Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Tecnologías Industriales

Tarjeta para configuración y comunicación con una FPGA Spartan 6 usando el controlador USB FT2232H

Autor:

Guillermo Sagrado Salvador

Tutor:

Santiago de Pablo Gómez

Valladolid, noviembre 2019

Resumen

El actual proyecto presenta una solución referida a la necesidad de configurar y comunicarse con una FPGA del modelo Spartan 6 del fabricante Xilinx a través de un dispositivo intermedio como es un controlador USB.

Se diseña una tarjeta que contenga las conexiones entre ambos dispositivos y que se conecta al host a través de USB. El trabajo abarca desde la búsqueda de las conexiones entre dispositivos hasta la fabricación de la tarjeta.

Se desarrollará también el interfaz de configuración JTAG para la FPGA Spartan 6 haciendo pruebas con el fichero de código tanto a través de cableado como con la tarjeta de este trabajo ya fabricada.

Palabras clave

Tarjeta, diseño, Spartan 6, configuración JTAG, USB

Abstract

The following project give us a solution so that we can configurate and comunicate with a FPGA Spartan 6 model by the electronic manufacturer Xilinx through an intermediate device which is an USB controller.

A carrier board which contains all the connections between both devices will be planned in this project. This work is not only about the connections research but the product manufacture.

Another of the tasks that have been carried out through this project has been the development of the JTAG configuration interface for testing purpose, executing the bitstream folder, first of all by wire and finally with the manufactured product.

Keywords

Carrier board, design Spartan 6, JTAG configuration, USB

Tabla de contenido

Índice de tablas

Tabla de ilustraciones

1. Introducción y objetivos generales

1.1 Marco general del proyecto

Desde principios del siglo XXI hasta ahora el uso de tarjetas comerciales FPGA´s (Field-Programmable Gate Array) ha crecido exponencialmente en el desarrollo tanto de prototipos como de desarrollos finales de productos industriales. Algunas de las características que han contribuido al éxito de estas tarjetas comerciales son que existe un gran abanico de tarjetas a bajo coste que incluyen una FPGA capaz de cubrir las necesidades de proyectos muy dispares y que dichas tarjetas vienen con la circuitería necesaria para utilizarlas al instante, por lo que evitamos desarrollos hardware que no aportan un valor añadido al proyecto pero que lo penalizan en costes y sobre todo en tiempo de desarrollo.

La principal ventaja que tienen las FPGA´s es que son reprogramables, si bien es cierto que hay ciertos fabricantes cuentan con modelos que solo pueden ser reprogramados una vez. Uno de los inconvenientes del uso de estas tarjetas de lógica programable es encontrar un diseño comercial que satisfaga las necesidades de nuestro proyecto en cuanto a interfaces disponibles y los conectores que permiten el acceso a los mismos se refiere.

Este problema se pone de manifiesto especialmente en los periféricos de comunicación y de programación. Con el paso de los años hemos pasado de buses de comunicación serie RS232 a los actuales buses de comunicación de USB, el cual tiene unas tasas de transferencia más elevadas. Sin embargo, también es más complejo que los antiguos buses de comunicación serie por lo que su implementación y control en los dispositivos de lógica programable se hace más complicado y consuma un alto número de recursos.

En cuanto al interfaz de programación ocurre algo parecido que con el de comunicaciones. El modelo de tarjeta que usemos determinará el interfaz de programación que debemos usar, en este caso tenemos dos opciones, a través de un dispositivo intermedio, que permita el acceso desde un puerto USB a la cadena de JTAG o utilizando un conector proporcionado por el fabricante.

En este trabajo de fin de grado se dará una solución para la conexión de un dispositivo intermedio entre el PC y la FPGA tanto para el interfaz de comunicaciones como para el de programación, así como el diseño y funcionamiento de una PCB que albergue las conexiones necesarias entre ambos dispositivos.

Los dispositivos elegidos para este trabajo serán: FPGA Spartan 6 del fabricante Xilinx, su carrier board TE0303 del mismo fabricante y el controlador de USB FT-2232H del fabricante FTDI.

1.2 Objetivos generales del proyecto

En este proyecto podemos distinguir dos objetivos principales.

El primer objetivo de este trabajo es el diseño y fabricación de una tarjeta que albergue las conexiones necesarias entre los dos dispositivos anteriormente descritos tanto para el interfaz de datos como para el de programación. Este segundo objetivo trae consigo la adquisición de una serie de competencias como son:

- Conocimiento del software OrCad Cadence en la versión 16.6 para el diseño y posterior generación de planos y Art Work para la fabricación de la tarjeta.
- Diseño de un primer prototipo teniendo únicamente la funcionalidad de la tarjeta.
- Proceso de optimización y reducción de costes.
- Finalización del proceso con la fabricación de la tarjeta.

El segundo objetivo que debemos destacar es el desarrollo del interfaz de configuración JTAG para la FPGA Spartan 6 y la realización de las pruebas necesarias para la validación tanto del producto fabricado anteriormente como del fichero que contiene la programación.

Este trabajo de fin de grado es únicamente una primera aproximación en el intento de aportar soluciones para las distintas posibilidades que existen para los dos interfaces, el de comunicación y el de programación. No es en ningún caso la única posibilidad de conexión, simplemente ha sido escogida en función a unos objetivos de simplicidad y funcionalidad para posteriores mejoras o ampliaciones.

1.3 Estructura de la memoria

La presente memoria se encuentra dividida en cuatro partes diferenciadas.

La primera parte, correspondiente a los capítulos primero y segundo de la memoria, constará de un pequeño planteamiento de los objetivos que se marcaron para este proyecto y demás consideraciones generales y, en segunda instancia, de los recursos con los que contaremos en este trabajo, dividiéndolos en hardware y software.

Una vez hayamos dejado claros tanto el planteamiento como el punto de partida del proyecto pasaremos en la segunda parte de la memoria a detallar todo lo relativo a las conexiones entre el dispositivo de lógica programable y el controlador del USB. Este apartado, que corresponderá con el capítulo 3 del trabajo, se centrará también en la elaboración de los esquemáticos del trabajo con el programa OrCad Capture CIS. En el capítulo 4, también perteneciente a esta parte de la memoria, encontraremos el diseño de la tarjeta en PCB Editor, dividiéndose a su vez en varios apartados como diseño, layout o routing.

En la tercera parte, que equivale al capítulo 5 veremos todo lo relativo al interfaz de programación JTAG, tanto su funcionamiento general como la implementación en este trabajo.

Finalmente, para concluir la memoria, se presentan las conclusiones alcanzadas, así como las referencias consultadas.

2. Herramientas Hardware/Software utilizados

2.1 FTDI 2232 Chip

Hay multitud de controladores de USB en el mercado, algunos de los más comunes son por ejemplo el controlador ISPI1362 de USB de Philips o el CY7C68013 de Cypress. En este trabajo utilizamos este debido a que nuestro punto de partida es que en proyectos anteriores a este ya se desarrolló el interfaz de comunicación y de programación de este controlador de USB para un dispositivo de lógica programable del fabricante Xilinx, por lo que será lógico seguir las pautas ya marcadas.

Algunas de las razones que llevaron a elegir este controlador de USB en detrimento de los competidores de mercado es que es un chip económico y de fácil disponibilidad ya que lo podemos adquirir a través de los distribuidores comunes de productos electrónicos. A parte de esto, tiene un doble puerto que nos permite implementar los interfaces de comunicación y de programación y es el componente utilizado en algunas placas de evaluación de Xilinx, por lo que se ajusta bastante a los requerimientos.

En este proyecto no vamos a trabajar en sí con este controlador de USB puesto que la implementación de los interfaces ya se realizó en el trabajo comentado anteriormente, pero si que es importante conocer tanto la arquitectura como los datos técnicos de este dispositivo debido a que se encuentra dentro de nuestro diseño y deben ser tenidos en cuenta por alguien que en un futuro use nuestra tarjeta.

2.1.1 Consideraciones generales

El FT2232H es un controlador de USB de 5ª generación, diseñado y fabricado por la compañía FTDI, que como hemos visto antes dispone de dos controladores independientes de propósito general (UART y FIFO) en un mismo chip, de forma que se pueden obtener dos canales de E/S de un único puerto USB sin necesidad de reconfigurar el controlador ni de multiplexar internamente las señales.

Otra característica a tener en cuenta en este dispositivo es la posibilidad de comunicación mediante el protocolo USB 2.0 (480Mbits/s) aunque también es compatible con la versión anterior 1.0 (12Mb/s). La alimentación del circuito integrado deberá ser a 3.3V aunque admite también 5V. El core, sin embargo, será alimentado a 1.8V lo que no es un problema para nosotros ya que esta alimentación está controlada con un regulador interno, a partir de la tensión de alimentación. El rango de temperatura se corresponde con el rango industrial extendido (-40ºC a 85ºC).

El dispositivo se encuentra disponible en encapsulados del tipo LQFP (Low Profile Quad Flat Pack) o QFN (Quad Flat Pack) de 64 pines libres de plomo.

2.1.2 Arquitectura Interna

Internamente, se distinguen varios modos funcionales:

- Dual Multi-Purpose UART/FIFO Controllers: Controlador de propósito general encargado de gestionar las transferencias entre los búferes de transmisión y recepción, TX y RX respectivamente, y los registros de transmisión y recepción UART/FIFO.
- USB Protocol engine and FIFO control. Es el bloque encargado de implementar el protocolo USB para establecer la comunicación con el host.
- Búfer de transmisión TX: Los datos recibidos por el puerto se almacenan en este búfer configurable de doble puerto antes de ser enviados al registro de transmisión del controlador.
- Búfer de recepción RX: Es exactamente igual que el de transmisión, pero en sentido contrario, sirve de búfer entre los datos recibidos del controlador UART/FIFO hasta ser enviados hacia el puerto USB, por una petición del host.
- Generador de RESET, el dispositivo implementa su propio circuito de reset. El pin RESET# puede ser usado de manera adicional, en caso de no hacerlo debe estar conectado a VCCIO.
- Generador de Baude Rate. Este bloque permite multiplicar x6 y x10 el reloj de entrada de la UART mediante la utilización de un reloj de referencia de 120 MHz.
- Regulador de 1.8V ya descrito anteriormente, encargado de transformar los 3.3V de la alimentación a los 1.8V a los que necesita ser alimentado el core.
- Interfaz EEPROM: Constituye una interfaz de tres pines con una memoria EEPROM externa de la clase 93C46, 93C56 o 93C66 que servirá para guardar datos de configuración del dispositivo y establecer el modo de configuración.

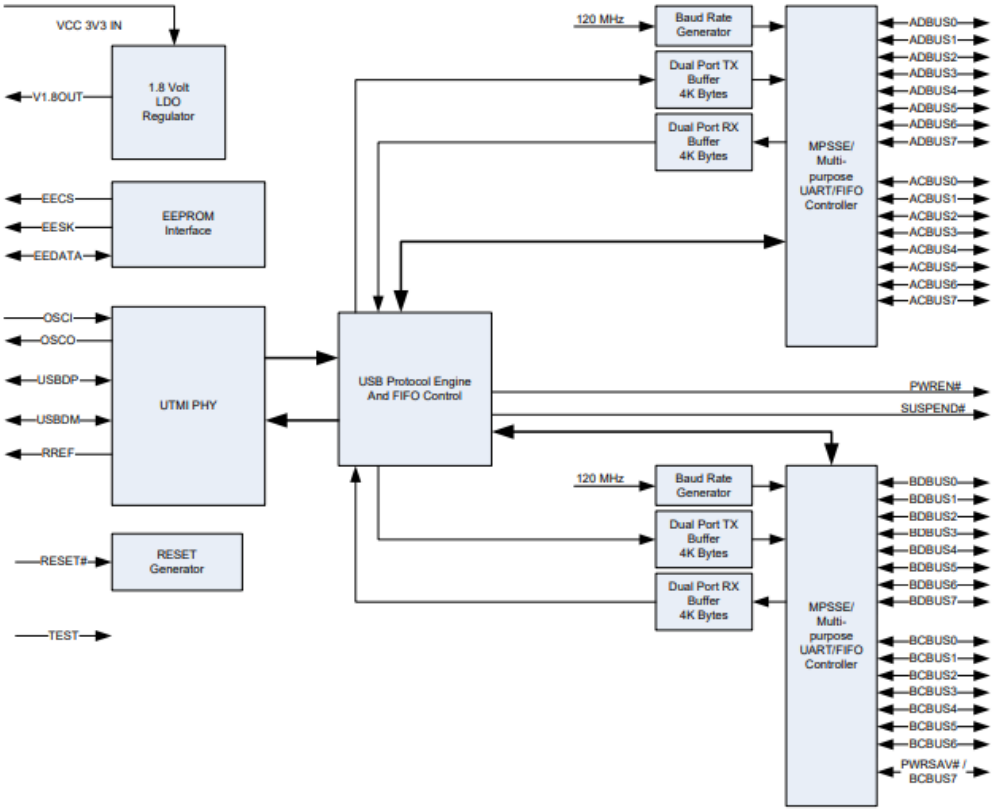

Figura 1 Arquitectura interna del dispositivo FT2232H.

2.1.3 Modos de funcionamiento

Existen muchos modos de comunicación en este dispositivo además de los ya descritos hasta ahora, estos son algunos otros:

- La interfaz UART (Universal Asynchronous Receiver-Transmitter) permite el uso del controlador de USB con dispositivos que sigan protocolos serie como RS-232, el RS422 o RS485, como ocurre en una gran cantidad de módems actuales.
- El modo 245 FIFO implementa un bus de comunicaciones de propósito general mucho más rápido que el de una UART. A su vez este modo puede ser implementado de manera síncrona o asíncrona.
- El modo MPSEE permite que el dispositivo FT2232H pueda comunicarse eficientemente con protocolos síncronos serie, como JTAG.

El resto de los modos de funcionamiento disponibles en el dispositivo son el Bit Bang Mode, tanto síncrono como asíncrono, el Fast Opto-Isolated Serial Interface y el MCU Host Bus Emulation Mode. Estos últimos modos de funcionamiento presentados no serán usados en este trabajo.

2.1.4 Configuración por EEPROM

La presencia o no de una EEPROM externa al chip FT2232H es clave en el funcionamiento de este. Si no conectamos un EEPROM, o si esta está vacía, el controlador USB adoptara el protocolo UART 323 Mode, que es el protocolo por defecto. En este trabajo vamos a utilizar otro protocolo, que ya hemos descrito anteriormente, como es el 245 FIFO. Para ello como se ha comentado anteriormente es imprescindible conectar el dispositivo a una EEPROM y especificar en la memoria el protocolo deseado.

Debemos tener en cuenta que este modo de funcionamiento tiene dos posibilidades la síncrona y la asíncrona, si utilizamos la primera el segundo canal queda desactivado estando únicamente operativo el puerto A, debido a que este modo utiliza todos los recursos de memoria del integrado.

2.1.5 Modo 245 FIFO

Este modo de funcionamiento será el utilizado en este trabajo, debido que disponemos de una EEPROM en la tarjeta del chip FT2232H.

Para comenzar veremos un desglose de las diferentes señales con las que cuenta este bus de comunicación:

- Señales de control:
  - Tenemos una señal de reloj CLKOUT con una frecuencia constante de 60MHz.
  - Una señal para pedir la lectura/salida RXF#, es activa a nivel alto. Cuando RFX# está a nivel bajo quiere decir que todavía no existen datos leídos. El RFX es considerado como buffer de recepción no vacío todavía. Cuando permanece a nivel alto, el último byte de datos en el buffer permanece en el bus de datos y no cambia.
  - Tenemos también una petición de escritura/salida TXE# que al igual que la de lectura es activa a nivel alto. Cuando se encuentra a nivel bajo quiere decir que existe espacio en el buffer interno de transmisión para escribir los datos en el controlador del USB. Sin embargo, cuando está a nivel alto nos dice que está llena y no pide operación de escritura.
  - #OE, señal que habilita los datos de entrada. Tiene que estar forzada a uno por el slave, antes de que el dato vaya a ser leído por el master.
  - RD#, señal mandada por el slave para confirmar una operación de lectura.
  - WR#, similar a la señal anterior, pero confirma una operación de escritura.
- 8 pines bidireccionales para datos y direcciones que van desde la señal D0 hasta la señal D7.

En los dos siguientes apartados vamos a dar unas pequeñas pinceladas sobre el modo de funcionamiento FIFO, tanto el síncrono como el asíncrono. No vamos a entrar en detalles de su funcionamiento teórico, pero sí que mostraremos los diagramas de tiempos y las especificaciones temporales. Ambos, diagrama y tabla de tiempos, los daremos tanto para lectura como escritura y para FIFO síncrona y FIFO asíncrona.

Mostramos el diagrama y la tabla de tiempos porque son datos a tener en cuenta para alguien que vaya a trabajar con la tarjeta que se persigue diseñar y fabricar en este trabajo.

2.1.5.1 FIFO síncrona

Por hacer una pequeña introducción del modo de funcionamiento síncrono diremos que se implementa un bus de comunicaciones serie, en el que las operaciones de lectura y de escritura se realizarán en el flanco de subida del reloj. Los datos serán síncronos con el reloj generado a 60 MHz, como comentamos anteriormente en la exposición de las señales que teníamos.

Podemos ver en la figura 2 el diagrama de tiempos de lectura síncrona y en la figura 3 la tabla de especificaciones temporales.

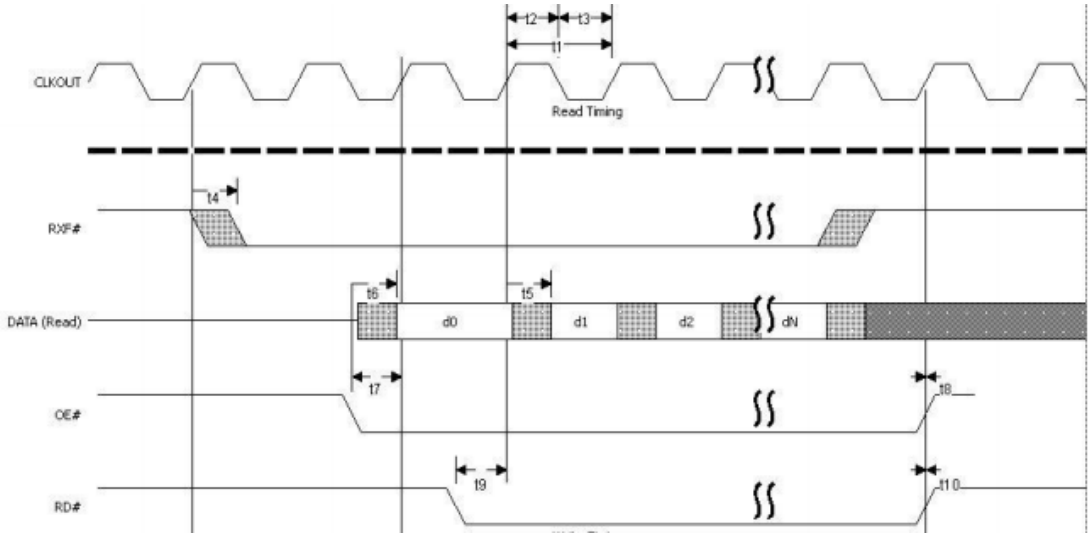

Figura 2 Diagrama de tiempo de lectura síncrona

Name	Minimum	Typical	Maximum	Units	Description
t1		16.67	16.67	ns	CLKOUT periodo
t2	7.5	8.33	9.17	ns	CLKOUT periodo a nivel alto
t3	7.5	8.33	9.17	ns	CLKOUT periodo a nivel bajo
t4	1		7.15	ns	CLKOUT a RXF#
t5	1		7.15	ns	CLKOUT para leer datos validos
t6	1		7.15	ns	OE# para leer datos validos
t7	8		16.67	ns	OE# tiempo de setup
t8	8			ns	OE# tiempo de hold
t9	8		16.67	ns	RD# tiempo de setup a CLKOUT
t10	0			ns	RD# tiempode hold

Tabla 1: Especificaciones temporales de lectura síncrona

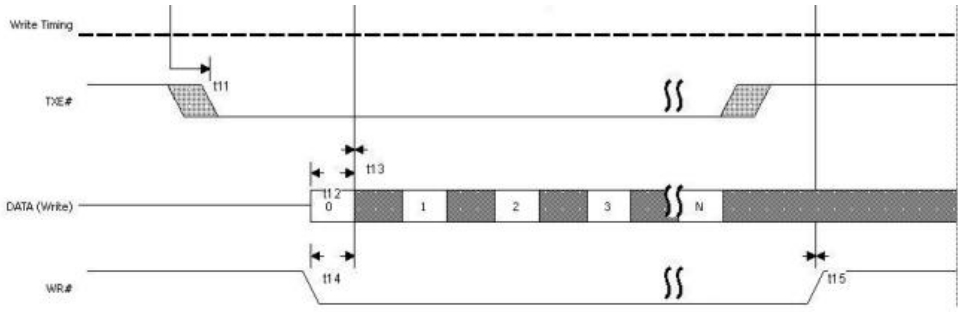

Figura 3: Diagrama de tiempo de escritura síncrona

t11	1		7.15	ns	CLKOUT TO TXE#
t12	8		16.67	ns	Write DATA setup time
t13	0			ns	Write DATA hold time
t14	8		16.67	ns	WR# setup time to CLKOUT (WR# low after TXE# low)
t15	0			ns	WR# hold time

Tabla 2: Especificaciones temporales de escritura síncrona

2.1.5.2 FIFO asíncrona

Este modo de funcionamiento es similar al anterior con la única diferencia de que los datos serán leídos o escritos en los flancos de señales de control RD# y WR# respectivamente.

Primero veremos el diagrama y la tabla de tiempos para la lectura asíncrona.

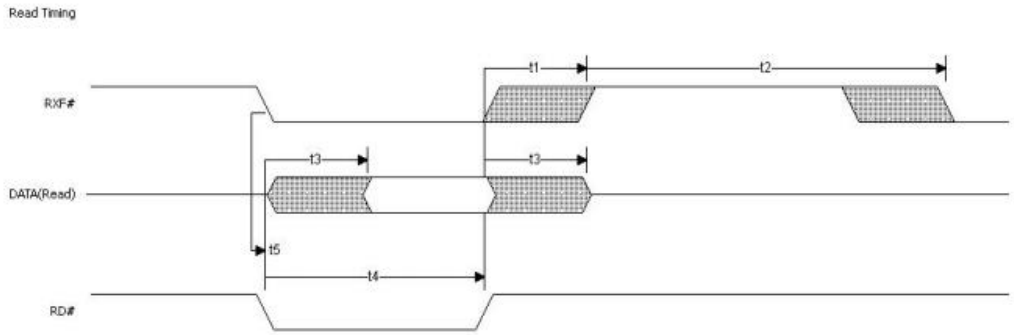

Figura 4: Diagrama de tiempo de lectura asíncrona

Name	Minimum	Typical	Maximum	Units	Description
t1	1		14	ns	RD# inactive to RX#
t2	49			ns	RXF# inactive after RD# cycle
t3	1		14	ns	RD# to DATA
t4	30			ns	RD# active pulse width
t5	0			ns	RD# active after RXF#

Tabla 3: Especificaciones temporales de lectura asíncrona

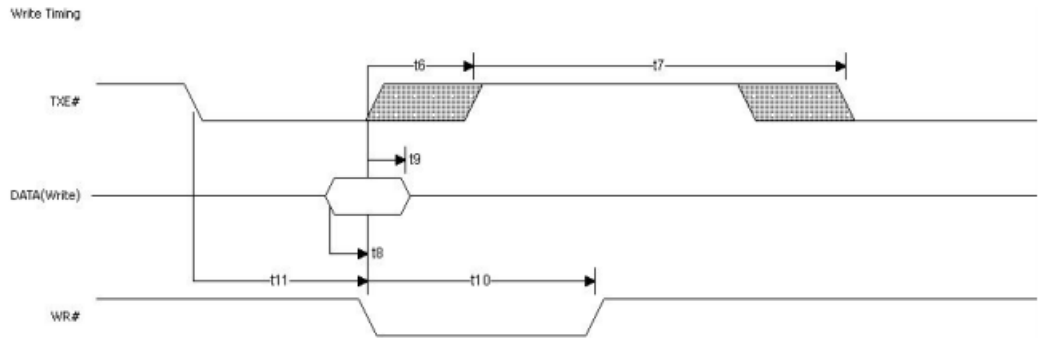

Figura 5: Diagrama de tiempo de escritura asíncrona

t6	1		14	ns	WR# active to TXE# inactive
t7	49			ns	TXE# inactive after WR# cycle
t8	5			ns	DATA to WR# active setup time
t9	5			ns	DATA hold time after WR# inactive
t10	30			ns	WR# active pulse width
t11	0			ns	WR# active after TXE#

Tabla 4: Especificaciones temporales de escritura asíncrona

2.1.6 MPSSE

El modo MPSEE ha sido deseñado para proveer al circuito del FT2232H de un interfaz eficiente con protocolos como JTAG, I2C y SPI.

El interfaz MPSSE es un interfaz flexible y puede ser configurado para cualquier protocolo asíncrono. Esta implementado en ambos canales del chip.

Existe la posibilidad de manejar un conjunto de GPIOS´s para implementar las señales de control que sean necesarias.

A continuación, podemos ver una tabla donde encontramos la correspondencia entre las señales del MPSSE y los interfaces serie más comunes. Para nuestro trabajo, como hemos dicho anteriormente, vamos a utilizar un interfaz JTAG, por lo que será el que nos interese.

Señales MPSSE	SPI	I2C	JTAG
Data Out (TDI/DO)	MOSI	SDA	TDI
Data In (TDO/DI)	MISO	SDA	TDO
Clock (TCK/CK)	SCLK	SCK	TCK
Chip Select (TMS/CS)	CS	Unused	TMS
GPIOL0	<disponible>	<disponible>	<disponible>
GPIOL1	WAIT or STOPCLK	WAIT or STOPCLK	WAIT or STOPCLK
GPIOL2	<disponible>	<disponible>	<disponible>
GPIOL3	<disponible>	<disponible>	RTCK
GPIOH0	<disponible>	<disponible>	<disponible>
GPIOH1	<disponible>	<disponible>	<disponible>
GPIOH2	<disponible>	<disponible>	<disponible>
GPIOH3	<disponible>	<disponible>	<disponible>
GPIOH4	<disponible>	<disponible>	<disponible>
GPIOH5	<disponible>	<disponible>	<disponible>
GPIOH6	<disponible>	<disponible>	<disponible>
GPIOH7	<disponible>	<disponible>	<disponible>

Tabla 5: Correspondencia de pinout del modo MPSSE con los principales buses serie

También es interesante ver el diagrama de tiempos de las señales que forman este interfaz y la tabla de tiempos asociada a estas señales.

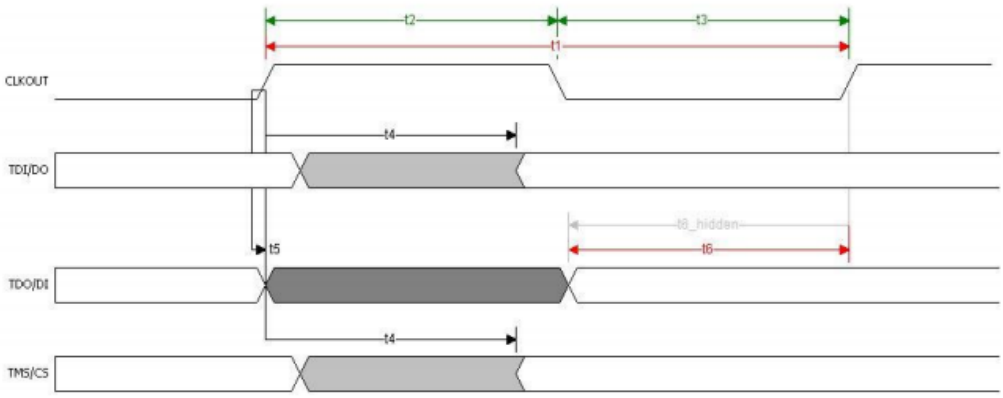

Figura 6: Diagrama de tiempos asociados a las señales para interfaz JTAG

Name	Minimum	Typical	Maximum	Units	Description
t1		33.33		ns	CLKOUT period
t2	15	16.67		ns	CLKOUT high period
t3	15	16.67		ns	CLKOUT low period
t4	1		7.15	ns	CLKOUT to TDI/DO delay
t5	0			ns	TDO/DI hold time
t6	11				TDO/DI setup time

Tabla 6: Especificaciones temporales del bus JTAG

El MPSSE va a ser siempre el maestro en la comunicación cuando estemos comunicándonos con un interfaz síncrono, por lo tanto, será el que se encargue de general el reloj y las señales de chip select requeridas por el interfaz.

El reloj del MPSSE lo podemos configurar eligiendo entre diferentes frecuencias de trabajo. También podemos configurar los flancos de reloj con los que vamos a trabajar en cada operación.

- El reloj interno del chip FT2232H es de 60 MHz y puede ser dividido de acuerdo con la siguiente fórmula:

$$\text{Data Speed} = \frac{60\ \text{MHz}}{((1 + \text{divisor}) + 2)}$$

  Donde el divisor será un valor de 16 bits, lo cual permite un rango de operaciones entre 30 y 460MHz.

- Podemos configurar también el flanco utilizado. Los datos de entrada/salida son almacenados de manera síncrona con el flanco de subida o de bajada del bus serie. El sentido del flanco que vamos a utilizar es configurable por el

usuario y puede configurarse de manera independiente para cada uno de los sentidos de transferencia de los datos.

2.1.6.1 Implementación de un bus JTAG

Además de la entrada/salida de datos y de la señal de reloj, el interfaz JTAG necesita implementar una cuarta señal de control que nos permita movernos a través de la máquina de estados.

Todas las comunicaciones en las que usemos el interfaz JTAG son LSB first, lo que quiere decir que los datos son desplazados a través de la máquina de estados en el flanco de subida. Esto implica que el interfaz MPSSE debe sacar los datos en el anterior flanco de bajada para garantizar el tiempo de setup.

Debemos configurar el sentido de entrada/salida de los pines dedicados a implementar el puerto serie y las señales GPIO, así como su valor por defecto.

2.2 FDI Mini-Module

El FT2232H Mini Module es un módulo de desarrollo de USB a serie/FIFO que el chip visto anteriormente FT2232H de dos puertos USB de alta velocidad para manejar los dos protocolos y la señalización de USB. Es ideal para tareas de desarrollo, pues permite demostrar de forma rápida la funcionalidad al añadir USB a un diseño de destino.

Figura 7: Dispositivo FTT2232H Mini Module

El FT2232H en el Mini Module permite el acceso al pinout del integrado del chip mediante dos tiras dobles de conectores. Estos dos conectores los denominaremos CN2 Y CN3 y tienen en cada fila 13 pines, por lo que, cada conector tiene un total de 26 pines distintos. Más tarde veremos el pinout y explicaremos con más detenimiento cuál es la función de cada uno de estos pines. El espaciado entre pines de este mini módulo es de 2.54mm, lo que equivale a una décima de pulgada.

A continuación, describiremos algunas de las características principales de este mini módulo:

- Es compatible con USB 2.0 de alta velocidad, pero también lo es con USB 1.0 al igual que ocurría con el chip FT2232H.
- Tiempo de desarrollo reducido.
- Rápida integración en sistemas existentes.
- Puede alimentarse a través del USB, no se requiere alimentación externa, pero también cuenta con esa posibilidad.
- Basado en el dispositivo FT2232H USB de alta velocidad.
- Protocolo USB manejado íntegramente por módulo USB.
- Velocidades de transferencia de datos serie asíncrona de 300 baudios a 12Mbaudios a niveles TTL.
- Velocidades de transferencia de datos serie asíncrona de hasta 30Mbps en JTAG.

- Soporta suspensión y reanudación de USB.
- Compatible con controlador host UHCI / OHC I/ EHCI.

2.2.1 Interfaz de entrada/salida y pinout

A continuación, veremos una imagen de una vista en planta del dispositivo donde podremos hacernos una idea más detallada de la distribución de los conectores CN2 y CN3.
Recordamos que ambos conectores son de dos filas de 13 pines cada una y con una separación entre pines de 2.54mm.
En la imagen hemos agrupado los pines por colores dependiendo de su funcionalidad, en blanco podemos ver los pines que no hemos utilizado en este trabajo, aunque en la parte de diseño veremos que tendremos que utilizar alguno de los que en principio están no conectados.

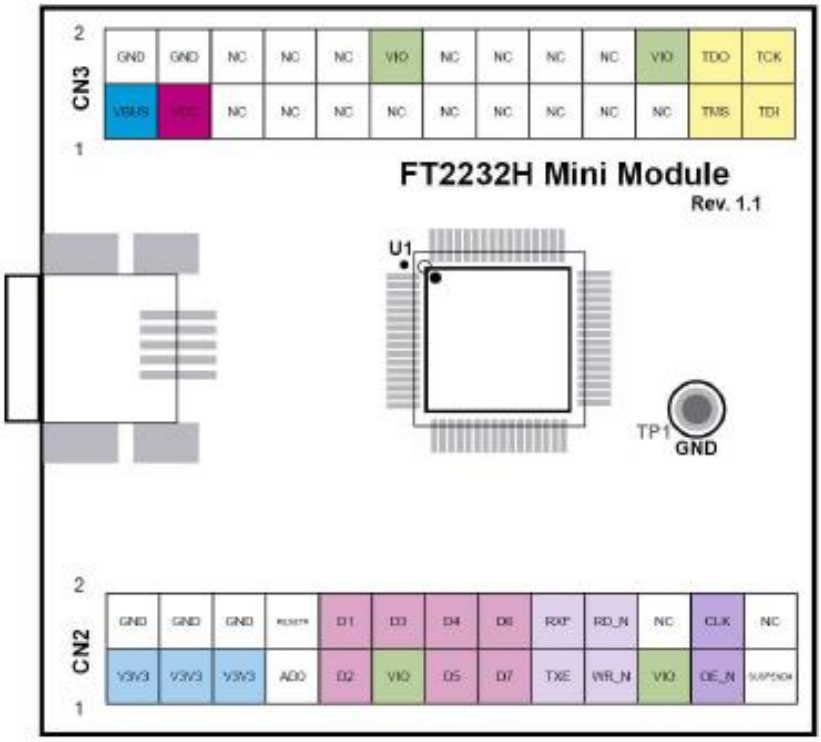

Figura 8: Pinout del dispositivo FT2232H Mini Module

Ahora veremos la presentación de las tablas con la función de cada pin dentro del integrado del mini módulo.

Connector Pin	Name	Description
CN2-1	V3V3	3.3VDC generated from VCC (output)
CN2-2	GND	0V Power pin
CN2-3	V3V3	3.3VDC generated from VCC (output)
CN2-4	GND	0V Power pin
CN2-5	V3V3	3.3VDC generated from VCC (output)
CN2-6	GND	0V Power pin
CN2-7	AD0	FT2232H AD0 pin
CN2-8	RESET#	FT2232H RESET# pin
CN2-9	AD2	FT2232H AD2 pin
CN2-10	AD1	FT2232H AD1 pin
CN2-11	VIO	Connected to all FT2232H VCCIO pins (input)
CN2-12	AD3	FT2232H AD3 pin
CN2-13	AD5	FT2232H AD5 pin
CN2-14	AD4	FT2232H AD4 pin
CN2-15	AD7	FT2232H AD7 pin
CN2-16	AD6	FT2232H AD6 pin
CN2-17	AC1	FT2232H AC1 pin
CN2-18	AC0	FT2232H AC0 pin
CN2-19	AC3	FT2232H AC3pin
CN2-20	AC2	FT2232H AC2 pin
CN2-21	VIO	Connected to all FT2232H VCCIO pins (input)
CN2-22	AC4	FT2232H AC4 pin
CN2-23	AC6	FT2232H AC6 pin
CN2-24	AC5	FT2232H AC5 pin
CN2-25	SUSPEND#	FT2232H SUSPEND# pin
CN2-26	AC7	FT2232H AC7 pin

Tabla 7: Pinout conector CN2

Connector Pin	Name	Description
CN3-1	VBUS	USB VBUS power pin (output)
CN3-2	GND	0V Power pin
CN3-3	VCC	+5V Power pin (input) used to generate V3V3, VPLL and VUSB
CN3-4	GND	0V Power pin
CN3-5	CS	FT2232H EECS pin
CN3-6	CLK	FT2232H EECLK pin
CN3-7	DATA	FT2232H EEDATA pin
CN3-8	PWREN#	FT2232H PWREN#
CN3-9	BC7	FT2232H BC7 pin
CN3-10	BC6	FT2232H BC6 pin
CN3-11	BC5	FT2232H BC5 pin
CN3-12	VIO	Connected to all FT2232H VCCIO pins (input)
CN3-13	BC4	FT2232H BC4 pin
CN3-14	BC3	FT2232H BC3 pin
CN3-15	BC2	FT2232H BC2 pin
CN3-16	BC1	FT2232H BC1 pin
CN3-17	BC0	FT2232H BC0 pin
CN3-18	BD7	FT2232H BD7 pin
CN3-19	BD6	FT2232H BD6 pin
CN3-20	BD5	FT2232H BD5 pin
CN3-21	BD4	FT2232H BD4 pin
CN3-22	VIO	Connected to all FT2232H VCCIO pins (input)
CN3-23	BD3	FT2232H BD3 pin
CN3-24	BD2	FT2232H BD2 pin
CN3-25	BD1	FT2232H BD1 pin
CN3-26	BD0	FT2232H BD0 pin

Tabla 8: Pinout conector CN3

Vamos a describirlos algo más detalladamente a continuación. Para ello, los agruparemos en tres bloques diferenciados, como son, alimentaciones, interfaz serie e interfaz de programación que esta formado por las señales del JTAG, que es el interfaz de programacion elegido para este trabajo.

- Alimentaciones: Con estos pines alimentaremos las diferentes partes del integrado, utilicemos el interfaz que utilicemos, estas señales deben estar conectadas. Distinguimos cuatro grupos.
  - VBUS, tensión de salida 5V proporcionada por el USB.

    - VCC, entrada de alimentacion de 5V a partir de la cual se generan V3V3, VPLL y VIO.
    - V3V3, tensión de salida 3,3V generada a partir de VCC.
    - VIO, tensión de alimentación para los bancos de entrada/salida, también de 3.3V.
- Interfaz serie, buses de datos que vamos a usar para la comunicación, distinguimos aquí dos grupos de señales perfectamente diferenciadas:
    - Buses de datos, comunes para el modo simétrico y asimétrico.
    - Buses de control formado por las señales RXF#, TXE#, WR#, RD#, OE# y CLK. Las dos últimas señales únicamente son necesarias si estamos utilizando el modo síncrono.
- Interfaz de programación, formado como hemos explicado anteriormente por las señales definidas en el estándar IEEE 1149.1, TDO, TDI, TMS y TCK.

Una de las características que definimos anteriormente en la introducción de este dispositivo es que no necesitaba una fuente de alimentación externa porque podía alimentarse a través del propio USB. Si bien es cierto, esto no quiere decir que sea la única forma de alimentación. En este apartado distinguimos dos formas principales de alimentación.

- Alimentación por USB:
    - Debemos conectar los pines VBUS a VCC (CN3, pin1 a CN3 pin3). De esta manera utilizaremos la tensión de 5V proporcionada por el bus del USB para alimentar el regulador interno del módulo. Como podemos ver en el esquemático, el regulador proporcionara las alimentaciones V3V3, VPLL y VIO.
    - El segundo paso será conectar los pines V3V3 a VIO (pines 1, 3 y 5 del conector CN2 a los pines 11 y 21 del mismo conector y pines 12 y 22 del conector CN3) para proporcionar alimentación a los bancos de entrada/salida.
- Modo auto-alimentación:
    - No se utilizara la tensión proporcionada por el bus USB, pin VBUS no conectado.
    - Debemos conectar una fuente externa de alimentación de 5V al pin 3 del conector CN3, proporcionando alimentación al regulador del módulo.
    - Conectar los pines V3V3 a VIO (pines 1, 3 y 5 del conector CN2 a los pines 11 y 21 del mismo conector y pines 12 y 22 del conector CN3) para proporcionar alimentación a los bancos entrada/salida.
    - Utilizar la herramienta software del fabricante MPROG para cambiar la configuración del módulo a Self-Powered. De esta manera se le indicuará al host USB que no es necesario suministrar tensión de alimentación a través del bus.

En el esquemático del dispositivo FT2232H Mini Module podemos ver mejor la situación de los pines que hemos descrito en el apartado anterior.

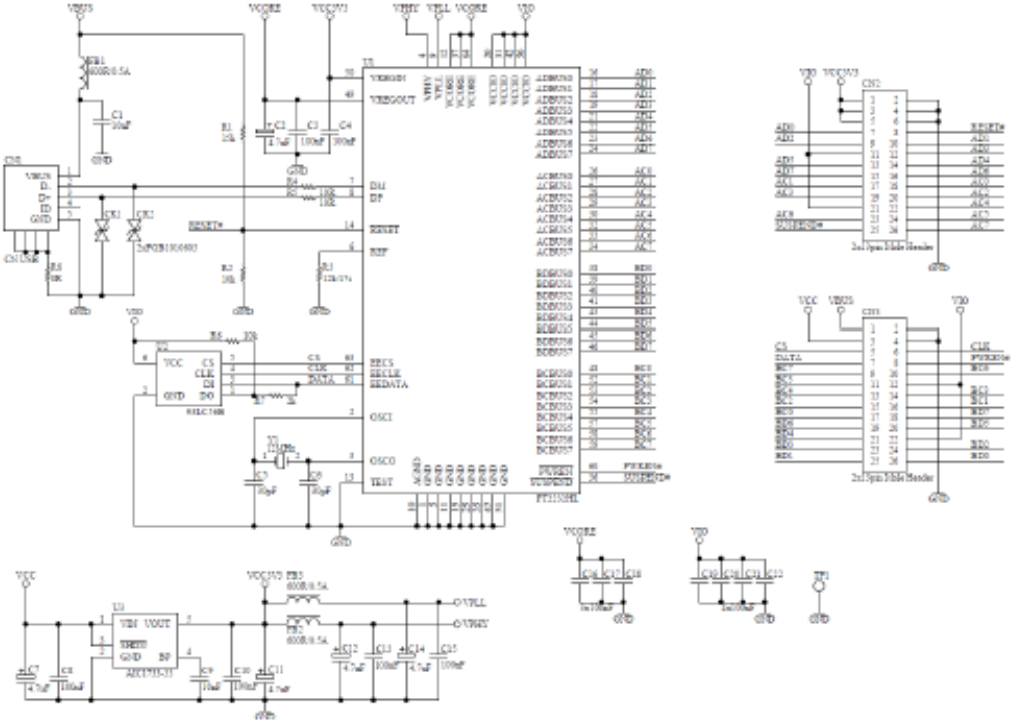

Figura 9: Esquemático del dispositivo FT2232H Mini Module

Para concluir con el tema de la alimentación, únicamente hacer una última aclaración sobre lo presentado en este capítulo. Las dos opciones que nos da el fabricante para alimentar el dispositivo no son las únicas y en la fase de diseño veremos alguna más y razonaremos cuál de todas ellas es por la que vamos a optar.

En este trabajo, uno de los objetivos descritos anteriormente en esta memoria era el diseño de una tarjeta que albergase varios dispositivos, entre ellos este módulo, por lo que será de gran utilidad tener un plano de fabricación del dispositivo que estamos tratando, porque más tarde en la fase de diseño necesitaremos saber y tener en cuenta las dimensiones del módulo.

Aquí vemos un plano de planta del FT2232H Mini Module y un alzado donde vemos su espesor.

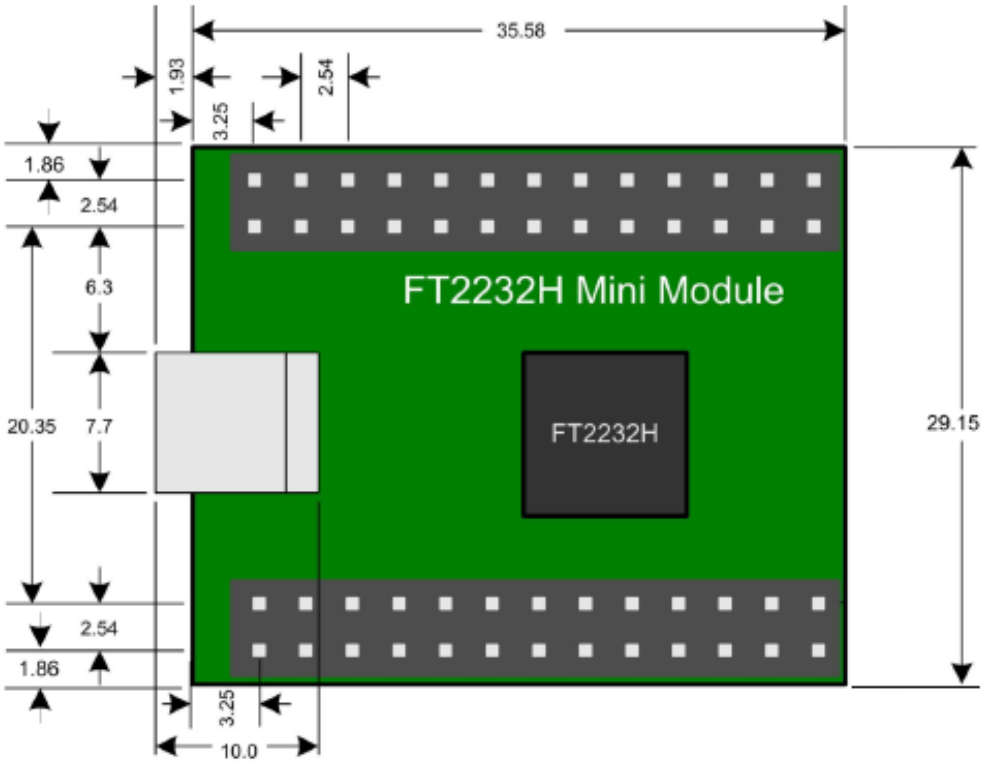

Figura 10: Plano planta fabricación FT2232H Mini Module

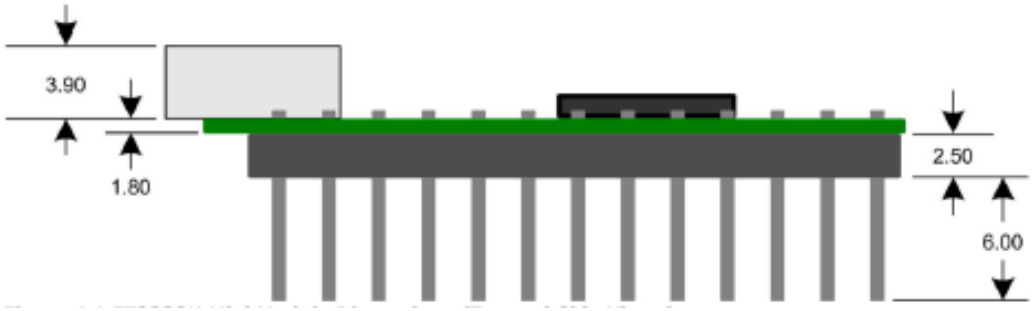

Figura 11: Plano alzado fabricación FT2232H Mini Module

Todas las dimensiones de los planos están expresadas en milímetros. Los dos conectores CN2 y CN3 están montados en la parte de debajo de la placa base. El espesor total de la pieza es de 8.5 mm y todos los pines están en una cuadrícula de decima de pulgada.

2.3 TE0630 (SPARTAN-6)

La TE0630 de Trenz Electronics es un mini módulo de FPGA destinada al uso industrial que se compone principalmente por la FPGA, del fabricante Xilinix, Spartan-6, un puerto mini USB 2.0 una SDRAM DDR3 de 1Gbit y 8Mbyte de memoria flash para configuración y operación. Además, cuenta también con una fuente de alimentación para todos los conectores de la placa.
Tiene un número considerable de entradas/salidas configurables de alta velocidad. Todos los componentes de este mini módulo tienen un rango de temperaturas de operación de entre -40ºC y 85ºC, aunque este rango está sujeto a variaciones en función del diseño donde vaya montado este dispositivo.

Figura 12: TE0630 planta superior

Figura 13: TE0630 planta inferior

2.3.1 Especificaciones técnicas

En el anterior apartado, donde explicábamos las características del FT232H Mini Module, decíamos que era importante tener una idea clara de sus dimensiones, no es el caso de este mini módulo de FPGA debido a que irá acoplado en nuestro diseño final en una carrier board TE0303 que describiremos en el siguiente apartado, y será esta la que nos condicione el diseño.
De todos modos, aquí tenemos un plano de fabricación en planta para hacernos una idea de las dimensiones del módulo.

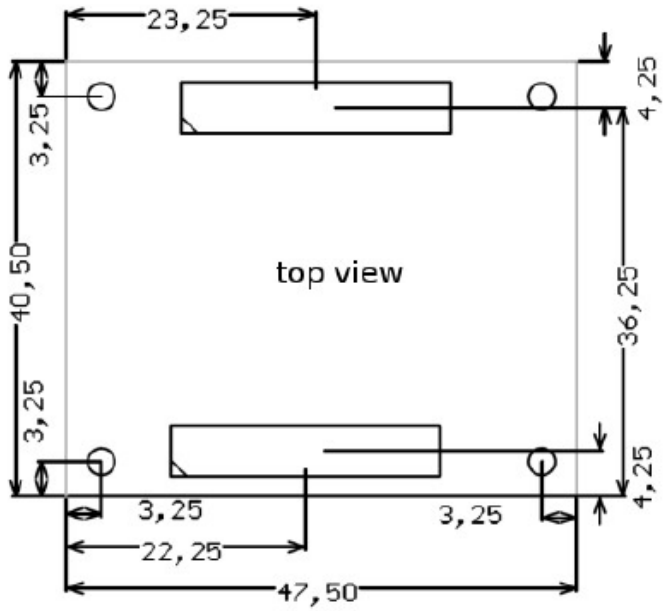

Figura 14: Plano de planta TE0630

Todas las dimensiones del anterior plano están en milímetros.

2.3.2 Descripción detallada.

A continuación, veremos algunas de las especificaciones de los componentes que forman parte de la FPGA.

2.3.2.1 Diagrama de bloques.

En la siguiente imagen vemos el diagrama de bloques del módulo TE0630.

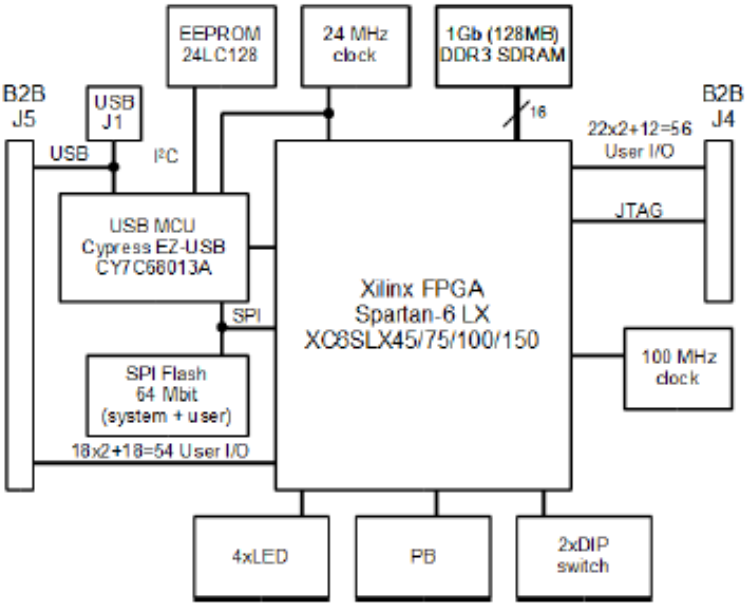

Figura 15: Diagrama de bloques del módulo TE0630

2.3.2.2 Alimentación.

El módulo puede ser alimentado a través de conector B2B J5 o a través del conector del USB. En caso de que los dos modos de alimentación estén disponibles, el conector B2B tiene preferencia sobre la alimentación del USB, deshabilitándola automáticamente.
A continuación, describiremos la alimentación de los distintos conectores que encontramos en el módulo.

- Alimentación del conector B2B: La alimentación de este conector requiere 5V. La alimentación normalmente viene de los pines (5Vb2b2) del conector J5. La recomendación es que el voltaje mínimo sean 4V mientras que el máximo sea de 5.5V. En cuanto a la corriente, no debería superar los 1.5A.
- Alimentación del conector del USB: Él módulo es alimentado a través del conector del USB si se dan las siguientes condiciones:
  - Cuando el módulo está equipado con un conector de USB.
  - Cuando el módulo está conectado a un bus de USB.
  - Cuando los conectores B2B no dan alimentación.

En este caso otros componentes, como pueden ser extensiones de una carrier board, serán alimentados con los 5V del conector B2B.

- Conexiones de alimentación en la placa: Tres reguladores de tensión ubicados en la placa son los que proveen de alimentación a las conexiones que necesitan los componentes del módulo.

Vemos un diagrama de la alimentación de la placa.

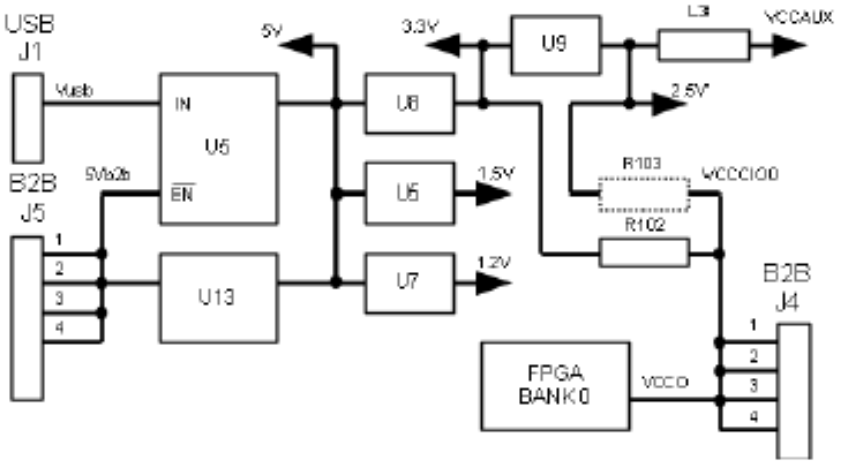

Figura 16: Diagrama de alimentación de TE0630

- Encendido de Reset: Durante el encendido de la FPGA, la señal de /RESET está controlada. A partir de entonces el supervisor de voltaje de alimentación monitorea la alimentación a 3.3V y mantiene la señal de /RESET activa a nivel bajo, mientras que las conexiones de alimentación tengan el voltaje crítico de 2.93V. Un temporizador interno retrasa el retorno de la señal de /RESET al estado inactivo (alto) para asegurar el correcto reseteo del sistema antes que un arranque del sistema normal.

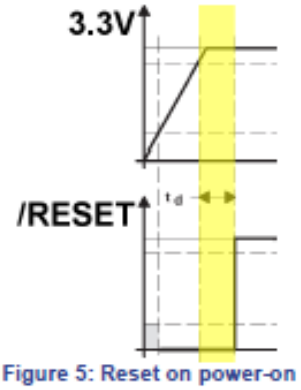

Figure 5: Reset on power-on

Figura 17: Diagrama señal de Reset

2.3.2.3 FPGA entradas/salidas

La tarjeta TE0630 provee de entradas/salidas disponibles conectadas a los conectores B2B, J4 y J5. Tenemos 3 tipos distintos de señales de entrada/salida:

- Simples
- Pares diferenciales en las que cada par es configurable como 2 entradas/salidas simples.
- Pares diferenciales, los cuales pueden ser usados como señales de reloj entrantes, cada par puede ser configurable como 2 entradas/salidas simples.

En la tabla siguiente podemos ver las entradas/salidas disponibles para los conectores J4 y J5.

	J4	J5	Total
Single ended	12	18	30
Differential	18	16	34
Differential (clock)	4	2	6
Total	56	54	110

Tabla 9: Entradas/salidas de los conectores J4 y J5

Y en esta otra vemos las entradas/salidas en la fuente de alimentación VCCIO.

	VCCIO0	3.3V	Total
Single ended	1	29	30
Differential	18	16	34
Differential (clock)	4	2	6
Total	45	65	110

Tabla 10: Entradas/salidas en la fuente de alimentación VCCIO

2.3.2.4 Conectores board to board (B2B)

El módulo tiene dos conectores board to board, que son los conectores J4 y J5 para un total de 160 pines. En la siguiente imagen podemos ver cómo son físicamente los conectores.

Figura 18: Conector B2B

2.3.2.5 Conector JTAG

El paso de los agujeros del conector J2, destinado al interfaz JTAG, es de décima de pulgada. Tenemos 6 agujeros al tresbolillo para que no sea necesario soldar correspondientes a las 4 señales de JTAG, una tierra y una alimentación.

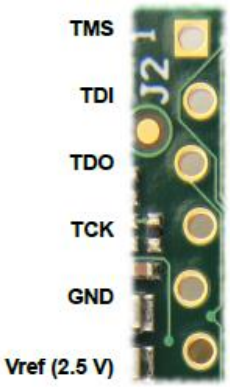

Figura 19: Conector JTAG

2.4 Carrier Board TE0303

La TE0303 es una carrier board para micromódulos como el que acabamos de ver, el TE0630, que contiene la FPGA Spartan-6. Este dispositivo es una solución barata para la extensión de la Spartan-6. Las señales del micromódulo tienen un montaje superficial y están conectadas a los conectores estándar.

Figura 20: Carrier board TE0303, vista de planta superior

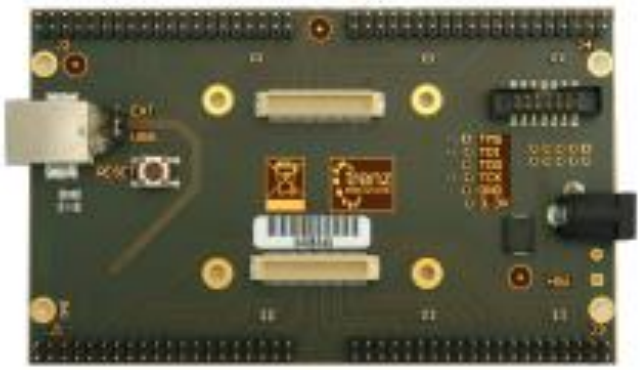

Figura 21: Carrier board TE0303, vista de planta inferior

2.4.1 Especificaciones técnicas

Todas las señales disponibles en los conectores J1, J2, J3, J4 están en una cuadrícula de 2.54mm que equivale a décima de pulgada. Tenemos 42 pares diferenciales para las señales de alta velocidad y 26 señales simples para las señales de media o baja velocidad.
Alimentación a través de Jack DC o bus USB.
Además de esto, tenemos varios conectores compatibles con JTAG, como son el J9, J10 y J11.
Como hemos hablado de varios conectores, será mejor ver el plano de ensamblaje para hacer una mejor idea de la situación de estos.

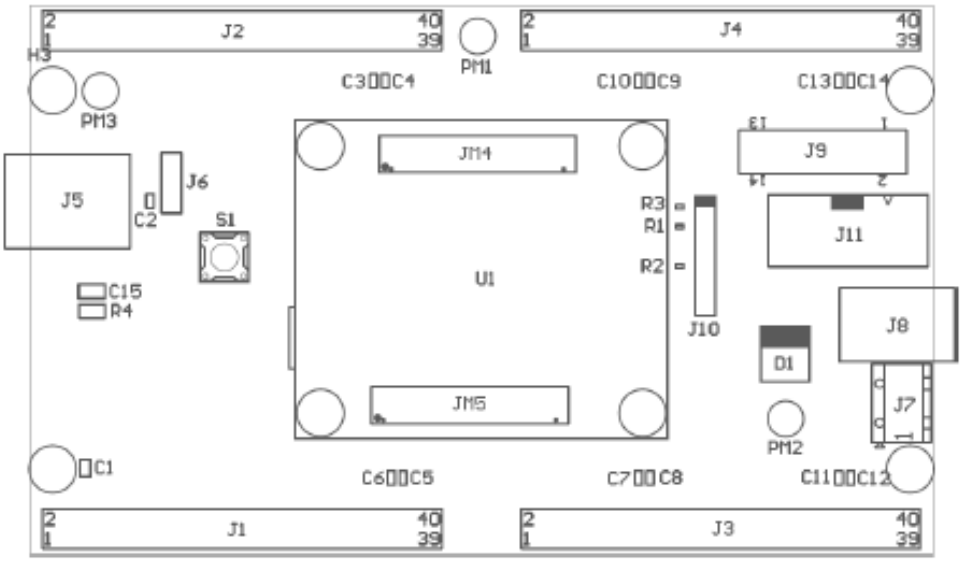

Figura 22: Plano de ensamblaje de la carrier board

2.4.2 Descripción detallada.

2.4.2.1 Alimentación

Existen tres opciones diferentes para alimentar la placa.

- 5V de continua vía Jack.
- 5V de continua vía terminales tornillo.
- A través del bus del USB.

Para alimentarlo con 5V de corriente continua debemos ajustar el jumper J6 a EXT y conectar a 5V de corriente exterior a través del Jack en uno de los laterales de la placa. Si, en cambio, queremos alimentar la placa a través del USB, debemos ajustar el jumper J6 a USB y conectar un cable estándar de USB a receptor J5.
A parte, tenemos en esta carrier board, ciertos pines de alimentación en algunos de los conectores que pueden alimentar circuitos externos, estos pines tienen voltajes de 1.2, 2.5 y 3.3V.

2.4.2.2 Bancos de entrada/salida

La carrier board tiene un total de 26 señales simples, conectadas a los bancos J1, J2, J3 y J4. Estas conexiones pueden ser utilizadas para bajas o medias velocidades.
También cuenta con 42 pares de señales diferenciales que están conectadas con una impedancia de 100 ohm a los conectores del J1 al J4. Estas conexiones están reservadas para señales de alta velocidad.

2.4.2.3 Conectores de JTAG

Tenemos tres opciones para elegir dónde queremos entrar con el interfaz de JTAG. Los conectores con disponibilidad para JTAG son el J9, J10 y J11.
El J9 es un conector de 14 pines para un cable del fabricante Xilinx o para USB.

Signal	Pin	Pin	Signal
GND	1	2	Vref
GND	3	4	TMS
GND	5	6	TCK
GND	7	8	TDO
GND	9	10	TDI
GND	11	12	n.c.
GND	13	14	n.c.

Tabla 11: Pinout para conector J9

En cuanto al J10 es un conector de 6 pines clásico de JTAG con las siguientes señales.

Pin	Signal
1	TMS
2	TDI
3	TDO
4	TCK
5	GND
6	Vref (3.3 V)

Tabla 12: Pinout para conector J10

Por último, el conector j11, este se ha añadido para facilitar el uso de la herramienta de Trenz Electronic, un programador de JTAG que en este trabajo no utilizaremos.

2.4.2.4 Pinout de los conectores J1 a J4

En posteriores capítulos deberemos escoger uno de estos 4 conectores para mandar las señales del FT2232H Mini Module hacia la FPGA. Es por esto por lo que debemos saber que señales hay en cada uno de los pines de esta carrier board y ver cuáles están libres. Para ello nos ayudara conocer el pinout.

pin	B2B name	FPGA pin	dir	dir	FPGA pin	B2B name	pin
1	+Vb2b	-	I	I	-	+Vb2b	2
3	+Vb2b	-	I	I	-	+Vb2b	4
5	5Vout	-	O	I	-	/MR	6
7	RESET	-	O	O	-	/RESET	8
9	V3_IO_17	U3	IO	IO	V1	V3_IO_18	10
11	GND	-	-	-	-	GND	12
13	V2_IO_01	V15	I	IO	U1	V3_IO_16	14
15	V3_IO_22	AB3	IO	IO	Y2	V3_IO_21	16
17	V3_IO_27	U8	IO	IO	Y3	V3_IO_23	18
19	GND	-	-	-	-	GND	20
21	V3_IO_13	T1	IO	IO	T2	V3_IO_12	22
23	V3_IO_15	AB2	IO	IO	AA2	V3_IO_14	24
25	V3_IO_20	Y1	IO	IO	V2	V3_IO_19	26
27	GND	-	-	-	-	GND	28
29	V3_IO_25	AA4	IO	IO	AB4	V3_IO_24	30
31	V2_IO_01_P	AA6	IO	IO	AB6	V2_IO_01_N	32
33	V2_IO_02_N	AB7	IO	IO	Y7	V2_IO_02_P	34
35	3.3 V	-	O	O	-	3.3 V	36
37	V2_IO_03_P	AA8	IO	IO	AB8	V2_IO_03_N	38
39	V2_IO_10_N	Y6	IO	IO	W6	V2_IO_10_P	40

Tabla 13: Pinout conector J1

Más tarde explicaremos por qué razón hemos elegido el conector J1 para hacer nuestras conexiones, es por esto por lo que solo se mostrará el pinout de este conector.

Si queremos ver el pinout de los demás conectores del dispositivo deberemos visitar la datasheet del fabricante.

Por último, veremos una imagen del esquemático del conector J1 para hacernos una mejor idea de como están distribuidas las señales.

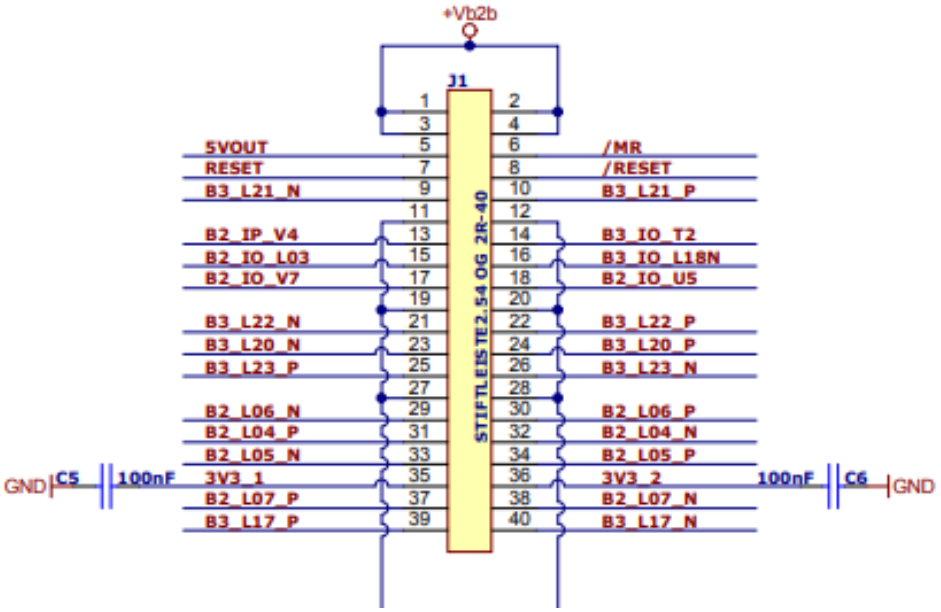

Figura 23: Esquemático del conector J1

2.5 Herramientas software utilizadas.

El software utilizado en este trabajo ha sido el OrCad Cadence 16.6. Orcad es un software propietario utilizado para automatización de diseño electrónico. El software es usado por técnicos e ingenieros de diseño, fundamentalmente para simulación electrónica, crear esquemas electrónicos y elaborar esquemas de circuito impreso para manufacturar placas de circuito impreso.

Dentro de este programa hay una gran cantidad de aplicaciones para distintas funcionalidades pero que están todas conectadas entre sí.

Durante el trascurso de este proyecto se han utilizado sobre todo dos de estos pequeños programas que están dentro del software.

- Capture CIS: Es el primer programa que hemos necesitado utilizar, tiene muchas posibilidades diferentes, pero sobre todo se centra en los esquemas de circuitos. En nuestro caso particular, lo hemos utilizado para realizar los esquemas de los distintos conectores de nuestra placa, para que al abrir el programa en el que diseñaremos la PCB, los pines ya estén unidos correctamente. La segunda tarea que hemos tenido que llevar a cabo con Capture ha sido, asignar los footprints de los elementos físicos creados en PCB Editor a los componentes en los esquemas.

- PCB Editor/Designer: Este es el segundo programa dentro de OrCad que hemos necesitado para el desarrollo del trabajo. Podemos distinguir dos tareas diferentes para las que hemos utilizado este software.
  En primer lugar, tuvimos que crear las huellas de todos los elementos que van a ir colocados más tarde en nuestra placa, para más tarde asignarlos a los componentes en el esquema. Y, por último, crear la PCB con las dimensiones correctas e importar el archivo desde Capture para conectar todos los elementos de forma correcta y generar sus planos, exportándolos a PDF.

A estos dos programas del entorno OrCad podríamos añadirle alguno más como PSpice que es usado para ver simulaciones eléctricas de los circuitos creados en Capture, o Padstack Editor con el que podemos editar y crear todo tipo de pines.

Una herramienta muy potente y con múltiples posibilidades, pero a su vez poco intuitiva y difícil de comprender para alguien que se esté iniciando en el diseño de PCB.

3. Planteamiento del diseño

En este capítulo vamos a plantear un problema que se nos ha dado, partiendo de algunos trabajos anteriores. Como ya sabemos, este trabajo consiste en el diseño y fabricación de una PCB que conecte varios módulos diferentes.
Una vez hemos descrito en los capítulos anteriores lo módulos que vamos a utilizar el proyecto, ahora veremos por qué queremos conectarlos entre sí y cómo lo vamos a hacer, siguiendo todo el proceso de búsqueda de soluciones y de prueba y error en la parte de diseño.

3.1 Problema inicial.

Como todo proyecto o trabajo a desarrollar, este, parte también de un problema que nos encontramos y al que debemos dar solución. En nuestro caso el problema que se nos presenta es que tenemos dos dispositivos como son el controlador de USB FT2232H y la FPGA Spartan-6 que necesitan ser conectados para realizar correctamente las funciones para las que están destinados, y estas son comunicarse con un host a través de un conector de USB y que a través de el podamos tener una comunicación de datos y configurar la FPGA.
Hasta ahora las pruebas que se habían hecho con estos dos dispositivos eran con un cableado externo, lo cual para unas primeras pruebas del interfaz de comunicación y programación está bien, pero cuando queremos que estos dispositivos trabajen conjuntamente de manera prolongada en el tiempo sería mejor crear una tarjeta en la que englobemos las conexiones entre los dos dispositivos.
El controlador de USB FT2232H está dentro del FT2232H Mini Module y las Spartan-6 está unida a través de los conectores J4 y J5 a la carrier board TE0303, por lo tanto, nosotros debemos centrarnos en las especificaciones técnicas y en las conexiones entre estos dos dispositivos.
Una vez que hemos presentado el problema vamos a pasar a comentar la solución que se nos ocurrió y como hemos ido desarrollándola.

3.2 Búsqueda de una solución.

Cuando se nos presentó el problema de conectar ambos dispositivos decidimos que la mejor opción era crear una placa base en la que se encontraran todas las conexiones entre ellos.
En este caso la solución era clara pero no sencilla debido a que existen una gran cantidad de posibilidades a la hora de conectar estos dos dispositivos y tenemos que intentar elegir la mejor teniendo en cuenta muchos aspectos.

- Diseño
- Económicos
- Robustez del dispositivo fabricado
- Alimentación

Los pasos seguidos en este proceso han sido los siguientes:

- Para empezar, debemos tener en cuenta los conectores que tienen los dos dispositivos con los que vamos a trabajar, por lo tanto, debemos hacer un estudio del pinout de ambos. Por un lado, tenemos el mini módulo del controlador de USB, en este caso el trabajo es algo más sencillo debido a que están claras las señales que vamos a usar, pero por el lado de la FPGA tenemos que ver las señales libres que tenemos y elegir el conector adecuado al que conectarlas.
- El segundo paso será elegir exactamente las conexiones, teniendo en cuenta ya un posterior diseño en PCB, lo que implica situación de los conectores y que no haya cruces de señales en la medida que sea posible.
- Una vez tengamos claras las conexiones que tenemos que realizar, nos iremos a la herramienta de software OrCad en el entorno de trabajo Capture CIS y continuaremos haciendo los esquemas de los conectores que vamos a poner en nuestro diseño.
- Por último, presentaremos los resultados de los esquemas en un formato adecuado para simplificar el trabajo del diseño de la placa posteriormente en la herramienta de OrCad PCB Editor.

3.3 Pinout de los dispositivos.

3.3.1 Pinout FT2232H Mini Module

Hemos comentado que el FT2232H Mini Module está más acotado en cuanto a las señales que tenemos que enviar hacia la FPGA y también respecto a la alimentación. Vamos a ver a continuación una imagen del pinout del dispositivo.

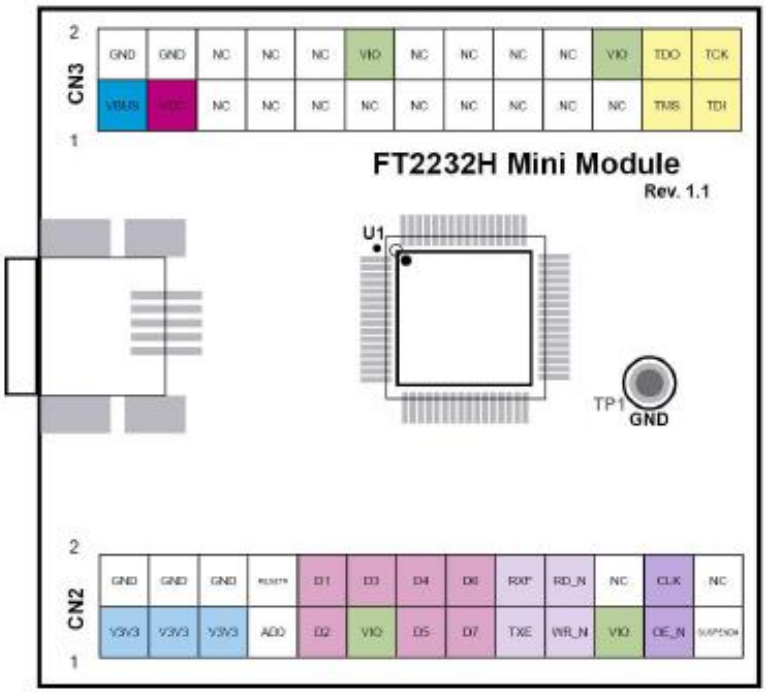

Figura 24: Pinout FT2232H Mini Module

Vamos a hacer un recuento de las señales que tenemos en este dispositivo distinguiéndolas entre alimentación, comunicación de datos, configuración y de control.

- Señales de datos: Tenemos 8 señales de datos ubicadas todas ellas en el conector CN2 que está formado por dos tiras de 13 pines cada uno, separados décima de pulgada entre ellos. Estas señales son: AD0, D1, D2, D3, D4, D5, D6 y D7. Todas estas señales deberemos llevarlas a algún conector de la carrier board que luego decidiremos.

- Señales de control: Son en total 6 señales de control, RXF, TXE, WR_N, RD_N, OE_N y CLK. Aunque ya explicamos a que se dedicaba cada una de estas señales cuando describíamos el controlador de USB, podemos recordar que RXF, TXE, WR_N y RD_N las usábamos en la lectura y escrituro en el modo de funcionamiento 245 FIFO, mientras que CLK es el reloj propio del core del controlador de USB.

- Señales de configuración: Son las 4 propias del interfaz de configuración JTAG, TCK, TMS, TDI, TDO.

- Por último, nos queda la alimentación, este tema lo tratamos también cuando describimos el mini módulo y las posibilidades que había de alimentarlo. Explicaremos más adelante la solución implementada para la alimentación del dispositivo y de la placa en general, de momento vamos a hacer un recuento de las señales de alimentación que tienen los conectores y que significado tiene cada una.
  Tenemos para empezar VBUS, que son los 5V que provienen del bus del USB y es una señal de salida. VCC es una señal de entrada de 5V también. V3V3 son señales de salida de 3.3V, mientras que VIO son las señales de 3.3v de entrada que alimentan las entradas/salidas del conector.

3.3.2 Pinout Carrier Board TE0303

Una vez que tenemos todas las señales que sabemos que hay que llevar a la FPGA, vamos a pasar al siguiente dispositivo. Con la carrier board el tenemos que abordar el problema de forma similar, pero con alguna diferencia.
En este caso, como son las señales del mini módulo las que tenemos que llevar a la carrier board, en esta deberemos buscar señales que no estén asignadas a nada para poder llevarlas a esos pines.
La tarjeta TE0303 tiene cuatro conectores principales, el J1, J2, J3 y J4. Están dispuestos en los laterales de la tarjeta con la siguiente disposición.

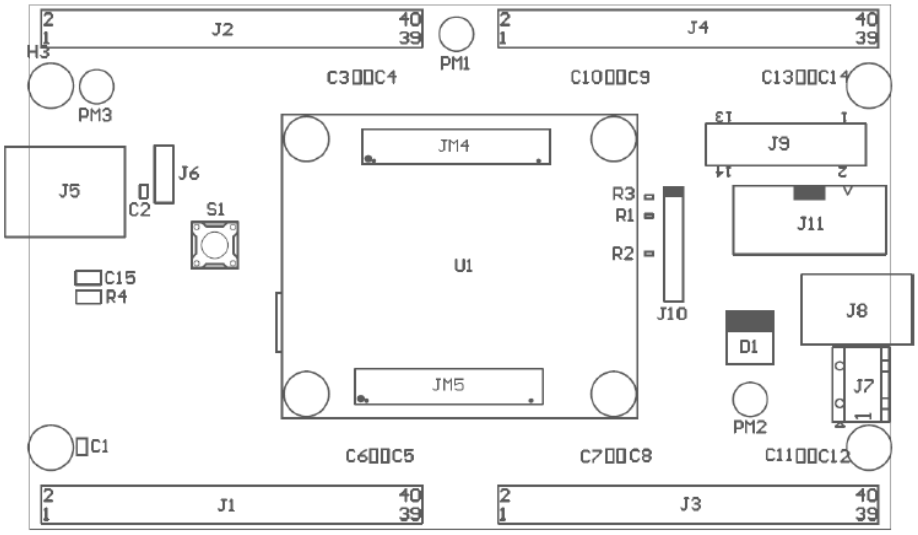

Figura 25: Distribución conectores TE0303

Vamos a seguir el mismo procedimiento que con el anterior dispositivo para ello vamos a ver el pinout de cada uno de los cuatro conectores que luego están conectados con la FPGA.

3.3.2.1 Pinout conector J1

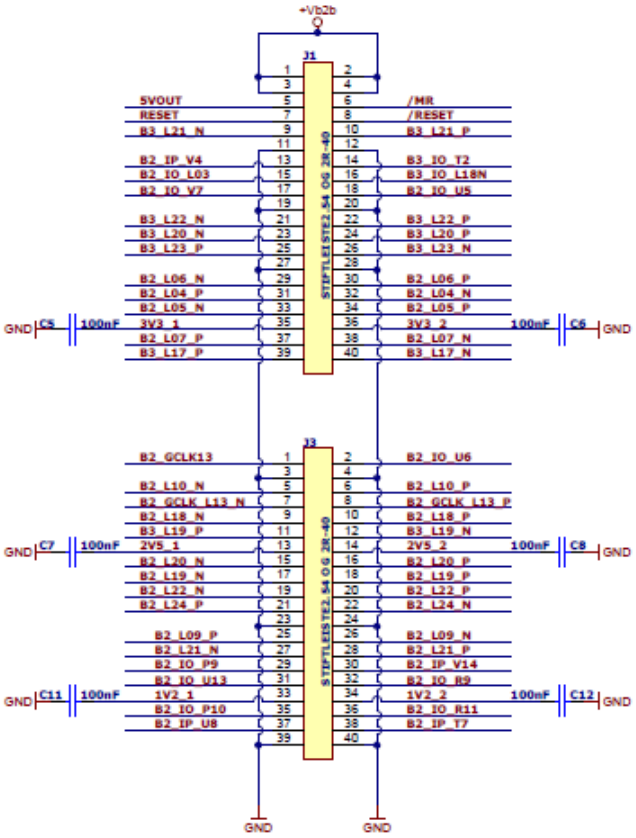

Figura 26: Esquemático J1

Empezaremos primero por el conector J1, vamos a hacer un recuento de las señales que tiene. En las señales 1, 2 ,3 y 4 vemos un Vb2b que son 5V de salida que pueden servir para alimentar algún otro dispositivo. Vemos en el pin 6 una señal llamada /MR y en el 8 una señal de /RESET, son ambas señales para resetear la FPGA. Los pines 11, 12, 19, 20, 27 y 28 están conectados a tierra, mientras que el resto son pines libres, a lo cuales podemos llevar señales del mini módulo.
En definitiva, en este conector tenemos:

- 4 pines de tensión de salida 5V.
- 6 pines conectados a tierra.
- /RESET y /MR.
- 24 pines libres.

3.3.2.2 Pinout conector J3.

El conector J3 tiene un aspecto similar con algún que otro matiz. Tenemos 10 señales conectadas a tierra que no podremos utilizar para nuestro diseño. El resto en principio son señales libres que podremos utilizar sin problema, a excepción de los pines 1, 7 y 8 que están destinados a funciones para controlar la frecuencia de reloj de la FPGA.
Por lo tanto, tendríamos:

- 10 pines conectados a tierra.
- 2 pines destinados a gestiones del reloj de la FPGA.
- 27 pines libres a los que podremos llevar señales.

Una vez analizados estos dos conectores pasaremos a analizar los otros dos restantes. Son similares a estos, pero se encuentran en el extremo opuesto de la tarjeta. Para ello vamos a ver una imagen del pinout de estos dos conectores.

3.3.2.3 Pinout conector J2

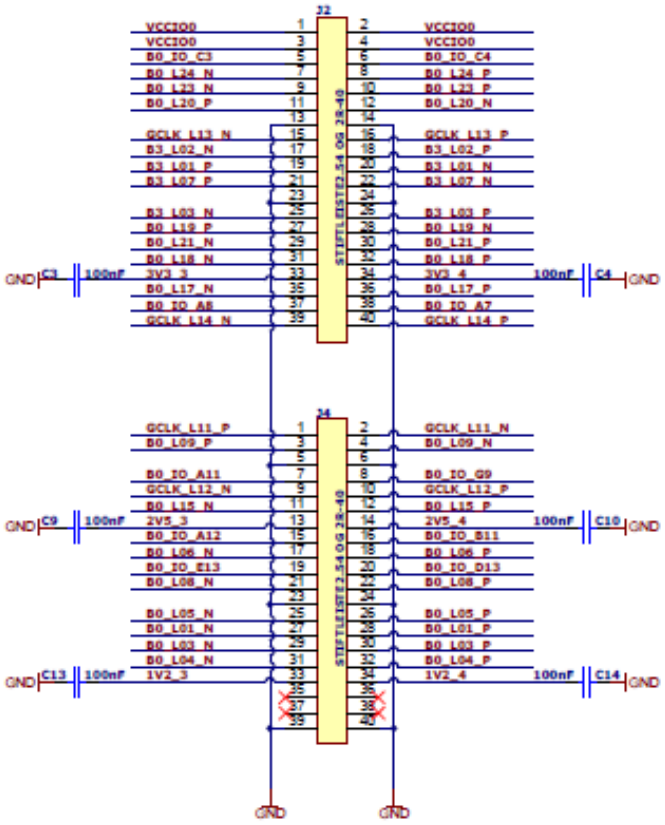

Figura 27: Pinout de los conectores J2 y J4

Los pines 1, 2, 3 y 4 denominados VCCOI0 son pines destinados a alimentación de las entradas/salidas. Tenemos, como en el caso del conector anterior, 4 pines dedicados a configurar la frecuencia del reloj de la FPGA y 6 pines conectados a tierra, que no podremos utilizar. El resto son pines libres para utilizarlos como entradas/salidas.
Por lo tanto, tendremos en este conector:

- 4 pines dedicados a alimentación de las entradas/salidas.
- 6 pines conectados a tierra.
- 4 pines relacionados con el reloj de la FPGA.

- 26 entradas/salidas con pines libres.

3.3.2.4 Pinout conector J4.

Nos queda únicamente un conector de los 4 principales por analizar. En este conector J4 encontramos de nuevo 4 pines con señales relacionadas con el reloj de la FPGA. También tenemos 10 pines conectados a tierra, que ya no podremos utilizar. El resto son pines libres a los que tenemos acceso.
En resumen:

- 4 pines relacionados con el reloj de la FPGA.
- 10 pines conectados a tierra.
- 4 pines no conectados.
- 22 pines con acceso para entradas/salidas.

Una vez hemos analizado los 4 conectores a los que llevaremos las señales de comunicación de datos y de control, hemos visto que en estos 4 conectores no tenemos ningún pin dedicado al interfaz JTAG. Como hemos comentado anteriormente en la descripción de la carrier board tiene unos conectores destinados a JTAG que vamos a describir a continuación.
Los conectores que vamos a ver son el J9, J10 y J11.

3.3.2.5 Pinout conector J9.

Vamos a comenzar con el conector J9:

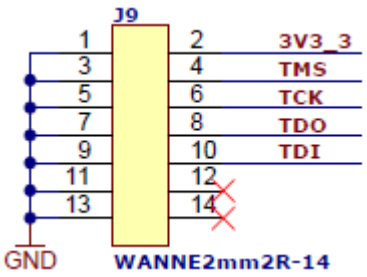

Figura 28: Pinout del conector J9

Es un conector 2 x 7 que tiene los pines 1, 3, 5, 7, 9, 11, 13 conectados a tierra mientras que el pin 2 es una entrada de 3.3V. Los pines 12 y 14 no están conectados mientras que los restantes son los destinados al interfaz JTAG.
TMS en el pin 4, TCK en el pin 6, TDO en el pin 8 y TDI en el 10.

3.3.2.6 Pinout conector J10

Pasamos ahora al conector J10, como vemos en la siguiente imagen es similar al conector J9 pero más sencillo.

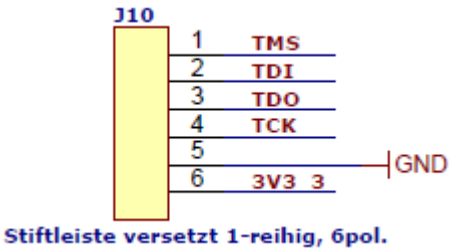

Figura 29: Pinout conector J10

Tiene únicamente una fila de 6 pines, el pin 6 debe estar conectado a 3.3V, el pin 5 a tierra y los demás son los habituales del interfaz JTAG.

3.3.2.7 Pinout conector J11

El J11 es un conector de una sola fila con 10 pines, el número 2 está no conectado, los números 1, 5 y 9 están conectados a tierra. Tenemos dos pines, el 3 y el 7 que necesitan 3.3V. Los restantes son lo pines del JTAG, TCK, TMS, TDI y TDO en los pines 4, 6, 8 y 10 respectivamente.

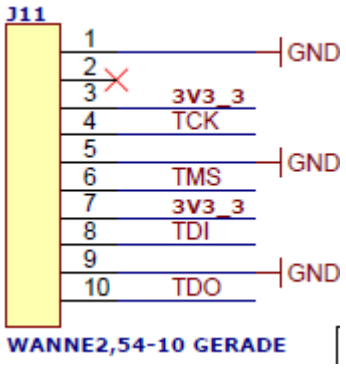

Figura 30: Pinout conector J11

Ya hemos visto la distribución de pines de todos los dispositivos que tenemos que conectar. La situación es la siguiente, tenemos en el FT2232H Mini Module una serie de señales que debemos llevar a la TE0303, a parte debemos resolver el problema de cómo alimentar nuestra placa.
Esto es lo que haremos en el siguiente apartado donde elegiremos las conexiones que habrá posteriormente en nuestro diseño y justificaremos cada una de ellas.

3.4 Conexiones en nuestra PCB.

Vamos a dividir este apartado en varios según la naturaleza de las señales de las que estemos hablando, hemos distinguido señales de alimentación, datos, configuración y el resto, que generalmente son señales de control, como algún reset.
Antes que todo esto daremos algunos detalles de los conectores que tendremos en nuestra tarjeta.

3.4.1 Conectores de nuestra tarjeta.

Para aclarar un poco el diseño de nuestra placa, aunque entraremos posteriormente más en detalle, como necesitamos crear una tarjeta para conectar el mini módulo con la carrier board de la FPGA, tendremos un conector de 2 x 20 del estilo de los conectores J1, J2, J3 y J4, luego especificaremos a cuál va conectado. Además, incorporaremos dos conectores de 2 x 13 como los del FT2232H Mini Module, donde irá conectado, y un conector JTAG de 1 x 6.

3.4.2 Conexiones de alimentación.

Cuando describimos en el capítulo 2 el FT2232H Mini Module, explicamos que había dos modos de alimentación, uno de ellos era una auto alimentación y el otro era alimentación a través del USB.
Para este proyecto hemos elegido la alimentación a través del USB porque creemos que será más robusta y estable, debido a que la alimentación viene directamente desde el host.
Recordemos que para ello necesitábamos conectar los siguientes pines los conectores de 2 x 13 del dispositivo.

- Conectar los pines VBUS a VCC (en el conector CN3, pin1 a pin3 del CN3). De esta manera utilizaremos la tensión proporcionada por el bus USB para alimentar el regulador interno del módulo.
- Conectar los pines V3V3 A VIO (pines 1, 3 y 5 del conector CN2 a los pines 11 y 21 del mismo conector y pines 12 y 22 del conector CN3 para proporcionar alimentación a los bancos de entrada/salida.

Esto sería una alimentación del USB simple. En este proyecto le hemos intentado dar una vuelta a esta alimentación, o al menos hemos encontrado una variante a esta forma de alimentación y en vez de tener que quedarnos con una de las dos, hemos intentado incorporar las dos a nuestro diseño aportándonos versatilidad y robustez, que es lo que se intenta en un prototipo como este, en caso de que alguna de las dos falle.

Cuando estábamos haciendo la descripción del pinout del conector J1 de la carrier board TE0303 dijimos que había 4 pines con una tensión saliente de 5V destinados a alimentar alguna otra placa que estuviese conectada al J1, como es nuestro caso.
Esta ha sido la razón fundamental por la cual hemos elegido el conector J1 para llevar las señales del mini módulo. Digo que ha sido la razón diferencial porque los 4 conectores cumplían con el otro requisito que era tener un mínimo de pines libres para poder acceder con todas las señales desde el FT2232H.

Debemos ahora explicar cómo hemos tratado esta cuestión de elegir entre dos tipos de alimentación. Para saber cómo encajar ambas alimentaciones debemos preguntarnos como integrar los 5V que vienen de la TE0303 en el formato de alimentación que tenemos a través del USB. En la alimentación a través del USB

conectamos el pin VBUS que es el que tiene los 5V al pin VCC, las demás conexiones que hemos hecho después es para alimentar al resto de los pines. Por lo tanto, es ahí donde está la cuestión para incorporar los 5V provenientes de la otra tarjeta.

La solución que hemos implementado en este caso es un jumper de tres entradas, en una de ellas 1 pin que vaya conectado a VBUS, otra que esté conectada a VCC y la otra conectada a los 4 pines Vb2b de 5V de la placa TE0303. Con esto lo que conseguimos es que si queremos que el circuito esté alimentado a través del USB conectaremos los pines VBUS y VCC quedando el pin Vb2b sin conectar, sin embargo, si queremos que sea la carrier board la que alimente la placa, conectaremos VCC a Vb2b quedando el pin VBUS sin conectar.
Para que nos hagamos una mejor idea quedaría de forma esquemática de la siguiente forma:

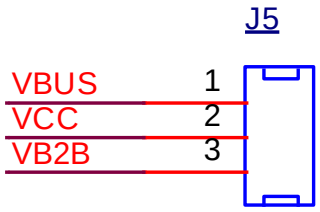

Figura 31: Jumper de tres pines para la alimentación

Como vemos en el esquema, la única condición que le debemos poner es que el pin VCC esté en el centro, para que podamos seleccionarlo tanto con VBUS como con Vb2b. El resto de las conexiones que comentamos para el modo de alimentación a través del USB deben ser permanente, tanto cuando alimentamos con USB como cuando alimentemos a través de la carrier board.

Vamos a ver en la siguiente imagen como quedaría el esquema final de la alimentación teniendo en cuenta las conexiones que debemos hacer en los conectores CN2, CN3, J1 y el jumper anteriormente descrito.

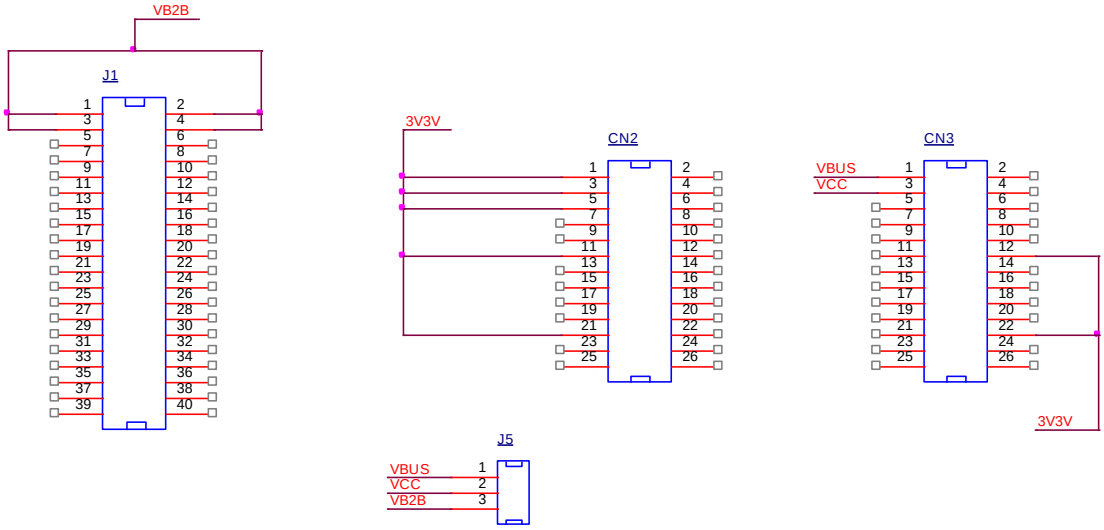

Figura 32: Conexiones generales para alimentación

3.4.3 Conexiones de datos.

En el caso de la alimentación, tenemos que conectar unos pines obligatoriamente a otros. No es el caso de las señales de datos, recordemos un momento las señales de datos que teníamos que llevar del FT2232H Mini Module a la carrier board y en qué pines los encontrábamos.

Teníamos 8 señales de datos, estas eran AD0, D1, D2, D3, D4, D5, D6 y D7, y estaban ubicadas en los pines 7, 10, 9, 12, 14, 13, 16 y 15 respectivamente del CN2 del mini módulo. Vamos a ver una imagen para hacernos una mejor idea de cómo están ubicados.

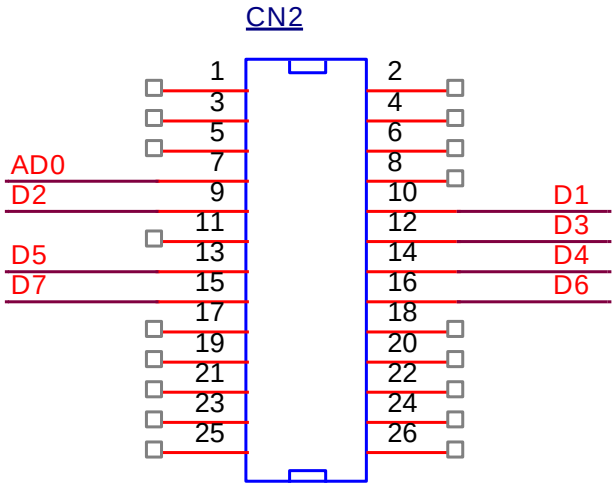

Figura 33: Esquema CN2 para señales de datos

Esas 8 señales debemos llevarlas a los pines del conector J1. No podemos llevarlas a cualquier pin, únicamente a los que dijimos anteriormente que estaban libres para entradas/salidas.

Vamos a ver también en un esquema para saber cómo quedan ubicados dichos pines. En el esquema únicamente se representan aquellos pines libres para entradas y salidas, los que no son para ello están representados con una x, pero pueden ser utilizados para otras señales de otro tipo, como alimentación.

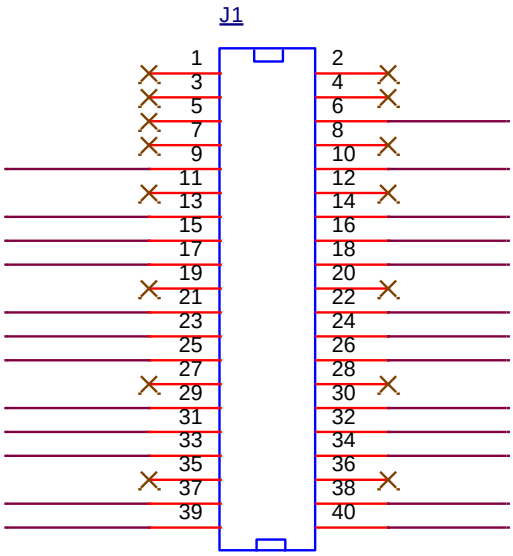

Figura 34: Esquema pines disponibles para señales en J1

De esos pines debemos elegir 8 a donde deben llegar las 8 señales de datos. Para ello debemos tener en cuenta la orientación que va a tener el conector J1 en la PCB. Como el conector J1 está girado 180º, con respecto al del anterior esquema, en la carrier board, así pondremos nosotros también nuestro conector J1 para que coincida físicamente con el de la tarjeta que contiene la FPGA.

El proceso de elegir a qué pines conectamos las 8 señales de datos, lo he hecho a través de varios croquis a mano, teniendo en cuenta las demás señales que vamos a ver después y las alimentaciones, para que no haya cruces entre ellas o al menos los menos posibles. El resultado de la elección de los pines en el conector J1 lo podemos ver en la siguiente imagen.

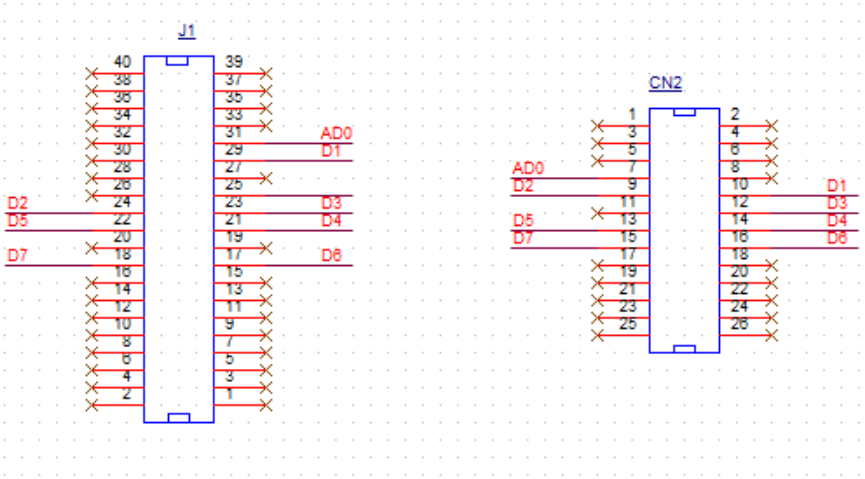

Figura 35: Esquema para conexión de datos

Esta sería la distribución elegida, teniendo en cuenta las señales que deberemos mandar más tarde, como son las de control y las alimentaciones que antes hemos tratado ya.
En esta imagen el conector J1 ya está correctamente orientado, por si quedaba alguna duda de cómo sería su posición final.

3.4.4 Conexiones de programación. JTAG.

Como vimos en la descripción del FT2232H Mini Module, los pines del interfaz de configuración JTAG se encuentran en el conector CN3, concretamente en los pines 23, 24, 25 y 26. Para verlo mejor pondremos una imagen del conector CN3 únicamente con las señales del JTAG.

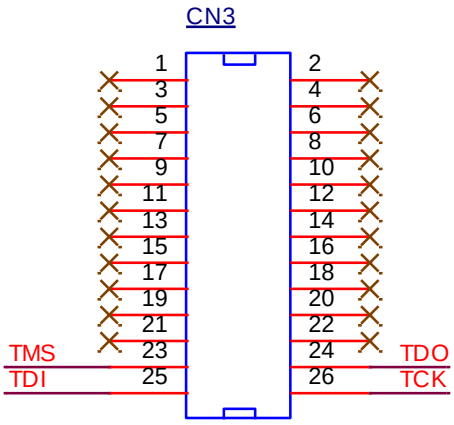

Figura 36: Conexiones del JTAG en CN3

Estas señales tenemos que llevarlas a la carrier board, igual que hicimos con las de control. No podemos llevarlas al conector J1 porque no son señales de entrada/salida, sino que se tienen que conectar a unos pines determinados que también estén destinados a configuración JTAG.
Los conectores de la carrier board que contienen estos pines configurados para JTAG son el J9, J10 y J11. El problema que se nos presenta es que tienen una posición muy complicada para acceder a ellos a través de nuestra placa, por lo que la solución que hemos llevado a cabo es la siguiente.

En vez de llevar los pines del JTAG a los conectores de la carrier board, incorporaremos en nuestra placa un conector 1 x 6 destinado al JTAG y llevaremos las señales a este conector. Más tarde, cuando el montaje esté completo llevaremos las señales hasta la FPGA mediante un conector con cable.
El conector que vamos a poner en nuestra PCB es similar al J10, un diseño sencillo. En la siguiente imagen podemos ver mejor como quedaría el conector.

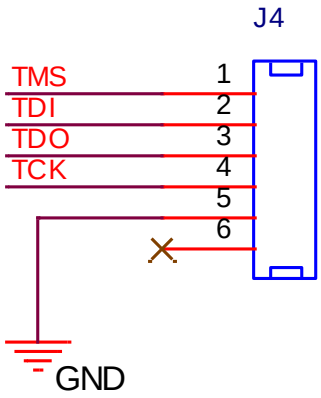

Figura 37: Conector 1 x 6 destinado a JTAG

Esos cuatro pines de JTAG, TMS, TDI, TDO y TCK irán conectados a los 4 del conector CN3.

3.4.5 Señales de control.

Las señales de control las tenemos divididas en dos, las que vienen del mini módulo y tenemos que llevarlas al conector J1 de la carrier board y las que vienen de la FPGA hacia la carrier board y tenemos que llevarlas a alguno de lo pines que quedaban como no conectados del dispositivo FT2232H.
Las primeras señales que hemos comentado, las que van hacía la carrier board son las siguientes: RESET#, CLK, OE_N, RD_N, WR_N, TXE y RXF. La primera es una señal del reloj del dispositivo FT2232H y las demás son del modo de funcionamiento 245 FIFO. Del segundo tipo únicamente tenemos una señal que es /MR que es un máster reset, para reiniciar la FPGA.

3.4.5.1 Señales de control del FT2232H Mini Module.

Vamos a comenzar por las señales que tenemos que llevar desde el conector CN2 hasta la carrier board. Como hemos dicho anteriormente, las señales que tenemos que llevar son: RESET#, CLK, OE_N, RD_N, WR_N, RXF y TXE y están en los pines 8, 24, 23, 20, 19, 18 y 17.
Para que podamos hacernos una idea mejor, vamos a poner el esquema a continuación con las señales de control del CN2. Únicamente están estas señales no las anteriormente presentadas ya, ni las que faltan por conectar.

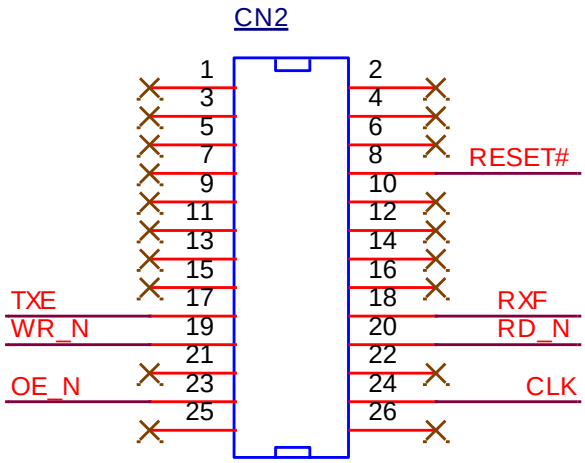

Figura 38: Esquema del CN2 para señales de control

Hemos seguido el mismo proceso para llevar estas señales a la carrier board que con las señales de datos, a través de varios croquis a mano alzada teniendo en cuenta todas las señales que tenemos que llevar e intentando que no se cruzasen. El resultado en el conector J1 es el siguiente. Solamente hemos presentado como ha quedado el conector con las señales de control conectadas, no se muestran el resto de las señales.

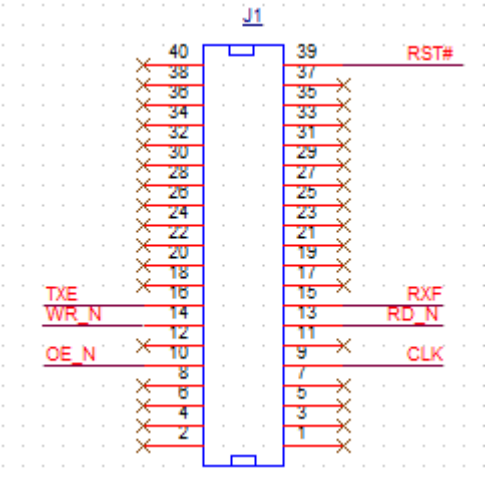

Figura 39: Esquema del J1 para señales de control

Al igual que en las señales de datos hemos presentado el resultado con el conector J1 en la orientación que va a tener finalmente en la PCB.

Como podemos observar, la señal de reset se llama de diferente forma en un conector que, en el otro, esto es debido a que hay algunas conexiones entre medias de los dos pines. Vamos a explicar a continuación esta señal.

La señal reset es activa a nivel bajo, entonces, para que no haya problemas de conflicto, haremos un pull-up para mantener esta señal a 1. Esto evita que se hagan lecturas erróneas. El pull-up se hace a través de una resistencia y una fuente de tensión, en este caso a 3.3V y la resistencia tiene un valor de 10kohm.

A mayores del pull-up le hemos añadido un jumper de dos pines con uno de ellos conectado a tierra por si queremos no utilizar esta señal, en un primer momento íbamos a poner un interruptor para la señal, pero al ser un prototipo pensamos que con esta solución temporal sería suficiente. Vamos a ver a continuación un esquema de cómo queda la conexión del pull-up y del jumper de 3 pines.

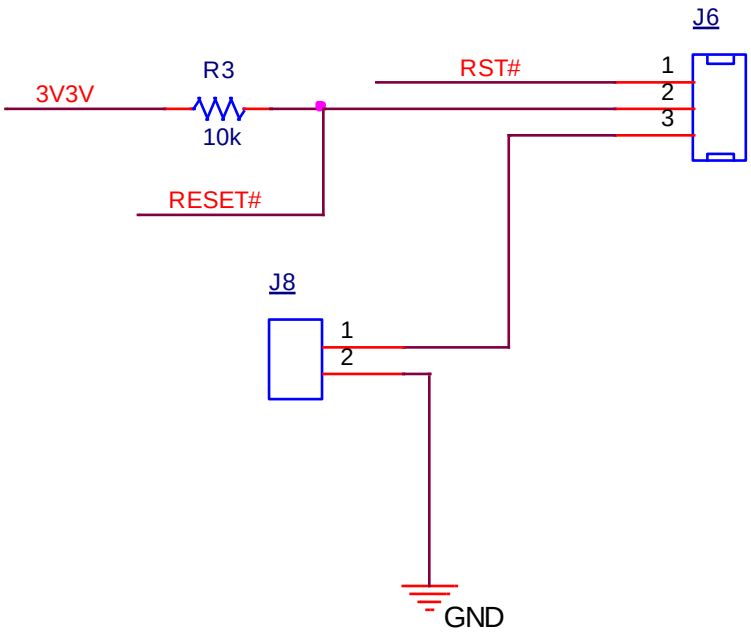

Figura 40: Esquema conexión RESET#

3.4.5.2 Señales de control de TE0303.

Comentamos al inicio del anterior apartado que teníamos una señal que llevar desde la carrier board hasta el FT2232H Mini Module. La señal es /MR, un máster reset que reinicia la FPGA.

Esta señal sale del pin 6 del conector J1 y debemos conectarlo a alguno de los pines que están no conectados del USB. En función de las conexiones que ya hemos hecho, con datos, JTAG, alimentaciones y las anteriores de control, hemos decidido conectarla al pin 5 del CN3.

Vamos a ver una imagen para hacernos idea mejor.

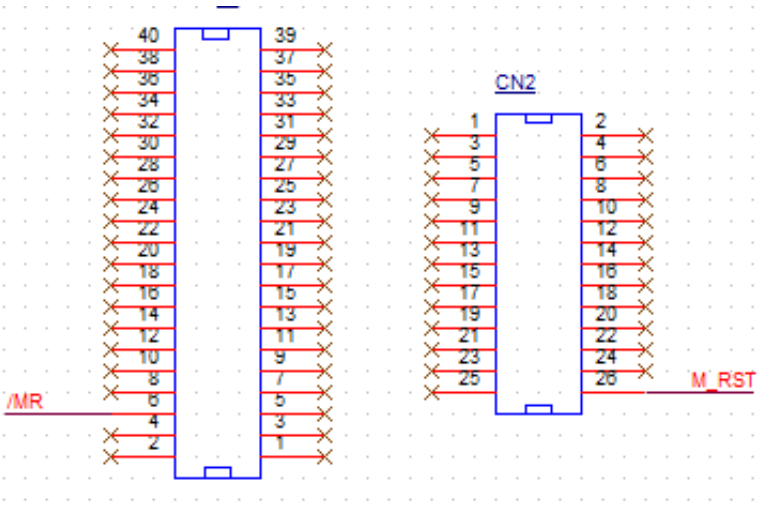

Figura 41: Esquema señal /MR conectores J1 y CN2

Como vemos en esta imagen, ocurre igual que con la señal de reset en el anterior apartado, como la señal /MR es activa a nivel bajo debemos mantener la señal a 1 con un pull-up para no tener problemas. Podemos ver en la siguiente las conexiones que existen entre ambos pines de esta señal. Lo mismo nos ocurre con las señales de control RD_N y WR_N y lo que hemos decidido es conectar 3 resistencias de 10k ohmios como indica la figura inferior.

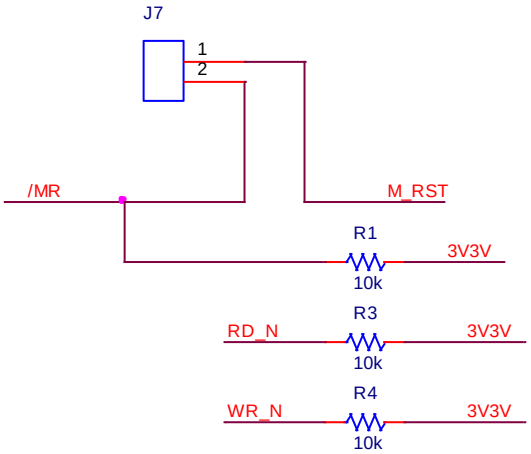

Figura 42: Esquema pull-up señales control

Para las otras dos señales de control que son de salida, hemos creído conveniente conectarlas también a un pull-up debido que son activas a nivel bajo. En este caso las resistencias que hemos elegido son de 150 ohmios.

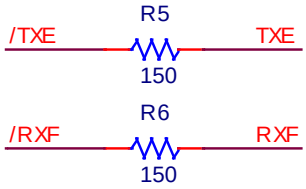

Figura 43: Esquema señales TXE y RXF

Con esto hemos concluido la presentación de señales de control tanto de la carrier board al controlador de USB como en sentido contrario.

3.4.5.3 Señales conectadas a tierra.

Por último, vamos a ver que pines de los conectores J1, CN2, CN3 y JTAG deben estar conectados a un plano de tierra.
Los pines que deben estar conectados a tierra vienen determinados en la hoja del fabricante por lo tanto no hay ninguna explicación adicional para este apartado. Vamos a ver en el siguiente esquema los pines a tierra de los conectores J1, CN2 y CN3.

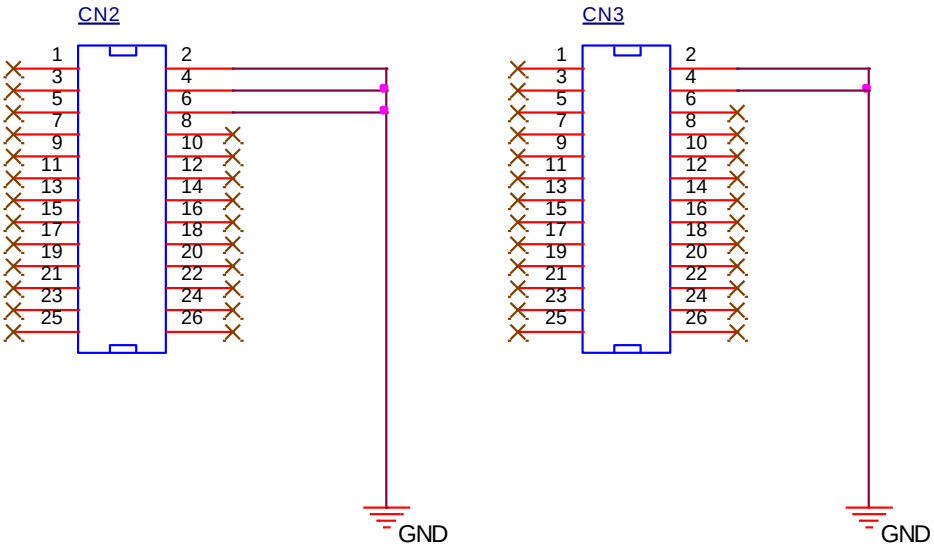

Figura 44: Esquema pines conectados a tierra en CN2 y CN3

Vemos ahora las conexiones a tierra en el conector de JTAG.

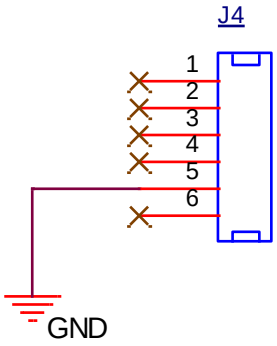

Figura 45: Esquema pines conectados a tierra en JTAG

3.5 Presentación esquemas en OrCad Capture CIS.

Vamos a presentar los resultados respetando el orden que hemos seguido hasta ahora. Por lo tanto, lo dividiremos en: alimentación, datos, configuración, control y un plano general.

3.5.1 Esquema alimentación.

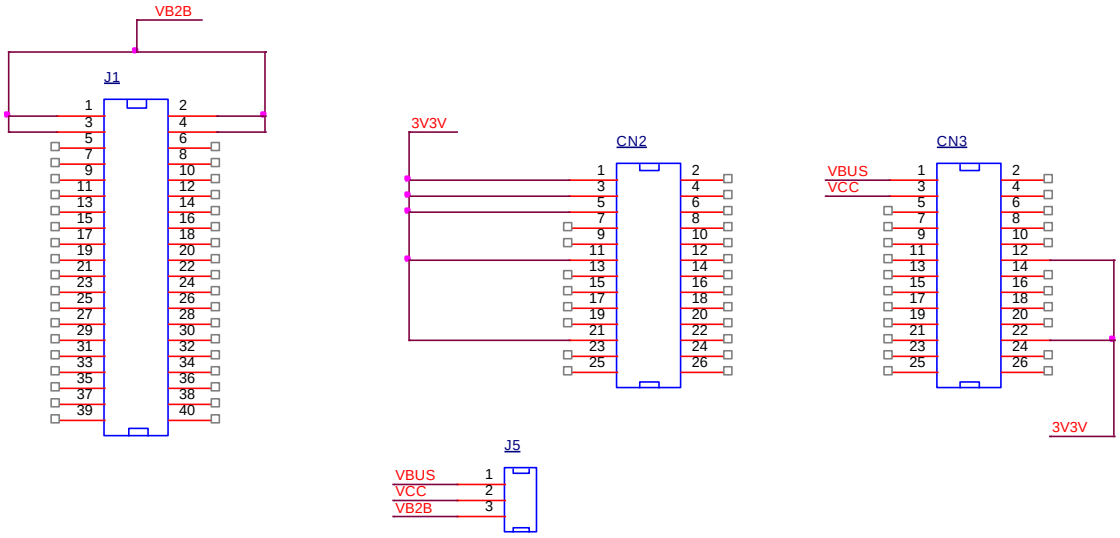

Figura 46: Conexiones generales para alimentación

3.5.2 Esquema datos.

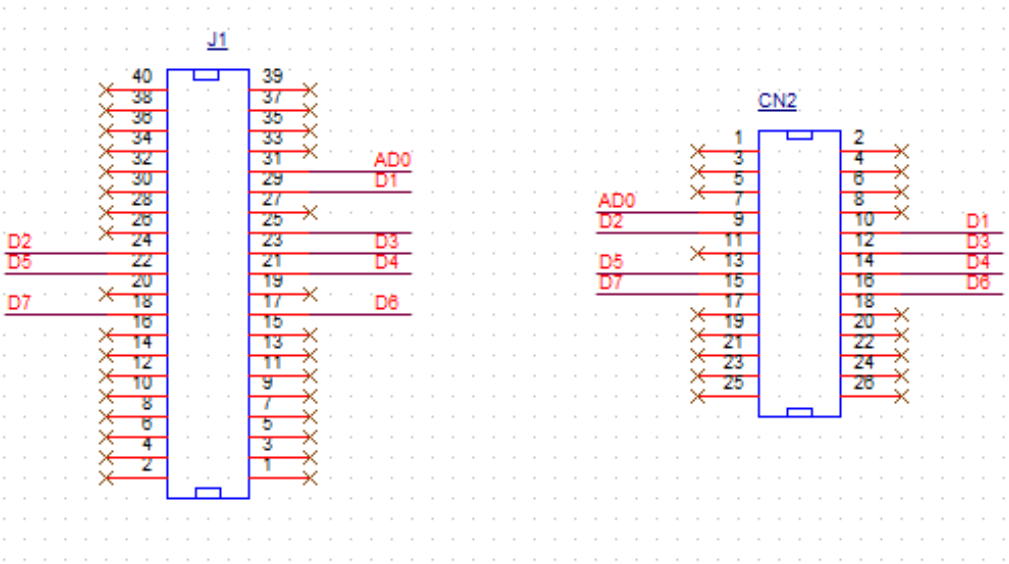

Figura 47: Esquema general de datos

3.5.3 Esquema configuración. JTAG.

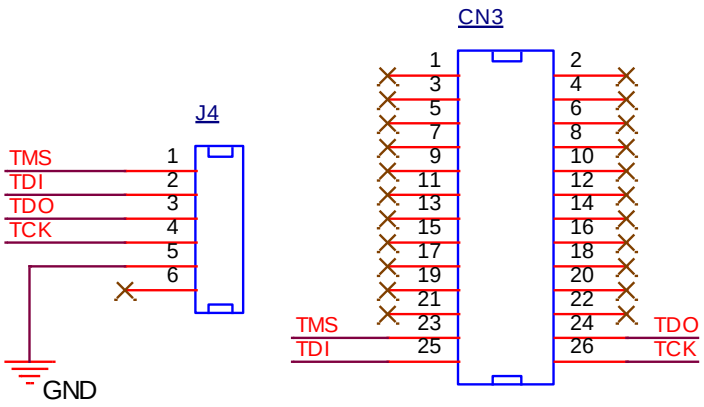

Figura 48: Esquema general configuración. JTAG

3.5.4 Esquema control.

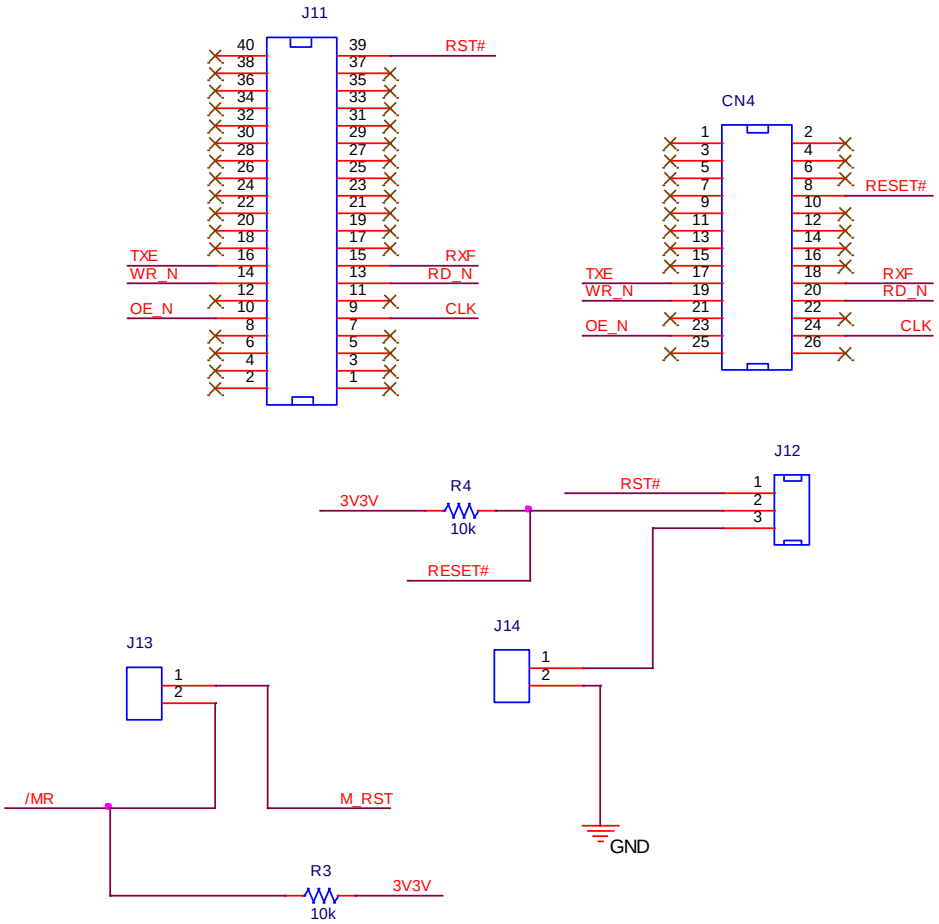

Figura 49: Esquema general control

3.5.5 Esquema general

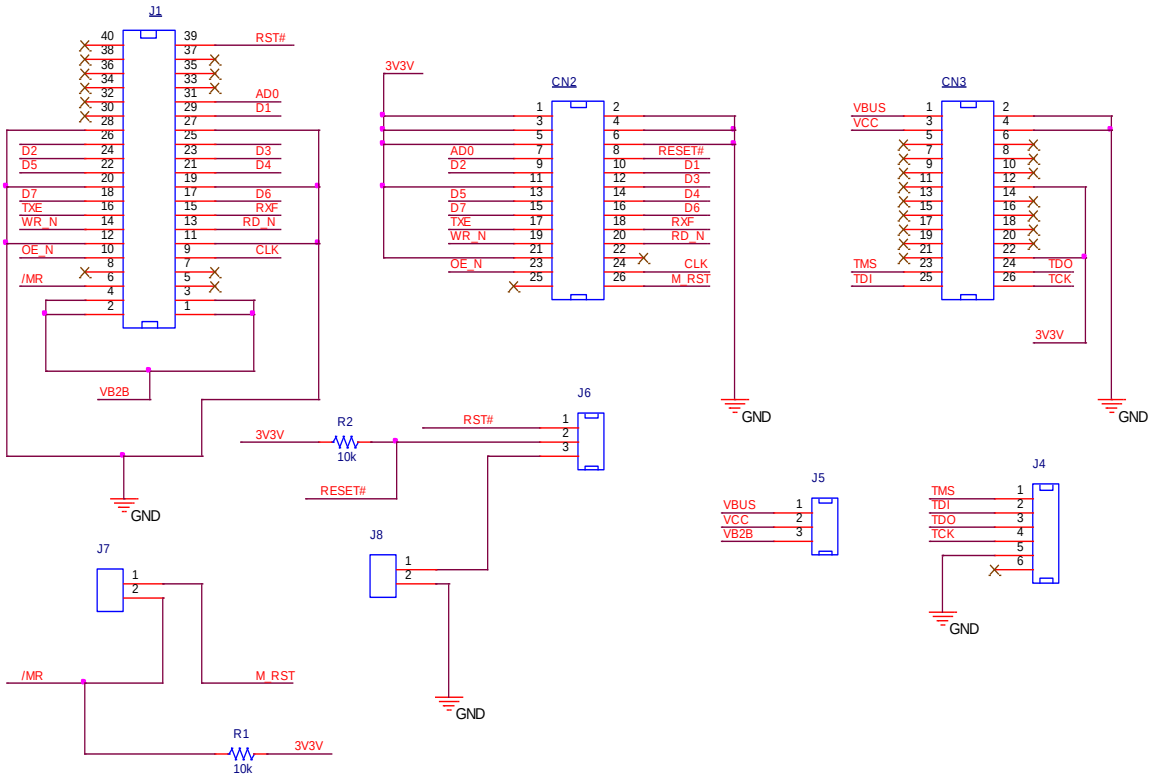

Figura 50: Esquema general

4. Diseño de la tarjeta

4.1 Consideraciones generales.

En el anterior capítulo nos centramos en los esquemas necesarios para las conexiones entre pines que iba a haber en nuestra tarjeta. Una vez tenemos los esquemas hechos en OrCad Capture CIS, pasamos a la otra herramienta que vamos a utilizar, PCB Editor, con la cual vamos a poder diseñar nuestra tarjeta.

4.2 Flujo de diseño

Estos pasos que vamos a ver a continuación serán los que daremos para un correcto planteamiento del trabajo.

- Diseño de los footprints de los conectores que debemos colocar en nuestra tarjeta. Deberán tener los mismos parámetros que los conectores físicos para recrearlos en el software.
- Asociación de los footprints a los conectores que hemos colocado en los esquemas. No solo debemos hacerlo de conectores sino también de resistencias y demás elementos que vayan colocados en nuestra tarjeta y que ocupen un espacio físico.
- Creación de la tarjeta, debemos crear nuestro espacio de trabajo que más tarde será la tarjeta, tendremos en cuenta el tamaño real que debe tener, el número de capas que utilicemos, así como el material.
- Layout de los componentes en la tarjeta. Una vez hayamos creado la tarjeta, importaremos los footprints desde Capture CIS, para poder trabajar con ellos. En este paso decidiremos dónde se colocan finalmente los componentes.
- Routing. Comenzamos a llevar las señales que anteriormente habíamos definido, especificando el espesor de las señales, en que capa se encuentran y demás aspectos a determinar.
- Por último, crearemos los planos del diseño para poder mandarlo a fabricar.

Una vez hemos hecho una introducción con los pasos que hemos seguido, comenzaremos a trabajar en el diseño.

4.3 Diseño de los footprints

Todo elemento que vaya a ir colocado en nuestra tarjeta debe tener su footprint correspondiente, para ello, debemos hacer un recuento de los componentes que necesitamos para poder crear cada una de las huellas.
A saber, en nuestro diseño contaremos con:

- Un conector 2x20, al que denominamos J1.
- Dos conectores de 2x13, denominados CN2 y CN3.
- Dos jumpers de 3 pines, J5 y J6.
- Un conector de 1x6 para JTAG, al que hemos llamado J4.

- Dos jumpers de 2 pines, J7 y J8.
- Dos resistencias de 10k llamadas R1 y R2.

Por lo tanto, vamos a seguir ese orden para su creación.

4.3.1 Conector J1

Acabamos de comentar que el conector J1 tiene dos tiras de pines de 20 cada una, por lo tanto, es un conector de 2x20. Para su diseño debemos tener en cuenta el conector que le vamos a colocar encima, debido a que debemos respetar el espacio que vaya a ocupar dicho conector y por supuesto tendremos que dimensionar los agujeros de nuestra placa para que quepan las patillas del conector.

En la tarjeta que vamos a usar necesitaremos conectores macho, por lo que hemos buscado en el fabricante Trenz Electronics algún componente que nos pueda servir.

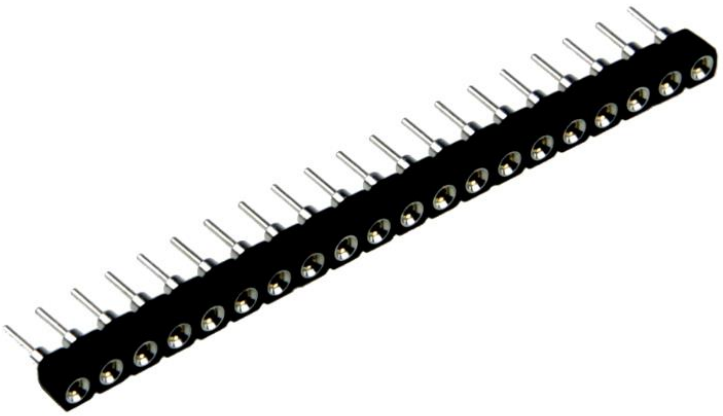

Figura 51: Conector hembra de 20 pines

Un conector como el de la figura superior nos viene bien para nuestro diseño. Para el conector J1 deberíamos poner dos como estos en paralelo.
A la hora de crear nuestro footprint, como hemos comentado antes, nos hace falta saber el tamaño del conector tanto de ancho como de largo y el diámetro de las patillas.

El conector de largo mide 5.2 cm y de ancho alrededor de 0.5 cm. Por lo que esos datos tomaremos a la hora de crearlo. En cuanto a las patillas del conector son de 0.5 mm.
Esto nos afecta para dimensionar el agujero de los pines en los que ira conectado. Como la patilla del conector es cuadrada debido a que esta torneada, su diagonal haciendo el cálculo son 0.7071 milímetros, esto pasado a pulgadas serían 0.0278 pulgadas, lo que equivale a 28 milésimas de pulgada, que es la medida que tienen los agujeros en el programa. Debemos dejar al menos 10 milésimas de pulgada de

tolerancia por lo que cogeremos un agujero circular de 35 mils de diámetro. Esto nos valdrá para todos los pines ya que los conectores que usaremos serán los mismos.

Para elegir el pad, tomaremos 20 mil más de lo que hemos tomado como diámetro del agujero, por lo tanto, nos iremos mínimo a 55, y el siguiente que encontramos es 60 por lo que elegiremos este. Pad60cir35d será nuestra elección para todo el trabajo.

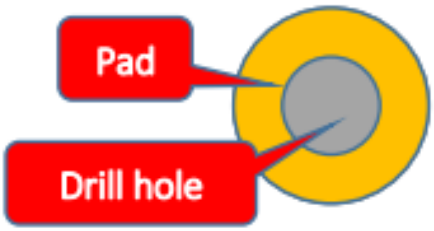

Figura 52: Agujero y pad

Con las consideraciones que hemos hecho anteriormente el footprint del elemento quedaría de la siguiente manera.

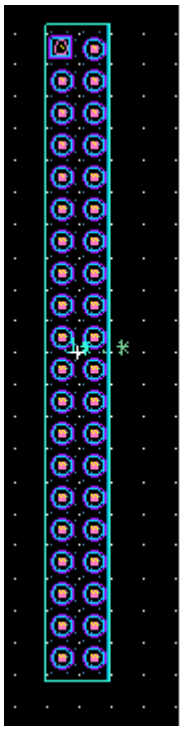

Figura 53: Footprint conector J1

4.3.2 Conectores CN2 y CN3

Englobamos a estos dos conectores en este apartado debido a que son idénticos. La única diferencia reseñable respecto del anterior conector es que este tiene dos tiras de pines de 13 cada una. El conector que usaremos en este caso será el mismo, pero

con 7 pines menos por fila, por tanto, el agujero y el pad elegidos en el anterior caso, nos valdrá también para este.
En este caso el footprint del elemento nos quedaría de la siguiente manera.

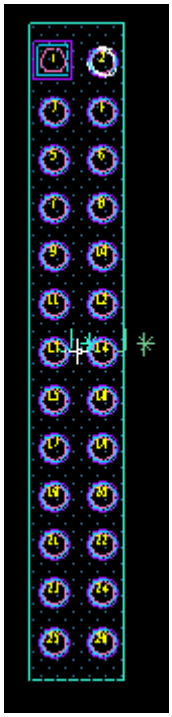

Figura 54: Footprint conectores CN2 y CN3

Podemos distinguir en diferentes colores el contorno exterior, los agujeros y el pad.

4.3.3 Conector J4

Este conector es el dedicado al interfaz de configuración JTAG y tiene una composición de una única fila con 6 pines. Ya hemos descrito en otros capítulos las señales incorporadas en este conector. Como hemos dicho en los demás casos usaremos un conector similar por lo que los pines elegidos serán los mismos, tanto el agujero como el pad.
El footprint del conector quedaría de la siguiente manera.

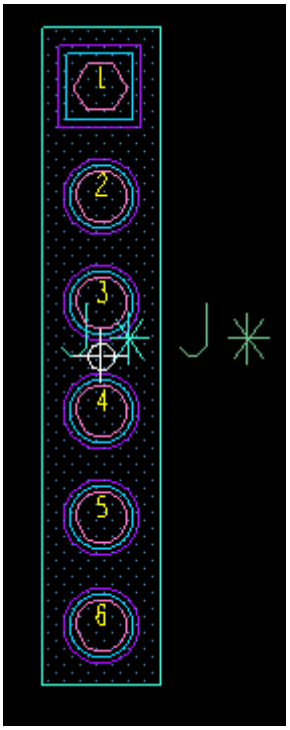

Figura 55: Footprint conector J4

4.3.4 Resistencias 10k

Al igual que hicimos con los conectores debemos mirar en algún fabricante unas resistencias de 10k. Si nuestra tarjeta fuese un producto final para comercializar usaríamos resistencias encapsuladas, pero en este caso al ser un prototipo hemos elegido resistencias de superficie.
El fabricante elegido ha sido RS Components, eligiendo la siguiente resistencia.

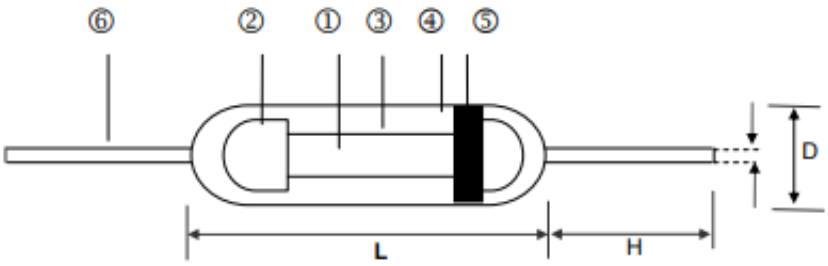

Figura 56: Resistencia 10k

Type	L	D	H	d	Weight (g) (1000pcs)
Carbon 0.125W	3.3+0.4/-0.2	1.8±0.3	29.3±2.0	0.452.3±0.03	92
Carbon 0.25W	6.3±0.5	2.3±0.3	28±2.0	0.55±0.03	155
Carbon 0.5W (H)	6.3±0.5	2.3±0.3	28±2.0	0.55±0.03	155
Carbon 1W (H)	9.0±0.5	3.2±0.5	26±2.0	0.65±0.03	352
Carbon 2W (H)	11.5±1.0	4.5±0.5	35±2.0	0.78±0.03	775

Tabla 14: Dimensiones resistencia

En función de la tabla anterior hemos elegido dejar una distancia entre pines de 4 décimas de pulgada. El footprint del elemento nos quedaría de la siguiente manera.

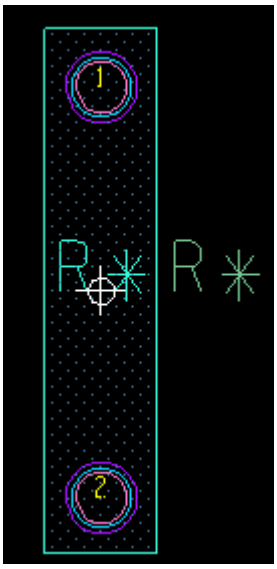

Figura 57: Footprint resistencia R1 y R2

4.3.5 Jumpers

En el apartado de jumpers necesitamos de dos tipos, dos de 3 pines y otros dos de 2 pines. Han sido creados de la misma forma y tienen la misma distancia entre pines que los demás elementos, décima de pulgada. Los jumpers los trataremos como si fuesen un conector de dos o tres pines.

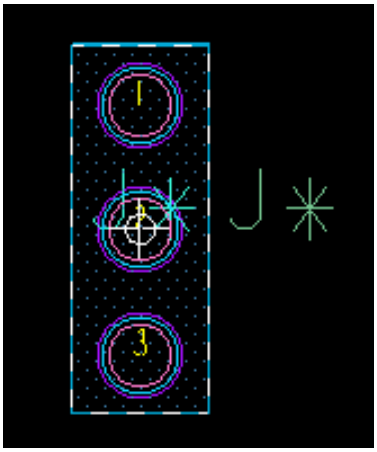

Figura 58: Footprint Jumper 1x3

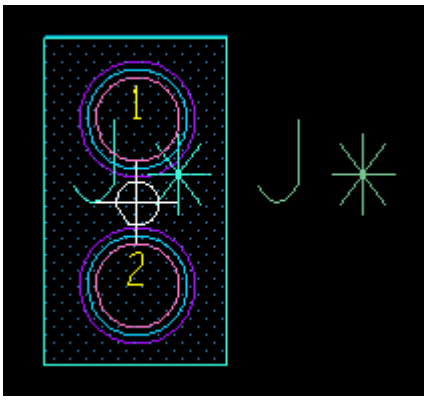

Figura 59: Jumper 1x2

Ahora que hemos creado todos los footprints de los elementos, el siguiente paso será asignar los footprints a los componentes en el esquema, para más tarde tenerlos asociados en PCB Editor.

4.3.6 Resistencias 150 ohmios

Como hemos explicado en el capítulo anterior, necesitamos incorporar para las señales de control TXE y RXF dos resistencias de 150 ohmios, para un pull-up. Las dimensiones de las resistencias que hemos elegido las podemos ver en la siguiente figura.

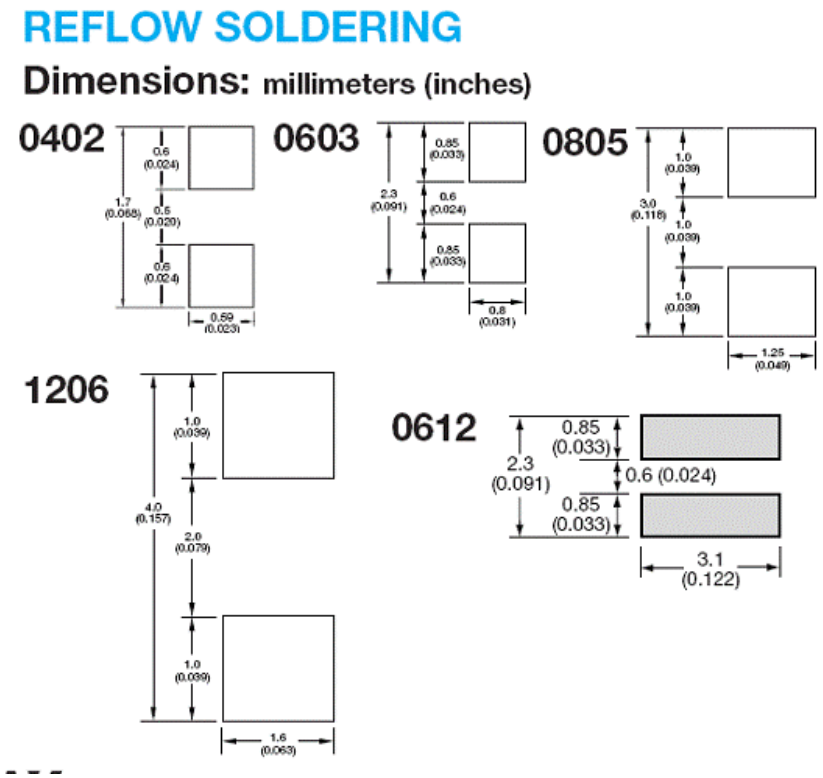

Figura 60: Dimensiones resistencia 150 ohmios

En este caso hemos elegido una resistencia 0805. Las unidades están en milílimetros (pulgadas).

El footprint que hemos elaborado de esta resistencia es el siguiente.

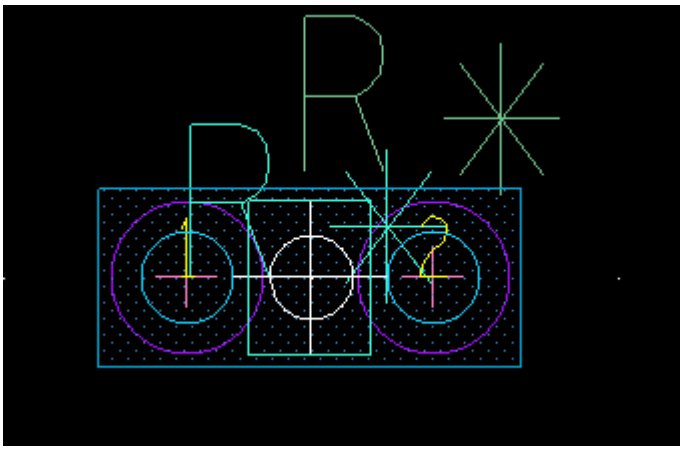

Figura 61: Footprint resistencia 150 ohmios

4.4 Asociar footprints a componentes en el esquema.

Para ello, debemos abrir Capture CIS, e ir al esquema que hayamos realizado de todos los componentes. Una vez hayamos hecho esto, seleccionaremos todos los

componentes y pulsaremos el botón derecho del ratón para que se nos despliegue el menú de opciones. Iremos a Edit Properties y allí veremos la siguiente ventana.

		Location Y-Coordinate	MAX_TEMP	Name	Part Reference	PCB Footprint	POWER	Power Pins Visible
1	SCHEMATIC1 : PAGE1	150		INS251	CN2	CONECTOR3		
2	SCHEMATIC1 : PAGE1	150		INS315	CN3	CONECTOR3		
3	SCHEMATIC1 : PAGE1	100		INS14368	J1	CONECTOR2x20		
4	SCHEMATIC1 : PAGE1	420		INS396	J4	CONECTOR1x6		
5	SCHEMATIC1 : PAGE1	430		INS431	J5	JUMPER1x3		
6	SCHEMATIC1 : PAGE1	370		INS7481	J6	JUMPER1x3		
7	SCHEMATIC1 : PAGE1	480		INS7597	J7	INTERRUPTOR		
8	SCHEMATIC1 : PAGE1	470		INS7611	J8	INTERRUPTOR		
9	SCHEMATIC1 : PAGE1	600	RTMAX	INS7509	R1	RESISTENCIA1k	RMAX	
10	SCHEMATIC1 : PAGE1	380	RTMAX	INS7525	R2	RESISTENCIA1k	RMAX	

Figura 62: Ventana Edit Properties

Como vemos en la anterior imagen tenemos un apartado para cada componente, veremos en qué componente estamos en la columna de Part Reference, mientras que en la ventana de PCB Footprint tendremos que rellenar cada fila con el nombre del footprint creado anteriormente. Debe ser exactamente el mismo nombre y ambos ficheros, el del esquemático y el de los footprints deben estar en el mismo archivo global, porque si no habrá problemas al asociarlos.

4.5 Creación de la PCB.

Ya tenemos asociados los footprints a los componentes, por tanto, es hora de crear nuestra base para el trabajo, la tarjeta, con todas sus dimensiones.

Para ello debemos abrir PCB y crear una nueva board. Vamos a elegir una PCB rectangular, de dimensiones 6mm de alto por 5mm de ancho, estas medidas, que pudiesen parecer un tanto aleatorias, son el resultado de un proceso de optimización del espacio en la tarjeta. Comenzamos con tamaños mucho mayores hasta que hemos conseguido reducirla hasta estas dimensiones.

La tarjeta que vamos a fabricar tendrá dos capas y ambas serán tanto de alimentación como de señales, esto lo hemos creído oportuno debido a que la densidad de señales no es muy grande como para hacer más capas intermedias, pero tampoco podemos hacerlo todo en una parte debido a que habría demasiados cruces de señales, por lo que hemos decidido que esta sería la mejor opción.
Otra de las opciones que deberíamos tener en cuenta a la hora de crear la tarjeta será la rejilla que usemos para el diseño, en nuestro caso, hemos elegido rejilla cuadrada de 1 décima de pulgada de espacio, debido a que todos los espacios entre pines son de esa dimensión.

Una vez hayamos creado la tarjeta nos quedaría algo similar a lo de la siguiente imagen, como podemos ver las líneas exteriores son el contorno de la placa.

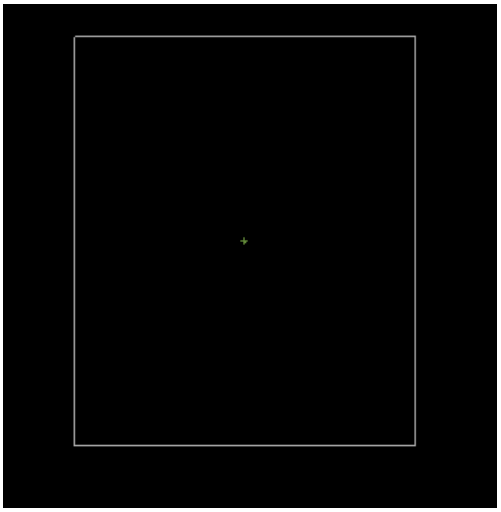

Figura 63: Contorno tarjeta en PCB Editor

4.6 Layout de la PCB

El siguiente paso es importar los archivos del programa Capture CIS a PCB Editor, para ello, con la hoja donde tenemos el esquemático seleccionada en el árbol del programa, debemos ir a Tools y seleccionar Create Netlist.

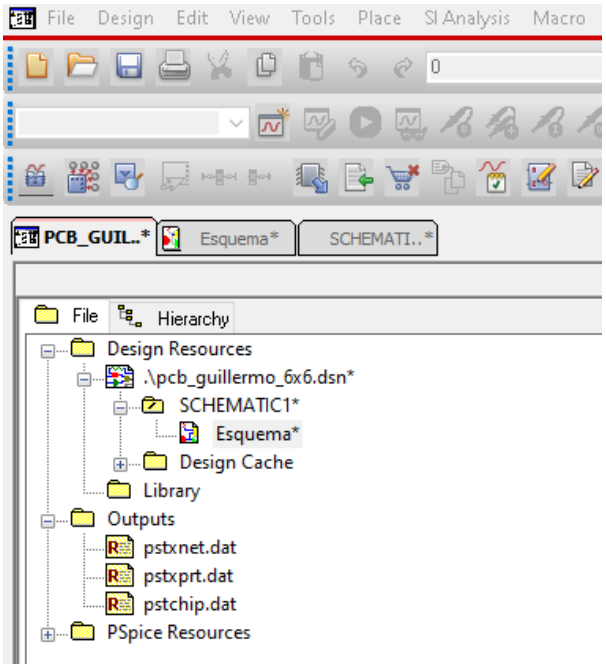

Figura 64: Árbol del Capture CIS

Al abrirse la ventana para elegir las posibilidades que tenemos al importar los archivos, lo que debemos tener en cuenta es importarlas a la tarjeta que hemos creado anteriormente.

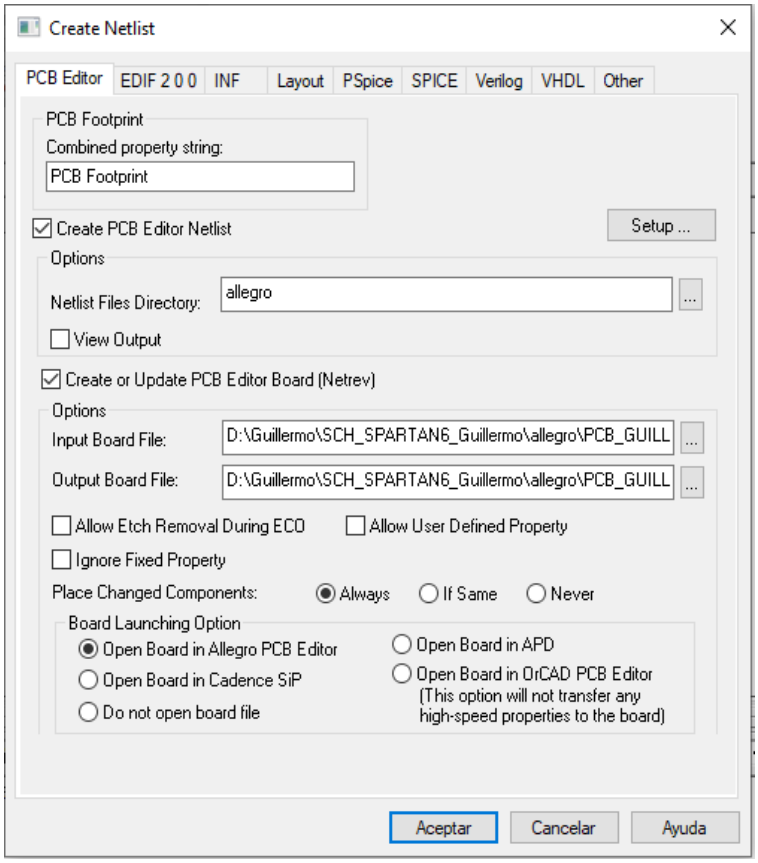

Figura 65: Create a Netlist

Tanto el apartado Input Board File como en Output Board File debemos rellenarlo con el archivo donde anteriormente guardamos nuestro diseño de tarjeta.

Cuando le demos a Aceptar el programa nos abrirá el PCB Editor y una vez en el comenzaremos a orientar y colocar los elementos de acuerdo con nuestros requerimientos.

En la imagen siguiente vamos a ver la disposición que hemos elegido nosotros y en base a qué condicionantes o restricciones lo hemos tenido que hacer.

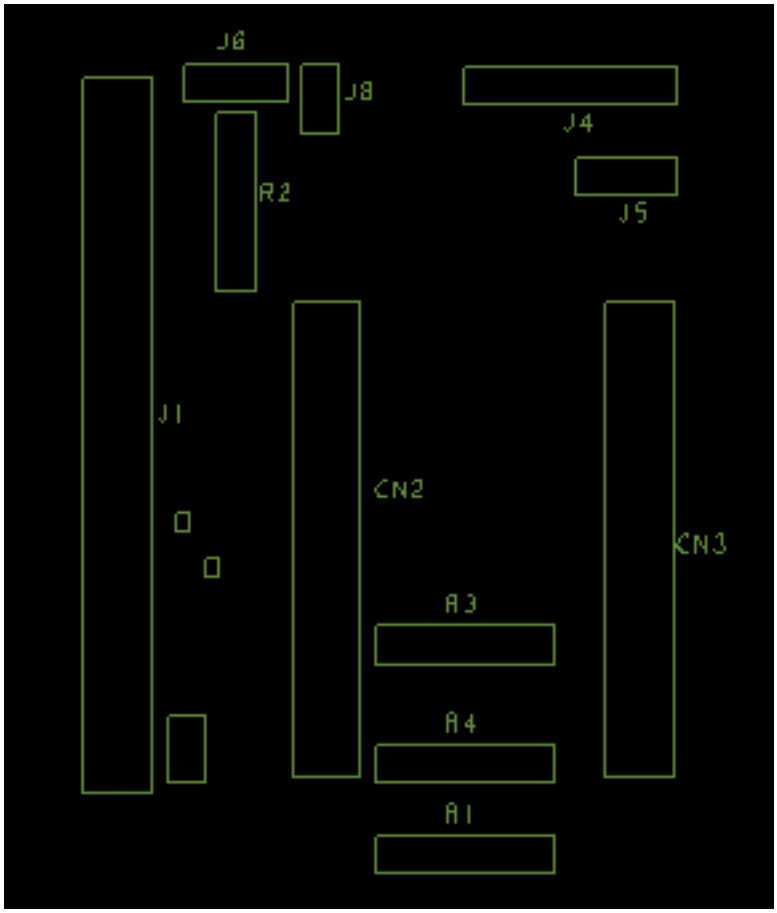

Figura 66: Disposición componentes PCB

Las restricciones o criterios que hemos tenido en cuenta han sido:

- El CN2 y el CN3 deben estar separados 8 décimas de pulgada, debido a que son dos conectores del mismo dispositivo, el controlador de USB, por lo que esa distancia no puede variar para que encaje cuando se proceda a su montaje.
- El pin 1 del conector J1 se encuentra en la parte inferior derecha del mismo, esto es para que coincida con el mismo pin del conector J1 de la carrier board TE0303.
- El conector J4, que es el que utilizamos para el interfaz JTAG, debe estar lo más cerca posible de la parte superior de la tarjeta, debido a que su conexión en tras la fabricación será por cable y el cable con el que contamos tiene una distancia limitada.
- El conector J5, que es un jumper de 3 pines, se ocupa de qué tipo de alimentación querremos a nuestra tarjeta, por lo tanto, debe estar cerca del conector CN3, que es el que tiene pines dedicados a la alimentación y desde donde salen las señales al jumper.

El resto de los elementos se han orientado y posicionado de acuerdo con que las señales no se cruzasen más tarde cuando pasemos a conectarlas. Los conectores CN2 y CN3 están orientados verticalmente ya que así ahorramos algo de espacio, puesto que la distancia vertical de la placa de 6cm es inevitable ya que está condicionada por el largo del conector J1, pero el ancho si que podíamos reducirlo si lo colocábamos de esta manera.
Como hemos dicho antes con el tamaño de la tarjeta, que los componentes se encuentren distribuidos de esta manera, es el resultado de un proceso de prueba y error en el que hemos tenido que empezar de cero varias veces.

4.7 Routing de la PCB

Vamos a comenzar este apartado, dando unas pequeñas explicaciones acerca de la distribución de señales que hemos seguido para este trabajo.
Como hemos dicho anteriormente cuando estábamos diseñando la base de la tarjeta, hemos elegido un diseño a dos caras, en el que ambas caras tienen señales y alimentación, mientras que en una de ellas va el plano de tierra.
A continuación, veremos por qué está compuesta cada cara:

- Capa inferior: En esta cara tenemos todas las señales de datos y configuración, así como señales de control. También encontraremos las señales de 5V que corresponden a la alimentación. Por último, es en esta capa en la que encontraremos todos los conectores, resistencias y jumpers. Hemos elegido que sea la capa inferior la de mayor densidad de señales y la que tiene los conectores debido a que nos favorece para el montaje posterior.
- Capa superior: En la capa superior domina el plano de tierra la mayor parte de la superficie y en el resto tenemos únicamente la alimentación de 3.3V.

Para que nos hagamos una mejor idea vamos a ver una imagen de como quedarían ya conectadas ambas capas.

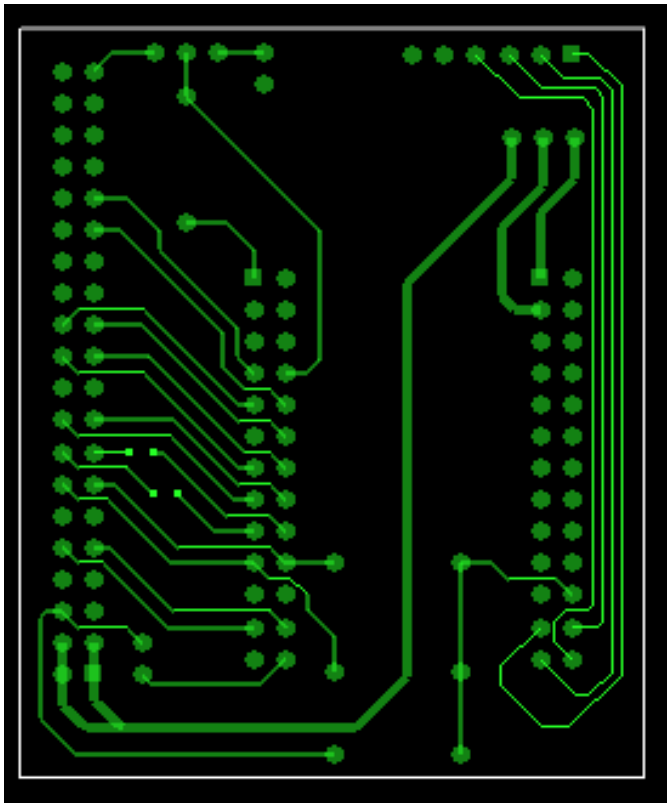

Figura 67: Routing capa inferior

Como podemos ver, las señales con mayor grosor son las de alimentación, debido a que por ellas pasa mayor intensidad de corriente. Hemos decidido hacer un diseño en el que las señales del CN2 que van hacia el J1 pasen a través del espacio entre sus pines para reducir el ancho de la tarjeta y así reducir también su coste.
Podemos dividir las señales en 3, las de la parte izquierda son las señales de comunicación de datos, las del centro las de alimentación y las del flanco derecho son las señales del interfaz JTAG.
Como podemos ver este diseño cumple con los requisitos que anteriormente habíamos mencionado.

Ahora vamos a pasar a ver la capa superior, que hemos dicho que estaba compuesta por el plano de tierra y las señales de alimentación de 3.3V. En ambas imágenes vemos también el contorno de la tarjeta para que nos podamos hacer una idea de cómo están aprovechados los extremos.

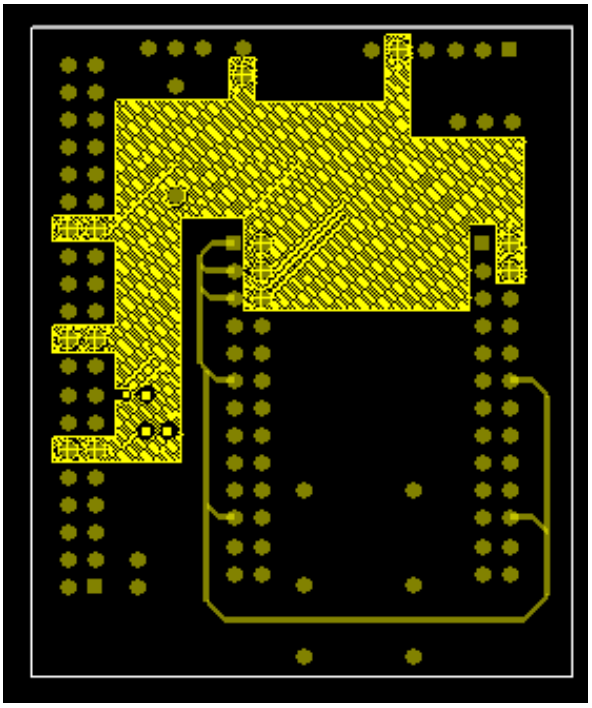

Figura 68: Routing capa superior

El plano de tierra debía pasar por aquellos pines que tienen que estar conectados a tierra. Se elige un plano como solución para la tierra debido a que es lo que menos impedancia tiene. El resto de las señales que vemos son gruesas debido a que son las de la alimentación de 3.3V.

Vamos a ver a continuación una imagen de las dos capas combinadas.

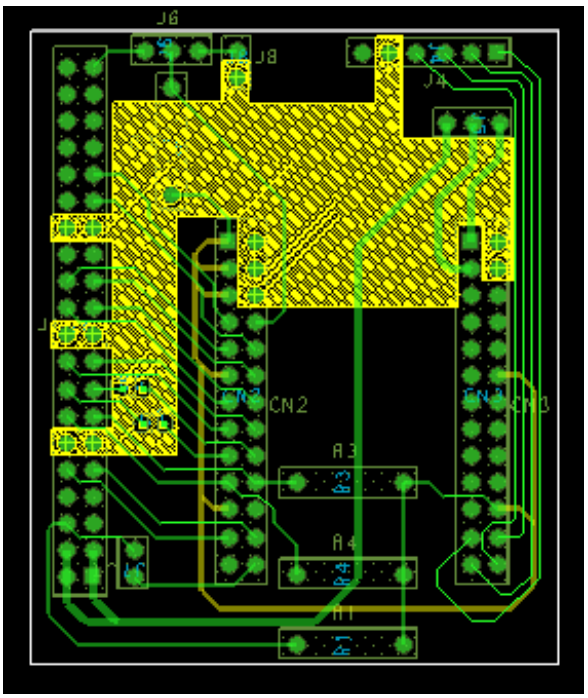

Figura 69: Routing general de la PCB

4.8 Fabricación

La tarjeta la hemos fabricado en el laboratorio de la escuela. Es por ello, que debido a la tecnología que existe actualmente hemos tenido que hacer algunas correcciones a nuestro diseño inicial, para adecuarnos al procedimiento que se usa para la fabricación.
Algunas de las correcciones han sido:

- El plano de tierra debe únicamente pasar por aquellos puntos que estén conectados a tierra. Inicialmente el plano que diseñamos ocupaba todo el espacio de la tarjeta a excepción del que ocupa la alimentación de 3.3V, pero para fabricarla hay complicaciones en los pines no conectados a tierra y que están dentro del plano.
- Las señales tienen cuellos en los estrechamientos de otros pines. Antes, todas las señales a excepción de las de alimentación eran del grosor de los estrechamientos, pero facilita las cosas hacer las señales más gruesas donde sea posible.
- El espacio entre el pad de los pines y el plano de tierra debía ser apreciable a simple vista en los planos de fabricación.
- En un principio la alimentación de 3.3V de la cara superior también iba a ser mediante un plano, pero los dos planos quedarían demasiado cerca, lo que complicaría enormemente la fabricación.

A continuación, vamos a ver los planos que hemos mandado a fabricación.

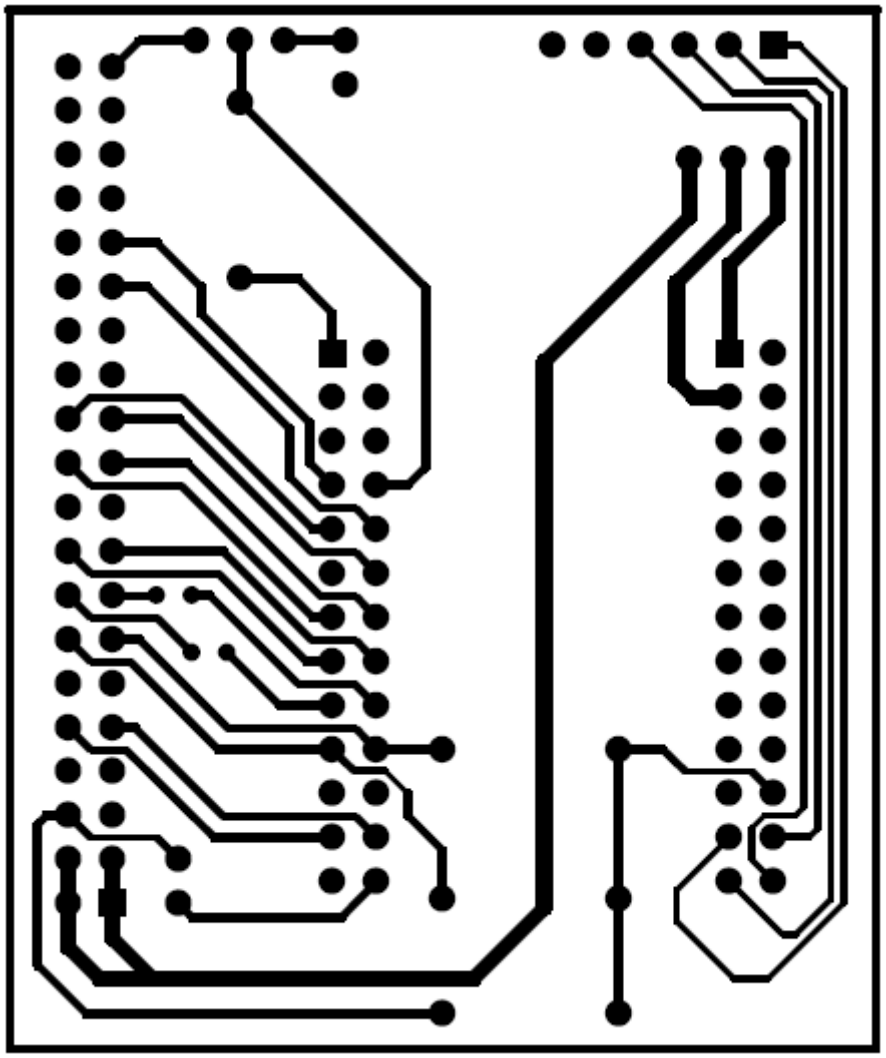

Figura 70: Plano fabricación capa inferior

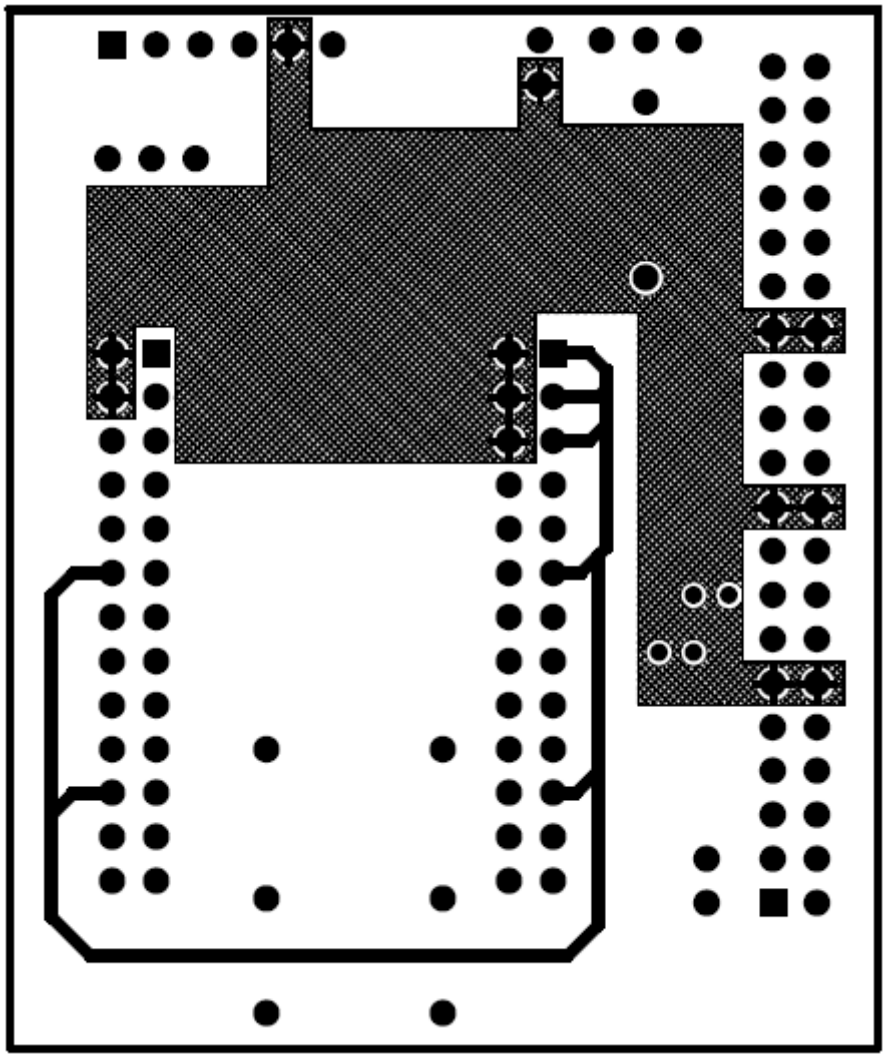

Figura 71: Plano fabricación capa superior

Como vemos, el plano de la cara superior es una imagen reflejada del plano real. Esto es para que en el proceso de fabricación ambas capas coincidan.

5. Interfaz de programación

5.1 Nociones básicas de JTAG

JTAG proviene de las siglas de Join Test Action Garoup, que es un grupo de la industria formado en 1885 cuya finalidad era resolver el problema de acceso físico a los pines de los circuitos integrados colocados sobre una PCB. A medida que las tarjetas de circuitos impresos crecen en complejidad y densidad de componentes, el proceso de verificación de estas se vuelve cada vez más complejo lo que deja algo atrasados los métodos que se usaban hasta entonces.

Es por ello que, el grupo JTAG comenzó a trabajar en una especificación para el diagnóstico de PCB´s mediante la exploración de contorno, denominada Boundary-Scan Testing. Esta norma se conoce abreviadamente como BST y ahora también como JTAG, por el nombre del grupo que la creó.

El gran valor añadido por el Boundary-Scan radica en su poder para controlar y observar mediante software los valores lógicos 0 y 1 en los pines de los circuitos impresos a través de un interfaz simple de cuatro bits.

En la actualidad el puerto JTAG se utiliza, además de como un medio de verificación de PCB´s, para la programación in-circuit de tarjetas de lógica programable. Este uso será en el que nosotros nos centraremos a lo largo de este trabajo.

5.2 IEEE 1149.1 – Test Access Port

La ilustración que podemos ver a continuación en forma de esquema es la estructura básica de la arquitectura BST. La cadena de exploración está formada por la concatenación de varias celdas, cada una ligada a un pin y/o al sistema lógico interno del chip. El dato puede ser desplazado por el contorno de exploración y capturado en los latches en los momentos indicados por el software.

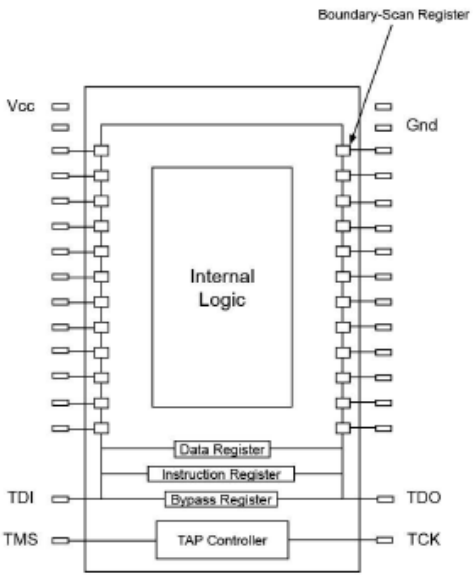

Figura 72: Test Access Port. Diagrama de bloques

Para la implementación de esta arquitectura necesitamos:

- Un interfaz serie formado por los cuatro pines habituales del interfaz JTAG, pudiendo añadirle un quinto en ciertas circunstancias.

    - TCK: Señal del reloj utilizada por el bus, esta señal de reloj deberá ser independiente del resto de los relojes del integrado y es a través de la cual se manejan los datos de entrada y salida.
    - TMS: Tes Mode Select, es una entrada del sistema cuyo valor, dependiendo de si es 0 o 1, nos permitirá movernos a través de la máquina de estados del TAP.
    - TDI: Test Data In, entrada de datos al sistema.
    - TDO: Test Data Out, salida de datos del sistema.
    - TCR: Test Command Reset, pin de entrada opcional que permite resetear la máquina de estados del TAP.

- La implementación de al menos tres registros como interfaz de control, el registro de datos, de instrucciones y de paso o también denominado bypass.
- Un controlador de acceso al puerto de pruebas, a partir de ahora, nos referiremos a él como TAP. El TAP debe controlar al menos tres instrucciones:

    - BYPASS, salto o paso de chip.
    - Data.
    - EXTEST, fijación de un pin a un valor dado, con prioridad sobre el valor lógico de la salida.

El fabricante del dispositivo con soporte para el JTAG es el responsable de la manera en la que decide implementar esta arquitectura y la inclusión o no de instrucciones específicos.

5.3 Características de Xilinx Boundary Scan

5.3.1 JTAG en Spartan-6 usando IEEE 1149.1

Todas las FPGA´s de Xilinx implementan un interfaz JTAG compuesto por los 4 pines obligatorios definidos en la norma. Como hemos visto en la parte del diseño de la placa y anteriormente, la Spartan-6 cuenta también con este interfaz. El JTAG está disponible siempre y cuando la FPGA esté alimentada. Veremos a continuación una imagen de la arquitectura del Boundary Scan para una FPGA del fabricante Xilinx.

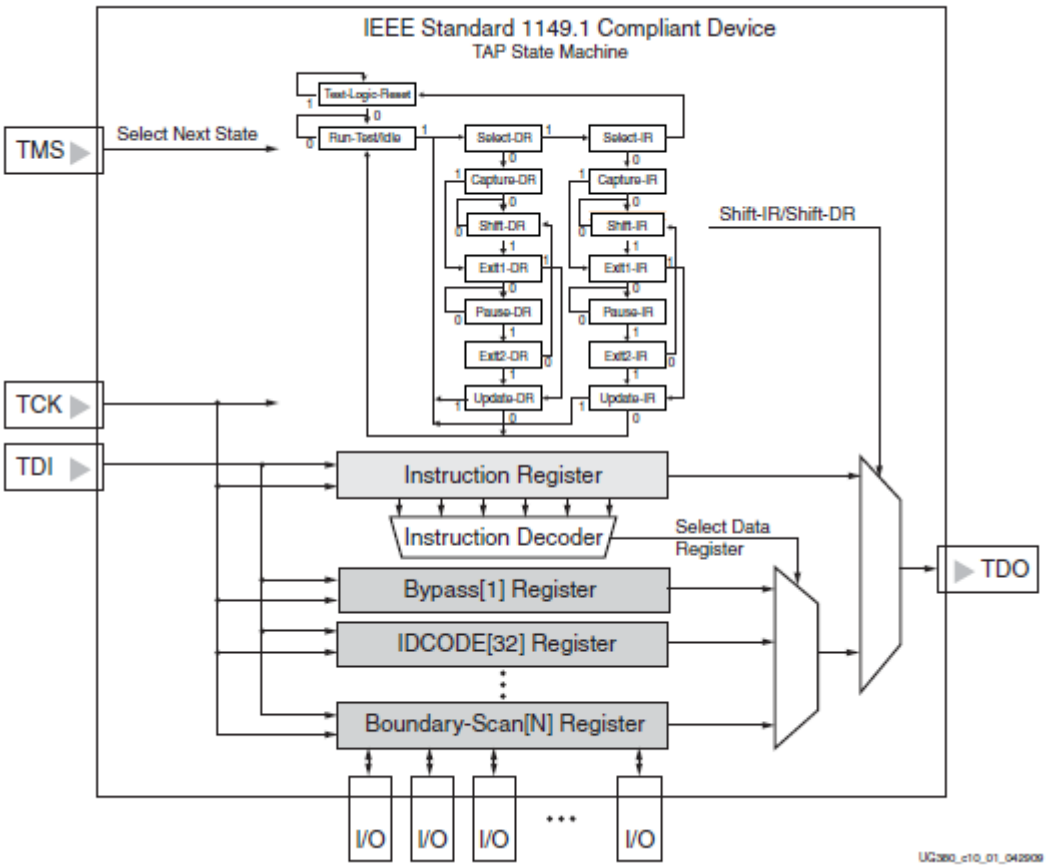

Figura 73: Arquitectura BST para fabricante Xilinx

5.3.1.1 Test Access Port (TAP)

Como hemos visto anteriormente, el TAP cuenta con los cuatro pines obligatorios que especifica el estándar del JTAG.

- TDI: Entrada serie tanto para el registro de datos como para el de instrucciones. El estado en el que se encuentre el controlador del TAP determinará el registro donde se cargarán los bits transmitidos a través de este pin. Este pin se encuentra conectado a un pull-up interno para que en caso de que no esté siendo utilizado se mantenga su valor a 1, estado de reposo ya que este pin es activo a nivel bajo. Los bits enviados mediante este pin serán registrados en el flanco de subida de la señal de reloj TCK.
- TDO: Salida serie tanto para el registro de instrucciones como para el de datos. Como ocurría en la señal TDI, dependerá del estado en el que se encuentre el controlador del TAP pudiendo ser un registro de datos o de instrucciones. El pin TDO cambia su estado con el flanco de bajada y está conectado a un pull-up interno.
- TMS: El valor lógico de este pin determina los movimientos a través de la máquina de estados del TAP. Esta señal será capturada en el flanco de subida de la señal TCK. Está conectado, al igual que las dos anteriores, a un pull-up

interno de manera que se encuentre manteniendo el valor 1 en caso de que no esté siendo manejada por ningún dispositivo.

- TCK: Es la entrada de reloj para el controlador del TAP y el circuito del BST.

5.3.1.2 Máquina de estados del TAP.

El controlador del TAP es una máquina síncrona de 16 estados. Nos movemos a través de ella dependiendo del valor lógico de la señal TMS. El reloj del controlador viene dado por la entrada TCK. Estando en cualquier estado se puede volver al estado inicial fijando TMS a nivel bajo y enviando 5 pulsos consecutivos de reloj por TCK. En la siguiente imagen podemos ver los valores que debe tomar la señal TMS para pasar de un estado a otro o permanecer en el mismo.

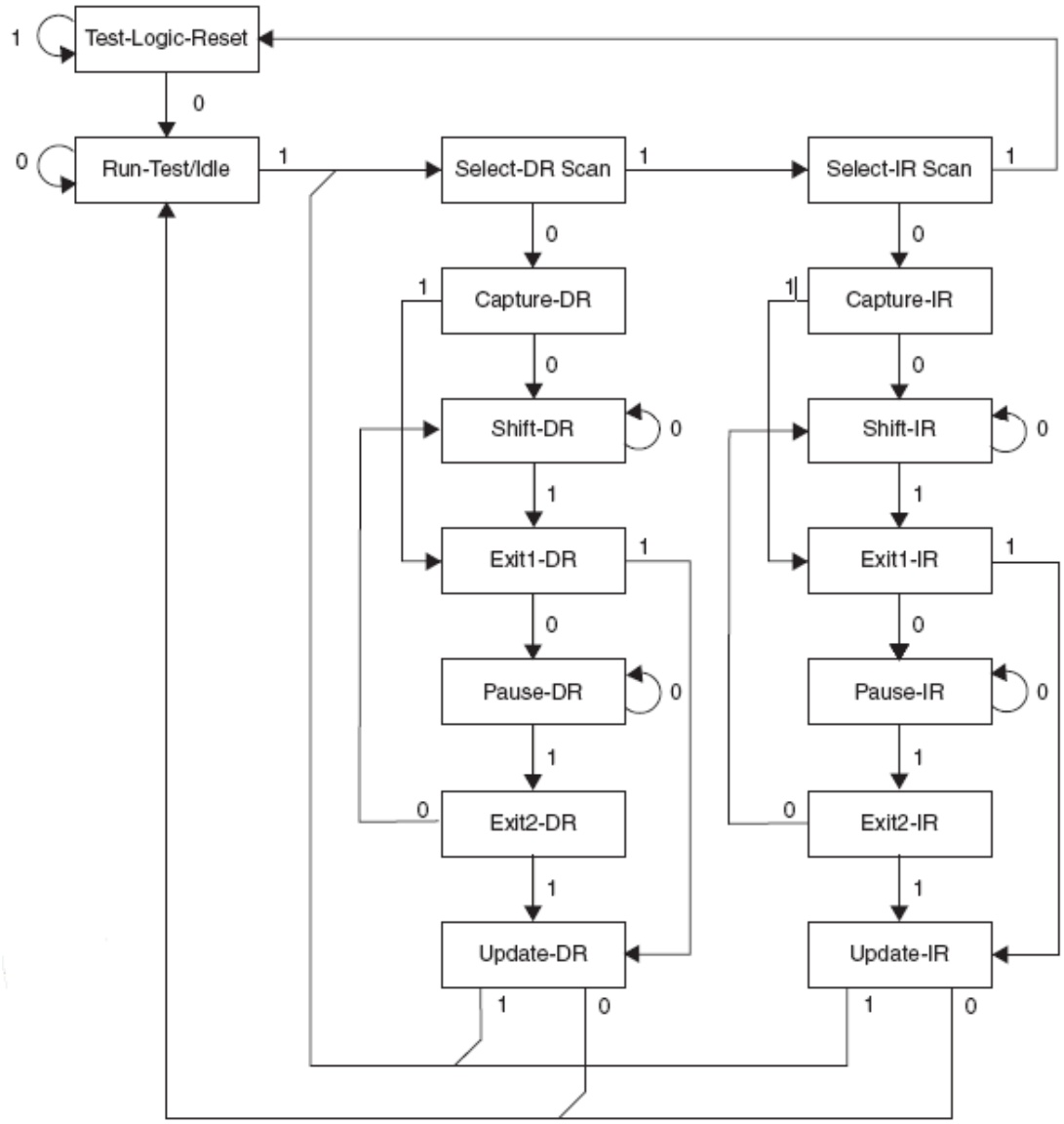

Figura 74: Máquina de estados del TAP

Como podemos observar en esta máquina existen dos secuencias de operaciones paralelas que se pueden realizar sobre un registro de datos o sobre el de instrucciones, DR o IR respectivamente. Podemos distinguir en cada una de esas secuencias tres operaciones: CAPTURE, SHIFT y UPDATE.

A continuación, vamos a explicar una a una las operaciones que nos encontramos en la máquina de estados del TAP.

- TEST-LOGIC-RESET: Cuando nos encontramos en este estado, el interfaz JTAG no está habilitado, por lo que la FPGA estará funcionando de forma ordinaria. Cada vez que encendamos el dispositivo este será el estado en el que se encuentre el controlador del TAP. Para volver a este estado inicial debemos mantener la señal TMS a nivel alto durante 5 ciclos de reloj.
- RUN-TEST-IDLE: Es el estado de reposo del TAP. También activa la lógica JTAG después de ejecutarse determinadas instrucciones.
- SELECT-DR-SCAN: Estado en el que tenemos acceso al registro de datos. Si al encontrarnos en esta operación mantenemos a nivel bajo el pin TMS se iniciará el acceso al registro de datos o bypass, dependiendo del valor que se haya cargado anteriormente en el registro de instrucciones.
- SELECT-IR-SCAN: Exactamente igual que el anterior, pero para el registro de instrucciones, únicamente cambia que si mantenemos la señal a nivel alto volveremos al estado de RESET.
- CAPTURE-IR: Conecta el registro de instrucciones al pin de entrada TDI y al pin de salida TDO.
- SHIFT-IR: Cada ciclo de reloj de reloj que permanezcamos en esta operación se cargara el bit transmitido a través de TDI en el bit mas significativo del registro de dirección. El resto de los bits se desplazarán una posición a la derecha. Todos los pines serán cargados o desplazados en el flanco de subida de la señal de reloj.
- EXIT1-IR: Estado de salida del modo desplazamiento. Durante la transición a este estado se producirá el último desplazamiento.
- UPDATE-IR: La instrucción previamente cargada en el registro de instrucciones será reconocida por el TAP y la instrucción tendrá efecto y será ejecutada hasta que volvamos al estado de reset o que otra instrucción la reemplace.

Las señales SHIFT-DR, EXIT1-DR, OAUSE-DR, EXIT2-DR, UPDATE-DR y CAPTURE-DR son exactamente igual que las de instrucciones, pero con datos.

Vamos a explicar el funcionamiento del controlador del TAP: El dispositivo después de reset arranca en el estado TEST-LOGIC-RESET, recomendamos antes de realizar ninguna operación forzar siempre este estado para partir de él de forma segura. A continuación, debemos navegar a través del controlador del TAP hasta que carguemos en el registro de instrucciones la que queremos ejecutar. Esta instrucción no va a ser reconocida hasta que no pasemos al estado UPDATE-IR.

Ejecutar una instrucción es conectar los puertos TDI y TDO a un registro de instrucciones. Para actuar sobre este registro navegaremos por el controlador del TAP para realizar las operaciones que deseemos.

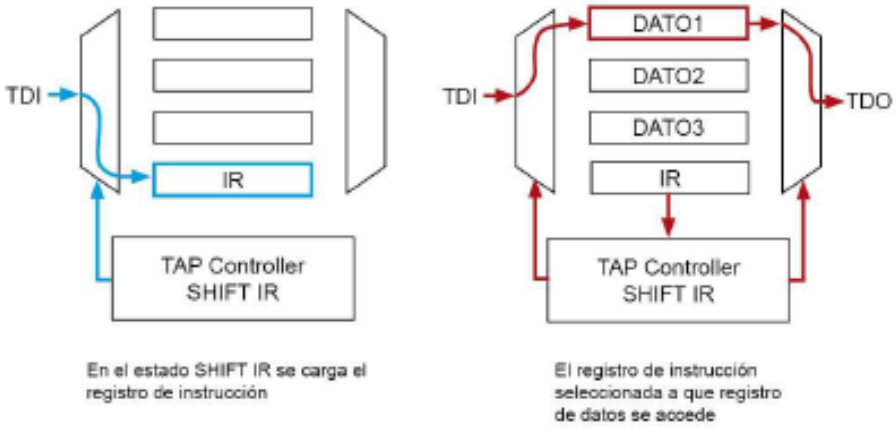

Figura 75: Lectura de un registro

5.3.2 Registro de instrucciones

Como acabamos de comentar en el funcionamiento de la máquina de estados del TAP, el registro de instrucciones es conectado entre los pines TDI y TDO de entrada y salida, mientras se está cargando la instrucción. Si queremos ejecutar o cargar una instrucción en concreto, lo que debemos hacer es transmitir bit a bit en cada señal del reloj TCK el código de instrucción que encontremos en el datasheet del fabricante. Debemos recordar que esta instrucción no se ejecutará hasta que no nos encontremos en el estado UPDATE-IR.
Vamos a describir a continuación las instrucciones que hemos utilizado en el código, en este caso son instrucciones del fabricante Xilinx debido a que estamos tratando de configurar la Spartan-6. Las instrucciones para cada FPGA las podemos encontrar en el datasheet del fabricante.

5.3.2.1 SAMPLE/PRELOAD

Esta instrucción tiene dos opciones, SAMPLE o PRELOAD. Mientras la ejecución de la instrucción se conectará el registro a las señales TDI y TDO.
Por un lado, SAMPLE carga una imagen del estado actual del sistema en el registro del BST. Las celdas del BST captura el valor de las entradas de todos sus pines y los valores de las salidas del dispositivo. Esto ocurrirá mientras estemos en el estado CAPTURE-DR y más tarde en el estado SHIFT-DR serán desplazados en cada señal de reloj bit a bit.
PRELOAD, sin embargo, coge el valor cargado dentro del BS registro desplazándolo a través de la salida paralelo.

5.3.2.2 EXTEST

Permite muestrear los valores de los pines de entrada, pero a diferencia de las anteriores, fuerza los valores de las salidas de manera simultánea. Durante la ejecución de esta instrucción el registro de BST está conectado entre TDI y TDO. Los valores de las entradas los capturaremos en el estado CAPTURE.DR, la instrucción se transmitirá bit a bit durante el estado SHIFT-DR del TAP para forzar las salidas cuando estemos en el estado UPDATE-DR.

5.3.2.3 BYPASS

Solemos utilizar esta instrucción para atravesar diferentes dispositivos de una misma cadena de manera rápida y sin que, si hacemos pruebas sobre un elemento de esa cadena, afecten al resto de dispositivos.
En esta instrucción se conecta la entrada TDI y TDO a un registro bypass.

5.3.2.4 Otras instrucciones para FPGA´s Xilinx.

Boundary-Scan Command	Instruction	Description
EXTEST	001111	Enables boundary-scan EXTEST operation.
SAMPLE	000001	Enables boundary-scan SAMPLE operation.
USER1	000010	Access user-defined register 1.
USER2	000011	Access user-defined register 2.
USER3	011010	User code that allows fabric access to/from the TAP controller from JTAG primitive instance 3.
USER4	011011	User code that allows fabric access to/from the TAP controller from JTAG primitive instance 4.
CFG_OUT	000100	Access the configuration bus for readback.
CFG_IN	000101	Access the configuration bus for configuration.
INTEST	000111	Enables boundary-scan INTEST operation.
USERCODE	001000	Enables shifting out user code.
IDCODE	001001	Enables shifting out of ID code.
HIGHZ	001010	3-state output pins while enabling BYPASS Register.
JPROGRAM	001011	Equivalent to and has the same effect as PROGRAM.
JSTART	001100	Clocks the startup sequence when Startup clock source is TCK (**StartupClk:JtagClk**).
JSHUTDOWN	001101	Clocks the shutdown sequence.
ISC_ENABLE	010000	Marks the beginning of ISC configuration. Full shutdown is executed.
ISC_PROGRAM	010001	Enables in-system programming.
ISC_NOOP	010100	No operation.
ISC_READ	010101	Used to read back battery-backed RAM.
ISC_DISABLE	010110	Completes ISC configuration. Startup sequence is executed.
ISC_DNA (ISC_FUSE_READ)	110000	Read Device DNA.

Tabla 15: Instrucciones FPGA´s Xilinx

Vamos a echarle un vistazo a alguna de las instrucciones más importantes de entre las anteriores.

- IDCODE: Es una instrucción implementada por el fabricante para poder identificar el dispositivo. Veremos cuando configuremos la FPGA que cuando ejecutamos el código nos indica que estamos ante una Spartan-6.
- USERCODE: Es una instrucción del fabricante para permitir el acceso al registro de usuario USER1.

- JPROGRAM: La ejecución de esta instrucción tendrá el efecto equivalente a la activación de la entrada PROGRAM del dispositivo.
- JSTART: Permite iniciar la secuencia de arranque cuando el reloj seleccionado es TCK.
- JSHUTDOWN: Activa la secuencia de apagado del dispositivo.

5.3.3 Registro de datos

Las FPGA´s del fabricante Xilinx tiene varios registros en la arquitectura del BST. Algunos de ellos son de propósito general y están contenidos en la norma IEEE 1149.1 y otros son opcionales para las operaciones de prueba y verificación.
La longitud de bits en los registros dependerá de la familia de FPGA que estemos tratando de configurar, en nuestro caso es una Spartan-6.
En la siguiente tabla podemos ver los registros para JTAG de la FPGA Spartan-6.

Register Name	Register Length	Description
Boundary-Scan Register	3 bits per I/O	Controls and observes input, output, and output enable
Instruction Register	6 bits	Holds current instruction opcode and captures internal device status
BYPASS Register	1 bit	Bypasses the device
Identification Register	32 bits	Captures the Device ID
JTAG Configuration Register	16 bits	Allows access to the configuration bus when using the CFG_IN or CFG_OUT instructions
USERCODE Register	32 bits	Captures the user-programmable code
User-Defined Registers (USER1, USER2, USER3, and USER4)	Design specific	Design specific

Tabla 16: Registros JTAG para Spartan-6

5.3.3.1 Boundary Scan Register

El registro BST es creado para controlar y probar el estado de los pines de entrada/salida. Todo pin de entrada/salida del circuito integrado, tiene tres celdas de registro BTS y no importa si no está conectado o no se encuentra disponible.

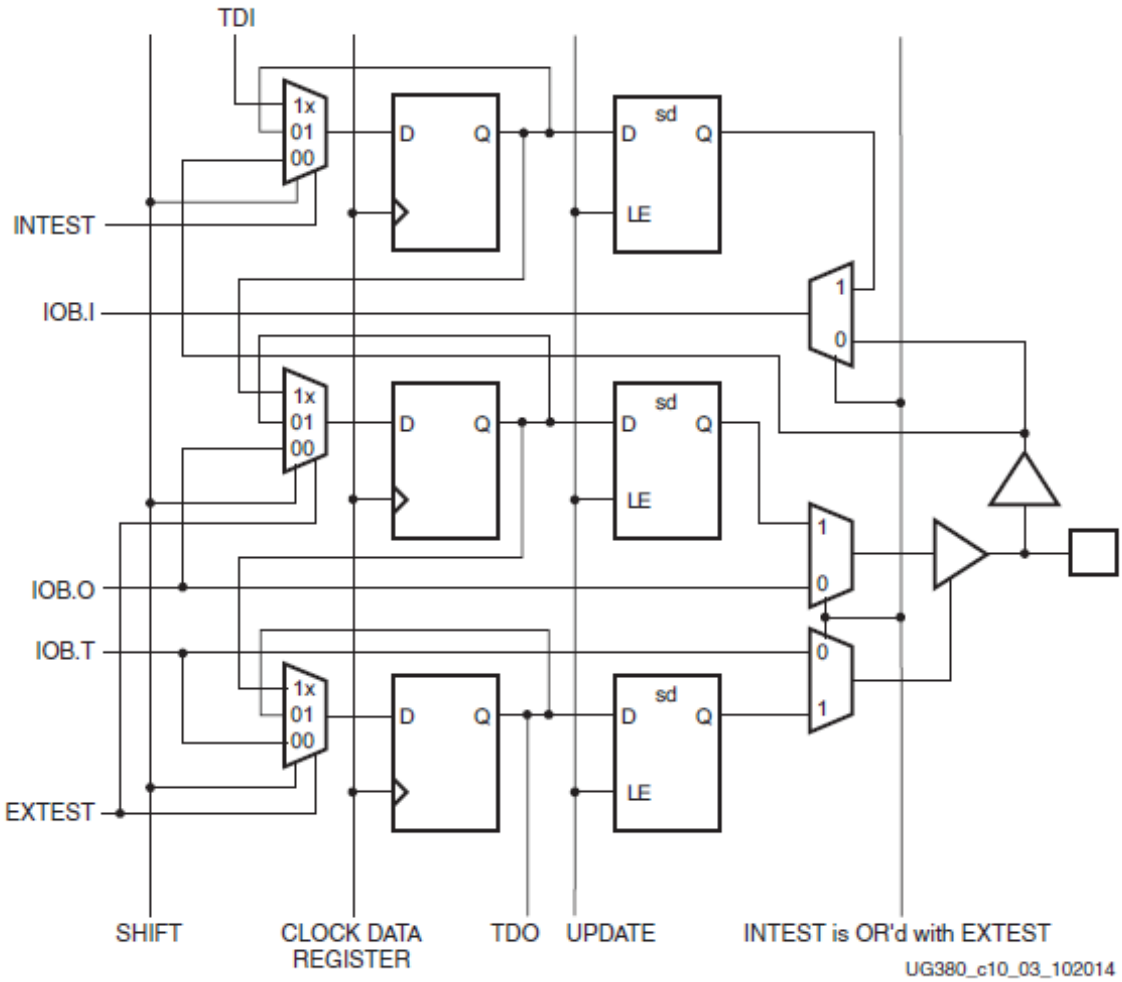

Figura 76: Registro Boundary Scan

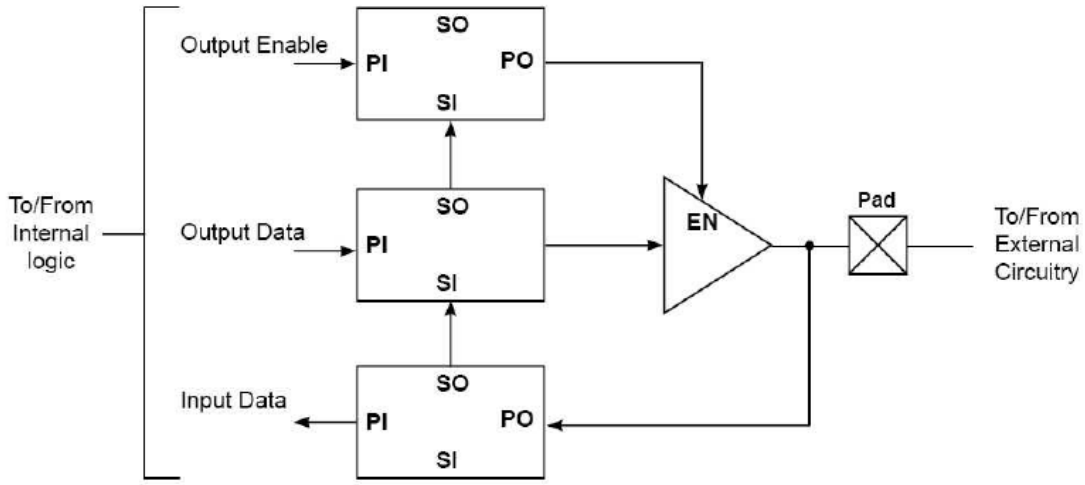

Figura 77: Celda del Boundary Scan

Cada una de las celdas tiene cuatro puertos, entrada serie SI, salida serie SO, entrada paralelo PI y salida paralelo PO. Los puertos SI y SO de cada una de las celdas estarán conectadas en serie a las demás del BTS.
Dicha cadena empieza en el pin TDI y recorrerá todos los pines del dispositivo hasta terminar en el pin de salida TDO, recorriendo los datos de test de la cadena en su desplazamiento.
Los puertos PI y PO de cada una de las celdas están conectados a la lógica interna de los buffers de entrada/salida. Cada operación de tipo Boundary Scan afecta a cada pin de entrada/salida de forma independiente.

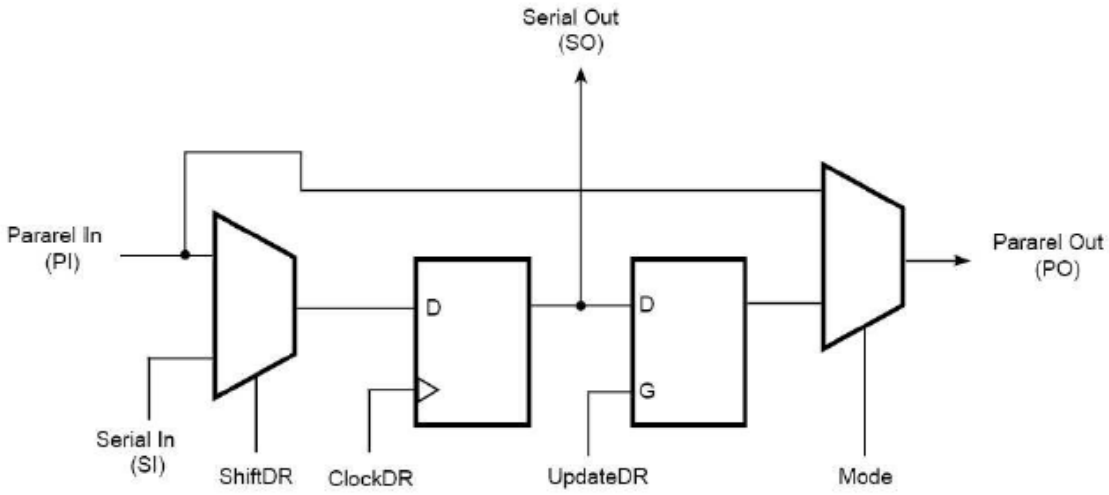

Figura 78: Puertos Boundary Scan

Todos los pines por defecto se encuentran configurados como bidireccionales con control triestado. Después de su configuración pasarán a ser entradas/salidas o triestado dependiendo del valor cargado. De todos modos, los tres bits del registro estarán disponibles para todos los pines.
En el caso de encontrarse seleccionado este registro y acceder a él mediante el controlador del TAP, se cargará el valor del registro. El orden de transmisión de los bits para este registro será el siguiente: primero el correspondiente a la entrada, seguido de la salida y finalmente el valor de control del buffer triestado.
La secuencia de bits de un dispositivo se puede obtener del fichero bsdl. Esta secuencia mantendrá siempre el mismo orden y el número de bits a transmitir será independiente del diseño.

5.4 Configuración de dispositivos Xilinx

5.4.1 Introducción

Las FPGA's del fabricante Xilinx deben ser configuradas mediante la descarga de un fichero de programación denominado bitstream, que se encuentra almacenado en la memoria interna. Las FPGA's pueden descargar el fichero de una memoria externa o utilizando un dispositivo intermedio, como es nuestro caso, con el controlador de

USB. Podemos diferenciar dos formas de configuración, bus serie o paralelo. El bus serie lo utilizamos para minimizar el número de pines que debemos usar en el proceso de configuración, mientras que el bus paralelo tiene un ancho de datos de 8 o 16 y aumenta la velocidad de configuración y se adapta mejor a los estándares.
Como también ocurre con los procesadores, las FPGA's de Xilinx podemos reprogramarlas las veces que sean necesarias.
El almacenamiento de estas FPGA's utiliza la tecnología CMOS, esto implica que la información se almacena de manera volátil y es eliminada en cada apagado. Por lo que cada vez que encendamos la FPGA debemos recargar el fichero de configuración que hemos nombrado anteriormente bitstream. Este proceso lo podemos ejecutar de varias maneras. El modo en que carguemos el bitstream dependerá del valor de los pines de configuración M [0:1] en el arranque según la siguiente tabla. Estos pines de configuración deben estar conectado a VCCO_2 del dispositivo o a tierra mediante pull up o pull down, cuyo valor encontramos en el manual de la Spartan-6.

Modo de configuración	M[1:0]	Ancho de Bus	Dirección CCLK
Master Serial/SPI	01	1, 2, 4	Output
Master SelectMAP/BPI	00	8, 16	Output
JTAG	xx	1	Input (TCK)
Sleve SelectMAP	10	8-16	Input
Slave Serial	11	1	Input

Tabla 17: Modos configuración. Pines M [1:0]

Los términos que vemos en la tabla anterior de maestro y esclavo hacen alusión a la dirección de configuración del reloj encargado de la operación. En este caso está configurada la FPGA como maestro y sería ella la encargada de generar el reloj CCLK. El valor de la frecuencia de configuración se determina como una opción en la herramienta que genera el bitstream.
El objetivo que perseguimos es utilizar el interfaz de configuración JTAG disponible en la FPGA Spartan-6 del fabricante Xilinx para configurarla teniendo en cuenta el dispositivo intermedio FT2232H. Es posible configurar un único dispositivo, que sería, o varios dispositivos en cadena. Cada uno de los dispositivos debe ser configurado de manera independiente, puenteando el resto.
Es necesario conocer el tamaño del registro de instrucciones de cada uno de los dispositivos que tuviéramos en la cadena para determinar la secuencia de datos. Podemos encontrar esta información en los archivos BDSL de cada uno de los dispositivos y en las diferentes datasheets.
A continuación, veremos un esquema de como funcionaría varios dispositivos en cadena con el interfaz de configuración JTAG. A esta configuración en cadena la denominamos Daisy chain.

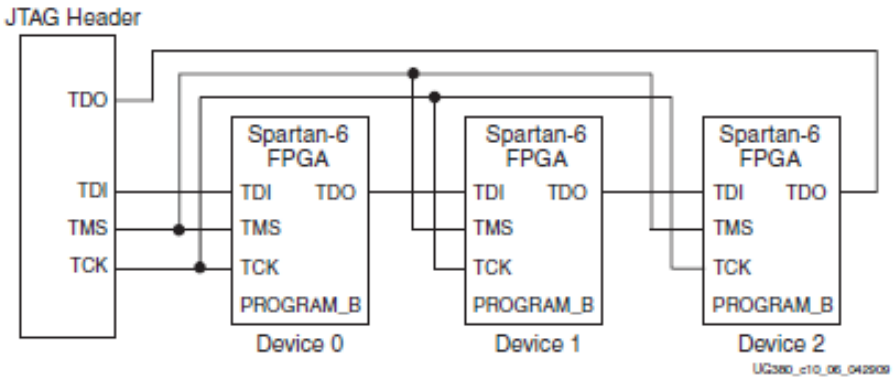

Figura 79: Esquema configuración en Daisy chain

5.4.2 Formatos para los ficheros de configuración

Podemos encontrar el fichero de configuración en diferentes formatos. Como veremos en la siguiente tabla, el programa BitGen, convierte el fichero NCD en un fichero bitstream. Más tarde, la herramienta PROMGen, convierte los ficheros bitstream en fichero de tipo PROM. Los ficheros de tipo PROM son los únicos que pueden presentar cambio de orden de los bits dentro de un byte, pero no será utilizado en este trabajo.

File Extension	Bit Swapping(1)	Xilinx Software Tool(2)	Description
BIT	Not Bit Swapped	BitGen (generated by default)	Binary configuration data file containing header information that does not need to be downloaded to the FPGA. Used to program devices from iMPACT software with a programming cable.
RBT	Not Bit Swapped	BitGen (generated if `-b` option is set)	ASCII equivalent of the BIT file containing a text header and ASCII 1s and 0s. (Eight bits per configuration bit.)
BIN	Not Bit Swapped	BitGen (generated if `-g Binary:yes` option is set) or PROMGen	Binary configuration data file with no header information. Similar to BIT file. Can be used for custom configuration solutions (for example, microprocessors), or in some cases to program third-party devices.
MCS EXO	Bit Swapped	PROMGen or iMPACT software	ASCII PROM file formats containing address and checksum information in addition to configuration data. Used mainly for device programmers and iMPACT software.
HEX	Determined by User	PROMGen or iMPACT software	ASCII PROM file format containing only configuration data. Used mainly in custom configuration solutions.
CFI	N/A	PROMGen or iMPACT software	Data file used by iMPACT software to determine PROM options to set such as x2 and x4 data width or version control.

Tabla 18: Tipos de ficheros de configuración

5.4.3 Secuencia de configuración de Spartan-6

Vamos a describir los estados por los que debe pasar la Spartan-6 para que sea configurada.

5.4.3.1 Set up

El proceso de inicialización es parecido en todos los interfaces de configuración. La Spartan-6 se encenderá desde el reset, debe iniciar su memoria de configuración interna y determinar el interfaz de configuración mediante el muestreo de sus pines.

5.4.3.1.1 Salida del reset

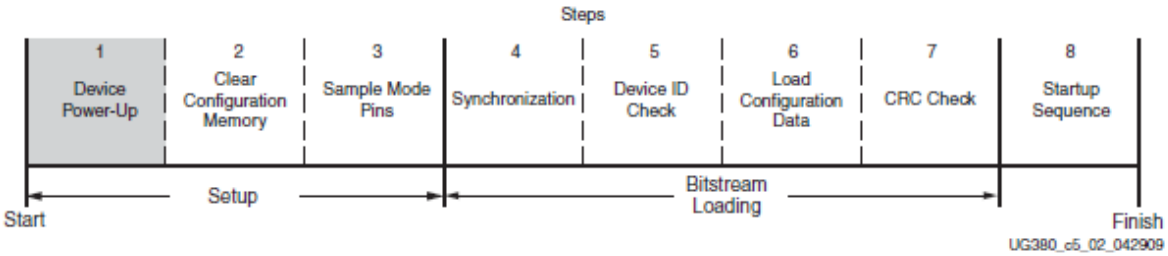

Figura 80: Paso 1 salida de reset

La Spartan-6 es inicializada y el modo de configuración es determinado a través de los pines de modo. El proceso de set up es parecido en todos los modos. Los pasos de set up son críticos para la correcta configuración del dispositivo.
Para configurar la Spartan-6, requiere alimentación, al menos, en los pines VCCO_2, VCCAUX y VCCINT más algún otro VCCO pin usado durante la configuración. Debemos conectar VCCO el último tras haber conectado los otros dos para asegurarnos de que las salidas están desactivadas hasta que el dispositivo se configure.
Podemos salir del reset de diferentes maneras:

- La Spartan-6 es alimentada y su circuito interno de Power-on-Reset mantiene la FPGA en reset hasta que las fuentes de alimentación alcanzan niveles apropiados.
- El pin PROG_B es forzado a nivel bajo para resetear la FPGA.
- La FPGA es llevada a reset mediante la instrucción JPROGRAM.

En la siguiente imagen podemos ver el timing seguido para el power-on-reset de la Spartan-6 a través de las señales:

- VCCINT, alimentación de la FPGA.
- VCCAUX, alimentación de los pines de configuración.
- VCCO, alimentación de los pines de entrada/salida.

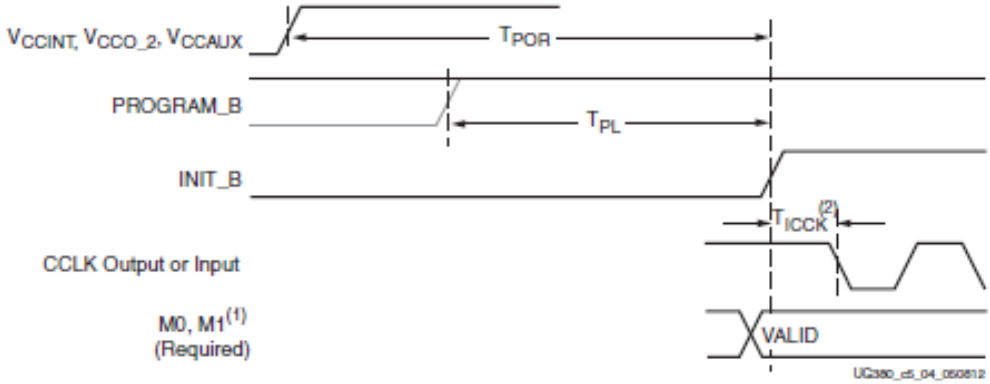

Figura 81: Timing del Power-on-Reset

5.4.3.1.2 Paso 2: Borrado de la memoria de configuración

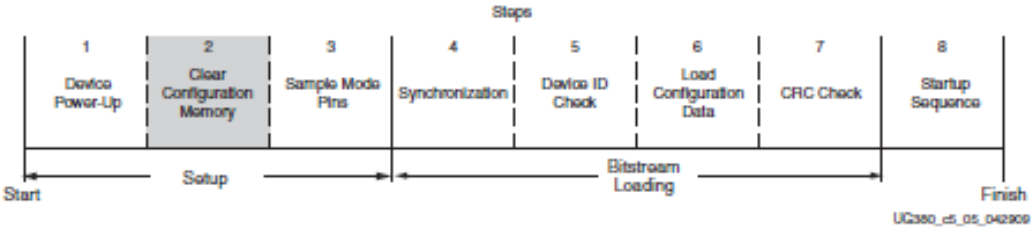

Figura 82: Paso 2: borrado de la memoria

La memoria de configuración es vaciada de manera secuencial cada vez que el dispositivo es encendido, después de que el pin PROGRAM_B sea forzado a nivel bajo y de que se ejecute la instrucción JPROGRAM. Durante este tiempo, las entradas/salidas están forzadas a nivel alto excepto los pines dedicados a la configuración.

El tiempo mínimo de la señal PROGRAM_B está definida por el tiempo de Tprogram. El pin PROGRAM-B puede mantenerse forzado a nivel alto el tiempo que sea necesario. El dispositivo borra la memoria de configuración dos veces tras que PROGRAM_B sea lanzado.

5.4.3.1.3 Paso 3: Muestreo de los pines de configuración.

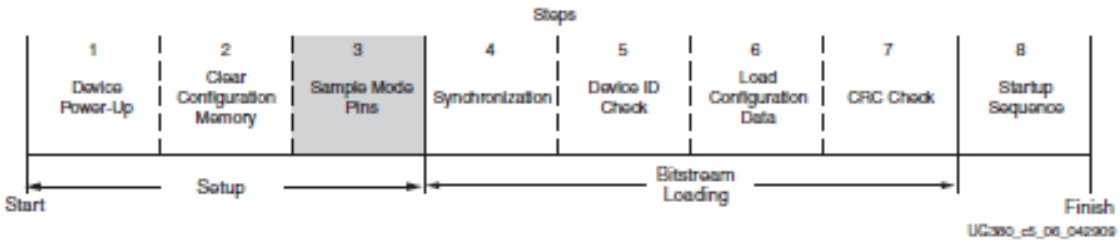

Figura 83: Paso 3: muestreo pines configuración

Cuando el pin INIT_B deje de estar forzado a nivel alto, la Spartan-6 muestreará los pines M [2.:0] y comenzará a generar la señal de reloj.

5.4.3.2 Carga del bitstream

La carga del bitstream es común a todos los modos de configuración. Los pasos dentro de esta etapa son:

5.4.3.2.1 Sincronización

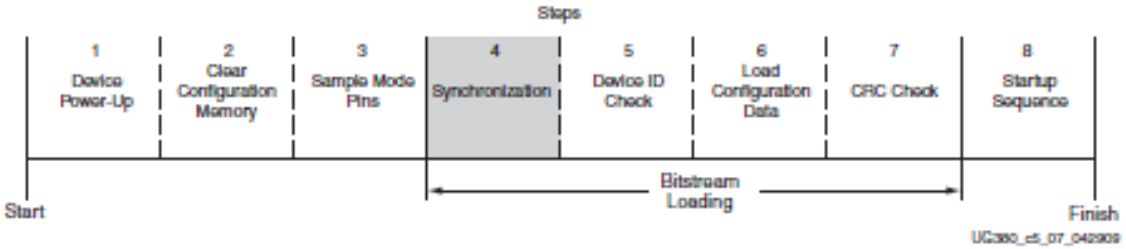

Figura 84: Paso 4: Sincronización

La palabra de sincronización alerta al dispositivo de la llegada de los datos de configuración y alinea los datos de configuración con la lógica interna de configuración. Cualquier dato enviado antes de la palabra de sincronización será ignorado. La longitud y el contenido de la palabra de configuración puede variar dependiendo del de la familia de la FPGA utilizada.

5.4.3.2.2 Comprobación del array IDCODE

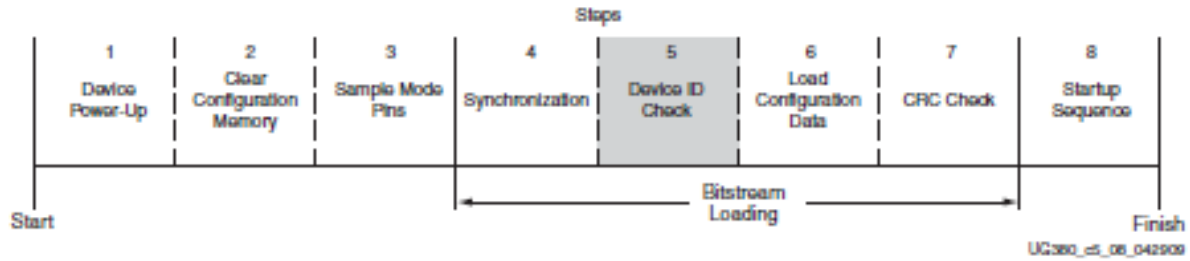

Figura 85: Paso 5: Identificación del dispositivo

Una vez el dispositivo ha sido sincronizado, el identificador del dispositivo debe pasar antes de que los datos de configuración sean cargados. Esto evita que sea cargado un bitstream de otro dispositivo distinto al nuestro. El chequeo de la identidad del dispositivo está dentro del código del bitstream, haciendo este proceso transparente a la mayoría de los diseñadores. La comprobación de la identidad del dispositivo es a

través de comandos desde el bitstream hasta la lógica de configuración no a través del registro JTAG IDCODE en este caso.

El registro JTAG IDCODE de la Spartan-6 tiene el siguiente formato:
vvvv:fffffff:aaaaaaaaa:ccccccccccc1 donde:
v = revisión
f = código familiar de 7 bits
a = código con array de 9 bits
c = código conjunto de 11 bits

Device	ID Code (Hex)
6SLX4	0x04000093
6SLX9	0x04001093
6SLX16	0x04002093
6SLX25	0x04004093
6SLX25T	0x04024093
6SLX45	0x04008093
6SLX45T	0x04028093
6SLX75	0x0400E093
6SLX75T	0x0402E093
6SLX100	0x04011093
6SLX100T	0x04031093
6SLX150	0x0401D093
6SLX150T	0x0403D093

Tabla 19: Código de identificación

La tabla anterior es una aclaración sobre el código de identificación en función del modelo de Spartan-6 que tengamos. En nuestro caso la Spartan-6 que estamos probando es la 6SLX75.

5.4.3.2.3 Paso 6: Carga de los frames de datos

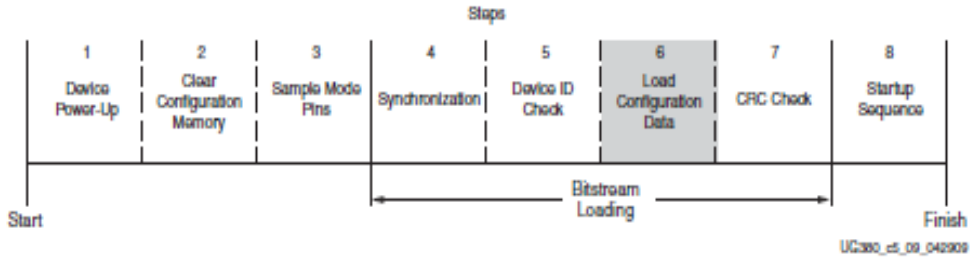

Figura 86: Paso 6: Carga de los frames de datos

Una vez la palabra de sincronización ha sido cargada y se ha comprobado la identidad del dispositivo, los frames de datos serán cargados. Este proceso es transparente para la mayoría de los usuarios.

5.4.3.2.4 Confirmación de redundancia cíclica

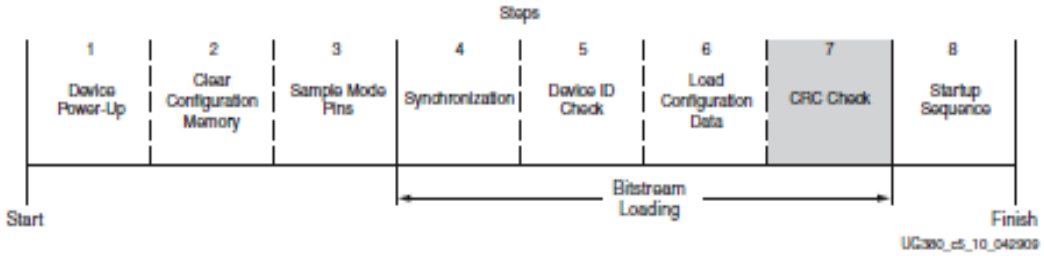

Figura 87: paso 7: Confirmación de redundancia cíclica

Mientras se cargan los frames de datos en la Spartan-6, esta irá calculando el código de redundancia cíclica de los datos recibidos. Una vez que todos los frames de datos hayan sido cargados y antes de la detección de la palabra de desincronización, el fichero de configuración puede contener una instrucción de comprobación del código CRC seguido por el código CRC esperado.

5.4.3.3 Paso 8: Start-up

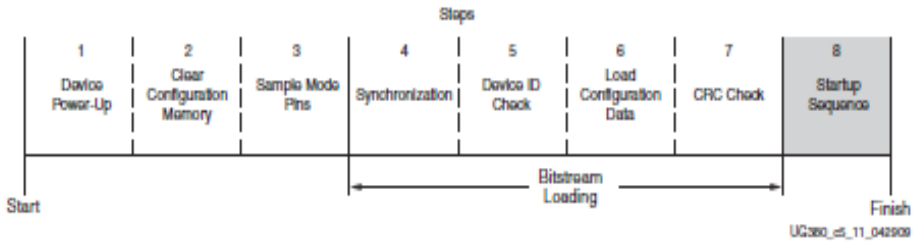

Figura 88: paso 8: Start-up

Una vez que el fichero de configuración haya sido carga debemos ordenar a la Spartan-6 que comience la secuencia de Start-up.
La secuencia se realizará de manera síncrona con la fuente de reloj que se haya configurado anteriormente en la creación del bitstream configurando la herramienta BitGen como hemos descrito con anterioridad, pudiendo ser:

- CCLK, configurado por defecto
- Reloj generado por la lógica interna de la FPGA mediante la primitiva START-UP.
- En el caso de que la configuración se realice a través de JTAG, se deberá utilizar el reloj proporcionado por este interfaz, que es la señal TCK.

En nuestro caso, deberemos usar la señal de reloj TCK puesto que usaremos el interfaz de configuración JTAG.

5.5 Configuracion de la Spartan-6 a través de Boundary Scan

La configuración de la Spartan-6 a través de BST se consigue siguiente el procedimiento descrito en la siguiente ilustración.

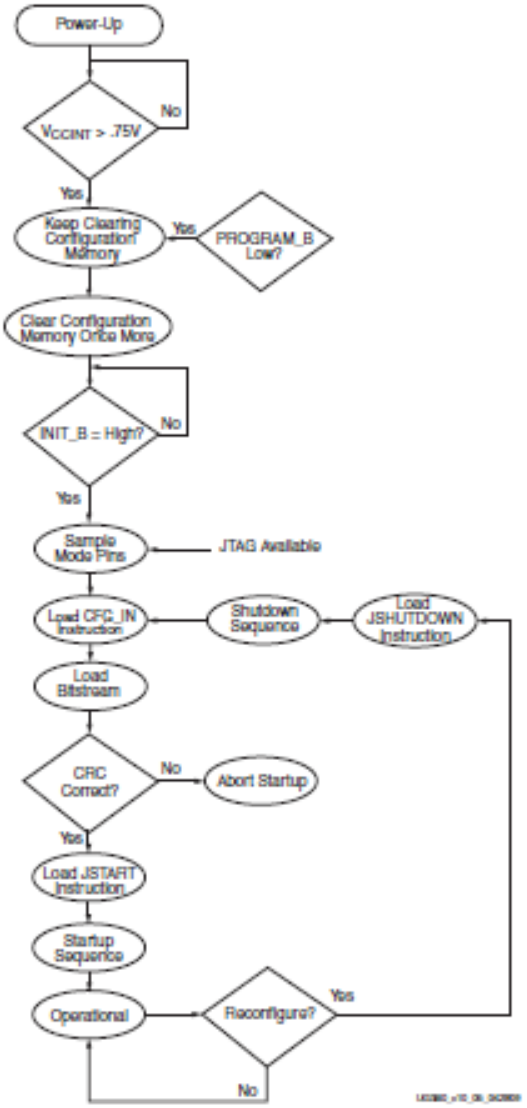

Figura 89: Proceso configuración

La siguiente tabla describe los comandos del controlador del TAP que se requieren para configurar la Spartan-6.

Extensión de archivo		Enviado y retenido		1Número de relojes
		TDI	TMS	TCK
1	En el encendido, coloque un "1" en el TMS y el reloj TCK cinco veces. (Esto garantiza el arranque en el estado TLR (Test-Logic-Reset))	x	1	5
2	Mueva al estado RTI	x	0	1
3	Mueva al estado SELECT-IR.	x	1	2
4	Introduzca el estado SHIFT-IR.	x	0	2
5	Comenzar a cargar la instrucción CFG_IN	0101	0	4
6	Cargue el último bit de la instrucción CFG_IN al salir de SHIFT-IR (definido en la norma IEEE).	0	1	1
7	Introduzca el estado SELECT-DR.	x	1	2
8	Introduzca el estado SIFT-DR.	x	0	2
9	Desplazamiento en el flujo de bits (bitN (MSB) es el primer bit en el flujo de bits	bit_1 bit_N	0	(Número de bits en bitstream)1
10	Cambia en el último bit del flujo de bits. (bit0 (LSB) se desplaza en la transición a EXIT1-DR)	bit_0	1	1
11	Introduzca el estado UPDATE-DR.	x	1	1
12	Introduzca el estado SELECT-IR.	x	1	2
13	Mueva al estado SHIFT-IR.	x	0	2
14	Comience a cargar la instrucción JSTART. (La instrucción JSTART inicializa la secuencia de inicio.)	1100	0	4
15	Cargue el último bit de la instrucción JSTART.	0	1	1
16	Mueva al estado SELECT-DR.	x	1	2
17	Desplácese a SHIFT-DR y marque la secuencia STARTUP. (aplicando un mínimo de 12 ciclos de reloj al TCK)	x	0	≥14
18	Mueva al estado UPDATE-DR.	x	1	2
19	Vuelva al estado RTI. (El dispositivo ya está funcionando).	x	0	1

Tabla 20: Comandos controlador del TAP Spartan-6

5.6 Implementación de un bus JTAG.

En el interfaz JTAG, tenemos las señales de entrada y salida TDI y TDO y la señal de reloj TCK, pero la que nos interesa en este momento es la señal de control, que es la que nos permite movernos a través de los estados en el controlador del TAP. Las comunicaciones que realizamos a través del interfaz JTAG son LSB first, lo que quiere decir que los datos son desplazados dentro de la máquina de estados en el flanco de subida, lo que implica que el MPSEE tiene que sacar esos datos en el flanco anterior.

5.6.1 Flujo de trabajo

Para configurar el dispositivo y establecer las comunicaciones a través del MPSEE debemos seguir los siguientes pasos:

1. Confirmar la disponibilidad del dispositivo y abrir el manejador, para lo cual llamaremos a las siguientes funciones:
   FT_CreateDeviceInfoList, es una función nos informa del número de dispositivos ft que encuentra en estado disponible.

FT_GetDeviceInfoList, nos devuelve la información de cada uno de los dispositivos que ha identificado previamente.

FT_Open, nos permite abrir el dispositivo a través del manejador que hemos usado en la función previa.

2. Configurar el puerto para ser usado en modo MPSSE. Una vez hayamos abierto el puerto debemos configurar algunos parámetros para poder usar las comunicaciones:

- Resetear los periféricos llamando a la función FT_ResetDevice.
- Dimensionar el tamaño de los buffers de recepción y transmisión del USB. FT_SetUSBParameters.
- Configurar la generación de eventos y definir los caracteres erróneos, para ellos utilizaremos la función FT_SetChars.
- Configurar los timeouts de lectura y escritura.
- Configurar el tiempo que debemos esperar para enviar un paquete incompleto de vuelta al periférico del host. Llamada a la función FT_SetLatencyTimer.

3. Configurar el MPSEE. En estos instantes, el controlador está listo para aceptar comandos. La función FT_Write la usamos para enviar instrucciones y parámetros al controlador. Las respuestas son leídas mediante FT_Read.
4. Sincronización y detección de comandos erróneos. Cuando un comando erróneo es detectado, el controlador devuelve un identificador seguido del identificado de comando erróneo recibido.
5. Configuración del controlador MPSSE. Una vez establecida la comunicación debemos configurar la velocidad del reloj, las direcciones de los pines y los estados iniciales.
6. Comunicación serie. Cuando hayamos configurado los parámetros podemos establecer las comunicaciones que tendremos con el dispositivo periférico.
7. Por último, debemos cerrar el manejador cuando la aplicación haya completado las comunicaciones. Esto lo hacemos mediante la función FT_Open. Es recomendable resetear el controlador antes del cierre.

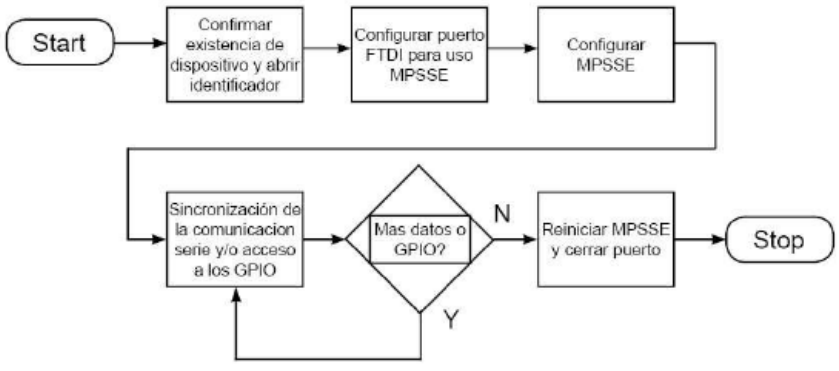

Figura 90: Diagrama flujo trabajo MPSEE

5.7 Solución implementada

Tras haber fabricado la anterior tarjeta que permite configuración y comunicación con la Spartan-6, el segundo objetivo que perseguimos es conseguir configurar la FPGA. Para este trabajo contamos con el código de una Spartan-3 por lo que debíamos cambiar el código para conseguir configurar una Spartan-6.
En esta ocasión no tuvimos que desarrollar la lógica programable dentro de la FPGA ya que pudimos utilizar la lógica de la que dispone la propia FPGA.

5.7.1 Cronología del diseño

Para afrontar el problema de la configuración de la Spartan-6 hemos ido dividiéndolo en problemas más pequeños:

- El primer paso será configurar el dispositivo FT2232H. No podemos hacer ninguna comunicación con el JTAG antes de haber configurado el dispositivo intermedio a través del MPSSE.
- El segundo paso será programar las funciones que harán que nos movamos a través del controlador del TAP. Debemos hacer una función para cada una de las operaciones entre estados, para poder acceder a los registros de instrucciones.
- Más tarde, debemos evaluar si en nuestra comunicación tenemos un diseño en cadena con varios dispositivos o si por el contrario es únicamente un dispositivo.
- Debemos acceder a registros básicos dentro del TAP del primer dispositivo de la cadena JTAG.
- Acceder de igual manera que en el apartado anterior, pero al resto de dispositivos de la cadena.
- Programación de la Spartan-6, cargaremos el archivo bitstream y modificaremos las funciones necesarias para cargar el fichero de programación generado previamente.

5.7.1.1 Configuración del dispositivo FT2232H

Antes de tratar con la configuración de la Spartan-6 debemos configurar el controlador del USB, ya que es él el que nos permite usar el interfaz JTAG. Este dispositivo debe ser configurado a través del modo MPSSE, que nos permite movernos a través de la máquina de estados.
Las comunicaciones en el modo MPSSE se realizan a través de las API's proporcionadas por FTDI, en este trabajo las hemos agrupado en dos librerías.

5.7.1.1.1 Librería ftdi_basics.cpp

Esta librería nos proporciona las funciones que necesitamos para confirmar la existencia y abrir el manejador del dispositivo. Vamos a ver las principales funciones que encontraremos en esta librería y lo que realiza cada una.

Int check_ftdi (DWORD] dwNumDevs, FT_HANDLE * ftHandle, int devToOpen, bool verbose, FILE * report)

- Obtener el número de dispositivos FTDI
- Abra el puerto

int config_ftdi_port (FT_HANDLE ftHandle, int bufferSize, int latency, int rx_timeout, int tx_timeout, bool verbose, FILE * report)

- Configurar los parámetros del puerto
- Restablecer el dispositivo FT2232H. Purgar el buffer de recepción USB
- Configurar los tamaños de transferencia de petición USB
- Deshabilitar caracteres de evento y error
- Establezca los tiempos de espera de lectura y escritura en milisegundos
- Activar el control de flujo para sincronizar solicitudes IN
- Controlador de reinicio
- Habilitar el modo MPSSE

5.7.1.1.2 Librería ftdi_jtag.cpp

Agruparemos en esta librería las funciones necesarias para la utilización del dispositivo intermedio en modo JTAG. La función que vamos a ver a continuación podemos configurar el modo MPSSE con los parámetros adecuados para implementar el JTAG.

Int config_ftdi_jtag (FT_HANDLE ftHandle, int dwClockDivisor, bool verbose, informe FILE *)

- Configurar el modo de comunicación MPSSE para JTAG. Enviamos los comandos con la función FT_Write.
- Ajustar la frecuencia de TCK a 60MHz
- Configurar el reloj del MPSSE
- Desactivar el reloj adaptativo
- Deshabilitar el sincronismo trifásico
- Inhabilitar el loopback interno
- Restablecer la máquina del TAP

5.7.1.2 Operaciones dentro del controlador del TAP.

Anteriormente en este capítulo hemos explicado con detenimiento cómo movernos a través de la máquina de estados del TAP. Ahora debemos programar una función para cada movimiento de un estado a otro de la máquina. Las funciones implementadas son las siguientes:

```
int config_ftdi_jtag (FT_HANDLE ftHandle, bool verbose) { ... }
void Test_Logic_Reset (FT_HANDLE ftHandle, bool verbose) { ... }
void TLT_to_RTI (FT_HANDLE ftHandle, bool verbose) { ... }
void RTI_clk_cycles (FT_HANDLE ftHandle, int num_clk_cycles, bool verbose) { ... }
void RTI_to_SDR (FT_HANDLE ftHandle, bool verbose) { ... }
void RTI_to_SIR (FT_HANDLE ftHandle, bool verbose) { ... }
void EDR_to_RTI (FT_HANDLE ftHandle, bool verbose) { ... }
void EIR_to_RTI (FT_HANDLE ftHandle, bool verbose) { ... }
```

Figura 91: Conjunto de funciones controlador TAP

Las funciones anteriores son semejantes, todas tienen la misma estructura. En cada flanco de subida deberemos forzar el pin de salida del TMS al valor lógico que necesitemos para pasar de un estado a otro. Para hacernos una idea de cómo trabaja alguna de estas funciones vamos a ver un ejemplo.

```
void RTI_to_SDR (FT_HANDLE ftHandle, bool verbose)
{

FT_STATUS ftStatus;
DWORD dwNumBytesToRead;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesRead;//Auxiliar variable used to store the number of read bytes
DWORD dwNumBytesToSend;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesSent;//Auxiliar variable used to store the number of read bytes
BYTE byOutputBuffer[4];     // Buffer to hold MPSSE commands and data to be sent to the FT2232H
BYTE byInputBuffer[4];      // Buffer to hold data read from the FT2232H
    dwNumBytesToSend = 0;// Reset output buffer pointer
//Navigage through  TAP Controler  until Test-Logic-Reset.
// Navigage TMS through Run-Test-Idle   ->      Select-DR-SCAN  -->     Capture-DR  -->     Shift-DR
//                                      TMS=1                   TMS=0               TMS= 0
    byOutputBuffer[dwNumBytesToSend++] = 0x4B;// Clock out TMS, no read.
    byOutputBuffer[dwNumBytesToSend++] = 0x02;// Number of clock pulses = Length + 1 (3 clocks here)
    byOutputBuffer[dwNumBytesToSend++] = 0x01;// Data is shifted LSB first, so the TMS pattern is 10
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
}
```

```
void RTI_clk_cycles (FT_HANDLE ftHandle, int num_clk_cycles, bool verbose)
{

FT_STATUS ftStatus;
DWORD dwNumBytesToRead;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesRead;//Auxiliar variable used to store the number of read bytes
DWORD dwNumBytesToSend;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesSent;//Auxiliar variable used to store the number of read bytes
BYTE byOutputBuffer[1024];      // Buffer to hold MPSSE commands and data to be sent to the FT2232H
BYTE byInputBuffer[1024];       // Buffer to hold data read from the FT2232H
    dwNumBytesToSend = 0;// Reset output buffer pointer
//Navigage through  TAP Controler  until Test-Logic-Reset.
// Navigage TMS through Run-Test-Idle   ->      Select-DR-SCAN  -->     Capture-DR  -->     Shift-DR
//                                      TMS=1                   TMS=0               TMS= 0
    int n_cycles;
    int i;
    n_cycles = num_clk_cycles/8;
    for( i=0; i< n_cycles; i++)
    {
    byOutputBuffer[dwNumBytesToSend++] = 0x4B;// Clock out TMS, no read.
    byOutputBuffer[dwNumBytesToSend++] = 0x07;// Number of clock pulses = Length + 1 (3 clocks here)
    byOutputBuffer[dwNumBytesToSend++] = 0x00;// Data is shifted LSB first, so the TMS pattern is 10
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
}
}
```

5.7.1.3 Identificación Daisy Chain

Durante esta etapa vamos a identificar cuantos dispositivos forman la cadena de configuración. El modo Daisy Chain quiere decir que los dispositivos están conectados en serie, esto se hace para únicamente establecer un puerto JTAG.
El sistema funciona de la siguiente manera, si queremos configurar cualquiera de los equipos de la cadena, debemos poner los demás en modo bypass. Los pines TDI y TDO son conectado a un registro de instrucciones de ancho de 1 bit, lo que introduce un ciclo de retraso por cada dispositivo que este conectado a la cadena.
Cuando la cadena JTAG tiene más de un dispositivo no podemos movernos a través del controlador del TAP, sino que debemos hacerlo de manera independiente para cada uno.

Para saber el número de elementos que componen la cadena hemos utilizado la siguiente función:

void discover_chain (FT_HANDLE ftHandle, unsigned int* number_jtagdev, bool verbose, FILE* report)

- Purgar los tampones TDI y TDO
- Reiniciar la máquina de estados del TAP. Los dispositivos inicializan el registro de instrucciones con la instrucción BYPASS
- Desplazar por la máquina del TAP de Test_Logic_Reset a Shift_Data Reg
- Los dispositivos en la cadena JTAG en modo derivación, si algún pin esta enviando el pin TDT, se devolverá el pin de TTGTTG, un número de ciclos de TCK después.

```
dwNumBytesToSend = 0;           //

// Here is the TMS command for one clock.  Data is also shifted in.

byOutputBuffer[dwNumBytesToSend++] = 0x4B;
                                // Clock out TMS, Read one bit.
byOutputBuffer[dwNumBytesToSend++] = 0x00;
                                // Number of clock pulses = Length + 0 (1 clock here)
byOutputBuffer[dwNumBytesToSend++] = 0x01;
                                // Data is shifted LSB first, so TMS becomes 1.  Also, bit 7 is shifted into TDI/DO, also a 1
                                // The 1 in bit 1 will leave TMS high for the next commands.
ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);

                                // Send off the TMS command
dwNumBytesToSend = 0;           // Reset output buffer pointer

 EDR_to_RTI (ftHandle,verbose);


    do
    {
        ftStatus = FT_GetQueueStatus(ftHandle, &dwNumBytesToRead);
                                    // Get the number of bytes in the device input buffer
    } while ((dwNumBytesToRead == 0) && (ftStatus == FT_OK));
                                    //or Timeout
    printf("LEIDO NUMERO is 0x%x\n",dwNumBytesToRead);
    ftStatus = FT_Read(ftHandle, &byInputBuffer, dwNumBytesToRead, &dwNumBytesRead);
                                    //Read out the data from input buffer

    printf("IDCODE: 0x");
            for(i=0 ; i<dwNumBytesRead; i++)
                printf("%02X", *(byInputBuffer+(dwNumBytesRead-1)-i));
            printf("\n");
//BUSCAR LA SECUENCIA PARA DESPLAZAR


ftStatus = FT_Purge(ftHandle, FT_PURGE_RX | FT_PURGE_TX); // Purge both Rx and Tx buffers
if (ftStatus == FT_OK) {
// FT_Purge OK
}
else {
// FT_Purge failed
```

Lo que hace la función es limpiar el buffer RX, leer los datos del buffer de entrada y enviar ocho para determinar cuántos dispositivos se asignan a la cadena JTAG.

5.7.1.4 Acceso a registros básicos dentro del TAP del primer dispositivo de la cadena JTAG

A continuación, vamos a ver las funciones a través de las que podemos acceder a los registros de datos y la de registro de instrucciones.

Para los registros de datos utilizaremos la función:

```
void read_idcode (FT_HANDLE ftHandle, unsigned int jcommand,unsigned int size_ir, bool verbose)
{
    FT_STATUS ftStatus;
DWORD dwNumBytesToRead;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesRead;//Auxiliar variable used to store the number of read bytes
DWORD dwNumBytesToSend;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesSent;//Auxiliar variable used to store the number of read bytes
BYTE byOutputBuffer[1024];      // Buffer to hold MPSSE commands and data to be sent to the FT2232H
BYTE byInputBuffer[1024];       // Buffer to hold data read from the FT2232H

    wjtag_command (ftHandle, jcommand,size_ir,verbose);
     RTI_to_SDR (ftHandle,verbose);
        // TMS is currently low.  State machine is in Shift-DR, so now use the TDI/TDO command to shift 101 out TDI/DO while reading TDO/DI
    // Although 3 bits need shifted in, only 2 are clocked here.  The 3rd will be in conjunciton with a TMS command, coming next
        dwNumBytesToSend = 0;
    byOutputBuffer[dwNumBytesToSend++] = 0x28;
                                // Clock data out throuth states Shift-DR and Exit-DR.
    byOutputBuffer[dwNumBytesToSend++] = 0x03;
                                // Number of clock pulses = Length + 1 (2 clocks here)
    byOutputBuffer[dwNumBytesToSend++] = 0x00;
                                // Shift out 101 (ignore last bit)
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
                                // Send off the TMS command
    dwNumBytesToSend = 0;           // Reset output buffer pointer

    // Here is the TMS command for one clock.  Data is also shifted in.

    byOutputBuffer[dwNumBytesToSend++] = 0x4B;
                                // Clock out TMS, Read one bit.
    byOutputBuffer[dwNumBytesToSend++] = 0x00;
                                // Number of clock pulses = Length + 0 (1 clock here)
    byOutputBuffer[dwNumBytesToSend++] = 0x01;
                                // Data is shifted LSB first, so TMS becomes 1.  Also, bit 7 is shifted into TDI/DO, also a 1
                                // The 1 in bit 1 will leave TMS high for the next commands.
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);

                                // Send off the TMS command
    dwNumBytesToSend = 0;           // Reset output buffer pointer


    // Navigage TMS through Update-DR -> Select-DR-Scan -> Select-IR-Scan -> Test Logic Reset
     //                     TMS=1        TMS=1             TMS=1             TMS=1

    byOutputBuffer[dwNumBytesToSend++] = 0x4B;
                                // Don't read data in Update-DR -> Select-DR-Scan -> Select-IR-Scan -> Test Logic Reset
    byOutputBuffer[dwNumBytesToSend++] = 0x01;
                                // Number of clock pulses = Length + 1 (4 clocks here)
    byOutputBuffer[dwNumBytesToSend++] = 0x01;
                                // Data is shifted LSB first, so the TMS pattern is 101100
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
                                // Send off the TMS command
    dwNumBytesToSend = 0;           // Reset output buffer pointer

    do
    {
        ftStatus = FT_GetQueueStatus(ftHandle, &dwNumBytesToRead);
                                // Get the number of bytes in the device input buffer
    } while ((dwNumBytesToRead < 4) && (ftStatus == FT_OK));
                                //or Timeout
    printf("LEIDO NUMERO is 0x%x\n",dwNumBytesToRead);
    ftStatus = FT_Read(ftHandle, &byInputBuffer, dwNumBytesToRead, &dwNumBytesRead);
                                //Read out the data from input buffer

    printf("status_register: 0x");
    unsigned int i = 0;
            for(i=0 ; i<dwNumBytesRead; i++)
                printf("%02X", *(byInputBuffer+(dwNumBytesRead-1)-i));
            printf("\n");
```

Para entender lo que realiza la función vamos a hacer un pequeño resumen.
Lo primero será escribir comando con la función wjtag_command y navegamos hasta el registro de datos. La función limpia los primeros bits que provienen de otros dispositivos asignados a la cadena. Más tarde limpia el buffer de recepción y lee los datos de entrada para seguir con el código de identificación. Se navega hasta actualizar el estado de los registros de datos. Si es una solicitud de ID, Print ID CODE,

se realiza una comprobación, ya que la función puede ser reutilizada para leer otros registros.

Una vez visto el acceso a registro de datos, vamos con el de instrucciones.

```
void wjtag_command (FT_HANDLE ftHandle, unsigned int jcommand,unsigned int size_ir, bool verbose)
{
FT_STATUS ftStatus;
DWORD dwNumBytesToRead;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesRead;//Auxiliar variable used to store the number of read bytes
DWORD dwNumBytesToSend;//Auxiliar variable used to store the number of bytes to read
DWORD dwNumBytesSent;//Auxiliar variable used to store the number of read bytes
BYTE byOutputBuffer[4];     // Buffer to hold MPSSE commands and data to be sent to the FT2232H
BYTE byInputBuffer[4];      // Buffer to hold data read from the FT2232H

    //TLR_TO_RTI
//  TLT_to_RTI (ftHandle, verbose);
    RTI_to_SIR (ftHandle, verbose);
    //write id code to spartan-3
    dwNumBytesToSend = 0;

    byOutputBuffer[dwNumBytesToSend++] = 0x1B;
    byOutputBuffer[dwNumBytesToSend++] = (size_ir -1) & 0xFF;// Number of clock pulses = Length + 1 (5 clocks here)
    byOutputBuffer[dwNumBytesToSend++] = jcommand & 0xFF; //001001 // Shift out 1111111 (ignore last bit)
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
    int i;
    for(i=0; i< 3 ; i++)
    {
    printf(" command %02X", *(byOutputBuffer+i));
    }
    printf("\n");
    //SIR_TO_EIR
    dwNumBytesToSend = 0;           // Reset output buffer pointer
    byOutputBuffer[dwNumBytesToSend++] = 0x4B;

    byOutputBuffer[dwNumBytesToSend++] = 0x00; // Number of clock pulses = Length + 0 (1 clock here)
    byOutputBuffer[dwNumBytesToSend++] = ((jcommand>>(size_ir -1)& 0x01)?(0x80):(0x00))|(0x01);// Data is shifted LSB first, so TMS becomes 1.
    ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend, &dwNumBytesSent);
    for(i=0; i< 3 ; i++)
    {
    printf(" command %02X", *(byOutputBuffer+i));
    }
    printf("\n");
    dwNumBytesToSend = 0;           // Reset output buffer pointer
    // Navigage TMS from Exit-IR through Update-IR -> Select-DR-Scan -> Capture-DR
    //                                  TMS=1       TMS=1            TMS=0
    EIR_to_RTI (ftHandle,verbose);

    }
```

El estado en el que se encuentra al iniciarse la función es SHIFT-IR, al igual que antes, utilizaremos el comando de lectura/escritura de MPSEE 0x39. El bit más significativo se escribirá al mismo tiempo que la transición TMS que mueve la máquina de estados al registro de instrucción de salida.

5.8 Programación de la Spartan-6. Fichero bitstream.

Por último, debemos usar dos funciones para cargar el fichero bitstream. Podemos encontrar las dos funciones en la librería bitstream.cpp

unsigned int read_bit_header(char bitfile[100], unsigned char **bitstream, unsigned int*bitfile_size, bool verbose)

Con ella lo que hacemos es abrir el fichero en modo lectura binaria, recorrerlo buscando los parámetros de la cabecera del fichero .bit, almacenamos el bitstream en un buffer intermedio e imprimimos la cabecera por consola o fichero dependiendo de los parámetros de la función.

int jtag:wdata (FT_HANDLE ftHandle, unsigned char *wdata, unsigned int wdatalen, bool verbose, FILE*report)

Divide el fichero en bloques de datos de 65536 bytes coincidiendo con el número máximo de bytes a transmitir con un comando MPSSE.
Procedemos luego a la escritura de N bloques del bitstream mediante el comando MPSSE 0x11. Se escribe el último byte del bloque comando 0zx11 y el último bit del comando 0x4B.

6. Conclusiones

En un primero momento se realizaron pruebas ejecutando el fichero bitstream con la Spartan 6 y el controlador del USB, detectando el programa dos dispositivos. En ese momento aún no se había fabricado la tarjeta sobre la que trata este trabajo, por lo tanto, lo hicimos con un prototipo de manera cableada. El resultado fue satisfactorio ya que éramos capaces de configurar la FPGA.
Una vez ya fabricada la tarjeta, hemos vuelto a realizar las pruebas conectando la tarjeta completa al ordenador y ejecutando el fichero bitstream y siendo capaces de nuevo de configurarla correctamente.

A la hora de valorar los objetivos que pusimos a principio del trabajo, lo hacemos de manera positiva ya que se ha conseguido el diseño y fabricación de la tarjeta de forma satisfactoria y posteriormente hemos comprobado, tanto el correcto funcionamiento de la tarjeta como del código para la Spartan-6.

En cuanto a posibles líneas futuras de este trabajo hay un proyecto claro, que es el de conseguir programar la comunicación de datos a través del controlador de USB. Con ese objetivo cumplido, se acabaría por completo el funcionamiento que se espera de la tarjeta que aquí hemos fabricado.

Una vez se haya conseguido la comunicación de datos, otra posibilidad para el desarrollo de la actual tarjeta sería la fabricación de una tarjeta de expansión, con varias memorias SRAM y la posibilidad de incluir la comunicación FIFO síncrona, con otro tipo de conectores ya que con la actual tarjeta no hay señales suficientes como para implementar este tipo de comunicación.

7. Bibliografía

Configuration Guide Spartan3 , Xilinx, Inc, 2016 (www.xilinx.com/support/documentation/user_guides/ug332.pdf)

Configuration Guide Spartan6 , Xilinx, Inc, 2017 www.xilinx.com/support/documentation/user_guides/ug380.pdf

Implementación de un interfaz doble de transmisión programación para módulos comerciales de FPGA basado en un conversor USB multiprotocolo genérico, María Dolores García Álvarez, 2017.

Planos Spartan6 TE0630 (www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0630_series/TE0630/documents/AD-TE0630-02I.PDF)

RS Components (es.rs-online.com/web/p/resistencias-fijas-de-orificio-pasante/0150928/)

RS Components (es.rs-online.com/web/p/resistencias-fijas-de-orificio-pasante/0150928/)

Schematics Spartan6 TE0630 (www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0630_series/TE0630/documents/SCH-TE0630-02.PDF)

Schematics Spartan6 TE0303 (www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0300_series/TE0303/documents/SCH-TE0303-01.pdf)

Trenz Electronics Connector (https://shop.trenz-electronic.de/de/21288-Zwei-Praezisions-Sockelstreifen-gerade-einreihig-2-54-mm-20-polig)

User Manual Guide resistencia 10k (docs-emea.rs-online.com/webdocs/09bb/0900766b809bb010.pdf)

User Manual Guide resistencia 150 ohmios (https://docs-emea.rs-online.com/webdocs/157b/0900766b8157b17e.pdf)

User Manual Guide Spartan6 TE0630, Xilinx, Inc, 2016 (www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0630_series/TE0630/documents/UM-TE0630.pdf)

User Manual Guide TE0303, Xilinx, Inc, 2017 (www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0300_series/TE0303/documents/UM-TE0303-01.pdf)