UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

Muise

Máster Universitario en Ingeniería de Sistemas Electrónicos

DISEÑO DE UN SOC (SYSTEM ON PROGRAMABLE CHIP) PARA LA CAPTURA DE SEÑALES ANALÓGICAS CON SENSORES DEL TIPO MOS (METAL OXIDE SEMICONDUCTOR) MEDIANTE EL ENTORNO DE LA TARJETA DE1-SOC DE ALTERA

Autor: Juan Bautista Talens Felis

Tutor : José Pelegí Sebastià
Tutora : Mª José Canet i Subiela

Trabajo Fin de Máster presentado en el **Departamento de Ingeniería Electrónica** de la **Universitat Politècnica de València** para la obtención del **Título de Máster Universitario en Ingeniería de Sistemas Electrónicos**
Curso 2019-20
Valencia, septiembre de 2019

***Resumen*—** El trabajo final de máster consiste en la captura de señales analógicas mediante la tarjeta de adquisición DE1-SoC para su uso en narices electrónicas. Se trata de la implementación hardware de un sistema completo de trabajo mediante un pequeño circuito y una tarjeta de desarrollo. Con el uso de Intellectual properties de Altera se desarrolla un conversor analógico digital para la captura de señales y su visionado en tiempo real en una pantalla.

***Índice de términos*—** Sigma-Delta,Adc,SoC,Altera,DE-1-SoC,MOS,gas sensor.

***Abstract*—** This master's thesis deals with the capture of analog signals by means of the DE1-SoC acquisition card to be used in electronic noses . It includes the hardware implementation of a complete work system through a small circuit and a development card. With the use of Altera's Intellectual Properties, a digital analog converter for signal capture and real-time viewing on a screen is developed.

***Index terms*—** Sigma-Delta,Adc,SoC,Altera,DE-1-SoC,MOS, gas sensor.

Índice de contenidos

1. Introducción

A lo largo del Máster Universitario en Ingeniería de Sistemas Electrónicos el alumnado recibe diversos contenidos que versan sobre, componentes electrónicos, sensores y tratamiento de señal. Se trabaja en diversos lenguajes de programación de software y configuración de hardware. Se plantean situaciones desde un punto de vista pragmático en donde el entorno real ejerce el protagonismo y media entre la solución teórica y la óptima. La elección de un trabajo final de Máster que englobe técnica y aprendizaje no resulta trivial. Aún más, si al mismo tiempo, permite realizar un estudio científico a aplicar a un elemento práctico del día a día y que, a su vez, asiente la base para el trabajo futuro . No obstante, el autor ha pretendido realizar un TFM centrado en un objetivo concreto pero que permita la exploración de diversos temas que, a su parecer, requieren de una mayor atención. Es por ello que se aborda la adquisición analógica de señales procedentes del medio físico pero implementando un Analog Digital Converter (ADC) usando pines digitales Low-Voltage Differential Signalling (LVDS) de una Field-Programmable Gate Array (FPGA). Esta técnica aporta la dificultad y el estudio propio de un proyecto científico y ofrece la posibilidad de estudiar profundamente una temática interesante y motivadora para el autor.

Una nariz electrónica consiste en un dispositivo capaz de capturar las variaciones de resistividad en valores de tensión con el fin de medir compuestos volátiles procedentes de un gas. Este trabajo fin de Máster parte de la idea de sustituir dos tarjetas de adquisición y un PC de laboratorio que constituyen una nariz electrónica por la tarjeta[1] DE1-SoC de TerasIC y un Monitor VGA[2] para la visualización en pantalla. Se pretende utilizar la mínima electrónica posible entre los sensores y el system on chip de Altera.

Este documento recoge el método de conversión abordado, la estrategia para la implementación en harware, el diseño de los test de verificación, las decisiones tomadas, los resultados y los pasos necesarios para la consecución de los objetivos que se describen a continuación.

1.1. Objetivos

Con el transfondo de mejorar en lo personal y profesional se plantean los siguientes objetivos:

- Realizar un conversor analógico digital mediante una FPGA.
- Convertir un sistema SoC de evaluación en un proyecto científico.
- Profundizar en herramientas avanzadas de diseño de hardware digital.
- Explorar otros lenguajes de alto nivel para el uso en sistemas embebidos.

1.2. Antecedentes

Los sensores del tipo semiconductores de metal óxido, en adelante MOS se caracterizan por variar su resistencia en función de las sustancias volátiles presentes en un gas [1]. Esto permite captar dichas variaciones en un intervalo de tiempo entre que el sensor químico reacciona en contacto con el estímulo y cuando lo pierde. Así pues, delante

[1] Se dispone de una tarjeta DE1-SoC por cortesía de un Concurso de Altera. Disponer de esta tarjeta permite abordar el proyecto a la par que se trabaja y se mantiene el cuidado de los hijos sin necesidad de desplazarse al laboratorio.

[2] VGA→ Video Graphics Array.

de una entrada analógica se pretende obtener un resultado digital para así analizar el comportamiento de determinados sensores frente a determinadas sustancias. Se trata por tanto de realizar un conversor analógico a digital. La característica principal que lo hace distinto es el hecho de que se adquiere mediante un pin digital de una FPGA y es controlado mediante un sistema embebido en un chip, en adelante SoC[2]. Se aborda así un sistema completo donde intervienen contenidos como sensores y adquisición de datos, procesado de la señal en sistemas electrónicos, procesado digital de la señal en FPGA y sistemas embebidos. El control ofrece la novedad del sistema de adquisición y amplia los contenidos de la asignatura de Sistemas Digitales Integrados.

La nariz electrónica que se pretende mejorar dispone de 32 sensores tipo MOS y es propiedad del grupo con el que colabora el autor. La etapa de adquisición contiene dos tarjetas de National Instruments de 12 y 14 bits controladas mediante un PC con LabVIEW [3]. Es por ello que a partir de la cuestión:

¿Podría una FPGA simplificar la etapa de adquisición y permitir el tratamiento de la señal?.

La tarjeta DE1-SoC [4] de Altera ofrece la posibilidad de implementar un sistema completo. Mediante un hardware capaz de realizar la adquisición, el control del proceso del tratamiento de señal y la existencia de varios estudios de adquisición de señales mediante pines digitales que se citan en el presente documento justifican el presente proyecto.

Las dos soluciones propuestas por [5] son perfectamente integrables dado que la solución SAR[3] utiliza 75 LUTS, la solución $\Sigma\Delta$ $1\,k$ LUTS y consiguen una resolución de entre 8 y 10 bits respectivamente. El estúdio teórico aportado por [6] del conversor $\Sigma\Delta$ hace preveer la posibilidad de llegar a 20 bits de resolución. A pesar que el circuito analógico que proponen es algo más complejo que un simple circuito RC se sustenta la justificación de la propuesta. Se utiliza el planteamiento de [7] con un simple circuito RC y un conversor digital analógico de un bit como se verá en la figura 5 del aparatado 2.1.2 sin utilizar una etapa de diezmado y filtrado. Así pues, el TFM contiene el estudio de las señales analógicas que se pretenden capturar, el modelo teórico o solución adoptada, el modelo físico, la configuración hardware del SoC y la programación software que permite la visualización de la señal de entrada en un visor VGA. Con ello se pretende validar la posibilidad de reducir el actual sistema. En la figura 1 puede verse el sistema completo de la nariz electrónica.

Figura 1: Sistema actual nariz electrónica.

[3]SAR→ Successive Approximation Register se explicará en el apartado 2.1.1.

1.3. Tarjeta DE1-SoC de Terasic

La tarjeta DE1-SoC es un kit de desarrollo entorno a un system on chip que combina una FPGA Cyclone V con un ARM cortex-A9 (HPS[4]). Dicha FPGA contiene $85k$ elementos programables. En la tabla 1 se detalla la FPGA que implementa la tarjeta DE1-SOC.

FPGA Device
Cyclone V SoC 5CSEMA5F31C6 Device
Dual-core ARM Cortex-A9 (HPS)
85K Programmable Logic Elements
4450 Kbits embedded memory
6 Fractional PLLs
2 Hard Memory Controllers

Tabla 1: Especificaciones FPGA DE1-SoC.

1.4. Señal de entrada

Para determinar las características del conversor analógico digital se requiere estudiar cómo es la señal de entrada. Para ello se parte de las señales procedentes del estudio previo[8] realizado por el autor para la determinación de la existencia o no de un patrón olfativo de cáncer de próstata en la orina.

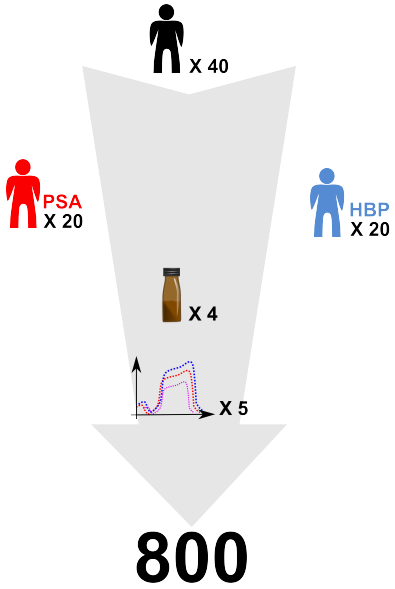

Figura 2: Origen de las muestras y curvas obtenidas.

En la figura 2 puede verse un resumen del procedimiento realizado para la obtención de curvas de respuesta de los sensores ante la orina. Inicialmente se obtuvieron $15ml$ de orina de 40 pacientes del Hospital Universitario La Fe de Valencia con niveles

[4]HPS→Hard processor system.

elevados de PSA[5]. La mitad de las muestras corresponden a cáncer de Próstata y la otra mitad a Hiperplasia benigna de próstata (HBP)[6]. Para aumentar el número de curvas se reparte el contenido de las muestras en cuatro recipientes y se realizan cinco adquisiciones de datos para cada recipiente con lo que se consiguen 800 curvas de señal adquiridas.

En la figura 3 se observa que las señales mantienen el rango positivo y son inferiores a los 2 Voltios. Además se observa que los cambios se manifiestan lentamente por lo que la respuesta del sensor requiere de más de 150 segundos para volver a su estado anterior. Otra característica de este tipo de sensores es el offset que experimenta. Como puede verse en la figura 3 a pesar de tener sensores iguales tienen offset diferentes.

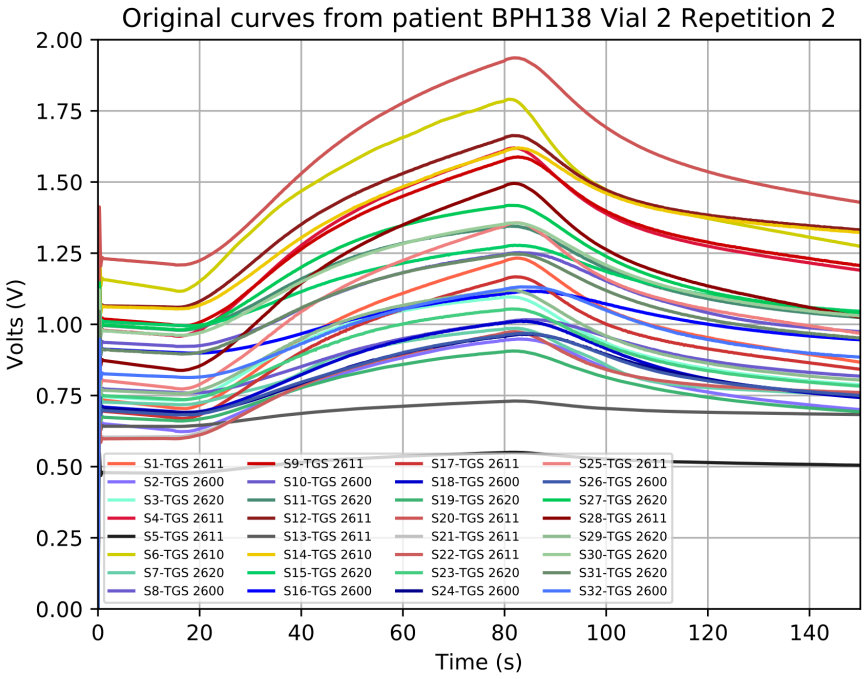

Figura 3: Curvas de un paciente extraidas con el eNose de 32 sensores denominada MOOSY32.

La nariz electrónica de 32 sensores utiliza una frecuencia de muestreo de $100\,Hz$. Las señales a capturar no deben superar el límite de Nyquist por lo que no deben superar los $50\,Hz$.

1.5. Señal objetivo

Debido a las características de las señales procedentes del eNose y a la frecuencia de muestreo deseada podría ser suficiente con una resolución de 10 bits. Las señales capturadas del eNose se truncaron a dos decimales, tienen frecuencias entorno a los $15\,mHz$ por lo que los incrementos de tiempo conllevan saltos en señal con una resolución que no superan los 10 bits. No obstante, siempre resulta interesante disminuir el

[5]PSA→Análisis del antígeno prostático específico.
[6]HBP→Trastorno benigno (no canceroso) en el que el tejido de la próstata crece en exceso.

ruido de quantificación. Dado que se dispone de un generador de señal de 16 bits y como se justifica en el apartado 2.1.2, se decide realizar un conversor analógico digital de 16 bits sin signo. Con ello se pretende alcanzar, al menos, 10 bits de precisión. Por tanto se tiene un formato $[16,14]$ con rango digital $[0,65535]$ y valores teóricos de voltaje:

$$Rango : [0,(2^N - 1)Q] \rightarrow [0,3{,}99994]\,V \tag{1}$$

$$Resolución : Q = 2^{-b} = \frac{1}{16384} \approx 61{,}04\,\mu V \tag{2}$$

1.6. Diseño del Test

Para la consecución de los objetivos, en concreto, de la realización de un conversor analógico digital mediante una FPGA, se requiere diseñar un Test capaz de poner a prueba el sistema y donde se observe su correcto funcionamiento.

El presente proyecto se enfoca básicamente en dos modelos de diseño. Un modelo desarrollado mediante capas que ofrece una solución en tiempo real y otro de post-producción que verifica y caracteriza la propuesta escogida.

1.6.1. Modelo de capas

Para un correcto desarrollo de un sistema completo se recurre a un modelo de capas donde cada capa interviene con el conjunto ofreciendo servicios a las capas superiores. De este modo se abordan las capas como elementos a desarrollar independientes. Dividiendo la propuesta en objetivos concretos con independencia de la tecnología a utilizar y el grado de desarrollo de la misma.

Aplicación: La capa de Aplicación se encarga de mostrar los datos al usuario y de guardarlos en un archivo para su posterior tratamiento. Esta recibe los datos de la capa de lectura. Para el test se requiere la inspección visual de los datos presentados en pantalla en tiempo de adquisición así como guardarlos en un archivo para su posterior análisis.

Lectura: La capa de Lectura se encarga de garantizar el acceso a los datos en tiempo y forma y ofrecer estos datos a la capa de Aplicación. Para el test se requiere el uso de salidas por consola y checkpoints en la depuración del software durante las pruebas de integración y validación.

Enlace: La capa de enlace se encarga del modelo físico que permite la transducción de un entorno real al digital ofreciendo datos de forma confiable a la capa de Lectura. Para el test se requiere del uso de modelos de test que simulen los estímulos del entorno real. El análisis requiere de analizadores lógicos que permitan la visualización de los datos.

Generador: La capa Generador se encarga de proporcional la señal a capturar en óptimas condiciones a la capa de Enlace. Para el test se utilizan el Osciloscópio y el Generador de Funciones como elementos principales.

En la figura 4 se observa el modelo de capas. Tras introducir los parámetros en el generador e iniciar la aplicación ésta devuelve un gráfico en tiempo real y, tras la captura de $150\,s$, un archivo que contiene los datos para su posterior procesado.

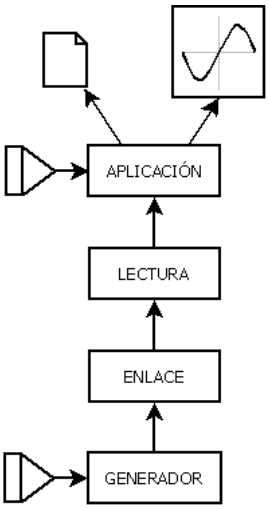

Figura 4: Modelo de Capas.

1.6.2. Modelo Postproductivo

Tras la interacción del usuario en el modelo de capas se requiere un modelo post-producción que se encarga de comparar los datos con funciones ideales y el cálculo de características intrínsecas con el fin de caracterizar el conversor diseñado. En el presente trabajo final de Máster se ha utilizado Matlab© para el análisis tras la obtención del archivo con los datos capturados.

2. Capa de Enlace. Conversión ADC

En la capa de enlace toma especial relevancia la conversión de los datos analógicos al dominio digital. En este apartado se tratan los dos métodos de conversión más comunes explicando en detalle la solución escogida para el presente trabajo final de Máster.

2.1. Métodos de conversión

Para la conversión analógica a digital se han estudiado dos métodos; el conversor del tipo sigma delta $\Sigma\Delta$ [6] y el registro de aproximaciones sucesivas (SAR) [5].

2.1.1. Aproximación sucesiva SAR

El método SAR consiste en la aproximación sucesiva del valor capturado. Se realiza una comparación del valor de entrada empezando por el valor medio de la escala y si se obtiene un valor mayor o menor se devuelve un decremento o incremento respectivamente. Con este método se consigue con *n* iteraciones capturar el valor deseado. Siendo *n* el número de bits del conversor. Este método requiere del uso de un conversor digital analógico, en adelante DAC, de n bits.

A pesar que resulta óptimo para su utilización en el e-Nose de 32 sensores, por su baja latencia y alta tasa de muestreo, se descarta inicialmente por requerir de un conversor digital-analógico de 16 bits. La única posibilidad de no incorporar hardware adicional sería la de utilizar la salida de audio del la DE1-SoC. Pero en ese caso, ¿por qué no utilizar la entrada y la salida jack?. El propósito radica en demostrar la posibilidad de

utilizar tecnología digital para la captura de datos analógicos en un sistema onchip que permita el manejo de los datos desde un sistema operativo. En esencia, trabajar con una FPGA-HPS realizando un ADC para su escalado en otros SoCs. No tendría sentido utilizar el conversor ADC que incorpora la própia tarjeta o, en este caso, el DAC. Si se cambia de tarjeta puede que no se disponga del mismo hardware y ya no sería apropiada la solución propuesta.

2.1.2. Sigma Delta $\Sigma\Delta$

El médoto de conversión $\Sigma\Delta$ utiliza tres etapas o bloques fundamentales donde la señal se suma, integra y se compara con la salida de un DAC de un bit [7] . Se utiliza una tasa de muestreo alta para intercambiarla por resolución. Este método simplifica el hardware a utilizar pero aumenta la latencia, no obstante, reduce la tasa de muestreo a cambio de una mayor resolución. Si se dispone de un reloj de $50Mhz$ y, como en el eNose, una señal muestreada a $100Hz$ se podría pretender una resolución[7] de 18 bits. No obstante, en el MOOSY32, al multiplexar con 32 sensores únicamente se puede pretender una resolución de 13 bits. El presente trabajo trata de demostrar la viabilidad o no de este tipo de tarjeta. Es por ello que se opta por una resolución objetivo intermedia que permita el escalado en número de sensores mayor a uno y mantega un número de bits suficiente. Todo ello independientemente de los posibles errores de cuantificación que puedan surgir. Así pues se fija como pretensión una resolución de 16 bits.

Para la conformación del conversor se utiliza un pequeño circuito RC y el uso de dos entradas LVDS así como un registro digital. De la comparación de la salida de 1-Bit tras el filtro analógico saldrán los pulsos necesarios para la cuantificación. La suma reiterada a una frecuencia f_H y el borrado[8] a f_L generan la salida registrada a la frecuencia de muestreo deseada.

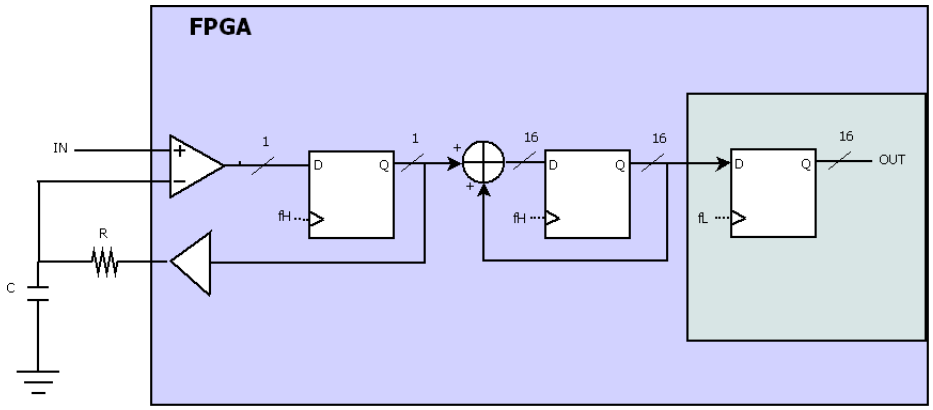

Figura 5: Conversor $\Sigma\Delta$ a implementar en FPGA.

En la figura 5 se pueden observar en distinto color los dos dominios de reloj dentro de la FPGA así como el circuito RC.

$$f_H = f_s \cdot 2^N \tag{3}$$

$$f_L = f_s \tag{4}$$

[7] Con $log_2(\frac{50e6}{100}) \approx 18{,}9$ bits.

[8] Si el acumulador no ha llegado a la cuenta máxima se requiere introducir un cero tras alcanzar el límite de 65535.

La tarjeta DE1-SoC permite trabajar con frecuencias de $50\,MHz$ por tanto la frecuencia de trabajo no sería un problema. La limitación es más bién física dado que se requieren 32 entradas analógicas. A pesar que la Cyclone V, que incorpora la tarjeta, permite entradas LVDS en los bancos de I/O está condicionada por los elementos montados en la misma. El uso del componente LVDS en el GPIO-0 condiciona todo el banco a usar tensiones de $2,5\,V$ haciendo imposible el uso de la memoria SDRAM que incorpora la tarjeta dado que trabaja a $3{,}3\,V$. No obstante no resulta crítico ya que el procesador utiliza la memoria DDR incorporada en la tarjeta como memoria principal. A continuación se describen los elementos que forma el conversor $\Sigma\Delta$ implementado en la FPGA.

2.1.2.1. LVDS El funcionamiento del comparador de la entrada LVDS es similar al de un amplificador operacional. Tiene una entrada inversora, otra no inversora y una salida como puede verse en la figura 6 . Trabaja de forma no lineal devolviendo la comparación de las dos entradas [9]. Se fabrican con cierta histéresis debido a que su propósito es ofrecer una salida estable a variaciones pequeñas de tensión. Esto produce una disminución de su resolución [10]. El threshold o tensión diferencial de entrada mínima requerida por el componente y descrita en [11] para una tensión de modo común de $1{,}25\,V$ es de $100\,mV$.

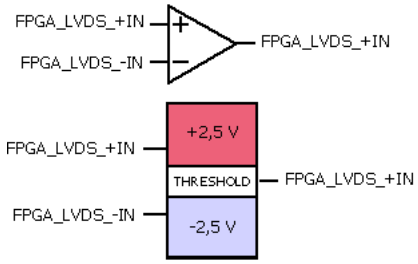

Figura 6: Etapa LVDS.

La configuración de dos pines de uno de los dos puertos GPIO es el elemento clave para el presente trabajo final de Máster. Estos deben emparejarse recíprocamente y se límita la tensión por la tecnología[9] a $2{,}5\,V$. Esta circunstancia afecta al rango de tensiones. Si se tiene un rango digital de $[0, 65535]$ con una tensión de referencia máxima de $2{,}5\,V$ la máxima tensión alcanzable no puede ser la descrita en la ecuación 2 sino el resultado de dividir el dato digital por 2^N y multiplicar por dicha tensión $2{,}49996\,V$

El elemento LVDS, usado en recepción y mediante una terminación resistiva para la adaptación de la línea, rechaza el ruido en modo común y tiene un comportamiento diferencial [12]. En este trabajo la señal de entrada analógica se resta con la señal que proviene del conversor digital analógico de 1 bit, que no es otro, que el resultado de registrar la señal comparada y conectarla al circuito RC de la figura 7.

La señal que proviene del DAC-1bit o dominio digital, a la salida de la FPGA, corresponde a una señal cuadrada modulada en frecuencia en función del resultado de la comparación. El comparador, con la ayuda del filtro RC, se encarga de utilizar la señal filtrada retrasada y restarla con la señal de entrada. El resultado es la pendiente de la función a digitalizar. Con ello se consigue el potencial positivo o negativo que genera los pulsos del DAC-1bit.

En el circuito de entrada, la integridad de la señal está afectada por el ruido. Como es común para las entradas es rechazado por el LVDS.

[9] Los niveles de señal de entrada o salida de señalización diferencial de bajo voltaje para LVDS se definen por la Asociación de la Industria de las Telecomunicaciones ANSI/TIA/EIA-644.

2.1.2.2. Diseño del filtro RC El filtro se diseña para que atenúe las frecuencias superiores a la frecuencia de muestreo final que se pretende digitalizar la señal. Con ello se consigue, por una parte, transformar la salida del DAC-1 bit en una señal similar al original y por otra mantener lineal la fase entorno a la frecuencia de muestreo. Si se

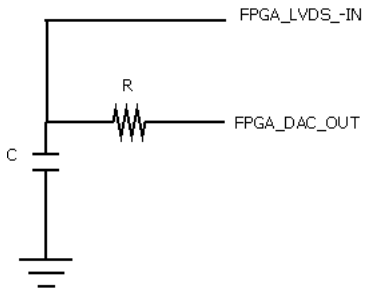

Figura 7: Estapa filtro RC.

parte de un filtro paso bajo normalizado, la constante de tiempo de un condesador para una frecuencia de corte de $100\,Hz$ se tiene:

$$\tau = \frac{1}{\omega} = \frac{1}{2\pi f_s} = \frac{1}{2\pi \cdot 100} \approx 1,59\,ms \tag{5}$$

Se pretende utilizar un condensador de $C = 150\,nF$ por tanto se calcula R para que no afecte el cambio de condensador al valor de τ visto en la ecuación 5.

$$R = \frac{C}{C'\omega} = \frac{1}{150nF \cdot 200\pi} \approx 10,61\,k\Omega \tag{6}$$

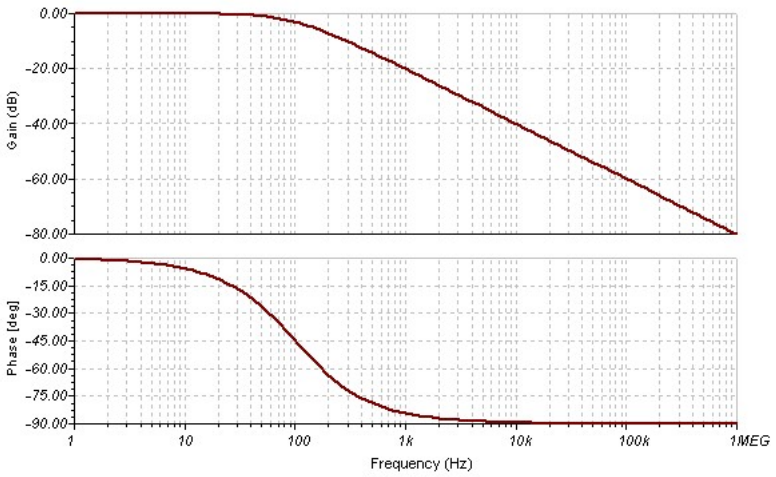

Figura 8: Repuesta filtro polo en $100\,Hz$, $R = 10,61\,k\Omega$, $C = 150\,nF$.

En la figura 8 puede verse que existe una atenuación de $-3\,dB$ para la frecuencia de muestreo y una zona lineal en fase entorno a f_s.

2.1.2.3. DAC-1 bit Se requiere comparar la señal de entrada con otra que se aproxime para así generar los estímulos necesarios entorno a una entrada digital. Posteriormente se integran dichos valores en una señal de mayor precisión.

Tal y como vemos en la figura 9 el convertidor digital a analógico de un bit consiste en un registro a la frecuencia f_H conectado al buffer de salida de la FPGA. El pin de salida

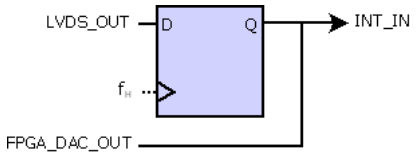

Figura 9: Etapa DAC-1bit.

se ha límitado a $2,5\,V$ para no superar la tensión de entrada del LVDS. En la figura 10

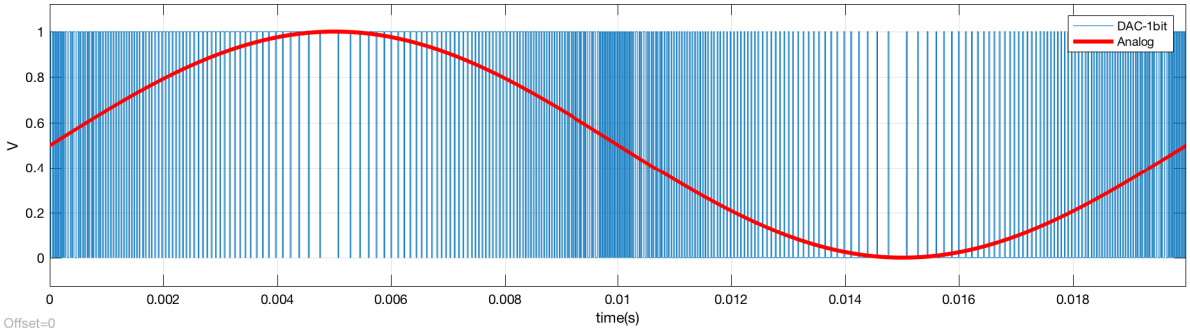

Figura 10: Entrada señal senoidal $v(t) = 0,5 + 0,5\sin(2\pi 50t)$ salida DAC-1-bit de $(0-1)\,V$, con $f_H = 65,536\,kHz$.

se ha usado un seno como señal analógica de entrada con valores de amplitud, offset y frecuencia para hacerla más representativa. Nótese el modulado del pulso en frecuencia de la señal de salida del conversor siguiendo la pendiente de la señal de entrada analógica.

2.1.2.4. Integrador El integrador está compuesto por un acumulador de 16 bits y un registro con realimentación al sumador. En la figura 11 puede verse como se recoge la señal del DAC de 1 bit INT_IN y se suma con al salida registrada del instante anterior. Se limita el crecimiento del sumador a 16 bits para que la cuenta vuelva a empezar tras alcanzar el valor de 65535 dado que este valor es el máximo valor de escala. Para las cuentas inferiores se realiza un borrado del registro cada $1/f_s$. Nótese que la entrada al sumador es de únicamente un bit. Por este motivo no se deja crecer el acumulador $\log_2(65536)$ ya que no alcanza el valor máximo hasta llegar al valor de fondo de escala.

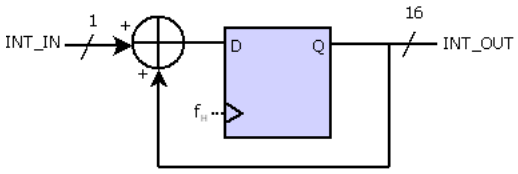

Figura 11: Etapa Integrador.

2.1.2.5. SAMPLE & HOLD En la figura 12 puede verse como se recoge el valor alcanzado por el integrador mediante un registro síncrono a la frecuencia f_s.

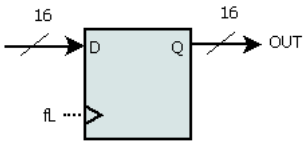

Figura 12: Etapa S & H.

3. Implementación en FPGA

La implementación en la FPGA no puede realizarse directamente sin antes verificar el modelo mediante un banco de pruebas. En la figura 13 se presenta el esquema a seguir para verificar el modelo. El testbench utiliza una señal de entrada producida mediante una simulación y capturada a un archivo. Los datos contenidos en el archivo constituyen los estímulos de entrada (data_dig). Mediante herramientas de compilación libres se verifica el modelo.

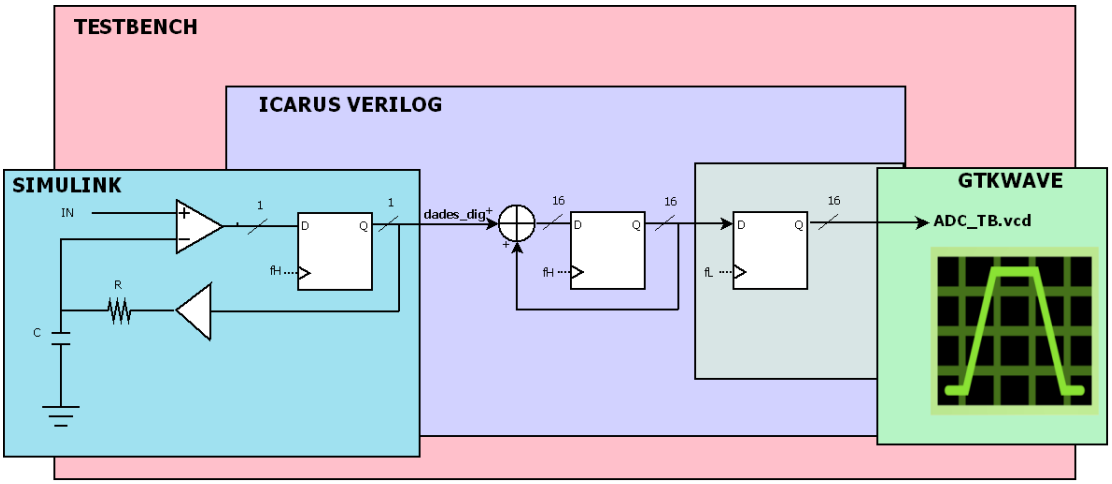

Figura 13: Esquema banco de pruebas del test inicial.

3.1. Verificación inicial del conversor $\Sigma\Delta$

Para la primera prueba se requiere de un comparador y un conversor digital a analógico de un bit. La dificultad radica en simular el dominio analógico con el digital.

3.1.1. Preparando el modelo

Para el estudio de la implementación del conversor $\Sigma\Delta$ en una FPGA se realizan simulaciones mediante Matlab© y Simulink© con la Toolbox de Simscape. En la figura 14 se observa el modelo completo que simula el comparador y el DAC-1bit.

El generador de la figura 14 proporciona una señal senoidal positiva:

$$v(t) = 1{,}25 + 1{,}25\sin(2\pi t) \tag{7}$$

Con la señal vista en 7 se optiene un máximo a $2{,}5\,V$ que corresponde al límite de escala[10] y una frecuencia de $1\,Hz$ para poder trabajar entre simulaciones sin demasiada demora y trabajar con una frecuencia similar a las frecuencias de la nariz electrónica.

[10] Se ha establecido un límite de escala en $2{,}5\,V$ proporcionado por el buffer de salida del DAC.

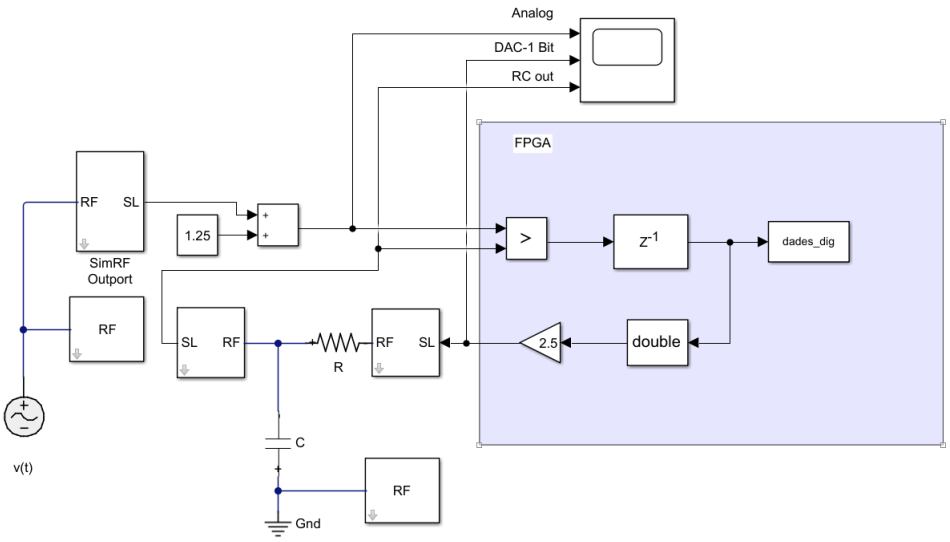

Figura 14: Modelo generador de señal a archivo de texto salida DAC 1-Bit.

La señal generada es comparada y se obtiene un bit como resultado como puede observarse en la Figura 15. Este bit se registra en un archivo de texto para su posterior uso en el techbench de verilog. Para los valores de R y C se han utilizado los calculados en el apartado 2.1.2.2. Para la simulación se utiliza el parámetro *Step Size* como $1/f_s$, *start* cero y *stop* uno con el *solver* en *Fixed-Step* y *auto* . El dato se guarda en decimal como 0 ó 1 línea a línea. La frecuencia de trabajo es 65536 veces la frecuencia de muestreo deseada como puede verse en el código siguiente:

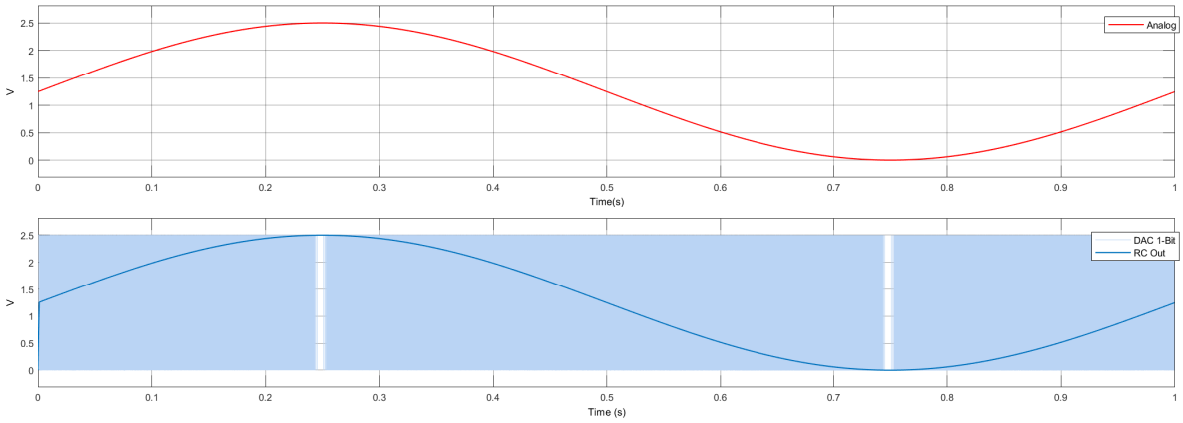

Figura 15: Señales de entrada, salida y comparación.

```
clear
format long
fs=6553600
open('Entradaanalogica.slx');
sim('Entradaanalogica.slx');
fid = fopen('salidaDAC.txt','wb');
fprintf(fid,'%d\n',dades_dig);
fclose(fid);
```

3.1.2. Modelo Verilog

Para la generación del modelo se utiliza Icarus Verilog[11] y al mismo tiempo se comprueba la compilación del modelo en ©Quartus. Este modelo no incluye el LVDS, el DAC-1 bit y el circuito RC ya que se utiliza como entrada el archivo obtenido con ©Simulink.

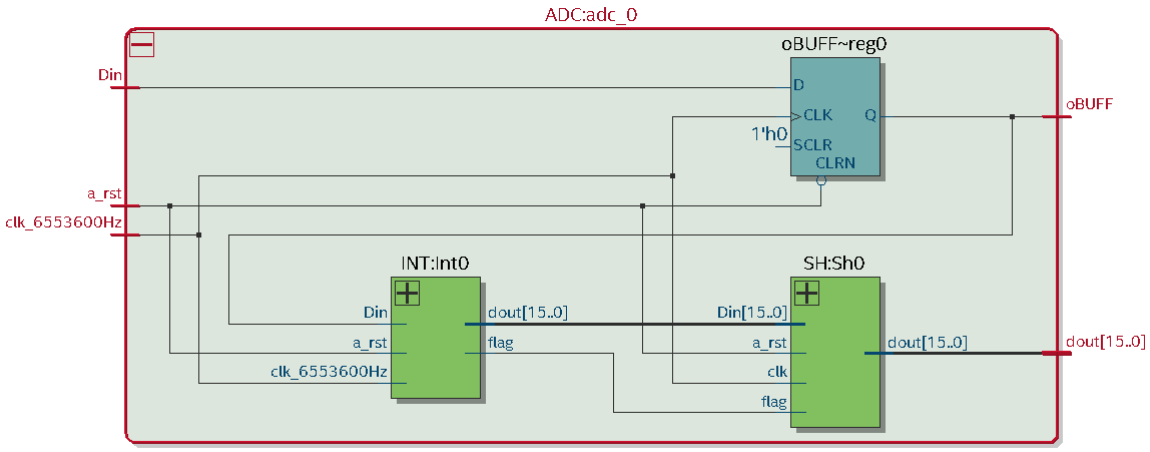

Figura 16: RTL del modelo ADC.v en ©Quartus.

```
module ADC
  #(parameter Nd=16,
  parameter Ng=0)
  (input Din,
  input  a_rst,clk_6553600Hz,
  output reg oBUFF,
  output [Nd-1+Ng:0] dout);

  wire [Nd-1+Ng:0] s0;
  wire [Nd-1+Ng:0] s;
  wire f;
  wire in;
  wire clk_100Hz,TC;
  assign  dout= (s[Nd-1+Ng:0]);
  assign in=oBUFF;

  always@(negedge a_rst, posedge clk_6553600Hz)
  if(!a_rst)
  oBUFF<=1'b0;
  else
  oBUFF<=Din;

  INT #(16,0) Int0  (in,clk_6553600Hz,a_rst,f,s0);
  SH #(16,0) Sh0  (s0, clk_6553600Hz,a_rst,f,s);
endmodule
```

En la figura 16 puede verse la representación del modelo Verilog en RTL desde la entrada al sumador del integrador *Din* hasta la salida interna a la FPGA *dout*. La señal flag se activa cuando el contador a alcanzado la cuenta de 65536.

[11] Icarus Verilog es una implementación del lenguaje de descripción de hardware Verilog. Admite las versiones de 1995, 2001 y 2005 del estándar, partes de SystemVerilog y algunas extensiones.

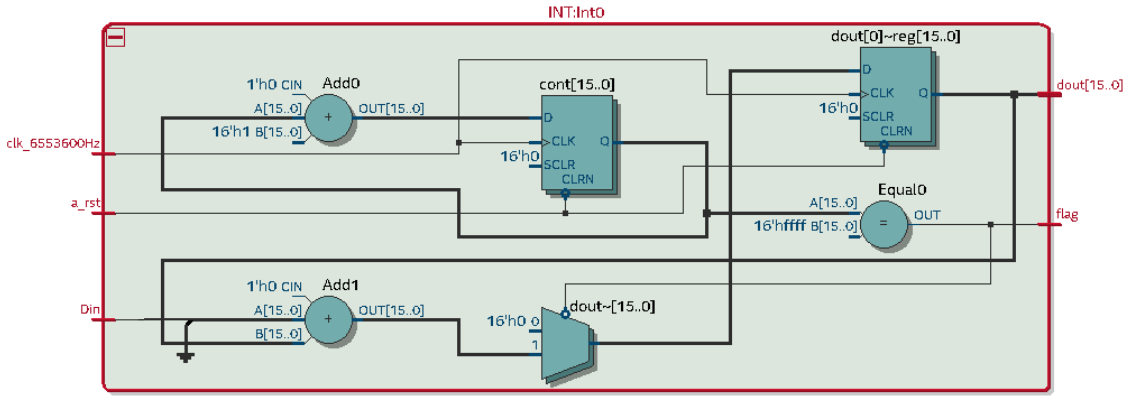

Figura 17: RTL Modelo ADC.v Integrador.

```
module INT
#(parameter Nd=16,
  parameter Ng=0)
  (input Din,
   input clk_6553600Hz, a_rst,
   output flag,
   output reg [Nd+Ng-1:0] dout);

  reg [15:0] cont;

  assign flag=(cont==16'd65535)?1'b1:1'b0;

  always@(negedge a_rst, posedge clk_6553600Hz)
  begin
  if(!a_rst)
  cont <=16'd0;
  else
  cont<=cont+1'b1;
  end


  always@(negedge a_rst, posedge clk_6553600Hz)
     if (!a_rst)
    dout<=16'd0;
    else
    if(!flag)
      dout<=Din+dout;
    else
      dout<=16'd0;
endmodule
```

La etapa del integrador requiere de un contador para borrar el dato sumado cuando la cuenta ha llegado a su fin pero la señal no ha alcanzado el máximo. Se ha limitado el crecimiento a 16 bits puesto que el máximo acanzable es de 65535.

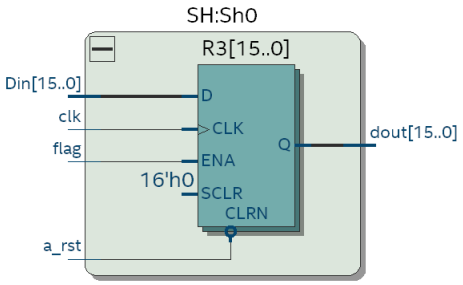

Figura 18: Módulo ADC.v Sample and Hold.

```
module SH
#(parameter Nd=16,
  parameter Ng=0)
  (input  [Nd-1+Ng:0] Din,
   input clk, a_rst, flag,
   output  [Nd+Ng-1:0] dout);

  reg  [Nd+Ng-1:0] R3;
  assign dout=R3;

  always@(negedge a_rst, negedge clk)
      if (!a_rst)
        R3<=0;
      else
        if(flag)
          R3<=Din;
        else
          R3<=R3;

endmodule
```

El módulo de sample and hold realiza un cambio de tasa mediante el uso de un Enable (flag) a f_s.

3.1.3. Verificando el modelo

Mediante un Testbench se generan los estímulos para verificar el funcionamiento del modelo verilog. Para ello se toma como entrada del diseño en verilog la salida del modelo de comparador, RC y DAC-1 bit (salidaDAC.txt).

```
`timescale 1ns / 1fs

module TB_ADC ();
parameter PER_6553600Hz=152.587890625;
parameter PER_100Hz=PER_6553600Hz*65536;
reg a_rst, clk, clk6553600Hz,clk_100Hz, Din;
wire oBUFF;
wire [15:0] dout;
wire clock_100_clk;
```

```
integer unsigned scan_in;
integer unsigned data_in;
reg [32:0] str;
ADC #(16,0) adc(Din, a_rst, clk6553600Hz,oBUFF, dout);

always #(PER_6553600Hz/2) clk6553600Hz = !clk6553600Hz;
always #(PER_100Hz/2) clk_100Hz = !clk_100Hz;

initial data_in = $fopen("./salidaDAC.txt", "r");


initial begin
  $dumpfile ("./ADC_TB.vcd");
  $dumpvars( 0, TB_ADC );
  {a_rst,clk,clk_100Hz,clk6553600Hz,Din}=0;
  #PER_6553600Hz a_rst=1;
  //Din=0;
      repeat (6553600)
    begin
    @(posedge clk6553600Hz)
    scan_in= $fscanf(data_in, "%x\n", Din);
      if ($feof(data_in))
        $fclose(data_in);
    end
$finish;
end
endmodule
```

Para conseguir la frecuencia de funcionamiento $f_H = 6{,}553\,MHz$ se requiere un *timescale* con una resolución de 10^{-18} para tener $152{,}587890625\,ns$. El compilador tiene una épsilon de 10^{-15} por lo que se produce un error en el ciclo de reloj que ahora es de $152{,}587890\,ns$.

$$E_a = \left| T_H - \widetilde{T_H} \right| = 6{,}25 \cdot 10^{-16}\,s \tag{8}$$

El testbench se simula mediante la compilación en Icarus Verilog y su representación mediante GkWave con las siguientes líneas de código:

```
$ iverilog -o salida.out TB_ADC.v ADC.v
$ vvp -n salida.out
$ gtkwave -o ADC_TB.vcd
```

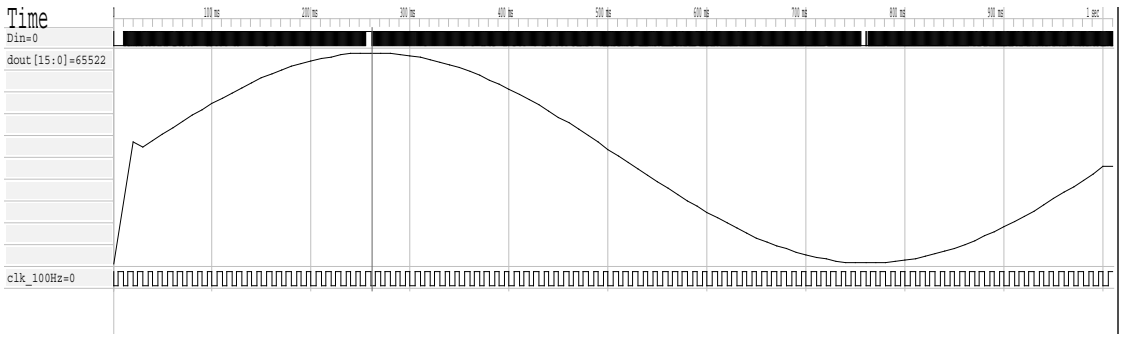

Figura 19: Techbench TB_ADC.v Señal $v(t) = 1,25 + 1,25\sin(2\pi t)$.

La señal de entrada vista en la ecuación 7 alcanza el valor máximo de tensión ofrecido por el buffer de salida de la FPGA y el admisible por el puerto LVDS de $2,5\,V$. Este valor

corresponde al tope de escala que debería ser 65535 dado que en 65536 cuentas se representa del $[0-65535]$.

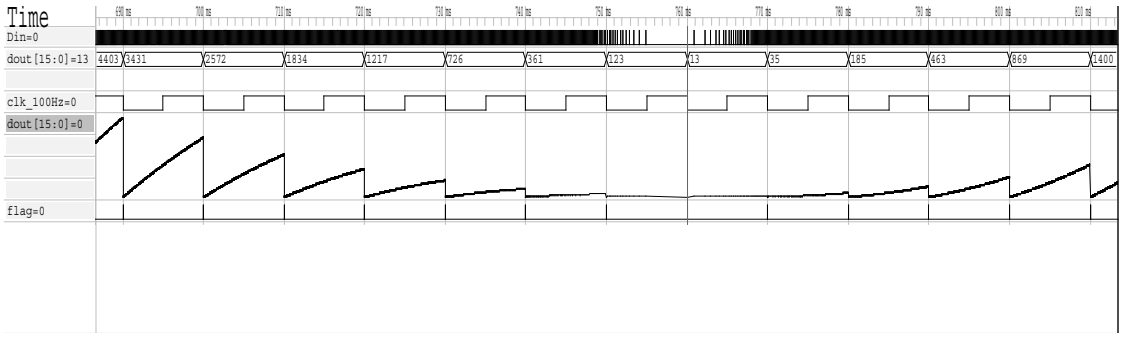

Figura 20: Techbench TB_ADC.v Detalle dout Integrador.

En las figuras 19 y 20 se observa que la salida no alcanza ni el máximo 65535 ni el mínimo 0 deseado. En su lugar se ontiene un rango de $[13-65522]$.

En la figura 20 se observa en el cursor que a la salida del integrador sí se alcanza el mínimo. No obstante, dado el registro del sample & hold, en el momento de la captura del dato éste se ha superado.

En la figura 21 se observa que el máximo alcanzado és de 65296. El rango obtenido en

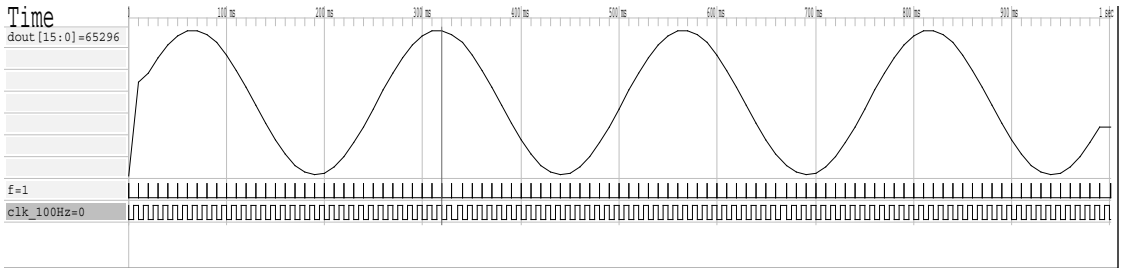

Figura 21: Techbench TB_ADC.v Señal $v(t) = 1,25 + 1,25\sin(2\pi 4t)$.

este caso es de [74-65296].

En la comparación de la señal analógica con la salida del DAC-1 bit, por los límites

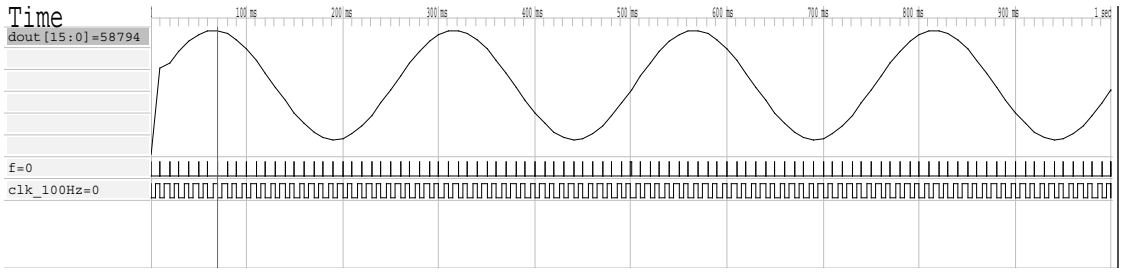

Figura 22: Techbench TB_ADC.v Señal $v(t) = 1,25 + 1\sin(2\pi 4t)$.

superior e inferior, podría estar produciéndose un error debido al retraso entre las dos señales. Se modifican los parámetros del generador para introducir una señal que no alcance ni el máximo ni el mínimo como por ejemplo $v(t) = 1,25 + 1\sin(2\pi 4t)$. Esta señal tiene el máximo en $2,25\,V$ y el mínimo en $250\,mV$. En la figura 22 puede verse el marcador que alcanza el valor decimal de 58794. El mínimo se ha obtenido en 6610. Estos valores corresponden en voltios a $2,2428\,V$ y a $252,2\,mV$ por tanto se entiende por vali-

dado el modelo y se procede a continuar con la implementación.

3.2. Generación de componentes

Mediante el uso de la herramienta *Platform Designer* se generan los componentes necesarios para la implementación en la FPGA. Se genera un archivo que deberá añadirse al proyecto para poder ser instanciado correctamente. Así pues se requiere de:

- Señal de Reloj de la FPGA.
- Señal de Reloj a f_H.
- Componente ADC.
- Componente adicional de salida.
- Modelo qsys.
- Generación de IP's.

3.2.1. Señal de Reloj de la FPGA

Se disponen de varias señales tal como se recoje en el manual de la DE1-SoC y que pueden verse en la tabla 2.

Se utiliza el elemento clk por defecto con la configuración de la figura 23.

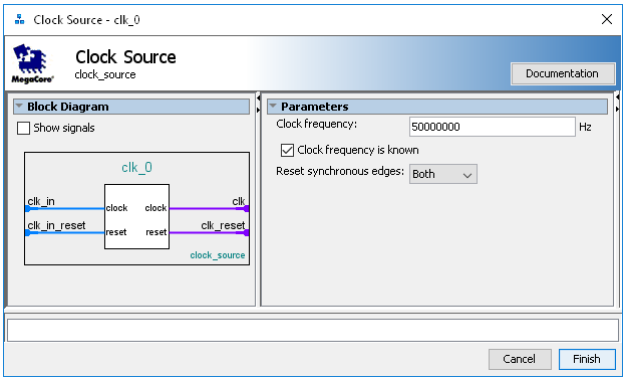

Figura 23: Componente de entrada de reloj de la DE1-SoC a $50 MHz$.

Nombre de señal	PIN FPGA	Descripción	I/O
CLOCK_50	PIN_AF14	Entrada reloj a $50 MHz$	$3,3\,V$
CLOCK2_50	PIN_AA16	Entrada reloj a $50 MHz$	$3,3\,V$
CLOCK3_50	PIN_Y26	Entrada reloj a $50 MHz$	$3,3\,V$
CLOCK4_50	PIN_K14	Entrada reloj a $50 MHz$	$3,3\,V$
HPS_CLOCK1_25	PIN_D25	Entrada reloj a $25 MHz$	$3,3\,V$
HPS_CLOCK2_25	PIN_F25	Entrada reloj a $25 MHz$	$3,3\,V$

Tabla 2: Relojes en la DE1-SoC.

3.2.2. Señal de Reloj a f_H

Mediante la herramienta *Platform Designer* se genera el componenente pll. En la figura 24 se observan los parámetros utilizados para conseguir el reloj a $6{,}5536\,MHz$.

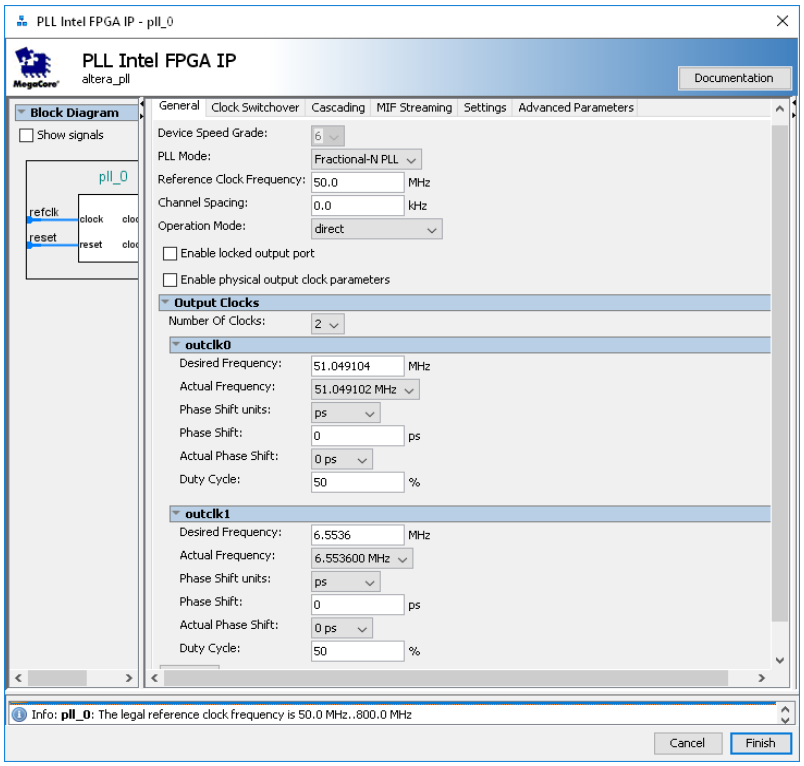

Figura 24: Configuración PLL $6{,}5536\,MHz$.

3.2.3. Componente Adc

Para generar el componente se requiere del archivo ADV.v que se incluye durante el proceso de creación. Es necesario crear las entidades del tipo *conduit* para los puertos de entrada y salida.

En la figura 25 se muestra la pestaña *Files* donde se añade la ruta del archivo y se establece como atributo *Top level File*. En la pestaña de *Signal & Interfaces* se generan las Interfaces siguientes:

- Din
- reset
- clk_6553600Hz

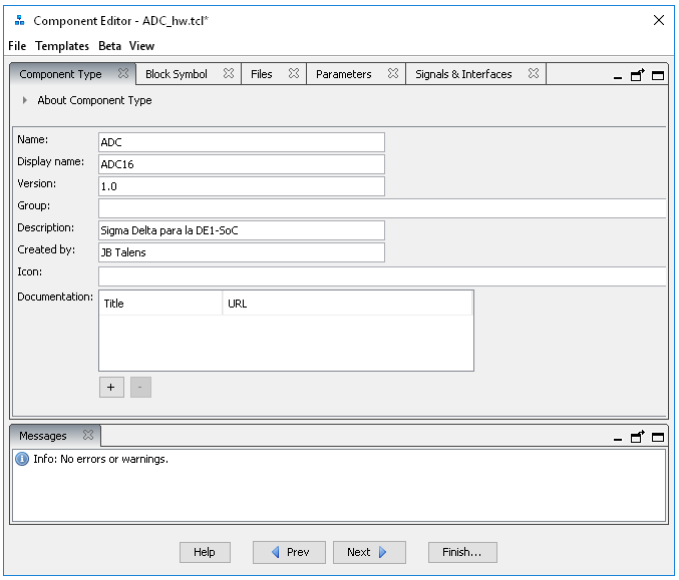

Figura 25: Componente ADC mediante Platform Designer.

- oBUFF
- dout

A cada interfaz se le añade la señal de entrada o salida corresponediente atendiendo a los puertos del modelo ADC.V.

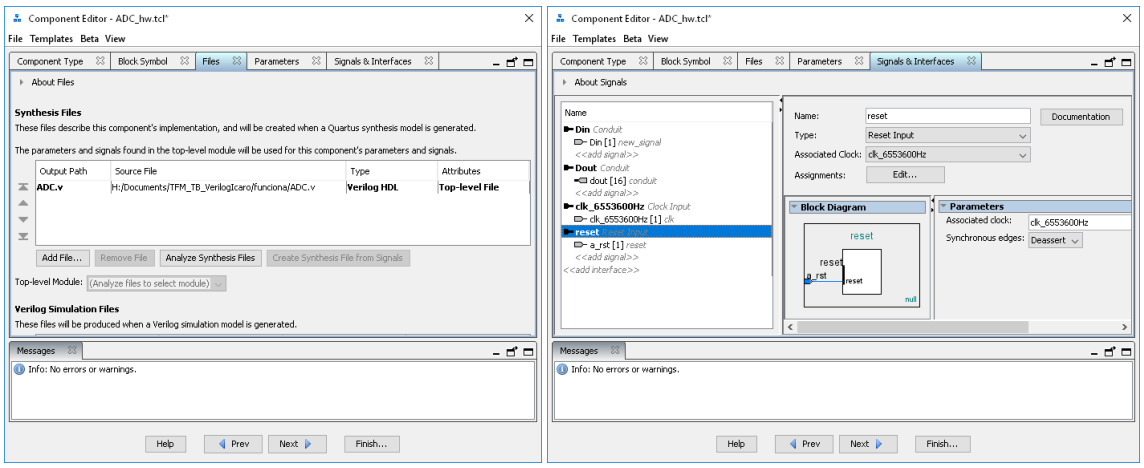

Figura 26: Componente ADC mediante Platform Designer.

3.2.4. Componente adicional de salida

Para la captura del dato se genera un componente paralelo de I/O, en este caso de entrada, con 16 bits. Este componente es de entrada debido a que recoge el dato de la salida del conversor ADC para ser leído por el sistema operativo. Esto es posible gracias al mapeado en memoria del componente dentro del bus de sistema.

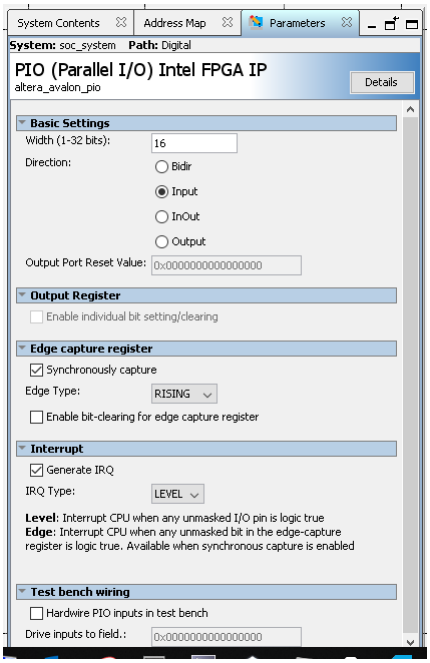

Figura 27: Componente de entrada de 16 bits.

3.2.5. Modelo qsys

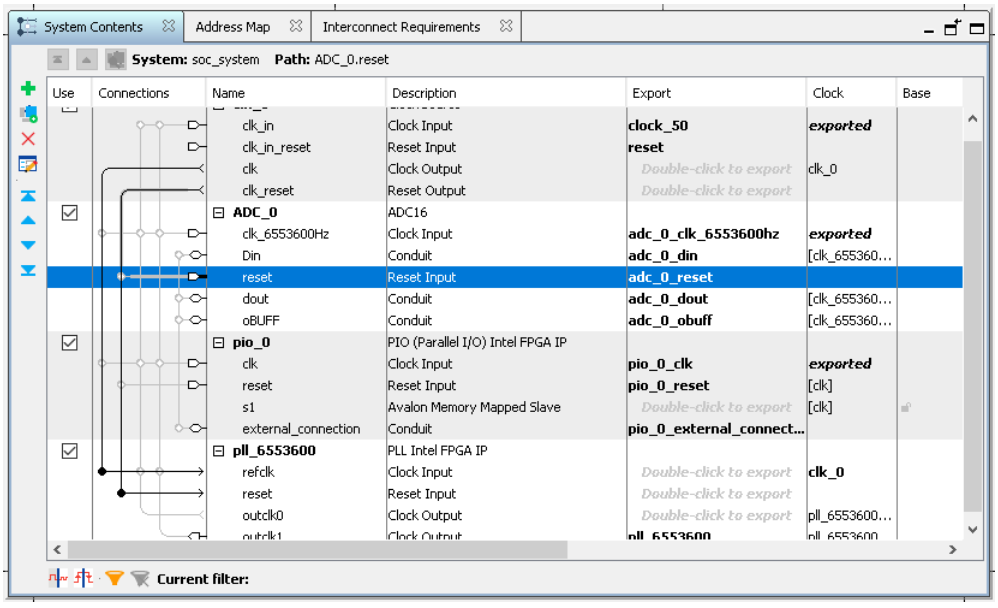

Figura 28: Detalle sistema qsys.

Nombre de señal	PIN FPGA	Descripción	I/O
GPIO0_[0]	PIN_AC18	PIN GPIO 0[0]	LVDS
GPIO0_[2]	PIN_AD17	PIN GPIO 0[2]	$2,5\,V$
GPIO1_[0]	PIN_AB17	PIN GPIO 1[0]	$2,5\,V$

Tabla 3: Puertos GPIO en la DE1-SoC.

El reloj sel sistema *clk* así como el reset se transfieren al componente *pll* y los demás estímulos se exportan para su conexión mediante la entidad top del proyecto.

Tras generar el componente qsys se instacia de la siguiente forma:

```
soc_system u0 (
      .clock_50_clk                    (CLOCK_50),
      .reset_reset_n                   (KEY[0]),
      .adc_0_din_new_signal            (lvds_out),
      .adc_0_reset_reset               (KEY[0]),
      .adc_0_clk_6553600hz_clk         (clk6553600),
      .adc_0_obuff_new_signal          (nbuffer),
      .adc_0_dout_conduit              (dout),
      .pio_0_clk_clk                   (clk6553600),
      .pll_6553600_clk                 (clk6553600),
      .pio_0_reset_reset_n             (KEY[0]),
      .pio_0_external_connection_export (dout)
    );
```

3.3. Generación IPs

Se requieren de un buffer de entrada para la entrada LVDS y uno de salida para la salidad del DAC-1bit que se gerenan mediante el catálogo de IPs de altera.

En la tabla 3 se observa la relación de pines y la asignación a realizar con la herramienta *Pin Planner*. Es necesario seleccionar el puerto GPIO0[0] tras la asiganción para cambiar el estandard I/O a LVDS.

Para el rechazo del modo común existe la posibilidad de seleccionar una terminación de $100\,\Omega$ interna al componente. Se reducen así los elementos externos y la distancia del elemento resistivo al mismo. Para ello se requiere habilitar la vista de la columna *Input Terminal* y seleccionar el modo diferencial. Con el botón derecho en cualquier columna se accede al menú *Customice Columns* como el de la figura 29. No obstante, en el presente proyecto, no se utiliza dado que no se trata de una línea de transmisión.

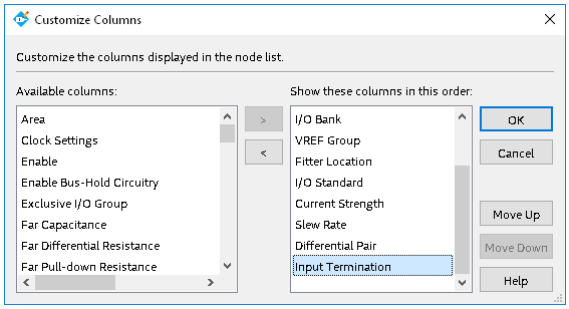

Figura 29: Herramienta Pin Planner. Habilitar el Input Terminal.

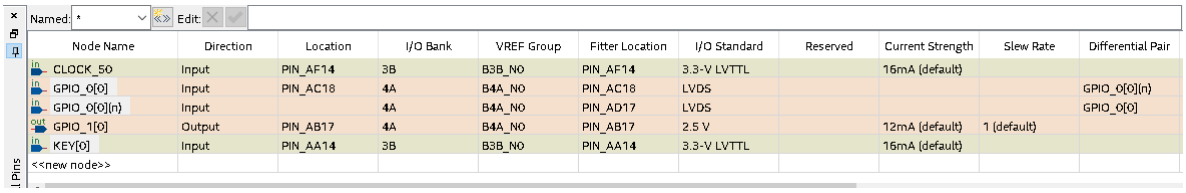

Figura 30: Editor Pin Planner.

En la figura 30 se observa el resultado tras la configurción propuesta. Se ha generado el puerto diferencial correspondiente en la asiganción PIN_AD17 que corresponde a la asignación natural del GPIO0[2].

3.3.1. Buffer de salida

Mediante el uso de la IP de Altera ALTIOBUF se genera el componente como *output buffer* y un solo buffer instanciado como parámetros. Dos señales auxiliares *nbuffer* y *obuffer* configuran las conexiones de entrada y salida del mismo con nuestro modelo. En la figura 30 puede verse la asignación al puerto de salida GPIO1[0] el pin PIN_AB17 que corresponde a la conexión con *obuffer*. Tras generar el componente se instacia de

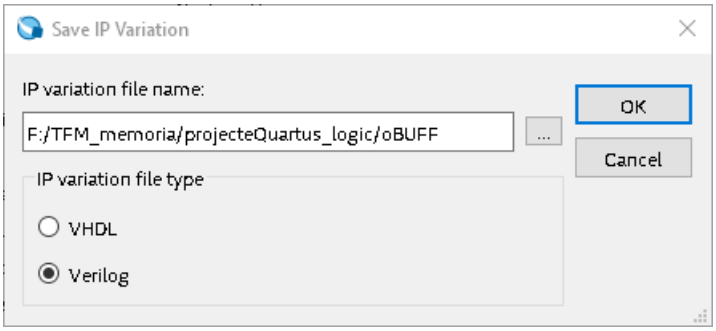

Figura 31: Generación del Buffer de salida mediante ALTIOBUF.

la siguiente forma:

```
oBUFF el_Buffer_Salida (
  .datain ( nbuffer ),
  .dataout ( obuffer )
  );
```

3.3.2. Buffer de entrada

La entrada LVDS requiere de un componente de la librería IP de altera. Se selecciona el ALTLVDS_RX con un único canal y el resto de parámetros por defecto. Tras generar el componente se instacia de la siguiente forma.

```
iBUFF an(
  GPIO_0[0],
  clk6553600,
  lvds_out
  );
```

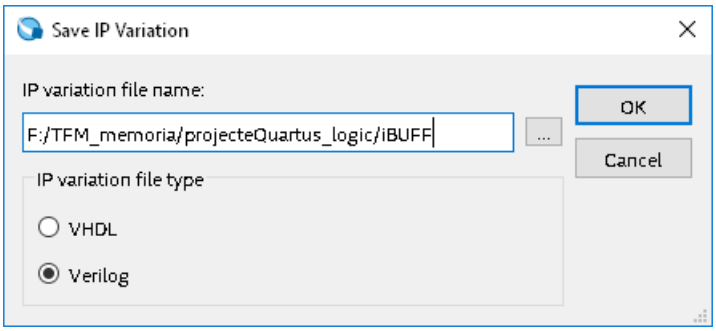

Figura 32: Generación del buffer de entrada mediante ALTLVDS_RX.

4. Test real del conversor $\Sigma\Delta$ implementado.

En el test anterior la señal de entrada era digital generada mediante Matlab©, sin embargo, en una aplicación real del eNose la entrada será analógica. En este apartado se diseña un modelo de test que es capaz de verificar el correcto funcionamiento del conversor ADC implementado con una entrada analógica. En la figura 33 se representa el diagrama de conexión utilizado. Un generador Rigol DS1054 de 16 bits conectado a la entrada de la FPGA y la salida del DAC a un circuito RC con la ayuda de una protoboard. Mediante el software el Quartus© en un PC de postproducción y conectado al puerto USB tipo B de la tarjeta DE1-SoC se implementa el analizador lógico.

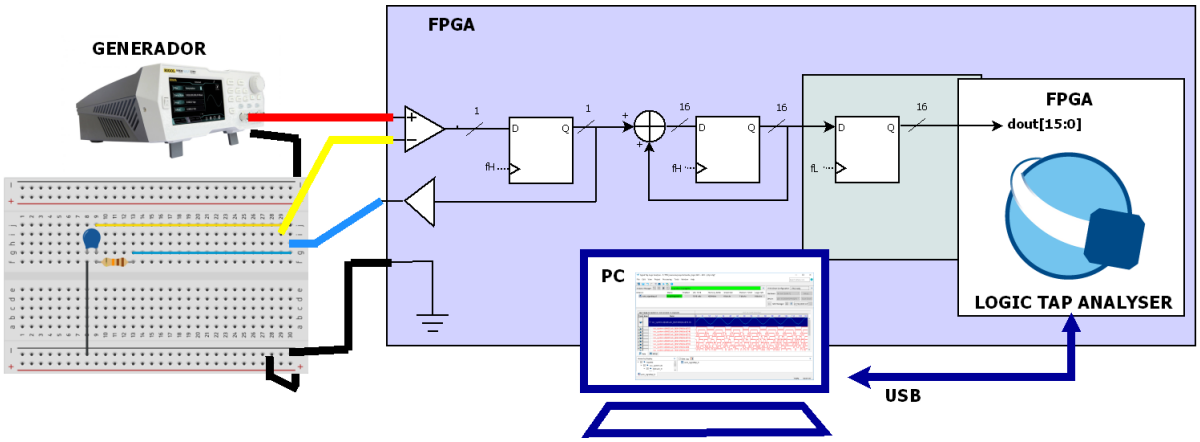

Figura 33: Test real mediante analizador lógico implementado en FPGA.

4.1. Modelo real de test

Dado el problema existente en la generación de estímulos, como la épsilon del reloj de $6{,}5536\,MHz$ visto en la simulación y que se requiere de la interacción con señales externas, se procede con una prueba real mediante la herramienta *Signal Tap Logic Analyser* y una señal de entrada. Esta herramienta implementa en la FPGA un analizador lógico. En el apartado 9.1 se detalla la configuración del necesaria así como los procedimientos para llevarla a cabo.

4.1.1. Capa Generador. Señal de entrada.

Se dispone de un generador de señal analógico con la intención de conseguir con señales con mayor resolución que la deseada. En este caso se trata de un PROMAX GF-232. La señal generada pretende ser $v(t) = 0{,}5+0{,}5\sin(2\pi 6t)$ pero dadas las limitaciones del generador se consigue la señal de la figura 34.

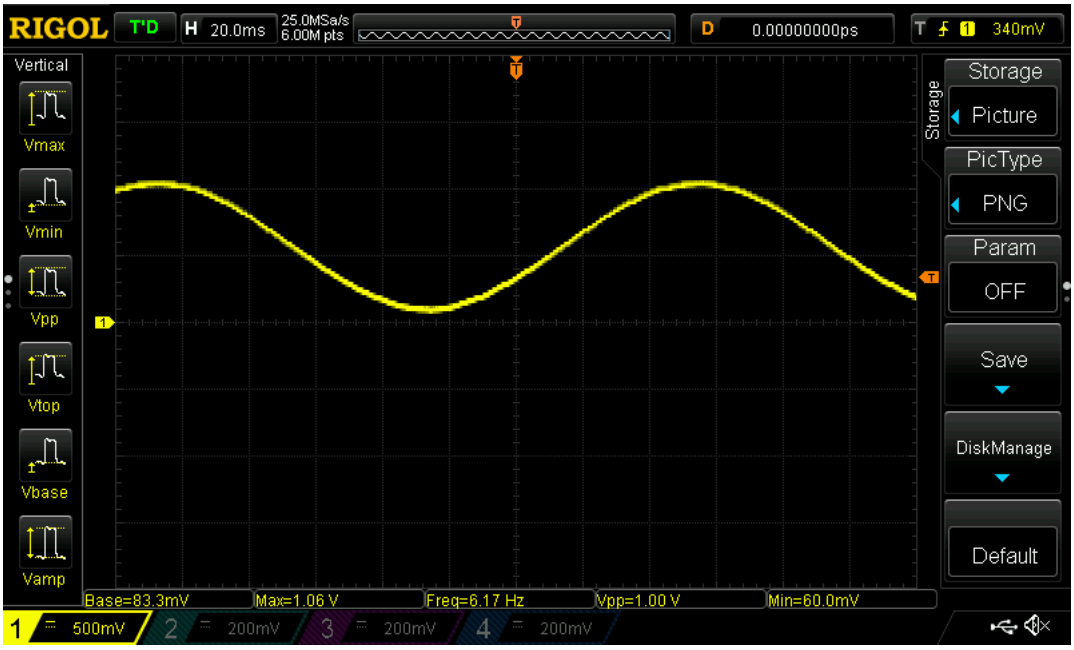

Figura 34: Señal $1\,V_{pp}$, $6{,}16\,Hz$ y offset= $0{,}5\,V$. Escala $500\,mV/div$ y $20\,ms/div$

4.1.2. Salida Analizador

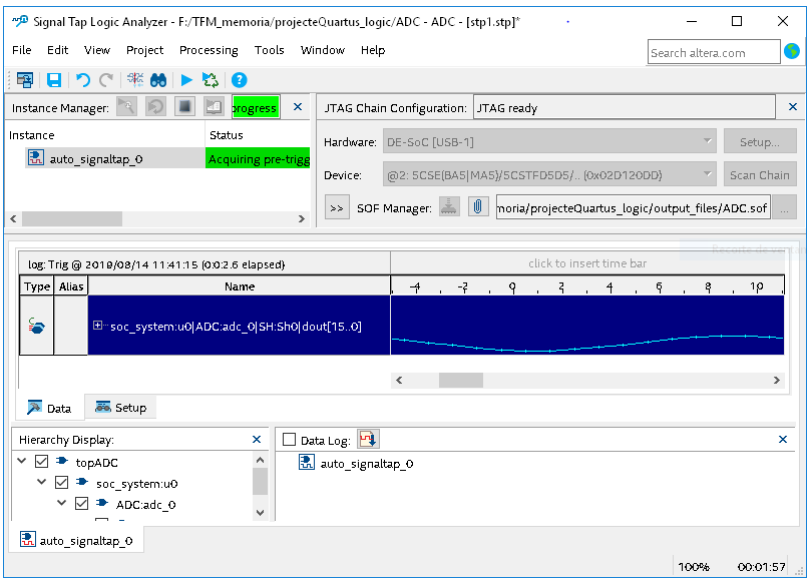

Figura 35: Salida *Logic Tap Analyser*, con *MSB on TOP* y *Unsigned Line Chard*.

Se debe tener en cuenta que el analizador lógico requiere de muchos recursos de la FPGA y no es posible añadir más señales en la visualización, como el reloj, para estudiar la señal en profundidad. No obstante, se obtiene una señal senoidal como se observa en las figuras 35 y 36.

En la figura 36 se observa que la salida tiene un máximo a 16'b19604 que equivale aproximadamente a $748\,mV$ si se toma como tensión de referencia $2{,}5\,V$. Por tanto existe una pequeña diferencia respecto de la señal de entrada. Esta diferencia hace intuir cierto error de ganancia. Pero dicho error no debería producirse, al menos, no de tal magnitud. Es por ello que se verifican la señal de salida del DAC-1 Bit accesible del exterior de la FPGA. En la figura 37 se observa que el pulso no es de $2{,}5\,V$ como se esperaba sino que es de aproximadamente $3{,}3\,V$.

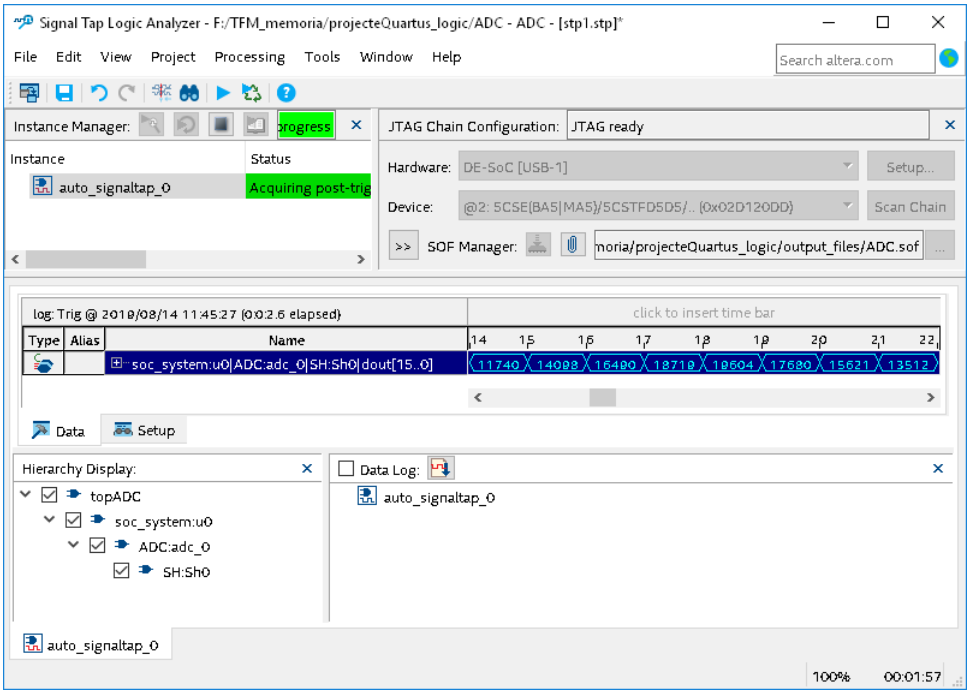

Figura 36: Salida *Logic Tap Analyser*, con *MSB on TOP*, *Invert signal* y *Unsigned Decimal*.

El fondo de escala se establece en $3{,}3\,V$ dado a un bug existente en pla DE1-SoC que a pesar de configurar el GPIO en el banco de $2{,}5\,V$ genera pulsos de $0 - 3{,}3\,V$ o incluso $3{,}4\,V$. Este error se había pasado por alto dado que en la verificación anterior no se dispone de la señal para su observación.

Con un fondo de escala de $3{,}3\,V$ el dato decimal obtenido mediante el test se convierte en aproximadamente $987\,mV$. Este dato se acerca más al valor introducido en el generador de $1\,V_{pp}$.

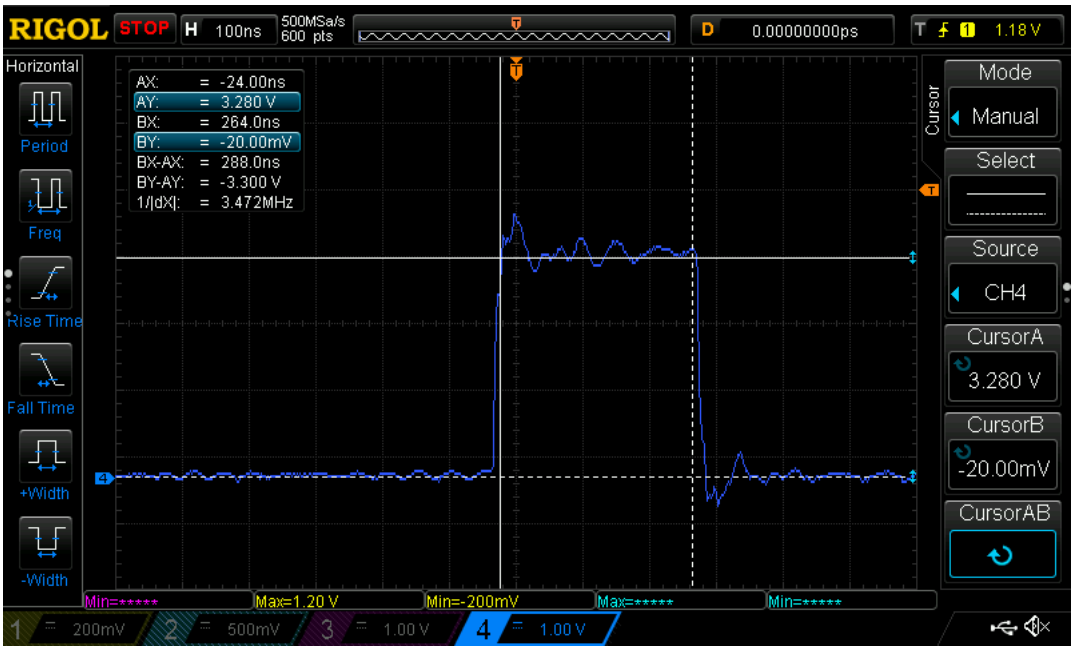

Figura 37: Captura señal salida DAC y entrada circuito RC.

5. Capa de Aplicación. Software y test final

Para el test final se requiere de un sistema operativo en la tarjeta SD, el monitor, ratón y teclado conectados así como el generador y el circuito RC a los pines GPIO. La figura 38 muestra un diagrama de conexión del test real final donde la tarjeta DE1-SoC está representada por el contenido de la Ciclone V, que se divide en dos mitades, con los elementos correspondientes conectados a cada parte(FPGA o HPS). Como puede observarse la tarjeta SD con el sistema Linux, el teclado y ratón son controlados por el procesador hard (HPS) y tanto el componente ADC como el controlador de la pantalla VGA implementados en la FPGA. Cabe destacar que en la figura no existe conexión al PC de postproducción. Es posible conectarse al HPS mediante los estándares Ethernet ó UART[12] para obtener el archivo generado por el test.

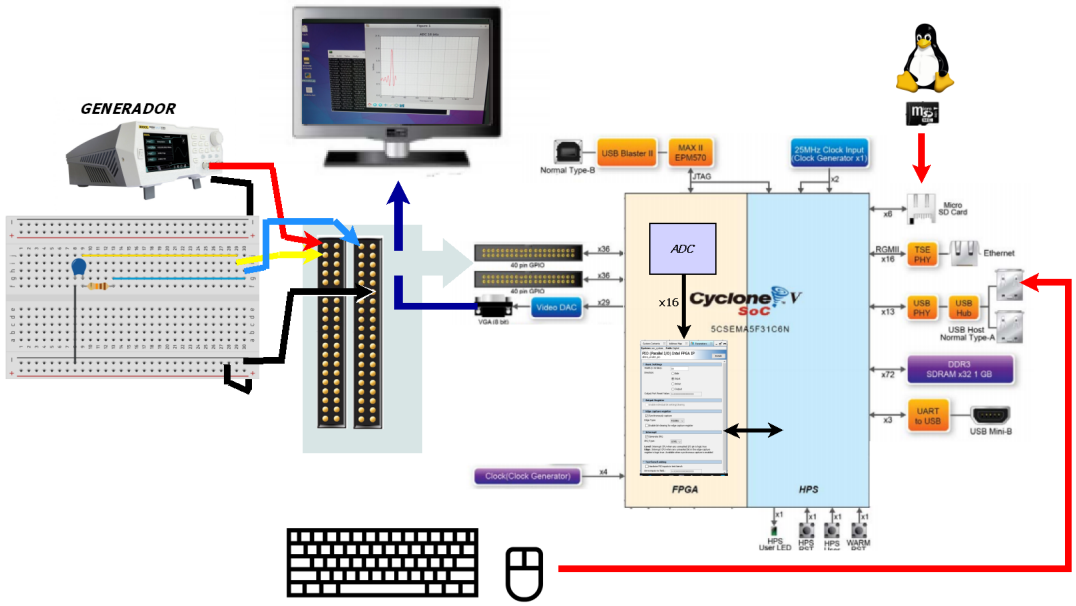

Figura 38: Diagrama Test real final.

Disponer de un de un sistema operativo capaz de gestionar el componente ADC implementado en la FPGA no es una tarea trivial. Para ello se describen las operaciones y procedimientos para la configuración de la tarjeta en el apartado 9.2. Una vez configurada la placa y el sistema Linux se requiere de ciertos preliminares descritos en el apartado 9.2.10 para tener la versión actualizada con las librerías necesarias.

[12] UART→ Universal Asynchronous Receiver Transmitter.

5.1. El scrip

Se define el scrip como el programa a ejecutar o proceso del cual subyacen los procesos secundarios hasta terminar con el salvado de los datos en un archivo. Tal y como puede verse en la Figura 39 el programa principal lanza dos hilos hijos que quedan huérfanos hasta que finaliza cada uno de ellos. Se han simplificado llamadas a métodos o creación de variables u objetos, que se verán más adelante, para un mayor entendimiento.

La salida del comando lscpu devuelve cierta información relevante para el desarrollo

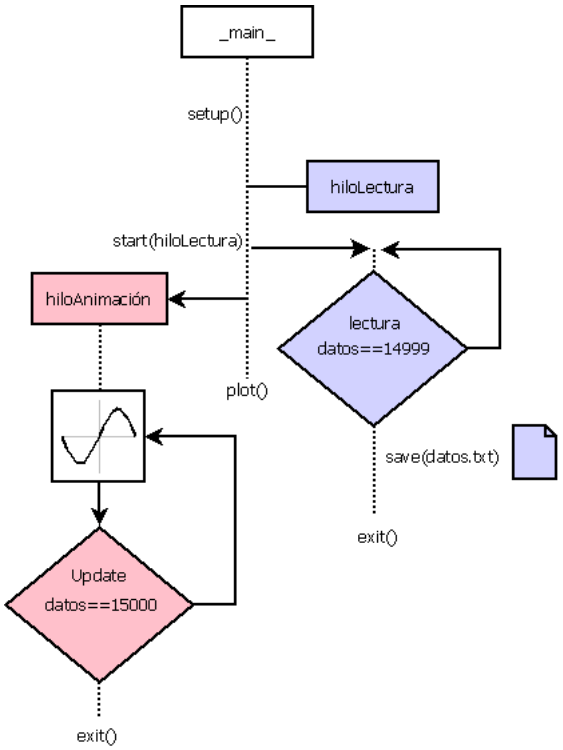

Figura 39: Diagrama de flujo del script en Python3.

del software. Como puede verse a continuación únicamente existe un hilo por cada core y dos cores en el único socket del que dispone la placa:

```
$ lscpu
Architecture:          armv7l
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             1
Model name:            ARMv7 Processor rev 0 (v7l)
```

El programa principal en sí es una lista de instrucciones que se convierte en un proceso a ejecutar por la cpu. En este caso se dispone de dos CPUs por lo que no es descabellado pensar en el uso de las mismas en paralelo. El problema reside en que para ello se requiere de dos procesos independientes: lectura del dato y presentación en pantalla del dato. Por tanto se requiere de un dato en común. Por otra parte el sistema continúa ejecutando las tareas y procesos de un sistema operativo en segundo plano y a la espera de que el usuario mueva el ratón, haga clic entre otras interacciones que puedan necesitar el uso de la CPU. En esta situación el sistema operativo dispone

de un elemento, el Dispacher, que se encarga de gestionar un cambio de contexto o lo que es lo mismo un cambio de proceso. Por lo tanto se expulsa un proceso en ejecución para ejecutar otro. Los procesos no comparten datos ni espacio de memoria por lo que el Dispacher tarda más en realizar la tarea de pasar de estado de ejecución a espera y poner otro proceso a ejecutar. En cambio los hilos comparten los datos y espacios de direccionamiento por lo que es mucho más rápido realizar un cambio de hilo.

Dividir el programa en dos hilos supone que pertenecen a un mismo proceso con lo que no es necesario un cambio de contexto por el Dispacher. Por otro lado, no es posible paralelizar la ejecución del programa con los dos cores disponibles pero sí permitir un core libre para la correcta ejecución de los procesos requeridos por el sistema operativo. Por tanto existirá un proceso único al ejecutar el programa y se crean dos hilos de ejecución.

5.1.1. Programa principal

El hilo principal encargado de inicializar las variables, generar el objeto lector, el hilo de lectura y la función de animación así como el gráfico se describe a continuación:

```
...

if __name__ == '__main__':
 global contador, elapsed, dato, cuenta, num
 #inicializar variables
 elapsed = 0
 contador = 0
 dato = 0
 cuenta = 0
 num = 0
 #crear archivo de datos
 os.system('touch dades.dat')
 #crear objeto lector
 rider=lector(0x00000010,0xff200000)

 #crear vectores
 x = np.linspace(0.0, 150.0,15000, dtype=None )
 y = np.zeros(len(x))

 #crear figura
 fig, ax = plt.subplots()
 line, = ax.plot(x, y, color='r')
 line.axes.axis([0,150,0,2.5])

 #crear hilo de lectura
 n_sem = 1
 semaforo = threading.Semaphore(n_sem)
 hiloLectura = threading.Thread(name='lectura',
                        target=lectura,
              daemon=None,
              args=(rider,semaforo))

 hiloLectura.start()

 #tiempo al hilo de lectura
 time.sleep(0.2)

 #crear bucle animado que actualice la figura
 hiloAnimacion=animation.FuncAnimation(fig, update, len(x), fargs=[x, y, line],
```

```
                                init_func=init, interval=1500, blit=True, repeat=
    False)
#representar figura
plt.show()
```

5.1.2. Librerías

Para la correcta ejecución del programa requiere importar las librerías necesarias. En el siguiente código se ha comentado la utilidad de cada una de ellas:

```
#!/usr/bin/python3
import threading #  Para Generar el hilo de lectura
import sys       #  Para salir del hilo de ejecucion
import time      #  Generar pausas
import struct    #  Transformar bytes en decimal
import numpy as np #Trabajar con matrices y numeros
from matplotlib import pyplot as plt # Plotear la senyal
from matplotlib import animation #Animar la representacion de la senyal
import os        #  Operar con el sistema operativo
import mmap      #  Leer memoria
...
```

5.1.3. El objeto lector

Para aprovechar el paradigma de la orientación a objetos de Python se genera un objeto de la classe *lector*. El uso de este objeto facilita el escalado del código para un conversor con más de una entrada. El diagrama de lenguaje unificado de modelado (UML) de la Figura 40 representa dicha clase. Este objeto, a su vez, representa la Capa de Lectura. Se ha optado por este método pero podría haberse implementado un código o programa subyacente en otro lenguaje y mediante una tubería o paso de parámetros intercambiar información con el script.

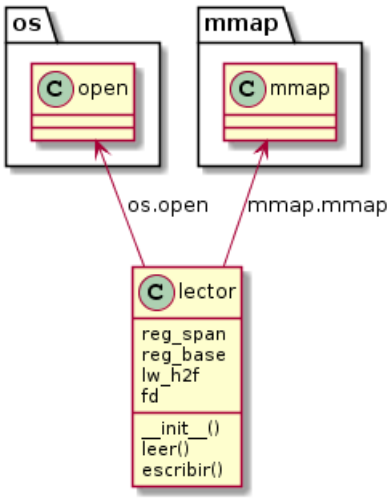

Figura 40: Diagrama UML de la clase lector.

El constructor recibe la dirección de memoria inicial y final donde se ubican los registros en cuestión. La función mmap se encarga de generar el puntero a memoria para su posterior uso.

```
...
class lector():
 def __init__(self,reg_span, reg_base):
  self.fd=os.open("/dev/mem",os.O_RDWR | os.O_SYNC)
  self.reg_span = reg_span #0x00000010
  self.reg_base = reg_base #0xff200000
  self.lw_h2f = mmap.mmap(self.fd, self.reg_span, mmap.MAP_SHARED, mmap.
    PROT_READ | mmap.PROT_WRITE, offset = self.reg_base)
...
```

En clase lector se definen dos métodos; leer y escribir. Tal y como se definen en primer lugar, leer, realiza la función de leer dos bytes(16 bits), situar el puntero en la posición base y convertir el dato a punto fijo $[N,b] = [16,14]$. Nótese que en la ecuación 9 se convierte el dato leído como *unsigned short* a punto fijo normalizado y se aplica la ganancia del fondo de escala.

$$dato = (datos[0] * 2 **(-16)) * 3{,}3 \tag{9}$$

Un contador se encarga de finalizar el hilo cuando se han alcanzado las 15000 muestras deseadas. Por último el método escribir guarda el arreglo con todos las muestras en un archivo. A continuación puede verse el código utilizado:

```
...
 def leer(self):
  global elapsed, contador, dato, num
  contador+=1
  #leer 2 bytes
  read_data1=self.lw_h2f.read(2)
  #retornar el puntero de lectura
  self.lw_h2f.seek(0x0,os.SEEK_SET)
  print('Proceso lectura:',threading.current_thread().getName())
  #datos a unsigned_short
  datos = struct.unpack('=H', read_data1)
  #datos a dato en punto fijo
  dato=(datos[0]*2**(-16))*3.3
  y[num]= dato
  num+=1

  #secuencia de escape y salvado en archivo
  if(contador >14999):
   contador=0
   self.escribir(str(y))
   sys.exit()
  else:
   return dato


 def escribir(self, dato):
  try:
   np.savetxt('dades.dat', dato)
  except ValueError:
   os.system('echo "error en escribir"')
...
```

5.1.4. Capa de Lectura. La función lectura

Fuera de la clase lector se declara el método lectura con el que trabajará uno de los dos posibles hilos de ejecución. Éste recibe el objeto lector y un semáforo de la clase Thread, llama a leer mientras tiene el semáforo y lo libera cuando ha leído los bytes para finalmente esperar $10\,ms$.

```
...
def lectura(rider, semaforo):
 global dato, num
 while(True):
  semaforo.acquire()
  dato=rider.leer()
  semaforo.release()
  time.sleep(0.009728397067388)
...
```

El tiempo de espera entre adquisición y adquisición o tiempo de lectura (t_{lec}), dependerá del tiempo que requiere el procesador para leer el dato. Como se observa en la ecuación 10 se trata del tiempo desde que pide el semáforo (t_{A-sem}) hasta que lo suelta(t_{R-sem}) más el tiempo en que duerme el hilo(t_{sleep}).

$$t_{lec} = (t_{A-sem} + t_{leer} + t_{R-sem}) + t_{sleep} \tag{10}$$

El tiempo t_{lec} debe ser igual al ciclo de reloj y por tanto aproximarse a los $10\,ms$. Con

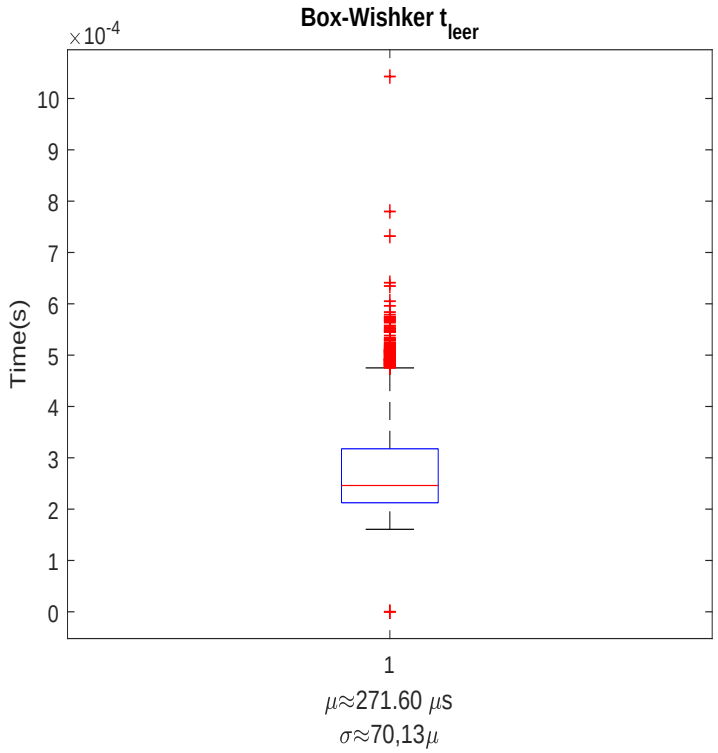

Figura 41: Diagrama Caja-Bigotes tiempo de lectura de dato.

una pequeña modificación en el programa[13] se capturan los tiempos para la adquisición

[13] Con la librería time y su función time() se captura el tiempo de lectura t_{leer} más el tiempo de adquisición t_{A-sem} y suelta del semáforo t_{R-sem} y se guarda en un vector para su posterior registro en un archivo.

de 15000 datos. En la Figura 41 puede verse el diagrama de caja-bigotes que representa la muestra.

A pesar que la mediana está ligeramente desplazada hacia a bajo, la desviación típica es muy pequeña por lo que el coeficiente de variación (cv)[14] está en torno al 25,8 % por lo que se considera que los datos están entorno al valor de la media. Los valores atípicos representan el 1,04 % con un único valor máximo de 1,04 ms y otro de cero por lo que resultan despreciables. En la tabla 4 se resumen los valores extraídos.

Tabla Resumen	
$\mu =$	$271{,}6029326121012\,\mu s$
$\sigma =$	$70{,}13492771052414$
cv=	$0{,}258225958887895 \rightarrow 25{,}82\,\%$
Máximo=	$1{,}042842864990\,ms$
Mayor$\approx$	$475{,}17\,ms$
$Q_{75} =$	$317{,}573547363281\,\mu s$
$Q_{50} =$	$246{,}047973632813\,\mu s$
$Q_{25} =$	$212{,}430953979492\,\mu s$
Menor$\approx$	$160{,}69\,\mu s$
Mínimo=	$0\,\mu s$
$Q_{75} - Q_{25} =$	$124{,}4544982910156\,\mu s$

Tabla 4: Datos estadísticos tiempo de lectura de dato.

Con la media extraída de la tabla se calcula el parámetro t_{lec} que hay que introducir a la función time.sleep(t_{lec}) según la ecuación 11 resultando de 0,009728397067388.

$$t_{lec} = 0{,}001 - \mu(s) \tag{11}$$

5.1.5. Gráfico animado

Los dos métodos siguientes los utiliza la animación para mostrar en pantalla los datos capturados. El primero, llamado init, define el layout del gráfico a plotear.

```
...
def init():
 ax.set_xlim(0, 150)
 ax.set_ylim(0, 2.5)
 ax.set_title('ADC 16 bits')
 ax.set_xlabel('Tiempo (s)')
 ax.set_ylabel('Voltios')
 ax.grid(b=True, color='#666666', which='major', linestyle='-', linewidth=0.5)
 ax.minorticks_on()
 ax.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
 return line,
...
```

El segundo, update, requiere especial consideración puesto que se realiza una implementación errónea a propósito para obtener el funcionamiento deseado. La función animate del hilo principal llama a la función update en función de los frames o iteraciones que se establecen. Para ello genera un bucle interno no accesible por el programador en

[14] El coeficiente de variación es $cv = \frac{\sigma}{|\mu|} \cdot 100\,\%$

el hilo principal que puede tener acceso a la cpu en tiempos no deseados. Cabe recordar que se requiere leer el dato cada $10\,ms$ sin interrupciones. Es por ello que se recurre a una variable global num que corrige el contador interno de animate. De este modo, aunque animate llame a la función en dos ocasiones durante los tiempos de espera, arbitrariamente o cada ciertos ms, el gráfico actualiza con los valores correctos.

```
...
def update(nume, x, y, line):
 global dato, num
 nume=num
 #print('Thread update:',threading.current_thread().getName())
 print(num)
 if(contador >14999):
  sys.exit()
 return line,
...
```

5.2. Test Final

Para ejecutar el script es posible que requiriera permisos de ejecución y como se accede a memoria debe identificarse como superusuario. Para ello se debe acceder a la carpeta y ejecutar el siguiente código:

```
$ sudo chmod 755 enose.py
$ sudo nice -n-17 python3 enose.py
```

Mediante el comando nice y la opción -17 le damos prioridad de ejecución sobre otros procesos del sistema o que arranque el propio usuario. En la figura 42 puede observarse el sistema funcionando en la adquisición de una señal del tipo sinc generada mediante el generador de 16 bits DG812 de Rigol.

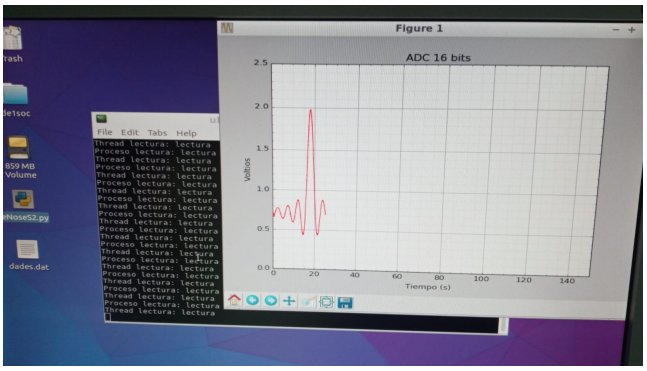

Figura 42: Sistema funcionando en la DE1-SoC.

Como se requiere acceso a memoria es necesario ejecutar la aplicación mediante permisos y por tanto, introducir la contraseña. Para evitar esto se modifica el archivo */etc/sudoers* mediante el siguiente comando:

```
$ sudo visudo
```

Tras la identificación como superusuario se añade al final del archivo la siguiene linea:

```
ubuntu ALL=(ALL) NOPASSWD:ALL
```

Es posible crear un lanzador en el Escritorio para ejecutar la aplicación. Para ello se requiere la instalación y ejecución del paquete gnome-desktop-item-edit mediante los siguientes comandos:

```
$ sudo aptitude install gnome-desktop-item-edit
$ gnome-desktop-item-edit ~/Desktop --create-new
```

En la pestaña *Desktop Entry* se seleccionan las opciones ejecutar en terminal y mantener el terminal abierto tras la ejecución y se introduce el comando con la ruta del archivo python. Es posible personalizar un icono o elegir cualquiera disponible en el sistema.

6. Presentación de resultados. Postproducción

Para el análisis se requiere guardar el archivo generado mediante la aplicación en un stick usb o bien copiarlo a través de la red o una conexión serie. En este caso, se ha optado por conectar la tarjeta DE1-SoC al PC de postproducción mediante un switch y cables ethernet. En la figura 43 se detalla el comando scp utilizado. Nótese que se debe conocer la direción IP[15] de la DE1-SoC para substituirla en el argumento del comando.

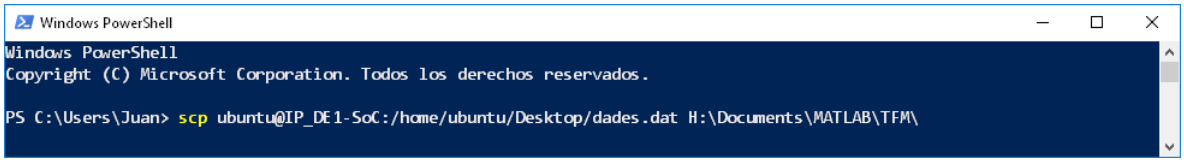

Figura 43: Comando Power Shell de Windows para la copia del fichero generado.

Para la caracterización del conversor se utilizan diferentes señales de entrada. Se observa la respuesta a diferentes frecuencias dentro del rango de adquisición como ya se determinó en el apartado 1.4. Se estudia el comportamiento en los extremos y se obtienen los valores de referencia que definan el conversor analógico digital diseñado.

Con el fin de tener cuatro ciclos completos en una captura de 15000 muestras se utiliza una señal senoidal de entrada de $35{,}5\,mHz$ con un máximo en $2\,V$ y un mínimo en $100\,mV$. De forma visual y empírica se capturan dos máximos consecutivos para disponer de un ciclo completo con el que comparar con una señal ideal.
La señal a comparar se genera mediante ©Matlab con:

```
t=linspace(0,1/0.0355,28.169);
y=1.05+0.95*cos(2*pi*t*(0.0355));
```

6.1. Ganancia

La señal capturada junto con la señal ideal se muestran en la figura 44. Tal y como se observa se tiene una ligera ganancia así como un offset. Estas diferencias vienen condicionadas por la señal de referencia que corresponde al pulso filtrado por la RC. Este pulso puede observarse en la figura 37.

Mediante la conversión es posible ajustar la ganancia por ello se modifica la ecuación 9 en la función leer:

$$dato = (datos[0] * 2 ** (-16)) * 3{,}342991226281 \tag{12}$$

[15] Para conocer la dirección IP basta con ejecutar en una terminal el comando *ip -c addr*

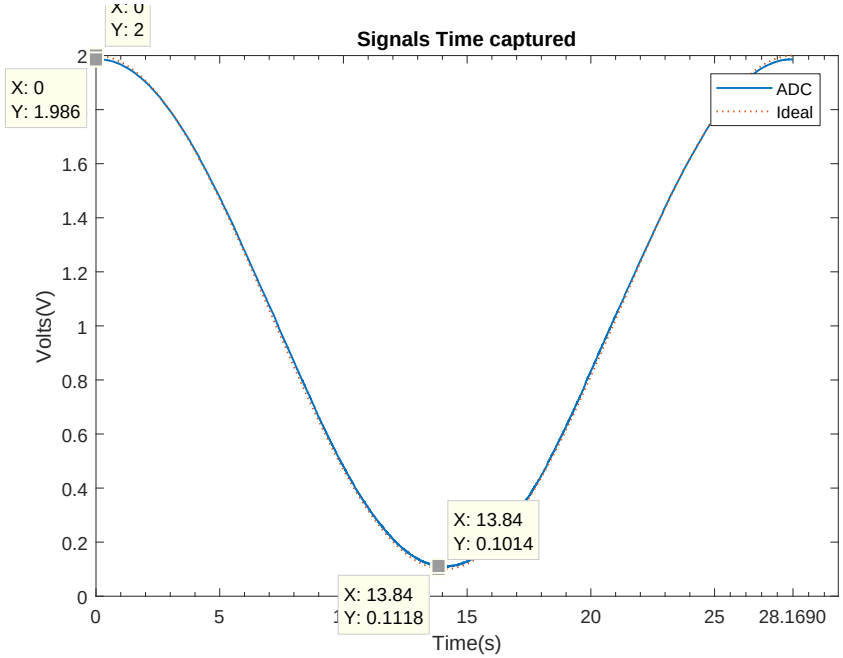

Figura 44: Comparación señal ideal con señal capturada sin ajustes.

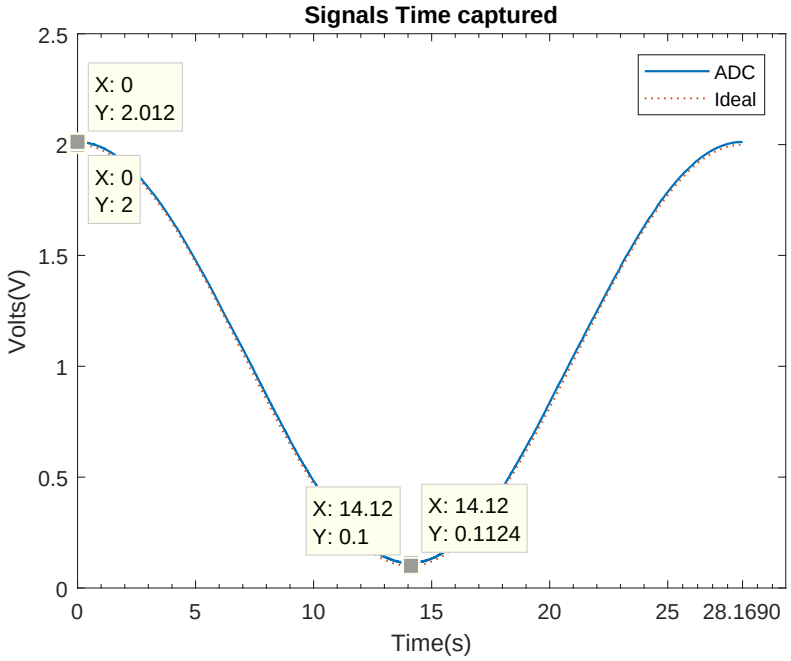

Figura 45: Comparación señal ideal con señal capturada con ajuste de ganancia.

6.2. Offset

Dada la señal capturada vista en la figura 45 se observa un offset respecto de la ideal de aproximadamente unos $12\,mV$.

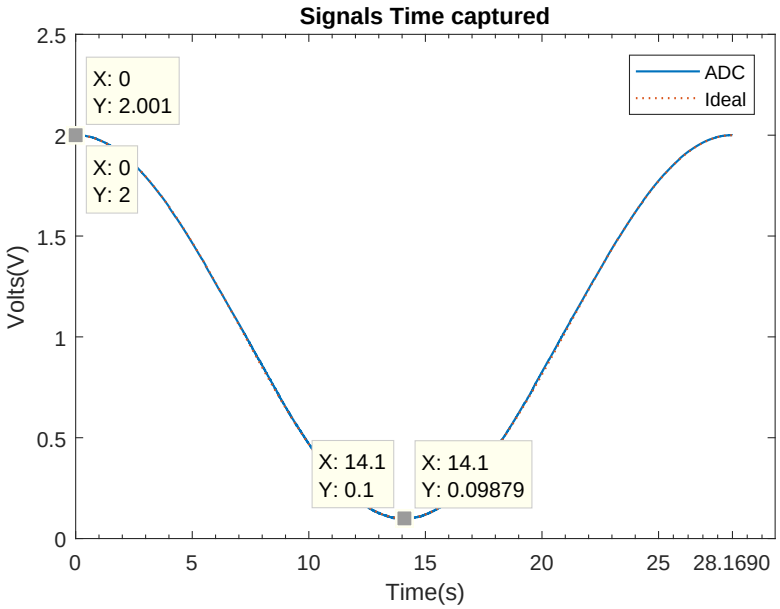

Figura 46: Comparación señal ideal con señal capturada con ajuste de offset.

6.3. Número efectivo de bits

En la medida del rango dinámico del conversor ADC el número efectivo de bits o ENOB para una señal senoidal que ocupe todo el rango posible se caracteriza por:

$$ENOB = \frac{SNR_{dB} - 1{,}761}{6{,}02} \tag{13}$$

Dada la ecuación vista en 13 para 16 bits se tiene aproximadamente una relación señal a ruido de $SNR_{max} \approx 98{,}02\,dB$. Si mediante el análisis espectral de potencia de la señal capturada visto en la figura 47 se obtiene una relación señal a ruido de $SNR \approx 75{,}97\,dB$ el número de bits efectivos tras aplicar la ecuación 13 son 12.

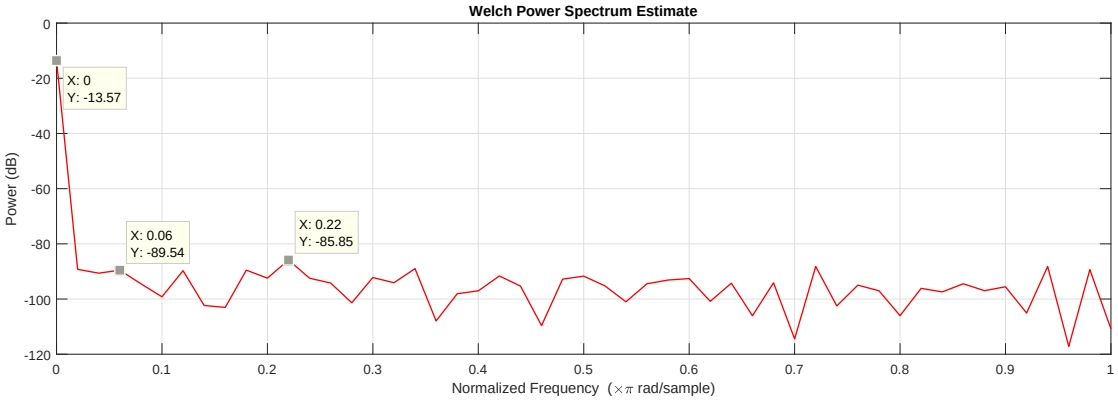

Figura 47: Estimación del espectro de potencia para el cáculo del ENOB.

6.4. Rango dinámico libre de espurios

El rango dinámico libre de espurios se determina por la ecuación 14:

$$SFDR \approx 6{,}02 \cdot N - 3{,}92\,dB \tag{14}$$

Para una señal de 16 bits se tiene un $SFDR \approx 98{,}42\,dB$ frente a los $SFDR \approx 72{,}28\,dB$ que se deducen de la figura 47. Este fenómeno supone la reducción a 11 bits efectivos.

6.5. Función de transferencia

Una vez compensado el offset y la ganancia se obtiene la función de transferencia de la figura 48.

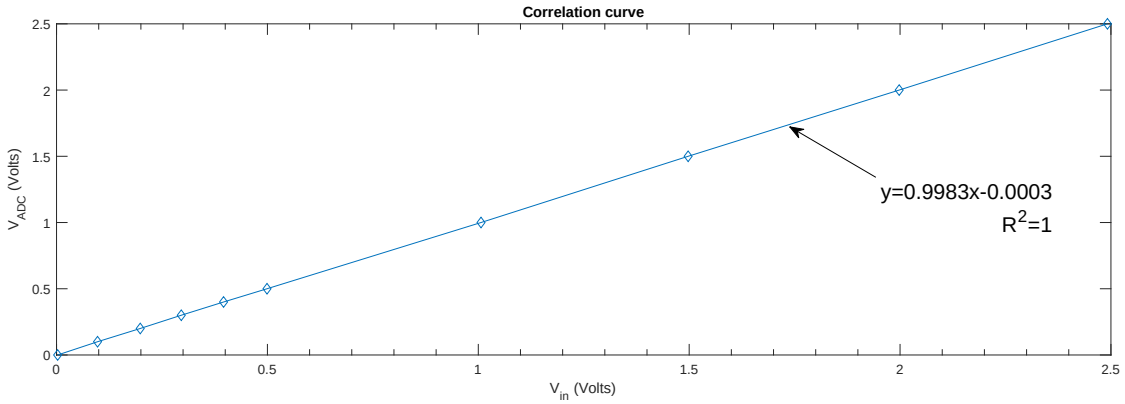

Figura 48: Función de transferencia señal de entrada respecto señal ideal.

Dada la figura 48 se observa que el sistema se comporta de forma lineal.

6.6. Barrido en frecuencia

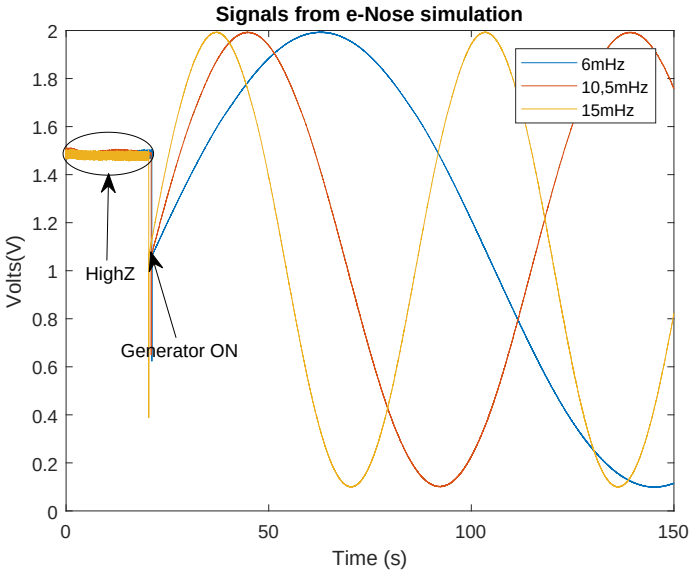

Figura 49: Señales de $6\,mHz$, $10{,}5\,mHz$ y $15\,mHz$ simulando sinusoides similares a las señales e-Nose.

Se observa el comportamiento en frecuencias diferentes dentro del rango de aplicación de una e-nose. Para ello se observan las pendientes de las señales vistas en la figura 3 y una supuesta senoidal con la misma pendiente estaría comprendida entre unos $6\,mHz$ y $15\,mHz$ aproximadamente. En la figura 49 se observan tres señales dentro del rango anterior.

Durante los ensayos se han utilizado diferentes señales donde se ha observado un resultado idéntico en el rango de frecuencias deseado. La figura 50 se ha capturado con el conversor realizado mediante la introducción de una señal $v(t) = 1{,}05 + 0{,}95\sin(\omega t)$ variando la frecuencia de entrada con el scroll del generasor. Se observa un barrido de $1\,mHz$ a $3\,Hz$ donde no presenta ninguna variación anómala.

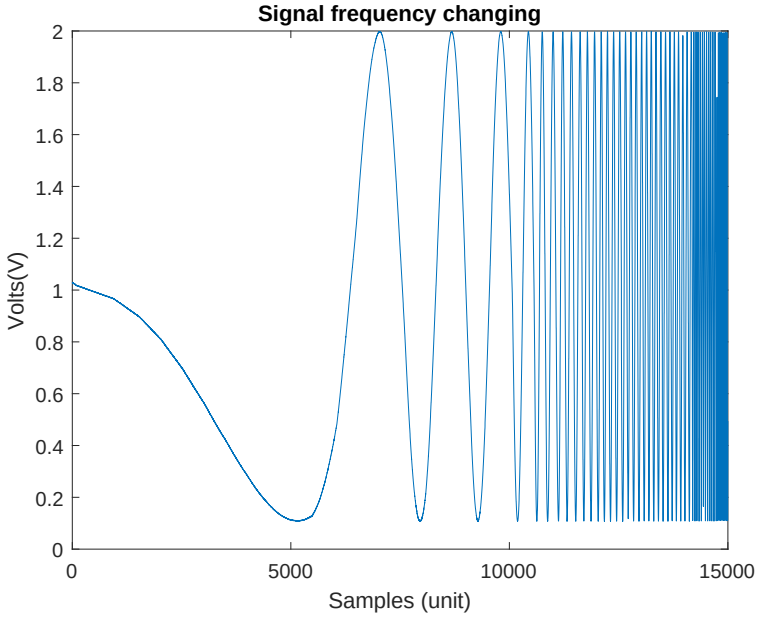

Figura 50: Barrido de $1\,mHz$ a $3\,Hz$ de una señal $v(t) = 1{,}05 + 0{,}95\sin(\omega t)$

7. Conclusiones

Las señales obtenidas de los 32 sensores de gas vistas en la figura 3 están truncadas a dos decimales por lo que no se tienen resoluciones menores a $10\ mV$ y se tiene una frecuencia de muestreo de $100\,Hz$. Las pendientes de las señales no superan los $15\,mHz$ por tanto, la resolución temporal está limitada en saltos de aproximadamente 9 bits.

Se ha conseguido digitalizar señales alcanzando una resolución de al menos 11 bits con menos recursos. El conversor visto en [7] se acerca con 9.8 bits de resolución pero con el uso de más recursos. Se ha detectado un recorte de la señal o su modificación por debajo de $10\,mV$ pero como puede verse en la figura 3 no es necesario introducirlo ya que este tipo de sensores introducen uno superior. Cabe recordar que el conversor implementado en [6] se volvía inestable por debajo de los $250\,mV$ y por encima de los $2,25\,V$. Este efecto se produce en los extremos de conversión. En el presente trabajo final de Máster disponer de una referencia mayor beneficia en el límite superior puesto que éste no se alcanza.

El ADC implementado en el presente proyecto permite su escalado a un diseño nuevo con más sensores. Es por ello que el resultado obtenido resulta más que óptimo para la aplicación.

Debido a que el sistema dispone de dos cores en un mismo socket y un hilo de ejecución por core la paralelización debe realizarse mediante hilos permitiendo así compartir variables entre los mismos y dejando el otro para el sistema. No obstante el escalado a varios sensores puede requerir un estudio mayor o incluso un cambio de estrategia como por ejemplo un algoritmo Round-Robin.

El hecho de disponer de una tarjeta ha resultado clave en el presente trabajo fin de Máster pero también parte del problema. Se han alcanzado todos los objetivos descritos en el apartado 1.1 y por supuesto, el objetivo principal de mejora personal. No obstante el esfuerzo realizado ha resultado extenuante y desesperante. Ha supuesto lidiar entre la frustración y el abandono. Los diversos bugs y problemas que se han tenido que superar han llevado al autor al borde de la desesperación. Posiblemente en un entorno productivo hubiese optado por abandonar esta tarjeta por otra. Afortunadamente se ha llegado a una solución más que óptima que presenta incluso innovación en la técnica.

8. Trabajo futuro

Se ha obtenido un conversor analógico digital óptimo para la captura de señales provinientes de un sensor del tipo MOS con el fin de clasificar curvas y con ello patrones olfativos. En concreto en la línea de investigación de la tesis doctoral; Análisis, diseño e implementación de instrumentación no invasiva para la detección de cáncer de próstata y vejiga mediante tecnología MOS. No obstante, este conversor sólo és válido para un sensor. Por ello, se requiere trabajar en un multiplexor así como en el desarrollo de una tarjeta propia que implemente el SoC. Con los elementos justos se puede conseguir el aprovechamiento de todos los pines LVDS para así disponer de más entradas. Este diseño debería de implementar un buffer para la captura de los datos.

Otra linea puede seguir por utilizar esta tarjeta con un único sensor para caracterizar sensores aislados en una cabina, así como el diseño de filtros para conseguir mayores resoluciones en el caso que fuese necesario.

Agradecimientos

El autor agradece la ayuda humana y técnica recibida por el profesorado del Máster Universitario de Ingeniería de Sistemas Electrónicos en especial a Mis tutores la Dra. Maria José Canet y el Dr. José Pelegrí, por mostrarle el camino al desarrollo profesional. Al servicio técnico de Terasic que sin sus archivos no hubiese salvado uno de los innumerables obstáculos que ha supuesto esta tesina. A Dr. Bruce Robert Land que aunque no me ayudó, su comentario sobre el abandono de esa placa contribuyó a solucionar otro de los muchos bugs y finalmente conseguir el entorno gráfico.

Y a todos aquellos que han tenido paciencia para el autor pudiese compaginar vida, trabajo y esta tesina, porque de todos ha aprendido.

Bibliografía

1. Di Natale, C. en *Handbook of Seafood and Seafood Products Analysis* (2009). ISBN: 9781420046359.
2. Altera Corporation. Cyclone V SoC HPS Release Notes (2014).
3. Cupane M, Pelegri-Sebastia J, Climent, Guarrasi, S. T. y MA, G. Application of MOOSY32 eNose to Assess the Effects of Some Post Harvest Treatments on the Quality of 'Salustiana' Orange Juice. *Journal of Biosensors & Bioelectronics.* doi:10.4172/2155-6210.1000184 (2015).
4. Terasic. *DE1-SoC* 2019. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English{\&}CategoryNo=205{\&}No=836 (2019).
5. Lattice Semiconductor. Leveraging FPGA and CPLD Digital Logic to Implement Analog-to-Digital Converters. *A Lattice Semiconductor White Paper,* 1-9 (2010).
6. Georgi Tsvetanov Tsenov, Snejana Dimitrova Terzieva, V. M. M. MODELING AND IMPLEMENTATION OF FIRST ORDER SIGMA-DELTA MODULATOR, 47-52 (2006).
7. Sousa, F., Mauer, V., Duarte, N., Jasinski, R. P. y Pedroni, V. A. Taking advantage of LVDS input buffers to implement sigma-delta A/D converters in FPGAS. *Proceedings - IEEE International Symposium on Circuits and Systems* **1,** 3-6. ISSN: 02714310 (2004).
8. Talens Felis, J.B, Sebastià Fabregat, N, Pelegrí-Sebastià, J., Sogorb Devesa, T., Loras Monfort, A, Ruiz-Cerdá, J. en *XII International Workshop on Sensors and Molecular Recognition* 33 (Burjassot, 2018).
9. Ali. Comparators. *High Speed Data Converters,* 191-221 (2016).
10. Lai, Y. MT-011 Find Those Elusive ADC Sparkle Codes and Metastable States. *Imid 2009,* 1069-1072 (2009).
11. Altera. Cyclone V Device Datasheet, 64 (2013).
12. Hellman, J. Implementation of a Low-Cost Analog-to-Digital Converter for Audio Applications Using an FPGA. *Institutionen för systemteknik Department of Electrical Engineering,* 13-34 (2013).

9. Anexos

9.1. Singal Tap Logic Analyser

El software de Altera incorpora un analizador lógico que permite observar el comportamiento de las señales dentro de la FPGA. Para ello se requiere configurar la tarjeta. Se selecciona el PIN MSEL situado en la parte inferior de la DE1-SoC con la siguiente configuración:

$$MSEL[4:0] = 6'b010010$$

El modo configurado permite trabajar con la tarjeta de desarrollo sin utilizar el ARM. Tras su selección e inicio debería mostrar una imagen en la pantalla VGA de la tarjeta así como una secuencia de luces en los leds y código exadecimal en los displays 7 segmentos.

Tras el inicio de ©Quartus y la herramienta *Signal tap Logic Analyser* se requiere conectar el cable de comunicación USB A/B entre el pc y la tarjeta. En caso de no haber comunicado el hardware con anterioridad puede requerir la configuración mediante la herramienta *Programmer* en el menú *hardware setup* tal y como se ve en la figura 51

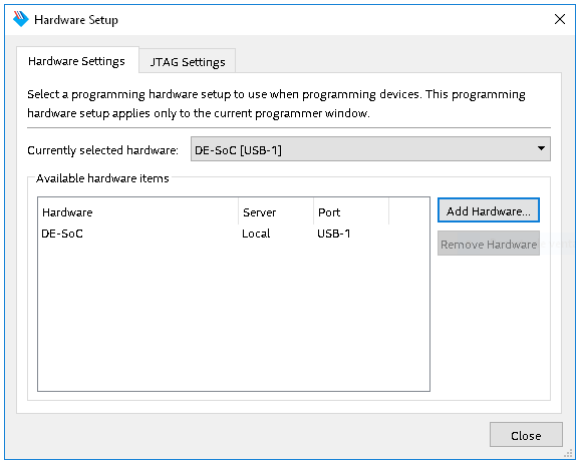

Figura 51: Configuración del dispositivo DE1-SoC mediante USB A/B.

Tras el inicio del Signal tap Analyser y el conexionado y configuración de la tarjeta se muestra el JTAG ready en la parte superior derecha tal y como se puede ver en la figura 52

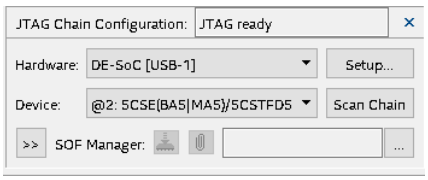

Figura 52: Captura *Signal tap Analyser* dispositivo conectado.

En la parte central con doble click en la pestaña *Setup* se añade la salida *dout* del S& H de 16 bits tal y como puede verse en la figura 53.

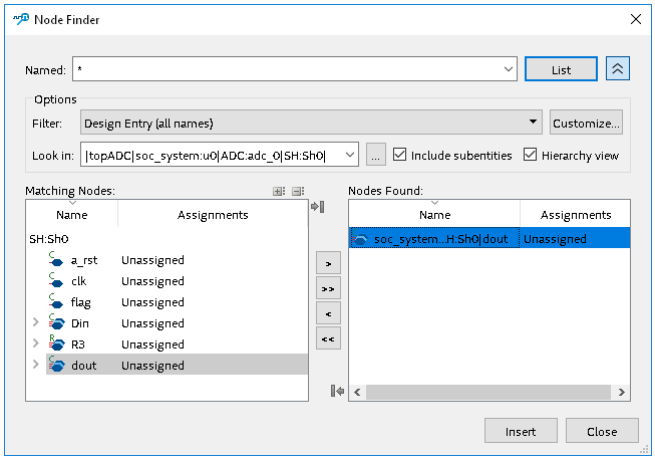

Figura 53: Configuración *Node Finder* para añadir señal a capturar.

En la parte central derecha se añade la señal de reloj con la que se obtendrán las capturas de la señal deseada. Se utiliza la señal de *flag* del S& H como se puede ver en la figura 54. Se utilizan las opciones de *segmented 128 1 sample* y *post trigger position*

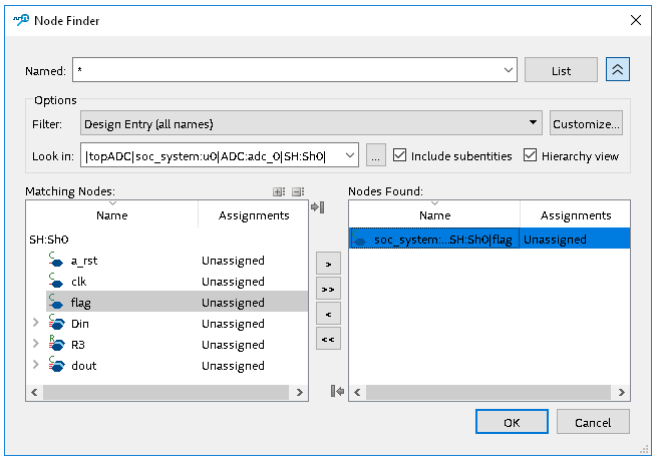

Figura 54: Configuración *Node Finder* para añadir señal de reloj.

Una vez añadidas la señal a capturar y el reloj se requiere compilar de nuevo. Este procedimiento se realiza desde el propio *Signal tap Analyser* que lanza la compilación en ©Quartus. Tras la compilación se carga el archivo ADC.sof en el JTAG y se configura el dispositivo tal y como puede verse en la figura 55. Mediante F5 o F6 se inicia el análisis. Los resultados se observan en la pestaña *Data* de la parte central.

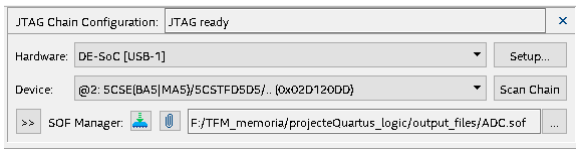

Figura 55: Carga del archivo ADC.sof en el JTAG.

9.2. Preparar Linux en DE1-SoC

Con el objetivo de presentar los datos en pantalla en un entorno amigable similar el de un PC, se recurre a un sistema operativo que utilice los recursos de la tarjeta. Para que el ARM pueda utilizar el sistema operativo Linux, en este caso Ubuntu, se requiere la preparación del sistema de archivos en una tarjeta SD y de la imagen del sistema. Los archivos necesarios son:

- Un Pre-loader[16] o programa secundario de carga del U-boot.
- El U-boot[17] que carga tres archivos desde la primera partición FAT de la SD.
- El Device Tree Blob o árbol de dispositivos que le dice al Kernel el hardware que tiene conectado.
- El Raw Binary File o flujo de bits para configurar el sistema FPGA.
- Linux Kernel o núcleo del sistema.
- Linux Root File system o sistema de archivos y carpetas donde montar el Linux.

Afortunadamente no hace falta montar el sistema manualmente ya que desde la web del fabricante ©Terasic es posible descargar la imagen[18] de la SD con un sistema de ejemplo.

Se parte de este sistema de ejemplo que posteriormente se modifica con la ip generada en el presente proyecto. Por tanto, salvo el sistema de ficheros y el archivo de imagen del kernel, es necesario reconstruir los demás archivos.

9.2.1. Tarjeta SD

El fabricante recomienda una SD de al menos clase 4 y entre 4 y 8 GB según versiones de Linux. La escritura de la imagen se realiza mediante ©Etcher.

En la SD se crean varias particiones. En la figura 57 puede verse como existe una

Figura 56: Copia de imagen Ubuntu sobre la SD.

pequeña partición FAT de $1\,MB$ al inicio donde irá el preloader. La siguiente partición

[16] El Pre-loader o precargador se utiliza debido a que todo el U-boot no cabe en la partición de inicio de la SD. Se realiza una precarga para posteriormente compilar todo el U-boot.

[17] El U-boot contiene un código de inicialización específico de la placa que configura los registros para reflejar la configuración realizada en el qsys.

[18] `http://www.terasic.com/downloads/cd-rom/de1-soc/linux_BSP/de1soc_ubuntu_1604.zip`

asignada de $819\,MB$ se trata de espacio en disco y es donde se almacenan los archivos necesarios listados a continuación:

```
Juan@DESKTOP-SR0FDP7 /cygdrive/k
$ ls -l
total 12292
-rw-r--r-- 1 Juan Juan 7007184  8 des.   2013 soc_system.rbf
-rw-r--r-- 1 Juan Juan   31245 14 oct.   2016 socfpga.dtb
drwxr-xr-x 1 Juan Juan       0 17 oct.   2016 System Volume Information
-rw-r--r-- 1 Juan Juan     200 16 febr.  2015 u-boot.scr
-rw-r--r-- 1 Juan Juan 5538512 27 oct.   2016 zImage
```

Existen otras particiones de $1,01\,GB$ y $1,4\,GB$ debido a que por defecto no se utiliza todo el espacio de la SD. Para poder utilizar todo el espacio se requiere arrancar el sistema Ubuntu en la DE-1SoC y expandir el disco. La siguiente partición de $4\,GB$ es una partición del tipo Ext4 donde se ubica la raíz del árbol del sistema Linux.

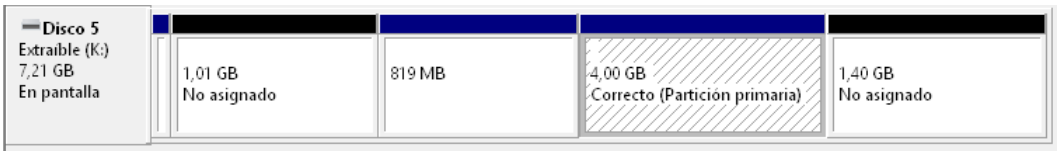

Figura 57: Detalle SD tras la copia de la imagen.

9.2.2. Configuración Tarjeta

Se requiere configurar la tarjeta en modo combinado FPGA y ARM. Se selecciona el PIN MSEL situado en la parte inferior de la DE1-SoC con la siguiente configuración:

$$MSEL[4:0] = 6'b000000$$

Dado que se pretende arrancar en modo gráfico se conectan los periféricos siguientes:

- Ratón y Teclado.
- Monitor VGA.
- Alimentación DE1-SoC.
- Conexión ethernet[19].
- DE1-SoC con Mini USB a USB PC.

9.2.3. Driver USB

Para la depuración puede resultar interesante iniciar sesión en remoto desde un sistema Windows a través del puerto de comunicaciones COM. Para la comunicación serie con el sistema Linux se requiere instalar el driver USB-UART FT232R[20].

Para configurar la UART se requiere:

- 115200 baud rate.

[19] El cable de ethernet es opcional pero mejora el tiempo de arranque ya que en caso contrario agota el tiempo de búsqueda de red.

[20] Es posible descarar el driver FT232R en `http://www.ftdichip.com/Drivers/VCP.htm`

- Paridad No.
- 1 bit de stop.
- No flow control settings.

Para el acceso al sistema desde la comunicación USB-UART se requiere el usuario **ubuntu** y el password **temppwd**.

9.2.4. Proyecto base FPGA-HPS

La generación de un proyecto que permita trabajar la FPGA conjuntamente con el ARM en un sistema Linux no es trivial y requiere de un grado de configuración y detalle elevado. Es por tanto que se parte de un sistema conocido y en funcionamiento. Entre los recursos del fabricante se encuentra un sistema de ejemplo DE1_SOC_Linux_FB[21]

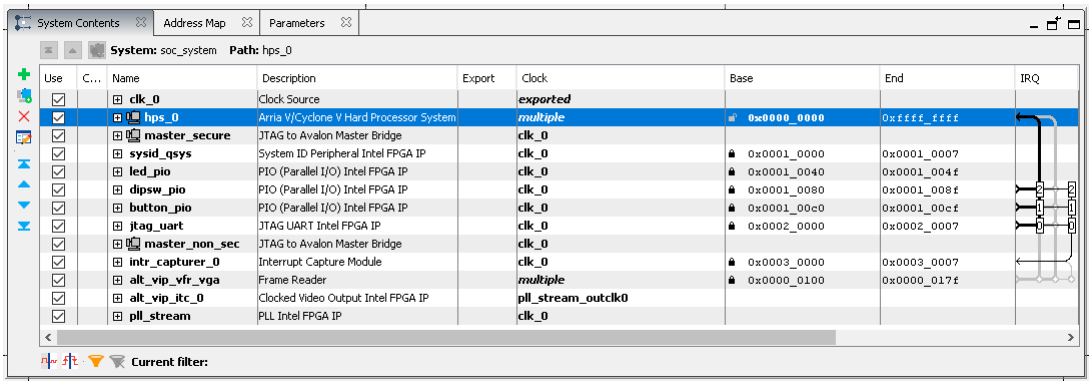

Figura 58: Proyecto base sistema qsys.

El proyecto base consta de los elementos necesarios para el funcionamiento del ARM en modo gráfico, el uso de los LEDs y de los switches así como los botones desde el sistema operativo. Por tanto se prepara la IP diseñada para la integración en el proyecto base.

9.2.5. Proyecto Base y ADC

Tras copiar el archivo .tcl del componente diseñado en el directorio del proyecto base se inicia la herramienta *Platform Designer* donde aparecerá la opción de añadirlo en la pestaña IP Catalog. Se añade el componente y se exportan las conexiones, se genera el pll6553600 y el PI/O tal y como se explicó en los apartados 3.2.2 y 3.2.4 y por último se realiza la conexión del elemento PI/O, en este caso llamado *Digital* al ARM:

- clk → clk_0.clk
- reset → clk_0.clk_reset
- s1 → hps_0.h2f_lw_axi_master
- irq → hps_0.f2h_irq0
- irq → intr_capturer_0.interrupt_reciver

En la pestaña *Adressmap* puede observarse la dirección relativa del bus AXI máster, en la cual, se ha mapeado el componente *Digital* desde $0x0000\_0000$ hasta $0x0000\_001f$.

En la parte del REG/WIRE declarations del Top Level Entity del proyecto se añaden los siguientes elementos:

```
...
wire lvds_out;
wire nbuffer;
wire obuffer;
wire [15:0] dout;
wire clk6553600;
...
```

En la llamada al componente soc_system se añaden las señales nuevas en el proyecto:

```
...
        .adc_0_obuff_new_signal                    (nbuffer),
        .adc_0_dout_conduit                        (dout),
        .adc_0_din_new_signal                      (lvds_out),
        .adc_0_reset_reset                         (hps_fpga_reset_n),
        .adc_0_clk_6553600hz_clk                   (clk6553600),
        .pll_6553600_outclk1_clk                   (clk6553600),
        .digital_external_connection_export        (dout),
...
```

La declaración de los puertos GPIO, por defecto, está en inout pero se cambia el GPIO0 a input i el GPIO1 a output:

```
...
//////// GPIO ////////
      input      [35:0]          GPIO_0,
      output      [35:0]          GPIO_1,
...
```

De este modo se puede añadir las dos IP de Altera; la LVDS de entrada y el buffer de salida con sus respectivas llamadas desde el Top Level como se realizó en los apartados 3.3.1 y 3.3.2.

```

assign GPIO_1[0]=obuffer;

iBUFF an(
  GPIO_0[0],
  clk6553600,
  lvds_out
  );

oBUFF el_Buffer_Salida (
  .datain ( nbuffer ),
  .dataout ( obuffer )
  );
```

[21] Ejemplos y más recursos para la DE1-SoC en `http://www.terasic.com/downloads/cd-rom/de1-soc`

Una vez creadas las dos IPs en el *Project* Navigator de ©Quartus, en la pestaña *IP components* deberían aparecer en el listado como en la figura 59.

	Entity	IP Component	Version	Supported Device Families	IP File
	soc_system	<Qsys System>		Cyclone V	soc_system/synthesis/soc_system.qip
	iBUFF	ALTLVDS_RX	18.0	Cyclone V	iBUFF.qip
	oBUFF	ALTIOBUF	18.0	Cyclone V	oBUFF.qip

Figura 59: Vista de componentes del proyecto en *Project* Navigator de ©Quartus

Antes de la compilación se asignan los pines LVDS y del buffer como en el apartado 3.3 dado que por defecto están a $3,3\,V$. Esto provoca un error en la compilación por incompatibilidad con el I/O bank 4A. En este banco de pines se encuentran los GPIO, los LEDR y el reloj CLOCK2_50. También ocurre una incompatibilidad con la DRAM de la Tarjeta DE1-SOC por lo que se eliminan las asignaciones a los pines de la DRAM, el GPIO0[2] y el CLOCK2_50. El banco 4A se cambia a $2,5\,V$ y se vuelve a asignar el pin LVDS al GPIO[0].

9.2.6. Generación del RBF

Para que la FPGA quede configurada se requiere del archivo con los bits necesarios o bitstream. Es por ello que se requiere el archivo .sof generado tras la copilación del proyecto. En el menú de ©Quartus → File → Convert Programmin Files se debe configurar los siguientes apartados:

- Programming file type → Raw Binary File(.rbf)
- Configuration device → EPCE16
- Mode → Passive Parallel x16
- File name → soc_system.rbf
- Input file to convert → Add File (se añade .sof)

Tras la configuración del menú indicado anteriormente o como puede verse en la figura 60 se genera el archivo .rbf en la ruta especificada.

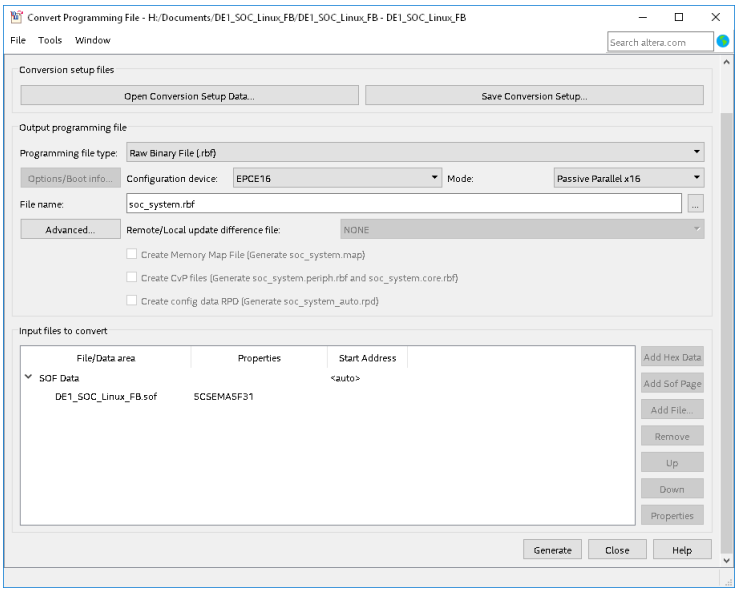

Figura 60: Herramienta generadora archivo rbf.

9.2.7. Generación del .dtb

Este archivo es uno de los más comprometidos del proyecto. Cualquier error aquí puede no ser detectado y generar que el sistema Linux no arranque. Para generar el dtb primero hay que generar el dts. Para generar el dts se requiere de la herramienta sopc2dts presente en el software SoC EDS Command Shell de Intel. Los archivos necesarios se encuentran en la propia carpeta del proyecto excepto el archivo:

hps_common_board_info.xml

Se encuentra en la ruta:

`X:\intelFPGA\18.0\embedded\examples\hardware\cv_soc_devkit_ghrd\`

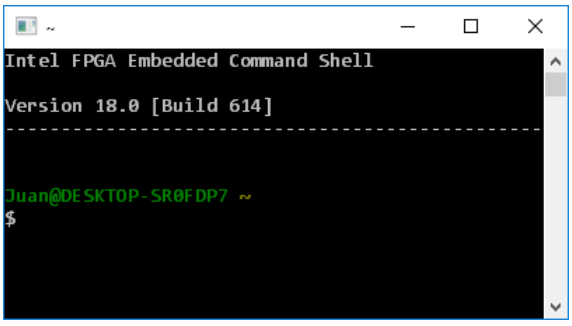

Figura 61: Intel FPGA Command Shell en modo Administrador.

```
$ cd /cygdrive/h/Documents/DE1_SOC_Linux_FB/
$ sopc2dts --input soc_system.sopcinfo  --output socfpga.dts  --type dts  --board
    soc_system_board_info.xml  --board hps_common_board_info.xml  --bridge-removal all
    --clocks
```

Este comando devuelve los siguientes errores:

```
Component ADC_0 of class ADC is unknown
DTAppend: Unable to find parent, null, for status. Adding to root
DTAppend: Unable to find parent, null, for status. Adding to root
DTAppend: Unable to find parent, null, for spidev@0. Adding to root
DTAppend: Unable to find parent, null, for spidev@0. Adding to root
```

Se edita el archivo ADC_hw.tcl y se añaden las siguientes lineas al principio del mismo:

```
set_module_assignment embeddedsw.dts.vendor "intel"
set_module_assignment embeddedsw.dts.compatible "dev,ADC_hw"
set_module_assignment embeddedsw.dts.group "ADC_hw"
```

Tras la generación del componente qsys y la compilación se vuelve a lanzar el comando anterior sopc2dts. En este caso desaparece el error del componente aunque siguen existiendo otros errores que se desprecian dado que así lo aconseja el fabricante.

```
DTAppend: Unable to find parent, null, for status. Adding to root
DTAppend: Unable to find parent, null, for status. Adding to root
DTAppend: Unable to find parent, null, for spidev@0. Adding to root
DTAppend: Unable to find parent, null, for spidev@0. Adding to root
```

Se prodece a generar el dtb mediante el siguiente comando:

```
$   dtc -I dts -O dtb -o socfpga.dtb  socfpga.dts
ERROR (duplicate_node_names): Duplicate node name /spidev@0
ERROR (duplicate_node_names): Duplicate node name /sopc@0/pmu0
ERROR (duplicate_node_names): Duplicate node name /sopc@0/fpgabridge@0
ERROR (duplicate_node_names): Duplicate node name /sopc@0/fpgabridge@1
ERROR (duplicate_node_names): Duplicate node name /sopc@0/fpgabridge@2
ERROR (duplicate_node_names): Duplicate node name /sopc@0/
flash@0xff704000/slot@0
ERROR (duplicate_node_names): Duplicate node name /sopc@0/pmu0/cti0@ff118000
ERROR (duplicate_node_names): Duplicate node name /sopc@0/pmu0/cti0@ff119000
ERROR: Input tree has errors, aborting (use -f to force output)
```

Para eliminar los errores resulta necesario editar el archivo dts. Se eliminan los duplicados como:

```
    pmu: pmu0 {
    }; //end pmu0 (pmu)

    fpgabridge0: fpgabridge@0 {
    }; //end fpgabridge@0 (fpgabridge0)

    fpgabridge1: fpgabridge@1 {
    }; //end fpgabridge@1 (fpgabridge1)

    fpgabridge2: fpgabridge@2 {
    }; //end fpgabridge@2 (fpgabridge2)


      slot_0: slot@0 {
      }; //end slot@0 (slot_0)

      cti0: cti0@ff118000 {
      }; //end cti0@ff118000 (cti0)

      cti1: cti0@ff119000 {
      }; //end cti0@ff119000 (cti1)
```

Pero también se borran dos componentes que no son necesarios:

```
spidev0: spidev@0 {
  compatible = "spidev";  /* appended from boardinfo */
  reg = <0>;  /* appended from boardinfo */
  spi-max-frequency = <100000000>;  /* appended from boardinfo */
  enable-dma = <1>; /* appended from boardinfo */
}; //end spidev@0 (spidev0)

spidev1: spidev@0 {
  compatible = "spidev";  /* appended from boardinfo */
  reg = <0>;  /* appended from boardinfo */
  spi-max-frequency = <100000000>;  /* appended from boardinfo */
  enable-dma = <1>; /* appended from boardinfo */
}; //end spidev@0 (spidev1)
```

Así como el componente alt_vip_vfr_vga al que se cambia una línea

```
clocks = <&pll_stream &clk_0>;
```

por esta otra:

```
clocks = <&periph_pll &clk_0>;
```

Se genera el dtb sin errores mediante el siguiente comando:

```
$ dtc -I dts -O dtb -o socfpga.dtb  socfpga.dts
```

9.2.8. Generación del U-Boot

Para la generación del archivo U-Boot se requiere de la compilación en C del proyecto. Para ello se utiliza de la herramienta *bsp_editor.exe* del Intel FPGA Command Shell. Tras la configuración del bsp se crea una carpeta en el proyecto llamada *software/spl_bsp* donde se encuentran entre otros el *Makefile*.

```
$ bsp-editor.exe
```

Tras el inicio del generador del bsp mediante File→ New HPS BSP se crea un nuevo BSP y se añade el directorio de los archivos de configuración:

```
H:\Documents\DE1_SOC_Linux_FB\hps_isw_handoff\soc_system_hps_0
```

Se activa la pestaña de FAT_SUPPORT y el resto por defecto.

```
$ cd software/spl_bsp/
$ make clean
$ make
$ make uboot
```

La primera vez que se compila el proyecto es necesario descomprimir los archivos de configuración en nuestro directorio. Existe un error en el descompresor y el make inicial genera un error, por tanto se debe proceder a descomprimir de forma manual y modificar el Makefile. Se procede del mismo modo si se borra la carpeta software. Una vez realizado ya no hace falta pero, tras la primera compilación, se realiza el make clean inicial antes del make.

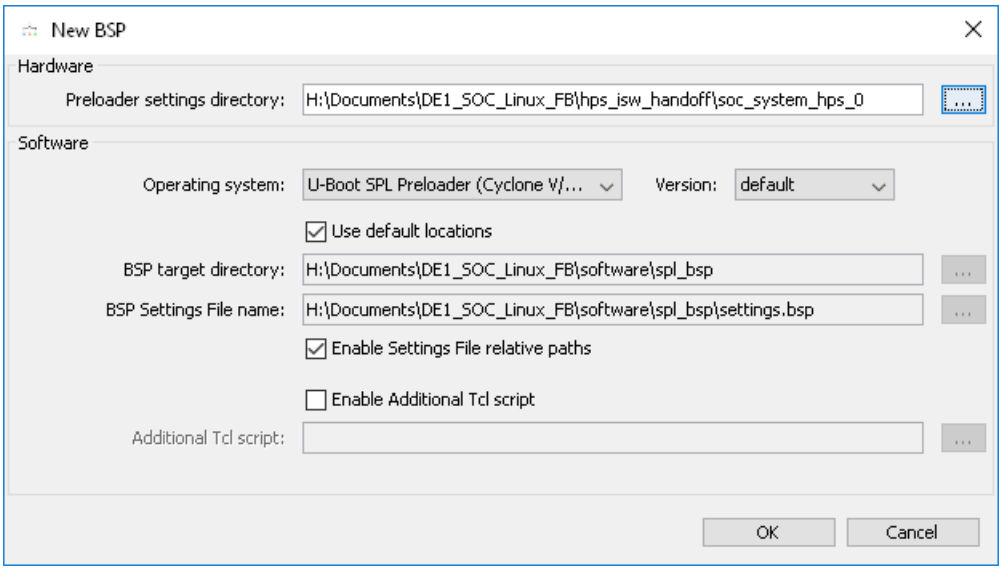

Figura 62: Nuevo BSP mediante bsp-editor.exe.

```
$ make

tar zxf /cygdrive/e/intelFPGA/18.0/embedded/host_tools/altera/
preloader/uboot-socfpga.tar.gz
tar: Error opening archive: Failed to open '/cygdrive/e/intelFPGA/18.0/
embedded/host_tools/altera/preloader/uboot-socfpga.tar.gz'
make: *** [uboot-socfpga/.untar] Error 1

$ tar zxf /cygdrive/e/intelFPGA/18.0/embedded/host_tools/altera/
preloader/uboot-socfpga.tar.gz
```

Se comentan las lineas 215 y 216 del Makefile:

```
213 ...
214 $(UNTAR_SRC): $(TGZ)
215 #@$(RM) $(PRELOADER_SRC_DIR)
216 #$(untar_recipe)
217 @$(CHMOD) -R 755 218(PRELOADER_SRC_DIR)
219 $(stamp)
220 ...
```

Tras comentar el archivo es necesario ejecutar el descompresor de nuevo. Si se realiza el *make* generá un error y se debe proceder con *make clean*, descomprimir y ejecutar *make* de nuevo.

```
$ tar zxf /cygdrive/e/intelFPGA/18.0/embedded/host_tools/altera/preloader/uboot-socfpga.
    tar.gz
$ make
$ make uboot
```

9.2.9. Copiar archivos a la tarjeta SD

La carga de los archivos necesarios a la tarjeta pasa por la preparación de la misma desde la imagen de ubuntu y la generación del preloader. Para ello se introduce la SD, ya preparada, en el PC donde se está ejecutando la Intel FPG Command Shell y en el

mismo directorio de trabajo *software/spl-bsp/* se ejecuta el siguiente script donde *k* es la letra de la unidad:

```
$ alt-boot-disk-util.exe -p preloader-mkpimage.bin -a write -d k

Altera Boot Disk Utility
Copyright (C) 1991-2014 Altera Corporation

Altera Boot Disk Utility was successful.
```

Tras la generación del preloader en la tarjeta se copian el resto de archivos a la SD mediante los comandos siguientes:

```
$ cp uboot-socfpga/u-boot.img /cygdrive/k
$ cp ../../socfpga.dtb /cygdrive/k
$ cp ../../soc_system.rbf /cygdrive/k
$ sync
```

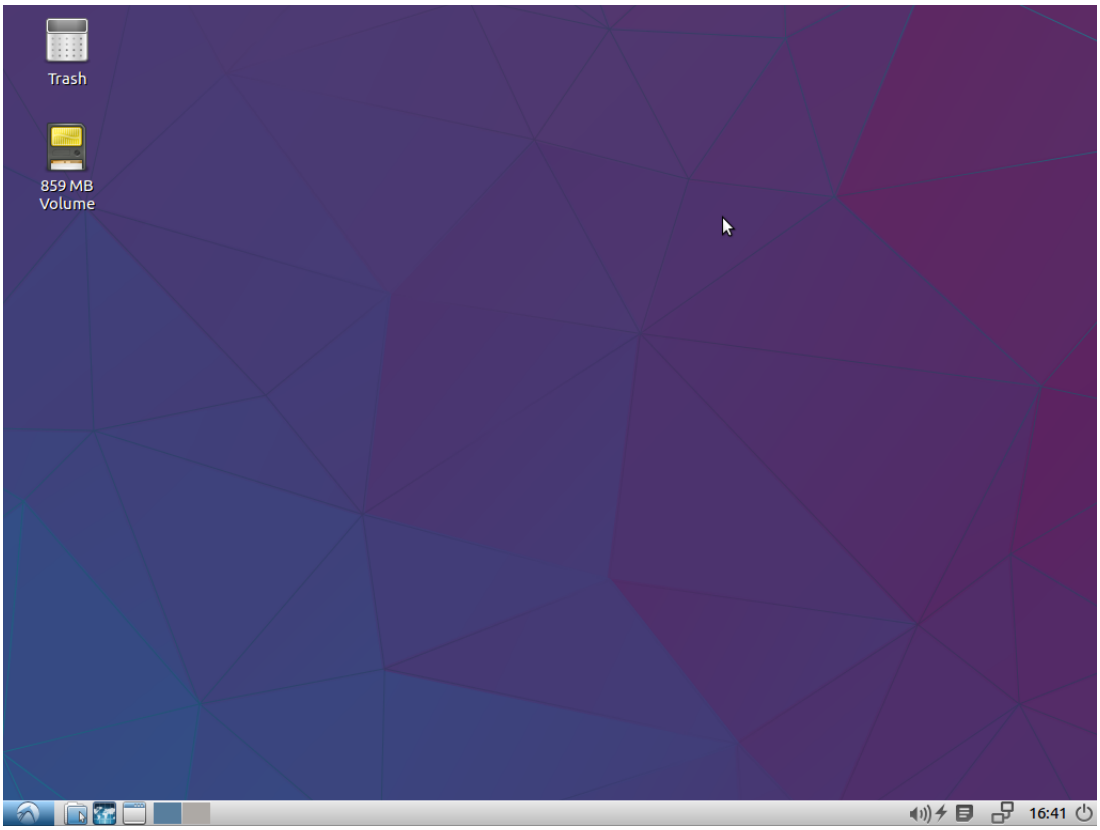

Figura 63: Entorno gráfico funcionando en DE1-SoC.

9.2.10. Preliminares

Para la captura del dato procedente del ADC y su muestra en pantalla se ha optado por un pequeño script de Python. En este caso se utiliza la versión nativa de Python 3.5.2. Una ayuda para conocer cómo se mapean los componentes consiste en generar el archivo hps_0.h. Para ello se ejecuta el siguiente script en el directorio del proyecto:

```
sopc-create-header-files \
"./soc_system.sopcinfo" \
--single hps_0.h \
--module hps_0
```

Dado que la versión del sistema Ubuntu 16.04.1 LTS Xenial, con la que cuenta la imagen de la tarjeta, se encuentra desactualizada. En primer lugar, se procede a actualizar los paquetes del sistema mediante el comando apt:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Una vez actualizado el sistema se instalan las librerías necesarias para ejecutar el script:

```
$ sudo apt-get install -y python3-pip
$ pip3 install numpy
$ sudo apt-get install aptitude
$ sudo aptitude install python3-matplotlib
```

Aunque resulte redundante se utilizan tres gestores de paquetes. Primero con *apt-get* se instala el gestor de paquetes *pip* del intérprete del lenguaje Python, éste se utiliza para instalar la librería *numpy*. Para la instalación de la librería *Matplotlib* es preferible utilizar el gestor de paquetes de la distribución Debian ya que la actual de Ubuntu presenta problemas de compatibilidad que no resuelve automáticamente como *aptitude*.

9.3. Manual de usuario

Para el correcto funcionamiento se requiere de la connexión previa del sistema a una pantalla VGA, la fuente de alimentación y a un ratón y teclado USB. Se introduce la tarjeta SD preparada y el sistema analógico de entrada de señal y circuito RC.

Tras pulsar el botón rojo de la DE1-SoC se iniciará el proceso de configuración de la placa y el arranque del sistema linux. Una vez iniciado el sistema puede que se requiera la intervención del usuario para cancelar la actualización del sistema u otro aviso que se pueda generar tras el arranque. Una vez finalizada dicha intervención se puede proceder a iniciar la captura de 15000 muestras mediante doble click en el icono eNose del Escritorio. Una imagen aparecerá en pantalla actualizando la el proceso de captura de la señal como puede verse en la figura 64. Al terminar la captura se genera un archivo llamado datos.dat en el Escritorio.

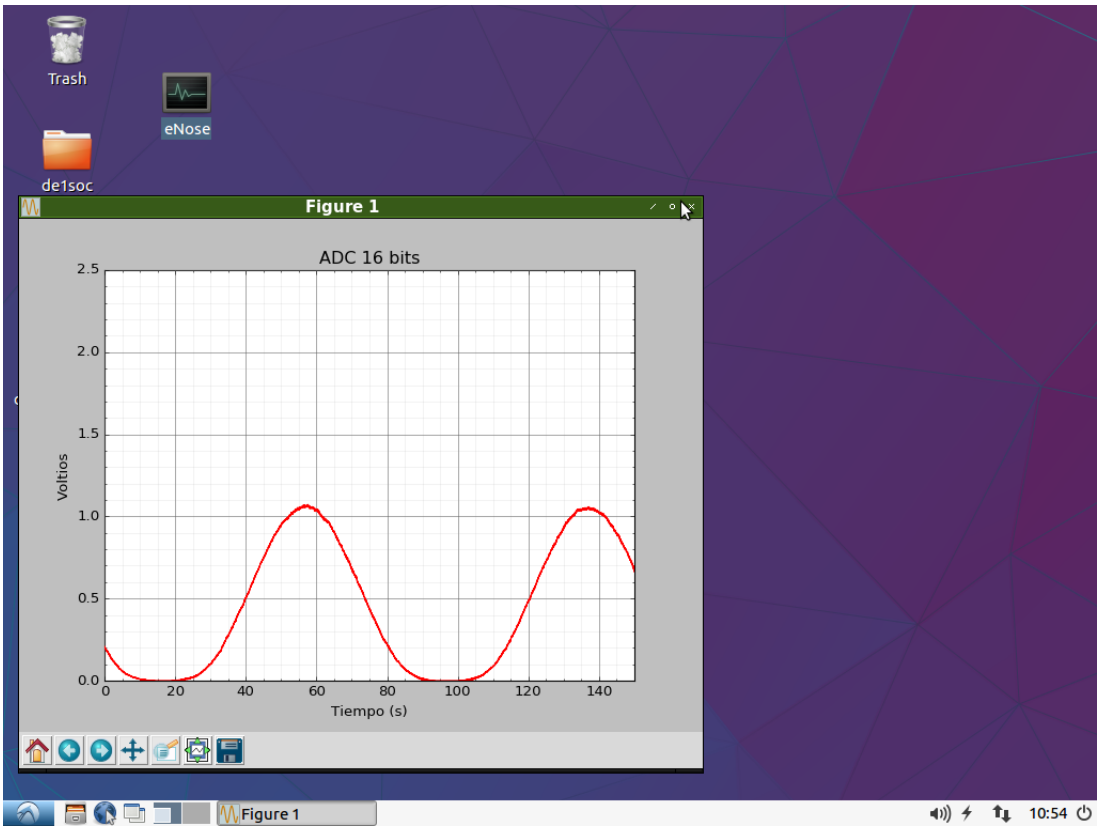

Figura 64: Acceso directo en escritorio y aplicación en ejecución.