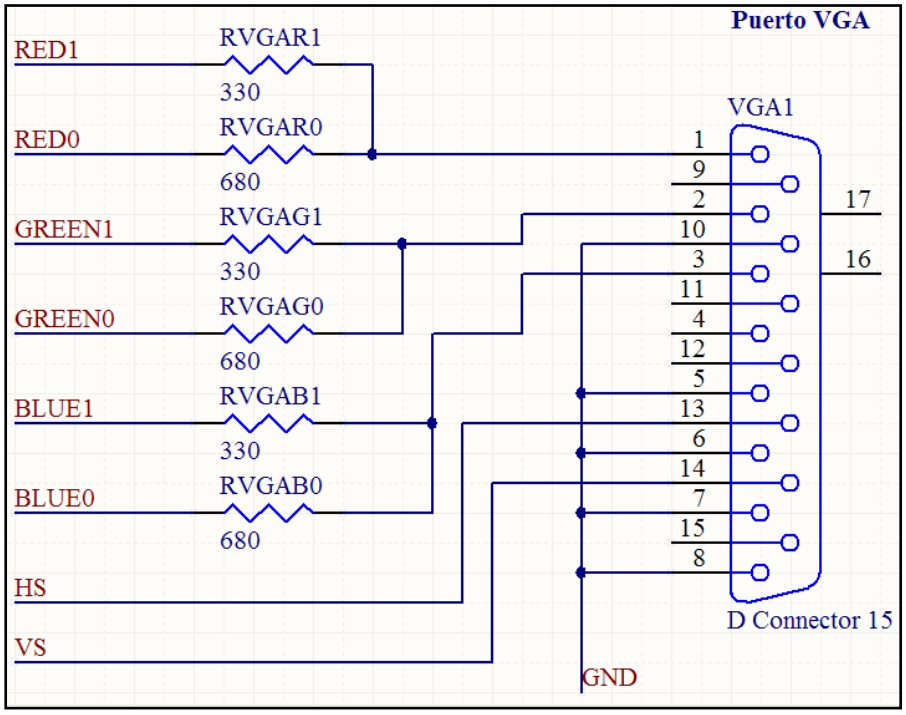


La FPGA en realidad dedicará 8 pines a la interfaz con el puerto VGA, ya que se destinarán dos pines para cada señal de color.

Patilla del puerto VGA	Señal	Patilla de la FPGA
1	RED	RED0 (41) y RED1 (44)
2	GREEN	GREEN0 (42) y GREEN1 (46)
3	BLUE	BLUE0 (43) y BLUE1 (47)
13	HORIZONTAL SYNC	HS (38)
14	VERTICAL SYNC	VS (40)
5, 6, 7, 8 , 10	GND	-

El esquemático del puerto VGA es el siguiente:

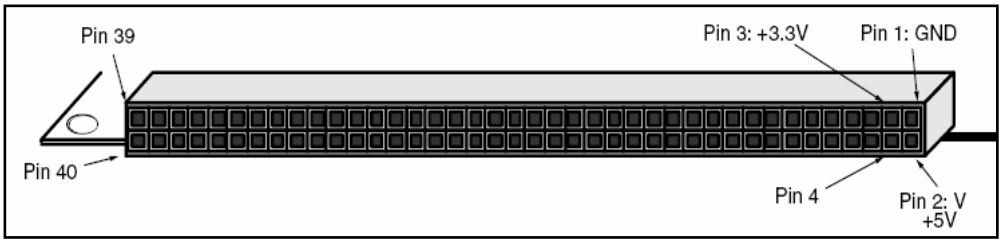
Figura 55: Esquemático del puerto VGA



Interfaz IDE

La placa didáctica contará con un conector de 40 pines conectados a 28 patillas de la FPGA para implementar una interfaz IDE (Integrated Drive Electronics), utilizada normalmente para interconectar un disco duro en paralelo con la placa base de un ordenador personal. Gracias a este conector, se podrá conectar cualquier dispositivo de expansión a la placa de prototipado.

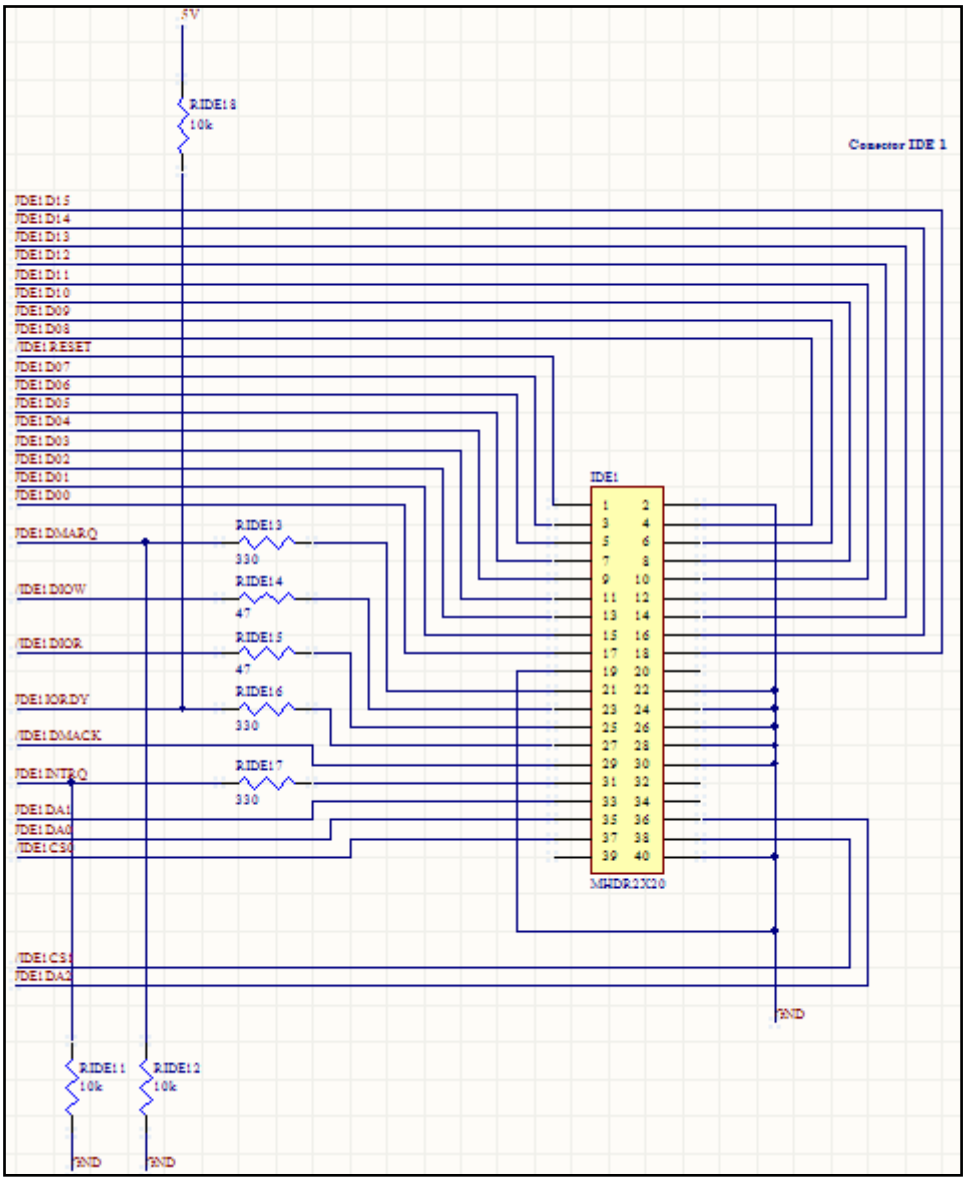
Figura 56: Conector IDE



Patilla del conector IDE	Señal	Patilla de la FPGA
1	RESET	/IDE1RESET (18)
2	GND	-
3	DATA7	IDE1D7 (4)
4	DATA8	IDE1D8 (3)
5	DATA6	IDE1D6 (5)
6	DATA9	IDE1D9 (141)
7	DATA5	IDE1D5 (6)
8	DATA10	IDE1D10 (140)
9	DATA4	IDE1D4 (7)
10	DATA11	IDE1D11 (139)
11	DATA3	IDE1D3 (10)
12	DATA12	IDE1D12 (138)
13	DATA2	IDE1D2 (11)
14	DATA13	IDE1D13 (137)
15	DATA1	IDE1D1 (12)
16	DATA14	IDE1D14 (135)
17	DATA0	IDE1D0 (13)
18	DATA15	IDE1D15 (134)
19	GND	-
20	KEY	-
21	DMARQ	IDE1DMARQ (19)
22	GND	-
23	/DIOW	/IDE1DIOW (20)
24	GND	-
25	/DIOR	/IDE1DIOR (21)
26	GND	-
27	IORDY	IDE1IORDY (22)
28	CSEL	-
29	/DMARK	/IDE1DMACK (23)
30	GND	-
31	INTRQ	IDE1INTRQ (26)
32	/IOCS16	-
33	DA1	IDE1DA1 (28)
34	/PDIAG	-
35	DA0	IDE1DA0 (29)
36	DA2	IDE1DA2 (27)
37	/CS1FX	/IDE1CS0 (30)
38	/CS3FX	/IDE1CS1 (31)
39	/DASP	-
40	GND	-

La conexión se muestra en la siguiente figura:

Figura 57: Esquemático de la interfaz IDE



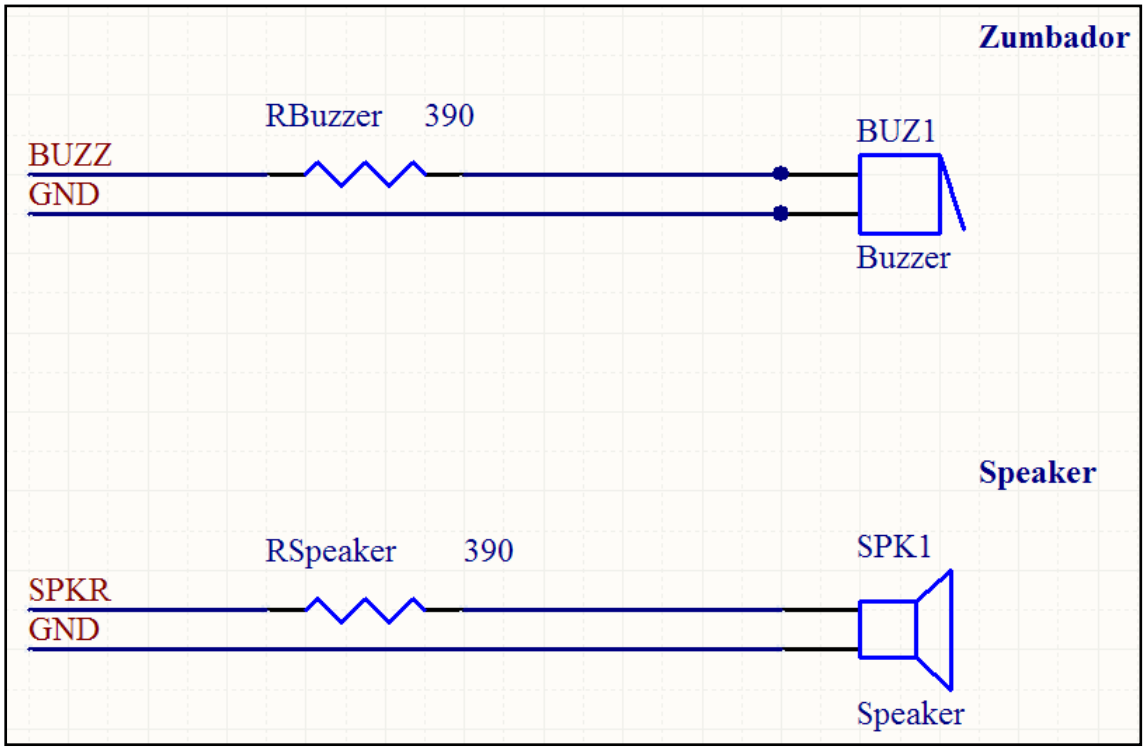
Zumbador y Speaker

Se podrá utilizar la placa para generar sonidos mediante un altavoz (enviando señales periódicas a través de un pin de usuario de la FPGA) o mediante un zumbador (haciendo que suene o no según el estado de otro pin de la FPGA). Las patillas utilizadas por el zumbador y el speaker son las siguientes:

Señal	Patilla de la FPGA
Altavoz	SPKR (132)
Zumbador	BUZZ (133)

El esquemático se muestra a continuación:

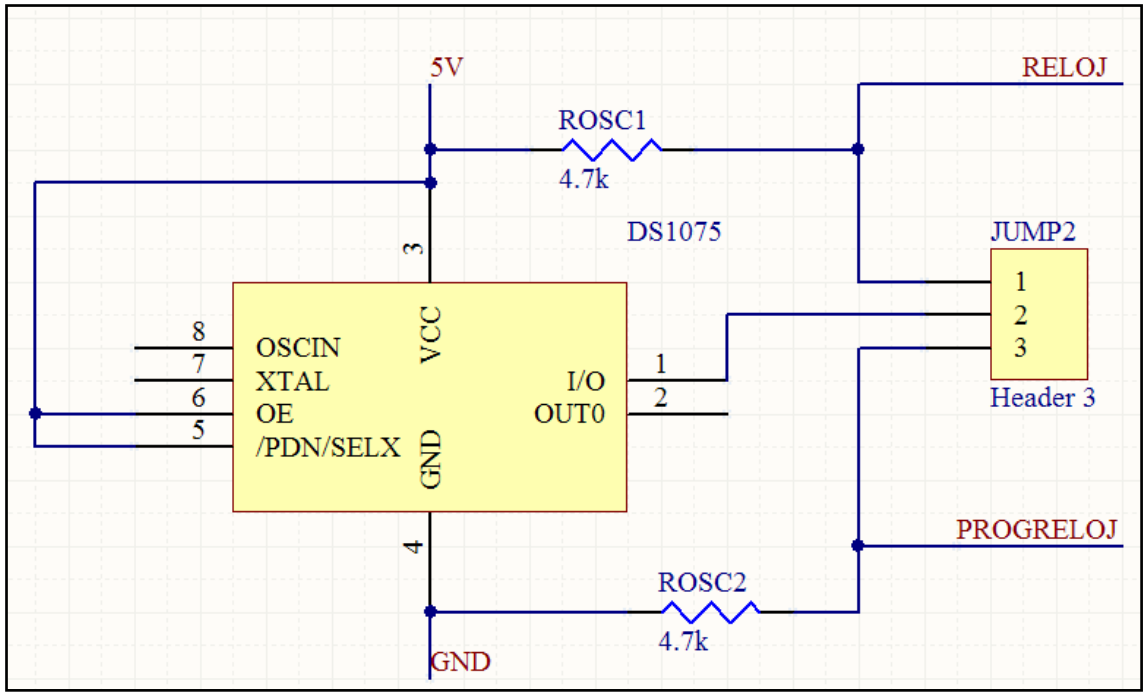
Figura 58: Esquemático del zumbador y del speaker



Oscilador programable

Para poder disponer de una fuente de reloj de usuario, se ha conectado a la FPGA un oscilador programable modelo 1075 de Dallas Semiconductor. Para ello, se han usado dos patillas: una para la programación de la frecuencia del reloj desde la FPGA y otra para introducir la señal de reloj generada en el oscilador en ella. La elección entre programación del oscilador o generación de reloj se realiza mediante un jumper.

Figura 59: Esquemático del oscilador programable



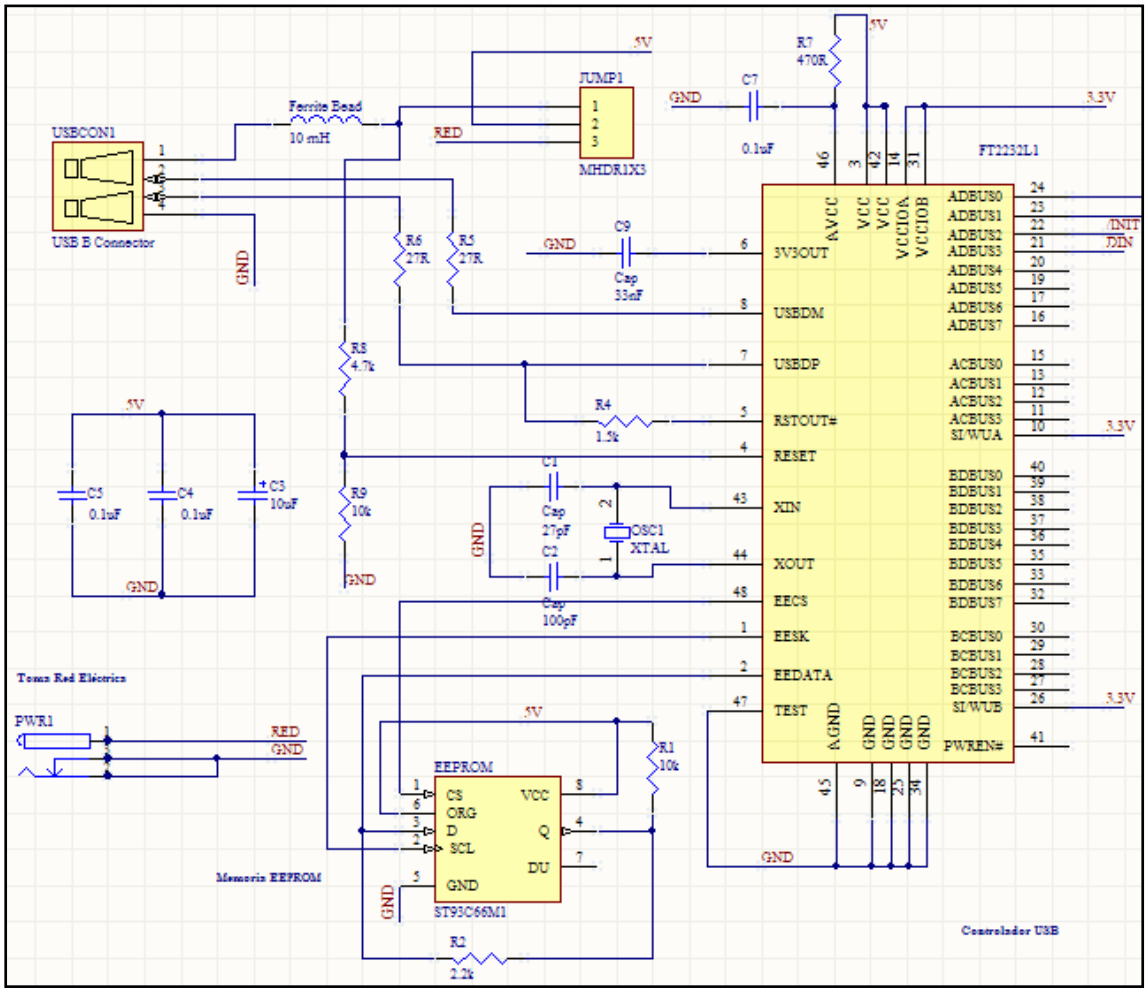
Los pines empleados de la FPGA son los siguientes:

Señal	Patilla de la FPGA
Señal del reloj del oscilador	RELOJ (88)
Señal de programación del oscilador	PROGRELOJ (87)

Alimentación

La placa dispondrá de dos alternativas a la hora de elegir la fuente de alimentación. Se podrá optar por una alimentación a través del bus USB o por usar una fuente de alimentación externa de 5 V. La selección entre una y otra se realizará mediante un jumper.

Figura 60: Esquemático de la interfaz eléctrica de la placa

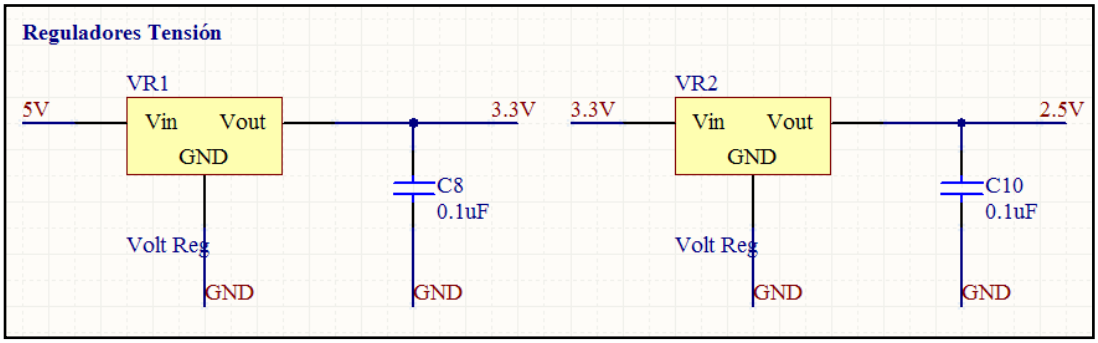


No todos los componentes presentes en la placa se alimentarán con 5 V. Es necesario usar un par de reguladores de tensión para obtener a partir de este voltaje uno de 3.3 V y otro de 2.5 V. El voltaje de alimentación correspondiente a cada dispositivo es el siguiente:

- Núcleo de la FPGA Spartan-II: 2.5 V.
- Pines de E/S de usuario de la FPGA Spartan-II: 3.3 V.
- Núcleo del controlador FT2232C: 5 V.
- Pines de E/S de los dos canales A y B del controlador FT2232C: 3.3 V.
- EEPROM: 5 V.
- Displays 7-segmentos, pulsadores, switches, teclado matricial, puerto PS/2: 3.3 V.
- Interfaz IDE y oscilador programable: 5 V.

Las redes de desacoplo de los reguladores de tensión serán más complejas que las mostradas y dependerán del modelo elegido:

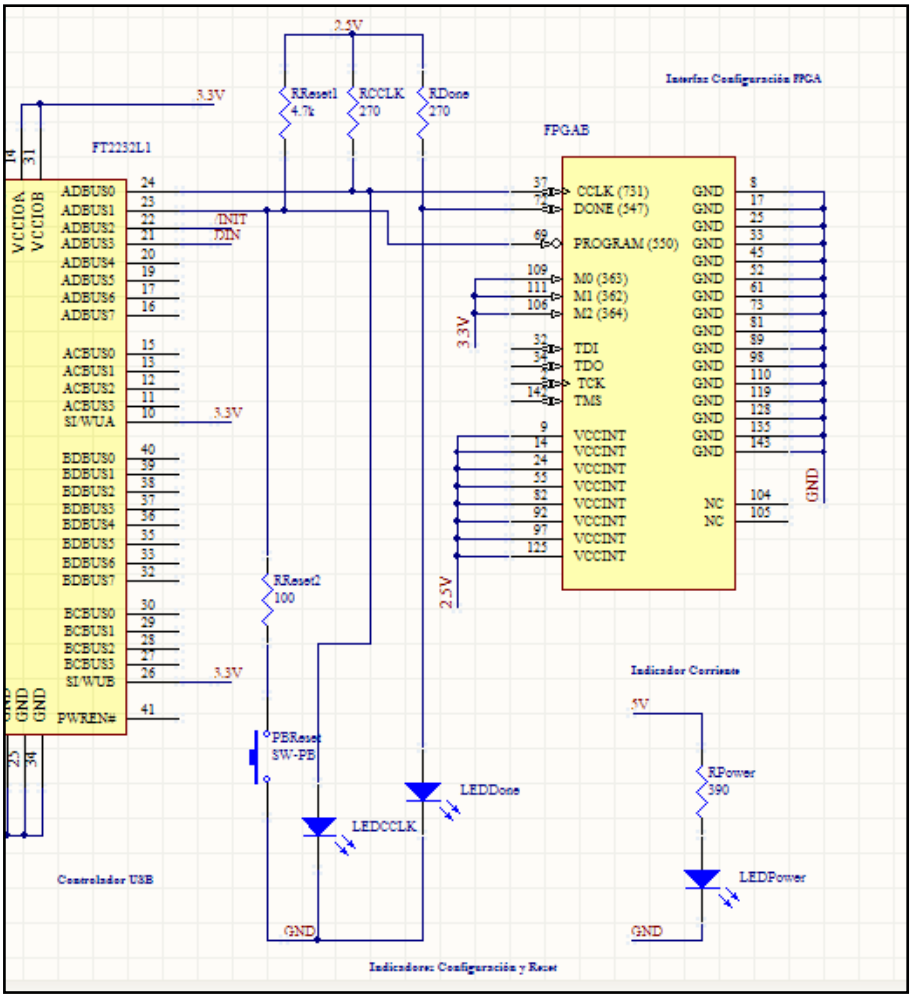
Figura 61: Esquemático de los reguladores de nivel



Indicadores de estado y botón de reset

La placa contará además con tres LEDs indicando el estado de la configuración y de la alimentación y con un pulsador para resetear la FPGA. Uno de los LEDs se encenderá durante la configuración (va conectado a la línea del reloj de configuración de la FPGA, generado por programa). Otro indicará si la configuración se ha realizado con éxito (conectado con la señal DONE de la FPGA). El indicador de alimentación se iluminará siempre que a la placa le llegue una tensión eléctrica de 5 V, ya sea a través del bus USB o a través de una fuente de alimentación externa. El botón de reset va unido a la patilla /PROGRAM de la FPGA.

Figura 62: Esquemático de la interfaz de configuración de la FPGA



Problemas encontrados

Durante el desarrollo del proyecto aparecieron varios problemas cuya resolución requirió bastante tiempo y que impidieron abordar la fabricación del diseño completo de la placa basada en FPGA.

El problema inicial fue abordar un campo en el que nunca antes se había entrado. Se había trabajado con placas FPGA en la asignatura Laboratorio de Tecnología de Computadores, pero no al nivel del diseño de la placa misma, ni del hardware asociado a ella. No se habían realizado trabajos anteriormente con protocolos de comunicación, ni con drivers de interfaces hardware. No se tenía experiencia alguna en diseño y fabricación de circuitos impresos. Por tanto, fue necesario familiarizarse con gran cantidad de conceptos y tecnologías nuevas, empleando toda documentación relacionada disponible en Internet. Todo esto, junto con la comprensión del sistema global, ocupó el tiempo disponible hasta el inicio del segundo cuatrimestre.

Después de esta fase, la principal dificultad encontrada fue la depuración de hardware. Una vez el proyecto estaba en marcha, la aparición de fallos en los circuitos impresos, bien fallos en el diseño, o bien fallos en el montaje del prototipo, suponía el empleo de varios días de depuración, retrasando los hitos de la planificación del proyecto.

Por otro lado, también costó aprender a manejar adecuadamente la Interfaz de Programación de Aplicaciones D2XX del controlador USB, provista de decenas de funciones, muchas de las cuales no tenían relevancia en el desarrollo del proyecto.

Otro problema software que apareció en más de una ocasión está relacionado con la configuración de la EEPROM. Si la información contenida en está memoria está corrupta o se usan unos campos VID y PID distintos a los que da el fabricante del chip FT2232L, al conectar el prototipo al PC, Windows no asociaba los drivers instalados con el chip, por lo que no se podía seguir trabajando con el interfaz D2XX, al no tener acceso al dispositivo. La primera vez que surgió esté problema, se solucionó reemplazando la EEPROM por una nueva y vacía (también se podría haber usado el analizador lógico para generar el cronograma que resetea la EEPROM, pero este procedimiento parecía más engorroso que reemplazar la memoria). La segunda vez que ocurrió este problema, se solucionó de una forma más cómoda, pues no requiere modificaciones en el circuito. Se modificaron los drivers con un editor de texto (en concreto el archivo de información de instalación ftd2xx.inf) y se cambiaron los valores del PID y del VID a los que se habían introducido por error en la memoria. Para ello, se usó el Administrador de Dispositivos para conocer estos valores y se sustituyeron en las siguientes líneas del archivo ftd2xx.dll:

```
USB\VID_0403&PID_6001.DeviceDesc="FTDI FT8U2XX Device"
USB\VID_0403&PID_6010&MI_00.DeviceDesc="FT2232C Channel A"
USB\VID_0403&PID_6010&MI_01.DeviceDesc="FT2232C Channel B"
```

Con este cambio, la placa si era detectada por los drivers al conectarla. Una vez que se configuró la EEPROM adecuadamente, se volvieron a usar los valores originales en el archivo ftd2xx.dll.

También hubo problemas a la hora de elegir el interfaz con el que usar el canal de entrada/salida del controlador USB. Inicialmente se pensó en el modo FIFO, pero cuando ya estaba prácticamente concluido el software de comuniación, se descubrió que la interfaz FIFO requería un dispositivo externo para generar las señales RD# y WR, para así indicar cuándo leer del buffer de transmisión o cuándo escribir en el de escritura. Finalmente, se decidió usar el Asynchronous Bit-Bang Mode, que funciona exactamente igual que el FIFO, excepto en que se escribe y se lee en cada canal con cada flanco del reloj interno del controlador USB. Luego no fue necesario añadir circutiería externa para gestionar las señales RD# y WR.

En conclusión podríamos decir que se han tenido problemas de tres tipos: conceptuales, de hardware, y de software.

CONCLUSIONES

En el desarrollo de este proyecto se han visto una gran cantidad de aspectos y conceptos nuevos, que nunca antes se habían estudiado con tanto detalle en las asignaturas relacionadas con la electrónica y la arquitectura de computadores. Ha sido un proyecto complejo, con tareas muy diversas y vertientes en todos los niveles de la Informática (programación de alto nivel, programación de bajo nivel, diseño lógico, implementación de protocolos de comunicación, diseño y depuración de circuitos electrónicos y fabricación de PCBs).

Se ha aprendido cuál es el proceso completo que se debe seguir para fabricar un dispositivo hardware, proceso dividido en cuatro fases (documentación y análisis de las tecnologías disponibles, diseño del sistema, fabricación del circuito impreso y testeo).

Se ha visto la importancia de la fase de documentación, en la que se ha empleado gran parte del tiempo. Antes de iniciar cualquier proyecto hardware, es necesario dedicar un gran esfuerzo al análisis de los protocolos, dispositivos y procesos de diseño existentes, para así conseguir una enfoque adecuado al problema.

Durante el diseño y fabricación de los prototipos se ha tenido constancia de la enorme dificultad que entraña todo el proceso de depuración, ya que siempre surgen problemas de muy distinta naturaleza, imposibles de prever a priori. Los problemas encontrados van desde errores en las conexiones físicas de los componentes, hasta problemas de incompatibilidades de voltajes de los distintos elementos, pasando por fallos de estos elementos.

Debido a los muchos problemas encontrados, no se ha podido cumplir totalmente el objetivo final del proyecto que era el diseño completo de una placa didáctica basada en FPGA. Se han llevado a término algunas de las partes más importantes, como la comunicación entre el PC y FPGA a través del puerto USB, la carga del fichero de configuración en la FPGA y una primera versión del diseño de la placa completa. Sin embargo, la fabricación y testeo del circuito impreso no se ha llegado a realizar por falta de tiempo.

Este proyecto no ha sido terminado en su totalidad, por lo que se presta a que sea continuado por otro grupo de alumnos que cursen la asignatura de Sistemas Informáticos en el próximo curso. Algunas de las posibles líneas de continuidad son la depuración del diseño de los esquemáticos de la placa didáctica completa, la fabricación de su prototipo y el posterior testeo y verificación del correcto funcionamiento de cada uno de sus componentes y de la placa en conjunto.

En definitiva, la realización del proyecto ha servido para acercarse a una de las ramas más complejas de la Informática, en la que nunca se había pensado anteriormente para encaminar una carrera profesional. Tras esta experiencia, el abanico de posibilidades de trabajo y el interés por la electrónica por parte de los autores han aumentado con creces.

También se ha llegado a la conclusión de que es necesario que la Facultad haga más hincapié en las asignaturas de la carrera de Ingeniería Informática relacionadas (Introducción a la Electrónica, Tecnología de Computadores, Ampliación de Tecnología de Computadores y Redes), tanto a nivel teórico como práctico.

GLOSARIO DE TÉRMINOS

API:
Application Programming Interface. Es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software.

ASCII:
American Standard Code for Information Interchange. Es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y en otras lenguas occidentales.

Asíncrono:
Hace referencia al suceso que no tiene lugar en total correspondencia temporal con otro suceso. Si se refiere a una comunicación asíncrona, ésta es en la que se realiza el envío de datos sin la sincronización de un reloj externo.

Asynchronous:
Ver Asíncrono.

Bit:
Dígito en el sistema binario.

Bitstream:
Flujo de bits.

Buffer:
Memoria usada para guardar temporalmente datos de entrada y salida de un dispositivo.

Bus:
Conjunto de conductores eléctricos en forma de pistas metálicas impresas sobre la placa base del computador, por donde circulan las señales que corresponden a los datos binarios del lenguaje máquina. Si el bus es de datos, este conectará dos dispositivos hardware.

Byte:
8 bits.

Canal:
Ver Puerto.

Chip:
Circuito integrado o pastilla en la que se encuentran todos o casi todos los componentes necesarios para que un ordenador pueda realizar alguna función.

Ciclo:
Un ciclo es la distancia temporal entre el principio y el final de una onda completa.

Circuito impreso:
Medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabados desde hojas de cobre laminadas sobre un sustrato no conductor.

Cristal:
En electrónica, es un circuito que es capaz de convertir la corriente continua en una corriente que varía de forma periódica en el tiempo (corriente periódica).

Frame:
ver Trama.

Depuración:
Proceso de mejora de un sistema hasta que realiza su función correctamente.

Drivers:
Programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz para usarlo.

EEPROM :
PROM eléctricamente borrable. Ver PROM.

Esquemático:
Es un diagrama, dibujo o boceto que detalla los elementos de un sistema.

FIFO:
First In, First Out. Protocolo que implementa una cola: lo que viene primero, se maneja primero, lo que viene segundo espera hasta que lo primero haya sido manejado, etc.

Flanco:
Transición del nivel bajo al alto (flanco de subida) o del nivel alto al bajo (flanco de bajada) de una señal.

Floorplanning:
En diseño de circuitos impresos, es el diagrama de emplazamiento de los footprints de los componentes.

Footprint:
En diseño de PCBs se refiere al mapa de las patas o pines de un dispositivo.

FPGA:
Dispositivo donde se pueden programar infinidad de diseños digitales distintos.

Hardware:
Es un término general usado para describir artefactos físicos de una tecnología.

IDE:
Integrated Device Electronics. Interfaz que controla los dispositivos de almacenamiento masivo de datos, como los discos duros.

Interfaz:
Interfaz software: Es la parte del programa informático que permite el flujo de información entre varias aplicaciones o entre el propio programa y el usuario.
Interfaz hardware: Es el puerto (circuito físico) a través del que se envían o reciben señales desde un sistema o subsistemas hacia otros. No existe un interfaz universal y la interconexión sólo es posible utilizando el mismo interfaz en origen y destino.

Jumper:
Conductor usado para conectar dos pines.

LED:
Light-emitting diode. Diodo emisor de luz.

Master:
Ver Master-Slave.

Master-Slave:
Protocolo de comunicación donde el dispositivo Master tiene control sobre el Slave, nunca al revés. La comunicación va en dirección del Master al Slave.

Paralelo:
En oposición a la comunicación en serie, la comunicación en paralelo se realiza enviando los datos en bloques de bits.

PCB:
Ver Circuito impreso.

PROM:
Programmable Read-Only Memory. Es una memoria digital donde el valor de cada bit depende del estado de un fusible (o antifusible), que puede ser quemado una sola vez.

Prototipo:
Ejemplar original o primer molde en que se fabrica un diseño.

Puerto:
Interfaz de comunicación físico entre un dispositivo y otro o varios dispositivos.

Registro:
Una memoria de alta velocidad y poca capacidad, integrada en el microprocesador.

Resonador:
Ver Cristal.

Serial:
Ver Serie.

Serie:
Una comunicación en serie es el proceso de enviar datos bit a bit.

Servidor:
Computadora que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

Síncrono:
Hace referencia al suceso que tiene lugar en correspondencia temporal con otro suceso. Si se refiere a una comunicación síncrona, ésta es en la que se realiza el envío de datos bajo la sincronización de un reloj externo.

Slave:
Ver Master-Slave.

SMD:
Surface Mount Technology. Es un método para construir circuitos electrónicos donde los componentes se montan directamente en la superficie del circuito impreso.

Software:
Son los programas y procedimientos requeridos para hacer capaz a una computadora de realizar una tarea específica.

SRAM:
Static Random Access Memory. Es un tipo de memoria semiconductora.

Switch:
Dispositivo para cambiar el curso (o flujo) de un circuito.

Trama:
Paquete de datos. En telecomunicaciones, especialmente en niveles bajos.

Trigger:
Condición fijada para provocar un evento.

UART:
Universal Asynchronous Receiver-Transmitter. Se trata de un circuito integrado que utilizan ciertos sistemas digitales basados en microprocesador, para convertir los datos en paralelo, que manda la CPU, en serie, con el fin de comunicarse con otro sistema externo

USB:
Universal Serial Bus. Es una interfaz que provee un estándar de bus serie para conectar dispositivos a un ordenador personal.

VGA:
Video Graphics Array. Es una norma de visualización de gráficos para ordenadores creada en 1987 por IBM.

Zumbador:
Aparato electrónico que produce un sonido continuo de un mismo tono pero que su construcción es más sencilla que la de un altavoz.

BIBLIOGRAFÍA

La mayoría de la documentación utilizada durante la realización del proyecto proviene de las páginas Web de los fabricantes de los componentes implicados y de los desarrolladores del software utilizado (www.xilinx.com, www.ftdichip.com y www.altium.com).

Referencias relacionadas con la FPGA:

- **DS001, "Spartan-II 2.5 V Family Complete Data Sheet":** Se trata de la hoja de especificaciones del fabricante (*datasheet*) de la familia de la FPGA Spartan-II.

- **XC4000E and XC4000X Series Field Programmable Gate Arrays:** Se trata de la hoja de especificaciones del fabricante (datasheet) de la familia de la FPGA XC4010XL utilizada en el prototipo desarrollado finalmente.

- **XAPP176, "Configuration and Readback of the Spartan-II and Spartan-IIE Families":** Notas de aplicación (*application notes*) que describen el proceso de configuración y el formato de los archivos .rbt y .bit de las FPGAs de las familias Spartan-II y Spartan-IIE.

Referencias relacionadas con el controlador USB:

- **DS2232C Version 1.5:** Se trata de la hoja de especificaciones del fabricante (*datasheet*) del dispositivo controlador USB FT2232C.

- **AN232B-01, "FT232BM/FT245BM Bit Bang Mode":** Notas de aplicación (*application notes*) que describen el modo de comunicación Bit Bang Mode de los dispositivos del fabricante FTDI que lo implementan, incluyendo código de ejemplo.

- **AN2232C-02, "Bit Mode Functions for the FT2232C":** Notas de aplicación (*application notes*) que describen la manera de establecer los distintos modos de comunicación posibles en un chip FT2232C.

- **D2XX Programmer's Guide:** Guía de programación con todos los tipos de datos y todas las funciones disponibles en la Interfaz de Programación de Aplicaciones D2XX de FTDI, necesaria para el acceso y manejo adecuado del dispositivo FT2232C.

Referencias relacionadas con el diseño y fabricación de placas de circuito impreso:

- **TU0117, "Getting Started with PCB Design":** Tutorial para aprender de forma rápida los conceptos básicos del manejo de la herramienta Altium DXP 2004, mediante la realización de un ejemplo en el que se diseña un pequeño circuito y se crea a partir de él una placa de circuito impreso.

Referencias con ejemplos de otras placas de prototipado:

- **Morph-IC Data Sheet Version 1.0:** Hoja de especificaciones de una placa de prototipado basada en una Altera que se comunica con el PC a través del puerto USB gracias a un dispositivo FT2232C.

- **Spartan-3 Starter Kit Board User Guide:** Guía de usuario de una placa de prototipado basada en una Spartan-3. Incluye abundante información acerca de los módulos adicionales que contiene (bancos de LEDs, switches, etc.) y de los protocolos y tramas involucrados en los puertos de expansión (VGA, PS/2, IDE…).

APÉNDICE

Fuentes

Funciones comunes a *EEPROM Loader* y *Loader*

AbrirPuerto

```
/* Método que lista todos los dispositivos conectados y abre el primero que
encuentra
cuya descripción termina con la letra A (canal A del chip FT2232C)
*/
private: void AbrirPuerto()
{
	// Variables locales para número de dispositivos encontrados y
	//descripciones
	DWORD numeroDispositivos;
	char descripcion[64];
	DWORD indice;

	// Variable local para el estado de las llamadas a la librería
	// FTD2XX.lib
	FT_STATUS estado;

	// Obtiene el número de dispositivos conectados
	estado = FT_ListDevices(&numeroDispositivos, NULL, FT_LIST_NUMBER_ONLY);

	// Si hay dispositivos conectados, los recorre
	if (estado == FT_OK && numeroDispositivos > 0)
	{
	  for(indice = 0; indice < numeroDispositivos; indice++)
		{
			// Lista el dispositivo actual por descripción
		estado = FT_ListDevices(indice, descripcion,
			FT_LIST_BY_INDEX|FT_OPEN_BY_DESCRIPTION);
		if (estado == FT_OK)
			{
				// Si el dispositivo corresponde al canal A,
				// finaliza
			if(Convert::ToString(descripcion)->EndsWith("A"))
			    break;
			    }
		 }
	}

	// Abre el dispositivo encontrado por su descripción
	estado = FT_OpenEx(descripcion, FT_OPEN_BY_DESCRIPTION, &handle);

	// Muestra si se ha concluido la llamada con éxito o no
	if(estado == FT_OK)
	  statusBarPanel2->Text = "Conexión establecida correctamente";
	else
		statusBarPanel2->Text = "Conexión no establecida";
}
```

CerrarPuerto

```
/* Método para cerrar los dispositivos que estén abiertos
*/
private: void CerrarPuerto()
{
	// Si hay un dispostivo abierto, lo cierra e inicializa su manejador
	if(handle) {
		FT_Close(handle);
		handle = NULL;
	}
}
```

Funciones de EEPROM Loader

ProgramarEEPROM

```
/* Método para programar la EEPROM y conseguir así que el protocolo usado por
el controlador del USB sea el 245 FIFO
*/
private: void ProgramarEEPROM()
{
	// Estado de las llamadas a la librería FTD2XX.lib
	FT_STATUS estado;

	// Estructura con la información de programación de la memoria EEPROM
	FT_PROGRAM_DATA ftData = {
		0x0,			// Header - must be 0x00000000
		0xffffffff,		// Header - must be 0xffffffff
		1,			// Header - FT_PROGRAM_DATA version
					//          0 = original
					//          1 = FT2232C extens
		0x0403,			//VID
		0x6010,			//PID
		"FTDI",			//Manufacturer string
		"FT",			//Manufacturer ID string
		"FPGA",			//description string
		"FT0001",		//serial number
		90,			//max current
		1,			//PNP
		0,			//Self Powered
		1,			//Remote wakeup
		//
		// Rev4 extensions
		//
		FALSE,			//Rev4
		FALSE,			//in endpoint type
		FALSE,			//out endpoint type
		FALSE,			//pulldown
		FALSE,			//serial number
		FALSE,			//USB Version select
		0,			//USB version
		//
		// FT2232C extensions
		//
		TRUE,			// non-zero if Rev5 chip, zero otherwise
		FALSE,			// TRUE if in endpoint is isochronous
		FALSE,			// TRUE if in endpoint is isochronous
		FALSE,			// TRUE if out endpoint is isochronous
		FALSE,			// TRUE if out endpoint is isochronous
		FALSE,			// TRUE if pull down enabled
		TRUE,			// TRUE if serial number to be used
		TRUE,			// TRUE if chip uses USBVersion
		0x0200,			// BCD (0x0200 => USB2)
```

```
            FALSE,              // TRUE if A interface is high current
            FALSE,              // TRUE if B interface is high current

            TRUE,               // TRUE if A interface is 245 FIFO
            FALSE,              // TRUE if A interface is 245 FIFO CPU target
            FALSE,              // TRUE if A interface is Fast serial
            FALSE,              // TRUE if A interface is to use VCP drivers

            TRUE,               // TRUE if B interface is 245 FIFO
            FALSE,              // TRUE if B interface is 245 FIFO CPU target
            FALSE,              // TRUE if B interface is Fast serial
            FALSE,              // TRUE if B interface is to use VCP drivers

            //
            // FT232R extensions (NUEVO)
            //
            FALSE, // Use External Oscillator
            FALSE, // High Drive I/Os
            FALSE, // Endpoint size
            FALSE, // non-zero if pull down enabled
            FALSE, // non-zero if serial number to be used
            FALSE, // non-zero if invert TXD
            FALSE, // non-zero if invert RXD
            FALSE, // non-zero if invert RTS
            FALSE, // non-zero if invert CTS
            FALSE, // non-zero if invert DTR
            FALSE, // non-zero if invert DSR
            FALSE, // non-zero if invert DCD
            FALSE, // non-zero if invert RI
            FALSE, // Cbus Mux control - Ignored for FT245R
            FALSE, // Cbus Mux control - Ignored for FT245R
            FALSE, // Cbus Mux control - Ignored for FT245R
            FALSE, // Cbus Mux control - Ignored for FT245R
            FALSE, // Cbus Mux control - Ignored for FT245R
            FALSE, // non-zero if using VCP drivers

    };

    try {
            // Lo transforma en un array de caracteres
            char fabricante[128];
            for(int i = 0; i < textBoxFabricante->Text->Length; i++)
                    fabricante[i] = textBoxFabricante->Text->Chars[i];
            ftData.Manufacturer = fabricante;

            // Lo transforma en un array de caracteres
            char fabricanteID[128];
            for(int i = 0; i < textBoxFabricanteID->Text->Length; i++)
                    fabricanteID[i] = textBoxFabricanteID->Text->Chars[i];
            ftData.ManufacturerId = fabricanteID;

            // Lo transforma en un array de caracteres
            char descripcion[128];
            for(int i = 0; i < textBoxDescripcion->Text->Length; i++)
                    descripcion[i] = textBoxDescripcion->Text->Chars[i];
            ftData.Description = descripcion;

            // Lo transforma en un array de caracteres
            char serial[128];
            for(int i = 0; i < textBoxSerie->Text->Length; i++)
                    serial[i] = textBoxSerie->Text->Chars[i];
            ftData.SerialNumber = serial;

            //      Se programa la EEPROM con la estructura anterior
            estado = FT_EE_Program(handle, &ftData);

            // Se informa del éxito o no de la llamada a la función de
```

```
            //programación
            if (estado == FT_OK)
                  MessageBox::Show(this,new String("EEPROM programada con
                  éxito"),new String("Información"),MessageBoxButtons::OK,
                  MessageBoxIcon::Information);
            else
                  MessageBox::Show(this,new String("Ocurrió un error al
                  programar la EEPROM"),new String("Error"),
                  MessageBoxButtons::OK, MessageBoxIcon::Error);
            }
            catch(...)
            {
                  MessageBox::Show(this,new String("Valor no válido en los
                  campos de texto"),new String("Advertencia"),
                  MessageBoxButtons::OK, MessageBoxIcon::Warning);
            };
}
```

BorrarEEPROM

```
private: void BorrarEEPROM()
{
      // Estado de las llamadas a la librería FTD2XX.lib
      FT_STATUS estado;

      // Borra el contenido de la EEPROM
      estado = FT_EraseEE(handle);

      // Informa del resultado de la llamada a la anterior función
      if (estado == FT_OK)
            MessageBox::Show(this,new String("EEPROM borrada con éxito"),new
            String("Información"),MessageBoxButtons::OK,
            MessageBoxIcon::Information);
      else
            MessageBox::Show(this,new String("Ocurrió un error al borrar la
            EEPROM"),new String("Error"),MessageBoxButtons::OK,
            MessageBoxIcon::Error);
}
```

LeerEEPROM

```
private: void LeerEEPROM()
{
      // Estado de las llamadas a la librería FTD2XX.lib
      FT_STATUS estado;

      FT_PROGRAM_DATA ftData;
      char vid[128];
      char pid[128];
      char fabricante[128];
      char fabricanteID[128];
      char descripcion[128];
      char serial[128];

      ftData.Signature1 = 0x00000000;
      ftData.Signature2 = 0xffffffff;
      ftData.VendorId = 403;
      ftData.ProductId = 6010;
      ftData.Manufacturer = fabricante;
      ftData.ManufacturerId = fabricanteID;
      ftData.Description = descripcion;
      ftData.SerialNumber = serial;
```

```
        estado = FT_EE_Read(handle, &ftData);

        if (estado == FT_OK)
        {
                textBoxFabricante->Text = ftData.Manufacturer;
                textBoxFabricanteID->Text = ftData.ManufacturerId;
                textBoxDescripcion->Text = ftData.Description;
                textBoxSerie->Text = ftData.SerialNumber;
        }
        else
        {
                MessageBox::Show(this,new String("No se pudo leer el contenido de
                la EEPROM"),new String("Error"),MessageBoxButtons::OK,
                MessageBoxIcon::Error);
    }
}
```

Funciones de Loader

CargarFPGA

```
/* Método que carga el diseño realizado en Xilinx mediante una configuración
inicial
de la FPGA y un volcado posterior del archivo .RBT serializado
*/
private: void CargarFPGA()
{
        // Variable local para almacenar el estado de la llamada a la librería
        FT_STATUS estado;

        if(handle == NULL)
        {
                statusBarPanel2->Text = "Conexión no establecida";
                MessageBox::Show(this,new String("Conexión no establecida"), new
                String("Error"),MessageBoxButtons::OK, MessageBoxIcon::Error);
                HabilitarBotones();
                this->DetenerHilo();
        }
        else
        {
                DeshabilitarBotones();
                // Establece el protocolo del USB (Asynchronous Bit Bang Mode)
                estado = FT_SetBitMode(handle,0xff,0x1);

                // Comprueba que se ha establecido correctamente el protocolo
                if(estado == FT_OK)
                {
                        estado = ConfiguracionInicial();
                        if(estado == FT_OK)
                        {
                                estado = SerializarArchivoYEnviar();
                                if(estado == FT_OK)
                                        ComprobacionFinal();
                        }
                }
                else
                {
                        MessageBox::Show(this,new String("Error al tratar de
                        establecer el protocolo USB"),new String("Error"),
                        MessageBoxButtons::OK, MessageBoxIcon::Error);
                        HabilitarBotones();
                        this->DetenerHilo();
                }
        }
}
```

ConfiguraciónInicial

```
/* Método que introduce las tramas de configuración inicial en la FPGA
*/
private: int ConfiguracionInicial()
{
	// Estado tras las llamadas a la librería FTD2XX.lib
	FT_STATUS estado;

	// Bytes a transmitir y transmitidos realmente
	DWORD TxBytes = 128;
	DWORD BytesTransmitidos;

	// Búffer de transmisión (esto es nuevo)
	const UInt32 TAM_BUFFER_TX = 128;
	char TxBuffer[TAM_BUFFER_TX];

	// Inicializa el búffer de transmisión
	for(int i=0;i<65 ;i++) //0..64 primer paquete USB a uno
		TxBuffer[i] = 0xFFFF;

	for(int i=65;i<127;i++)
		TxBuffer[i] = INIT|CCLK|DIN; //65..128 segundo paquete usb con la
					     //señal /PROGRAM en baja
	TxBuffer[127] = 0xFFFF;

	// Escribe la trama en la FPGA
	estado = FT_Write(handle, TxBuffer, TxBytes, &BytesTransmitidos);


	// Comprueba que se ha introducido bien la trama
	if(estado != FT_OK || BytesTransmitidos != TxBytes)
	{
		MessageBox::Show(this,new String("Error en la transmisión de la
		trama"),new String("Error"),MessageBoxButtons::OK,
		MessageBoxIcon::Error);
		// Detiene el hilo de ejecución
		DetenerHilo();
		return estado;
	}
	else
	{
		// Retardo de 3 milisegundo para darle tiempo a la FPGA para que
		//limpie la memoria de configuración
		Thread::Sleep(3);
		return FT_OK;
	}
}
```

SerializarArchivoYEnviar

```
/* Método que se encarga de serializar el archivo .RBT generado por la
herramienta Xilinx Foundations y que contiene el la información necesaria para
la configuración de la FPGA. El contenido de este archivo se va introduciendo
por los pines de la FPGA en forma de caracteres que representan las tramas que
van por las líneas CCLK y DIN. Este método usa un búffer de 128 bytes (el
tamaño del búffer TX del chip)
*/
private: int SerializarArchivoYEnviar()
{
	try {

		// Búffer de lectura y cantidad de bloques a leer en cada llamada
		// a fopen()
		const UInt32 TAM_BUFFER = 64;
```

```
char buffer[TAM_BUFFER];
int n_read, caracteres_a_enviar;

// Inicializa el búffer de lectura del fichero
for(int i=0;i<TAM_BUFFER;i++)
	buffer[i] = 0;

// Número de línea que está siendo procesada
int linea = 0;

// Número de bits que forman la trama del archivo .RBT
UInt32 nbits = 0;

// Número de bits procesados
UInt32 leidos = 0;

// Estado tras las llamadas a la librería FTD2XX.lib
FT_STATUS estado;

// Bytes a transmitir y transmitidos realmente
DWORD TxBytes = 128;
DWORD BytesTransmitidos;

// Búffer de transmisión
const UInt32 TAM_BUFFER_TX = 128;
char TxBuffer[TAM_BUFFER_TX];

// Comienza a procesar el archivo .RBT
if(fichero != NULL)
{

	// Lee las primeras líneas del archivo, descartándolas
	do {
		n_read = fread(buffer, 1, 1, fichero);
		if(buffer[0] =='\n')
	 linea++;
	}
	while(linea != 6);

	// Si se está tratando la línea que indica el número de
	// bits
	if (linea == 6)
	{
		// Buffer para almacenar la cifra que indica el
		// tamaño de la trama
		char buffer2[128];

		// Índice y centinela de control
		int pos = 0;
		bool cifra = false;

		// Extrae el número de bits de la línea actual
		do
		{
			n_read = fread(buffer, 1, 1, fichero);
			if(cifra)
				buffer2[pos++] = buffer[0];
			else
			{
				if(buffer[0] == ':')
					cifra = true;
			}
		}
		while(buffer[0] != '\n');

		// Traduce la cadena leída a un entero
		nbits = Convert::ToInt32(buffer2);
```

```
                        // Actualiza la barra de progreso
                        progressBar1->Step = 64;
                        progressBar1->Maximum = nbits;

                        // Pasa a las líneas de las tramas de bits
                        linea = linea++;

                }

        // Si se trata del cuerpo del archivo .RBT (trama)
        do
        {
                n_read = fread(buffer, TAM_BUFFER, 1, fichero);
                if(linea >= 7)
                {
                        int posBufferTX = 0;
                        for(int posBuffer=0;posBuffer<TAM_BUFFER;posBuffer++)
                        {
                                // Genera las tramas con la señal de reloj añadida y
                                // las envía
                                if(buffer[posBuffer] == '1')
                                {
                                        // Envía la trama (reloj en baja)
                                        TxBuffer[posBufferTX++] = DIN|PROGRAM;
                                        // Envía la trama (reloj en alta)
                                        TxBuffer[posBufferTX++] = DIN|CCLK|PROGRAM;
                                }
                                else if(buffer[posBuffer] == '0')
                                {
                                        // Envía la trama (reloj en baja)
                                        TxBuffer[posBufferTX++] = PROGRAM;
                                        // Envía la trama (reloj en alta)
                                        TxBuffer[posBufferTX++] = CCLK|PROGRAM;
                                }
                                else
                                {
                                        // Si es fin de línea, no se hace nada
                                }
                        }
                        estado = FT_Write(handle, TxBuffer, posBufferTX,
                                        &BytesTransmitidos);

                        // Comprueba que la trama se ha enviado correctamente
                        if(estado != FT_OK || posBufferTX != BytesTransmitidos)
                        {
                                n_read = 0;
                                MessageBox::Show(this,new String("Error en la
                                transmisión de la trama"),new String("Error"),
                                MessageBoxButtons::OK, MessageBoxIcon::Error);
                                // Detiene el hilo de ejecución
                                DetenerHilo();
                                return estado;
                        }
                        else
                        {
                                // Actualiza la barra de progreso y su label
                                // asociado
                                if (leidos < nbits)
                                        leidos = leidos + TAM_BUFFER;
                                else
                                        leidos = nbits;
                                        labelProgreso->Text = String::Concat(new
                                        String("Transmitidos "),
                                        String::Concat(Convert::ToString((UInt32)leido
                                        s), String::Concat(new String(" de "),
                                        String::Concat(Convert::ToString((UInt32)nbits
                                        ),new String(" bits"))))));
                                        progressBar1->PerformStep();
```

```
                }
            }
        }
        while(n_read != 0);

        // Se comprueba el indicador de fin de fichero del flujo del fichero
                if(feof(fichero) == 0)
                {
                    MessageBox::Show(this,new String("No se llegó al
                    final del archivo .RBT"),new String("Error"),
                    MessageBoxButtons::OK, MessageBoxIcon::Error);
                    // Detiene el hilo de ejecución
                    DetenerHilo();
                    return estado;
                }
                else
                {
                    return FT_OK;
                }
            }
        }
        catch(...) {
            MessageBox::Show(this,new String("Archivo .RBT corrupto"),new
            String("Error"),MessageBoxButtons::OK, MessageBoxIcon::Error);
            this->DetenerHilo();
            return FT_IO_ERROR;
        }
}
```

ComprobaciónFinal

```
/* Método que comprueba que la señal DONE de la FPGA está en alta tras una
transmisión
de tramas
*/
private: void ComprobacionFinal()
{
    // Estado tras las llamadas a la librería FTD2XX.lib
       FT_STATUS estado;

       // Bytes a leer y leídos realmente
       DWORD RxBytes = 1;
       DWORD BytesLeidos;

       // Búffer de lectura
       char RxBuffer[1];

       // Inicializa la máscara a cero (todas las señales en baja)
       RxBuffer[0] = 0;
       estado = FT_Purge(handle,FT_PURGE_RX);

       if(estado != FT_OK)
       {
            MessageBox::Show(this,new String("No se pudo determinar si la
            trama se transmitió correctamente"),new
            String("Advertencia"),MessageBoxButtons::OK,
            MessageBoxIcon::Warning);
       }
       else
       {

            // Lee la trama en la FPGA
            estado = FT_Read(handle, RxBuffer, RxBytes, &BytesLeidos);

            // Comprueba que se ha leido bien la trama
            if(estado != FT_OK || BytesLeidos != RxBytes)
            {
```

```
                        MessageBox::Show(this,new String("No se pudo determinar si
                        la trama se transmitió correctamente"),new
                        String("Advertencia"), MessageBoxButtons::OK,
                        MessageBoxIcon::Warning);
                }
                else
                {
                        if(RxBuffer[0] >= 16)
                                MessageBox::Show(this,new String("La trama se
                                transmitió correctamente"),new
                                String("Información"),MessageBoxButtons::OK,
                                MessageBoxIcon::Information);
                        else
                                MessageBox::Show(this,new String("Error en la
                                transmisión de la trama"),new
                                String("Error"),MessageBoxButtons::OK,
                                MessageBoxIcon::Error);
                }
        }
        // Detiene el hilo de ejecución
        DetenerHilo();
}
```