UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

TESIS

OSCILOSCOPIO DIGITAL PORTÁTIL CON COMUNICACIÓN BLUETOOTH, BASADO EN PROCESADOR DE SEÑALES Y DISPOSITIVO ANDROID

Que para obtener el Título de

INGENIERO EN ELECTRÓNICA

Presenta

Baldwin Rainiero Cortés Hernández

balcortex@hotmail.com

Asesor de Tesis

Maestro en Ciencias en Ingeniería Electrónica

Félix Jiménez Pérez

felichefie@gmail.com

Morelia, Michoacán, Enero de 2015

Agradecimientos

- *A mis padres Ma. Angélica Hernández Hernández y J. Jesús Cortés García, que me han apoyado durante toda mi existencia y han dado siempre lo mejor de ellos, acompañándome en cada momento de mi vida escolar.*
- *A mi hermana Carolina Stephanie Cortés Hernández, que ha sido mi cómplice y compañera de vida.*
- *Al Ing. Félix Jiménez Pérez, por su incondicional apoyo y los conocimientos que me ha aportado.*
- *A mis amigos Gilberto Francisco Orozco Hernández y Juan Pablo Márquez Vargas, por todas las horas de que han pasado junto a mí, y la infinita cantidad de veces que me han hecho sonreír.*
- *A todos mis maestros, por todo su conocimiento compartido.*
- *A mi novia Marisela Quevedo Ochoa, el motor de mi vida.*

Dedicatoria

- *A mis padres, que a pesar de mis equivocaciones nunca me han dejado de apoyar.*
- *A la memoria de mis abuelos Irma Hernández y Roberto Cortés.*

Resumen

En este trabajo se presenta el diseño y construcción de un osciloscopio digital, el cual permite observar y analizar señales de voltaje de alta y baja velocidad, así como medir sus características, obtener y visualizar su espectro en frecuencia.

El hardware se compone básicamente de cinco elementos:

- Controlador digital de señales dsPIC33FJ16GS502, el cual se utiliza para realizar el muestreo de la señal.
- Amplificadores operacionales *rail-to-rail* de baja potencia para acondicionar la señal de entrada.
- Interruptores analógicos que permiten seleccionar la ganancia de la etapa de acondicionamiento y el tipo de acoplo de entrada en AC y DC.
- Módulo de comunicación Bluetooth, para transmitir datos y comandos.
- Dispositivo Android® que proporciona la interfaz con el usuario, en la cual se visualiza la información de la señal y se configura el dispositivo.

Operación básica: Las muestras se adquieren en tiempo real a través del convertidor analógico-digital integrado en el dsPIC y se almacenan en SRAM. Mediante el módulo de comunicación UART (también integrado en el dsPIC), se envían hacia el módulo Bluetooth, y posteriormente al dispositivo Android donde se realiza el procesamiento y visualización de la información. Éste permite tener una interfaz gráfica intuitiva para el control del instrumento de medición, el software escala de manera automática la gráfica al tamaño de la pantalla.

El equipo desarrollado presenta las siguientes características:

- Canales de entrada: 2.
- Resolución vertical: 8 bits.
- Velocidad de muestreo: 2 Msps.
- Voltaje máximo de entrada: ±10 V.
- Voltaje de alimentación: 3.3 V.

Palabras clave: Osciloscopio, dsPIC, Bluetooth, Android, FFT.

Abstract

In this paper, the design and construction of a digital oscilloscope is presented, which allows us to observe and analyze low-speed and high-speed voltage signals, measure their characteristics, obtain and display their frequency spectrum.

The hardware consists of basically five elements:

- A dsPIC33FJ16GS502 digital signal controller, which is used to sample the signal.
- Low power rail-to-rail operational amplifiers to condition the input signal.
- Analog switches that allow gain selection of the signal conditioning phase and allow switching the input coupling between AC and DC.
- Bluetooth communication module to transmit data and commands.
- Android device that provides the user interface, in which the signal information is displayed and the device is configured.

Basic operation: Samples are acquired in real time by the built-in analog-digital converter of the dsPIC and stored in SRAM. Using the UART communication module (also integrated into the dsPIC), these samples are sent to the Bluetooth module, which transfers them to the Android device where the signal processing and information display is done. By using an Android device, it is possible to control the measuring instrument with an intuitive graphical interface, the software automatically scale the graph to screen size.

The digital oscilloscope has the following characteristics:

- Input channels: 2.
- Vertical resolution: 8 bits.
- Sampling rate: 2 Msps.
- Maximum input voltage: ±10 V.
- Supply Voltage: 3.3 V.

Keywords: Oscilloscope, dsPIC, Bluetooth, Android, FFT.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Ecuaciones

Lista de Abreviaturas

AC	Corriente alterna
ADC	Convertidor analógico-digital
ANSI	Instituto Nacional Estadounidense de Estándares
API	Interfaz de programación de aplicaciones
APLL	Lazo auxiliar de seguimiento de fase
BLE	Bluetooth de baja energía
CMRR	Rechazo al modo común
CPU	Unidad central de procesamiento
CRT	Tubo de rayos catódicos
DC	Corriente directa
DSO	Osciloscopio de almacenamiento digital
DSP	Procesador digital de señales
dsPIC	Controlador digital de señales de interfaz periférico
F_{CY}	Frecuencia de instrucción
FFT	Transformada rápida de Fourier
F_{OSC}	Frecuencia de reloj
LCD	Pantalla de cristal líquido
MCRL	Borrado general
MIPS	Millones de instrucciones por segundo
Msps	Millones de muestras por segundo
OLED	Diodo orgánico de emisor de luz
Oscilador FRC	Oscilador rápido resistivo-capacitivo
PC	Computadora personal
PIC	Controlador de interfaz periférico

PLL	Lazo de seguimiento de fase
RISC	Computador con conjunto de instrucciones reducidas
RX	Línea de recepción en UART
SPI	Interfaz de periféricos serie
SRAM	Memoria estática de acceso aleatorio
TX	Línea de transmisión en UART
UART	Transmisor-receptor asíncrono universal
USB	Bus universal en serie
VCO	Oscilador controlador por tensión
V_{DD}	Fuente positiva de voltaje
V_{SS}	Nodo común de la fuente de voltaje

Glosario

Rail-to-rail	La característica rail-to-rail en un amplificador operacional permite que el voltaje de salida pueda estar muy cercano al voltaje de las fuentes de alimentación; mientras que en los amplificadores operacionales normales, el voltaje de salida es entre 1 V – 1.5 V menos que el voltaje de las fuentes.
Bytecode	Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código máquina.

Capítulo 1

Introducción

Un osciloscopio, anteriormente llamado oscilógrafo, es un tipo de instrumento de prueba electrónico que permite la observación de señales de voltaje, usualmente como una gráfica bidimensional de una o varias señales en función del tiempo. La imagen obtenida se denomina oscilograma.

1.1 Antecedentes

1.1.1 Oscilogramas dibujados a mano

El primer método de la historia para crear una imagen de una forma de onda era a través de un minucioso y laborioso proceso de medición de la tensión o corriente de un rotor giratorio en puntos específicos alrededor del eje del rotor, tomando los datos de un galvanómetro. Al desplazarse lentamente alrededor del rotor, una onda estacionaria podía ser dibujada en papel siguiendo la lectura de los grados de rotación y la amplitud denotada por el galvanómetro.

Este proceso fue parcialmente automatizado por Jules François Joubert con su método "paso a paso" para la medición de ondas mostrado en la Figura 1.1.

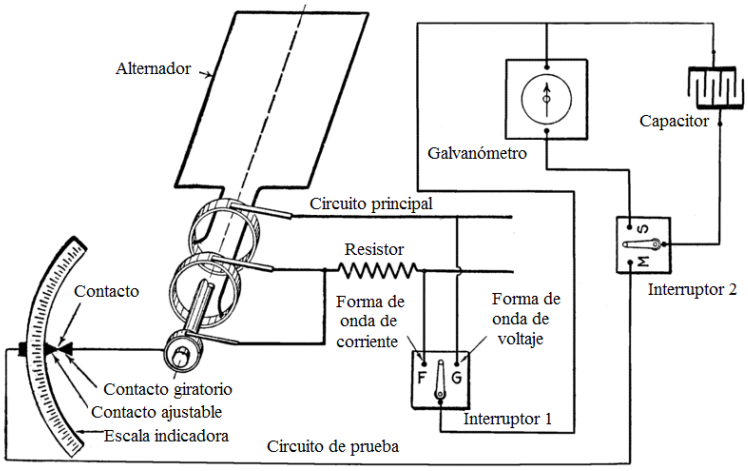

Figura 1.1 Ilustración del método de Joubert para dibujar manualmente una onda

Éste consistía en un conmutador especial de un solo contacto unido al eje de un rotor giratorio. El punto de contacto podía moverse alrededor del rotor siguiendo una escala indicadora de grados y la salida en el galvanómetro, esto para ser dibujado a mano por el operador [Hawkins 1917]. Este proceso sólo podía producir una tosca aproximación a la onda debido a que ésta se formaba durante un periodo de varios miles de ciclos de onda, sin embargo; fue el primer paso en la ciencia de la proyección de imágenes de forma de onda.

1.1.2 Oscilógrafo en papel tirado automáticamente

El primer oscilógrafo automatizado utilizado por primera vez empleaba un galvanómetro para mover una pluma a través de un rollo de papel, capturando formas de onda con el continuo movimiento del rollo. Debido a la relativa gran velocidad de las ondas de alta frecuencia en comparación con la lenta reacción de los componentes mecánicos, la imagen de la onda no era dibujada directamente, sino era construida en un periodo de tiempo combinando pequeños fragmentos de varios ciclos, creando una forma de onda promedio.

El dispositivo conocido como *ondógrafo hospitalario* estaba basado en este método de visualizar las ondas. Cargaba automáticamente un capacitor a partir de la centésima onda, y descargaba la energía a través de un galvanómetro de grabación, cada carga sucesiva del capacitor se hacía un poco más delante de la onda [Hawkins 1917]. La Figura 1.2 muestra una imagen de este dispositivo.

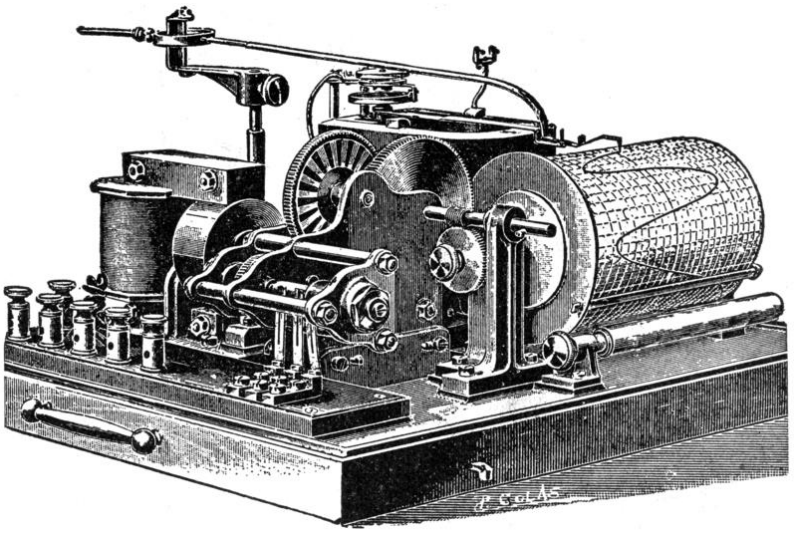

Figura 1.2 Ondógrafo hospitalario

1.1.3 Oscilógrafo fotográfico

A fin de lograr que la forma de onda fuera tomada directamente en un solo ciclo era necesario que el dispositivo de grabación empleara un sistema de medición muy ligero que se pudiera mover con la suficiente velocidad para alcanzar a las ondas medidas. Esto fue posible con el *oscilógrafo de bobina móvil* desarrollado por William Dudell, conocido en tiempos modernos como *galvanómetro de espejo,* el cual se muestra en la Figura 1.3. El tamaño del sistema de medición fue reducido a un pequeño espejo que se podía mover velozmente para emparejarse con velocidad de la onda y que reflejaba un haz de luz.

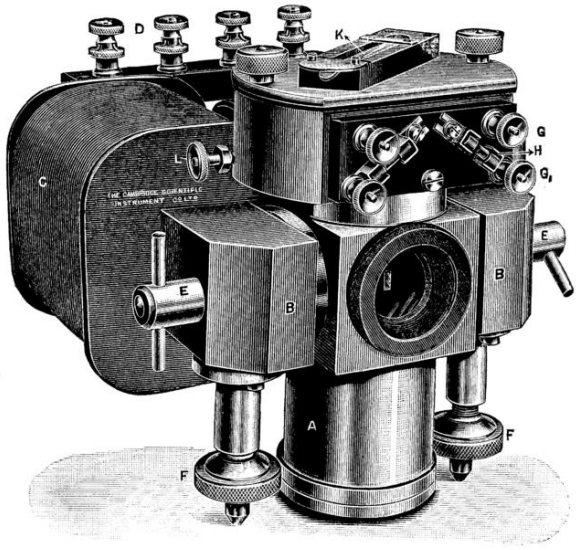

Figura 1.3 Oscilógrafo de bobina móvil

Para realizar la medición de la forma de onda, una diapositiva fotográfica es deslizada a través de una ventana por donde el haz de luz emerge para tomar una sola imagen de la onda, o un rollo de película cinematográfica es desplazado a través de la abertura para registrar la forma de onda a través del tiempo. A pesar de las mediciones eran mucho más precisas que con los oscilogramas en papel, todavía había margen para mejorar debido a que las fotografías tenían que ser reveladas antes de que se pudieran examinar. La Figura 1.4 muestra la fotografía de un oscilograma registrado con un oscilógrafo fotográfico.

En la década de 1920, un pequeño espejo reclinable sujeto a un diafragma en el ápice de una bocina proporcionó una buena respuesta de hasta 10 kHz. Una base de tiempo, no sincronizada, era proporcionada por un pequeño espejo giratorio en forma de polígono, y un

haz de luz colimada proveniente de una lámpara de arco proyectaba la forma de onda en la pared del laboratorio.

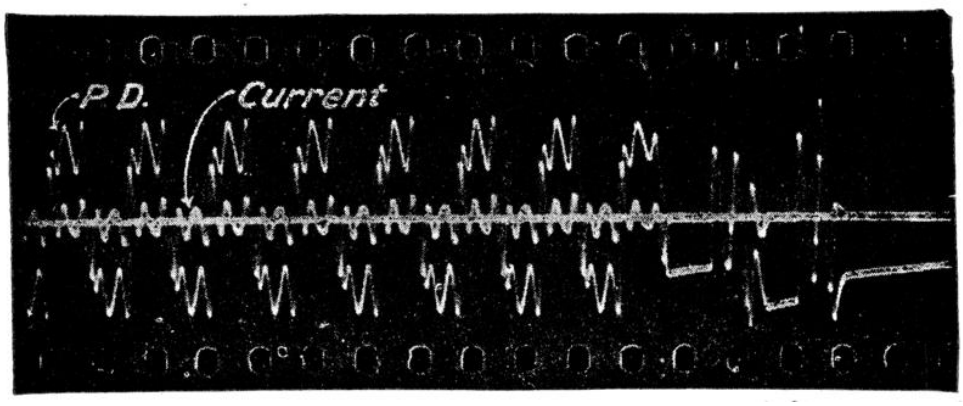

Figura 1.4 Oscilograma registrado en película

1.1.4 Osciloscopio de rayos catódicos

El tubo de rayos catódicos (CRT, del inglés *Cathode Ray Tube*) es una tecnología que permite visualizar imágenes mediante un haz de rayos catódicos constante dirigido contra una pantalla de vidrio recubierta de fósforo y plomo. El fósforo permite reproducir la imagen proveniente del haz de rayos catódicos, mientras que el plomo bloquea los rayos X para proteger al usuario de sus radiaciones.

Karl Ferdinand Braun inventó el osciloscopio de rayos catódicos como curiosidad de la física en 1897, al aplicar una señal oscilante a placas deflectoras cargadas eléctricamente en un tubo de rayos catódicos recubierto de fósforo (tubo Braun). El tubo Braun, mostrado en la Figura 1.5, utilizaba un emisor de cátodo frío y voltajes muy altos (entre 20,000 a 30,000 volts) [Abramson 1995]. En 1989 Jonathan Zenneck equipó el tubo de rayos catódicos con placas de formación de haces y utilizó un campo magnético para realizar el barrido de la traza [Kularatna 2003].

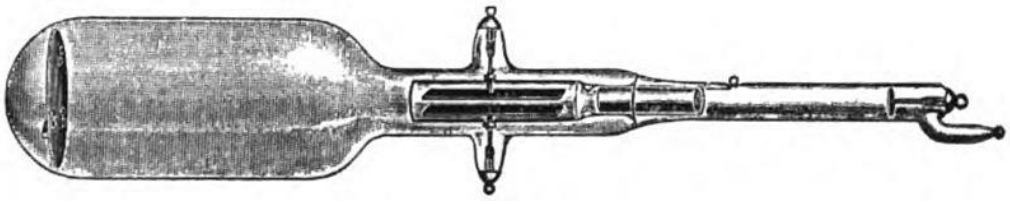

Figura 1.5 Tubo de rayos catódicos de cátodo frío desarrollado por Braun

Los primeros tubos de rayos catódicos se habían aplicado experimentalmente para mediciones de laboratorio a principios de 1919 [Burns 1998] pero sufrían de inestabilidad en el vacío y los cátodos emisores. La aplicación del emisor termoiónico permitió que el voltaje de operación descendiera hasta unos cientos de volts.

V. K. Zworykin describió un tubo de rayos catódicos al alto vacío y permanentemente sellado con un emisor termoiónico en 1931. Este componente estable y reproducible permitió a General Radio fabricar un osciloscopio capaz de utilizarse fuera del entorno de un laboratorio [Kularatna 2003].

Este tipo de osciloscopio fue ampliamente usado durante la segunda guerra mundial para el desarrollo y, posteriormente, mantenimiento del equipo de radar. A pesar de que era extremadamente útil para analizar el comportamiento de circuitos de pulsos, no estaba calibrado y por lo tanto no era posible utilizarlo como un instrumento de medición.

1.1.5 Osciloscopio de barrido disparado

Los osciloscopios se convirtieron en una herramienta de mayor utilidad en 1946 cuando Howard Vollum y Jack Murdock inventaron el osciloscopio de barrido disparado: el Tektronix® 511, el cual se muestra en la Figura 1.6.

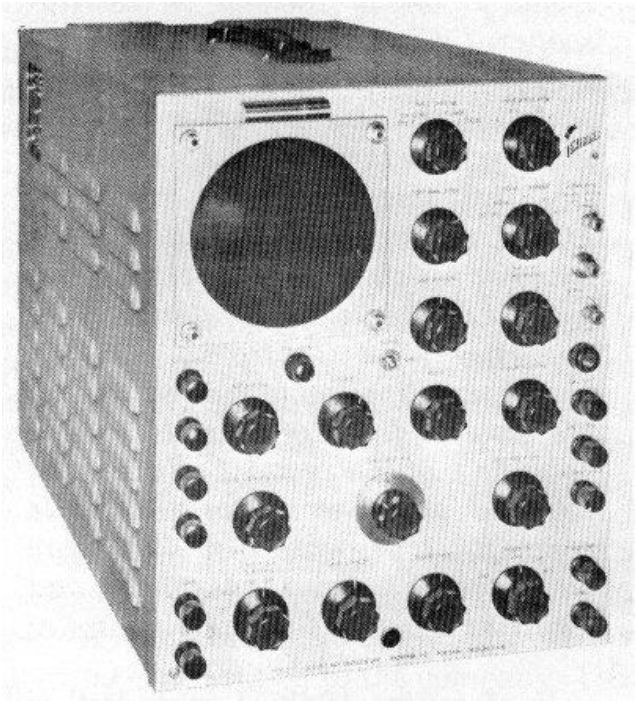

Figura 1.6 Primer osciloscopio de barrido disparado

Antes de que el barrido disparado se utilizara, la deflexión horizontal del haz del osciloscopio era controlada por un generador independiente de una señal de diente de sierra. Si el periodo del barrido horizontal no coincidía con el periodo de la onda a observarse, cada trazo siguiente comenzaría en un lugar diferente de la onda, y por lo tanto se tendría una imagen en continuo movimiento.

El disparo permite una representación estacionara de la onda, debido a que múltiples ciclos de ella se dibujan exactamente sobre el mismo trazo en la pantalla de fósforo. Un barrido disparado mantiene la calibración de la velocidad de barrido, haciendo posible la medición de propiedades de la onda tales como frecuencia, amplitud, fase, tiempo de subida, entre otras, que de otra manera serían imposibles de medir.

Los osciloscopios de barrido disparado compara la deflexión vertical de la señal con un umbral ajustable, denominado nivel de disparo. El circuito de disparo también es capaz de reconocer la pendiente de la señal cuando cruza el umbral (flanco de subida o flanco de bajada), esta función se conoce polaridad del disparo. Cuando la señal cruza el nivel de umbral definido y con la pendiente deseada, el circuito de disparo borra la pantalla del osciloscopio y comienza un barrido lineal exacto.

Las características del barrido disparado hicieron que los osciloscopios se convirtieran en instrumentos extremadamente útiles para mediciones y pruebas.

1.2 Objetivo

1.2.1 Objetivo general

Desarrollar una herramienta inalámbrica de graficación de voltaje de alta velocidad basada en el procesador de señales dsPIC33FJ16GS502 con interfaz Bluetooth y dispositivos móviles con sistema operativo Android. Al finalizar la etapa de pruebas, es posible que esta herramienta sea utilizada por alumnos y profesores de la Facultad de Ingeniería Eléctrica.

1.2.2 Objetivos particulares

Elaborar un manual para el desarrollo de aplicaciones gráficas y de Bluetooth en la plataforma Android.

Diseñar e implementar un circuito de acondicionamiento para realizar mediciones de hasta ±10 V con acoplo AC o DC, y entrada mínima de 10 mV.

Configurar el convertidor analógico-digital a máxima velocidad.

Implementar dos canales de entrada.

Implementar opción de disparo.

Utilizar un algoritmo FTT y proveer el espectro de la señal al usuario.

Probable uso en la Facultad y comercialización de la herramienta.

1.3 Justificación

La tecnología está en constante desarrollo y es importante adquirir nuevos conocimientos para aprovechar ésta al máximo. Adicionalmente, se amplía la capacidad de resolver los problemas con más ingenio y menos recursos materiales.

El osciloscopio es uno de los instrumentos más versátiles que existen y es utilizado principalmente en el diseño y análisis de circuitos electrónicos. Es un dispositivo de medición esencial para el pleno desarrollo de los estudiantes de electrónica, sin embargo, es muy difícil de adquirir para la gran mayoría de los alumnos. Además, los osciloscopios comerciales estándar son voluminosos y en ocasiones pesados, lo que dificulta su traslado.

Utilizando un procesador de señales y un dispositivo Android, es posible desarrollar un osciloscopio digital portátil de bajo costo. De esta manera, los alumnos tendrían la oportunidad de fabricar su propio osciloscopio y utilizarlo para desarrollar sus habilidades fuera de las instalaciones del laboratorio.

Además de contar con un osciloscopio portátil propio de bajo costo, los alumnos de la Facultad de Ingeniería Eléctrica podrán obtener conocimiento de dispositivos que hasta este momento no se encuentran dentro del plan de estudios de la carrera como el módulo Bluetooth y los DSPs. Asimismo, tendrán la oportunidad de conocer los aspectos y programación básica para el desarrollo de aplicaciones en Android. Es una gran oportunidad de contribuir significativamente al desarrollo de los alumnos y de la Facultad.

1.4 Metodología

Investigación bibliográfica del principio de funcionamiento de un osciloscopio de almacenamiento digital. Asistencia clases de Android, consulta de libros y exploración de las APIs distribuidas por Google® para el desarrollo de programas en Android. Revisión de la hoja de datos del procesador de señales dsPIC33FJ16GS502.

Implementación del hardware y, posteriormente, desarrollo del software del procesador de señales y dispositivo Android. Se efectuaron pruebas para evaluar el rendimiento del osciloscopio, así como para el diagnóstico y corrección de errores.

1.5 Contenido de la tesis

En el Capítulo 1 se da una breve introducción a este trabajo, se muestran los antecedentes históricos del osciloscopio, las mejoras y la importancia que ha tenido para el desarrollo de nuevas tecnologías. Se describe el objetivo y, por último, se explica por qué la decisión de desarrollar este tema.

En el Capítulo 2 se muestran los principales usos del osciloscopio, el funcionamiento de los osciloscopios digitales, los instrumentos más modernos y la justificación para la implementación de uno basado en DSP, Bluetooth y Android.

En el Capítulo 3 se abordan los conceptos de hardware: características de los elementos base, diseño e implementación de la etapa de acondicionamiento y, finalmente, el diagrama de conexión general.

En el Capítulo 4 se desarrolla el software del osciloscopio: iniciar la conexión Bluetooth, transmisión y procesamiento de datos, visualización de la señal.

En el Capítulo 5 se presentan las características del dispositivo desarrollado y los resultados obtenidos.

En el Capítulo 6 se exponen las conclusiones generales, aportaciones y trabajos futuros.

Capítulo 2

El osciloscopio de almacenamiento digital

En la actualidad, los osciloscopios analógicos están siendo desplazados por los osciloscopios de almacenamiento digital (DSO), entre otras razones por la facilidad de poder transferir los resultados hacia una computadora para su posterior análisis.

2.1 El Osciloscopio digital

Al igual que un osciloscopio analógico, el digital también permite visualizar una señal eléctrica variante en el tiempo. Mientras los primeros trabajan con señales continuas, el DSO lo hace con señales discretas.

Estos osciloscopios utilizan un convertidor analógico-digital de alta velocidad y circuitos de memoria para registrar y mostrar una representación digital de la onda, produciendo una mayor flexibilidad para el disparo, análisis y visualización, en comparación con los osciloscopios analógicos clásicos. Al contrario de su predecesor analógico, el DSO puede mostrar eventos anteriores al disparo. En la Figura 2.1 se observa un osciloscopio digital típico.

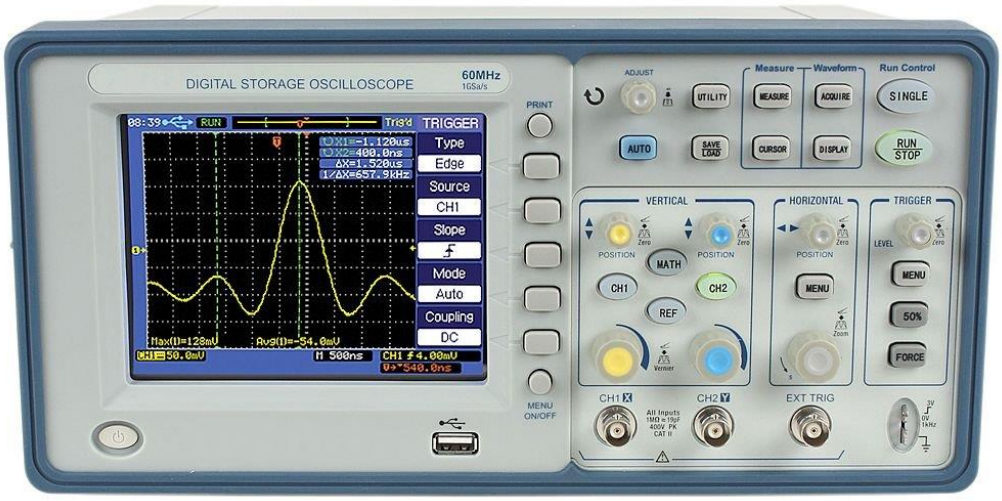

Figura 2.1 Osciloscopio de almacenamiento digital

2.2 Funcionamiento de un osciloscopio digital

Un osciloscopio digital se compone principalmente de cuatro etapas:

1. Acondicionamiento
2. Adquisición
3. Procesamiento
4. Visualización

En la Figura 2.2 se observa el diagrama de bloques de estas etapas.

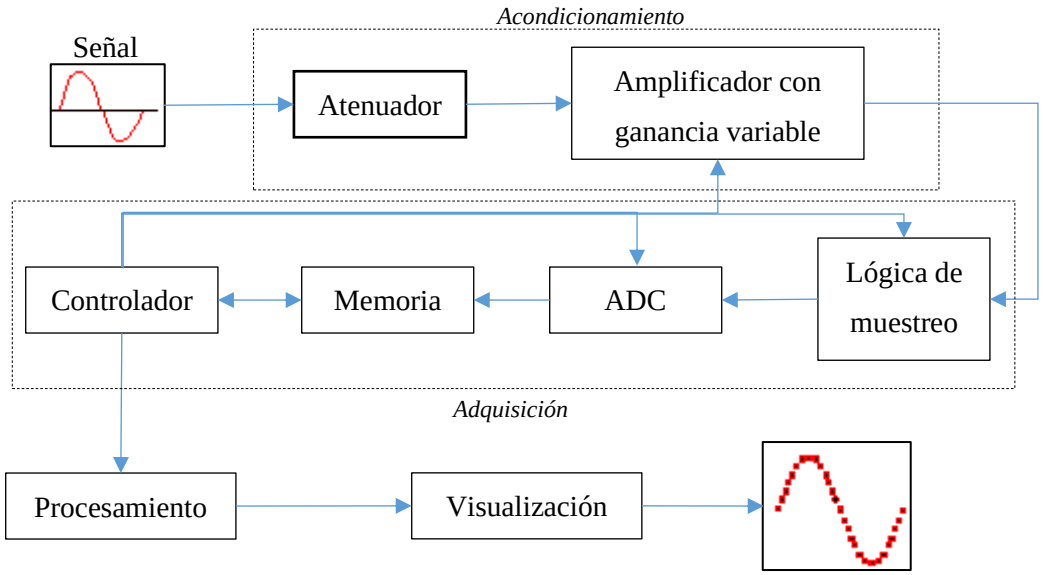

Figura 2.2 Diagrama a bloques del funcionamiento de un osciloscopio digital

2.2.1 Etapa de acondicionamiento

Su función es adaptar la amplitud de la señal de entrada a un voltaje adecuado para posteriormente introducirla en la etapa de adquisición.

Los dispositivos de adquisición trabajan a un nivel de voltaje bajo (generalmente entre 0 y 5 V), por lo que esta etapa es importante para no dañar dicho dispositivo. También es necesario agregar un desplazamiento a la salida para colocar la parte negativa de la señal, si existe, por arriba del nivel de referencia (0 V).

2.2.2 Etapa de adquisición

En esta etapa se convierte la señal analógica en una señal digital; es decir, la amplitud de la señal toma valores discretos dentro de un rango finito. Esto es posible mediante un convertidor analógico-digital.

Los valores de la señal digital son almacenados en memoria para posteriormente procesarlos.

2.2.3 Etapa de procesamiento

Una de las características principales de los sistemas digitales es la gran facilidad que se tiene para manipular los datos. Gracias a esto se pueden implementar algoritmos para calcular la frecuencia, amplitud, voltaje eficaz, e incluso el espectro de frecuencia.

Además en esta etapa también se configura la variable del disparo (trigger), seleccionando entre diferentes opciones como: disparo único, disparo por nivel en flanco de subida o bajada y disparo por ancho de pulso.

2.2.4 Etapa de visualización

Es la etapa de interacción entre el usuario y del osciloscopio digital. Aquí se muestra la señal digital en alguna pantalla, generalmente en una LCD, como una gráfica en dos dimensiones y adicionalmente se pueden agregar las características de la onda (que han sido calculadas en la etapa previa).

2.3 Utilidad de un osciloscopio

Los osciloscopios son utilizados para observar el cambio de una señal eléctrica en el tiempo, de tal manera que el voltaje y el tiempo describen una onda que se grafica constantemente contra una escalada calibrada.

Típicamente, el ojo humano procesa una imagen en 150 microsegundos; es decir, puede procesar entre 30 y 40 imágenes por segundo. Esto significa que si una persona quisiera visualizar una señal de 60 Hz sólo conseguiría ver una serie de ondas superpuestas. Mientras que con la ayuda de un osciloscopio se pueden visualizar señales de varios cientos de MHz, incluso hasta decenas de GHz.

Los osciloscopios son usados en diversas áreas como son ingeniería, medicina, ciencias, telecomunicaciones y más. Los osciloscopios de propósito general se utilizan para el mantenimiento de equipo electrónico y trabajo de laboratorio, mientras que los de propósito específico pueden ser utilizados para analizar el sistema de ignición de un automóvil, como se muestra en la Figura 2.3, o incluso para registrar la actividad eléctrica del corazón, indicado en la Figura 2.4.

Figura 2.3 Osciloscopio digital para diagnóstico de fallas en automóviles

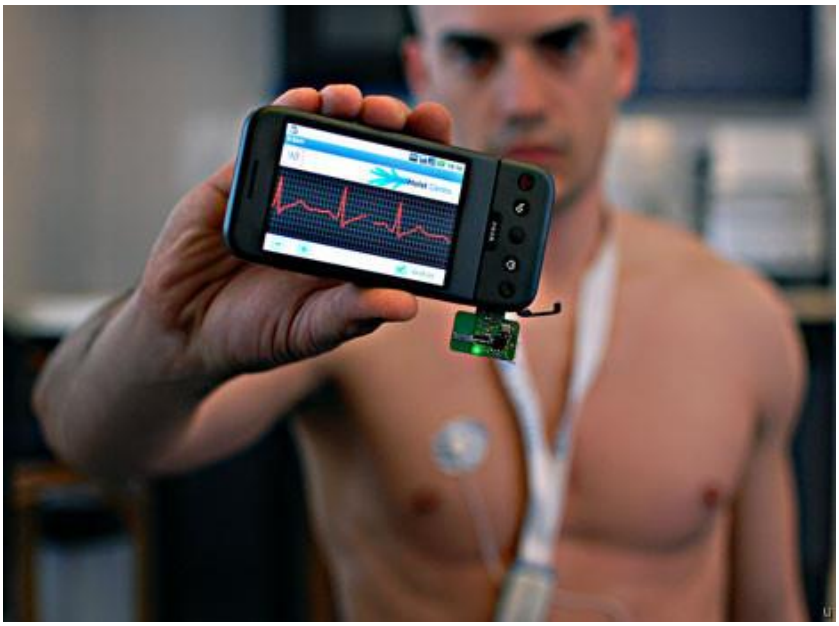

Figura 2.4 Sistema de electrocardiograma portátil basado en Android

2.4 Osciloscopios digitales en la actualidad

Los osciloscopios digitales actuales se basan en los mismos principios que los primeros desarrollados en la década de 1980, pero con el desarrollo de componentes más potentes se han visto mejorados drásticamente.

2.4.1 Osciloscopios digitales de mesa

Son utilizados ampliamente en ciencias e ingeniería para el análisis, reparación y diseño de circuitos electrónicos. Este tipo de osciloscopio es el más confiable para realizar mediciones, pero también presenta la desventaja de ser costoso y no portátil. Un osciloscopio digital de mesa es mostrado en la Figura 2.5.

El precio de éstos va desde los $5,000 pesos, pero incluso se pueden encontrar equipos de más de $300,000 pesos. Tienen el ancho de banda más amplio de entre todas las variedades, el mínimo suele ser 200 MHz y pueden llegar alcanzar hasta más de 60 GHz.

Figura 2.5 Osciloscopio digital de mesa alimentado por baterías

2.4.2 Osciloscopios basados en PC

Debido al reciente predominio de las computadoras personales, los osciloscopios basados en PC se han vuelto comunes. Típicamente, la señal es capturada en hardware externo y transmitida a la computadora, donde es procesada y visualizada. La Figura 2.6 muestra un osciloscopio digital para PC.

Estos osciloscopios se pueden conseguir en un amplio rango de precios que van desde los $800 pesos hasta más de $100,000 pesos, dependiendo de sus capacidades y el uso destinado. Los dispositivos más capaces tienen un ancho de banda de hasta 20 GHz.

Figura 2.6 Osciloscopio digital para PC con conexión USB

2.4.3 Osciloscopios de mano o *handheld*

Son osciloscopios pequeños alimentados por baterías y que, como su nombre lo indica, caben en la palma de la mano. La Figura 2.7 muestra un osciloscopio de mano en operación.

Se componen principalmente de una pantalla LCD y botones físicos. Tienen un puerto USB e incluso una ranura microSD. Suelen tener dos canales de entrada, pero también se pueden encontrar de cuatro. Tienen anchos de banda de hasta 200 MHz, y su precio oscila alrededor de $5,000 y $60,000 pesos.

Figura 2.7 Osciloscopio de mano

2.4.4 Osciloscopios de bolsillo

Son similares a los de mano, a excepción de que sólo tienen algunos botones físicos y son de tamaño más reducido, como se muestra en la Figura 2.8.

Se pueden conseguir en precios de alrededor de $1,500 pesos, y su ancho de banda es de hasta 200 KHz.

Figura 2.8 Osciloscopio de bolsillo

2.4.5 Osciloscopio para protoboard

Es uno de los osciloscopios más pequeños que existen. Cuenta con una pantalla OLED de 128x64 pixeles, conectividad USB para PC y Android. Además de también tener

función de generar señales. La Figura 2.9 muestra este osciloscopio en funcionamiento, montado en protoboard.

Su precio es de poco más de $1,000 pesos y tiene un ancho de banda de hasta 200 KHz.

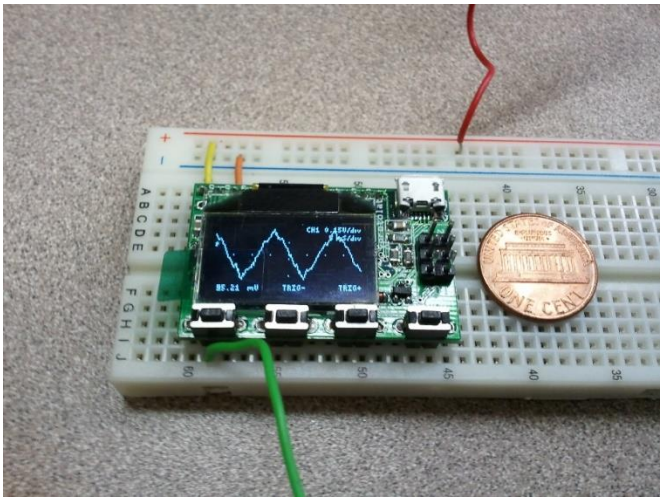

Figura 2.9 Osciloscopio para protoboard

2.4.6 Osciloscopio para dispositivos móviles

Para iOS® se pueden conseguir por alrededor de $5,000 pesos, con las siguientes características: dos canales analógicos y cuatro digitales, 5 MHz de ancho de banda y rango de entrada entre -8 y 13 V. En la Figura 2.10 se muestra un osciloscopio diseñado para iOS.

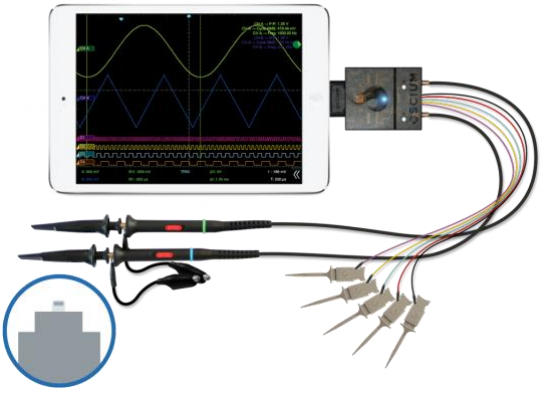

Figura 2.10 Osciloscopio diseñado para dispositivos iOS

Para Android existe una opción de alrededor de $4,000 pesos, y presenta las siguientes características: dos canales analógicos, ancho de banda de hasta 8 MHz y voltaje de entrada entre -16 y 16 V. En la Figura 2.11 se muestra una fotografía de un osciloscopio con conexión USB desarrollado para Android.

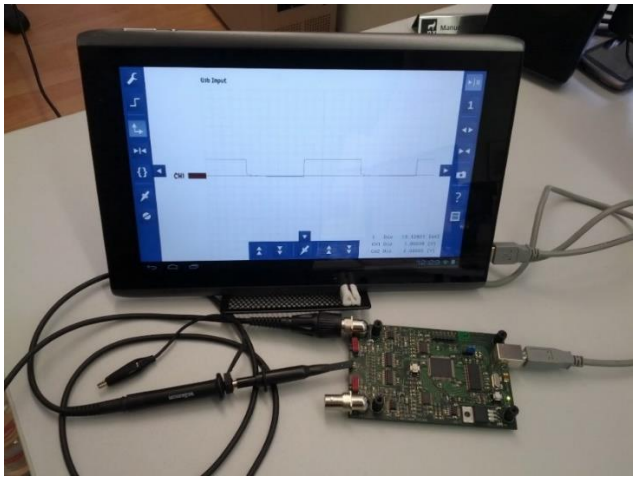

Figura 2.11 Osciloscopio para Android con conexión USB

2.4.7 Reloj osciloscopio

Cuenta con las funciones de un reloj (hora, alarma, calendario, etc.) combinadas con las funciones de un osciloscopio digital y un generador de señales. Su precio es cercano a los $2,000 pesos y cuenta con las siguientes características: dos canales analógicos, ocho canales digitales, ancho de banda de 200 kHz y rango de entrada entre -14 y 20 V. Además cuenta con conexión USB para PC y Android. La Figura 2.12 muestra una serie de fotografías desde distintos ángulos de este dispositivo.

Figura 2.12 Reloj osciloscopio

2.5 Osciloscopio digital basado en procesador de señales, Bluetooth y Android

2.5.1 Microcontrolador o DSP

Como se había mencionado anteriormente, la parte más importante de un osciloscopio digital es el convertidor analógico-digital. El dsPIC, además de contar con un módulo de conversión analógico-digital, también tiene integrado un puerto de comunicación UART, el cual se utiliza para realizar la transferencia de datos hacia el módulo de comunicación Bluetooth.

Se pueden conseguir dsPICs por menos de $100 pesos, por lo que el factor económico queda cubierto.

No se requiere aprender lenguajes complicados de programación. Los programas se pueden escribir en lenguaje C.

En la Facultad de Ingeniería Eléctrica se imparten dos cursos de microcontroladores, por lo que los alumnos de la facultad están familiarizados con ellos y pueden comprender fácilmente el funcionamiento del programa.

2.5.2 Comunicación Bluetooth

La tendencia de las nuevas tecnologías es eliminar la transferencia de datos por cables, los cuales afectan directamente la movilidad de los dispositivos. Además, al quitar la conexión física, protegemos el dispositivo móvil de las conexiones realizadas en el instrumento.

La conexión Bluetooth facilita la portabilidad del programa hacia otros dispositivos que cuenten con adaptador Bluetooth como PC, iOS, etc.

Con el nuevo protocolo de Bluetooth de baja energía (BLE) se ha solucionado el alto consumo de energía de los protocolos Bluetooth anteriores, llegando a consumir sólo 0.04 mA en modo espera y 4 mA en operación.

2.5.3 Android

Android es el sistema operativo más usado en dispositivos móviles con más de mil millones de usuarios actualmente (existen 7,000 millones de habitantes en el mundo) y cada día se activan alrededor de 1.5 millones de dispositivos (celulares y tabletas)

Además de ser el preferido, también es el más barato. Se pueden conseguir dispositivos Android por menos de $1,000 pesos.

Las herramientas y documentos para el desarrollo de programas se encuentran disponibles para cualquier persona en su página web. Aunado a esto, existe una gran cantidad de soporte para nuevos desarrolladores en los foros de internet.

En resumen, en este capítulo se dio a conocer el principio de funcionamiento de un osciloscopio digital, sus principales aplicaciones y se enlistaron distintos tipos de osciloscopios que existen en la actualidad. Además se justificó la implementación de uno utilizando un DSP, comunicación Bluetooth y un dispositivo Android,

A continuación, en el Capítulo 3, se aborda el concepto de todo lo relacionado con el hardware y se mencionan las principales características de los elementos elegidos para el desarrollo del equipo.

Capítulo 3

Hardware

Una de las características más importantes a tomar en cuenta a la hora de elegir un osciloscopio digital es el ancho de banda. Éste es determinado directamente por la velocidad del convertidor analógico-digital.

3.1 Procesador de señales dsPIC33FJ16GS502

3.1.1 Introducción

Los PIC® (Controlador de Interfaz Periférico) son una familia de microcontroladores fabricados por Microchip®. Son muy populares debido a su costo reducido, amplia disponibilidad, gran base de usuarios, vasta colección de notas, bajo costo de herramientas de desarrollo y programación serial.

El PIC usa un juego de instrucciones tipo RISC (Computador con Conjunto de Instrucciones Reducidas), cuyo número puede variar desde 35 para PICs de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada *sleep*.

Los dsPIC son los primeros PICs con bus de datos inherente de 16 bits. Incorporan todas las posibilidades de los anteriores PICs y añaden varias operaciones de procesamiento digital de señales (DSP) implementadas en hardware.

3.1.2 Descripción

El dsPIC33FJ16GS502 es un controlador digital de señales. Integra la funcionabilidad de un microcontrolador de alto desempeño y de un procesador digital de señales.

Su módulo CPU tiene una arquitectura Harvard modificada de 16 bits y un conjunto de instrucciones mejoradas, incluyendo soporte para DSP.

En la Figura 3.1 se observa el diagrama de terminales del dsPIC33FJ16GS502.

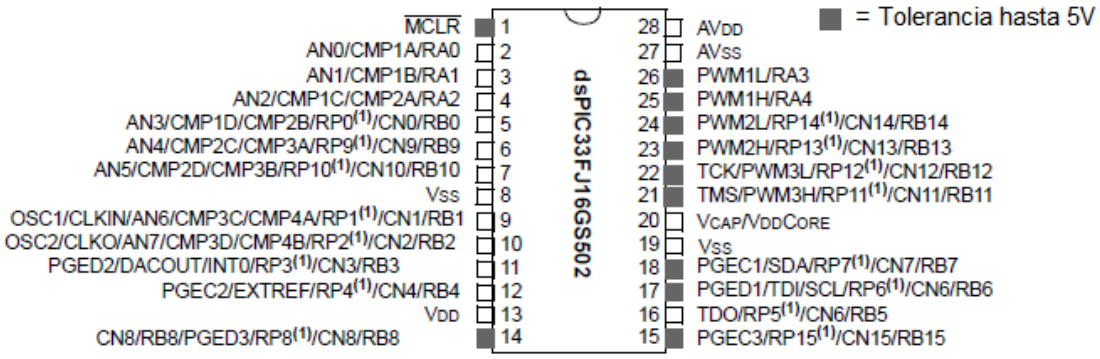

Figura 3.1 Diagrama de terminales del dsPIC33FJ16GS502

3.1.3 Características

Características eléctricas

- Voltaje de alimentación: 3.6 V
- Corriente de operación típica a 40 MIPS: 98 mA

Rango de operación

- Hasta 40 MIPS (a 3.0 – 3.6V)

Entradas/salidas digitales

- Funcionalidad de terminal periférica seleccionable
- 21 entradas/salidas digitales programables

Memoria

- Memoria flash de programa (hasta 16 kB)
- SRAM de datos (hasta 2 kB)

Características de periféricos

- 3 contadores (*timer*) de 16 bits.
- Módulo de comunicación SPI
- Módulo de comunicación UART
- ADC de alta velocidad (10 bits)

Interrupciones

- Hasta 35 fuentes de interrupción
- Siete niveles con prioridad programable

3.1.4 Sistema de reloj

El dsPIC33F cuenta con un oscilador FRC (*Fast RC Oscillator*), que trabaja a una velocidad nominal de 7.37 MHz, y un PLL para escalar la frecuencia interna de operación a la requerida por el reloj del sistema, pudiendo alcanzar la velocidad máxima de operación (80 MHz) sin la necesidad de hardware externo de generación de reloj. El diagrama del sistema de reloj se muestra en la Figura 3.2.

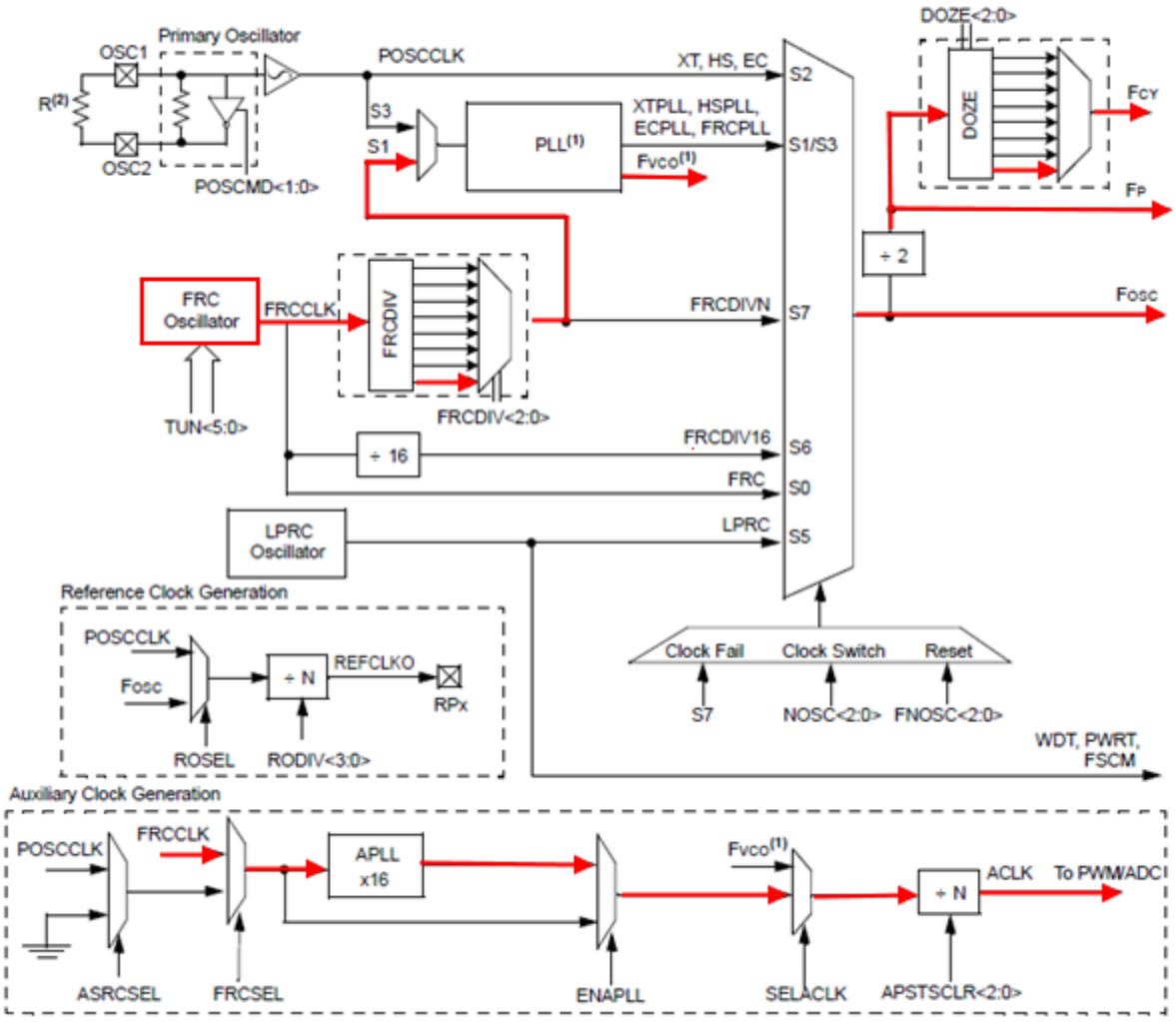

Figura 3.2 Diagrama del sistema de reloj del dsPIC33FJ16GS502

Del diagrama anterior se observa el sistema del reloj de trabajo y del reloj auxiliar (el cual se usa en el ADC). El oscilador utilizado para el reloj de trabajo es el FRC el cual entra a un *postscaler* y a continuación al PLL, donde finalmente se hacen operaciones (como se explicará en el Capítulo 4.1.2) para alcanzar la frecuencia máxima de trabajo. El oscilador

para el reloj auxiliar también es el FRC, éste se introduce en un PLL auxiliar (APLL) para posteriormente entrar en un *postscaler* y finalmente utilizarse como reloj para el ADC.

3.1.5 Puertos de entrada/salida

Todos las terminales del dsPIC (excepto V_{DD}, V_{SS} y $\overline{MCLR}$) son compartidos entre los periféricos y los puertos paralelos de entrada/salida. Generalmente un puerto paralelo de entrada/salida que comparte una terminal con un periférico está subordinado a este último. Los buffers de datos de salida y de señal de control del periférico se proporcionan a un par de multiplexores. Los multiplexores seleccionan si el periférico o el puerto paralelo asociado tienen la pertenencia sobre los datos de salida y señal de control de la terminal. Esta lógica previene un ciclo en el cual un puerto de salida digital pueda controlar la entrada del periférico que comparte la misma terminal. La Figura 3.3 muestra como los puertos son compartidos con otros periféricos y su terminal asociada de entrada/salida a la cual están conectados.

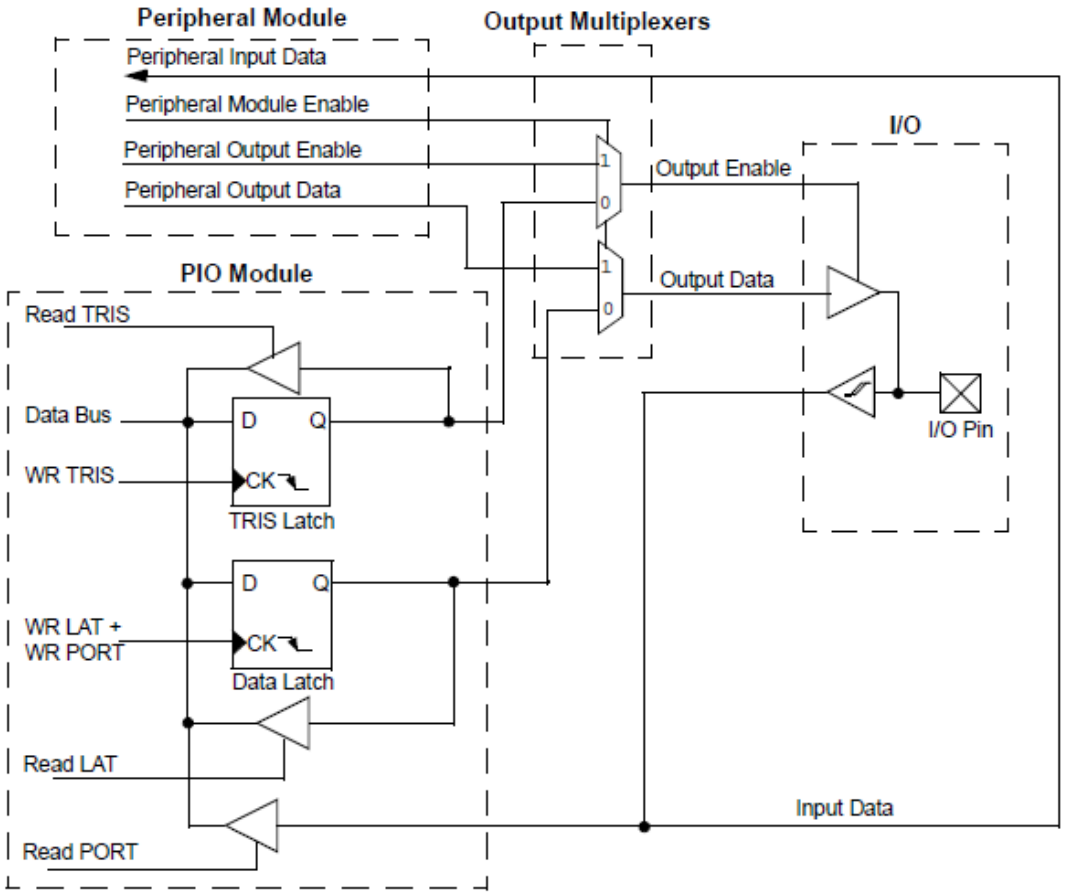

Figura 3.3 Diagrama de la estructura de un puerto típico compartido

Cuando un periférico es habilitado y éste controla activamente la terminal asociada, el uso de la terminal como una salida de propósito general es desactivado. La terminal de entrada/salida puede ser leída, pero el controlador de salida para el puerto paralelo es desactivado. Si el periférico es habilitado, pero no ejerce un control activo sobre la terminal, ésta puede ser controlada por el puerto.

Todos las terminales de puerto tienen tres registros directamente asociados con su operación como una entrada/salida digital. El registro de dato de dirección (TRISx) determina si la terminal de puerto es una entrada o salida. Si el dato de dirección es un *1*, entonces la terminal de puerto es una entrada. Todas las terminales de puerto están definidos como entradas después de un reinicio. Leer el *latch* (LATx) lee el *latch*. Escribir sobre el *latch* escribe el *latch*. Leer el puerto (PORTx) lee la terminal de puerto, mientras escribir sobre la terminal de puerto escribe el *latch*. El valor del *latch* es escrito por el programa, mientras el valor del puerto es el voltaje actual leído sobre la terminal. Generalmente estos valores son iguales, pero existen ocasiones en los que podrían ser diferentes. Por ejemplo, si la terminal se cortocircuita al valor de referencia y se escribe un valor en alto en el *latch*, en éste se leería el valor en alto, pero en el puerto se leería un valor en bajo.

Los registros ADPCFG y TRIS controlan la operación de los puertos del convertidor analógico-digital. Las terminales de puerto que funcionan como entradas analógicas deben tener su correspondiente registro TRIS en *1*. El registro ADPCFG tiene un valor por defecto de 0x0000; por lo tanto, todas las terminales que comparten funciones ANx son analógicas por defecto.

3.1.6 Convertidor analógico-digital de alta velocidad

Utiliza un ADC de aproximaciones sucesivas con las siguientes características:

- Resolución de 10 bits
- Entradas unipolares
- Precisión de ±1 LSB a 3.3V
- 2 Msps por canal

Puede muestrear y convertir simultáneamente dos entradas analógicas en 0.5 microsegundos. Es posible elegir la fuente de disparo (manual, PWM o *timer*) para iniciar la conversión.

La Figura 3.4 muestra el diagrama del convertidor analógico-digital integrado en el dsPIC33FJ16GS502. Éste contiene dos SARs (registro de aproximaciones sucesivas), uno para las entradas pares y otro para las impares. Las entradas pares tienen un circuito amplificador-retenedor dedicado, mientras las entradas impares comparten todas el mismo circuito amplificador-retenedor. A continuación de los SARs, se da formato al dato (dependiendo de la elección del usuario) y se guarda en el registro correspondiente.

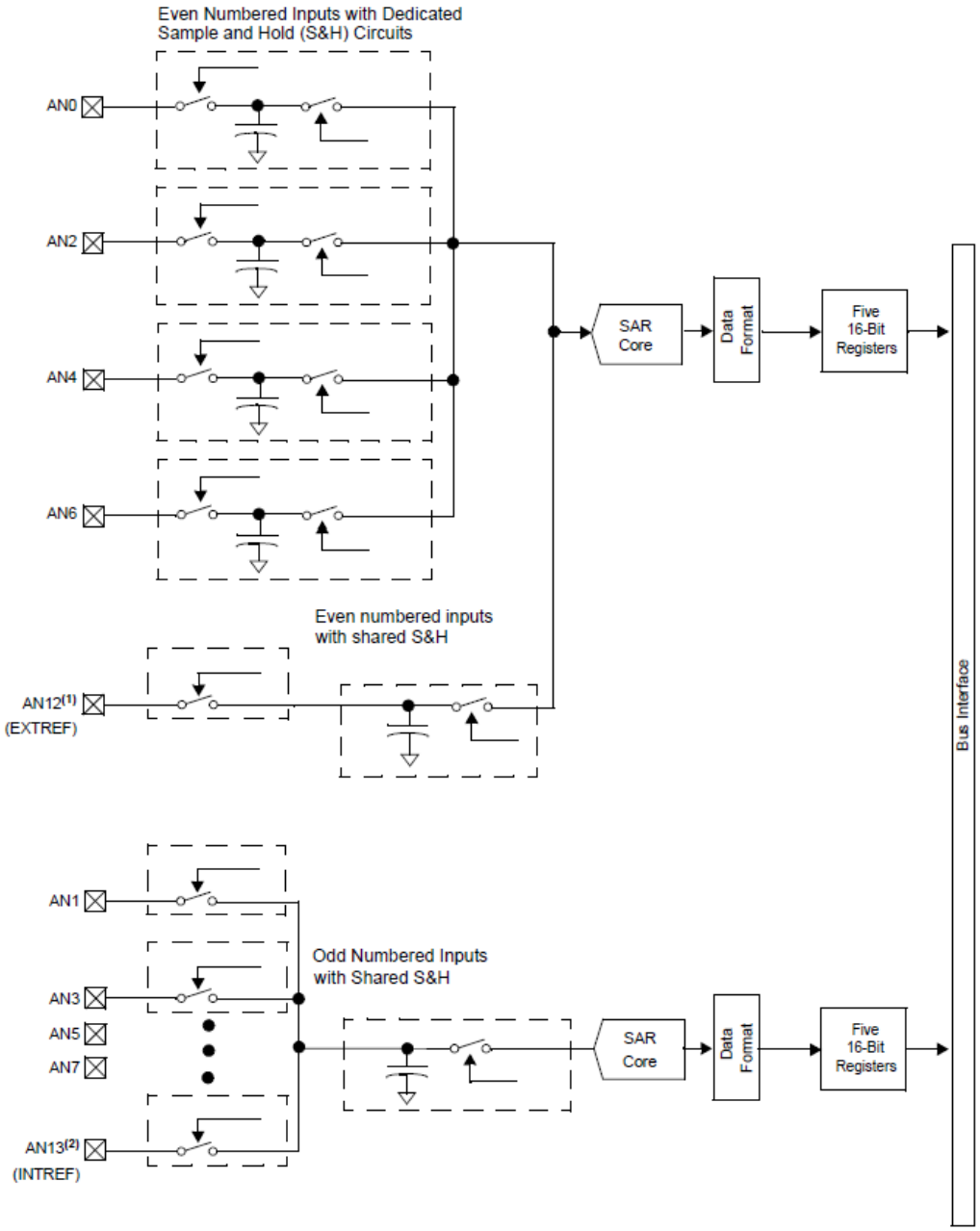

Figura 3.4 Diagrama del ADC del dsPIC33FJ16GS502

3.1.7 Módulo de comunicación UART

Es un sistema asíncrono full dúplex que puede comunicarse con dispositivos periféricos que usan el estándar RS-232, utilizando el adaptador de nivel de voltaje adecuado. Las características principales de este módulo son:

- Comunicación de 8 o 9 bits
- Bit de paridad (par, impar o ninguna) para 8 bits.
- Uno o dos bits de paro
- Generador de *Baud Rate* (15 bps – 4 Mbps)
- Registro FIFO de 4 palabras de profundidad
- Detección de errores
- Interrupciones por recepción y transmisión

3.2 Módulo de comunicación Bluetooth HC-06

3.2.1 Descripción

La finalidad del módulo es reemplazar las conexiones físicas de una comunicación serial. Para la conexión sólo es necesario conectar las terminales RX y TX del módulo Bluetooth a las terminales TX y RX, respectivamente, del dsPIC. Este dispositivo mantiene un socket abierto esperando conexiones, por lo que la conexión debe realizarse hacia un cliente. La Figura 3.5 muestra el diagrama de terminales del módulo.

3.2.2 Características

- Voltaje de alimentación: 3.3V
- Comunicación punto a punto
- Protocolo SPP (*Serial Port Profile*)
- *Baud Rate* por defecto: 9600 (1200 a 1.3M seleccionable)
- Consumo antes de establecer comunicación: 25 mA (promedio)
- Consumo durante la comunicación: 8 mA
- Tamaño: 28mm x 15mm x 2.35mm

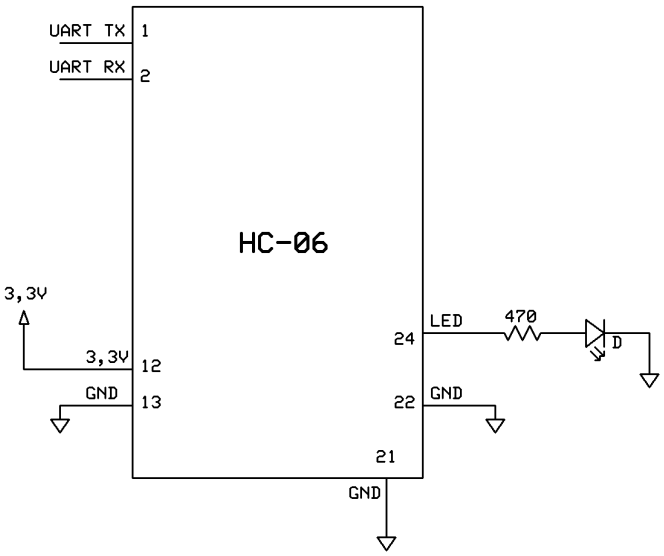

Figura 3.5 Diagrama de terminales del módulo Bluetooth HC-06

3.3 Amplificador operacional rail-to-rail LM6134

3.3.1 Descripción

Contiene cuatro amplificadores operacionales de baja potencia. La característica rail-to-rail permite que el voltaje de salida pueda estar muy cercano al voltaje de las fuentes de alimentación, esto es particularmente importante cuando se utilizan fuentes de bajo voltaje. La Figura 3.6 muestra el diagrama de terminales del LM6134.

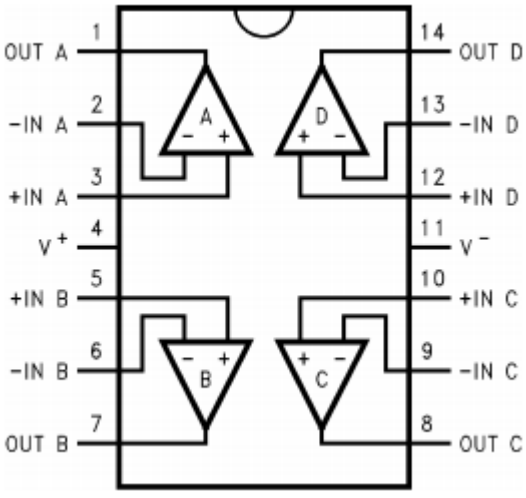

Figura 3.6 Diagrama de terminales del LM6134

3.3.2 Características

- Amplio voltaje de alimentación: 2.7 V a más de 24 V
- Ancho de banda de 10 MHz
- Baja corriente de alimentación: 360 µA por amplificador
- CMRR 100 dB
- 100 dB de ganancia con R_L = 10kΩ
- Voltaje de salida entre 0.01V y 4.99V para fuente simple de 5V

3.4 Interruptor analógico bidireccional TC4066

3.4.1 Descripción

Contiene cuatro interruptores analógicos bidireccionales independientes. Cuando la entrada de control es puesta en alto, la impedancia entre la entrada y salida del interruptor se vuelve baja; cuando es puesta en bajo, la impedancia se vuelve alta. Se puede utilizar tanto en señales analógicas como en digitales. En la Figura 3.7 se puede observar el diagrama de terminales del interruptor analógico bidireccional.

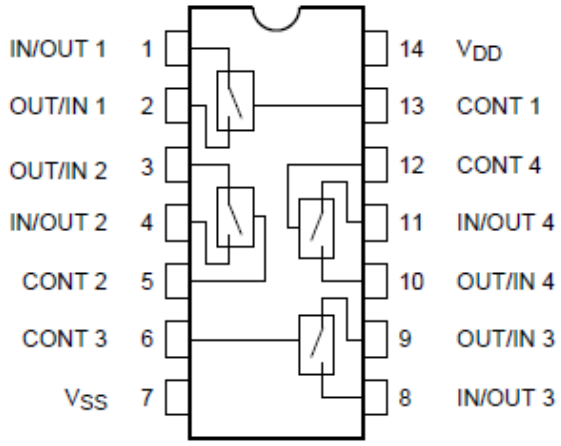

Figura 3.7 Diagrama de terminales del TC4066

3.4.2 Características

- Voltaje de operación: 3V a 20V
- Resistencia en encendido: 50 a 500 Ω
- Resistencia en apagado: >100 MΩ

3.5 Convertidor de voltaje conmutado elevador-reductor ICL7660

3.5.1 Descripción

Es un convertidor que invierte, dobla, divide o multiplica un voltaje de alimentación positivo. Se usa principalmente para generar un voltaje de alimentación negativo a partir de un voltaje positivo para energizar circuitos analógicos. El diagrama de terminales de este convertidor se muestra en la Figura 3.8.

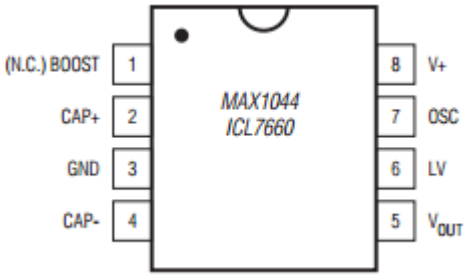

Figura 3.8 Diagrama de terminales del ICL7660

3.5.2 Características

- Voltaje de operación: 1.5 V a 10 V.
- Corriente de salida: 10 mA.
- Eficiencia alimentación-salida: 98%.
- Utiliza 2 capacitores de 10 µF

3.6 Etapa de acondicionamiento

El rango máximo de entrada en el osciloscopio es de ±10 volts y el rango del ADC en el dsPIC es de 0 a 3.3 volts. Debido a esta condición se debe diseñar una etapa de acondicionamiento que entregue una señal positiva en el rango de operación del ADC.

3.6.1 Circuito atenuador

Su función es entregar sólo una fracción proporcional del voltaje que ingresa en él. Se puede implementar con un arreglo de resistores en serie para formar un divisor de tensión como se muestra en la Figura 3.9.

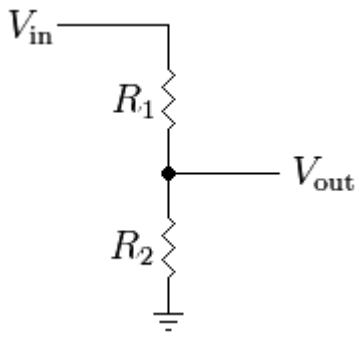

Figura 3.9 Arreglo de resistores en serie

La resistencia estándar en la entrada de un osciloscopio es de 1 MΩ, por lo que la resistencia equivalente del arreglo de resistores debe ser igual a 1 MΩ. La ganancia de un arreglo de resistores en serie está definida por la Ecuación (3.1).

$$\frac{V_{out}}{V_{in}} = A_v = \frac{R_2}{R_1 + R_2} \tag{3.1}$$

La amplitud máxima que se desea medir en la entrada es de 20 V, mientras que la salida debe tener una amplitud máxima de 3.3 V, por lo tanto la ganancia debe ser:

$$A_v = \frac{3.3\ \text{V}}{20\ \text{V}} = 0.165$$

Considerando la resistencia equivalente de 1 MΩ, la ganancia máxima de 0.165 y valores comerciales de resistores, se propone el circuito atenuador mostrado en la Figura 3.10. La impedancia total de este circuito es de 1.002 MΩ y la ganancia de 0.14. El voltaje *V1* de este circuito estará en un rango de ±1.4 V.

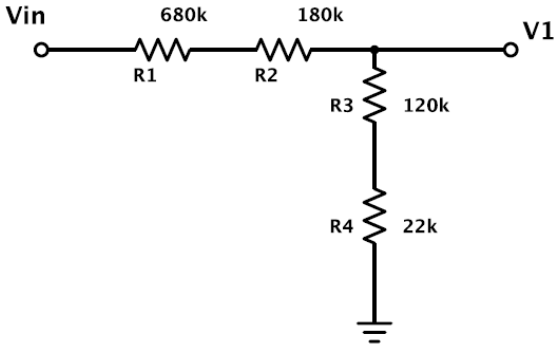

Figura 3.10 Circuito atenuador en etapa de acondicionamiento

3.6.2 Circuito selector de acoplo

Para permitir al usuario elegir acoplo en corriente directa o corriente alterna se utiliza un interruptor analógico controlado digitalmente por el dsPIC y un filtro pasa-altas. Cuando se cierra el interruptor, el filtro se cortocircuita y por lo tanto no tiene efecto; mientras que si el interruptor está abierto, el filtro elimina la componente de DC, permitiendo solamente el paso de corriente alterna.

El diagrama de este circuito se muestra en la Figura 3.11. El voltaje *V2* se encuentra en un rango de ±1.6 V.

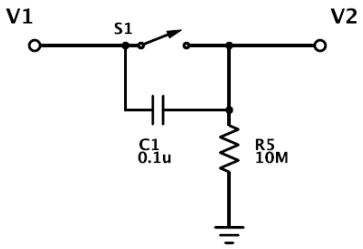

Figura 3.11 Circuito selector de acoplo en etapa de acondicionamiento

3.6.3 Circuito amplificador de ganancia variable

Debido a la atenuación de la señal de entrada en la etapa previa, es necesario utilizar un amplificador con ganancia variable para tener distintos rangos en el osciloscopio. Al igual que con el acoplo, la ganancia se selecciona por medio de interruptores analógicos controlados digitalmente por el dsPIC, los cuales activan o desactivan resistores de retroalimentación en un amplificador no inversor como se muestra en la Figura 3.12.

Se debe considerar que la resistencia del interruptor cuando está cerrado es de ~200 Ω. La ganancia para cada uno de los interruptores se muestra en la Tabla 3.1.

Tabla 3.1 Ganancias seleccionables del circuito amplificador de ganancia variable

Interruptor activado	**Resistencia de retroalimentación**	**Ganancia**	**Rango señal de entrada**
S2	~200 Ω	1.01	±10 V
S3	~82.2 kΩ	5.11	±2 V
S4	~182.2 kΩ	10.11	±1 V

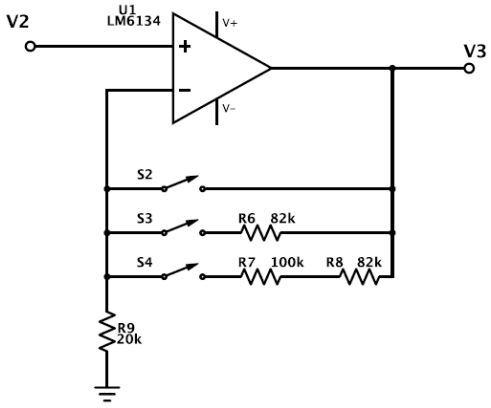

Figura 3.12 Circuito amplificador de ganancia seleccionable en etapa de acondicionamiento

3.6.4 Circuito sumador

Hasta la etapa anterior, el voltaje puede ser negativo. El dsPIC sólo admite voltajes de entrada entre 0 V y 3.3 V, un sumador permite eliminar la parte negativa. El diagrama de un circuito sumador no inversor se observa en la Figura 3.13.

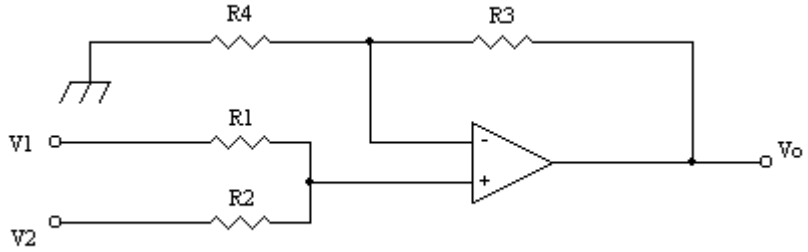

Figura 3.13 Diagrama de un circuito sumador no inversor

El voltaje de salida de un sumador no inversor se determina mediante la Ecuación (3.2).

$$V_o = \frac{V_1 R_2 + V_2 R_1}{R_1 R_2} \cdot \frac{R_3 + R_4}{R_3} \tag{3.2}$$

Si el valor en paralelo de R_1 y R_2 es igual al paralelo de R_3 y R_4, la Ecuación (3.2) se puede reducir como se muestra en la Ecuación (3.3).

$$V_o = \frac{R_3}{R_1} V_1 + \frac{R_3}{R_2} V_2 \tag{3.3}$$

Si las resistencias son iguales, el voltaje de salida está determinado por la Ecuación (3.4).

$$V_o = V_1 + V_2 \tag{3.4}$$

El voltaje de referencia (cuando la entrada está conectada a 0 V) en el dsPIC se determina de acuerdo a la Ecuación (3.5)

$$V_{ref} = \frac{V_{DD} + V_{SS}}{2} \tag{3.5}$$

Para el dsPIC33FJ16GS502, el voltaje de referencia es:

$$V_{ref} = \frac{3.3\ \text{V} + 0\ \text{V}}{2} = 1.65\ \text{V}$$

Sumando 1.65 V a la señal, la relación entre voltaje de la señal de entrada y voltaje de entrada en el dsPIC será la siguiente:

- Los voltajes positivos en la entrada estarán en el rango de 1.65 V a 3.3 V en la entrada del dsPIC.
- El voltaje de referencia en la entrada (0 V) se convertirá en una referencia virtual en el dsPIC con valor de 1.65 V.
- Los voltajes negativos en la entrada estarán en el rango de 1.65 V a 0 V (siendo 0 V el máximo de una señal negativa en la entrada).

El voltaje de alimentación de los circuitos es 3.3 V, un divisor de tensión con resistencias iguales se puede proporcionar 1.65 V para introducirlos en el sumador, junto con la señal proveniente del amplificador de ganancia variable como se observa en la Figura 3.14.

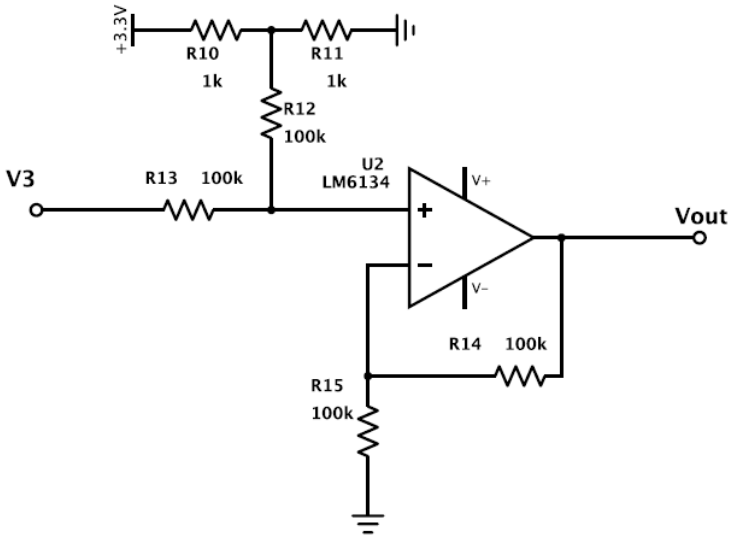

Figura 3.14 Circuito sumador en etapa de acondicionamiento

3.7 Diagrama de conexión

La Figura 3.15 muestra el diagrama de conexión del hardware utilizado para el desarrollo del prototipo.

En resumen, en este capítulo se revisaron todos los elementos que componen el hardware del equipo y sus características, además se diseñó e implementó una etapa para el acondicionamiento de la señal de entrada.

A continuación, en el Capítulo 4, se plantea el software y se establece la unión con el hardware.

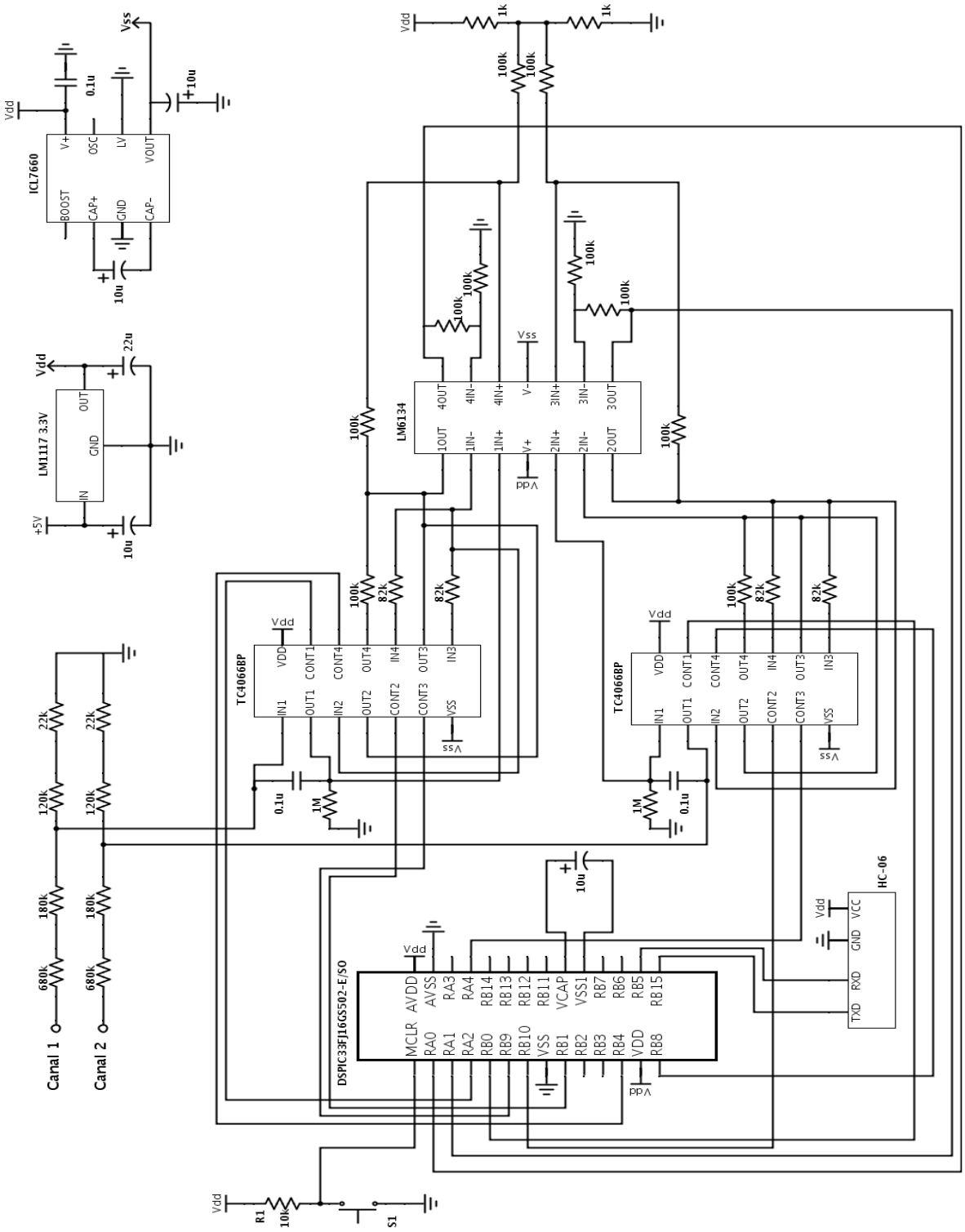

Figura 3.15 Diagrama de conexión del hardware del osciloscopio

Capítulo 4

Software

4.1 Software del dsPIC

El desarrollo de programas para la familia dsPIC se realiza en lenguaje C. Se requieren dos herramientas adicionales proporcionadas por Microchip: el entorno de programación MPLAB® X IDE y el compilador C XC16. Este último compila programas escritos en ANSI C a instrucciones de 16 bits en lenguaje del dsPIC.

4.1.1 Bits de configuración

Los bits de configuración son opciones que controlan el comportamiento del dsPIC en cada reinicio. Esto es particularmente importante en las opciones de oscilador y el *watchdog timer*. Si los bits de configuración están determinados para una fuente de oscilación externa pero ninguna está presente, el dsPIC no iniciará. Estos bits están almacenados en los últimos bytes de la memoria de programa.

En el Anexo A.1 se muestra el código utilizado para establecer los bits de configuración.

4.1.2 Oscilador

Es posible alcanzar la máxima velocidad de operación (80 MHz) del dsPIC utilizando el oscilador interno FRC y el PLL. En la Figura 4.1 se observa el diagrama del PLL y el rango de frecuencia aceptado en cada una de sus etapas.

La salida del oscilador primario o FRC, denotada como $F_{IN,}$ es dividida por un preescalador (N1) antes de ingresar el VCO del PLL. La salida del VCO debe estar en el rango de 0.8 MHz a 8 MHz. El preescalador *N1* es seleccionado usando los bits PLLPRE<4:0> (CLKDIV<4:0>).

El divisor de retroalimentación del PLL, bits PLLDIV<8:0> (PLLFDB<8:0>), proporciona un factor *M* que multiplica a la entrada del VCO. Este factor debe ser elegido de tal manera que la frecuencia de salida del VCO esté en el rango de 100 MHz a 200 MHz.

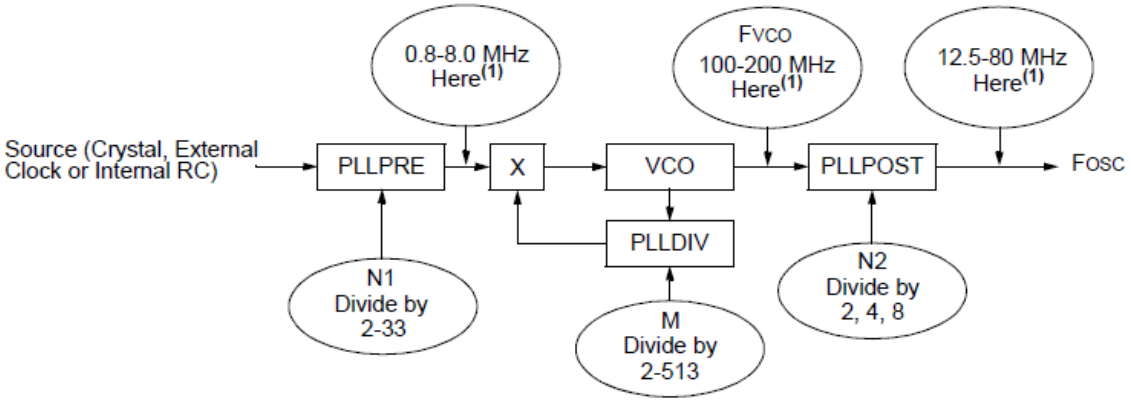

Figura 4.1 Diagrama del PLL en el dsPIC33FJ16GS502

La salida del VCO es posteriormente divida por un post-escalador *N2*. Este factor es seleccionado con los bits PPLPOST<1:0> (CKLDIV<7:6>). *N2* debe ser elegido de tal manera que la frecuencia de salida del PLL (F_{OSC}) esté en el rango de 12.5 MHz a 80 MHz, lo cual genera velocidades de operación de 6.25 a 40 MIPS.

La frecuencia de operación del dsPIC está determinada por la Ecuación (4.1).

$$F_{CY} = F_{OSC}/2 \tag{4.1}$$

Para un oscilador primario o un oscilador FRC con frecuencia de salida F_{IN}, la frecuencia de salida F_{OSC} del PLL está dada por la Ecuación (4.2).

$$F_{OSC} = F_{IN} * \left(\frac{M}{N1 * N2}\right) \tag{4.2}$$

donde:

$$N1 = \text{PLLPRE} + 2$$
$$N2 = 2 * (\text{PLLPOST} + 1)$$
$$M = \text{PLLDIV} + 2$$

La frecuencia nominal del oscilador interno FRC es de 7.37 MHz. Para tener una frecuencia de oscilación máxima (≈80 MHz) la configuración del PLL debe ser:

- PLLPRE<4:0> = 0, así *N1* = 2. La entrada en el VCO es 3.685 MHz.
- PLLDIV<8:0> = 41, así *M* = 43. La salida del VCO es 158.455 MHz.

- PLLPOST<1:0> = 0, así *N2* = 2. F_{OSC} es de 79.22 MHz, F_{CY} = 39.61 MIPS.

El código para la configuración del oscilador se encuentra en el Anexo A.2.

4.1.3 Generación auxiliar de reloj

El reloj auxiliar es utilizado para periféricos que necesitan operar a frecuencias no relacionadas con el reloj del sistema, como es el caso del ADC. El oscilador interno FRC puede ser utilizado junto con un PLL auxiliar (APLL) para obtener un reloj auxiliar de alta velocidad.

Los pasos para activar el reloj auxiliar a máxima velocidad mediante el APLL y el oscilador interno FRC son los siguientes:

a) Poner a *1* el bit FRCSEL (ACLKCON<6>) para seleccionar el oscilador interno FRC como referencia al APLL.
b) Habilitar el APLL escribiendo un *1* en el bit ENAPLL (ACLKCON<15>).
c) Seleccionar el APLL como fuente de reloj para la salida del divisor de reloj auxiliar estableciendo en *1* el bit SELACLK (ACLKCON<13>).
d) Escribir un *7* decimal en APSTSCLR<2:0> (ACLKCON<10:8>) para obtener una post-escalamiento 1:1.
e) Asegurarse de que el APLL está enganchado y listo para operar. Esto se logra mediante el poleo del bit APLLCK (ACLKCON<14>).

El código en C de los pasos anteriores puede consultarse en el Anexo A.3.

4.1.4 Puertos de entrada/salida

En la Tabla 4.1 se muestra la distribución de los puertos utilizados.

Tabla 4.1 Resumen de los puertos utilizados en el dsPIC33FJ16GS502

Terminal	Puerto	TRIS	A/D	Periférico	Descripción
2	AN0	Entrada	Analógico	ADC	Canal 1 del osciloscopio
3	AN1	Entrada	Analógico	ADC	Canal 2 del osciloscopio
4	RA2	Salida	Digital	-	Selección de acoplo en canal 1
5	RB0	Salida	Digital	-	Selección de acoplo en canal 2

6	RB9	Salida	Digital	-	Selección de ganancia en canal 1
7	RB10	Salida	Digital	-	Selección de ganancia en canal 2
9	RB1	Salida	Digital	-	Selección de ganancia en canal 1
12	RB4	Salida	Digital	-	Selección de ganancia en canal 1
14	RB8	Salida	Digital	-	Selección de ganancia en canal 2
15	RP15	Entrada	Digital	UART	Recepción UART (RX)
16	RP5	Salida	Digital	UART	Transmisión UART (TX)
25	RA4	Salida	Digital	-	Selección de ganancia en canal 2

El código utilizado para la configuración de las terminales de puerto puede consultarse en el Anexo A.4.

4.1.5 Configuración del módulo de comunicación UART

El módulo UART se configura con las siguientes características:

- *Baud rate*: 9600
- Bits de datos: 8
- Bit de paridad: ninguno
- Bits de paro: 1

Este módulo tiene integrado un BRG (generador de *baud rate*) de 16 bits. El registro UxBRG controla el periodo de un *timer* de 16 bits de funcionamiento libre. Se puede configurar el bit BRGH (UxMODE<3>) en *1* para generar velocidades altas o en *0* para velocidades bajas. La Ecuación (4.3) muestra la forma de calcular el valor del registro UxBRG para un valor seleccionado de *baud rate* con BRGH=0.

$$UxBRG = \frac{F_{CY}}{16 * (baud\ rate)} - 1 \tag{4.3}$$

Para un *baud rate* de 9600, el valor de UxBRG es:

$$UxBRG = \frac{40 \times 10^6}{16(9600)} - 1 = 259$$

A continuación se muestra el procedimiento para inicializar el módulo de comunicación UART1:

a) Poner a *0* el bit STSEL (U1MODE<0>) para seleccionar un bit de paro.
b) Seleccionar comunicación de 8 bits sin paridad escribiendo *0* en los bits PDSEL<1:0> (U1MODE<2:1>).
c) Deshabilitar el reconocimiento automático de *baud rate* limpiando el bit ABAUD (U1MODE<5>).
d) Seleccionar la generación de velocidades bajas para el *baud rate* poniendo el bit BRGH (U1MODE<3>) en *0*.
e) Escribir en el registro U1BRG el valor decimal *259* para obtener un *baud rate* de 9600.
f) Habilitar el módulo de UART poniendo en *1* el bit UARTEN (U1MODE<15>).
g) Habilitar el control de la terminal U1TX por el módulo UART1 estableciendo un *1* en el bit UTXEN (U1STA<10>).

El código utilizado para inicializar el módulo de comunicación UART se presenta en el Anexo A.5.

4.1.6 Configuración del ADC

El convertidor analógico-digital se configura con las siguientes características:

- Operación en modo *Idle.*
- El reloj es proporcionado por el PLL auxiliar.
- Resultado de conversión en entero.
- La interrupción se genera después de la segunda conversión (conversión simultánea de dos entradas).
- El muestreador-retenedor constantemente está muestreando y termina tan pronto se detecte el disparo de la conversión.
- División del reloj 1:1.

Las entradas utilizadas en el convertidor (AN0 y AN1) corresponden al Par 0, por lo que hay que configurar apropiadamente las interrupciones y el modo de disparo para éste. Se

determinó utilizar el modo de disparo manual debido a que es el que presenta la mayor velocidad.

A continuación se muestra el procedimiento para inicializar el módulo convertidor analógico-digital:

a) Asegurarse de que el convertidor analógico-digital esté desactivado poniendo en *0* el bit ADON (ADCON<15>).
b) Escribir *0x00* en el registro ADCON para configurar la mayoría de las características mostradas anteriormente.
c) Poner en *1* el bit ASYNCSAMP (ADCON<4>) para el continuo muestreo de la señal.
d) Generar interrupciones en el Par 0 escribiendo *1* en el bit IRQEN0 (ADCPC0<7>).
e) Elegir modo de disparo manual en el Par 0 configurando los bits TRGSRC0 (ADCPC0<4:0>) como *0b00001*.
f) Limpiar la bandera de interrupción del Par 0, P0RDY (ADSTAT<0>).
g) Habilitar interrupción general del ADC poniendo en *1* el bit ADIE (IEC0<13>).
h) Habilitar el convertidor escribiendo un *1* en ADON (ADCON<15>).

El código utilizado para inicializar el módulo convertidor analógico-digital puede consultarse en el Anexo A.6.

4.1.7 Almacenamiento en SRAM de muestras

Los resultados de la conversión analógico-digital se guardan en un arreglo *char* con una longitud de 400 para cada canal. Este arreglo se modifica en cada interrupción del ADC, la cual indica que ha finalizado una nueva conversión para ambos canales de entrada.

Debido a que el ADC tiene una resolución de 10 bits y el *char* tiene 8 bits de longitud, sólo se utilizan los 8 bits más significativos del resultado de la conversión. Esto se logra al hacer un corrimiento de 2 bits hacia la izquierda al momento de asignar el dato a la variable.

El resultado de la conversión del canal 1 (AN0) se encuentra en el registro ADCBUF0, mientras que el del canal 2 (AN1) se encuentra en ADCBUF1.

A continuación se muestra el proceso de la función de interrupción:

a) Guardar los 8 bits más significativos del canal 1.
b) Guardar los 8 bits más significativos del canal 2.
c) Limpiar la bandera de interrupción del Par 0, P0RDY (ADSTAT<0>).
d) Limpiar la bandera de interrupción general del ADC, ADIF (IFS0<13>).
e) Iniciar una nueva conversión habilitando el bit SWTRG0 (ADCPC0<5>).

El código de la interrupción del ADC para el almacenamiento de las muestras se muestra en el Anexo A.7.

4.1.8 Algoritmo de disparo (*trigger*)

El disparo es una característica de los osciloscopios que permite estabilizar la imagen de la señal en la pantalla; es decir, dibuja la señal una y otra vez comenzando siempre en el mismo punto.

En este trabajo se presenta únicamente el pre-disparo mediante umbral, y se puede configurarse tanto en flanco de subida como en flanco de bajada. Para flanco de subida se propone el siguiente algoritmo:

a) Definir el nivel en el cual ocurrirá el disparo.
b) Comparar cada uno de los elementos del arreglo que se encuentren entre la posición 100 y 300 (se envían 200 muestras por cada canal) con el nivel del disparo. Se prueban los datos del arreglo y se compara cada uno, si el dato actual del arreglo es menor que el disparo, y el siguiente dato es mayor que éste, se ha cumplido la condición.
c) En caso de que la condición se haya cumplido, elegir este dato como la mitad del arreglo (para poder visualizar eventos anteriores al disparo) y enviar 200 muestras por canal.
d) En caso de la que la condición no se haya cumplido, enviar las primeras 200 muestras para evitar pérdida de información.

La Figura 4.2 muestra el diagrama de bloques de este algoritmo.

El código con el algoritmo implementado se muestra en el Anexo A.8.

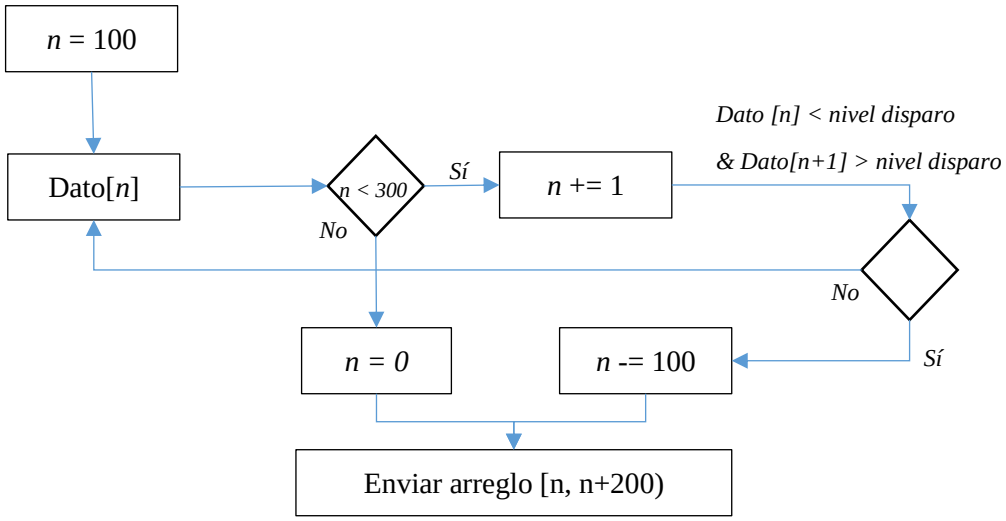

Figura 4.2 Diagrama de bloques del algoritmo de disparo

4.1.9 Transmisión de datos mediante UART1

Una vez que se tienen ambos vectores (uno por canal) listos con datos, se procede a transmitir la información. Es importante primero enviar uno o varios datos conocidos para saber con certeza dónde comienza cada uno de los arreglos para no mezclarlos; además, en caso de que el disparo esté activado y se haya cumplido, la señal siempre se dibuje comenzando en el mismo punto.

Se determinó utilizar dos datos como marca de inicio: *0xFE* y *0x01*. A continuación de éstos se envía el arreglo con las 200 muestras del canal 1 (AN0) y finalmente las 200 muestras del canal 2 (AN1).

A continuación se muestran los pasos a realizar para efectuar correctamente la transmisión de datos:

a) Esperar a que la última transmisión se complete verificando que el bit TRMT (U1STA<8>) se encuentre en *1*.
b) Cargar el dato en el registro U1TXREG.

El código utilizado para la transmisión de los datos está disponible en el Anexo A.9.

4.1.10 Recepción de datos mediante UART1

El proceso de recepción de datos es más delicado que el de transmisión, debido a que el usuario puede modificar las características del osciloscopio en cualquier momento, incluso durante una transmisión. Por ello es necesario utilizar una interrupción por recepción de datos, la cual por defecto tiene mayor prioridad que la del convertidor analógico-digital, por lo que no es necesario modificar el registro de prioridades.

En la Tabla 4.2 se muestran los comandos utilizados y su respectiva función en el dsPIC.

Tabla 4.2 Lista de comandos para modificar características del osciloscopio

Comando	Función
0x01 + 0xHH + 0xLL	Cambia el periodo de muestreo
0x02 + 0x01 + 0x00	Modificar volt/div en canal 1 con ganancia mínima
0x02 + 0x01 + 0x01	Modificar volt/div en canal 1
0x02 + 0x01 + 0x02	Modificar volt/div en canal 1 con ganancia máxima
0x02 + 0x02 + 0x00	Modificar volt/div en canal 2 con ganancia mínima
0x02 + 0x02 + 0x01	Modificar volt/div en canal 2
0x02 + 0x02 + 0x02	Modificar volt/div en canal 2 con ganancia máxima
0x03 + 0x00 + 0xXX	Disparo desactivado
0x03 + 0x01 + 0xHH	Disparo activado en canal 1, flanco de subida
0x03 + 0x02 + 0xHH	Disparo activado en canal 1, flanco de bajada
0x03 + 0x03 + 0xHH	Disparo activado en canal 2, flanco de subida
0x03 + 0x04 + 0xHH	Disparo activado en canal 2, flanco de bajada
0x04 + 0x01 + 0x00	Acoplo AC en canal 1
0x04 + 0x01 + 0x01	Acoplo DC en canal 1
0x04 + 0x02 + 0x00	Acoplo AC en canal 2
0x04 + 0x02 + 0x01	Acoplo DC en canal 2

Donde *0xHH y 0xLL*, depende directamente de la selección del usuario como se verá en los temas 4.1.11, 4.1.12 y 4.1.13, respectivamente. Mientras que el valor de *0xXX* es indiferente y sólo se usa para completar el estándar de tres datos por comando.

A continuación se muestra el proceso de la interrupción por recepción de datos:

a) Determinar el primer dato del comando.
b) Esperar a que se complete la recepción de un nuevo dato.
c) Determinar el segundo dato del comando.
d) Esperar a que se complete la recepción de un nuevo dato.
e) Determinar el tercer dato del comando.
f) Realizar las operaciones correspondientes.
g) Apagar la bandera de interrupción U1RXIF (IFS0<11>).

El código con la rutina de interrupción por recepción de datos puede consultarse en el Anexo A.10.

4.1.11 Configuración del periodo de muestreo (time/div)

La configuración del periodo de muestreo se puede conseguir con diferentes procedimientos:

- Modificar el reloj del ADC.
- Esperar un periodo de tiempo entre las conversiones.
- Ignorar un cierto número de conversiones.

Se optó por la última opción debido a que la modificación del reloj sólo permite pocos cambios, mientras que esperar un cierto retardo entre las conversiones disminuye el ancho de banda a la mitad (haciendo la conversión más rápida por lo menos en el doble de tiempo que sin retardo).

Como se observó anteriormente, el comando para modificar el periodo de muestre se compone de tres datos: *0x01*, *0xHH*, *0xLL*. El primer dato indica al dsPIC que el usuario requiere cambiar el periodo de muestreo, mientras que los otros datos son integrados en un *int* (*0xHHLL*) que indica el número de conversiones que se deben ignorar antes de guardar una en el arreglo de muestras.

Es posible dejar pasar desde 0 conversiones (guardar en cada una de ellas), con lo que se consigue observar señales de 50 kHz; hasta un máximo de 65535, para señales alrededor de 1 Hz.

El código de las funciones relacionadas con la modificación del periodo de muestreo pueden consultarse en el Anexo A.7 y en al Anexo A.10.

4.1.12 Selección de la ganancia en la sección de acondicionamiento (volt/div)

La ganancia en la etapa de acondicionamiento se selecciona por medio de un puerto de salida que controla el estado de un interruptor analógico: en *1* el interruptor está cerrado, mientras que en *0* está abierto.

El comando para modificar la ganancia se compone de tres datos:

- *0x02* que indica la opción de modificar la ganancia.
- *0x01* o *0x02* para seleccionar el canal 1 o el canal 2, respectivamente.
- *0x00*, *0x01* o *0x02* para seleccionar apagar o encender los interruptores dependiendo de la selección del usuario (donde *0x00* es la ganancia mínima y *0x02* la ganancia máxima).

El código de la función relacionada con la selección de la ganancia se encuentra en el Anexo A.10, en la sección respectiva.

4.1.13 Selección del modo de acoplo

Al igual que la ganancia, el acoplo se selecciona por medio de un puerto de salida que controla el estado de un interruptor analógico.

El comando para modificar la ganancia se compone de tres datos:

- *0x04* que indica la opción de modificar el acoplo.
- *0x01* o *0x02* para seleccionar el canal 1 o el canal 2, respectivamente.
- *0x00* para elegir el modo de corriente alterna o *0x01* para el modo de corriente directa.

El código de la función relacionada con la selección del modo de acoplo se encuentra en el Anexo A.10, en la sección respectiva.

4.2 Software en Android

Las aplicaciones se desarrollan habitualmente en el lenguaje Java y se necesita tener instalado el kit de desarrollo *Android Software Development Kit* (Android SDK) provisto por Google, el cual se puede descargar gratuitamente. Todas las aplicaciones están comprimidas en formato APK y se pueden instalar sin dificultad desde cualquier explorador de archivos en la mayoría de dispositivos.

El SDK también cuenta con emuladores de las diferentes versiones de Android, pero éstos cuentan con algunas limitantes como el hecho de no soportar la emulación del adaptador Bluetooth. El emulador es útil para obtener un vistazo del aspecto de una aplicación en diferentes tamaños y densidades de pantalla.

Además del SDK, se requiere un entorno de desarrollo integrado (IDE) y el kit de desarrollo *Java Development Kit* (JDK). El IDE más utilizado para el desarrollo de programas en Android es Eclipse, el cual es gratuito.

4.2.1 Introducción a Android

Android es de las plataformas más populares por el simple motivo de que se encuentra en los móviles. Android es un sistema operativo basado en Linux. Se desarrolló por una compañía llamada Android, Inc. En 2005 Google adquiere la empresa para seguir trabajando en el mismo proyecto que después conociera la luz como un sistema operativo para móviles denominado finalmente como Android.

4.2.2 Estructura de Android

Ya se mencionó que Android está basado en Linux. Para ser más específicos, hablamos del kernel. Android utiliza como base el kernel de Linux.

En la Figura 4.3 se muestra la estructura de Android, donde se pueden observar 5 capas: kernel, bibliotecas, tiempo de ejecución, marco de aplicación y aplicaciones.

Kernel

En esta capa se encuentra el corazón de Android: el manejo de memoria, procesos, drivers, etc. Aquí es donde se da la comunicación con el hardware y se administran los recursos, memoria, energía, etc. Esta capa sirve para no tener dificultades con los distintos fabricantes de dispositivos, si se desea utilizar la cámara del dispositivo no se requiere saber

cómo funcionar la cámara del fabricante X o la del fabricante Y, solamente hay que utilizar las funciones proporcionadas.

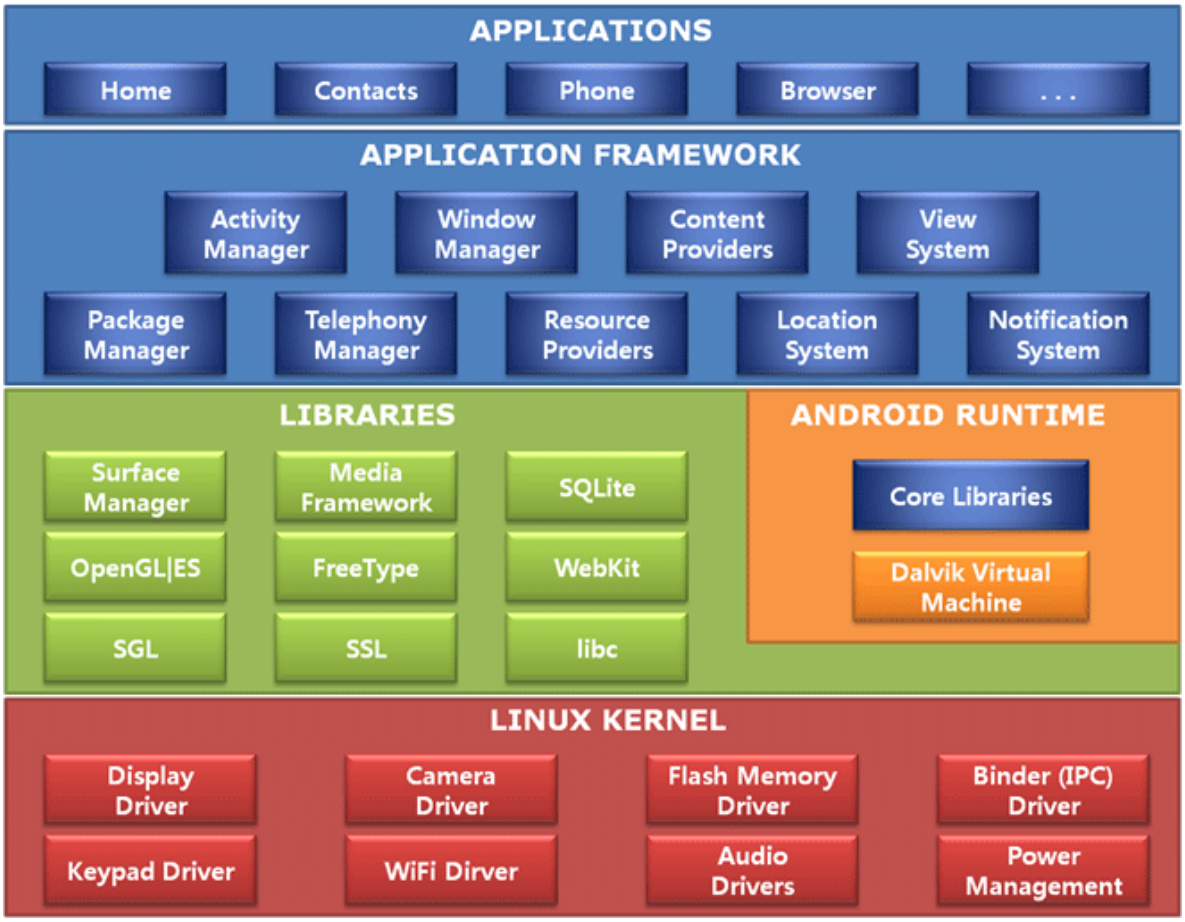

Figura 4.3 Estructura del sistema operativo Android

Bibliotecas

Esta capa tiene las bibliotecas nativas de Android, están escritas en C o C++ y tienen tareas específicas:

- *Surface manager*: gestión del acceso a la pantalla.
- *Media Framework*: reproducción de imágenes, audio y vídeo.
- *SQLite*: bases de datos.
- *Webkit*: navegador.
- *SGL*: gráficos 2D.
- *OpenGL*: gráficos 2D intensos y gráficos 3D.
- *Freetype*: renderización de vectores o imágenes.

Tiempo de ejecución

Aquí se encuentra *Dalvik*, la máquina virtual de Android, que no es lo misma que la *Java Virtual Machine*. Esto quiere decir que cuando compilamos en Java lo que se genera solamente va a funcionar en la JVM, porque *Dalvik* es una máquina virtual, pero de Android, así que el *bytecode* que genera Java es inservible para *Dalvik*.

Algunas de las características de *Dalvik* son:

- Trabaja en entorno con restricción de memoria y procesador.
- Ejecuta el formato .dex.
- Convierte .class en .dex.

Marco de aplicación

Esta capa es la es más visible para el desarrollador, ya que la mayoría de los componentes que forman parte del desarrollo se encuentran aquí:

- *Activity Manager*: rige las actividades de nuestra aplicación y el ciclo de vida.
- *View System*: administra la interfaz gráfica e interacción con el usuario.
- *Notification System*: dispone las notificaciones.
- *Telephony Manager*: administra telefonía, llamadas, mensajes.
- *Resource Provider*: gestiona los recursos de la aplicación, como los xml, imágenes, sonido.
- *Location System*: gestiona la posición geográfica.

Aplicaciones

Aquí se encuentran las aplicaciones que vienen integradas con el dispositivo móvil: reproductor de música, gestor de correos, mensajes, etc.

4.2.3 Actividades en una aplicación

El concepto *Activity* (actividad) es muy utilizado en Android, pero no es sencillo de comprender al principio. Una sola aplicación puede tener una o más actividades. Es válido decir que todas las diferentes pantallas o ventanas de la aplicación son actividades, aunque existen excepciones. Si una aplicación tiene 5 diferentes pantallas, entonces tendrá 5 diferentes actividades.

Casi todas las actividades interactúan con el usuario, para lo cual la clase *Activity* crea una ventana en la que se puede colocar la interfaz gráfica con la función `setContentView(View)`. Mientras la mayoría de las actividades se presentan como una ventana en pantalla completa, se pueden utilizar de otras maneras: como una ventana flotante o integrada en otra actividad. Existen dos métodos que todas las subclases de *Activity* deben implementar:

- `onCreate(Bundle)`: es donde se inicializa la actividad.
- `onPause()`: es donde se trata la situación de que el usuario salga de la actividad. Aquí deberían guardarse los cambios que el usuario realizó para un próximo lanzamiento de la actividad.

4.2.4 Ciclo de vida de una actividad

Las actividades en el sistema se administran como un *activity stack* (pila de actividades). Cuando una nueva actividad es lanzada, se coloca en la cima de la pila y se convierte en la actividad en ejecución. La actividad anterior siempre permanece debajo en la pila, y no será visible hasta que la nueva actividad finalice.

Una actividad tiene básicamente cuatro estados:

- Si la actividad está visible, se encuentra activa.
- Si la actividad ha perdido el foco pero aún es visible, está pausada. Una actividad en pausa está completamente viva (mantiene el estado e información de sus miembros y se encuentra ligada al administrador de ventanas), pero puede ser finalizada por el sistema en situaciones críticas de baja memoria.
- Si la actividad es completamente cubierta por otra actividad, se encuentra detenida. Todavía retiene el estado e información de sus miembros, sin embargo, no se encuentra visible al usuario por lo que su ventana se oculta y típicamente será finalizada por el sistema cuando éste necesite memoria.
- Si la actividad es pausada o detenida, el sistema puede quitarla de la memoria finalizándola, o bien pidiendo al usuario que la finalice. Cuando esta actividad vuelva a ser visible, será reiniciada y restaurada a su estado anterior.

El diagrama de la Figura 4.4 muestra las rutas de los estados de una actividad. Los rectángulos representan métodos *callback* que se pueden implementar para realizar ciertas

operaciones cuando la actividad se mueve entre estados, mientras que los óvalos representan estados importantes en los que la actividad puede encontrarse.

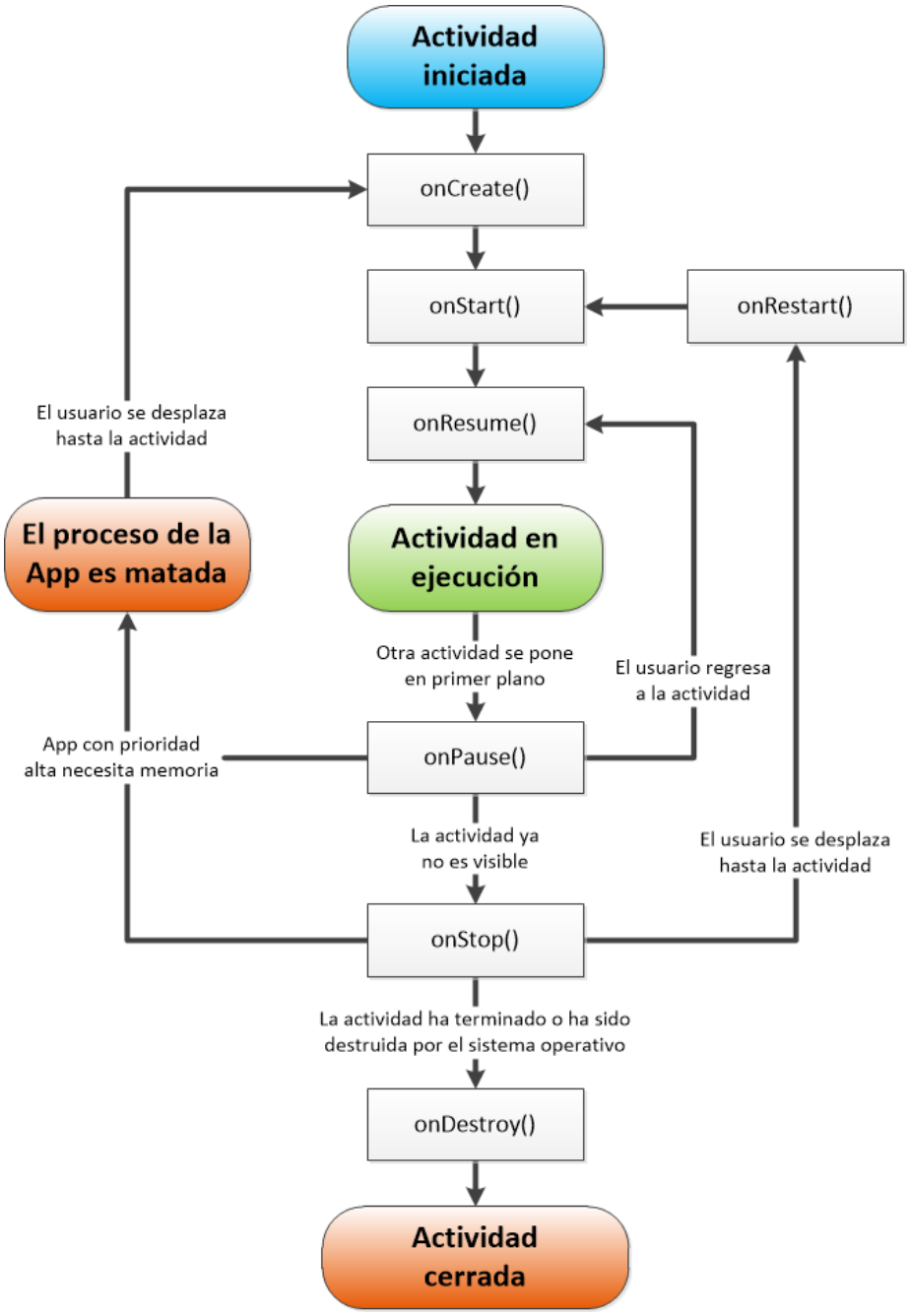

Figura 4.4 Ciclo de vida de una actividad en Android

Es importante mencionar que los métodos `onStop()` y `onDestroy()` no siempre son llamados, por lo que cualquier información que quiera preservarse deberá hacerse en el método `onPause()`.

4.2.5 Permisos de aplicación

Android tiene un sistema de seguridad muy elaborado. Cada aplicación es ejecutada en su propio proceso y máquina virtual, con su propio usuario Linux y grupo, y no puede influir sobre otras aplicaciones.

Android también restringe el uso de los recursos del sistema, como los servicios de red, la memoria SD, el hardware para grabar audio, etc. Si la aplicación requiere usar cualquiera de los recursos del sistema, se requiere solicitar permiso. Esto se hace en el *Manifest* el cual es un archivo que contiene información sobre todas las actividades, los permisos, versiones de Android compatibles con la aplicación, etc.

Al instalar una aplicación, se muestra al usuario una lista con los permisos que requiere la aplicación y mediante ello puede decidir proceder con la instalación o cancelarla

Para la aplicación del osciloscopio sólo se necesita solicitar dos permisos:

- `android.permission.BLUETOOTH`: necesario para establecer comunicación Bluetooth con los dispositivos sincronizados.
- `android.permission.BLUETOOTH_ADMIN`: necesario para encender y apagar el adaptador Bluetooth, escanear nuevos dispositivos y para sincronizarse con ellos.

Es importante señalar que si no se declaran los permisos necesarios para una determinada acción, al ejecutarla, la aplicación se cerrará.

4.2.6 Hilo principal e hilos en paralelo

La interfaz de usuario (UI) en los sistemas operativos modernos (como Android) no trabaja en tiempo real. Utilizan un paradigma basado en eventos, donde el sistema operativo informa a la aplicación los eventos de entrada, así como cuando renderizarse a sí misma. Esto se consigue con *callbacks* que la aplicación registra con el sistema operativo en el arranque; éstos son los responsables de procesar las notificaciones de los eventos recibidos. Lo anterior ocurre en el llamado hilo UI, el hilo principal de nuestra aplicación UI. Es una buena idea regresar de los *callbacks* tan rápido como sea posible, por lo que no es recomendado hacer operaciones intensivas en ellos.

En el hilo UI se procesa todo lo relacionado con la interacción del usuario, por lo que si se colocan demasiadas operaciones o métodos de bloqueo en este hilo, el programa no

responderá inmediatamente a los eventos del usuario, lo cual se puede percibir como que la aplicación dejo de funcionar. En Android, si el hilo UI de la aplicación tarda entre 3 – 5 segundos en responder, se presenta un cuadro de diálogo al usuario en el que se le solicita finalizar la aplicación.

Los hilos en paralelo se ejecutan "simultáneamente" con el hilo principal. En ellos se pueden ejecutar operaciones intensivas o métodos de bloqueo, sin que esto afecte directamente al hilo principal y, en consecuencia, a la aplicación.

La comunicación Bluetooth requiere 2 hilos, uno para establecer la conexión y otro para administrarla. El primero sólo se utiliza de manera temporal hasta que el segundo hilo es lanzado, el cual estará en ejecución hasta que, típicamente, la aplicación finalice. Es importante ejecutar la administración de la conexión en un hilo dedicado, ya que la lectura de datos es un método de bloqueo.

4.2.7 Comunicación entre hilos a través de un Handler

Un *handler* permite enviar y procesar *mensajes* y objetos ejecutables asociados con la cola de mensajes de un hilo determinado. Cada instancia de un *handler* es asociado únicamente con un hilo y su cola de mensajes. Cuando se crea un *handler*, éste queda ligado al hilo que lo creó, y entregará mensajes y ejecutables a esa cola de mensajes y serán ejecutados tan pronto como salgan de ahí.

Existen dos razones principales para usar un *handler*:

- Programar mensajes y ejecutables que serán llevados a cabo en algún momento en el futuro.
- Colocar en la cola (encolar) una acción que será ejecutada en un hilo diferente.

El método `sendMessage(Message)` permite encolar un objeto *message* que contiene un paquete de datos que serán procesados por el método `handleMessage(Message)` del *handler* (se requiere implementar una subclase de *handler*).

Cuando se envía a un *handler*, se puede elegir procesar el mensaje tan pronto como sea posible, o elegir que transcurra un tiempo antes de hacerlo.

Cuando un proceso es creado para la aplicación, su hilo principal se dedica a ejecutar una cola de mensajes que se encarga de administrar los objetos de alto nivel de la aplicación

(como las actividades) y las ventanas. Se pueden crear hilos en paralelo, y comunicarse hacia el hilo principal a través de un *handler*. Esto se hace con el método `handleMessage(Message)`, como se había comentado anteriormente, pero desde el nuevo hilo.

4.2.8 Conexiones Bluetooth en Android

Android incorpora las herramientas necesarias para intercambiar datos con otros dispositivos a través de una red Bluetooth.

Una aplicación Android puede:

- Buscar dispositivos Bluetooth.
- Obtener los dispositivos sincronizados anteriormente.
- Establecer canales RFCOMM (emulación de puerto serie).
- Conectar a otros dispositivos.
- Transmitir y recibir datos.
- Administrar múltiples conexiones.

Las clases e interfaces básicas para administrar las estas conexiones son: `BluetoothAdapter`, `BluetoothDevice`, `BluetoothSocket` y `BluetoothServerSocket`.

BluetoothAdapter

Representa el adaptador Bluetooth local. Es el punto de partida para toda la interacción Bluetooth. Con esta clase se pueden buscar nuevos dispositivos, obtener la lista de dispositivos sincronizados, instanciar un *BluetoothDevice* (dispositivo Bluetooth) a través de su dirección MAC y crear *BluetoothServerSocket* (socket de servidor) para establecer *conexión con otros dispositivos.*

BluetoothDevice

Representa el dispositivo Bluetooth remoto. Se utiliza para requerir la conexión con un dispositivo remoto a través de un *BluetoothSocket*, o para obtener información del dispositivo como nombre, dirección, clase y estado.

BluetoothSocket

Representa la interfaz de un socket Bluetooth (similar a un socket TCP). Este es el punto de conexión que permite a la aplicación el intercambio de información con un dispositivo Bluetooth a través de *InputStream* (flujo de entrada) y *OutputStream* (flujo de salida).

BluetoothServerSocket

Representa un socket servidor que escucha conexiones entrantes (similar a un socket de servidor TCP). Cuando el dispositivo remoto realiza una petición de conexión y es aceptada, el *BluetoothServerSocket* regresa un *BluetoothSocket* conectado.

4.2.9 Inicialización y conexión del adaptador Bluetooth

A continuación se muestra el procedimiento para inicializar el adaptador Bluetooth y establecer la conexión con un dispositivo:

a) Antes que la aplicación pueda comunicarse a través del enlace Bluetooth, es necesario verificar que existe un adaptador Bluetooth en el dispositivo, y verificar que éste se encuentre activado. Si el Bluetooth es soportado pero se encuentra deshabilitado, es posible realizar una petición al dispositivo para activarlo sin necesidad de salir de la aplicación a través de una nueva actividad flotante.

b) Una vez que el dispositivo Bluetooth está activado, obtener la lista de dispositivos que ya han sido sincronizados anteriormente. Si el dispositivo no se encuentra en esta lista, iniciar el modo escaneo y sincronizar el dispositivo.

c) Obtener un *BluetoothSocket* con el UUID 00001101-0000-1000-8000-00805F9B34FB. Mediante este UUID se estable un canal RFCOMM con el módulo Bluetooth. Es importante realizar esta etapa en un hilo adicional, ya que el método para obtener el socket es un método de bloqueo (finaliza cuando se regresa un socket conectado u ocurre una excepción) y si se realiza en el hilo principal se bloqueará cualquier posible interacción con la aplicación. Una vez conectado el socket es posible finalizar la ejecución de este hilo.

d) Obtener un flujo de entrada (*InputStream*) y un flujo de salida (*OutputStream*) a partir del socket conectado. Efectuar la lectura a través del flujo de entrada

y la escritura a través del flujo de salida. La lectura también es un método de bloqueo (y la escritura en algunas ocasiones) por lo que es necesario dedicar un hilo adicional que se encargue de realizar estas tareas. Como se había comentado anteriormente, es necesario el uso de un *handler* para enviar la información hacia el hilo principal.

La Figura 4.5 muestra el diagrama de bloques del proceso para inicializar el adaptador Bluetooth y establecer la conexión con otro dispositivo.

4.2.10 Dibujar en pantalla

Clase *View*

La clase *View* representa el bloque básico para construir interfaces de usuario. Un *View* ocupa un espacio rectangular en pantalla y es responsable de dibujar y manejar eventos.

Para implementar una vista personalizada solamente se necesita sobrescribir el método `onDraw(Canvas)`, y colocar la vista en pantalla en el método *callback* `onCreate(Bundle)` con el método `setContentView(View).`

Clase *Canvas*

El método `onDraw()` requiere un objeto de la clase *Canvas.* Esta clase contiene las llamadas para dibujar. Se requieren 4 componentes básicos para dibujar algo: un *Bitmap* para dibujar los pixeles, un *Canvas* para llamar a los métodos, un dibujo primitivo (puntos, líneas, rectángulos, círculos, texto, etc) y un *paint* (que describe el color y el estilo del dibujo).

Dibujando con un *Canvas* en un *View*

Si la aplicación no requiere una cantidad significativa de procesamiento o demasiados cuadros por segundo, es buena idea crear un *View* personalizado y dibujar con un *Canvas* en `View.onDraw()`. Al utilizar el método *callback* `onDraw()`, el *Canvas* es automáticamente proporcionado y solamente es necesario colocar las llamadas para dibujar en él.

Para comenzar se debe extender de la clase *View* y definir el método `onDraw()`. Este método será llamado por el marco de aplicación de Android para pedirle a la vista que se dibuje a sí misma. Aquí deberán hacerse todas las llamadas para dibujar a través del *Canvas*, que es pasado como parámetro al método.

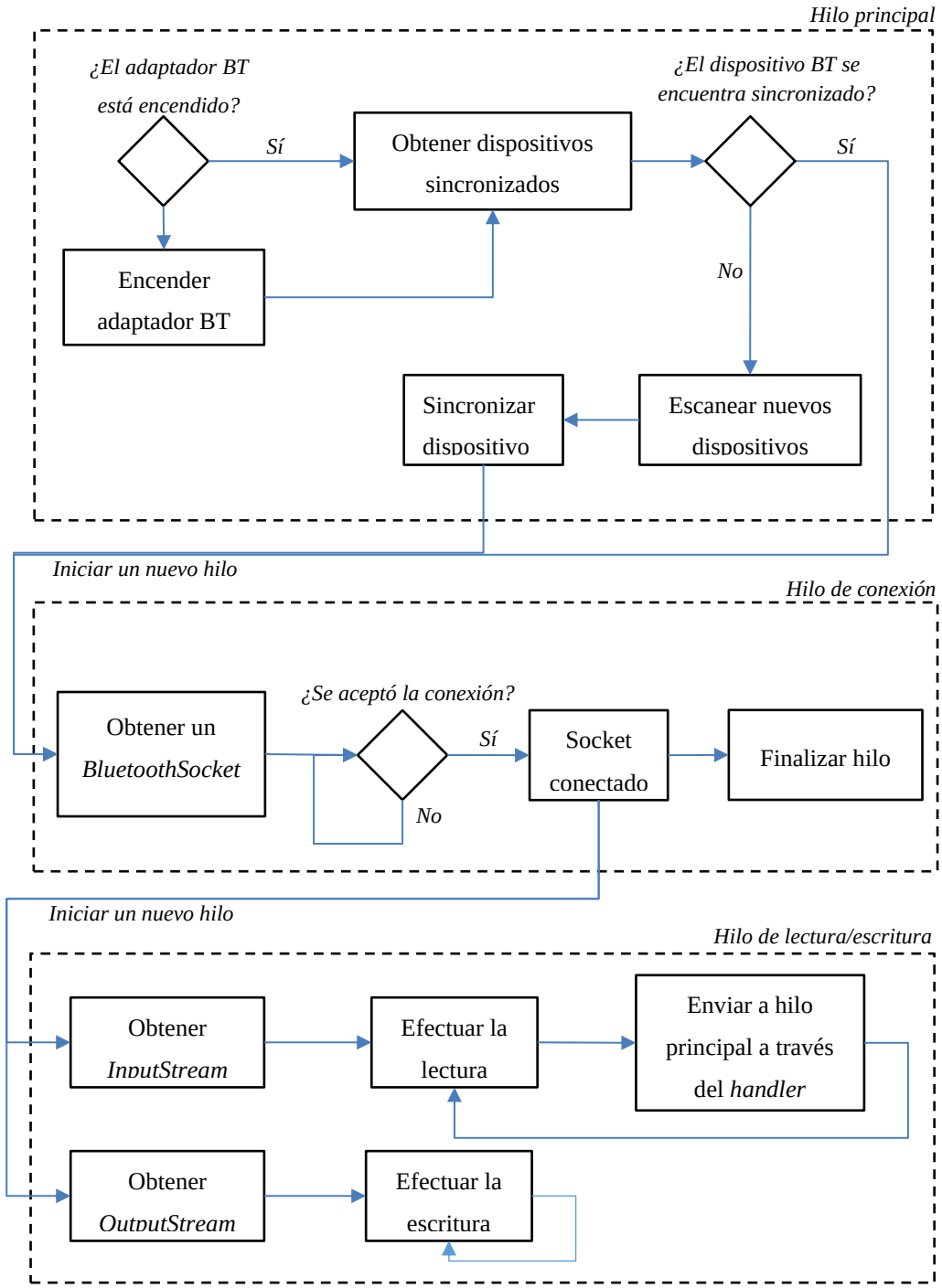

Figura 4.5 Diagrama de bloques del proceso de comunicación Bluetooth en Android

El marco de aplicación sólo llamará a `onDraw()` cuando sea necesario. Cada vez que la aplicación esté lista para ser dibujada, se debe pedir que la vista sea invalidada llamando

al método `invalidate()`. Esto le indica al sistema que la vista debe dibujarse otra vez haciendo una llamada forzosa al método `onDraw()`.

Es importante mencionar que el origen de las coordenadas de la pantalla se ubica en la parte superior izquierda (y no en la parte inferior izquierda como normalmente se trata en sistemas coordenados) y el máximo se ubica en la parte inferior derecha. En dispositivo con resolución de 320 x 480, dibujar una línea recta desde el origen (0, 0) hasta el máximo (319, 479) se hace mediante el método `drawLine(0, 0, 319, 479, myPaint)`, el resultado se muestra en la Figura 4.6.

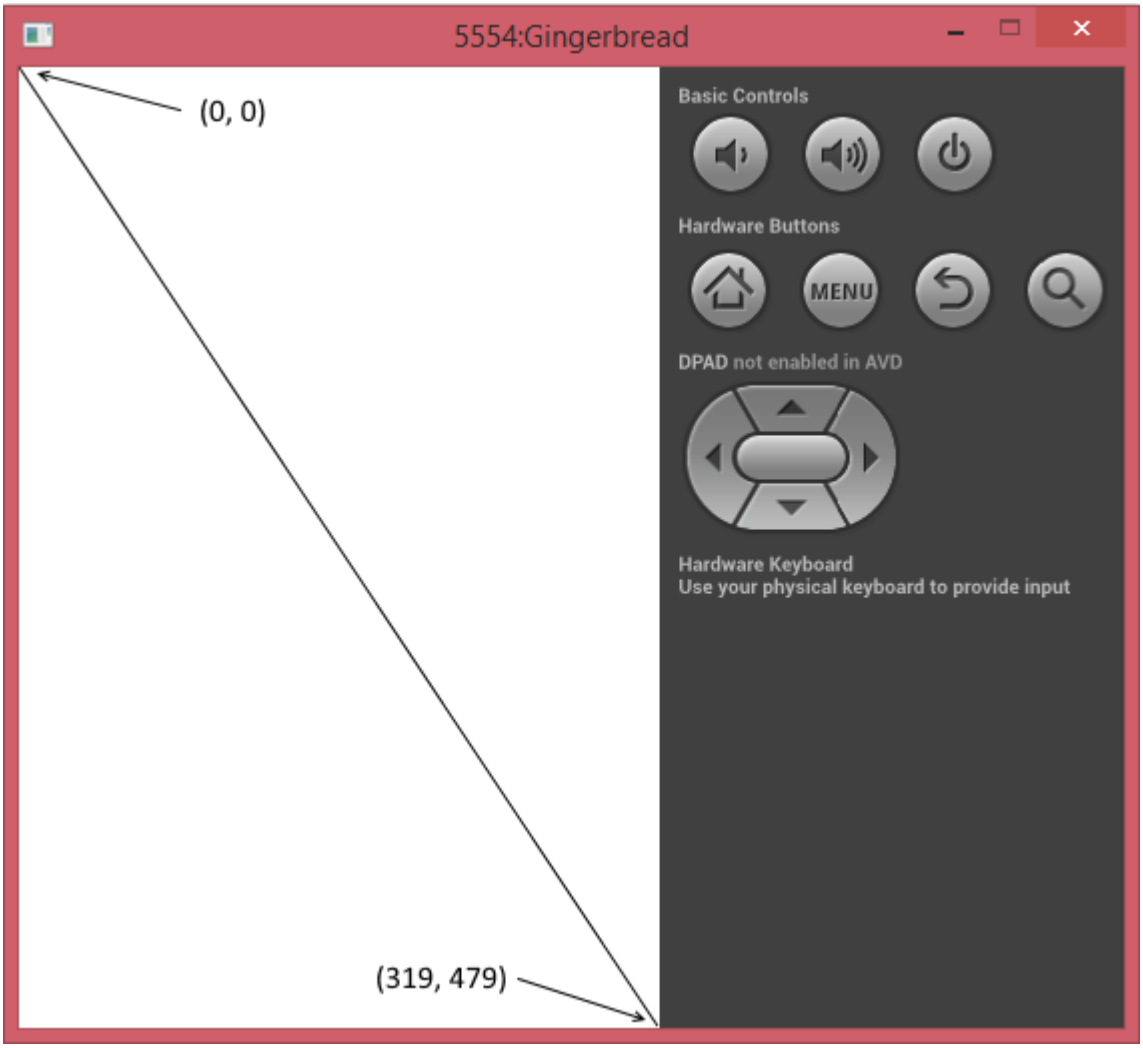

Figura 4.6 Línea recta diagonal dibujada del origen hacia el otro extremo de la pantalla

4.2.11 Gestionar la interacción con el usuario

Como se mencionó anteriormente, la clase *View* se encarga de manejar los eventos del usuario mediante los siguientes métodos:

- `onKeyDown(int, KeyEvent)`
- `onKeyUp(int, KeyEvent)`
- `onTrackballEvent(MotionEvent)`
- `onTouchEvent(MotionEvent)`

Debido a que en la aplicación de osciloscopio sólo es necesario manejar los eventos en pantalla, se requiere implementar únicamente el último método: `onTouchEvent(MotionEvent)`. Este es un método *callback* que es llamado automáticamente cuando se detecta un *MotionEvent* en pantalla.

MotionEvent

Objeto que es utilizado para reportar eventos de movimiento (mouse, lápiz, dedo, *trackball*). Estos eventos pueden contener movimientos absolutos o relativos y otro tipo de información, dependiendo del evento.

Los eventos *Motion* describen movimiento en términos de un código de acción y un conjunto de coordenadas. Esta acción indica el cambio de estado que ha ocurrido, como tocar o dejar de tocar la pantalla. Las coordenadas describen la posición y otras propiedades del movimiento.

Por ejemplo, cuando el usuario toca por primera vez la pantalla, el sistema entrega un evento de toque al *View* apropiado con el código de acción `ACTION_DOWN` y un conjunto de valores que incluyen las coordenadas X y Y, la presión, tamaño y orientación del área de contacto.

Existen tres códigos de acción fundamentales:

- `ACTION_DOWN`: el usuario toca por primera vez la pantalla.
- `ACTION_MOVE`: el usuario desplaza el puntero por la pantalla.
- `ACTION_UP`: el usuario levanta el puntero (deja de tocar) de la pantalla.

La clase *Motion* posee varios métodos para determinar la posición del puntero en la pantalla, entre la cuales están:

- `getX()`: regresa la ubicación del puntero en el eje X (eje horizontal).

- `getY()`:regresa la ubicación del puntero en el eje Y (eje vertical).

Por ejemplo, se coloca un botón personalizado con la esquina superior izquierda en (100, 300) y la esquina inferior derecha en (120, 340), para determinar si el usuario lo ha pulsado es necesario realizar el siguiente procedimiento:

a) Verificar que el código de acción sea `ACTION_UP` (para ejecutar la acción una vez que el usuario deje de presionar el botón).
b) Verificar que la posición del puntero esté en 100 < X < 120 y 300 < Y < 340.
c) Ejecutar la función del botón.

4.2.12 Procesamiento de los datos recibidos

Los datos son recibidos en el hilo secundario dedicado al Bluetooth, el cual siempre está pendiente de la llegada de datos. Una vez que éstos han sido recibidos, son enviados al hilo principal mediante el *Handler*. Si no se requiere una cantidad significativa de procesamiento, es posible utilizar el hilo UI para realizar el procesamiento. En el caso del osciloscopio, se recibe un paquete de 400 muestras máximo cada 0.5 segundos (conversión, procesamiento y envío en el dsPIC) lo que requiere procesar 2 cuadros por segundo, por lo que se optó por procesar la información en el hilo principal.

Las tres etapas del procesamiento de los datos son: ajuste de coordenadas, escalamiento a tamaño de pantalla y reordenamiento del vector. La Figura 4.7 muestra el diagrama de esta etapa.

Ajuste de coordenadas

En Android, el eje coordenado Y comienza en la parte superior de la pantalla y no en la parte inferior como normalmente se trabaja. Para graficar, primero se tiene ajusta cada uno de los datos recibidos de acuerdo con la Ecuación (4.4).

$$Dato^* = (Resolución_{ADC} - 1) - Dato \tag{4.4}$$

Por ejemplo, para una conversión con valor de *75* en el dsPIC, en Android el valor correspondiente es *180*.

$$Dato^* = (2^8 - 1) - 75 = 180$$

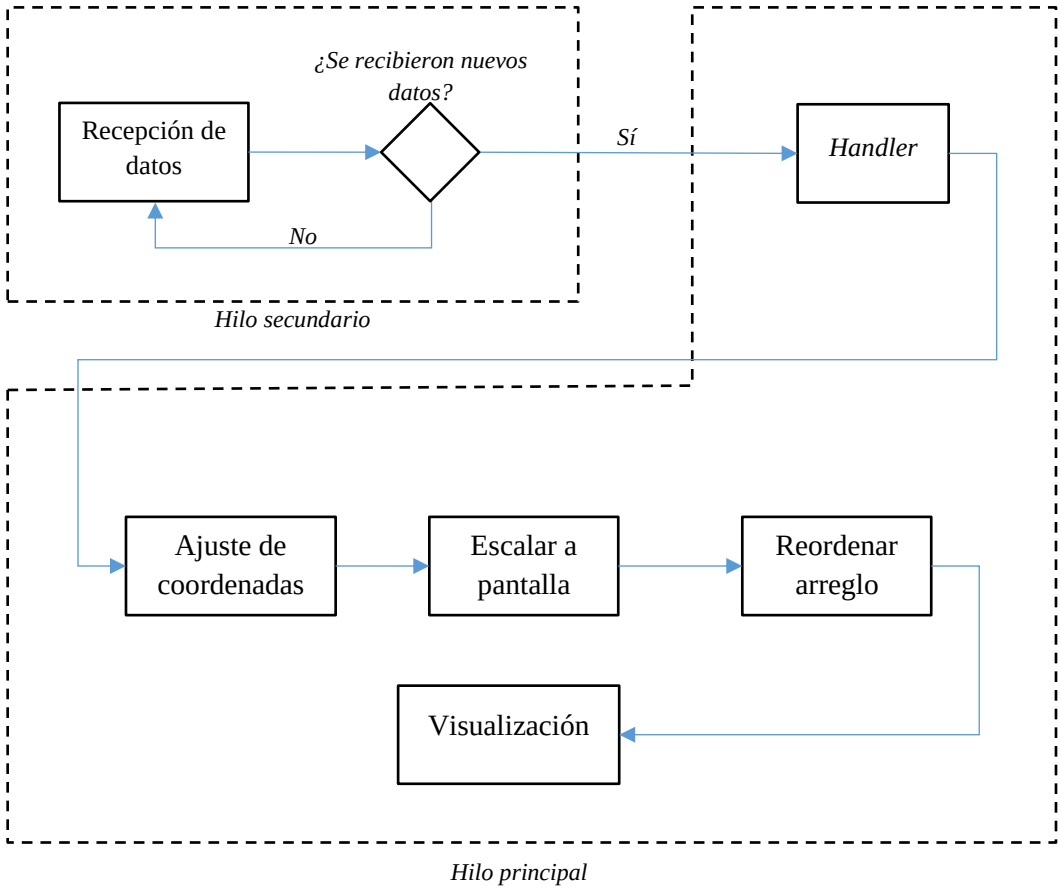

Figura 4.7 Diagrama de bloques de la etapa de procesamiento de datos recibidos en Android

Escalamiento a alto de pantalla

Existe una gran variedad de tamaños y densidades de pantalla en dispositivos con Android, por lo que la gráfica se debe ajustar al dispositivo. Entre las opciones para obtener las dimensiones de la pantalla, la clase *View* provee el método *callback* `onSizeChanged(int w, int h, int oldw, int oldh)` que regresa los valores de alto y ancho del *View* (y por tanto del *Bitmap* en el dibuja el *Canvas*). En la Ecuación (4.5) se muestra cómo escalar los datos al ancho y alto de la pantalla.

$$Dato^{**} = Dato^{*} * (Alto - 1)/(Resolución_{ADC} - 1) \qquad (4.5)$$

Por ejemplo, para una conversión con valor de *180* en Android (*75* en dsPIC) y una pantalla de 480 x 320, el valor correspondiente a graficar es *225*.

$$Dato^{**} = 180 * (320 - 1)/(2^8 - 1) = 225$$

Escalamiento a ancho de pantalla

Los valores del eje X son constantes durante todo el ciclo de la actividad, por lo que es necesario calcularlos una sola vez.

La Ecuación (4.6) muestra la forma de crear un vector, con los datos fijos de X en las posiciones pares, escalado al ancho de pantalla del dispositivo.

$$pts[i * 2] = i * ancho/pts.lenght \quad (4.6)$$

Reordenar el arreglo para dibujarlo

Es necesario unir los datos recibidos mediante líneas. El método `drawLines(float[] pts, int offset, int count, Paint paint)` dibuja una serie de líneas, cada línea es tomada mediante 4 valores consecutivos del arreglo `pts` de la siguiente manera: línea de pts[0], pts[1] a pts[2], pts[3]; línea de pts[4], pts[5] a pts[6], pts[7]; etc.

Los valores enviados por el dsPIC únicamente contienen la amplitud de la señal, es decir, los valores del eje Y. Debido a que los valores del eje X permanecen constantes, como se mencionó anteriormente, sólo es necesario modificar el arreglo en las posiciones impares en cada nuevo paquete de datos. De tal manera que el arreglo tenga la siguiente estructura: $pts[] = [x_0, y_0, x_1, y_1, x_2, y_2 \dots x_{n-1}, y_{n-1}, x_n, y_n]$.

4.2.13 Visualización de la información

Una vez finalizado el procesamiento de la señal, es necesaria mostrarla al usuario de la siguiente manera:

a) Establecer el color del fondo mediante el método `drawColor(int color)`. Este método sobrescribe los dibujos anteriores, y es útil para limpiar la pantalla.
b) Dibujar la cuadrícula del osciloscopio mediante el método `drawLine(float startX, float startY, float stopX, float stopY, Paint paint)`.

c) Mostrar la información del canal 1 (si se encuentra activo) mediante el método `drawLines(float[] pts, int ofsset, int count, Paint paint)`.
d) Mostrar la información del canal 2 (si se encuentra activo) mediante el método `drawLines(float[] pts, int ofsset, int count, Paint paint)`.
e) Dibujar la línea de disparo (si se encuentra activo) mediante el método `drawLine(float startX, float startY, float stopX, float stopY, Paint paint)`.
f) Dibujar la información adicional (como time/div y volt/div) utilizando el método `drawText(String text, float x, float y, Paint paint)`.

Este procedimiento se repite durante toda la vida de la actividad.

En la Figura 4.8 se muestra el ejemplo de este proceso para una pantalla de 3 *x* 10 pixeles y los siguientes datos recibidos [16, 21, 23].

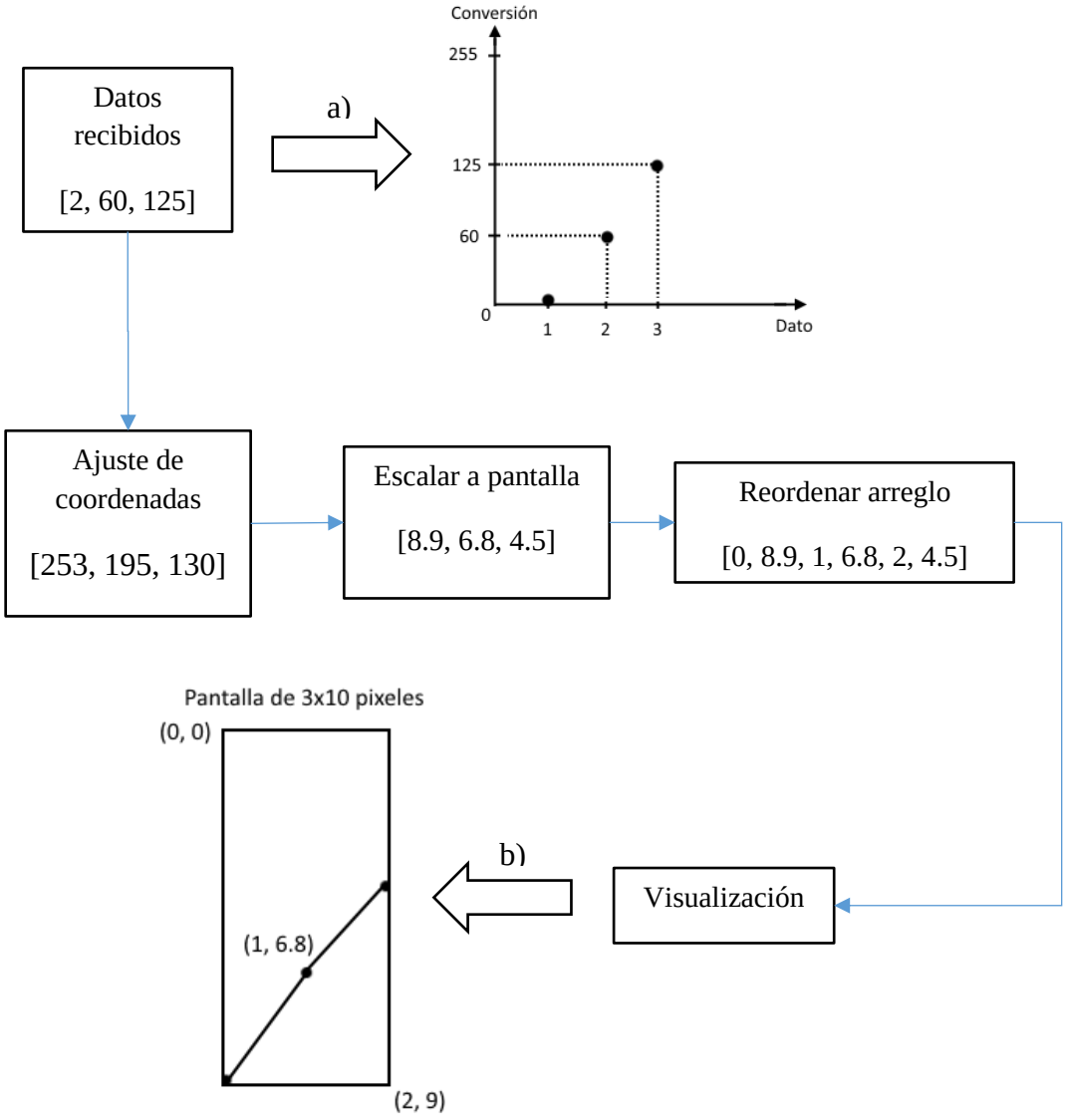

Figura 4.8 Etapa de procesamiento para una pantalla de 3 *x* 10 pixeles y un arreglo de 3 datos recibidos. a) Gráfica en un plano cartesiano ordinario de los datos recibidos. b) Gráfica de los datos recibidos en el plano cartesiano utilizado en Android.

En resumen, en este capítulo se implementó el software del equipo: iniciliazación del dsPIC, mustreo de la señal, establecimiento de la comunicación Bluetooth entre el dsPIC y el dispositivo Android, actividades e hilos en Android y, finalmente, visualización de la información.

A continuación, en el Capítulo 5, se presentan los resultados y las pruebas efectuadas al prototipo.

Capítulo 5

Resultados

Utilizando una etapa de acondicionamiento, el convertidor analógico-digital del dsPIC16FJ16GS502 y un dispositivo Android para la visualización de la señal se logró construir un osciloscopio digital con las siguientes características:

Hardware

- 2 canales de entrada.
- Resolución vertical de 8 bits.
- Velocidad de muestreo: 2 Msps.
- Voltaje máximo de entrada de ±10 volts.
- 3 niveles analógicos de ganancia.
- Voltaje de alimentación: 3.3 V.

Software

- 2 canales individuales.
- Pre-disparo por nivel en flanco de subida/bajada canal 1/canal 2.
- Desplazamiento horizontal y vertical de la señal.
- Medición de amplitud pico a pico y frecuencia.
- Visualización de espectro en frecuencia utilizando un algoritmo FFT.
- 15 niveles seleccionables de tiempo/div.

El hardware se implementó en protoboard utilizando circuitos integrados con encapsulado DIP (*Dual in-line package*) como se muestra en la Figura 5.1.

Se utilizó un gabinete de plástico de 19.4 cm x 7.4 cm x 11.4 cm para contener el hardware del osciloscopio, la fuente de alimentación y un generador digital de señales (proyecto adicional) como se observa en la Figura 5.2. La fuente de alimentación está conformada por 4 pilas AA de 1.2 V en serie (4.8 V en total).

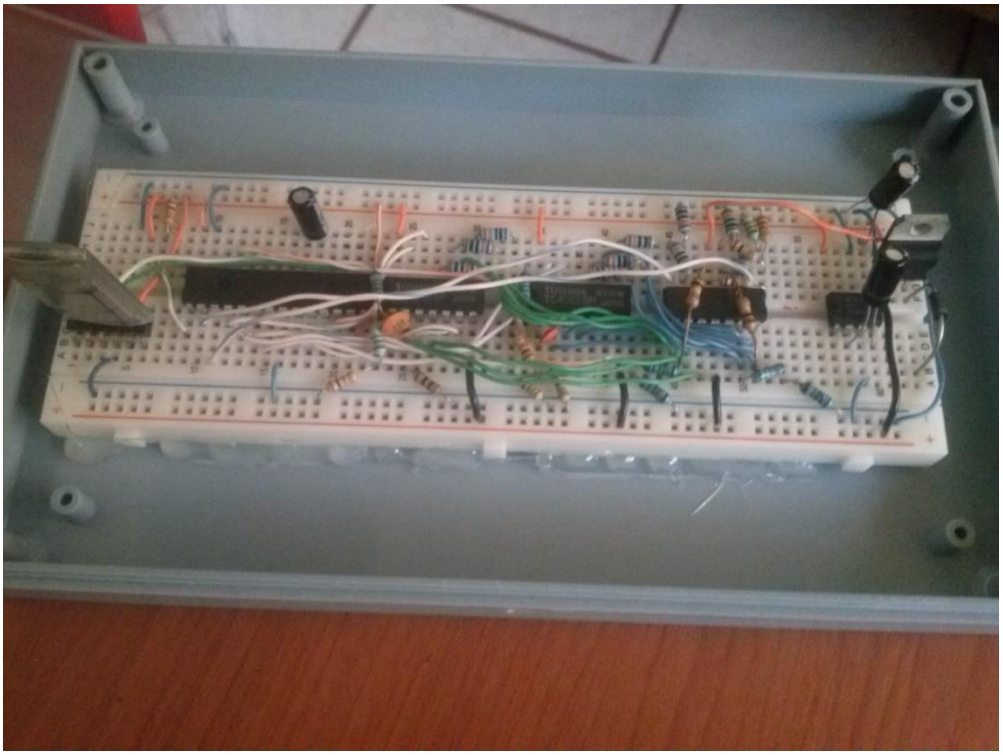

Figura 5.1 Hardware del osciloscopio montado en protoboard

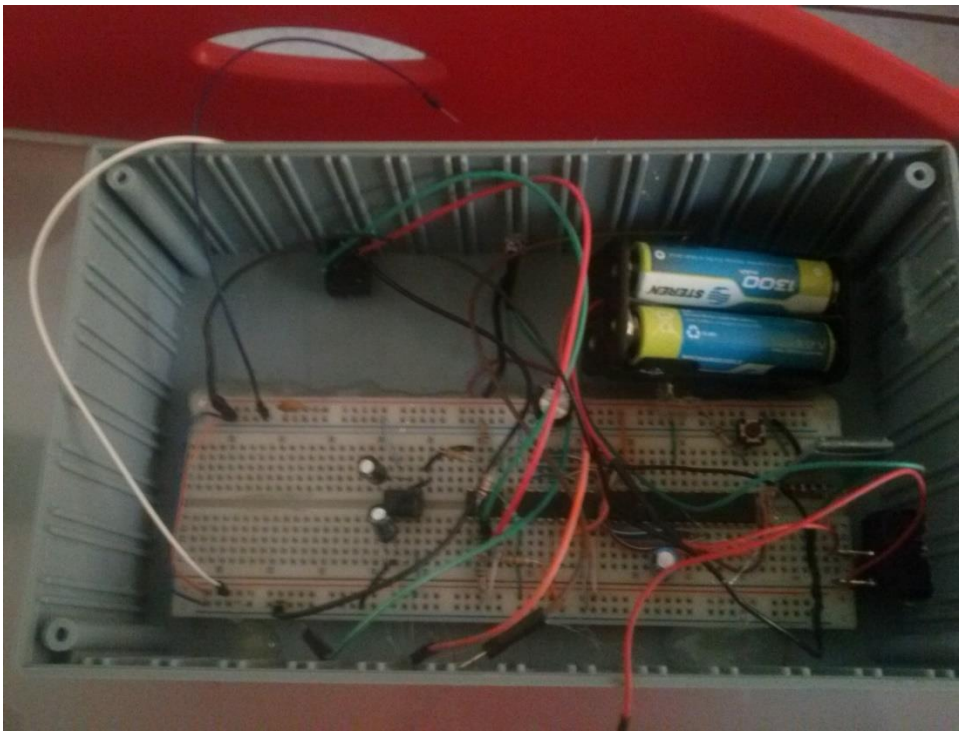

Figura 5.2 Fuente de alimentación con baterías y hardware de un generador digital de señales (proyecto adicional)

La Figura 5.3 muestra la vista frontal del gabinete, el cual cuenta con 2 *jacks* de 3.5 mm y un led indicador de encendido.

Figura 5.3 Vista externa del osciloscopio con entrada de 3.5 mm

La punta de prueba del osciloscopio está compuesta como por un *plug* de 3.5 mm en un extremo y tres caimanes en el otro extremo. El caimán negro corresponde a la referencia, el caimán rojo al canal 1 y el caimán verde al canal 2. La Figura 5.4 muestra la punta de prueba conectada a la entrada del gabinete.

En la parte trasera del gabinete se encuentra un interruptor de encendido/apagado. En la Figura 5.5 se observa el dispositivo encendido.

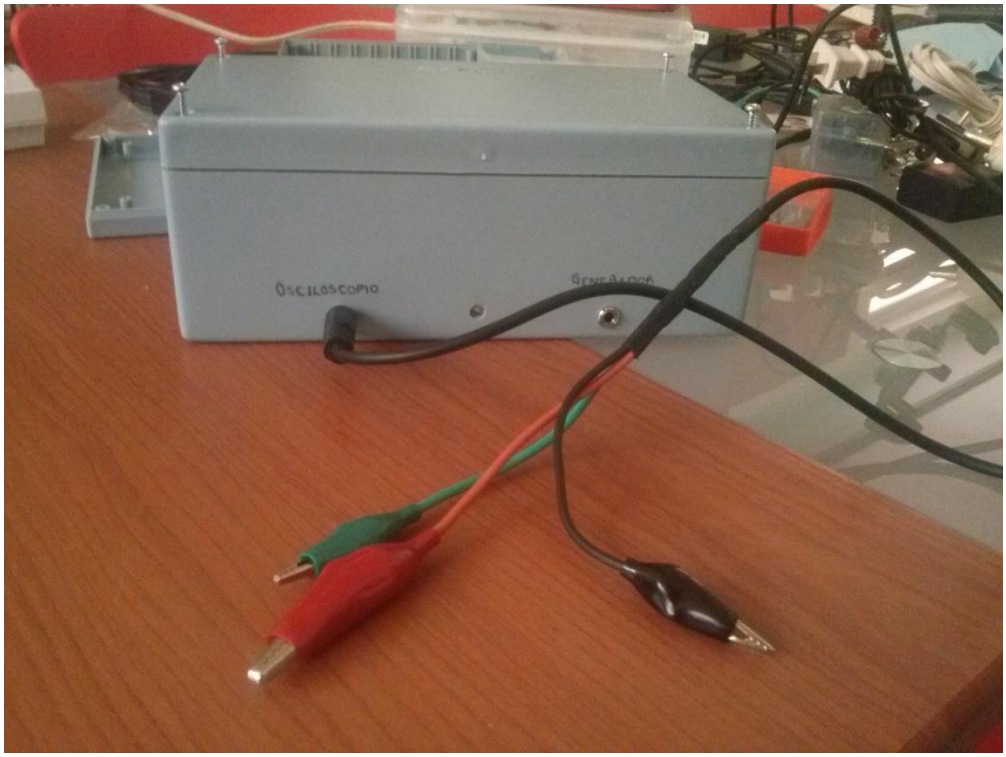

Figura 5.4 Puntas de prueba conectadas al osciloscopio

Figura 5.5 Osciloscopio encendido

Para comprobar el funcionamiento del osciloscopio se midieron 2 señales: una sinusoidal pura con componente de CD generada por un generador comercial y otra sinusoidal con componente de segundo armónico generada por el generador digital de señales integrado en el gabinete. La Figura 5.6 muestra ambas señales visualizadas en el dispositivo Android con sus respectivos valores de voltaje y frecuencia.

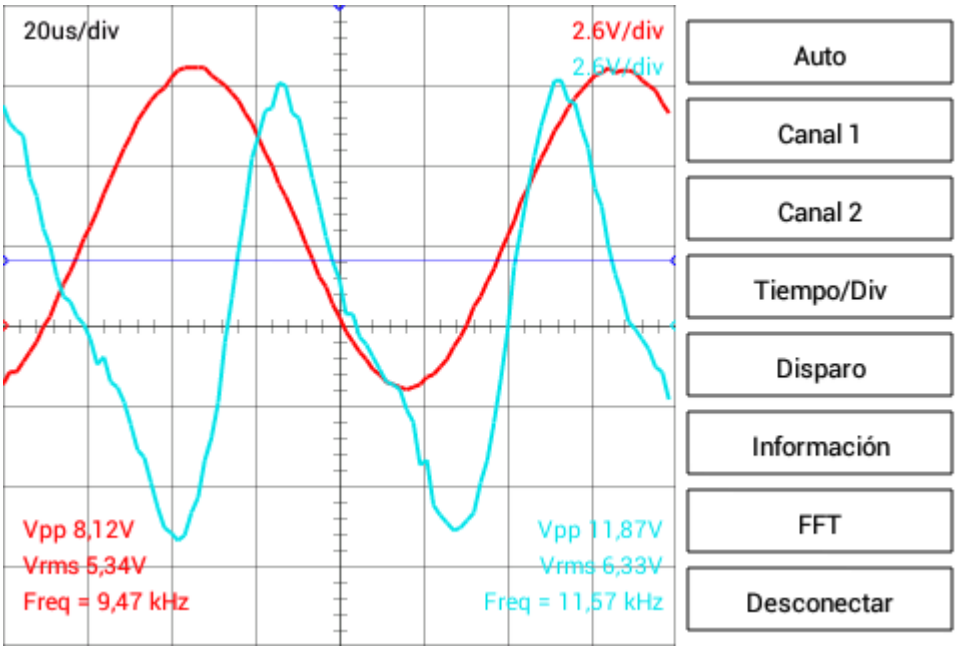

Figura 5.6 Interfaz de usuario del osciloscopio en Android con dos señales: una sinusoidal pura (en rojo) y otra sinusoidal con segundo armónico (en azul), y sus respectivos valores de voltaje y frecuencia.

Utilizando un algoritmo FFT (mostrado en Anexo B.1) para procesar señales de audio en tiempo real se calcula el espectro en frecuencia de las señales de entrada y se muestra al usuario. La Figura 5.7 muestra el espectro en frecuencia de la señal sinusoidal pura con componente de CD, mientras que la Figura 5.8 muestra el espectro de la señal sinusoidal con componente de segundo armónico. De ambas figuras se puede observar que el espectro en frecuencia de cada señal coincide con los parámetros de entrada.

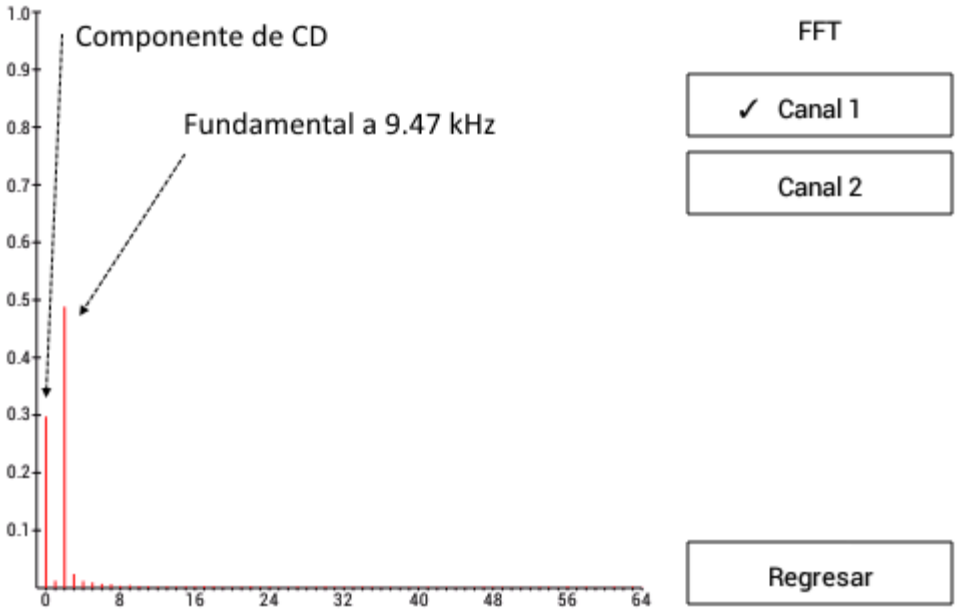

Figura 5.7 Espectro en frecuencia de una señal sinusoidal con componente de CD

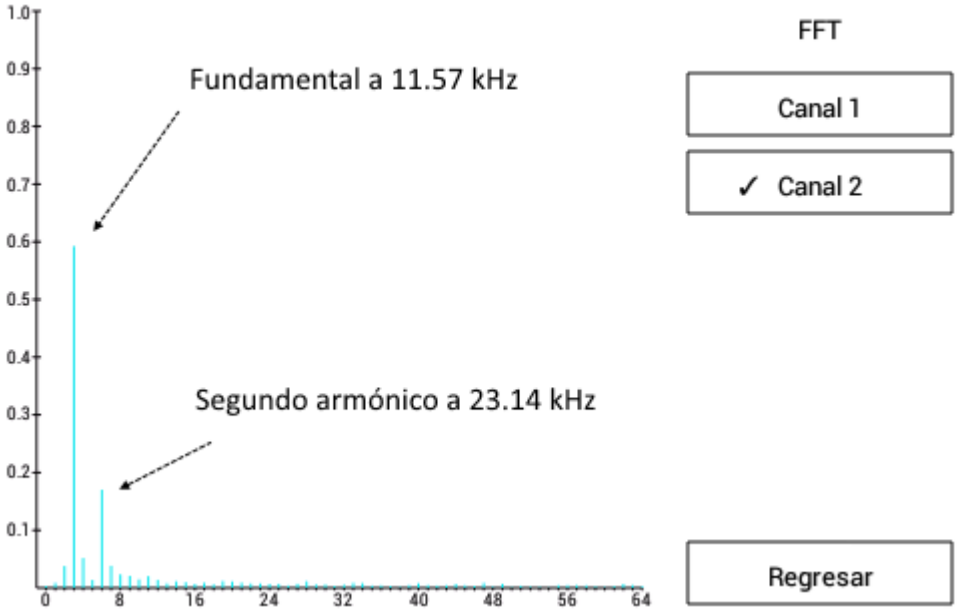

Figura 5.8 Espectro en frecuencia de una señal sinusoidal con segundo armónico

Después de las señales sinusoidales se probó con una señal diente de sierra y una cuadrada, ambas con componente de CD. La Figura 5.9 muestra la visualización de ambas señales en la pantalla del dispositivo Android.

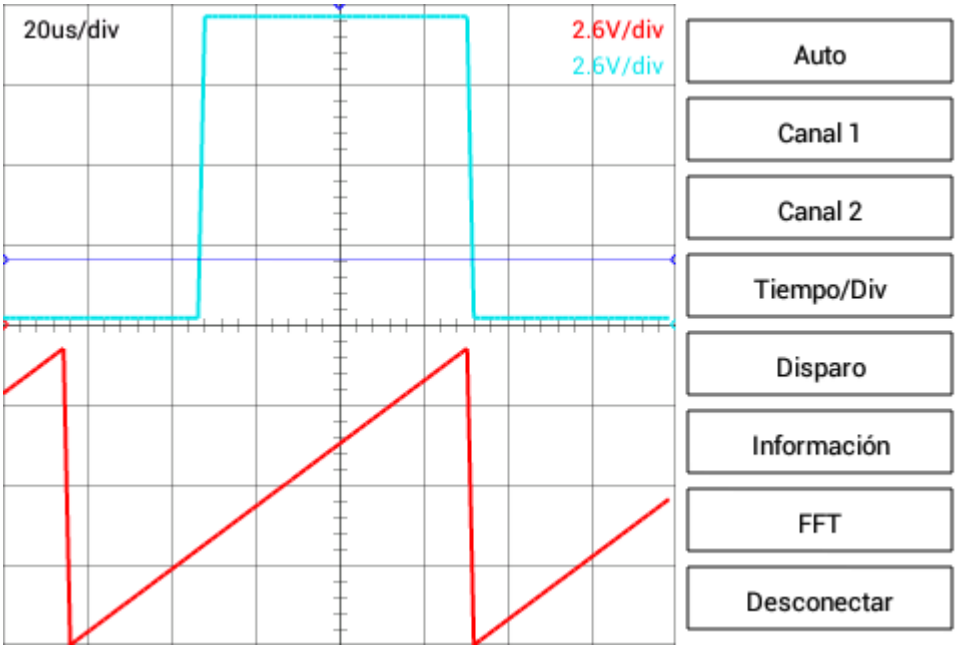

Figura 5.9 Interfaz de usuario del osciloscopio en Android con una señal cuadrada y una señal diente de sierra

De igual manera que con las señales sinusoidales, se obtuvo el espectro en frecuencia de la señal diente de sierra y cuadrada. La Figura 5.10 muestra el espectro de la señal diente sierra y la Figura 5.11 muestra el espectro de la señal cuadrada.

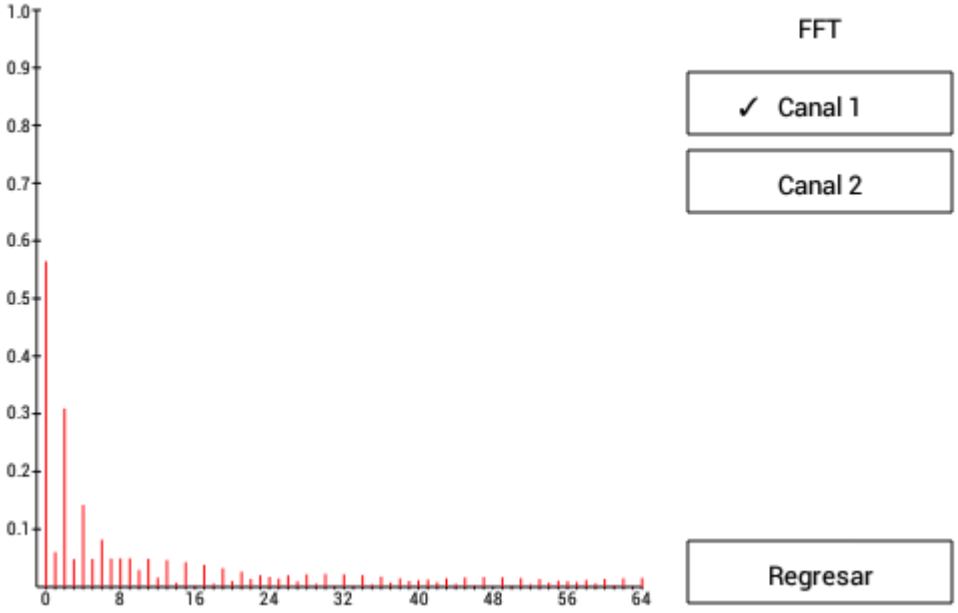

Figura 5.10 Espectro en frecuencia de una señal diente de sierra con componente de CD

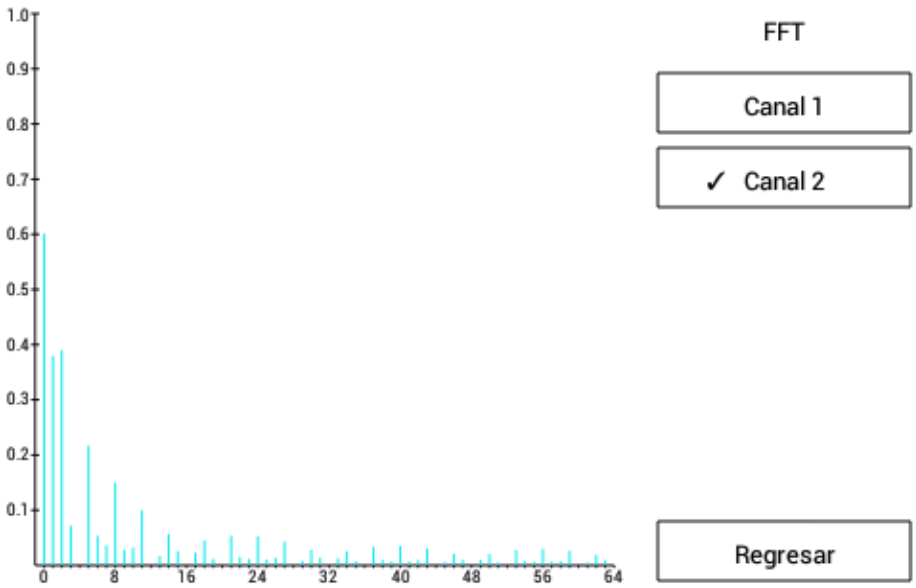

Figura 5.11 Espectro en frecuencia de una señal cuadrada con componente de CD

Para corroborar el correcto funcionamiento del algoritmo FFT se utilizó el software matemático MATLAB® para recrear una señal diente de sierra y obtener su espectro en frecuencia, lo cual se muestra en la Figura 5.12.

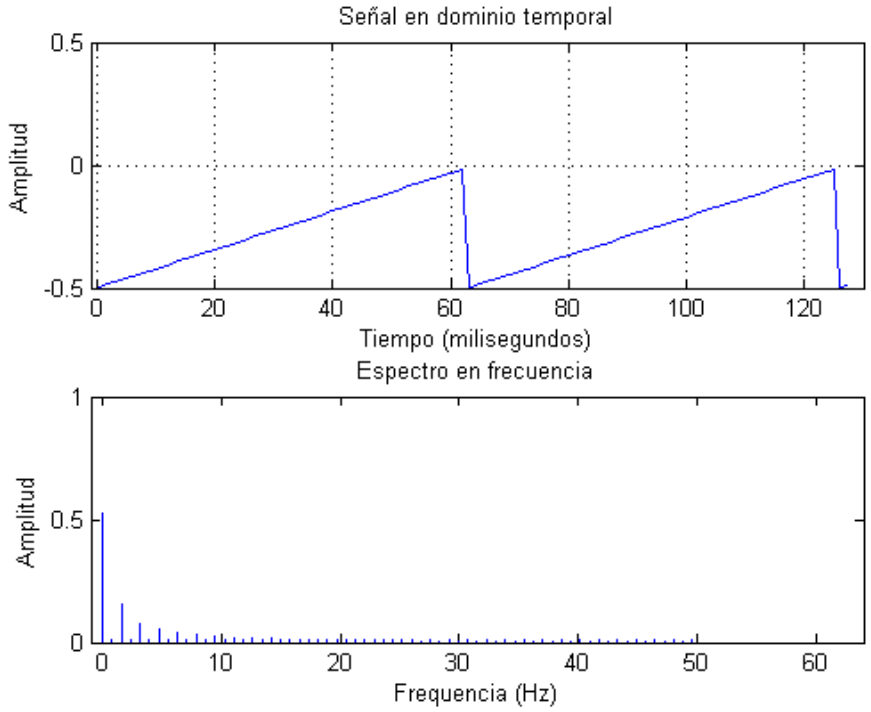

Figura 5.12 Señal diente de sierra y su espectro en frecuencia en MATLAB

Al comparar el espectro calculado en MATLAB y el espectro calculado en el osciloscopio se observa que ambos son similares, a excepción de que se muestran algunos armónicos adicionales en este último debido a que el procesamiento es en tiempo real ya que señal a la entrada del osciloscopio está variando constantemente.

En resumen, en este capítulo se observó que el funcionamiento del osciloscopio implementado en este trabajo resultó satisfactorio. Este equipo permite observar y calcular el espectro de cualquier tipo de señal eléctrica de hasta 50 kHz y ±10 V de amplitud, lo cual es suficiente para señales de audio en tiempo real.

A continuación, en el Capítulo 6, se exponen las conclusiones y los trabajos futuros.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusión general

El osciloscopio digital es un instrumento versátil que se utiliza para visualizar una señal variante en el tiempo, medir sus características e incluso mostrar el espectro en frecuencia. Es utilizado ampliamente en electrónica para análisis y diseño; aunque también se emplea en áreas como medicina, ingeniería automotriz, ingeniería eléctrica, telecomunicaciones, entre otras. Estos osciloscopios están desplazando en gran medida a su contraparte analógica, debido a que presenta las siguientes ventajas:

- Tamaño más compacto: están compuestos en gran medida por diminutos circuitos integrados.
- Procesamiento digital: permite realizar fácilmente tareas que son complicadas de manera analógica, incluso imposibles.
- Memoria digital: almacenamiento de información temporal o permanentemente.
- LCD: gran cantidad de colores para distinguir los trazos.

Existe una gran variedad de osciloscopios digitales disponibles en la actualidad, los cuales están enfocados a distintos mercados: los hay en precios reducidos, con gran movilidad debido a su pequeño tamaño y presentan las características básicas; precios intermedios, con tamaños accesibles para desplazarlos eventualmente y características añadidas como un ancho de banda superior; y finalmente existen equipos extremadamente caros, de gran tamaño y poca movilidad, pero con características muy superiores y de gran precisión y exactitud.

Si bien existen osciloscopios basados en dispositivos móviles, todos se conectan mediante el puerto USB. Esto supone que existe una porción libre del mercado para comercializar un osciloscopio con comunicación inalámbrica.

6.2 Conclusión personal

El desarrollo de este proyecto presentó un gran reto debido a que éste integra distintos temas de diversas materias. Además de aplicar conocimientos adquiridos en la Facultad, también fue necesario aprender de temas nuevos.

Antes de comenzar el proyecto fue necesario tomar clases de desarrollo de aplicaciones en Android, donde se adquirió conocimiento básico para iniciar la comunicación Bluetooth y para manejar la interfaz gráfica. Además fue necesario iniciarse en la programación orientada a objetos en lenguaje Java consultando libros y manuales tanto impresos como en línea.

En la primera versión del osciloscopio se utilizó el microcontrolador PIC18F4550, del que ya se tenían conocimientos previos. Además se utilizó una etapa similar a la desarrollada en este trabajo para el acondicionamiento de la señal. El instrumento construido presentaba un excelente funcionamiento y era bastante preciso y exacto en las mediciones, pero sólo alcanzaba un ancho de banda de 6 kHz, por lo que se decidió cambiar el microcontrolador.

La siguiente versión utiliza el controlador de señales dsPIC33FJ16GS502, del que ya se habló en el desarrollo, y una etapa de acondicionamiento donde la ganancia era controlada por un par de potenciómetros digitales MCP4161 (uno por canal) que utilizan comunicación SPI. El nuevo dsPIC cumplió las expectativas de ancho de banda, alcanzando hasta 50 kHz (suficiente para señales de audio utilizadas en el laboratorio de la Facultad); y los potenciómetros digitales lograban que el circuito fuera más compacto ya que se eliminaban los interruptores analógicos. El problema surgió cuando se detectó que la resistencia de estos potenciómetros sufría variaciones de hasta ±20%, lo cual afectaba directamente tanto la precisión como la exactitud de las mediciones.

Posteriormente, sin cambiar el dsPIC30FJ16GS502, se recurrió nuevamente a los interruptores analógicos para modificar la ganancia de la etapa de acondicionamiento. Esta es la versión que se explica ampliamente en el desarrollo.

6.3 Trabajos futuros

Es del interés del autor continuar con el desarrollo del osciloscopio digital, añadiendo nuevas características y mejorando las actuales. Se plantea la posibilidad de producir estos instrumentos en masa para comercializarlos.

Utilizar un convertidor analógico-digital más rápido, ya sea reemplazando el controlador de señales o añadiendo un convertidor independiente, para lograr un ancho de banda de al menos 200 kHz.

Mejorar el aspecto de la interfaz gráfica al eliminar el área reservada para los botones, maximizando el espacio para visualizar la señal. Añadir una barra de estado en la parte inferior en la que se desplieguen valores como tiempo/división, voltaje/división, modo de disparo, entre otros. Implementar gestos con varios dedos para mejorar la experiencia del usuario, así como la función de ajuste automático de la señal. Permitir al usuario guardar el vector de muestras en la memoria del dispositivo Android, así como agregar una opción nativa de la aplicación para registrar video y capturas de pantalla.

Compatibilidad con otros dispositivos que soporten Bluetooth como iOS y PC.

Anexo A. Código fuente del dsPIC

A.1. Establecer configuración

Conjunto de macros para establecer los parámetros iniciales del dsPIC en cada reinicio.

```
// Select Internal FRC at POR
_FOSCSEL(FNOSC_FRC);
// Enable Clock Switching, Fail-Safe Clock Monitor is Disabled
// OSC2 is general purpose digital I/O pin
// Allow multiple PPS reconfigurations
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF & IOL1WAY_OFF);
// Watchdog timer off
_FWDT(FWDTEN_OFF);
```

A.2. Configurar el PLL para utilización del oscilador interno FRC

Se muestran las instrucciones necesarias para obtener la máxima velocidad de trabajo mediante el uso del PLL y del oscilador interno FRC a 7.37 MHz.

```
// Configure PLL prescaler, PLL postscaler, PLL divisor
PLLFBD = 41;                    // M = 43
CLKDIVbits.PLLPOST = 0;         // N2 = 2
CLKDIVbits.PLLPRE = 0;          // N1 = 2
// Initiate Clock Switch to Internal FRC with PLL (NOSC = 0b001)
__builtin_write_OSCCONH(0x01);
__builtin_write_OSCCONL(0x01);
// Wait for Clock switch to occur
while (OSCCONbits.COSC != 0b001);
// Wait for PLL to lock
while (OSCCONbits.LOCK != 1);
```

A.3. Configurar la generación auxiliar de reloj

Conjunto de instrucciones para activar la señal auxiliar de reloj. Ésta se utiliza como señal de reloj para el ADC.

```
// Internal FRC is clock source for auxiliary PLL
ACLKCONbits.FRCSEL = 1;
ACLKCONbits.ENAPLL = 1;         // APLL is enabled
// Auxiliary PLL provides the source clock for the clock divider
ACLKCONbits.SELACLK = 1;
// Auxiliary Clock Output Divider is Divide by 1
```

```
ACLKCONbits.APSTSCLR = 7;
while(ACLKCONbits.APLLCK != 1);    // Wait for Auxiliary PLL to Lock
```

A.4. Configurar entradas/salidas y periféricos

Se presenta el código para inicializar todos los puertos utilizados, tanto entradas como salidas.

```
// ADC
ADPCFG = 0b11111100;    // AN7-AN2 digital, AN1 & AN0 analogic
TRISAbits.TRISA0 = 1;   // RA0 (Pin 2) as input
TRISAbits.TRISA1 = 1;   // RA1 (Pin 3) as input

// Coupling
TRISAbits.TRISA2 = 0;   // Output
TRISBbits.TRISB0 = 0;   // Output
PORTAbits.RA2 = 1;      // Ch1 DC
PORTBbits.RB0 = 1;      // Ch2 DC

// Channel 1 Gain
TRISBbits.TRISB1 = 0;   // Output
TRISBbits.TRISB9 = 0;   // Output
TRISBbits.TRISB4 = 0;   // Output
PORTBbits.RB1 = 1;      // Minimun gain
PORTBbits.RB9 = 0;      // Open switch
PORTBbits.RB4 = 0;      // Open switch

// Channel 2 Gain
TRISBbits.TRISB10 = 0;  // Output
TRISAbits.TRISA4 = 0;   // Output
TRISBbits.TRISB8 = 0;   // Output
PORTBbits.RB10 = 1;     // Minimun gain
PORTAbits.RA4 = 0;      // Open switch
PORTBbits.RB8 = 0;      // Open switch

// UART
RPINR18bits.U1RXR = 15; // RP15 (Pin 15) tied to UART1 receive
RPOR2bits.RP5R = 0b011; // RP5 (Pin 16) tied to UART1 transmit
```

A.5. Inicialización del módulo de comunicación UART1

Código para inicializar el módulo de comunicación UART1 con un *baud rate* de 9600, 8 bits, sin paridad y un bit de paro.

```
// Configure UART registers
U1MODEbits.STSEL = 0;   // 1 Stop bit
U1MODEbits.PDSEL = 0;   // No Parity, 8 data bits
```

```
U1MODEbits.ABAUD = 0;   // Auto-Baud Disabled
U1MODEbits.BRGH = 0;    // Low Speed mode
U1BRG = 259;            // BaudRate = 9600
U1MODEbits.UARTEN = 1;  // Enable UART
U1STAbits.UTXEN = 1;    // Enable data transmission

// Interrupt Request settings
IFS0bits.U1RXIF = 0; // Clear RX interrupt flag
IEC0bits.U1RXIE = 1; // Receiver interrupt request enable
```

A.6. Inicialización del módulo convertidor analógico-digital (ADC)

Código para configurar y activar el ADC a máxima velocidad con disparo manual y generando una interrupción al finalizar la conversión del par de entradas.

```
// Configure ADCON registers
ADCONbits.ADON = 0;         // Disables ADC module
ADCON = 0x00;               // Operation in Idle mode
                            // ADC is clocked by the auxiliary PLL
                            // Data output as integer
                            // Interrupt after second conversion
                            // 1:1 clock divider
ADCONbits.ASYNCSAMP = 1;    // Continuos sampling

// AN1 & AN0 pair control register
ADCPC0bits.IRQEN0 = 1;          // IRQ is generated
ADCPC0bits.TRGSRC0 = 0b00001;   // Individual software trigger

// Interrupt Request settings
ADSTATbits.P0RDY = 0;           // Clear ADC pair 0 interrupt flag
IEC0bits.ADIE = 1;              // Enable ADC interrupt

ADCONbits.ADON = 1;             // Enables ADC module
```

A.7. Rutina de interrupción del ADC

Código con la rutina de interrupción generada al finalizar la conversión analógica-digital, almacena los 8 bits más significativos de la conversión en un arreglo *char*.

```
// ADC receive interrupt routine
void __attribute__((interrupt, no_auto_psv)) _ADCInterrupt(void) {
    if(j == subsampling) {              // Change sampling period
        data_Ch1[i] = ADCBUF0 >> 2;     // Save conversion Ch1 (8-bit)
        data_Ch2[i++] = ADCBUF1 >> 2;   // Save conversion Ch2 (8-bit)
        j = 0;
    else
        j++;
```

```
    ADSTATbits.P0RDY = 0;           // Clear ADC pair 0 interrupt flag
    IFS0bits.ADIF = 0;              // Clear ADC interrupt flag
    ADCPC0bits.SWTRG0 = 1;          // Start conversion of AN0 & AN1
}
```

A.8. Implementación del algoritmo de disparo

Código del algoritmo de disparo activado por nivel en canal 1 (AN0) en flanco de subida.

```
// Trigger
for (i = 100; i <= 300; i++) {
    if (data_Ch1[i] < triggerLevel && data_Ch1[i + 1] >= triggerLevel) {
        i -= 100;  // Put trigger in the middle of data to send
        break;
     }
}

if (i > 300)       // If trigger was not activated
    i = 0;         // Send the first samples
```

A.9. Transmisión de los datos mediante UART

Código utilizado para transmitir los datos mediante el módulo UART1.

```
// Send header [0xFE, 0x01]
UART1_Transmit(254);          // Send 0xFE
UART1_Transmit(1);            // Send 0x01

//Send channel 1 vector data
for (j = i; j < (200 + i); j++) {
    UART1_Transmit(data_Ch1[j]);
}

// Send channel 2 vector data
for (j = i; j < (200 + i); j++) {
    UART1_Transmit(data_Ch2[j]);
}

// Transmit function
void UART1_Transmit(unsigned char byte) {
    while (!U1STAbits.TRMT);   // Wait until last transmision is complete
    U1TXREG = byte;            // Load one byte in TRMT -> send
}
```

A.10. Rutina de interrupción por recepción de datos

Código con la rutina de interrupción generada al recibir un dato mediante el módulo UART1, se muestra cada uno de los posibles comandos utilizados para modificar las características del osciloscopio.

```
// UART1 receive interrupt routine
void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void) {
    unsigned char dataRX = 0;

    dataRX = U1RXREG; // Read byte received
    switch (dataRX) {
        case 1: // Change Time/Div
            while (!U1STAbits.URXDA); // Wait one more byte
            subsampling = 0;
            subsampling = U1RXREG << 8; // Read byte received
            while (!U1STAbits.URXDA); // Wait one more byte
            subsampling += U1RXREG;
            break;
        case 2: // Change Volt/Div
            while (!U1STAbits.URXDA); // Wait one more byte
            dataRX = U1RXREG;
            if (dataRX == 1) { // Ch1 gain
                while (!U1STAbits.URXDA); // Wait one more byte
                dataRX = U1RXREG;
                switch(dataRX){
                    case 0: // Minimun gain
                        PORTBbits.RB1 = 1;
                        PORTBbits.RB9 = 0;
                        PORTBbits.RB4 = 0;
                        break;
                    case 1:
                        PORTBbits.RB1 = 0;
                        PORTBbits.RB9 = 1;
                        PORTBbits.RB4 = 0;
                        break;
                    case 2: // Maximun gain
                        PORTBbits.RB1 = 0;
                        PORTBbits.RB9 = 0;
                        PORTBbits.RB4 = 1;
                        break;
                }
            } else { // Ch2 gain
                while (!U1STAbits.URXDA); // Wait one more byte
                dataRX = U1RXREG;
                switch(dataRX){
                    case 0: // Minimun gain
                        PORTBbits.RB10 = 1;
                        PORTAbits.RA4 = 0;
```

```
                    PORTBbits.RB8 = 0;
                    break;
                case 1:
                    PORTBbits.RB10 = 0;
                    PORTAbits.RA4 = 1;
                    PORTBbits.RB8 = 0;
                    break;
                case 2: // Maximun gain
                    PORTBbits.RB10 = 0;
                    PORTAbits.RA4 = 0;
                    PORTBbits.RB8 = 1;
                    break;
            }
        }
        break;
    case 3: // Change trigger
        while (!U1STAbits.URXDA); // Wait one more byte
        dataRX = U1RXREG;
        switch (dataRX) {
            case 0: //trigger off
                while (!U1STAbits.URXDA); // Wait one more byte
                triggerLevel = U1RXREG;
                triggerConf = 0;
                break;
            case 1: //trigger on, positive slope, channel 1
                while (!U1STAbits.URXDA); // Wait one more byte
                triggerLevel = U1RXREG;
                triggerConf = 1;
                break;
            case 2: //trigger on, negative slope, channel 1
                while (!U1STAbits.URXDA); // Wait one more byte
                triggerLevel = U1RXREG;
                triggerConf = 2;
                break;
            case 3: //trigger on, positive slope, channel 2
                while (!U1STAbits.URXDA); // Wait one more byte
                triggerLevel = U1RXREG;
                triggerConf = 3;
                break;
            case 4:   //trigger on, negative slope, channel 2
                while (!U1STAbits.URXDA); // Wait one more byte
                triggerLevel = U1RXREG;
                triggerConf = 4;
                break;
        }
        break;
    case 4: // Change coupling
        while (!U1STAbits.URXDA); // Wait one more byte
        dataRX = U1RXREG;
        if (dataRX == 1) { // Ch1 coupling
```

```
                while (!U1STAbits.URXDA); // Wait one more byte
                PORTAbits.RA2 = U1RXREG; // 0 = AC, 1 = DC
            } else { // Ch2 coupling
                while (!U1STAbits.URXDA); // Wait one more byte
                PORTBbits.RB0 = U1RXREG; // 0 = AC, 1 = DC
            }
            break;
        default:
            break;
    }

    IFS0bits.U1RXIF = 0; // Clear RX interrupt flag
}
```

Anexo B. Código fuente Android

B.1. Algoritmo FFT

El código del algoritmo FFT utilizado para calcular el espectro en frecuencia es lo suficientemente rápido para procesar señales de audio en tiempo real. Éste es un fragmento tomado de https://www.ee.columbia.edu/~ronw/code/MEAPsoft/doc/html/FFT_8java-source.html perteneciente a la Universidad de Columbia tomado con licencia *GNU General Public License* v2.0 (GPLv2).

```
public class FFT {

    int n, m;

    // Lookup tables. Only need to recompute when size of FFT changes.
    double[] cos;
    double[] sin;

    public FFT(int n) {
        this.n = n;
        this.m = (int) (Math.log(n) / Math.log(2));

        // Make sure n is a power of 2
        if (n != (1 << m))
            throw new RuntimeException("FFT length must be power of 2");

        // precompute tables
        cos = new double[n / 2];
        sin = new double[n / 2];

        for (int i = 0; i < n / 2; i++) {
            cos[i] = Math.cos(-2 * Math.PI * i / n);
            sin[i] = Math.sin(-2 * Math.PI * i / n);
        }

    }

    public void fft(double[] x, double[] y) {
        int i, j, k, n1, n2, a;
        double c, s, t1, t2;

        // Bit-reverse
        j = 0;
```

```
        n2 = n / 2;
        for (i = 1; i < n - 1; i++) {
            n1 = n2;
            while (j >= n1) {
                j = j - n1;
                n1 = n1 / 2;
            }
            j = j + n1;

            if (i < j) {
                t1 = x[i];
                x[i] = x[j];
                x[j] = t1;
                t1 = y[i];
                y[i] = y[j];
                y[j] = t1;
            }
        }

        // FFT
        n1 = 0;
        n2 = 1;

        for (i = 0; i < m; i++) {
            n1 = n2;
            n2 = n2 + n2;
            a = 0;

            for (j = 0; j < n1; j++) {
                c = cos[a];
                s = sin[a];
                a += 1 << (m - i - 1);

                for (k = j; k < n; k = k + n2) {
                    t1 = c * x[k + n1] - s * y[k + n1];
                    t2 = s * x[k + n1] + c * y[k + n1];
                    x[k + n1] = x[k] - t1;
                    y[k + n1] = y[k] - t2;
                    x[k] = x[k] + t1;
                    y[k] = y[k] + t2;
                }
            }
        }
    }
}
```

Bibliografía

[Darcey *et al.* 2011]

L. Darcey y S. Conder, *Sams teach yourself Android application development in 24 hours*, USA: Sams, 2011, 2da edición.

[Tomás 2013]

J. Tomás, *El gran libro de Android*, México: Alfaomega, 2013, 3ra edición.

[Amaro 2012]

J. E. Amaro, *El gran libro de programación avanzada con Android*, México: Alfaomega, 2012, 1ra edición.

[Zechner *et al.* 2012]

M. Zechner y R. Green, *Beginning Android games*, USA: Apress, 2012, 2da edición.

[García de Jalón *et al.* 2000]

J. García de Jalón *et al.*, *Aprenda Java como si estuviera en primero*, España: TECNUN.

[Sánchez 2004]

J. Sánchez, *Java2*, Documento electrónico.

[Microchip 2009]

Microchip, *dsPIC33FJ06GS101/X02 and dsPIC33FJ16GSX02/X04 Data Sheet*, USA, 2009.

[Microchip 2012]

Microchip, *dsPIC33F/PIC24H Family Reference Manual*, USA, 2012.

[Microchip 2013]

Microchip, *MPLAB XC16 C Compiler User's Guide*, USA, 2013.

[Di Cerbo 2010]

M. Di Cerbo y A. Rudolf, *Using Android in Industrial Automation*, Suiza: FHNW/IA, 2010.

[ProjectProto 2014]

ProjectProto. *Android Bluetooth Oscilloscope*. Enero de 2014.

http://projectproto.blogspot.mx/2010/09/android-bluetooth-oscilloscope.html

[Android 2014]

Android. *Bluetooth* | *Android Developers*. Enero de 2014.

http://developer.android.com/guide/topics/connectivity/bluetooth.html

[Android 2014]

Android. *View* | *Android Developers*. Enero de 2014.

http://developer.android.com/reference/android/view/View.html

[Android 2014]

Android. *SurfaceView* | *Android Developers*. Enero de 2014.

http://developer.android.com/reference/android/view/SurfaceView.html

[Android 2014]

Android. *Thread* | *Android Developers*. Enero de 2014.

http://developer.android.com/reference/java/lang/Thread.html

[Android 2014]

Android. *Handler* | *Android Developers*. Enero de 2014.

http://developer.android.com/reference/android/os/Handler.html

[Wikipedia 2014]

Wikipedia. *Oscilloscope – Wikipedia, the free encyclopedia*. Mayo de 2014.

http://en.wikipedia.org/wiki/Oscilloscope

[OsciPrime 2014]

OsciPrime. *OsciPrime – An Open Source Android Oscilloscope*. Junio de 2014.

http://www.osciprime.com/index.php?p=source

[Desarrolloweb.com 2014]

Desarrolloweb.com. *Introducción a Android*. Octubre de 2014

http://www.desarrolloweb.com/articulos/introduccion-android.html