Programa de doctorado:
Ingeniería Agraria, Alimentaria, Forestal y del Desarrollo Rural Sostenible

NUEVAS TÉCNICAS PARA LA MEJORA DE LOS PROCESOS DE ELABORACIÓN DE LA ACEITUNA DE MESA DE ESTILO SEVILLANO

NEW TECHNIQUES FOR THE IMPROVEMENT OF THE ELABORATION PROCESSES OF "ESTILO SEVILLANO" TABLE OLIVES

TESIS DOCTORAL

DIRECTORES:

Rafael E. Hidalgo Fernández

Antonio Madueño Luna

AUTOR:
José Miguel Madueño Luna
Depositada en IDEP: 13/02/2021

Departamento de Ingeniería Gráfica

Universidad de Córdoba

Marzo 2021

TITULO: *Nuevas técnicas para la mejora de los procesos de elaboración de la aceituna de mesa de estilo sevillano*

AUTOR: *José Miguel Madueño Luna*

© Edita: UCOPress. 2021
Campus de Rabanales
Ctra. Nacional IV, Km. 396 A
14071 Córdoba

https://www.uco.es/ucopress/index.php/es/
ucopress@uco.es

TÍTULO DE LA TESIS:
NUEVAS TÉCNICAS PARA LA MEJORA DE LOS PROCESOS DE ELABORACIÓN DE LA ACEITUNA DE MESA DE ESTILO SEVILLANO

DOCTORANDO/A:
José Miguel Madueño Luna

INFORME RAZONADO DEL/DE LOS DIRECTOR/ES DE LA TESIS

La presente tesis doctoral aporta al estado del arte una nueva metodología que permite caracterizar variedades de aceituna en verde y procesadas haciendo uso únicamente de la impedancia eléctrica medida en la pulpa, redes neuronales (NN) e Internet de las cosas (IoT) y con ello recolectar información de las muestras de aceitunas analizadas tanto en explotaciones agrícolas como en fábricas.

Para la consecución de ese objetivo, el doctorando ha desarrollado previamente un equipo basado en el System on Chip (SoC) AD5933 que ejecuta una transformada discreta de Fourier (DFT) de 1024 puntos para devolver el valor de la impedancia como módulo y fase y que trabajando junto a dos multiplexores analógicos AD706 y un reloj externo programable basado en un DDS sintetizado en una FPGA XC3S250E-4VQG100C, permite la medida de la impedancia en productos agroalimentarios con un barrido en frecuencia desde 1 Hz a 100 kHz.

Como indicios de calidad esta tesis ha dado lugar a la siguiente publicación científica:

Título:
Characterization and Differentiation between Olive Varieties through Electrical Impedance Spectroscopy, Neural Networks and IoT
https://doi.org/10.3390/s20205932
Autores:
José Miguel Madueño Luna
Antonio Madueño Luna
Rafael E. Hidalgo Fernández
Revista:
https://www.mdpi.com/journal/sensors
Indicios de calidad:
Q1, IF=3,275

Por todo ello, se autoriza la presentación de la tesis doctoral.

HIDALGO FERNANDEZ RAFAEL ENRIQUE - 30514517A

Firmado digitalmente por HIDALGO FERNANDEZ RAFAEL ENRIQUE - 30514517A
Fecha: 2021.02.12 20:19:38 +01'00'

Sevilla, 12 de febrero de 2020

Firma del/de los director/es

MADUEÑO LUNA ANTONIO - 30791753C

Firmado digitalmente por MADUEÑO LUNA ANTONIO - 30791753C
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-30791753C, givenName=ANTONIO, sn=MADUEÑO LUNA, cn=MADUEÑO LUNA ANTONIO - 30791753C
Fecha: 2021.02.12 15:27:44 +01'00'

Fdo.: Rafael E. Hidalgo Fernández Fdo.: Antonio Madueño Luna

Agradecimientos

- *Gracias a Rafael y Antonio, tutores de este trabajo, por toda la ayuda prestada.*

José Miguel Madueño Luna

Córdoba, 2021

Resumen

La medida de la impedancia eléctrica ha demostrado ser útil para medir propiedades y características de productos agroalimentarios: calidad de frutos, contenido de humedad y capacidad de germinación en semillas o la resistencia de los frutos a las heladas. En el caso de aceitunas se ha usado para determinar contenido de grasa y el momento óptimo de recolección.

En el presente trabajo se ha desarrollado un equipo basado en el System on Chip (SoC) AD5933 que ejecuta una transformada discreta de Fourier (DFT) de 1024 puntos para devolver el valor de la impedancia como módulo y fase y que trabajando junto a dos multiplexores analógicos AD706 y un reloj externo programable basado en un DDS sintetizado en una FPGA XC3S250E-4VQG100C, permite la medida de la impedancia en productos agroalimentarios con un barrido en frecuencia desde 1 Hz a 100 kHz.

Con esta aportación se demuestra cómo la impedancia eléctrica se ve afectada por la temperatura tanto en las aceitunas recién recolectadas como en las procesadas en salmuera y proporciona una nueva técnica para la mejora de los procesos de elaboración de la aceituna de mesa de estilo sevillano al caracterizar los cultivares empleando unicamente la impedancia eléctrica, redes neuronales (NN) e Internet de las cosas (IoT).

Palabras clave:

Impedancia eléctrica; SoC AD5933; redes neuronales artificiales (ANNs); internet de las cosas (IoT); temperatura

Abstract

Electrical impedance has shown itself to be useful in measuring the properties and characteristics of agri-food products: fruit quality, moisture content, the germination capacity in seeds or the frost-resistance of fruit. In the case of olives, it has been used to determine fat content and optimal harvest time.

In this paper, a system based on the System on Chip (SoC) AD5933 running a 1024-point discrete Fourier transform (DFT) to return the impedance value as a magnitude and phase and which, working together with two ADG706 analog multiplexers and an external programmable clock based on a synthesized DDS in a FPGA XC3S250E-4VQG100C, allows for the impedance measurement in agri-food products with a frequency sweep from 1 Hz to 100 kHz.

With this contribution, it is demonstrated how the electrical impedance is affected by the temperature both in the olives just harvested and in those processed in brine and provides a new technique for the improvement of the production processes of the “Estilo sevillano” table olives when characterizing cultivars using only electrical impedance, neural networks (NN) and Internet of things (IoT).

Keywords:

Electrical impedance; SoC AD5933; artificial neural networks (ANNs); internet of things (IoT); temperature

INDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

AC	Corriente alterna
ADC	Convertidor analógico/digital
ARM CORTEX M3	Tipo de procesador de 32 bits empleado
C	Condensador
CLK	Reloj
DDS	Generador de síntesis digital
DDS1	DDS empleado con AD9850
DFT	Transformada discreta de Fourier
DUE	Arduino modelo "DUE"
DUT	Dispositivo bajo medición (en nuestro caso impedancia)
f	Frecuencia
F	Fase de la impedancia eléctrica
f_0	Frecuencia de corte del filtro elíptico empleado por el AD9850
fa(i)	Vector de fases durante la medida del DUT
fc(i)	Vector de fases durante la calibración
FCLK	Frecuencia de reloj
FMAX	Valor máximo de la frecuencia de reloj
FMIN	Valor mínimo de la frecuencia de reloj
I^2C	Interfaz serie de dos hilos de Philips
IDE	Entorno de desarrollo (para Arduino)
L	Bobina
ma(i)	Vector de módulos durante la medida del DUT
mc(i)	Vector de módulos durante calibración
MCLK	Pin de entrada del reloj de AD5933
MSPS	Millones de muestras por segundo
PCB	Placa de circuito impreso
PGA	Amplificador de ganancia programable
Q	Factor de calidad de una bobina
R	Parte real de la impedancia eléctrica o resistor
RCL/RLC	Circuito con resistencia, bobina y condensador
RS232C	Comunicación serie
TTL	Tecnología transistor-transistor
USB	Comunicación USB
VDD	Tensión de alimentación a 3.3V
VIN	Pin de entrada de la tensión de excitación
VOUT	Pin de salida de la tensión de excitación
V_P	Tensión de pico
VPP	Tensión pico a pico
X	Parte imaginaria de la impedancia eléctrica
Z	Módulo de la impedancia eléctrica
ZBF	Resistencia de FeedBack
Zcal	Impedancia de calibración
Zin	Impedancia a medir (DUT)

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Efecto de la temperatura

La temperatura afecta la textura de frutas y hortalizas, tanto crudas [1] como cocidas [2], y también afecta a sus parámetros eléctricos como la impedancia eléctrica de su pulpa. Existe una amplia variedad de procedimientos y equipos [3,4] para obtener diversos parámetros asociados a la pulpa de estas frutas y verduras (resistencia a la penetración, viscosidad, etc.)

En el caso de la aceituna, no existen estudios que analicen explícitamente la influencia de la temperatura en el fruto, especialmente en sus parámetros eléctricos, aunque se conoce su efecto en determinados procesos mecanizados de la aceituna. Este es el caso, por ejemplo, del proceso industrial de deshuesado de la aceituna [5], que se realiza mediante máquinas automáticas donde se utilizan mordazas para aprisionar la aceituna, mientras que un punzón deshuesador atraviesa dichas mordazas y, en consecuencia, las aceitunas, provocando el deshuesado de las mismas. Este tipo de máquina alcanza una velocidad de deshuesado considerable, lo que obviamente supone un coste mínimo por unidad de fruta deshuesada. Sin embargo, las mismas mordazas que sujetan el fruto y el deshuesador provocan la rotura de un número considerable de aceitunas durante su funcionamiento normal, que puede ser del orden del 14%, especialmente en la variedad de aceituna “Gordal Sevillana”. Si bien la productividad de estas deshuesadoras las hace más rentables a pesar del porcentaje de rotura de aceitunas en comparación con los anteriores sistemas de deshuesado, estas máquinas presentan un grave problema, ya que el porcentaje de aceitunas rotas es bastante considerable.

Si bien no existen soluciones mecánicas para reducir el porcentaje de aceitunas rotas en tales máquinas deshuesadoras, ha podido comprobarse que un enfriamiento de las aceitunas, previo a su deshueso, minimiza el problema, hasta el punto de que, con una refrigeración adecuada, el porcentaje de aceitunas rotas durante el proceso de deshuesado, se sitúa en el orden del 2%, para las aceitunas de la variedad “Gordal Sevillana” anteriormente citada.

Por otra parte, la tendencia actual de los consumidores por valores más bajos de acidez y sal [6], hace que gran parte de las aceitunas vayan deteriorándose durante el tiempo que pasan en los fermentadores. Al cabo de los meses y con una temperatura media de 25°C, las probabilidades de obtener gran cantidad de mermas en el proceso del deshuesado son muy altas, debido a la pérdida de dureza de las aceitunas, especialmente las de gran tamaño, que son las más cotizadas en el mercado. Mediante el enfriamiento a temperaturas cercanas a los 7°C (en algunos casos se llega a 0°C en función del sistema de enfriamiento y los costes que desea asumir la empresa en este proceso), es posible incrementar la dureza de las aceitunas manteniendo todas sus características y, así reducir las mermas debido a la textura más rígida de la aceituna y, por otra

parte minimizar las horas-máquina de funcionamiento, lo que se traduce en una disminución de atascos y horas de parada de las punzonadoras.

Una manera de caracterizar el estado de las aceitunas previo a su deshuesado sería realizando una medida de su impedancia eléctrica de una muestra representativa. Como se va a demostrar con el presente estudio, la impedancia eléctrica de la pulpa de la aceituna no está afectada sólo por la variedad o el tipo de procesamiento industrial (aceitunas verdes "Estilo Sevillano" o negras "Estilo Californiano") [7], sino también por la temperatura. Por ello caracterizar este parámetro eléctrico versus temperatura (en especial entre 7°C y 0°C) para un procesamiento industrial dado permitiría reconocer a través de la impedancia eléctrica sí las aceitunas son aptas para el deshuesado.

1.2 La impedancia eléctrica: forma de medirla

La determinación de propiedades eléctricas se utiliza en una amplia gama de disciplinas e industrias [8]. En el sector agroalimentario, el uso de la conductividad eléctrica se aplica para la determinación de diversas características en productos agroalimentarios [9,10] como sensibilidad a las heladas, tolerancia a la congelación, contenido de humedad y germinación de semillas [11]. El uso de espectroscopia de impedancia eléctrica es una técnica que puede proporcionar muy buenos resultados en la etapa de madurez de frutas y hortalizas. En este artículo, nos centraremos en un System on Chip (SoC) capaz de realizar una medición de impedancia compleja: el SoC AD5933 [12].

La impedancia es un parámetro de gran importancia para la caracterización de circuitos y componentes electrónicos [13], así como de los materiales utilizados en su producción. La impedancia (Z) se define generalmente como la oposición total que ofrece un dispositivo o circuito al flujo de una corriente alterna (CA) a una frecuencia específica, y se representa como un número complejo con una representación gráfica en un plano complejo. Un vector de impedancia consta de la parte real (resistencia, R) y la parte imaginaria (reactancia, X). La impedancia se puede expresar usando las coordenadas rectangulares en forma de $R + j \bullet X$, o en forma polar como magnitud y ángulo de fase: $| Z | \angle Ø$ [14].

Los instrumentos más utilizados para medir la impedancia son: el medidor LCR o puente LCR y el analizador de impedancia. El primero proporciona una medición de impedancia simple y exacta para un valor de frecuencia específico. Sin embargo, para componentes que no sean inductores puros (L), condensadores (C) o resistencias (R), es inadecuado para determinar el valor. En estos casos, se utiliza un analizador de impedancia para medir y representar gráficamente la impedancia compleja del dispositivo que se está probando en un rango de frecuencias [15]. Al tratarse de dispositivos de alto coste [16,17], existen diseños alternativos más económicos (medidores LCR) que permiten obtener un sistema analizador de impedancia al combinarlos con instrumentación virtual [12,18].

Existen varias configuraciones para el diseño de puentes medidores de impedancia como: el puente Schering [19] y el puente Maxwell [20]. Estas tienen la dificultad de que requieren obtener la condición de balance. Además, se utilizan generalmente para la medición de impedancias inductivas o capacitivas puras. Para la obtención de la impedancia en forma compleja se utilizan métodos electrónicos, tales como el método vectorial y el método que utiliza dos ondas sinusoidales en cuadratura [21], [22]. Otra forma de medir la impedancia en forma compleja es el método de las tres tensiones [23], sin embargo, este precisa de tensiones elevadas al cuadrado lo que hace que los errores de la medición sean mayores, además de que requiere instrumentos muy precisos para realizar las mediciones.

Un analizador de Impedancia/Ganancia-Fase [24] es un instrumento de medición de gran utilidad para el estudio y diseño de circuitos electrónicos. Este potente dispositivo es capaz de obtener por separado los diagramas tanto en módulo como en fase de cualquier red o circuito electrónico que disponga de una entrada y una salida. Con esta información, es posible, por ejemplo, obtener la función de transferencia de un determinado circuito, aunque se desconozca su implementación.

Un analizador de Impedancia/Ganancia-Fase debe ser capaz de obtener con fiabilidad los diagramas de amplitud y fase correspondientes a un determinado circuito. En ambos casos, se representarán dichas variables en función de la frecuencia. Estos tipos de analizadores deben poseer la capacidad de generar una señal de carácter sinusoidal y aplicarla directamente sobre la entrada de la red que se desea medir. De este modo se realiza un barrido de frecuencia sobre la red con un determinado criterio en lo que se refiere a los valores de frecuencia inicial, intermedios, final, número de puntos, de barrido lineal o logarítmico, etc. que normalmente es seleccionado por el usuario. Para representar el diagrama de módulo es necesario realizar el cociente entre las amplitudes de salida y de entrada de la red que se desea medir para cada uno de los valores de frecuencia del barrido. Por otro lado, para realizar la representación del diagrama de fase es necesario obtener la diferencia de fase entre las señales de salida y entrada de la red, de nuevo para cada uno de los valores de frecuencia. Por lo tanto, es obvio que para poder construir un analizador de impedancia/ganancia-fase se precisa como mínimo, de un generador senoidal para generar un barrido de frecuencias, de un circuito capaz de medir amplitudes y de otro circuito capaz de medir la diferencia de fase entre dos señales.

1.3 Ejemplos previos de uso de la impedancia eléctrica

En [25] se hace uso de la medida de conductividad eléctrica empleando sólo el módulo de la misma para caracterizar diferentes variedades de aceitunas (Olea europea L.) concretamente cuatro cultivares diferentes, 'Picual', 'Manzanilla de Sevilla', 'Hojiblanca' y 'Gordal Sevillana"con el objeto de establecer un índice objetivo de madurez del fruto y así determinar el momento óptimo de recolección basado en parámetros como el rendimiento y la calidad del aceite, demostrándose

que la conductividad aumentaba con la maduración del fruto y que cada variedad tenía un valor promedio característico de conductividad eléctrica en las últimas etapas de maduración.

Estudios previos se han centrado en especial en determinar índices de madurez y parámetros de calidad [26], [7], [27], o en la manera de retrasar el periodo de maduración, sobre todo en frutos climatéricos, con el fin de alargar el periodo entre la recolección y el consumo con fines comerciales [28].

1.4 Redes neuronales (NN) de ajuste y de clasificado

Las redes neuronales artificiales son un modelo computacional inspirado en el comportamiento observado en su homólogo biológico [29]. Su uso está muy extendido en multiples de campos. Centrándonos en aceituna de mesa, podemos ver su uso en [30] o en [31].

En este estudio vamos a emplear dos tipos de redes neuronales, por una parte una red neuronal de ajuste [32] para valorar la efectividad de esta técnica a la hora de generar un modelo válido del comportamiento de la impedancia de la pulpa de la aceituna y por otra parte una red neuronal clasificadora [33] para distinguir entre variedades de aceitunas a varias temperaturas.

1.5 Internet de las cosas (IoT)

El uso del Internet de las Cosas (IoT) está actualmente muy extendido existiendo numerosos casos de aplicación; por ejemplo [34], [35], [36], [37], [38] en agricultura de precisión, riego, control de temperatura, monitorización de procesos productivos en agricultura, en la cadena de valor del olivar (automated-olive-chain) o en el funcionamiento de las propias máquinas deshuesadoras, rodajadoras y rellenadoras (DRR) [30], [31].

El presente trabajo muestra el desarrollo de un equipo basado en este SoC AD5933 [12] que junto una red neuronal y un sistema IoT permite el análisis en finca o fábrica del estado de las aceitunas previo a su deshuesado.

1.6 Objetivos

El objetivo general que se plantea con el presente trabajo es desarrollar un sistema de medida de la impedancia eléctrica adaptado a aceituna de mesa haciendo uso del SoC AD5933. Para ello se van a llevar a cabo dos tareas:

- Modelización de la impedancia mediante redes neuronales para dos variedades de aceitunas (“Gordal Sevillana” y “Hojiblanca”) cocidas en sosa cáustica y fermentadas en salmuera (proceso industrial conocido como “Estilo Sevillano”).
- Clasificación mediante redes neuronales de cada variedad a tres temperaturas (25°C, 7°C y 0°C).

Para ello se va a desarrollar:

- Un dispositivo específico con el SoC AD5933 [12], que incluirá la interface de comunicaciones I^2C [39], un generador externo DDS basado en una FPGA XC3S250E-4VQG100C [40] para poder realizar un barrido completo desde 1 Hz a 100 kHz y dos multiplexores analógicos ADG706 [41] para fijar el rango de impedancia a medir. Este dispositivo está controlado por un microcontrolador ARM CORTEX M3 de 32 bits AT91SAM3X8E trabajando a 84 MHz [42].
- El software para el control del sistema empleando como lenguaje de programación Matlab [43].
- Comunicación IoT basada en [44] para generar una base de datos con los resultados de los ensayos.

2. Materiales y Métodos

2.1. Variedades de aceituna y proceso industrial empleado

El fruto del olivo (Olea europaea L.) [25] es una drupa ovoide cuyo tamaño oscila entre 0,6 y 2 cm de diámetro y entre 1 y 4 cm de longitud. Este tamaño depende de la variedad (ver Figura 1 en [25]), el estado vegetativo del árbol, las condiciones ambientales y las técnicas de cultivo [45]. Se utilizaron muestras de 100 aceitunas verdes de olivares (200 árboles ha^{-1}, Sevilla, España), en condiciones de riego y nutrientes no limitantes con recolección mecánica y muestras de 100 aceitunas en salmuera de una fábrica de Sevilla. En este trabajo utilizamos dos variedades: "Hojiblanca" y "Gordal Sevillana" recién recolectadas en las clases 0, 1 y 2 [46] y tratadas en salmuera al "Estilo Sevillano". Este tipo de aceituna, aderezada de esta manera, se procesa en cuatro etapas [47-49]:

- Cocido en NaOH 2–4% (p/v) durante 6–12 h
- Lavado en agua (12–15 h)
- Fermentación en salmuera (10–12% (p/v) durante 60–300 días)
- Deshuesado, relleno o rodajado.

2.2 El SoC AD5933 para la medida de impedancia

Según la hoja de datos del SoC AD5933 [12] "el AD5933 es un convertidor de impedancia de alta precisión que combina un generador de frecuencia integrado con un convertidor analógico-digital (ADC) de 12 bits y 1 MSPS. ...) La señal de respuesta de la impedancia (DUT) es muestreada por el ADC interno y es procesada con una transformada discreta de Fourier (DFT)" [50]. El voltaje de excitación de salida y la frecuencia de medición son totalmente programables. La comunicación se lleva a cabo por una interfaz I^2C.

Revisando la bibliografía disponible se constata que este SoC AD5933 tiene notables aplicaciones biológicas, así por ejemplo se ha usado para monitorizar el crecimiento de cultivos celulares [51], [52], mediciones en células aisladas [53], detección de coagulación en la sangre [54], aplicaciones como biosensor [55], medidas de bio-impedancia [56-60]. Asimismo se usa en la vigilancia de "objetos técnicos", por ejemplo, para el análisis de la corrosión en estructuras de acero [61], [62]. Para obtener una información completa sobre las propiedades eléctricas de un objeto medido y utilizar un método conveniente del análisis de espectros de impedancia (modelado de circuitos equivalentes), la impedancia debe medirse en una amplia gama de frecuencias [63].

En la tabla 1 se muestran los datos técnicos de varios medidores de impedancia basados en el SoC AD5933. Sólo tres de los dispositivos descritos permiten medir la impedancia en más de tres órdenes de magnitud de frecuencias. El rango de impedancia medida es típicamente de 10 Ω a

más de 1 MΩ. Sin embargo, en muchos casos no se da el rango exacto. La mayoría de los medidores de impedancia mencionados requieren "front-end" analógicos adicionales para proporcionar una interfaz adecuada entre el SoC AD5933 y el dispositivo bajo medida (DUT) [64].

Tabla 1. Lista de medidores de impedancia eléctrica basados en el SoC AD5933 (tomada de [64])

Autor	Propósito	Rango de frecuencias	Rango de impedancia	Error máximo
C. J. Chen et al. [46]	Monitoreo de cultivos de células	Fijo a 10 Hz	Sin especificar	Sin especificar
T. Schwarzenberger et al. [47]	Monitoreo de cultivos de células	100 Hz – 100 kHz	Sin especificar	2 % - módulo, 2 %- argumento
M. H. Wang et al. [48] (usa un AD5934)	Medidas en células aisladas	0.1 Hz – 100 kHz	100 Ω – 10 MΩ	Entorno al 10 % para medidas en células
J. Broeders et al. [50]	Aplicación como biosensor	10 Hz – 100 kHz	10 Ω – 5 MΩ	Sin especificar
P. Bogónez-Franco et al. [52]	Monitor de Bioimpedancia	100 Hz – 200 kHz	10 Ω – 1 kΩ	2.5 % - módulo, 4.5 % - argumento
J. Ferreira et al. [53]	Electrodos en ropa para Bioimpedancia	5 kHz – 450 kHz	Sin especificar	0.7 % - resistencia, 17 % - reactancia
C. Margo et al. [54]	Aplicaciones "embebidas" para bioimpedancia	1 kHz – 100 kHz	Sin especificar No data	2.5 % - módulo, 1.3 % - argumento
A. Melwin y K. Rajasekaran [55]	Medidas de la composición del cuerpo	Fijo a 50 kHz	Sin especificar	2 % (sin especificar)
J. Hoja y G. Lentka [56],[57]	Monitorización técnica de objetos	0.01 Hz – 100 kHz	10 Ω – 10 GΩ	1.6 % - módulo, 0.6 % - argumento

Otros autores [65-67], proponen modificaciones en la topología original proporcionada por el fabricante con el uso de un sistema de multiplexado para ajustar el rango de medida de impedancia si bien no incluyen cambios en la fuente reloj que se inyecta al SoC AD5933.

En los **ANEXOS** y en la referencia [68] se describe el funcionamiento del SoC AD5933, el hardware, el software desarrollado y el sistema IoT.

2.3. Redes neuronales para modelizado y clasificación

Se han empleado dos tipos de redes neuronales de las librerías de Matlab:

- Una red neuronal de ajuste del tipo fitnet [32] que permite obtener una descripción de la evolución de la impedancia compleja de la pulpa de la aceituna en las 2 variedades y que es una mejora sustancial sobre el modelo de Hayden [69].

- Una red clasificadora tipo patternnet [33] para distinguir entre 6 casos (2 variedades y 3 temperaturas) en verde y otras tantas para aceitunas procesadas al “Estilo Sevillano”.

3 Resultados

Se han efectuado tres pruebas sobre el hardware desarrollado (ver apartados 3.1-3.3). Un estudio de como afecta la variedad de aceitunas verdes sobre la impedancia (apartado 3.4) e igualmente para las aceitunas aderezadas en salmuera (apartado 3.5).

Se ha desarrollado un modelo de redes neuronales capaz de mostrar la evolución de la impedancia en aceitunas verdes y aderezadas en salmuera para dos variedades de aceitunas (ver apartado 3.6). Se ha implementado un clasificador neuronal que distingue entre 6 casos posibles en aceitunas en verde (ver apartado 3.7.1) y aderezadas (ver apartado 3.7.2.) y finalmente (ver apartado 3.8.) el sistema IoT empleado.

3.1 Ensayo de verificación del sistema de medida de la impedancia: Dispositivo bajo medida (DUT): Resistencia pura de valor 10 kΩ y 5% de tolerancia

Se ha realizado una prueba usando una resistencia pura de 10 KΩ, los resultados están en la Figura 1. El objeto de la prueba es asegurarse de que el equipo está funcionando correctamente. Como puede verse, la respuesta es la misma tanto en magnitud como en la fase en todo el barrido en frecuencia desde 1 Hz hasta 100 kHz.

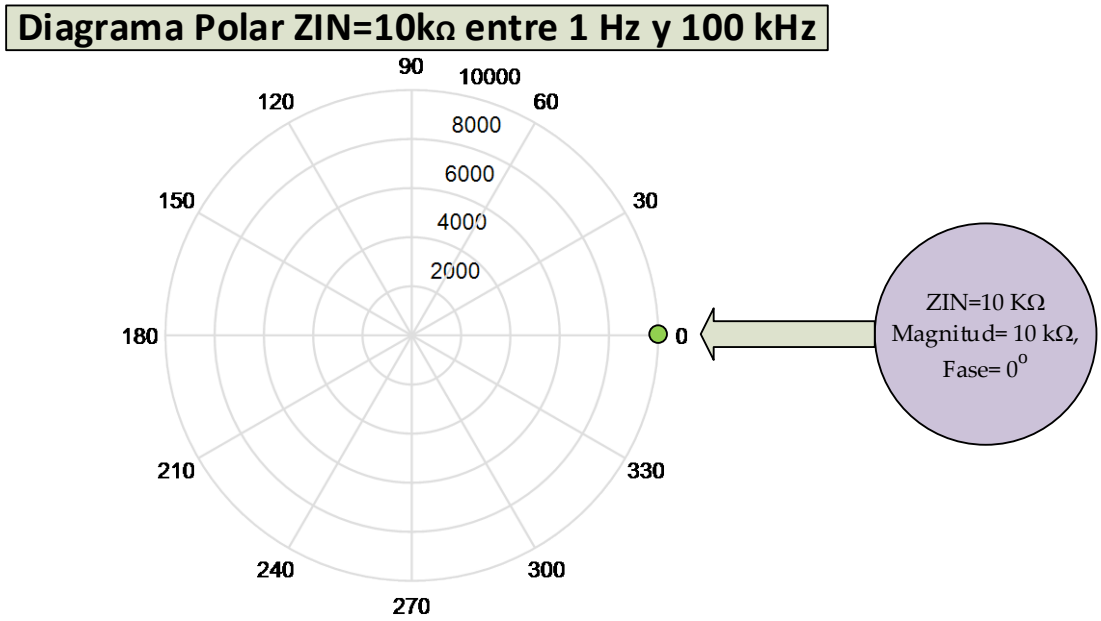


Figura 1. Diagrama Polar: Barrido completo desde 1 Hz hasta 100 kHz con ZIN = 10 KΩ (Magnitud 10 kΩ, Fase 0°).

3.2 Ensayo de verificación del sistema de medida de la impedancia. Dispositivo bajo medida (DUT): Circuito con componentes RLC en serie

Se ha realizado una prueba en un circuito RLC en serie con una inductancia de L = 56 mH, capacidad C = 1500 pF and resistencia R = 5.1 kΩ, con una resistencia de calibración Zcal = 10 kΩ, con una resistencia de realimentación ZBF = 10 kΩ, Rango 1: 2 Vpp, PGA = ×1, Multiplicador = ×1 desde 1 Hz to 100 kHz (ver Tablas 4 y 5 en el ANEXO 1 o las tablas 1y 2 en [68]).

Para la prueba, se conectaron tres elementos en serie para construir un circuito RLC típico. La frecuencia de resonancia teórica es:

$$f_o = \frac{1}{2\cdot\pi\cdot\sqrt{L\cdot C}} = \frac{1}{2\cdot\pi\cdot\sqrt{56\times10^{-3}\times1500\times10^{-12}}} = 17365.22\ \text{Hz} \quad (1)$$

A bajas frecuencias predomina la parte capacitiva con fases negativas que tienden a -90 °. En las proximidades de la zona de resonancia, la fase tiende a cero y desde ese momento comienza a predominar la parte inductiva, la fase, en ese caso positiva, tiende a 90 °. Los resultados teóricos y experimentales se muestran en las Figuras 2a y 2b. Las pequeñas discrepancias entre los datos teóricos y experimentales se deben a la tolerancia de los componentes utilizados en la prueba real.

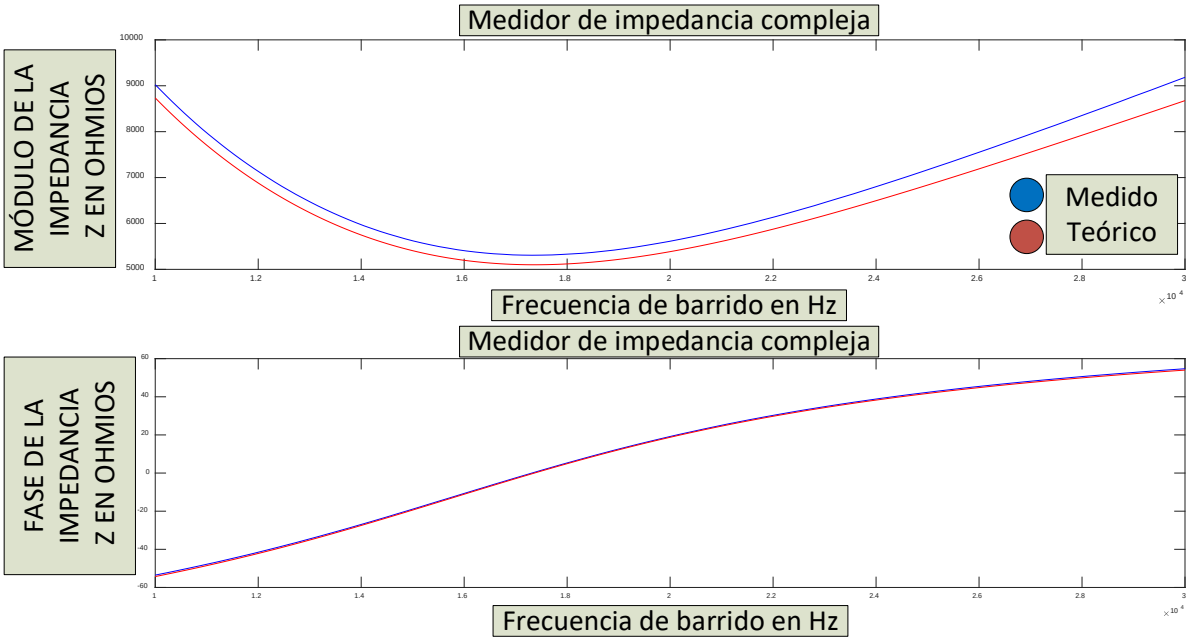


Figura 2a. Diagrama X-R del test con circuito con componentes RLC en serie efectuado entre 1 kHz y 30 KHz. En azul se representan los valores medidos y en rojo los teóricos.

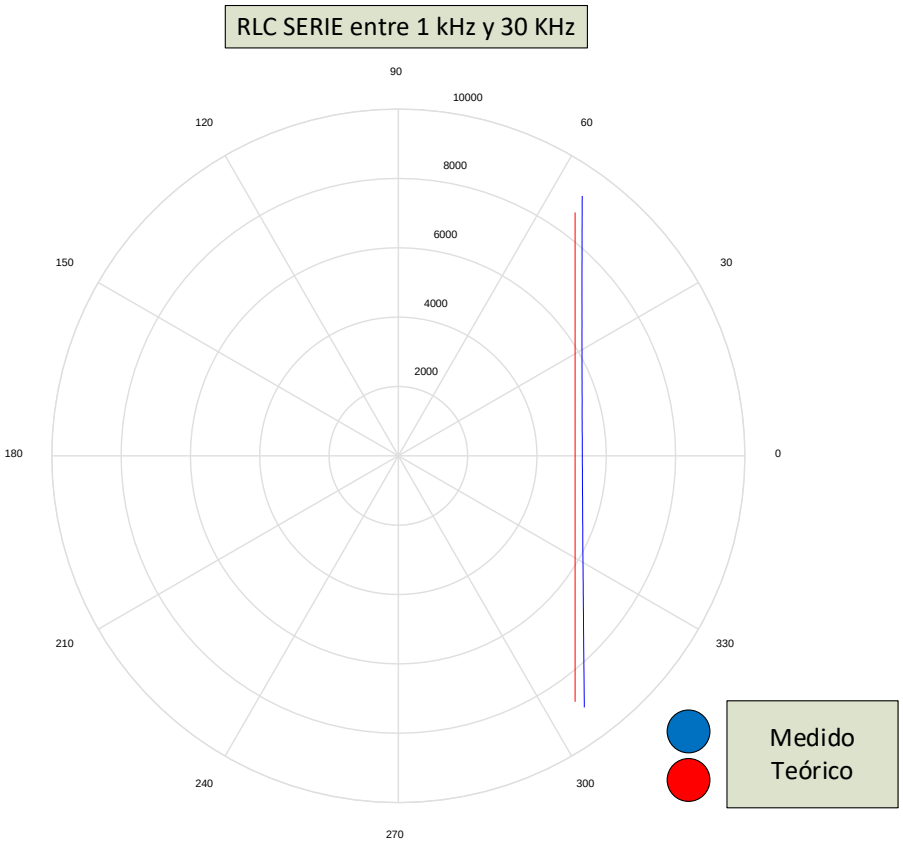


Figura 2b. Diagrama polar del test con circuito con componentes RLC en serie efectuado entre 1 kHz y 30 KHz. En azul se representan los valores medidos y en rojo los teóricos.

3.3 Ensayo de verificación del sistema de medida de la impedancia: Dispositivo bajo medida (DUT): Circuito con componentes RLC en paralelo

Se ha realizado una prueba en un circuito RLC en paralelo con una inductancia de L = 56 mH, capacidad C = 1500 pF and resistencia R = 5.1 kΩ, con una resistencia de calibración Zcal = 10 kΩ, con una resistencia de realimentación ZBF = 10 kΩ, Rango 1: 2 Vpp, PGA = ×1, Multiplicador = ×1 desde 1 Hz to 100 kHz (ver Tablas 4 y 5 en el ANEXO 1 o las tablas 1 and 2 en [68]).

Para este ensayo se conectaron los tres components en paralelo formando un circuito RLC paralélo típico. La frecuencia teórica es la misma que la del ensayo anterior.

Los resultados teóricos y experimentales se muestran en las Figuras 3a y 3b. Una vez más, las diferencias observadas son debidas a las tolerancias de los componentes empleados.

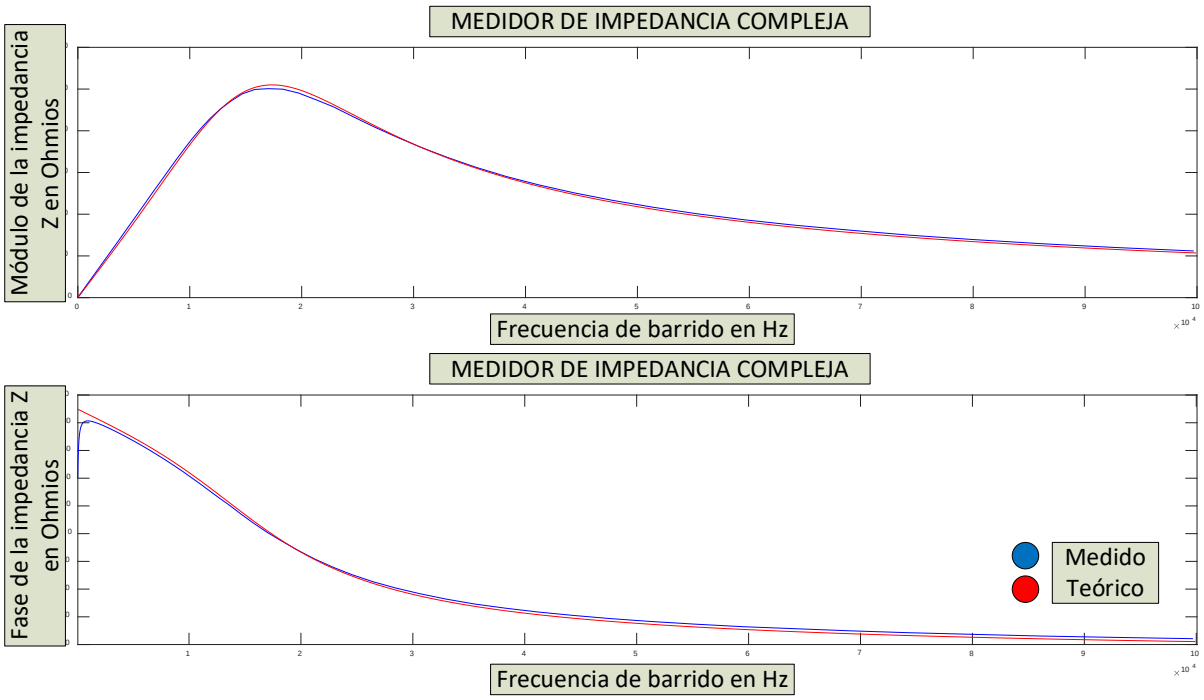


Figura 3a. Diagrama X-R del test con circuito con componentes RLC en paralelo efectuado entre 1 Hz y 100 KHz. En azul se representan los valores medidos y en rojo los teóricos.

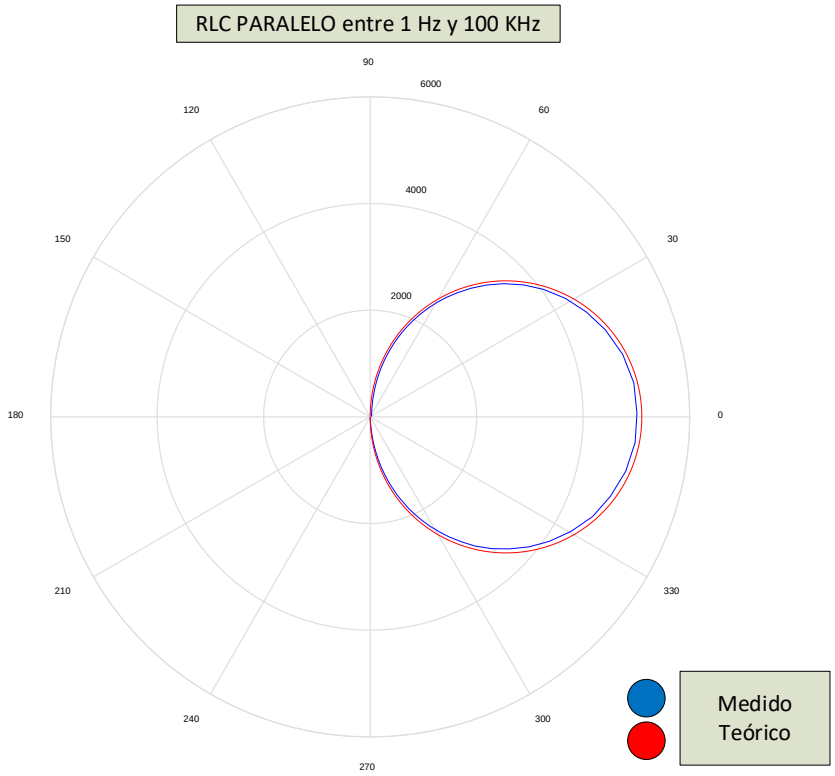


Figura 3b. Diagrama polar del test con circuito con componentes RLC en paralelo efectuado entre 1 Hz y 100 KHz. En azul se representan los valores medidos y en rojo los teóricos.

3.4. Prueba en aceitunas verdes: el efecto de la variedad de aceitunas en la impedancia eléctrica

Este estudio se ha ocupado de muestras de aceitunas de las variedades “Gordal Sevillana” y “Hojiblanca”, mostradas en la Figura 4, de dos olivos situados uno al lado del otro y por tanto sometidos a las mismas condiciones ambientales y de riego. Estas muestras se recolectaron el mismo día a la misma hora (16/06/2019).

En la Figura 5, la evolución del perfil de impedancia obtenido (una media de 100 pruebas) aparece presentada como sus componentes X-R para aceitunas verdes de ambas variedades a tres temperaturas: 0 ° C (azul), 7 ° C (negro) y 25 ° C. (rojo).

Como se puede apreciar, presentan un perfil característico diferenciado tanto por temperatura como por valores máximos, siendo la “Hojiblanca” la que presenta valores más altos en ambos componentes X, R, y por ende el módulo Z, tanto a 0 ° C como a 7 ° C, mientras que a 25 ° C su perfil es similar al del “Gordal Sevillana”.

Pruebas anteriores han demostrado que estos perfiles de impedancia no se ajustan bien a modelos del tipo descrito en [70] (ver Figura 3b de dicha cita) con un mínimo en X cercano a cero. Por el contrario, en aceitunas verdes a baja frecuencia, hay valores altos tanto en R como en X (círculos rojos en los gráficos). Asimismo, se pueden observar dos mínimos relativos en el valor de reactancia X (rombos verdes) a 7 ° C y 25 ° C. La tendencia en el gráfico de 0 ° C sugiere que a frecuencias superiores a 100 kHz también existiría en este caso.

Uno de los problemas que a veces aparece es la distinción entre las variedades inmaduras, debido a que su apariencia exterior y tamaño provocan dudas. Esta prueba demostró cómo estas variedades (inmaduras) afectan la impedancia eléctrica al igual que el resto de parámetros (especialmente la temperatura). Este sistema permitiría identificar la variedad de aceituna de la muestra analizada a partir de la impedancia eléctrica sin utilizar otros parámetros caracterizadores.

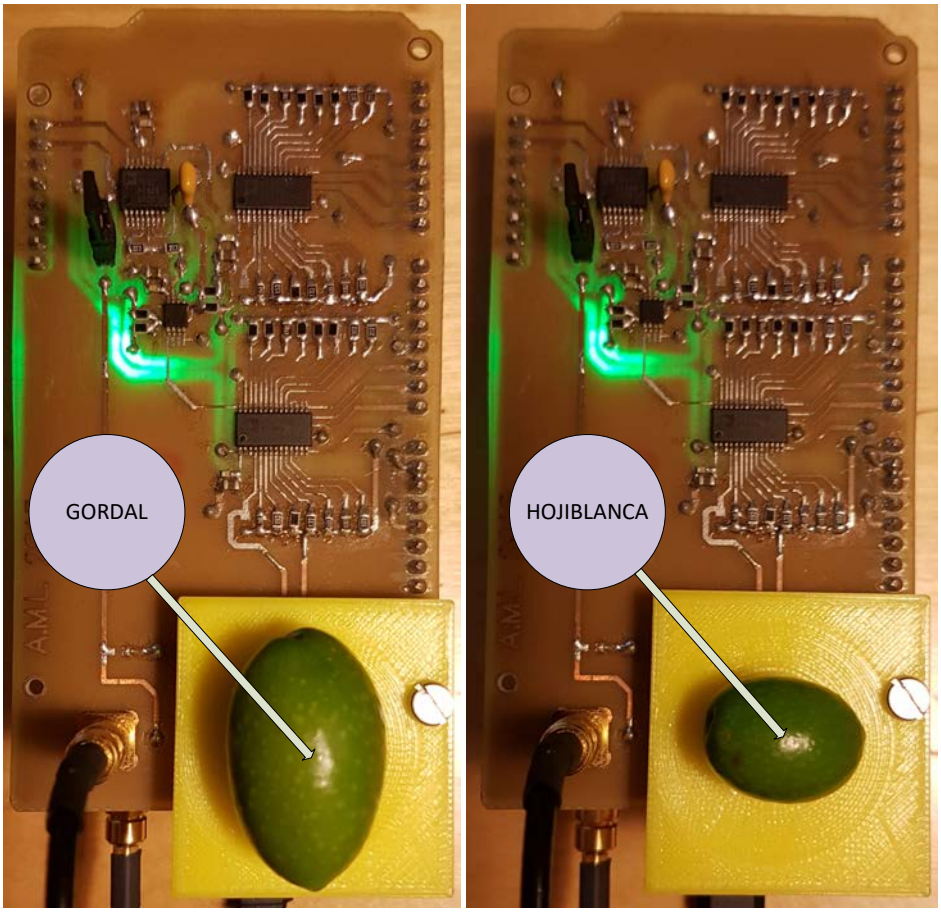


Figura 4. Variedades de aceituna (a) “Gordal Sevillana” y (b) “Hojiblanca” durante la medida de la impedancia eléctrica.

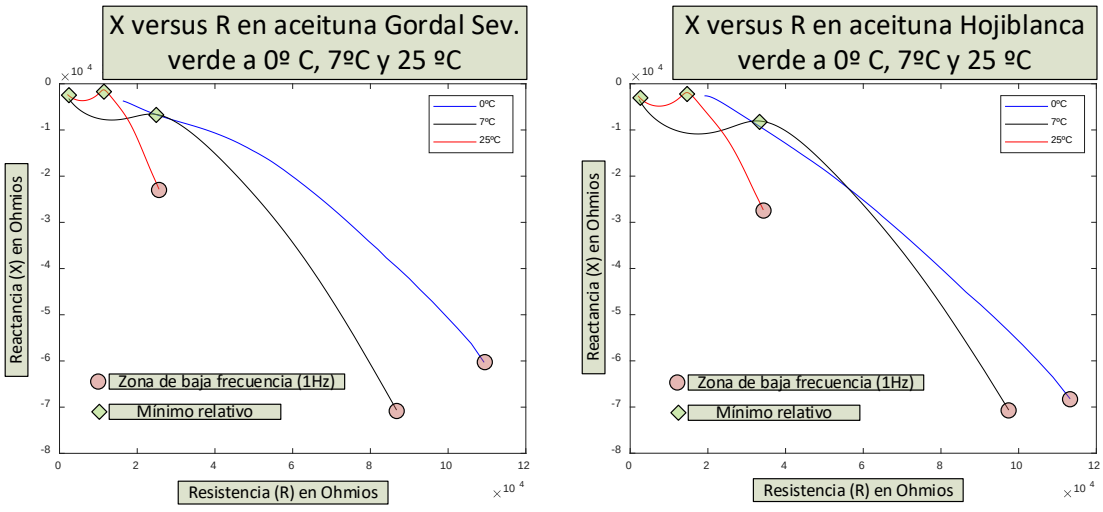


Figura 5. Evolución de la impedancia en (a) en las variedades en verde “Gordal Sevillana” y (b) “Hojiblanca” a 3 temperaturas: 0 °C, 7 °C, y 25 °C (promedio de 100 tests). Los círculos rojos corresponden a f = 1 Hz

En la Figura 6, el espectro de impedancia de ambas variedades se muestra conjuntamente (en diagrama polar). Cada una tiene un perfil característico a cada temperatura y la máxima diferencia se encuentra a bajas frecuencias, lo que podría servir para distinguir entre variedades de aceitunas verdes como también se indica en [25].

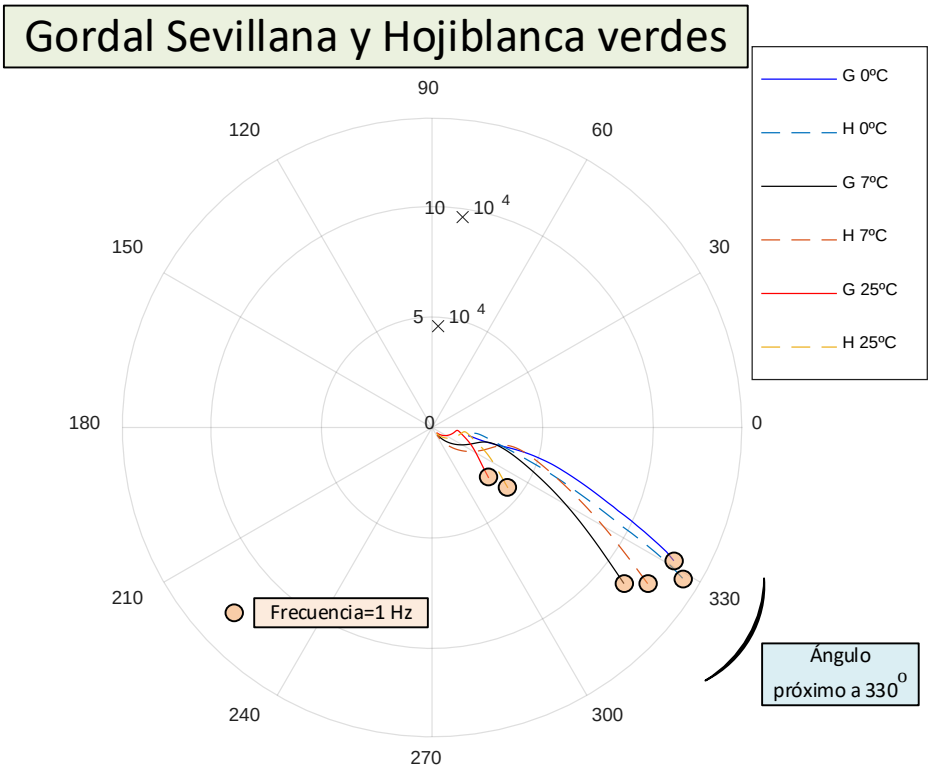


Figura 6. Barrido en frecuencia para medir la impedancia eléctrica en las variedades en verde “Gordal Sevillana” y “Hojiblanca” (valores promedio de 100 ensayos).

3.5. Prueba en aceitunas procesadas: efecto de la variedad de aceituna sobre la impedancia eléctrica en aceitunas procesadas al estilo sevillano

Al igual que ocurre con las aceitunas verdes, a veces es difícil distinguir entre variedades procesadas por su forma, tamaño y especialmente por su color (debido a los procesos químicos que sufren) siendo muy similares. Una vez más, se realizó un estudio sobre las dos variedades anteriormente mencionadas (“Gordal Sevillana” y “Hojiblanca”) aliñadas al “Estilo Sevillano”.

En la Figura 7, se muestra el efecto de la temperatura en estos tipos de aceituna con 100 muestras de cada variedad a 3 temperaturas (0 ° C, 7 ° C y 25 ° C). Una vez más, cada variedad tiene una característica de espectro de impedancia eléctrica a cada temperatura y las diferencias máximas entre las dos variedades se encuentran en frecuencias más bajas.

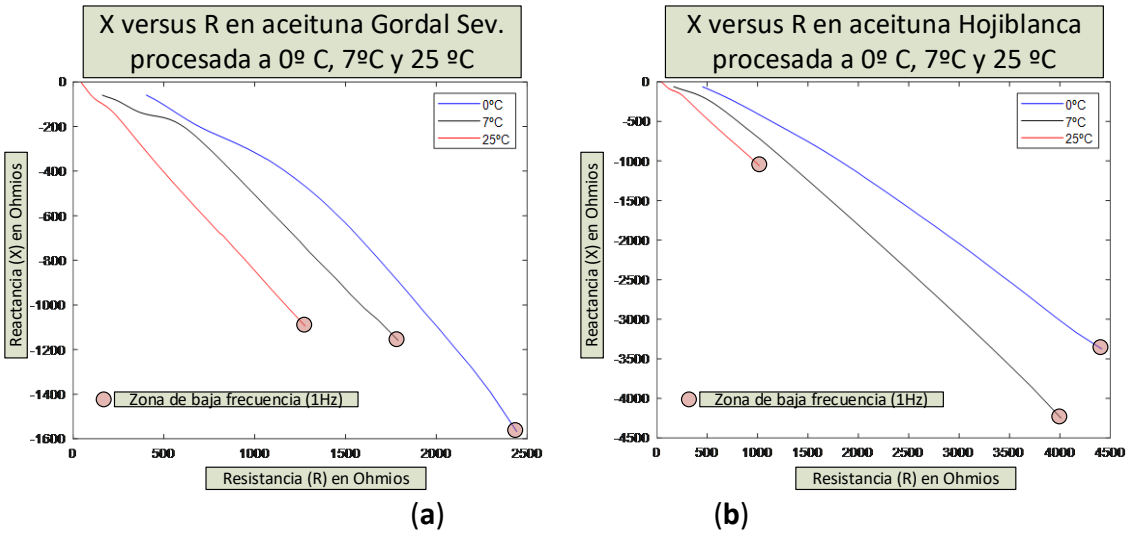


Figura 7. Efecto de la temperatura en la impedancia eléctrica para variedades procesadas al "Estilo Sevillano" (a) "Gordal Sevillana" (b) "Hojiblanca" a 3 temperaturas: 0 °C, 7 °C, y 25 °C (valores promedio de 100 ensayos).

Cabe resaltar que en el caso de las curvas R-X, los mínimos relativos a X no se presentan como sucedió en el caso de las aceitunas verdes. Tampoco son de aplicación los modelos clásicos descritos en [69,70].

Otro detalle a destacar es la drástica reducción de los valores en ambos componentes (R, X) (alrededor de 20 veces) debido a la presencia de salmuera, que hace que la pulpa de aceituna sea altamente conductora de la electricidad.

Finalmente, como se puede observar, comparando las Figuras 6-8, la presencia de salmuera hace que la pulpa se comporte de manera más capacitiva con un ángulo de desviación promedio alrededor de 310 ° para aceitunas en salmuera y alrededor de 330 ° para aceitunas verdes.

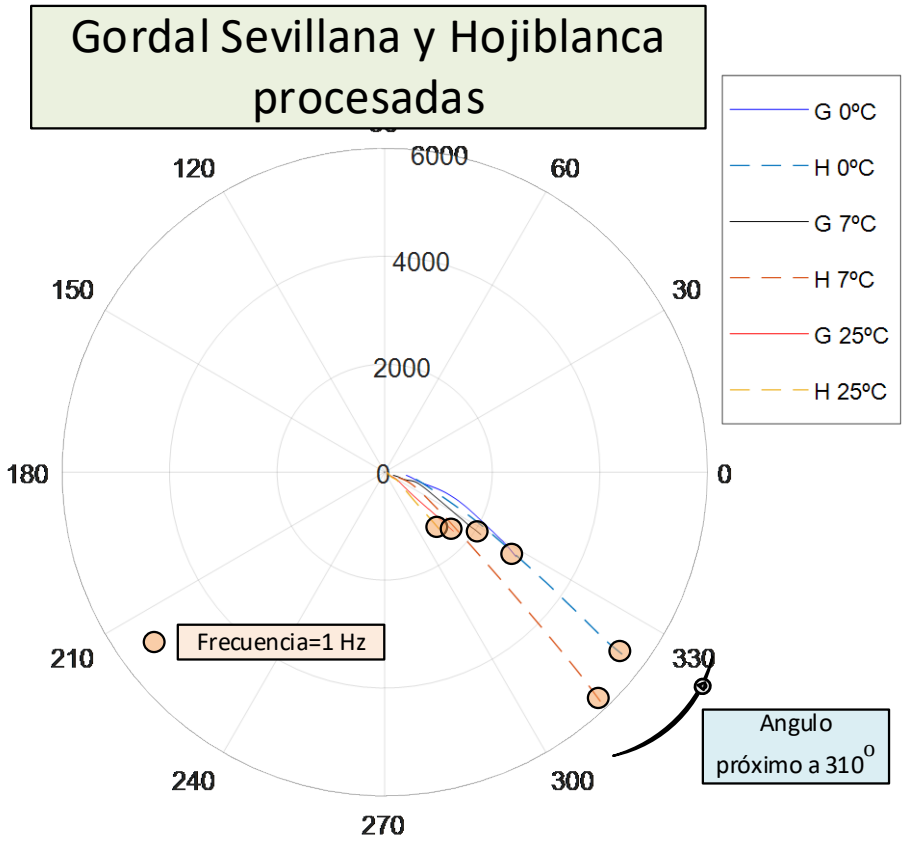


Figura 8. Influencia de la salmuera en la impedancia de las aceitunas procesadas.

3.6. Modelo de impedancia compleja para las variedades de aceituna "Gordal Sevillana" y "Hojiblanca" inmaduras y procesadas al estilo "Estilo Sevillano" a través de redes neuronales

Los modelos clásicos [69,70] no se adaptan bien para caracterizar la evolución de la impedancia en aceitunas; es decir, en aceitunas verdes o procesadas al estilo Sevillano, para cada variedad y temperatura. Así por ejemplo, el modelo de Hayden [69] no tiene suficientemente en cuenta los componentes R2 y C3, reduciendo sus efectos.

Por esta razón y tal como se adelantó en el apartado 2.3, se ha calibrado un modelo basado en redes neuronales para ambas variedades, tanto en aceitunas inmaduras como en aceitunas procesadas a las tres temperaturas vistas anteriormente correlacionando directamente los valores de las componentes X reactiva y R resistiva. Las Figuras 9-12 muestran algunos resultados utilizando las redes neuronales de la Toolbox de Matlab y su función fitnet [32], siendo suficiente una capa oculta con 5 neuronas para obtener un ajuste adecuado.

La bondad de ajuste se expresa de acuerdo con [71] y como error relativo en % en las Tablas 2 y 3.

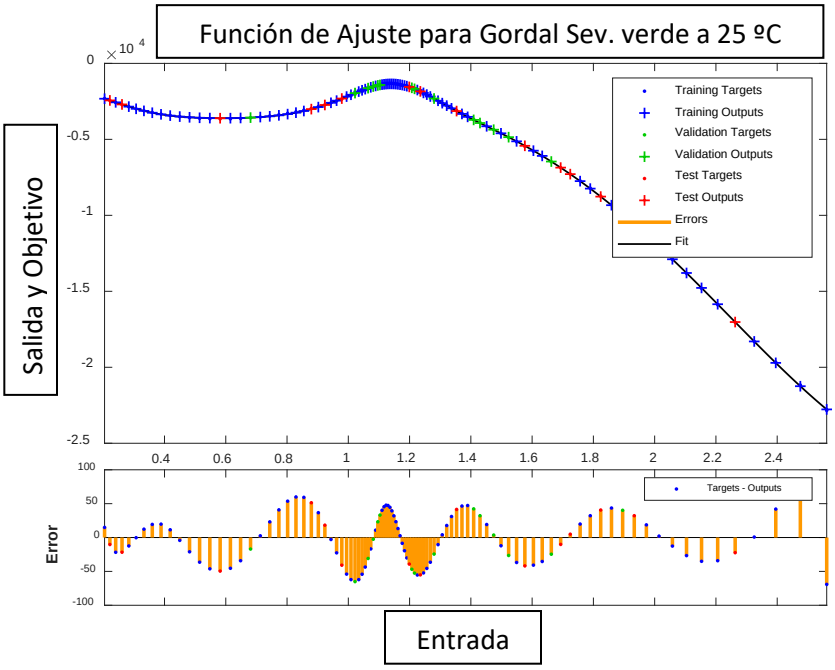


Figura 9. “Gordal Sevillana” verde a 25 °C

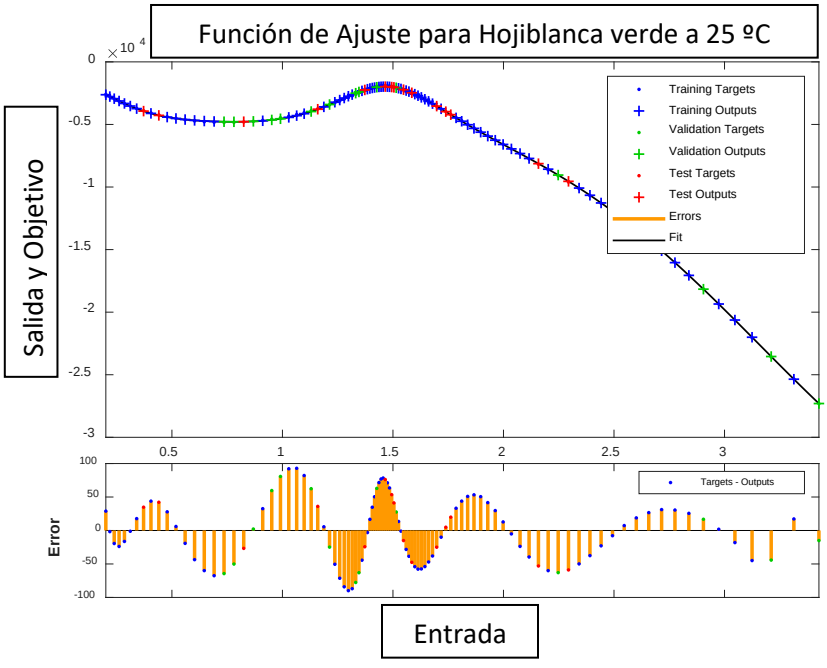


Figura 10. “Hojiblanca” verde a 25 °C.

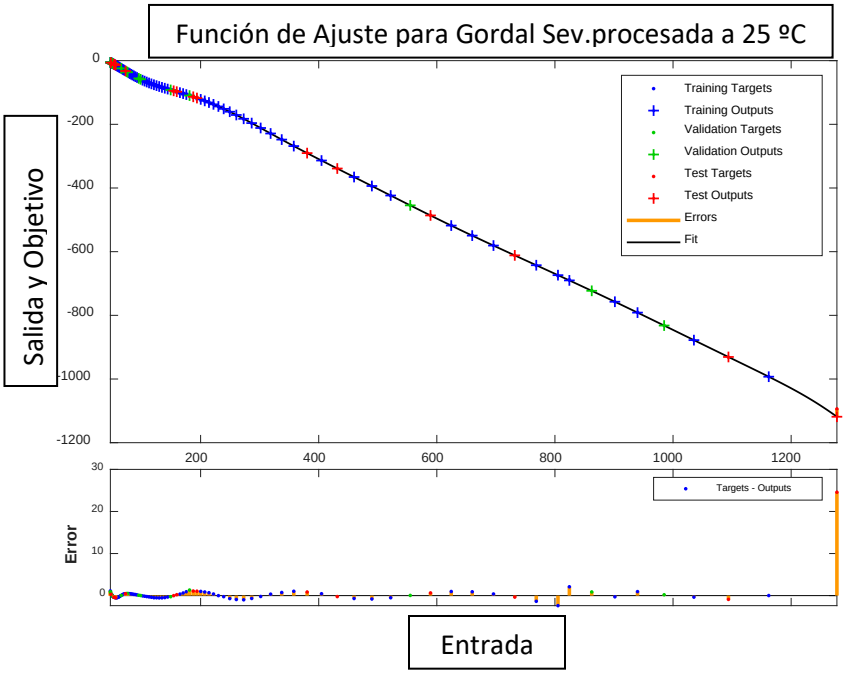


Figura 11. “Gordal Sevillana” procesada a 25 °C.

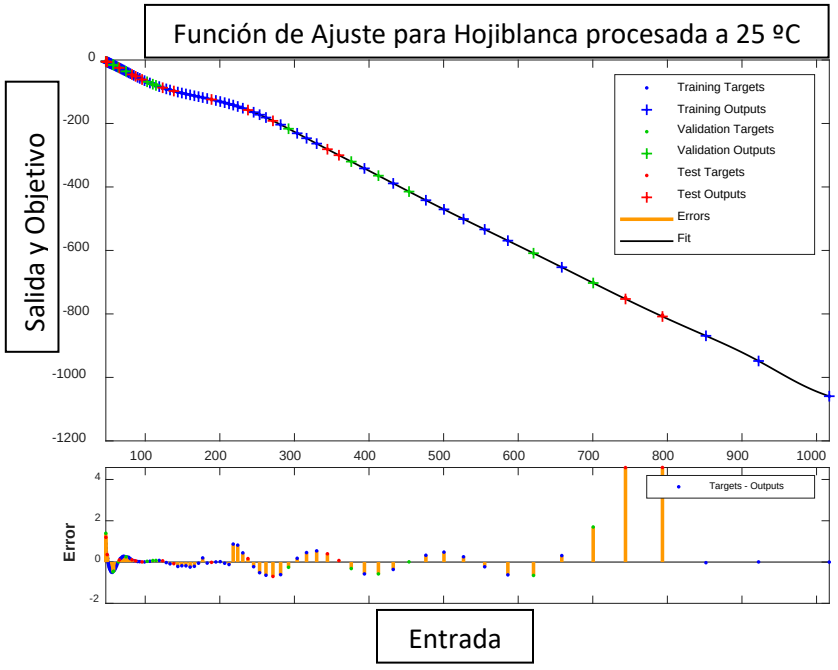


Figura 12. “Hojiblanca” procesada a 25 °C.

Tabl*a* 2. Bondad del ajuste en aceitunas verdes según la variedad y la temperatura

Variedad	Temperatura	P	Error Relativo (%)
"Gordal Sev."	0 °C	7.3753×10^3	1.33%
	7 °C	788.9215	0.08%
	25 °C	1.2965×10^3	0.33%
"Hojiblanca"	0 °C	1.0014×10^3	0.18%
	5 °C	294.0392	0.05%
	20 °C	2.1827×10^3	0.36%

Tabl*a* 3. Bondad del ajuste en aceitunas procesadas según la variedad y la temperatura.

Variedad	Temperatura	P	Error Relativo (%)
"Gordal Sev."	0 °C	1.6073×10^3	1.14%
	7 °C	1.885×10^3	0.34%
	25 °C	6.3865	1.27%
"Hojiblanca"	0 °C	1.4252×10^3	0.25%
	7 °C	6.321×10^3	1.14%
	25 °C	2.218×10^3	0.4%

3.7. Clasificación con redes neuronales según variedad y temperatura

Distinguir entre variedades, en verde o procesadas (en este caso, al "Estilo Sevillano"), puede ser de interés como se menciona en el apartado 2.3 para diferenciar entre variedades en caso de duda. En esta sección, la clasificación se realiza utilizando la Toolbox de Matlab y, específicamente, su clasificador neuronal "patternnet" [33].

En todos los casos, los ajustes se han realizado utilizando solo una capa oculta de 8 neuronas, 2 variedades y 3 temperaturas, para que así el clasificador distinguiera entre 6 opciones posibles.

Realizar el proceso de clasificación a 3 temperaturas permite generalizarlo dado que las condiciones de temperatura en la finca y en la fábrica pueden ser diferentes a lo largo de la jornada de trabajo o la época del año en que se realiza la prueba. Esto es de especial interés en la industria donde las aceitunas se encuentran generalmente a estas tres temperaturas: temperatura ambiente durante el almacenamiento de aproximadamente 25 °C en promedio, a 7 °C antes del deshuesado / relleno / rodajado y a 0 °C durante el deshuesado / relleno / rodajado.

3.7.1. Clasificación con redes neuronales para aceitunas verdes

El clasificador fue entrenado con 10 muestras de cada estado y 5 desafíos, es decir 60 muestras en total para entrenamiento y 30 para probar el clasificador, los resultados se muestran en la Figura 13. La red neuronal fue capaz de clasificar correctamente el 100% de los casos de aceitunas en verde.

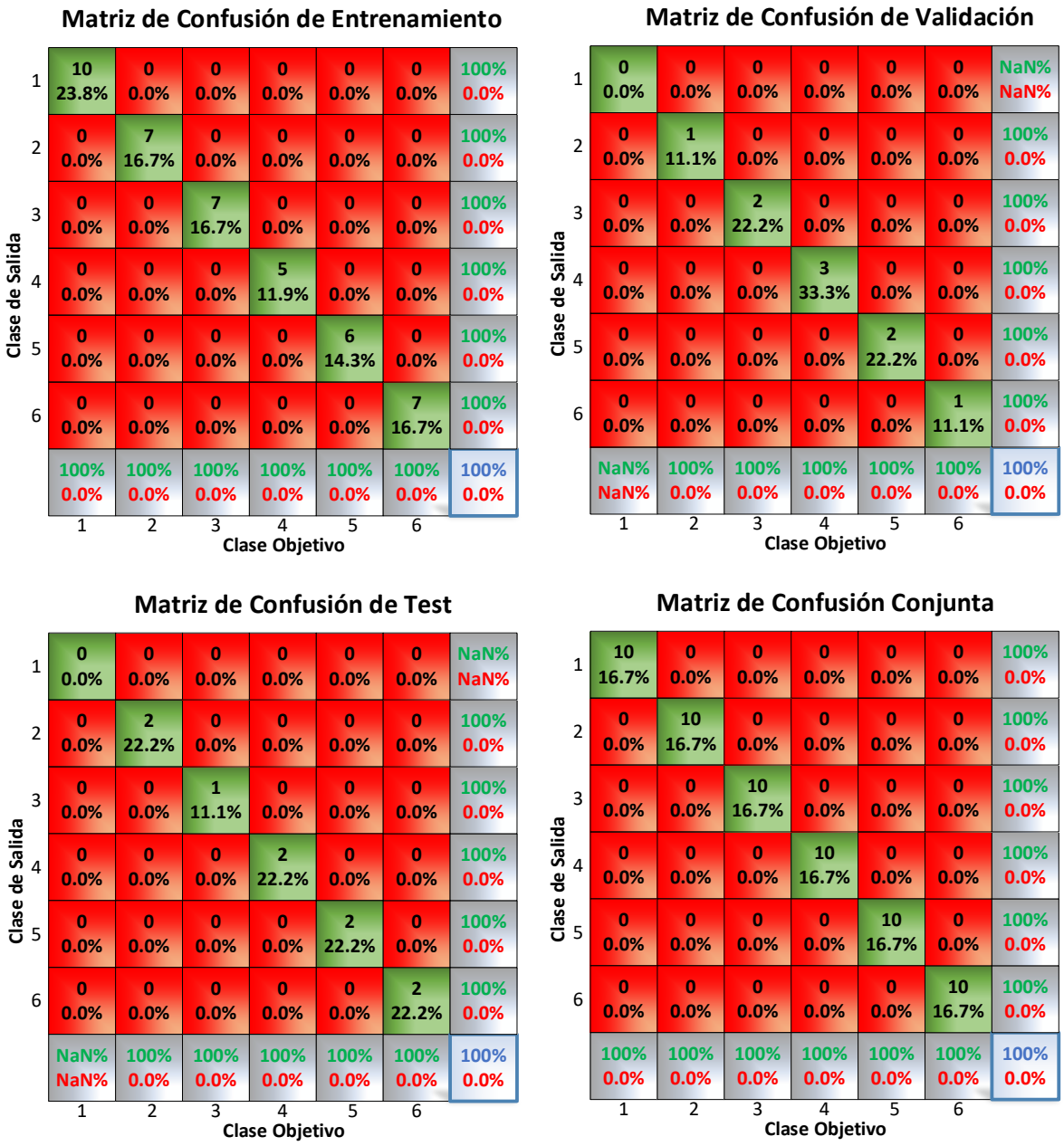


Figura 13. Matriz de confusión para el clasificado de aceitunas verdes.

3.7.2. Clasificación con redes neuronales para aceitunas procesadas al “Estilo sevillano”

Nuevamente el clasificador es entrenado con10 muestras de cada estado y 5 desafíos, es decir, 60 muestras en total para entrenamiento y 30 para probar el clasificador, los resultados se muestran en la Figura 14. La red neuronal clasificó correctamente más del 98% de casos.

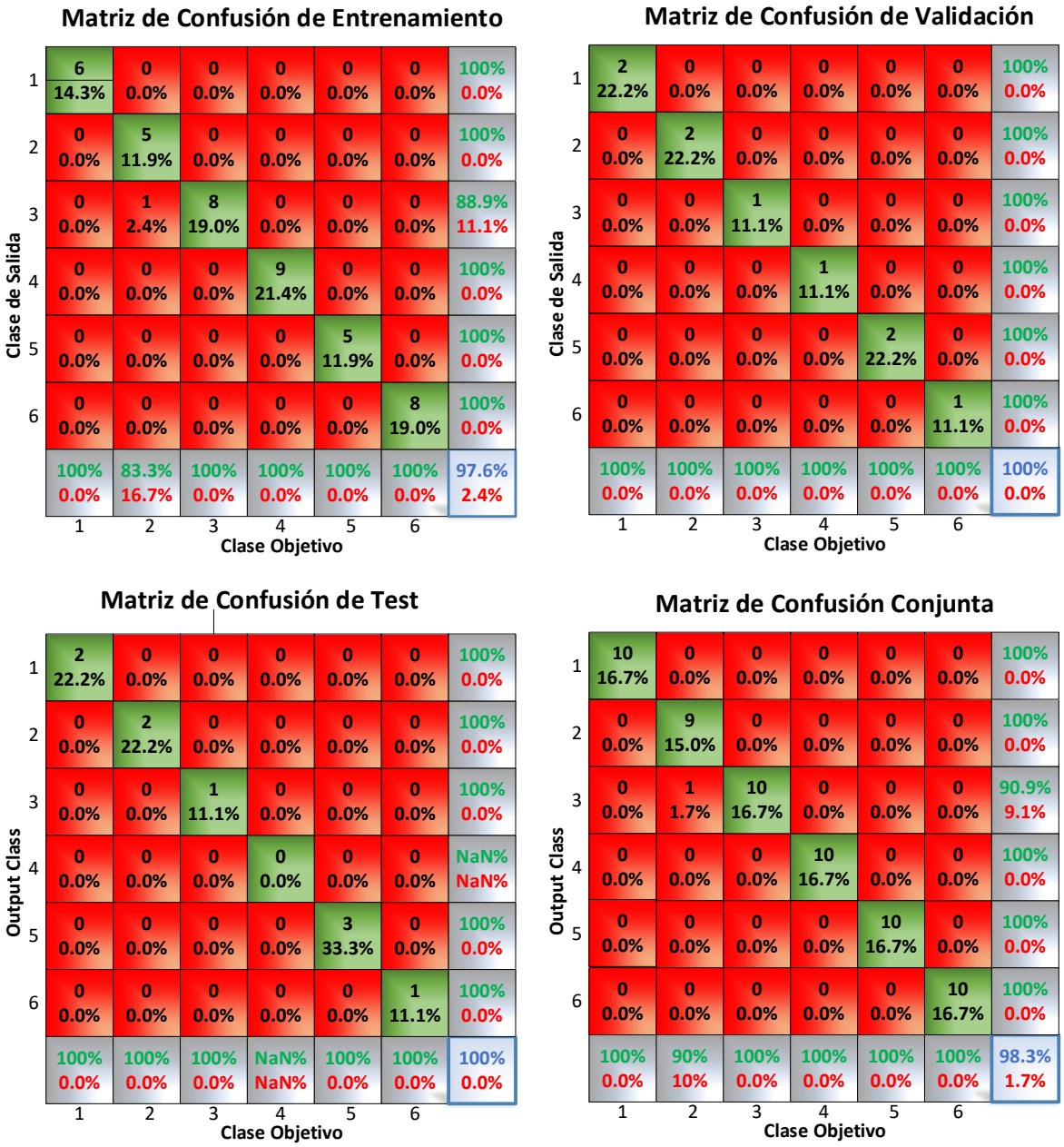


Figura 14. Matriz de confusión para el clasificado de aceitunas procesadas al “Estilo Sevillano

3.8. Internet de las cosas (IoT)

Con el objetivo de evaluar el prototipo para las pruebas de campo y fábrica, se eligió un sistema IoT simple basado en el uso de un ordenador portátil que ejecuta una aplicación escrita en Matlab y envía el archivo con los resultados a una carpeta compartida de Dropbox. De esta forma, las muestras se pueden analizar de forma remota y desde cualquier otro ordenador, como se muestra en la Figura 15.

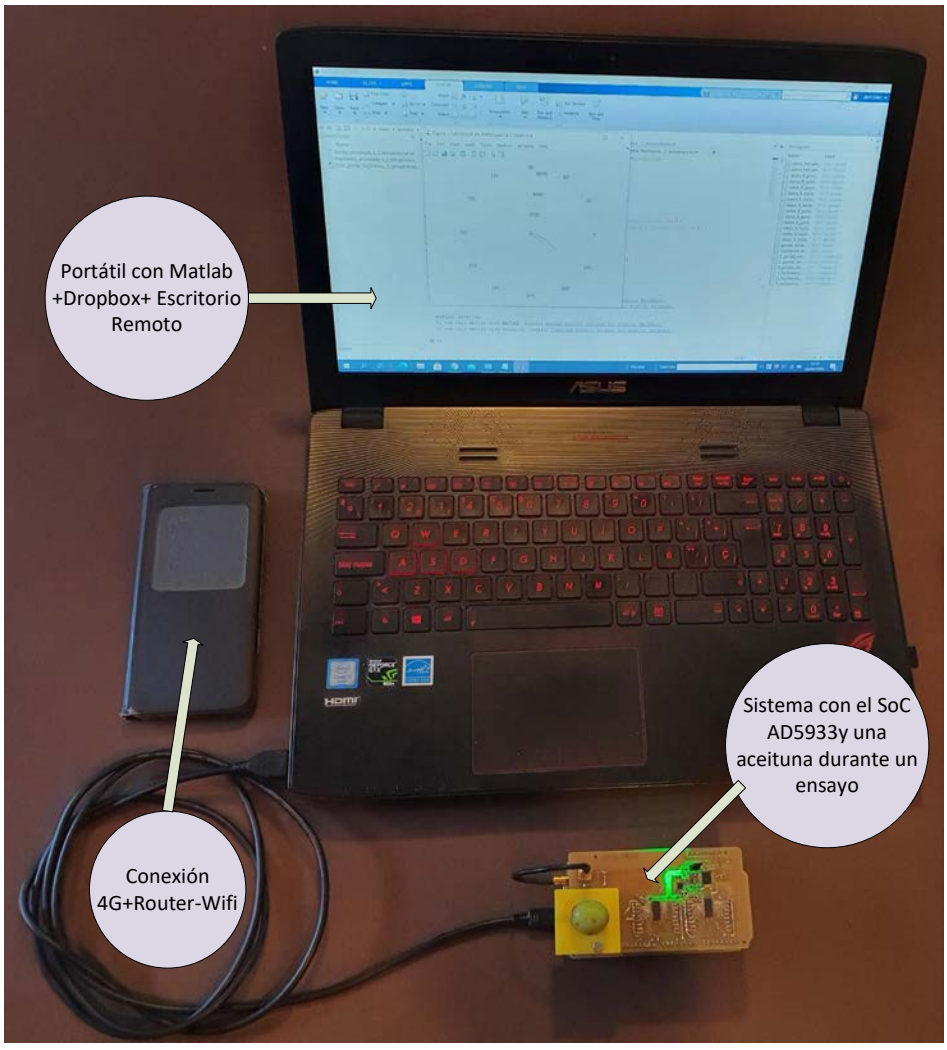


Figura 15. Sistema IoT durante el test en fábrica.

4. Conclusiones, puntos débiles y líneas de trabajo futuras

Con este trabajo se ha podido comprobar que:

• Tanto la aceituna en verde como la procesada presentan un perfil de impedancia equivalente a un modelo R-C sin componente inductivo con una fase en torno a 330 ° en aceitunas verdes y 310 ° en aceitunas en salmuera.

• Con aceitunas verdes y procesadas, se puede observar un perfil de impedancia característico para cada variedad a cada temperatura. Es a baja frecuencia donde las diferencias se acentúan más.

• Con aceitunas verdes en alta frecuencia, los mínimos relativos observados son similares a los descritos por modelos clásicos como el de Hayden, lo que no ocurre con las aceitunas procesadas en salmuera.

• Con las aceitunas procesadas en salmuera, el valor de impedancia de los componentes R y X se reduce en 20 veces, debido al efecto de la salmuera.

• Los modelos desarrollados con redes neuronales como fitnet para representar la evolución de la impedancia permiten obtener un modelo de tipo R versus X con solo 5 neuronas en la capa oculta.

• Se ha comprobado que redes neuronales como patternnet y 8 neuronas en la capa oculta, permiten distinguir entre los 6 casos estudiados (2 variedades y 3 temperaturas) tanto en aceitunas verdes como procesadas.

• El uso de un sencillo sistema IoT basado en Dropbox permite obtener muestras en campo y en la fábrica para su posterior estudio utilizando la combinación de un portátil con conexión 4G y el prototipo desarrollado. Por otro lado, para alcanzar estos resultados se ha optado por el diseño de un hardware propio que nos ha permitido obtener las máximas prestaciones del SoC AD5933:

• Se ha desarrollado un circuito que implementa un SoC AD5933 con todos los elementos periféricos necesarios para su funcionamiento. Este prototipo incluye un par de multiplexores analógicos ADG706 para cubrir todo el rango de impedancia a medir.

• Para lograr la máxima resolución en el DFT, se ha utilizado un DDS basado en un FPGA para generar una señal de reloj programable a voluntad según los límites del barrido de frecuencia a realizar durante la medida de impedancia.

• La programación de la aplicación principal se ha realizado en Matlab. Para controlar todos los elementos se ha utilizado un microcontrolador ARM CORTEX M3 (AT91SAM3X8E) con un Arduino DUE, implementando todo el firmware necesario para controlar el hardware. Por último,

se ha implementado en ASM las rutinas de control del microcontrolador Picoblaze embebido en la FPGA del DDS.

• Las pruebas con resistencia pura y circuitos RLC en serie / paralelo han demostrado que el sistema funciona correctamente.

• Existe una limitación funcional en el chip donde la velocidad del DSP interno es proporcional a la velocidad del reloj aplicada externamente. En estas circunstancias, para la medición de baja frecuencia (de 1 Hz a 30 Hz), se ha utilizado un reloj de 25 kHz que, en comparación con la frecuencia estándar utilizada (16 MHz), hace que el proceso de medición sea 640 veces más lento a bajas frecuencias.

• El circuito construido es experimental y, para poder utilizarlo directamente en las fincas donde se ubica el cultivo, se debe producir una versión capaz de soportar esas condiciones de trabajo. Asimismo, la aplicación debe ser una “app” para teléfono móvil o una tablet, lo que permitiría una conexión a la computadora a través de Bluetooth.

• Una posible opción es implementar todas las rutinas desarrolladeas en un microcontrolador embebido en la FPGA programado directamente en C (por ejemplo, Microblaze).

Finalmente, entre las áreas de trabajo futuras, estamos considerando:

• Aumentar el rango de frecuencia a 25 MHz de forma que superando los 100 kHz actuales constatar si son de aplicación los modelos estándar como el de Hayden.

• Estudiar otras variedades de aceituna de relevancia comercial como la aceituna “Manzanilla” o “Cacereña”.

• Estudiar otros tratamientos industriales como la aceituna negra oxidada “Estilo Californiano”.

• Ejecutar un análisis que correlacione el porcentaje de rotura en máquinas DRR directamente con el valor de impedancia medido de diferentes variedades, procesos y temperaturas. Los estudios futuros incluirán la aplicación de esta metodología en otras frutas como el tomate o la cereza.

Bibliografía

1. Bourne, M.C. Effect of Temperature on Firmness of Raw Fruits and Vegetables. *J. Food Sci.* **1982**, *47*, 440–444, doi:10.1111/j.1365-2621.1982.tb10099.x.
2. Bourne, M.C.; Comstock, S.H. Effect of Temperature on Firmness of Thermally Processed Fruits and Vegetables. *J. Food Sci.* **1986**, *51*, 531–533, doi:10.1111/j.1365-2621.1986.tb11179.x.
3. Bourne, M.C. *Food Texture and Viscosity: Concept and Measurement*; Academic Press:, Harcourt place, 32 James Road, London NW1 7BY, UK 2002; ISBN 9780080491332.
4. Kılıçkan, A.; Güner, M. Physical properties and mechanical behavior of olive fruits (*Olea europaea* L.) under compression loading. *J. Food Eng.* **2008**, *87*, 222–228, doi:10.1016/j.jfoodeng.2007.11.028.
5. Gutierrez Rubio, J. *Maquina Enfriadora de Aceitunas*; ES 1 003 639 U, 1988. Available online: http://www.oepm.es/pdf/ES/0000/000/01/00/36/ES-1003639_U.pdf (accessed on 15 September 2020).
6. Leiva, D.; Tapia, F. Elaboración de Aceitunas Con Bajo Contenido de Sodio. In *Producción de Aceitunas con Bajo Contenido de Sodio ("Light")*; Instituto de Investigaciones Agropecuarias, Centro Regional de Investigación Intihuasi: La Serena, Chile, 2015. Available online: http://biblioteca.inia.cl/medios/biblioteca/boletines/NR40474.pdf. (accessed on 13 September 2020)
7. Gómez, A.H.S.; García, P.; Navarro, L.R. Elaboration of table olives. *Grasas Aceites* **2006**, *57*, 86–94.
8. Hlaváčová, Z. Low frequency electric properties utilization in agriculture and food treatment. *Res. Agric. Eng.* **2003**, *49*, 125–136, doi:10.17221/4963-RAE.
9. Mitchell, F.R.G.; Alwis, A.A.P.D. Electrical conductivity meter for food samples. *J. Phys. E* **1989**, *22*, 554–556, doi:10.1088/0022-3735/22/8/004.
10. Nelson, S.O. Dielectric properties of agricultural products-measurements and applications. *IEEE Trans. Electr. Insul.* **1991**, *26*, 845–869, doi:10.1109/14.99097.
11. Repo, T.; Paine, D.H.; Taylor, A.G. Electrical impedance spectroscopy in relation to seed viability and moisture content in snap bean (*Phaseolus vulgaris* L.). *Seed Sci. Res.* **2002**, *12*, 17–29, doi:10.1079/SSR200194.
12. *AD5933*; Analog Devices: Norwood, MA, USA, 2005. Available online: http://www.analog.com/media/en/technical-documentation/data-sheets/AD5933.pdf. (accessed on 15 September 2020).
13. Rodríguez Gómez, R.; Cruz Hurtado, J. Sistema de medición y análisis de impedancia. *Ing. Electrónica Automática Comun.* **2015**, *36*, 56–66.
14. Okada, K.; Sekino, T. *Agilent Impedance Measurement Handbook;* A guide to measurement technology and techniques; Agilent Technologies: Santa Clara, CA, USA, 2009.
15. Tegam. The LCR Meter as an Impedance Analyzer. Available online: http://www.tegam.com/wp-content/uploads/2015/10/AN303.pdf. (accessed on 15 September 2020).
16. Keysight 4395A Network/Spectrum/Impedance Analyzer. Available online: http://www.keysight.com/en/pd-1000000864%3Aepsg%3Apro-pn-4395A/network-%0Aspectrum-impedance-analyzer?cc=ES&lc=eng (accessed on 15 September 2020).
17. Keysight 4194A Impedance/Gain-Phase Analyzer. Available online: https://www.keysight.com/en/pd-1000003398%3Aepsg%3Apro-pn-4194A/impedance-gain-phase-analyzer?cc=ES&lc=eng. (accessed on 05 September 2020).
18. Juping, G.; Long, J.; Shenbei, Q.; Xinjian, W.; Zhike, X. Researching on the automatic impedance measurement system. In Proceedings of the Eighth International Conference on Electrical Machines and Systems, Nanjing, China, 27–29 September 2005.
19. Ingenieria Eléctrica Fravedsa. Available online: http://ingenieriaelectricafravedsa.blogspot.com.es/2014/11/puente-schering.html. (accessed on 15 September 2020).

20. Wikipedia. Maxwell Bridge. Available online: https://en.wikipedia.org/wiki/Maxwell_bridge. (accessed on 18 September 2020).
21. Ibrahim, K.M.; Abdul-Karim, M.A.H. Digital Impedance Measurement by Generating Two Waves. *IEEE Trans. Instrum. Meas.* **1985**, *IM-34*, 2–5, doi:10.1109/TIM.1985.4315245.
22. Taha, S.M.R. Digital measurement of the polar and rectangular forms of impedances. *IEEE Trans. Instrum. Meas.* **1989**, *38*, 59–63, doi:10.1109/19.19999.
23. Steber, G.R. A Low Cost RF Impedance Analyzer. *Nuts and Volts*, February 2008, pp. 38–41. Available online: https://www.nutsvolts.com/magazine/article/a_low_cost_rf_impedance_analyzer (accessed on 15 September 2020).
24. Castelló, J.; Espí, J.; García, R.; Esteve, V. Analizador de Impedancia/Ganancia-Fase para PC. *Revista Española de Electrónica*, 2001, pp. 70–75.
25. Justicia, M.; Madueño, A.; Ruiz-Canales, A.; Molina, J.M.; López, M.; Madueño, J.M.; Granados, J.A. Low-frequency characterisation of mesocarp electrical conductivity in different varieties of olives (*Olea europaea* L.). *Comput. Electron. Agric.* **2017**, *142*, 338–347, doi:10.1016/j.compag.2017.09.021.
26. Weaver, G.M.; Jackson, H.O. Electric impedance, an objective index of maturity in peach. *Can. J. Plant Sci.* **1966**, *46*, 323–326.
27. Ezeike, G.O.I. A resistive probe moisture sensor for tropical root crops and vegetables. *J. Agric. Eng. Res.* **1987**, *37*, 15–26, doi:10.1016/0021-8634(87)90128-4.
28. Montoya Lirola, M. Estudio de la Conductividad Eléctrica Como Índice de Madurez en Frutos Climatéricos y su Evolución Durante la Conservación Frigorífica en Atmosfera Normal y Modificada. Ph.D. Thesis, UNED, Madrid, Spain, 1992. Available online: https://dialnet.unirioja.es/servlet/tesis?codigo=40951 (accessed on 15 September 2020).
29. Van Gerven, M.; Bohte, S. Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Front. Comput. Neurosci.* **2017**, *11*, doi:10.3389/fncom.2017.00114.
30. De Jódar Lázaro, M.; Madueño Luna, A.; Lucas Pascual, A.; Martínez, J.M.M.; Canales, A.R.; Madueño Luna, J.M.; Segovia, M.J.; Sánchez, M.B. Deep learning in olive pitting machines by computer vision. *Comput. Electron. Agric.* **2020**, *171*, 105304, doi:10.1016/j.compag.2020.105304.
31. Lucas Pascual, A.; Madueño Luna, A.; de Jódar Lázaro, M.; Molina Martínez, J.M.; Ruiz Canales, A.; Madueño Luna, J.M.; Justicia Segovia, M. Analysis of the Functionality of the Feed Chain in Olive Pitting, Slicing and Stuffing Machines by IoT, Computer Vision and Neural Network Diagnosis. *Sensors* **2020**, *20*, 1541, doi:10.3390/s20051541.
32. The MathWorks Inc. Function Fitting Neural Network (Fitnet). Available online: https://es.mathworks.com/help/deeplearning/ref/fitnet.html. (accessed on 15 September 2020).
33. The MathWorks Inc. Pattern Recognition Network (Patternnet). Available online: https://es.mathworks.com/help/deeplearning/ref/patternnet.html;jsessionid=370562d44f3c46b93a717f92677f. (accessed on 18 September 2020).
34. Lougheed, E.C.; Miller, S.R.; Ripley, B.D.; Cline, R.A. Electrical impedance of daminozide- and calcium-treated McIntosh apples. *Experientia* **1981**, *37*, 835–837, doi:10.1007/BF01985666.
35. Jackson, P.J.; Harker, F.R. Apple Bruise Detection by Electrical Impedance Measurement. *HortScience* **2000**, *35*, 104–107, doi:10.21273/HORTSCI.35.1.104.
36. Stout, D.G.; Hall, J.W.; McLaughlin, N.B. In vivo plant impedance measurements and characterization of membrane electrical properties: The influence of cold acclimation. *Cryobiology* **1987**, *24*, 148–162, doi:10.1016/0011-2240(87)90017-4.
37. Stout, D.G. Effect of Cold Acclimation on Bulk Tissue Electrical Impedance. *Plant Physiol.* **1988**, *86*, 283–287, doi:10.1104/pp.86.1.283.
38. Bauchot, A.D.; Harker, F.R.; Arnold, W.M. The use of electrical impedance spectroscopy to assess the physiological condition of kiwifruit. *Postharvest Biol. Technol.* **2000**, *18*, 9–18, doi:10.1016/S0925-5214(99)00056-3.
39. NXP Semiconductors. I²C Bus. Available online: http://www.interfacebus.com/Design_Connector_I2C.html (accessed on 14 September 2020).

40. Xilinx Inc. Spartan-3E Fpga Available online: https://www.digikey.es/es/datasheets/xilinxinc/xilinx-inc-ds312. (accessed on 15 September 2015).
41. *ADG706 Analog Multiplexer*; Analog Devices, Norwood, MA, USA, 2005. Available online: https://www.analog.com/media/en/technical-documentation/data-sheets/ADG706_707.pdf. (accessed on 15 September 2020).
42. Microchip ARM Cortex-M3. Available online: https://www.microchip.com/wwwproducts/en/ATSAM3X8E (accessed on 16 September 2020).
43. The MathWorks Inc. Matlab Software. Available online: https://es.mathworks.com/products/matlab.html (accessed on 11 September 2020).
44. Dropbox. Available online: https://www.dropbox.com/. (accessed on 19 September 2020).
45. Rappoport, H. Botánica y Morfología. In *El Cultivo del Olivo*; Mundi-Prensa: Madrid, Spain, 2008; pp. 35–60.
46. Ferreira, J. *Explotaciones Olivareras Colaboradoras*; Number 5; Ministerio de Agricultura: Madrid, Spain, 1979.
47. Garrido, A.; García, P.; Brrenes, M. Olive fermentations. In *Biotechnology: A Multivolume Comprehesive Treatise*; Reed, H.J., Nagodawitana, T.W., Eds.; Wiley-VCH Verlag GmbH, 1995; pp. 593–625.
48. Estrada, J.M. *La Aceituna de Mesa: Nociones Sobre sus Características,Elaboración y Cualidades*; Fundacion Para El Fomento y Promocion de la Aceituna de Mesa: Sevilla, Spain; Diputación de Sevilla: Sevilla, Spain, 2011. Available online: http://www.besana.es/sites/default/files/libroaceitunamaqueta080411.pdf. (accessed on 15 September 2020).
49. Garrido, A.; García, P.; López, A.; Arroyo, F.N. *Processing Technology in Olive Oil and Table Olive*. International Olive Council, Madrid, 2006. Available online: https://pdfs.semanticscholar.org/5999/f039244e30eda8538c20a2f2b47092c4f55e.pdf. (accessed on 17 September 2020).
50. Stockham, T.G. High-speed convolution and correlation. In *Proceedings of the Spring Joint Computer Conference on XX—AFIPS '66 (Spring),* Boston, MA. USA, *April 26–28 1966*; ACM Press: New York, NY, USA, 1966; p. 229.
51. Chen, C.J.; Liu, J.T.; Chang, S.J.; Lee, M.W.; Tsai, J.Z. Development of a portable impedance detection system for monitoring the growth of mouse L929 cells. *J. Taiwan Inst. Chem. Eng.* **2012**, *43*, 678–684, doi:10.1016/j.jtice.2012.04.008.
52. Schwarzenberger, T.; Wolf, P.; Brischwein, M.; Kleinhans, R.; Demmel, F.; Lechner, A.; Becker, B.; Wolf, B. Impedance sensor technology for cell-based assays in the framework of a high-content screening system. *Physiol. Meas.* **2011**, *32*, 977–993, doi:10.1088/0967-3334/32/7/S18.
53. Wang, M.H.; Kao, M.F.; Jang, L.S. Single HeLa and MCF-7 cell measurement using minimized impedance spectroscopy and microfluidic device. *Rev. Sci. Instrum.* **2011**, *82*, 064302, doi:10.1063/1.3594550.
54. Helen Berney, H.; O'Riordan, J.J. Impedance Measurement Monitors Blood Coagulation. *Analog Dialogue* **2008**, *42*, 2–4.
55. Broeders, J.; Duchateau, S.; van Grinsven, B.; Vanaken, W.; Peeters, M.; Cleij, T.; Thoelen, R.; Wagner, P.; de Ceuninck, W. Miniaturised eight-channel impedance spectroscopy unit as sensor platform for biosensor applications. *Phys. Status Solidi A* **2011**, *208*, 1357–1363, doi:10.1002/pssa.201001199.
56. Seoane, F.; Ferreira, J.; Sanchéz, J.J.; Bragós, R. An analog front-end enables electrical impedance spectroscopy system on-chip for biomedical applications. *Physiol. Meas.* **2008**, *29*, S267–S278, doi:10.1088/0967-3334/29/6/S23.
57. Bogónez-Franco, P.; Bayés-Genís, A.; Rosell, J.; Bragós, R. Performance of an implantable impedance spectroscopy monitor using ZigBee. *J. Phys. Conf. Ser.* **2010**, *224*, 012163, doi:10.1088/1742-6596/224/1/012163.
58. Ferreira, J.; Seoane, F.; Lindecrantz, K. AD5933-based electrical bioimpedance spectrometer. Towards textile-enabled applications. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Boston, MA, USA, 30 August–3 September 2011*; IEEE: Piscataway, NY, USA, 2011; pp. 3282–3285.

59. Margo, C.; Katrib, J.; Nadi, M.; Rouane, A. A four-electrode low frequency impedance spectroscopy measurement system using the AD5933 measurement chip. *Physiol. Meas.* **2013**, *34*, 391–405, doi:10.1088/0967-3334/34/4/391.
60. Melwin, A.; Rajasekaran, K. Implementation of Bioimpedance Instrument Kit in ARM7. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2013**, *3,5*, 1271–1273.
61. Hoja, J.; Lentka, G. Interface circuit for impedance sensors using two specialized single-chip microsystems. *Sens. Actuators A Phys.* **2010**, *163*, 191–197, doi:10.1016/j.sna.2010.08.002.
62. Hoja, J.; Lentka, G. A Family of New Generation Miniaturized Impedance Analyzers for Technical Object Diagnostics. *Metrol. Meas. Syst.* **2013**, *20*, 43–52, doi:10.2478/mms-2013-0004.
63. Grimnes, S.; Martinsen, O.G. *Bioimpedance and Bioelectricity Basics*, 3rd ed.; Elsevier Academic Press: Amsterdam, 1000AE, Netherlands 2014; ISBN 9780124115330.
64. Chabowski, K.; Piasecki, T.; Dzierka, A.; Nitsch, K. Simple Wide Frequency Range Impedance Meter Based on AD5933 Integrated Circuit. *Metrol. Meas. Syst.* **2015**, *22*, 13–24, doi:10.1515/mms-2015-0006.
65. Simic, M. Realization of Complex Impedance Measurement System Based on the Integrated Circuit AD5933. In *Proceedings of the 21st Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 26–28 November 2012*; IEEE: Piscataway, NY, USA, 2013; pp. 573–576.
66. Simić, M. Complex Impedance Measurement System for the Frequency Range from 5 kHz to 100 kHz. *Key Eng. Mater.* **2015**, *644*, 133–136, doi:10.4028/www.scientific.net/KEM.644.133.
67. Simic, M. Realization of digital LCR meter. In *Proceedings of the International Conference and Exposition on Electrical and Power Engineering (EPE), Iasi, Romania, 16–18 October 2014*; IEEE: Piscataway, NY, USA, 2014; pp. 769–773.
68. Madueño, J.M. Papers Appendix. Available online: https://www.dropbox.com/sh/wg1hmyxdgt558nc/AACPWe1XJi_tYcJoFzVihv3Ya?dl=0 (accessed on 15 September 2020).
69. Hayden, R.I.; Moyse, C.A.; Calder, F.W.; Crawford, D.P.; Fensom, D.S. Electrical Impedance Studies on Potato and Alfalfa Tissue. *J. Exp. Bot.* **1969**, *20*, 177–200, doi:10.1093/jxb/20.2.177.
70. Wu, L.; Ogawa, Y.; Tagawa, A. Electrical impedance spectroscopy analysis of eggplant pulp and effects of drying and freezing-thawing treatments on its impedance characteristics. *J. Food Eng.* **2008**, *87*, 274–280, doi:10.1016/j.jfoodeng.2007.12.003.
71. The MathWorks Inc. Network Performance. Available online: https://es.mathworks.com/help/deeplearning/ref/perform.html (accessed on 14 September 2020).

ANEXO 1: Funcionamiento del SoC AD5933 y hardware desarrollado

2.2.1. El SoC AD5933

En la figura 16, se muestra el diagrama de bloques del SoC AD 5933 [12].

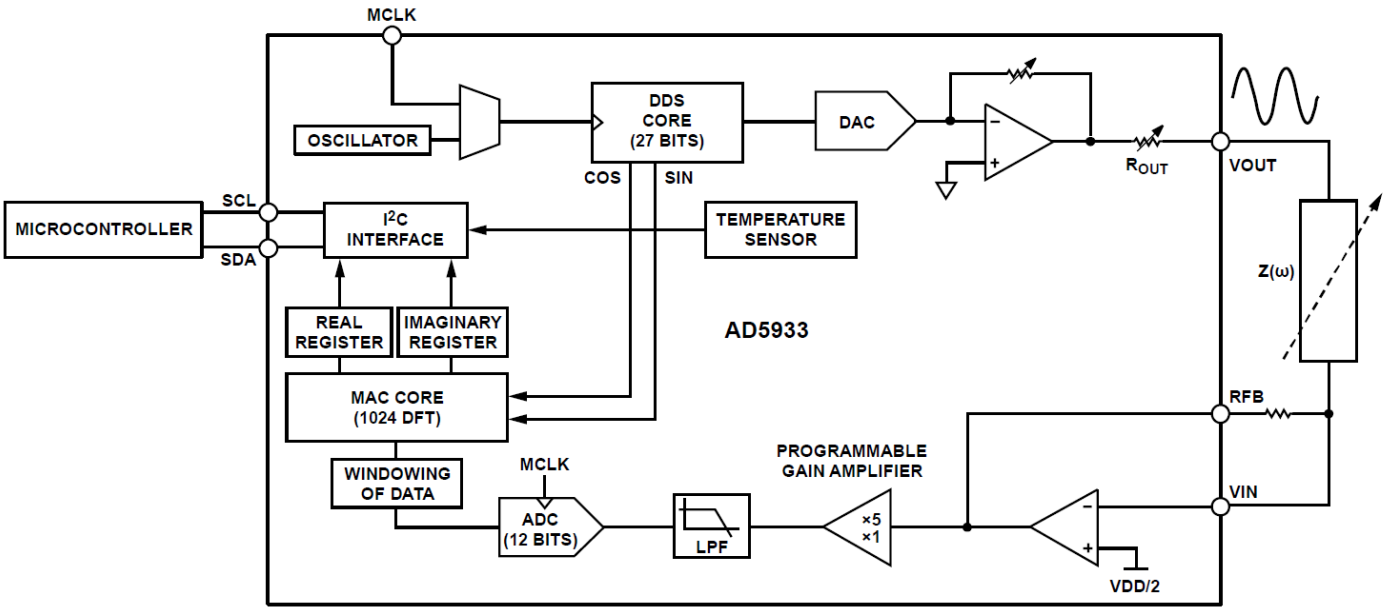


Figura 26 Diagrama de bloques del SoC AD5933 (tomado de [12])

El SoC AD5933 excita la impedancia externa (Z) con una señal de frecuencia conocida producida por un generador senoidal sintetizado interno (DDS). La señal de respuesta de la impedancia es muestreada por el ADC interno lo que permita al DSP efectuar una transformada discreta de Fourier DFT de 1024 pasos. El algoritmo DFT devuelve una palabra de datos real (R) y otra imaginaria (I) en cada punto de frecuencia a lo largo del barrido. La magnitud y la fase de la impedancia se calculan fácilmente utilizando las siguientes ecuaciones:

$$Magnitud = \sqrt{R^2 + I^2} \quad (2)$$

$$Fase = tan^{-1}\left(\frac{I}{R}\right) \quad (3)$$

El SoC AD5933 permite realizar un barrido de frecuencia con una frecuencia de inicio definida por el usuario, resolución de frecuencia y número de puntos en el barrido. Además, el dispositivo permite al usuario programar el valor pico a pico de la señal sinusoidal de salida que sirve de excitación a la impedancia externa desconocida conectada entre los pines VOUT y VIN así como la amplificación posterior de la misma con un amplificador de ganancia programable (PGA).
La etapa de síntesis DDS del chip permite resoluciones por debajo del hertzio. El reloj para el DDS se genera a partir de un reloj de referencia externo, que es proporcionado por el usuario en MCLK,

o por el oscilador interno del chip de 16.776 MHz. Se muestra a continuación con un ejemplo, la forma en la que opera el chip.

2.2.2. Cálculo de la magnitud

Sea R=907 el valor real procedente del cálculo de la DFT, este valor queda almacenado en los registros de dirección 0x94 y 0x95 del chip, e I=516 el valor imaginario, en este caso almacenado en los registros de posición 0x96 y 0x97.

$$Magnitud = \sqrt{R^2 + I^2} = \sqrt{907^2 + 516^2} = 1043{,}506 \quad (4)$$

Para transformar este valor en impedancia es necesario multiplicarlo por un factor de escala llamado factor de ganancia. Este factor de ganancia se debe calcular previamente con un ensayo de calibración empleando una impedancia patrón de valor conocido.

2.2.3. Cálculo del factor de ganancia

Sea:
Tensión de excitación de salida = 2 V p-p
Impedancia patrón= 200 kΩ
Ganancia del PGA = ×1
Resistencia de ajuste de ganancia del amplificador corriente a tensión = 200 kΩ
Frecuencia de calibración = 30 kHz
Supongamos que se obtiene para los registros R e I los siguientes valores:
R=-3996 e I=8830.
Con lo que:

$$Magnitud = \sqrt{R^2 + I^2} = \sqrt{-3996^2 + 8830^2} = 9692.106 \quad (5)$$

$$Factor\ de\ Ganancia = \frac{\frac{1}{Impedancia\ patrón}}{Magnitud} = \frac{\frac{1}{200\ k\Omega}}{9692.106} = 515.819x\ 10^{-12} \quad (6)$$

2.2.4. Cálculo de la impedancia empleando el factor de ganacia

Supongamos que la impedancia desconocida a calcular vale=510 kΩ, y que esta es medida a una frecuencia de 30 kHz, suponiendo también R=-1473 e I=3507

$$Magnitud = \sqrt{R^2 + I^2} = \sqrt{-1473^2 + 3507^2} = 3802.863 \quad (7)$$

El valor de la impedancia se obtiene como:

$$Impedancia = \frac{1}{Factor\ de\ Ganancia\ x\ Magnitud} = \frac{1}{515.819275x10^{-12}x3802.863} = 509.791\ k\Omega \quad (8)$$

2.2.5. Hardware específico desarrollado

Se ha procedido al desarrollo de un hardware específico que permite extraer todo el potencial del SoC AD5933 y por otra parte facilita que el proceso de medida de la impedancia durante un barrido en frecuencia se pueda realizar de manera automática empleando para ello el entorno de Matlab.

2.2.6. Tarjeta con SoC AD5933 y multiplexores analógicos ADG706

Se ha desarrollado una tarjeta (figuras 17 y 18), que además del SoC AD5933, incluye dos multiplexores analógicos ADG706 para, mediante control externo con un microcontrolador AT91SAM3X8E, poder conmutar (entre un conjunto de 15), las resistencias ZBF (tensión/corriente) y Zcal (impedancia patrón), (Tabla 4). Esto permite mejorar la precisión en las medidas de la impedancia bajo test Zin (DUT). El esquema eléctrico de esta tarjeta se encuentra en el Apéndice 2 y [68].

Tabla 4. Valor de las resistencias de calibración (ZBF/ZCAL)

	Resistencias														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valor (KΩ)	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Resistencias de tolerancia 0.1% y coeficiente térmico de 10ppm

El formato que se ha dado a esta tarjeta es el mismo de un Arduino DUE, para poderla ensamblar sobre el mismo, es lo que se conoce como “Shield para Arduino”.

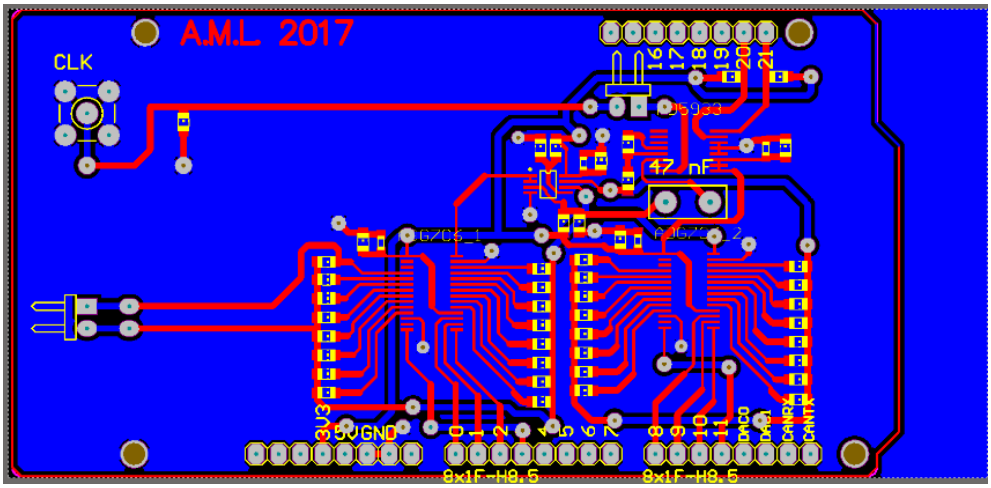


Figura 17. Diseño de la PCB de la tarjeta AD5933 con multiplexores AD706

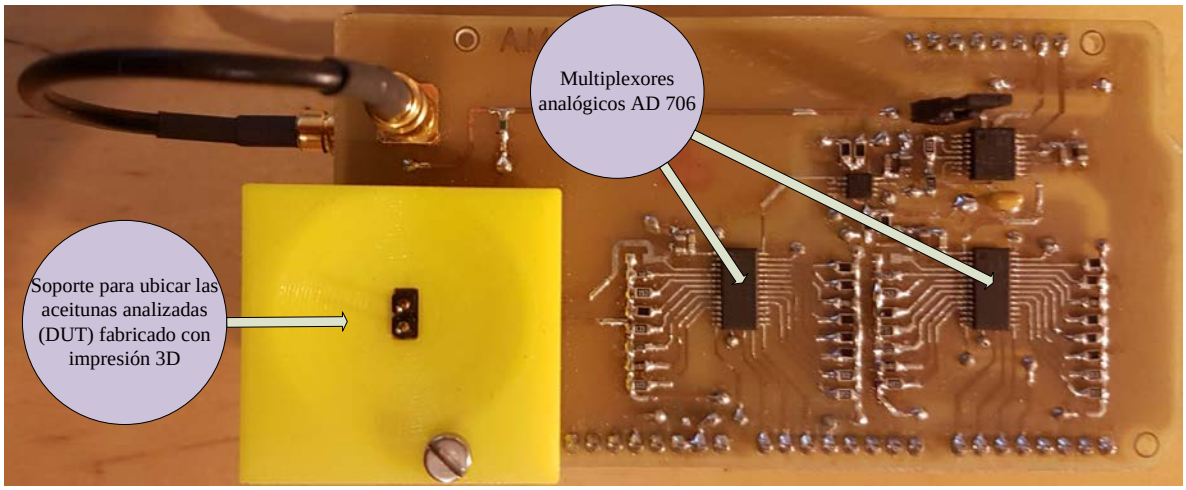


Figura 18. PCB de la Tarjeta con SoC AD5933 y multiplexores AD706

2.2.7. Elección de la fuente de reloj para el SoC AD5933

Una de las cuestiones de diseño que se han planteado es la fuente de reloj necesaria para el funcionamiento del SoC AD5933. Se va a usar un módulo generador de síntesis digital directa (DDS) implementado con una FPGA.

2.2.8. Tarjeta con DDS (Direct Digital Synthesis) basado en una FPGA 3E XC3S250E-4VQG100C de Xilinx

Como se ha indicado, para conseguir la máxima precisión a la hora de realizar la DFT es necesario que el reloj externo que alimenta al módulo DDS del AD5933 tenga una frecuencia dentro de unos rangos apropiados (Tabla 5):

Tabla 5. Límites experimentales para el valor mínimo de frecuencia versus MCLK

Intervalo de Frecuencia	**Frecuencia mínima del AD59333**	**Frecuencia del reloj aplicado al pin MCLK**
1	100 kHz to 5 kHz	16 MHz
2	5 kHz to 1 kHz	4 MHz
3	5 kHz to 300 Hz	2 MHz
4	300 Hz to 200 Hz	1 MHz
5	200 Hz to 100 Hz	250 kHz
6	100 Hz to 30 Hz	100 kHz
7	30 Hz to 20 Hz	50 kHz
8	20 Hz to 10 Hz	25 kHz

Para conseguir esto, se ha diseñado un DDS externo programable (en adelante DDS1), mediante una FPGA Spartan 3E XC3S250E-4VQG100C de Xilinx insertada en un módulo DIP de 40 pines [69], (ver Figura 19), que incluye un microcontrolador para facilitar la programación. El programa (.bin), se carga a través de una micro-SD [68].

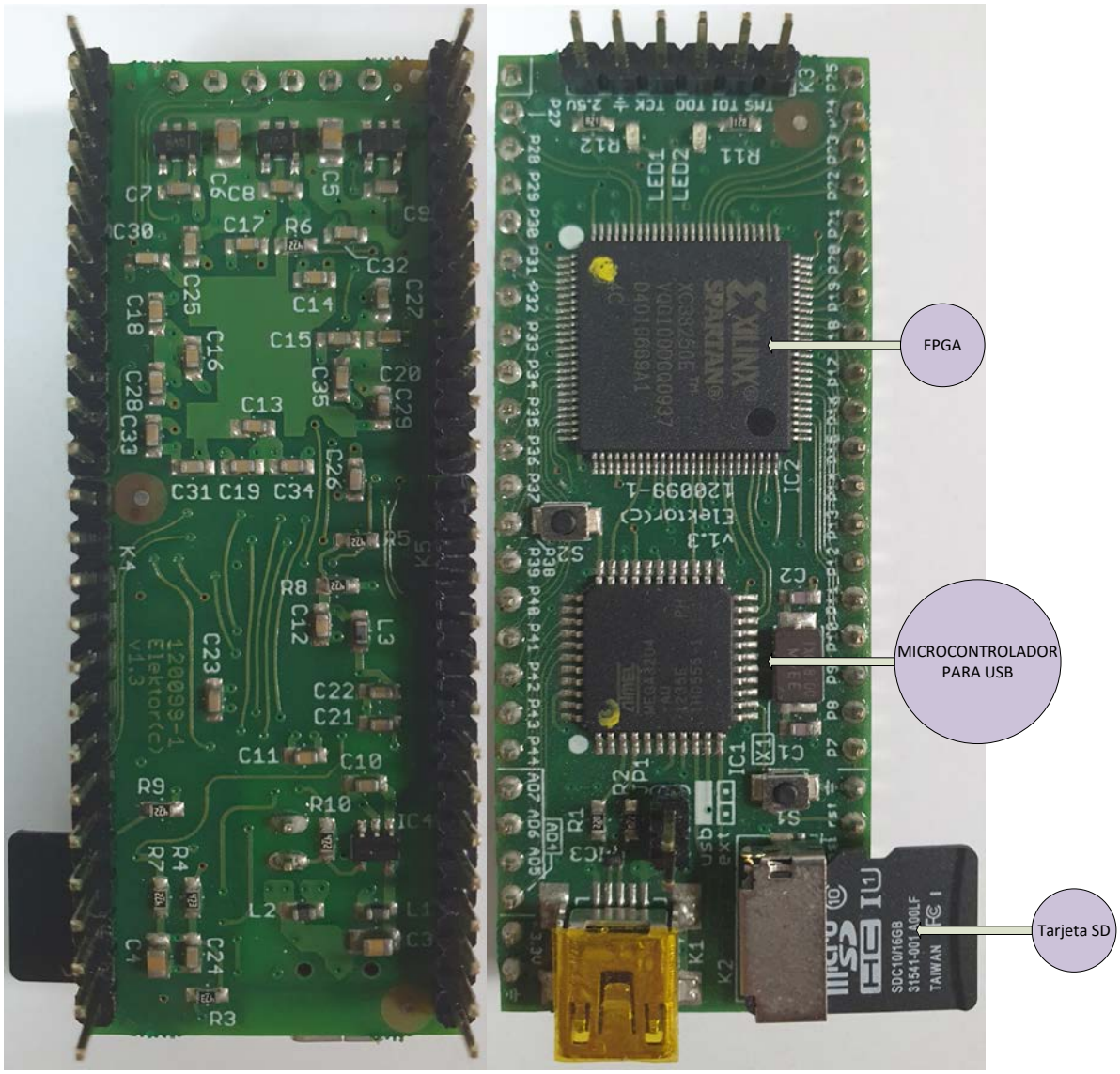


Figura 19. Módulo FPGA con Spartan 3E XC3S250E-4VQG100C de Xilinx

Este DDS1 emplea un PicoBlaze [70], [71], para gestionar todas las funciones necesarias de generación de la señal cuadrada (a niveles de 3.3 Vpp), así como la comunicación con el exterior. Este microcontrolador embebido ha sido programado con el compilador KCPSM3 [72]. Por otra parte el fichero UCF se ha redefinido para poder usar como FPGA una Spartan 3E XC3S250E-4VQG100C.

Para la síntesis VHDL se ha usado Xilinx Ise 9.2i [73]. En la Figura 20 se muestra el circuito desarrollado en formato Arduino DUE, sobre el aparece ensamblado el módulo FPGA con una Spartan 3E XC3S250E-4VQG100C y tarjeta SD.

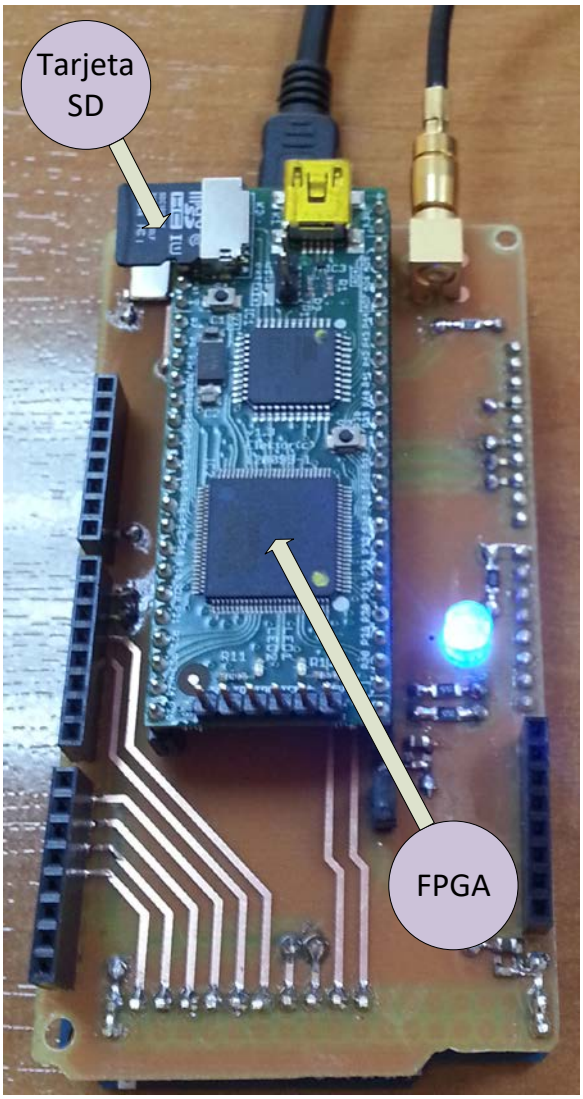


Figura 20. Módulo FPGA con Spartan 3E XC3S250E-4VQG100C en una Shield para Arduino DUE

2.2.9. Arduino DUE para el control del SoC AD5933 y la FPGA

El control del módulo DDS1 y AD5933, ver figura 21, se han encomendado a un microcontrolador ARM Cortex M3, concretamente un AT91SAM3X8E de 32 bits de un Arduino DUE trabajando con un reloj a 84 MHz [68].

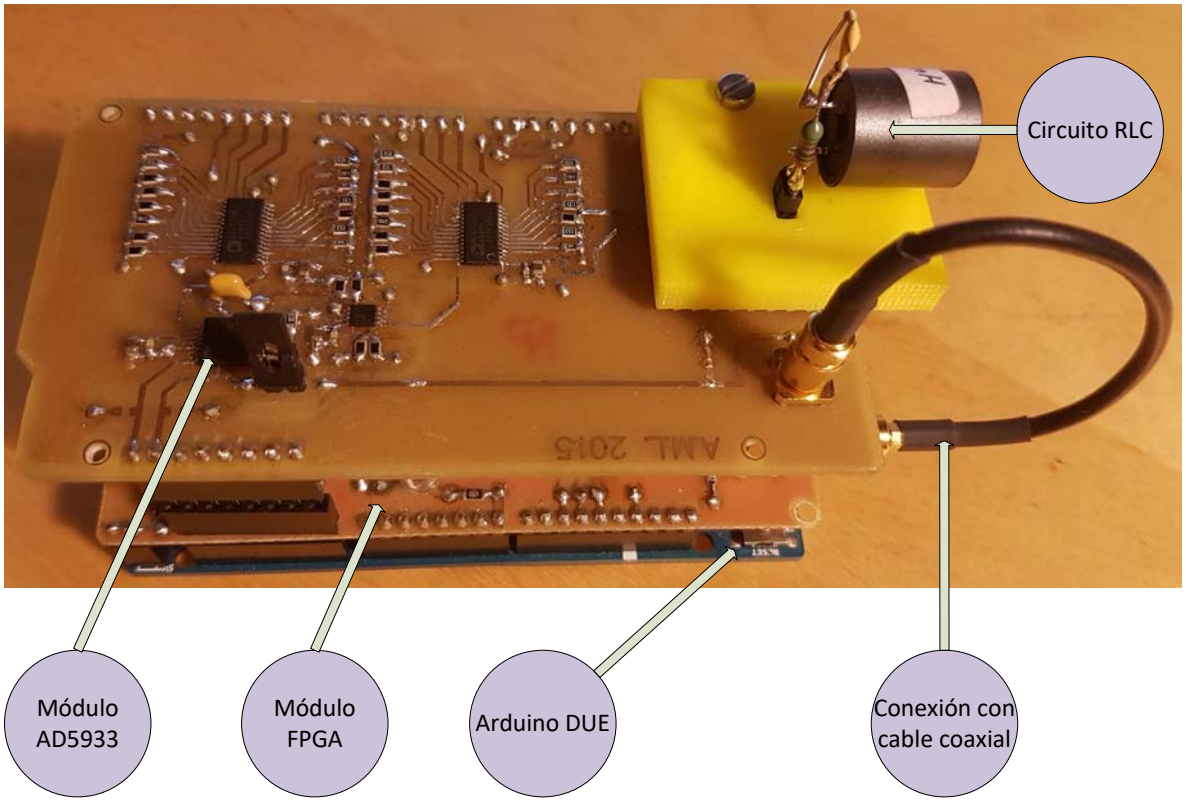


Figura 21. Conjunto ensamblado (El Arduino DUE está en el piso inferior, el modulo con la FPGA en el piso intermedio y el SOC AD5933 en el piso superior), llevando a cabo un barrido de frecuencia en un circuito RLC.

2.2.10. Aplicación en Matlab

Para la gestión del conjunto hardware desarrollado se va a emplear un script escrito para Matlab que permite seleccionar las diferentes funciones a realizar: Selección de rangos del barrido en frecuencia, intervalos en frecuencia entre puntos de medida, configuración del PGA, tiempo de muestreo, etc. [68]

2.2.11. Organigrama de adquisición, calibración y medida e IoT

Para el proceso de adquisición, calibración y medida e IoT se sigue el siguiente procedimiento, (ver Figura 22):

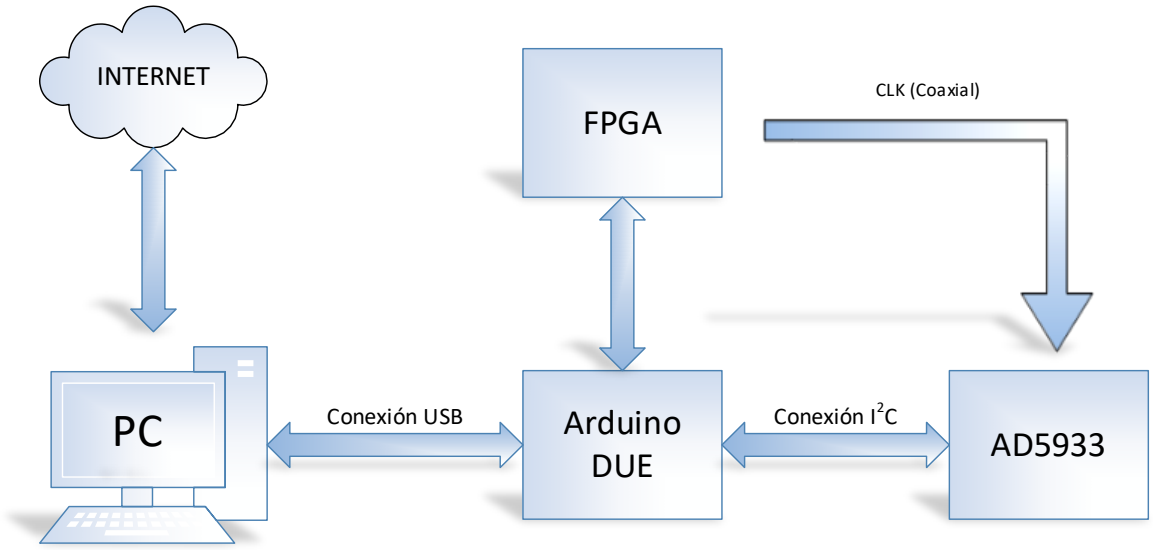


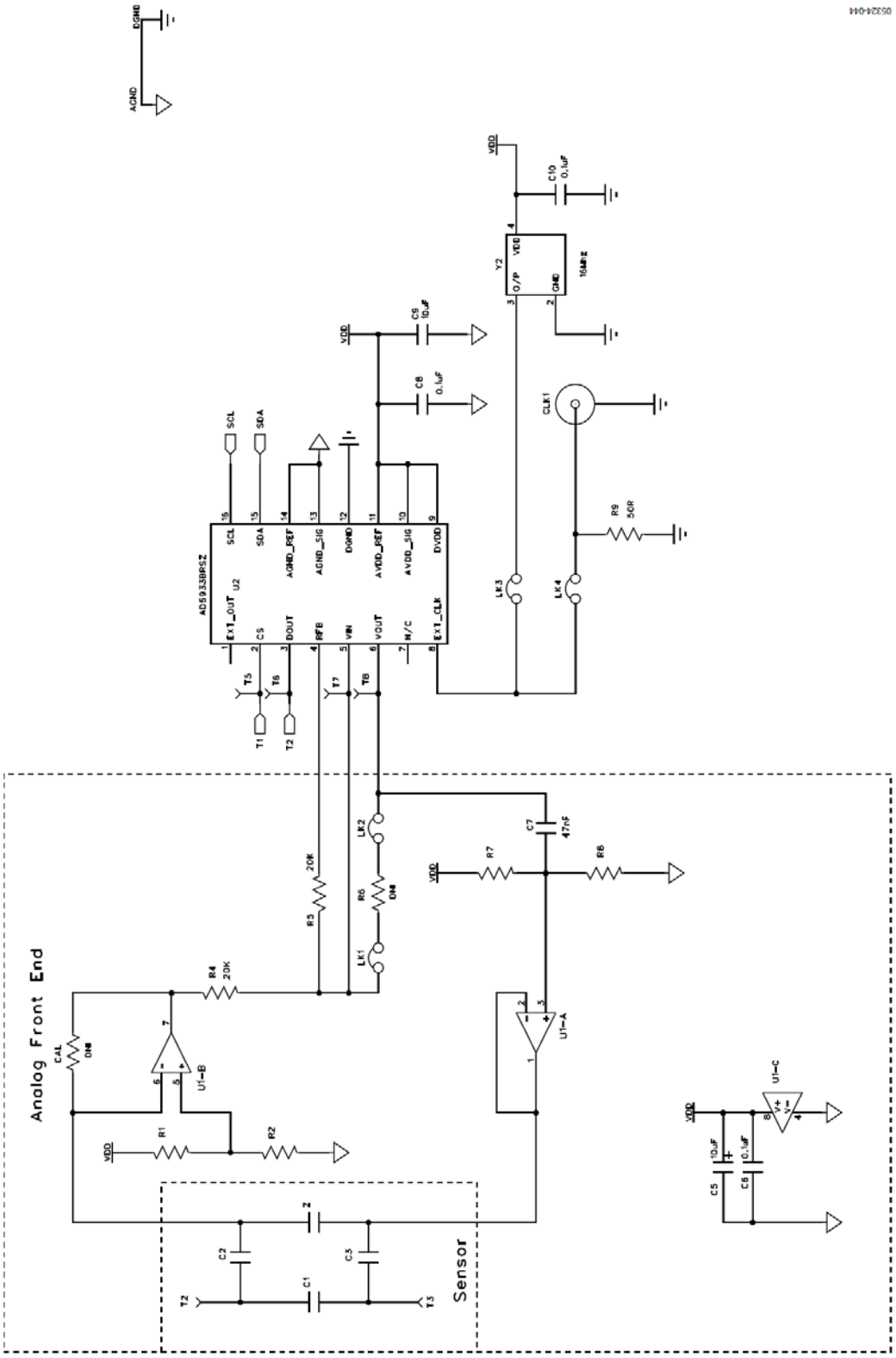
Figura 22. Organigrama de adquisición, calibración, medida e IoT

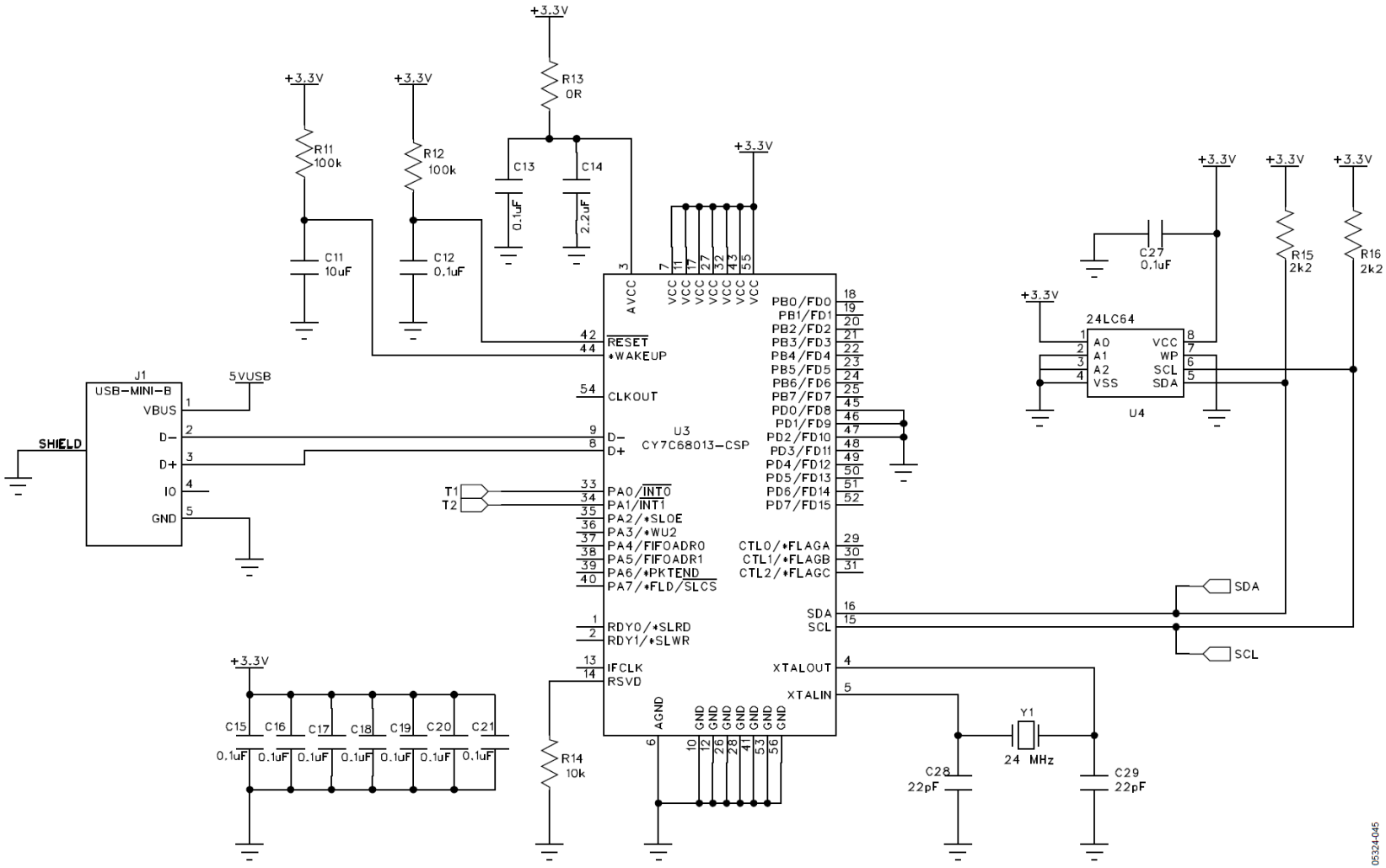
1.-Se fija el valor de ZBF y ZCAL actuando sobre los multiplexores analógicos y acto seguido se realiza el barrido en frecuencia, obteniéndose las componentes real e imaginaria con las que se calcula el módulo mc(i) y la fase ϕc(i) para la calibración. A partir de mc(i), se obtiene el valor del factor de ganancia G(i) (ver apartado 2.2.2.).

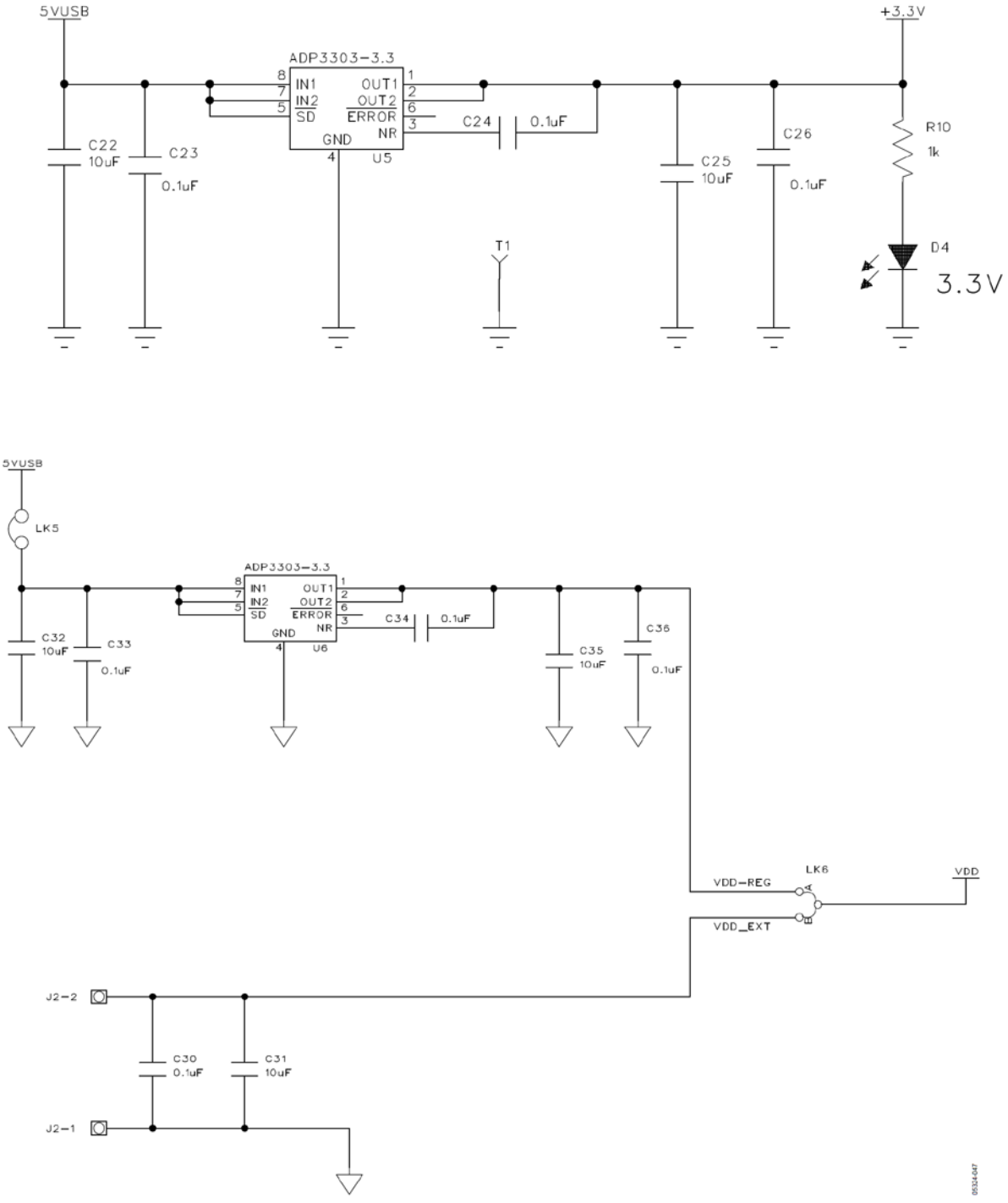
2.-Se repite el proceso, pero en este caso se sustituye la ZCAL por la ZIN obteniéndose para cada frecuencia el módulo ma(i) y la fase ϕa(i) para la medida. Con cada factor de ganancia G(i) y el módulo ma(i), se obtiene el valor del módulo de la impedancia ZIN (i) para cada frecuencia (ver apartado 2.2.2.).
En relación a la fase la fase correspondiente al DUT se obtiene como:

$$\phi_{ZIN} = \phi a(i) - \phi c(i) \quad (9)$$

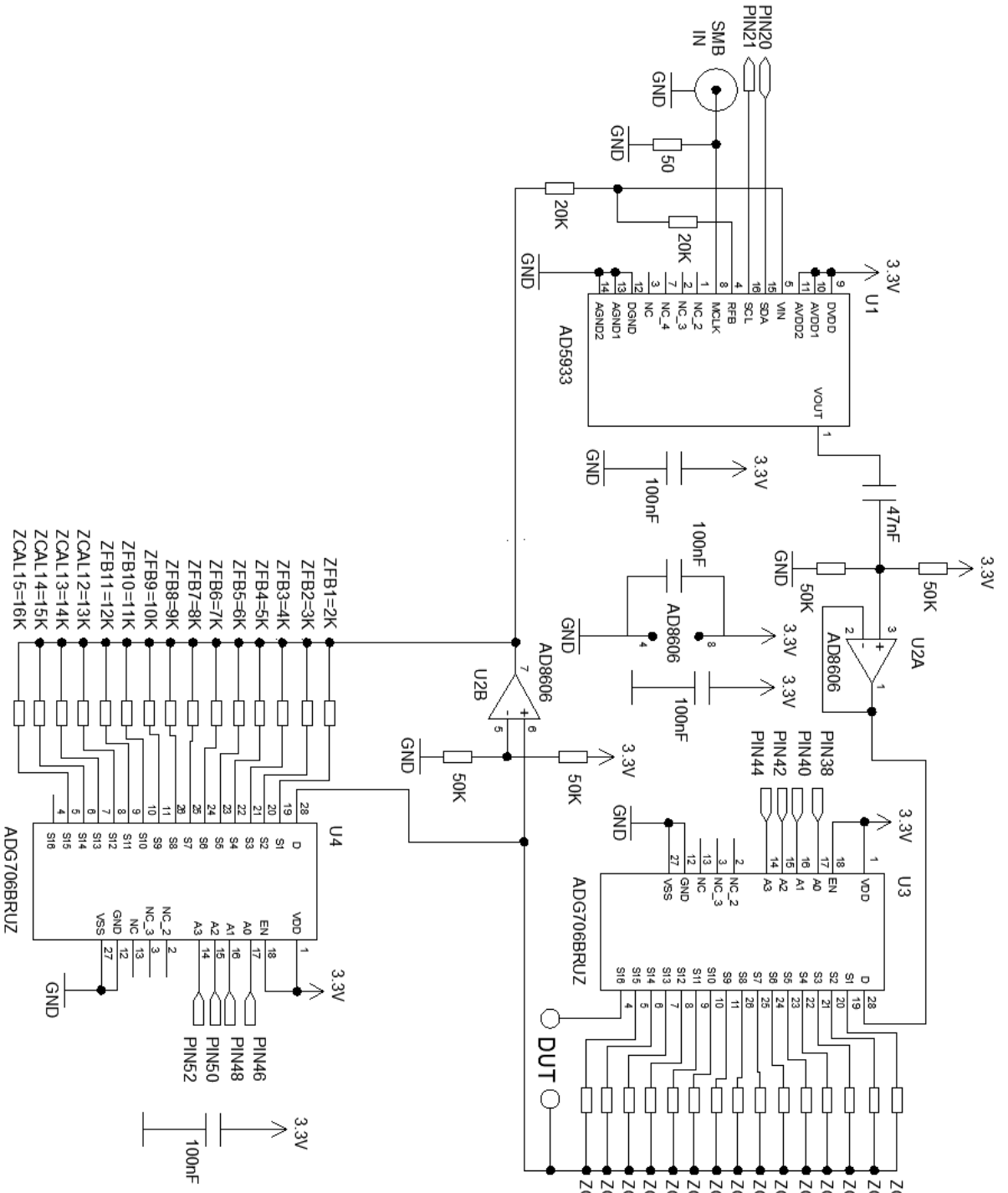
ANEXO 2: Esquemático de la tarjeta de evaluación EVAL-AD5933EBZ (punto de partida para el desarrollo hardware presentado en este trabajo)



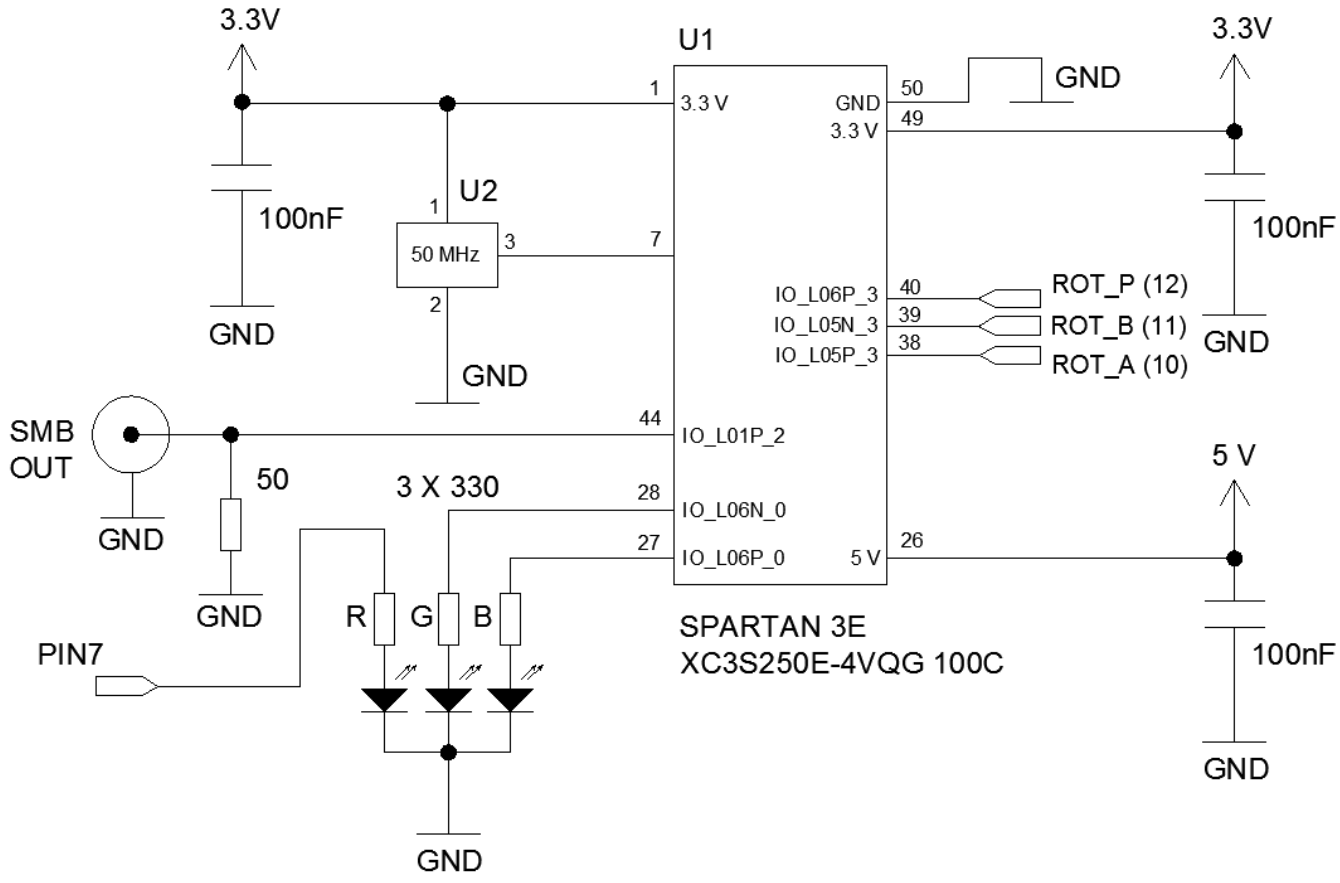

+3.3V
R13
0R
C13
0.1uF
C14
2.2uF
R11
100k
R12
100k
C11
10uF
C12
0.1uF
C27
0.1uF
R15
2k2
R16
2k2
24LC64
U4
A0
A1
A2
VSS
VCC
WP
SCL
SDA
J1
USB-MINI-B
5VUSB
VBUS
D-
D+
ID
GND
SHIELD
U3
CY7C68013-CSP
AVCC
VCC
RESET
*WAKEUP
CLKOUT
PB0/FD0
PB1/FD1
PB2/FD2
PB3/FD3
PB4/FD4
PB5/FD5
PB6/FD6
PB7/FD7
PD0/FD8
PD1/FD9
PD2/FD10
PD3/FD11
PD4/FD12
PD5/FD13
PD6/FD14
PD7/FD15
PA0/INT0
PA1/INT1
PA2/*SLOE
PA3/*WU2
PA4/FIFOADR0
PA5/FIFOADR1
PA6/*PKTEND
PA7/*FLD/SLCS
CTL0/*FLAGA
CTL1/*FLAGB
CTL2/*FLAGC
RDY0/*SLRD
RDY1/*SLWR
IFCLK
RSVD
XTALOUT
XTALIN
AGND
GND
T1
T2
SDA
SCL
C15
C16
C17
C18
C19
C20
C21
0.1uF
R14
10k
Y1
24 MHz
C28
22pF
C29
22pF
05324-045


5VUSB
+3.3V
ADP3303-3.3
IN1
IN2
SD
GND
OUT1
OUT2
ERROR
NR
U5
C22
10uF
C23
0.1uF
C24
0.1uF
C25
10uF
C26
0.1uF
R10
1k
D4
3.3V
T1
5VUSB
LK5
ADP3303-3.3
U6
C32
10uF
C33
0.1uF
C34
0.1uF
C35
10uF
C36
0.1uF
LK6
VDD-REG
VDD_EXT
VDD
J2-2
J2-1
C30
0.1uF
C31
10uF

ANEXO 3: Esquemático de la tarjeta con chip AD5933 y multiplexores analógicos ADG706



ANEXO 4: Esquemático de la tarjeta DDS1 con FPGA



ANEXO 5: Sketch de Arduino

```
// Control de AD5933
// Firmware para el Arduino DUE
// Sevilla 8 de mayo de 2017

#include "Wire.h"

// Nombre de los registros y direcciones fisicas dentro del chip AD5933
#define SLAVE_ADDR 0x0D
#define ADDR_PTR 0xB0

// Nombre de variables asociadas al i2c

byte inData[3];  // Allocate some space for the Bytes
byte inByte;   // Where to store the Bytes read
byte indice = 0;   // Index into array; where to store the Bytes

byte dispositivo=0;
byte direccion=0;
byte dato=0;
byte resultado=0;

int pausa=5;

// Definimos el pin del LED rojo
int LED_R = 7;

// Definimos los nombres de las salidas que actuan como encoder
int ENC_A = 10;
int ENC_B = 11;
int ENC_PUL = 12;

// Definimos los nombres de las entradas que testean los valores del encoder
int LED_V = 34;
int LED_A = 36;

// Pines de los multiplexores
// Multiplexor de ZIN:
int M_ZIN_A0=38;
int M_ZIN_A1=40;
int M_ZIN_A2=42;
int M_ZIN_A3=44;

// Multiplexor de ZBF:
int M_ZBF_A0=46;
int M_ZBF_A1=48;
int M_ZBF_A2=50;
int M_ZBF_A3=52;

int retardo=10;
int f_estado;
int d_LED_V=0;
int d_LED_A=0;


void setup() {

  // Inicialización de pines digitales como salidas
  pinMode(LED_R, OUTPUT);
  pinMode(ENC_A, OUTPUT);
  pinMode(ENC_B, OUTPUT);
```

```
  pinMode(ENC_PUL, OUTPUT);

  pinMode(M_ZIN_A0, OUTPUT);
  pinMode(M_ZIN_A1, OUTPUT);
  pinMode(M_ZIN_A2, OUTPUT);
  pinMode(M_ZIN_A3, OUTPUT);

  pinMode(M_ZBF_A0, OUTPUT);
  pinMode(M_ZBF_A1, OUTPUT);
  pinMode(M_ZBF_A2, OUTPUT);
  pinMode(M_ZBF_A3, OUTPUT);

  // Inicialización de pines digitales como entradas
  pinMode(LED_V, INPUT);
  pinMode(LED_A, INPUT);

  // Inicialización a cero las salidas al encoder del DDS1
  digitalWrite(ENC_A, LOW);
  digitalWrite(ENC_B, LOW);
  digitalWrite(ENC_PUL, LOW);

  // Fijamos el estado inicial por defecto de los multiplexores (2 kohmios)
  digitalWrite(M_ZIN_A0, LOW);
  digitalWrite(M_ZIN_A1, LOW);
  digitalWrite(M_ZIN_A2, LOW);
  digitalWrite(M_ZIN_A3, LOW);

  digitalWrite(M_ZBF_A0, LOW);
  digitalWrite(M_ZBF_A1, LOW);
  digitalWrite(M_ZBF_A2, LOW);
  digitalWrite(M_ZBF_A3, LOW);

  // Inicializacion del i2c y puerto serie
  Wire.begin();
  Serial.begin(115200);

  // Esperamos a la activación del modulo DDS1
  digitalWrite(LED_R, HIGH);
  delay (500);
  while ((d_LED_V==0)&&(d_LED_A==0)) {
      d_LED_V = digitalRead(LED_V);
      d_LED_A = digitalRead(LED_A);
   }
  digitalWrite(LED_R, LOW);

// Fijamos a 0 Hz la frecuencia por defecto del DDS1 en el arranque
f_100_MHz_OFF();
}

void loop()
{
  while(Serial.available() > 0){

    if(indice < 3) // One less than the size of the array
    {
      inByte = Serial.read(); // Read a Byte
      inData[indice] = inByte; // Store it
      // Esperamos a la sincronizacion
      if ((inData[0]==65) ||(inData[0]==13) || (inData[0]==14)|| (inData[0]==15)|| (inData[0]==16)|| (inData[0]==17)) {
        indice++;
      }
    }
    if(indice==3) // One less than the size of the array
    {
       // Calculamos las ordenes a enviar por i2c
       dispositivo=inData[0];
```

```
direccion=inData[1];
dato=inData[2];

indice=0;

// Ejecutamos test de lectura
      if (dispositivo==65) {
         //delay (pausa);
         Serial.write(64);
       }
// Ejecutamos la escritura
      if (dispositivo==13) {
        writeData(direccion, dato);
      }
// Ejecutamos la lectura
      if (dispositivo==14) {
         resultado=readData(direccion);
         //delay (pausa);
         Serial.write(resultado);
      }
 // Programamos el DDS1
      if (dispositivo==15) {
          switch(dato) {

        // Frecuencia del DDS a 25 kHz
        case 1:
            f_OFF();
            f_25_kHz_ON();
        break;

        // Frecuencia del DDS a 50 kHz
        case 2:
            f_OFF();
            f_50_kHz_ON();
        break;

        // Frecuencia del DDS a 250 kHz
        case 3:
            f_OFF();
            f_250_kHz_ON();
        break;

        // Frecuencia del DDS a 1 MHz
        case 4:
            f_OFF();
            f_1_MHz_ON();
        break;

        // Frecuencia del DDS a 2 MHz
        case 5:
            f_OFF();
            f_2_MHz_ON();
        break;

        // Frecuencia del DDS a 4 MHz
        case 6:
            f_OFF();
            f_4_MHz_ON();
        break;

        // Frecuencia del DDS a 16 MHz
        case 7:
            f_OFF();
            f_16_MHz_ON();
        break;
      }
```

```
      }

// Calibraciones del MUX ZIN:
  if (dispositivo==16) {
    switch(dato) {
// c1: 2k
 case 1:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN1");
 break;

// c2: 3k
 case 2:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN2");
 break;

// c3: 4k
 case 3:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN3");
 break;

// c4: 5k
 case 4:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN4");
 break;

// c5: 6k
 case 5:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN5");
 break;

// c6: 7k
 case 6:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN6");
 break;

// c7: 8k
 case 7:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN7");
```

```
 break;

// c8: 9k
 case 8:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, LOW);
 //Serial.println("Calibracion ZIN8");
 break;

// c9: 10k
 case 9:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN9");
 break;

// c10: 11k
 case 10:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN10");
 break;

// c11: 12k
 case 11:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN11");
 break;

// c12: 13k
 case 12:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, LOW);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN12");
 break;

// c13: 14k
 case 13:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN13");
 break;

// c14: 15k
 case 14:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, LOW);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN14");
 break;

// c15: 16k
```

```
 case 15:
 digitalWrite(M_ZIN_A0, LOW);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN15");
 break;

// c16: Para medir la impedancia externa
 case 16:
 digitalWrite(M_ZIN_A0, HIGH);
 digitalWrite(M_ZIN_A1, HIGH);
 digitalWrite(M_ZIN_A2, HIGH);
 digitalWrite(M_ZIN_A3, HIGH);
 //Serial.println("Calibracion ZIN16");
 break;

  }
 }
      // Calibraciones del MUX ZBF:
     if (dispositivo==17) {
         switch(dato) {
// d1: 2k
 case 1:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF1");
 break;

// d2: 3k
 case 2:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF2");
 break;

// d3: 4k
 case 3:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF3");
 break;

// d4: 5k
 case 4:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF4");
 break;

// d5: 6k
 case 5:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, HIGH);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF5");
 break;
```

```
// d6: 7k
 case 6:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, HIGH);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF6");
 break;

// d7: 8k
 case 7:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, HIGH);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF7");
 break;

// d8: 9k
 case 8:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, HIGH);
 digitalWrite(M_ZBF_A3, LOW);
 //Serial.println("Calibracion ZBF8");
 break;

// d9: 10k
 case 9:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, HIGH);
 //Serial.println("Calibracion ZBF9");
 break;

// d10: 11k
 case 10:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, LOW);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, HIGH);
 //Serial.println("Calibracion ZBF10");
 break;

// d11: 12k
 case 11:
 digitalWrite(M_ZBF_A0, LOW);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, HIGH);
 //Serial.println("Calibracion ZBF11");
 break;

// d12: 13k
 case 12:
 digitalWrite(M_ZBF_A0, HIGH);
 digitalWrite(M_ZBF_A1, HIGH);
 digitalWrite(M_ZBF_A2, LOW);
 digitalWrite(M_ZBF_A3, HIGH);
 //Serial.println("Calibracion ZBF12");
 break;

// d13: 14k
 case 13:
```

```
        digitalWrite(M_ZBF_A0, LOW);
        digitalWrite(M_ZBF_A1, LOW);
        digitalWrite(M_ZBF_A2, HIGH);
        digitalWrite(M_ZBF_A3, HIGH);
        //Serial.println("Calibracion ZBF13");
        break;

       // d14: 15k
        case 14:
        digitalWrite(M_ZBF_A0, HIGH);
        digitalWrite(M_ZBF_A1, LOW);
        digitalWrite(M_ZBF_A2, HIGH);
        digitalWrite(M_ZBF_A3, HIGH);
        //Serial.println("Calibracion ZBF14");
        break;

       // d15: 16k
        case 15:
        digitalWrite(M_ZBF_A0, LOW);
        digitalWrite(M_ZBF_A1, HIGH);
        digitalWrite(M_ZBF_A2, HIGH);
        digitalWrite(M_ZBF_A3, HIGH);
        //Serial.println("Calibracion ZBF15");
        break;

          }
        }
       }
     }
  }

// Rutina de escritura de 1 byte por I2C
void writeData(int addr, int data) {

 Wire.beginTransmission(SLAVE_ADDR);
 Wire.write(addr);
 Wire.write(data);
 Wire.endTransmission();
 delay(1);
}

// Rutina de lectura de 1 byte por I2C
int readData(int addr){
  int data;

  Wire.beginTransmission(SLAVE_ADDR);
  Wire.write(ADDR_PTR);
  Wire.write(addr);
  Wire.endTransmission();

  delay(1);

  Wire.requestFrom(SLAVE_ADDR,1);

  if (Wire.available() >= 1){
    data = Wire.read();
  }
  else {
    data = -1;
  }

  delay(1);
  return data;
}

// Selección del modo del encoder
```

```
void modo() {
  // Con este pulso cambiamos de modo
  digitalWrite(ENC_PUL, HIGH);
  delay(retardo);
  digitalWrite(ENC_PUL, LOW);
  delay(retardo);
}

// Rotación a la izquierda del encoder
void izqda() {
  // Izquierda:
  digitalWrite(ENC_A, LOW);
  digitalWrite(ENC_B, LOW);
  delay(retardo);
  digitalWrite(ENC_A, LOW);
  digitalWrite(ENC_B, HIGH);
  delay(retardo);
  digitalWrite(ENC_A, HIGH);
  digitalWrite(ENC_B, HIGH);
  delay(retardo);
  digitalWrite(ENC_A, HIGH);
  digitalWrite(ENC_B, LOW);
  delay(retardo);
}

// Rotación a la derecha del encoder
void dcha() {
  // Derecha:
  digitalWrite(ENC_A, LOW);
  digitalWrite(ENC_B, LOW);
  delay(retardo);
  digitalWrite(ENC_A, HIGH);
  digitalWrite(ENC_B, LOW);
  delay(retardo);
  digitalWrite(ENC_A, HIGH);
  digitalWrite(ENC_B, HIGH);
  delay(retardo);
  digitalWrite(ENC_A, LOW);
  digitalWrite(ENC_B, HIGH);
  delay(retardo);
}

// Selección de f=25 kHz a la salida del DDS1
void f_25_kHz_ON() {
  dcha();
  dcha();
  dcha();
  dcha(); // Escala de 10 kHz
  modo();
  dcha(); // 10 kHz
  dcha(); // 20 kHz
  modo();
  dcha(); // Escala de 1 kHz
  modo();
  dcha(); // 21.1 kHz
  dcha(); // 21.2 kHz
  dcha(); // 21.3 kHz
  dcha(); // 21.4 kHz
  dcha(); // 21.5 kHz
  modo();
  f_estado=1;

}

// Selección de f=50 kHz a la salida del DDS1
void f_50_kHz_ON() {
```

```
  dcha();
  dcha();
  dcha();
  dcha(); // Escala de 10 kHz
  modo();
  dcha(); // 10 kHz
  dcha(); // 20 kHz
  dcha(); // 30 kHz
  dcha(); // 40 kHz
  dcha(); // 50 kHz
  modo();
  f_estado=2;

}

// Selección de f=250 kHz a la salida del DDS1
void f_250_kHz_ON() {
  dcha();
  dcha();
  dcha(); // Escala de 100 kHz
  modo();
  dcha(); // 100 kHz
  dcha(); // 200 kHz
  modo();
  dcha(); // Escala de 10 kHz
  modo();
  dcha(); // 210 kHz
  dcha(); // 220 kHz
  dcha(); // 230 kHz
  dcha(); // 240 kHz
  dcha(); // 250 kHz
  modo();
  f_estado=3;
}

// Selección de f=1 MHz a la salida del DDS1
void f_1_MHz_ON() {
  dcha();
  dcha();
  modo();
  dcha(); // 1 MHz
  modo();
  f_estado=4;
}

// Selección de f=2 MHz a la salida del DDS1
void f_2_MHz_ON() {
  dcha();
  dcha();
  modo();
  dcha(); // 1 MHz
  dcha(); // 2 MHz
  modo();
  f_estado=5;
}

// Selección de f=4 MHz a la salida del DDS1
void f_4_MHz_ON() {
  dcha();
  dcha();
  modo();
  dcha(); // 1 MHz
  dcha(); // 2 MHz
  dcha(); // 3 MHz
  dcha(); // 4 MHz
  modo();
```

```
  f_estado=6;
}

// Selección de f=16 MHz a la salida del DDS1
void f_16_MHz_ON() {
  dcha();
  modo();
  dcha(); // 10 MHz
  modo();
  dcha();
  modo();
  dcha(); // 11 MHz
  dcha(); // 12 MHz
  dcha(); // 13 MHz
  dcha(); // 14 MHz
  dcha(); // 15 MHz
  dcha(); // 16 MHz
  modo(); //  por defecto salimos en modo desplazamiento
  f_estado=7;
}

// Anulación de f=25 kHz a la salida del DDS1
void f_25_kHz_OFF() {
  modo();
  izqda(); // 24 kHz
  izqda(); // 23 kHz
  izqda(); // 22 kHz
  izqda(); // 21 kHz
  izqda(); // 20 kHz
  modo();
  izqda(); // Escala de 10 kHz
  modo();
  izqda(); // 10 kHz
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 100 kHz
  izqda(); // Escala de 1 MHz
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=50 kHz a la salida del DDS1
void f_50_kHz_OFF() {
  modo();
  izqda(); // 40 kHz
  izqda(); // 30 kHz
  izqda(); // 20 kHz
  izqda(); // 10 kHz
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 100 kHz
  izqda(); // Escala de 1 MHz
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=250 kHz a la salida del DDS1
void f_250_kHz_OFF() {
  modo();
  izqda(); // 240 kHz
  izqda(); // 230 kHz
  izqda(); // 220 kHz
  izqda(); // 210 kHz
  izqda(); // 200 kHz
```

```
  modo();
  izqda(); // Escala de 100 kHz
  modo();
  izqda(); // 100 kHz
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 1 MHz
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=1 MHz a la salida del DDS1
void f_1_MHz_OFF() {
  modo();
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=2 MHz a la salida del DDS1
void f_2_MHz_OFF() {
  modo();
  izqda(); // 1 MHz
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=4 MHz a la salida del DDS1
void f_4_MHz_OFF() {
  modo();
  izqda(); // 3 MHz
  izqda(); // 2 MHz
  izqda(); // 1 MHz
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 10 MHz
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=16 MHz a la salida del DDS1
void f_16_MHz_OFF() {
  modo();
  izqda(); // 15 MHz
  izqda(); // 14 MHz
  izqda(); // 13 MHz
  izqda(); // 12 MHz
  izqda(); // 11 MHz
  izqda(); // 10 MHz
  modo();
  izqda(); // Escala de 10 MHz
  modo();
  izqda(); // 0 Hz
  modo();
  izqda(); // Escala de 100 MHz
  f_estado=0;
}

// Anulación de f=100 MHz a la salida del DDS1
void f_100_MHz_OFF() {
```

```
// Nos aseguramos que estamos en modo desplazamiento
  d_LED_A = digitalRead(LED_A);
  if (d_LED_A==1) {
     modo();
  }
  izqda();
  izqda();
  modo();
  izqda(); // cero Hz
  modo(); //  por defecto salimos en modo desplazamiento
  f_estado=0;
}

// Selector de anulación de salidas del DDS1
void f_OFF() {
     switch(f_estado) {

       case 0: // estamos a frecuencia 0 Hz y no hacemos nada
          break;

       case 1: // estamos a frecuencia 25 kHz
          f_25_kHz_OFF();
          break;

       case 2: // estamos a frecuencia 50 kHz
          f_50_kHz_OFF();
          break;

       case 3: // estamos a frecuencia 250 kHz
          f_250_kHz_OFF();
          break;

       case 4: // estamos a frecuencia 1 MHz
          f_1_MHz_OFF();
          break;

       case 5: // estamos a frecuencia 2 MHz
          f_2_MHz_OFF();
          break;

       case 6: // estamos a frecuencia 4 MHz
          f_4_MHz_OFF();
          break;

       case 7: // estamos a frecuencia 16 MHz
          f_16_MHz_OFF();
          break;
     }
}
```

ANEXO 6: Archivo .psm del PicoBlaze

(En verde aparecen las líneas editadas para inhibir las llamadas a una LCD). Se ha incluido al final el fichero .ucf adaptado a la FPGA Spartan 3E XC3S250E-4VQG100C.

```
; KCPSM3 Program - Control and calculation for Frequency Generator design using the
;              Spartan-3E Starter Kit.
;
; Interfaces with the rotary encoder and LCD display to enable a frequency to be set.
; Converts the BCD frequency value into a binary integer and then performs the high
; precision calculation necessary to derive the control numbers required by the high
; performance Direct Digital Synthesis (DDS) circuit implemented in hardware.
;
; LEDs are connected and used as edit mode indicators.
;
; Substantial comments are included in line with the code below and should be used
; in conjunction with the documentation provided with the complete reference design.
;
;
;
; Ken Chapman - Xilinx Ltd
;
; Version v1.00 - 13th July 2006
;
;*****************************************************************************************
;Port definitions
;*****************************************************************************************
;
;
;
CONSTANT LED_port, 80            ;8 simple LEDs
CONSTANT LED0, 01                ;   LED 0 - bit0
CONSTANT LED1, 02                ;       1 - bit1
CONSTANT LED2, 04                ;       2 - bit2
CONSTANT LED3, 08                ;       3 - bit3
CONSTANT LED4, 10                ;       4 - bit4
CONSTANT LED5, 20                ;       5 - bit5
CONSTANT LED6, 40                ;       6 - bit6
CONSTANT LED7, 80                ;       7 - bit7
;
;
CONSTANT rotary_port, 00         ;Read status of rotary encoder
CONSTANT rotary_left, 01         ; Direction of last move Left=1 Right=0  - bit0
CONSTANT rotary_press, 02        ;    Centre press contact (active High) - bit1
;
;
;LCD interface ports
;
;The master enable signal is not used by the LCD display itself
;but may be required to confirm that LCD communication is active.
;This is required on the Spartan-3E Starter Kit if the StrataFLASH
;is used because it shares the same data pins and conflicts must be avoided.
;
CONSTANT LCD_output_port, 40     ;LCD character module output data and control
CONSTANT LCD_E, 01               ;  active High Enable      E - bit0
CONSTANT LCD_RW, 02              ;  Read=1 Write=0          RW - bit1
CONSTANT LCD_RS, 04              ;  Instruction=0 Data=1    RS - bit2
CONSTANT LCD_drive, 08           ;  Master enable (active High) - bit3
CONSTANT LCD_DB4, 10             ;  4-bit           Data DB4 - bit4
CONSTANT LCD_DB5, 20             ;  interface       Data DB5 - bit5
CONSTANT LCD_DB6, 40             ;                  Data DB6 - bit6
```

```
CONSTANT LCD_DB7, 80            ;               Data DB7 - bit7
;
;
CONSTANT LCD_input_port, 01     ;LCD character module input data
CONSTANT LCD_read_DB4, 10       ;   4-bit       Data DB4 - bit4
CONSTANT LCD_read_DB5, 20       ;   interface   Data DB5 - bit5
CONSTANT LCD_read_DB6, 40       ;               Data DB6 - bit6
CONSTANT LCD_read_DB7, 80       ;               Data DB7 - bit7
;
;
;
;DDS control ports
;
;DDS control word is 32-bits
;
CONSTANT DDS_control0_port, 02   ; dds_control_word(7:0)
CONSTANT DDS_control1_port, 04   ; dds_control_word(15:8)
CONSTANT DDS_control2_port, 08   ; dds_control_word(23:16)
CONSTANT DDS_control3_port, 10   ; dds_control_word(31:24)
;
;Frequency scaling control word is 5-bits
;
CONSTANT DDS_scaling_port, 20    ; dds_scaling_word(4:0)
;
;
;*****************************************************************************
;Special Register usage
;*****************************************************************************
;
;*****************************************************************************
;Scratch Pad Memory Locations
;*****************************************************************************
;
CONSTANT rotary_status, 00      ;Status of rotary encoder
CONSTANT rotary_event, 80       ;  flag set by interrupt in 'rotary_status' - bit7
;
CONSTANT ISR_preserve_s0, 01    ;Preserve s0 contents during ISR
;
CONSTANT LED_pattern, 02        ;LED pattern used in rotation mode
;
;
;BCD digits representing selected and displayed frequency
;
CONSTANT BCD_digit0, 03         ; value         1
CONSTANT BCD_digit1, 04         ;              10
CONSTANT BCD_digit2, 05         ;             100
CONSTANT BCD_digit3, 06         ;           1,000
CONSTANT BCD_digit4, 07         ;          10,000
CONSTANT BCD_digit5, 08         ;         100,000
CONSTANT BCD_digit6, 09         ;       1,000,000
CONSTANT BCD_digit7, 0A         ;      10,000,000
CONSTANT BCD_digit8, 0B         ;     100,000,000
;
;
;Binary integer representation of BCD value
;
CONSTANT frequency0, 0C         ;LS byte
CONSTANT frequency1, 0D
CONSTANT frequency2, 0E
CONSTANT frequency3, 0F         ;MS byte
;
;
;Control of frequency selection values
;
CONSTANT cursor_position, 10    ; Pointer to edit position on LCD
CONSTANT edit_digit_pointer, 11  ; BCD digit to be changed
```

```
;
;
;
;80-bit product resulting from 32-bit frequency x 48-bit scaling constant
;
CONSTANT product0, 12           ;LS byte
CONSTANT product1, 13
CONSTANT product2, 14
CONSTANT product3, 15
CONSTANT product4, 16
CONSTANT product5, 17
CONSTANT product6, 18
CONSTANT product7, 19
CONSTANT product8, 1A
CONSTANT product9, 1B           ;MS byte
;
;Local copies of the DDS control word and DDS scaling word
;
CONSTANT DDS_control0, 1C       ; dds_control_word(7:0)
CONSTANT DDS_control1, 1D       ; dds_control_word(15:8)
CONSTANT DDS_control2, 1E       ; dds_control_word(23:16)
CONSTANT DDS_control3, 1F       ; dds_control_word(31:24)
CONSTANT DDS_scaling, 20        ; dds_scaling_word(4:0)
;
;*****************************************************************************
; Useful data constants
;*****************************************************************************
;
; To convert the frequency into a DDS control value a high precision scaling
; factor is used. This is a 48-bit number which converts the frequency presented
; as an 32-bit integer into the 32-bit value required by the phase accumulator
; to synthesize the desired frequency. The scaling factor is derived using the
; following method. First I will consider the scaling factor which results in the
; desired frequency being generated directly at the output of the phase accumulator
; which is suitable for low frequencies in which a few ns of jitter is acceptable.
;
; 'Fpa' is frequency generated by the MSB of the phase accumulator.
; 'p' is number of phase accumulator which in this case is 32 bits.
; 'clk' is the input clock frequency to the phase accumulator which is 200MHz.
; 'N' is the DDS control word value which is also 'p' bits (32 in this case).
;
; Frequency at MSB of phase accumulator is then
;
;     Fpa = clk x N / (2^p)
;
; Note that the maximum value allowed for 'N' is (2^p)/2 which results in Fpa=clk/2.
;  for 'N' greater than that value 'Fpa' would decrease in frequency (aliasing).
;
;
; By simple reorganisation of the equation we can compute 'N'
;
;     N = Fpa x (2^p) / clk
;
;
; Now it is easier to approach the next step using specific example.
;
; So for a frequency of Fpa = 1MHz then
;     N = 1MHz x (2^32)/200MHz = 21474836.48
;
; We must use the nearest 32-bit integer value 21474836 and this in turn
; is best viewed as the 32-bit hexadecimal value 0147AE14.
;
; In this case the value we have to work with is a 32-bit integer frequency
; value of 1 million which is 000F4240.
;
; So now we need to translate the value 000F4240 into 0147AE14. This is
```

```
; where a 48-bit scaling value is used together with a full precision multiplier
; as this ensures adequate accuracy of the final frequency.
;
;       32-bit frequency value              ffffffff
;       48-bit scaling value          x ssssssssssss
;                              --------------------
;       80-bit product          nnnnnnnnnnnnnnnnnnnn
;
; The art is to organise the scaling factor into the range where the most is made of
; the 48-bit resolution available but which will result in the correct 32-bit output.
; The way this is achieved is the select an appropriate 32-bits from the available 80-bit
; product for use as 'N' and truncate 'y' least significant bits.
;
; From this we can deduce that for a target frequency 'Ft' at the input then the
; scaling value 'S' is given by
;
;    S = N x (2^y) / Ft    with the condition that S < 2^48 but as large as possible
;
; For best accuracy we calculate 'S' using the full precision value of 'N' divided
; by Ft and then multiply continuously by 2 until we reach the biggest value less
; that 2^48. The number of multiplications by 2 indicating the value of 'y'.
;
; In this case we find that 'y' is 43.....
;       S = 21474836.48 x (2^43) / 1000000 = 21.47483648 x (2^43)
;                                  = 188894659314785.80854784
;
;  ...round to nearest integer and convert to hexadecimal S = ABCC77118462
;
; N will be taken from the 80 bit product by removing the 43 LSBs and the 5 MSBs
; to leave the 32 active bits required. This is best achieved by shifting left
; by 5 places (multiply by 2^5=32) and keeping the upper 32-bits.
;
;
; Sanity check....
;   Note that most calculators do not support >64 bit values to you will either
;   need to decompose your calculation and perform some of it manually or trust
;   the PicoBlaze implementation :-)
;
;     Ft = 1MHz =                  000F4240
;           S =              x  ABCC77118462
;                       --------------------
;                       000A3D70A3D70A405C80
;
;    shift left 5 places                x 20
;                       --------------------
;                       0147AE147AE1480B9000
;
; As expected, the most significant 32-bit (4 bytes) are 0147AE14 hex which is
; the DDS control word for 1MHz calculated previously.
;
; ***
;
; Now I will consider how this needs to be modified for the circuit presented
; which has a second DCM connected to the output of the phase accumulator to
; multiply the synthesized frequency and reduce cycle to cycle jitter at
; the same time. There is then a clock divider circuit connected to the output
; of the DCM which allows lower frequencies to be formed a different way (more of
; that later). As a minimum that divider circuit will divide by 2 which ensures that
; a square wave is presented to the clocked put pin. So in this circuit the fundamental
; multiplication factor is 8 formed by a 16 times multiplication by the DCM (256/16) and
; then a divide by 2.
;
; The overall multiplication factor of this sebsequent circuit means that for final
; output from the DCM to be the desired frequency, the output from the phase accumulator
; needs to be the same number of times smaller. This is not a bad thing because the
; percentage jitter of waveforms produced by the phase accumulator is better for lower
```

```
; frequencies made from more clock cycles.
;
; So we modify the basic equation to
;
;   Fout = Frequency at output of DCM
;      M = Multiplying factor of DCM
;
;   Fout = M x Fpa = M x clk x N / (2^p)
;
;
; By simple reorganisation of the equation we can compute 'N'
;
;      N = Fout x (2^p) / (clk x M)
;
;
; In this design M=8, p=32, clk=200MHz
;
; So now consider generating a nominal maximum frequency of 100MHz which will require
; the frequency synthesized by the phase accumulator to be 12.5MHz.
;
;      N = 100MHz x (2^32) / (200MHz x 8) = 268435456 = 10000000 Hex
;
; This all seems like a very convenient number but it simply reflects that 12.5MHz
; is a perfect division of the 200MHz clock and that that output from the phase
; accumulator will be formed perfectly of 16 of the 200MHz clock periods every time
; (8 Low and 8 High) with no additional jitter.
;
; So now we work out the scaling factor with the same rules as used previously that
; the scaling factor should be as large as possible within the 48-bits allocated.
;
;   S = N x (2^y) / Ft    with the condition that S < 2^48 but as large as possible
;
; In this case Ft = 100MHz = 055FE100 and the biggest value for S is found when using
; y=46
;
;   S = 268435456 x (2^46) / 100000000 = 2.68435456 x (2^46)
;                              = 188894659314785.80854784
;
;     round to 188894659314786  = ABCC77118462
;
; Actually this is the exact same scaling constant as previously because the
; frequency to be synthesized by the phase accumulator is 8 times smaller but the
; value of 'S' is deliberate scaled to be as large as possible. In fact, 'S' in this
; case has been scaled up by a factor of 8 to arrive at the same value. So after
; using the scaling constant to form the 80 bit product, this time we will remove
; the 46 LSBs and the 2 MSBs to leave the 32 active bits required. This is best
; achieved by shifting left by 2 places (multiply by 2^2=4) and keeping the upper
; 32-bits (last time we multiplied by 32 which was 8 times more).
;
;
; Sanity check....
;
;    Ft = 100MHz =                    055FE100
;            S =               x  ABCC77118462
;                        --------------------
;                        04000000000001242200
;
;      shift left 5 places                 x 20
;                        --------------------
;                        1000000000001C908800
;
; As expected, the most significant 32-bit (4 bytes) are 10000000 hex which is
; the DDS control word for 12.5MHz at the phase accumulator output calculated
; previously.
;
;
```

```
; ********
;
;
; 48-bit Scaling factor constant to generate the phase accumulator control word
; from the integer frequency value.
;
;   S = AB CC 77 11 84 62
;
; Notes
;
; The 80-bit product must be shifted left 5 times and then most significant 32-bits
; used to provide DDS control word if the frequency required is to be synthesized
; directly at the output of the phase accumulator.
;
; The 80-bit product must be shifted left 2 times and then most significant 32-bits
; used to provide DDS control word if the frequency required is to be synthesized
; by the phase accumulator followed by a multiplying DCM and divider with overall
; frequency gain of 8 times.
;
CONSTANT scale_constant0, 62       ;LS byte
CONSTANT scale_constant1, 84
CONSTANT scale_constant2, 11
CONSTANT scale_constant3, 77
CONSTANT scale_constant4, CC
CONSTANT scale_constant5, AB       ;MS byte
;
;
;
; ************************
;
;Constant to define a software delay of 1us. This must be adjusted to reflect the
;clock applied to KCPSM3. Every instruction executes in 2 clock cycles making the
;calculation highly predictable. The '6' in the following equation even allows for
;'CALL delay_1us' instruction in the initiating code.
;
; delay_1us_constant =  (clock_rate - 6)/4      Where 'clock_rate' is in MHz
;
;Example: For a 50MHz clock the constant value is (10-6)/4 = 11  (0B Hex).
;For clock rates below 10MHz the value of 1 must be used and the operation will
;become lower than intended.
;
CONSTANT delay_1us_constant, 0B
;
;
;
;ASCII table
;
CONSTANT character_a, 61
CONSTANT character_b, 62
CONSTANT character_c, 63
CONSTANT character_d, 64
CONSTANT character_e, 65
CONSTANT character_f, 66
CONSTANT character_g, 67
CONSTANT character_h, 68
CONSTANT character_i, 69
CONSTANT character_j, 6A
CONSTANT character_k, 6B
CONSTANT character_l, 6C
CONSTANT character_m, 6D
CONSTANT character_n, 6E
CONSTANT character_o, 6F
CONSTANT character_p, 70
CONSTANT character_q, 71
CONSTANT character_r, 72
CONSTANT character_s, 73
```

```
CONSTANT character_t, 74
CONSTANT character_u, 75
CONSTANT character_v, 76
CONSTANT character_w, 77
CONSTANT character_x, 78
CONSTANT character_y, 79
CONSTANT character_z, 7A
CONSTANT character_A, 41
CONSTANT character_B, 42
CONSTANT character_C, 43
CONSTANT character_D, 44
CONSTANT character_E, 45
CONSTANT character_F, 46
CONSTANT character_G, 47
CONSTANT character_H, 48
CONSTANT character_I, 49
CONSTANT character_J, 4A
CONSTANT character_K, 4B
CONSTANT character_L, 4C
CONSTANT character_M, 4D
CONSTANT character_N, 4E
CONSTANT character_O, 4F
CONSTANT character_P, 50
CONSTANT character_Q, 51
CONSTANT character_R, 52
CONSTANT character_S, 53
CONSTANT character_T, 54
CONSTANT character_U, 55
CONSTANT character_V, 56
CONSTANT character_W, 57
CONSTANT character_X, 58
CONSTANT character_Y, 59
CONSTANT character_Z, 5A
CONSTANT character_0, 30
CONSTANT character_1, 31
CONSTANT character_2, 32
CONSTANT character_3, 33
CONSTANT character_4, 34
CONSTANT character_5, 35
CONSTANT character_6, 36
CONSTANT character_7, 37
CONSTANT character_8, 38
CONSTANT character_9, 39
CONSTANT character_colon, 3A
CONSTANT character_stop, 2E
CONSTANT character_semi_colon, 3B
CONSTANT character_minus, 2D
CONSTANT character_divide, 2F      ;'/'
CONSTANT character_plus, 2B
CONSTANT character_comma, 2C
CONSTANT character_less_than, 3C
CONSTANT character_greater_than, 3E
CONSTANT character_equals, 3D
CONSTANT character_space, 20
CONSTANT character_CR, 0D          ;carriage return
CONSTANT character_question, 3F    ;'?'
CONSTANT character_dollar, 24
CONSTANT character_exclaim, 21     ;'!'
CONSTANT character_BS, 08          ;Back Space command character
;
;
;
;
;
;*****************************************************************************
;Initialise the system
```

```
        ;*******************************************************************************
        ;
cold_start: ;CALL LCD_reset                 ;initialise LCD display
        ;
        ;Write 'Frequency Generator' to LCD display and display for 4 seconds
        ;
        LOAD s5, 10                 ;Line 1 position 0
        ;CALL LCD_cursor
        ;CALL disp_Frequency
        LOAD s5, 22                 ;Line 2 position 2
        ;CALL LCD_cursor
        ;CALL disp_Generator
        CALL delay_1s                ;wait 4 seconds
        CALL delay_1s
        CALL delay_1s
        CALL delay_1s
        ;CALL LCD_clear               ;clear screen
        ;
        ;
        ;Initial frequency of 100MHz
        ;
        LOAD s0, 00
        LOAD s1, 01
        STORE s0, BCD_digit0
        STORE s0, BCD_digit1
        STORE s0, BCD_digit2
        STORE s0, BCD_digit3
        STORE s0, BCD_digit4
        STORE s0, BCD_digit5
        STORE s0, BCD_digit6
        STORE s0, BCD_digit7
        STORE s1, BCD_digit8
        ;
        LOAD s0, 04                 ;Start position for editing frequency is 1MHz digit
        STORE s0, cursor_position
        LOAD s0, BCD_digit6
        STORE s0, edit_digit_pointer
        ;
        ;
        ENABLE INTERRUPT            ;interrupts are used to detect rotary controller
        CALL delay_1ms
        LOAD s0, 00                 ;clear the status of any spurious rotary events
        STORE s0, rotary_status      ;   as a result of system turning on.
        ;
        ;*******************************************************************************
        ; Main program
        ;*******************************************************************************
        ;
        ; The main program is centred on the task of editing the frequency. It waits until the
        ; rotary control is used and then makes the appropriate changes. If the actual digit
        ; digit value is changed then the calculation to drive the DDS is performed each time.
        ;
        ; The start state is that of allowing the edit cursor position to be moved. Rotary
        ; inputs are detected by the interrupt service routine and set a flag bit which the
        ; main program then uses to adjust the cursor position and pointer to the corresponding
        ; BCD digit in memory.
        ;
        ; A press of the rotary control is detected by polling and used to change to the digit
        ; editing mode.
        ;
        ;
move_mode: CALL compute_DDS_words       ;compute DDS control values
        ;CALL display_freq           ;refresh display with cursor position shown
        LOAD s0, LED0               ;indicate move mode on LEDs
        OUTPUT s0, LED_port
move_wait: INPUT s0, rotary_port        ;read rotary encoder
```

```
            TEST s0, rotary_press           ;test for press of button which changes mode
            JUMP NZ, edit_mode
            FETCH s0, rotary_status         ;check for any rotation of rotary control
            TEST s0, rotary_event
            JUMP Z, move_wait
            ;
            AND s0, 7F                  ;clear flag now that action it is being processed
            STORE s0, rotary_status
            FETCH sA, cursor_position       ;read current position
            FETCH sB, edit_digit_pointer
            TEST s0, rotary_left            ;determine direction to move cursor
            JUMP Z, move_right
            ;
    move_left: COMPARE sB, BCD_digit8           ;can not move left of 100MHz digit
            JUMP Z, move_mode
            ADD sB, 01                  ;move to next higher BCD digit
            SUB sA, 01                  ;move cursor to match digit to be edited
            COMPARE sA, 09                  ;must skip over space separator
            JUMP Z, skip_left
            COMPARE sA, 05                  ;must skip over decimal point
            JUMP NZ, edit_point_update
    skip_left: SUB sA, 01                   ;move cursor further left
            JUMP edit_point_update
            ;
   move_right: COMPARE sB, BCD_digit0           ;can not move right of 1Hz digit
            JUMP Z, move_mode
            SUB sB, 01                  ;move to next lower BCD digit
            ADD sA, 01                  ;move cursor to match digit to be edited
            COMPARE sA, 09                  ;must skip over space separator
            JUMP Z, skip_right
            COMPARE sA, 05                  ;must skip over decimal point
            JUMP NZ, edit_point_update
   skip_right: ADD sA, 01                   ;move cursor further right
            ;
edit_point_update: STORE sA, cursor_position        ;update edit value in memory
            STORE sB, edit_digit_pointer
            JUMP move_mode
            ;
            ;
            ; The edit mode is reached by pressing the rotary control. Since this is a simple switch
            ; a software de-bounce delay is used to wait for the knob to be released fully before
            ; entering the digit editing mode fully.
            ;
            ; In this mode rotations of the detected by the interrupt service routine are used to
            ; increment or decrement the digit value at the cursor position with carry/borrow to
            ; the left.
            ;
            ; A new press of the rotary control is detected by polling and used to change back to the
            ; cursor moving mode.
            ;
            ;
    edit_mode: CALL wait_switch_release         ;wait for switch press to end
  edit_display: CALL compute_DDS_words          ;compute DDS control values
            ;CALL display_freq              ;refresh display with new values
            LOAD s0, LED1                   ;indicate edit mode on LEDs
            OUTPUT s0, LED_port
    edit_wait: INPUT s0, rotary_port            ;read rotary encoder
            TEST s0, rotary_press           ;test for press of button which changes mode
            JUMP NZ, end_edit_mode
            FETCH s0, rotary_status         ;check for any rotation of rotary control
            TEST s0, rotary_event
            JUMP Z, edit_wait
            ;
            AND s0, 7F                  ;clear flag now that action it is being processed
            STORE s0, rotary_status
            FETCH sB, edit_digit_pointer    ;read pointer to BCD digit for initial change
```

```
            TEST s0, rotary_left            ;determine direction to increment or decrement
            JUMP Z, inc_digit
            ;
            ; Decrement the value starting at the current position and borrowing from the left.
            ; However the value needs to bottom out at all 0's from the editing position.
            ;
            ;
    dec_digit: FETCH sA, (sB)               ;read digit value at pointer position
            SUB sA, 01                  ;decrement digit
            COMPARE sA, FF              ;test for borrow from next digit
            JUMP Z, dec_borrow
            STORE sA, (sB)              ;store decremented digit value
            JUMP edit_display           ;decrement task complete
  dec_borrow: LOAD sA, 09                   ;current digit rolls over to nine
            STORE sA, (sB)              ;store '9' digit value
            COMPARE sB, BCD_digit8          ;check if working on 100MHz digit
            JUMP Z, set_min_value
            ADD sB, 01                  ;increment pointer to next most significant digit
            JUMP dec_digit              ;decrement next digit up.
            ;
 set_min_value: FETCH sB, edit_digit_pointer    ;Must fill digits from insert to MS-Digit with 000...
            LOAD sA, 00
   fill_min: STORE sA, (sB)
            COMPARE sB, BCD_digit8          ;check if filled to 100MHz digit
            JUMP Z, edit_display
            ADD sB, 01                  ;fill next higher digit
            JUMP fill_min
            ;
            ; Increment the value starting at the current position and carrying to the left.
            ; However the value needs to saturate to all 9's from the editing position.
            ;
    inc_digit: FETCH sA, (sB)               ;read digit value at pointer position
            ADD sA, 01                  ;increment digit
            COMPARE sA, 0A              ;test for carry to next digit
            JUMP Z, inc_carry
            STORE sA, (sB)              ;store incremented digit value
            JUMP edit_display           ;increment task complete
    inc_carry: LOAD sA, 00                  ;current digit rolls over to zero
            STORE sA, (sB)              ;store zero digit value
            COMPARE sB, BCD_digit8          ;check if working on 100MHz digit
            JUMP Z, set_max_value
            ADD sB, 01                  ;increment pointer to next most significant digit
            JUMP inc_digit              ;increment next digit up.
            ;
 set_max_value: FETCH sB, edit_digit_pointer    ;Must fill digits from insert to MS-Digit with 999...
            LOAD sA, 09
   fill_max: STORE sA, (sB)
            COMPARE sB, BCD_digit8          ;check if filled to 100MHz digit
            JUMP Z, edit_display
            ADD sB, 01                  ;fill next higher digit
            JUMP fill_max
            ;
 end_edit_mode: CALL wait_switch_release        ;wait for end of switch press
            JUMP move_mode              ;then go to move cursor mode
            ;
            ;
            ; Routine to poll the press switch with de-bounce delay and wait for it to be
            ; released. Any rotation inputs detected by the interrupt
            ; service routine are cleared before returning.
            ;
wait_switch_release: CALL delay_20ms            ;delay to aid switch de-bounce
            INPUT s0, rotary_port           ;read rotary encoder
            TEST s0, rotary_press           ;test if button is still being pressed
            JUMP NZ, wait_switch_release
            LOAD s0, 00                 ;clear flag indicating any rotary events
            STORE s0, rotary_status
```

```
RETURN
;
;*******************************************************************************
; Compute DDS control words from currently display frequency value
;*******************************************************************************
;
; This routine reads the current BCD value and converts it into a 32-bit binary
; integer. It then multiplies it by a 48-bit scaling factor (see notes in the
; constants section above) to form a full precision 80-bit product.
;
; From this product the 32-bit DDS control word must be extracted. For frequencies of
; 50MHz or above the DDS control word is formed by shifting the product left by 2 places
; (multiply by 4) and then keeping only the most significant 32-bits (4 bytes).
;
; Also for frequencies of 50MHz and above, there is no additional division performed
; after the DCM which multiplies frequency and reduces the jitter. Therefore the DDS_scaling
; word will be set to zero and the output of the DCM will divide by 2.
;
;  Freq    DDS control word   DDS Scaling     Synthesized Frequency
;                             of Phase Accumulator
;
; 50MHz     08000000           00              6.25MHz
;100MHz     10000000           00              12.50MHz
;
; You will notice that for frequencies of 50MHz and above, the upper byte of the
; DDS control word is 08 hex or greater. In other words, bit3 and/or bit4 of that byte
; are High (bits 27 and/or 28 of the full 32-bit word). This is the indication that
; the control words are complete.
;
; For frequencies below 50MHz an additional process is required. The reason for this
; becomes clear if we think about the lowest frequency of 1Hz. In that case the 80-bit
; product is the same as the 48-bit scaling constant 00000000ABCC77118462. Once this
; has been multiplied by 4 (shifted left 2 places) it becomes 00000002AF31DC461188 and the
; most significant 32-bits are only 00000002 hex. If we put this back into the basic
; equations for the phase accumulator we find that the output frequency of the phase
; accumulator would be
;
;   Fout = M x clk x N / (2^p)
;
;       = 8 x 200MHz x 2 / (2^32) = 0.745 Hz
;
; There are two important observations we can make. Firstly we have lost accuracy because
; the resolution of the DDS control word has become too granular at low amplitudes.
; Secondly this would never even work because the frequency synthesized by the phase
; accumulator would be 0.745/8 = 0.0931 Hz which is seriously slow and a way below the
; frequency at which the DCM can even work.
;
; The solution to both of these issues is to ensure that the DDS control word is always
; formed to be in the range that would result in an output of 50MHz or above. In other
; words to keep the phase accumulator output in the range 6.25MHz to 12.5MHz such that
; the DCM is able to work and only has to deal with one octave of input variation. This
; can be achieved by shifting the 80-bit product left more times until bits 27 and 28
; of the most significant 32-bits are not zero.
;
; For each shift left the synthesized frequency is being doubled and therefore the final
; output from the DCM must be divided by a further factor of 2. This is achieved using
; a multiplexer which is guided to select the appropriate output from a simple binary
; counter.
;
; Returning to the example of 1Hz, the 80-bit product will be shifted left by the default
; 2 places (multiplied by 4), but will then need to be shifted left by a further 26 places
; which is like multiplying by 67108864 (04000000 hex).
;
;                   00000000ABCC77118462
;                  x              4
;                   --------------------
```

```
;                    00000002AF31DC461188
;
;
;                  x         04000000
;                    --------------------
;                    0ABCC771184620000000
;
; So now the DDS control word is 0ABCC771 (180143985 decimal)and the frequency synthesized
; by the phase accumulator will be....
;
;   Fpa = clk x N / (2^p) = 200MHz x 180143985 / (2^32) = 8388608Hz
;
; The DCM will multiply this by a factor of 16 to give 134217728Hz and this will then
; be divided by the counter of which the 26th bit selected (26 decimal = 1A hex).
;
;   Fout = Fpa x 16 / (2^(D+1)) =  8388608Hz x 16 / (2^(26+1)) = 0.99999999947 Hz
;
;    'D' is the DDS Scaling factor
;       Note that bit0 of a counter is a clock division by 2 and hence the 'D+1'
;
; Clearly this implementation style has provided much greater accuracy and enables
; the DCM to work for all desired frequencies.
;
;
;   Freq    DDS control word    DDS Scaling     Synthesized Frequency
;                                of Phase Accumulator
;
; 100 MHz     10000000           00              12.50MHz
;  50 MHz     08000000           00              6.25MHz
;  25 MHz     08000000           01              6.25MHz
;  12.5 MHz   08000000           02               6.25MHz
;
;   1Hz      0ABCC771           1A              8.388608 MHz
;
;
;
; In order to ensure the DCM is always provided with a frequency in an acceptable
; range, the value of absolute zero is never implemented and instead just a very low
; frequency is produced.
;   6.25MHz x 16 / (2^31+1) = 0.0233 Hz
;      which is 1 cycle every 43 seconds and that is pretty slow  :-)
;
;
;
;
compute_DDS_words: CALL BCD_to_integer             ;convert BCD display value to 32-bit value
            CALL scale_frequency             ;80-bit product of 32-bit frequency x 48-bit scaling value
            FETCH sA, product9               ;read the upper part of the 80-bit product into [sA,s9,s8,s7,s6,s5,s4]
            FETCH s9, product8               ; The least significant 24-bits of the 80-bit product will never
            FETCH s8, product7               ; be used for frequencies above 1Hz.
            FETCH s7, product6               ;The final 32-bit DDS control word will be formed in
            FETCH s6, product5               ; [sA,s9,s8,s7]
            FETCH s5, product4
            FETCH s4, product3
            CALL shift80_left                ;multiply DDS control word by 4 to achieve default value
            CALL shift80_left
            LOAD sB, 00                    ;default scaling factor is 2 (select counter bit0)
  normalise_loop: TEST sA, 18                    ;Test bits 27 and 28 of 32-bit DDS control word
            JUMP NZ, store_DDS_words          ;DDS control word is normalised to above 50MHz output
            CALL shift80_left                ;multiply DDS control word by 2
            ADD sB, 01                     ;Divide final value by 2 to compensate
            COMPARE sB, 1F                   ;Test for maximum division factor
            JUMP NZ, normalise_loop
            LOAD sA, 08                    ;Set for minimum frequency
            LOAD s9, 00                    ; with phase accumulator set to generate 6.25MHz
            LOAD s8, 00
```

```
            LOAD s7, 00
store_DDS_words: STORE s7, DDS_control0          ;store local copy of control word
            STORE s8, DDS_control1          ;store local copy of control word
            STORE s9, DDS_control2          ;store local copy of control word
            STORE sA, DDS_control3          ;store local copy of control word
            STORE sB, DDS_scaling
            CALL drive_DDS_words            ;output control words to DDS circuit
            RETURN
            ;
  shift80_left: SL0 s4                      ;shift (most of the) 80-bit value in
            SLA s5                    ;  [sA,s9,s8,s7,s6,s5,s4] left 1 place
            SLA s6
            SLA s7
            SLA s8
            SLA s9
            SLA sA
            RETURN
            ;
            ;*****************************************************************************
            ; Set DDS control words
            ;*****************************************************************************
            ;
            ; Because multiple ports are used, the idea is to update all of them in
            ; rapid succession to avoid too much disturbance in the frequency synthesis.
            ;
            ; dds_control_word should be supplied in register set [sA,s9,s8,s7]
            ; dds_scaling_word should be supplied in register s6.
            ;
drive_DDS_words: FETCH s7, DDS_control0
            FETCH s8, DDS_control1
            FETCH s9, DDS_control2
            FETCH sA, DDS_control3
            FETCH s6, DDS_scaling
            OUTPUT s7, DDS_control0_port
            OUTPUT s8, DDS_control1_port
            OUTPUT s9, DDS_control2_port
            OUTPUT sA, DDS_control3_port
            OUTPUT s6, DDS_scaling_port
            RETURN
            ;
            ;
            ;*****************************************************************************
            ; Display frequency on top line of the LCD and DDS data on the lower line
            ;*****************************************************************************
            ;
            ; The BCD value should be stored in scratch pad memory in 9 ascending locations
            ; called BCD_digit0 to BCD_digit8.
            ;
            ; The value is displayed in the format     xxx.xxx xxxMHz
            ;
            ; However, the most significant 2 digits will be blanked if zero.
            ;
            ; registers used s0,s1,s2,s3,s4,s5,s6,s7
            ;
            ;
  display_freq: ;CALL display_DDS_data          ;display DDS information on lower line
            LOAD s5, 12                 ;Line 1 position 2
            ;CALL LCD_cursor
            FETCH s5, BCD_digit8          ;read 100MHz digit
            COMPARE s5, 00              ;test for blanking
            JUMP Z, blank_100M_digit
            ;CALL display_digit          ;display non zero digit
            FETCH s5, BCD_digit7          ;read 10MHz digit and display
            ;CALL display_digit
            JUMP disp_1M_digit
            ;
```

```
blank_100M_digit: ;CALL display_space              ;blank 100MHz digit
            FETCH s5, BCD_digit7            ;read 10MHz digit
            COMPARE s5, 00                  ;test for blanking
            JUMP Z, blank_10M_digit
            ;CALL display_digit             ;display non zero digit
            JUMP disp_1M_digit
            ;
 blank_10M_digit: ;CALL display_space              ;blank 10MHz digit
            ;
  disp_1M_digit: FETCH s5, BCD_digit6           ;read 1MHz digit and display
            ;CALL display_digit
            LOAD s5, character_stop         ;display decimal point
            ;CALL LCD_write_data
            ;
            LOAD s2, BCD_digit5             ;set pointer to 100KHz digit
            ;CALL display_3_digits
            ;CALL display_space
            LOAD s2, BCD_digit2             ;set pointer to 100Hz digit
            ;CALL display_3_digits
            LOAD s5, character_M             ;display 'MHz'
            ;CALL LCD_write_data
            LOAD s5, character_H
            ;CALL LCD_write_data
            LOAD s5, character_z
            ;CALL LCD_write_data
            ;
            FETCH s5, cursor_position       ;reposition edit cursor on display
            ADD s5, 10                  ;on line 1
            ;CALL LCD_cursor
            RETURN
            ;
display_3_digits: LOAD s3, 03                   ;3 digits to display
   3digit_loop: FETCH s5, (s2)
            ;CALL display_digit
            SUB s2, 01                  ;decrement digit pointer
            SUB s3, 01                  ;count digits displayed
            JUMP NZ, 3digit_loop
            RETURN
            ;
  display_digit: ADD s5, 30                 ;convert BCD to ASCII character
            ;CALL LCD_write_data
            RETURN
            ;
  display_space: LOAD s5, character_space
            ;CALL LCD_write_data
            RETURN
            ;
            ;
            ;*****************************************************************************
            ; Convert 9 digit BCD frequency into 32-bit binary integer
            ;*****************************************************************************
            ;
            ;Both values are stored in scratch pad memory
            ;   BCD values in ascending locations BCD_digit0 to BCD_digit8
            ;   Binary frequency in ascending locations frequency0 to frequency3
            ;
            ;Each digit is read in turn and its value is determined by repeated
            ;decrement until reaching zero. Each decrement causes a value to be added
            ;to the memory locations forming the frequency value as binary integer.
            ;The process requires approximately 1600 instructions to convert the highest
            ;value 999,999,999 which is approximately 64us at 50MHz clock rate.
            ;
            ;Registers used s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,sA,sB
            ;
  BCD_to_integer: LOAD s2, 09               ;9 digits to convert
            LOAD s0, 00                 ;clear frequency value ready to accumulate result
```

```
            STORE s0, frequency0
            STORE s0, frequency1
            STORE s0, frequency2
            STORE s0, frequency3
            LOAD sB, 00                 ;initialise BCD digit weighting [sB,sA,s9,s8] to 1
            LOAD sA, 00
            LOAD s9, 00
            LOAD s8, 01
            LOAD s3, BCD_digit0         ;locate LS-digit
next_BCD_to_int_digit: FETCH s1, (s3)
  BCD_digit_convert: COMPARE s1, 00             ;test for zero
            JUMP Z, next_digit_value
            FETCH s0, frequency0        ;add 32-bit digit weighting to memory value
            ADD s0, s8
            STORE s0, frequency0
            FETCH s0, frequency1
            ADDCY s0, s9
            STORE s0, frequency1
            FETCH s0, frequency2
            ADDCY s0, sA
            STORE s0, frequency2
            FETCH s0, frequency3
            ADDCY s0, sB
            STORE s0, frequency3
            SUB s1, 01                  ;decrement digit value
            JUMP BCD_digit_convert
            ;Increase weighting by 10x
   next_digit_value: LOAD s7, sB                ;copy existing weighting
            LOAD s6, sA
            LOAD s5, s9
            LOAD s4, s8
            SL0 s8                      ;multiply weight by 4x (shift left 2 places)
            SLA s9
            SLA sA
            SLA sB
            SL0 s8
            SLA s9
            SLA sA
            SLA sB
            ADD s8, s4                  ;add previous weight to form 5x multiplication
            ADDCY s9, s5
            ADDCY sA, s6
            ADDCY sB, s7
            SL0 s8                      ;multiply weight by 2x (shift left 1 places)
            SLA s9
            SLA sA
            SLA sB                      ;weight value is now 10x previous value
            ADD s3, 01                  ;move to next digit for conversion
            SUB s2, 01
            JUMP NZ, next_BCD_to_int_digit
            RETURN
            ;
            ;
            ;*****************************************************************************
            ; 32-bit x 48-bit multiply to scale the integer frequency
            ;*****************************************************************************
            ;
            ;Multiply the 32-bit frequency binary integer by the 48-bit scaling factor
            ;to form a full precision 80-bit product.
            ;
            ;The frequency binary integer is stored in scratch pad memory using ascending
            ;locations frequency0 to frequency3
            ;
            ;The product will be stored in scratch pad memory using ascending
            ;locations product0 to product9
            ;
```

```
            ;The scaling factor is provided directly as constants
            ; scale_constant0 to scale_constant5
            ;
            ;The multiplication is performed as a 32-bit 'shift and add' process in which the
            ;integer frequency is examined LSB first using a register set [sB,sA,s9,s8] and
            ;a scaling accumulator is formed directly in the 'product' memory locations.
            ;
            ;The process requires up to 1772 instructions which is 3544 clock cycle or
            ;approximately 71us at 50MHz clock rate.
            ;
            ;Registers used s0,s1,s8,s9,sA,sB (s1,s8,s9,sA,sB clear on return)
            ;
scale_frequency: LOAD s0, 00                ;clear accumulator section of 'product'
            STORE s0, product9
            STORE s0, product8
            STORE s0, product7
            STORE s0, product6
            STORE s0, product5
            STORE s0, product4
            FETCH sB, frequency3            ;read frequency integer value
            FETCH sA, frequency2
            FETCH s9, frequency1
            FETCH s8, frequency0
            LOAD s1, 20                 ;32-bit multiply
 scale_mult_bit: SR0 sB                     ;shift right frequency integer
            SRA sA
            SRA s9
            SRA s8
            JUMP NC, product_shift          ;no add if bit is zero (note carry is zero)
            FETCH s0, product4              ;addition of scaling factor to most significant bits of product
            ADD s0, scale_constant0
            STORE s0, product4
            FETCH s0, product5
            ADDCY s0, scale_constant1
            STORE s0, product5
            FETCH s0, product6
            ADDCY s0, scale_constant2
            STORE s0, product6
            FETCH s0, product7
            ADDCY s0, scale_constant3
            STORE s0, product7
            FETCH s0, product8
            ADDCY s0, scale_constant4
            STORE s0, product8
            FETCH s0, product9
            ADDCY s0, scale_constant5
            STORE s0, product9              ;carry holds any overflow of addition
  product_shift: FETCH s0, product9             ;Divide product by 2 (shift right by 1)
            SRA s0                      ;overflow of addition included in shift
            STORE s0, product9
            FETCH s0, product8
            SRA s0
            STORE s0, product8
            FETCH s0, product7
            SRA s0
            STORE s0, product7
            FETCH s0, product6
            SRA s0
            STORE s0, product6
            FETCH s0, product5
            SRA s0
            STORE s0, product5
            FETCH s0, product4
            SRA s0
            STORE s0, product4
            FETCH s0, product3
```

```
            SRA s0
            STORE s0, product3
            FETCH s0, product2
            SRA s0
            STORE s0, product2
            FETCH s0, product1
            SRA s0
            STORE s0, product1
            FETCH s0, product0
            SRA s0
            STORE s0, product0
            SUB s1, 01                  ;move to next bit
            JUMP NZ, scale_mult_bit
            RETURN
            ;
            ;***************************************************************************************
            ; Display DDS control information on the lower line of the LCD display.
            ;***************************************************************************************
            ;
            ;Display the 32-bit DDS control word and 8-bit DDS scaling word.
            ;
display_DDS_data: LOAD s5, 20               ;Line 2 position 0
            ;CALL LCD_cursor
            LOAD s5, character_N
            ;CALL LCD_write_data
            LOAD s5, character_equals
            ;CALL LCD_write_data
            LOAD s7, DDS_control3           ;pointer to most significant byte in memory
            ;CALL display_hex_32_bit
            ;CALL display_space
            LOAD s5, character_D
            ;CALL LCD_write_data
            LOAD s5, character_equals
            ;CALL LCD_write_data
            FETCH s0, DDS_scaling
            ;CALL display_hex_byte
            RETURN
            ;
            ;***************************************************************************************
            ; Routines to display hexadecimal values on LCD display
            ;***************************************************************************************
            ;
            ;
            ; Convert hexadecimal value provided in register s0 into ASCII characters
            ;
            ; The value provided must can be any value in the range 00 to FF and will be converted into
            ; two ASCII characters.
            ;    The upper nibble will be represented by an ASCII character returned in register s3.
            ;    The lower nibble will be represented by an ASCII character returned in register s2.
            ;
            ; The ASCII representations of '0' to '9' are 30 to 39 hexadecimal which is simply 30 hex
            ; added to the actual decimal value. The ASCII representations of 'A' to 'F' are 41 to 46
            ; hexadecimal requiring a further addition of 07 to the 30 already added.
            ;
            ; Registers used s0, s2 and s3.
            ;
hex_byte_to_ASCII: LOAD s2, s0              ;remember value supplied
            SR0 s0                      ;isolate upper nibble
            SR0 s0
            SR0 s0
            SR0 s0
            CALL hex_to_ASCII           ;convert
            LOAD s3, s0                 ;upper nibble value in s3
            LOAD s0, s2                 ;restore complete value
            AND s0, 0F                  ;isolate lower nibble
            CALL hex_to_ASCII           ;convert
```

```
            LOAD s2, s0                 ;lower nibble value in s2
            RETURN
            ;
            ; Convert hexadecimal value provided in register s0 into ASCII character
            ;
            ;Register used s0
            ;
  hex_to_ASCII: SUB s0, 0A              ;test if value is in range 0 to 9
            JUMP C, number_char
            ADD s0, 07                  ;ASCII char A to F in range 41 to 46
   number_char: ADD s0, 3A              ;ASCII char 0 to 9 in range 30 to 40
            RETURN
            ;
            ;
            ; Display the two character HEX value of the register contents 's0' on the LCD
            ; at the current cursor position.
            ;
            ; Registers used s0, s1, s2, s3, s4, s5
            ;
display_hex_byte: CALL hex_byte_to_ASCII
            LOAD s5, s3
            ;CALL LCD_write_data
            LOAD s5, s2
            ;CALL LCD_write_data
            RETURN
            ;
            ;
            ;
            ; Display the 32-bit value stored in 4 ascending memory locations as an 8 character
            ; HEX value at the current cursor position. Register s7 must contain the memory
            ; location of the most significant byte (which is also the highest address).
            ;
            ; Registers used s0, s1, s2, s3, s4, s5, s6, s7
            ;
display_hex_32_bit: LOAD s6, 04         ;4 bytes to display
   disp32_loop: FETCH s0, (s7)          ;read byte
            ;CALL display_hex_byte      ;display byte
            SUB s7, 01                  ;decrement pointer
            SUB s6, 01                  ;count bytes displayed
            RETURN Z
            JUMP disp32_loop
            ;
            ;
            ;*****************************************************************************
            ;LCD text messages
            ;*****************************************************************************
            ;
            ;
            ;Display 'Frequency' on LCD at current cursor position
            ;
 disp_Frequency: LOAD s5, character_F
            ;CALL LCD_write_data
            LOAD s5, character_r
            ;CALL LCD_write_data
            LOAD s5, character_e
            ;CALL LCD_write_data
            LOAD s5, character_q
            ;CALL LCD_write_data
            LOAD s5, character_u
            ;CALL LCD_write_data
            LOAD s5, character_e
            ;CALL LCD_write_data
            LOAD s5, character_n
            ;CALL LCD_write_data
            LOAD s5, character_c
            ;CALL LCD_write_data
```

```
            LOAD s5, character_y
            ;CALL LCD_write_data
            RETURN
            ;
            ;Display 'Generator' on LCD at current cursor position
            ;
disp_Generator: LOAD s5, character_G
            ;CALL LCD_write_data
            LOAD s5, character_e
            ;CALL LCD_write_data
            LOAD s5, character_n
            ;CALL LCD_write_data
            LOAD s5, character_e
            ;CALL LCD_write_data
            LOAD s5, character_r
            ;CALL LCD_write_data
            LOAD s5, character_a
            ;CALL LCD_write_data
            LOAD s5, character_t
            ;CALL LCD_write_data
            LOAD s5, character_o
            ;CALL LCD_write_data
            LOAD s5, character_r
            ;CALL LCD_write_data
            ;CALL display_space
            LOAD s5, character_v
            ;CALL LCD_write_data
            LOAD s5, character_1
            ;CALL LCD_write_data
            LOAD s5, character_stop
            ;CALL LCD_write_data
            LOAD s5, character_2
            ;CALL LCD_write_data
            RETURN
            ;
            ;
            ;
            ;
            ;*****************************************************************************************
            ;Software delay routines
            ;*****************************************************************************************
            ;
            ;
            ;
            ;Delay of 1us.
            ;
            ;Constant value defines reflects the clock applied to KCPSM3. Every instruction
            ;executes in 2 clock cycles making the calculation highly predictable. The '6' in
            ;the following equation even allows for 'CALL delay_1us' instruction in the initiating code.
            ;
            ; delay_1us_constant =  (clock_rate - 6)/4       Where 'clock_rate' is in MHz
            ;
            ;Registers used s0
            ;
   delay_1us: LOAD s0, delay_1us_constant
    wait_1us: SUB s0, 01
            JUMP NZ, wait_1us
            RETURN
            ;
            ;Delay of 40us.
            ;
            ;Registers used s0, s1
            ;
  delay_40us: LOAD s1, 28                   ;40 x 1us = 40us
   wait_40us: CALL delay_1us
            SUB s1, 01
```

```
        JUMP NZ, wait_40us
        RETURN
        ;
        ;
        ;Delay of 1ms.
        ;
        ;Registers used s0, s1, s2
        ;
 delay_1ms: LOAD s2, 19                 ;25 x 40us = 1ms
  wait_1ms: CALL delay_40us
        SUB s2, 01
        JUMP NZ, wait_1ms
        RETURN
        ;
        ;Delay of 20ms.
        ;
        ;Delay of 20ms used during initialisation.
        ;
        ;Registers used s0, s1, s2, s3
        ;
delay_20ms: LOAD s3, 14                 ;20 x 1ms = 20ms
 wait_20ms: CALL delay_1ms
        SUB s3, 01
        JUMP NZ, wait_20ms
        RETURN
        ;
        ;Delay of approximately 1 second.
        ;
        ;Registers used s0, s1, s2, s3, s4
        ;
  delay_1s: LOAD s4, 32                 ;50 x 20ms = 1000ms
   wait_1s: CALL delay_20ms
        SUB s4, 01
        JUMP NZ, wait_1s
        RETURN
        ;
        ;
        ;
        ;*****************************************************************************
        ;LCD Character Module Routines
        ;*****************************************************************************
        ;
        ;LCD module is a 16 character by 2 line display but all displays are very similar
        ;The 4-wire data interface will be used (DB4 to DB7).
        ;
        ;The LCD modules are relatively slow and software delay loops are used to slow down
        ;KCPSM3 adequately for the LCD to communicate. The delay routines are provided in
        ;a different section (see above in this case).
        ;
        ;
        ;Pulse LCD enable signal 'E' high for greater than 230ns (1us is used).
        ;
        ;Register s4 should define the current state of the LCD output port.
        ;
        ;Registers used s0, s4
        ;
LCD_pulse_E: XOR s4, LCD_E              ;E=1
        OUTPUT s4, LCD_output_port
        CALL delay_1us
        XOR s4, LCD_E               ;E=0
        OUTPUT s4, LCD_output_port
        RETURN
        ;
        ;Write 4-bit instruction to LCD display.
        ;
        ;The 4-bit instruction should be provided in the upper 4-bits of register s4.
```

```
        ;Note that this routine does not release the master enable but as it is only
        ;used during initialisation and as part of the 8-bit instruction write it
        ;should be acceptable.
        ;
        ;Registers used s4
        ;
LCD_write_inst4: AND s4, F8                 ;Enable=1 RS=0 Instruction, RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
        ;CALL LCD_pulse_E
        RETURN
        ;
        ;
        ;Write 8-bit instruction to LCD display.
        ;
        ;The 8-bit instruction should be provided in register s5.
        ;Instructions are written using the following sequence
        ; Upper nibble
        ; wait >1us
        ; Lower nibble
        ; wait >40us
        ;
        ;Registers used s0, s1, s4, s5
        ;
LCD_write_inst8: LOAD s4, s5
        AND s4, F0                  ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
        OR s4, LCD_drive            ;Enable=1
        ;CALL LCD_write_inst4            ;write upper nibble
        CALL delay_1us              ;wait >1us
        LOAD s4, s5                 ;select lower nibble with
        SL1 s4                  ;Enable=1
        SL0 s4                  ;RS=0 Instruction
        SL0 s4                  ;RW=0 Write
        SL0 s4                  ;E=0
        ;CALL LCD_write_inst4            ;write lower nibble
        CALL delay_40us             ;wait >40us
        LOAD s4, F0                 ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ;Release master enable
        RETURN
        ;
        ;
        ;
        ;Write 8-bit data to LCD display.
        ;
        ;The 8-bit data should be provided in register s5.
        ;Data bytes are written using the following sequence
        ; Upper nibble
        ; wait >1us
        ; Lower nibble
        ; wait >40us
        ;
        ;Registers used s0, s1, s4, s5
        ;
 LCD_write_data: LOAD s4, s5
        AND s4, F0                  ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
        OR s4, 0C                   ;Enable=1 RS=1 Data, RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
        ;CALL LCD_pulse_E            ;write upper nibble
        CALL delay_1us              ;wait >1us
        LOAD s4, s5                 ;select lower nibble with
        SL1 s4                  ;Enable=1
        SL1 s4                  ;RS=1 Data
        SL0 s4                  ;RW=0 Write
        SL0 s4                  ;E=0
        OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
        ;CALL LCD_pulse_E            ;write lower nibble
        CALL delay_40us             ;wait >40us
```

```
            LOAD s4, F0                 ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
            OUTPUT s4, LCD_output_port      ;Release master enable
            RETURN
            ;
            ;
            ;
            ;
            ;Read 8-bit data from LCD display.
            ;
            ;The 8-bit data will be read from the current LCD memory address
            ;and will be returned in register s5.
            ;It is advisable to set the LCD address (cursor position) before
            ;using the data read for the first time otherwise the display may
            ;generate invalid data on the first read.
            ;
            ;Data bytes are read using the following sequence
            ; Upper nibble
            ; wait >1us
            ; Lower nibble
            ; wait >40us
            ;
            ;Registers used s0, s1, s4, s5
            ;
LCD_read_data8: LOAD s4, 0E                 ;Enable=1 RS=1 Data, RW=1 Read, E=0
            OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
            XOR s4, LCD_E               ;E=1
            OUTPUT s4, LCD_output_port
            CALL delay_1us              ;wait >260ns to access data
            INPUT s5, LCD_input_port        ;read upper nibble
            XOR s4, LCD_E               ;E=0
            OUTPUT s4, LCD_output_port
            CALL delay_1us              ;wait >1us
            XOR s4, LCD_E               ;E=1
            OUTPUT s4, LCD_output_port
            CALL delay_1us              ;wait >260ns to access data
            INPUT s0, LCD_input_port        ;read lower nibble
            XOR s4, LCD_E               ;E=0
            OUTPUT s4, LCD_output_port
            AND s5, F0                  ;merge upper and lower nibbles
            SR0 s0
            SR0 s0
            SR0 s0
            SR0 s0
            OR s5, s0
            LOAD s4, 04                 ;Enable=0 RS=1 Data, RW=0 Write, E=0
            OUTPUT s4, LCD_output_port      ;Stop reading 5V device and release master enable
            CALL delay_40us             ;wait >40us
            RETURN
            ;
            ;
            ;Reset and initialise display to communicate using 4-bit data mode
            ;Includes routine to clear the display.
            ;
            ;Requires the 4-bit instructions 3,3,3,2 to be sent with suitable delays
            ;following by the 8-bit instructions to set up the display.
            ;
            ;  28 = '001' Function set, '0' 4-bit mode, '1' 2-line, '0' 5x7 dot matrix, 'xx'
            ;  06 = '000001' Entry mode, '1' increment, '0' no display shift
            ;  0E = '00001' Display control, '1' display on, '1' cursor off, '0' cursor blink off
            ;  01 = '00000001' Display clear
            ;
            ;Registers used s0, s1, s2, s3, s4
            ;
  LCD_reset: CALL delay_20ms            ;wait more that 15ms for display to be ready
            LOAD s4, 30
            ;CALL LCD_write_inst4        ;send '3'
```

```
            CALL delay_20ms             ;wait >4.1ms
            ;CALL LCD_write_inst4       ;send '3'
            CALL delay_1ms              ;wait >100us
            ;CALL LCD_write_inst4       ;send '3'
            CALL delay_40us             ;wait >40us
            LOAD s4, 20
            ;CALL LCD_write_inst4       ;send '2'
            CALL delay_40us             ;wait >40us
            LOAD s5, 28                 ;Function set
            ;CALL LCD_write_inst8
            LOAD s5, 06                 ;Entry mode
            ;CALL LCD_write_inst8
            LOAD s5, 0E                 ;Display control
            ;CALL LCD_write_inst8
LCD_clear: LOAD s5, 01                  ;Display clear
            ;CALL LCD_write_inst8
            CALL delay_1ms              ;wait >1.64ms for display to clear
            CALL delay_1ms
            RETURN
            ;
            ;Position the cursor ready for characters to be written.
            ;The display is formed of 2 lines of 16 characters and each
            ;position has a corresponding address as indicated below.
            ;
            ;           Character position
            ;       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
            ;
            ; Line 1 - 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
            ; Line 2 - C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
            ;
            ;This routine will set the cursor position using the value provided
            ;in register s5. The upper nibble will define the line and the lower
            ;nibble the character position on the line.
            ; Example s5 = 2B will position the cursor on line 2 position 11
            ;
            ;Registers used s0, s1, s2, s3, s4
            ;
LCD_cursor: TEST s5, 10                 ;test for line 1
            JUMP Z, set_line2
            AND s5, 0F                  ;make address in range 80 to 8F for line 1
            OR s5, 80
            ;CALL LCD_write_inst8       ;instruction write to set cursor
            RETURN
set_line2: AND s5, 0F                   ;make address in range C0 to CF for line 2
            OR s5, C0
            ;CALL LCD_write_inst8       ;instruction write to set cursor
            RETURN
            ;
            ;This routine will shift the complete display one position to the left.
            ;The cursor position and LCD memory contents will not change.
            ;
            ;
            ;Registers used s0, s1, s2, s3, s4, s5
            ;
LCD_shift_left: LOAD s5, 18             ;shift display left
            ;CALL LCD_write_inst8
            RETURN
            ;
            ;*****************************************************************************
            ;Interrupt Service Routine (ISR)
            ;*****************************************************************************
            ;
            ;Interrupts occur when the rotary control has been moved.
            ;
            ;The ISR captures the state of the direction which it writes to scratch pad memory (SPM).
            ;The most significant bit is also set at this location to provide a 'flag' to the
```

```
    ;main body of the program.
    ;
    ;
ISR: STORE s0, ISR_preserve_s0        ;preserve s0
    INPUT s0, rotary_port            ;read rotary encoder
    OR s0, rotary_event              ;set flag
    STORE s0, rotary_status          ;put result in SCM
    FETCH s0, ISR_preserve_s0        ;restore s0
    RETURNI ENABLE
    ;
    ;
    ;*****************************************************************************************
    ;Interrupt Vector
    ;*****************************************************************************************
    ;
    ADDRESS 3FF
    JUMP ISR
    ;
    ;
```

Archivo .ucf:

```
# Constraints for reference design 'frequency_generator'.
#
# Revision C of the Spartan-3E Starter Kit.
#
# Ken Chapman - Xilinx Ltd
#
# 28th June 2006
#
# Period constraint for 50MHz operation of control logic
#
NET "clk" PERIOD = 20.0ns HIGH 50%;
#
#
# Period constraint for 200MHz operation of DDS phase accumulator
#   (Specification used actually equates to 151.5MHz)
#
NET "dds_clk" PERIOD = 5.0ns HIGH 50%;
#
# Period constraint for up to 200MHz operation of frequency divider
#   In practice this frequency is variable and only reaches 200MHz as a maximum.
#   (Specification used actually equates to 151.5MHz)
#
NET "synth_clk" PERIOD = 5.0ns HIGH 50%;
#
#
# IMPORTANT NOTE - All paths between clock domains are non critical.
# They are not given timing specifications and therefore can be as
# slow as they like.
#
#
#
# Lock the position of the DCM used in frequency aligned mode so that the special
# settings can be applied in BITGEN.
#
INST "frequency_aligned_dcm" LOC=DCM_X1Y1;
#
#
#
```

```
# I/O contraints
#
#
#
# soldered 50MHz Clock.
#
NET "clk" LOC = "P48" | IOSTANDARD = LVTTL;
#
#
# SMA socket.
#
NET "sma_out" LOC = "P24" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
#
#
# Simple LEDs
# Require only 3.5mA.
#
NET "led<0>" LOC = "P94" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<1>" LOC = "P95" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<2>" LOC = "P86" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<3>" LOC = "P49" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<4>" LOC = "P53" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<5>" LOC = "P54" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<6>" LOC = "P57"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "led<7>" LOC = "P58"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
#
#
# LCD display
# Very slow so can use lowest drive strength.
#
NET "lcd_rs"   LOC = "P60" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_rw"   LOC = "P61" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_e"    LOC = "P62" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<4>" LOC = "P65" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<5>" LOC = "P66" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<6>" LOC = "P67" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<7>" LOC = "P68" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
#
# Strata Flash (need to disable to use LCD display)
#
NET "strataflash_oe" LOC = "P70" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_ce" LOC = "P71" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_we" LOC = "P83" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
#
#
# Rotary encoder.
#   Rotation contacts require pull UP resistors to provide High level.
#   Press contact requires pull DOWN resistor to provide Low when not pressed..
#
NET "rotary_a"     LOC = "P15" | IOSTANDARD = LVTTL | PULLDOWN;
NET "rotary_b"     LOC = "P16" | IOSTANDARD = LVTTL | PULLDOWN;
NET "rotary_press" LOC = "P17" | IOSTANDARD = LVTTL | PULLDOWN;
#
#
# Simple I/O pins on connector J4
#
NET "simple<9>"  LOC = "P5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "simple<10>" LOC = "P9" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "simple<11>" LOC = "P10" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "simple<12>" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
#
#
# End of File
#
```

ANEXO 7: Script de Matlab

```
% Control de la tarjeta AD5933



% Función de control del AD5933
function AD5933
% Borramos ventana de comandos y memoria
clc;
clear all
% Seleccionamos formato largo para las cifras
format long

% Añadimos configuración por defecto del AD5933
Gain=1;              % Gain X1
OutputVoltage=0;     % Output excitation range 1
r_cal=0;

% Vectores procedentes de la calibración multipunto
calibration_GainFactors=0;
calibration_phase_multi_point=0;

%Vectores procedentes de la medida de la impedancia
MagnitudeArray=0;
PhaseArray=0;
RealDataArray=0;
ImagineryDataArray=0;
FrequencyPoints=0;

% Configuración de la comunicación serie
h='0';
PS=serial('COM3');
set(PS,'Baudrate',115200); % se configura la velocidad a 9600 Baudios
set(PS,'StopBits',1); % se configura bit de parada a uno
set(PS,'DataBits',8); % se configura que el dato es de 8 bits, debe estar
entre 5 y 8
set(PS,'Parity','none'); % se configura sin paridad
set(PS,'Terminator','CR/LF');% “c” caracter con que finaliza el envío
set(PS,'OutputBufferSize',1); % ”n” es el número de bytes a enviar
set(PS,'InputBufferSize' ,1); % ”n” es el número de bytes a recibir
set(PS,'Timeout',5); % 5 segundos de tiempo de espera
%PS.RequestToSend='off';
PS.DataTerminalReady= 'off'; % Eliminamos el reset al Arduino conectar el
puerto serie

% Abrimos el puerto serie
fopen(PS);

% Selector de casos
while h~= 0
h=input('orden a ejecutar: ');
    switch h
```

```
         % Medida de la temperatura del AD5933
        case 1
         MeasureTemperature

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=25 kHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 2
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,1);   % CLK=25 kHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(25000,10,4,5,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(25000,10,4,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=50 kHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 3
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,2);   % CLK=50 kHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(50000,31,14,5,5,10000);
```

```
        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(50000,31,14,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=250 kHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 4
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,3);   % CLK=250 kHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(250000,102,20,5,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(250000,102,20,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=1 MHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 5
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,4);   % CLK=1 MHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
```

```
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(1000000,203,120,5,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(1000000,203,120,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=2 MHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 6
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,5);   % CLK=2 MHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(2000000,813,38,5,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(2000000,813,38,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
```

```
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=4 MHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 7
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,6);   % CLK=4 MHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(4000000,1010,250,16,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(4000000,1010,250,16,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=16 MHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 8
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,7);   % CLK=16 MHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(16000000,5011,250,179,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
```

```
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(16000000,5011,250,179,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

        % Test con ZIN=Zcal=10k, ZBF=10k, CLK=16 MHz, Rango 1: 2 Vpp,
PGA=x1, Multiplicador=x1
        case 9
         PortWrite(16,128,9);   % mux ZIN=10k
         PortWrite(17,128,9);   % mux ZBF=10k
         PortWrite(15,128,7);   % CLK=16 MHz
         pause(1);              % Esperamos a que la salida del DDS1 sea
estable
         OUTPUT_excitation(1);  % Range 1: 2 Vpp
         PGA_control(1);        % X1
         Multiply_by_one;       % DDS Settlig time cicles x1

        % Cálculo del Factor de Ganancia en modo Multipunto con ZB=10k y
Zin=Zcal=10k
         % function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
         CalculateGainFactor_multipoint(16000000,50011,10000,5,5,10000);

        % Hacemos la lectura de la impedancia externa (DUT)
         PortWrite(16,128,16);   % mux ZIN=Zext
         pause(0.1);             % Esperamos a la comuntación del
multiplexor
         % function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
         Sweep_single(16000000,50011,10000,5,5);

         % Grabación de los datos a fichero
         vector_de_datos=[FrequencyPoints' RealDataArray'
ImagineryDataArray' MagnitudeArray' PhaseArray'];
         save ('datos.txt', 'vector_de_datos', '-ascii', '-append');

         case 10
         % Cargamos el fichero
         datos_fichero=load('datos.txt');

         % Preparamos los 3 vectores con los datos
         datos_frecuencia=datos_fichero(:,1);
         datos_parte_real=datos_fichero(:,2);
         datos_parte_imaginaria=datos_fichero(:,3);
         datos_modulo=datos_fichero(:,4);
         datos_fase=datos_fichero(:,5);
```

```
            % Determinamos los extremos en abcisas
            [minimo_frec posicion_1]=min(datos_frecuencia);
            [maximo_frec posicion_2]=max(datos_frecuencia);

            % Creamos las ventanas gráficas
            % POLAR:
            figure('Name','MEDIDOR DE IMPEDANCIA COMPLEJA')
            polar(datos_fase*pi/180,datos_modulo,'m');
            title('Diagrama polar');

            % Real e Imaginaria
            figure('Name','MEDIDOR DE IMPEDANCIA COMPLEJA')
            subplot(2,1,1)
            plot(datos_frecuencia, datos_parte_real, 'c','linewidth',2);
            title('MEDIDOR DE IMPEDANCIA COMPLEJA');
            xlabel('Frecuencia de barrido en Hz');
            ylabel('Parte real (Re) de la impedancia Z en Ohmios');
            xlim([minimo_frec maximo_frec])
            ylim([-50000 50000])
            grid on
            subplot(2,1,2)
            plot(datos_frecuencia, datos_parte_imaginaria,
'y','linewidth',2);
            xlabel('Frecuencia de barrido en Hz');
            ylabel('Parte imaginaria (Im) de la impedancia Z en Ohmios');
            xlim([minimo_frec maximo_frec])
            ylim([-50000 50000])
            grid on

            % Modulo y fase
            figure('Name','MEDIDOR DE IMPEDANCIA COMPLEJA')
            subplot(2,1,1)
            plot(datos_frecuencia, datos_modulo, 'r','linewidth',2);
            title('MEDIDOR DE IMPEDANCIA COMPLEJA');
            xlabel('Frecuencia de barrido en Hz');
            ylabel('Módulo de la impedancia Z en Ohmios');
            xlim([minimo_frec maximo_frec])
            ylim([0 50000])
            grid on
            subplot(2,1,2)
            plot(datos_frecuencia, datos_fase, 'linewidth',2);
            xlabel('Frecuencia de barrido en Hz');
            ylabel('Fase de la impedancia Z en Ohmios');
            xlim([minimo_frec maximo_frec])
            ylim([-180 180])
            grid on
        end
    end

    % Cerramos el puerto serie y vaciamos su buffer
    fclose(PS);
    delete(PS);
    clear PS;

    %=====================================================================
```

```
% Funciones:
%=========================================================================
% Escritura de 1 byte en el AD5933
    function PortWrite (Dispositivo, RegisterAddress, RegisterData)
         fwrite(PS,Dispositivo,'uint8');
         fwrite(PS,RegisterAddress,'uint8');
         fwrite(PS,RegisterData,'uint8');
    end
%=========================================================================
% Lectura de 1 byte del AD5933
    function Lectura=PortRead (Dispositivo, RegisterAddress)
         PortWrite (Dispositivo, RegisterAddress, 5);
         Lectura=fread(PS);
    end
%=========================================================================
% Escritura de 1 byte en el AD5933 con chequeo
    function WritetToPart(RegisterAddress, RegisterData)
         PortWrite (13, RegisterAddress, RegisterData);
%          Readback = PortRead(14, RegisterAddress);
%          if (Readback ~=RegisterData)
%             disp('error')
%          end
    end
%=========================================================================
% Escritura en el registro de control con chequeo con PGA y VOUT
    function WritetToControlRegister(RegisterAddress, RegisterData)
         RegisterData = int8(RegisterData + Gain + OutputVoltage);
         PortWrite (13, RegisterAddress, RegisterData);
%          Readback1 = PortRead(14, RegisterAddress);
%          if (Readback1 ~=RegisterData)
%             disp('error_1')
%          end
    end
%=========================================================================
% Escritura en el registro de control con chequeo sin PGA y VOUT
    function WritetToControlRegister2(RegisterAddress, RegisterData)
         RegisterData = int8(RegisterData);
         PortWrite (13, RegisterAddress, RegisterData);
%          Readback2 = PortRead(14, RegisterAddress);
%          if (Readback2 ~=RegisterData)
%             disp('error_2')
%          end
    end
%=========================================================================
% Rutina para la medida de la temperatura del AD5933
    function MeasureTemperature
         WritetToPart (128, 144);
        % pause(0.005);
         ReadbackStatusRegister = PortRead(14, 143);
         ReadbackStatusRegister = ReadbackStatusRegister & 1;
         while(ReadbackStatusRegister ~= 1)
           ReadbackStatusRegister = PortRead(14, 143);
           ReadbackStatusRegister = ReadbackStatusRegister & 1;
         end
         TemperatureUpper = PortRead(14, 146);
         TemperatureLower = PortRead(14, 147);
         Temperature = TemperatureLower + (TemperatureUpper * 256);
```

```
         if (Temperature <= 8191) % msb =0.
            % Positive Temperature.
            Temp_val = (Temperature / 32)
         else
            % Negative Temperature.
            Temp_val = (Temperature - 16384) / 32
         end
            %re-assign variables used.
            TemperatureUpper = 0;
            TemperatureLower = 0;
            Temperature = 0;
    end
%=====================================================================
% Rutina de refresco de la frecuencia de inicio del barrido
    function UpdateStartFrequency(DDSRefClockFrequency, StartFrequency)
            TempStartFrequency = (StartFrequency / (DDSRefClockFrequency
/ 4)) * 16777215 * 8;
            TempStartFrequency = floor(TempStartFrequency);
            if (StartFrequency < 0)
            % Convert Start Frequency to 3 Byte local Variables.
                StartFrequencybyte0 = 0;
                StartFrequencybyte1A = 0;
                StartFrequencybyte2 = 0;
                % Write in data to Start frequency register and verify
                % correct contents written
                WritetToPart (132, StartFrequencybyte0);   %84 hex lsb
                WritetToPart (131, StartFrequencybyte1A);  %83 hex
                WritetToPart (130, StartFrequencybyte2);   %82 hex msb
             else
                %-------------------------------------------------------
--
                StartFrequencybyte0 = int8(bitand(TempStartFrequency,
255));
                StartFrequencybyte1A = int8(bitand(TempStartFrequency,
65280) / 256);
                StartFrequencybyte2 = int8(bitand(TempStartFrequency,
16711680) / 65536);

                StartFrequencybyte1B = StartFrequencybyte1A; % TEMPORY
COPY OF StartFrequencybyte1A

                if (StartFrequencybyte1A >= 255)        % is Dim
StartFrequencybyte1a too big to transmit as 1 byte i.e ffhex
                                                        % if it is it
must be broken up resulting in any overflow into a second byte being
carried over to the third byte
                                                        % remember that
the part will only store the bottom three bytes
                    StartFrequencybyte1B =
int8(bitand(StartFrequencybyte1B, 255));
                    MSBfromStartFrequencybyte1B =
int8(bitand(StartFrequencybyte1A, 65280) / 256);
                    StartFrequencybyte2 = int8((StartFrequencybyte2 *
256) + MSBfromStartFrequencybyte1B);
                    StartFrequencybyte2 = int8(((StartFrequencybyte2 *
256) + MSBfromStartFrequencybyte1B) & 255);
                end
```

```
                % Write in data to Start frequency register and verify
                % correct contents written
                WritetToPart (132, StartFrequencybyte0);   %84 hex lsb
                WritetToPart (131, StartFrequencybyte1B);  %83 hex
                WritetToPart (130, StartFrequencybyte2);   %82 hex msb
             end
    end
%=====================================================================
% Rutina de refresco del número de incrementos del barrido
    function UpdateNumberIncrements(NumberIncrements)
            if (NumberIncrements >= 511)
                NumberIncrements = 511;
            elseif (NumberIncrements <= 0)
                NumberIncrements = 1;
            else
                NumberIncrements = NumberIncrements;
            end
            % Convert Number Increments to 2 Byte local Variables.
            NumberIncrementsbyte0 = int8(bitand(NumberIncrements, 255));
            NumberIncrementsbyte1 = int8(bitand(NumberIncrements, 3840) /
511);

            % Write in data to Number Increments register and verify
            % correct contents written
            WritetToPart (137, NumberIncrementsbyte0);
            WritetToPart (136, NumberIncrementsbyte1);
    end
%=====================================================================
% Rutina de refresco del incremento en frecuencia del barrido
    function
UpdateFrequencyIncrement(DDSRefClockFrequency,FrequencyIncrements)
          TempStartFrequency = ((FrequencyIncrements /
(DDSRefClockFrequency / 4)) * 16777215) * 8;
          TempStartFrequency = floor(TempStartFrequency);
          if (FrequencyIncrements < 0)
            % Convert  increment Frequency to 3 Byte local Variables.
             FrequencyIncrementbyt0 = 0;  %84 hex lsb
             FrequencyIncrementbyt1B = 0; %83 hex
             FrequencyIncrementbyt2 = 0; %82 hex msb

            % Write in data to frequency increment register and verify
            % correct contents written
             WritetToPart (135, FrequencyIncrementbyt0);   %87 hex lsb
             WritetToPart (134, FrequencyIncrementbyt1B);  %86 hex
             WritetToPart (133, FrequencyIncrementbyt2);   %85 hex msb
          else
            %-------------------------------------------------------------
---------------------------
            % Convert Frequencyincrement to 3 Byte local Variables.
             FrequencyIncrementbyt0 = int8(bitand(TempStartFrequency,
255));
             FrequencyIncrementbyt1A = int8(bitand(TempStartFrequency,
65280) / 256);
             FrequencyIncrementbyt2 = int8(bitand(TempStartFrequency,
16711680) / 65536);
```

```
            FrequencyIncrementbyt1B = FrequencyIncrementbyt1A; % TEMPORY
COPY OF FrequencyIncrementbyte1A

            if (FrequencyIncrementbyt1A >= 255)       % is Dim
StartFrequencybyte1a too big to transmit as 1 byte i.e ffhex
                                                      % if it is it must
be broken up resulting in any overflow into a second byte being carried
over to the third byte
                                                      % remember that
the part will only store the bottom three bytes
                  FrequencyIncrementbyt1B =
int8(bitand(FrequencyIncrementbyt1B, 255));
                  MSBfromFrequencyIncrementbyt1B =
int8(bitand(FrequencyIncrementbyt1A, 65280) / 256);
                  FrequencyIncrementbyt2 = int8((FrequencyIncrementbyt2
* 256) + MSBfromFrequencyIncrementbyt1B);
                  FrequencyIncrementbyt2 = int8(((FrequencyIncrementbyt2
* 256) + MSBfromFrequencyIncrementbyt1B) & 255);
             end
             % Write in data to Start frequency register and verify
             % correct contents written
            WritetToPart (135, FrequencyIncrementbyt0);   %87 hex lsb
            WritetToPart (134, FrequencyIncrementbyt1B);  %86 hex
            WritetToPart (133, FrequencyIncrementbyt2);   %85 hex msb
           end
    end
%=========================================================================
% Rutina de refresco del tiempo de muestreo
    function UpdateSettlingTime(SettlingTime)
           if (SettlingTime >= 511)
              SettlingTime = 511;
           else
              SettlingTime = SettlingTime;
           end

           % Convert Number Increments to 2 Byte local Variables.
           SettlingTimebyte0 = int8(bitand(SettlingTime, 255));
           SettlingTimebyte1 = int8(bitand(SettlingTime, 65280) / 511);

           % Write in data to Number Increments register and verify
           % correct contents written
           WritetToPart (139, SettlingTimebyte0);
           WritetToPart (138, SettlingTimebyte1);
    end
%=========================================================================
% Función para poner al chip en estado de reposo
    function EnterStandbyMode
        %-----------------------------------------------------------------
----------------------------------------------------
        %This routine is used program the AD5933/34 into standby mode
where all internal circuitry is powered up but the vin/vout pins are
internally grounded.
        %-----------------------------------------------------------------
----------------------------------------------------
        % Enter Standby Mode Set Bits D15,D13,D12 in Control Register.
        WritetToPart (128, 176);
```

```
    end
%=========================================================================
% Función para la programación del PGA
    function PGA_control(Xi)
        if (Xi==1)
          Gain = 1;
        elseif (Xi==5)
          Gain = 0;
        end
        EnterStandbyMode
        WritetToControlRegister (128, 0);
        WritetToControlRegister (129, 0);
    end
%=========================================================================
% Función para la selección de la tensión de salida
    function OUTPUT_excitation(Range)
        if (Range==1)
            OutputVoltage=0;
        end
        if (Range==2)
            OutputVoltage=6;
        end
        if (Range==3)
            OutputVoltage=4;
        end
        if (Range==4)
            OutputVoltage=2;
        end
    end
%=========================================================================
% Función para la selección del tipo de reloj (externo o interno)
%     function System_clock(clock)
%         if (clock==0)
%             systemclock = 0; % Internal oscillator
%         elseif (clock==1)
%             systemclock = 1; % External clock
%         end
%     end
%=========================================================================
% Multiplicador x 1 para el tiempo de muestreo
    function Multiply_by_one
        Readback_upper_byte_of_settling_time_regC = PortRead(14, 138);
        Readback_upper_byte_of_settling_time_regC =
bitand(Readback_upper_byte_of_settling_time_regC, 1);  % used to isolate
d8 (when set) ,if the user wants to toggle between
        Readback_upper_byte_of_settling_time_regC =
int8(bitor(Readback_upper_byte_of_settling_time_regC, 0));  % various
settings x1 x2 x3
        WritetToPart (138, Readback_upper_byte_of_settling_time_regC);
    end
%=========================================================================
% Multiplicador x 2 para el tiempo de muestreo
    function Multiply_by_two
        Readback_upper_byte_of_settling_time_regC = PortRead(14, 138);
        Readback_upper_byte_of_settling_time_regC =
bitand(Readback_upper_byte_of_settling_time_regC, 1);  % used to isolate
d8 (when set) ,if the user wants to toggle between
```

```
        Readback_upper_byte_of_settling_time_regC =
int8(bitor(Readback_upper_byte_of_settling_time_regC, 2));  % various
settings x1 x2 x3
        WritetToPart (138, Readback_upper_byte_of_settling_time_regC);
    end
%=====================================================================
% Multiplicador x 4 para el tiempo de muestreo
    function Multiply_by_four
        Readback_upper_byte_of_settling_time_regC = PortRead(14, 138);
        Readback_upper_byte_of_settling_time_regC =
bitand(Readback_upper_byte_of_settling_time_regC, 1);  % used to isolate
d8 (when set) ,if the user wants to toggle between
        Readback_upper_byte_of_settling_time_regC =
int8(bitor(Readback_upper_byte_of_settling_time_regC, 6));  % various
settings x1 x2 x3
        WritetToPart (138, Readback_upper_byte_of_settling_time_regC);
    end
%=====================================================================
% Función para el cálculo del valor de la fase según el cuadrante
    function phase_sweep=phase_sweep(img,real)
        %-------------------------------------------------------------
-----------------------------------------------------
        %This routine is used to calculate the theorical phase profile
displayed in the front profile.
        %The routine uses the ideal impedance equations for a
resistor,capacitor, r+c and r||C etc.
        %The equations displayed is determined by the calibration
topology chosen and set by the user in the front panel.
        %-------------------------------------------------------------
-----------------------------------------------------
          if ((real >= 0) && (img >= 0))
              theta = atan(img / real);   % theta = arctan (imaginary
part/real part) 1st quadrant
              phase_sweep = (theta * 180) / pi; %convert to degrees

          elseif ((real >= 0) && (img < 0))
              theta = atan(img / real);    %4th quadrant theta = minus
angle
              phase_sweep = ((theta * 180) / pi) + 360;

          elseif ((real < 0) && (img < 0))
              theta = pi + atan(img / real); %3rd quadrant img/real is
positive
              phase_sweep = (theta * 180) / pi;

          elseif ((real < 0) && (img >= 0))
              theta = pi + atan(img / real); %2nd quadrant img/real is
neg
              phase_sweep = (theta * 180) / pi;
          end
    end
%=====================================================================
% Función para el cálculo del factor de ganancia multipunto
    function
CalculateGainFactor_multipoint(DDSRefClockFrequency,StartFrequency,Freque
ncyIncrements,NumberIncrements,SettlingTime,CalibratedImpedance1)
```

```
        IndexArray = 1;                     %initialise counter variable
        calibration_AveGainFactor = 0;      %initialise counter
variable:All of the multipoint gain factors are stored and used but only
the average is displayed in the front panel.
        Increment = NumberIncrements + 1;   %initialise counter variable:
this variable is used to keep count of the number of increments
"remaining" in the sweep.This variable gets decremented after each sweep.
        Frequency = StartFrequency;         %start frequency is a global
variable containing the start frequency (In hz).The Frequency variable is
used in plotting of |Z| vs Frequency and Ø vs frequency

        EnterStandbyMode                    %put the AD5933/34 into
standby mode, all internal circuitry powered up but the output/input pins
remain internally grounded.

        UpdateStartFrequency(DDSRefClockFrequency, StartFrequency);
%refresh the programmed start frequency to ensure the correct contents
are programmed prior to the sweep commencing.
        UpdateNumberIncrements(NumberIncrements);
%refresh the programmed num of increments to ensure the correct contents
are programmed prior to the sweep commencing.
        UpdateFrequencyIncrement(DDSRefClockFrequency,
FrequencyIncrements);      %refresh the programmed frequency increments
to ensure the correct contents are programmed prior to the sweep
commencing.
        UpdateSettlingTime(SettlingTime);
%refresh the programmed settling time cycles (delay between sweep points
conversions) to ensure the correct contents are programmed prior to the
sweep commencing.

        % Enable external Oscillator
        WritetToControlRegister2(129, 8)

        %----------------------------initialise sensor-------------------
---
        WritetToControlRegister(128, 16)   % issue an "Initialise Sensor
with Start Frequency" command to the control register.
        %pause(0.002);                      % this is a nominal delay
which is user determined by the application/impedance under test.
        %-------------------------start the frequency sweep--------------
----
        WritetToControlRegister(128, 32)   % issue an "Start Frequency
Sweep" command to the control register.
        %-----------------------------------------------------------------
----

        % Enter Frequency Sweep Loop
        ReadbackStatusRegister = PortRead(14, 143);
        ReadbackStatusRegister = bitand(ReadbackStatusRegister, 4); %
mask off bit D2 (Frequency sweep complete)
        while ((ReadbackStatusRegister ~= 4) && (Increment ~= 0))
            % check to see if current sweep point complete
            ReadbackStatusRegister = PortRead(14, 143);
            ReadbackStatusRegister = bitand(ReadbackStatusRegister, 2); %
mask off bit D1  (valid real and imaginary data availab to read)
```

```
            %-------------------------------------------------------------
------------
            if (ReadbackStatusRegister == 2)
                % this sweep point has returned valid data so we can
proceed with the sweep
            else
                %Do
                while 1
                    %if valid data has not been returned then we need to
pole stat reg until such time as valid data
                    %has been returned
                    %i.e. if point is not complete then Repeat sweep
point and pole staus reg until valid data returned
                    WritetToControlRegister (128, 64); %repeat sweep
point
                           %Do
                           while 1
                                ReadbackStatusRegister = PortRead(14,
143);
                                ReadbackStatusRegister =
bitand(ReadbackStatusRegister, 2); % mask off bit D1- Wait until dft
complete
                                if (ReadbackStatusRegister==2)
                                    break
                                end
                           %Loop While (ReadbackStatusRegister ~= 2)
                           end
                    if (ReadbackStatusRegister==2)
                        break
                    end
                %Loop Until (ReadbackStatusRegister = 2)
                end
            end
            %-------------------------------------------------------------
-------------
            RealDataUpper = PortRead(14, 148);
            RealDataLower = PortRead(14, 149);
            RealData = RealDataLower + (RealDataUpper * 256);
            %The Real data is stored in a 16 bit 2's complement format.
            %In order to use this data it must be converted from 2's
complement to decimal format
            if RealData <= 32767
                % Positive
            else
                % Negative
                RealData = RealData - 65536;
            end

            %The imaginary data is stored in a 16 bit 2's complement
format.
            %In order to use this data it must be converted from 2's
complement to decimal format
            ImagineryDataUpper = PortRead(14, 150);
            ImagineryDataLower = PortRead(14, 151);
            ImagineryData = ImagineryDataLower + (ImagineryDataUpper *
256);
```

```
            if ImagineryData <= 32767
                % Positive
            else
                % Negative
                ImagineryData = ImagineryData - 65536;
            end

            Magnitude = ((RealData ^ 2) + (ImagineryData ^ 2)) ^ 0.5;
%calculates the magnitude of the dft real and imaginary components.The
gain factor = Admittance/"magnitude"
            sweep_phase = phase_sweep(ImagineryData, RealData);
%calculates the phase of the dft real and imaginary components
            %-------------------------------------------------------------
--------
            GainFactor_multipoint = 1 / (CalibratedImpedance1 *
Magnitude);

            %-------------------------------------------------------------
-------------------
            calibration_GainFactors(IndexArray) = GainFactor_multipoint;
%write gain factor Data to global array.
            calibration_phase_multi_point(IndexArray) = sweep_phase;
%write phase data to global array.
            Increment = Increment - 1;
%incremet was set to number of increments+1 of sweep at the start
            FrequencyPoints(IndexArray) = Frequency;
%write frequency data to global array.
            Frequency = Frequency + FrequencyIncrements;
%holds the current value of the current sweep frequency
            IndexArray = IndexArray + 1;
%increment variable

            %------------- Check to see if sweep complete ---------------
-------
            ReadbackStatusRegister = PortRead(14, 143);
            ReadbackStatusRegister = bitand(ReadbackStatusRegister, 4); %
mask off bit D2
            % Increment to next frequency point
            WritetToControlRegister (128, 48);                    % issue
an "increment frequency command" to the control regsister, if the sweep
is complete the device will ignore this command.
        end

        %-------------------------------------------------------------
-------------------------------------------------------------------------
-----------------------------------------------------
        %----------------------The multipoint sweep used to determine the
multipoint gain factors is complete at this point-----------------------
-----------------------------------------------------
        %-------------------------------------------------------------
-------------------------------------------------------------------------
-----------------------------------------------------

%        % calculate the average gain factor to display in the box: we
use all the gain factors in a multipoint sweep but we only display the
avaerage gain factor
```

```
%         for i = 1: IndexArray -1
%           calibration_AveGainFactor = calibration_AveGainFactor +
calibration_GainFactors(i);
%         end
%         calibration_AveGainFactor=calibration_AveGainFactor /
IndexArray;
    end
%=====================================================================
% Función para el barrido en frecuencia
function
Sweep_single(DDSRefClockFrequency,StartFrequency,FrequencyIncrements,Numb
erIncrements,SettlingTime)
            IndexArray = 1;                     %initialise counter
variable
            Increment = NumberIncrements + 1;   %initialise counter
variable: this variable is used to keep count of the number of increments
"remaining" in the sweep.This variable gets decremented after each sweep.
            Frequency = StartFrequency;         %start frequency is a
global variable containing the start frequency (In hz).The Frequency
variable is used in plotting of |Z| vs Frequency and Ø vs frequency

            EnterStandbyMode;

            UpdateStartFrequency(DDSRefClockFrequency, StartFrequency);
%refresh the programmed start frequency to ensure the correct contents
are programmed prior to the sweep commencing.
            UpdateNumberIncrements(NumberIncrements);
%refresh the programmed num of increments to ensure the correct contents
are programmed prior to the sweep commencing.
            UpdateFrequencyIncrement(DDSRefClockFrequency,
FrequencyIncrements);   %refresh the programmed frequency increments to
ensure the correct contents are programmed prior to the sweep commencing.
            UpdateSettlingTime(SettlingTime);
%refresh the programmed settling time cycles (delay between sweep points
conversions) to ensure the correct contents are programmed prior to the
sweep commencing.

            % Enable external Oscillator
            WritetToControlRegister2(129, 8)

            %----------------------------initialise sensor---------------
-------
            WritetToControlRegister(128, 16)   % issue an "Initialise
Sensor with Start Frequency" command to the control register.
          % pause(0.002);                       % this is a nominal delay
which is user determined by the application/impedance under test.
            %-------------------------start the frequency sweep-----------
--------
            WritetToControlRegister(128, 32)   % issue an "Start
Frequency Sweep" command to the control register.
            %-----------------------------------------------------------
--------

            % Enter Frequency Sweep Loop
            ReadbackStatusRegister = PortRead(14, 143);
```

```
            ReadbackStatusRegister = bitand(ReadbackStatusRegister, 4); %
mask off bit D2 (Frequency sweep complete)
            while ((ReadbackStatusRegister ~= 4) && (Increment ~= 0))
                % check to see if current sweep point complete
                ReadbackStatusRegister = PortRead(14, 143);
                ReadbackStatusRegister = bitand(ReadbackStatusRegister,
2); % mask off bit D1  (valid real and imaginary data availab to read)
                %-------------------------------------------------------
----------------
                if (ReadbackStatusRegister == 2)
                    % this sweep point has returned valid data so we can
proceed with the sweep
                else
                    %Do
                    while 1
                        %if valid data has not been returned then we need
to pole stat reg until such time as valid data
                        %has been returned
                        %i.e. if point is not complete then Repeat sweep
point and pole staus reg until valid data returned
                        WritetToControlRegister (128, 64); %repeat sweep
point
                                %Do
                                while 1
                                     ReadbackStatusRegister = PortRead(14,
143);
                                     ReadbackStatusRegister =
bitand(ReadbackStatusRegister, 2); % mask off bit D1- Wait until dft
complete
                                     if (ReadbackStatusRegister==2)
                                         break
                                     end
                                %Loop While (ReadbackStatusRegister ~= 2)
                                end
                        if (ReadbackStatusRegister==2)
                            break
                        end
                    %Loop Until (ReadbackStatusRegister = 2)
                    end
                end
                %-------------------------------------------------------
----------------
                RealDataUpper = PortRead(14, 148);
                RealDataLower = PortRead(14, 149);
                RealData = RealDataLower + (RealDataUpper * 256);
                %The Real data is stored in a 16 bit 2's complement
format.
                %In order to use this data it must be converted from 2's
complement to decimal format
                if RealData <= 32767
                    % Positive
                else
                    % Negative
                    RealData = RealData - 65536;
                end
```

```
                %The imaginary data is stored in a 16 bit 2's complement
format.
                %In order to use this data it must be converted from 2's
complement to decimal format
                ImagineryDataUpper = PortRead(14, 150);
                ImagineryDataLower = PortRead(14, 151);
                ImagineryData = ImagineryDataLower + (ImagineryDataUpper
* 256);

                if ImagineryData <= 32767
                    % Positive
                else
                    % Negative
                    ImagineryData = ImagineryData - 65536;
                end

                %***
                Magnitude = ((RealData ^ 2) + (ImagineryData ^ 2)) ^ 0.5;
%calculates the magnitude of the dft real and imaginary components.The
gain factor = Admittance/"magnitude"
                sweep_phase = -(calibration_phase_multi_point(IndexArray)
- phase_sweep(ImagineryData, RealData));
                GainFactor = calibration_GainFactors(IndexArray);
                Impedance = 1 / (Magnitude * GainFactor);

                % Write Data to each global array.
                MagnitudeArray(IndexArray) = Impedance;
                PhaseArray(IndexArray) = sweep_phase;
                ImagineryDataArray(IndexArray) = ImagineryData;
                %code(IndexArray) = Magnitude;
                RealDataArray(IndexArray) = RealData;
                Increment = Increment - 1; % incremet was set to number
of increments of sweep at the start
                FrequencyPoints(IndexArray) = Frequency;
                Frequency = Frequency + FrequencyIncrements; % holds the
current value of the sweep freq
                IndexArray = IndexArray + 1;
%increment variable

                %------------- Check to see if sweep complete -----------
-----------
                ReadbackStatusRegister = PortRead(14, 143);
                ReadbackStatusRegister = bitand(ReadbackStatusRegister,
4); % mask off bit D2
                % Increment to next frequency point
                WritetToControlRegister (128, 48);                  %
issue an "increment frequency command" to the control regsister, if the
sweep is complete the device will ignore this command.
            end
end
%=====================================================================
% Función para selección de la frecuencia del DDS1
    function selec_frecuencia_DDS(frecuencia_DDS)
           switch frecuencia_DDS
             case 25000
                  PortWrite(15,128,1);   % CLK=25 kHz
```

```
            case 50000
                 PortWrite(15,128,2);   % CLK=50 kHz
            case 250000
                 PortWrite(15,128,3);   % CLK=250 kHz
            case 1000000
                 PortWrite(15,128,4);   % CLK=1 MHz
            case 2000000
                 PortWrite(15,128,5);   % CLK=2 MHz
            case 4000000
                 PortWrite(15,128,6);   % CLK=4 MHz
            case 16000000
                 PortWrite(15,128,7);   % CLK=16 MHz
          end
    end
%=========================================================================
% Función para la selección de las resistencias patrón por los
% multiplexores
    function select_mux(patron)
            switch patron
              case 1
                PortWrite(17,128,1);   % mux ZBF=2k
                PortWrite(16,128,1);   % mux ZIN=2k
                r_cal=2000;
              case 2
                PortWrite(17,128,2);   % mux ZBF=3k
                PortWrite(16,128,2);   % mux ZIN=3k
                r_cal=3000;
              case 3
                PortWrite(17,128,3);   % mux ZBF=4k
                PortWrite(16,128,3);   % mux ZIN=4k
                r_cal=4000;
              case 4
                PortWrite(17,128,4);   % mux ZBF=5k
                PortWrite(16,128,4);   % mux ZIN=5k
                r_cal=5000;
              case 5
                PortWrite(17,128,5);   % mux ZBF=6k
                PortWrite(16,128,5);   % mux ZIN=6k
                r_cal=6000;
              case 6
                PortWrite(17,128,6);   % mux ZBF=7k
                PortWrite(16,128,6);   % mux ZIN=7k
                r_cal=7000;
              case 7
                PortWrite(17,128,7);   % mux ZBF=8k
                PortWrite(16,128,7);   % mux ZIN=8k
                r_cal=8000;
              case 8
                PortWrite(17,128,8);   % mux ZBF=9k
                PortWrite(16,128,8);   % mux ZIN=9k
                r_cal=9000;
              case 9
                PortWrite(17,128,9);   % mux ZBF=10k
                PortWrite(16,128,9);   % mux ZIN=10k
                r_cal=10000;
              case 10
                PortWrite(17,128,10);   % mux ZBF=11k
```

```
                PortWrite(16,128,10);   % mux ZIN=11k
                r_cal=11000;
              case 11
                PortWrite(17,128,11);   % mux ZBF=12k
                PortWrite(16,128,11);   % mux ZIN=12k
                r_cal=12000;
              case 12
                PortWrite(17,128,12);   % mux ZBF=13k
                PortWrite(16,128,12);   % mux ZIN=13k
                r_cal=13000;
              case 13
                PortWrite(17,128,13);   % mux ZBF=14k
                PortWrite(16,128,13);   % mux ZIN=14k
                r_cal=14000;
              case 14
                PortWrite(17,128,14);   % mux ZBF=15k
                PortWrite(16,128,14);   % mux ZIN=15k
                r_cal=15000;
              case 15
                PortWrite(17,128,15);   % mux ZBF=16k
                PortWrite(16,128,15);   % mux ZIN=16k
                r_cal=16000;
            end
    end
%=========================================================================
end
```

ANEXO 8: Artículo en la revista SENSORS (Q1, IF=3,275).
https://doi.org/10.3390/s20205932

Characterization and Differentiation between Olive Varieties through Electrical Impedance Spectroscopy, Neural Networks and IoT

José Miguel Madueño Luna [1,*], Antonio Madueño Luna [2] and Rafael E. Hidalgo Fernández [3]

[1] Graphics Engineering Department, University of Seville, 41013 Seville, Spain
[2] Aerospace Engineering and Fluid Mechanical Department, University of Seville, 41013 Seville, Spain; antonio@madueno.es
[3] Graphics Engineering and Geomatics Department, University of Córdoba, 14014 Córdoba, Spain; ig1hifer@uco.es
* Correspondence: jmadueno@us.es

Received: 20 September 2020; Accepted: 16 October 2020; Published: 20 October 2020

Abstract: Electrical impedance has shown itself to be useful in measuring the properties and characteristics of agri-food products: fruit quality, moisture content, the germination capacity in seeds or the frost-resistance of fruit. In the case of olives, it has been used to determine fat content and optimal harvest time. In this paper, a system based on the System on Chip (SoC) AD5933 running a 1024-point discrete Fourier transform (DFT) to return the impedance value as a magnitude and phase and which, working together with two ADG706 analog multiplexers and an external programmable clock based on a synthesized DDS in a FPGA XC3S250E-4VQG100C, allows for the impedance measurement in agri-food products with a frequency sweep from 1 Hz to 100 kHz. This paper demonstrates how electrical impedance is affected by the temperature both in freshly picked olives and in those processed in brine and provides a way to characterize cultivars by making use of only the electrical impedance, neural networks (NN) and the Internet of Things (IoT), allowing information to be collected from the olive samples analyzed both on farms and in factories.

Keywords: electrical impedance; SoC AD5933; artificial neural networks (ANNs); internet of things (IoT); temperature

1. Introduction

1.1. The Effect of Temperature

Temperature affects the texture of fruit and vegetables, both raw [1] and cooked [2], and it also affects their electrical parameters such as the electrical impedance of their pulp. A wide variety of procedures and equipment [3,4] exist to obtain various parameters associated with the pulp of these fruits and vegetables (penetration resistance, viscosity, etc.)

In the case of olives, there are no studies explicitly analyzing the influence of temperature on the fruit, especially in its electrical parameters, although its effect is known in certain mechanized

olive processing procedures. This is the case, for example, in the industrial olive pitting process [5], which is done by automatic machines where clamps are used to trap the olive, while a punch needle goes through the clamps and, consequently, the olives, causing them to be de-stoned. This type of machine reaches a considerable pitting speed, which obviously means a minimum cost per unit or fruit de-stoned. However, the very clamps that hold the fruit in place and the de-stoning tool cause breakage in a considerable number of olives during normal operation, which can be to the order of 14%, especially in the "Gordal Sevillana" olive variety. While the productivity of these pitting machines makes them more cost-effective despite the percentage of olive breakage compared to previous de-stoning systems, these machines present a serious problem, as the percentage of broken olives is quite considerable.

While mechanical solutions reducing the percentage of olive breakage in such de-stoning machines do not exist, it has been proven that cooling the olives prior to pitting minimizes the problem to the point that, with adequate refrigeration, the percentage of broken olives during the pitting process is to the order of 2%, for the "Gordal Sevillana" olive variety previously cited.

Furthermore, the current consumer trend for lower salt and acidity [6] means that a significant proportion of the olives deteriorate during the time they spend in fermenters. After a few months at an average temperature of 25 °C, the probabilities of having a significant loss of product in the pitting process are very high, due to a loss in firmness in the olives, especially the larger ones, which are the most in demand on the market. By cooling to temperatures close to 7 °C (in some cases it reaches 0 °C, depending on the cooling system and the costs the company wants to take on in the process), it is possible to increase the firmness of the olives while maintaining all of their characteristics, thereby reducing the loss of product due to the olives having a more rigid texture. In addition, this cuts down on the hours the machine is working, meaning a decrease in machine jams and hours when the punch needles are stopped.

One way of characterizing the state of the olives prior to de-stoning would be by doing an electrical impedance measurement on a representative sample. As will be demonstrated in this study, the electrical impedance of the olive pulp is not only affected by the olive variety or the type of industrial processing (green "Sevillana Style" or black "Californian Style" olives) [7], but also by the temperature. For this reason, characterizing this electrical parameter versus the temperature (especially between 7 °C and 0 °C) for a given type of industrial processing would show whether or not olives are apt before pitting.

1.2. Electrical Impedance: Measurement

The determination of electrical properties is used in a wide range of disciplines and industries [8]. In the agri-food sector, the use of electrical conductivity is applied for the determination of diverse characteristics in agri-food products [9,10] such as frost sensitivity, freezing tolerance, moisture content, and seed germination [11]. The use of electrical impedance spectroscopy is a technique that can provide very good results in the maturity stage of fruit and vegetables. In this paper, we are going to focus on a System on Chip (SoC) capable of a complex impedance measurement: the SoC AD5933 [12].

Impedance is a parameter of great importance to the characterization of circuits and electronic components [13], as well as the materials used in their production. Impedance (Z) is generally defined as the total opposition a device or circuit offers to the flow of an alternating current (AC) to a specific frequency, and it is represented as a complex number with a graphic representation on a complex plane. An impedance vector consists of the real part (resistance, R) and the imaginary part (reactance, X). Impedance can be expressed by using the rectangular coordinates in the form of $R + j \cdot X$, or in the polar form as a magnitude and phase angle: $|Z| \angle Ø$ [14].

The instruments most commonly used to measure impedance are: the LCR meter or LCR bridge and the impedance analyzer. The first provides a simple and exact impedance measurement for a specific frequency value. However, for components other than pure inductors (L), capacitors ©, or resistors (R), it is inadequate to determine value. In these cases, an impedance analyzer is used to measure and graphically represent the complex impedance of the device being tested over a range of frequencies [15]. As they are high cost devices [16,17] , the reason why designs using LCR meters (which are cheaper) exist, is to obtain a system impedance analyzer combining it with virtual instrumentation [12,18].

There are various configurations for the design of impedance measuring bridges such as Schering [19] and the Maxwell bridge [20]. The difficulty with these is that they need to have the balance condition. Moreover, they are generally used for pure inductive or capacitive impedance measurements. In order to obtain the complex impedance, electronic methods are used, such as the vectoral method and the method using two quadrature sinusoidal waves [21,22].Another way to measure the complex impedance is the three-voltage method [23]; however, this requires voltages to be raised to the square, which makes the measurement errors greater, in addition to the fact that it requires very precise instruments to run the measurements.

An Impedance/Gain-Phase analyzer [24] is a measuring instrument of great value for the study and design of electronic circuits. This powerful device is capable of obtaining diagrams separately, both in magnitude and in phase of any network or electronic circuit which has an input and an output. With this information, it is possible, for example, to obtain the transfer function of a specific circuit, although its implementation is not known.

An Impedance/Gain-Phase analyzer should be capable of obtaining a reliable amplitude and phase diagrams corresponding to a specific circuit. In both cases, these variables will be represented according to frequency. These types of analyzers must be capable of generating a sinusoidal signal and directly applying it to the input of the network to be measured. Thus, a frequency sweep is done on the network with a specific criterion regarding the initial, intermediate, final frequency values, number of points, linear or logarithmic sweep, etc., which is normally selected by the user. To generate the magnitude diagram, it is necessary to find the quotient between the amplitudes of the network output and input for each of the values to be measured in the frequency sweep. Furthermore, in order to do the phase diagram, it is necessary to obtain the phase difference between the network output and input signals again for each of the frequency values. Therefore, it is obvious that in order to build an impedance/gain-phase analyzer, a sine wave generator is necessary, at the very least, to generate a circuit frequency sweep capable of measuring amplitude and of another circuit capable of measuring the phase difference between two signals.

1.3. Previous Examples of the Use of Electrical Impedance in Olive Production and Other Fruits

Electric conductivity has been used (only its magnitude) to characterize different olive varieties (*Olea europa* L.) [25], specifically, four different cultivars: “Picual”, “Manzanilla de Sevilla”, “Hojiblanca” and “Gordal Sevillana” with the objective of establishing a fruit maturity index and, thus of determining the optimal time for harvesting based on parameters such as oil yield and quality, demonstrating that the conductivity increased with fruit maturity and that each variety had an average characteristic electric conductivity value in the last stages of maturation. Previous studies have focused in particular on determining maturity indexes and quality parameters [7,26,27], or on ways of slowing the maturation period, above all in climacteric fruits, with the aim of lengthening the period between harvesting and commercial consumption [28].

1.4. Neural Networks (NN) for Adjustment and Sorting

Artificial neural networks are computational models inspired by the behavior observed in their biological counterparts [29]. Their use is widespread in an abundance of fields. Focusing on the table olive, we can see their use in [30] or in [31].

In this study, we are going to use two types of neural network. On the one hand, a neural network for adjustment [32] to evaluate the effectiveness of this technique in generating a valid impedance behavior mode for olive pulp and, on the other hand, a neural network for sorting [33] to distinguish between olive varieties at different temperatures.

1.5. The Internet of Things (IoT)

The use of the IoT is widely extended nowadays and there are numerous cases (for example [34–38]) in precision agriculture, irrigation, temperature control, monitoring of the agricultural production process, in automated olive chain processes, or in the operation of the pitting, slicing, and stuffing machines themselves (DRR) [30,31]. This paper shows the development of equipment based on the SoC AD5933 [12] which, together with a neural network and an IoT system, allows for the analysis of the state of the olives on the farm or in the factory before pitting.

The general objective set out by this paper is to develop an electrical impedance measurement system adapted to table olives, making use of the SoC AD5933. In order to do this, two tasks are going to be done:

(1) Impedance modeling through neural networks for two varieties of olives ("Gordal Sevillana" and "Hojiblanca"), cured in caustic soda and fermented in brine (an industrial process known as "Estilo Sevillano").
(2) Classification via neural networks for each olive variety at three temperatures (25 °C, 7 °C, and 0 °C).

To this end, these systems will be developed:

- A specific device with the SoC AD5933 [12], that includes an I²C communication interface [39], an external DDS generator based on a FPGA XC3S250E-4VQG100C [40] to conduct a complete sweep from 1 Hz to 100 kHz and two ADG706 analog multiplexers [41] to set the impedance range to be measured. This device is controlled by a 32-bit ARM CORTEX M3 AT91SAM 3 × 8 E microcontroller working at 84 MHz [42].
- The system control software using Matlab programing language [43].
- IoT communication is based on [44] to generate a database with the trial results.

2. Materials and Methods

2.1. Olive Varieties and Industrial Process Used

The olive fruit (*Olea europaea* L.) [25] is an ovoid drupe whose size oscillates from 0.6 to 2 cm in diameter and from 1 to 4 cm in length. This size depends on the variety (see Figure 1 in [25]), the vegetative state of the tree, the environmental conditions, and the cultivation techniques [45]. We used samples of 100 unripe olives from olive orchards (200 trees ha^{-1}, Seville, Spain), under irrigation and non-limiting nutrient conditions with mechanical harvesting, and samples of 100 olives in brine from a factory in Seville. In this work we used two varieties: "Hojiblanca" and "Gordal Sevillana" freshly picked in class 0, 1, and 2 [46] and processed in brine the "Estilo Sevillano" way. This type of olive, seasoned this way, is processed in four stages [47–49]:

- Lye treatment in NaOH 2–4% (p/v) for 6–12 h
- Rinsed in water (12–15 h)

- Fermented in brine (10–12% (p/v) for 60–300 days)
- Pitted and stuffed or sliced.

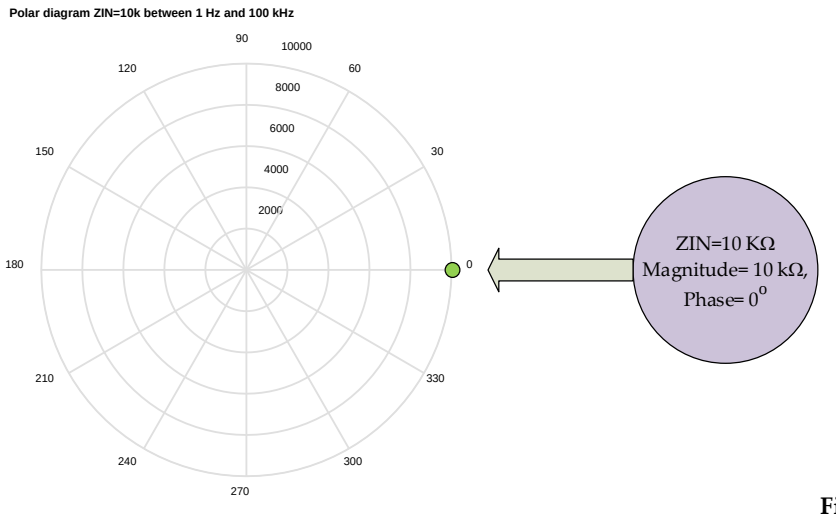


Figure 1. Polar diagram: Complete sweep from 1 Hz to 100 kHz with ZIN = 10 KΩ (Magnitude 10 kΩ, Phase 0°).

2.2. The SoC AD5933 to Measure Impedance

According to the SoC AD5933 datasheet [12], "the AD5933 is a high precision impedance converter combining an integrated frequency generator with a 12-bit and 1 MSPS analogue-to-digital converter (ADC)". The impedance of the device under test (DUT) is sampled by the internal ADC and is processed with a discrete Fourier transform (DFT)" [50]. The output excitation voltage and the measurement frequency are totally programmable. Communication is done via an I2C interface.

Reviewing the available bibliography confirms that the SoC AD5933 has significant biological applications, such that it has been used to monitor the growth of cell cultures [51,52] in measurements in isolated cells [53], detection of blood clotting [54], biosensor applications [55], and bio-impedance measurements [56–60]. Likewise, it is used in the monitoring of "technical objects", for example, for corrosion analysis in steel structures [61,62]. In order to obtain comprehensive information about the electrical properties of a measured object and to use a suitable impedance spectrum analysis method (equivalent circuit modelling), impedance must be measured at a wide range of frequencies [63].

In Table 1, the technical data of various impedance meters based on the SoC AD5933 are shown. Only three of the devices described allow impedance were to be measured in more than three orders of magnitude of frequency. The impedance range measured is typically from 10 Ω to more than 10 MΩ. However, in many cases, it does not give the exact range. The majority of the impedance meters mentioned require additional analog front-ends to provide an adequate interface between the SoC AD5933 and the device under test (DUT) [64].

Table 1. List of electrical impedance meters based on the SoC AD5933 (taken from [64]).

Author	Purpose	Frequency Range	Impedance Range	Maximum Error
C. J. Chen et al. [51]	Monitoring Cell Cultures	Set to 10 Hz	Not specified	Not specified
T. Schwarzenberger et al. [52]	Monitoring Cell Cultures	100 Hz–100 kHz	Not specified	2%–magnitude, 2%–argument

M. H. Wang et al. [53] (uses an AD5934)	Measuring Isolated Cells	0.1 Hz–100 kHz	100 Ω–10 MΩ	Around 10% for cell measurement
J. Broeders et al. [55]	Biosensor Application	10 Hz–100 kHz	10 Ω–5 MΩ	Not specified
P. Bogónez-Franco et al. [57]	Bioimpedance Monitor	100 Hz–200 kHz	10 Ω–1 kΩ	2.5%–magnitude, 4.5%–argument
J. Ferreira et al. [58]	Bioimpedance Electrodes In Clothing	5 kHz–450 kHz	Not specified	0.7%–resistance, 17%–reactance
C. Margo et al. [59]	"Embedded" applications for bioimpedance	1 kHz–100 kHz	Not specified. No data	2.5%–magnitude, 1.3%–argument
A. Melwin y K. Rajasekaran [60]	Body composition measurements	Set to 50 kHz	Not specified	2% (not specified)
J. Hoja y G. Lentka [61,62]	Object monitoring technique	0.01 Hz–100 kHz	10 Ω–10 GΩ	1.6%–magnitude, 0.6%–argument

Other authors [65–67] proposed modifications in the original topology provided by the manufacturer with the use of a multiplex system to adjust the range of the impedance measurement, although this does not include changes in the source clock input into the SoC AD5933. In [68] the operation of the SoC AD5933, the hardware and software developed, and the IoT system are described.

2.3. Neural Networks to Modeling and Sorting

Two types of neural networks were used from the Matlab libraries:

- ◎A neural network for adjustment, "fitnet" [32], allows a description of the evolution of the complex impedance in the pulp of 2 varieties of olives to be obtained. It is a considerable improvement over the Hayden model [69].
- ◎A sorting network, "patternnet" [33], to distinguish between 6 cases (2 varieties and 3 temperatures) in unripe and another 6 cases (2 varieties and 3 temperatures) in olives processed the "Estilo Sevillano" way.

3. Results

Three verification tests of the developed hardware have been carried out (Sections 3.1–3.3) as well as a study on the effect of variety on electrical impedance in unripe olives (Section 3.4) and on olives in brine (Section 3.5). A model with neural networks on the evolution of impedance in unripe and brined olives of the two varieties studied in Section 3.6. and a classifier based on neural networks capable of distinguishing between 6 different cases in unripe olives was trained (Section 3.7.1), in brined olives (Section 3.7.2), and finally (Section 3.8) the IoT system used is shown.

3.1. Impedance Meter Verification Testing: DUT Made Up of a Pure Resistance of 10 kΩ

A test has been done using a pure resistance of 10 KΩ, the results are in Figure 1. The object of the test is to make sure that the equipment is working correctly. As can be seen, the answer is the same in both the magnitudes as in the phase throughout the sweep from 1 Hz to 100 kHz.

3.2. Impedance Meter Verification Testing: DUT Comprised of a SERIAL RLC Circuit

We conducted a test on the SERIAL RLC circuit with an inductance of L = 56 mH, C = 1500 pF and R = 5.1 kΩ, with Zcal = 10 k, ZBF = 10 k, Range 1: 2 Vpp, PGA = ×1, Multiplier = ×1 from 1 Hz to 100 kHz (see Tables 1 and 2 in [68]).

For the test, three elements were connected in series to build a typical RLC circuit. The theoretical resonance frequency is:

$$f_o = \frac{1}{2\cdot\pi\cdot\sqrt{L\cdot C}} = \frac{1}{2\cdot\pi\cdot\sqrt{56\times10^{-3}\times1500\times10^{-12}}} = 17365.22\ \text{Hz} \quad (2)$$

A low frequency predominates the capacitive part with negative phases that tend to −90°. In the proximity of the resonance zone, the phase tended to zero and from that moment the inductive part began to predominate, the phase in that case positive tends to 90°. The theoretical and experimental results are shown in Figure 2. The small discrepancies between theoretical and experimental data are due to the tolerance of the components used in the real test.

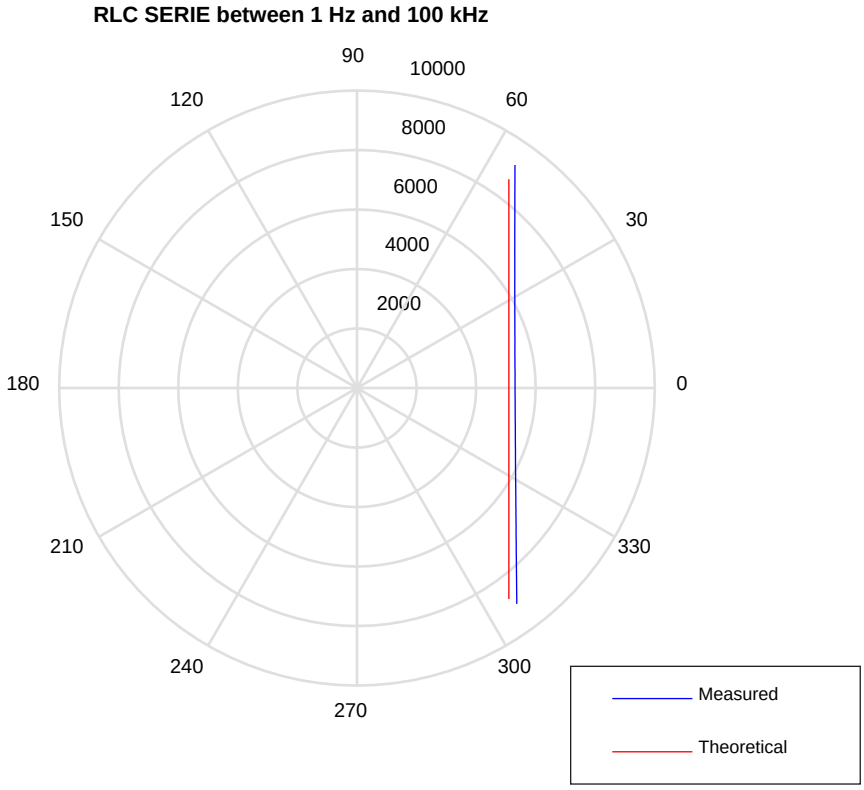


Figure 2. Polar diagram of the test with SERIAL RLC circuit between 1 Hz and 100 KHz. The blue values are measured and the red ones are theoretical.

3.3. Impedance Meter Verification Testing: DUT Comprised of a PARALLEL RLC Circuit

The test on the PARALLEL RLC circuit with an inductance of L = 56 mH, C = 1500 pF and R = 5.1 kΩ, with Zcal = 10 k, ZBF = 10 k, Range 1: 2 Vpp, PGA = ×1, Multiplier = ×1 from 1 Hz to 100 kHz (see Tables 1 and 2 in [68]).

For this test, three elements have been connected in parallel to build a typical RLC circuit in parallel. The theoretical frequency will be the same as in the previous test.

The theoretical and experimental results are shown in Figure 3. Once again, the differences are due to the tolerance of the components used.

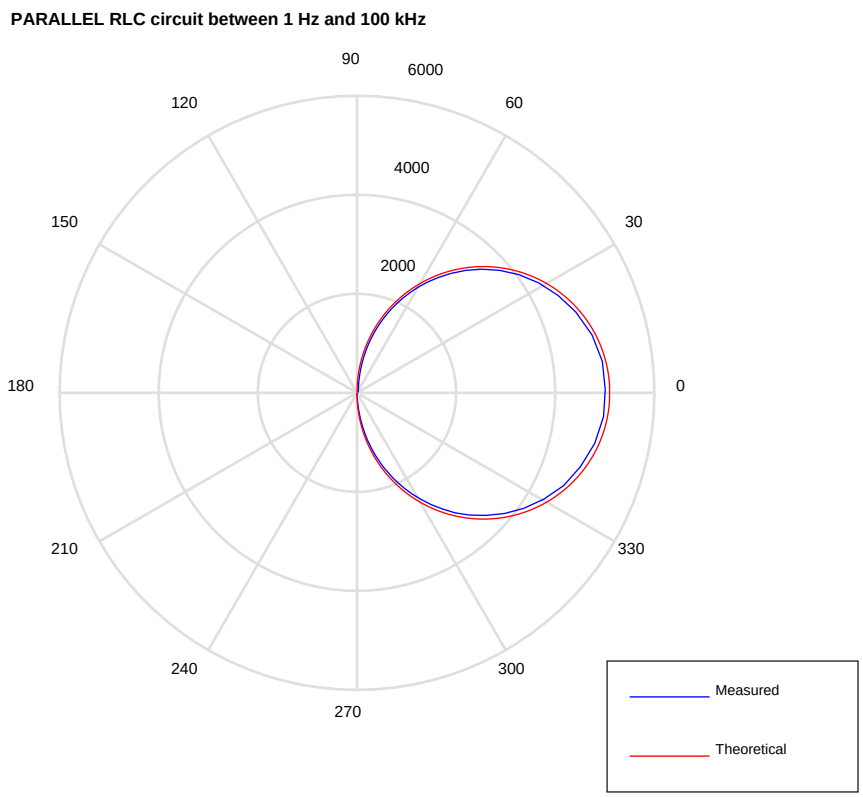


Figure 3. Polar diagram of the test with the PARALLEL RLC circuit between 1 Hz and 100 kHz. The blue values are measured and the red ones are theoretical.

3.4. Test on Unripe Olives: The Effect of Olive Variety on Electric Impedance

This study has dealt with olive samples from the "Gordal Sevillana" and "Hojiblanca" varieties, shown in Figure 4, from two olive trees situated side by side and therefore subject to the same environmental and watering conditions. These samples were harvested on the same day at the same time (16/06/2019).

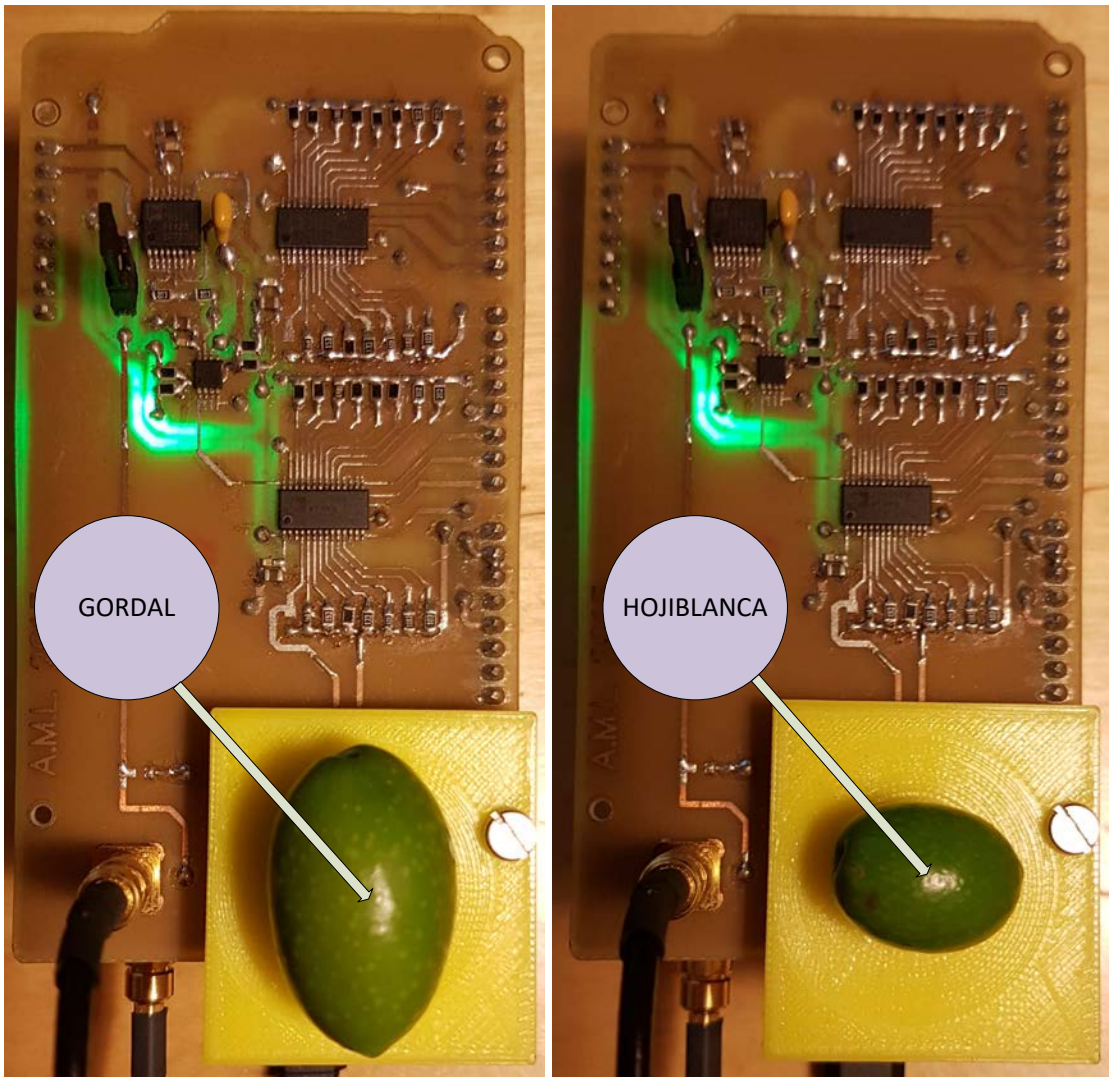
In Figure 5, the evolution of the impedance profile obtained (an average of 100 tests) appears presented as its components X-R for unripe olives of both varieties at three temperatures: 0 °C (blue), 7 °C (yellow), and 25 °C (red).

As can be seen, they present a characteristic profile differentiated both by temperature and maximum values, the "Hojiblanca" being the one with higher values in both components X, R, and thereby the module Z, both at 0 °C and 7 °C, while at 25 °C its profile is similar to the "Gordal Sevillana".

Previous tests have demonstrated that these impedance profiles do not adjust well to models of the type described [70] (see Figure 3b of said citation) with minimum at X close to zero. Conversely, in unripe olives at a low frequency, there are high values in both R and X (red circles on the graphs). Likewise, two relative minimums can be observed in the reactance X value (green diamonds) at 7 °C and 25 °C. The trend on the graph of 0 °C suggests that at frequencies over 100 kHz it would also exist in this case.

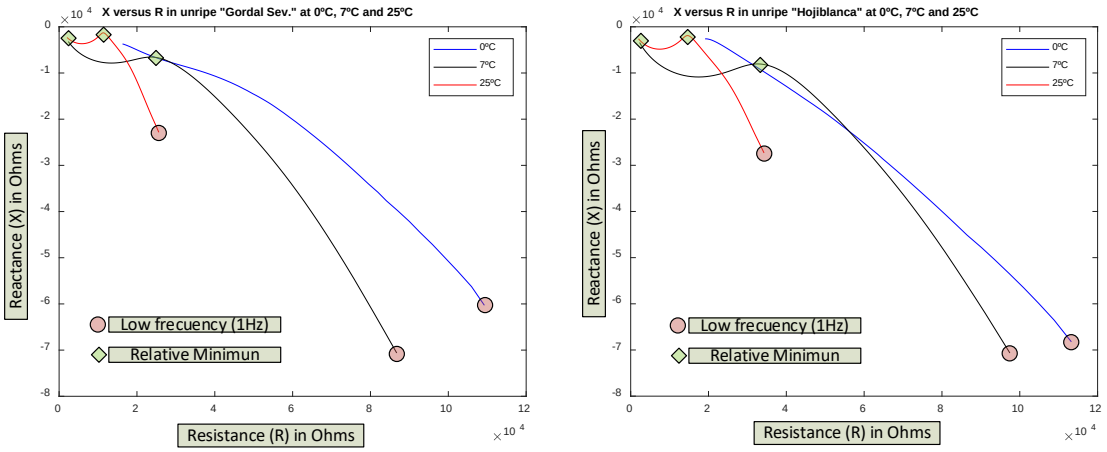
One of the problems that sometimes appears is distinguishing between the unripe varieties, due to the fact that their outward appearance and size cause doubt. This test demonstrated how these varieties (unripe) affect the electrical impedance the same as the rest of the parameters

(especially temperature). This system would allow the sample variety of olive to be identified from the electrical impedance without using other characterized parameters.



(**a**) (**b**)

Figure 4. (**a**) "Gordal Sevillana" and (**b**) "Hojiblanca" olive varieties during the electrical impedance measurement.



(**a**) (**b**)

Figure 5. Evolution of the impedance in (**a**) unripe "Gordal Sevillana" and (**b**) "Hojiblanca" varieties at 3 temperatures: 0 °C, 7 °C, and 25 °C (average of 100 tests). The red circles corresponds to f = 1 Hz.

In Figure 6, the impedance spectrum of both varieties is shown together (in polar diagram). Each one has a characteristic profile at each temperature and the maximum difference is found at low frequencies, which could serve to distinguish between unripe olive varieties as is also indicated in [25].

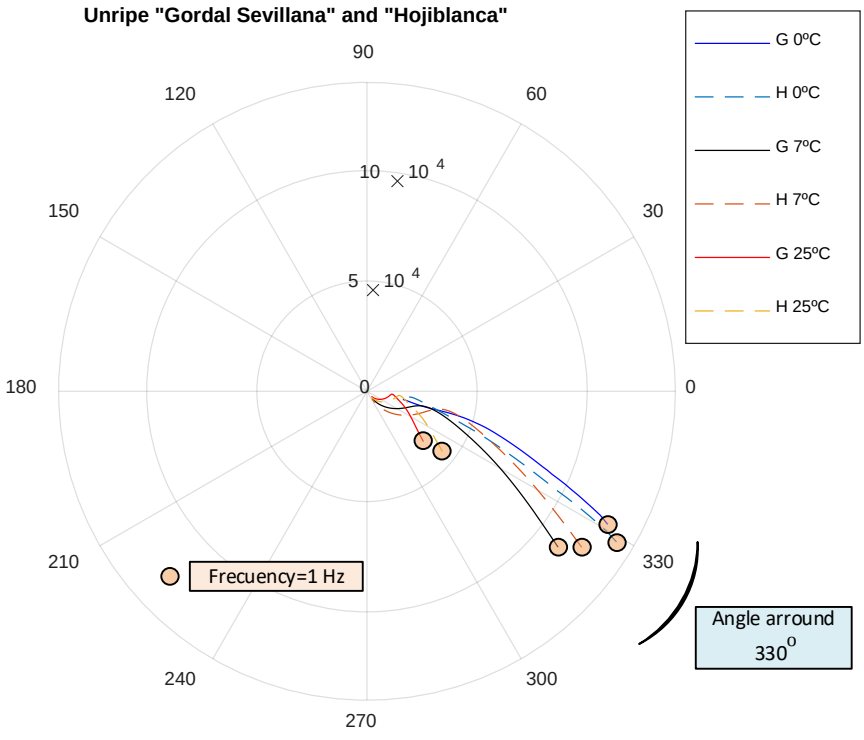
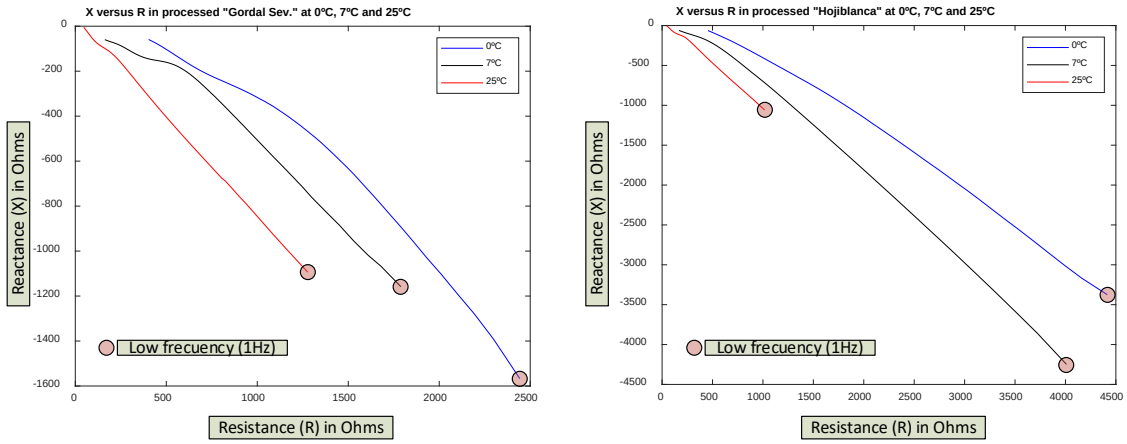


Figure 6. Frequency sweep to measure the electrical impedance of the "Gordal Sevillana" and "Hojiblanca" varieties when unripe (average values from 100 tests).

3.5. Test on Processed Olives: The effect the Olive Variety Has on Electrical Impedance in Olives Processed the "Estilo Sevillano" Way

Just as with unripe olives, it is sometimes difficult to distinguish between processed varieties because of their shape, size, and special color (due to the chemical processes they undergo) and they are very similar. Once again, a study was done on the two previously mentioned varieties ("Gordal Sevillana" and "Hojiblanca") seasoned the "Estilo Sevillano" way.

In Figure 7, the effect of temperature is shown on these types of olive with 100 samples of each variety at 3 temperatures (0 °C, 7 °C, and 25 °C). Once again, each variety has an electrical impedance spectrum characteristic at each temperature and the maximum differences between the two varieties are at lower frequencies.



(**a**) (**b**)

Figure 7. The effect of temperature on electric impedance on the (**a**) "Gordal Sevillana" and (**b**) "Hojiblanca" olive varieties processed in the "Estilo Sevillano" way in at 3 temperatures: 0 °C, 7 °C, and 25 °C (average values of 100 tests).

It must be highlighted that in the case of R-X curves, minimums relative to X do not present as happened in the case of unripe olives. The classic models described in [69,70] are not applied.

Another detail to emphasize is the drastic reduction of the values in both components (R, X) (around 20 times) due to the presence of brine, which makes the olive pulp highly conductive.

Finally, as can be observed, comparing Figures 6–8, the presence of brine makes the aggregate behave in a more capacitive way with an average offset angle around 310° for olives in brine and around 330° for unripe olives.

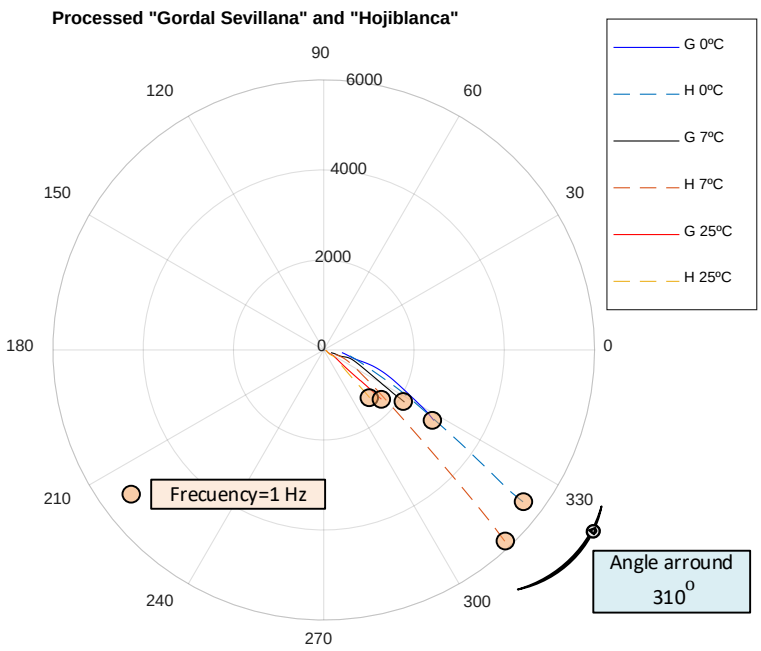


Figure 8. Influence of the brine on the impedance of processed olives.

3.6. Complex Impedance Model for "Gordal Sevillana" and "Hojiblanca" Olive Varieties Unripe and Processed the "Estilo Sevillano" Way Via Neural Networks

As mentioned in Section 2.3, the classic models [69,70] do not adapt well to characterize the evolution of impedance in olives; not in unripe olives or in those processed the "Estilo Sevillano" way, for each variety and each temperature. Thus, the Hayden model [69] does not take components R2 and C3 sufficiently into account, reducing their effect.

A model has been calibrated based on neural networks for both varieties, both in unripe and in processed olives at the three temperatures previously seen and directly correlating to the values of the reactive X and resistive R components. Figures 9–12 show some results using the Matlab Toolbox neural networks and its fitnet function [32], with a hidden layer hiding 5 neurons being enough to obtain the adequate adjustment.

The goodness of fit is expressed in agreement to [71] and as relative error in % in Tables 2 and 3.

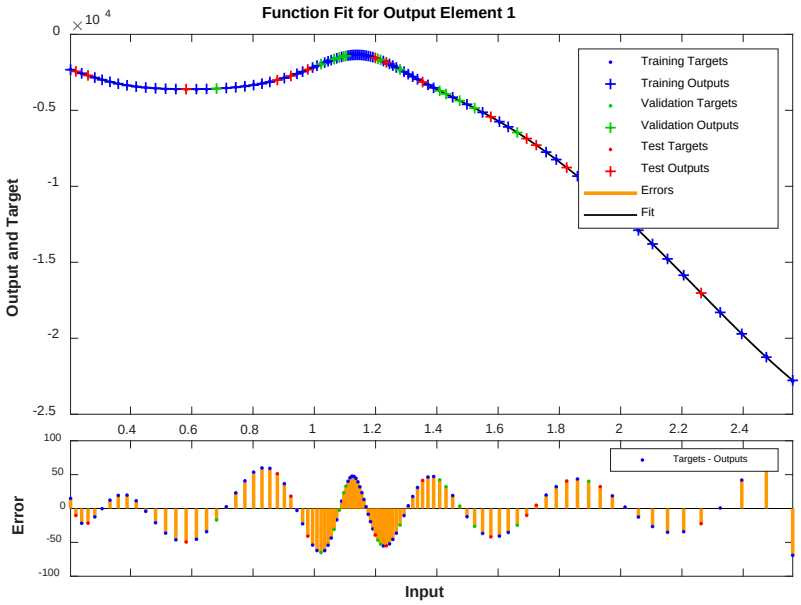


Figure 9. Unripe "Gordal Sevillana" at 25 °C.

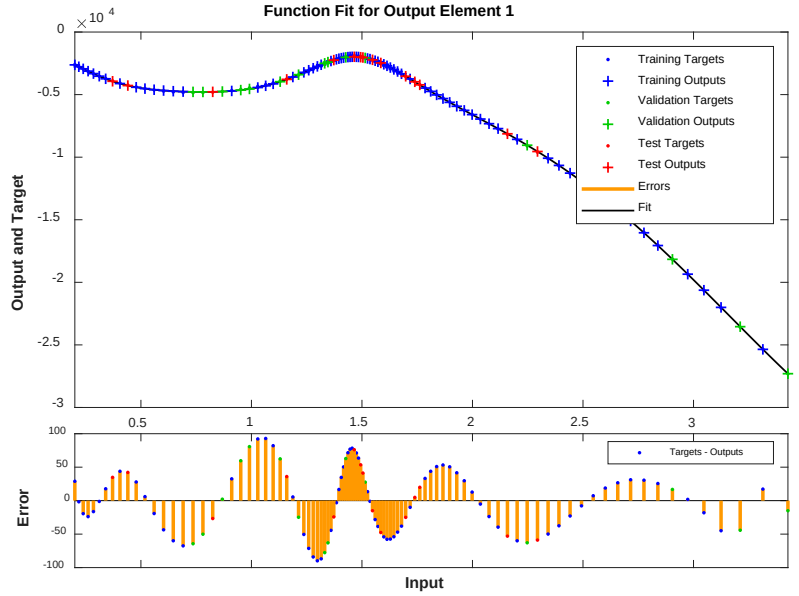


Figure 10. Unripe "Hojiblanca" at 25 °C.

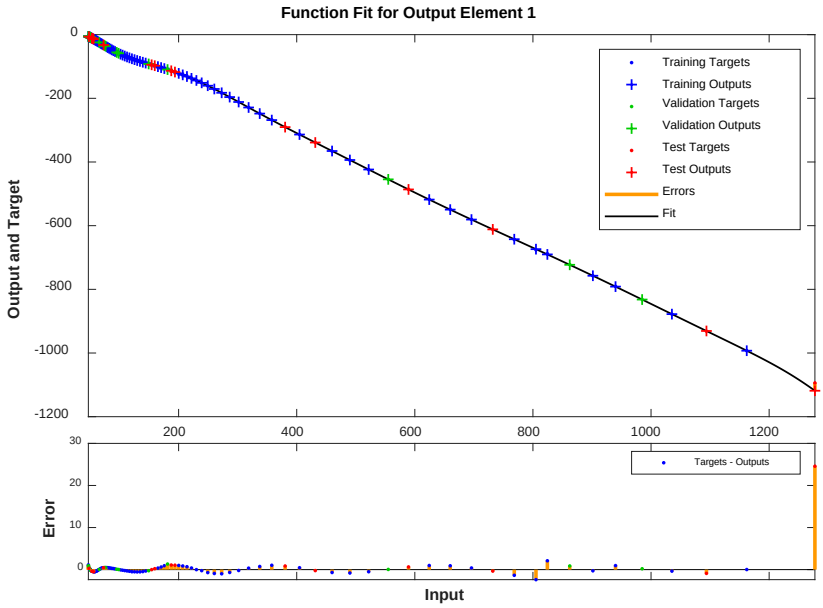


Figure 11. Processed "Gordal Sevillana" at 25 °C.

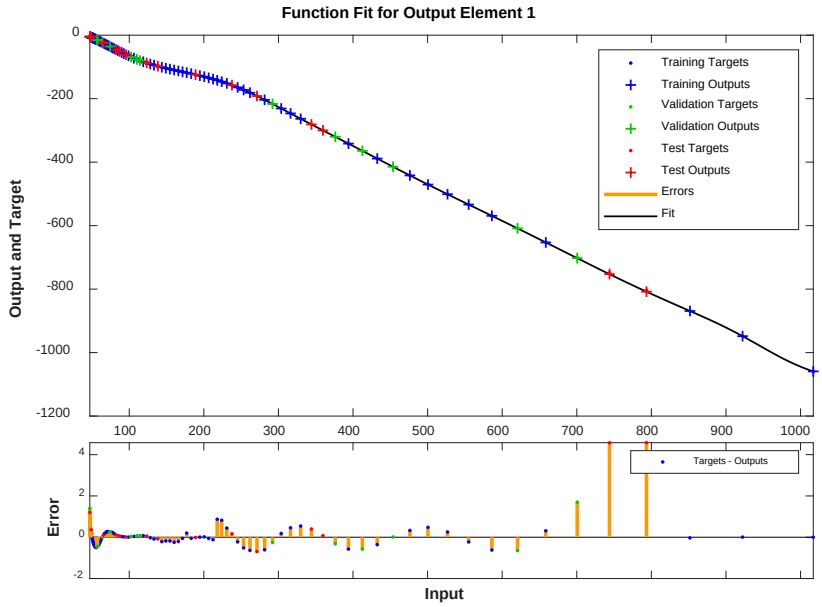


Figure 12. Processed "Hojiblanca" at 25 °C.

Table 2. Goodness of fit for unripe olives according to variety and temperature.

Variety	Temperature	P	Relative Error (%)
"Gordal Sev."	0 °C	7.3753×10^3	1.33%
	7 °C	788.9215	0.08%
	25 °C	1.2965×10^3	0.33%
"Hojiblanca"	0 °C	1.0014×10^3	0.18%
	5 °C	294.0392	0.05%
	20 °C	2.1827×10^3	0.36%

Table 3. Goodness of fit for processed olives according to variety and temperature.

Variety	Temperature	P	Relative Error (%)
"Gordal Sev."	0 °C	1.6073×10^3	1.14%
	7 °C	1.885×10^3	0.34%
	25 °C	6.3865	1.27%
"Hojiblanca"	0 °C	1.4252×10^3	0.25%
	7 °C	6.321×10^3	1.14%
	25 °C	2.218×10^3	0.4%

3.7. Sorting with Neural Networks According to Variety and Temperature

Distinguishing between varieties, either unripe or processed (in this case, "Estilo Sevillano"), can be of interest as mentioned in Section 2.3 to differentiate between varieties in case of doubt. In this section, the sorting is done using Toolbox by Matlab and, specifically, its neural sorter patternnet [33]. In all cases, adjustments were made by using just one layer of 8 neurons, 2 varieties, and 3 temperatures, in order for the sorter to distinguish between 6 possible options. Conducting the sorting process at 3 temperatures allows it to be generalized given that the temperature conditions on the farm and in the factory can be different throughout the work day or the time of year when the test is done. This is of special interest in the industry where olives are generally found at these three temperatures (room temperature during storage is about 25 °C on average, at 7 °C prior to pitting/stuffing/slicing and 0 °C during the pitting/stuffing/slicing).

3.7.1. Sorting with Neural Networks for Unripe Olives

Once the sorter is trained (10 samples of each state and 5 challenges, which is to say 60 samples all together for training and 30 to test the sorter), the results are shown in Figure 13. The neural network detected 100% of cases correctly.

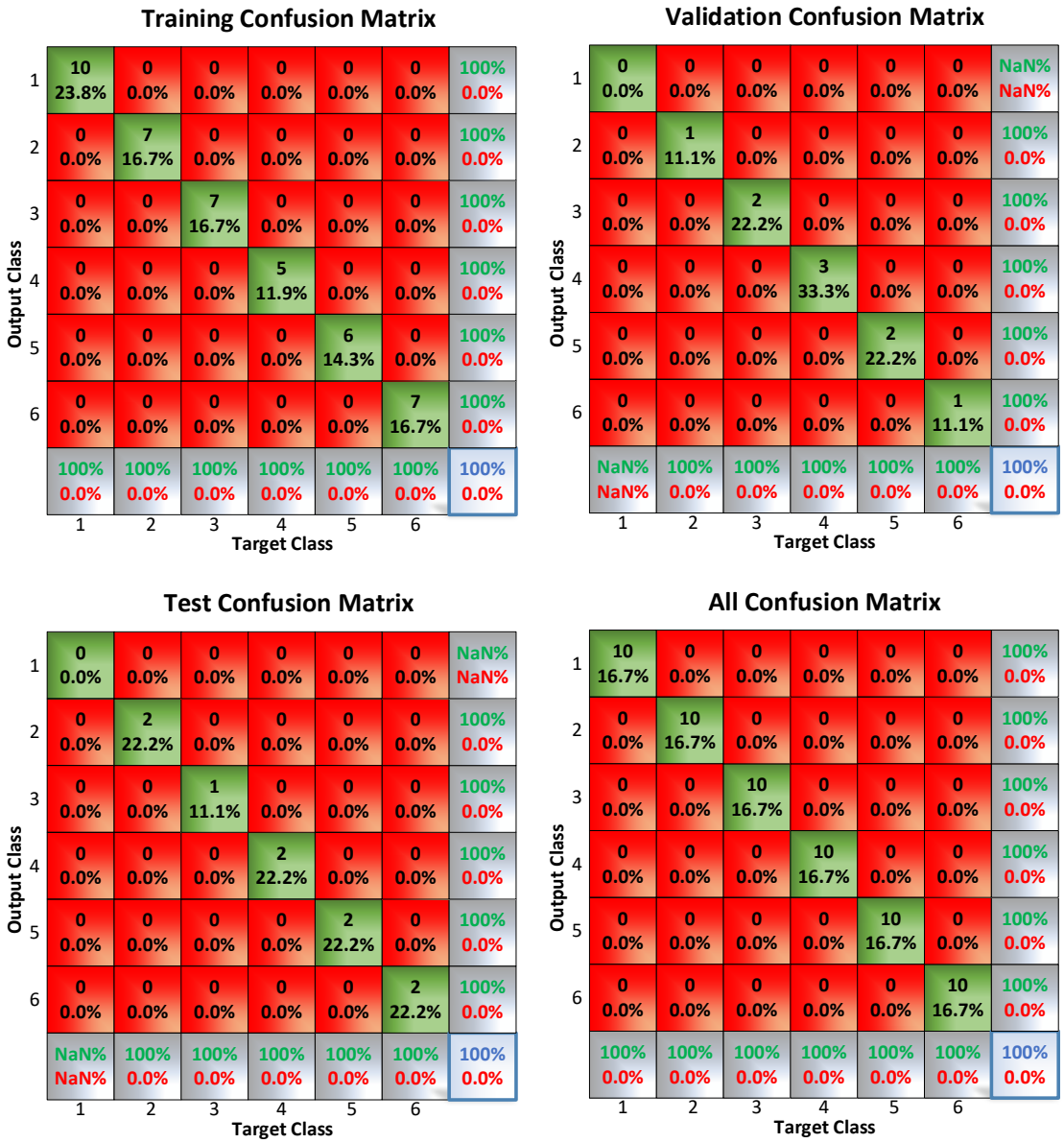


Figure 13. Training confusion matrix for the sorting of unripe olives.

3.7.2. Sorting with Neural Networks for "Estilo Sevillano" Processed Olives

Once the sorter is trained (10 samples of each state and 5 challenges, which is to say 60 samples all together for training and 30 to test the sorter), the results are shown in Figure 14. The neural network detected more than 98% of cases correctly.

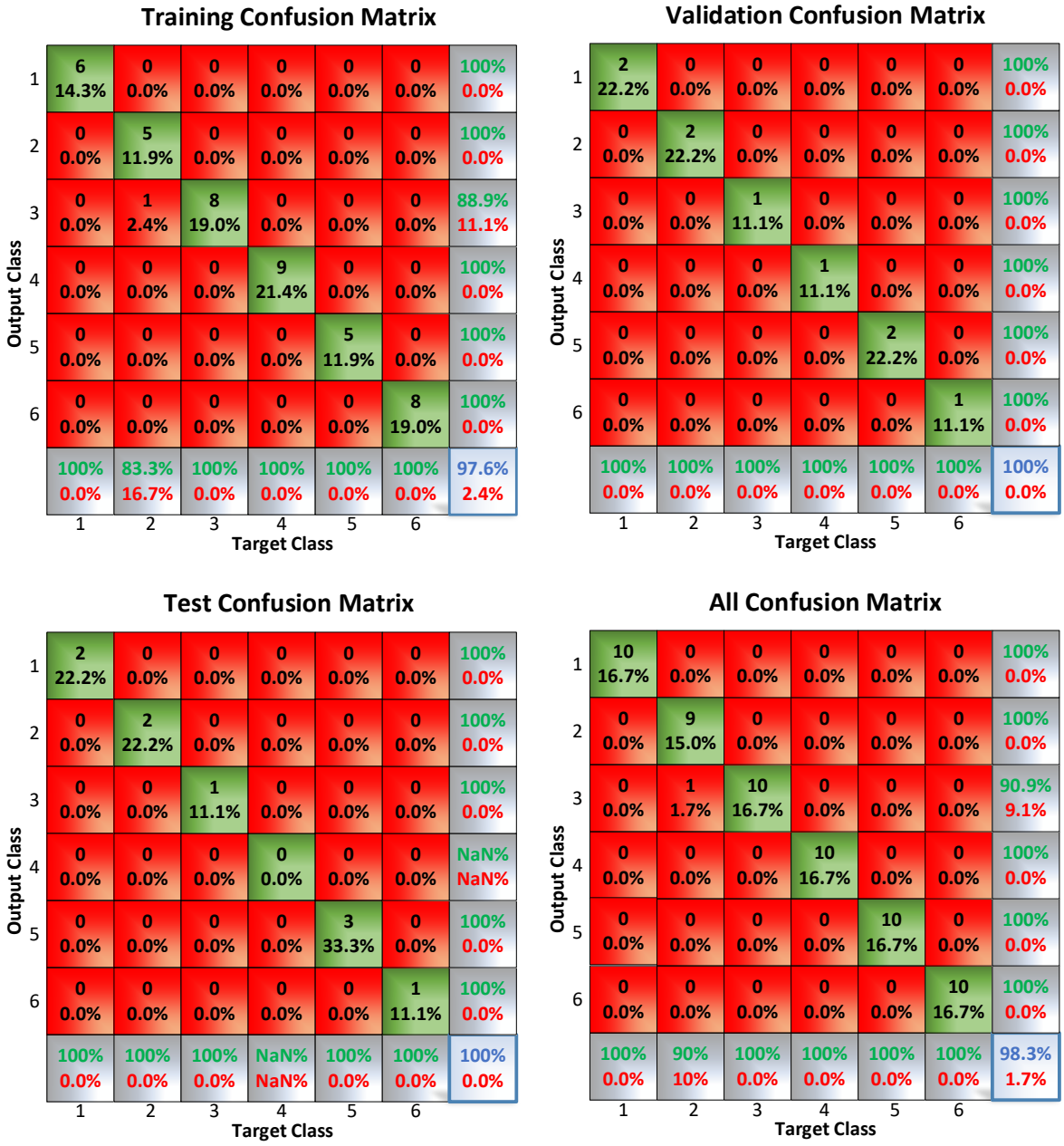


Figure 14. Training confusion matrix for sorting "Estilo Sevillano" processed olives.

3.8. The Internet of Things (IoT)

With the objective of evaluating the prototype for the farm and factory tests, a simple IoT system was chosen based on the use of a laptop which executes the Matlab application and sends the file with the results to a shared folder in a Dropbox. This way, the samples can be analyzed remotely and from any other computer, as shown in Figure 15.

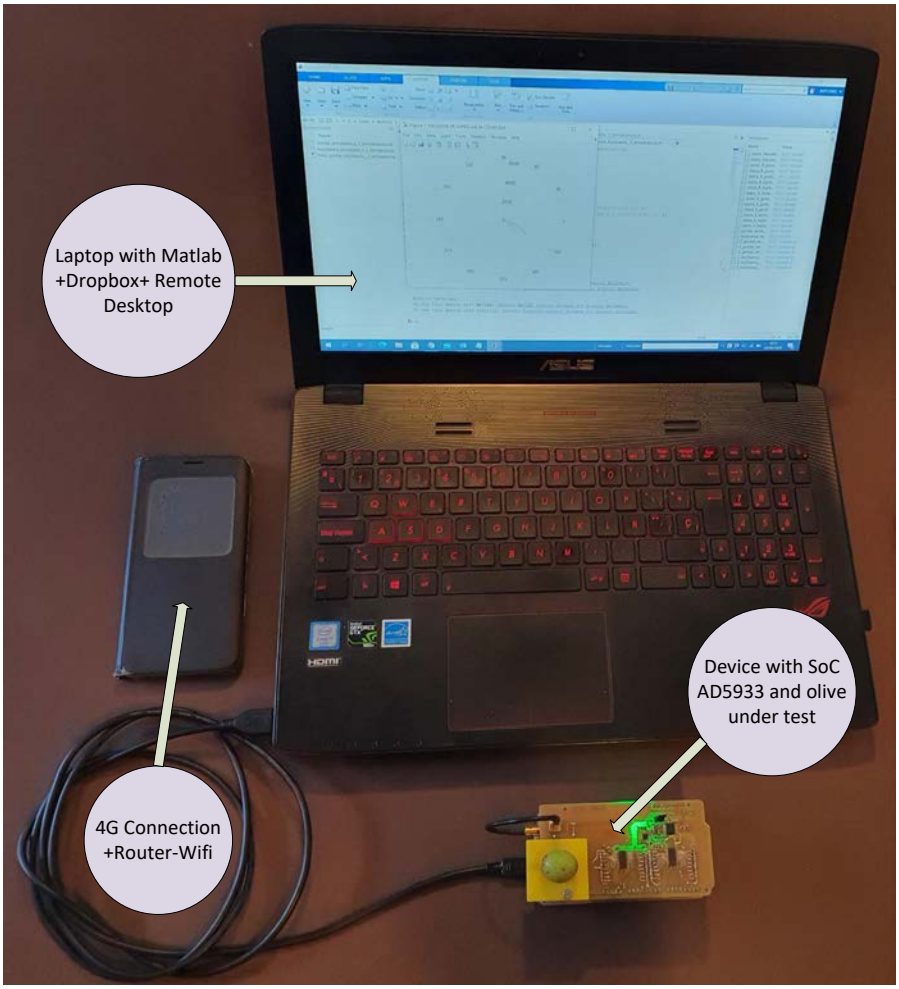


Figure 15. IoT System during test in factory.

4. Conclusions

With this work it has been possible to verify that:

- Both unripe and processed olives present an impedance profile equivalent to an R-C model without an inductive component with a phase around 330° in unripe olives and 310° in brined olives.
- With unripe and processed olives, a characteristic impedance profile can be observed for each variety at each temperature. It is at a low frequency where the differences are more accentuated.
- With unripe olives at a high frequency, the relative minimums observed are similar to those described by classic models like that of Hayden, which does not happen with olives processed in brine.
- With olives processed in brine, the impedance value of the components R and X are reduced by 20 times, due to the effect of the brine.
- The models developed with neural networks like fitnet to represent the evolution of the impedance allow a model of type R versus X be obtained with only 5 neurons in the hidden layer.

- It has been verified that neural networks like patternnet and 8 neurons in the hidden layer, allow it to distinguish between the 6 cases studied (2 varieties and 3 temperatures) both in unripe and processed olives.
- The use of a simple IoT system based in Dropbox allows samples to be obtained on the farm and in the factory for later study using the combination of a laptop with a 4G connection and the prototype developed.

On the other hand, to reach these results, the design of a hardware has been chosen that has allowed us to obtain the maximum benefits of the SoC AD5933:

- A circuit implementing an SoC AD5933 has been developed with all the peripheral elements necessary for it to run. This prototype includes a pair of ADG706 analog multiplexers in order to convert the range in the impedance module to be measured.
- In order to achieve the maximum resolution in the DFT, a DDS based on an FPGA has been used to generate a clock signal to be programmed at will according to the limits of the frequency sweep to be carried out during the impedance measurement.
- Programming the main application has been done in Matlab. In order to control all of the elements, an ARM CORTEX M3 (AT91SAM3X8E) microcontroller has been used with an Arduino DUE, implementing all of the firmware necessary to control the hardware. Lastly, the Picoblaze routine control embedded in the FPGA of the DDS has been implemented in ASM.
- The resistance tests and the RLC series/parallel circuits have shown that the system works properly.
- There is a functional limitation to the chip where the internal DSP speed is proportional to the clock speed applied externally. In these circumstances, for the low frequency measurement (from 1 Hz to 30 Hz), a 25 kHz clock has been used which, compared to the frequency used (16 MHz), makes the measurement process 640 times slower at low frequencies.
- The circuit built is experimental and, in order to use it directly on the farms where the crop is located, a version capable of withstanding those working conditions ought to be produced. Likewise, the application should be an app, for example on a cell phone or a tablet, where it could connect to the computer via Bluetooth.
- One possible option is to implement all the routines through a microcontroller embedded in the FPGA, using a programable microcontroller directly in C (for example, Microblaze) in this case.

Finally, among the future areas of work, we are considering:

- Increasing the frequency range to 25 MHz (likely on a system which allows it to go over 100 kHz) with the objective of seeing if, after that point, they are standard application models like in Hayden's.
- Studying other olive varieties of commercial relevance like "Manzanilla" or "Cacereña" olives.
- Studying other industrial treatments like the oxidized black olive (California style).
- Running an analysis which correlates the breakage percentage in DRR machines directly with the measured impedance value of different varieties, processes, and temperatures.

Future studies will include the application of this methodology in other fruits such as tomato or cherry.

Author Contributions: Conceptualization, A.M.L.; Data curation, J.M.M.L.; Investigation, A.M.L. and R.E.H.F.; Supervision, J.M.M.L., A.M.L. and R.E.H.F.; Validation, J.M.M.L., A.M.L. and R.E.H.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

72. Bourne, M.C. Effect of Temperature on Firmness of Raw Fruits and Vegetables. *J. Food Sci.* **1982**, *47*, 440–444, doi:10.1111/j.1365-2621.1982.tb10099.x.
73. Bourne, M.C.; Comstock, S.H. Effect of Temperature on Firmness of Thermally Processed Fruits and Vegetables. *J. Food Sci.* **1986**, *51*, 531–533, doi:10.1111/j.1365-2621.1986.tb11179.x.
74. Bourne, M.C. *Food Texture and Viscosity: Concept and Measurement*; Academic Press:, Harcourt place, 32 James Road, London NW1 7BY, UK 2002; ISBN 9780080491332.
75. Kılıçkan, A.; Güner, M. Physical properties and mechanical behavior of olive fruits (*Olea europaea* L.) under compression loading. *J. Food Eng.* **2008**, *87*, 222–228, doi:10.1016/j.jfoodeng.2007.11.028.
76. Gutierrez Rubio, J. *Maquina Enfriadora de Aceitunas*; ES 1 003 639 U, 1988. Available online: http://www.oepm.es/pdf/ES/0000/000/01/00/36/ES-1003639_U.pdf (accessed on 15 September 2020).
77. Leiva, D.; Tapia, F. Elaboración de Aceitunas Con Bajo Contenido de Sodio. In *Producción de Aceitunas con Bajo Contenido de Sodio ("Light")*; Instituto de Investigaciones Agropecuarias, Centro Regional de Investigación Intihuasi: La Serena, Chile, 2015. Available online: http://biblioteca.inia.cl/medios/biblioteca/boletines/NR40474.pdf. (accessed on 13 September 2020)
78. Gómez, A.H.S.; García, P.; Navarro, L.R. Elaboration of table olives. *Grasas Aceites* **2006**, *57*, 86–94.
79. Hlaváčová, Z. Low frequency electric properties utilization in agriculture and food treatment. *Res. Agric. Eng.* **2003**, *49*, 125–136, doi:10.17221/4963-RAE.
80. Mitchell, F.R.G.; Alwis, A.A.P.D. Electrical conductivity meter for food samples. *J. Phys. E* **1989**, *22*, 554–556, doi:10.1088/0022-3735/22/8/004.
81. Nelson, S.O. Dielectric properties of agricultural products-measurements and applications. *IEEE Trans. Electr. Insul.* **1991**, *26*, 845–869, doi:10.1109/14.99097.
82. Repo, T.; Paine, D.H.; Taylor, A.G. Electrical impedance spectroscopy in relation to seed viability and moisture content in snap bean (*Phaseolus vulgaris* L.). *Seed Sci. Res.* **2002**, *12*, 17–29, doi:10.1079/SSR200194.
83. *AD5933*; Analog Devices: Norwood, MA, USA, 2005. Available online: http://www.analog.com/media/en/technical-documentation/data-sheets/AD5933.pdf. (accessed on 15 September 2020).
84. Rodríguez Gómez, R.; Cruz Hurtado, J. Sistema de medición y análisis de impedancia. *Ing. Electrónica Automática Comun.* **2015**, *36*, 56–66.
85. Okada, K.; Sekino, T. *Agilent Impedance Measurement Handbook;* A guide to measurement technology and techniques; Agilent Technologies: Santa Clara, CA, USA, 2009.
86. Tegam. The LCR Meter as an Impedance Analyzer. Available online: http://www.tegam.com/wp-content/uploads/2015/10/AN303.pdf. (accessed on 15 September 2020).
87. Keysight 4395A Network/Spectrum/Impedance Analyzer. Available online: http://www.keysight.com/en/pd-1000000864%3Aepsg%3Apro-pn-4395A/network-%0Aspectrum-impedance-analyzer?cc=ES&lc=eng (accessed on 15 September 2020).
88. Keysight 4194A Impedance/Gain-Phase Analyzer. Available online: https://www.keysight.com/en/pd-1000003398%3Aepsg%3Apro-pn-4194A/impedance-gain-phase-analyzer?cc=ES&lc=eng. (accessed on 05 September 2020).
89. Juping, G.; Long, J.; Shenbei, Q.; Xinjian, W.; Zhike, X. Researching on the automatic impedance measurement system. In Proceedings of the Eighth International Conference on Electrical Machines and Systems, Nanjing, China, 27–29 September 2005.
90. Ingenieria Eléctrica Fravedsa. Available online: http://ingenieriaelectricafravedsa.blogspot.com.es/2014/11/puente-schering.html. (accessed on 15 September 2020).
91. Wikipedia. Maxwell Bridge. Available online: https://en.wikipedia.org/wiki/Maxwell_bridge. (accessed on 18 September 2020).
92. Ibrahim, K.M.; Abdul-Karim, M.A.H. Digital Impedance Measurement by Generating Two Waves. *IEEE Trans. Instrum. Meas.* **1985**, *IM-34*, 2–5, doi:10.1109/TIM.1985.4315245.

93. Taha, S.M.R. Digital measurement of the polar and rectangular forms of impedances. *IEEE Trans. Instrum. Meas.* **1989**, *38*, 59–63, doi:10.1109/19.19999.
94. Steber, G.R. A Low Cost RF Impedance Analyzer. *Nuts and Volts*, February 2008, pp. 38–41. Available online: https://www.nutsvolts.com/magazine/article/a_low_cost_rf_impedance_analyzer (accessed on 15 September 2020).
95. Castelló, J.; Espí, J.; García, R.; Esteve, V. Analizador de Impedancia/Ganancia-Fase para PC. *Revista Española de Electrónica*, 2001, pp. 70–75.
96. Justicia, M.; Madueño, A.; Ruiz-Canales, A.; Molina, J.M.; López, M.; Madueño, J.M.; Granados, J.A. Low-frequency characterisation of mesocarp electrical conductivity in different varieties of olives (*Olea europaea* L.). *Comput. Electron. Agric.* **2017**, *142*, 338–347, doi:10.1016/j.compag.2017.09.021.
97. Weaver, G.M.; Jackson, H.O. Electric impedance, an objective index of maturity in peach. *Can. J. Plant Sci.* **1966**, *46*, 323–326.
98. Ezeike, G.O.I. A resistive probe moisture sensor for tropical root crops and vegetables. *J. Agric. Eng. Res.* **1987**, *37*, 15–26, doi:10.1016/0021-8634(87)90128-4.
99. Montoya Lirola, M. Estudio de la Conductividad Eléctrica Como Índice de Madurez en Frutos Climatéricos y su Evolución Durante la Conservación Frigorífica en Atmosfera Normal y Modificada. Ph.D. Thesis, UNED, Madrid, Spain, 1992. Available online: https://dialnet.unirioja.es/servlet/tesis?codigo=40951 (accessed on 15 September 2020).
100. Van Gerven, M.; Bohte, S. Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Front. Comput. Neurosci.* **2017**, *11*, doi:10.3389/fncom.2017.00114.
101. De Jódar Lázaro, M.; Madueño Luna, A.; Lucas Pascual, A.; Martínez, J.M.M.; Canales, A.R.; Madueño Luna, J.M.; Segovia, M.J.; Sánchez, M.B. Deep learning in olive pitting machines by computer vision. *Comput. Electron. Agric.* **2020**, *171*, 105304, doi:10.1016/j.compag.2020.105304.
102. Lucas Pascual, A.; Madueño Luna, A.; de Jódar Lázaro, M.; Molina Martínez, J.M.; Ruiz Canales, A.; Madueño Luna, J.M.; Justicia Segovia, M. Analysis of the Functionality of the Feed Chain in Olive Pitting, Slicing and Stuffing Machines by IoT, Computer Vision and Neural Network Diagnosis. *Sensors* **2020**, *20*, 1541, doi:10.3390/s20051541.
103. The MathWorks Inc. Function Fitting Neural Network (Fitnet). Available online: https://es.mathworks.com/help/deeplearning/ref/fitnet.html. (accessed on 15 September 2020).
104. The MathWorks Inc. Pattern Recognition Network (Patternnet). Available online: https://es.mathworks.com/help/deeplearning/ref/patternnet.html;jsessionid=370562d44f3c46b93a717f92677f. (accessed on 18 September 2020).
105. Lougheed, E.C.; Miller, S.R.; Ripley, B.D.; Cline, R.A. Electrical impedance of daminozide- and calcium-treated McIntosh apples. *Experientia* **1981**, *37*, 835–837, doi:10.1007/BF01985666.
106. Jackson, P.J.; Harker, F.R. Apple Bruise Detection by Electrical Impedance Measurement. *HortScience* **2000**, *35*, 104–107, doi:10.21273/HORTSCI.35.1.104.
107. Stout, D.G.; Hall, J.W.; McLaughlin, N.B. In vivo plant impedance measurements and characterization of membrane electrical properties: The influence of cold acclimation. *Cryobiology* **1987**, *24*, 148–162, doi:10.1016/0011-2240(87)90017-4.
108. Stout, D.G. Effect of Cold Acclimation on Bulk Tissue Electrical Impedance. *Plant Physiol.* **1988**, *86*, 283–287, doi:10.1104/pp.86.1.283.
109. Bauchot, A.D.; Harker, F.R.; Arnold, W.M. The use of electrical impedance spectroscopy to assess the physiological condition of kiwifruit. *Postharvest Biol. Technol.* **2000**, *18*, 9–18, doi:10.1016/S0925-5214(99)00056-3.
110. NXP Semiconductors. I²C Bus. Available online: http://www.interfacebus.com/Design_Connector_I2C.html (accessed on 14 September 2020).
111. Xilinx Inc. Spartan-3E Fpga Available online: https://www.digikey.es/es/datasheets/xilinxinc/xilinx-inc-ds312. (accessed on 15 September 2015).
112. *ADG706 Analog Multiplexer*; Analog Devices, Norwood, MA, USA, 2005. Available online: https://www.analog.com/media/en/technical-documentation/data-sheets/ADG706_707.pdf. (accessed on

15 September 2020).
113. Microchip ARM Cortex-M3. Available online: https://www.microchip.com/wwwproducts/en/ATSAM3X8E (accessed on 16 September 2020).
114. The MathWorks Inc. Matlab Software. Available online: https://es.mathworks.com/products/matlab.html (accessed on 11 September 2020).
115. Dropbox. Available online: https://www.dropbox.com/. (accessed on 19 September 2020).
116. Rappoport, H. Botánica y Morfología. In *El Cultivo del Olivo*; Mundi-Prensa: Madrid, Spain, 2008; pp. 35–60.
117. Ferreira, J. *Explotaciones Olivareras Colaboradoras*; Number 5; Ministerio de Agricultura: Madrid, Spain, 1979.
118. Garrido, A.; García, P.; Brrenes, M. Olive fermentations. In *Biotechnology: A Multivolume Comprehesive Treatise*; Reed, H.J., Nagodawitana, T.W., Eds.; Wiley-VCH Verlag GmbH, 1995; pp. 593–625.
119. Estrada, J.M. *La Aceituna de Mesa: Nociones Sobre sus Características,Elaboración y Cualidades*; Fundacion Para El Fomento y Promocion de la Aceituna de Mesa: Sevilla, Spain; Diputación de Sevilla: Sevilla, Spain, 2011. Available online: http://www.besana.es/sites/default/files/libroaceitunamaqueta080411.pdf. (accessed on 15 September 2020).
120. Garrido, A.; García, P.; López, A.; Arroyo, F.N. *Processing Technology in Olive Oil and Table Olive*. International Olive Council, Madrid, 2006. Available online: https://pdfs.semanticscholar.org/5999/f039244e30eda8538c20a2f2b47092c4f55e.pdf. (accessed on 17 September 2020).
121. Stockham, T.G. High-speed convolution and correlation. In *Proceedings of the Spring Joint Computer Conference on XX—AFIPS '66 (Spring),* Boston, MA. USA, *April 26–28 1966*; ACM Press: New York, NY, USA, 1966; p. 229.
122. Chen, C.J.; Liu, J.T.; Chang, S.J.; Lee, M.W.; Tsai, J.Z. Development of a portable impedance detection system for monitoring the growth of mouse L929 cells. *J. Taiwan Inst. Chem. Eng.* **2012**, *43*, 678–684, doi:10.1016/j.jtice.2012.04.008.
123. Schwarzenberger, T.; Wolf, P.; Brischwein, M.; Kleinhans, R.; Demmel, F.; Lechner, A.; Becker, B.; Wolf, B. Impedance sensor technology for cell-based assays in the framework of a high-content screening system. *Physiol. Meas.* **2011**, *32*, 977–993, doi:10.1088/0967-3334/32/7/S18.
124. Wang, M.H.; Kao, M.F.; Jang, L.S. Single HeLa and MCF-7 cell measurement using minimized impedance spectroscopy and microfluidic device. *Rev. Sci. Instrum.* **2011**, *82*, 064302, doi:10.1063/1.3594550.
125. Helen Berney, H.; O'Riordan, J.J. Impedance Measurement Monitors Blood Coagulation. *Analog Dialogue* **2008**, *42*, 2–4.
126. Broeders, J.; Duchateau, S.; van Grinsven, B.; Vanaken, W.; Peeters, M.; Cleij, T.; Thoelen, R.; Wagner, P.; de Ceuninck, W. Miniaturised eight-channel impedance spectroscopy unit as sensor platform for biosensor applications. *Phys. Status Solidi A* **2011**, *208*, 1357–1363, doi:10.1002/pssa.201001199.
127. Seoane, F.; Ferreira, J.; Sanchéz, J.J.; Bragós, R. An analog front-end enables electrical impedance spectroscopy system on-chip for biomedical applications. *Physiol. Meas.* **2008**, *29*, S267–S278, doi:10.1088/0967-3334/29/6/S23.
128. Bogónez-Franco, P.; Bayés-Genís, A.; Rosell, J.; Bragós, R. Performance of an implantable impedance spectroscopy monitor using ZigBee. *J. Phys. Conf. Ser.* **2010**, *224*, 012163, doi:10.1088/1742-6596/224/1/012163.
129. Ferreira, J.; Seoane, F.; Lindecrantz, K. AD5933-based electrical bioimpedance spectrometer. Towards textile-enabled applications. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Boston, MA, USA, 30 August–3 September 2011*; IEEE: Piscataway, NY, USA, 2011; pp. 3282–3285.
130. Margo, C.; Katrib, J.; Nadi, M.; Rouane, A. A four-electrode low frequency impedance spectroscopy measurement system using the AD5933 measurement chip. *Physiol. Meas.* **2013**, *34*, 391–405, doi:10.1088/0967-3334/34/4/391.
131. Melwin, A.; Rajasekaran, K. Implementation of Bioimpedance Instrument Kit in ARM7. *Int. J. Adv. Res.*

Comput. Sci. Softw. Eng. **2013**, *3,5*, 1271–1273.
132. Hoja, J.; Lentka, G. Interface circuit for impedance sensors using two specialized single-chip microsystems. *Sens. Actuators A Phys.* **2010**, *163*, 191–197, doi:10.1016/j.sna.2010.08.002.
133. Hoja, J.; Lentka, G. A Family of New Generation Miniaturized Impedance Analyzers for Technical Object Diagnostics. *Metrol. Meas. Syst.* **2013**, *20*, 43–52, doi:10.2478/mms-2013-0004.
134. Grimnes, S.; Martinsen, O.G. *Bioimpedance and Bioelectricity Basics*, 3rd ed.; Elsevier Academic Press: Amsterdam, 1000AE, Netherlands 2014; ISBN 9780124115330.
135. Chabowski, K.; Piasecki, T.; Dzierka, A.; Nitsch, K. Simple Wide Frequency Range Impedance Meter Based on AD5933 Integrated Circuit. *Metrol. Meas. Syst.* **2015**, *22*, 13–24, doi:10.1515/mms-2015-0006.
136. Simic, M. Realization of Complex Impedance Measurement System Based on the Integrated Circuit AD5933. In *Proceedings of the 21st Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 26–28 November 2012*; IEEE: Piscataway, NY, USA, 2013; pp. 573–576.
137. Simić, M. Complex Impedance Measurement System for the Frequency Range from 5 kHz to 100 kHz. *Key Eng. Mater.* **2015**, *644*, 133–136, doi:10.4028/www.scientific.net/KEM.644.133.
138. Simic, M. Realization of digital LCR meter. In *Proceedings of the International Conference and Exposition on Electrical and Power Engineering (EPE), Iasi, Romania, 16–18 October 2014*; IEEE: Piscataway, NY, USA, 2014; pp. 769–773.
139. Madueño, J.M. Papers Appendix. Available online: https://www.dropbox.com/sh/wg1hmyxdgt558nc/AACPWe1XJi_tYcJoFzVihv3Ya?dl=0 (accessed on 15 September 2020).
140. Hayden, R.I.; Moyse, C.A.; Calder, F.W.; Crawford, D.P.; Fensom, D.S. Electrical Impedance Studies on Potato and Alfalfa Tissue. *J. Exp. Bot.* **1969**, *20*, 177–200, doi:10.1093/jxb/20.2.177.
141. Wu, L.; Ogawa, Y.; Tagawa, A. Electrical impedance spectroscopy analysis of eggplant pulp and effects of drying and freezing-thawing treatments on its impedance characteristics. *J. Food Eng.* **2008**, *87*, 274–280, doi:10.1016/j.jfoodeng.2007.12.003.
142. The MathWorks Inc. Network Performance. Available online: https://es.mathworks.com/help/deeplearning/ref/perform.html (accessed on 14 September 2020).