

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**MÁSTER UNIVERSITARIO EN INGENIERÍA
DE SISTEMAS ELECTRÓNICOS
TRABAJO FIN DE MÁSTER**

**DESARROLLO DE UN SISTEMA DE BAJO
COSTE PARA LA ADQUISICIÓN,
PROCESADO Y PRESENTACIÓN DE
SEÑALES DE ALTA FRECUENCIA**

**Carlos Cortés Sánchez
2018**

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

TRABAJO FIN DE MÁSTER

Título: Desarrollo de un sistema de bajo coste para la adquisición, procesado y presentación de señales de alta frecuencia.

Autor: D. Carlos Cortés Sánchez

Tutor: D. Francisco José López Hernández

Ponente: D.

Departamento: Tecnología Fotónica y Bioingeniería.

MIEMBROS DEL TRIBUNAL

Presidente: D.

Vocal: D.

Secretario: D.

Suplente: D.

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de:
.....

Madrid, a de de 20...

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**MÁSTER UNIVERSITARIO EN INGENIERÍA
DE SISTEMAS ELECTRÓNICOS
TRABAJO FIN DE MÁSTER**

**DESARROLLO DE UN SISTEMA DE BAJO
COSTE PARA LA ADQUISICIÓN,
PROCESADO Y PRESENTACIÓN DE
SEÑALES DE ALTA FRECUENCIA**

**CARLOS CORTÉS SÁNCHEZ
2018**

RESUMEN

En el marco de la comunidad creadora de contenido físico mediante placas de desarrollo y prototipado, se plantea el diseño de un sistema de adquisición de datos con el objetivo de obtener un sistema abierto, de bajo coste, y fácilmente ampliable para facilitar el acceso a sistemas de medida tanto a la comunidad como al mundo académico. Como valor añadido, se implementará el sistema con la posibilidad de medir señales periódicas de frecuencias por encima de las posibilidades del conversor analógico digital (ADC) que incorpora el sistema. Con el objetivo de poder reconstruir señales periódicas de alta frecuencia en sistemas limitados en ancho de banda utilizando técnicas de interpolación y diezmado. Además, se realizará un análisis exhaustivo del proceso de desarrollo en esta memoria con el objetivo de que se sea fácilmente ampliable por quien tenga a bien desarrollar, mediante la base que se proporciona en este trabajo, un sistema mejorado. El sistema está compuesto por un generador de señales para pruebas desarrollado con Arduino, una tarjeta de adquisición de datos formados por una placa de desarrollo de Digilent que, incorpora: ADC, matriz de puertas lógicas programables y un transmisor USB-Serie. Dicha placa, se comunica mediante USB con una Raspberry Pi que, mediante el lenguaje Python, procesa y muestra los datos además de servir como interfaz de usuario.

SUMMARY

Supported by the “Makers” community, in this project, a data acquisition system is developed to create an open, low cost, and scalable system. The objective of this project is making the community easier to have access to data acquirement.

However, the system will be able to resample undersampled signals to bring the possibility of using limited analog to digital converters. In addition, a deep analysis of the development process will be done because of the will of making this system improvable. The system is composed by: a signal generator developed in Arduino system, an evaluation board Arty S that contains an analog to digital converter (ADC) and a field programable gate array (FPGA). This board will be used for acquiring the data that will be transmitted to a Raspberry Pi through USB. Then, Raspberry pi will process and print the data using Python.

PALABRAS CLAVE

Adquisición de datos, reconstrucción de señales periódicas, Spartan 7, submuestreo, Python.

KEYWORDS

Data acquisition, resample, periodic signal, undersampling, Python.

ÍNDICE DEL CONTENIDO

1. INTRODUCCIÓN Y OBJETIVOS.....	1
1.1. Introducción	1
1.2. Objetivos	4
2. DESARROLLO	7
2.1. Introducción	7
2.2. Osciladores de síntesis digital directa (DDS).....	7
2.3. Placa de evaluación arty S7.....	10
2.3.1. XADC	12
2.3.2. FIFO.....	16
2.3.3. Medidor de frecuencia	17
2.3.4. Transmisión serie mediante usb.....	19
2.4. Raspberry Pi	21
2.4.1. Sistema operativo Raspbian	22
2.4.2. Recepción de datos	24
2.4.3. Interfaz gráfica en PyQt	25
2.4.4. Uso de SCIPY para cálculo de transformada rápida de Fourier discreta.....	27
2.4.5. Ajuste de frecuencia mediante interpolación y diezmado en python	28
3. RESULTADOS	29
3.1. Introducción	29
3.2. Señales procedentes de FPGA.....	30
3.3. Comparativa de señales medidas generadas con el generador de funciones DS345 31	
3.4. Comparativa de señales generadas con el DDS AD9850	39
4. CONCLUSIONES Y LÍNEAS FUTURAS	41
4.1. Conclusiones	41
4.2. Líneas futuras	41
5. BIBLIOGRAFÍA.....	43
ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y	
AMBIENTALES	44
ANEXO B: CÓDIGO DE LOS DISTINTOS BLOQUES	44

ÍNDICE DE FIGURAS

Figura 1.1 Proceso de adquisición de datos	2
Figura 1.2 Ranking de Spectrum IEEE.....	4
Figura 2.1 Diagrama de bloques del funcionamiento del sistema	7
Figura 2.2 Diagrama de bloques del funcionamiento de un DDS completo para generar un tren de pulsos.[6].....	8
Figura 2.3 Interfaz gráfica creada con Processing para programar DDS desde Arduino	9
Figura 2.4 Esquema de conexiones entre AD9852 y Arduino.....	9
Figura 2.5 Salida sinusoidal de AD9850 configurado a 100 kHz medida con osciloscopio Tektronix TDS 2022B.....	10
Figura 2.6 Placa de evaluación Arty S7 de Digilent[8]	11
Figura 2.7 Periféricos de Arty S7	11
Figura 2.8 Diagrama de bloques del XADC[5]	12
Figura 2.9 Esquema eléctrico de la entrada analógica de la FPGA.[8]	13
Figura 2.10 Puertos del bloque XADC[5]	13
Figura 2.11 Entidad que contiene al XADC y se comunica con la FIFO.....	15
Figura 2.12 Entidad de la FIFO generada por el IPcore de Xilinx	16
Figura 2.13 Diagrama de la máquina de estados finito que controla la FIFO.	17
Figura 2.14 Cronograma de las señales principales que controlan el medidor de frecuencia.	18
Figura 2.15 Selección de entradas para el bloque transmisor.	18
Figura 2.16 Bloque de la entidad generada para medir frecuencias.	19
Figura 2.17 Diagrama de estados de la FSM que controla el medidor de frecuencias.	19
Figura 2.18 Forma de onda de una transmisión serie de 8 bits con un bit de parada.	20
Figura 2.19 Conversor USB-UART.[8].....	20
Figura 2.20 Entidad del bloque transmisor en la FPGA.	21
Figura 2.21 Raspberry Pi 3 modelo B.....	22
Figura 2.22 Ejemplo de ejecución de Gedit en RPi desde SSH en modo gráfico.	23
Figura 2.23 Trama inicial.....	25
Figura 2.24 GUI creada con PyQt.....	26
Figura 2.25 Proceso de remuestreo	28
Figura 3.1 Sistema de adquisición de datos.	30
Figura 3.2 Señal fin de conversión del XADC medida con osciloscopio TDS 2022B de Textronix.....	30
Figura 3.3 Señal de transmisión serie muestreando una señal continua	31
Figura 3.4 Señal de 70 kHz medida con osciloscopio TDS 2022B	32
Figura 3.5 Señal sinusoidal de 70kHz medido con el sistema de adquisición de datos.	32
Figura 3.6 Señal sinusoidal de 20 kHz medida con el sistema de adquisición de datos.....	33
Figura 3.7 Periodograma normalizado de la señal sinusoidal de 20 kHz medida con el sistema de adquisición de datos.	33
Figura 3.8 Señal Cuadrada de 20 kHz medida con el sistema de adquisición de datos.....	34
Figura 3.9 Periodograma de señal cuadrada de 20 kHz medida con el sistema de adquisición de datos.	34
Figura 3.10 Señal triangular de 20 kHz medida con el sistema de adquisición de datos.	35
Figura 3.11 Periodograma de señal triangular de 20 kHz medida con el sistema de adquisición de datos.	35
Figura 3.12 Periodograma de señal sinusoidal de 350 kHz medida con el sistema de adquisición de datos.	36
Figura 3.13 Señal de 350 kHz medida con el sistema de adquisición de datos.	36

Figura 3.14 Señal de 700 kHz medida con el sistema de adquisición de datos.	37
Figura 3.15 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos.	37
Figura 3.16 Señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.	38
Figura 3.17 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.	38
Figura 3.18 Señal sinusoidal de 1 MHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.	38
Figura 3.19 Señal sinusoidal de 5 MHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.	39
Figura 3.20 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.	39
Figura 3.21 Señal sinusoidal de 10 kHz generada por el circuito de test.	40
Figura 3.22 Señal sinusoidal de 900 kHz generada por el circuito de test.	40
Figura 3.23 Periodograma de la señal sinusoidal de 900 kHz generada por el circuito de test.	40

1. INTRODUCCIÓN Y OBJETIVOS

1.1. INTRODUCCIÓN

La idea de fábrica está cambiando, de igual forma que internet supuso una democratización en cuanto a información, una nueva clase de tecnología “Maker”, es decir, creadora, surge de la aparición de numerosas herramientas que permiten un acceso rápido y sencillo a tecnologías clásicas para crear sistemas físicos de forma sencilla e intuitiva. Desde impresoras 3D, placas de desarrollo, entornos de desarrollo especialmente creados para principiantes, y, sin lugar a duda, una comunidad de usuarios en la que se comparten libremente los proyectos creados ha generado un fenómeno parecido al Web que tuvo lugar hace unos años.

Según [1], el surgimiento de la comunidad “Maker”, es la vuelta a la escuela-taller adaptada a la era Web. En el mismo documento, se hace referencia al incremento del hardware abierto dentro de la comunidad. La aparición de la placa de desarrollo Arduino y la liberación de componentes electrónicos por parte de Google, son ejemplos de gran beneficio económico basadas en el hardware libre.

Continuando con la idea anterior, miles de proyectos de hardware libre han conseguido generar beneficio económico, cambiando así, tanto el paradigma de la industria como el de la propia comunidad creadora. Es decir, tanto por parte de la industria que se abre a utilizar estas herramientas para prototipado y formación, así como la comunidad creadora está empezando a transformarse en una comunidad emprendedora con iniciativas como “Kickstarter” (Web para financiación de proyectos).

Dentro de este marco se encuentra el diseño que en este trabajo se plantea. Utilizando plataformas con el soporte de la comunidad creadora: Raspberry Pi, Arduino, e, incluso, una placa de desarrollo FPGA pensada para “Makers”. Estos elementos se combinan en este trabajo para construir un sistema de adquisición de datos (DAQ).

La idea es crear un prototipo funcional utilizando un amplio repertorio de programas de diseño, lenguajes de programación además de sistemas físicos abiertos pensando en la escalabilidad del proyecto y el uso libre de otras personas en el futuro. De hecho, en este mismo trabajo, se deja la puerta abierta en el capítulo 4 de la memoria, a mejoras múltiples que no se han podido abarcar, pero sí analizar durante el desarrollo del trabajo.

En 1981 IBM lanzó el primer ordenador personal (PC). El carácter de libre diseño del PC permitió que se creasen multitud de aplicaciones por parte de desarrolladores independientes.

Acompañado por un descenso significativo en el coste y una expansión veloz del software disponible, el hecho de ser una plataforma basada en microprocesador ha convertido al PC en la plataforma más usada en áreas como el procesamiento de señal y de imagen, el control industrial y la adquisición de datos.

Los sistemas de adquisición de datos (DAQ) son ampliamente utilizados en ámbitos industriales y de laboratorio como aplicaciones de control, monitorización, test, medición y automatización entre otros. El objetivo principal de los sistemas de adquisición de datos es tomar muestras de fenómenos físicos convirtiendo la señal analógica en digital y siendo enviada a otros sistemas para un análisis en profundidad. Los sistemas de adquisición comerciales se diferencian entre sí atendiendo a sus características en ancho de banda, rango dinámico, frecuencia de muestreo, pero, normalmente, necesitan de un PC para recolectar los datos. [2]

Un DAQ se compone de un conjunto de bloques hardware que desempeñan funciones distintas e, incluso, pueden utilizar distintas plataformas. Es tarea del ingeniero el conseguir que estos bloques independientes funcionen en el sistema completo.

Los elementos básicos de un sistema de adquisición de datos son[3]:

1. Sensores y transductores.
2. Acondicionamiento de señal.
3. Hardware de adquisición de datos.
4. Sistema operativo (PC).
5. Software de adquisición de datos.

Los sensores y transductores actúan como interfaces entre el mundo real y el sistema de adquisición de datos. Con el surgimiento del internet de las cosas (IoT) existen numerosos sensores que realizan por si mismos la conversión analógico-digital e, incluso, algunos procesados de señal. Bajo este paradigma se extrae la necesidad de sistemas de adquisición de datos versátiles y compatibles con estas formas de medir. Las formas más utilizadas para comunicarse digitalmente con los sensores son I2C y SPI[3]

El acondicionamiento de señal hace referencia a los procesos por los cuales la señal analógica de entrada se modifica para adaptarla al hardware de adquisición de datos. Entre dichos procesos se encuentran: Filtrado, amplificación linealización, aislamiento y excitación.[3]

El hardware de adquisición de datos es el componente que recoge las muestras y controla el sistema. Las tareas que puede desarrollar el DAQ son [3]:

1. La entrada, procesamiento y conversión a formato digital, utilizando ADC o una señal de datos medido de un sistema o proceso, estos datos son transferidos a un ordenador para su muestra almacenaje y análisis.
2. La entrada de señales digitales que contienen información de un sistema o proceso.
3. Generar una salida digital de control.

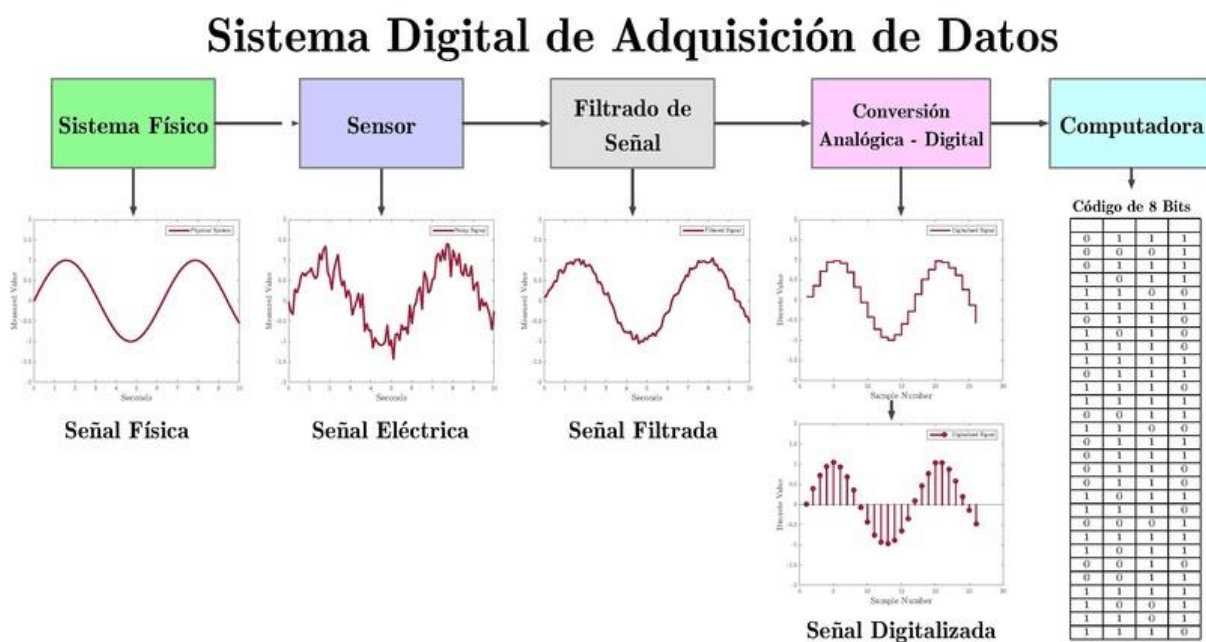


Figura 1.1 Proceso de adquisición de datos

Los sistemas empotrados para la adquisición de datos normalmente necesitan de un sistema operativo para su correcto funcionamiento. Aun así, la ejecución de partes críticas en hardware

dedicado programable como las FPGA reducen el tiempo de ejecución de dichas secciones críticas. La posibilidad de programación de las FPGA permite desarrollar funciones software específica optimizadas en HW[2].

En un sistema de adquisición de datos, a medida que aumenta la frecuencia de la señal a muestrear, se hace cada vez más importante que la frecuencia de muestreo se encuentre perfectamente acotada para ofrecer una buena representación de la señal una vez medida. Las FPGA ofrecen una muy eficaz gestión de las señales de reloj. Las FPGAs modernas utilizan gestores de reloj de modo mixto (MMCM). Su funcionalidad es muy similar a un lazo de seguimiento de fase.

Respecto al software de adquisición de datos, se define como el software que acompaña al dispositivo de adquisición de datos. Según el nivel al que se programe el dispositivo se diferencian 3 casos[3]:

1. Programar directamente los registros del hardware.
2. Utilizar software de bajo nivel específico proporcionado por el fabricante
3. Utilizar aplicaciones externas como LabVIEW para extraer los datos.

Si se requiere de una aplicación en tiempo real, la potencia del procesador de la computadora que alberga el sistema operativo puede limitar la capacidad del DAQ. Técnicas para abstraer del procesador de la adquisición de datos es el uso de acceso directo a memoria (DMA). En este aspecto, sistemas operativos multitarea, dificultan el normal funcionamiento del DMA retrasando las señales que lo controlan. Por otra parte, sistemas operativos multitarea abren un abanico importante de funcionalidad al DAQ pudiendo realizar múltiples tareas al mismo tiempo, ofreciendo una experiencia de usuario más completa[3].

Python

Python es un lenguaje de programación libre y de alto nivel que destaca por su flexibilidad y por la posibilidad que ofrece de implementar diferentes paradigmas de programación, como programación orientada a objetos, en un mismo lenguaje. Esta flexibilidad es una gran ventaja sobre otros lenguajes de programación.

Además, es el lenguaje más utilizado en computadoras de placa única como Raspberry Pi o Beaglebone lo que, unido a su gran comunidad, ofrecen numerosos ejemplos, librerías y documentación que ayuda a completar proyectos con un reducido tiempo de desarrollo. Las librerías disponibles para Python permiten comunicarse sobre RS232 y GPIB que son estándares de comunicación para test y medición.

Según el último ranking de lenguajes de programación de IEEE Spectrum Python ha obtenido el primer puesto en programación para sistemas empujados.























Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	98.4
3. C	  	98.2
4. Java	  	97.5
5. C#	  	89.8
6. PHP		85.4
7. R		83.3
8. JavaScript	 	82.8
9. Go	 	76.7
10. Assembly		74.5

Figura 1.2 Ranking de Spectrum IEEE

En el informe, se destaca Python como un lenguaje emergente, capaz de desbancar a los tradicionales lenguajes de programación para sistemas empujados debido a los cada vez más potentes sistemas que permiten evaluar código al vuelo. Y, como se ha comentado anteriormente, Python se relaciona fácilmente con el hardware de los sistemas empujados.

Otra razón que se destaca en el informe es la existencia de librerías de alta calidad en campos estadísticos y de aprendizaje automatizado ganándole terreno al lenguaje R, tradicionalmente utilizado en dichos campos.

Uno de los parámetros fundamentales en los DAQ es su ancho de banda. A la vez que se eleva la frecuencia a la que se trabaja, los DAQ tienden a elevar su coste de forma considerable. Este ancho de banda suele limitarlo el hardware de adquisición de datos. Según el criterio de Nyquist, es necesario muestrear al doble de la frecuencia máxima de una señal para poder reconstruirla sin solape espectral[4].

Sin embargo, si se conocen de antemano las condiciones de las señales a recolectar, por ejemplo, si son periódicas, si son tonos puros y su ancho de banda en el que trabaja la señal, se pueden reconstruir señales por encima del criterio de Nyquist. Al muestrear por encima de la frecuencia máxima que permite Nyquist en señales periódicas, se produce solape espectral, que, en señales periódicas y conociendo la frecuencia original, puede reconstruirse utilizando decimación e interpolación de la señal[4]. Una vez reconstruida la señal, es necesario ajustarla en frecuencia teniendo en cuenta el proceso de decimación e interpolación (remuestreo) en conjunto.

1.2.OBJETIVOS

Como trabajo de fin de máster de la titulación Máster Universitario en Ingeniería de Sistemas Electrónicos (MUISE), se propone el desarrollo de un sistema de adquisición, procesado y presentación de datos de bajo coste con capacidad de trabajar con señales de alta frecuencia. Los elementos que compondrán el sistema serán una placa de evaluación FPGA Arty S7 y un ADC para la adquisición y registro de datos en memoria y, para su visionado, se hará uso de una Raspberry Pi 3, la cual procesará y mostrará los datos recibidos mediante comunicación

serie desde la FPGA. Como ADC se hará uso del que incorpora la placa y, que está limitado a 1 MSPS.

Por lo tanto, los objetivos del trabajo fin de titulación son:

1. Estudio de la FPGA Spartan 7 que incluye la placa de evaluación Arty S7.
2. Estudio de la placa de evaluación Arty-S7, así como de sus periféricos (XADC, Memoria, GPIO, Pmod).
3. Estudio de las posibilidades de procesamiento en Raspberry Pi.
4. Diseñar la interfaz de usuario para utilizar la Raspberry Pi 3 basado en Python.
5. Diseñar el código de descripción hardware para implementar todas las funcionalidades de la FPGA: configuración de ADC, comunicación serie con Raspberry Pi y registro de datos ADC.

Otros objetivos:

1. Diseñar un sistema basado en Arduino y Processing y sintetizadores digitales de señal para, mediante interfaz de usuario en un PC, generar las señales de prueba para el sistema.

Como metodología se plantea el diseñar un sistema base que contenga todas las funcionalidades descritas para señales que no sean de alta frecuencia. Más adelante, y, una vez finalizado el sistema base, se irán añadiendo y mejorando las funcionalidades adecuadas para poder adquirir procesar y muestrear señales a alta frecuencia como: ADCs más rápidos, comunicaciones serie con mayor tasa de transmisión, uso distinto de la memoria, procesados de señal más rápidos utilizando la unidad de procesamiento gráfico (GPU) de la Raspberry Pi 3.

A continuación, se detalla el precio de los componentes utilizados y el lugar de su compra:

Componente	Precio (IVA incluido)	Lugar de compra
Arduino mini	11,70 €	RS
Kit de desarrollo Raspberry Pi 3B	55,47€	RS
Arty S7-50	145,26 €	RS
AD9850	30,75 €	Lualtec

Tabla 1.1 Lista de materiales para el desarrollo del proyecto.

Suman 243,18 €, aunque, el AD9850 y el Arduino mini, son parte del circuito de test, realmente, el sistema de adquisición sumaría 200,73 €. En realidad, se han reutilizado muchos componentes, pero, si se decidiese realizar desde cero, estos serían los precios. Además, no se ha tenido en cuenta los honorarios del ingeniero que realiza el proyecto ni el posible alquiler de los equipos de medida.

2. DESARROLLO

2.1. INTRODUCCIÓN

Durante el desarrollo del sistema se ha trabajado en distintas plataformas y lenguajes de programación. En la figura 2.1 se muestra un diagrama de los sistemas utilizados y cómo se relacionan entre ellos. En este apartado se realizará un recorrido explicando brevemente las funciones principales y las formas de interconexión de cada una de las partes que aparecen en la figura 2.1. Posteriormente, en este mismo capítulo, se llevará a cabo un análisis más detallado del funcionamiento de los bloques en cuestión.

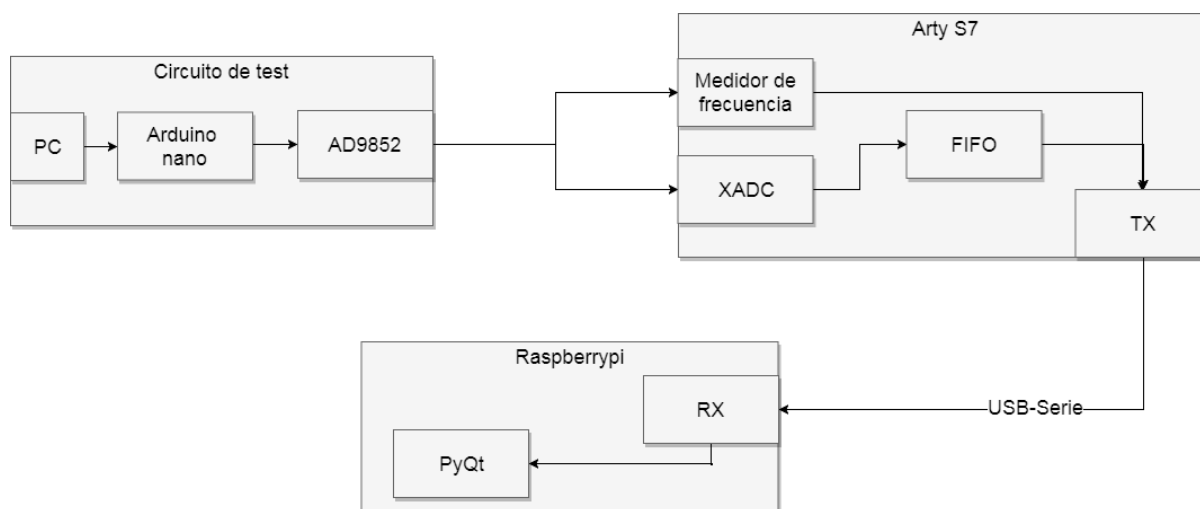


Figura 2.1 Diagrama de bloques del funcionamiento del sistema

La parte encuadrada dentro de “Circuito de test” está compuesta por un PC, una placa de evaluación Arduino mini y un circuito impreso AD9852. La función del Circuito de test es generar una señal senoidal con control sobre la frecuencia para, mediante el PC, probar el sistema de adquisición de datos objeto de este trabajo.

Por otro lado, encontramos una placa de evaluación Arty S7 cuyo núcleo es una FPGA modelo Spartan 7 cuyas funciones son adquirir las muestras de la señal generada, ya sea por el circuito de test o cualquier otra señal, y, además, medir la frecuencia de esta. La medición de frecuencia y el muestreo se realizan de forma independiente. La máxima frecuencia de muestreo es de 1 MHz[5], lo que, teóricamente, limita las señales a 500 kHz debido al teorema de Nyquist. La función del medidor de frecuencia es precisamente medir a una mayor velocidad la frecuencia de la señal que está muestreando el ADC, que, tal y como se comenta en la introducción, sirve para reconstruir la señal mediante diezmado e interpolación.

Una vez recibidos los datos por la Raspberry Pi, se toman para imprimir, aquellos que se ajusten en tiempo y amplitud a los parámetros de disparo y tiempo seleccionados por el usuario en la interfaz de usuario. Además, si el usuario lo desea, puede detener y reiniciar el proceso de muestra de datos, así como visualizar el Periodograma de la señal mediante la opción “Transformada de Fourier”.

2.2. OSCILADORES DE SÍNTESIS DIGITAL DIRECTA (DDS)

Un oscilador de síntesis digital directa (DDS) permite obtener señales analógicas mediante la generación de una señal digital que es transformada mediante un conversor analógico-digital,

podemos obtener la señal deseada. Debido a que el control y la configuración se realizan de forma digital, el DDS permite una rápida respuesta a cambios de frecuencia, una gran resolución en frecuencia y un gran ancho de banda. Entre otras ventajas de los DDS podemos encontrar su reducido tamaño y su bajo consumo.[6]

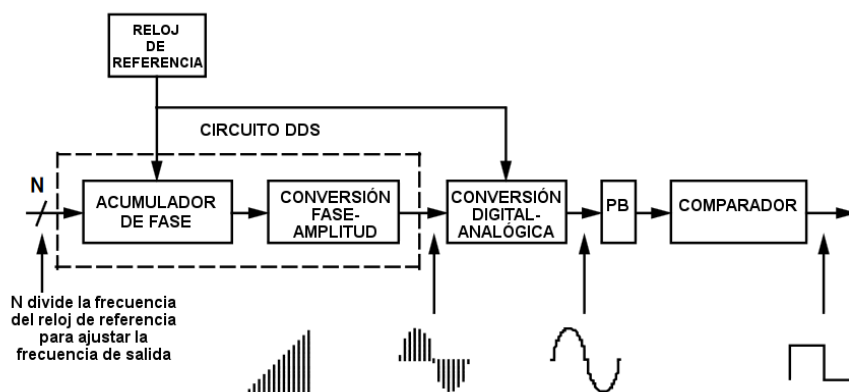


Figura 2.2 Diagrama de bloques del funcionamiento de un DDS completo para generar un tren de pulsos.[6]

El funcionamiento de los DDS se ilustra en la figura 2.2, los elementos principales de un circuito DDS son:

1. El acumulador de fase
2. La conversión de fase-amplitud.

El acumulador de fase se controla con la palabra sintonizadora (N). El funcionamiento del acumulador de fase se basa en actuar como un simple contador de módulo N en formato digital con una resolución fija de 2^n , de forma que se determina cuantos saltos de N muestras por cada ciclo de la señal de frecuencia de referencia ocurren. Es decir, a mayor N , menos tardará el acumulador en llegar a la máxima salida y, por tanto, mayor frecuencia. El proceso descrito genera una señal digital representada en la figura 2.2 mediante un triángulo a la salida del acumulador de fase. La frecuencia de salida F_o viene representada por la expresión:

$$F_o = \frac{N \cdot F_{ref}}{2^n} \quad (2.1)$$

dónde N es la palabra sintonizadora, F_{ref} es la frecuencia de referencia y n es el número de bits del acumulador de fase.

El siguiente elemento que aparece es el conversor fase-amplitud, este genera una señal de salida mediante un mapeo de la salida del acumulador con una tabla en memoria para generar un seno discreto. Dicho seno está representado a la salida del conversor fase-amplitud en la figura 2.2.

Mediante un conversor analógico digital y un filtro paso bajo se obtiene la señal seno deseada, que, en el ejemplo que ilustra la Figura 2.2, se utiliza a la entrada de un comparador para generar una señal cuadrada de la frecuencia deseada.

El DDS utilizado en este proyecto es el AD9850 debido a que se ajusta a los rangos de frecuencia deseados para probar el circuito (hasta 62,5 MHz)[7]. También existe una librería

en Arduino para programarlo de forma sencilla, lo que reduce mucho el tiempo de desarrollo. El AD9850 se programa desde la plataforma Arduino mediante la programación serie que permite dicho AD9850. La frecuencia a sintetizar se controla mediante el PC, el cual lanza una aplicación en Processing.

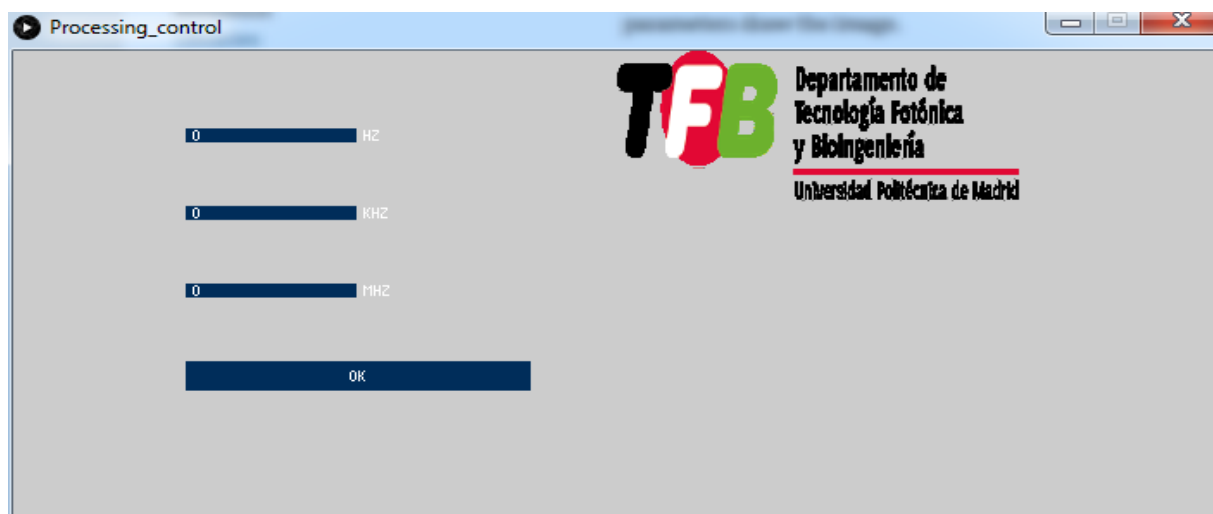


Figura 2.3 Interfaz gráfica creada con Processing para programar DDS desde Arduino

Processing es un lenguaje de programación para generar entornos gráficos que se ha utilizado para crear una interfaz de usuario con la que programar la frecuencia del DDS. La interfaz se comunica a través del puerto serie-USB que incorpora la plataforma Arduino. Cuando se pulsa OK en la interfaz se envía una trama con el formato “XXXHXXKXXM” donde las “XXX” delante de las letras “H”, “K” y “M” representan valores de 000 a 999 que determinan el valor en Hz, KHz y MHz respectivamente. Ésta trama es traducida por el código Arduino que programa, mediante la librería anteriormente citada, el AD9850.

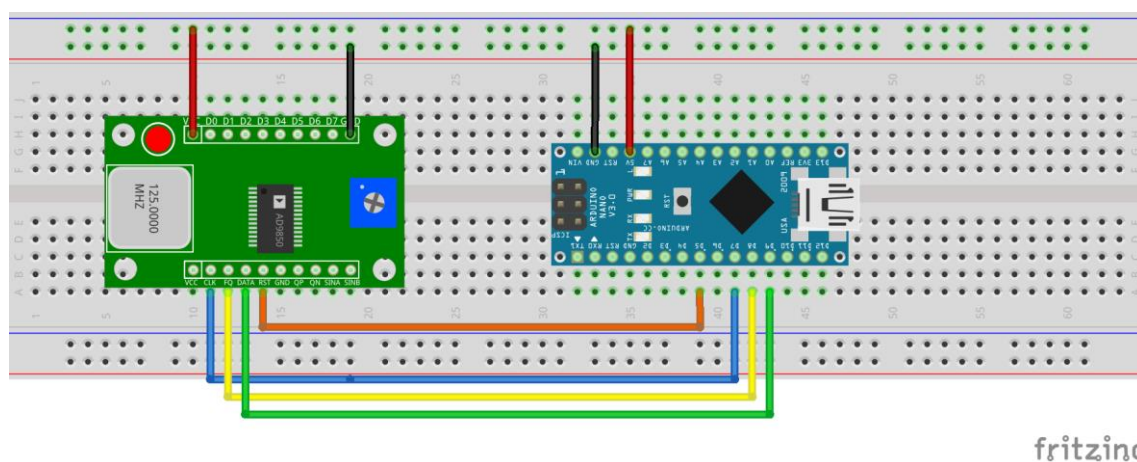


Figura 2.4 Esquema de conexiones entre AD9852 y Arduino

Para programar el AD9852 se utilizan los pines digitales D7, D5, D8 y D9 del Arduino mini, conectados a las entradas CLK, RST, FQ y DATA respectivamente. CLK se utiliza como reloj de la transmisión Data como bus, RST para reiniciar y FQ como señal de actualización de la

salida del DDS a la frecuencia que se ha transmitido. Para alimentar el integrado AD9850 se utiliza el propio Arduino mediante sus pines de 5V y GND.

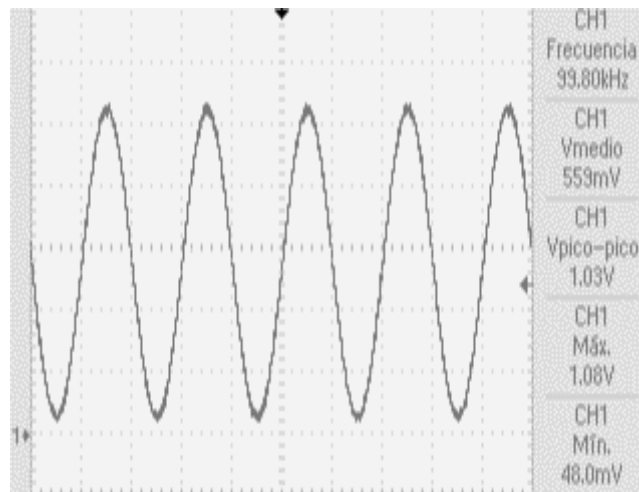


Figura 2.5 Salida sinusoidal de AD9850 configurado a 100 kHz medida con osciloscopio Tektronix TDS 2022B.

Se puede observar en la figura 2.5 una onda sinusoidal de 1V pico a pico con un offset de aproximadamente 500 mV.

Con este circuito se pretende poder realizar pruebas del sistema de adquisición de datos habiéndose diseñado, finalmente, un generador de señales sinusoidales mediante PC, Arduino y un DDS.

2.3.PLACA DE EVALUACIÓN ARTY S7

La placa de evaluación Arty S7 integra una FPGA modelo Spartan-7 de Xilinx. El modelo Spartan-7 ofrece buenas prestaciones en cuanto a tamaño, rendimiento y bajo coste de la familia Spartan. Este último modelo es compatible con la herramienta Vivado de Xilinx. Al integrar la FPGA en una placa de evaluación como Arty S7 se obtienen multitud de opciones de interconexión y de funcionalidad. Por ejemplo, en este trabajo, se utilizan el ADC, el conversor Serie-USB y, eventualmente, leds y conmutadores para configuración y pruebas. Además, la placa de evaluación Arty S7 incluye conectores Pmod para utilizar multitud de bloques funcionales prediseñados de Digilent como extensiones para VGA, ethernet o sensores.[8]

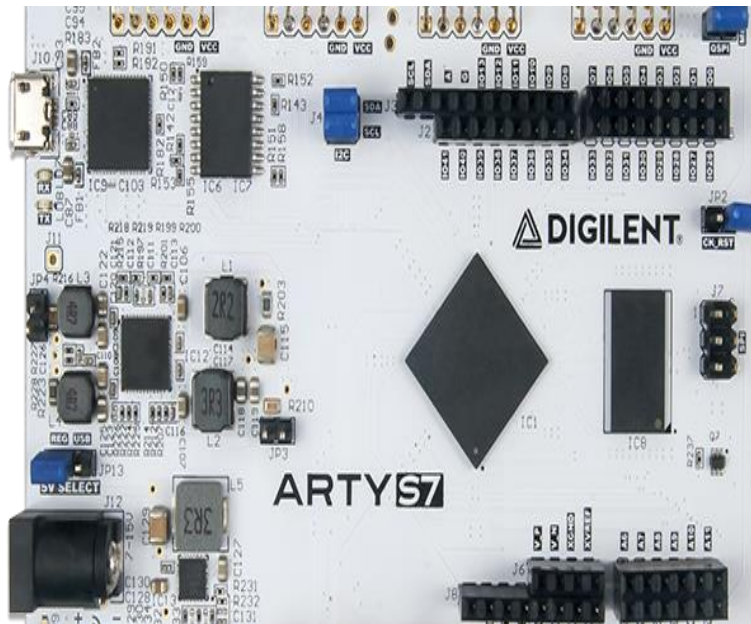


Figura 2.6 Placa de evaluación Artys S7 de Digilent[8]

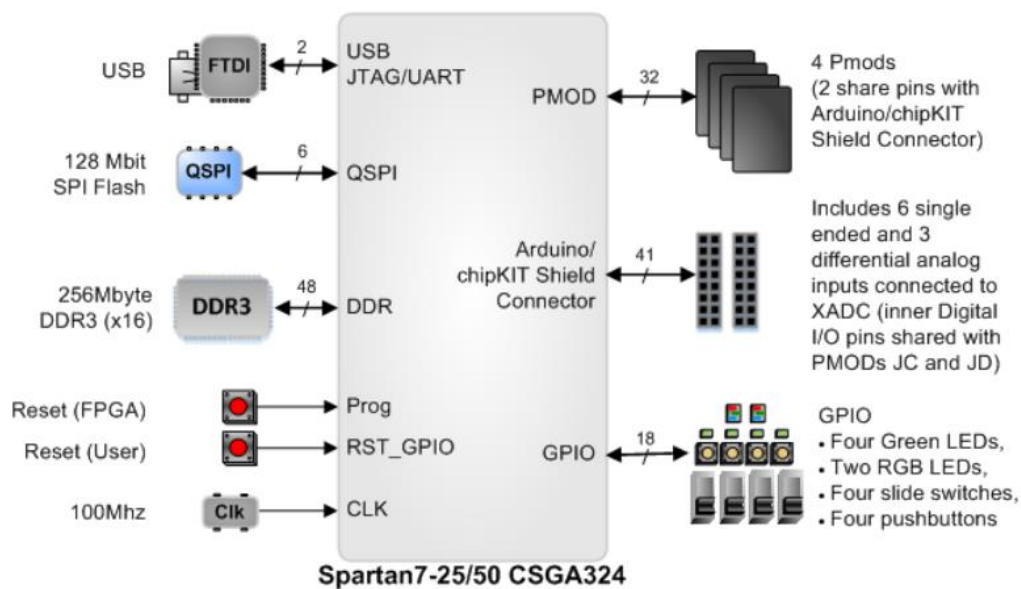


Figura 2.7 Periféricos de Artys S7

A nivel de periféricos y funcionalidades que añade la placa de evaluación sobre la FPGA encontramos:

1. Conversor serie a USB.
2. Flash programable desde SPI de 128 Mbit.
3. RAM DDR3 de 256Mbyte.
4. Botones de reinicio.
5. 4 puertos PMods.
6. Entradas analógicas.
7. Puertos entrada y salida de propósito general (GPIO).

Además, se añade un reloj de 100Mhz y, conectado a las entradas GPIO:

1. 4 leds verdes.
2. Dos leds RGB
3. Cuatro conmutadores
4. Cuatro botones

2.3.1. XADC

El XADC es un bloque que ofrece Xilinx para configurar y utilizar recursos analógicos en sus FPGA de la serie 7, en el caso de este trabajo se utiliza dicho bloque IP para tomar muestras utilizando el ADC que incluye la placa Arty S7. Las posibilidades de configuración son muy extensas. En este apartado se comentarán las distintas opciones seleccionadas dentro del vasto abanico de opciones que ofrece el bloque. Aun así, solo se detallarán las opciones seleccionadas, ofreciendo un análisis temporal y funcional del bloque configurado.

El XADC incluye dos ADC con una resolución de 12 bits y una tasa de muestreo de una megamuestra por segundo.

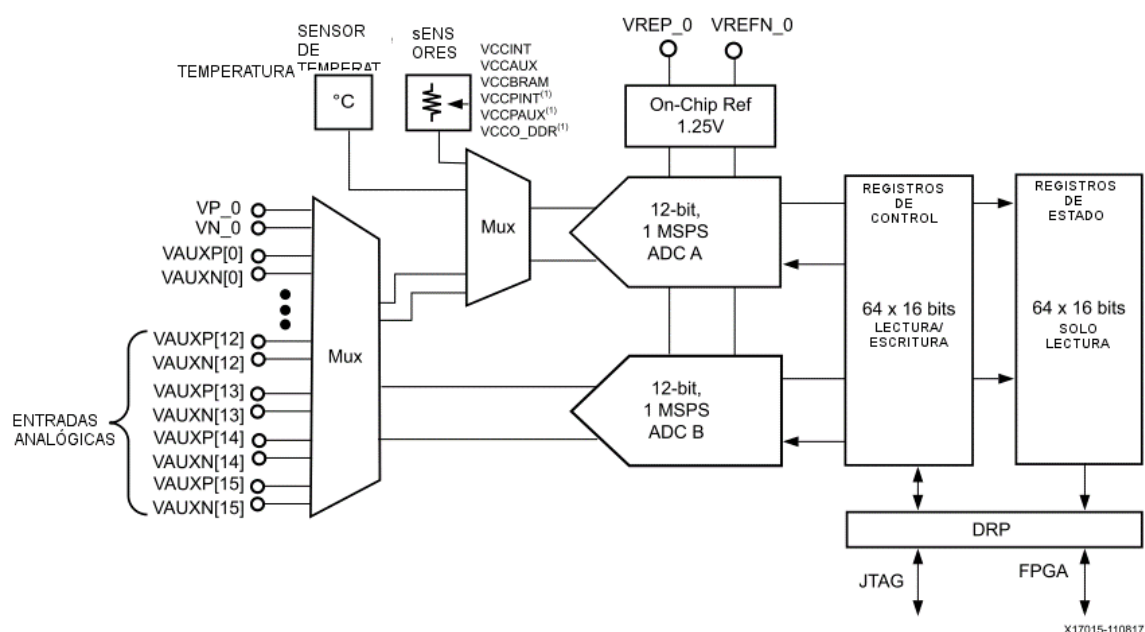


Figura 2.8 Diagrama de bloques del XADC[5]

Como se puede observar en la figura el bloque XADC está compuesta por distintos módulos, en esencia, existen dos ADC que, alimentados con una referencia bipolar de 1.25V, muestrean una señal diferencial según la configuración que toman de los registros de control. La señal muestreada se almacena en los registros de estado. Las entradas del XADC están multiplexadas de forma que se pueden seleccionar muchas entradas, además, se puede configurar un secuenciador para muestrear todos los canales uno a uno. Sin embargo, en este trabajo no se ha planteado ya que reduciría la frecuencia de muestreo. La única entrada que se muestrea en este trabajo es la VAUXP[0] – VAUXN[0], que, en la Figura 2.9 se muestran como AD0_P y AD0_N respectivamente.

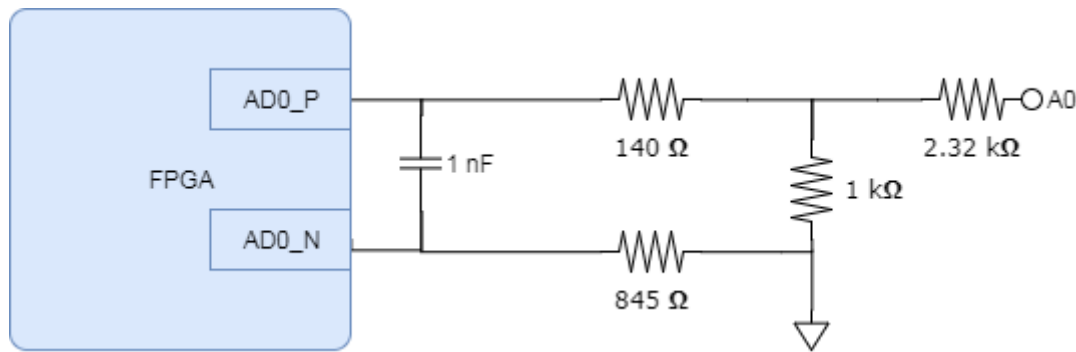


Figura 2.9 Esquema eléctrico de la entrada analógica de la FPGA.[8]

Se puede observar en la Figura 2.9 una red de resistencias y condensador desde el pin de entrada A0, hasta la entrada diferencial ADP0. Las primeras dos resistencias, que tienen valores de $2.32k\Omega$ y $1k\Omega$, forman un divisor de tensión que reduce la señal de $3.3V$ a $1V$ mediante la expresión $\frac{A0}{3.32}$, por otra parte, las otras dos resistencias, junto con el condensador de $1nF$ forman un circuito RC que funciona como filtro. La constante de tiempo del circuito RC es, aproximadamente, de $1\mu s$, calculada como $\tau = RC$, siendo R la suma de las resistencias de 140Ω y 845Ω y C el condensador de $1nF$.

Teniendo en cuenta esta modificación de la señal, la resolución de la muestra será de $\frac{1}{2^{12}} = 244\mu V$. Al transmitir la muestra, se envían solo los 8 bits más significativos. Suponen un desplazamiento de 4 bits hacia la izquierda, lo que, a su vez, suponen una resolución en la recepción de $2^4 \cdot 244\mu V = 3,9\mu V$. Además, hay que tener en cuenta que el acondicionamiento de señal que se muestra en la Figura 2.9 reduce la señal por un factor de 3,32. Teniendo en cuenta todos los factores, el bit menos significativo (LSB) recibido por la Raspberry Pi representa $13mV$.

INSTANCIACIÓN DEL XADC

A nivel de entidad VHDL, el XADC tiene las siguientes entradas y salidas

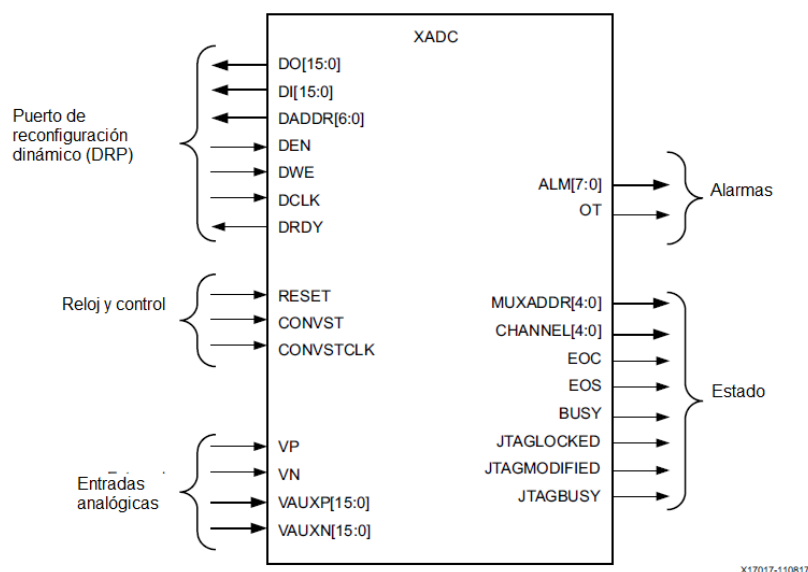


Figura 2.10 Puertos del bloque XADC[5]

Nombre del puerto	Tipo	Función	Uso
DO[15:0]	Salida	Bus de datos DRP	Muestras tomadas
DI[15:0]	Entrada	Datos de entrada para el DRP	Sin uso
DADDR[6:0]	Entrada	Selección de bus para DRP	Para seleccionar canal de entrada del XADC
DEN	Entrada	Habilita el DRP	Conectado al EOC para que cada vez que finalice una conversión, cambie el canal si es necesario.
DWE	Entrada	Habilita la escritura del DRP	Desactivado ya que no se escribe en el DRP
DCLK	Entrada	Entrada de reloj para DRP	Debe ser debidamente configurado para ofrecer una megamuestra por segundo.
DRDY	Salida	Señal de dato disponible para DRP	Si se usan más canales, se utilizaría para sincronizar los cambios de estos.
RESET	Entrada	Reinicio del XADC	
CONVST	Entrada	Inicio de conversión	No se utiliza ya que se muestrea de forma continua.
CONVSTCLK	Entrada	Reloj para el inicio de conversión	No se utiliza
VP	Entrada	Entrada diferencial dedicada	No se utiliza
VN	Entrada	Entrada diferencial dedicada	No se utiliza
VAUXP[15:0]	Entradas	Pares de entradas diferenciales	Se utiliza la entrada diferencial VAUXP[0]
VAUXN[15:0]	Entradas	Pares de entradas diferenciales	VAUXN[0] conectada a tierra por defecto en Arty S7
ALM[7:0]	Salidas	Distintas alarmas de sensores	No se utilizan
OT	Salida	Alarma por sobre temperatura	No se utiliza
MUXADDR[4:0]	Salidas	Informa del siguiente canal a muestrear	No se utiliza ya que no se configura el modo de multiplexor externo
CHANNEL[4:0]	Salidas	Informa del canal seleccionado una vez se ha terminado de muestrear	No se utiliza

Nombre del puerto	Tipo	Función	Uso
EOC	Salida	Informa del fin de conversión	Salida conectada al reloj de escritura de la FIFO.
EOS	Salida	Informa del fin de una secuencia de conversiones	No se utiliza.
BUSY	Salida	Informa sobre el estado de conversión XADC	No se utiliza
JTAGLOCKED	Salida	Indica bloqueo del JTAG	No se utiliza
JTAGMODIFIED	Salida	Indica que se ha escrito desde el JTAG	No se utiliza
JTAGBUSY	Salida	Indica que se está escribiendo desde el JTAG	No se utiliza

Se puede observar que la mayoría de las señales que ofrece el XADC no son tomadas en cuenta en este trabajo. Esto es debido a que se utiliza la más simple de las configuraciones. Modo continuo y solo un canal. Aun así, con vistas a futuras modificaciones se dejan indicadas las distintas posibilidades. Para más información sobre los distintos modos que ofrece este bloque, acceder al documento [5].

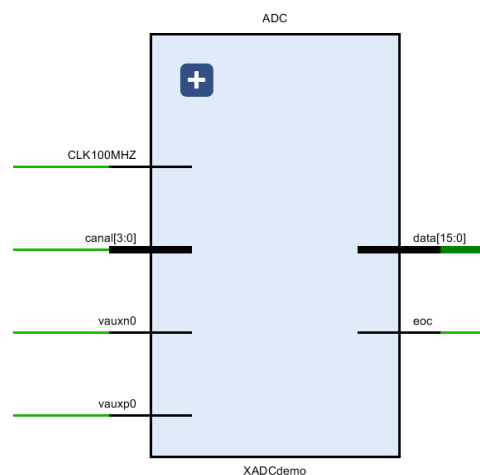


Figura 2.11 Entidad que contiene al XADC y se comunica con la FIFO.

Finalmente, para el objetivo de este trabajo, solo necesitamos la señal de Reloj, las dos entradas que se encaminan hacia las entradas del ADC que incorpora la placa Arty S7. Intencionadamente, se permite seleccionar el canal en el bloque diseñado. Pero para realizar el cambio de canal sería necesario encaminar las entradas diferenciales hacia los pines dedicados a las entradas. Estos pines son modificables desde el archivo .xdc que proporciona Xilinx junto con la placa. En el caso de este trabajo está fijo a la entrada A0. De la salida data[15:0] se

seleccionan los 8 bits más significativos y se encaminan a la FIFO. Como se ha comentado anteriormente, la señal eoc se conecta al reloj de escritura de la FIFO.

2.3.2. FIFO

Una memoria en la que se almacenan los datos de forma que el primero que se escribe es el primero en ser leído se denomina como una cola FIFO.

Tal y como se puede observar en la Figura 2.1, las muestras tomadas por el XADC son almacenadas en una cola que será la misma que sirva para surtir al transmisor de datos. Dicha cola está implementada utilizando un IPcore de Xilinx incluido en la herramienta Vivado 2017.2. La FIFO está implementada utilizando los bloques RAM de la FPGA.

Respecto a su configuración, se han establecidos señales de estado que advierten del estado de la cola, ya sea debido a que se ha completado o bien que se ha quedado vacía.

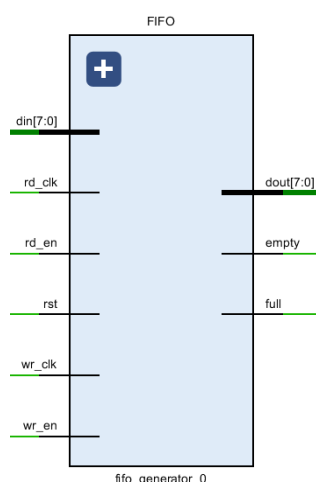


Figura 2.12 Entidad de la FIFO generada por el IPcore de Xilinx

Puerto	Tipo	Función	Uso
din[7:0]	Entrada	Bus de datos para escritura	Entrada conectada a los 8 bits más significativos del XADC
rd_clk	Entrada	Señal de reloj para lectura	Controla la lectura de los datos de la cola, se conecta a la señal de fin de transmisión del bloque TX.
rd_en	Entrada	Habilitar lectura	Permite la lectura de datos de la cola.
Rst	Entrada	Reinicia la cola	Reinicia la cola estableciendo las salidas dout y full a nivel bajo y, empty, a nivel alto.
wr_clk	Entrada	Señal de reloj para la escritura.	Controla la escritura de los datos de la cola, se conecta a la señal de fin de conversión (EOC).
wr_en	Entrada	Habilitar escritura.	Permite la escritura de los datos en la cola.
dout[7:0]	Salida	Bus de datos de lectura.	Salida conectada a la entrada del bloque TX.

Puerto	Tipo	Función	Uso
empty	Salida	Señal de “cola vacía”.	Tanto “empty” como “full” se utilizan para sincronizar los ciclos de lectura y escritura de forma que primero se muestree y luego se transmita mediante el bloque TX.
full	Salida	Señal de “cola llena”	

Tabla 2.1 Puertos de la entidad FIFO generada mediante el IPCore de Xilinx

La cola se controla mediante una máquina de estados síncrona cuyo comportamiento corresponde al siguiente diagrama:

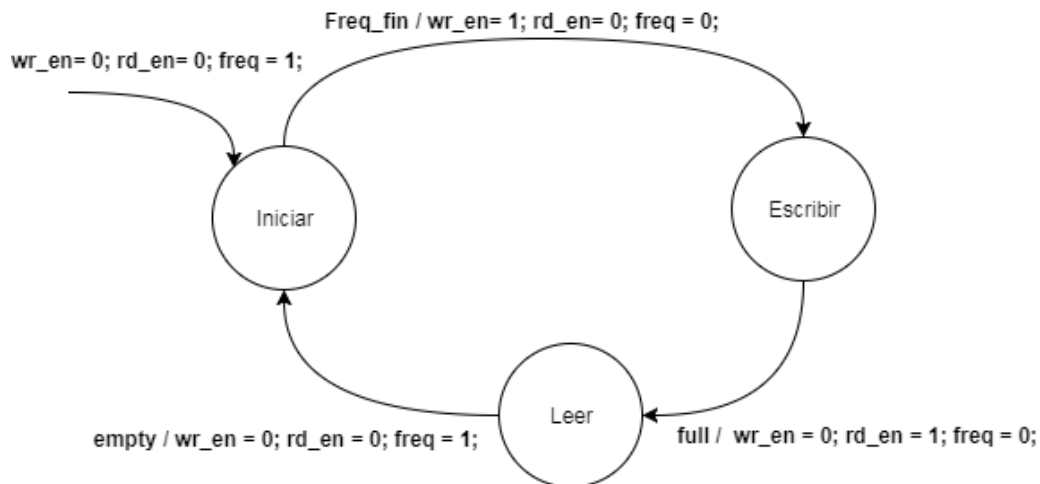


Figura 2.13 Diagrama de la máquina de estados finito que controla la FIFO.

Parte de las señales que controlan la máquina de estados finita (FSM) se corresponden con las expuestas en la Tabla 2.1. “Freq_fin” y “freq”, en cambio, se utilizan para sincronizar la FIFO con el envío de la frecuencia medida, cuyo proceso se detalla en el siguiente apartado de este capítulo. De igual forma, el estado “Iniciar” se utiliza a modo de espera que corresponde con dicha sincronización.

Como se puede apreciar en la Figura 2.13, una vez que se activa “Freq_fin” se escriben en la FIFO las señales muestreada una a una hasta que se llena, una vez llena la FIFO, se transmite mediante el bloque TX hasta que queda vacía, pasando entonces al estado “Iniciar” que espera otra activación de “Freq_fin”.

2.3.3. MEDIDOR DE FRECUENCIA

Para poder reconstruir señales periódicas a través del diezmado e interpolación, es necesario conocer la frecuencia de la señal en cuestión. Para ello, necesitamos que la FPGA mida la frecuencia de la señal y la transmita junto a la señal muestreada.

El método por el cual se mide la frecuencia de la señal es el de registrar el número de ciclos de un reloj “rápido” que proporciona la FPGA que se producen en un ciclo de la señal externa. La señal externa, para este medidor, se conecta a una entrada digital de la FPGA. Esto último implica que el nivel de la señal se ha de ajustar a los niveles lógicos y de frecuencia máxima de los pines de la FPGA. En la FPGA se pueden configurar los distintos niveles lógicos según se requiera. Para consultar los distintos valores se puede recurrir al siguiente documento[9]. Para este trabajo se ha seleccionado el LVCMOS33 que responde a los rangos máximos de [-3,0.8] V para detección de nivel bajo y [2, 3.45] V para la detección de nivel alto[9].

Existe un problema relacionado con la selección de voltajes en los bancos de entrada-salida de la FPGA que integra la placa de evaluación Arty S7. Todos los bancos disponibles para entrada-salida, es decir, los encaminados a pines de salida, se encuentran a un nivel ya fijado para el correcto funcionamiento de la placa, por lo que, a priori, no es posible seleccionar otros niveles de voltaje para implementar el frecuencímetro.

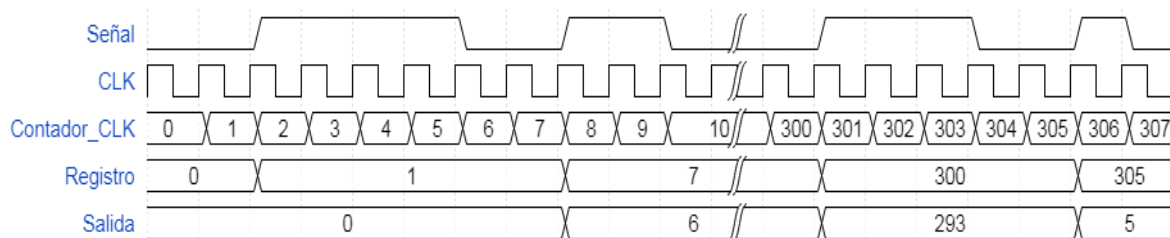


Figura 2.14 Cronograma de las señales principales que controlan el medidor de frecuencia.

Como se puede apreciar en la Figura 2.14, lo que está implementado es un medidor de pulsos. Al detectarse un flanco de subida en “Señal”, se registra el valor del contador de “CLK”, para, una vez que detecta el siguiente flanco de subida, establecer en la salida la diferencia entre el primer instante y el segundo del “Registro”. El cálculo de la frecuencia de la señal a partir del número de pulsos se realiza en la Raspberry Pi a través de la siguiente relación.

$$f_{señal} = \frac{f_{CLK}}{n^{\circ} \text{ pulsos}}$$

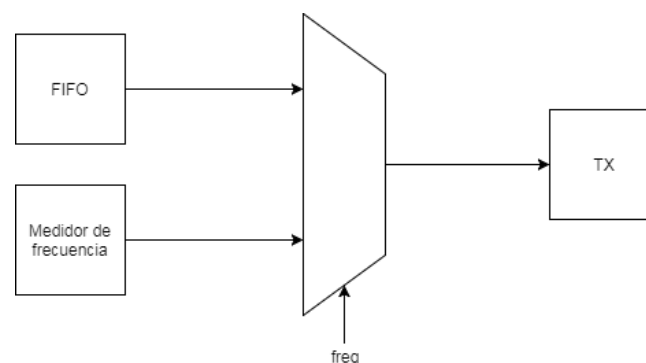


Figura 2.15 Selección de entradas para el bloque transmisor.

A nivel de entidad de VHDL, la salida del medidor de frecuencia es de 48 bits, mientras que la transmisión se realiza con una longitud de 8 bits. Para solventar este problema se ha optado por implementar un contador que depende de una señal del bloque transmisor que indica el fin de transmisión (EOT). Cada activación de EOT aumenta el índice de un array en el que se almacenan de 8 bits en 8 bits el número de pulsos medidos. Además, en dicho array también se almacenan dos caracteres especiales para que la Raspberry Pi pueda extraer el número de pulsos dentro de entre los datos de la señal muestreada con el XADC.

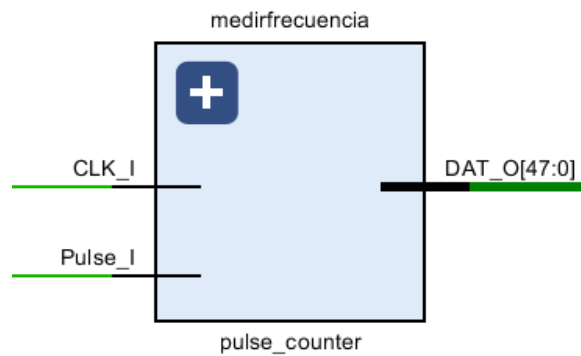


Figura 2.16 Bloque de la entidad generada para medir frecuencias.

Para poder transmitir el número de ciclos de reloj medidos en un pulso de señal, es necesario seleccionar la entrada al bloque transmisor TX para que escoja entre la salida de la FIFO y la salida del bloque medidor de frecuencia. Dicha selección se controla mediante la señal “freq” que aparece en la máquina de estados de la Figura 2.13. Esta señal es activada por la máquina de estados que controla la FIFO cada vez que esta se vacía.

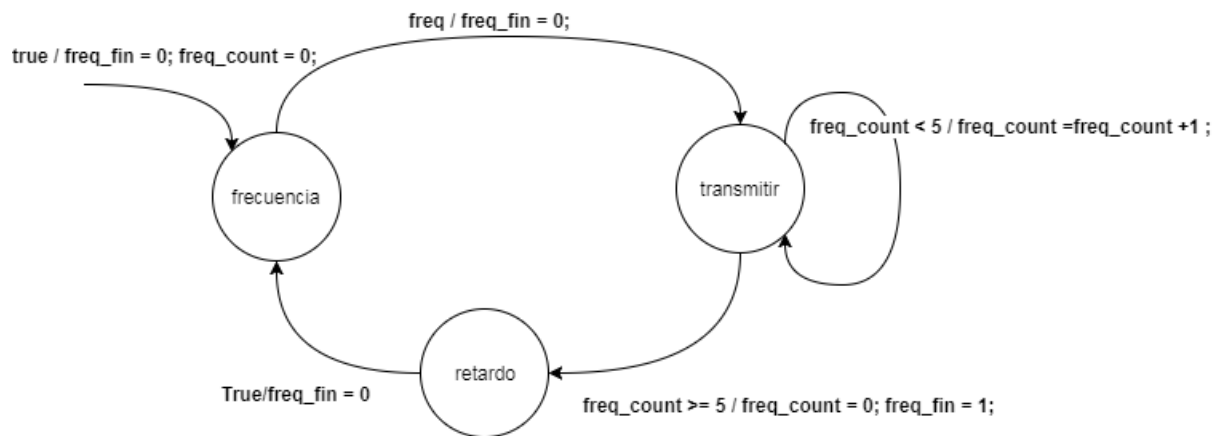


Figura 2.17 Diagrama de estados de la FSM que controla el medidor de frecuencias.

2.3.4. TRANSMISIÓN SERIE MEDIANTE USB

El transmisor-receptor asíncrono universal (UART) es un dispositivo para la comunicación mediante el puerto serie. Está formado por un convertor paralelo serie de los datos de entrada al puerto serie y, en la recepción, de un convertor serie paralelo, además, contiene una memoria para almacenar los datos que aún no han sido procesados. Además de estos procesos de conversión, la UART contiene numerosas señales que se utilizan para conocer el estado de la

transmisión que permiten controlar las entradas y las salidas en cada momento para su correcta transmisión-recepción.

La forma de onda de la transmisión de un dato en la línea de transmisión serie tiene el siguiente formato:

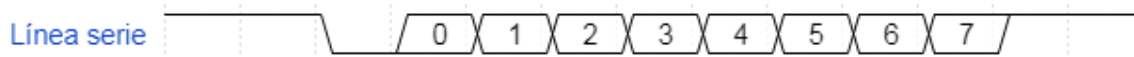


Figura 2.18 Forma de onda de una transmisión serie de 8 bits con un bit de parada.

Cuando no se está transmitiendo, la línea se mantiene a nivel alto. Para comenzar la transmisión, se establece la línea a nivel bajo durante un tiempo de bit, este bit se conoce como bit de comienzo. Una vez finaliza el bit de comienzo, se transmiten uno a uno por la línea serie los bits que son convertidos de paralelo a serie por la UART. Tras la transmisión, se envía un bit de parada, que consiste en establecer la línea a nivel alto durante un tiempo de bit. En este trabajo se ha configurado la transmisión serie con un solo bit de parada. De igual forma, tampoco se contemplan control de flujo.

La placa de evaluación Arty S7 incluye un circuito integrado FT2232HQ que actúa como convertidor USB-UART. Gracias a este circuito integrado, podemos conectar la placa de evaluación directamente mediante un cable micro USB a la Raspberry Pi para, mediante comunicación serie, transmitir los datos desde la FPGA a la Raspberry Pi. Además, esto permite alimentar la placa de evaluación sin un adaptador de corriente dedicada, es alimentada por la Raspberry a través del puerto USB.

El circuito integrado FT2232 también sirve para programar la FPGA desde el PC mediante JTAG. Según la guía de referencia de la placa de evaluación, estas dos funciones son totalmente independientes y no hay que prestar especial atención a la hora de usar el convertidor. [8]

Para comunicarse con la FPGA, se utilizan dos hilos TXD y RXD tal y como se muestra en la Figura 2.19. Además de estos dos hilos, tal y como se ha comentado anteriormente, existe el conector JTAG que sirve tanto para programar la FPGA desde el PC, como para programar la memoria estática para que la FPGA arranque por defecto con el programa desarrollado.

En este trabajo la Raspberry no necesita transmitir información hacia la FPGA, pero, en el capítulo de líneas futuras, sí que se mencionan implementaciones que necesitarían de esta posibilidad.

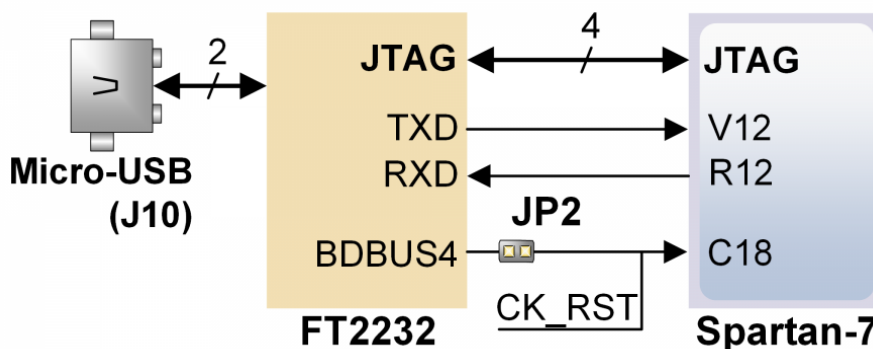


Figura 2.19 Convertidor USB-UART.[8]

La placa Arty S7 incluye dos leds que informan del estado de la UART, V12 y R12 en Figura 2.19. En la placa, están serigrafiados como TX y RX. Se puede apreciar como es justamente al contrario a la Figura 2.19 ya que están señalizadas desde el punto de vista del PC.[8]

El CK_RST y el JP2 están diseñados para simular el comportamiento de Arduino y las placas chipKit cuando son programadas generando un reinicio compatible con el microprocesador software para FPGAs “Microblaze”. Según el manual de referencia[8], no es recomendable activar el conector si no se va a utilizar Arduino. Como no se va a utilizar en este trabajo, el conector está desactivado.

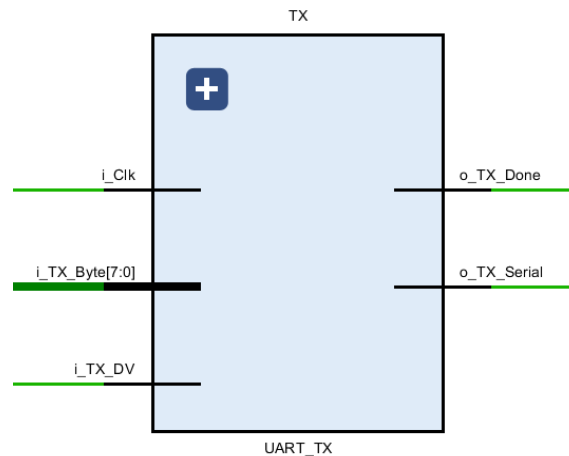


Figura 2.20 Entidad del bloque transmisor en la FPGA.

En la Figura 2.20 se pueden observar las señales de entrada y salida del bloque transmisor. La señal que gobierna la transmisión es *i_TX_DV*, se activa bien con la señal “vacía” o con la señal “freq” reflejadas en Figura 2.13. Con dicha señal habilitada, se carga el dato *i_TX_Byte* en el siguiente flanco de subida de *i_CLK*. Internamente, se realiza la correcta división de la frecuencia de reloj para respetar la tasa de baudios de la transmisión serie. Dicha división es ajustable mediante un genérico de VHDL. Una vez que los 8 bits de entrada han sido serializados y transmitidos, se activa la señal *o_TX_Done* que, según se esté transmitiendo el número de pulsos o la salida de la fifo, actualizará el valor de la entrada del propio transmisor respectivamente.

2.4.RASPBERRY PI

Raspberry Pi es un sistema en chip (SOC) desarrollado por la fundación Raspberry Pi con la intención de promover la enseñanza. Algunas de sus bondades son el bajo coste, su relativamente elevada capacidad de procesamiento tanto en su unidad central (CPU) como en su unidad gráfica (GPU). Debido a estas razones su uso ha excedido al objetivo inicial y es usado en numerosos usos profesionales.

A continuación, se enumeran los recursos más importantes que se utilizan en este trabajo

Tipo de recurso	Recurso
CPU + GPU	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Memoria dinámica	1GB RAM
Memoria estática	Puerto micoSD
Salidas de video	HDMI y video compuesto
Puertos USB	4

Raspberry Pi 3 modelo B está dotada de puertos de comunicaciones SPI, miniUart, I2C así como puertos entrada/salida de propósito general (GPIO) que extienden la conectividad de la Raspberry Pi para ser usada junto a sensores y otros sistemas embebidos.

Como recursos de conectividad incorpora Bluetooth, Ethernet y WiFi.



Figura 2.21 Raspberry Pi 3 modelo B.

Además, para operaciones con grandes volúmenes de datos, cuenta con una GPU, que, en este trabajo no se utilizará. Sí que se realizará un análisis de las posibilidades para una futura mejora del proceso de señal que se recibe a través de la transmisión serie.

Como se comenta en la introducción de este capítulo, la placa Raspberry Pi, se encarga de recibir, procesar y mostrar los datos enviados a través de la FPGA. En este apartado se van a explicar las partes más importantes en la que se dividen estas tareas, así como comentarios acerca de las opciones y decisiones finales para cada uno de los cometidos.

Todo el código que se ha utilizado en la Raspberry Pi se ha desarrollado en Python. Las distintas librerías que se utilizan para llevar a cabo las distintas funciones se detallan a continuación:

Función	Librería	Descripción
Recepción de señal	Serial	Recepción de datos a través del puerto serie USB
Manipulación de datos	NumPy	Manipulación de datos mediante listas numéricas, búsqueda, indexado.
Mostrar los datos	Matplotlib	Muestra los datos de forma similar a la herramienta Matlab
Interfaz de usuario	PyQt	Permite encapsular todas las funciones anteriores y añadir interacción con el usuario mediante Qt.
Reconstrucción de señales y medida de frecuencias	Scipy	Realizar la DFT, interpolación y diezmado (remuestreo) de la señal para, posteriormente, ajustar en frecuencia.

Tabla 2.2 Librerías utilizadas para llevar a cabo la recepción el procesado y la muestra de la señal.

2.4.1. SISTEMA OPERATIVO RASPBIAN

Raspbian es el nombre de la versión adaptada para Raspberry Pi de la distribución de Linux Debian. Debian es una de las distribuciones de Linux más longevas, por lo que ofrece

estabilidad, gran compatibilidad y un rendimiento óptimo incluso en sistemas limitados. Por todo esto se apetece como una base atractiva para utilizar en Raspberry Pi. Como se ha comentado en la introducción, la memoria estática depende de una tarjeta flash microSD. Por lo que el sistema operativo adaptado ha sido mermado para utilizar el menor espacio en memoria. Entre las aplicaciones que incluye se encuentran: navegador web, programación en Python, software Mathematica, interfaz gráfica para usar la Raspberry Pi. Además, se incluye un entorno gráfico de escritorio conocido como “Lightweight X11”[10].

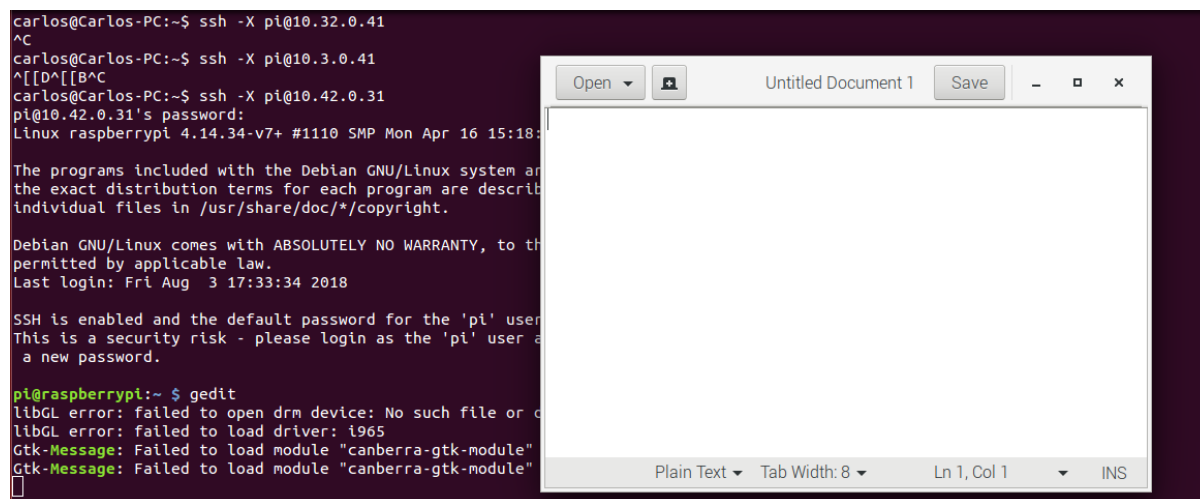


Figura 2.22 Ejemplo de ejecución de Gedit en RPi desde SSH en modo gráfico.

Con dicho entorno se puede lanzar el programa desde SSH, lo que hace posible ejecutar la aplicación desde otro lugar siempre que la Raspberry Pi esté conectada a la misma red que un PC utilizando la opción “-X” al conectar mediante SSH a la Raspberry Pi. No solo es posible lanzar la aplicación objeto de este trabajo, es posible lanzar cualquier aplicación que genere una ventana en el entorno. Lo que permite trabajar sin monitor ni teclado ni ratón sobre la Raspberry Pi. En la Figura 2.22 se puede observar un documento abierto desde Ubuntu mediante SSH con la opción -X para poder escribir documentos en la Raspberry Pi sin necesidad de conectar dispositivos a la misma.

El Sistema de archivos de Raspbian se organiza de la siguiente manera:

Directorio	Descripción
/	El directorio principal que contiene todos los directorios y el sistema de archivos
/bin	Contiene los ejecutables del sistema operativo
/boot	Contiene los archivos necesarios para iniciar el sistema operativo
/dev	Contiene los ficheros relacionados con los dispositivos conectados al sistema
/etc	Contiene los archivos de configuración del sistema así como los ejecutables para el arranque del sistema
/home	Directorio de usuario

Directorio	Descripción
/lib	Contiene las librerías compartidas del sistema
/opt	Contiene los paquetes instalados
/proc	Directorio virtual que contiene información sobre el kernel y los procesos del sistema
/root	Directorio de superusuario
/sbin	Contiene ejecutables utilizados en el proceso de arranque o rescate del sistema
/sys	Contiene archivos del sistema
/tmp	Contiene los archivos temporales
/usr	Contiene los ficheros relacionados con los programas instalados por el usuario
/var	Contiene ficheros que cambian de tamaño durante la ejecución del sistema

Tabla 2.3 Estructura del sistema de ficheros de Raspbian[11]

La recepción de datos a través del puerto serie que se genera en forma de archivo en la carpeta /dev dentro del sistema de archivos de Raspbian.

2.4.2. RECEPCIÓN DE DATOS

LIBRERÍA PYSERIAL

La librería PySerial ofrece soporte para utilizar comunicaciones serie desde Python ejecutado desde distintas plataformas. Al instanciar la clase principal “serial”, es necesario configurar la transmisión serie. Los parámetros a configurar son:[12]

1. Puerto: La dirección del archive que representa el puerto serie.
2. Tasa de baudios: debe coincidir con la tasa configurada en la FPGA
3. Tamaño de byte– número de bits por símbolo sus posibles valores son: FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS.
4. Paridad– Habilita o deshabilita el chequeo de paridad, sus posibles valores son: PARITY_NONE, PARITY_EVEN, PARITY_ODD PARITY_MARK, PARITY_SPACE.
5. Bits de parada–Número de bits de para de tras la transmisión de un símbolo, sus posibles valores son: STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO.
6. Tiempo de espera para lectura(float): establece un tiempo de espera para que la lectura no sea bloqueante.
7. Xonxoff: Habilita el control de flujo software.

8. Rtscts: Habilita el control de flujo hardware (RTS/CTS).
9. Dsrdrtr: Habilita el control de flujo hardware e (DSR/DTR).
10. Tiempo de espera para la escritura: establece un tiempo de espera para que la escritura no sea bloqueante.
11. Tiempo de espera entre caracteres: Deshabilitado por defecto, establece el tiempo entre caracteres.

En este trabajo se utiliza una configuración de 8 bits por símbolo, con un bit de parada, ningún control de flujo, sin paridad y sin tiempos de espera, esto quiere decir que las lecturas son bloqueantes. Aun así, como se está transmitiendo continuamente información no se generarán cuelgues en el programa.

La tasa de baudios es variable y debe ser ajustada en los dispositivos. Se asume durante el desarrollo del trabajo que la tasa de baudios es de 115200.

PROCESO DE RECEPCIÓN SERIE, TRAMAS Y DECODIFICACIÓN

Dentro del programa principal, existe una función llamada “serialanddecode (trigger, muestras):” que devuelve una lista de tamaño “muestras”, un valor de frecuencia y un valor que indica si las muestras son válidas. Cuando las muestras no son válidas, indican que en el número de muestras que se han recogido, no existe ningún valor que haya superado el valor de disparo (trigger). Siempre se leen el doble de “muestras” desde el puerto serie y se devuelven el número de muestras indicado a partir de que se detecte un valor igual o superior al disparo. Si después de encontrar dicho valor, el número de muestras restantes es inferior al indicado, o, no se ha hallado dicho valor, se devuelve un -1 en el valor que informa sobre la validez de las muestras. En este caso, no se actualizaría la interfaz de usuario.

FF	F7	00	A5	00	01	02
----	----	----	----	----	----	----

Figura 2.23 Trama inicial

La trama completa enviada por la FPGA es variable, pero, siempre comienza por ‘0xFF7F112233’ donde 112233 indican el número de pulsos medidos en la FPGA en base hexadecimal. El resto de la trama es decodificado como se indica anteriormente.

La decisión de no medir la frecuencia en cada lectura se ha tomado en aras de potenciar la fluidez del programa.

2.4.3. INTERFAZ GRÁFICA EN PYQT

Qt es una herramienta habitual para el diseño de interfaces gráficas de usuario (GUI) en sistemas empujados. Actualmente, no es necesaria una versión específica para distintas plataformas[13]. Además, es posible trasladar código entre plataformas de forma que lo convierten en una herramienta versátil a la hora de crear GUIs. En lo que a este trabajo se refiere, se va a utilizar su variante para Python.

La GUI desarrollada está basada en señales generadas al interactuar con los distintos botones disponibles en la ventana. Al generarse dichas señales, funciones son lanzadas para modificar el funcionamiento periódico de la GUI. El funcionamiento periódico de la GUI consiste en un

temporizador que genera una señal (de igual forma que los botones), pero, sin que el usuario necesite activarlo cada vez. La señal que genera el temporizador realiza una lectura del puerto serie, tal y como se describe en la sección anterior, mediante la librería PySerial.

A continuación, se van a presentar los distintos modos de funcionamiento, así como los distintos botones y los comportamientos que estos generan en la GUI.

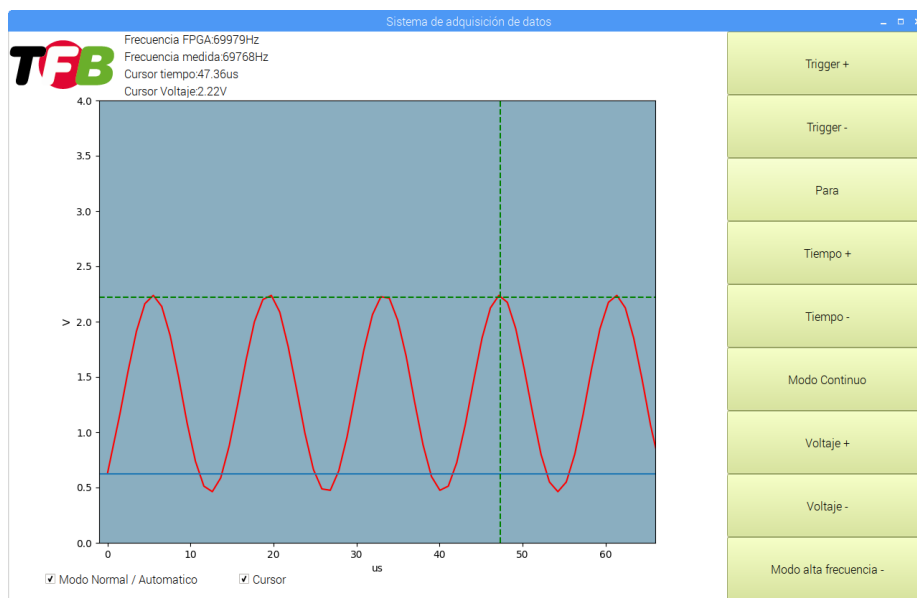


Figura 2.24 GUI creada con PyQt

Como se puede apreciar, se disponen de 9 botones, 6 de ellos son pares que manejan la escala de tiempos, la escala de voltaje y, el nivel de disparo. Además, se puede observar en la parte superior de la Figura 2.24 la frecuencia recibida desde la FPGA, la frecuencia medida en el último análisis espectral y los valores de los cursores. En el ejemplo, podemos observar una señal de 70 kHz de unos 1.75 Vpp aproximadamente. Como se ha comentado anteriormente, estos botones activan señales conectadas a funciones. Los botones “Disparo”, “Tiempo” y “Voltaje” modifican variables globales que controlan distintos aspectos de la recepción, así como de la representación de los datos.

Según el estado de la casilla “Modo normal/automatico” los requisitos de disparo resultan distintos. Estando desmarcada, es condición suficiente que el nivel de la señal sea mayor que el establecido por la línea trigger (color azul). Si está marcada, en cambio, se necesita un paso por la línea de forma ascendente consiguiendo sincronizar la señal.

Para conseguir detectar flanco se realiza el siguiente control:

1. Al detectar una muestra de mayor nivel que la línea de trigger se analiza que las dos muestras anteriores sean menores y las dos consecutivas mayores en nivel.
2. Una vez comprobada la condición 1, se calcula la diferencia de la señal medida con la señal de disparo fija y se interpola la distancia en tiempo entre ambas.
3. Se desplaza el eje X sumando el desplazamiento necesario y se establece en la primera muestra representada el nivel de disparo.

El par de botones “Disparo” controlan el nivel de voltaje a partir del cual se tomarán el número de muestras controlado por el par de botones “Tiempo”. El par “Voltaje” no influye en la recepción serie, solo fija el valor máximo del eje Y.

Botón	Mínimo	Máximo	Por defecto	Paso
Disparo	125 mV	3,125 V	625 mV	62,5 mV
Tiempo	—	0,1 s	1 ms	± 10%
Voltaje	0	4	4	250 mV

Tabla 2.4 Parámetros de disparo, tiempo y voltaje de la GUI.

El botón “Detener” detiene el temporizador que activa la recepción serie de forma periódica, permitiendo detener la imagen. Al pulsar dicho botón el texto cambia a “Comenzar” que reanuda el temporizador. Dicho temporizador solo detiene la recepción de datos, dejando intacto el refresco de la GUI que puede seguir siendo utilizada. Este comportamiento se consigue utilizando dos hilos de ejecución. Uno para la representación de datos y otro para la recepción serie. Los dos hilos comparten las variables de datos, muestras y frecuencia recibida por parte de la FPGA.

El último botón disponible en la GUI es el de “Modo Continuo”, al pulsarse, se puede apreciar el espectro de la señal recibida. Para poder mostrar el espectrómetro, se hace uso de la librería Numpy para realizar una transformada discreta de Fourier a las muestras tomadas. Para que este modo funcione de forma correcta, es necesario un correcto valor de “Tiempo”. En el próximo apartado se explica el proceso por el cual, utilizando las funciones adecuadas de la librería SciPy se consigue la función de densidad de potencia en frecuencia de la señal.

Existe la posibilidad de utilizar cursores en el modo continuo para medir tiempo y voltaje. Se pueden activar marcando la casilla Cursores en la parte inferior izquierda de la GUI. Utilizando las teclas “WSAD” a modo de cruceta se pueden mover las líneas que marcan los niveles de voltaje y tiempo.

2.4.4. USO DE SCIPY PARA CÁLCULO DE TRANSFORMADA RÁPIDA DE FOURIER DISCRETA

Para obtener la estimación de la densidad de potencia espectral de la señal (PSD) es necesario utilizar la función “periodogram” del paquete signal de la librería SciPy para Python. Esta forma de estimar la PSD se basa en la transformada de Fourier de la función de autocorrelación de la señal.

$$F \{ x(t) * x(-t) \} = X(f) \cdot X^*(f) = |X(f)|^2$$

Existen muchas formas de definir la transformada de Fourier en tiempo discreto. En la implementación de SciPy se define de esta manera.

$$A_k = \sum_{m=0}^{n-1} a_m \exp\{-2\pi i \frac{mk}{n}\} \quad k = 0, \dots, n-1$$

La transformada de Fourier en tiempo discreto se define mediante entradas y salidas complejas además de una componente lineal de frecuencia.

$$a_m = e^{2\pi i m \Delta t}$$

El valor que devuelve la llamada es una lista en la que el primer elemento contiene la frecuencia cero, es decir, la media de la señal. El resto de los valores de la lista se ordenan en frecuencia siendo la parte positiva de frecuencia se encuentra desde el segundo elemento hasta n/2. A partir de n/2, hasta el final, se corresponde con los términos negativos de frecuencia. Para ajustar el valor resultante y centrar los valores se hace uso de la llamada a función `fft.freq()` que

devuelve un array con las posiciones naturales de frecuencia. Sabiendo cual es la posición de la muestra máxima que no sea la componente continua (posición 0) obtenemos la frecuencia fundamental de la señal. Para obtener dicho máximo se utiliza la función de *maxarg()* de la librería Numpy que devuelve el primer índice del máximo valor en la lista. Evaluando dicho índice en el eje de las frecuencias, se obtiene la frecuencia fundamental.

Para obtener el periodograma desde la transformada discreta de Fourier se realizan las siguientes modificaciones respecto a el cálculo de la DFT.

$$S\left(\frac{k}{NT}\right) = \left| \sum_N x_N[n] \cdot e^{-i2\pi \frac{kn}{N}} \right|^2$$

2.4.5. AJUSTE DE FRECUENCIA MEDIANTE INTERPOLACIÓN Y DIEZMADO EN PYTHON

Para reconstruir una señal submuestreada, es necesario aplicar técnicas de modificación de la frecuencia de muestreo de la señal adquirida. Estos métodos se conocen como up-sampling y down-sampling. Para aplicarlos se utilizan la interpolación y el diezmado. La interpolación realiza el proceso de up-sampling, mientras que el diezmado realiza el proceso de down-sampling. Es necesario combinar las dos técnicas ya que en la práctica estas técnicas solo admiten números enteros, si queremos realizar una transformación de la frecuencia de muestreo por un número racional, es necesario realizar ambos procesos, lo que se conoce como remuestreo[14].

El proceso de interpolación puede ser llevado a cabo en dos dominios, el de la frecuencia y el del tiempo. La función de la librería SciPy que se usa en el sistema utiliza la interpolación en el dominio de la frecuencia. El algoritmo de interpolación utilizado en el dominio de la frecuencia puede ser consultado en [14]. El proceso de resample se ilustra en la Figura 2.25:

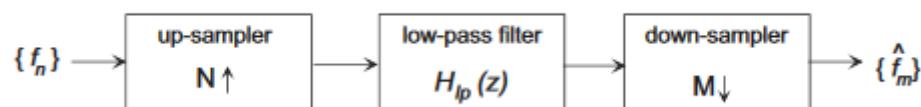


Figura 2.25 Proceso de remuestreo

Disponiendo de la frecuencia medida con un reloj más rápido en la FPGA, es posible, reconstruir la señal original a partir de las muestras medidas a una tasa de muestreo no apta para su representación de forma habitual. Existen ciertas necesidades para que esto sea posible en el sistema:

1. Frecuencia de la señal menor o igual a 50 MHz, que, es la mitad de la frecuencia del reloj con el que trabaja el frecuencímetro en la FPGA.

2. Señal acorde con los niveles de amplitud que se detallan en el apartado del frecuencímetro.

Una vez que se activa en la interfaz de usuario la opción de alta frecuencia se modifica el número de muestras que se obtienen en cada recepción serie, aumentándolo de esta forma hasta la capacidad máxima de la FIFO. De esta forma, nos aseguramos obtener en una transmisión la trama de la frecuencia, así como de obtener una representación amplia de la señal periódica. Al tener un número considerable de muestras de la señal periódica, se aplica un remuestreo de la señal. El valor de dicho remuestreo, tendrá que ver con la relación de la frecuencia de la señal submuestreada y la frecuencia transmitida por la FPGA medida con el frecuencímetro. Como ejemplo práctico se ilustra el siguiente escenario (presente en el capítulo de resultados):

1. Se genera una señal sinusoidal generada de 3 Vpp centrada en 1,5V de 750 kHz de frecuencia.
2. Al obtener la señal, se mide, mediante DFT, la frecuencia de la señal periódica, se obtienen 250kHz debido al submuestreo.
3. Se compara la medición con la frecuencia transmitida por la FPGA y se obtiene la relación directa, se obtiene una relación de 3.
4. Se sobremuestra la señal obtenida utilizando la librería SciPy.
5. Tanto en el modo FFT como en el modo continuo se ajustan los ejes a los nuevos términos de frecuencia y tiempo respectivamente.

En el capítulo 3, se ilustra un ejemplo parecido al anterior, atendiendo de forma más precisa a los resultados e incidencias.

3. RESULTADOS

3.1.INTRODUCCIÓN

En este capítulo se van a mostrar y comentar capturas de los resultados obtenidos. Para una mayor fidelidad y con el objetivo de comparar el sistema desarrollado con otros sistemas comerciales se van a hacer uso tanto para la generación de la señal de test como para medición de señales de:

1. Generador de señales
2. Osciloscopio Textronix TDS 2022B
3. El generador de señales diseñado mediante Arduino PC y Processing.
4. El sistema de adquisición de datos basado en FPGA y Raspberry Pi.

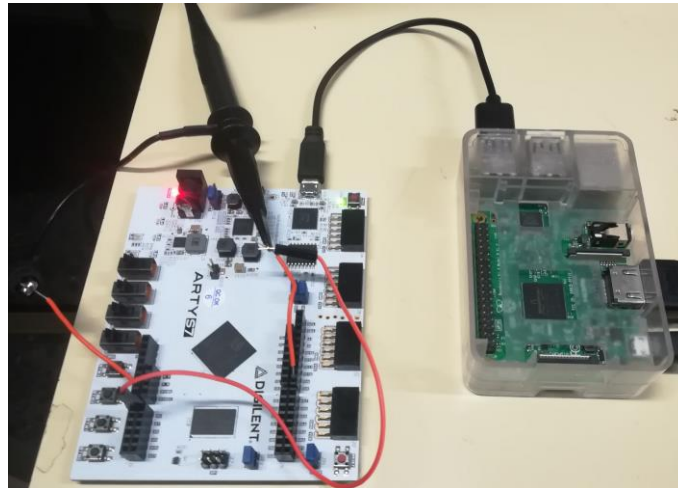


Figura 3.1 Sistema de adquisición de datos.

Como se muestra en la figura Figura 3.1, la señal es conectada a 2 pines de la placa, A0 y D8, correspondientes a la adquisición de señal y medida de la frecuencia. Se conecta a través de USB a la Raspberry Pi, que ya se encuentra preparada para ejecutar el programa escrito en Python.

3.2.SEÑALES PROCEDENTES DE FPGA

Para comenzar se van a analizar las señales que genera la la FPGA al recolectar datos. Se recuerda que la señal de fin de conversión se utiliza como reloj que gobierna la escritura de datos en la FIFO. Para medir la señal de fin de conversión del XADC se ha utilizado uno de los pines de entrada y salida digital disponibles en la placa Arty S7 estos a su vez se conectan con los bancos, también de entrada y salida, de la FPGA, que se configuran según lo descrito en el capítulo 2.

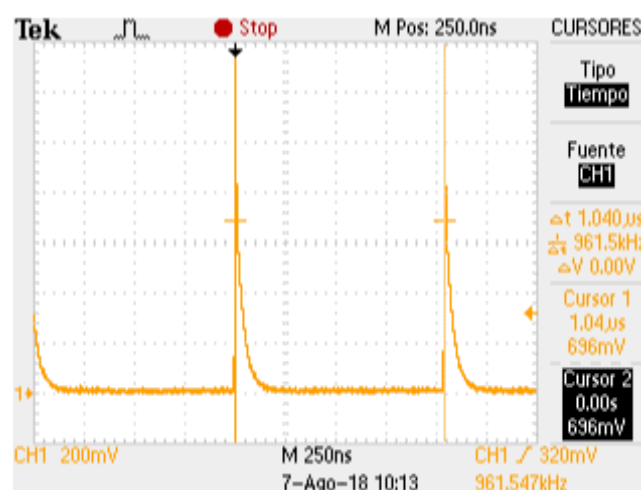


Figura 3.2 Señal fin de conversión del XADC medida con osciloscopio TDS 2022B de Textronix.

Como se puede observar en la figura Figura 3.2, la señal de fin de conversión que genera el XADC tiene un periodo aproximado de 1 microsegundo, lo que equivaldría aproximadamente a 1 MHz. Sin embargo, se puede apreciar que la frecuencia real es de 961 kHz. En el diseño

final, no es posible acceder a dicha señal. Se estima pues, que la distancia entre las muestras es de 1 microsegundo. Posteriormente, se podrá apreciar como esta aproximación no es válida a la hora de submuestrear, pero, como se ajusta automáticamente a la frecuencia medida por la FPGA, no reviste de importancia.

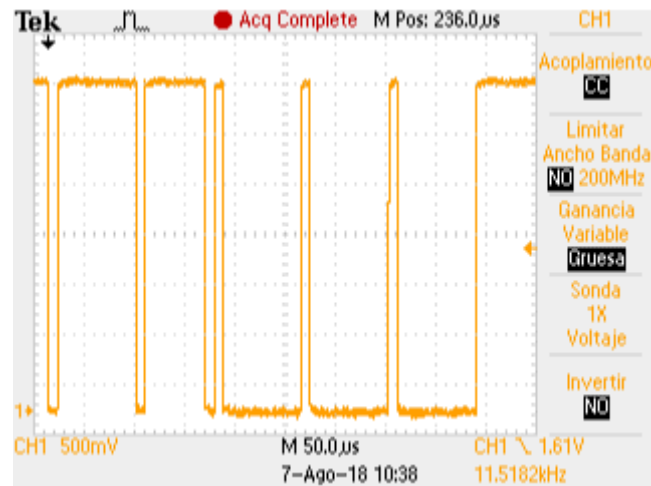


Figura 3.3 Señal de transmisión serie muestreando una señal continua

En la figura Figura 3.3 se muestra la línea serie al comienzo de una transmisión que envía una señal continua muestreada. Tras el bit de inicio (primer flanco de bajada por la izquierda), se puede observar que se transmite 0xFF, es decir, toda la línea a nivel alto. El siguiente byte que se transmite, en cambio, corresponde al valor 0x7F. Los 3 siguientes bytes corresponden al número de pulsos medidos en un periodo de la señal, que, al ser continua, corresponden a 0x000000.

3.3.COMPARATIVA DE SEÑALES MEDIDAS GENERADAS CON EL GENERADOR DE FUNCIONES DS345

A continuación, se van a mostrar capturas del DAQ en distintos escenarios involucrando, además, distintas funcionalidades descritas en el capítulo 2.

En las Figura 3.4 y Figura 3.5 se puede apreciar una señal sinusoidal generada con el generador de funciones medida con el osciloscopio Textronix TDS 2022B y la misma señal medida con el DAQ.

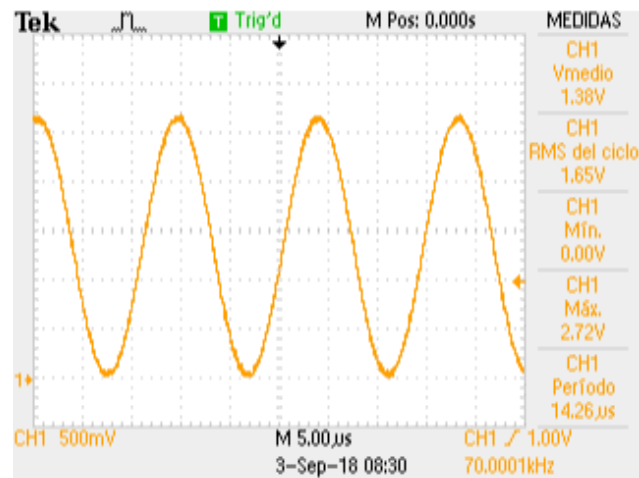


Figura 3.4 Señal de 70 kHz medida con osciloscopio TDS 2022B

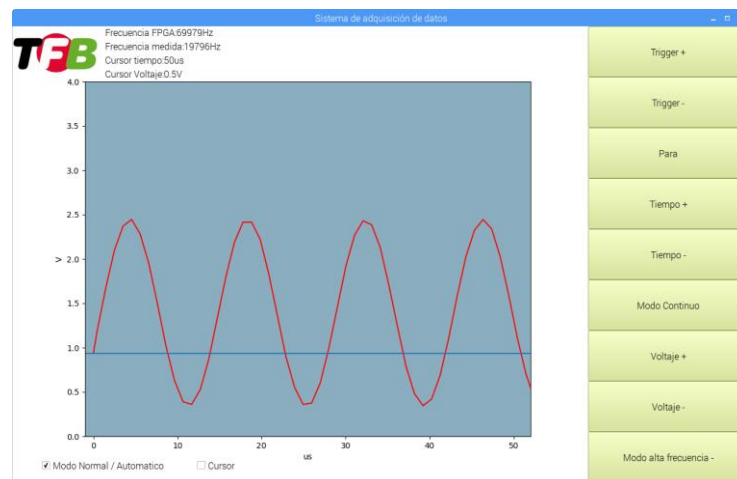


Figura 3.5 Señal sinusoidal de 70kHz medido con el sistema de adquisición de datos.

Podemos observar como a esta frecuencia la señal se representa con suficiente precisión. A 70 kHz, tendríamos una cantidad de 14 muestras por ciclo, lo que explica la representación fidedigna.

Una vez probado el sistema con una señal de 70 kHz, se procede a probar distintas señales de 20 kHz para observar el distinto comportamiento, tanto en el dominio de la frecuencia como en el del tiempo del sistema de adquisición de datos.

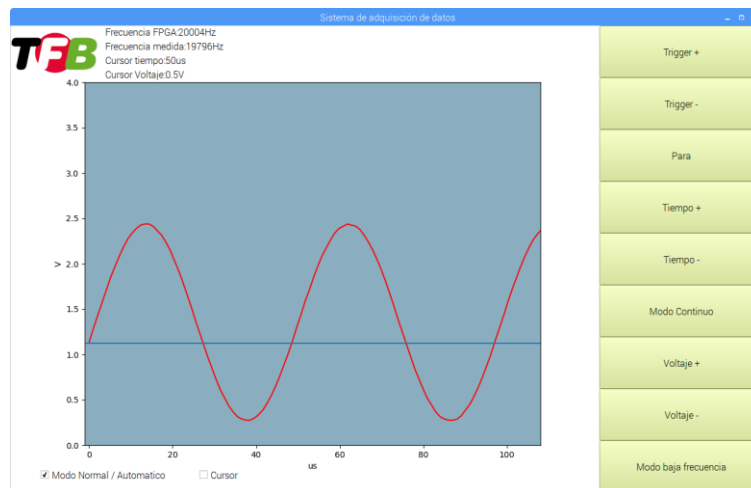


Figura 3.6 Señal sinusoidal de 20 kHz medida con el sistema de adquisición de datos.

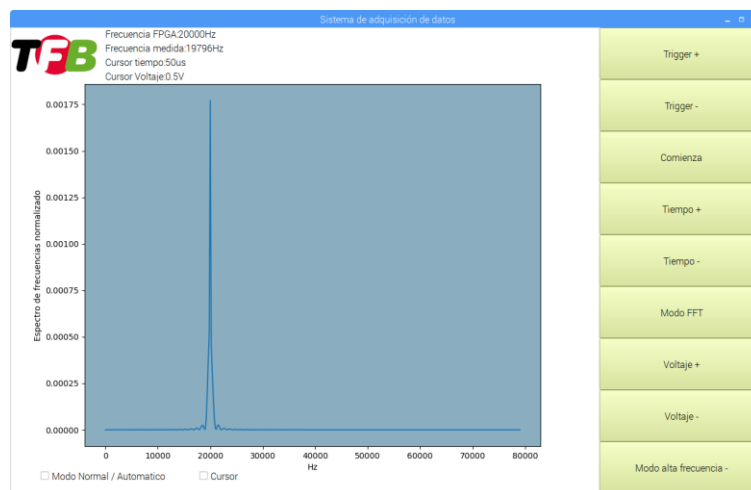


Figura 3.7 Periodograma normalizado de la señal sinusoidal de 20 kHz medida con el sistema de adquisición de datos.

En la Figura 3.6 se puede apreciar cómo se representa de forma correcta casi dos ciclos del periodo de la señal sinusoidal. Al pasar al modo FFT, obtenemos en la interfaz la Figura 3.7 en la que se puede apreciar, en frecuencia, la misma señal en forma de tono en 20kHz. La siguiente señal medida será una onda cuadrada tal y como se puede observar en la Figura 3.8.

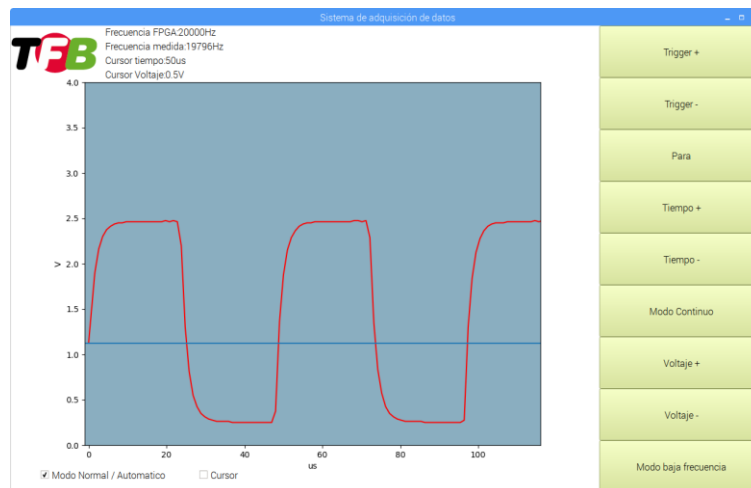


Figura 3.8 Señal Cuadrada de 20 kHz medida con el sistema de adquisición de datos.

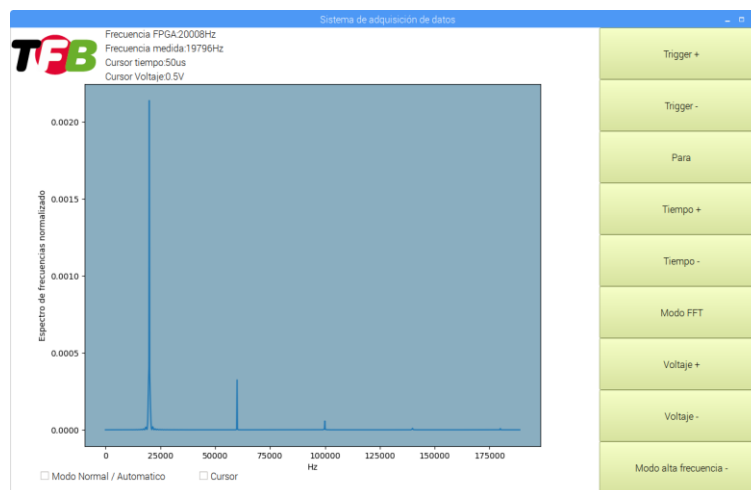


Figura 3.9 Periodograma de señal cuadrada de 20 kHz medida con el sistema de adquisición de datos.

Este par de figuras (Figura 3.8 y Figura 3.9) tienen como diferencia, que, en el periodograma se observa la transformada de Fourier del tren de pulsos (en valor absoluto). A continuación, se utiliza como señal de excitación un tren de señales triangulares.

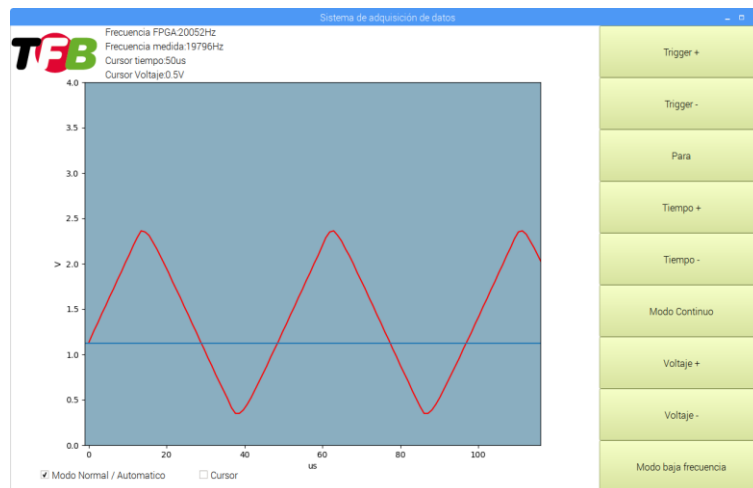


Figura 3.10 Señal triangular de 20 kHz medida con el sistema de adquisición de datos.

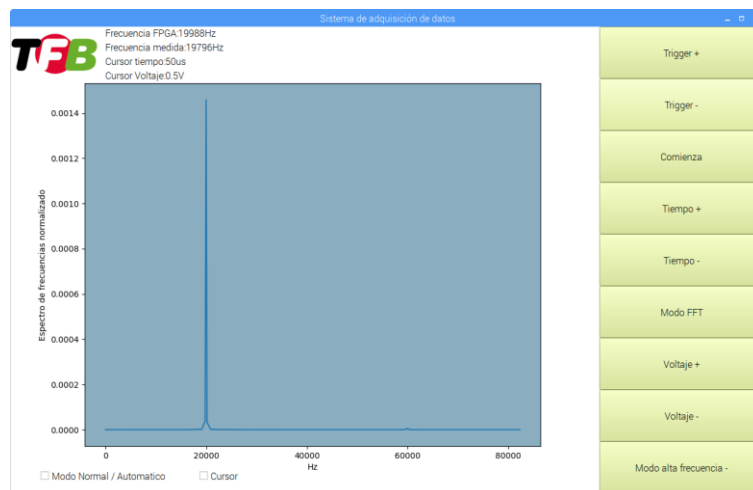


Figura 3.11 Periodograma de señal triangular de 20 kHz medida con el sistema de adquisición de datos.

Se concluye el buen funcionamiento del sistema para señales periódicas de frecuencias cercanas a los 20 kHz.

Al acercarnos a frecuencias más altas, se pueden comprobar dos efectos negativos para el sistema:

1. Reducción del nivel de señal debido a desadaptación de impedancia.
2. Pocas muestras por periodos que resultan en una pobre representación.

En la siguiente prueba, se va a comprobar el comportamiento del sistema para muestrear una señal sinusoidal de 350 kHz.

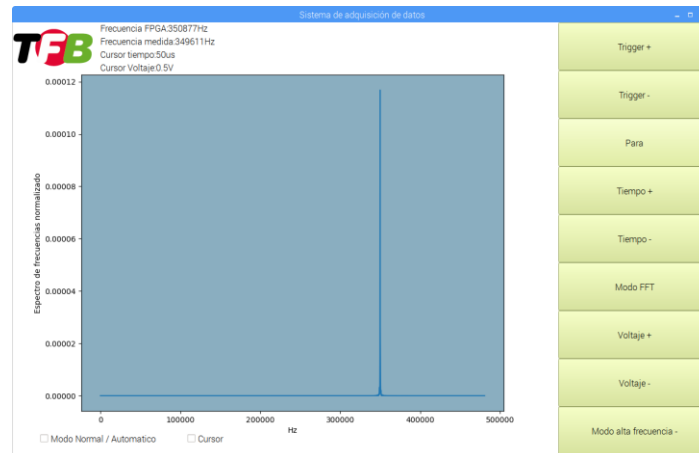


Figura 3.12 Periodograma de señal sinusoidal de 350 kHz medida con el sistema de adquisición de datos.

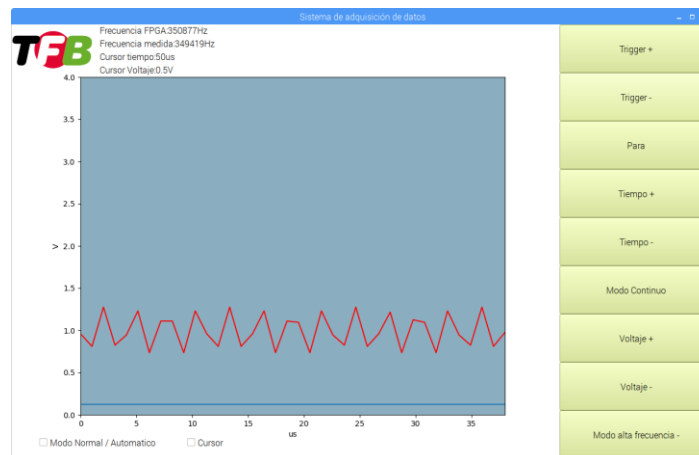


Figura 3.13 Señal de 350 kHz medida con el sistema de adquisición de datos.

Es recomendable tener presente la limitación de la tasa de muestreo del XADC, que, condiciona, a señales de menos de 500 kHz. Pero, a 350 kHz tendremos menos de 4 muestras por ciclo. Aun así, la representación en frecuencia sí que se aprecia correcta, siendo corroborable mediante los indicadores de Frecuencia FPGA y medida.

Para la siguiente prueba, se va a sobrepasar la frecuencia de muestreo máxima que respeta el criterio de Nyquist para el ADC que incluye la placa de evaluación Arty S7, es decir, supera los 500 kHz.

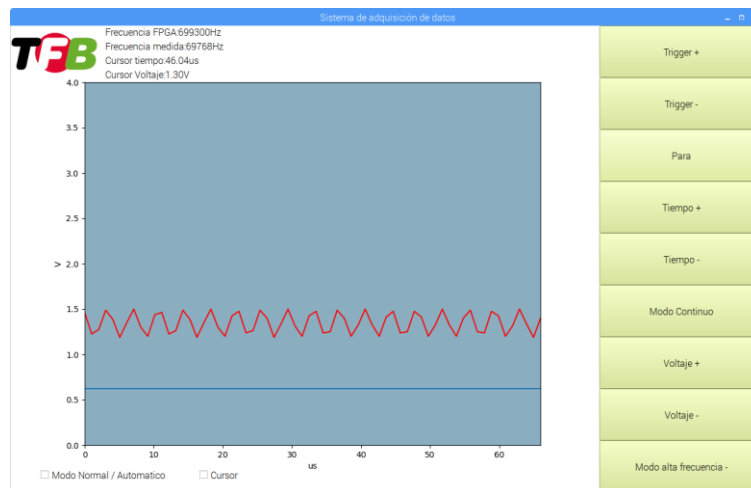


Figura 3.14 Señal de 700 kHz medida con el sistema de adquisición de datos.

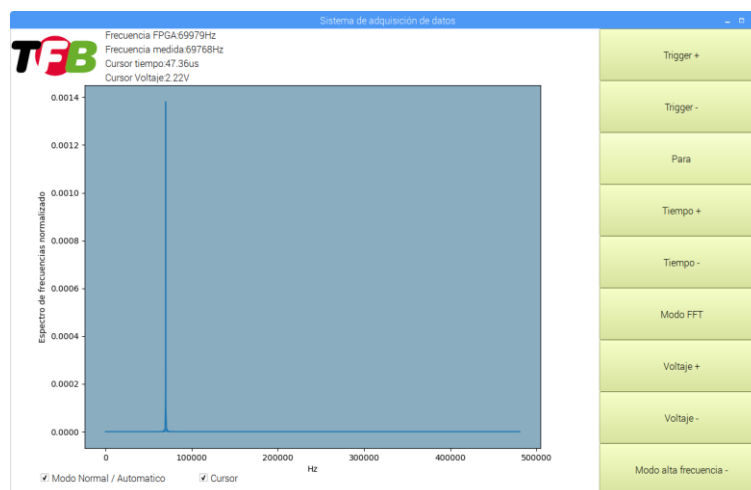


Figura 3.15 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos.

Podemos observar como la señal de 700 kHz, además de mostrarse desplazada en frecuencia, se ha reducido mucho el nivel de señal, en parte, debido a la desadaptación de impedancias.

En cambio, a 700kHz podemos observar tanto una señal incorrectamente representada, al sobrepasar 260kHz la frecuencia máxima (alrededor de 500 kHz) que el tono en la frecuencia máxima se encuentra en 300 kHz. Aun así, la FPGA sí que mide correctamente la frecuencia de 700 kHz con lo que hace posible reconstruir la señal.

Al activar el modo de alta frecuencia descrito en el capítulo 2 obtenemos:

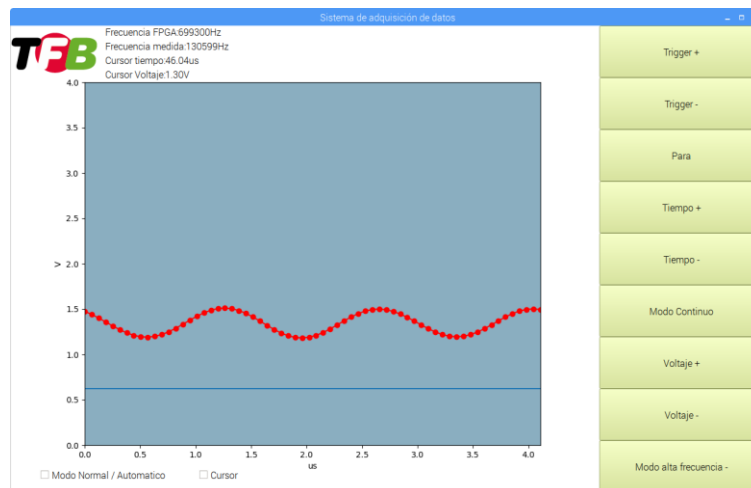


Figura 3.16 Señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.

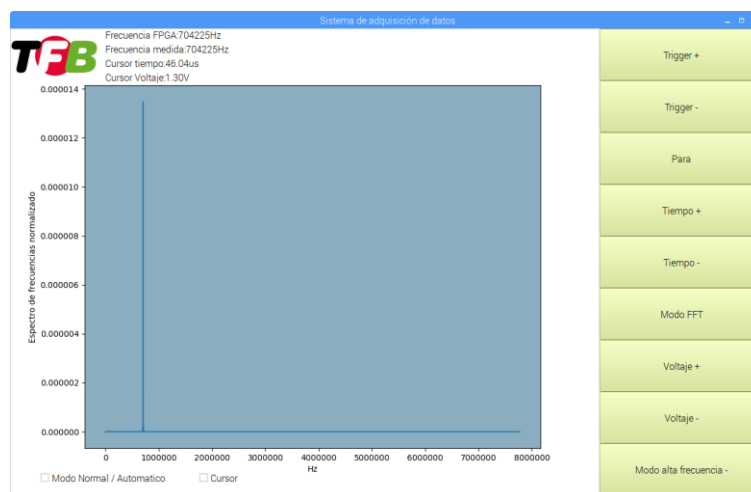


Figura 3.17 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.

Se puede observar la correcta reconstrucción de la señal y su coherencia en frecuencia con la señal medida en la FPGA.

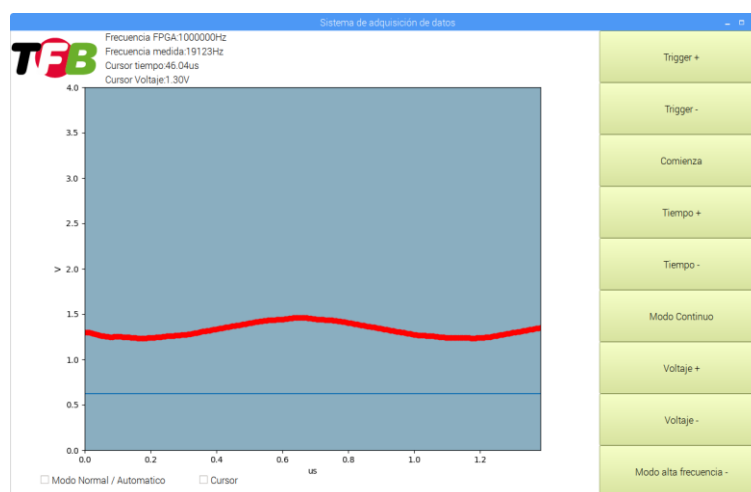


Figura 3.18 Señal sinusoidal de 1 MHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.

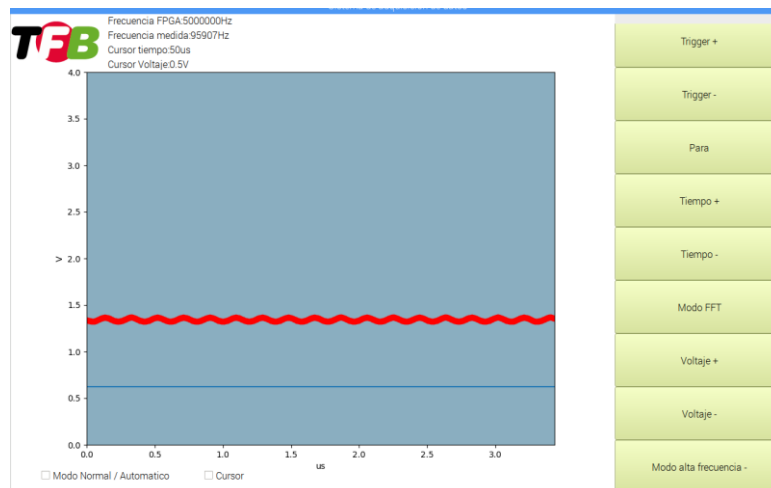


Figura 3.19 Señal sinusoidal de 5 MHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.

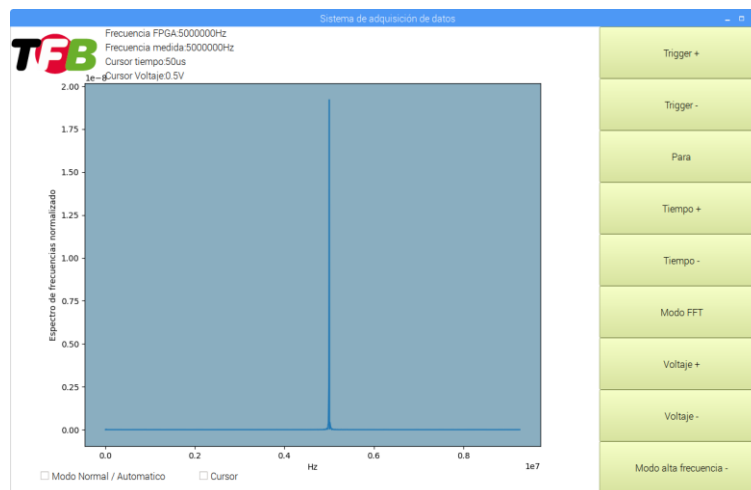


Figura 3.20 Periodograma de señal sinusoidal de 700 kHz medida con el sistema de adquisición de datos reconstruida y ajustada en frecuencia.

A pesar de la desadaptación de impedancia y el filtro paso bajo a la entrada, que provocan un nivel bajo de señal, seguimos obteniendo una señal correcta en frecuencia.

3.4.COMPARATIVA DE SEÑALES GENERADAS CON EL DDS AD9850

A continuación, se van a mostrar distintas mediciones de señales sinusoidales generados con el circuito de test que se analiza en el capítulo 2. Dicho circuito se controla desde la interfaz de usuario creada en Processing que también se explica en capítulos anteriores.

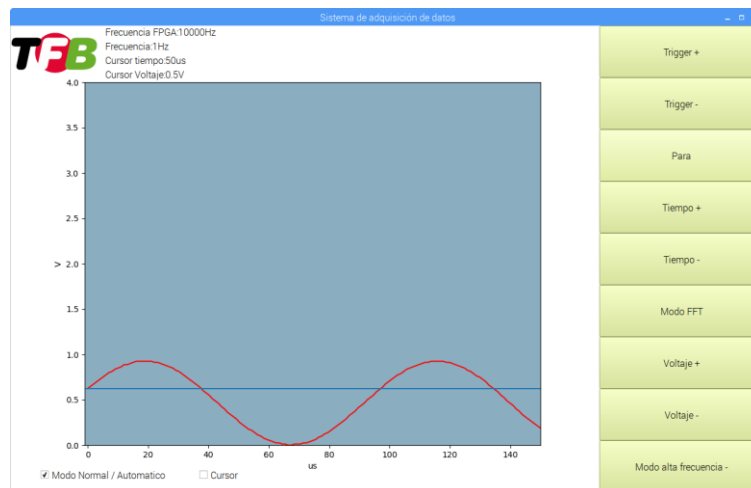


Figura 3.21 Señal sinusoidal de 10 kHz generada por el circuito de test.

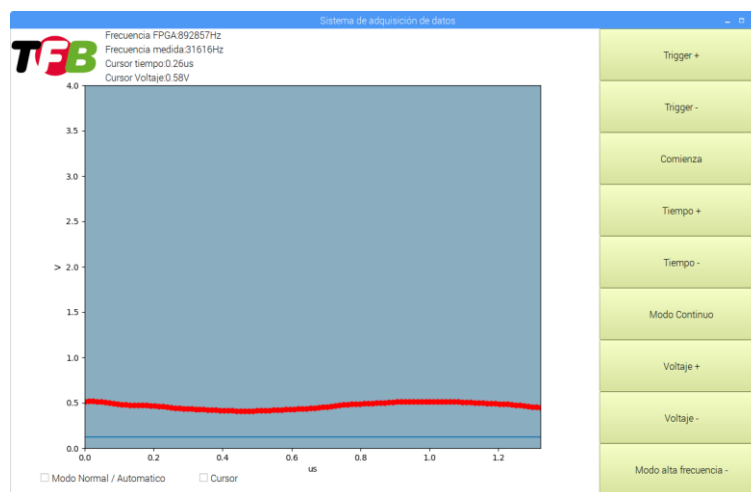


Figura 3.22 Señal sinusoidal de 900 kHz generada por el circuito de test.

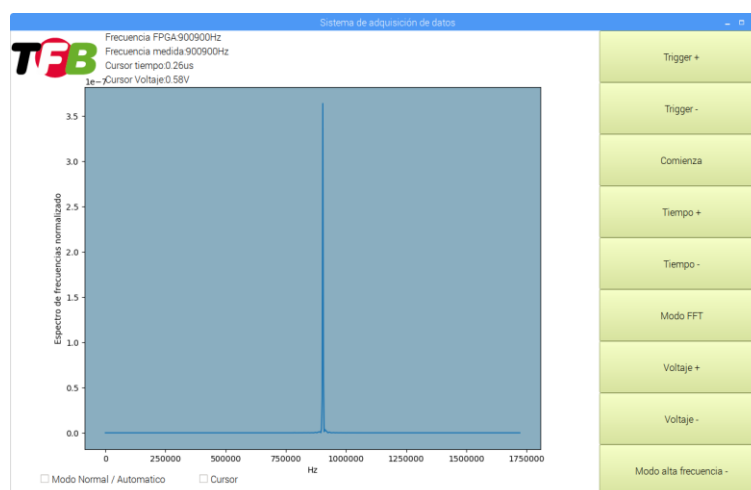


Figura 3.23 Periodograma de la señal sinusoidal de 900 kHz generada por el circuito de test.

En las figuras de este apartado se puede comprobar como el sistema sigue ajustando correctamente la frecuencia de las señales que sobrepasan los 500 kHz, sin embargo, el nivel de la señal es más bajo debido a que el circuito de test genera una señal sinusoidal de 1 a 0 V.

Esta limitación en amplitud provoca que la desadaptación de impedancias haga que la señal merme a frecuencias menores.

4. CONCLUSIONES Y LÍNEAS FUTURAS

4.1. CONCLUSIONES

Haciendo uso de las pruebas realizadas en el capítulo 3, se puede concluir que el sistema base tiene una funcionalidad que, dentro de unas condiciones de señal bastante estrictas como:

1. En cuanto a nivel deben ser señales positivas de entre 0 V y 3,3 V
2. En cuanto a frecuencia, a partir de cierto nivel de frecuencia, la desadaptación de impedancias debida a la red de resistencias que ajustan el voltaje de 3,3 V a 1 V , merman el nivel de señal.
3. En cuanto a la frecuencia máxima de la señal, a partir de 500 kHz, se empieza a submuestrear y es necesario desplazar en frecuencia la señal para la correcta muestra.
4. En cuanto al frecuencímetro, es necesario que la señal se encuentre en los rangos permitidos para LVCMOS33 que son: $[-0,3, 0,8]V$ para detección de nivel bajo y $[2, 3,45]V$ para detección de nivel alto.[9]

Bajo estas condiciones, el funcionamiento se muestra correcto, aun así, más allá de los requisitos para el correcto funcionamiento, existen otros componentes a mejorar en cuanto a la experiencia de uso como:

1. El funcionamiento del sistema podría realizarse de forma más veloz.
2. Mejorar la interfaz de usuario para hacerla más intuitiva.

Aun sabiendo los fallos que alberga el sistema, teniendo en cuenta el coste de este, disponemos de un sistema que puede desarrollar de forma correcta la mayoría de las funcionalidades normales en un laboratorio de electrónica de forma académica, e, incluso, dado que es un sistema asequible, es susceptible de préstamo.

Además, la mayoría de las incidencias registradas tienen que ver con la forma en la que la placa de evaluación Arty S7 recoge las muestras, más allá, el ADC que incorpora la placa de evaluación, es limitado tanto en frecuencia como en rango de voltaje.

La forma que tiene la placa de evaluación de acondicionar la señal para el ADC que se implementa mediante un divisor de tensión es bastante inadecuado. Por lo que se puede concluir que el principal defecto del sistema reside en la implementación de la conversión analógico digital de la placa Arty S7.

4.2. LÍNEAS FUTURAS

El principal problema que reviste el sistema es el ADC que se utiliza, las limitaciones que ofrece el módulo XADC de Xilinx, a saber:

1. Medida limitada al rango de 1V.
2. Máxima frecuencia de muestreo de 1 megamuestra por segundo.
3. Resolución de 12 bits de máximo.

Todas estas limitaciones se antojan suficientes para pensar en utilizar un ADC comercial e integrarlo junto a la FPGA. Como opción para utilizar en vez del ADC que incorpora la placa

de evaluación Arty S7 se propone el circuito integrado ADC08100 de Texas Instruments. Las características principales del ADC con respecto a las deficiencias encontradas en el sistema de origen son:[15]

1. Tasa de muestreo máxima de 100 megamuestras por segundo.
2. Rango de operación ajustable dentro de unos márgenes.
3. Se reduce el número de bits, que pasan de 12 a 8

Con estas características podemos aumentar el rango de normal funcionamiento del sistema desarrollado, además, se dispone de como se debe realizar el acondicionamiento de la señal en las notas de aplicación de su manual de referencia[15]. En dicho manual, aparece como se debe polarizar y como se adquieren las señales. En este caso, se utilizaría una interfaz en paralelo para recoger las muestras desde la FPGA. Dichas interfaces suelen estar contraindicadas para conexiones de longitud considerable y dispositivos con mala gestión de la sincronización. Así pues, es perfectamente recomendable para una FPGA. Para integrarla en el sistema completo bastaría con conectar las señales de fin de conversión al reloj de la FIFO y la entrada de datos de la FIFO a la entrada paralela.

Sería necesario, además, en el código de la interfaz de usuario, modificar la tasa de muestreo para la representación correcta de los datos. Con este ADC sería posible adaptar correctamente las impedancias al tener la posibilidad de realizar el circuito de acondicionamiento de la señal como aparece en la nota de aplicación.[15]

Otra línea de mejora es el uso de la GPU de la Raspberry Pi para el procesamiento de señal.

El concepto de computación de propósito general en unidades de procesamiento gráfico (GPGPU) trata de utilizar y estudiar las capacidades de cómputo de una unidad de procesamiento gráfica. El uso de GPU para el procesamiento de señales es de sobra conocido en el ámbito de las telecomunicaciones. Arquitecturas y lenguajes orientados a este concepto son CUDA y OpenGL. Raspberry Pi 3 utiliza un chip Broadcom que integra una GPU compatible con OpenGL. PyOpenGL es una librería para Python que permite utilizar OpenGL en la Raspberry Pi. Enmarcado en este proyecto se propone implementar funciones disponibles en la librería SciPy pero utilizando procesamiento paralelo en aquellas partes donde se concentre el cálculo iterativo y de vectores.

5. BIBLIOGRAFÍA

- [1] C. Anderson, *Makers: la nueva revolución industrial*. Empresa Activa, 2013.
- [2] J. G. Velasquez-Aguilar, F. Aquino-Roblero, M. Limon-Mendoza, L. Cisneros-Villalobos, and A. Zamudio-Lara, "Multi-channel data acquisition and wireless communication FPGA-based system, to real-time remote monitoring," *Proc. - 2017 Int. Conf. Mechatronics, Electron. Automot. Eng. ICMEAE 2017*, vol. 2017–Janua, pp. 181–186, 2017.
- [3] J. Park and S. Mackay, *Practical Data Acquisition for Instrumentation and Control Systems*. Elsevier, 2003.
- [4] A. J. Silva, "Reconstruction of Undersampled Periodic Signals," *Electr. Eng.*, 1988.
- [5] Xilinx Inc., "7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter - User Guide," vol. 480, p. 92, 2016.
- [6] E. Murphy and C. Slattery, "All about direct digital synthesis," *Analog Dialogue*, vol. 38, no. 3, pp. 8–12, 2004.
- [7] Analog Devices, "AD9850 - Complete DDS Synthesizer," *Current*, p. 20, 2004.
- [8] Digilent, "Arty S7 Reference Manual [Reference.Digilentinc]." [Online]. Available: <https://reference.digilentinc.com/reference/programmable-logic/arty-s7/reference-manual>. [Accessed: 01-Aug-2018].
- [9] Xilinx, "Spartan-7 FPGAs Data Sheet: DC and AC Switching Characteristics," vol. 182, p. 34, 2014.
- [10] E. Upton and G. Halfacree, *Raspberry Pi User Guide*, 4th ed. Wiley, 2012.
- [11] S. Pomyen, "Signal and Image Processing With Matlab on Raspberry Pi Platform," no. February, 2015.
- [12] C. Liechti, "pySerial Documentation," 2013.
- [13] Q. Jinhui, L. D. Hui, and Y. Junchao, "The Application of Qt/Embedded on Embedded Linux," *2012 Int. Conf. Ind. Control Electron. Eng.*, pp. 1304–1307, 2012.
- [14] M. I. T. Opencourseware, "Data Resampling: Interpolation (Up-sampling) and Decimation Up-Sampling (Interpolation) by an Integer Factor," *Signal Processing*, 2008.
- [15] Ti, "ADC08100 8-Bit, 20 Msps to 100 Msps, 1.3 mW/Msps A/D Converter Check for Samples: ADC08100 1FEATURES DESCRIPTION," no. May, 2000.

ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

En la introducción se hace referencia a los aspectos económicos, sociales y éticos en los que se enmarca este trabajo.

ANEXO B: CÓDIGO DE LOS DISTINTOS BLOQUES

El código de todas las partes que involucran el sistema se puede encontrar en el siguiente enlace:

<https://github.com/rino202CarlosCortes/TFM>