Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Sistema de adquisición y procesamiento de audio.

Autor:

Pérez Segurado, Rubén

Tutor:

Andrés Rodríguez-Trelles, Francisco José de

Departamento de Tecnología Electrónica

Valladolid, Julio de 2015.

Agradecimientos.

Lo primero es agradecer a mi familia por haberme proporcionado las mejores lecciones de la vida, haberme inculcado que con esfuerzo y constancia se pueda conseguir todo.

Agradecer también a Francisco y a Oliver, que sin su ayuda y sin su paciencia no habría sido posible realizar este proyecto.

Por último, dar las gracias también a todos aquellos que durante estos años han estado a mi lado, en las buenas y en las malas situaciones, porque sin ellos no hubiera podido proseguir en mi sueño.

Resumen:

El objetivo de este proyecto es el diseño y la implementación de una plataforma para un sistema de procesamiento de audio.
El sistema recibirá una señal de audio analógica desde una fuente de audio, permitirá realizar un tratamiento digital de dicha señal y generará una señal procesada que se enviará a unos altavoces externos.

Para la realización del sistema de procesamiento se empleará:

- Un dispositivo FPGA de Lattice, modelo MachX02-7000-HE, en la cual estarán todas las configuraciones programadas para dicho procesamiento.
- Un códec de audio, en el que su entrada será la línea de entrada de audio analógica y su salida la señal analógica enviada a los altavoces.
- La lógica de control y el establecimiento de la interfaz entre los distintos dispositivos para realizar los diferentes objetivos.

Palabras clave:

Adquisición, procesamiento, audio, FPGA, control.

Índice:

Capítulo 1. Introducción y objetivos.

En este apartado se procederá a realizar una introducción, de manera clara y concisa, del presente trabajo de fin de grado.

1.1 Descripción del proyecto.

Se pretende programar un dispositivo capaz de realizar la adquisición de audio a través de un códec, convirtiendo la señal de entrada analógica a señal digital, o bien directamente desde una fuente analógica de audio a través de un conector JACK 3.5 mm.

Esta señal digital se envía a un dispositivo FPGA, el cual dependiendo de su configuración realizará un tipo de procesamiento digital u otro, pudiendo programar con anterioridad cualquier tipo de efecto sonoro que queramos aplicar a la señal de audio obtenida en la fuente analógica.

Por último, la señal procesada se envía a un sistema de escucha sonora, en este caso unos altavoces convencionales.

1.2 Objetivos

El objetivo principal que se pretende alcanzar es el desarrollo y puesta a punto de una plataforma que permita el tratamiento digital de audio, basado en una FPGA. Para alcanzar este objetivo, se deberán cumplir otros objetivos parciales como:

- Analizar los diferentes tipos de efectos sonoros, que se estudiarán dentro del campo de procesamiento de audio, y más tarde se implementarán sus algoritmos en el dispositivo FPGA.

- Analizar los sistemas de adquisición de audio y sus convertidores A/D y D/A para poder trabajar con sus diferentes elementos, realizando estas conversiones de manera óptima para un buen comportamiento del sistema. También se deberá tener muy en cuenta todos los requisitos y características de dichos convertidores y acondicionadores de la señal de audio.

- Analizar la estructura de un dispositivo FPGA, inicialización, características y posibilidades del mismo. Además, se deberá programar e introducir las configuraciones con el software de diseño electrónico.

- Emplear herramientas de diseño electrónico, simulación y síntesis de circuitos electrónicos modernos, analizando las diferentes posibilidades de diseño y evaluar sus posibilidades.

- Emplear el lenguaje VHDL y sus herramientas de simulación y síntesis para programar en Lattice Diamond, haciendo estructuras mixtas formadas por HDL y esquemas.

- Analizar los resultados y extraer conclusiones del trabajo realizado. Por último, se deberá verificar que el comportamiento del sistema es correcto y estable.

1.3 Hardware y Software.

Para la realización de este proyecto, utilizaremos un hardware en el cual algunos módulos han sido creados con anterioridad, y donde otros módulos serán necesarios crearlos para realizar algunas de las tareas específicas que competen a este trabajo. Por ello, se utilizarán los correspondientes programas de creación y diseño electrónico para crear algunas PCB (placa de circuito impreso).

Como dispositivo para realizar dicho control, se utilizará un dispositivo FPGA de Lattice, modelo MachX02-7000-HE, que estará montado sobre la PCB principal. La elección de este dispositivo se debe a su disponibilidad y la gran memoria interna que dispone para realizar el procesamiento.

El software utilizado para programar este dispositivo es el proporcionado por la propia empresa Lattice Semiconductor, llamado Lattice Diamond 3.2. Este software nos permite programar el dispositivo FPGA y realizar las simulaciones necesarias para comprobar su correcto funcionamiento.

También se dispondrá de un códec de audio Wolfson, montado por la empresa Mikroelektronika, el cual tendrá una entrada de señal analógica por la que se introducirá la señal de audio, conversores A/D y D/A, filtros digitales y analógicos, acondicionadores para hacer el procesamiento digital, y por último dispondrá de una salida de audio analógica.

Capítulo 2. Procesamiento de audio.

2.1 Digitalización de audio:

Lo primero que debemos analizar en el procesamiento digital es la conversión de la entrada de audio desde nivel analógico a un nivel digital, para que pueda darse dicho procesamiento digital.

Las señales u ondas sonoras se pueden transformar a señales eléctricas analógicas a través de algunos dispositivos como el micrófono, que convierte estas vibraciones en una señal de voltaje dependiente del tiempo.

Existen aplicaciones donde la señal analógica es la adecuada, pero en los ordenadores o máquinas digitales, sus procesos se basan en señales discretas, y por ello la señal de audio analógica se debe transformar a una señal de audio digital binaria.

Los pasos para su digitalización son:

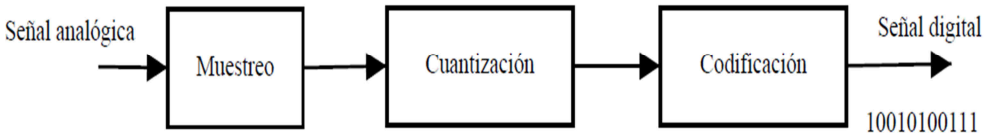


Figura 1: Esquema de digitalización de audio.

- **Muestreo:** Mide la amplitud de la señal cada ciertos intervalos de tiempo predefinidos y de igual duración. Cada valor de la señal de ese intervalo se llama muestra, y el periodo con el que se toman las muestras es el periodo de muestreo, que se mide en segundos. Su inversa es la frecuencia de muestreo o "sampling rate", que se mide en hercios. Según el teorema de Nyquist, la frecuencia de muestreo debe ser el doble de la frecuencia máxima de la señal de entrada para que no se produzca aliasing y se pierda información de la señal muestreada.

- **Cuantización:** Un vez muestreada la señal, procedemos a limitar los valores de la amplitud de dichas muestras estableciendo una serie de números o valores de la señal. El número de valores que puede tomar esta muestra corresponde a la resolución que posea el convertidor A/D, que se mide por el número de bits del mismo. Por ejemplo si tuviera 8 bits, tendría una resolución de 2^8=256 valores o estados diferentes que podría tomar la señal.

- **Codificación:** este proceso consiste en asignar un valor o código binario al conjunto de la cuantización de la muestra. Esta codificación se puede realizar con un códec, que incluye los parámetros necesarios para el proceso de codificación, como número de canales, frecuencia de muestreo, resolución y bits rate (velocidad de transmisión de los bit en el circuito).

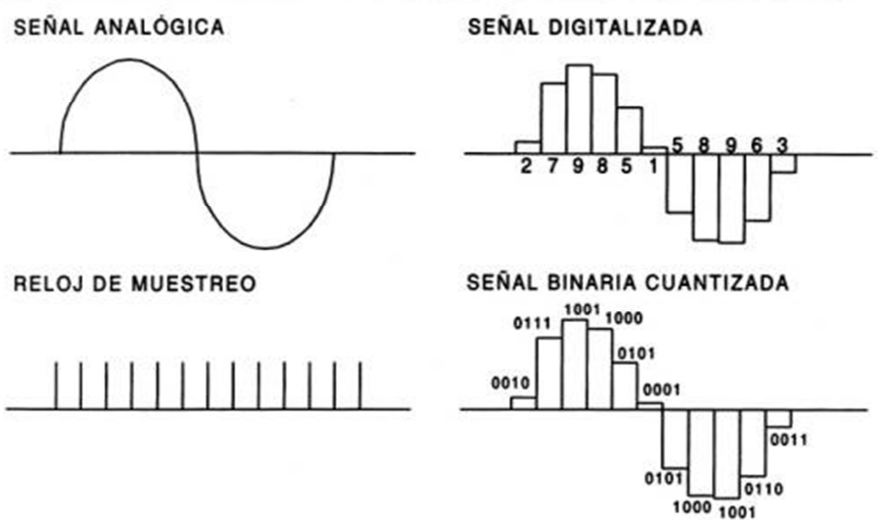


Figura 2: Procesos de digitalización de audio.

Las ventajas del procesamiento digital frente al procesamiento analógico son las siguientes:

- Flexibilidad: facilidad de modificar el algoritmo que se le aplica al circuito de control.
- Repetitividad: Este algoritmo de control siempre va a obtener la misma salida ante un mismo estímulo.
- Coste: el coste de sus componentes es mucho menor que el de los circuitos analógicos.
- Implementación digital sin equivalentes analógicos: existen componentes con características específicas que no se podrían aplicar analógicamente.
- Existencia de un gran número de sistemas o herramientas de diseño electrónico: con algoritmo de control preprogramados, que hace que el proceso sea rápido y fácil de diseñar.

El esquema general del proyecto a desarrollar tiene las siguientes partes, divididas en obtención de la señal de audio, procesamiento digital en el dispositivo FPGA y la reproducción del audio mediante unos altavoces.

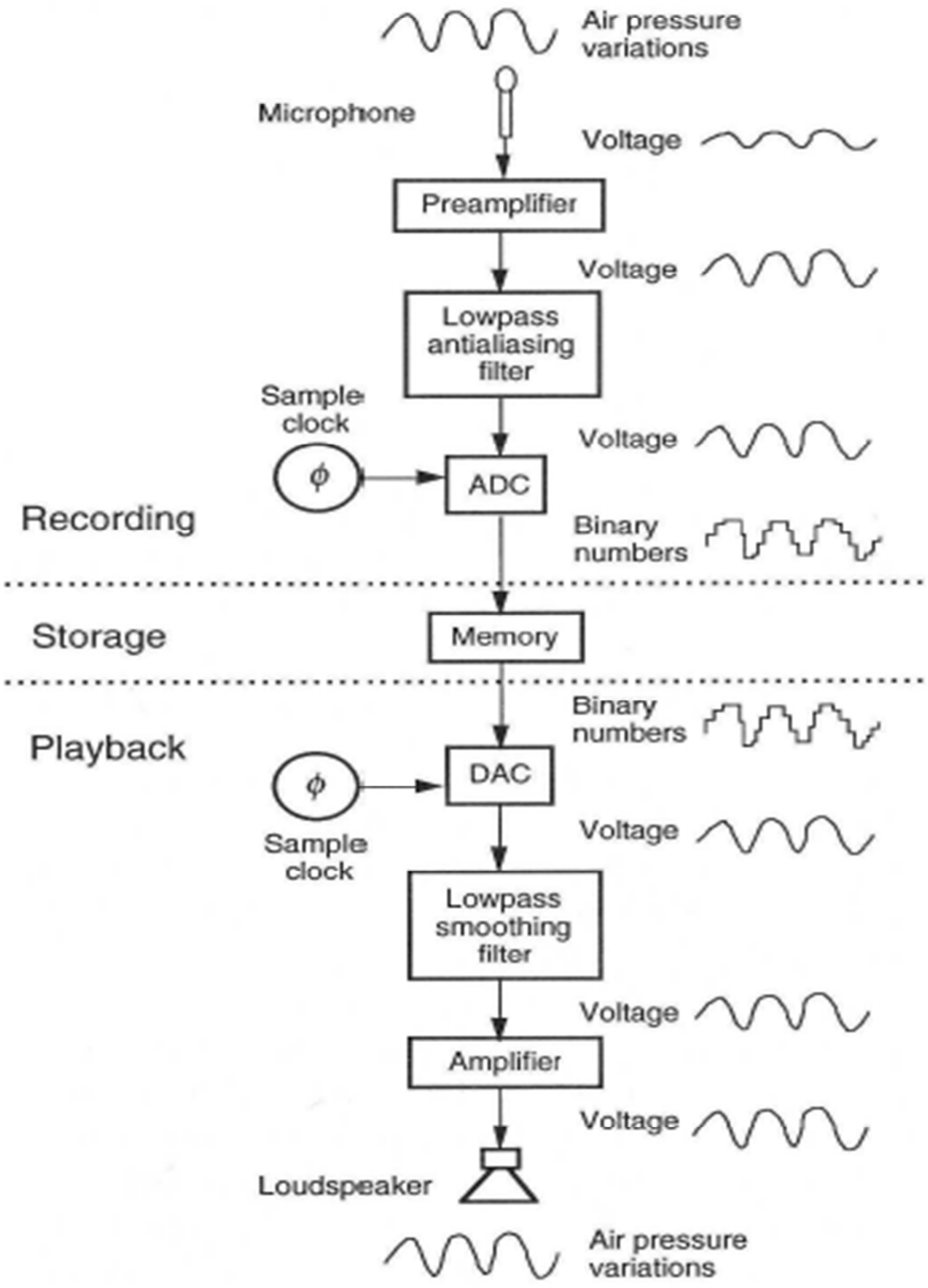


Figura 3: Etapas de procesamiento digital de audio.

2.2 Procesamiento digital:

Podemos afirmar que realizamos procesamiento digital de señales cuando modificamos la representación digital de la señal. En este caso, la señal de la magnitud de interés es el sonido. Esta modificación se realizará a través de efectos de audio.

Un efecto de audio o efecto sonoro es la modificación hecha a una señal de audio que genera un cambio en la manera que percibimos dicha señal. Pueden ser desde pequeños cambios como la eliminación de ruidos de fondo, hasta grandes cambios como la variación completa de la señal original.

Prácticamente, todos los sistemas de tratamiento de audio permiten realizar cambios a las características de la señal mediante la variación de los parámetros de control, que varían la señal inicial de entrada a una señal de salida deseada.

El procesamiento digital está presente en todo el sistema de audio, desde la generación, tratamiento y reproducción del mismo, para obtener las características deseadas en esta señal.

Existen muchos tipos de efectos o procesos digitales de audio, clasificados en dos grandes grupos:

- Dominio temporal: estos procesos trabajan directamente en la modificación de dichas muestras de la señal de audio para generar las transformaciones.

- Dominio frecuencial: en este tipo de procesamiento lo primero que se debe realizar es el análisis de la señal para obtener su representación en frecuencia. Una vez obtenida esa información, se transforma al dominio de frecuencia, y por último se obtiene la síntesis para convertir la representación frecuencial modificada en el nuevo dominio de frecuencia, que generará la señal de audio transformada.

1.2.1 Dominio del tiempo:

Los efectos de audio más utilizados son los retardos temporales. Un retardo digital puede considerarse como un espacio de memoria que contiene un número de casillas para almacenar muestras de audio. Este tipo de estructuras se denominan normalmente FIFO, y se puede representar como un buffer o una memoria circular, en la que existe un punto entrada, un espacio de almacenamiento de dato digital y un punto de salida del mismo.

El tiempo de duración del retardo depende de un puntero de lectura de la memoria FIFO, pero el número de posiciones determina el retardo máximo que puede tener la memoria.

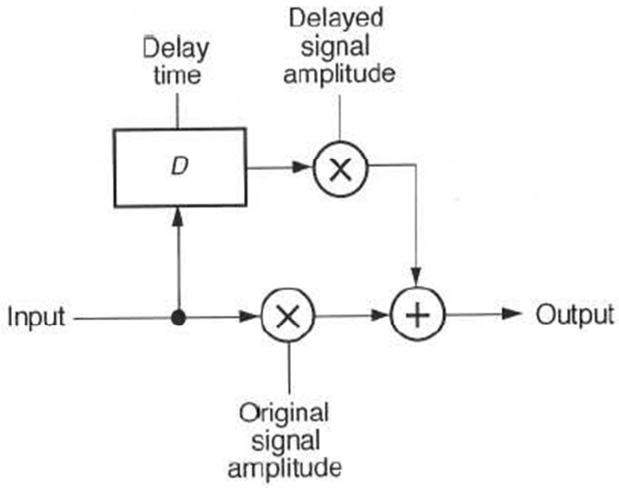


Figura 4: Esquema de bloques de procesamiento de audio con retardo.

La duración del retardo puede ser fija o variable en el tiempo.

A. Retardo fijo: Se divide principalmente en tres grupos:

1. Retardos fijos de duración inferior a 10 ms: a este efecto se le denomina efecto de filtro LP. Prácticamente no es audible por el oído humano.

2. Retardos fijos de duración entre 10-40 ms: se le denomina efecto de filtro en peine, y proporciona una sensación de crecimiento o duplicación de la señal. Con ello se consigue la anulación del refuerzo de la fase entre la señal retardada y la señal original.

3. Retardos de duración mayor a 50 ms: es el denominado efecto de eco (echo en inglés), y se produce cuando comenzamos a diferenciar entre la señal retardada y la original. Este efecto tendrá una sensación de duplicación de sonido, y es similar a si la señal de audio chocará contra un muro y esta rebotara. Como sabemos que la velocidad del sonido es 340 m/s, se comenzará a producir efecto de eco cuando existe una pared a una distancia de 7 metros en la que rebote la onda, ya que se reenvía 50 ms después de ser producida.

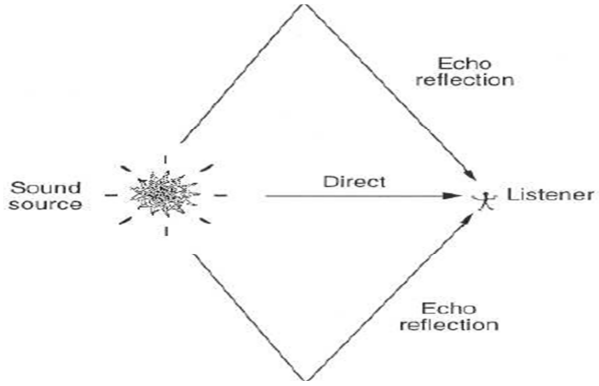


Figura 5: Representación gráfica del efecto de eco.

B. Retardos variables: se realizan a través de la modulación de la posición de los puntos de lectura de la memoria circular, es decir, variando la duración del retardo.

Estos retardos variables producen cuatro efectos principalmente:

1. Flange: Este efecto se produce cuando se escucha un sonido metalizado oscilante en la señal de salida. El tiempo de retardo se controla con un oscilador de baja frecuencia, que forma oscilaciones con frecuencias entre 0.1 y 20 Hz. Este efecto se puede conseguir duplicando la señal original, manteniendo una de las ondas limpia de procesado y a la otra onda se le aplica un delay de 5 a 25 ms.
   Algunos parámetros que se podrán modificar son la profundidad del efecto con la proporción de la señal retardada a la suma del total, el tiempo de retardo medio y la frecuencia del oscilador de baja frecuencia.

Las imágenes siguientes corresponden a la representación gráfica del efecto de flange y la implementación en un sistema digital.

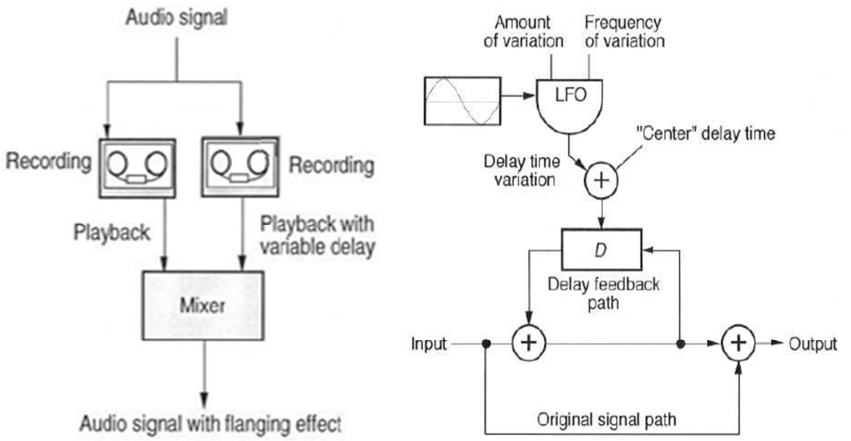


Figura 6: Efectos de flange e implementación en un sistema digital.

2. Phasing: es un efecto similar al efecto de flange, pero la alteración de su timbre no es tan acusada. Se produce a través de un filtro pasa-todo, con respuesta plana que solo afecta a la fase de la señal. Se utiliza un oscilador de baja frecuencia para controlar la cantidad que se desplaza la fase introducida en el filtro pasa-todo. Por último, se suma la señal del filtro con la señal original.

3. Coro: este efecto digital necesita diferencias entre las voces de la mezcla como retardos, alteraciones en la frecuencia y modificaciones en su tono. Se puede producir enviando la señal de audio a una línea con retardo multi-etapa, donde el tiempo de retardo varía en un rango pequeño de frecuencias, modificando la señal y produciendo desafinaciones en el tiempo.

4. Transposición: Al avanzar o retroceder el puntero de lectura de la memoria FIFO para producir retardos, observamos que la señal de sonido aumenta o disminuye de tonalidad según variamos su tiempo de retardo. Esto se produce ya que la frecuencia de muestreo se acelera o se reduce según la velocidad del retardo, pudiendo variar así la frecuencia de muestreo. Si esta frecuencia es continua, la transposición que obtenemos en la señal es contante.

1.2.2 Dominio de la frecuencia:

Son los efectos en los que se varía su contenido de frecuencia para crear modificaciones en la señal de salida. Pueden ser:

A. Transposición: Se puede hacer independientemente de los retardos, variando el contenido frecuencial de dicha señal. Este efecto consiste en variar la altura de la señal manteniendo constante la relación de frecuencia entre sus armónicos, por ejemplo, multiplicando sus cuatro primeros armónicos por una constante numérica. Este efecto se puede utilizar para un armonizador, ya que simula que hay una serie de acordes, o para el efecto de coro dando la sensación que hay más de un instrumento sonando simultáneamente.
B. Desplazamiento en frecuencia o modulación de banda lateral: Se basa en sumar o restar la misma frecuencia a todos sus armónicos, cambiando así su altura y su timbre, obteniendo sonidos inarmónicos debido a que varían las relaciones entre estos.
C. Morphing: Se entremezclan las propiedades de las señales de audio para obtener una señal híbrida, mezclándose los parámetros de Fourier o del espectro del sonido contenidos en el modelo senoidal.

Reverberación:

Es un fenómeno que resulta de la interacción entre la fuente de sonido y el medio en el que se propagan dichas ondas. Es el resultado de la mezcla de miles de ecos producidos al chocar las ondas sonoras contra los objetos del medio, tales como paredes, mesas, muebles, etc. Esta concentración debe ser lo suficientemente densa para dar una sensación de continuidad del sonido. La reverberación artificial se produce a partir de algoritmos que utilizan filtros y retardos. Los retardos se usan para reproducir las reflexiones primarias (mayor de 10 ms) y los filtros, principalmente en peine, se usan para representar los múltiples ecos y la reinyección de las señales digitales.

Técnicas futuras:

El proceso digital de la señal de audio es un campo aún con muchas posibilidades de desarrollo, ya que cada poco tiempo surgen nuevas técnicas de procesamiento digital. La más novedosa es la síntesis cruzada, basada en los análisis de la transformada rápida de Fourier (FFT) que permiten obtener métodos híbridos de señales. El análisis de la FFT permite caracterizar la respuesta en frecuencia en un sistema lineal, ya que para analizar un conjunto de señales infinitas, como una señal senoidal, se necesitan trabajar con señales discretas y periódicas.

Capítulo 3. Alternativas de diseño.

En esta parte del trabajo se analizarán las posibles alternativas existentes para la implantación del sistema que se podrían haber desarrollado en lugar de un dispositivo FPGA.

Esta comparación se realizará tanto económica como tecnológicamente, de manera que quede claro cuál sería la mejor opción que se podría haber implementado para dicha plataforma de audio.

Para su implementación se partirá de un circuito integrado, también llamado chip o microchip, compuesto de una oblea de un material semiconductor, generalmente silicio, sobre el que se fabrican ciertos circuitos electrónicos que cumplen con una determinada función. Estos son creados a partir de un fotolito, protegido con un elemento plástico o cerámico.

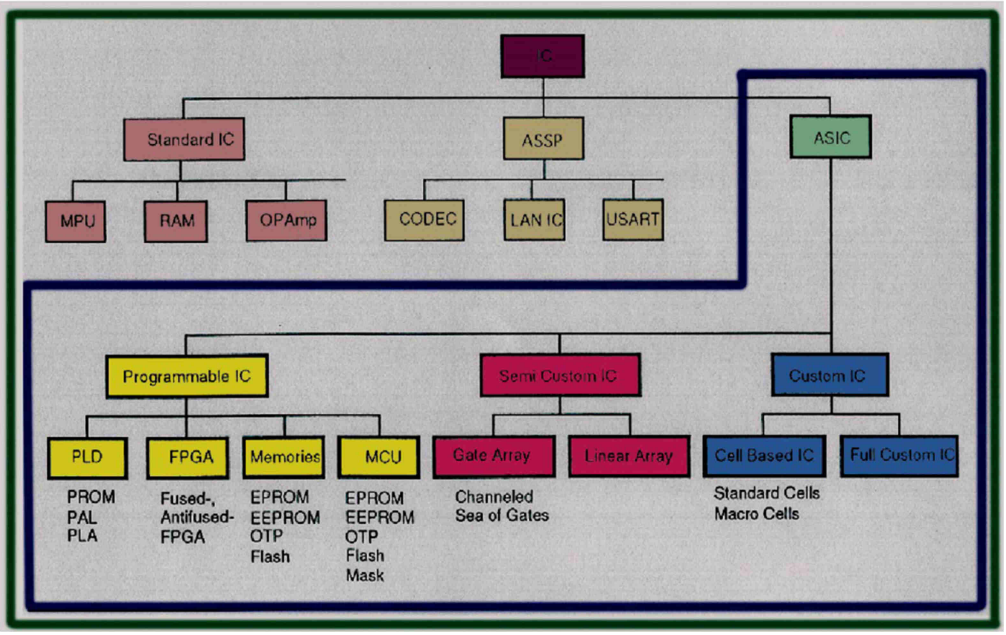


Figura 7: Grafo de clasificación de dispositivos electrónicos.

Comenzaremos analizando los circuitos integrados estándar, con las unidades de procesamiento o microprocesadores, memorias RAM o ROM y los amplificadores operacionales.

Más tarde definiremos a los ASSP, que son circuitos para una aplicación específica no reprogramables, y cuáles son sus diferentes topologías y sistemas de fabricación.

Por último, analizaremos el campo de los ASIC, que son circuitos de aplicaciones específicas, también no reprogramables, y de cuáles son sus diferentes topologías y propiedades.

3.1 Microprocesadores.

Los circuitos integrados estándar constituyen bloques funcionales, que son suministrados por los fabricantes con unas características lógicas y eléctricas perfectamente definidas. El diseño de sistemas con este tipo de topología presenta ventajas como su coste reducido y su gran facilidad de fabricación en serie con grandes lotes. Sin embargo, presentan varios inconvenientes, como falta de protección ante copias no autorizadas y el gran número de circuitos necesarios para la realización de su diseño.

El grupo más popular son los llamados microcontroladores. Estos son circuitos integrados programables, capaces de ejecutar órdenes grabadas previamente por el programador en su memoria. Los microcontroladores están diseñados para reducir el costo económico y el consumo de energía para un sistema en particular.

Sus principales unidades funcionales son las mismas que las de una computadora, y se dividen en unidad central de procesamiento (CPU), memoria (normalmente de tipo RAM, ROM, PROM y EPROM) y periféricos de entrada y de salida.

A. CPU: La unidad de control es uno de los elementos fundamentales que determinan las prestaciones del procesador, ya que su tipo y estructura determinan parámetros tales como el tipo de conjunto de instrucciones, velocidad de ejecución, tiempo del ciclo de máquina, tipo de buses que puede tener el sistema, manejo de interrupciones y un buen número de más propiedades que en cualquier procesador van a parar a este bloque.

B. Memoria: En los microcontroladores la memoria RAM no suele ser abundante, ya que aquí no se hablará de Gigabytes de memoria, como en las computadoras personales, sino que será de unos pocos Megabytes de almacenamiento interno. En aplicaciones quye precise mucha memoria RAM, esta deberá ser externa al dispositivo.

C. Periféricos: Cuando observamos la organización básica de un microcontrolador, señalamos que dentro de este se ubican un conjunto de periféricos de entrada y salida. Alguno de estos periféricos se clasifican en:

   - Entradas y salidas de propósito general: También conocidos como puertos de E/S, permiten leer datos del exterior o escribir en ellos desde el interior del microcontrolador. Algunos puertos de E/S tienen características especiales que le permiten manejar salidas con determinados requerimientos de corriente y/o tensión, o incorporan mecanismos especiales de interrupción.

   - Conversor analógico/digital: Como es muy frecuente el trabajo con señales analógicas, estas deben ser convertidas a formato digital y por ello muchos microcontroladores incorporan un conversor analógico-digital (A/D), el cual se utiliza para tomar datos de varias entradas diferentes. Las resoluciones más frecuentes son 8 y 10 bits, que son suficientes para aplicaciones sencillas.

A continuación podemos ver una lista con la evolución histórica de los microcontroladores que han existido:

- **Intel 8048:** es el primer microcontrolador moderno que se fabricó. Ganó su popularidad debido a su reducido precio, disponibilidad y herramientas de diseño electrónico.
- **Intel 8051:** Fue el más popular en la década de los 80, ya que era muy potente y fácil de programar.
- **Motorola 68HC11:** Es un microcontrolador de 8 bits, potente y con gran cantidad de aplicaciones y variantes.
- **Intel 80186**: Primera versión del microcontrolador de Intel que era capaz de aprovechar los programas y herramientas de desarrollo para computadoras personales,
- **Microchip PIC:** es una arquitectura de microprocesadores con características RISC (conjunto de instrucciones reducidas), en el cual las instrucciones son fijas y solo las instrucciones de carga y almacenamiento pueden acceder a la memoria.

Estos microcontroladores tienen diferentes características principales, como:

- Memorias Flash y ROM hasta 256 Kbyte
- Núcleos CPU de 8 o 16 bits.
- Módulos de captura y comparación PWM.
- Temporizadores de hasta 32 bits.
- Puertos de entrada/salida.
- Soportes de interfaz USB.
- Soportes para controladores Ethernet, CAN, LIN.
- Controladores LCD.

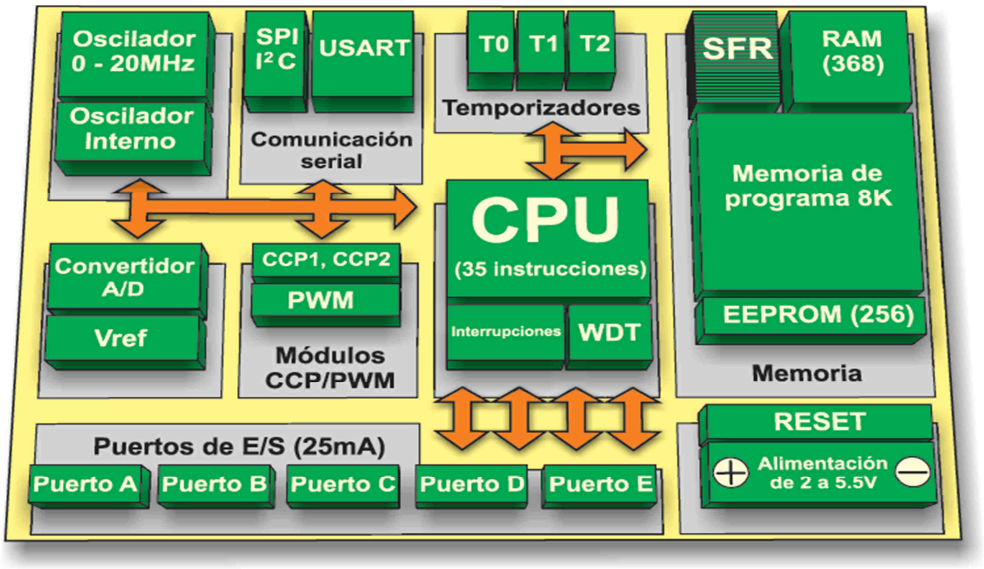


Figura 8: Estructura de microchip PIC.

- **Arduino:** es una plataforma de hardware libre, basada en una PCB (Printed Circuit Board) con un microcontrolador Atmel y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. Debido a su bajo coste y sencillez de programación, es una alternativa importante a la hora de automatizar procesos mediante un microprocesador con distintos elementos de control y procesamiento. Además su entorno de programación es sencillo, siendo un código muy parecido al lenguaje C.

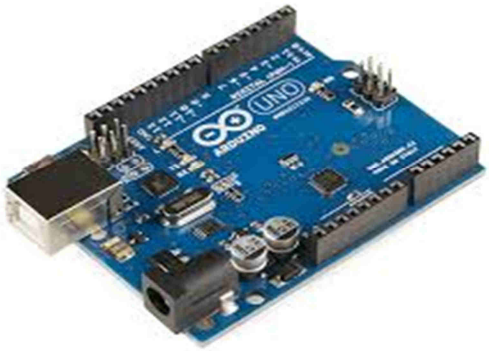


Figura 9: Fotografía de Arduino UNO.

Conclusión: Si hubiésemos decidido hacer el procesamiento de este proyecto mediante un microcontrolador, habría sido una opción perfectamente válida. Su precio reducido y la existencia de entornos de programación sencillos la convierten en una propuesta válida para la mayoría de los procesos de procesamiento y control.

Sin embargo, el principal inconveniente de estos dispositivos es la resolución que poseen los conversores A/D y D/A es muy pequeña (de unos 8 bits normalmente), y la resolución necesaria es de 16 bits para el desarrollo del proyecto, debido a que el formato de audio que vamos a utilizar es de 16 bits.

Además tiene una memoria RAM muy escasa, por lo que para el procesamiento digital sería imposible realizarlo eficientemente. Por ello, en este trabajo se desarrolla con un hardware FPGA y no se aplicarán los diseños de dichos microprocesadores.

3.2 ASSP (Productos estándar de aplicación específica):

Un ASSP (productos estándar de aplicación específica) es un dispositivo o circuito integrado que se dedica a una aplicación específica de mercado y es vendido a más de un solo usuario, por tanto, es un producto estándar.

El ASSP se comercializa a múltiples clientes como un producto de uso general para una aplicación específica. Al igual que un circuito ASIC (circuito integrado de aplicación específica), el ASSP es para una aplicación especial, pero se vende a cualquier empresa que necesite realizar esa aplicación. Sin embargo, un ASIC está diseñado y construido por encargo de una empresa específica, por lo que no es general.

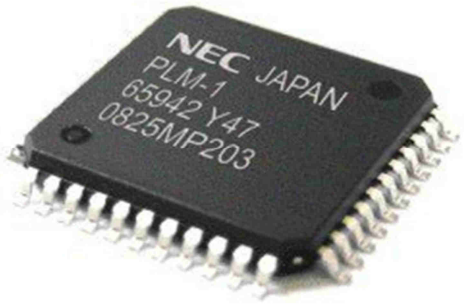


Figura 10: Fotografía de dispositivo ASSP.

Conclusión: Un ASSP sería una posibilidad si deseamos construir un gran número de circuitos con una misma configuración, ya que económicamente el precio es relativamente bajo con respecto a las otras opciones. También tiene muchos inconvenientes, como la imposibilidad de modificar sus configuraciones, por lo que lo convierte en un dispositivo muy rígido.

Independientemente de estos inconvenientes, la dificultad reside principalmente en encontrar un circuito estándar que realice exactamente lo que queremos implementar en este proyecto, pero este no es el objetivo, ya que lo que se pretende es diseñarlo y no comprar el diseño.

Por tanto, dentro de las opciones que consideraríamos en el mercado, esta no es una opción válida para nuestro proyecto.

3.3 ASIC (Circuito integrados de aplicación específica):

Se conoce como un circuito ASIC a un circuito integrado, diseñado a medida para un propósito o aplicación específica. Al contrario de la mayoría de alternativas, este puede tener funciones analógicas, digitales o combinación entre ambas funciones.

La gran ventaja de estos circuitos diseñados a medida es su tamaño reducido, por lo que tiene un menor coste por unidad. Además, en algunos casos se puede obtener mejoras en su rendimiento y un consumo menor, ya que al estar diseñado específicamente para una aplicación concreta, contiene el menor número de puertas posibles. Sus desventajas son el gran aumento del coste debido a su tiempo de desarrollo para obtener las configuraciones óptimas y la necesidad de usar un software de diseño electrónico CAD más complejo en equipos más cualificados.

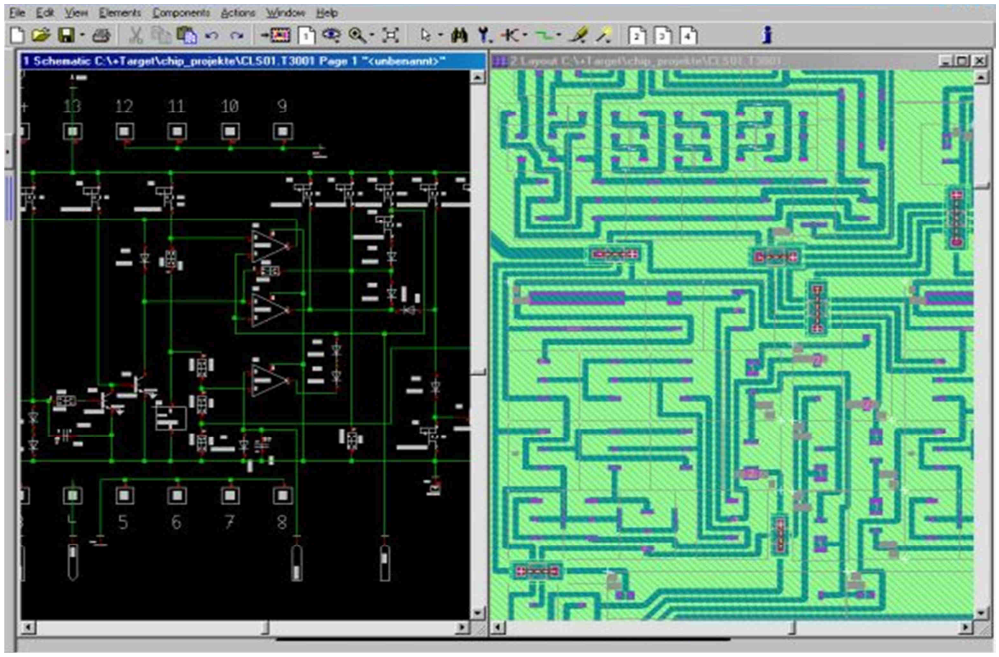


Figura 11: Herramienta de diseño digital del dispositivo ASIC.

Conclusión: Esta es una alternativa interesante en dispositivos electrónicos que vayan a ser fabricados en grandes series, ya que al necesitar una gran inversión inicial y que sea rentable es necesario producir un número mínimo. También es la opción idónea si se necesita una elevada velocidad de procesamiento, ya que son de las opciones más rápidas en el mercado actual.

3.4 PLD (Dispositivos lógicos programables).

Un dispositivo lógico programable o PLD es un componente electrónico estándar usado para construir sistemas digitales. Contiene una arquitectura general predefinida en la que el usuario puede programar la configuración final del dispositivo usando herramientas de diseño electrónico.

En muchos casos, estos dispositivos cuentan con la posibilidad de ser reprogramados y reconfigurados, lo que permite la reutilización en otros montajes con otras configuraciones.

Podemos distinguir dos tipos de PLDs dependiendo de la manera en la que son programados:

A. **PLD programables en fábrica:** estos son construidos en la fábrica, a partir de diseños electrónicos de acuerdo a los requerimientos del cliente. Esta construcción se lleva a cabo mediante procesos irreversibles, por lo que no se podrán hacer futuras modificaciones en el circuito. Las topologías más importantes son los PLD pregrabados y las memorias ROM.
B. **PLD programables en campo:** estos están diseñados para ser programados por el usuario con herramientas de diseño electrónico. Generalmente son reprogramables un número alto de veces. Las topologías más populares son las SPLD, CPLD y FPGA.

Actualmente existen cuatro posibles alternativas para circuitos PLD:

1. **ROM:** además de las unidades de memoria, estos dispositivos pueden actuar para realizar lógica combinacional. Tienen las siguientes características:

   - Son mucho más lentos que los circuitos lógicos dedicados, y son circuitos poco eficientes cuando el número de variables es elevado.
   - Consumen mayor potencia que el resto de alternativas.
   - Sólo se utiliza una pequeña fracción de su capacidad en una sola aplicación, por lo que tienen un uso ineficiente de los recursos.
   - Por sí misma no se puede utilizar para crear circuitos de lógica secuencial, puesto que no contienen elementos biestables.

2. **SPLD (Simple Programmable Logic Devices):** Son los dispositivos más simples, pequeños y baratos que podemos implementar en circuitos lógicos. Normalmente se le suelen introducir una macrocelda para conectar el dispositivo al exterior.

   A esta familia pertenecen dos topologías distintas:

   - **PAL:** Es un dispositivo de matriz programable. Su arquitectura interna se compone de un conjunto de puertas AND con sus conexiones programables que alimenta una matriz de puertas OR fijas. Es el dispositivo más usado en SPLD y CPLD debido a que son extremadamente rápidas y flexibles.

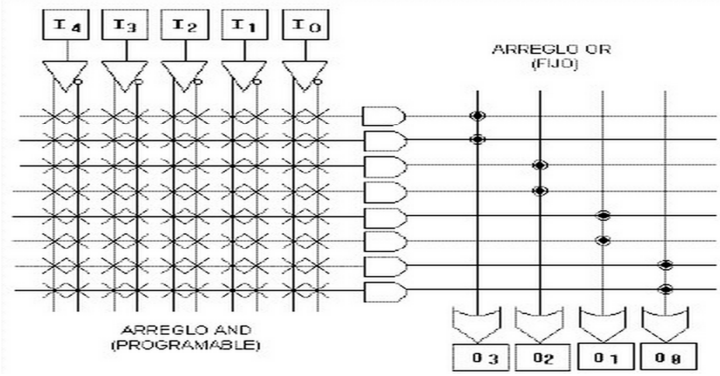


Figura 12: Estructura PAL.

   - **PLA:** se compone de dos matrices de conexiones que alimentan un conjunto de puertas AND y de puertas OR. En este caso, ambas matrices son programables, por lo que da la mayor flexibilidad de todas las PLD, pero son bastante lentas ya que necesita pasar por más enlaces que las otras PLD.

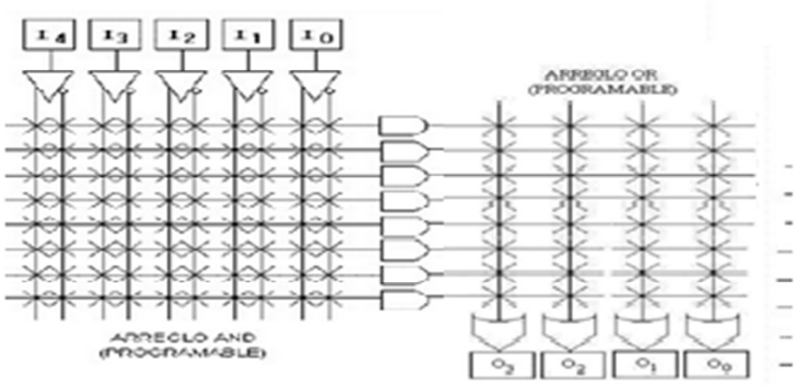


Figura 13: Estructura PLA.

3. **CPLD (Complex Programmable Logic Devices):** Estos dispositivos amplían el concepto de PLD a un mayor nivel de integración. Permiten la creación de circuitos integrados equivalentes a varios PLDs en un solo chip. Además utilizan menos espacio, mejoran la fiabilidad del diseño y reducen el coste.

   En cuanto a su arquitectura, un CPLD es una unión de una estructura PAL, una macrocélula y un asignador lógico. Los bloques lógicos se comunican entre sí utilizando una matriz programable de conexiones. Casi cualquier diseño puede ser implementado con estos dispositivos.

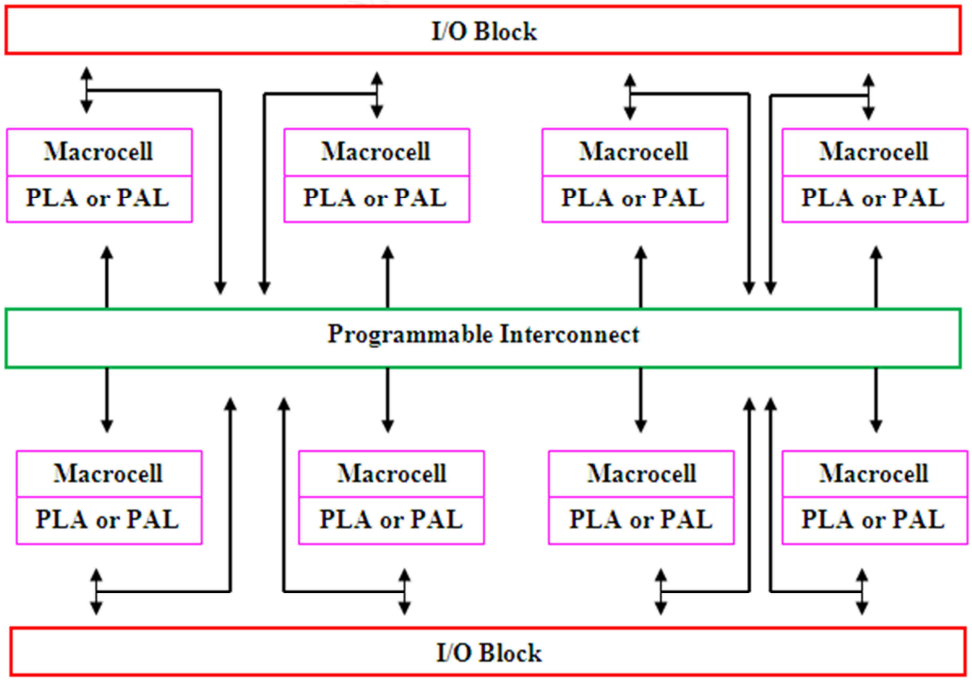


Figura 14: Esquema de la estructura del dispositivo CPLD.

4. **Dispositivo FPGA (Field Programmable Gate Array):** Un dispositivo FPGA es un circuito integrado que contiene matrices con bloques de lógica, cuyas interconexiones y funcionalidades pueden ser configuradas mediante un lenguaje de descripción especializado.

   Se caracterizan por las altas densidades de integración, alto rendimiento, un gran número de entradas y salidas definidas por el usuario, un esquema de interconexiones flexible y un entorno similar al de la matriz de puertas lógicas.

Los FPGAs son más lentos que los circuitos ASIC, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, los FPGAs tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Más adelante se definen con más detalle los dispositivos FPGA, tanto sus características como sus diferentes funcionalidades, modelos y usos actuales.

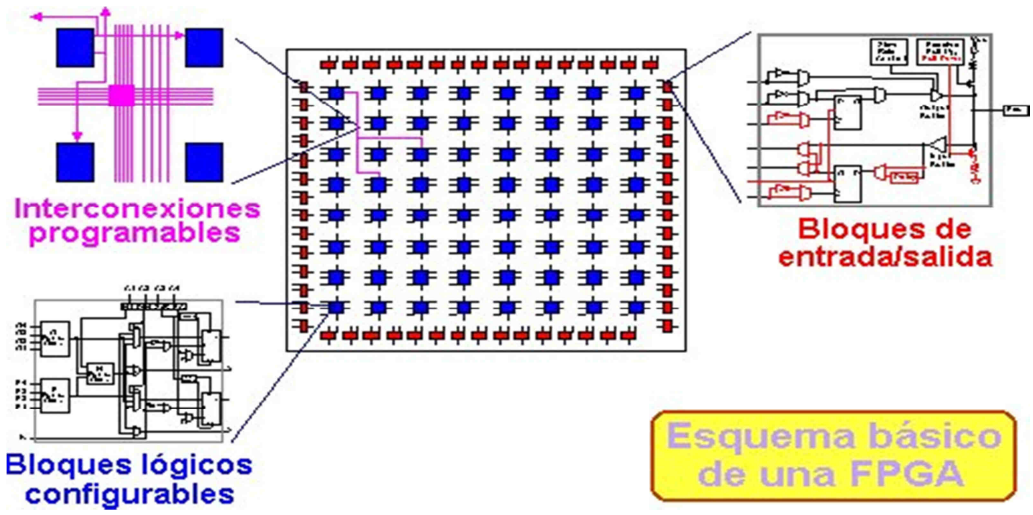


Figura 15: Esquema de estructura de un dispositivo FPGA.

Las diferencias, tanto ventajas como desventajas de los FPGA con respecto a las CPLD, son:

- La diferencia fundamental entre los FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida, ya que está formada por una o varias sumas de productos. La arquitectura de los FPGAs se basa en un gran número de bloques pequeños usados para realizar operaciones lógicas sencillas.
- Tanto los CPLDs como los FPGAs contienen un gran número de elementos lógicos programables, pero midiendo la densidad de los elementos lógicos programables en puertas lógicas equivalentes (número de puertas NAND equivalentes que podríamos programar en un dispositivo), podríamos decir que los FPGAs tienen del orden de 10 a 100 veces más cantidad que las CPLDs.

- En la mayoría de los FPGAs se pueden encontrar funciones de alto nivel (como sumadores y multiplicadores), además de bloques de memoria, en cambio en las CPLDs no existen dichos módulos.

Conclusión: Para la realización de este proyecto, el dispositivo que usaremos será un dispositivo FPGA debido a su gran memoria RAM y su flexibilidad lógica para desarrollar tareas específicas.

También se ha decidido usarlos en este dispositivo por la existencia de un puerto I2C, el cual proporcionará toda la función de inicialización y configuración de los dispositivos.

A pesar de ello, estas también tienen características negativas como su menor velocidad y la imposibilidad de abarcar procesos y sistemas tan complejos como otros dispositivos de las alternativas propuestas.

Capítulo 4. Estado del arte, selección y breve descripción de los dispositivos.

Como ya se indicó en el primer capítulo, una de las metas de este trabajo fin de grado es el estudio de las posibilidades de los dispositivos FPGA.

En este capítulo se analizarán las características de los dispositivos elegidos para la realización e implementación del proyecto. Comenzaremos analizando primeramente el dispositivo FPGA, en el cual se analizarán de manera breve los fabricantes con sus diferentes topologías y el uso que se hará de dicho dispositivo en el trabajo. Después, se describirá de manera concreta el códec de adquisición y conexión de audio que se ha utilizado para establecer dichas conexiones.

4.1 Dispositivos FPGA

Un dispositivo FPGA es un circuito integrado que contiene matrices con bloques de lógica, cuyas interconexiones y funcionalidades pueden ser configuradas mediante un lenguaje de descripción electrónico especializado.

Ciertos fabricantes cuentan con FPGAs que sólo se pueden programar una vez, por lo que sus ventajas e inconvenientes se encuentran a medio camino entre los ASICs y los FPGAs reprogramables.

En el FPGA no se realiza programación tal cual como se realiza en otros dispositivos como CPLD o microcontroladores. Los FPGA realizan su lógica en base a celdas que se configuran con una función específica, ya sea como memoria (FLIP-FLOP tipo D), como multiplexor o con una función lógica tipo AND, OR, XOR. Por ello, la tarea del usuario será programar, tanto las funciones realizadas por cada bloque como las interconexiones entre ellos.

El programador cuenta con la ayuda de software y entornos de desarrollo especializados para realizar los diseños de sistemas a implementar en un dispositivo FPGA. Un diseño puede ser capturado de diferentes maneras, ya sea como diseño esquemático o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL (lenguajes de descripción de hardware).

Los lenguajes HDL más conocidos y utilizados son:

- VHDL
- ABEL
- Verilog

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando el dispositivo FPGA disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan los FPGA son los DSP (procesamiento digital de señales), ya sea en radio definido por software, sistemas aeroespaciales y defensa, prototipos de ASICs, sistemas de tratamiento de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática y emulación de hardware de computadora entre otras.

Cabe destacar que su uso en otras áreas es cada vez mayor, sobre todo en aplicaciones que requieren un alto grado de paralelismo y concurrencia.

4.1.1 Fabricantes:

Dentro del mundo de los dispositivos FPGA, existen diferentes fabricantes como Altera, Xilinx o Lattice Semiconductor. A continuación se explicarán brevemente las características de los dispositivos que ofrece cada uno de los fabricantes mencionados.

Altera:

Es una de las tres empresas líderes en la fabricación y desarrollo de los dispositivos FPGAs. Dispone de tres familias de FPGAs, y que dependiendo de las especificaciones del diseñador, deberá escoger entre una de ellas:

- Cyclone Serie: es la serie de FPGAs más simple, con menor potencia y con menor coste. Esta elección es correcta para las aplicaciones que no requieran gran poder de computación, memoria o procesamiento.
- Arria Serie: es la serie de la gama intermedia, que proporciona mejores características que las cyclone a un precio moderado.
- Stratix Serie: Es la serie de dispositivos FPGAs de alto rendimiento, con gran poder de computación pero con elevado precio.

La empresa Altera dispone de una herramienta de software de programación propio llamado Quartus II, utilizado para el análisis y la síntesis de diseños realizados en HDL. Quartus II permite al programador compilar sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino.

Xilinx:

Es la mayor de las tres empresas líderes en la fabricación y desarrollo de los dispositivos FPGAs. Entre su oferta de dispositivos FPGAs, destacan seis familias clasificadas dependiendo de su escala de integración y su densidad de puertas lógicas.

45nm	28nm	20nm	16nm
	VIRTEX 7	VIRTEX UltraSCALE	VIRTEX UltraSCALE+
	KINTEX 7	KINTEX UltraSCALE	KINTEX UltraSCALE+
SPARTAN 6	ARTIX 7		

Figura 16: Clasificación dispositivos FPGA de la compañía XILINX.

Se dividen sobre todo en tres topologías, siendo Artix la de menor rendimiento pero menor coste, Kintex la de especificaciones y precio intermedios, y por último Virtex, que es la de mejores características y mayor precio.

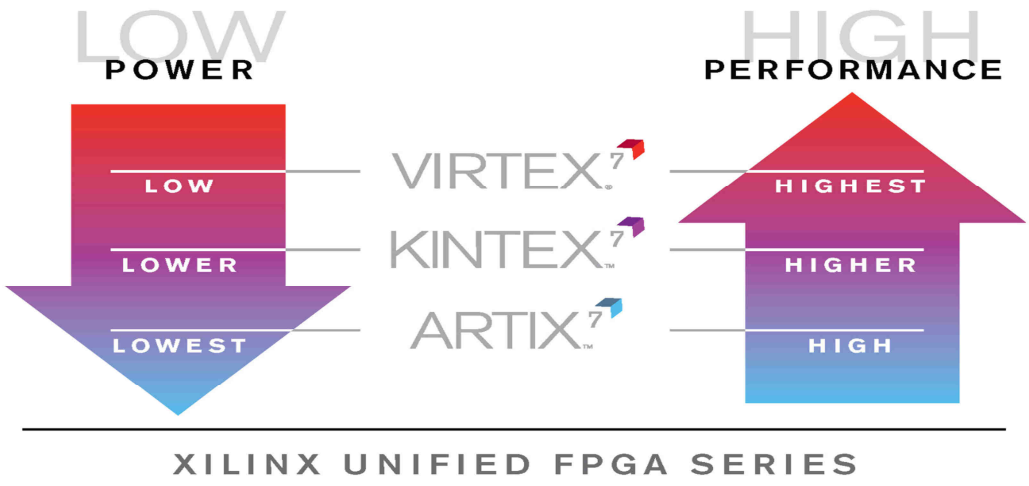


Figura 17: Clasificación de las FPGA de XILINX por potencia y prestaciones.

La empresa Xilinx dispone de una herramienta de software de programación y diseño electrónico propio llamado Xilinx-ISE, utilizado en el análisis y la síntesis de diseños realizados en lenguaje HDL.

Lattice semiconductor:

Es una de las tres empresas líderes en la fabricación y desarrollo de los dispositivos FPGAs, líder en diseño de circuitos integrados programables de ultra bajo consumo. Entre sus ofertas de dispositivos FPGAs, existen multitud de opciones dependiendo del nivel de prestaciones y precios.

A continuación se presentan las principales familias con sus características:

- ECP5 FPGA Family: dispositivos de baja densidad, poco consumo y precio asequible. Es ideal en aplicaciones con alto volumen de pedidos, en la que el coste es un factor muy determinante.
- iCE40 FPGA Family: dispositivo con características y precio moderados. Se comercializa en tres distinciones: bajo consumo, bajo consumo con IP embebido o alto rendimiento.
- MACHX03 FPGA Family: dispositivos con memoria no volátil. Poseen mejores características y un rendimiento mejorado.
- MACHX02 FPGA Family: dispositivos perfectos para una rápida implementación de sistemas de control y procesamiento. Su precio es relativamente bajo.
- ispMACH 4000ZE FPGA Family: dispositivos diseñados para aplicaciones móviles donde sus especificaciones exigen un mínimo consumo. La corriente típica de standby de estos dispositivos es de 10 microAmperios.

La empresa Lattice Semiconductor dispone de una herramienta de software de programación propio llamado Lattice Diamond, utilizado para el análisis y la síntesis de diseños realizados en HDL. Lattice Diamond permite al desarrollador compilar diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino.

Una de las características más importante de esta empresa es la construcción de dispositivos FPGA protegidos frente a copias no autorizadas, siendo uno de los dispositivos más seguros del mercado.

Conclusión: podemos afirmar que cualquiera de estas tres empresas disponen de dispositivos capaces para satisfacer las necesidades del trabajo. Pero hay que remarcar que los dispositivos de Lattice Semiconductor disponen de dispositivos con memoria no volátil y protegido frente a copias, siendo ventajas muy importantes con respecto a sus competidores.

4.2 Modelo elegido y software utilizado:

Como ya se adelantó en el primer capítulo, el modelo elegido para el control y procesamiento del proyecto es una FPGA Lattice MACHX02-7000-HE, ya que es el dispositivo que está integrado dentro de la placa preestablecida, disponible en el departamento de Tecnologías Electrónicas.

El significado de su nomenclatura es:

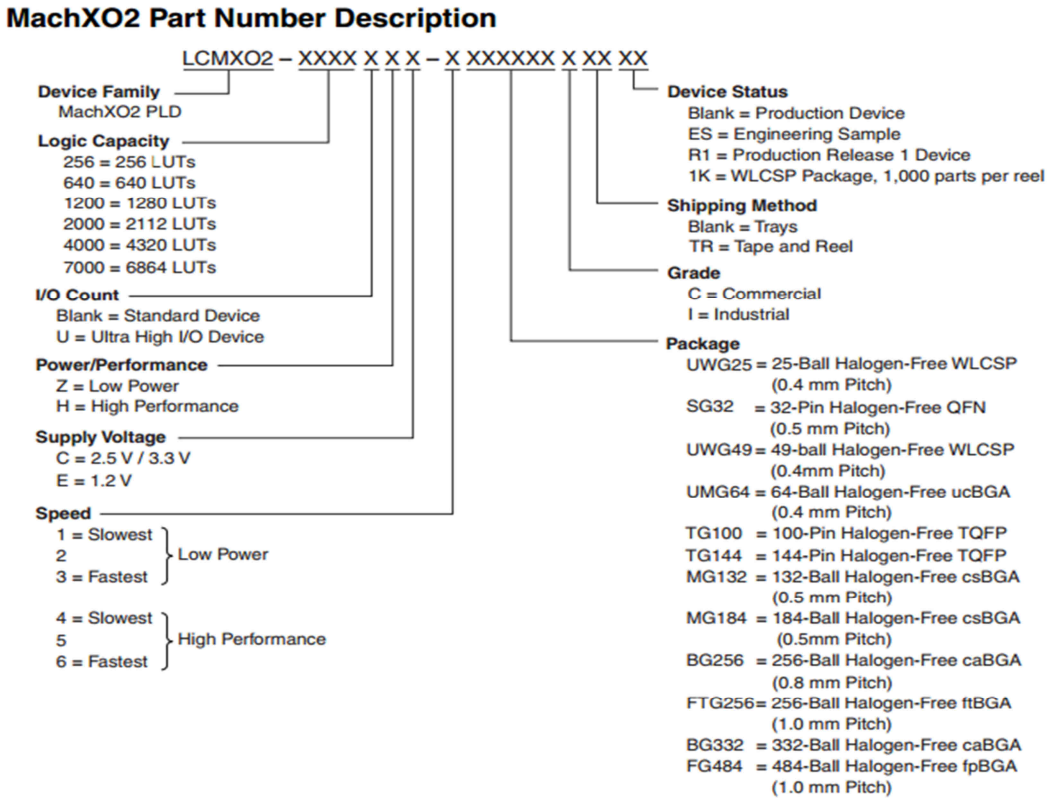


Figura 18: Nomenclatura de los dispositivos FPGA MACHX02 de Lattice.

En nuestros dispositivos tenemos:

- MachX02: corresponde a la familia del dispositivo FPGA.
- 7000: Capacidad lógica de 6864 LUTs.
- HE: Posee altas prestaciones a una tensión de 1.2V.

Otra ventaja que presenta este dispositivo es la existencia de un software gratuito de la propia empresa Lattice Semiconductor para programar sus dispositivos. Se trata del programa Lattice Diamond, que permite entre otras cosas el diseño, síntesis, análisis, programación, debugger, simulación e implementación de esquemas y circuitos electrónicos. En este proyecto se utiliza la versión 3.2 (para sistemas operativos de 64 bits).

A continuación se analizarán con más detalle las características del dispositivo FPGA y el software usado para realizar el diseño electrónico.

4.3 Características y protocolos del dispositivo:

En cuanto a la arquitectura, la familia MACHX02 se compone de bloques lógicos rodeados por E/S programables (PIO), además de bloques sysCLOCK PLL y bloques sysMEM RAM embebidos (EBRs)

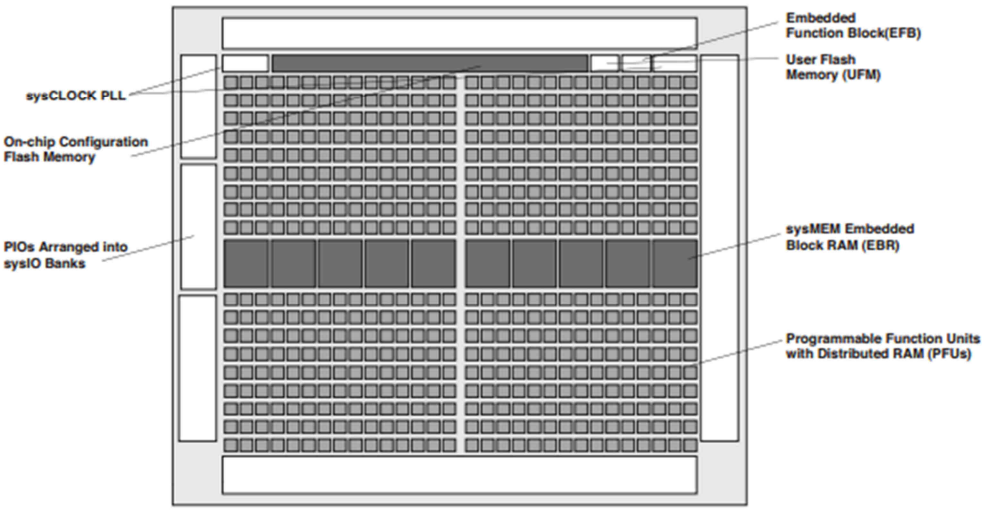


Figura 19: Estructura de un dispositivo FPGA.

- Los bloques lógicos, Unidad Funcional Programable (UFP) y EBR, están situados en una cuadricula de dos dimensiones (filas y columnas). Cada fila puede tener bloques lógicos o bloques EBR.
- Las UFP contienen los bloques correspondientes a la lógica aritmética, RAM, ROM y registros.
- Las PIO utilizan un buffer de I/O flexible y compatible con varios interfaces estándar.
- Los bloques PLL son muy flexibles y útiles. Estos circuitos se utilizan principalmente para ajustar la frecuencia de la onda de una fuente externa o interna a la frecuencia deseada por el programador para una determinada aplicación.

Los bloques se interconectan entre sí mediante pistas horizontales y verticales. La herramienta de diseño es la que se encarga de asignar automáticamente estos recursos de enrutamiento.

Todos los dispositivos de la familia MachXO2 incluyen un módulo EFB (Embedded Function Block). Este bloque permite implementar funciones de control que facilitan el diseño y permiten ahorrar recursos de uso general como LUTs, registros, señales de reloj y rutado.

Los módulos EFB convencionales contienen los siguientes elementos:

- Dos núcleos I2C.
- Un núcleo SPI.
- Un timer/counter de 16 bits.
- Interfaz para configurar las características del bloque PLL.
- Interfaz para configurar la lógica.
- Interfaz para controlar la disipación de potencia del chip.
- Memoria flash de usuario (UFM).

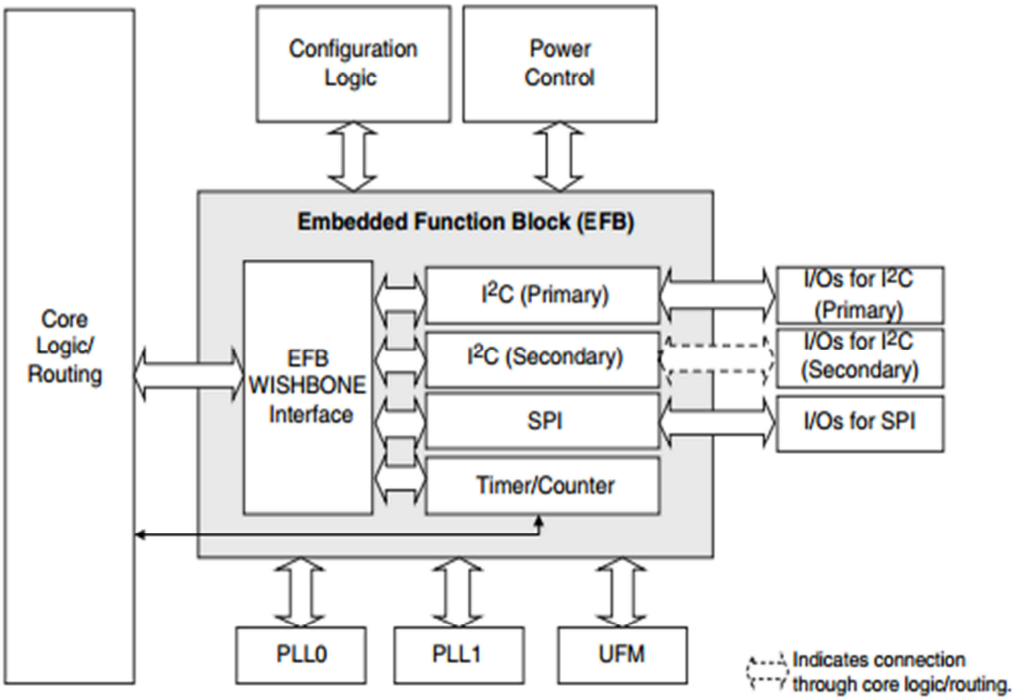


Figura 20: Esquema del bloque EFB.

4.3.1 Bus WISHBONE:

Es un hardware de código abierto destinado a permitir que las diferentes partes del circuito se comuniquen entre sí. Proporciona una conectividad eficiente entre el núcleo lógico del dispositivo con el bloque EFB, así como comunicación entre las distintas funciones individuales del mismo.

Esto se lleva a cabo mediante el bus WISHBONE, como puede verse en el esquema siguiente:

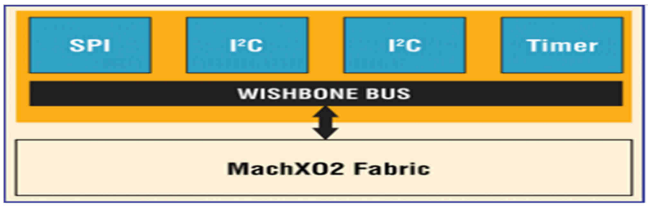


Figura 21: Esquema conexiones del bus WISHBONE.

Mediante la herramienta IP_Express, el usuario será capaz de crear y configurar un bloque EFB que implemente las funciones que necesite para su diseño. Se generará un fichero Verilog o VHDL que podrá ser usado en el diseño y funcionamiento de los proyectos.

La estructura en el bus de comunicación es maestro-esclavo. El usuario deberá diseñar su propio maestro para que interactúe con el esclavo.

Las señales de comunicación entre el WISHBONE maestro y el esclavo son las siguientes:

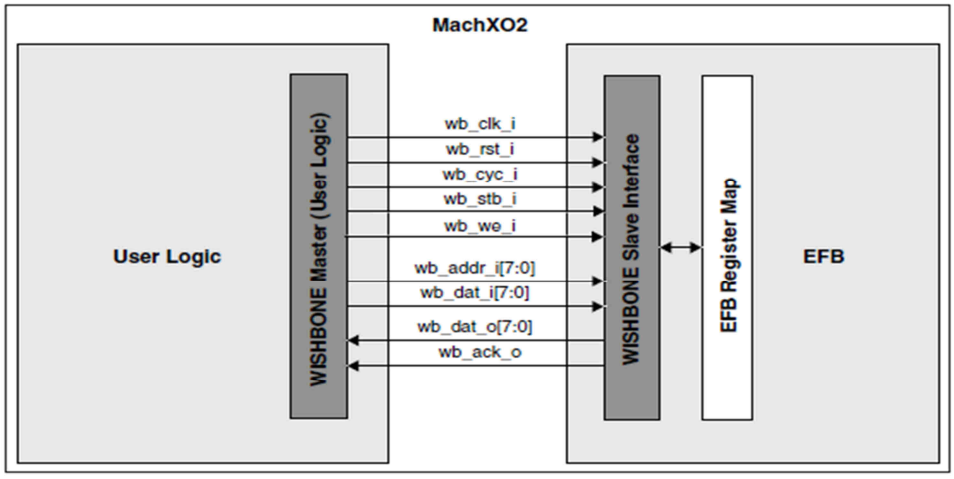


Figura 22: Esquema de conexiones del EFB mediante el bus WISHBONE.

- CLK_I: es la señal de reloj. Trabaja con frecuencias máximas de 133 MHz. La señal de configuración del bus debe ser como mínimo 7,5 veces inferior a la señal del oscilador.
- RST_I: señal síncrona de reset, activa a nivel alto.
- CYC_I: se activa para indicar el inicio de un ciclo de lectura/escritura en el bus maestro.
- STB_I: se activa para indicar el inicio de un ciclo de lectura/escritura en el bus esclavo.
- WE_I: indica si es de lectura (nivel bajo) o escritura (nivel alto).
- ADR_I: es un bus de 8 bits que indica la dirección del mapa de registro del bloque EFB.
- DAT_I: es un bus de 8 bits que contiene el byte a escribir.
- DAT_O: es un bus de 8 bits que contiene el byte leído.
- ACK_O: es un bit que confirma que el ciclo lectura/escritura ha finalizado correctamente.

El bus WISHBONE del MachXO2 es compatible con el estándar WISHBONE de OpenCores. Proporciona conectividad entre la lógica de aplicación FPGA y los bloques funcionales de EFB. El usuario puede implementar una interfaz WISHBONE maestro para interactuar con la interfaz WISHBONE esclava.

Ciclo de escritura en el bus WISHBONE:

En un ciclo de escritura, el WISHBONE maestro escribe un byte de datos en el bloque EFB. Se requieren al menos tres ciclos de reloj para completar la operación.

La forma de cada una de las señales para que este ciclo se lleve a cabo correctamente es la siguiente:

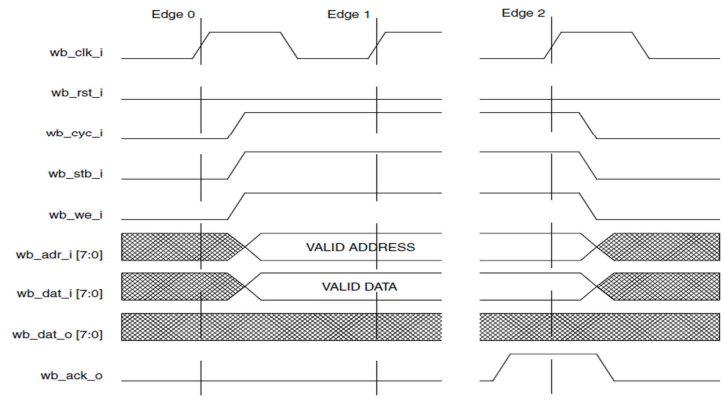


Figura 23: Esquema del ciclo de escritura del bus WISHBONE.

Ciclo de lectura en el bus WISHBONE:

De manera análoga al caso anterior, en un ciclo de lectura se lee un byte de datos del bloque EFB. Se requieren al menos tres ciclos de reloj para completar esta operación.

La forma de onda de cada una de las señales es la siguiente para llevar a cabo esta operación correctamente:

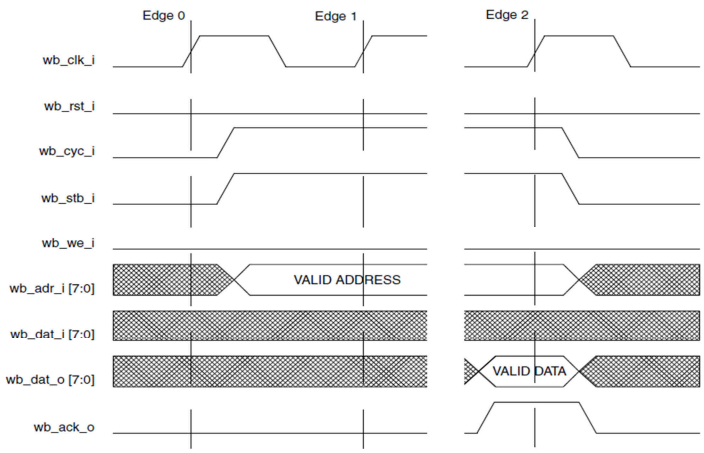


Figura 24: Esquema del ciclo de lectura del bus WISHBONE.

Existe un protocolo, llamado Hanshake, para la transferencia de datos, en el cual el maestro activa la señal STBO y lo mantiene hasta que el esclavo activa la señal de ACK

4.3.2 Timer/Counter:

Este módulo proporciona un temporizador/contador de 16 bits de propósito general. Es bidireccional y cuenta con una salida independiente. Permite además implementar control por modulación de ancho de pulso (PWM).

Dispone de señales específicas para comunicarse con el núcleo lógico de nuestro dispositivo a través del bus WISHBONE, aunque también existe la posibilidad de incluirlo en el diseño aunque no lo podría modificar de forma dinámica.

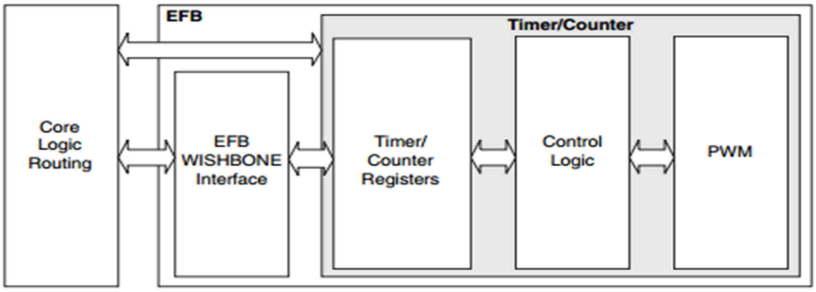


Figura 25: Esquema del contador de tiempo del bloque EFB.

Este contador se puede utilizar de cuatro modos de funcionamiento:

- Clear time on Comparate March: El contador se reinicia y se pone a 0x0000 cuando el valor del registro TCCNT alcanza el valor cargado en el registro TCTOP, que por defecto es el valor 0xFFFF.
- Watchdog Timer: Estos contadores se usan para controlar posibles bloqueos en los procesos de sistemas operativos o microcontroladores. Si detecta una interrupción, el contador Watchdog ejecuta una interrupción.
- Fast PWM: La salida se pone a nivel bajo cuando el valor del contador alcanza el valor cargado en el registro TCTOP, o se pone a nivel alto cuando alcanza el valor cargado en el registro de comparación TCOCR.
- Phase and Frecuency Correct PWM: El contador permite el funcionamiento de manera ascendente y descendente, lo que permite ajustar también la frecuencia y fase de la onda.

Toda esta información ampliada se puede encontrar en el manual del dispositivo “MACHX02 Family Handbook”, que puede ser descargado en la página web de Lattice Semiconductor.

4.3.3 Protocolo I2C:

El protocolo I2C es un estándar que facilita la comunicación o la trasferencia de información entre los microcontroladores, memorias y otros dispositivos, requiriendo únicamente dos líneas de señal y un común o masa. La velocidad estándar es de 100 KHz.

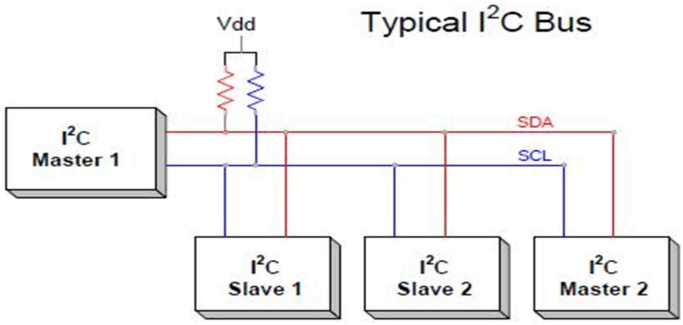


Figura 26: Esquema típico del bus I2C

Las líneas de este protocolo de comunicaciones son:

- SCL (system clock): es la línea de pulso de reloj que sincroniza el dispositivo.
- SDA (system data): es la línea por la que se mueven los datos del dispositivo.
- GND (ground): es línea común del dispositivo.

Las líneas SCL y SDA son de drenador o colector abierto. Se debe polarizar en estado alto por medio de resistencias de pull-up, lo que permite la conexión en paralelo de las I/O.

El protocolo de comunicación se establece sobre varios dispositivos y se distingue entre dispositivos maestros y esclavos. La condición inicial de bus libre es cuando ambas señales (SCL y SDA) están en estados lógicos altos. En este estado, el dispositivo maestro puede ocuparlo, estableciendo la condición inicial o de start condition (SDA=0 y SCL=1).

Si el dispositivo maestro quiere dejar libre el bus, necesitará generar la condición de parada o de Stop condition (SCL=1 y SDA=1).

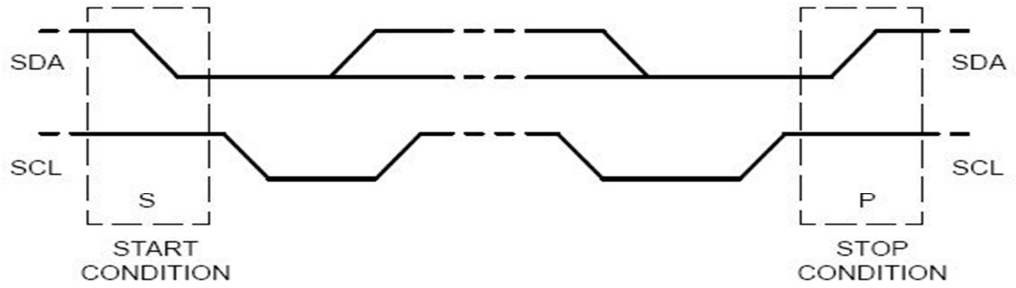


Figura 27: Esquema del ciclo del protocolo I2C.

El primer byte que se transmite después de la condición inicial consta de siete bits que transmite la dirección del dispositivo a seleccionar (pudiendo existir hasta 128 dispositivos con 7 bits) y el octavo bit es el que indica si es de lectura o escritura (1 si es escritura o 0 si es lectura). Si el dispositivo existe y está disponible, este contesta con un bit en nivel bajo, ubicado en el octavo bit que se envía al maestro (bit de ACK), que será 0 si es correcto. El dispositivo maestro procede entonces al envió de la información, hasta que ya se ha enviado toda y por último genera la condición de parada.

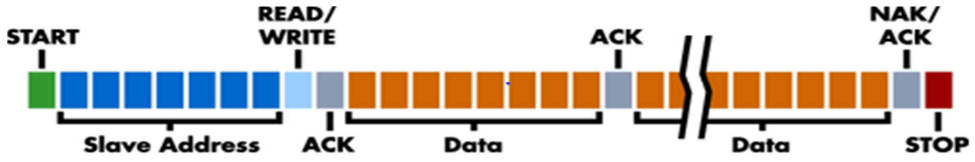


Figura 28: Estructura de datos del protocolo I2C.

El ciclo completo de escritura del protocolo I2C maestro es:

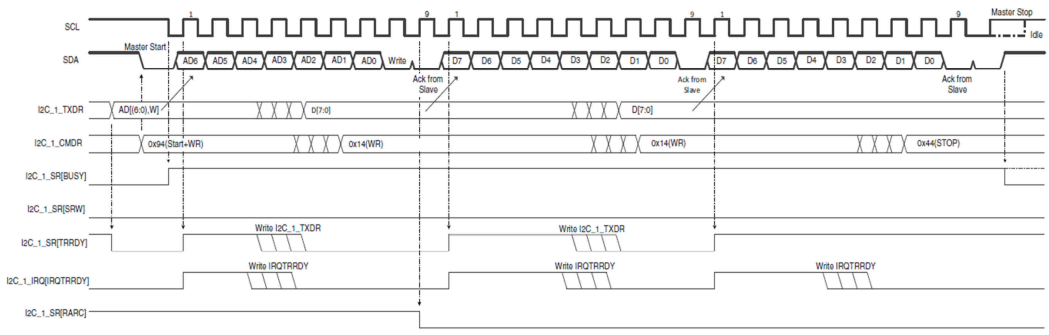


Figura 29: Esquema de ciclo de escritura del protocolo I2C maestro.

El ciclo completo de lectura del protocolo I2C maestro es:

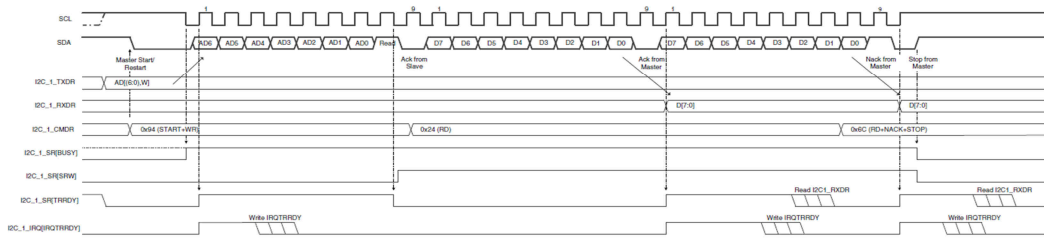


Figura 30: Esquema de ciclo de lectura del protocolo I2C maestro.

4.3.4 Protocolo SPI:

El protocolo SPI (serial peripheral interface) es un estándar de comunicaciones, usado para transferir información entre circuitos en un equipo electrónico. Este protocolo tiene interfaz serie, que controla cualquier dispositivo electrónico digital que acepte un flujo de bus serie regulado por un reloj (síncrono).

Tiene ventajas como la minimización en el número de conductores, pines, tamaño. Tiene también desventajas como mayor consumo que en el protocolo I2C, necesidad de direccionamientos exclusivos y no existe un bit de reconocimiento (ACK).

El ciclo de funcionamiento es el siguiente:

1. Se habilita el chip al cual hay que enviar la información mediante un bus de habilitación.
2. Se carga el buffer de salida.
3. El reloj comienza a generar una señal cuadrada, en el que por cada flanco se pone un bit en MISO.
4. El receptor en cada flanco de subida captura el bit de la línea MISO y lo incorpora al buffer.
5. Acabado de transmitir toda la información, la línea de habilitación pasa a un estado de reposo.

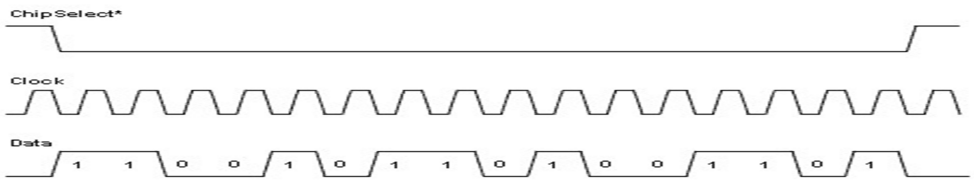


Figura 31: Esquema de ciclo del protocolo SPI.

4.4 Características de la placa:

El dispositivo FPGA está implementado dentro de una Breakout Board MACHXO2 del fabricante Lattice. Esta placa es barata y sencilla de programar. Se puede encontrar en la página web de Lattice Semiconductor, en la sección de placas y kits.

Cada pin de entrada/salida del dispositivo FPGA está conectado a un orificio de 100 milésimas de pulgada para poder conectar los diseños propios del usuario, que suelen estar en PCB externas.

El tamaño de la Breakout Board es de 7,5 x 7,5 cm. Cuenta en la parte superior con una matriz de diodos led que se usarán para las notificaciones, una matriz de prototipos para el desarrollo de pruebas. También contiene diodos led para hacer pruebas de diseño y para comprobar su consumo de potencia.

Por último, un puerto USB Mini-B por el que se conectará al ordenador y se podrá configurar el FPGA desde un software de diseño externo, generalmente por ser dispositivos de Lattice se usará el software de Lattice Diamond.

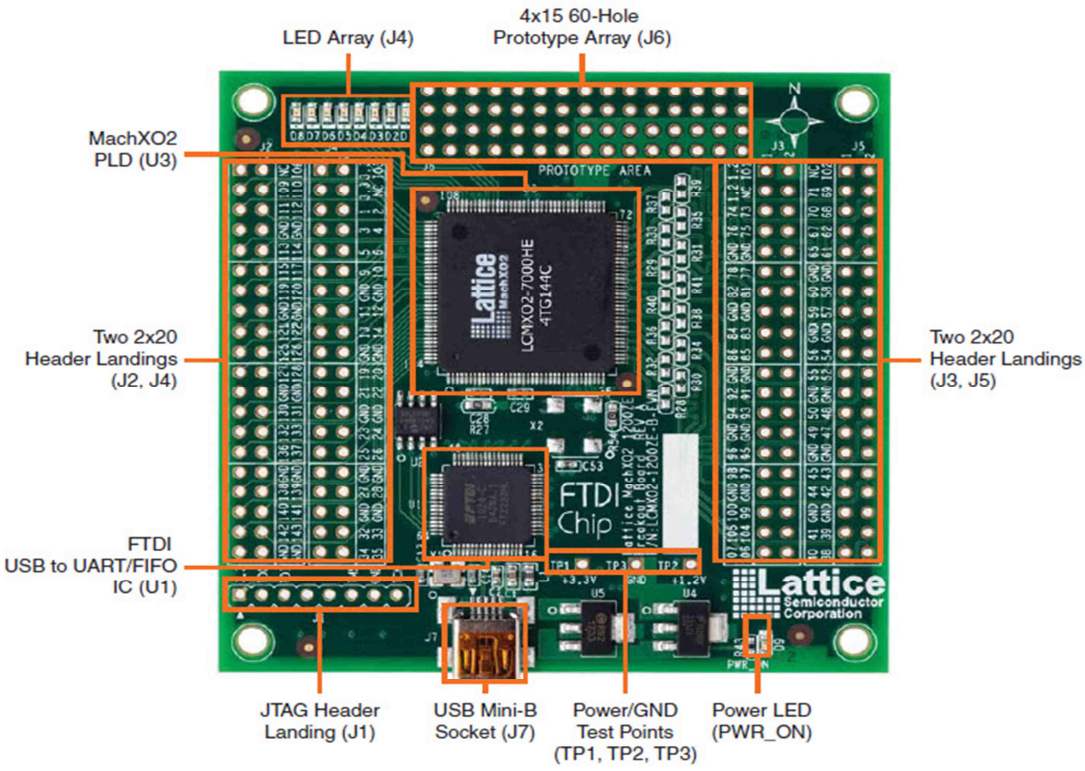


Figura 32: Hardware de Breakout Board.

La placa, en su conjunto, estará constituida por dos bloques adicionales:

1. **Placa PCB Display**: Esta PCB contiene dos dispositivos Display de siete segmentos, así como 16 resistencias que limitan la intensidad que le llega a sus entradas.

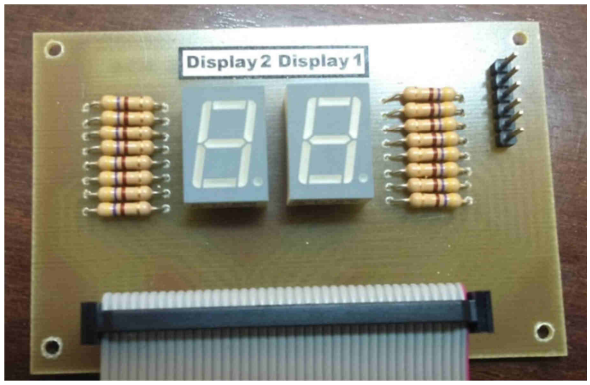


Figura 33: Fotografía de la placa PCB display.

En esta placa tendremos un modelo de conexión en anodo común de todos los diodos que forman el display. Su estructura es tal que:

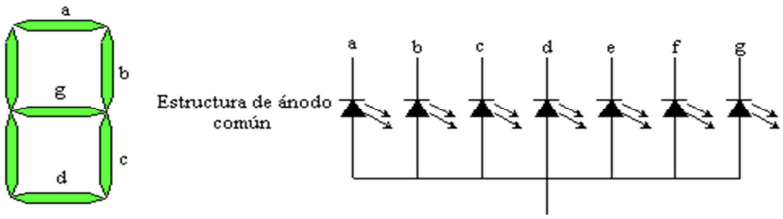


Figura 34: Esquema de la organización del display.

Cada pin de entrada del Display de siete segmentos está conectado a una señal de salida del dispositivo FPGA, siendo sus conexiones:

Segmentos	Salidas del FPGA	
	Display 1	Display 2
A	21	23
B	22	25
C	11	33
D	10	34
E	13	35
F	12	26
G	14	24
Dp	9	32

2. **Placa PCB Swich y Pulsadores:** Esta PCB contiene ocho micro-interruptores y ocho pulsadores para poder aplicar opciones de control de una manera simple al dispositivo FPGA.

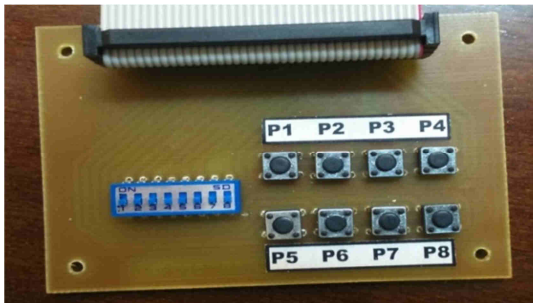


Figura 35: Fotografía PCB Swich y Pulsadores.

Al poner uno de esos micro-interruptores a una posición de ON o presionar un pulsador, el estado de lógica de la entrada correspondiente a esa FPGA variará su nivel, poniéndose a un nivel alto (3.3V). Si se pone a una posición OFF o se suelta el pulsador, este proporcionará un estado de la entrada del FPGA de nivel bajo (0V).

Cada pin de entrada de los micro-interruptores y micro-pulsadores está conectado a una entrada del dispositivo FPGA, siendo sus conexiones:

Interruptores	Entrada FPGA	Pulsadores	Entrada FPGA
1	86	1	76
2	85	2	75
3	84	3	74
4	83	4	73
5	82	5	96
6	81	6	95
7	78	7	94
8	77	8	93

Por tanto, el hardware completo del proyecto será:

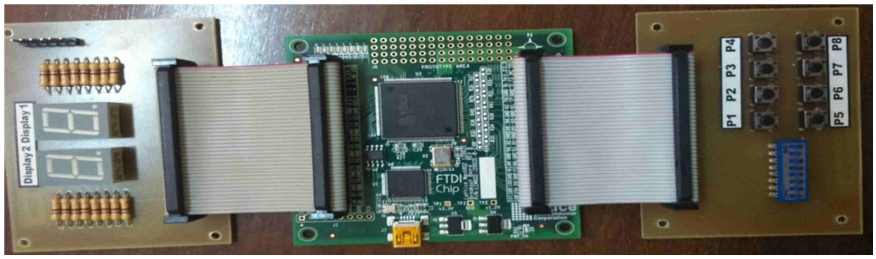


Figura 36: Fotografía de la placa general.

4.5 Características del software de diseño electrónico:

El programa utilizado para el diseño electrónico es el software que ofrece la empresa Lattice Semiconductor para programar sus dispositivos. Este programa es el Lattice Diamond versión 3.2, que se puede descargar de manera gratuita en la página oficial del fabricante. Además, en la página web en la sección de productos existe un manual a disposición del usuario, llamado "Lattice Diamond User Guide", en el cual explica detalladamente el funcionamiento de dicho programa con ejemplos y tutoriales.

Lattice Diamond ofrece un entorno de desarrollo basado en proyectos, incorporando además herramientas de diseño, simulación y síntesis. Permite trabajar de manera estructurada y jerárquica.

Una característica interesante de este programa es que se puede usar para realizar técnicas de diseño mixto, en el que una parte de las descripciones estará realizada por bloques que definiremos previamente por VHDL.

A continuación se realizará un análisis más detallado de las principales herramienta que disponemos en este software, las cuales usaremos en este proyecto. También se incorporan las páginas del manual de usuario del programa donde se puede ampliar esta información.

4.5.1 Spreadsheet View:

Proporciona una interfaz desde la que visualizar y modificar distintas restricciones de diseño. En el caso del proyecto, se ha utilizado esta herramienta para la asignación de los puertos de la placa, pudiendo modificar los puertos de dichas conexiones y sus topologías. [Lattice Diamond User Guide]

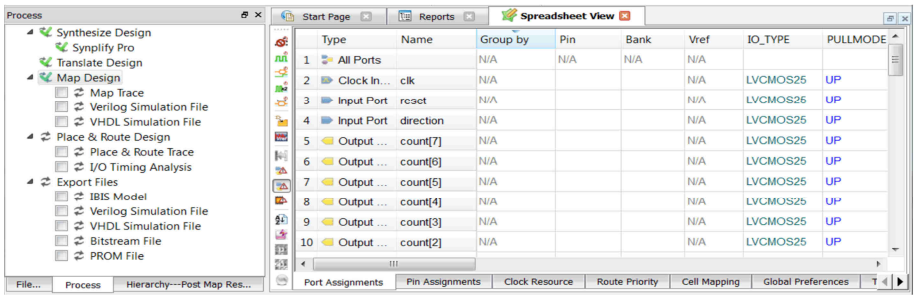


Figura 37: Captura de la herramienta Spreadshett View.

4.5.2 IP_Express:

Esta herramienta recoge una amplia colección de modelos funcionales que pueden ser utilizados para generar bloques personalizados Verilog o VHDL, listos para incluir en nuestro diseño. Están optimizados para que la arquitectura de los dispositivos de la marca Lattice mejore su rendimiento y su velocidad en el circuito. Una vez generado el bloque, se añade al diseño y estaría listo para poder trabajar con él. [Lattice Diamond User Guide].

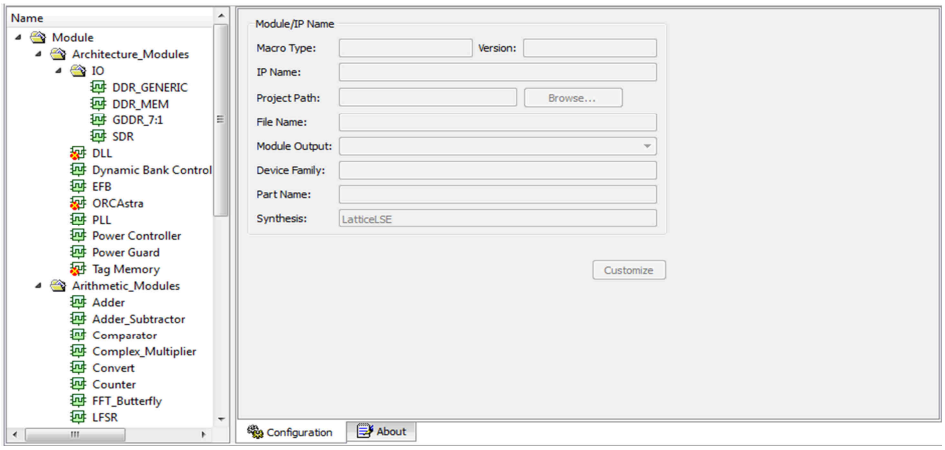


Figura 38: Captura de la herramienta IP_Express.

4.5.3 Programmer:

Una vez finalizado el diseño y generado el fichero JEDEC, esta herramienta permite transferir la información del proyecto al dispositivo de Lattice que vamos a utilizar. [Lattice Diamond User Guide].

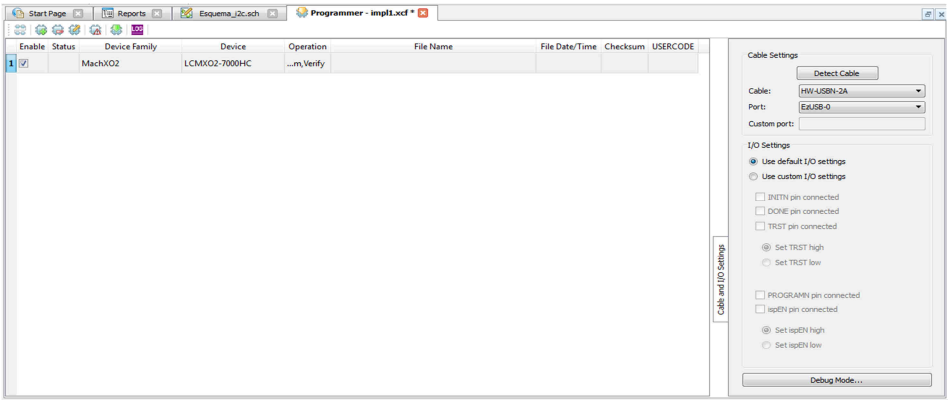


Figura 39: Captura de la herramienta Programmer.

4.5.4 Active-HDL Lattice edition:

Esta herramienta no pertenece al entorno de desarrollo del proyecto como tal, sino que se trata de un simulador que permite comprobar el correcto funcionamiento de cada módulo por separado o al conjunto del diseño completo. [Lattice Diamond User Guide].

Figura 40: Captura de la herramienta Active-HDL.

4.5.5 Editor de VHDL:

Esta herramienta permite definir el código VHDL que se aplicará para dicho proyecto, pudiendo más adelante simular el comportamiento descrito, o bien crear bloques propios para implementarlos en los dispositivos. [Lattice Diamond User Guide].

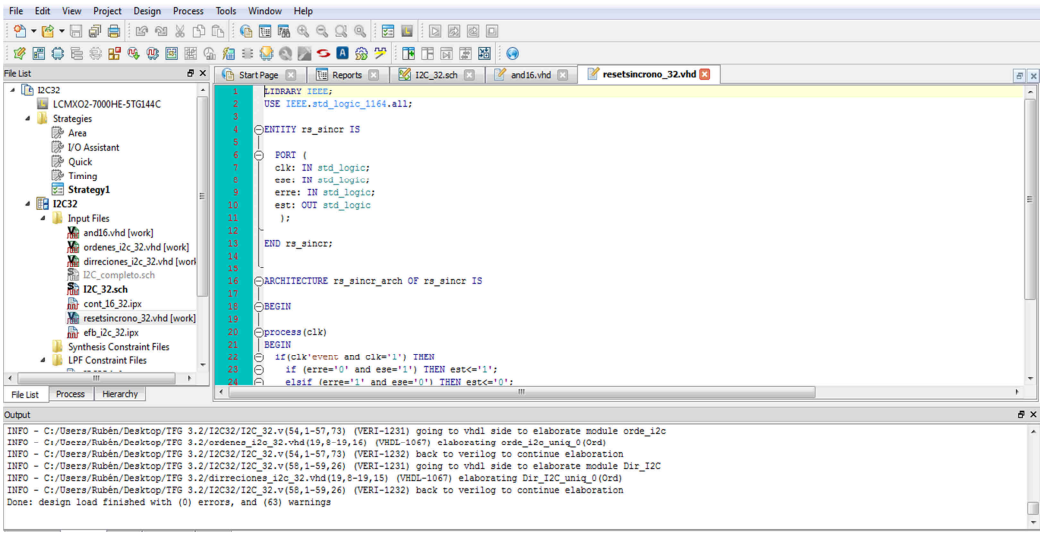


Figura 41: Captura de la herramienta del editor VHDL.

4.5.6 Editor de esquemas:

Esta herramienta permite establecer de manera gráfica las conexiones realizada en los bloques que queremos implementar dentro del dispositivo a programar. [Lattice Diamond User Guide].

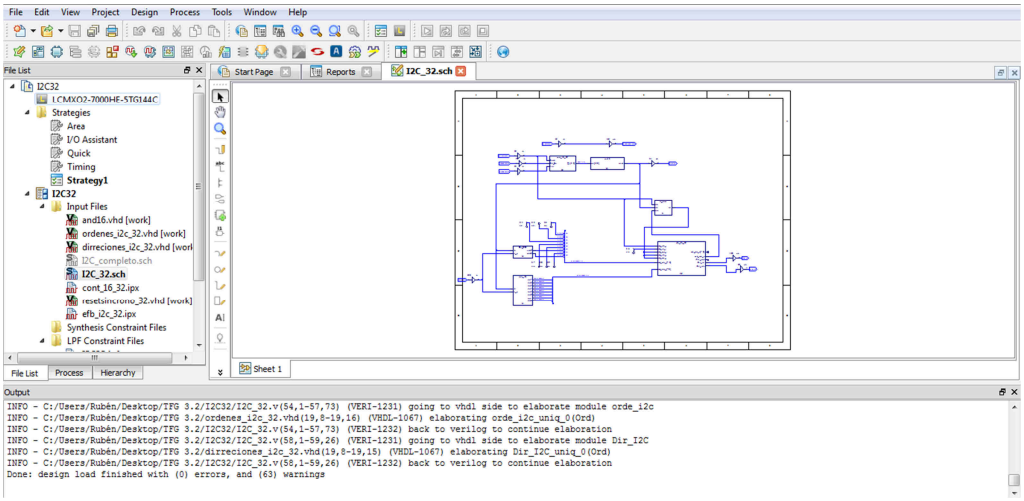


Figura 42: Captura de la herramienta de editor de esquemas.

4.6 Códec de audio:

Un códec (abreviación de la palabra codificador/decodificador) es cualquier dispositivo, incluyendo el uso de un módulo de software, que realiza la función de codificación y/o decodificación de los datos recibidos por un periférico de entrada/salida.

Para la realización de la codificación y decodificación de los datos de audio en este proyecto, usaremos el periférico proporcionado por la compañía MikroElektronika, llamado Audio Codec Proto, que contiene el circuito integrado WOLFSON. Este driver es el encargado de obtener la señal de audio de entrada y de proporcionar la señal de salida.

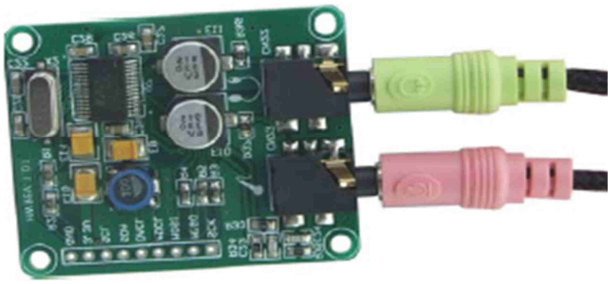


Figura 43: Fotografía del códec de audio de MikroElektronika.

La interfaz de comunicación del códec está configurada para ajustarse a las especificaciones de los protocolos I2C y SPI.

El protocolo I2C proporciona los datos de inicialización y modificación de las características de funcionamiento del códec online, como valor de registros o el volumen del audio, mandados desde el usuario a través del bus WISHBONE primario, para que la reconfiguración o el control no requieran operaciones adicionales. Esto solo significa que los usuarios deben cumplir con los formatos de datos, volumen, silenciamiento y otros ajustes que tienen sido preseleccionados para ellos. Se podrán comprobar que la placa del códec contiene las patillas SCL y SDA propias del protocolo I2C.

El protocolo SPI se usará cuando los datos sean transformados de analógico-digital o viceversa, ya que será el encargado de transportar una gran cantidad de información entre el códec y el FPGA. Se podrá comprobar que la placa del códec contiene las patillas SCK, MISO y MOSI propias del protocolo SPI.

Además, ambos protocolos de comunicación de información necesitan una patilla de puesta a tierra o masa, que también estará disponible dentro de la propia tarjeta del códec.

Conexión:

La placa adicional está conectada a un dispositivo FPGA a través de conectores CN2. Un conector de 3,5 mm CN23 que se utiliza para conectar un micrófono o una entrada de audio digital, mientras que el conector CN22 se utiliza para conectar los auriculares o altavoces.

Uso:

El circuito del códec se utiliza para convertir la entrada de audio analógica en una señal de audio digital para su procesamiento, y una vez procesada dicha señal se transforma a una señal analógica para que se reproduzca a través de auriculares o altavoces. Esta también se utiliza para convertir la señal de audio analógica del micrófono en una señal de audio digital.

La transferencia de datos entre el dispositivo FPGA y placa adicional se realiza por el Interfaz Periférico Serial (SPI), mientras que la inicialización y el control de las funciones son controlados por el FPGA a través de la comunicación I2C.

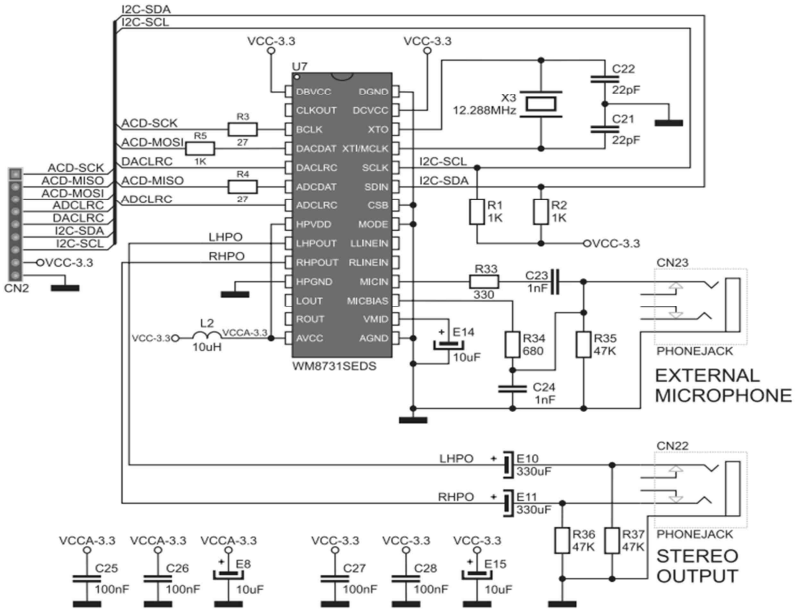


Figura 44: Esquema del circuito integrado WOLFSON del códec de audio.

Modos de funcionamiento:

El esquema completo del códec es:

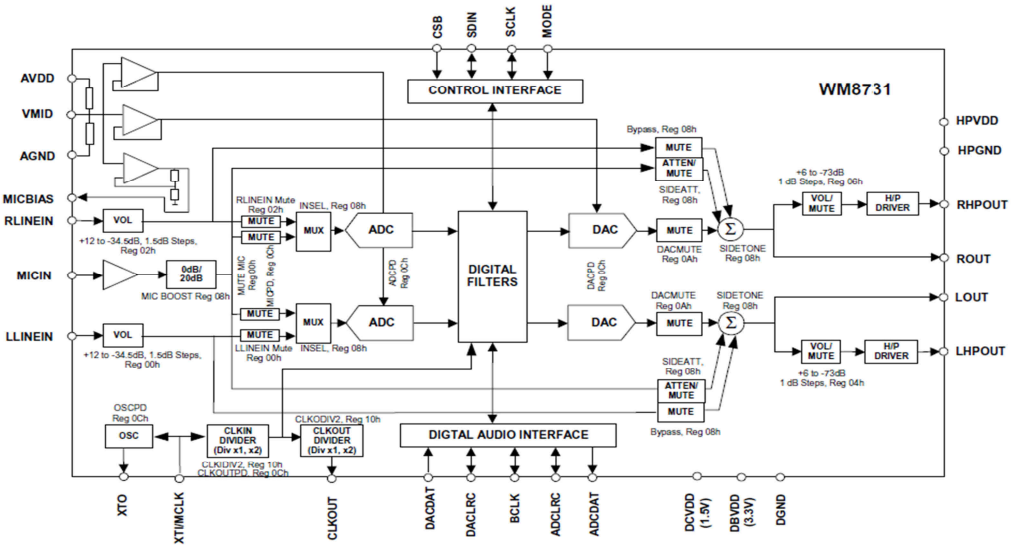


Figura 45: Esquema de las I/O del circuito integrado WOLFSON.

Dependiendo de las configuraciones adoptadas en su esquema, el funcionamiento del códec Wolfson WM8731, puede tomar las siguientes entradas y salidas:

- **LINEIN:** Se puede variar las líneas de entrada configurando su impedancia, de tal modo que se pueda regular la línea derecha o izquierda para que el sonido sea mayor o menor. Con esto también se pueden añadir filtros activos o pasivos para controlar las frecuencias de entrada.
- **MICIN:** Se pueden variar las configuraciones del micrófono, pudiéndose ajustar el volumen e incluir una función de silencio. También se pueden incluir filtros para regular las frecuencias de entrada.
- **MICBIAS:** Esta salida aporta un voltaje de referencia con bajo nivel de ruido, utilizado principalmente para la polarización del micrófono.
- **LLINEOUT y RLINEOUT:** Este códec proporciona dos salidas de baja impedancia, adecuado para manejar cargas de línea normales. Estas no son ajustables con niveles analógicos.
- **HPOUT:** La entrada analógica se envía directamente a la salida analógica, normalmente con la configuración SIDESTONE, amplificándola para ser usada por altavoces.

Capítulo 5. Implementación.

5.1 Implementación del bypass.

5.1.1 Desarrollo del hardware.

Placa PCB del Códec:

Una vez realizada la configuración para inicializar el dispositivo FPGA, se procede a realizar la placa PCB para el códec. Esta PCB se utilizará para dar soporte y transportar la información de la fuente analógica a través de una entrada de audio al códec. Más tarde, trasportará la salida analógica después del procesamiento digital, según sea la configuración que le demos al dispositivo FPGA.

Esta configuración está recogida en el datasheet del códec WOLFSON, en la sección de signal audio path, y es necesario fabricarla ya que el códec del que disponemos solo presenta una configuración para conectar directamente un micrófono, sin embargo nosotros vamos a introducir el audio directamente a partir de una fuente analógica, eliminando los niveles de continua de la señal. Además, la salida del códec sirve para el uso de cascos, y queremos enviarlo a unos altavoces con una impedancia diferente, por lo que tendremos que modificarlo debido a que posee diferente impedancia de salida.

- El circuito de entrada para esta aplicación es:

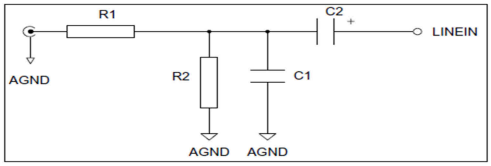


Figura 46: Esquema del circuito de entrada.

Según el datasheet del códec WOLFSON, los valores de los componentes son:

- Resistencias: 5600 Ω.
- Condensador 1: 220 pF.
- Condensador 2: 1 µF.

- El circuito de salida para esta aplicación es:

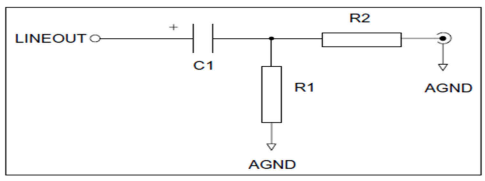


Figura 47: Esquema del circuito de salida.

Según el datasheet del códec WOLFSON, los valores de los componentes son:

- Resistencia 1: 47000 Ω.
- Resistencia 2: 100 Ω.
- Condensador: 10 µF.

Por tanto, una vez terminada la PCB, se soldarán los componentes descritos en los esquemas del datasheet del códec y se añadirán los pines para conectarlo a la placa del códec, quedando:

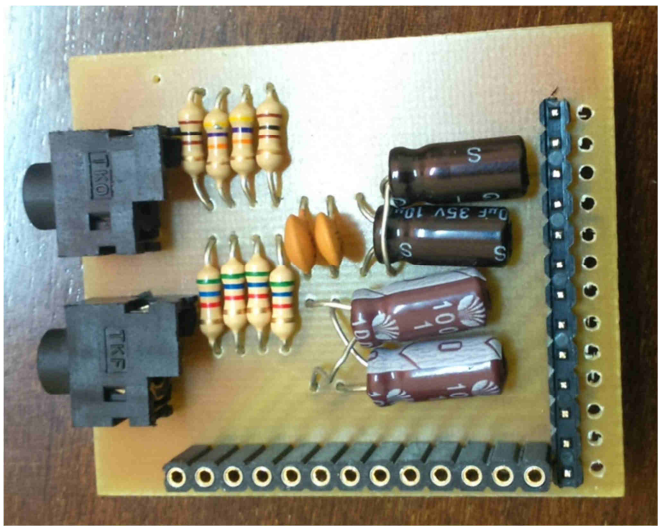


Figura 48: Fotografía de la placa de conexiones del códec.

Los pines de entrada (pines hembra) son los que se conectarán a las entradas de la placa del códec para que haga el procesamiento digital de la señal de audio y realice las funciones propias del mismo.

Los pines de salida (pines machos) son los que se conectarán al dispositivo FPGA, para que puedan enviar y recibir información, ya que estos serán los pines del protocolo I2C y del protocolo SPI, además de la toma a tierra del circuito.

5.1.2 Desarrollo de la configuración:

5.1.2.1 Bloque EFB:

El hardware que se utiliza para trasferir la información y los datos está organizado según el siguiente esquema:

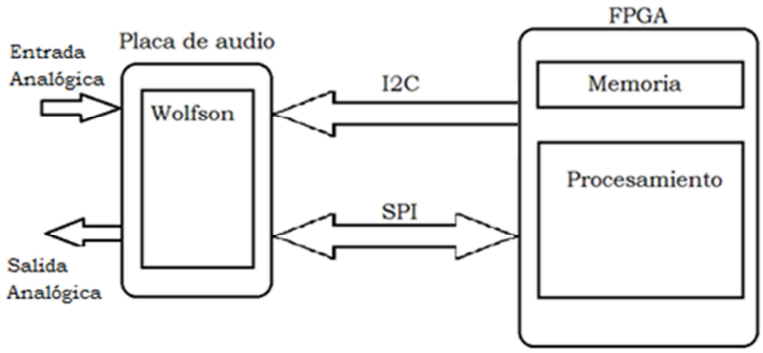


Figura 49: Esquema general de la placa de procesamiento y sus protocolos.

Para transferir la información de control del FPGA al códec de audio, se usará el protocolo de comunicación I2C, explicado anteriormente en el apartado 4.3. Este protocolo se puede implementar con el bloque EFB, localizado en el propio dispositivo FPGA.

Más tarde, también se podría implementar el protocolo SPI con un bloque EFB, usado para enviar la información digital del audio procesado desde el dispositivo FPGA al códec de audio, para convertirlo a analógico y ser escuchado por los altavoces. Pero para este protocolo SPI se utilizará un protocolo creado exclusivamente para esta aplicación, como se explicará más adelante en el apartado 5.2 implementación del bypass digital.

Para crear un módulo EFB con las funciones del protocolo I2C primario en el software de Lattice Diamond, se deberán seguir los siguientes pasos:

1º Seleccionamos la opción IP_Express del panel de configuraciones de Lattice Diamond. Esta opción está destinada para crear bloques predefinidos rápidamente con configuraciones básicas.

Figura 50: Captura de la selección de la opción IP_Express.

2º Se selecciona la opción de bloque EFB, y saldrá un panel de configuración tal que:

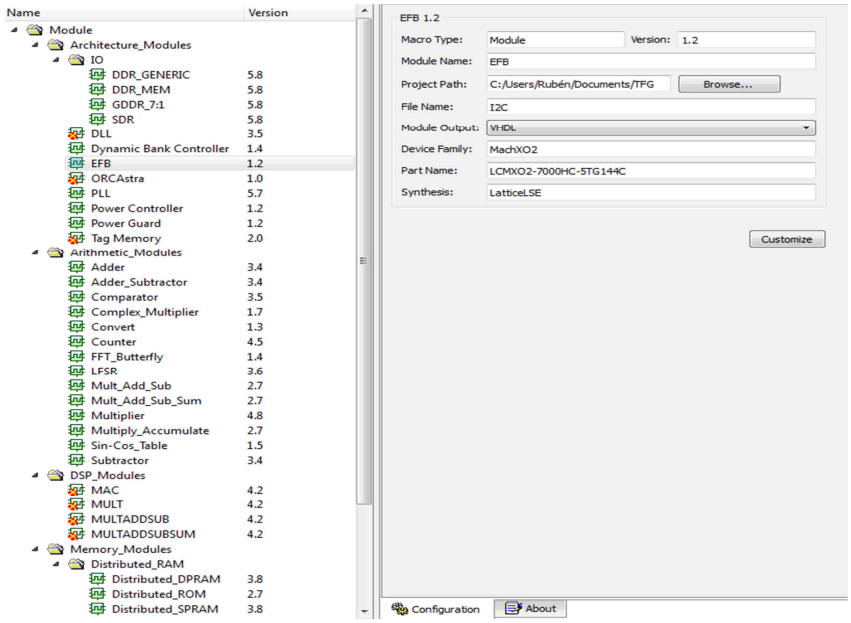


Figura 51: Selección del bloque EFB dentro de la opción IP_Express.

3º Se introducen las opciones y configuraciones necesarias y pulsamos en la opción Customize, obteniendo el siguiente cuadro de configuración :

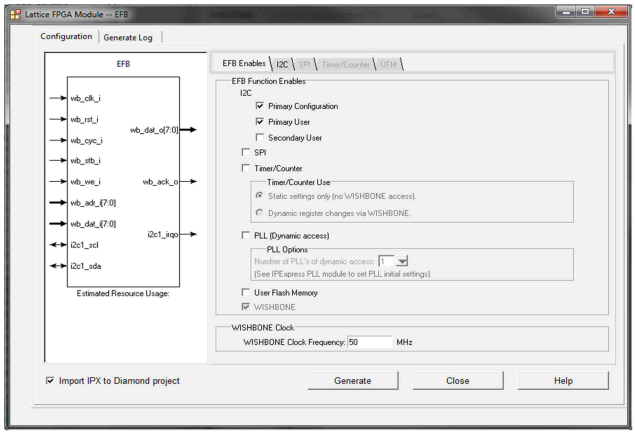


Figura 52: Selección de las configuraciones del bloque EFB.

4º Se introducen de nuevo las configuraciones correctas para el protocolo I2C primario y seleccionamos la opción de Generate.

El módulo EFB tiene un mapa del registro para el servicio de las funciones ensambladas a través de la interfaz del bus WISHBONE de operaciones de lectura y escritura. Cada función ensamblada dedica 8 bits de datos y registros de control, con excepción de la memoria Flash.

A estas funciones se puede acceder a través del mismo conjunto de registros internos, que serán:

Address (Hex)	Hardened Function
0x00-0x1F	PLL0 Dynamic Access1
0x20-0x3F	PLL1 Dynamic Access1
0x40-0x49	I²C Primary
0x4A-0x53	I²C Secondary
0x54-0x5D	SPI
0x5E-0x6F	Timer/Counter
0x70-0x75	Flash Memory (UFM/Configuration)
0x76-0x77	EFB Interrupt Source

Figura 53: Registros del bus WISHBONE.

Al crear instancias en el protocolo I2C para las operaciones de esclavos, la entrada del EFB 'wb_clk_i' deberá estar conectada a una fuente de reloj válida de al menos 7.5 la velocidad del bus I2C (por ejemplo, si la frecuencia del CLK es de 7.5 MHz, la frecuencia mínima de trabajo del I2C será de 100 kHz).

5.1.2.2 Registros I2C:

Ambos núcleos I2C se comunican con el interfaz WISHBONE EFB a través de un conjunto de controles, comandos, estados y/o registros de datos. Esta tabla muestra los nombres de los registros internos y sus funciones.

Estos registros son un subconjunto del mapa de registros EFB:

I²C Primary Register Name	I²C Secondary Register Name	Register Function	Address I²C Primary	Address I²C Secondary	Access
I2C_1_CR	I2C_2_CR	Control	0x40	0x4A	Read/Write
I2C_1_CMDR	I2C_2_CMDR	Command	0x41	0x4B	Read/Write
I2C_1_BR0	I2C_2_BR0	Clock Pre-scale	0x42	0x4C	Read/Write
I2C_1_BR1	I2C_2_BR1	Clock Pre-scale	0x43	0x4D	Read/Write
I2C_1_TXDR	I2C_2_TXDR	Transmit Data	0x44	0x4E	Write
I2C_1_SR	I2C_2_SR	Status	0x45	0x4F	Read
I2C_1_GCDR	I2C_2_GCDR	General Call	0x46	0x50	Read
I2C_1_RXDR	I2C_2_RXDR	Receive Data	0x47	0x51	Read
I2C_1_IRQ	I2C_2_IRQ	IRQ	0x48	0x52	Read/Write
I2C_1_IRQEN	I2C_2_IRQEN	IRQ Enable	0x49	0x53	Read/Write

Note: Unless otherwise specified, all reserved bits in writable registers shall be written '0'.

Figura 54: Registros del protocolo I2C en el bus Wishbone.

Las operaciones con los diferentes registros se encuentran en el ANEXO I: registros internos del Wishbone.

5.1.2.3 Circuitería necesaria para inicializar el I2C:

La escritura en este bus Wishbone se desarrollará con pulsos de periodo de 1 milisegundo. Cada pulso es una orden para mandar una serie de direcciones o datos, que desarrolla una acción. El periodo de 1 milisegundo se establece por seguridad, para que los cambios no se realicen con una rapidez mayor de la que pueda soportar el bus Wishbone para transferir esta información.

El ciclo de escritura del bus Wishbone es el siguiente:

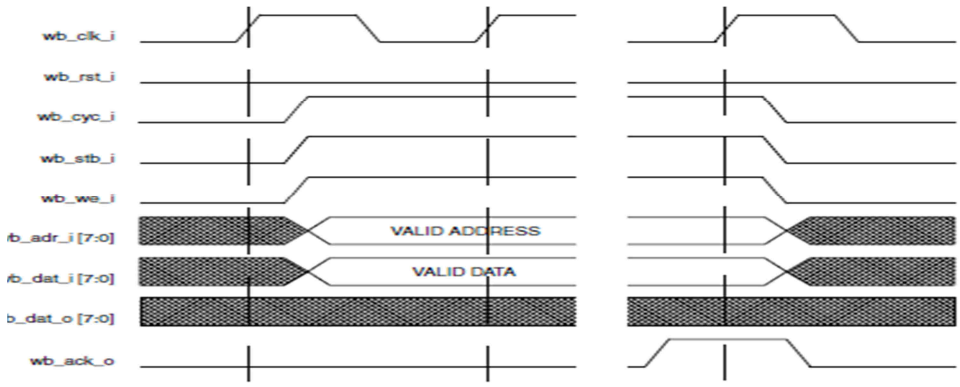


Figura 55: Ciclo de escritura del bus WISHBONE.

En la placa breakout board disponemos de un oscilador de frecuencia 50 Megahercios. Se puede generar un oscilador de frecuencia de 1KHz (periodo de 1 milisegundo) con un contador de 16 bits y una puerta AND de 16 entradas. Con ello reduciríamos la frecuencia de oscilación en aproximadamente 65000 veces, siendo esta frecuencia apropiada para generar las ordenes de operaciones.

Con un pulso de 20ns se podría realizar el diseño, pero para mejorar la seguridad de que estas órdenes serán leídas, y puesto que no hay problemas de tiempo disponible, se cuadriplicará dicho pulso para asegurar su correcta interpretación.

Por ello, en la puerta AND se concretará que cuando del bit 15 al bit 2 estén a uno, se genere un pulso que dure hasta que los 16 bits de la puerta AND sean 1 y el contador se reinicie, siendo el periodo del pulso 1 KHz y su duración 4x50MHz, es decir, 80 nanosegundos.

Por ello, necesitaremos un pulso tal que:

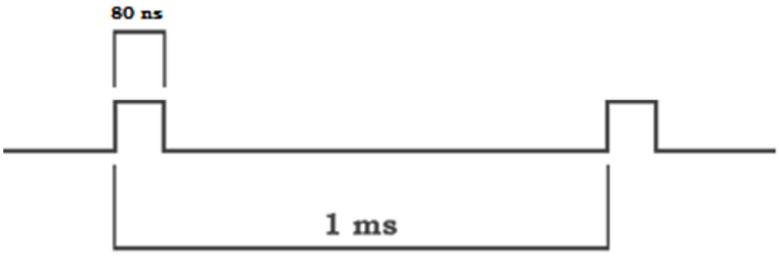


Figura 56: Pulso de 80 ns y periodo 1 ms.

Este pulso se generará con el oscilador de 50 MHz previamente mencionado, al cual estará conectado un bloque contador de 16 bits que podemos encontrar dentro del dispositivo FPGA en la opción de IP_Express, y un bloque AND de 16 entradas que generaremos específicamente para esta tarea en lenguaje VHDL, generando posteriormente su esquema para poder trabajar con él en el editor de esquemas.

Para crear el contador de 16 bits se hará con la opción IP_Express. Se selecciona esta opción del cuadro de comandos, y nos saldrá el siguiente cuadro de configuraciones:

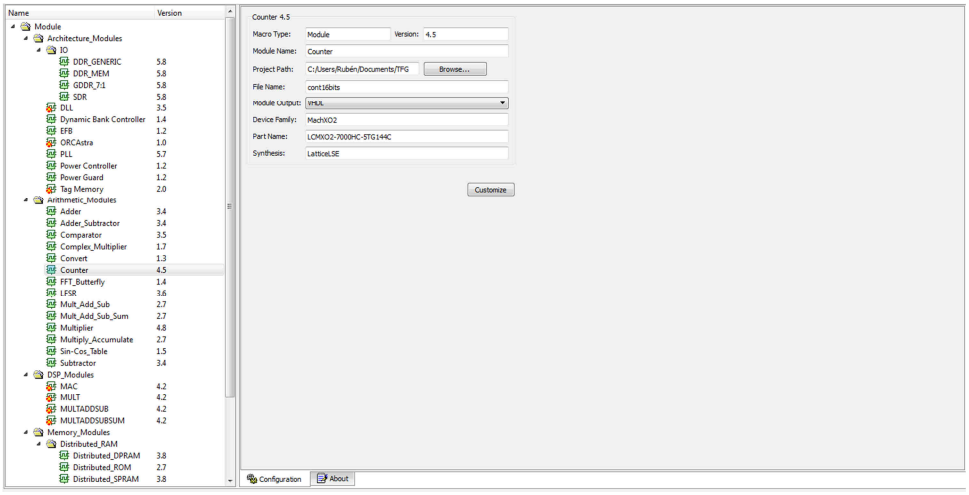


Figura 57: Captura del dispositivo contador de la opción IP_Express.

Seleccionamos la opción de “counter” y se introducen los datos correctos de funcionamiento. Una vez terminado de introducir los datos, se selecciona la opción “Customice”, obteniendo las siguientes opciones de configuraciones:

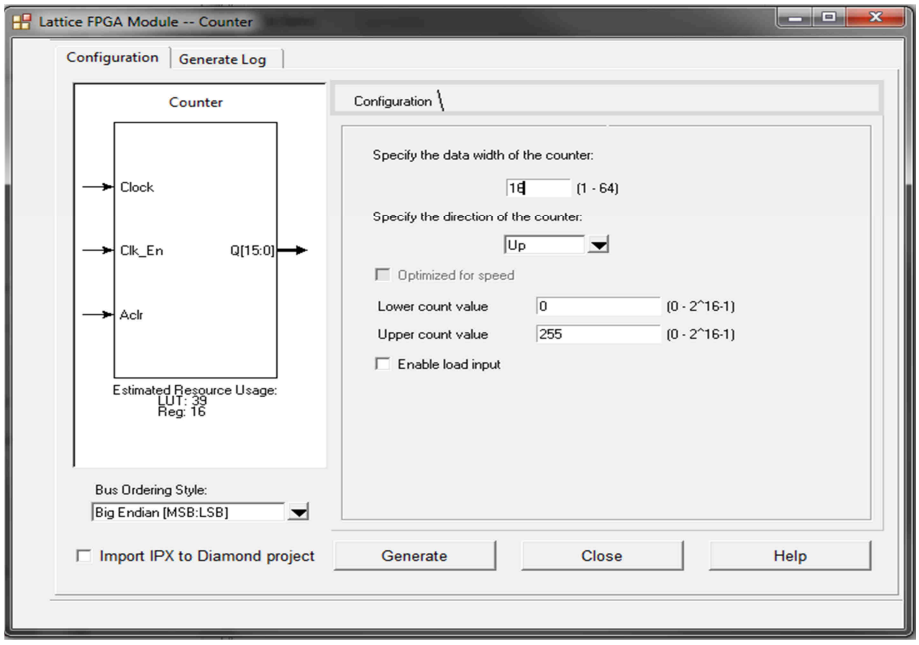


Figura 58: Captura de la configuración del contador del IP_Express.

Una vez terminados de introducir los valores correctos, se selecciona "Generate" y aparecerá un contador de 16 bits propio del FPGA.

Para la puerta AND de 16 bits, el código en VHDL del bloque es:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY and16 IS

  PORT (ent: IN std_logic_vector(15 downto 0);
        salida: OUT std_logic);

END and16;

ARCHITECTURE and16_arch OF and16 IS

begin
salida<=ent(15)and ent(14)and ent(13)and ent(12)and ent(11)and ent(10)and ent(9)and ent(8)and ent(7)and ent(6)and ent(5)and ent(4)and ent(3)and ent(2);

end and16_arch;

```

Figura 59: Captura del código VHDL del bloque AND de 16 bits.

Para generar el bloque funcional, se guarda el código VHDL y le damos a crear símbolo presionando botón derecho en las opciones de la izquierda, donde aparecen todos los ficheros de VHDL.

Conectando todos los bloques correctamente en el editor de esquemas del software, el esquema completo de la señal de periodo 1 ms quedará:

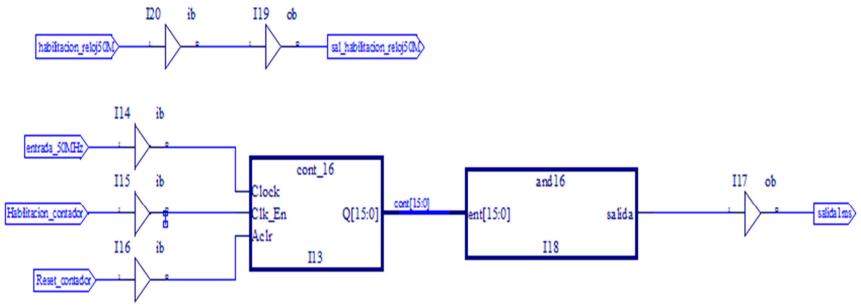


Figura 60: Esquema del generador de pulsos de periodo 1 ms.

Más tarde, se generará un reset síncrono, el cual se habilitará con cada pulso de 1 ms, poniéndose su salida en valor alto. Este se conecta a las entradas del bloque Wishbone cyc-i, stb-i y we-i, que es necesario que estén a nivel alto para comenzar a leer el ciclo de escritura del bus Wishbone.

Una vez traspasado el byte de información, el dispositivo Wishbone esclavo mandará un bit ACK o bit de confirmación. Este bit cambiará la salida del reset a nivel bajo, inhabilitando así la transferencia de datos por el Wishbone, hasta que este no vuelva a recibir otra señal del bloque de frecuencia de 1KHz para comenzar con una nueva transferencia.

El código VHDL del Reset síncrono es:

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3
4  ENTITY rs_sincr IS
5
6    PORT (
7    clk: IN std_logic;
8    ese: IN std_logic;
9    erre: IN std_logic;
10   est: OUT std_logic
11    );
12
13 END rs_sincr;
14
15
16 ARCHITECTURE rs_sincr_arch OF rs_sincr IS
17
18 BEGIN
19
20 process(clk)
21 BEGIN
22   if(clk'event and clk='1') THEN
23     if (erre='0' and ese='1') THEN est<='1';
24     elsif (erre='1' and ese='0') THEN est<='0';
25     elsif (erre='1' and ese='1') THEN est<='0';
26
27     end if;
28   end if;
29   end process;
30
31 END rs_sincr_arch;
32
```

Figura 61: Captura del código VHDL del reset síncrono.

El esquema total del circuito de habilitación del Wishbone quedará:

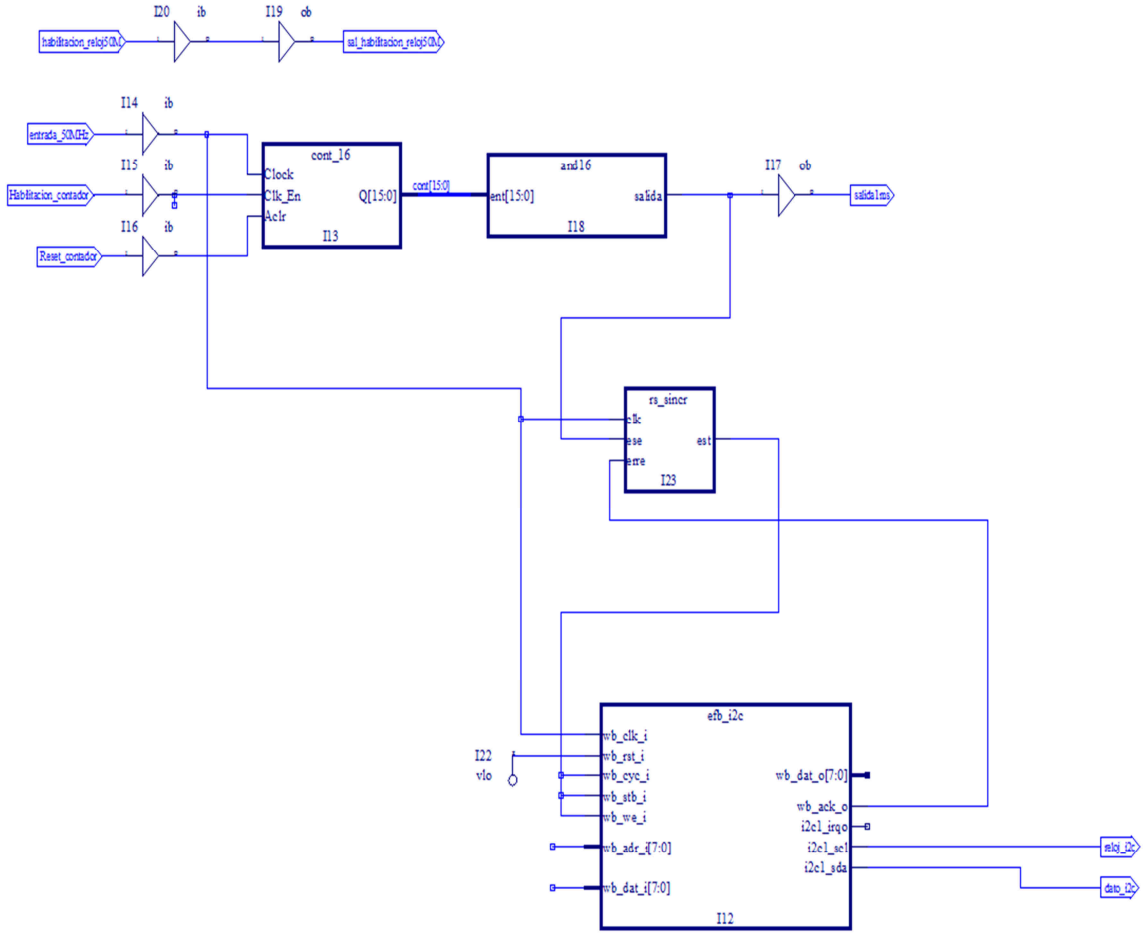


Figura 62: Captura del esquema de pulsos de activación del bypass.

5.1.2.4 Transferencia del I2C:

Se irán dando sucesivas órdenes en el I2C para su inicialización. Lo primero que se deberá programar será la habilitación del protocolo I2C, para después poder configurar el códec. Por último, una vez enviada toda la información, se inhabilitará el protocolo I2C.

1. La primera escritura que se debe realizar es la de habilitación del protocolo I2C, usando para ello el bloque EFB. Esto se realizará con la dirección de registros internos del bus Wishbone referidos al I2C, especificados en el catálogo "User Flash Memory and Hardener Control Functions in MachX02 Devices Reference Guide", especificados en el Anexo I.
   El registro interno al que se debe mandar esta orden es al registro 0x40 en el maestro, que es el registro I2C_1_CR. La orden de información a mandar será 1000 || 0000. Esta orden habilita el protocolo I2C y establece un delay o retardo de 300ns, propio de dicho protocolo.

2. La segunda orden es la de enviar la información al códec de audio. Esto se realizará en tres secuencias, ya que la información necesaria para cada modificación del códec consta de 24 bits, y por el bus Wishbone solo se pueden enviar tramas de 1 byte (8 bits).
   Para ello, se divide la información en tres tramas de 8 bits, y se enviará primero al registro interno 0x44 en el maestro, cuyo registro es I2C_1_TXDR. Una vez guardado el byte en el registro 0x44, que actúa como almacenamiento de información, se mandará la orden de enviarlo al Wishbone esclavo para que realice la orden establecida. Para ello se envía la orden al registro interno 0x41, cuya función es enviar la información guardada en el registro 0x44 I2C_1_TXDR al códec de audio.
   Las divisiones de la cadena de información se realizarán tantas veces como requiera la misma, realizando en cada separación las tareas de almacenamiento y envío.

3. Cuando no hay más bytes que enviar, se vuelve al registro interno 0x40, que es el registro de control del I2C, y se escribe la orden 0000||0000 para inhabilitar de nuevo el protocolo I2C.

En conclusión, primero es necesario definir la dirección del dispositivo al que lo queremos mandar, después se debe mandar la dirección del registro interno que queremos modificar, y por último las nuevas configuraciones del registro que queremos modificar.

Esto se puede establecer con dos máquinas de estados finitos o FSM, que con un mismo reloj sincronizadas, vayan poniendo con una FSM la dirección del protocolo I2C, en formato de 1 byte, en el bus de direcciones del Wishbone wb_addr [7:0], y con la otra FSM la orden que queremos enviar, con formato de 1 byte, en la entrada Wishbone wb_dat [7:0].

El ciclo de escritura en el protocolo I2C es el siguiente:

Figura 63: Ciclo de escritura del protocolo I2C.

Inicialmente, se implementará la configuración de bypass en el códec Wolfson, por el que el audio analógico llegará al códec, y sin convertirse a formato digital pasará directamente a la salida de audio sin realizar modificaciones en los datos. Por tanto, los datos solo estarán en forma analógica, sin usar los filtros digitales o su procesamiento digital.

Mediante los diferentes valores de los registros internos del dispositivo Wolfson, obtenemos sus diferentes configuraciones y controles especificados en el manual para la configuración deseada.

Los valores de los registros internos del Wolfson vienen determinados por defecto para el bypass, pero nosotros vamos a resetear primero el dispositivo. Más tarde, se cambiará el control de consumo de los dispositivos del códec, ya que en este caso un consumo mínimo del dispositivo no es importante y por defecto estaría en modo bajo consumo (power down mode).

La configuración de los registros internos del códec, por defecto, quedará con los siguientes valores, obtenidos de las tablas del ANEXO II:

Acción	Registro interno	Valor
Línea izquierda	0000000	0||10010111
Línea derecha	0000001	0||10010111
Salida altavoz izquierda	0000010	0||01111001
Salida altavoz derecha	0000011	0||01111001
Control analógico	0000100	0||00001010
Control digital	0000101	0||00001000
Control de energía	0000110	0||00000000
Formato digital	0000111	0||10011111
Muestreo de control	0001000	0||00000000
Control activo	0001001	0||00000000
Borrado de registros	0001111	0||00000000

Solo se mandarán las órdenes en los casos que los registros sean configurados para realizar la función especificada. En este primer caso, como lo único que queremos es poner el modo bypass, resetearemos la información inicial y más tarde modificaremos el control de la energía, poniendo a estado de consumo alto a todos sus elementos. Cada información se envía en tres tramas de 1 byte cada una. La primera trama incluirá la dirección del dispositivo al que queremos acceder, la segunda trama contendrá el registro interno del dispositivo seleccionado (en este caso el códec) más el bit más significativo (MSB) del valor de los nuevos datos del registro a modificar. En la tercera trama incluye el resto de datos de los nuevos valores de los registros.

La dirección del códec viene determinada por el estado de la señal CSB del códec, que normalmente es 0011010+ R/not W. En este caso, como lo que queremos es escribir, se pondrá el último bit 0. Por tanto, la trama quedará 00110100. La señal CSB sirve para establecer la dirección cuando existen varios dispositivos, pero si solo existe un dispositivo por defecto tiene nivel de tensión bajo. Si tuviera nivel alto, la dirección sería 0011011+ R/not W.

Al principio de cada información a enviar, en la primera trama debemos enviar al registro 0x44 un bit de start y en la última trama un bit de stop. Esto indicará cuando iniciamos y terminamos la transmisión de un dato entero.

Estas máquinas se harán con el software libre Qsfm, especializado en crear e implementar, en lenguaje VHDL, la descripción de las máquinas de estado finitos de una manera simple y eficaz, creados en un editor gráfico. El software ha sido proporcionado por el Departamento de Tecnología Electrónica. Una vez creadas las FSM, se pasarán al software de diseño electrónico para poder simularlo y comprobar que su funcionamiento es el correcto.

Comprobado su funcionamiento, se creará el esquema completo con el editor de esquemas del bus Wishbone, que transmitirá la información del protocolo I2C al dispositivo del códec de audio.

Por lo tanto, resumiendo las tareas necesarias para implementar la función bypass en la plataforma, obtendríamos la siguiente programación de registros internos:

Descripción de la tarea.	Descripción de cada paso.	Dirección del registro.	Datos del registro.
Inicio.	Se inicializa el protocolo I2C.	0x40	1000 0000
Habilitar I2C.	Se habilita el protocolo I2C.	0x40	1000 0000
Configurar el códec de audio.	Poner la dirección del dispositivo en TXDR.	0x44	0011 0100
	Se envía al códec por el WISHBONE con el bit de start.	0x41	1001 0100
	Poner dirección del registro interno + MSB de datos en TXDR.	0x44	0001 111 + 0
	Se envía al códec por el WISHBONE.	0x41	0001 0100
	Poner el nuevo dato del registro en TXDR.	0x44	0000 0000
	Se envía al códec por el WISHBONE con el bit de stop.	0x41	0101 0100
	Poner la dirección del dispositivo en TXDR.	0x44	0011 0100
	Se envía al códec por el WISHBONE con el bit de start.	0x41	1001 0100
	Poner dirección del registro interno + MSB de datos en TXDR.	0x44	0000 110 + 0
	Se envía al códec por el WISHBONE.	0x41	0101 0100
	Poner el nuevo dato del registro en TXDR.	0x44	0000 0000
	Se envía al códec por el WISHBONE con el bit de stop.	0x41	0101 0100
Inhabilitar I2C.	Se deshabilita el protocolo I2C	0x40	0000 0000

Direcciones del registro:

Las secuencias de bits de las direcciones de los registros internos del WISHBONE a los que accedemos son tres, mostradas con mayor detalle en el ANEXO I.

Dirección hexadecimal	Dirección binaria
0x40	0100 || 0000
0x41	0100 || 0001
0x44	0100 || 0100

Por lo tanto, las direcciones de los registros que tendremos que mandar ordenadamente serían:

Orden/Bit.	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Función
0	0	1	0	0	0	0	0	0	Inicializar I2C.
1	0	1	0	0	0	0	0	0	
2	0	1	0	0	0	1	0	0	Reset.
3	0	1	0	0	0	0	0	1	
4	0	1	0	0	0	1	0	0	
5	0	1	0	0	0	0	0	1	
6	0	1	0	0	0	1	0	0	
7	0	1	0	0	0	0	0	1	
8	0	1	0	0	0	1	0	0	Control de consumo.
9	0	1	0	0	0	0	0	1	
10	0	1	0	0	0	1	0	0	
11	0	1	0	0	0	0	0	1	
12	0	1	0	0	0	1	0	0	
13	0	1	0	0	0	0	0	1	
14	0	1	0	0	0	0	0	0	Inhabilitar.

Podemos ver que salvo los bits A2 y A0, todos los demás bit se mantienen a un nivel fijo. Los bits que varían de valor son los que se programarán en la máquina de estados finitos o FSM, e irán cambiado en función a la variación de un reloj de periodo 1 ms creado anteriormente.

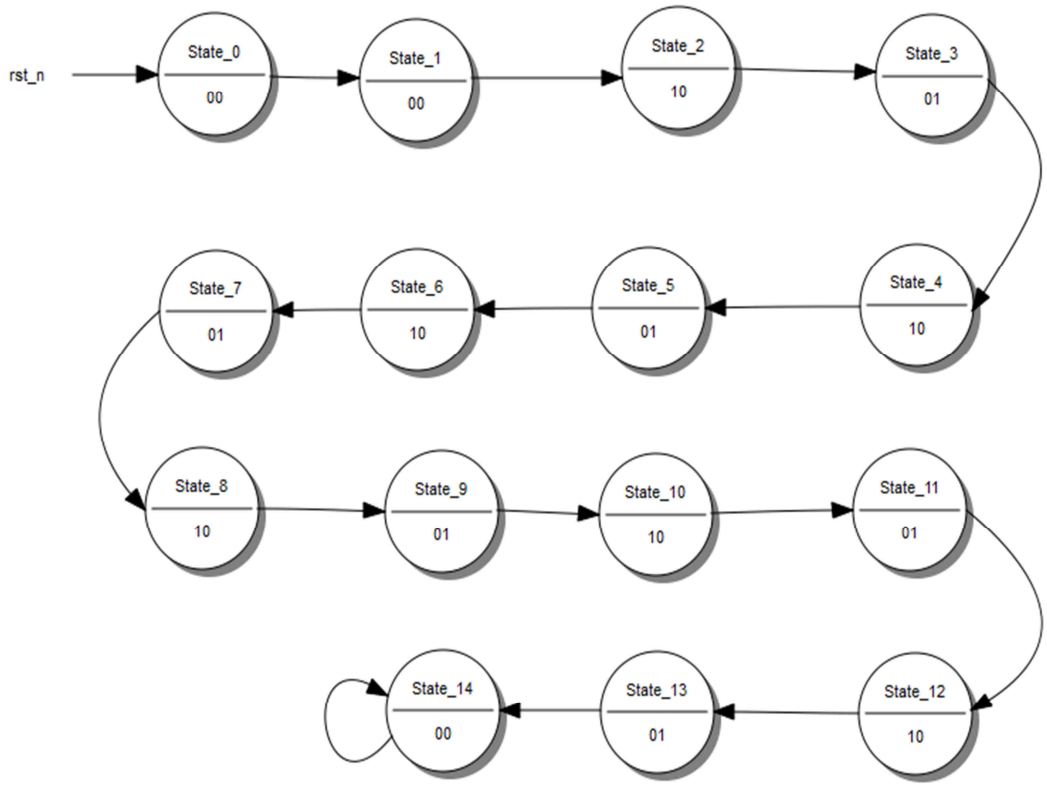


Figura 64: Esquema de la FSM de la dirección de los registros I2C.

El diagrama del bloque en el editor de esquemas quedaría:

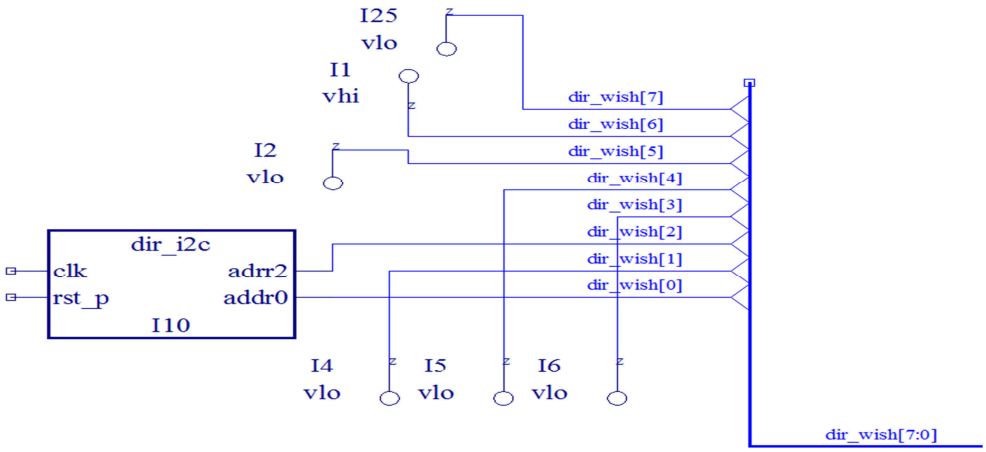


Figura 65: Esquema en el editor de la FSM de la dirección registros I2C.

El código VHDL de la máquina de estados finitos del bloque de direcciones del protocolo I2C es:

```
-- This file was generated by
-- Qfsm Version 0.53
-- (C) Stefan Duffner, Rainer Strobel


-- State/Output adrr2 addr0
-- State_0      0     0
-- State_1      0     0
-- State_2      1     0
-- State_3      0     1
-- State_4      1     0
-- State_5      0     1
-- State_6      1     0
-- State_7      0     1
-- State_8      1     0
-- State_9      0     1
-- State_10     1     0
-- State_11     0     1
-- State_12     1     0
-- State_13     0     1
-- State_14     0     0

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

ENTITY direcciones IS
  PORT (clk: IN std_ulogic;
        rst_n: IN std_ulogic;
        adrr2: OUT std_ulogic;
        addr0: OUT std_ulogic);
END direcciones;

ARCHITECTURE behave OF direcciones IS

SIGNAL current_state, next_state : std_ulogic_vector(3 DOWNTO 0);

CONSTANT State_0 : std_ulogic_vector(3 DOWNTO 0) := "0000";
CONSTANT State_1 : std_ulogic_vector(3 DOWNTO 0) := "0001";
CONSTANT State_2 : std_ulogic_vector(3 DOWNTO 0) := "0010";
CONSTANT State_3 : std_ulogic_vector(3 DOWNTO 0) := "0011";
CONSTANT State_4 : std_ulogic_vector(3 DOWNTO 0) := "0100";
CONSTANT State_5 : std_ulogic_vector(3 DOWNTO 0) := "0101";
CONSTANT State_6 : std_ulogic_vector(3 DOWNTO 0) := "1111";
CONSTANT State_7 : std_ulogic_vector(3 DOWNTO 0) := "0111";
CONSTANT State_8 : std_ulogic_vector(3 DOWNTO 0) := "1000";
CONSTANT State_9 : std_ulogic_vector(3 DOWNTO 0) := "1001";
CONSTANT State_10 : std_ulogic_vector(3 DOWNTO 0) := "1010";
CONSTANT State_11 : std_ulogic_vector(3 DOWNTO 0) := "1011";
CONSTANT State_12 : std_ulogic_vector(3 DOWNTO 0) := "1100";
CONSTANT State_13 : std_ulogic_vector(3 DOWNTO 0) := "1101";
CONSTANT State_14 : std_ulogic_vector(3 DOWNTO 0) := "1110";

BEGIN
  state_register: PROCESS (rst_n, clk)
  BEGIN
    IF rst_n='0' THEN
      current_state <= State_0;
    ELSIF rising_edge(clk) THEN
      current_state <= next_state;
    END IF;
  END PROCESS;
```

```
63
64    next_state_and_output_logic: PROCESS (current_state)
65      VARIABLE temp_output : std_ulogic_vector(1 DOWNTO 0);
66    BEGIN
67      CASE current_state IS
68        WHEN State_0 => temp_output := "00";
69        next_state <= State_1;
70        WHEN State_1 => temp_output := "00";
71        next_state <= State_2;
72        WHEN State_2 => temp_output := "10";
73        next_state <= State_3;
74        WHEN State_3 => temp_output := "01";
75        next_state <= State_4;
76        WHEN State_4 => temp_output := "10";
77        next_state <= State_5;
78        WHEN State_5 => temp_output := "01";
79        next_state <= State_6;
80        WHEN State_6 => temp_output := "10";
81        next_state <= State_7;
82        WHEN State_7 => temp_output := "01";
83        next_state <= State_8;
84        WHEN State_8 => temp_output := "10";
85        next_state <= State_9;
86        WHEN State_9 => temp_output := "01";
87        next_state <= State_10;
88        WHEN State_10 => temp_output := "10";
89        next_state <= State_11;
90        WHEN State_11 => temp_output := "01";
91        next_state <= State_12;
92        WHEN State_12 => temp_output := "10";
93        next_state <= State_13;
94        WHEN State_13 => temp_output := "01";
95        next_state <= State_14;
96        WHEN State_14 => temp_output := "00";
97        next_state <= State_14;
98        WHEN OTHERS => temp_output := (OTHERS =>'X');
99        next_state <= State_0;
100     END CASE;
101     adrr2 <= temp_output(1);
102     addr0 <= temp_output(0);
103   END PROCESS;
104
105 END behave;
106
```

Figura 66: Código VHDL de las direcciones del I2C en el editor de VHDL.

Para verificar que el código VHDL es correcto, se realiza una simulación con la herramienta de simulación Active-HDL del software de Lattice Diamond 3.2. Como podemos ver, el código corresponde a la perfección con la máquina de estados finitos de las direcciones del I2C.

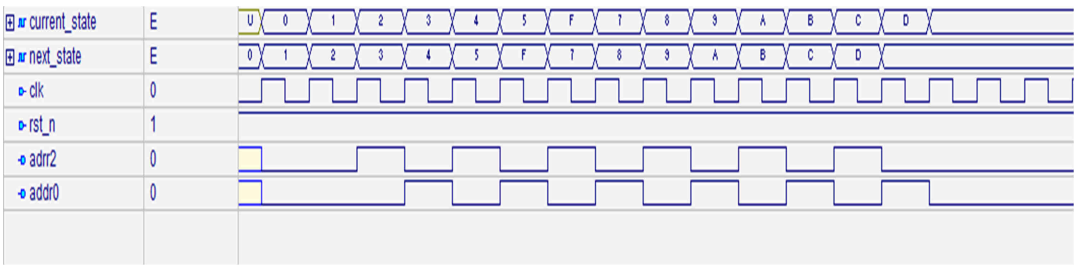


Figura 67: Simulación del VHDL de las direcciones del I2C.

Datos del registro:

Los datos a enviar a los registros de los diferentes dispositivos en función de los bits es la siguiente:

Orden/Bit.	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Función
0	1	0	0	0	0	0	0	0	Habilitar I2C.
1	1	0	0	0	0	0	0	0	
2	0	0	1	1	0	1	0	0	Reset.
3	1	0	0	1	0	1	0	0	
4	0	0	0	1	1	1	1	0	
5	0	0	0	1	0	1	0	0	
6	0	0	0	0	0	0	0	0	
7	0	1	0	1	0	1	0	0	
8	0	0	1	1	0	1	0	0	Control de consumo.
9	1	0	0	1	0	1	0	0	
10	0	0	0	0	1	1	0	0	
11	0	1	0	1	0	1	0	0	
12	0	0	0	0	0	0	0	0	
13	0	1	0	1	0	1	0	0	
14	0	0	0	0	0	0	0	0	Inhabilitar I2C.

Podemos ver que todos los bits varían en cada estado. Estos bits son los que se programarán en la máquina de estados finitos e irán cambiando su valor en función de un reloj de periodo 1 ms creado anteriormente.

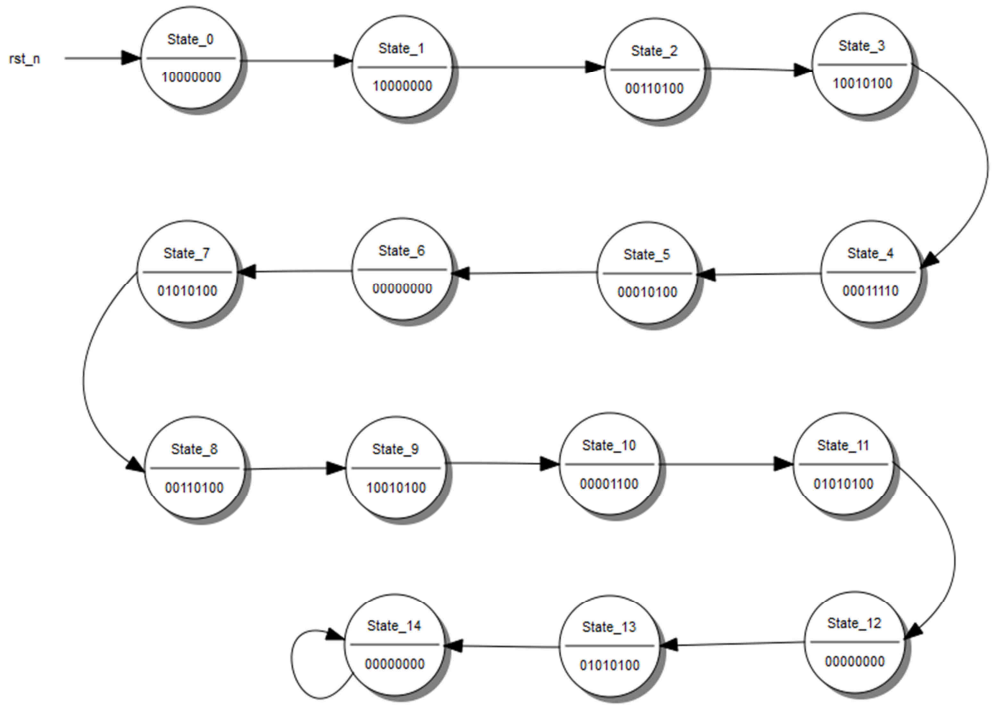


Figura 68: Esquema de la FSM de los datos I2C.

El diagrama del bloque en el editor de esquemas quedaría:

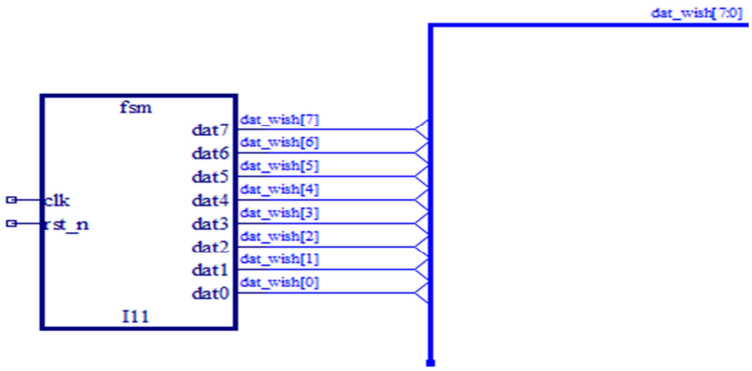


Figura 69: Esquema en el editor de la FSM de los datos I2C.

```
-- This file was generated by
-- Qfsm Version 0.53
-- (C) Stefan Duffner, Rainer Strobel


-- State/Output dat7 dat6 dat5 dat4 dat3 dat2 dat1 dat0
-- State_0      1    0    0    0    0    0    0    0
-- State_1      1    0    0    0    0    0    0    0
-- State_2      0    0    1    1    0    1    0    0
-- State_3      1    0    0    1    0    1    0    0
-- State_4      0    0    0    1    1    1    1    0
-- State_5      0    0    0    1    0    1    0    0
-- State_6      0    0    0    0    0    0    0    0
-- State_7      0    1    0    1    0    1    0    0
-- State_8      0    0    1    1    0    1    0    0
-- State_9      1    0    0    1    0    1    0    0
-- State_10     0    0    0    0    1    1    0    0
-- State_11     0    1    0    1    0    1    0    0
-- State_12     0    0    0    0    0    0    0    0
-- State_13     0    1    0    1    0    1    0    0
-- State_14     0    0    0    0    0    0    0    0

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

ENTITY datos IS
  PORT (clk: IN std_ulogic;
        rst_n: IN std_ulogic;
        dat7: OUT std_ulogic;
        dat6: OUT std_ulogic;
        dat5: OUT std_ulogic;
        dat4: OUT std_ulogic;
        dat3: OUT std_ulogic;
        dat2: OUT std_ulogic;
        dat1: OUT std_ulogic;
        dat0: OUT std_ulogic);
END datos;

ARCHITECTURE behave OF datos IS

SIGNAL current_state, next_state : std_ulogic_vector(3 DOWNTO 0);

CONSTANT State_0 : std_ulogic_vector(3 DOWNTO 0) := "0000";
CONSTANT State_1 : std_ulogic_vector(3 DOWNTO 0) := "0001";
CONSTANT State_2 : std_ulogic_vector(3 DOWNTO 0) := "0010";
CONSTANT State_3 : std_ulogic_vector(3 DOWNTO 0) := "0011";
CONSTANT State_4 : std_ulogic_vector(3 DOWNTO 0) := "0100";
CONSTANT State_5 : std_ulogic_vector(3 DOWNTO 0) := "0101";
CONSTANT State_6 : std_ulogic_vector(3 DOWNTO 0) := "0110";
CONSTANT State_7 : std_ulogic_vector(3 DOWNTO 0) := "0111";
CONSTANT State_8 : std_ulogic_vector(3 DOWNTO 0) := "1000";
CONSTANT State_9 : std_ulogic_vector(3 DOWNTO 0) := "1001";
CONSTANT State_10 : std_ulogic_vector(3 DOWNTO 0) := "1010";
CONSTANT State_11 : std_ulogic_vector(3 DOWNTO 0) := "1011";
CONSTANT State_12 : std_ulogic_vector(3 DOWNTO 0) := "1100";
CONSTANT State_13 : std_ulogic_vector(3 DOWNTO 0) := "1101";
CONSTANT State_14 : std_ulogic_vector(3 DOWNTO 0) := "1110";

BEGIN
  state_register: PROCESS (rst_n, clk)
  BEGIN
    IF rst_n='0' THEN
      current_state <= State_0;
    ELSIF rising_edge(clk) THEN
      current_state <= next_state;
    END IF;
  END PROCESS;

  next_state_and_output_logic: PROCESS (current_state)
    VARIABLE temp_output : std_ulogic_vector(7 DOWNTO 0);
  BEGIN
    CASE current_state IS
      WHEN State_0 => temp_output := "10000000";
      next_state <= State_1;
      WHEN State_1 => temp_output := "10000000";
      next_state <= State_2;
      WHEN State_2 => temp_output := "00110100";
      next_state <= State_3;
      WHEN State_3 => temp_output := "10010100";
      next_state <= State_4;
      WHEN State_4 => temp_output := "00011110";
      next_state <= State_5;
      WHEN State_5 => temp_output := "00010100";
      next_state <= State_6;
      WHEN State_6 => temp_output := "00000000";
      next_state <= State_6;
```

```
88          WHEN State_7 => temp_output := "01010100";
89          next_state <= State_8;
90          WHEN State_8 => temp_output := "00110100";
91          next_state <= State_9;
92          WHEN State_9 => temp_output := "10010100";
93          next_state <= State_10;
94          WHEN State_10 => temp_output := "00001100";
95          next_state <= State_11;
96          WHEN State_11 => temp_output := "01010100";
97          next_state <= State_12;
98          WHEN State_12 => temp_output := "00000000";
99          next_state <= State_13;
100         WHEN State_13 => temp_output := "01010100";
101         next_state <= State_14;
102         WHEN State_14 => temp_output := "00000000";
103         next_state <= State_14;
104         WHEN OTHERS => temp_output := (OTHERS =>'X');
105         next_state <= State_0;
106       END CASE;
107       dat7 <= temp_output(7);
108       dat6 <= temp_output(6);
109       dat5 <= temp_output(5);
110       dat4 <= temp_output(4);
111       dat3 <= temp_output(3);
112       dat2 <= temp_output(2);
113       dat1 <= temp_output(1);
114       dat0 <= temp_output(0);
115     END PROCESS;
116
117 END behave;
118
119
```

Figura 70: Código VHDL de los datos del I2C en el editor de VHDL.

Para verificar que el código VHDL es correcto, se realiza una simulación con la herramienta de simulación Active-HDL del software de Lattice Diamond 3.2. Como podemos ver, el código corresponde a la perfección con la máquina de estados finitos de los datos del I2C.

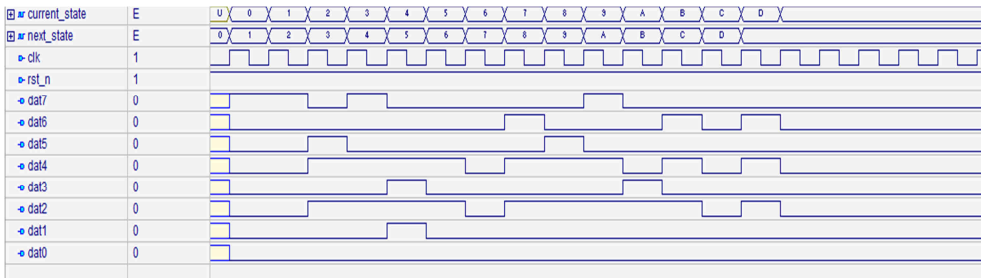


Figura 71: Simulación del VHDL de los datos del I2C.

El spreadsheet view del circuito para las diferentes entradas y salidas del esquema será:

	Name	Group By	Pin	BANK	BANK_VCC	VREF	IO_TYPE	PULLMODE	DRIVE
1	All Ports	N/A	N/A	N/A	N/A	N/A			N/A
1.1	Input	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.1.1	Clock	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.1.1.1	Entrada50Mhz	N/A	27(27)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)
1.1.2	HAB_MSF	N/A	84(84)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)
1.1.3	Hab_50MHz	N/A	83(83)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)
1.1.4	Hab_contador	N/A	86(86)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)
1.1.5	Rst_contador	N/A	85(85)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)
1.2	Output	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.2.1	Sal_Hab_50MHz	N/A	32(32)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)
1.2.2	Salida1ms	N/A	97(97)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)
1.2.3	scl	N/A	126(126)	0(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)
1.2.4	sda	N/A	125(125)	0(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)

El esquema total del bus Wishbone maestro, desarrollado en el editor de esquemas, quedaría de la siguiente manera:

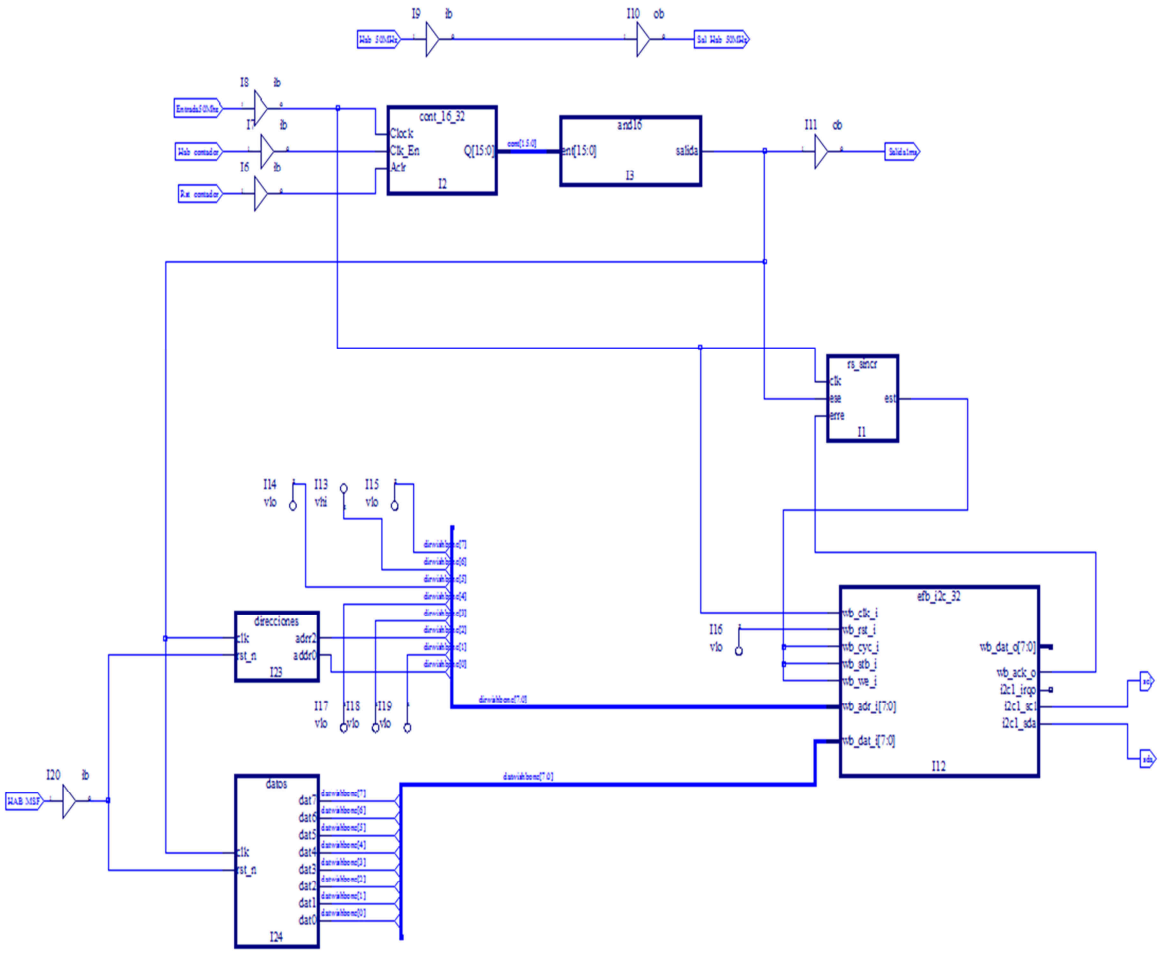


Figura 72: Esquema total del circuito de bypass.

5.1.3 Consideraciones finales:

El desarrollo el bypass analógico, a través del códec de audio, se ha realizado con la placa de apoyo necesaria y la puesta a punto de la plataforma del sistema de adquisición y procesamiento de audio.

Se ha comprobado el correcto funcionamiento del conjunto, desde la adquisición de audio a través de la fuente de audio analógica, hasta la escucha del audio por medio de unos altavoces.

5.2 Implementación del bypass digital.

5.2.1 Desarrollo del hardware.

Configuración del Códec:

La función del códec es la de convertir la señal analógica que le llega de la entrada a una señal digital para realizar el procesamiento. Una vez realizado el procesamiento en el dispositivo FPGA, vuelve a convertir la señal digital a analógica para su reproducción por unos altavoces.

La transferencia de datos entre el códec y el dispositivo FPGA se realiza a través del protocolo de comunicaciones SPI, previamente inicializado y controlado por el protocolo de comunicaciones I2C.

Se usará una señal estéreo, con un canal izquierdo y un canal derecho. Estos canales analógicos son:

- De entrada: Llinein, Rlinein.
- De salida: Lout, Rout.

Las señales de entrada son entradas de alta impedancia y baja capacitancia, ideal para recibir niveles de señales de un equipo externo de audio.

El esquema interno de entrada del códec es:

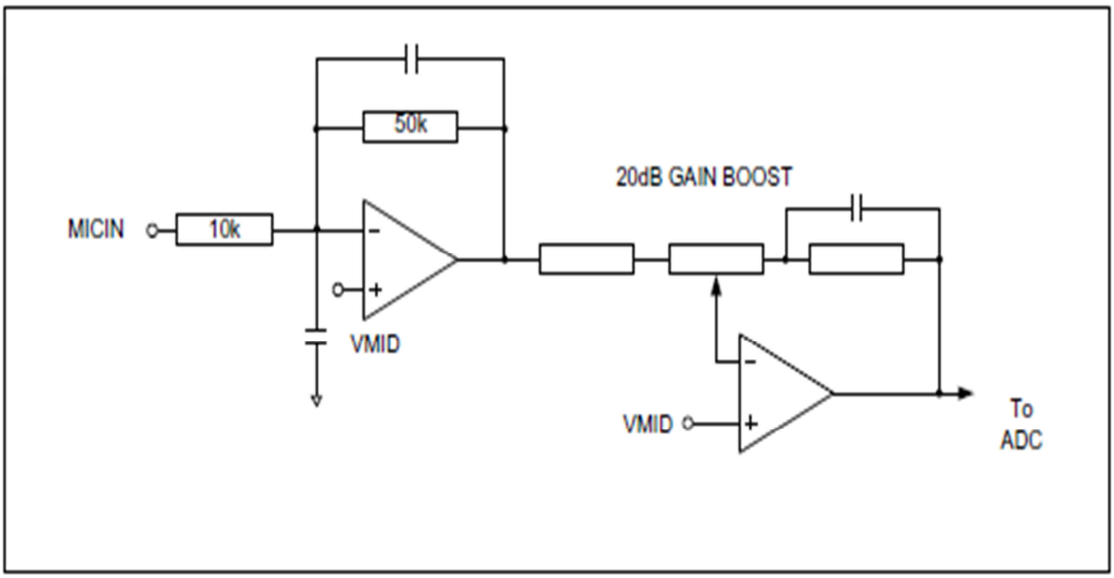


Figura 73: Esquema de entrada del códec.

Ambas líneas de entrada incluyen diferentes niveles programables y un ADC mute, que desactiva el convertidor A/D interno del códec.

Las señales de salida tienen baja impedancia y una alta capacitancia, y pueden ser usadas desde la salida del convertidor D/A interno del códec o desde la línea de entrada, cuando se configura para que funcione por el método bypass. Estas señales no tienen niveles programables o ajustables de ganancia, sino que esta es fija de 0 decibelios.

El esquema interno de salida del códec es:

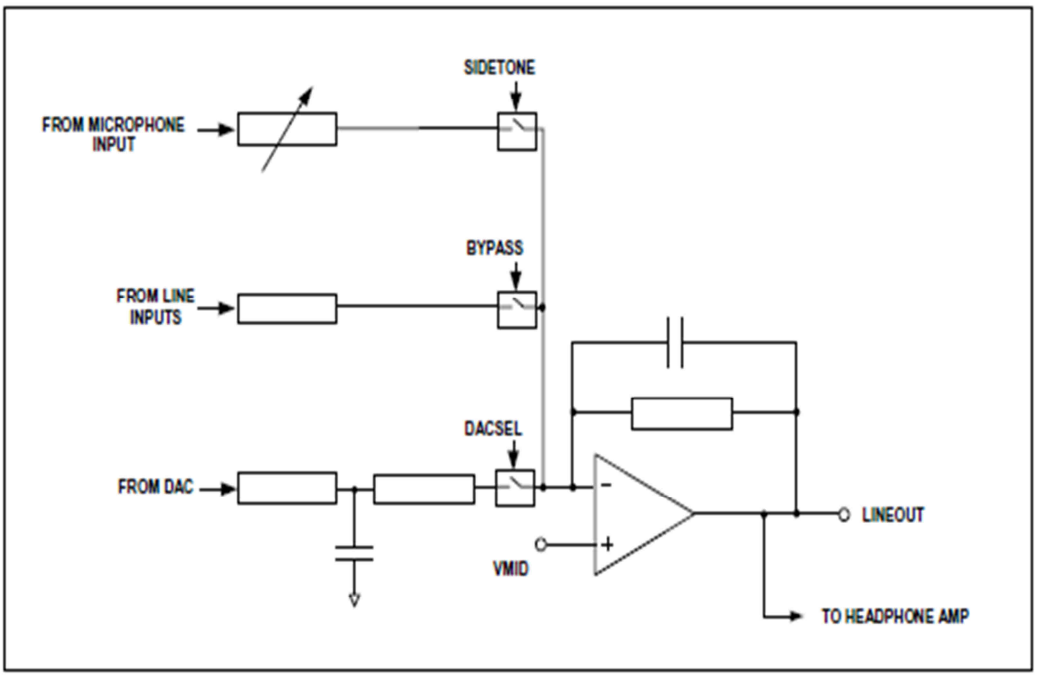


Figura 74: Esquema de salida del códec.

5.2.2 Desarrollo de la configuración:

Introducción y bloque EFB:

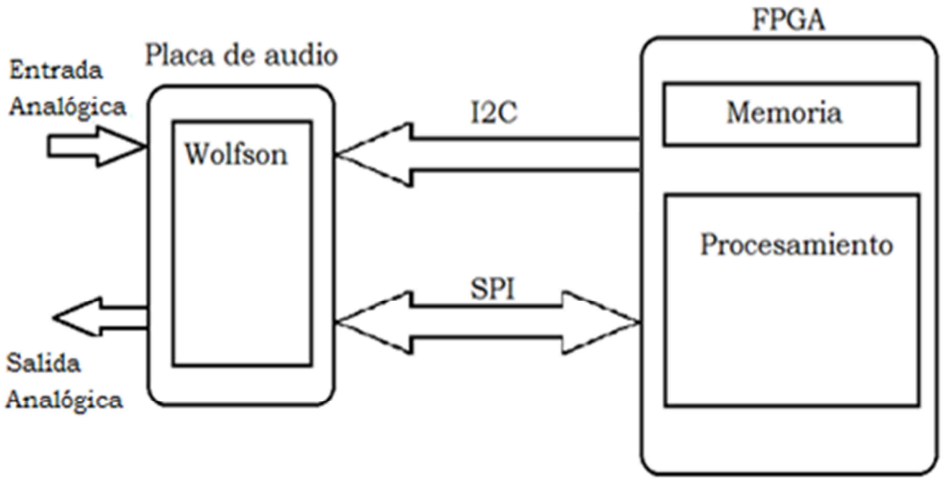


Figura 75: Esquema general de la placa de procesamiento y sus protocolos.

Como ya se explicó en el apartado 5.1 en la implementación del bypass, será necesario crear primeramente un protocolo I2C para inicializar el dispositivo y poder así enviar la información inicial correspondiente que se desea al códec de audio.

Para ello se necesitará crear un bloque EFB para transferir esta información al códec por el bus Wishbone, y que este pueda activar elementos necesarios utilizados para digitalizar la información analógica de la entrada, contenidos en su estructura interna como los conversores A/D y D/A.

Una vez inicializado el dispositivo, se procederá al protocolo de transporte de datos, que en este caso será el protocolo SPI. Los datos de entrada DATAIN de la señal analógica son enviados al conversor D/A, donde se mandará al dispositivo FPGA para que realice el correspondiente procesamiento digital. Una vez procesado dentro del dispositivo FPGA mediante sus estructuras correspondientes, se mandará de vuelta a códec de audio por el mismo protocolo SPI, donde se transformará la señal digital a una señal analógica en el conversor A/D y se enviarán a DATAOUT para su posterior escucha por los altavoces.

La creación del bloque EFB se realizará de la misma manera que se creó para la configuración de bypass, explicado en el apartado Bloque EFB, con sus mismas características y valores.

Modificación del valor de los registros I2C:

Ambos núcleos I2C se comunican con el interfaz bus WISHBONE del módulo EFB, a través de un conjunto de controles, comandos, estados y/o registros de datos.

Esta tabla muestra los nombres de los registros internos y sus funciones. Estos registros son un subconjunto del mapa del registro EFB.

I²C Primary Register Name	I²C Secondary Register Name	Register Function	Address I²C Primary	Address I²C Secondary	Access
I2C_1_CR	I2C_2_CR	Control	0x40	0x4A	Read/Write
I2C_1_CMDR	I2C_2_CMDR	Command	0x41	0x4B	Read/Write
I2C_1_BR0	I2C_2_BR0	Clock Pre-scale	0x42	0x4C	Read/Write
I2C_1_BR1	I2C_2_BR1	Clock Pre-scale	0x43	0x4D	Read/Write
I2C_1_TXDR	I2C_2_TXDR	Transmit Data	0x44	0x4E	Write
I2C_1_SR	I2C_2_SR	Status	0x45	0x4F	Read
I2C_1_GCDR	I2C_2_GCDR	General Call	0x46	0x50	Read
I2C_1_RXDR	I2C_2_RXDR	Receive Data	0x47	0x51	Read
I2C_1_IRQ	I2C_2_IRQ	IRQ	0x48	0x52	Read/Write
I2C_1_IRQEN	I2C_2_IRQEN	IRQ Enable	0x49	0x53	Read/Write

Note: Unless otherwise specified, all reserved bits in writable registers shall be written '0'.

Figura 76: Registros del protocolo I2C en el bus Wishbone.

Las operaciones con los diferentes registros se encuentran en el ANEXO I: registros internos del Wishbone.

Para ello, a partir del circuito establecido anteriormente para la configuración I2C, en el cual había un pulso de 1ms y un bloque EFB para transferir dicha información a través del bus WISHBONE, sería necesario cambiar el valor de varios de los registros internos del códec, que realizarán tareas, por ejemplo, de activación de los convertidores o de modificación de su circuito para adaptarse a las configuraciones requeridas.

Toda la información y los pasos de la trasferencia del protocolo I2C están explicados y argumentados en el punto 5.1 de bypass en el apartado de transferencia del I2C. Lo único que sería necesario modificar es el valor de dicha información a enviar al códec, ya que antes se realizaba un bypass analógico y ahora se desea realizar un bypass digital.

Direcciones del registro:

Las secuencias de bits de las direcciones de los registros internos del WISHBONE a los que accedemos son tres principalmente, mostrados con detalle en el ANEXO I.

Dirección hexadecimal	Dirección binaria
0x40	0100 || 0000
0x41	0100 || 0001
0x44	0100 || 0100

Por lo tanto, las direcciones de los registros que tendremos que mandar ordenadamente serían:

Estado/Bit.	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Función
0	0	1	0	0	0	0	0	0	Inicializar I2C
1	0	1	0	0	0	0	0	0	
2	0	1	0	0	0	1	0	0	Reset
3	0	1	0	0	0	0	0	1	
4	0	1	0	0	0	1	0	0	
5	0	1	0	0	0	0	0	1	
6	0	1	0	0	0	1	0	0	
7	0	1	0	0	0	0	0	1	
8	0	1	0	0	0	1	0	0	LlineIN
9	0	1	0	0	0	0	0	1	
10	0	1	0	0	0	1	0	0	
11	0	1	0	0	0	0	0	1	
12	0	1	0	0	0	1	0	0	
13	0	1	0	0	0	0	0	1	
14	0	1	0	0	0	1	0	0	RlineIN
15	0	1	0	0	0	0	0	1	
16	0	1	0	0	0	1	0	0	
17	0	1	0	0	0	0	0	1	
18	0	1	0	0	0	1	0	0	
19	0	1	0	0	0	0	0	1	
20	0	1	0	0	0	1	0	0	Control Analógico
21	0	1	0	0	0	0	0	1	
22	0	1	0	0	0	1	0	0	
23	0	1	0	0	0	0	0	1	
24	0	1	0	0	0	1	0	0	
25	0	1	0	0	0	0	0	1	
26	0	1	0	0	0	1	0	0	

27	0	1	0	0	0	0	0	1	Control Digital
28	0	1	0	0	0	1	0	0	
29	0	1	0	0	0	0	0	1	
30	0	1	0	0	0	1	0	0	
31	0	1	0	0	0	0	0	1	
32	0	1	0	0	0	1	0	0	Formato Digital
33	0	1	0	0	0	0	0	1	
34	0	1	0	0	0	1	0	0	
35	0	1	0	0	0	0	0	1	
36	0	1	0	0	0	1	0	0	
37	0	1	0	0	0	0	0	1	
38	0	1	0	0	0	1	0	0	Control de consumo.
39	0	1	0	0	0	0	0	1	
40	0	1	0	0	0	1	0	0	
41	0	1	0	0	0	0	0	1	
42	0	1	0	0	0	1	0	0	
43	0	1	0	0	0	0	0	1	
44	0	1	0	0	0	0	0	0	Inhabilitar.

Podemos ver que salvo los bits A2 y A0, todos los demás bit se mantienen a un nivel fijo. Los bits que varían de valor son los que se programarán en la máquina de estados finitos o FSM, e irán cambiado en función a la variación de un reloj de periodo 1 ms creado anteriormente.

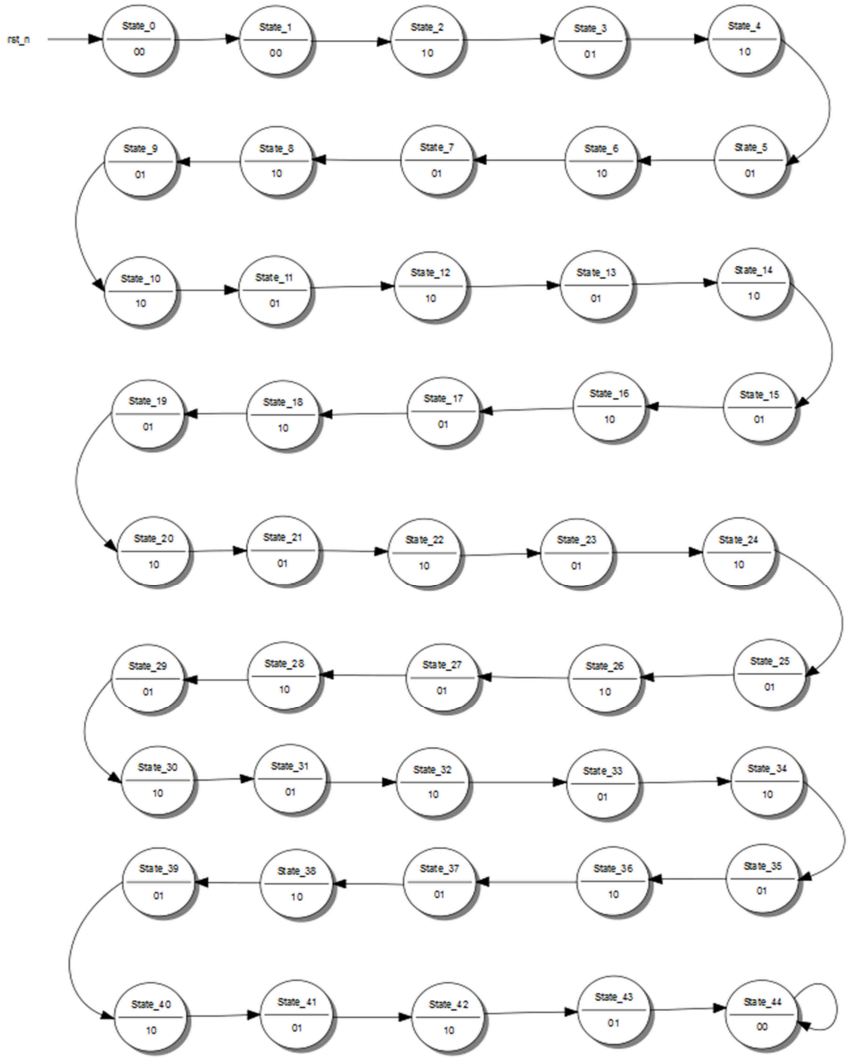


Figura 77: Esquema de la FSM de la dirección de los registros del I2C.

El diagrama del bloque quedaría:

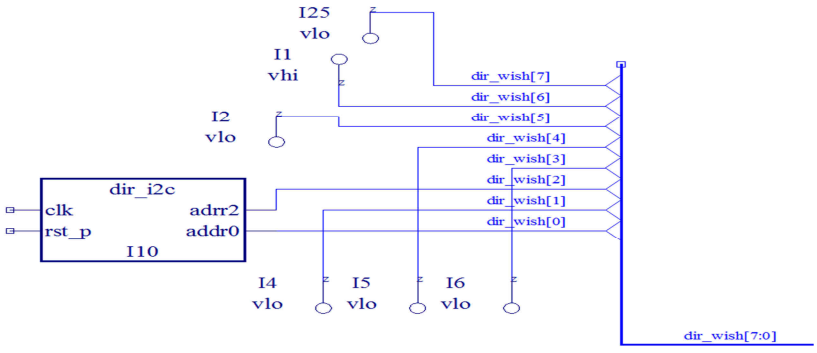


Figura 78: Esquema en el editor de la FSM de dirección de los registros I2C.

Datos del registro:

Las órdenes a realizar en los registros en función de los bits es la siguiente:

Estado/Bit.	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Función
0	1	0	0	0	0	0	0	0	Inicializar I2C
1	1	0	0	0	0	0	0	0	
2	0	0	1	1	0	1	0	0	Reset
3	1	0	0	1	0	1	0	0	
4	0	0	0	1	1	1	1	0	
5	0	0	0	1	0	1	0	0	
6	0	0	0	0	0	0	0	0	
7	0	1	0	1	0	1	0	0	
8	0	0	1	1	0	1	0	0	LlineIN
9	1	0	0	1	0	1	0	0	
10	0	0	0	0	0	0	0	0	
11	0	1	0	1	0	1	0	0	
12	0	0	0	1	0	1	1	1	
13	0	1	0	1	0	1	0	0	
14	0	0	1	1	0	1	0	0	RlineIN
15	1	0	0	1	0	1	0	0	
16	0	0	0	0	0	0	1	0	
17	0	0	0	1	0	1	0	0	
18	0	0	0	1	0	1	1	1	
19	0	1	0	1	0	1	0	0	
20	0	0	1	1	0	1	0	0	Control Analógico
21	1	0	0	1	0	1	0	0	
22	0	0	0	0	1	0	0	0	
23	0	0	0	1	0	1	0	0	
24	0	0	0	1	0	0	1	0	
25	0	1	0	1	0	1	0	0	
26	0	0	1	1	0	1	0	0	Control Digital
27	1	0	0	1	0	1	0	0	
28	0	0	0	0	1	0	1	0	
29	0	0	0	1	0	1	0	0	
30	0	0	0	0	1	1	1	0	
31	0	1	0	1	0	1	0	0	
32	0	0	1	1	0	1	0	0	Formato Digital
33	1	0	0	1	0	1	0	0	
34	0	0	0	1	0	1	0	0	
35	0	0	0	1	0	1	0	0	
36	0	1	0	0	0	0	0	0	
37	0	1	0	1	0	1	0	0	

38	0	0	1	1	0	1	0	0	Control de consumo.
39	1	0	0	1	0	1	0	0	
40	0	0	0	0	1	1	0	0	
41	0	1	0	1	0	1	0	0	
42	0	0	0	0	0	0	0	0	
43	0	1	0	1	0	1	0	0	
44	0	0	0	0	0	0	0	0	Deshabilitar

Con estas modificaciones lo que hemos hecho es:

- Modificar la entrada de los canales izquierdo y derecho, ya que hemos habilitado el camino al convertidor A/D (ADC) debido que estaba desactivado por defecto.

- En el control analógico hemos habilitado el convertidor D/A (DAC) con el registro DACSEL, y hemos inhabilitado la configuración de bypass que estaba seleccionado por defecto en el códec.

- En el control digital hemos habilitado el control de fase a 48 KHz, para que un dato completo se registre cada 1/48000 segundos.

- El formato digital lo hemos modificado para que sea el códec el dispositivo maestro, el canal derecho se habilite cuando la señal DACLRC este a nivel bajo, sea un dato de un tamaño de 16 bits y los datos estén justificados a la derecha. Todas estas configuraciones se explicarán más adelante en el formato de audio.

- Hemos reseteado del contenido anterior del dispositivo con la nueva configuración del registro interno de reset.

- Se ha puesto a nivel de tensión alto todos los elementos del dispositivo, ya que el consumo en esta aplicación no es importante.

Podemos ver que todos los bits varían en cada estado. Estos bits son los que se programarán en la máquina de estados finitos e irán cambiando su valor en función de un reloj de periodo 1 ms creado anteriormente.

Figura 79: Esquema de la FSM de los datos de I2C.

El diagrama del bloque quedaría:

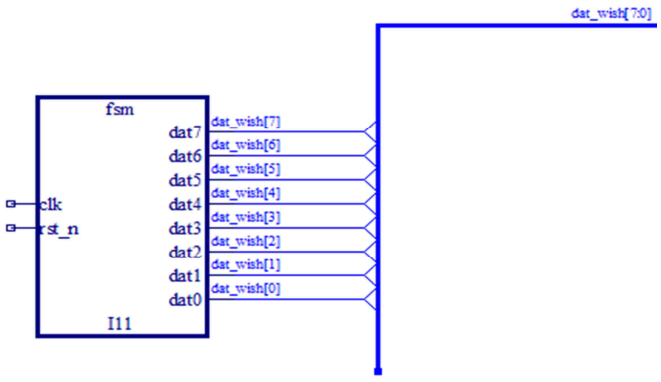


Figura 80: Esquema en el editor de la FSM de los datos I2C.

El esquema total del Wishbone maestro quedaría:

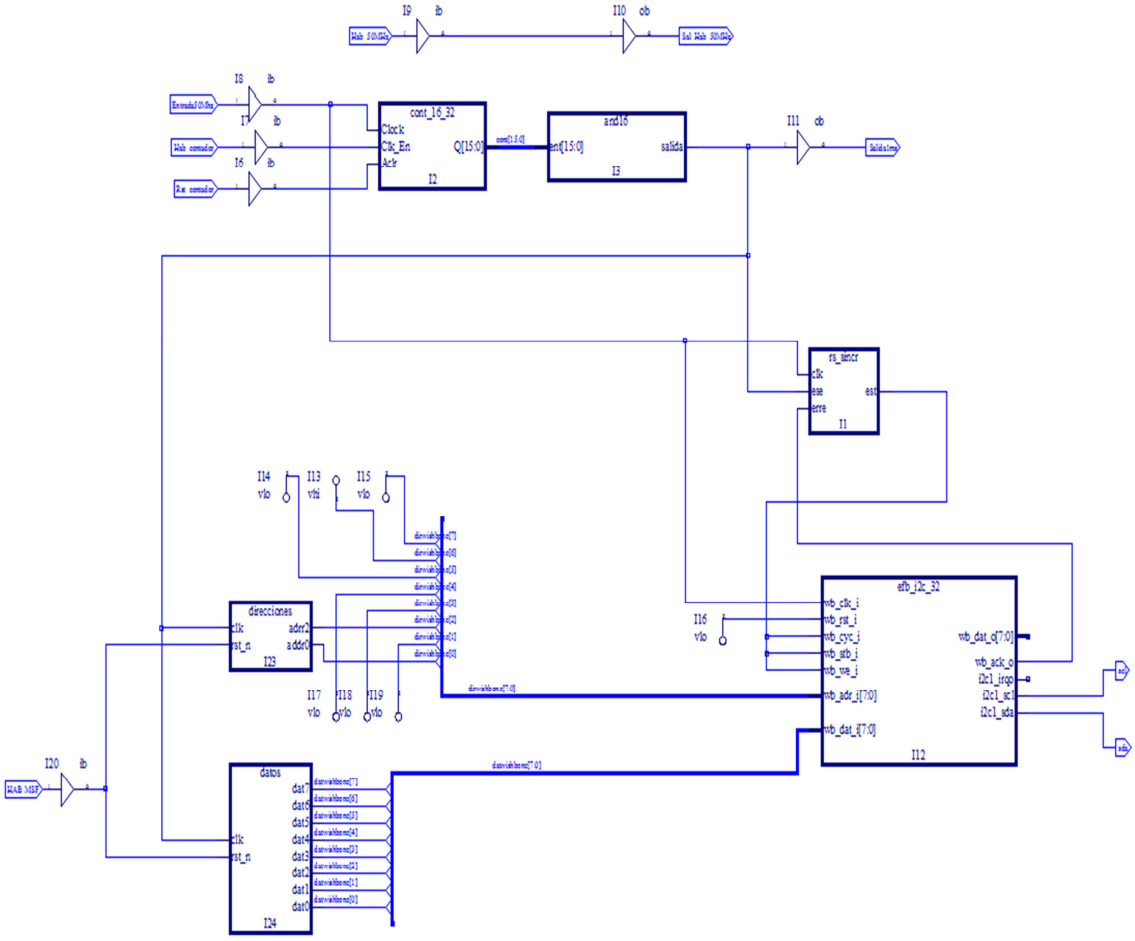


Figura 81: Esquema total del protocolo I2C.

Formato de audio:

Como hemos explicado en el punto anterior, el códec de audio se convierte en el dispositivo maestro para la transferencia de datos. Esto es así ya que necesitamos el propio reloj del códec, de 12,288 MHz, y su control de transferencia de datos se realiza a 48 KHz.

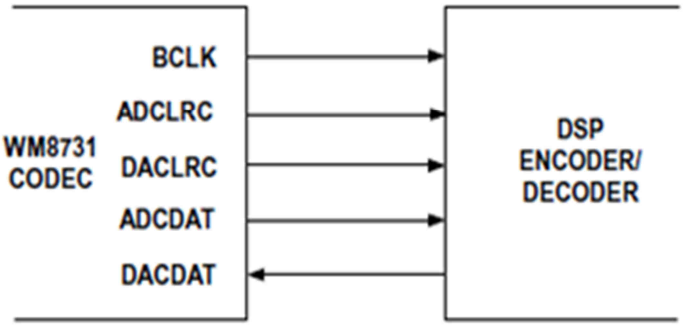


Figura 82: Esquema de conexión maestra del códec.

El reloj BCLK se usará para transferir la información debido a que es un hardware ya existente, y así no tenemos que crear un reloj con los recursos disponibles del dispositivo FPGA.

Los datos de audio se transfieren a una frecuencia de 48 KHz, y cada dato consta de 16 bits.

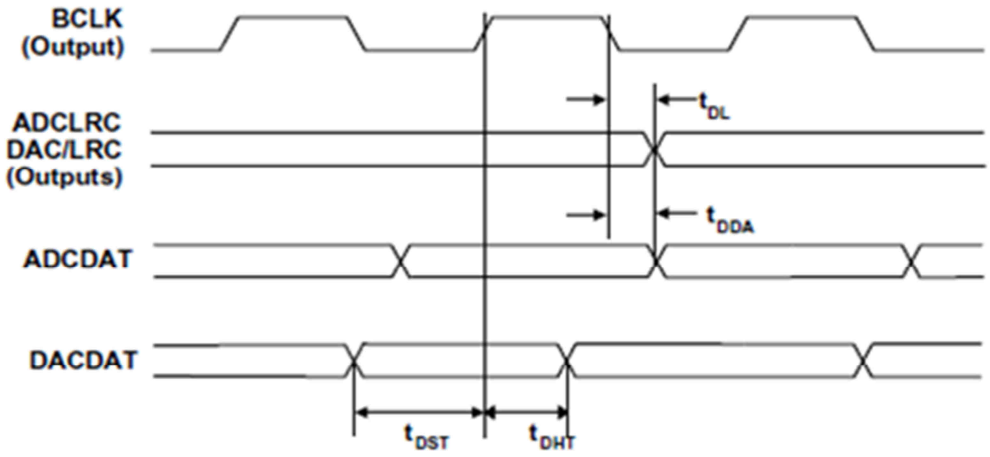


Figura 83: Datagrama de envío de datos digitales en modo maestro.

Los datos de tiempos mínimos de envió de datos digitales en modo maestro del códec son:

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Audio Data Input Timing Information						
ADCLRC/DACLRC propagation delay from BCLK falling edge	t_{DL}		0		10	ns
ADCDAT propagation delay from BCLK falling edge	t_{DDA}		0		15	ns
DACDAT setup time to BCLCK rising edge	t_{DST}		10			ns
DACDAT hold time from BCLK rising edge	t_{DHT}		10			ns

Como se puede comprobar, cumple con los requisitos temporales de establecimiento de datos, ya que los tiempos mínimos son de magnitud de nanosegundos, y nosotros trabajamos en microsegundos.

Según hemos expresado con anterioridad, el canal derecho se habilitará cuando el DACLRC esté a nivel bajo.

El esquema de envió de datos será:

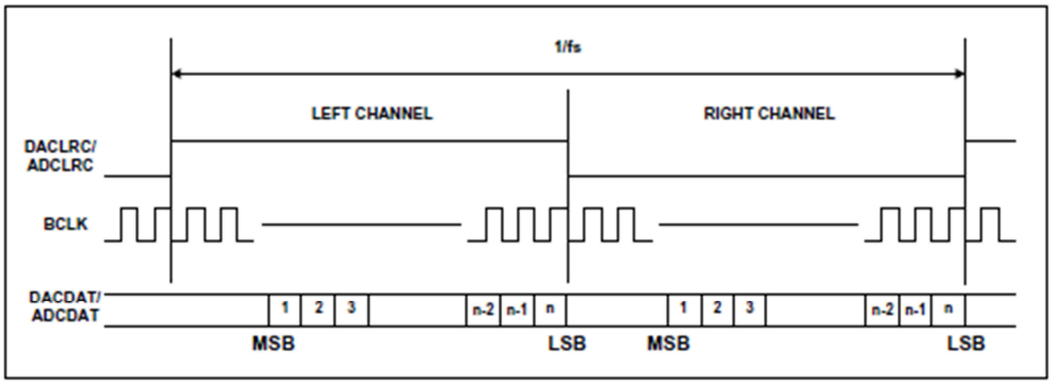


Figura 84: Envío de modo justificado a la derecha.

Como podemos ver, está los datos del byte de audio están justificados a la derecha. La explicación se realizará en el siguiente apartado.

El tiempo de lectura de todo el dato de 16 bits es de 1,4 microsegundos, ya que para cada bit tendría como máximo 81 nanosegundos (debido a que la frecuencia de los bits es de 12,288 MHZ → $81x10^{-9}$ s x 16bits ≈ $1,4x10^{-6}$ s).

Como máximo teóricamente, tendría un periodo de lectura de 20 microsegundos (frecuencia de 48 KHz para cada dato entero), muy superior al de lectura de los 16 bits. Por tanto, si se podría realizar esta implementación de frecuencias en 16 bits.

Protocolo SPI:

El protocolo SPI es el encargado de transferir estos datos digitales entre el códec y el dispositivo FPGA. El esquema del bypass digital será:

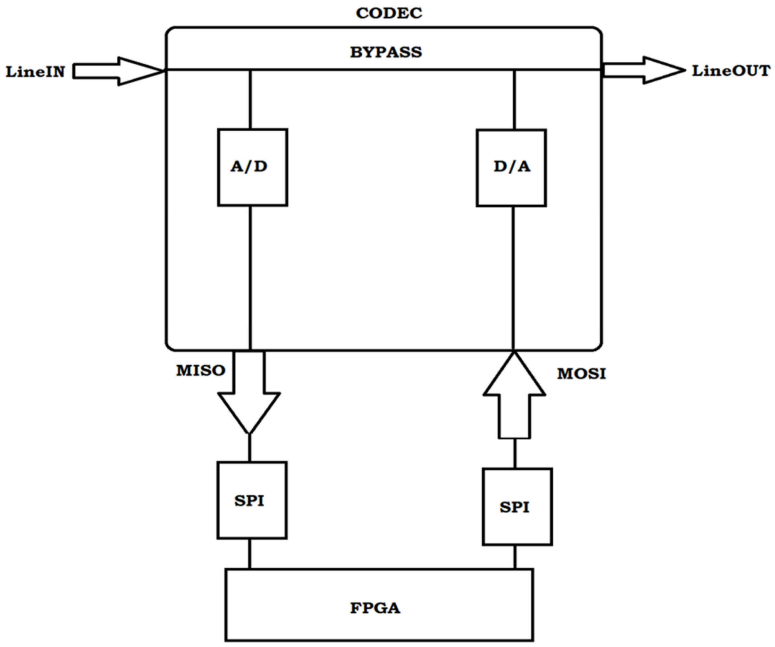


Figura 85: Esquema general del bypass digital.

El protocolo SPI será creado en el dispositivo FPGA. Se creará una interfaz nueva, en vez de utilizar la interfaz del SPI creado con las configuraciones rápidas de IP_Express del software Lattice Diamond, debido a que para esta aplicación es mejor usar un protocolo propio adaptado a nuestras necesidades para la obtención de un sonido estéreo. La principal limitación es el tamaño, ya que con el SPI pre-implantado propio del FPGA, el tamaño de transferencia es de 8 bits, mientras que para el tratamiento de sonido en esta aplicación necesitamos un tamaño de 16 bits.

Para ello, será necesario crear la lógica y la circuitería necesaria para aplicar dicho protocolo, tanto a la línea izquierda como a la línea derecha, empleando dos protocolos SPI diferenciados para la entrada y otros dos SPI para la salida.

Las dos de la entradas de audio (canal izquierdo y canal derecho) serán similares, pero con la diferencia de que tienen la habilitación contraria. Lo mismo pasaría con las configuraciones de la salida, donde su habilitación será contraria.

El SPI de entrada constará de 2 registros de desplazamiento, un registro por cada canal. Cada registro de desplazamiento tendrá un tamaño de 16 bits, que es el tamaño de cada muestra de sonido. Tendrán una habilitación o Enable, cuya señal corresponderá a la señal de ADCLRC/DACLRC, siendo su frecuencia de 48 KHZ.

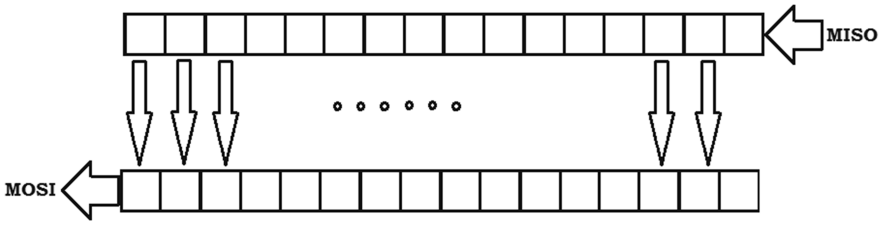


Figura 86: Esquema del registro de desplazamiento del SPI.

Por último, el reloj de los biestables será el de 12,288 MHz, que irá desplazando cada bit por cada registro de desplazamiento. Como el método de transporte es desplazado a la derecha, los últimos 16 bits que se mandarán por cada canal serán los que corresponden al dato de audio. Si no estuvieran desplazados a la derecha, estos bits se perderían debido a que solo existe 1 registro de desplazamiento con 16 bits. Por ello, si están desplazados a la derecha los bits de sonido se mantendrán en el registro de desplazamiento de 16 bits.

El esquema será:

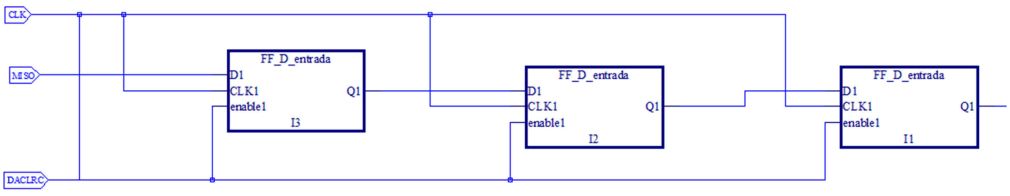


Figura 87: Esquema reducido del SPI entrada.

Cuando llegan los bits a todos los biestables del registro de desplazamiento, el de más a la derecha tiene el bit más significativo (MSB), y si se pone a nivel bajo la señal DACLRC/ADCLRC, inhabilita así la trasferencia de los registros en el canal izquierdo.

Una vez inhabilitado el circuito de entrada, se pasan los bits de entrada al circuito de salida, constituido también con 16 biestables y un multiplexor que decidirá si se realiza precarga en paralelo para ir guardando la información de los bits de entrada, o se realiza el desplazamiento para sacar dicha información por la salida del protocolo SPI de MOSI.

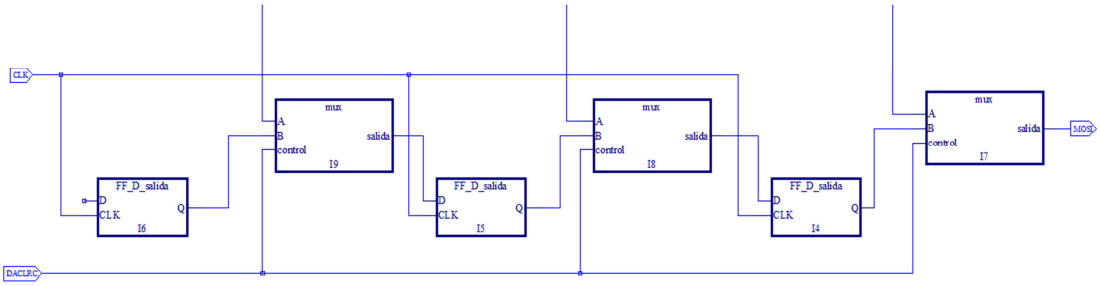


Figura 88: Esquema reducido del SPI salida.

El control irá conectado a la señal ADCLRC/DACLRC con una frecuencia de 48 KHz. Cuando esta señal esté a nivel alto, el circuito de entrada cogerá los datos y el circuito de salida precarga los datos. Cuando la señal este a nivel bajo, el circuito de entrada estará inhabilitado y el circuito de salida desplazará los datos a la salida MOSI.

El circuito que hemos realizado es el del canal izquierdo, y el del canal derecho sería similar pero con la señal de control de ADCLRC/DACLRC negada, para que cuando uno envié datos el otro reciba los datos de su canal.

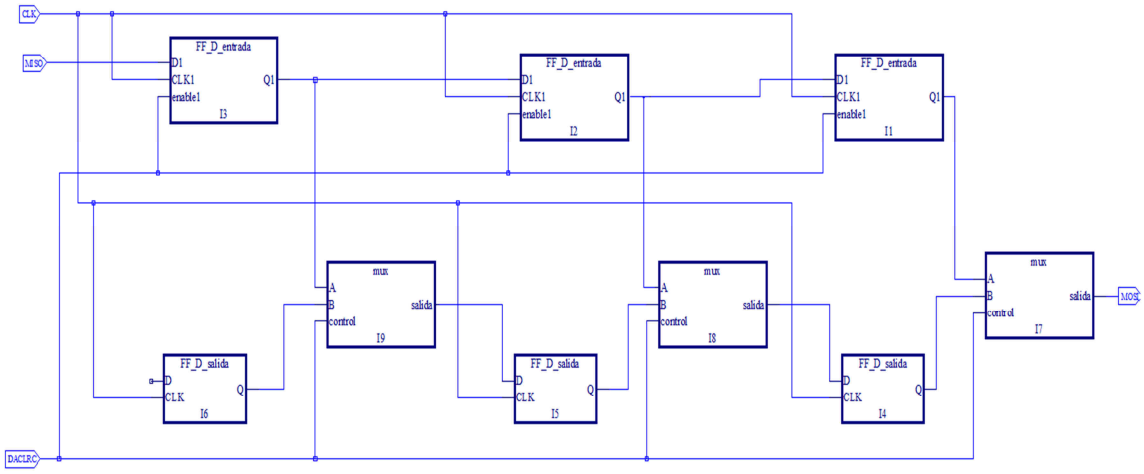


Figura 89: Esquema reducido del protocolo SPI.

Los códigos VHDL serán:

Start Page | Reports | FF_tipoD_entrada.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3
4  entity FF_D_entrada is
5  port(
6  D: in std_logic;
7  CLK: in std_logic;
8  enable: in std_logic;
9  Q: out std_logic);
10 end FF_D_entrada;
11
12 architecture FF_D_entrada_arch of FF_D_entrada is
13 begin
14 process(enable,CLK)
15 begin
16 if(enable='0')then D <= Q;
17 elsif (CLK'event and CLK='1')then Q <= D;
18 end if;
19 end process;
20 end FF_D_entrada_arch;
```

Figura 90: Código VHDL del biestable de entrada.

Start Page | Reports | FF_tipoD_salida.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3
4  entity FF_D_salida is
5  port(
6  D:in std_logic;
7  CLK:in std_logic;
8  Q:out std_logic);
9  end FF_D_salida;
10
11 architecture FF_D_salida_arch of FF_D_salida is
12 begin
13 process(CLK)
14 begin
15 if (CLK'event and CLK='1')then
16 Q <= D;
17 end if;
18 end process;
19 end FF_D_salida_arch;
```

Figura 91: Código VHDL del biestable de salida.

Start Page | Reports | multiplexor.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  entity mux is
4  port(
5  A,B,control:in std_logic;
6  salida: out std_logic);
7  end mux;
8
9  architecture mux_arch of mux is
10 begin
11 process(A,B,control)
12 begin
13 if control='1' then
14 salida<=A;
15 else salida<=B;
16 end if;
17 end process;
18 end mux_arch;
```

Figura 92: Código VHDL del multiplexor.

5.2.3 Consideraciones finales:

Se ha realizado un nuevo desarrollo del protocolo SPI ajustado a nuestras necesidades del proyecto, pero no se ha implementado en la plataforma debido a la falta de tiempo y no ser un objetivo de este trabajo de fin de grado.

Sería necesario implementar el esquema del registro de desplazamiento definido anteriormente, en el dispositivo FPGA para que actúe como protocolo SPI, y se pueda realizar el bypass digital en la plataforma del sistema de adquisición y procesamiento de audio.

Capítulo 6. Conclusiones.

La conclusión principal del trabajo realizado es la construcción y la puesta a punto de una plataforma para el procesamiento digital de audio, mediante los dispositivos FPGA y códec. Para alcanzar este dispositivo final, se han establecido resultados parciales como:

- Análisis de las diferentes alternativas de diseño que existen a un dispositivo FPGA para la implementación en esta aplicación, y se ha comprobado que esta era la mejor opción para las especificaciones.

- Se han analizado las posibilidades de un códec de audio, así como sus diferentes configuraciones y el significado de sus registros internos para obtener las diferentes configuraciones.

- Se ha dotado el hardware del diseño con una interfaz a base de interruptores, pulsadores y display de siete segmentos, los cuales indicarán la configuración que se desea aplicar al procesamiento.

- Se han utilizado las herramientas de diseño, simulación y síntesis modernas propias de la compañía de Lattice semiconductor, obteniendo un diseño estructurado y jerárquico.

- Se han empleado técnicas de diseño mixtas, usando lenguajes de descripción del hardware y esquemas para su realización. La descripción global se ha realizado en base a diferentes bloques funcionales con lenguaje VHDL, que luego se han establecido relaciones entre ellos con el editor de esquemas propio del software de diseño electrónico.

- Se han usado herramientas automáticas para la generación y la simulación de las descripciones de VHDL.

- Se han estudiado los diferentes tipos de procesamiento de audio, analizando cómo se puede conseguir los diferentes tipos de efectos y lo necesario para digitalizar una señal analógica de audio.

- Se han desarrollado los protocolos de comunicación I2C y SPI, ambos para la trasferencia de información del dispositivo FPGA. El protocolo I2C lo hemos implementado a partir de un bloque EFB del dispositivo FPGA, mientras que el protocolo SPI lo hemos desarrollado específicamente para esta aplicación a partir de dos registros de desplazamiento con 16 bits cada uno.

En cuanto a las líneas fututas de desarrollo de este proyecto, el principal desarrollo sería la implantación de los diferentes algoritmos de procesamiento de audio sobre la plataforma que se ha creado.

Esto se podría subdividir en secciones como:

- Completar la puesta a punto del bypass digital en la plataforma, implementando el esquema propuesto para el protocolo SPI utilizado para la transferencia de los datos digitales.

- Desarrollar las diferentes configuraciones del códec y del FPGA para implementar los diferentes efectos de procesamiento, explicados anteriormente en el apartado de procesamiento de audio.

- Establecer las diferentes configuraciones de dispositivo FPGA, y sus diferentes efectos a través de los diferentes pulsadores que integra la placa del hardware general, así como la visualización de los diferentes efectos que se están realizando a partir con una numeración en los display de siete segmentos.

- Utilizar la entrada de micrófono propia de la placa del códec WOLFSON para realizar el procesamiento digital a una señal procedente de un micrófono analógico polarizado, en vez de la entrada analógica procedente de una fuente digital como hemos realizado en este proyecto.

Capítulo 7. Bibliografía.

Para la realización y documentación de este trabajo de fin de grado, se ha seguido la siguiente bibliografía:

7.1 Libros:

- K.C Chang:" Digital desing and Modeling with VHDL And Synthesis". 1 º edición. IEEE Computer Society Press, 1997.
- Pong P.Chu: "FPGA prototyping by VHDL examples", 3 º edición. WILEY.
- Zwolinski Mark: "Digital System Desing with VHDL", 2 º edición. Pearson Education.

7.2 Manuales y datasheet:

- Manual del códec de audio WOLFSON WM8731.
- Manual del dispositivo FPGA MACHX02-7000-HE.
- Manual del software de creación de máquinas de estados finitos QFSM.
- Manual del software de diseño electrónico Lattice Diamond 3.2.

7.3 Web:

- http://www.latticesemi.com/Products/FPGAandCPLD/MachXO2.aspx [consulta: 10/05/2015]
- http://www.latticesemi.com/latticediamond [consulta: 10/05/2015]
- http://cdn.opencores.org/downloads/wbspec_b3.pdf [consulta: 27/05/2014]

- http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/ [consulta: 15/03/2015]

- http://robots-argentina.com.ar/Comunicacion_busI2C.htm [consulta: 15/03/2015]

- http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf [consulta: 25/04/2015]

- http://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/clase05.pdf [consulta: 20/04/2015]

- http://www.dtic.upf.edu/~egomez/teaching/sintesi/SPS1/Tema10-EfectosDigitales.pdf [consulta: 20/04/2015]

7.4 Otros proyectos final de carrera:

- Agustín García Gallego, Gonzalo: “Monitorización y control de intensidad luminosa mediante FPGA”.

- Lorente Izquierdo, David: “Implementación de un procesador digital de audio basado en FPGA”.

ANEXO I: Registros internos del I2C.

Los diferentes registros internos del WISHBONE, referidos al protocolo I2C, son los siguientes:

I²C Primary Register Name	I²C Secondary Register Name	Register Function	Address I²C Primary	Address I²C Secondary	Access
I2C_1_CR	I2C_2_CR	Control	0x40	0x4A	Read/Write
I2C_1_CMDR	I2C_2_CMDR	Command	0x41	0x4B	Read/Write
I2C_1_BR0	I2C_2_BR0	Clock Pre-scale	0x42	0x4C	Read/Write
I2C_1_BR1	I2C_2_BR1	Clock Pre-scale	0x43	0x4D	Read/Write
I2C_1_TXDR	I2C_2_TXDR	Transmit Data	0x44	0x4E	Write
I2C_1_SR	I2C_2_SR	Status	0x45	0x4F	Read
I2C_1_GCDR	I2C_2_GCDR	General Call	0x46	0x50	Read
I2C_1_RXDR	I2C_2_RXDR	Receive Data	0x47	0x51	Read
I2C_1_IRQ	I2C_2_IRQ	IRQ	0x48	0x52	Read/Write
I2C_1_IRQEN	I2C_2_IRQEN	IRQ Enable	0x49	0x53	Read/Write

Note: Unless otherwise specified, all reserved bits in writable registers shall be written '0'.

Table 4. I²C Control (Primary/Secondary)

I2C_1_CR / I2C_2_CR								0x40/0x4A
Bit	7	6	5	4	3	2	1	0
Name	I2CEN	GCEN	WKUPEN	(Reserved)	SDA_DEL_SEL[1:0]		(Reserved)	
Default	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	—	R/W	R/W	—	—

Note: A write to this register will cause the I²C core to reset.

I2CEN — I²C System Enable Bit – This bit enables the I²C core functions. If I2CEN is cleared, the ²C core is disabled and forced into idle state.
0: I²C function is disabled
1: I²C function is enabled

GCEN — Enable bit for General Call Response – Enables the general call response in slave mode.
0: Disable
1: Enable

The General Call address is defined as 0000000 and works with either 7- or 10-bit addressing

WKUPEN — Wake-up from Standby/Sleep (by Slave Address matching) Enable Bit – When this bit is enabled the, I²C core can send a wake-up signal to the on-chip power manager to wake the device up from standby/sleep. The wake-up function is activated when the MachXO2 Slave Address is matched during standby/sleep mode.
0: Disable
1: Enable

SDA_DEL_SEL[1:0] — SDA Output Delay (Tdel) Selection (see Figure 15)
00: 300 ns (min) 300 ns + 2000/[wb_clk_i frequency in MHz] (max)
01: 150 ns (min) 150 ns + 2000/[wb_clk_i frequency in MHz] (max)
10: 75 ns (min) 75 ns + 2000/[wb_clk_i frequency in MHz] (max)
11: 0 ns (min) 0 ns + 2000/[wb_clk_i frequency in MHz] (max)

Table 5. FC Command (Pri/Sec)

I2C_1_CMDR / I2C_2_CMDR								0x41/0x4B
Bit	7	6	5	4	3	2	1	0
Name	STA	STO	RD	WR	ACK	CKSDIS	(Reserved)	
Default	0	0	0	0	0	1	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	—	—

STA — Generate START (or Repeated START) condition (Master operation)

STO — Generate STOP condition (Master operation)

RD — Indicate Read from slave (Master operation)

WR — Indicate Write to slave (Master operation)

ACK — Acknowledge Option – when receiving, ACK transmission selection
0: Send ACK
1: Send NACK

CKSDIS — Clock Stretching Disable.

Clock stretching is not supported in this device. Please refer to Lattice Product Change Notice (PCN) #10A-13 for further information.

This bit must be set to '1' for all writes to this register.

I2C_1_BR0 / I2C_2_BR0								0x42/0x4C
Bit	7	6	5	4	3	2	1	0
Name	I2C_PRESCALE[7:0]							
Default[1]	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

1. Hardware default value may be overridden by EFB component instantiation parameters. See discussion below.

Table 7. I²C Register Clock Prescale 1 (Primary/Secondary)

I2C_1_BR1 / I2C_2_BR1								0x43/0x4D
Bit	7	6	5	4	3	2	1	0
Name	(Reserved)						I2C_PRESCALE[9:8]	
Default[1]	0	0	0	0	0	0	0	0
Access	—	—	—	—	—	—	R/W	R/W

1. Hardware default value may be overridden by EFB component instantiation parameters. See discussion below.

I2C_PRESCALE[9:0] — I²C Clock Prescale value. A write operation to I2CBR [9:8] will cause an I²C core reset. The WISHBONE clock frequency is divided by (I2C_PRESCALE*4) to produce the Master I²C clock frequency supported by the I²C bus (50 kHz, 100 kHz, 400 kHz).

Note: Different from transmitting a Master, the practical limit for Slave I²C bus speed support is (WISHBONE clock)/2048. For example, the maximum WISHBONE clock frequency to support a 50 kHz Slave I²C operation is 102 MHz.

Note: The digital value is calculated by IPexpress™ when the I²C core is configured in the I²C tab of the EFB GUI. The calculation is based on the WISHBONE Clock Frequency and the I²C Frequency, both entered by the user. The digital value of the divider is programmed in the MachXO2 device during device programming. After power-up or device reconfiguration, the data is loaded onto the I2C_1_BR1/0 and I2C_2_BR1/0 registers.

Table 8. I²C Transmit Data Register (Primary/Secondary)

I2C_1_TXDR / I2C_2_TXDR								0x44/0x4E
Bit	7	6	5	4	3	2	1	0
Name	I2C_Transmit_Data[7:0]							
Default	0	0	0	0	0	0	0	0
Access	W	W	W	W	W	W	W	W

I2C_Transmit_Data[7:0] — I²C Transmit Data. This register holds the byte that will be transmitted on the I²C bus during the Write Data phase. Bit 0 is the LSB and will be transmitted last. When transmitting the slave address, Bit 0 represents the Read/Write bit.

Table 9. I²C Status (Primary/Secondary)

I2C_1_SR / I2C_2_SR								0x45/0x4F
Bit	7	6	5	4	3	2	1	0
Name	TIP[1]	BUSY[1]	RARC	SRW	ARBL	TRRDY	TROE	HGC
Default	—	—	—	—	—	—	—	—
Access	R	R	R	R	R	R	R	R

1. These bits exhibit 0.5 SCK period latency before valid in R1 devices. For more details on the R1 to Standard migration refer to AN8086, Designing for Migration from MachXO2-1200-R1 to Standard (Non-R1) Devices.

TIP — Transmit In Progress. The current data byte is being transferred. Note that the TIP flag will suffer one-half SCL cycle latency right after the START condition because of the signal synchronization. Also note that this bit could be high after configuration wake-up and before the first valid I²C transfer start (when BUSY is low), and it is not indicating byte in transfer, but an invalid indicator.
- 1: Byte transfer in progress
- 0: Byte transfer complete

BUSY — I²C Bus busy. The I²C bus is involved in transaction. This is set at START condition and cleared at STOP. Note only when this bit is set should all other I²C SR bits be treated as valid indicators for a valid transfer.
- 1: I²C bus busy
- 0: I²C bus not busy

RARC — Received Acknowledge. An acknowledge response from the addressed slave (during master write) or from receiving master (during master read) was received.
- 1: No acknowledge received
- 0: Acknowledge received

SRW — Slave Read/Write. Indicates transmit or receive mode.
- 1: Master receiving / slave transmitting
- 0: Master transmitting / slave receiving

Note: SRW is valid after TRRDY=1 following a synchronization delay of up to four WISHBONE clock cycles. Do not test both SRW and TRRDY in the same WISHBONE transaction, but test SRW at least four WISHBONE clock cycles after TRRDY is tested true. This delay is represented in Figure 16.

ARBL — Arbitration Lost. The core has lost arbitration in Master mode. This bit is capable of generating an interrupt.
- 1: Arbitration Lost
- 0: Normal

TRRDY — Transmitter or Receiver Ready. The I²C Transmit Data register is ready to receive transmit data, or the I²C Receive Data Register contains receive data (dependent upon master/slave mode and SRW status). This bit is capable of generating an interrupt.
1: Transmitter or Receiver is ready
0: Transmitter of Receiver is not ready

TROE — Transmitter/Receiver Overrun Error or NACK received. A transmit or receive overrun error has occurred (dependent upon master/slave mode and SRW status), or a No Acknowledge was received (only when RARC also set). This bit is capable of generating an interrupt.
1: Transmitter or Receiver Overrun detected or NACK received
0: Normal

HGC — Hardware General Call Received. A hardware general call has been received in slave mode. The corresponding command byte will be available in the General Call Data Register. This bit is capable of generating an interrupt.
1: General Call Received in slave mode
0: Normal

Table 10. I²C General Call Data Register (Primary/Secondary)

I2C_1_GCDR / I2C_2_GCDR								0x46/0x50
Bit	7	6	5	4	3	2	1	0
Name	I2C_GC_Data[7:0]							
Default	—	—	—	—	—	—	—	—
Access	R	R	R	R	R	R	R	R

I2C_ GC _Data[7:0] — I²C General Call Data. This register holds the second (command) byte of the General Call transaction on the I²C bus.

Table 11. I²C Receive Data Register (Primary/Secondary)

I2C_1_RXDR / I2C_2_RXDR								0x47/0x51
Bit	7	6	5	4	3	2	1	0
Name	I2C_Receive_Data[7:0]							
Default	—	—	—	—	—	—	—	—
Access	R	R	R	R	R	R	R	R

I2C_ Receive _Data[7:0] — I²C Receive Data. This register holds the byte captured from the I²C bus during the Read Data phase. Bit 0 is LSB and was received last.

Table 12. I²C Interrupt Status (Primary/Secondary)

I2C_1_IRQ / I2C_2_ IRQ								0x48/0x52
Bit	7	6	5	4	3	2	1	0
Name	(Reserved)				IRQARBL	IRQTRRDY	IRQTROE	IRQHGC
Default	—	—	—	—	—	—	—	—
Access	—	—	—	—	R/W	R/W	R/W	R/W

IRQARBL — Interrupt Status for Arbitration Lost.
When enabled, indicates ARBL was asserted. Write a '1' to this bit to clear the interrupt.
1: Arbitration Lost Interrupt
0: No interrupt

IRQTRRDY — Interrupt Status for Transmitter or Receiver Ready.
When enabled, indicates TRRDY was asserted. Write a '1' to this bit to clear the interrupt.
1: Transmitter or Receiver Ready Interrupt
0: No interrupt

IRQTROE — Interrupt Status for Transmitter/Receiver Overrun or NACK received.
When enabled, indicates TROE was asserted. Write a '1' to this bit to clear the interrupt.
1: Transmitter or Receiver Overrun or NACK received Interrupt
0: No interrupt

IRQHGC — Interrupt Status for Hardware General Call Received.
When enabled, indicates HGC was asserted. Write a '1' to this bit to clear the interrupt.
1: General Call Received in slave mode Interrupt
0: No interrupt

Table 13. I²C Interrupt Enable (Primary/Secondary)

I2C_1_ IRQEN / I2C_2_IRQEN								0x49/0x53
Bit	7	6	5	4	3	2	1	0
Name	(Reserved)				IRQARBLEN	IRQTRRDYEN	IRQTROEEN	IRQHGCEN
Default	0	0	0	0	0	0	0	0
Access	—	—	—	—	R/W	R/W	R/W	R/W

IRQARBLEN — Interrupt Enable for Arbitration Lost
1: Interrupt generation enabled
0: Interrupt generation disabled

IRQTRRDYEN — Interrupt Enable for Transmitter or Receiver Ready
1: Interrupt generation enabled
0: Interrupt generation disabled

IRQTROEEN — Interrupt Enable for Transmitter/Receiver Overrun or NACK Received
1: Interrupt generation enabled
0: Interrupt generation disabled

IRQHGCEN — Interrupt Enable for Hardware General Call Received
1: Interrupt generation enabled
0: Interrupt generation disabled

ANEXO II: Registros internos del Códec.

Los diferentes registros internos del Wolfson son los siguientes:

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000000 Left Line In	4:0	LINVOL[4:0]	10111 (0dB)	Left Channel Line Input Volume Control 11111 = +12dB . . 1.5dB steps down to 00000 = -34.5dB
	7	LINMUTE	1	Left Channel Line Input Mute to ADC 1 = Enable Mute 0 = Disable Mute
	8	LRINBOTH	0	Left to Right Channel Line Input Volume and Mute Data Load Control 1 = Enable Simultaneous Load of LINVOL[4:0] and LINMUTE to RINVOL[4:0] and RINMUTE 0 = Disable Simultaneous Load
0000001 Right Line In	4:0	RINVOL[4:0]	10111 (0dB)	Right Channel Line Input Volume Control 11111 = +12dB . .1.5dB steps down to 00000 = -34.5dB
	7	RINMUTE	1	Right Channel Line Input Mute to ADC 1 = Enable Mute 0 = Disable Mute
	8	RLINBOTH	0	Right to Left Channel Line Input Volume and Mute Data Load Control 1 = Enable Simultaneous Load of RINVOL[4:0] and RINMUTE to LINVOL[4:0] and LINMUTE 0 = Disable Simultaneous Load
0000010 Left Headphone Out	6:0	LHPVOL [6:0]	1111001 (0dB)	Left Channel Headphone Output Volume Control 1111111 = +6dB . . 1dB steps down to 0110000 = -73dB 0000000 to 0101111 = MUTE
	7	LZCEN	0	Left Channel Zero Cross detect Enable 1 = Enable 0 = Disable
	8	LRHPBOTH	0	Left to Right Channel Headphone Volume, Mute and Zero Cross Data Load Control 1 = Enable Simultaneous Load of LHPVOL[6:0] and LZCEN to RHPVOL[6:0] and RZCEN 0 = Disable Simultaneous Load

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000011 Right Headphone Out	6:0	RHPVOL [6:0]	1111001 (0dB)	Right Channel Headphone Output Volume Control 1111111 = +6dB . . 1dB steps down to 0110000 = -73dB 0000000 to 0101111 = MUTE
	7	RZCEN	0	Right Channel Zero Cross detect Enable 1 = Enable 0 = Disable
	8	RLHPBOTH	0	Right to Left Channel Headphone Volume, Mute and Zero Cross Data Load Control 1 = Enable Simultaneous Load of RHPVOL[6:0] and RZCEN to LHPVOL[6:0] and LZCEN 0 = Disable Simultaneous Load
0000100 Analogue Audio Path Control	0	MICBOOST	0	Microphone Input Level Boost 1 = Enable Boost 0 = Disable Boost
	1	MUTEMIC	1	Mic Input Mute to ADC 1 = Enable Mute 0 = Disable Mute
	2	INSEL	0	Microphone/Line Input Select to ADC 1 = Microphone Input Select to ADC 0 = Line Input Select to ADC
	3	BYPASS	1	Bypass Switch 1 = Enable Bypass 0 = Disable Bypass
	4	DACSEL	0	DAC Select 1 =Select DAC 0 = Don't select DAC
	5	SIDETONE	0	Side Tone Switch 1 = Enable Side Tone 0 = Disable Side Tone
	7:6	SIDEATT[1:0]	00	Side Tone Attenuation 11 = -15dB 10 = -12dB 01 = -9dB 00 = -6dB

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000101 Digital Audio Path Control	0	ADCHPD	0	ADC High Pass Filter Enable 1 = Disable High Pass Filter 0 = Enable High Pass Filter
	2:1	DEEMP[1:0]	00	De-emphasis Control 11 = 48kHz 10 = 44.1kHz 01 = 32kHz 00 = Disable
	3	DACMU	1	DAC Soft Mute Control 1 = Enable soft mute 0 = Disable soft mute
	4	HPOR	0	Store dc offset when High Pass Filter disabled 1 = store offset 0 = clear offset
0000110 Power Down Control	0	LINEINPD	1	Line Input Power Down 1 = Enable Power Down 0 = Disable Power Down
	1	MICPD	1	Microphone Input an Bias Power Down 1 = Enable Power Down 0 = Disable Power Down
	2	ADCPD	1	ADC Power Down 1 = Enable Power Down 0 = Disable Power Down
	3	DACPD	1	DAC Power Down 1 = Enable Power Down 0 = Disable Power Down
	4	OUTPD	1	Outputs Power Down 1 = Enable Power Down 0 = Disable Power Down
	5	OSCPD	0	Oscillator Power Down 1 = Enable Power Down 0 = Disable Power Down
	6	CLKOUTPD	0	CLKOUT power down 1 = Enable Power Down 0 = Disable Power Down
	7	POWEROFF	1	POWEROFF mode 1 = Enable POWEROFF 0 = Disable POWEROFF

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000111 Digital Audio Interface Format	1:0	FORMAT[1:0]	10	Audio Data Format Select 11 = DSP Mode, frame sync + 2 data packed words 10 = I^2S Format, MSB-First left-1 justified 01 = MSB-First, left justified 00 = MSB-First, right justified
	3:2	IWL[1:0]	10	Input Audio Data Bit Length Select 11 = 32 bits 10 = 24 bits 01 = 20 bits 00 = 16 bits
	4	LRP	0	DACLRC phase control (in left, right or I^2S modes) 1 = Right Channel DAC data when DACLRC high 0 = Right Channel DAC data when DACLRC low (opposite phasing in I^2S mode) or DSP mode A/B select (in DSP mode only) 1 = MSB is available on 2nd BCLK rising edge after DACLRC rising edge 0 = MSB is available on 1st BCLK rising edge after DACLRC rising edge
	5	LRSWAP	0	DAC Left Right Clock Swap 1 = Right Channel DAC Data Left 0 = Right Channel DAC Data Right
	6	MS	0	Master Slave Mode Control 1 = Enable Master Mode 0 = Enable Slave Mode
	7	BCLKINV	0	Bit Clock Invert 1 = Invert BCLK 0 = Don't invert BCLK
0001000 Sampling Control	0	USB/ NORMAL	0	Mode Select 1 = USB mode (250/272fs) 0 = Normal mode (256/384fs)
	1	BOSR	0	Base Over-Sampling Rate USB Mode: 0 = 250fs, 1 = 272fs Normal Mode: 0 = 256fs, 1 = 384fs
	5:2	SR[3:0]	0000	ADC and DAC sample rate control; See USB Mode and Normal Mode Sample Rate sections for operation
	6	CLKIDIV2	0	Core Clock divider select 1 = Core Clock is MCLK divided by 2 0 = Core Clock is MCLK
	7	CLKODIV2	0	CLKOUT divider select 1 = CLOCKOUT is Core Clock divided by 2 0 = CLOCKOUT is Core Clock

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0001001 Active Control	0	ACTIVE	0	Activate Interface 1 = Active 0 = Inactive
0001111 Reset Register	8:0	RESET	not reset	Reset Register Writing 00000000 to register resets device

ANEXO III: Puertos de la Breakout Board.

Figure 4. J2/J4 Header Landing Callout

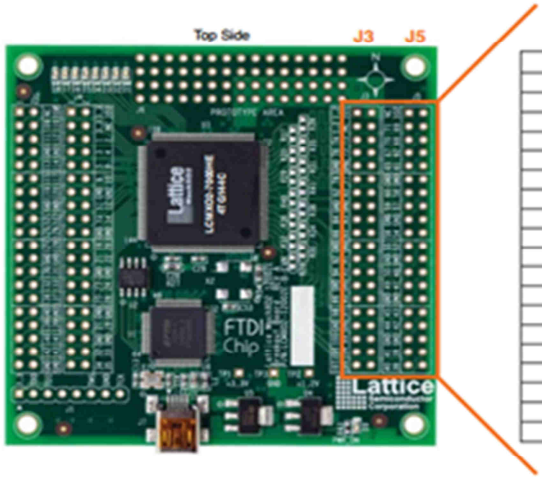
J2

1	2
NC	IO0
109	110
111	112
GND	GND
113	114
115	117
119	120
GND	GND
121	122
125	126
127	128
GND	GND
130	131
132	133
136	137
GND	GND
138	139
140	141
142	143
GND	GND

J4

1	2
3.3	IO3
3.3	NC
1	2
3	4
5	6
9	10
GND	GND
11	12
13	14
GND	GND
19	20
21	22
GND	GND
23	24
25	26
GND	GND
27	28
GND	GND
32	33
34	35

Figure 5. J3/J5 Header Landing Callout



J3

1	2
1.2	IO1
1.2	NC
74	73
76	75
GND	GND
78	77
82	81
GND	GND
84	83
86	85
GND	GND
92	91
94	93
GND	GND
96	95
98	97
GND	GND
100	99
105	104
107	106

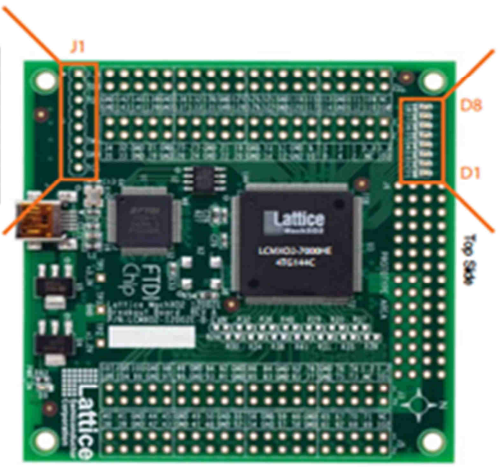
J5

1	2
NC	IO2
71	69
70	68
67	62
65	61
GND	GND
60	58
59	57
GND	GND
56	54
55	52
GND	GND
50	48
49	47
GND	GND
45	43
44	42
GND	GND
41	39
40	38

Figure 6. J1 Header Landing and LED Array Callout

J1

1
3.3
TDO
TDI
NC
NC
TMS
GND
TCK
8



LED Array

LED	Function	MachXO2 Pin
D8	LED7	107
D7	LED6	106
D6	LED5	105
D5	LED4	104
D4	LED3	100
D3	LED2	99
D2	LED1	98
D1	LED0	97