Sistema de adquisición y generación de señal basado en FPGA y placa embedded

Trabajo de fin de grado

Alumno: Sergio López Casablanca
Tutor: Juan Antonio Chávez Domínguez

Grado en Ingeniería de Sistemas Electrónicos
Escola Tècnica Superior de Telecomunicacions de Barcelona
Universitat Politècnica de Catalunya

Julio 2017

TEXIS v.1.0.

Sistema de adquisición y generación de señal basado en FPGA y placa embedded

Versión 1.0

Grado en Ingeniería de Sistemas Electrónicos
Escola Tècnica Superior de Telecomunicacions de Barcelona
Universitat Politècnica de Catalunya

Julio 2017

A mi familia.

Agradecimientos

Me gustaría empezar por agradecer a mi compañero Daniel su ayuda, ya que sin él este documento y probablemente este proyecto no habría sido posible. También dar las gracias a cada uno de mis compañeros del taller electrónico del ICFO por todo lo que he aprendido de ellos en este último año. Espero que trabajemos juntos algún día de nuevo.

Agradecer a mis padres por ser como son, por el empeño y esfuerzo que han puesto siempre en que nunca dejara los estudios.

Y por último, agradecer a todos los profesores que han pasado por mi vida académica, tanto los buenos por todo lo que me han enseñado, como los no tan buenos que han hecho que sepa aprender por mí mismo.

Abstract

This project is the result of the study, design and implementation of a signal acquisition and generation system for a FPGA platform remotely configurable by a minicomputer. A printed circuit board that includes the digital-analogue and analogue-digital converters and their connection with the FPGA module was developed. Furthermore, a VHDL described hardware was implemented in the FPGA in order to configure the converters. A Python3 based application was created to establish a serial communication between the FPGA and the minicomputer. The integration of the different parts is compatible with a modular rack system developed by ICFO´s Electronic Workshop.

Resumen

Este proyecto es el resultado del estudio, diseño e implementación de un sistema de adquisición y generación de señal para una plataforma FPGA configurable remotamente desde un miniordenador. Para ello se realizó una placa de circuito impreso con los conversores analógico-digital y digital-analógico y la conexión de cada uno de ellos con un módulo de FPGA. También se describió mediante VHDL el circuito implementado en la FPGA para la configuración de los conversores. Para la comunicación serie entre la FPGA y el miniordenador se creó una aplicación en Python3. La integración de las diferentes partes se realizó de manera compatible con un sistema de rack modular propio del taller electrónico del ICFO.

Resum

Aquest projecte és el resultat de l´estudi, disseny i implementació d´un sistema d´adquisició i generació de senyal per una plataforma FPGA configurable remotament des d´un miniordinador. Per a això es va realitzar una placa de circuit imprès amb els conversors analògic-digital i digital-analògic i la connexió de cada un d´ells amb un mòdul de FPGA. També va descriure mitjançant VHDL el circuit implementat en la FPGA per a la configuració dels conversors. Per a la comunicació sèrie entre la FPGA i el miniordinador es va crear una aplicació en Python3. La integració de les diferents parts es va realitzar de manera compatible amb un sistema de rack modular propi del taller electrònic del ICFO.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción y objetivos

1.1. Introducción

El trabajo de fin de grado se desarrolla en el taller electrónico del Institut de Ciències Fotòniques (ICFO), cuya función es brindar soporte técnico en el área de electrónica a los diversos grupos de investigación. Su principal actividad es el desarrollo de instrumentos electrónicos a medida, incluyendo el diseño y realización de circuitos impresos, programación de microcontroladores y descripción de sistemas digitales mediante VHDL.

Habitualmente, las necesidades que debe atender el taller electrónico requieren una solución en la máxima brevedad posible. Una buena estrategia para responder a este requerimiento es contar con recursos estandarizados y configurables que agilicen el desarrollo de los proyectos. Además, resulta conveniente unificar (en la medida de lo posible) el tipo de tecnologías utilizadas para realizar estos recursos. En este sentido, desde el taller electrónico se ha optado por la utilización de FPGA para aplicaciones de procesado digital de señales. Para aquellas aplicaciones que por sus características necesiten de un microprocesador, se prevé aprovechar los procesadores embedidos que ofrecen algunas familias de FPGA de Xilinx.

Una opción para implementar esta estrategia es utilizar módulos configurados con la circuitería básica para trabajar con una FPGA. En el taller electrónico se ha elegido el módulo XC7A35T-2CSG324C de la empresa Trenz Electronic, el cual incluye una FPGA de la familia ARTIX-7 de Xilinx. Esta familia de FPGA ofrece la implementación del IP Core MicroBlaze Soft Processor que permite disponer de un microprocesador RISC de 32 bits dentro de la FPGA. Añadiendo a esta nueva plataforma conversores analógico-digital (ADC) y digital-analógico (DAC), se consigue unificar con una misma tecnología necesidades cuya solución óptima requiere de una FPGA con aquellas que se solucionan mejor mediante un microcontrolador. De esta forma se consigue una alternativa a los diseños de procesado digital de señales que se

realizan actualmente en el taller electrónico basados en un microcontrolador de la familia ADuC70xx de Analog Devices. Estos diseños utilizan los ADC y DAC de 12 bits incluidos en el microcontrolador para implementar funciones tales como: controladores PID, filtros digitales, correladores, operaciones matemáticas, etc.

Con la finalidad de simplificar el desarrollo de los proyectos basados en el módulo XC7A35T-2CSG324C mencionado anteriormente, se dispone de un hardware descrito en VHDL denominado IOBOX, el cual consiste en una tabla de registros que pueden ser utilizados como entradas/salidas para implementar funciones específicas. Por otro lado, se cuenta con una aplicación denominada ICFOserial que permite la comunicación serie entre un PC y la FPGA. Finalmente, para facilitar la integración de las placas de circuito impreso y proveer la alimentación necesaria, el taller electrónico del ICFO utiliza un sistema de rack modular denominado ICFORACK, [1] Cifuentes (2016).

Con estos antecedentes, el proyecto aquí presentado consiste en un sistema de adquisición y generación de señales mediante DAC y ADC que añade funcionalidad al módulo de FPGA XC7A35T-2CSG324C mencionado anteriormente. El sistema desarrollado utiliza el hardware IOBOX implementado en la FPGA, se comunica con un miniordenador mediante una migración de la aplicación ICFOserial y es compatible con el sistema de rack modular ICFORACK.

1.2. Objetivos

Objetivo general

El objetivo general del proyeto es el desarrollo de un módulo de conversión analógico-digital y digital-analógico gestionado mediante un hardware prestablecido (IOBOX) implementado en la FPGA configurable desde un miniordenador por medio de una aplicación (ICFOserial) para realizar una comunicación serie con ésta. La integración del hardware debe ser compatible con el sistema de rack modular (ICFORACK) utilizado por el taller electrónico del ICFO.

Objetivos particulares

- Estudio comparativo de conversores AD y DA de distintos fabricantes para poder elegir los que mejor se ajustan a los requerimientos del sistema.

- Realización de una placa de circuito impreso compatible con el sistema de rack modular, que contenga los conversores y que sea compatible con la FPGA utilizada en el taller.

- Descripción VHDL compatible con IOBOX para hacer funcionar los conversores.

- Estudio comparativo de miniordenadores, con miras a elegir aquel que sea el más adecuado tanto para esta aplicación como para futuras necesidades del taller electrónico del ICFO.

- Realización de una placa de circuito impreso compatible con el sistema de rack modular, pudiendo así adaptar el miniordenador a éste.

- Desarrollo de aplicación (ICFOserial) para la comunicación serie entre miniordenador y FPGA.

1.3. Especificaciones

- 16 bits de resolución, de los cuales 14 han de ser útiles.
- Tasa de muestreo entre 0.8 MHz y 1.2 MHz.
- Miniordenador compatible para comunicación con FPGA.
- Descripción de hardware implementable en el módulo FPGA de Xilinx.
- Placa de circuito impreso compatible con ICFORACK.

1.4. Propuesta del sistema

En la figura 1.1 se muestran las diferentes partes que conforman el sistema completo propuesto, donde FPGA y conversores estarán ubicados en la misma placa de circuito impreso. Además el miniordenador debe estar acoplado a una placa que sea compatible con la mencionada anteriormente para su comunicación e inserción en el ICFORACK.

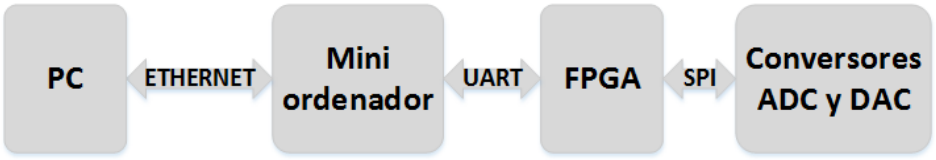


Figura 1.1: Sistema completo

Capítulo 2

Miniordenador

Como puede observarse en la figura 1.1 el miniordenador se debe comunicar con la FPGA por el protocolo serie UART y con un PC a través de Ethernet. Su función dentro del sistema de adquisición y generación de señal es la escritura y lectura de los registros de la IOBOX implementada en la FPGA. Esto permite habilitar y deshabilitar los conversores, escribir el código de conversión para enviar al DAC y leer el código recibido desde el ADC.

2.1. Estudio comparativo y selección

Para la selección de un miniordenador los principales parámetros a tener en cuenta son: conectividad, comunidad (información disponible) y precio. Por ser muy similares, no se tienen en cuenta parámetros como tamaño y consumo.

Como posibles candidatos se comparan los últimos modelos de miniordenadores en el mercado de los siguientes fabricantes: Raspberry Pi, Beaglebone, Avnet y ODROID, todos ellos mostrados en la tabla 2.1.

	Conectividad	Comunidad	Precio
Raspberry Pi 3	26 GPIO	+++	31.69 €
Beaglebone Black	65 GPIO	++	43.07 €
MicroZed	100 GPIO	+	175 €
ODROID C2	40 GPIO	+	40.61 €

Tabla 2.1: Comparativa para selección de miniordenador

Se llega a la conclusión que la *BeagleBone Black* es la mejor opción para el proyecto, ya que ofrece muy buena conectividad para proyectos electrónicos, destacando las posibilidades de configuración para cada pin de entrada/salida (PWM, timers, entradas analógicas, UART, SPI, etc.).

2.2. Comunicación con la FPGA

Actualmente, existe una aplicación para comunicar la FPGA con un PC vía USB denominada ICFOserial. La comunicación se realiza mediante un conversor serie-USB. En esta aplicación se escriben unos comandos que permiten leer y escribir en los distintos registros de la IOBOX. En la tabla 2.2 se describen los comandos usados y las distintas maneras de escribir un mismo valor:

	Comando	Registro	Valor
Escritura	10	X	Z
Escritura	10	X	YDZ
Escritura	10	X	YHZ
Escritura	10	X	YbZ
Lectura	20	X	

Tabla 2.2: Comandos IOBOX

Donde X es la posición del registro, Y es la cantidad de bytes a enviar y Z es el valor que se quiere en el registro de 16 bits. El valor a escribir en el registro se puede enviar de distintas maneras, pudiendo escribir directamente el valor decimal a enviar sin especificar cuantos bytes de información o escribir la cantidad de bytes además de la letra que indica si el valor Z está en decimal, hexadecimal o binario.

Se debe desarrollar de nuevo la aplicación que permita enviar por comunicación serie, UART, los comandos e información necesaria para que se pueda seguir utilizando el entorno IOBOX, ya que la aplicación ya diseñada no se puede utilizar en distribuciones Linux y tampoco es compatible con Windows 10. Se elige como entorno de desarrollo Python3, ya que ofrece la posibilidad de uso de la aplicación tanto desde el miniordenador como desde un PC con Windows.

Esta aplicación de poder ser ejecutada desde el terminal directamente, abierto con Python3 teniendo líneas de comandos predefinidas, pudiendo cargar un archivo `.txt` o escribiendo manualmente línea a línea, o pudiendo ser utilizado desde otro script. También se ha de poder escoger a que velocidad se comunica el puerto serie,en el caso de la comunicación con la FPGA es de 921600 bauds, además de abrir y cerrar el puerto cuando se desee. Y por último añadir una nueva funcionalidad a los mensajes a enviar, pudiendo

añadir al final de línea en los archivos `.txt` un signo '#'que da paso a escribir un comentario, siendo esta información descartada para el envío.

En la figura 2.1 se muestra el diagrama de flujo y en el anexo B se proporciona el script realizado para el envío y recepción de los datos.

2.3. Interfaz gráfica

Para hacer más cómodo el uso de la aplicación anterior, gracias al paquete tkinter, [3] Grayson (2000), se desarrolla una interfaz gráfica que llama al script explicado en la sección anterior para así poder interactuar con la FPGA de forma más rápida, teniendo en distintos archivos de texto los comandos y con un solo click ejecutar los deseados.

En la imagen 2.2 se observa el terminal desde donde se ejecuta el interface y el propio interface gráfico, donde se dispone de dos espacios para introducir el nombre del puerto por el que comunicarse, la velocidad de éste y de ocho botones donde cada uno llama a un archivo `.txt` donde están los comandos a ejecutar.

En el anexo B se dispone del código de la interfaz básica que se puede usar para cualquier proyecto.

2.4. Configuración

Para configurar la BeagleBone Black para poder usar el paquete serial con Python3 que servirá para la comunicación UART con la FPGA se debe seguir unos pasos. Para empezar, hay que tener bien sincronizada la hora y revisar y actualizar las reposiciones y la distribución.

```
1 sudo ntpdate pool.ntp.org
2 sudo apt-get update
3 sudo apt-get dist-upgrade
4 sudo apt-get upgrade
```

Una vez aquí, ya se dispone de lo necesario para poder instalar el paquete serial siguiendo los pasos marcados, [2] Liechti (2001).

Por último hay que activar los puertos UART que se quieran utilizar, en este caso, el puerto UART2. Se accede al archivo /boot/uboot/uEnv.txt y se añade capemgr.enable_partno=BB-UART2.

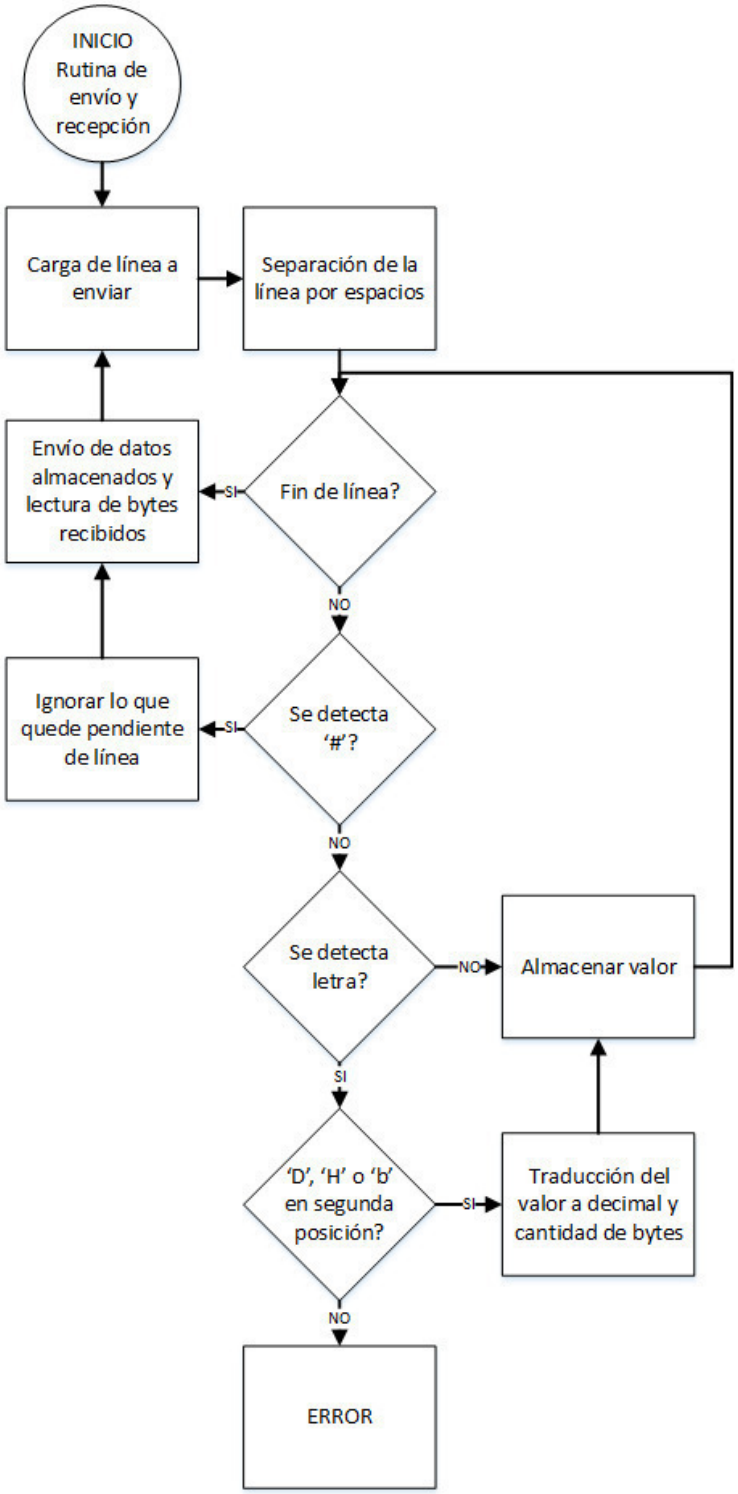


Figura 2.1: Diagrama de flujo de la rutina de envío y recepción

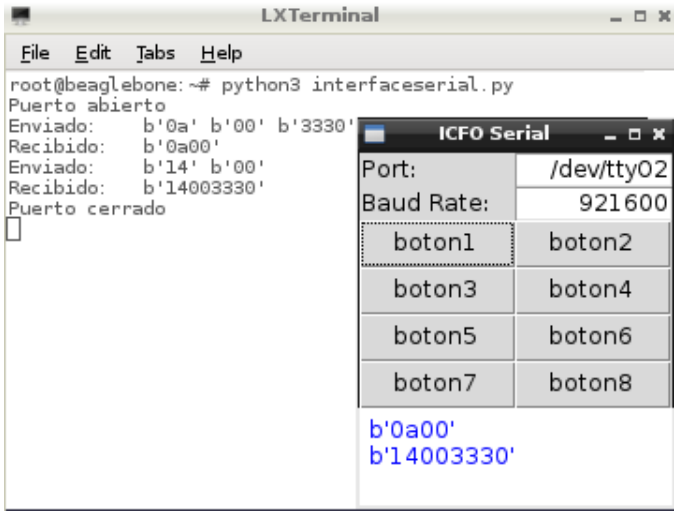


Figura 2.2: Interfaz ICFOserial

2.5. Placa de adaptación

Para que BeagleBone Black y FPGA sean cien por cien funcionales entre ellas y poder integrar el miniordenador en el ICFORACK, se diseña una placa de circuito impreso de tamaño Eurocard, la cual es posible conectar con la placa de circuito impreso que contenga la FPGA, quedando apilada una con otra.

Esta placa ha de disponer en el frontal de dos botones, uno para el encendido/apagado y otro para el reset del miniordenador. Debe contar con alimentación por parte del backplane y gracias a un conversor DC/DC se alimenta el miniordenador.

En un lateral de la placa se añade un conector PC104 de 32x2 que permite la comunicación con otras placas. Se elige conectar los pines de la BeagleBone Black que aporten más opciones de configuración, por ejemplo aquellos que dispongan de timers, PWM, etc.

En el anexo B se puede encontrar el esquemático y el layout de esta placa de adaptación.

Capítulo 3

Conversores AD y DA

3.1. Analógico-Digital

3.1.1. Requisitos

Según las necesidades del taller electrónico se determinan los siguientes requisitos para el conversor AD:

- 16 bits de resolución, de los cuales 14 han de ser útiles.
- Entre 0.8 MHz y 1.2 MHz.
- 1 canal.
- Tipo de entrada "single-ended" o pseudodiferencial.
- Margen de entrada de 0 V a 5 V.
- Interfaz serie.
- Posibilidad de referencia externa.
- Placa de evaluación.
- Encapsulado con pines expuestos para soldadura manual.
- Disponibilidad.

3.1.2. Estudio comparativo y selección

En la tabla 3.1 se muestran únicamente los conversores que cumplen con todos los requisitos mencionados, mostrando en la tabla aquellas características que difieren entre los conversores.

#	MSPS	Encapsulado	Precio
AD7980	1	MSOP-10	18.40 €
AD7983	1.33	MSOP-10	20.19 €
ADS8860	1	VSSOP-10	16.99 €

Tabla 3.1: Selección ADC

Finalmente, se elige el `AD7983`, ya que ofrece algo más de velocidad para no tener que trabajar al límite del conversor y poder alcanzar la velocidad de 1 millón de muestras por segundo. El resto de características son muy similares.

3.1.3. Placa de evaluación

Para evaluar las características y conocer el funcionamiento del conversor se adquiere una de las dos placas de evaluación disponibles, [4] Placa de evaluación ADC. Se elige un kit que contiene una placa que se asemeja mucho a lo que se busca conseguir en nuestro diseño, pudiendo utilizarse la placa con nuestra propia FPGA, y una DSP para la comunicación con el PC pudiendo usar un software específico.

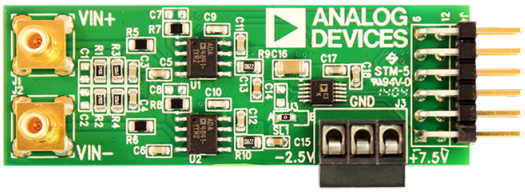


Figura 3.1: Placa de evaluación conversor analógico-digital

Para empezar, se debe realizar el montaje siguiendo el tutorial de la web. El fabricante ofrece una aplicación que permite analizar las señales que convierte el conversor, pudiendo ver así el comportamiento y características del conversor. Una vez se consigue conectar con la aplicación, se realizan distintos análisis para obtener las características que ofrece la placa que contiene el conversor.

En la figura 3.2 se muestra la lectura del conversor en un histograma durante un corto periodo de tiempo cuando en el conector SMA hay un circuito abierto. Según la configuración del front-end se tienen 2.5 V de tensión continua, donde, con 16 bits de resolución del conversor supone 32 768 cuentas, tal y como se puede comprobar en la siguiente ecuación, donde M_{ADC} es el valor digital obtenido, V_{ADC} la tensión de entrada y $V_{Ref_{ADC}}$ la tensión de referencia externa.

$$M_{ADC} = \left(\frac{2^{16}}{V_{Ref_{ADC}}} \right) \cdot V_{ADC}$$

Se puede observar que se tiene una pequeña desviación de 13 cuentas, pudiendo comprobar con el multímetro que la tensión en el pin de entrada del conversor no es exactamente 2.5 V, sino algo menor debido a que para

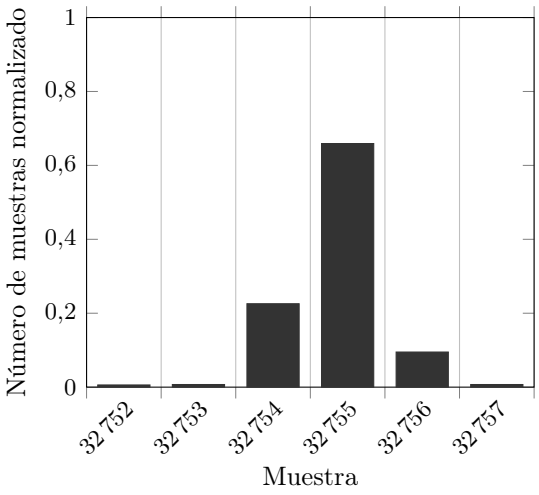


Figura 3.2: Representación cantidad de muestras

obtener este voltaje se hace un divisor de tensión a partir de la referencia externa con dos resistencias de 1 kΩ que tienen una tolerancia del 1 %, y una pequeña dispersión de valores aunque la gran mayoría concentrados en la cuenta 32 755.

3.2. Digital-Analógico

3.2.1. Requisitos

Según las necesidades del taller electrónico se determinan los siguientes requisitos para el conversor DA:

- 16 bits de resolución, de los cuales 14 han de ser útiles.
- Tasa de muestreo entre 0.8 MHz y 1.2 MHz.
- 1 canal.
- Interfaz serie.
- Margen de salida de 0 V a 5 V.
- Posibilidad de referencia externa.
- Placa de evaluación disponible.
- Encapsulado pequeño con pines expuestos para soldadura manual.
- Disponibilidad.

3.2.2. Estudio comparativo y selección

Para el caso del conversor digital-analógico se decide buscar un conversor de características similares al analógico-digital, para así facilitar el trabajo

a la hora de hacer el diseño VHDL, ya que los bloques usados para un conversor, deberían servir para el otro.

En la tabla 3.2 se muestran únicamente los conversores que cumplen con todos los requisitos mencionados, mostrando en la tabla aquellas características que difieren entre los conversores.

#	MSPS	Encapsulado	Precio
AD5061	1.3	SOT-23-8	8.82 €
AD5541	1.5	SOIC-8	18.34 €
DAC8830	1	SOIC-8	14.18 €

Tabla 3.2: Selección DAC

Finalmente, se escoge el `AD5061`, ya que dispone de un buffer a la salida que permite poner cargas menores que en los otros conversores y además su placa de evaluación es compatible con la DSP utilizada con la placa de evaluación del ADC.

3.2.3. Placa de evaluación

Para el DAC se escoge la siguiente placa de evaluación, [5] Placa de evaluación DAC, la cual se puede utilizar con la DSP que se muestra en la figura 3.3 con un software que permite escribir un valor en hexadecimal para comprobar que concuerda con la tensión mostrada en la salida del conversor, o utilizar directamente la placa con nuestra FPGA pudiendo comprobar el funcionamiento del VHDL desarrollado sabiendo que la placa funciona correctamente y solo pudiendo tener errores en nuestra descripción de circuito VHDL.

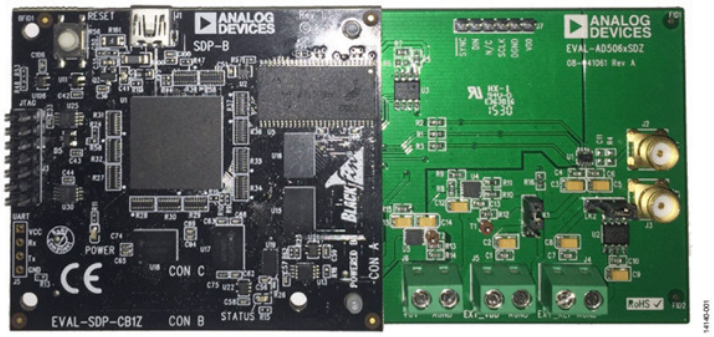


Figura 3.3: Kit de evaluación conversor digital-analógico

Capítulo 4

Diseño VHDL

4.1. Introducción al sistema

El objetivo de este capítulo es diseñar el bloque de comunicaciones y control de los conversores. La FPGA utilizada en el taller electrónico dispone de un hardware el cual permite la lectura y escritura de distintos registros de 16bits, siendo estos almacenados en el `Master Registers`. Hay que hacer la descripción de circuito en VHDL enlazando el bloque mencionado con el que se diseñe para que permita la comunicación SPI con los conversores, disponiendo de los distintos registros para utilizarlos como inputs y outputs de dichos bloques. En la figura 4.1 se muestra un diagrama del sistema, incluida la comunicación entre BeagleBone y FPGA, donde la parte interna recuadrada de la FPGA será la que se implementará en este capítulo, a excepción del protocolo SPI, [6] Larson (2010).

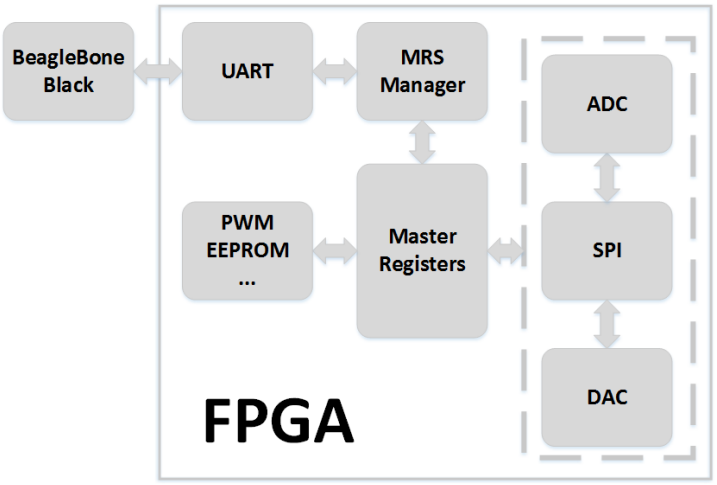


Figura 4.1: Diagrama del sistema IOBOX

4.2. Descripción de circuitos

Para el diseño que se va a realizar se estudia el datasheet de los conversores para así saber las señales a enviar y recibir.

Bloque Analógico-Digital

En el caso del ADC, se necesita generar una señal *cnv* donde será de nivel alto mientras se convierta la señal analógica y de nivel bajo mientras se transmitan los 16 bits de señal digital por parte del conversor. Según especificaciones el tiempo mínimo de nivel alto deberá ser de 500 ns y el de nivel bajo de mínimo 250 ns. Para cada conversión deseada se debe generar esta señal dando en el flanco de bajada de *cnv* el enable para dar comienzo al envío del clock del spi.

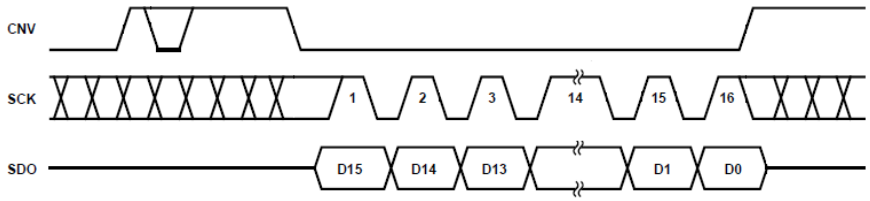


Figura 4.2: Timing de señales del ADC

Por lo tanto, se debe generar una señal *sck* en el tiempo de adquisición de 16 periodos, donde cada periodo tiene una duración de mínimo 9 ns hasta 16 ns dependiendo de VIO. Este clock se genera a partir de divisiones por 2 del clock de 200 MHz de la FPGA, por lo que se usará un *sck* de 50 MHz, equivalente a un tiempo de periodo de 20 ns, y por tanto hay que ampliar el tiempo de adquisición hasta 320 ns. Finalmente, sumando los 500 ns de conversión más los 320 ns dan un tiempo de 820 ns, con lo que se consiguen 1.22 MHz de frecuencia de muestreo.

Una vez teniendo claro el funcionamiento del conversor, se dibuja un diagrama de bloques funcional de como ha de ser el VHDL para que FPGA y ADC trabajen correctamente. En la figura 4.3 se representa la versión final del diagrama de bloques.

Para realizar la descripción de circuito que genera la señal *cnv* se crea una máquina de estados. En la figura 4.4 se muestra la máquina de estados empleada.

Por último se comprueba mediante el osciloscopio que la comunicación SPI funciona correctamente, consiguiendo una frecuencia de muestreo de 1.18 MHz, tal y como se observa en la figura 4.5.

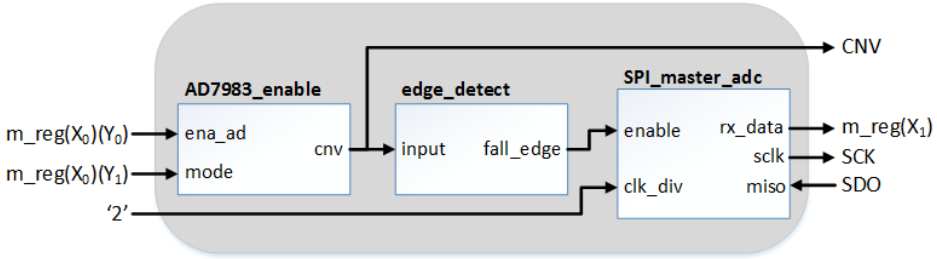


Figura 4.3: Bloques VHDL para el ADC

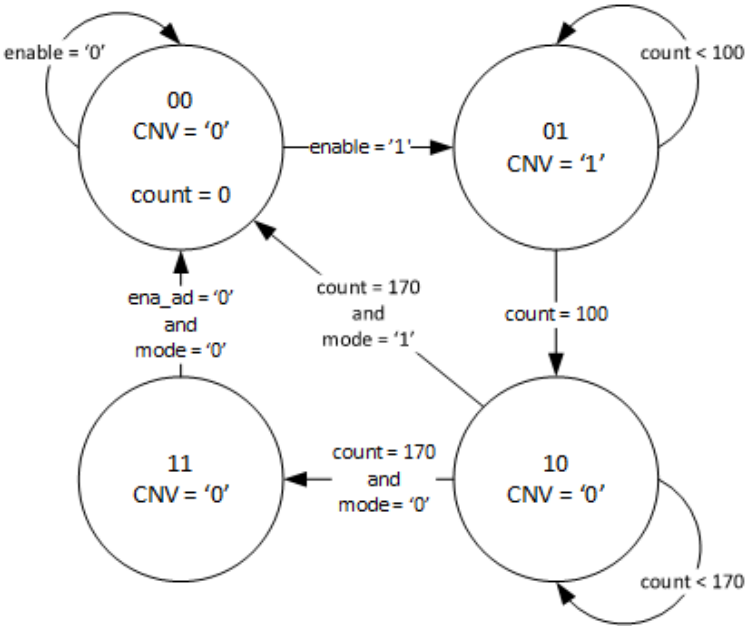


Figura 4.4: Máquina de estados para el bloque AD7983_enable

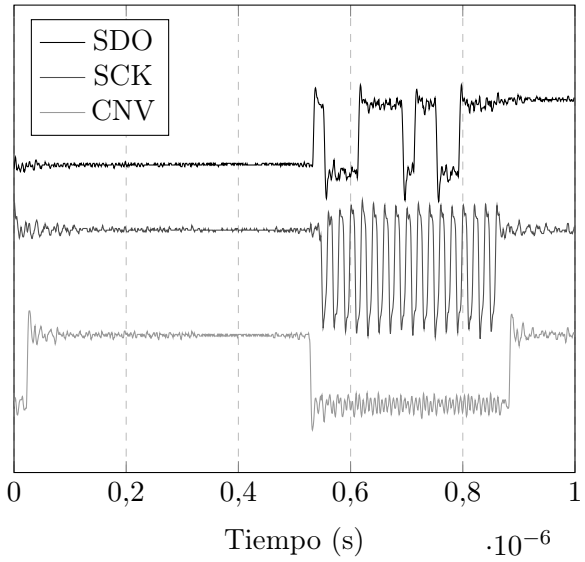


Figura 4.5: Señales ADC capturadas por el osciloscopio

Bloque Digital-Analógico

Para el caso del DAC, se necesita generar una señal *sync* donde será de nivel bajo mientras se transmiten los bits al conversor, y de nivel alto cuando se tendrá finalmente el valor de tensión analógico disponible. Según especificaciones el tiempo mínimo de nivel alto deberá ser de 12 ns y el de nivel bajo teniendo en cuenta que cada clock de los 24 a enviar ha de durar mínimo 33 ns, se necesitarán 792 ns. Para cada conversión deseada, se generará esta señal dando en el flanco de bajada de *sync* el enable para dar comienzo al envío del clock del spi.

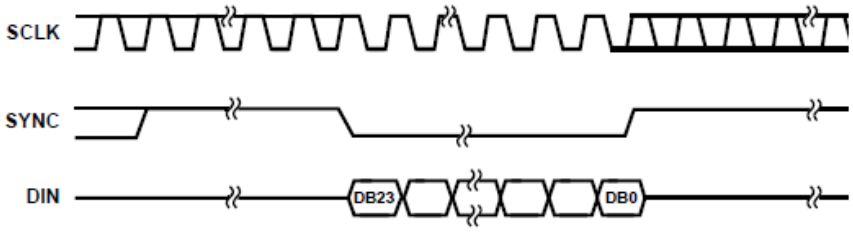


Figura 4.6: Timing de señales del DAC

La FPGA debe generar una señal *sck* en el tiempo de envío al DAC de 24 periodos, en 8 clocks se envía el tipo de modo y en los 16 clocks restantes los bits de información, donde cada periodo ha de tener una duración mínima de 33 ns. Este clock se genera a partir de divisiones por 2 del clock de 200 MHz de la FPGA, por lo que se usará un *sck* de 25 MHz, equivalente a un tiempo

de periodo de 40 ns, y por tanto hay que ampliar el tiempo de adquisición hasta 960 ns. Finalmente, sumando los 15 ns más los 960 ns dan un tiempo de 975 ns, con lo que se consiguen 1.02 MHz de frecuencia de muestreo.

Una vez teniendo claro el funcionamiento del conversor, se dibuja un diagrama de bloques funcional de como ha de ser el VHDL para que FPGA y DAC trabajen correctamente. En la figura 4.7 se representa la versión final del diagrama de bloques.

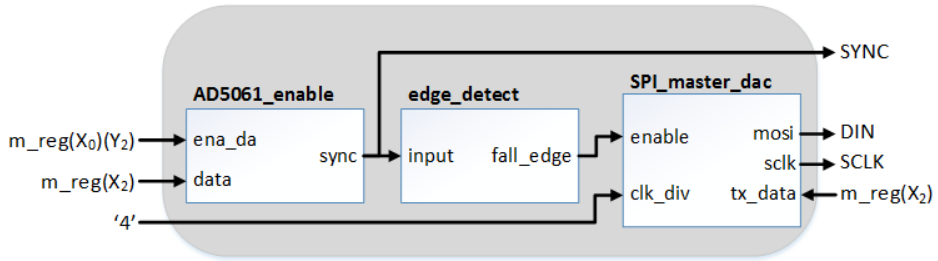


Figura 4.7: Bloques VHDL para el DAC

Para realizar la descripción de circuito que genera la señal *sync* se crea una máquina de estados. En la figura 4.4 se muestra la máquina de estados empleada en el código final.

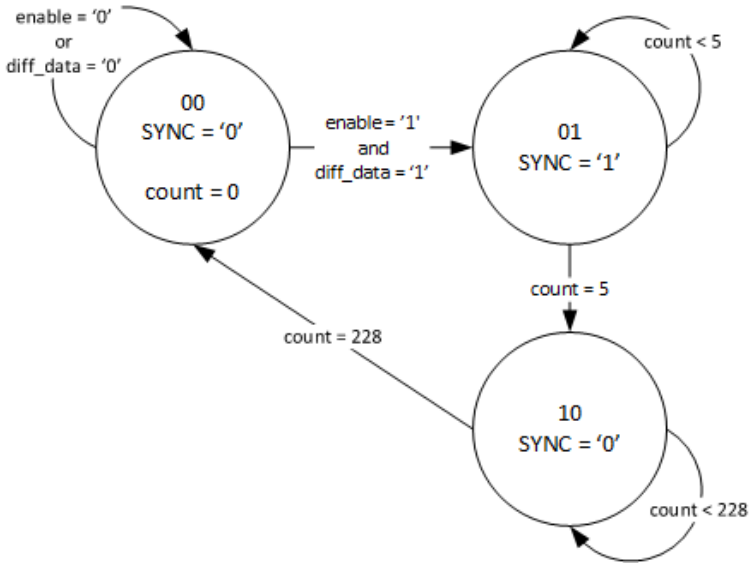


Figura 4.8: Máquina de estados para el bloque AD5061_enable

Añadir que una vez funcionando el código VHDL, al poner una tensión contínua a la salida del DAC, se observa que hay una pequeña desviación de pocos centenares de µV. Para solucionar el problema, se modifica el código VHDL del enable para el conversor haciendo que únicamente se comunique

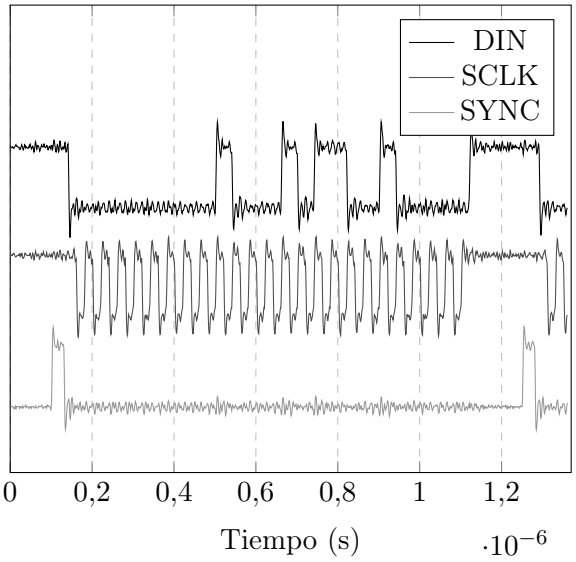


Figura 4.9: Señales DAC capturadas por el osciloscopio

por SPI cuando haya un valor diferente a enviar y así mantenga el valor de tensión marcado si no hay cambios, dando más tiempo al conversor a estabilizar dicha tensión.

Además, se observa el problema que el conversor desprecia un envío de cada dos ya que cuando empieza el segundo envío aún no ha estabilizado la señal anterior, por lo que para solucionar el problema se reduce la velocidad de la señal *sync* a un tiempo de 1.14 µs, lo que supone una frecuencia de muestreo de 871 kHz. Por último se comprueba mediante el osciloscopio que la comunicación SPI funciona correctamente, tal y como se observa en la figura 4.5.

Capítulo 5

Diseño de esquemático

5.1. Introducción

Para que el diseño sea compatible con el ICFORACK se trabaja con una placa de tamaño eurocard, pero se verá reducido ya que se ha de contar con la FPGA que va conectada mediante conectores a nuestra placa y los conectores ya dispuestos del puerto trasero, donde se obtiene la alimentación de 15 V y −15 V que se debe regular para trabajar con las tensiones adecuadas. En la figura 5.1 se puede observar la disposición de la placa, quedando disponible el espacio donde se nombra a los conversores para situar los componentes necesarios para el funcionamiento de éstos.

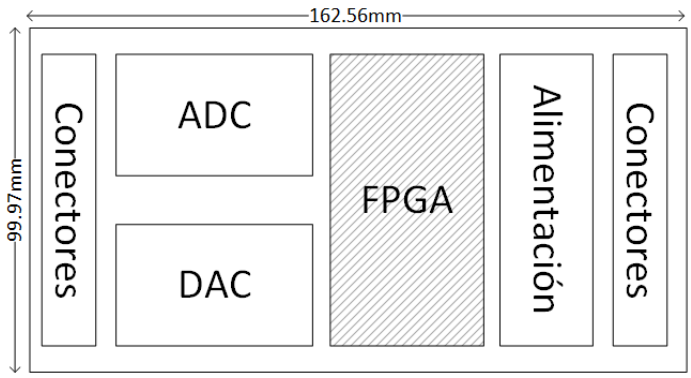


Figura 5.1: Esquema del circuito impreso

También hay que tener en cuenta la parte que ocupará la alimentación para los componentes utilizados y los conectores para las entradas/salidas analógicas al extremo de la placa que irán al frontal del cassette.

Para el diseño del circuito impreso se opta por hacer el diseño en cuatro capas, permitiendo así llevar por las caras intermedias alimentaciones y masa, y distinguiendo claramente partes analógicas de partes digitales para evitar

problemas de ruido.

A continuación, se explica el diseño para el esquemático y el diseño de circuito impreso. El esquemático completo del circuito y layouts se encuentran en el anexo D.

5.2. Conversor Analógico-Digital

A continuación, se explica la elección de algunos de los íntegrados y componentes utilizados en el diseño para el uso del conversor AD.

A la hora de elegir la referencia externa para el conversor, se debe buscar una referencia que tenga una tensión de ruido menor al equivalente en tensión de una cuenta. Para una tensión de referencia, $V_{Ref_{ADC}}$, de 5 V, una cuenta equivale a 76.3 µV, lo que lleva a la elección del ADR445B que tiene una tensión de ruido de 2.24 µV, bastante por debajo del valor deseado, y una precisión de 2 mV sobre los 5 V. Además, antes de tener la tensión de referencia en el conversor, según datasheet, se debe haber alimentado primero el conversor, por lo que se usa el íntegrado ADG1401, forzando a que no se tenga la tensión de referencia en el pin del conversor hasta que éste no se haya alimentado a los 2.5 V de la tensión de alimentación. En la figura 5.2 se observa el circuito propuesto.

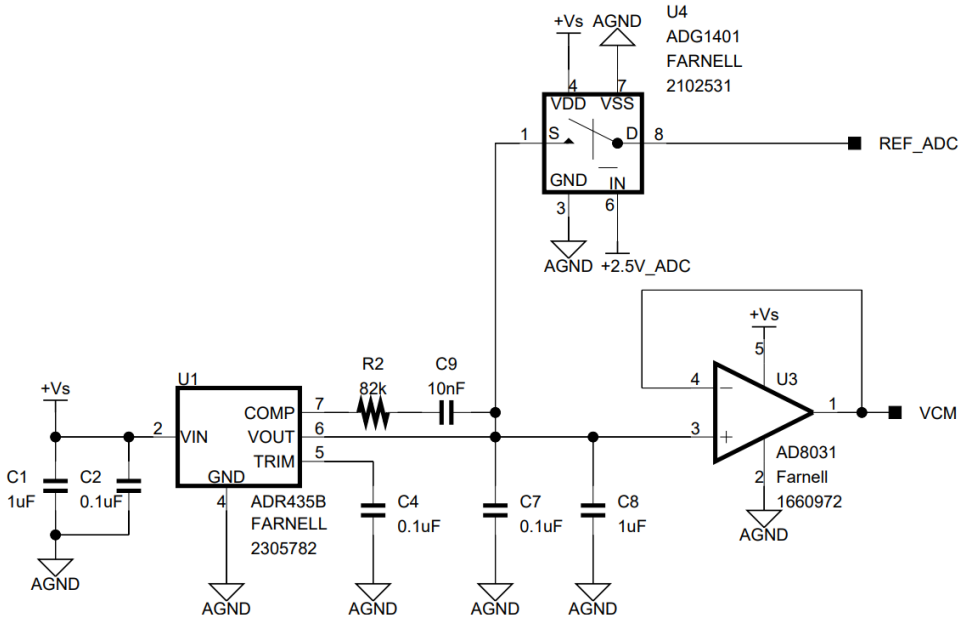


Figura 5.2: Diseño de la referencia del conversor analógico-digital

Finalmente, se ha de disponer en el front-end de dos tipos de configuración, por lo que dependiendo de los componentes que se suelden en la figura

5.3, una resistencia de $0\,\Omega$ en C_{18} y sin soldar R_4 y R_5 o un condensador de $1\,\mu F$ y soldando las dos resistencias mencionadas, se puede introducir una señal continua entre $0\,V$ y $5\,V$ o una alterna entre $-2.5\,V$ y $2.5\,V$. Para evitar tener que elegir una configuración u otra se decide incorporar dos conversores analógico-digital al diseño.

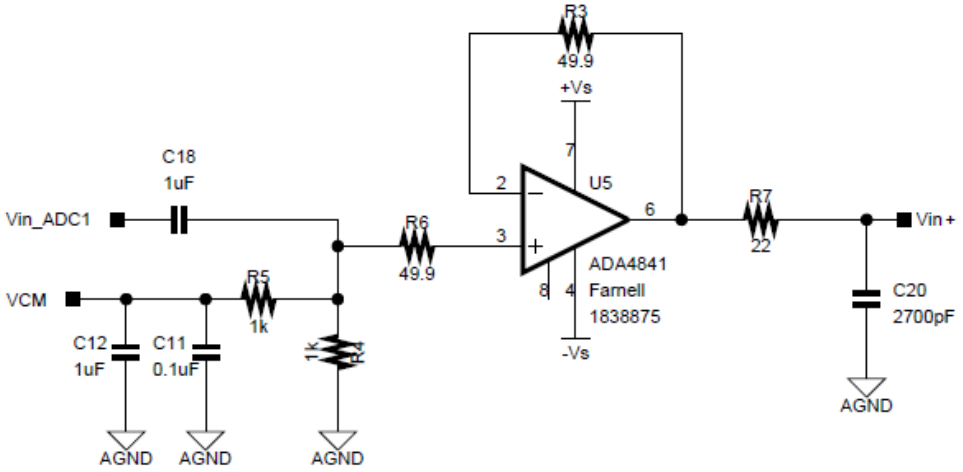


Figura 5.3: Front-end del conversor analógico-digital

5.3. Conversor Digital-Analógico

El diseño del esquemático para hacer funcionar el DAC necesita únicamente buscar una tensión de referencia, $V_{Ref_{DAC}}$, de 5 V, ya que en este caso no se dice en datasheet que necesite ser alimentado antes que dar la tensión de referencia. Por otro lado, se utiliza el mismo integrado, ADR445B, que se ha buscado para el ADC.

Como este diseño ocupa poco espacio se decide incluir hasta tres DAC para que dependiendo de la aplicación se puedan soldar los conversores necesarios, evitando así tener que utilizar más de una placa en caso de necesitar más de un conversor.

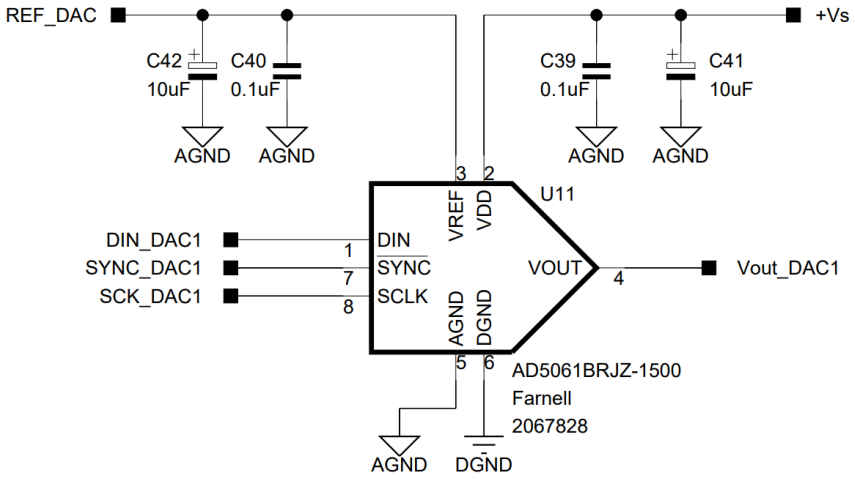


Figura 5.4: Diseño de esquemático del conversor digital-analógico

Capítulo 6

Medidas

Para comprobar el correcto funcionamiento del sistema y con la finalidad de comparar con las especificaciones del proyecto y las de los fabricantes de los dispositivos se elabora un banco de test con la disposición de la figura 6.1.

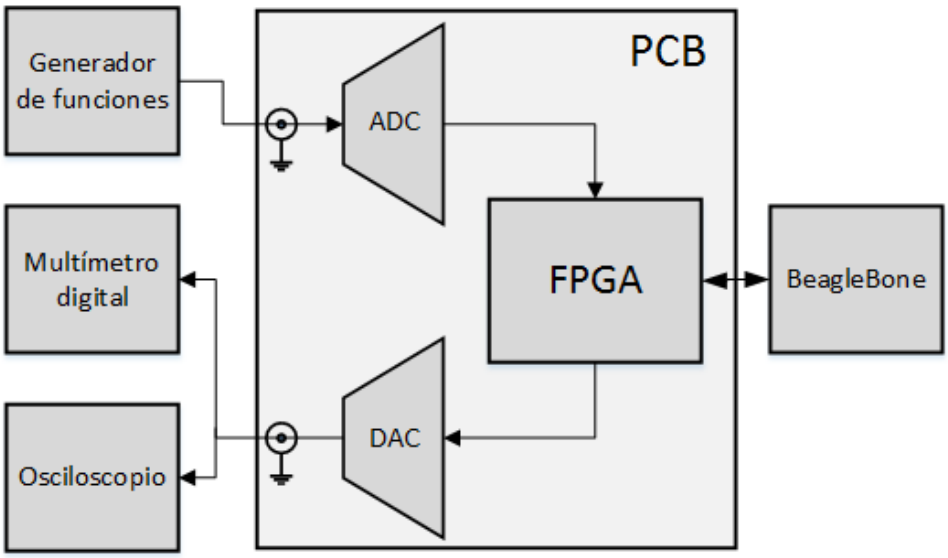


Figura 6.1: Setup para medidas

Donde los instrumentos utilizados son los siguientes:
- Generador de funciones: Tabor Electronics WW1281A.
- Multímetro digital: KEITHLEY 2100.
- Osciloscopio: Agilent Technologies DSO-X 3034A.

6.1. Conversor Analógico-Digital

Para comprobar la dispersión del conversor AD al convertir un valor constante se necesita una fuente de bajo ruido. Para ello se utiliza la misma tensión de referencia utilizada para el conversor aplicando un divisor de tensión con dos resistencias de 1 kΩ y se recoge la lectura del conversor en un registro de la FPGA. En la figura 6.2 se muestra el resultado de leer un total de 1 000 valores.

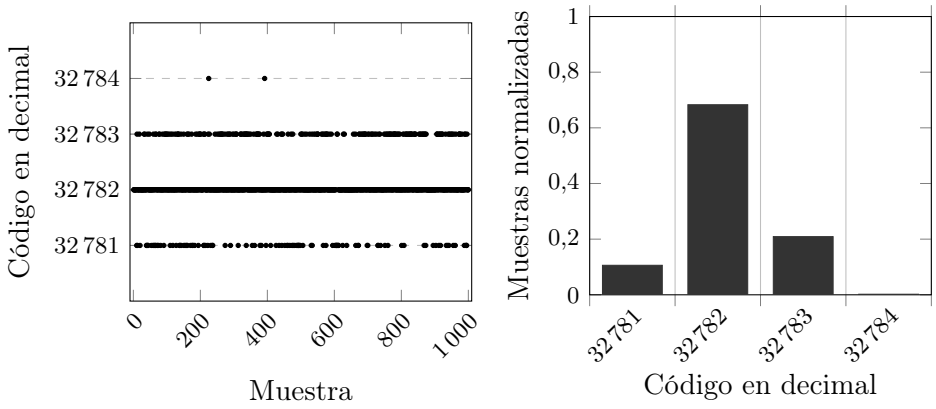


Figura 6.2: Tensión continua aplicada al ADC

Se observa que la desviación máxima del valor central es de 2 cuentas, correspondiéndose con las especificaciones marcadas por el fabricante. Cabe mencionar que el valor central, 32 782, es 16 cuentas mayor que el valor de media escala, 32 768. Esta desviación se debe a que las resistencias utilizadas para hacer el divisor de tensión tienen una tolerancia del 1 %, lo que supondrá hasta ±328 cuentas en el peor caso posible.

6.2. Conversor Digital-Analógico

Para verificar las características del DAC se utiliza el multímetro digital conectado a la salida del DAC mediante un cable coaxial RG174 para comprobar distintos valores de tensión. Los valores de tensión obtenidos se comparan con el valor de tensión ideal teniendo en cuenta el valor de $V_{Ref_{DAC}}$ medido, cuya tensión en este caso es de 5.0005 V.

Error de cero

El error de cero es el error en la salida cuando en el registro del conversor se carga el valor 0x0000. Idealmente, la salida debe ser 0 V. Este error es siempre positivo porque la salida no puede estar por debajo de 0 V y es debido a la combinación del error de offset en el conversor y el amplificador de salida interno. El error se muestra en la figura 6.3 expresado en µV.

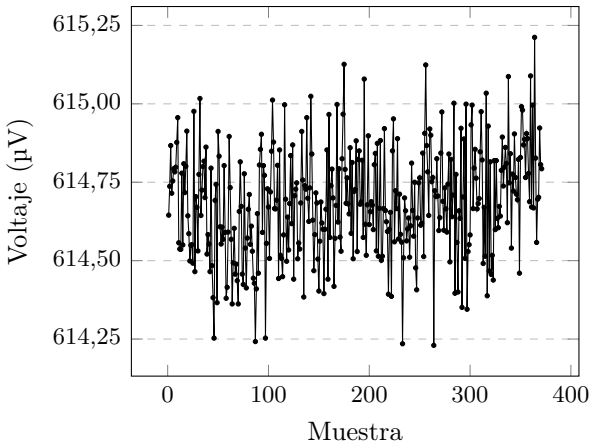


Figura 6.3: Error de cero

Se puede observar un error de cero de 614 µV. No se puede comparar este error ya que no hay medidas por parte del fabricante.

Error de final de escala

El error de final de escala es la medida de error de la salida cuando el código 0xFFFF se carga en el registro del conversor. Idealmente, la salida debería ser $V_{Ref_{DAC}} - 1LSB$. El error se muestra en la figura 6.4 representado en tanto por ciento respecto el rango de fin de escala.

Se puede observar un error de fin de escala del 0.134 % respecto el fin de escala medido. No se puede comparar este error ya que no hay medidas por parte del fabricante.

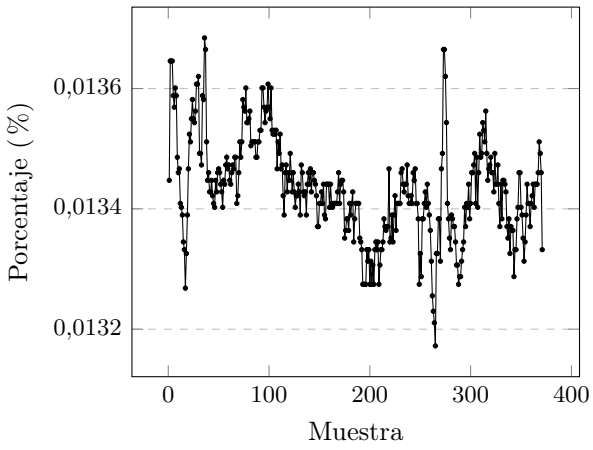


Figura 6.4: Error de fin de escala

Lectura de tensión continua

En la figura 6.5 se observa le lectura de una tensión DC generada por el DAC cuando se le envía el código decimal 32 768.

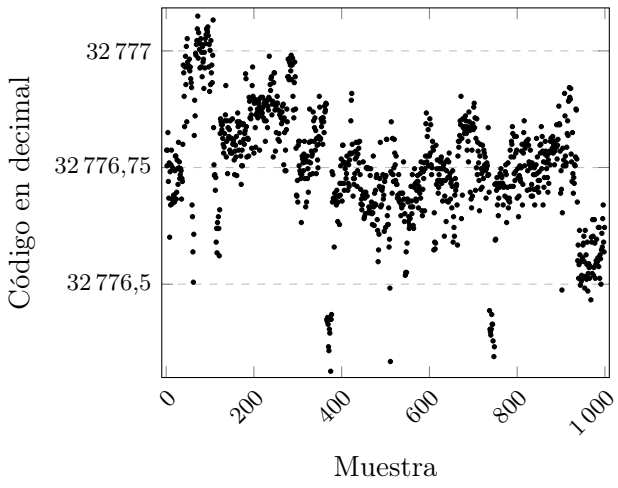


Figura 6.5: Tensión DC generada por el conversor digital-analógico

Como se puede apreciar esta tensión es poco ruidosa y tiene un offset de 8 cuentas respecto al valor que se debía obtener. Esto puede ser debido a un error del instrumento de medida.

6.3. Combinación de conversores

Con el fin de comprobar el comportamiento dinámico de ambos conversores, se realiza una medida utilizando los dos conversores simultáneamente.

En la figura 6.6 se observa la señal cuadrada generada por el generador de funciones a digitalizar por el conversor y los datos obtenidos se envían posteriormente por el DAC. Se puede observar que el slew-rate del DAC es de 1 V/µs tal y como se especifica en el datasheet.

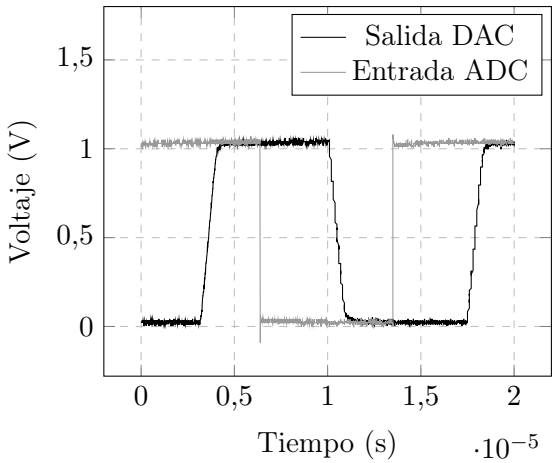


Figura 6.6: Slew-Rate del conversor digital-analógico

En la figura 6.7 se muestra la digitalización de una señal triangular y después muestreada por el DAC donde se puede observar los saltos de tensión por parte del DAC, siendo éstos de un tiempo de 1.14 µs.

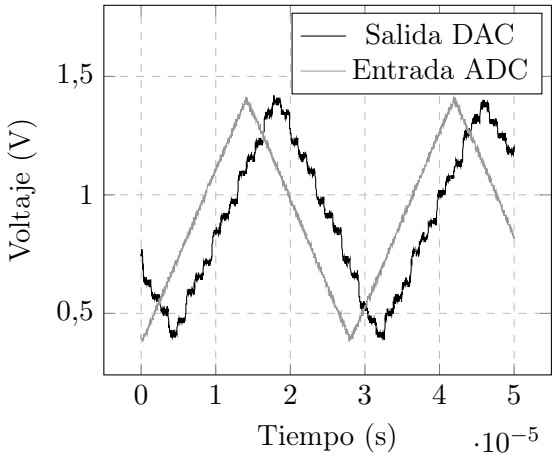


Figura 6.7: Captura y generación de señal triangular

En la figura 6.8 se muestra la lectura del ADC realizando con el DAC un barrido con un incremento de 100 cuentas y tomando 10 muestras de cada valor. En la figura de la izquierda se muestra la diferencia máxima entre las diez muestras viendo así la precisión. Por otro lado, la figura de la derecha muestra la diferencia entre el valor ideal, teniendo en cuenta el valor de $V_{Ref_{DAC}}$ y de $V_{Ref_{ADC}}$, y la exactitud del valor promediado.

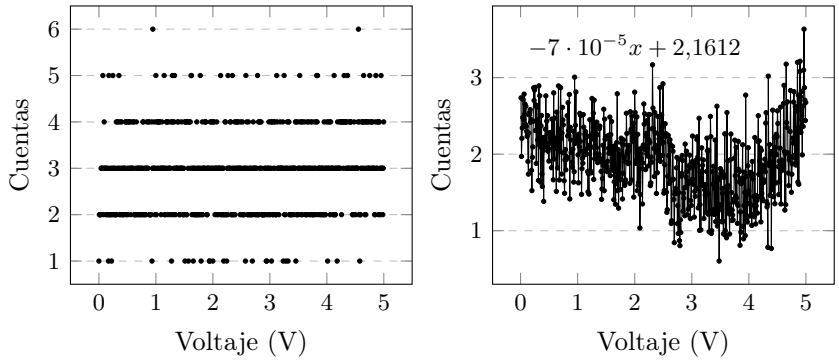


Figura 6.8: Generación y captura de señal DC

En la figura 6.9 se muestran los datos obtenidos al escribir en el registro del DAC el valor decimal 32 768 y recoger los datos con el ADC uniendo los conectores SMA de salida y entrada de los dos conversores. Se puede observar como el valor 32 777 es el más obtenido, suponiendo esto una desviación de nueve cuentas respecto al valor esperado.

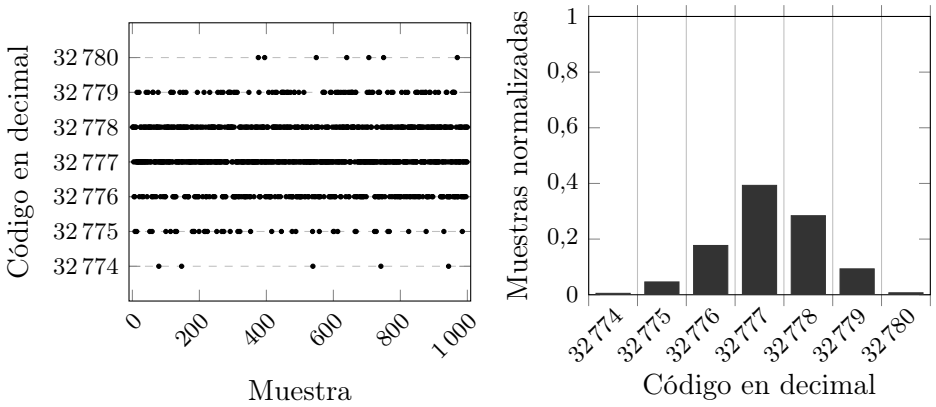


Figura 6.9: Generación y captura de señal DC

Se puede demostrar el desvío de estas cuentas mediante la siguiente ecuación, donde sabiendo que al tomar las medidas, las tensiones de referencia son $V_{Ref_{ADC}} = 4.9988\,\mathrm{V}$ y $V_{Ref_{DAC}} = 5.0005\,\mathrm{V}$.

$$M_{ADC} - M_{DAC} = 2^N \cdot V_{DAC}(M_{DAC}) \cdot \left[\frac{V_{Ref_{DAC}} - V_{Ref_{ADC}}}{V_{Ref_{ADC}} \cdot V_{Ref_{DAC}}} \right]$$

Esta diferencia da como resultado un error de once cuentas. Además, como en la figura 6.8 se ha demostrado, el valor ideal en media escala es dos cuentas mayor al valor medido. Por lo que finalmente cuadra las nueve cuentas de diferencia entre el código introducido en el DAC y el ADC.

Capítulo 7

Conclusiones y mejoras

7.1. Conclusiones

Se realizó un sistema de generación y adquisición de señales implementado utilizando el conversor digital-analógico AD5061 (Analog Devices) y el conversor analógico-digital AD7983 (Analog Devices) configurados desde el módulo de FPGA XC7A35T-2CSG324C (Trenz Electronic) mediante el hardware IOBOX utilizado por el taller electrónico del ICFO. La comunicación con la FPGA se establece por medio del miniordenador Beaglebone Black a través de la interfaz ICFOserial implementada en Python3. El diseño de la placa de circuito impreso incluye conectores para el módulo de la FPGA y es compatible con el sistema de rack modular ICFORACK.

Las medidas realizadas confirman el correcto funcionamiento del sistema, cumpliéndose las especificaciones planteadas al inicio del proyecto. Las frecuencias de muestreo logradas son de 871 kHz para el DAC y 1.18 MHz para el ADC. Las medidas realizadas con el ADC demuestran una resolución de 14 bits efectivos, mientras que el DAC presenta una resolución de 15 bits efectivos. En las medidas del DAC aparece un pequeño offset (610 µV) que podría deberse a un error en el instrumento de medida.

Estos resultados hacen prever que el sistema podrá ser utilizado para aplicaciones de procesamiento digital de señales valiéndose del microprocesador MicroBlaze a implementar en la FPGA. De esta forma será posible realizar digitalmente algunas funciones que actualmente se realizan el dominio analógico, tales como: controladores, PID, filtros digitales, correladores, operaciones matemáticas, etc.

7.2. Mejoras a futuro

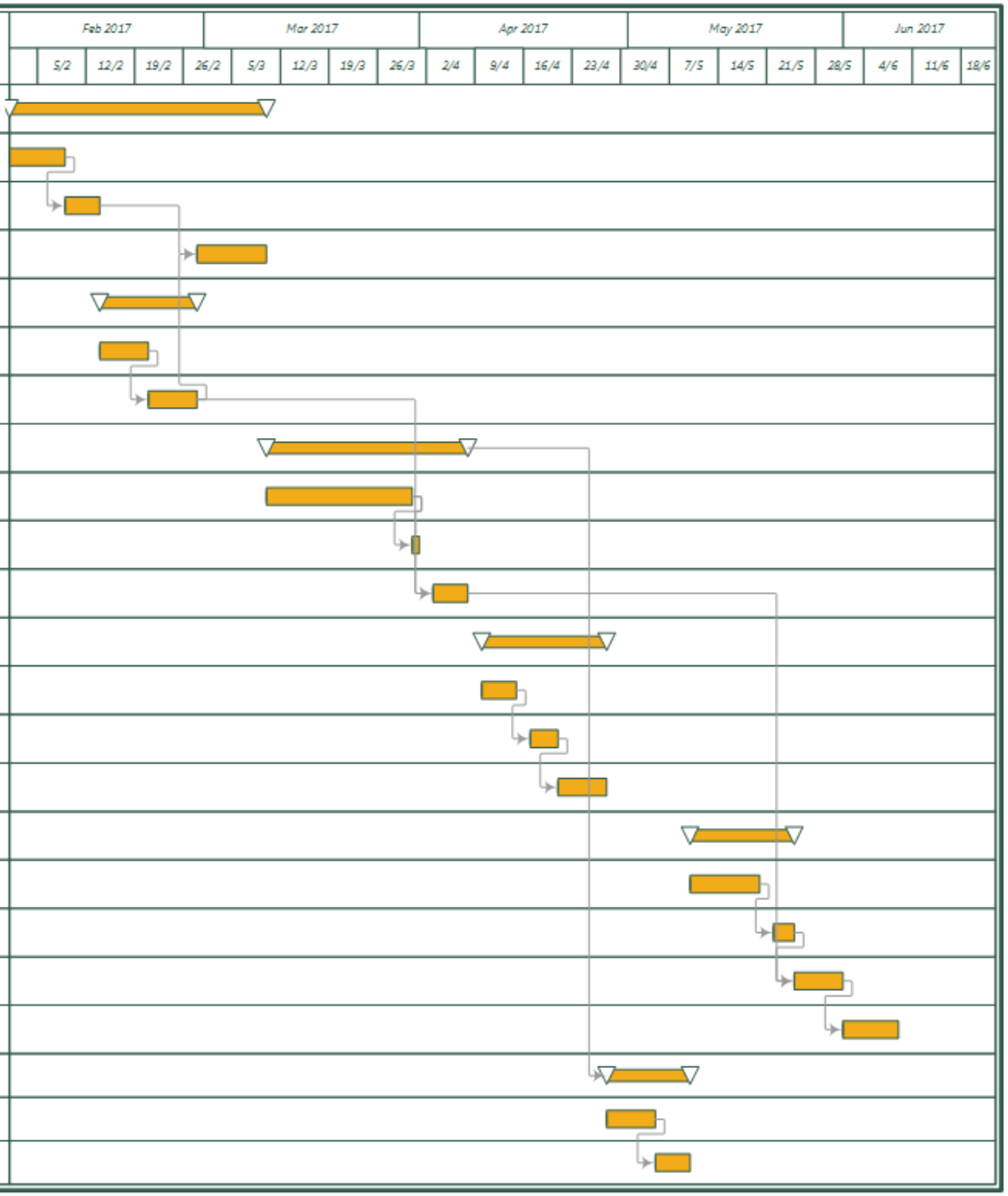
- Como se vio en el apartado de medidas, las referencias tienen una pequeña desviación de tensión respecto su valor teórico. Para poder ajustar este valor, las referencias disponen de un trimmer el cual no se ha utilizado hasta el momento, por lo que sería algo a tener en cuenta en futuras versiones.

- Además hay que añadir la posibilidad de disponer de otro clock a otra frecuencia, ya que el clock del SPI funciona con divisores de la frecuencia de clock, por lo que no se ha podido alcanzar en el conversor digital-analógico la frecuencia de muestreo deseada de un millón de muestras por segundo.

Apéndice A

Diagrama de Gantt

ID	Task Name	Start	Finish
1	**Conversores**	**01/02/2017**	**09/03/2017**
2	Tabla y elección	01/02/2017	08/02/2017
3	Estudio datasheets	09/02/2017	13/02/2017
4	Test kits	28/02/2017	09/03/2017
5	**Firmware**	**14/02/2017**	**27/02/2017**
6	Funcionamiento IOBOX	14/02/2017	20/02/2017
7	Diseño VHDL	21/02/2017	27/02/2017
8	**PCB Conversores**	**10/03/2017**	**07/04/2017**
9	Diseño	10/03/2017	30/03/2017
10	Costes	31/03/2017	31/03/2017
11	Test	03/04/2017	07/04/2017
12	**Minicomputer**	**10/04/2017**	**27/04/2017**
13	Tabla y elección	10/04/2017	14/04/2017
14	Configuración	17/04/2017	20/04/2017
15	Interface	21/04/2017	27/04/2017
16	**PCB Minicomputer**	**10/05/2017**	**24/05/2017**
17	Diseño	10/05/2017	19/05/2017
18	Test	22/05/2017	24/05/2017
19	**Test final**	**25/05/2017**	**31/05/2017**
20	**Apps**	**01/06/2017**	**08/06/2017**
21	**PCB Conversores 2**	**28/04/2017**	**09/05/2017**
22	Diseño	28/04/2017	04/05/2017
23	Test	05/05/2017	09/05/2017



Apéndice B

Miniordenador

ICFOserial

Interface

Esquemático de placa de adaptación

Layout de placa de adaptación

Costes

ICFOserial

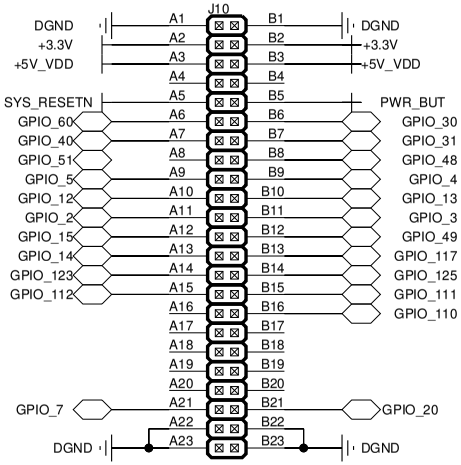
```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
-- Company: ICFO
-- Engineer: Sergio López Casablanca
-- Revision 2.0
"""

import os
import sys
import serial
import binascii

def func_send(input_send,ser):
    try:
        ser.read() #Comprobar que el puerto esta abierto.
    except:
        print("Error, el puerto no esta abierto")
        return b''
    else:
        r=b''
        send = b''
        input_send = input_send.split()
        for i in input_send: #Discernir dec, hexa, bin
            if '#' in i: break
            elif len(i)==1:
                x=int(i).to_bytes(1,'big')
                send=send+x
            elif 'D' in i[1]:
                sub_send=i.split("D")
                x=int(sub_send[1]).to_bytes(int(sub_send[0]),'big')
                send=send+x
            elif 'H' in i[1]:
                sub_send=i.split("H")
                x=int(sub_send[1],16).to_bytes(int(sub_send[0]),'big')
                send=send+x
            elif 'b' in i[1]:
                sub_send=i.split("b")
                x=int(sub_send[1],2).to_bytes(int(sub_send[0]),'big')
                send=send+x
            else:
                if int(i) > 65535:
                    x=int(i).to_bytes(3,'big')
                    send=send+x
                elif int(i) > 255:
                    x=int(i).to_bytes(2,'big')
                    send=send+x
                else:
                    x=int(i).to_bytes(1,'big')
                    send=send+x
        if(send!=b''):
            ser.write(send)
            r=ser.read(10000)
            print("\nRecibido:  ",binascii.b2a_hex(r))
        return str(binascii.b2a_hex(r))
def func_load(archivo,ser):
    archivo = open(archivo, "r")
    r=""
```

```
59      for linea in archivo.readlines():
60          if(linea[0]=="#"):
61              pass
62          else:
63              rr=func_send(linea,ser)
64              try:
65                  r=r+rr+"\n"
66              except:
67                  return 'ERROR'
68      archivo.close()
69      return r
70  def func_open(config):
71      config = config.split()
72      try:
73          ser = serial.Serial(config[0],int(config[1]),
74              timeout=0.025)
75      except:
76          print("Error al abrir el puerto")
77      else:
78          print("Puerto abierto")
79          return ser
80  def func_close(ser):
81      try:
82          ser.close()
83      except:
84          print("Error al cerrar el puerto")
85      else:
86          print("Puerto cerrado")
87
88  if __name__ == '__main__':
89      while True:
90          if(len(sys.argv) > 2):
91              ser = serial.Serial(sys.argv[1],int(sys.argv[2]),
92                  timeout=0.025)
93              if sys.argv[3] == "send":
94                  func_send(sys.argv[4:],ser)
95              elif sys.argv[3] == "load":
96                  func_load(sys.argv[4],ser)
97              else:
98                  print("error, tercer argumento incorrecto")
99                  ser.close()
100                 break
101         else:
102             inp = input("ICFOserial>> ")
103             inp = inp.split(" ",1)
104             if inp[0]=="open":
105                 serial=func_open(inp[1])
106             elif inp[0]=="send":
107                 func_send(inp[1],serial)
108             elif inp[0]=="load":
109                 func_load(inp[1],serial)
110             elif inp[0]=="close":
111                 func_close(serial)
112             elif inp[0]=="help":
113                 print("open\nsend\nload\nclose\nexit")
114             elif inp[0]=="exit":
115                 break
116             else:
117                 print("help para ver comandos...\nPulsa enter
118                     para continuar")
```

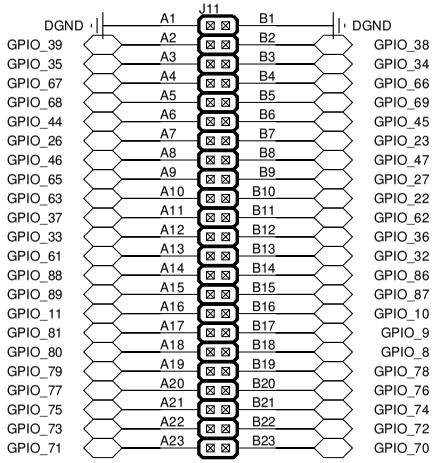
Interface

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
-- Company: ICFO
-- Engineer: Sergio López Casablanca
-- Revision 2.0
"""

from tkinter import *
from tkinter import ttk
from ICFOserial import *
import serial
import binascii

class Aplicacion():
    def __init__(self):
        self.raiz = Tk()
        self.raiz.title("ICFO Serial")
        self.portname= StringVar(value="COM")
        self.portspeed= StringVar(value="921600")
        self.received   = StringVar()

        box1text = "boton1"
        box2text = "boton2"
        box3text = "boton3"
        box4text = "boton4"

        self.etiq1 = ttk.Label(self.raiz, text="Port:")
        self.portnamelab = ttk.Entry(self.raiz,
            textvariable=self.portname, width=10, justify=RIGHT)
        self.etiq2 = ttk.Label(self.raiz, text="Baud Rate:")
        self.portspeedlab = ttk.Entry(self.raiz,
            textvariable=self.portspeed, width=10, justify=RIGHT)
        self.boton1 = ttk.Button(self.raiz, text=box1text,
            command=self.conf1)
        self.boton2 = ttk.Button(self.raiz, text=box2text,
            command=self.conf2)
        self.boton3 = ttk.Button(self.raiz, text=box3text,
            command=self.conf3)
        self.boton4 = ttk.Button(self.raiz, text=box4text,
            command=self.conf4)
        self.receivedlab = ttk.Label(self.raiz, textvariable=
            self.received, foreground="yellow", background="black",
            borderwidth=5, anchor=NW, justify=LEFT)

        self.etiq1.grid(column=0, row=0, sticky=W)
        self.portnamelab.grid(column=1,  row=0)
        self.etiq2.grid(column=0, row=2, sticky=W)
        self.portspeedlab.grid(column=1, row=2)
        self.boton1.grid(column=0, row=4)
        self.boton2.grid(column=1, row=4)
        self.boton3.grid(column=0, row=5)
        self.boton4.grid(column=1, row=5)
        self.receivedlab.grid(column=0, columnspan=2, row=8,
            sticky=W+E)
        self.raiz.mainloop()

```

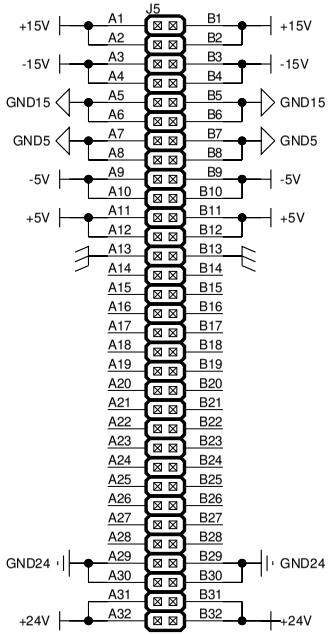
```
59      def openport(self, *args):
60          portcom=self.portname.get()+" "+self.portspeed.get()
61          self.serial=func_open(portcom)
62      def closeport(self, *args):
63          func_close(self.serial)
64      def conf1(self, *args):
65          try:
66              self.openport()
67              x=func_load("boton1.txt",self.serial)
68              self.received.set(x)
69              self.closeport()
70          except:
71              print("Error con el puerto o archivo .txt")
72              self.received.set("ERROR")
73      def conf2(self, *args):
74          try:
75              self.openport()
76              x=func_load("boton2.txt",self.serial)
77              self.received.set(x)
78              self.closeport()
79          except:
80              print("Error con el puerto o archivo .txt")
81              self.received.set("ERROR")
82      def conf3(self, *args):
83          try:
84              self.openport()
85              x=func_load("boton3.txt",self.serial)
86              self.received.set(x)
87              self.closeport()
88          except:
89              print("Error con el puerto o archivo .txt")
90              self.received.set("ERROR")
91      def conf4(self, *args):
92          try:
93              self.openport()
94              x=func_load("boton4.txt",self.serial)
95              self.received.set(x)
96              self.closeport()
97          except:
98              print("Error con el puerto o archivo .txt")
99              self.received.set("ERROR")
100
101 def main():
102     mi_app = Aplicacion()
103     return 0
104
105 if __name__ == '__main__':
106     main()
```



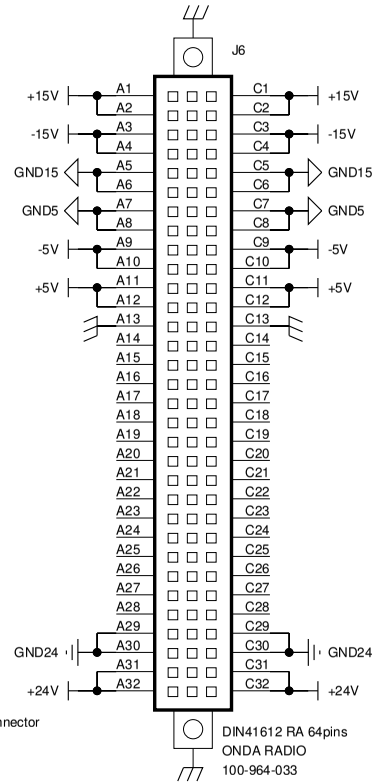
PC/104 Non-Stackthrough Connector
FARNELL



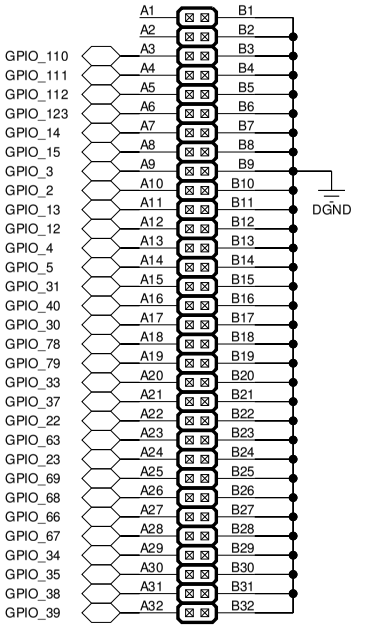
PC/104 Non-Stackthrough Connector
FARNELL



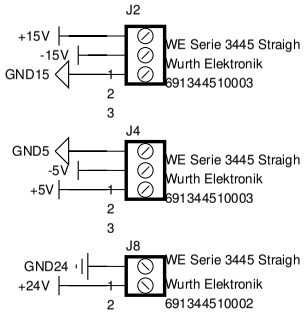
PC/104 Non-Stackthrough Connector
FARNELL
359993

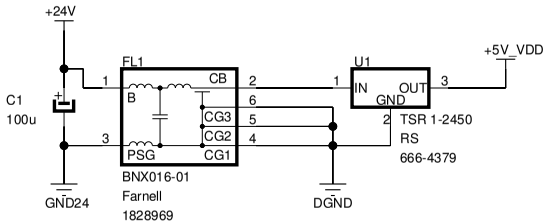


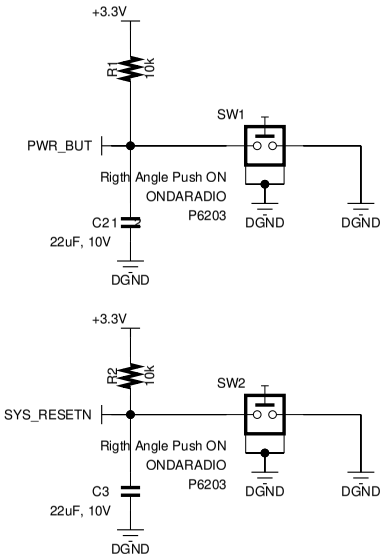
DIN41612 RA 64pins
ONDA RADIO
100-964-033



PC/104 Non-Stackthrough Connector
FARNELL
359993







ICFO's IP policy:	Any inventions, techniques, know-how, designs, plans, technical designs, processes, formulae, data, machinery, programmes, tools or instruments, experiments, trials etc developed at ICFO's workshops are ICFO's property. Therefore, it must remain at ICFO, and can be used outside ICFO only after written authorization by ICFO Director. For additional information please contact ICFO's KTT office at ktt@icfo.es.

BB_Adaptor_v3a

ICFO Institut de Ciències Fotòniques

ORDER: TE2016.526 REV: 3a

AUTHOR: Sergio López

DATE: 26-06-2017_11:08

SHEET: Power OF 1

Placa de adaptación

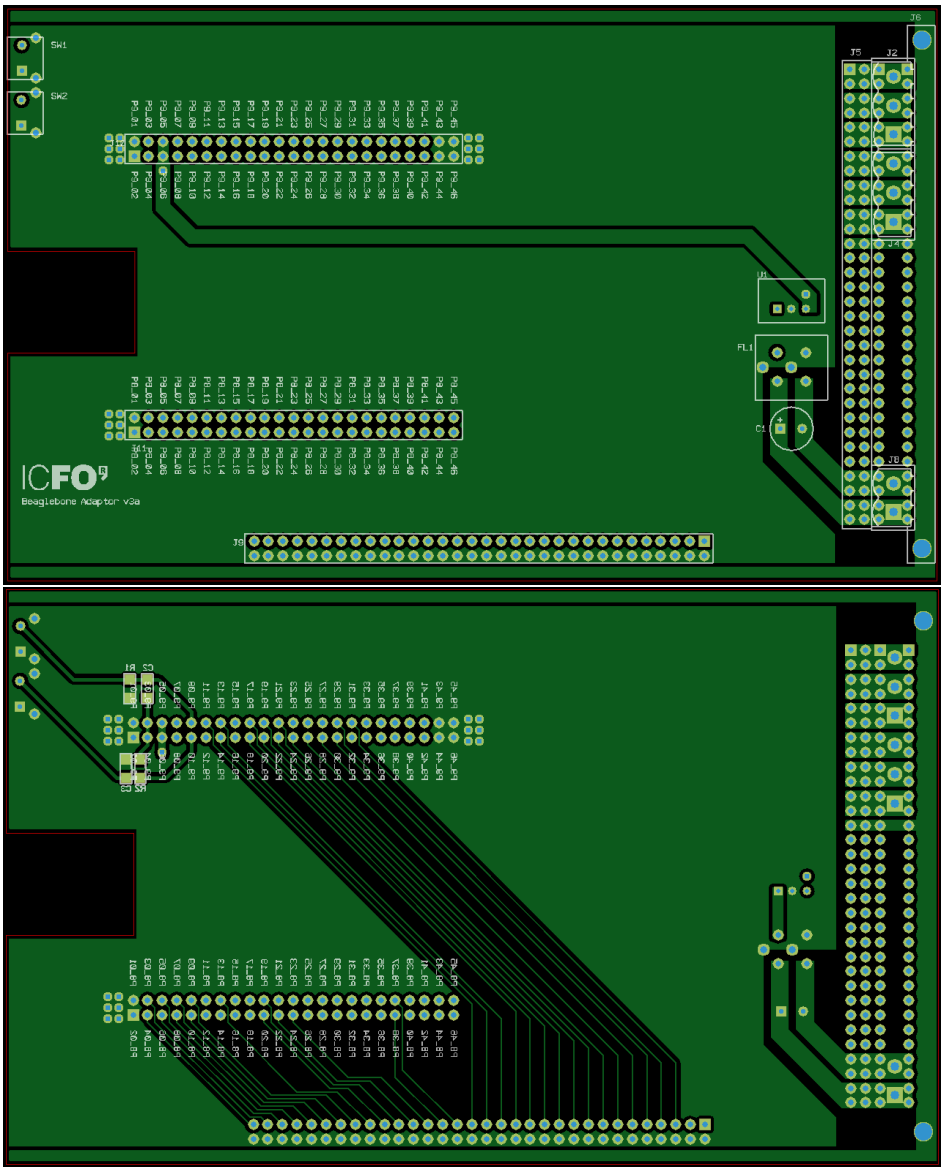


Figura B.1: Caras Top y Bottom

#	Ref	Material	Precio/u	Total
2	P6203	Pulsadores	0.2 €	0.4 €
1	6664379	Regulador 5VDC 1A	5.5 €	5.5 €
2	855-M20-6112045	Harwin PC / 104	3.2 €	6.4 €
2	CO3080	Tira Poste Macho Recto 80pins	0.4 €	0.8 €
2	R_SMD_1206	Resistencia Genérica	0.03 €	0.06 €
1	1828969	BNX016-01 - Filtro	3.4 €	3.4 €
1	1800677	Condensador electrolítico 100 µF	5.5 €	5.5 €
1	359993	Harwin PC104, 64vías	4.8 €	4.8 €
2	C_SMD_1206_X5R	Condensador genérico	0.1 €	0.2 €
1	100-964-033	Conector macho DIN41612 64vías acodado	1.2 €	1.2 €
				28.26 €

Tabla B.1: Costes placa de adaptación

Apéndice C

Descripción de circuitos en VHDL

TopLevel.ucf

TopLevel.vhd

AD7983_Enable.vhd

AD5061_Enable.vhd

Edge_Detect.vhd

TopLevel.ucf

```
# GLOBAL CLOCK
NET clk LOC = P55 | IOSTANDARD = LVCMOS33;

#ADC
NET adc_out_cnv LOC = P17 | IOSTANDARD = LVCMOS33;
NET adc_SCLK LOC = P22 | IOSTANDARD = LVCMOS33;
NET adc_SDO_to_miso LOC = P21 | IOSTANDARD = LVCMOS33;

#DAC1
NET dac_out_sync LOC = P14 | IOSTANDARD = LVCMOS33;
NET dac_SCLK LOC = P15 | IOSTANDARD = LVCMOS33;
NET dac_mosi_to_DIN LOC = P16 | IOSTANDARD = LVCMOS33;

# USB-RS232 (FTDI-UB232B)
NET TXD LOC = P50 | IOSTANDARD = LVCMOS33; #input
NET RXD  LOC = P51 | IOSTANDARD = LVCMOS33; #output
```

TopLevel.vhd

```
----
-- Company: ICFO
-- Engineer: Sergio López Casablanca
----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library unisim;
use unisim.vcomponents.all;

entity TopLevel is
    Port (
      clk : in  STD_LOGIC;
      RXD : out  STD_LOGIC;
      TXD : in  STD_LOGIC;

      adc_out_cnv              : out STD_LOGIC;
      adc_SDO_to_miso             : in STD_LOGIC;
      adc_SCLK                 : out STD_LOGIC;
      adc_mosi_to_DIN             : out STD_LOGIC;

      dac_out_sync             : out STD_LOGIC;
      dac_SDO_to_miso             : in STD_LOGIC;
      dac_SCLK               : out STD_LOGIC;
      dac_mosi_to_DIN          : out STD_LOGIC;
   );
end TopLevel;

architecture Behavioral of TopLevel is
   constant number_of_master_registers : integer := 90;
   signal locked:  STD_LOGIC;
```

```
35    signal globalCLK_in: STD_LOGIC;
36    signal globalCLK_neg: STD_LOGIC;
37    signal globalCLK: STD_LOGIC;
38    signal new_data_rx: STD_LOGIC;
39    signal data_rx: STD_LOGIC_VECTOR (7 downto 0);
40    signal data_tx: STD_LOGIC_VECTOR (7 downto 0);
41    signal new_data_tx: STD_LOGIC;
42    signal data_send: STD_LOGIC;
43    signal RXD_in : STD_LOGIC;
44    signal clk_out : STD_LOGIC;
45    signal enable_read : STD_LOGIC := '0';
46    signal counterDataRX : std_logic_vector (7 downto 0) := "00000000";
47    signal data_send_pre : std_logic := '0';
48    signal data_send_new : std_logic := '0';
49    signal captureData_send : std_logic := '0';
50    signal new_data_rx_pre : std_logic := '0';
51    signal new_data_rx_new : std_logic := '0';
52    signal captureNewDataRX : std_logic := '0';
53    signal new_value : std_logic;
54    signal new_value_reg : std_logic := '0';
55    signal new_value_reg_pre : std_logic := '0';
56    signal new_value_reg_new : std_logic := '0';
57    signal captureNewValue : std_logic := '0';
58    type registers is array (0 to 255) of std_logic_vector(15 downto 0)
          ;
59    signal master_register_in, master_register_in_aux,
          master_register_out : registers := (others => (others => '0'));
60    signal address: STD_LOGIC_VECTOR (7 downto 0);
61    signal write_value: STD_LOGIC_VECTOR (15 downto 0);
62    signal read_value: STD_LOGIC_VECTOR (15 downto 0);
63    signal read_or_write : STD_LOGIC;
64    signal new_answer : STD_LOGIC := '0';
65    signal not_data_send : std_logic;
66    signal data_in_DIN_1 : std_logic_vector (15 downto 0);
67    signal data_out_DOUT_1 : std_logic_vector (15 downto 0);
68
69    signal adc_cnv_aux            : std_logic;
70    signal adc_enable_aux         : std_logic;
71    signal adc_data_rx_spi        : std_logic_vector (15 downto 0);
72    signal adc_sync_aux         : std_logic;
73    signal adc_sclk_signal        : STD_LOGIC;
74    signal adc_mosi             : STD_LOGIC;
75    signal adc_n_slaves         : std_logic_vector (0 downto 0);
76    ---
77    signal dac_syncnv_aux         : std_logic;
78    signal dac_enable_aux         : std_logic;
79    signal dac_data_rx_spi        : std_logic_vector (15 downto 0);
80    signal dac_SYNC_aux         : std_logic;
81    signal dac_sclk_aux           : STD_LOGIC;
82    signal dac_mosi             : STD_LOGIC;
83    signal dac_n_slaves         : std_logic_vector (0 downto 0);
84
85    signal nreset : STD_LOGIC;
86
87    component DCM
88       port (
89          -- Clock in ports
90          CLK_IN1           : in     std_logic;
91          -- Clock out ports
92          CLK_OUT1          : out    std_logic;
93          LOCKED              : out std_logic;
94          -- Status and control signals
```

```
95          RESET               : in      std_logic
96      );
97  end component;
98
99  component UART
100     port (
101         clk : in  STD_LOGIC;
102         rxd : in  STD_LOGIC;
103         new_data_rx : out  STD_LOGIC;
104         data_rx : out  STD_LOGIC_VECTOR (7 downto 0);
105         txd : out  STD_LOGIC;
106         data_tx : in  STD_LOGIC_VECTOR (7 downto 0);
107         new_data_tx : in  STD_LOGIC;
108         data_send : out  STD_LOGIC;
109         enable_read : in STD_LOGIC;
110         new_value : out STD_LOGIC
111     );
112 end component;
113
114 component MasterRegister
115     port(
116         clk : in STD_LOGIC;
117         register_in : in  STD_LOGIC_VECTOR (15 downto 0);
118         register_in_aux : in STD_LOGIC_VECTOR (15 downto 0);
119         register_out : out  STD_LOGIC_VECTOR (15 downto 0)
120     );
121 end component;
122
123 component MRSManager
124     port(
125         clk : in  STD_LOGIC;
126         data_rx : in  STD_LOGIC_VECTOR (7 downto 0);
127         new_data_rx : in  STD_LOGIC;
128         enable_read_rx : out  STD_LOGIC;
129         new_value_in : in STD_LOGIC;
130         data_tx : out  STD_LOGIC_VECTOR (7 downto 0);
131         address : out  STD_LOGIC_VECTOR (7 downto 0);
132         write_value : out  STD_LOGIC_VECTOR (15 downto 0);
133         new_value_out : out STD_LOGIC;
134         read_value : in  STD_LOGIC_VECTOR (15 downto 0);
135         new_answer : in STD_LOGIC;
136         read_or_write : out STD_LOGIC;
137         enable_read : in STD_LOGIC;
138         new_value : out STD_LOGIC
139     );
140 end component;
141
142 component DOUT
143     port(
144         clk : in  STD_LOGIC;
145         enable : in  STD_LOGIC;
146         data_out_register : in  STD_LOGIC_VECTOR (15 downto 0);
147         data_out : out  STD_LOGIC_VECTOR (15 downto 0)
148     );
149 end component;
150
151 component DIN
152     port(
153         clk             : in  STD_LOGIC;
154         enable          : in  STD_LOGIC;
155         data_in         : in  STD_LOGIC_VECTOR (15 downto 0);
156         data_in_register  : out  STD_LOGIC_VECTOR (15 downto 0)
```

```
157         );
158     end component;
159 
160     component AD7983_Enable
161         port(clk, nrst : in  std_logic;
162              ena_ad    : in  std_logic;
163              mode      : in  std_logic;
164              cnv       : out     std_logic
165             );
166     end component;
167 
168      component AD5061_Enable
169         port (clk      : in  std_logic;
170                nrst        : in  std_logic;
171              ena_ad    : in  std_logic;
172              data      : in  std_logic_vector(15 downto 0);
173                 sync        : out     std_logic
174              );
175       end component;
176 
177     component EDGE_DETECT
178         port (clk, nrst       : in std_logic;
179              INPUT            : in std_logic;
180              RISE_EDGE        : out std_logic;
181              FALL_EDGE        : out std_logic
182             );
183     end component;
184 
185     component SPI_master_ADC
186         generic(
187             slaves    : INTEGER := 1;
188             bus_width    : INTEGER := 16);
189         port(
190             clk       : in     STD_LOGIC;
191             nrst   : in     STD_LOGIC;
192             enable    : in     STD_LOGIC;
193             cpol      : in     STD_LOGIC;
194             cpha      : in     STD_LOGIC;
195             cont      : in     STD_LOGIC;
196             clk_div   : in     INTEGER;
197             addr      : in     INTEGER;
198             tx_data   : in     STD_LOGIC_VECTOR(bus_width-1 DOWNTO 0);
199             miso      : in     STD_LOGIC;
200             sclk      : out    STD_LOGIC;
201             ss_n      : out    STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);
202             mosi      : out    STD_LOGIC;
203             busy      : out    STD_LOGIC;
204             rx_data   : out    STD_LOGIC_VECTOR(bus_width-1 DOWNTO 0)
205         );
206     end component;
207 
208     component SPI_master_DAC
209         generic(
210             slaves    : INTEGER := 1;
211             bus_width    : INTEGER := 24);
212         port(
213             clk       : in     STD_LOGIC;
214             nrst   : in     STD_LOGIC;
215             enable    : in     STD_LOGIC;
216             cpol      : in     STD_LOGIC;
217             cpha      : in     STD_LOGIC;
218             cont      : in     STD_LOGIC;
```

```
219             clk_div : in      INTEGER;
220             addr    : in      INTEGER;
221             tx_data : in      STD_LOGIC_VECTOR(bus_width-1 downto 0)
222             miso    : in      STD_LOGIC;
223             sclk    : out     STD_LOGIC;
224             ss_n    : out     STD_LOGIC_VECTOR(slaves-1 downto 0);
225             mosi    : out     STD_LOGIC;
226             busy    : out     STD_LOGIC
227         );
228     end component;
229 
230 
231 begin
232 
233     DCM_1 : DCM
234         port map(
235             CLK_IN1 => clk,
236             CLK_OUT1 => globalCLK_in,
237             LOCKED => locked,
238             RESET  => '0'
239         );
240 
241     globalCLK_neg <= not(globalCLK_in);
242 
243     ODDR2_1 : ODDR2
244         generic map(
245             DDR_ALIGNMENT => "NONE",
246             INIT => '0',
247             SRTYPE => "SYNC")
248         port map (
249             Q => globalCLK,
250             C0 => globalCLK_in,
251             C1 => globalCLK_neg,
252             CE => '1',
253             D0 => '1',
254             D1 => '0',
255             R => '0',
256             S => '0'
257         );
258 
259     OBUF_1 : OBUF
260         generic map (
261             DRIVE => 12,
262             IOSTANDARD => "DEFAULT",
263             SLEW => "SLOW")
264         port map (
265             O => clk_out,
266             I => globalCLK
267         );
268 
269     UserInterface : UART
270         port map (
271             clk => globalCLK_in,
272             rxd => TXD,
273             new_data_rx => new_data_rx,
274             data_rx => data_rx,
275             txd => RXD_in,
276             data_tx => data_tx,
277             new_data_tx => new_data_tx,
278             data_send => data_send,
279             enable_read => enable_read,
280             new_value => new_value
```

```
281         );
282
283     process(globalCLK_in)
284     begin
285         if (globalCLK_in'event and globalCLK_in = '1') then
286             RXD <= RXD_in;
287         end if;
288     end process;
289
290     MRS_masterRegisters : for number in 0 to number_of_master_registers
            generate
291     begin
292         MR : MasterRegister
293             port map (
294                 clk => globalCLK_in,
295                 register_in => master_register_in(number),
296                 register_in_aux => master_register_in_aux(number),
297                 register_out => master_register_out(number)
298             );
299     end generate MRS_masterRegisters;
300
301
302     process (globalCLK_in)
303     begin
304         if (globalCLK_in'event and globalCLK_in = '1') then
305             not_data_send <= not(data_send);
306         end if;
307     end process;
308
309     MRS_Manager : MRSManager
310         port map(
311             clk => globalCLK_in,
312             data_rx => data_rx,
313             new_data_rx => new_data_rx,
314             enable_read_rx => enable_read,
315             new_value_in => new_value,
316             data_tx => data_tx,
317             address => address,
318             write_value => write_value,
319             new_value_out => new_value_reg,
320             read_value => read_value,
321             new_answer => new_answer,
322             read_or_write => read_or_write,
323             enable_read => not_data_send,
324             new_value => new_data_tx
325         );
326
327     process (globalCLK_in)
328     begin
329         if (globalCLK_in'event and globalCLK_in = '1') then
330             new_value_reg_pre <= new_value_reg;
331             new_value_reg_new <= new_value_reg_pre;
332             if (new_value_reg_pre = '1' and new_value_reg_new = '0') then
333                 captureNewValue <= '1';
334             end if;
335             if (captureNewValue = '1') then
336                 captureNewValue <= '0';
337                 if (read_or_write = '1') then --WRITE
338                     master_register_in(conv_integer(address)) <=
                            write_value;
339                     new_answer <= '1';
340                 elsif (read_or_write = '0') then --READ
```

```
341                 read_value <= master_register_out(conv_integer(address)
                        );
342                 new_answer <= '1';
343              end if;
344           else
345              new_answer <= '0';
346           end if;
347        end if;
348     end process;
349
350     DIN_1 : DIN
351        port map(
352           clk => globalCLK_in,
353           enable => master_register_out(0)(1),
354           data_in => data_in_DIN_1,
355           data_in_register => master_register_in_aux(2)
356        );
357
358     DOUT_1 : DOUT
359        port map(
360           clk => globalCLK_in,
361           enable => master_register_out(0)(0),
362           data_out_register => master_register_out(3),
363           data_out => data_out_DOUT_1
364        );
365
366     data_in_DIN_1 <= data_out_DOUT_1;
367
368     ENABLE_AD7983 : AD7983_ENABLE
369     port map(
370        clk    => globalCLK_in,
371        nrst   => nreset,
372        ena_ad   => master_register_out(0)(4),
373        mode   => master_register_out(0)(5),
374        cnv      => adc_cnv_aux
375        );
376        adc_out_cnv <= adc_cnv_aux;
377
378     EDGE_DETECT_SPI_ADC : EDGE_DETECT
379     port map(
380        clk    => globalCLK_in,
381        nrst   => nreset,
382        INPUT => adc_cnv_aux,
383        RISE_EDGE    => open,
384        FALL_EDGE    => adc_enable_aux
385        );
386
387     ADC_SPI : SPI_master_ADC
388     generic map(
389        slaves => 1,
390        bus_width => 16)
391     port map(
392        clk    => globalCLK_in,
393        nrst   => nreset,
394        enable    => adc_enable_aux,
395        cpol   => '1',
396        cpha   => '0',
397        cont     => '0',
398        clk_div => 2,
399        addr     => 0,
400        tx_data => "0000000000000000",
401        miso => adc_SDO_to_miso,
```

```
402         sclk => adc_sclk_signal,
403         ss_n => adc_n_slaves,
404         mosi => adc_mosi,
405         busy => open,
406         rx_data => master_register_in_aux(11)
407         );
408     adc_sync_aux <= adc_n_slaves(0);
409     adc_SCLK <= adc_sclk_signal when adc_sync_aux = '0' else '1';
410     adc_mosi_to_DIN   <=    adc_mosi when adc_sync_aux = '0' else '1';
411
412
413     ENABLE_AD5061 : AD5061_ENABLE
414     port map(
415         clk    => globalCLK_in,
416         nrst   => nreset,
417         ena_ad    => master_register_out(0)(12),
418         data   => master_register_out(31),
419         sync   => dac_syncnv_aux
420         );
421         dac_out_sync <= dac_syncnv_aux;
422
423     EDGE_DETECT_SPI_DAC : EDGE_DETECT
424     port map(
425         clk    => globalCLK_in,
426         nrst   => nreset,
427         INPUT => dac_syncnv_aux,
428         RISE_EDGE    => open,
429         FALL_EDGE    => dac_enable_aux
430         );
431
432     DAC_SPI : SPI_master_DAC
433     generic map(
434         slaves => 1,
435         bus_width => 24)
436     port map(
437         clk    => globalCLK_in,
438         nrst   => nreset,
439         enable    => dac_enable_aux,
440         cpol   => '1',
441         cpha   => '0',
442         cont     => '0',
443         clk_div => 4,
444         addr     => 0,
445         tx_data => "00000000"&master_register_out(31),
446         miso => dac_SDO_to_miso,
447         sclk => dac_sclk_aux,
448         ss_n => dac_n_slaves,
449         mosi => dac_mosi,
450         busy => open
451         );
452     dac_SYNC_aux <= dac_n_slaves(0);
453     dac_SCLK     <=     dac_sclk_aux when dac_SYNC_aux = '0' else '1';
454     dac_mosi_to_DIN   <=    dac_mosi when dac_SYNC_aux = '0' else '1';
455
456     nreset <= '1';
457 end Behavioral;
```

AD7983_Enable

```
1  ----
2  -- Company: ICFO
3  -- Engineer: Sergio López Casablanca
4  -- Module Name:    AD7983_Enable
5  -- Description: This block generates the cnv signal for the AD7983
6  -- Revision 0.01 - File Created
7  ----
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12
13 entity AD7983_Enable is
14    port (clk : in std_logic;
15       nrst : in std_logic;
16       ena_ad   : in std_logic;
17       mode  : in std_logic;
18       cnv : out std_logic
19       );
20 end AD7983_Enable;
21
22 architecture synth of AD7983_Enable is
23    signal pstate, n_state : std_logic_vector(1 downto 0);
24    signal count   : integer;
25    signal mode_aux   : std_logic;
26
27 begin
28    AD7983_Enable: process (clk, nrst)
29     begin
30       if (nrst = '0') then
31          pstate <= "00";
32          count <= 0;
33       elsif (clk'event and clk = '1' ) then
34          pstate <= n_state;
35          mode_aux <= mode;
36          if pstate = "00" then
37             count <= 0;
38          elsif (pstate = "01" or pstate = "10") then
39             count <= count+1;
40          end if;
41       end if;
42     end process;
43
44    n_state <= "01" when (pstate = "00" and ena_ad = '1') else
45       "01" when (pstate = "01" and count < 100) else
46         "10" when (pstate = "01" and count = 100) else
47       "10" when (pstate = "10" and count < 170) else
48       "00" when (pstate = "10" and count = 170 and mode_aux = '1')
             else
49         "11" when (pstate = "10" and count = 170 and mode_aux = '0')
               else
50       "00" when (pstate = "11" and ena_ad = '0' and mode_aux = '0')
             else
51       "11" when (pstate = "11") else
52       "00";
53
54
55    cnv <= '0' when pstate = "00" else --
```

```
56        '1' when pstate = "01" else --
57        '0' when pstate = "10" else --
58        '0' when pstate = "11" else
59        '0';
60 end synth;
```

AD5061_Enable

```
1  ----
2  -- Company: ICFO
3  -- Engineer: Sergio López Casablanca
4  -- Module Name:    AD5061_Enable
5  -- Description: This block generates the sync signal for the AD5061
6  -- Revision 0.01 - File Created
7  ----
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12
13 entity AD5061_enable is
14    port (clk : in std_logic;
15       nrst : in std_logic;
16       ena_ad   : in std_logic;
17       data  : in  std_logic_vector(15 downto 0);
18       sync : out std_logic
19       );
20 end AD5061_enable;
21
22 architecture synth of AD5061_Enable is
23    signal pstate, n_state : std_logic_vector(1 downto 0);
24    signal count   : integer range 0 to 255;
25    signal old_data : std_logic_vector(15 downto 0);
26    signal diff_data : std_logic;
27
28 begin
29     AD5061_Enable: process (clk, nrst)
30     begin
31       if (nrst = '0') then
32          pstate <= "00";
33          count <= 0;
34       elsif (clk'event and clk = '1' ) then
35          pstate <= n_state;
36          old_data <= data;
37          if data /= old_data then
38             diff_data <= '1';
39          elsif count = 1 then
40             diff_data <= '0';
41          end if;
42          if pstate = "00" then
43             count <= 0;
44          elsif (pstate = "01" or pstate = "10") then
45             count <= count+1;
46          end if;
47
48       end if;
49    end process;
```

```
50
51    n_state <= "01" when (pstate = "00" and ena_ad = '1' and diff_data
          = '1') else
52       "01" when (pstate = "01" and count < 5) else
53        "10" when (pstate = "01" and count = 5) else
54       "10" when (pstate = "10" and count < 228) else --200
55       "00" when (pstate = "10" and count = 228) else
56       "00";
57
58    sync <= '0' when pstate = "00" else --
59       '1' when pstate = "01" else --
60       '0' when pstate = "10" else --
61       '0' when pstate = "11" else
62       '0';
63 end synth;
```

Edge_Detect

```
1 ----------------------------------------------------------------------------------
2 -- Company: ICFO
3 -- Engineer: Sergio López Casablanca
4 -- Module Name:    EDGE_DETECT
5 -- Description: This block generates an enable signal for one clock
      when an edge is detected
6 -- Revision 0.01 - File Created
7 ----------------------------------------------------------------------------------
8
9 library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.std_logic_arith.all;
12
13 entity EDGE_DETECT is
14    port(clk, nrst     : in std_logic;       -- clk from DCM = 350 MHz
15         INPUT         : in std_logic;
16         RISE_EDGE     : out std_logic;  -- The enable signal generated
               when rise edge
17         FALL_EDGE     : out std_logic       -- The enable signal
               generated when rise edge
18       );
19 end EDGE_DETECT;
20
21 architecture behaviour of EDGE_DETECT is
22    signal input_edge : std_logic;
23
24 begin
25    EDGE_DETECT: process(clk, nrst)
26    begin
27          if (nrst = '0') then
28             input_edge <= '0';
29
30          elsif rising_edge(clk) then
31             input_edge <= INPUT;
32          end if;
33    end process;
34
35 RISE_EDGE <= not(input_edge) and INPUT;
36 FALL_EDGE <= input_edge and not(INPUT);
```

```
37
38 end behaviour;
```

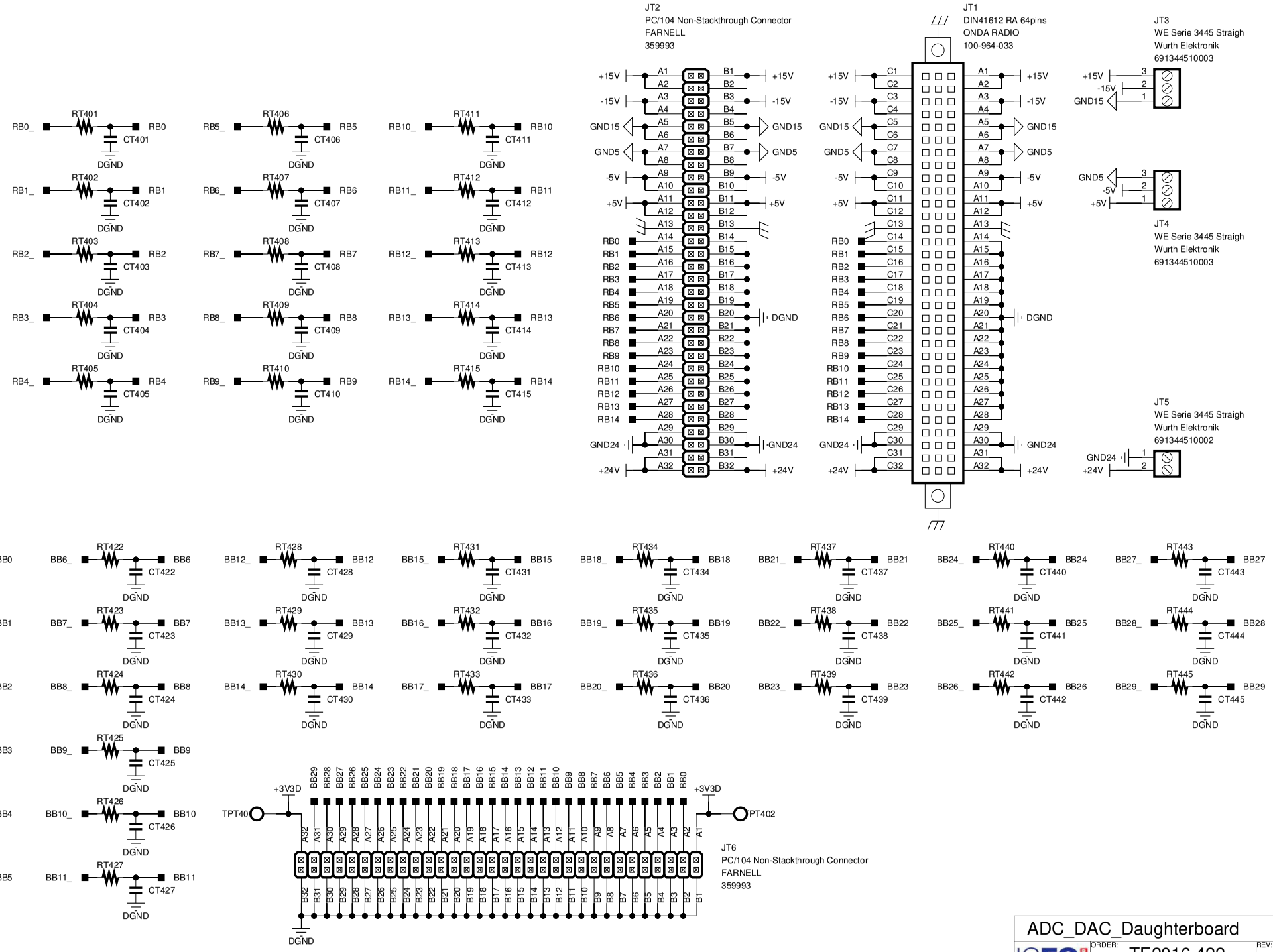
Apéndice D

Placa de conversores

Esquemático

Layout

Costes



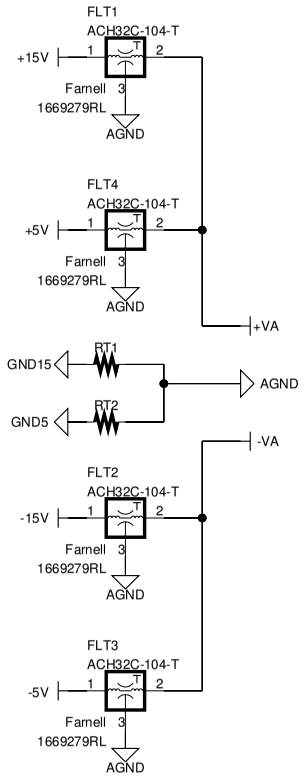
ADC_DAC_Daughterboard

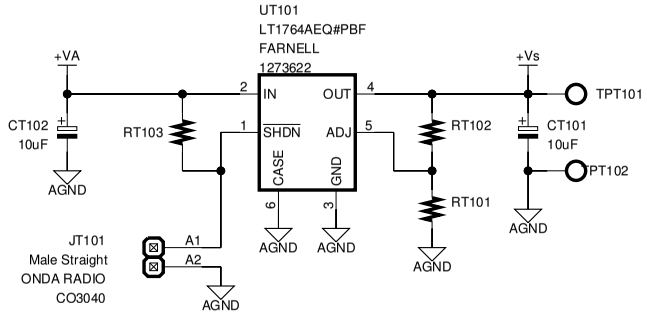
ICFO Institut de Ciències Fotòniques

ORDER: TE2016.422 REV: 4a

AUTHOR: Sergio López

DATE: 26-06-2017_09:13 SHEET: 1 OF 5



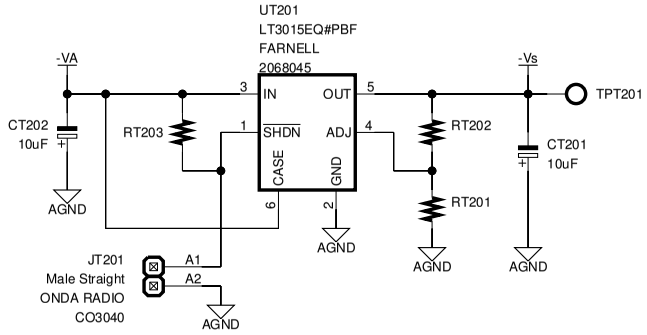


LT1764 Application Information

Input Voltage Range: 2.7V to 20V
Dropout Voltage: 340mV
Output Current: 3.0A
Output Voltage Range: 1.21V to 20V
Output Noise: 40uVrms (10Hz to 100kHz)

Vo = 1.21V·(1+R2/R1)+(Iadj)·(R2)
Vadj = 1.21V
Iadj = 3uA at 25ºC

Vo Resistor Dividers with current of aprox 100uA :



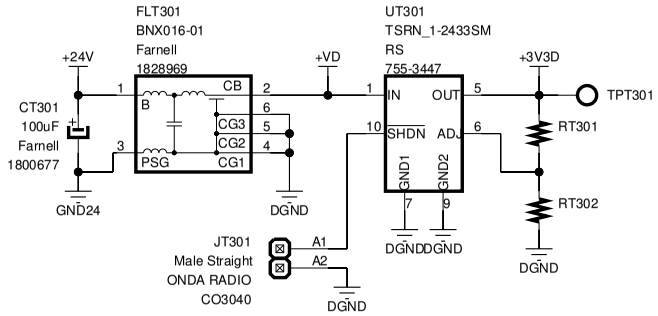
LT3015 Application Information

Input Voltage Range: -1.8V to -30V
Dropout Voltage: 310mV
Output Current: 1.5A
Output Voltage Range: -1.22V to -29.3V
Output Noise: 60uVrms (10Hz to 100kHz)

Vo = -1.22V·(1+R2/R1)+(Iadj)·(R2)
Vadj = -1.22V
Iadj = 30nA at 25ºC

Vo Resistor Dividers with current of aprox 100uA :

Vo= -2.5V: R1=12.1k, R2=12.7k
Vo= -3.0V: R1=12.1k, R2=17.8k
Vo= -3.3V: R1=12.1k, R2=20.5k
Vo= -5.0V: R1=12.1k, R2=37.4k
Vo= -12.0V: R1=12.1k, R2=107k
Vo= -15.0V: R1=12.4k, R2=140k



TSRN_1-2433SM:

Input Voltage Range: 4.6V to 42V
Output Current: 1.0A
Nominal output voltage: 3.3V
Trim output voltage range: 1.5V to 5.5V

Trim up (RU=>RT302):
RU (kohm)= (Vo-3.3)/26.4
Trim down (RD=>RT301):
RD (kohm)= (Vo·3.3-26.4)/(3.3-Vo)

Remote ON/OFF:
ON: 2V-5V or open circuit
OFF: 0V-0.8V or short circuit

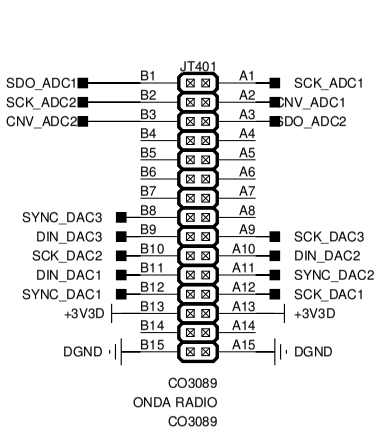
ADC_DAC_Daughterboard

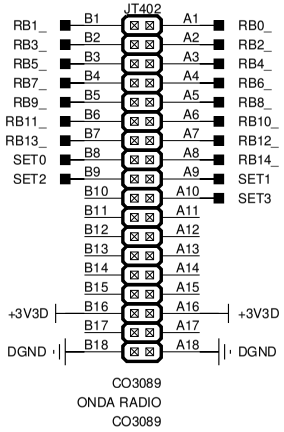
ICFO Institut de Ciències Fotòniques

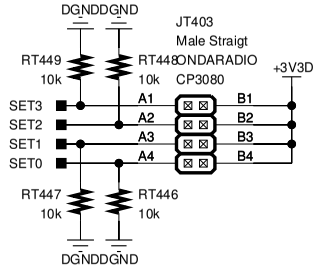
ORDER: TE2016.422 REV: 4a

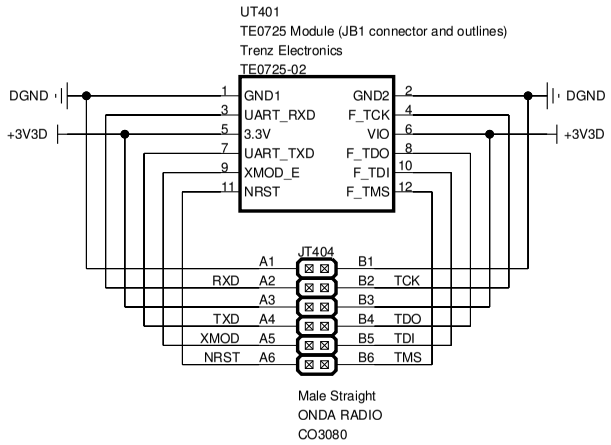
AUTHOR: Sergio López

DATE: 26-06-2017_09:13









ADC_DAC_Daughterboard

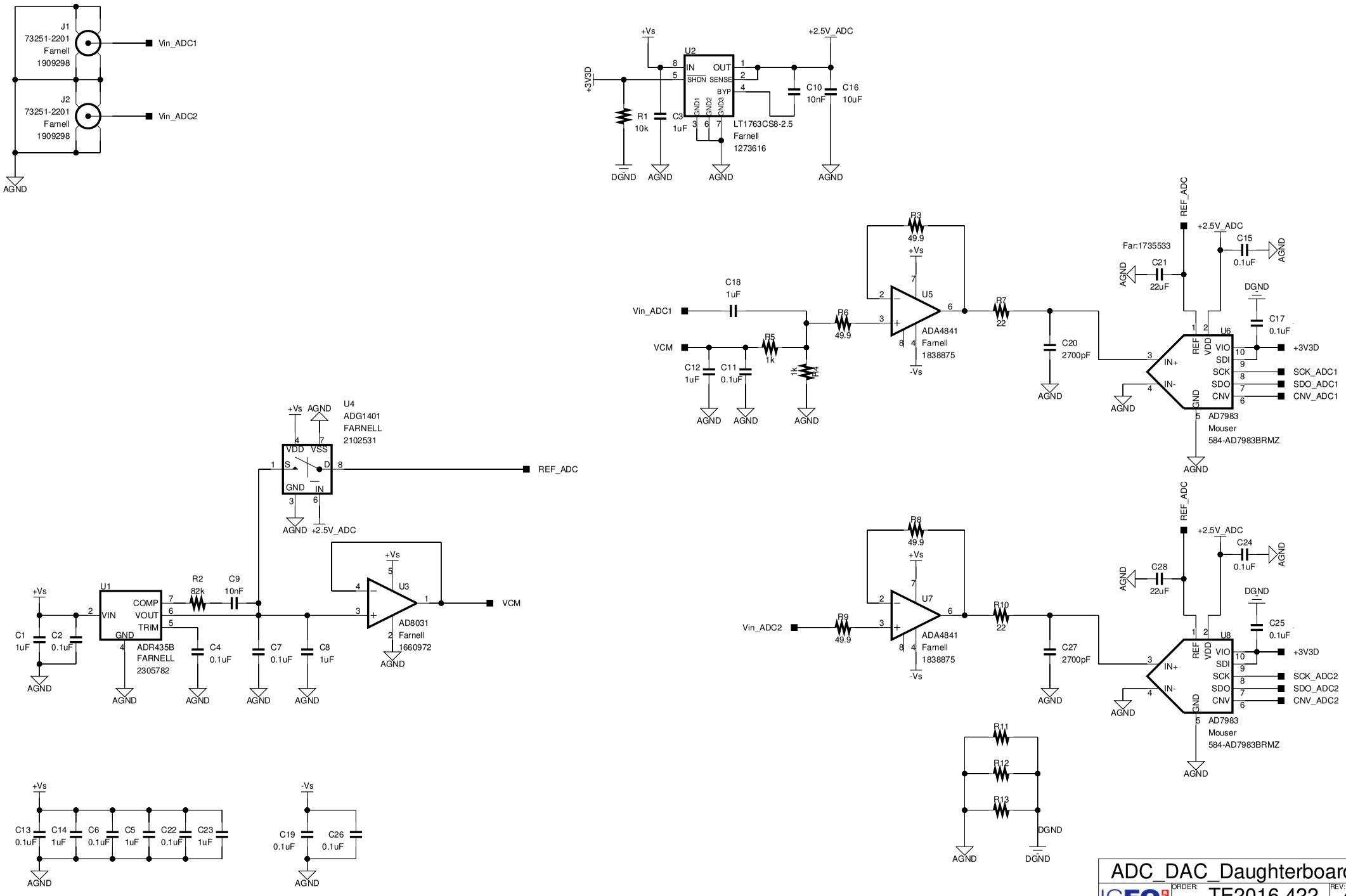
ICFO Institut de Ciències Fotòniques

ORDER: TE2016.422 REV: 4a

AUTHOR: Sergio López

DATE: 26-06-2017_09:13

SHEET: 3 OF 5



ICFO's IP policy: Any inventions, techniques, know-how, designs, plans, technical designs, processes, formulae, data, machinery, programmes, tools or instruments, experiments, trials etc developed at ICFO's workshops are ICFO's property. Therefore, it must remain at ICFO, and can be used outside ICFO only after written authorization by ICFO Director. For additional information please contact ICFO's KTT office at ktt@icfo.es.

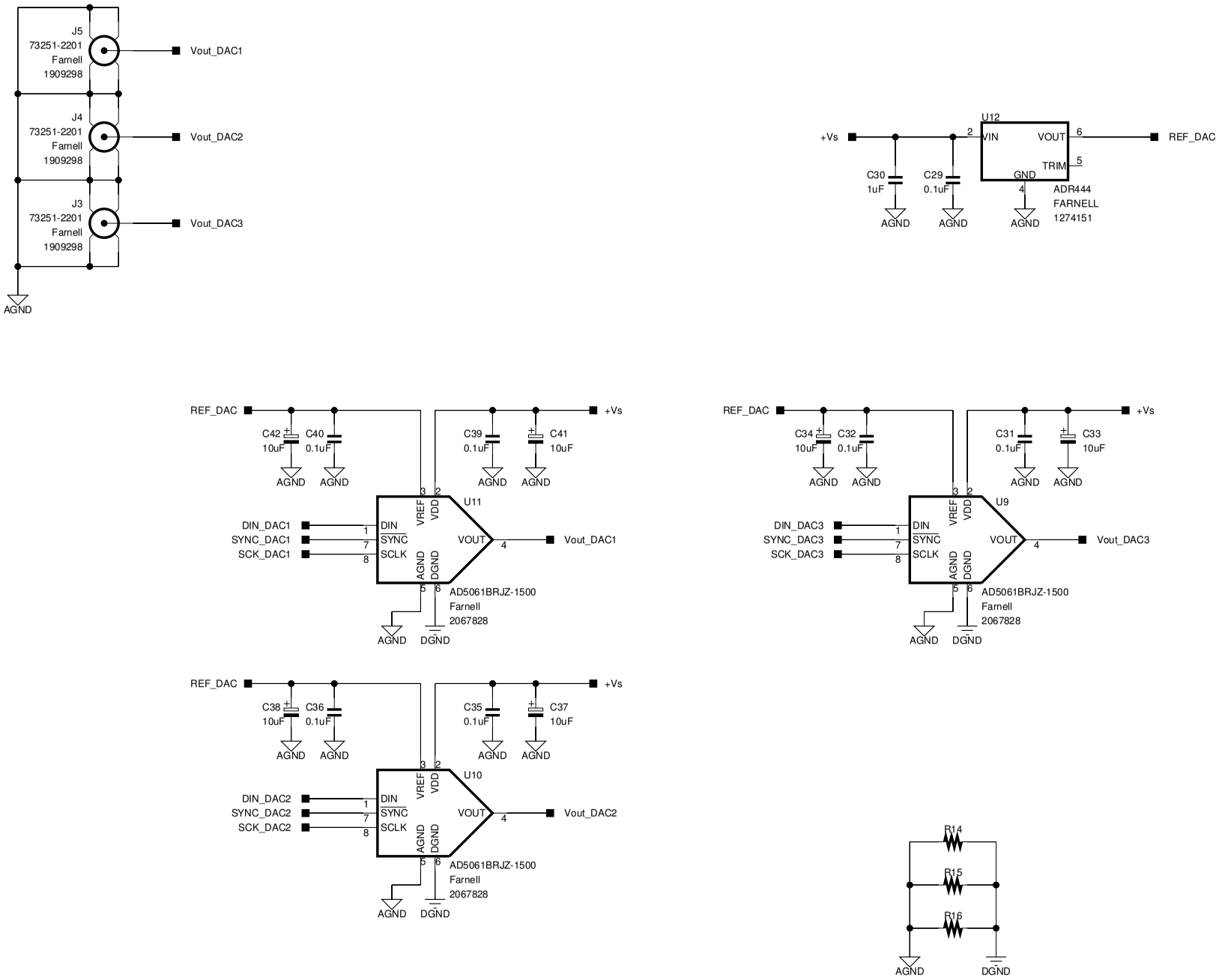
ADC_DAC_Daughterboard

ICFO Institut de Ciències Fotòniques

ORDER: TE2016.422 REV: 4a

AUTHOR: Sergio López

DATE: 26-06-2017_09:13 SHEET: 4 OF 5



ADC_DAC_Daughterboard

ICFO Institut de Ciències Fotòniques

ORDER: TE2016.422 REV: 4a

AUTHOR: Sergio López

DATE: 26-06-2017_09:18 SHEET: 5 OF 5

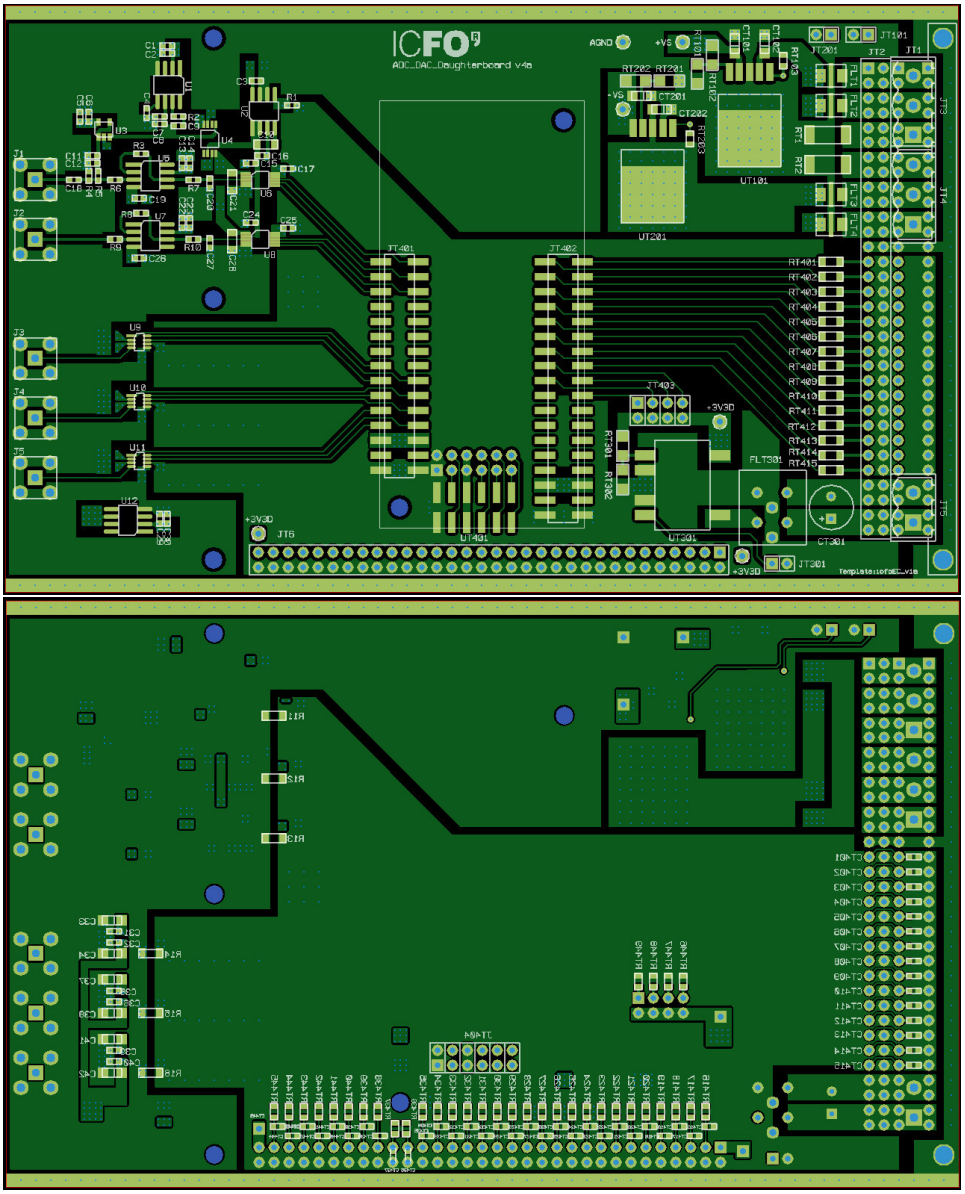


Figura D.1: Cara Top y Bottom

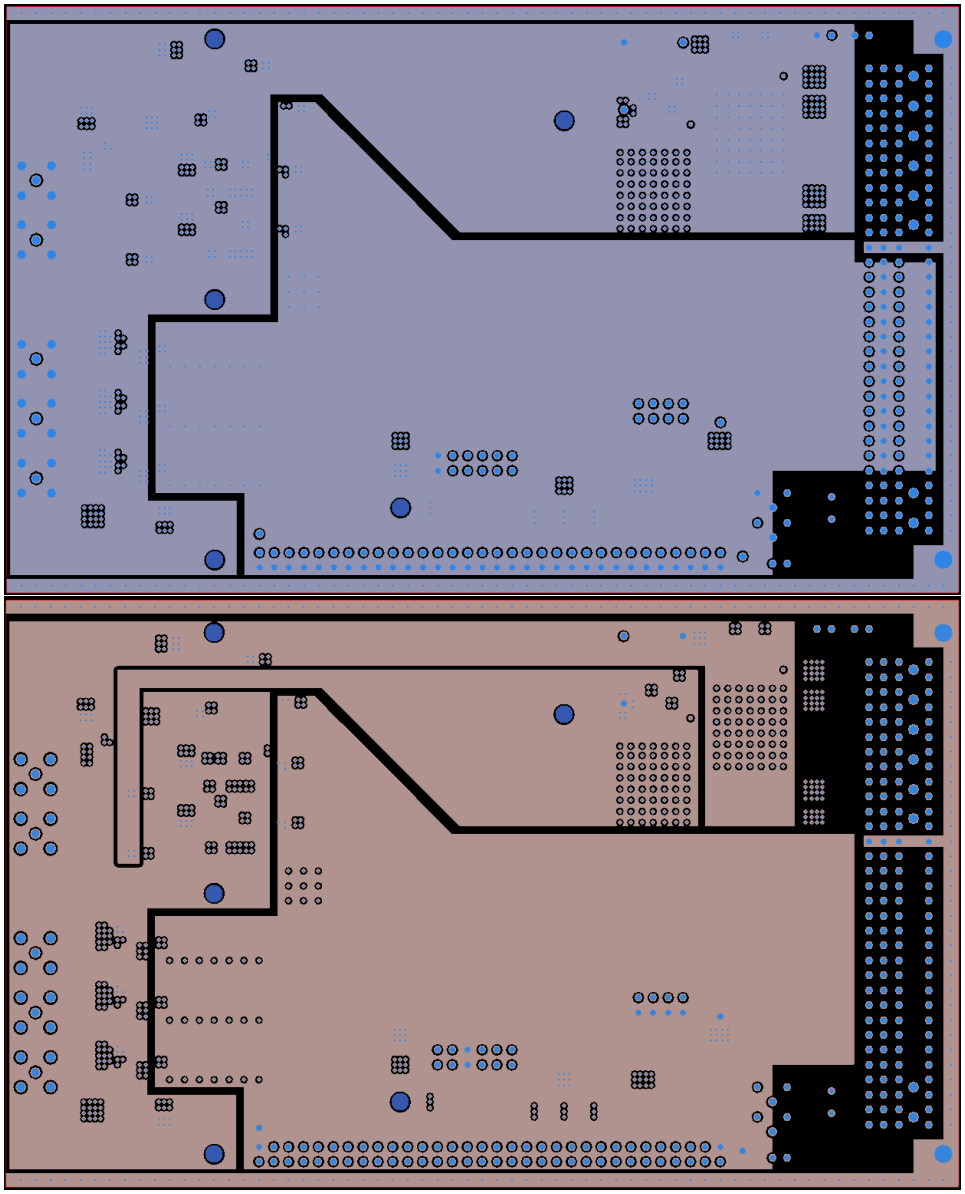


Figura D.2: Caras intermedias

#	Ref	Material	Precio/u	Total
1	1660972	AD8031ARTZ	2.46 €	2.46 €
1	2102531	ADG1401BRMZ	2.15 €	2.15 €
1	1273616	LT1763CS8-2.5	2.02 €	2.02 €
2	CO3089	TIRA POSTE SMD	4.41 €	8.82 €
1	1828969	BNX016-01	3.44 €	3.44 €
2	1838875	ADA4841	4.66 €	9.32 €
2	359993	PC104 20x02	4.8 €	9.6 €
2	CO3080	Tira Poste Macho Recto 80pins	0.4 €	0.8 €
1	1800677	C. Elect. 100 µF	0.3 €	0.3 €
1	1663484	LT1764AET	3.86 €	3.86 €
1	2068046	LT3015ET	3.98 €	3.98 €
2	584-AD7983BRMZ	AD7983BRMZ	23.81 €	47.62 €
1	100-964-033	DIN41612 64vías acodado	1.2 €	1.2 €
1	7553447	Regulador 3.3V salida 1A	8.9 €	8.9 €
3	2067828	AD5061BRJZ	5.66 €	16.98 €
1	CO3040	Tira poste macho recto 40pins	0.2 €	0.2 €
4	1669279	Filtro, Línea de potencia, 3.5-200MHZ	0.6 €	2.4 €
5	1248989	SMA	1.20 €	6 €
10	2309065	C tant., 10uF, 20V	0.54 €	5.4 €
27	R_SMD_1206	R genérico	0.03 €	0.81 €
10	R_SMD_0603	R genérico	0.03 €	0.3 €
36	R_SMD_0805	R genérico	0.03 €	1.08 €
3	C_SMD_1206	C genérico	0.10 €	0.3 €
78	C_SMD_0603	C genérico	0.10 €	7.8 €
				145.74 €

Tabla D.1: Costes placa de circuito impreso de los conversores

Bibliografía

[1] Cifuentes, J. C. *Plataforma para el desarrollo rápido de prototipos electrónicos a medida*. Proyecto final de carrera, Universitat Politécnica de Catalunya, 2016.

[2] Liechti, C. *pySerial*. 2001. Disponible en http://pyserial.readthedocs.io/en/latest/pyserial.html (último acceso, 2017).

[3] Grayson, J. E. *Python and Tkinter Programming*. Manning Publications, 2000. Disponible en https://www.manning.com/books/python-and-tkinter-programming.

[4] Placa de evaluación ADC. *PulSAR ADC PMODs*. Disponible en http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/PulsarPMODs.html.

[5] Placa de evaluación DAC. *EVAL-AD504X-506X*. Disponible en http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-AD504X-506X.html.

[6] Larson, S. *Serial Peripheral Interface (SPI) Master (VHDL)*. 2010. Disponible en https://eewiki.net/pages/viewpage.action?pageId=4096096 (último acceso, 2017).

No creas que estoy huyendo;
si me ves retroceder,
espera,
que estoy cogiendo carrera.

Roberto Iniesta