UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema inalámbrico para la medición de energía de un auto eléctrico

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Julián de Gortari Briseño

DIRECTOR DE TESIS

M.I. José Castillo Hernández

Ciudad Universitaria, Cd. Mx., 2019

Agradecimientos

Al laboratorio de Electrónica del Instituto de Ciencias Aplicadas y Tecnología, especialmente a mi asesor de tesis M.I. José Castillo Hernández por todo el apoyo, paciencia y conocimiento compartido a lo largo del desarrollo del proyecto.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por haberme ayudado a generar el conocimiento para convertirme en profesionista y al Programa de Apoyo a Proyectos para la Innovación y Mejoramiento de la Enseñanza por haberme brindado apoyo económico durante la realización de este trabajo.

La tesis se la dedico a mi familia, en especial a mis padres y mi hermano por todo el apoyo incondicional, al igual que a mis compañeros de la Facultad y del GAR.

Índice

Lista de figuras

Nomenclatura

Símbolo	Significado
A/D	Analógico-digital
CSV	Valores separados por coma
DC	Corriente directa
EUSART	Transmisor-receptor síncrono-asíncrono universal mejorado
LCD	Pantalla de cristal líquido
LiPo	Polímero de litio
RAM	Memoria de acceso aleatorio
SPI	Interfaz periférica serial
UART	Transmisor-Receptor Asíncrono Universal

Introducción

La medición es el proceso por el cual se le asigna un número a una cantidad física o fenómeno. Para esto se utilizan sistemas que generalmente consisten de tres etapas: una en la que se detecta la variable física y realiza una transformación de un efecto físico a otro, haciendo más sencilla la manipulación de la señal resultante; una etapa intermedia que modifica la señal mediante amplificación, filtrado u otro medio para adaptarla a las características de las etapas posteriores; y una parte final que indica, registra o controla la variable que se mide (Holman, 1986). Los sistemas de medición pueden incluir dispositivos para transmitir las señales a largas distancias, de modo que no sea necesario que el operador se encuentre cerca del dispositivo para monitorear la medición; a este tipo de sistemas se les conoce como sistemas de telemetría. Este trabajo se enfoca en la medición de energía eléctrica por medio de un sistema de telemetría, por lo que a continuación haremos un breve recuento de los métodos usados para la medición de esta variable eléctrica. Con base en esto, en los siguientes apartados de la introducción explicaremos las consideraciones que se tuvieron para integrar un sistema de telemetría particular a un auto de pruebas eléctrico.

La medición de energía eléctrica se realiza comúnmente con watthorímetros, los cuales calculan la energía empleada por una carga cada cierto tiempo mediante la medición de su potencia instantánea, para esto es necesaria la utilización de sensores de voltaje y corriente, cuyo funcionamiento es detallado en (Holman, 1986). Cada una de estas mediciones de energía se van adicionando para formar un total que corresponde a lo consumido desde el tiempo que se activó el medidor. Los watthorímetros comúnmente utilizan los watts-hora como unidad de medición, que es equivalente a 3600 joules.

Estos instrumentos se pueden clasificar principalmente en electromecánicos y digitales:

Los electromecánicos consisten en 2 electroimanes conectados a las líneas de alimentación que producen campos magnéticos variantes desfasados en 90°, uno de los cuales es proporcional al voltaje (electroimán de tensión) y el otro a la intensidad de corriente (electroimán de flujo) (Raman, 2012). Un disco de

aluminio montado sobre un eje separa a los dos electroimanes, dentro del cual se generan corrientes circulares como consecuencia de los campos magnéticos desfasados, que a su vez producen un torque (Electrical4U, 2018). La velocidad angular del disco es entonces proporcional al producto del voltaje con la corriente, o mejor dicho, a la potencia instantánea. Un sistema de engranes se conecta con el eje del disco de aluminio y por medio de una serie de marcadores indica la energía empleada a través del tiempo, que es proporcional al número de revoluciones completadas por el disco. Por último, un sistema de frenado que consiste en un pequeño imán cuyos extremos envuelven al disco, genera un torque en sentido opuesto al producido por los electroimanes, con lo que se controla la velocidad del disco y se logra que deje de girar en el momento que se detiene el suministro de energía.

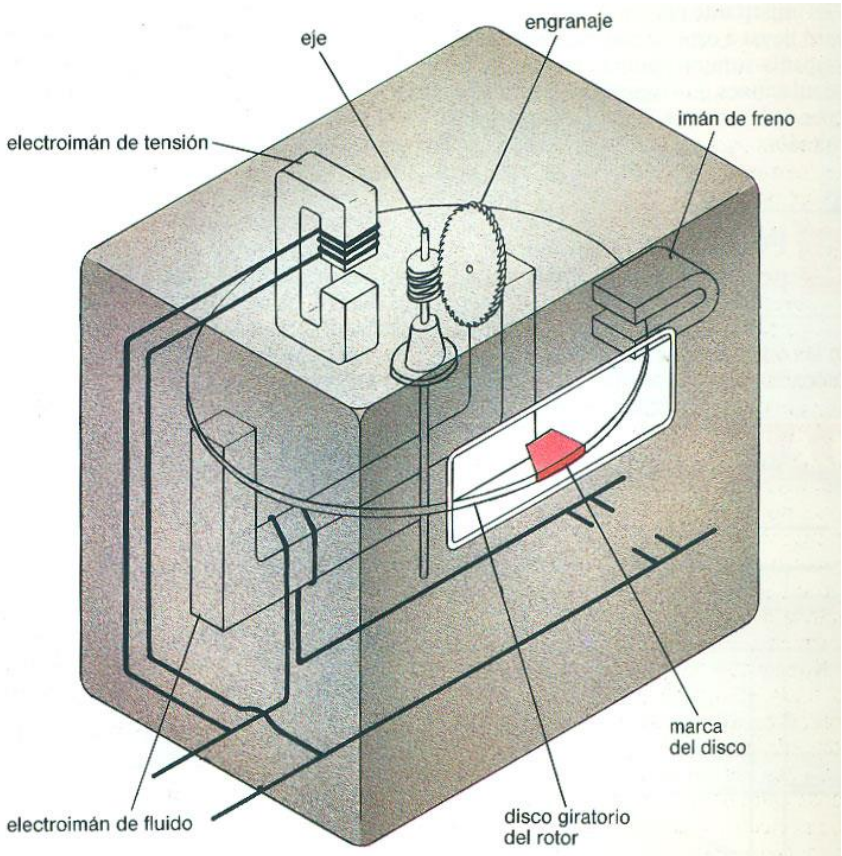


Figura 1. Watthorímetro electromecánico

Fuente: Bricolaje. (2011) ***Funcionamiento contador electrico*** *[Internet]. Disponible dese <http://bricolage-pvc.com/2011/04/funcionamiento-contador-electrico.html> [Acceso 5 de marzo 2019].*

Los watthorímetros digitales, en contraste, utilizan convertidores analógicos digitales para obtener la representación digital del nivel de corriente y voltaje. Para esto es necesario primero transformar el valor de la corriente en un valor de voltaje proporcional y ajustar los niveles de las dos señales al intervalo soportado por el convertidor. Este dispositivo obtiene muestras periódicas de las señales que llegan y estas muestras son las que posteriormente cuantifica y codifica, de forma que se obtienen valores digitales a su salida. El posterior procesamiento de estos valores para obtener el cálculo de la energía consumida en un tiempo dado puede ser realizado por medio de un conjunto de circuitos digitales discretos o a través de un microcontrolador. Por ejemplo, para el primer caso, los valores digitales obtenidos del convertidor pueden servir de entradas para un multiplicador digital cuya salida represente la potencia instantánea; este valor puede ser usado a su vez por un convertidor digital a frecuencia, con su salida conectada a un contador digital, con el fin de que se registre la energía demandada.

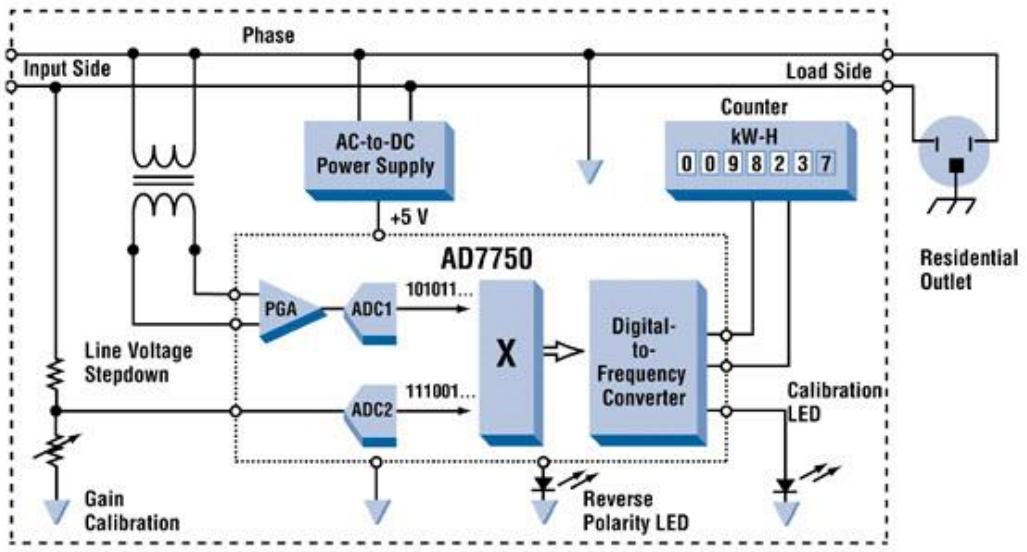


Figura 2. Ejemplo de watthorímetro digital, donde el convertidor analógico digital, el multiplicador digital y el convertidor digital a frecuencia están contenidos dentro de un mismo circuito integrado.

Fuente: Daigle, P. (1999) All-Electronic Power and Energy Meters. ***Analogue Dialogue****. 33(2), febrero, p. 1.*

Por otro lado, las ventajas que suponen el uso de microcontroladores se encuentran en la reducción del número de circuitos utilizados y la gran flexibilidad que presentan como consecuencia de permitir programar el funcionamiento del

dispositivo. Para esto, los microcontroladores ponen a disposición de los programadores toda una serie de interrupciones con las cuales se consigue que el código responda de forma inmediata a los cambios que suceden fuera de su ámbito de procesamiento, como por ejemplo, el término de la conversión de un valor analógico en digital (Basile et al., 2014). Realizar las operaciones aritméticas correspondientes al cálculo de la potencia es tan sencillo como agregar ciertas instrucciones al código que usen los resultados de las conversiones, y mantener el registro de la energía consumida se reduce a utilizar la memoria RAM integrada para almacenar este valor. El hecho de que los microcontroladores generalmente ya tengan integrados distintos módulos de comunicación serie (SPI, UART, etc.) permite que la transmisión y despliegue de los resultados sea de igual forma más sencilla, requiriendo solamente la configuración de ciertos registros especiales para su operación.

Definición del problema

El diseño de un vehículo eléctrico tiene varias etapas, entre las cuales se encuentran el diseño mecánico, la implementación electrónica, el control de los motores, la instrumentación del auto, por mencionar algunos; cada una de estas etapas a su vez se prueban y se rediseñan para obtener un desempeño más eficiente. Particularmente, una parte importante de los vehículos eléctricos es su consumo eléctrico, del cual depende que su autonomía sea rentable, por lo que durante la fase de diseño y pruebas, contar con un medio para contabilizar su consumo es necesario. Sin embargo, en ocasiones esto no es fácil debido a que no es posible que el conductor nos ofrezca esta información de primera mano (debe estar atento a la conducción del vehículo), por lo tanto, un instrumento que se puede anexar durante las pruebas del auto es un medidor de consumo eléctrico que usualmente se conoce como juliómetro o watthorímetro. Dentro de las características deseables del instrumento se considera el registro del voltaje, la corriente y la energía consumida, pero algo sumamente útil sería contar con un sistema de telemetría que permita realizar pruebas y valorar el desempeño del auto en el instante mismo en el que ocurren.

Actualmente con los avances en el desarrollo de tarjetas prototipo, y la amplia gama de módulos de radiofrecuencia y módulos de instrumentación, el diseño de un watthorímetro es más que factible.

En este trabajo se plantea el diseño y desarrollo de un instrumento que permita medir las variables de voltaje y corriente, el consumo energético y realizar la telemetría de un motor alimentado por una batería, que forma parte del sistema de tracción de un auto de pruebas eléctrico. Con la información recabada se evaluará el desempeño del auto, y de ser necesario, se puede plantear un rediseño para mejorar su eficiencia.

Entorno actual

En el Laboratorio de Electrónica del Instituto de Ciencias Aplicadas y Tecnología (ICAT) se ha ido gestando el desarrollo de un auto eléctrico con base en el control de motores brushless y baterías de ion-litio. En este contexto se han desarrollado tarjetas para controlar la potencia que se suministra al motor; estos diseños están basados en electrónica de potencia con una filosofía basada en sistemas conmutados. Estos sistemas se han probado en un auto eléctrico que participará en las competencias del Ecomaratón Shell que se llevará a cabo en Sonoma, California. A partir de esto se consideró necesario desarrollar un sistema de telemetría que permita monitorear las variables eléctricas del auto, y de esta forma, tener un mejor desempeño del vehículo. Actualmente el Laboratorio de Electrónica cuenta con infraestructura de diseño analógico y digital necesario para el desarrollo del presente proyecto, así como experiencia en el diseño de sistemas de telemetría (Castillo et al., 2013).

Descripción del problema

El proceso de medición de energía con un microcontrolador requiere de la utilización de su convertidor analógico digital integrado con el fin de obtener una representación digital de las muestras de voltaje y corriente necesarias, y a partir de esto, realizar los cálculos correspondientes al consumo energético a través de un algoritmo. Para todo esto, es necesario conocer la función de entrada/salida no solo del propio convertidor, sino, en el caso de la corriente, del sensor que devuelve un voltaje de salida proporcional a la medición; y en el caso

del voltaje, del factor de proporción que relaciona los niveles de tensión generados por el motor con aquellos soportados por el convertidor. Debido a que el cálculo de la energía consumida requiere de una estimación bastante aproximada sobre el intervalo de tiempo al cual corresponde, y a que con la medición del voltaje y corriente de la carga no se puede calcular más allá de la velocidad a la cual se consume tal energía (potencia instantánea), se hace también indispensable tener un control confiable sobre el tiempo durante el cual operan los diferentes módulos y se ejecutan las distintas rutinas dentro del microcontrolador, a fin de evitar imprecisiones en los cálculos.

Por otro lado, la transmisión y despliegue de los resultados constituyen otra serie de disyuntivas sobre el tipo de comunicación que se utiliza y los sistemas operativos de los dispositivos a los que van dirigidos los datos. Por ejemplo, debido a que el sistema de medición va montado directamente sobre el auto y este último está en constante movimiento, es necesario considerar un sistema de transmisión y recepción que sea inalámbrico, tenga un gran intervalo de alcance y que sea fácil de operar con un microcontrolador.

El receptor, instalado en una estación fija, debe recibir los datos transmitidos por el sistema de medición en el auto. Se debe contar con un sistema de retransmisión hacia los dispositivos finales en esta estación que tome en consideración las tecnologías de comunicación soportadas por aquéllos. Por ejemplo, es más fácil para los smartphones utilizar comunicación inalámbrica por medio del estándar Bluetooth, mientras que las computadoras se acoplan mejor a métodos alámbricos, como la comunicación serie.

La realización de programas especiales para el despliegue de los resultados de forma gráfica debe tomar en cuenta las diferentes capacidades que tienen los dispositivos para procesar de forma eficiente los datos recibidos, por lo que es necesario que, por ejemplo, desde el sistema de medición se agregue el resultado de un contador a cada una de las muestras enviadas con el fin de mantener intacta la secuencia de medidas y lograr posicionarlas de forma certera en la escala de tiempo sin importar el momento en que son recibidas. Asimismo, es necesario que los dispositivos finales verifiquen que los datos recibidos no estén corruptos. Finalmente, ante la necesidad de guardar los datos para futuros análisis, se debe implementar la funcionalidad de almacenaje en archivos.

Relevancia y limitaciones

Este trabajo permitirá diseñar y probar un sistema digital de monitoreo de consumo energético que se distinga de otros por su capacidad de desplegar los resultados de forma gráfica en una variedad de dispositivos, haciendo uso para ello de distintas tecnologías de comunicación. Cabe resaltar que está diseñado para ser utilizado con un auto de pruebas eléctrico en particular, pero es posible adaptarlo para su uso con otros modelos ya que cuenta con la capacidad para ser reprogramado, de forma que también se le pueden agregar nuevas funcionalidades en caso de que así lo requiera el usuario final.

Método

En la figura 3 se presenta el diagrama de bloques del sistema propuesto. El diagrama se divide en dos bloques principales que corresponden al sistema de medición de energía y la estación retransmisora.

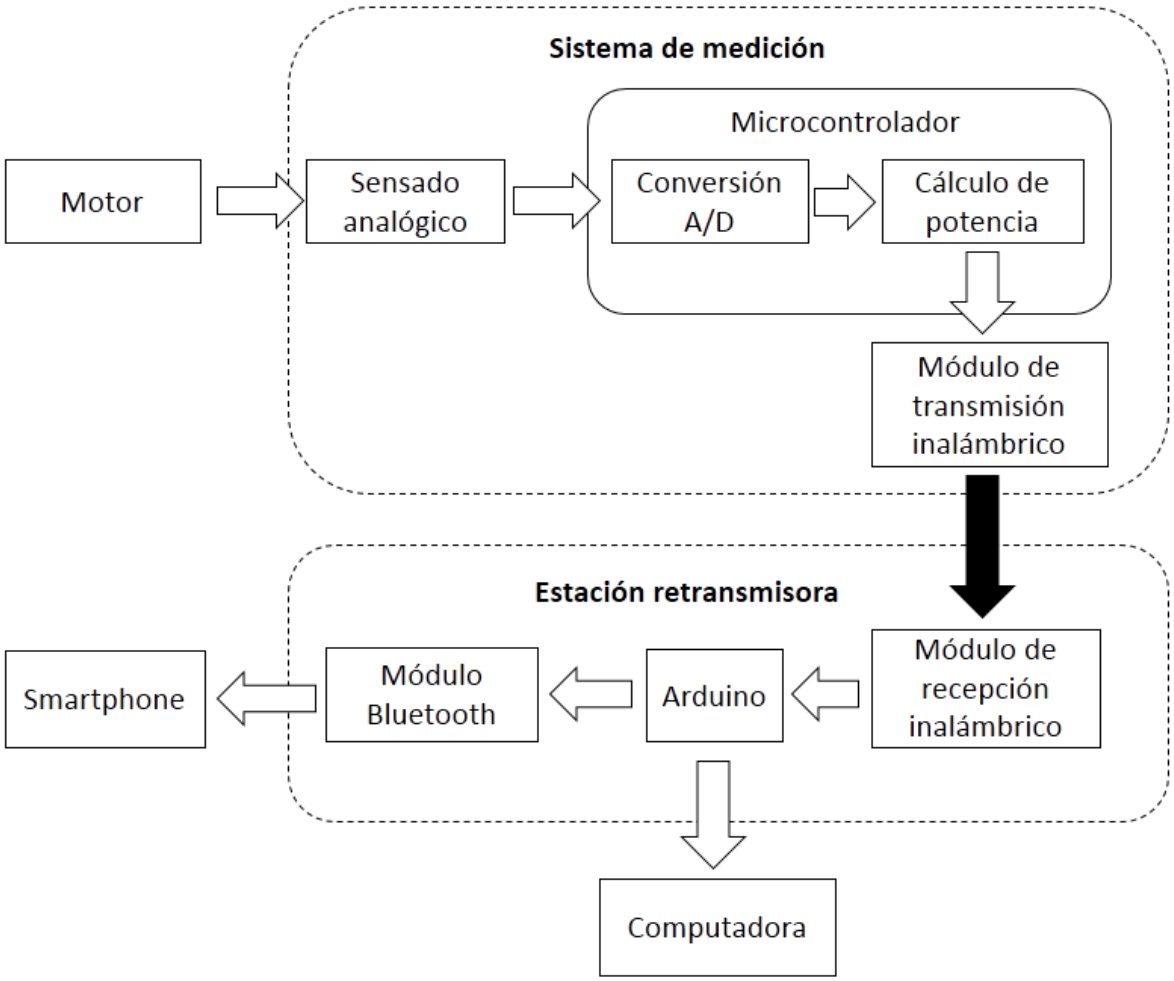


Figura 3. Diagrama de bloques del sistema propuesto

El sistema de medición está compuesto por una etapa de sensado analógico, un microcontrolador y un módulo de transmisión inalámbrica. Dentro de la etapa de sensado analógico, se realiza el sensado del voltaje por medio de un arreglo de resistencias divisor de voltaje, mientras que para la intensidad de corriente se utiliza un sensor de efecto Hall junto con su electrónica de acondicionamiento respectivo. Las señales resultantes del sensor y del divisor de voltaje son leídas por el microcontrolador a través de su convertidor analógico digital y los valores

devueltos por este módulo son utilizados para hacer el cálculo de potencia por medio de un algoritmo programado dentro del mismo microcontrolador. Este dispositivo se comunica de forma serial (SPI o UART) con un módulo de transmisión inalámbrico en el sistema de medición, que se encarga de enviar los datos al módulo de recepción en la estación retransmisora.

La estación retransmisora consiste del módulo de recepción inalámbrico, una tarjeta Arduino y un módulo Bluetooth. La tarjeta Arduino obtiene los datos del módulo de recepción por medio de comunicación serie y los retransmite de la misma forma hacia una computadora, a la que puede estar o no conectada, y a un módulo Bluetooth; este último dispositivo sirve para transmitir los datos a algún smartphone con el que haya iniciado una conexión.

Por último, el sistema de medición se integra al sistema controlador del motor del auto eléctrico, que es producto de otro trabajo, y se polariza a partir de la salida de un convertidor de DC-DC que se encuentra en el circuito controlador, el cual adapta el nivel de voltaje generado por la batería a los 5 [V] necesarios por el circuito digital; en el caso de la estación retransmisora, ésta puede utilizar una batería externa para su polarización.

Objetivo y resultados esperados

Objetivo general

Diseñar y desarrollar un sistema de telemetría para supervisar y recabar información de la batería del sistema de tracción de un vehículo eléctrico con el fin de mejorar su desempeño energético.

Objetivos particulares

1. Desarrollar la electrónica de los sistemas de monitoreo, transmisión y recepción de datos.
2. Desarrollar el software con el que se puedan visualizar los datos generados, tanto en computadoras personales como en smartphones.
3. Realizar las pruebas en tiempo real del sistema y proponer cambios y mejoras para un trabajo futuro.

Organización del escrito

En el capítulo 1 se da una descripción del hardware utilizado para acondicionar las señales proporcionales al voltaje y corriente del motor, de los circuitos usados para procesar los datos, transmitirlos y recibirlos vía inalámbrica, y de las consideraciones tomadas en la conversión de las muestras analógicas a digitales. En el capítulo 2 se hace un análisis del código utilizado en el programa del microcontrolador y en las aplicaciones creadas para la visualización de los resultados para los dispositivos con sistema operativo Android, Windows y GNU/Linux. En el capítulo 3 se presentan los resultados de las pruebas realizadas tanto para el sistema de medición como para la estación retransmisora. Por último, en el capítulo 4 se hace un análisis de lo logrado con esta tesis y las posibles mejoras futuras a los sistemas construidos.

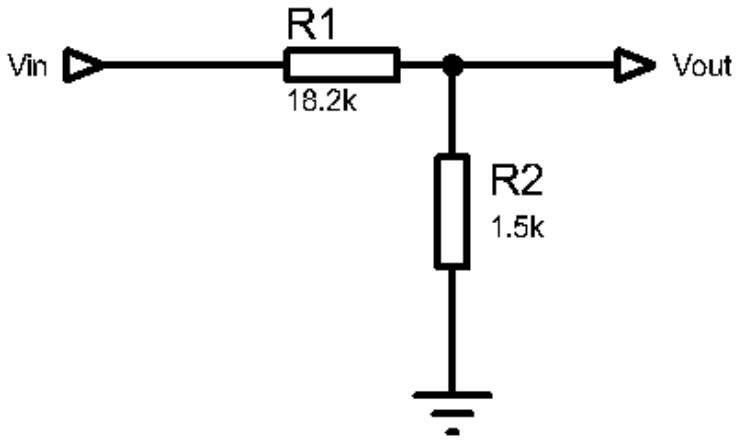
1 Hardware

1.1 Acondicionamiento analógico de señales

El acondicionamiento analógico de señales es el proceso por el cual se adaptan las propiedades de éstas de acuerdo con las especificaciones impuestas por el sistema dado. En el caso de esta tesis, el acondicionamiento es necesario para poder procesar el voltaje y corriente aplicados al motor. A continuación se describen las consideraciones tomadas para el acondicionamiento de cada tipo de medición.

Sensado de voltaje

La diferencia de potencial máxima generada por el motor eléctrico es de 60 [V], por lo que es necesario el uso de un divisor de voltaje para ajustar la amplitud de la señal a un intervalo de 0 a 5 [V] compatible con las entradas del microcontrolador utilizadas para la conversión analógica-digital. El arreglo de resistencias para tal divisor quedó como sigue:



Donde *Vin* es el voltaje aplicado al motor y *Vout* es proporcional a este nivel de tensión, pero adaptado al intervalo soportado por el convertidor. La relación matemática que representa este ajuste se muestra a continuación:

$$V_{OUT} = \frac{R2}{R1 + R2} V_{IN} \tag{1.1}$$

Tomando en cuenta los valores de las resistencias que se muestran en el esquema anterior, V_{out} es entonces igual a $\left(\frac{15}{197}\right)V_{in}$.

Dentro del programa se introdujo una constante de multiplicación *K* con el fin de poder calcular el valor de tensión aplicado al motor V_{in} a partir del valor V_{out} devuelto por el divisor de voltaje. Esta constante resultó igual al inverso de la función salida/entrada de este divisor:

$$K = \frac{R1 + R2}{R2} \tag{1.2}$$

Por lo que *K* resultó en un valor de $13.1\overline{3}$.

En la práctica, la tensión resultante del divisor de voltaje fue ligeramente menor al valor teórico calculado por medio de la ecuación 1.1, por lo que fue necesario ajustar el cálculo del voltaje del motor para considerar este error, que resultó igual a 0.03 [V].

Sensado de corriente

Se utilizó un sensor de corriente bidireccional ACS756KCA-050B (Allegro MicroSystems, 2018) que se basa en un circuito de efecto Hall encargado de convertir el campo magnético generado por el paso de corriente, sobre las terminales del sensor, en un voltaje proporcional (Electronics Tutorials,). Este sensor cuenta con un intervalo de medición de ±50 [A] y regresa un valor de voltaje proporcional V_{iout}, del cual puede obtenerse el valor de la corriente de entrada I_p mediante la fórmula:

$$I_p = \frac{(V_{iout} - V_{offset})}{sensibilidad} \tag{1.3}$$

Donde V_{offset} corresponde a la salida del sensor cuando no hay flujo de corriente sobre sus entradas y la sensibilidad es de aproximadamente 40 [mV/A] de acuerdo con la información proveída por el fabricante. Idealmente V_{offset} es igual a la mitad del voltaje de polarización que se aplica sobre el sensor, pero en la práctica puede existir una ligera variación que hace necesario medir este valor de forma directa. La relación entre la entrada y salida de este sensor es idealmente lineal, pero en la práctica llegan a presentarse ligeras desviaciones por varios factores que no serán considerados para este trabajo.

Debido a que solamente quisimos medir la intensidad de corriente en una sola dirección, se empleó un amplificador operacional TLV271 de tipo rail-to-rail

polarizado por una única fuente, con el cual se buscó obtener 0 [V] para niveles de tensión de entrada menores a 2.5 [V], mientras que para aquellas entradas en el intervalo de 2.5 [V] a 5 [V], se buscó una salida proporcional dentro del intervalo de 0 [V] a 5 [V].

El análisis del circuito de acondicionamiento mostrado en la figura 4 se realiza a continuación con el fin de demostrar la relación entre la entrada y salida del amplificador.

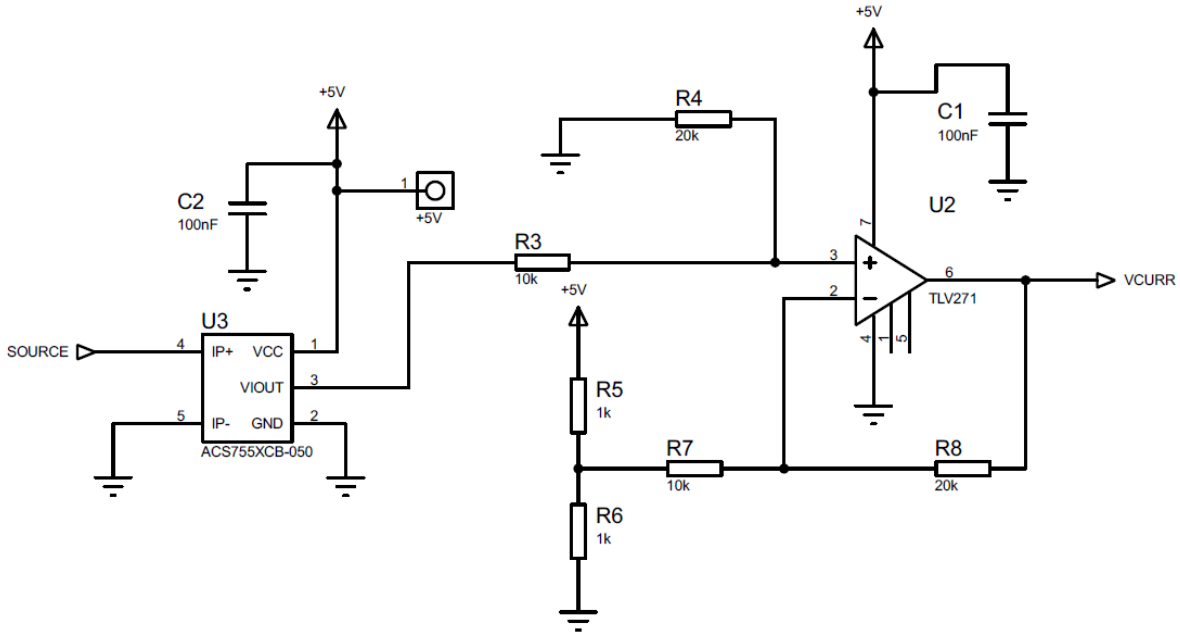


Figura 4. Circuito de acondicionamiento

Nota: los capacitores C1 y C2 son simplemente capacitores de bypass con los que se logra filtrar el ruido que llega a generarse.

Primero, es conveniente convertir el divisor de voltaje que aparece en el circuito compuesto por las resistencias *R5* y *R6*, y la fuente de voltaje *Vcc* de 5 [V], en un circuito equivalente conformado por una fuente de voltaje en serie con una sola resistencia, utilizando para ello la transformación Thévenin-Norton-Thévenin. El voltaje y la resistencia de Thévenin equivalentes se muestran ya en la figura 5 y se calculan como sigue:

$$V_{th} = \frac{R6}{R5 + R6} Vcc \qquad (1.4)$$

$$R_{th} = \frac{R5 \cdot R6}{R5 + R6} \tag{1.5}$$

El resultado es un voltaje V_{th} de 2.5 [V] y una resistencia R_{th} de 500 [Ω].

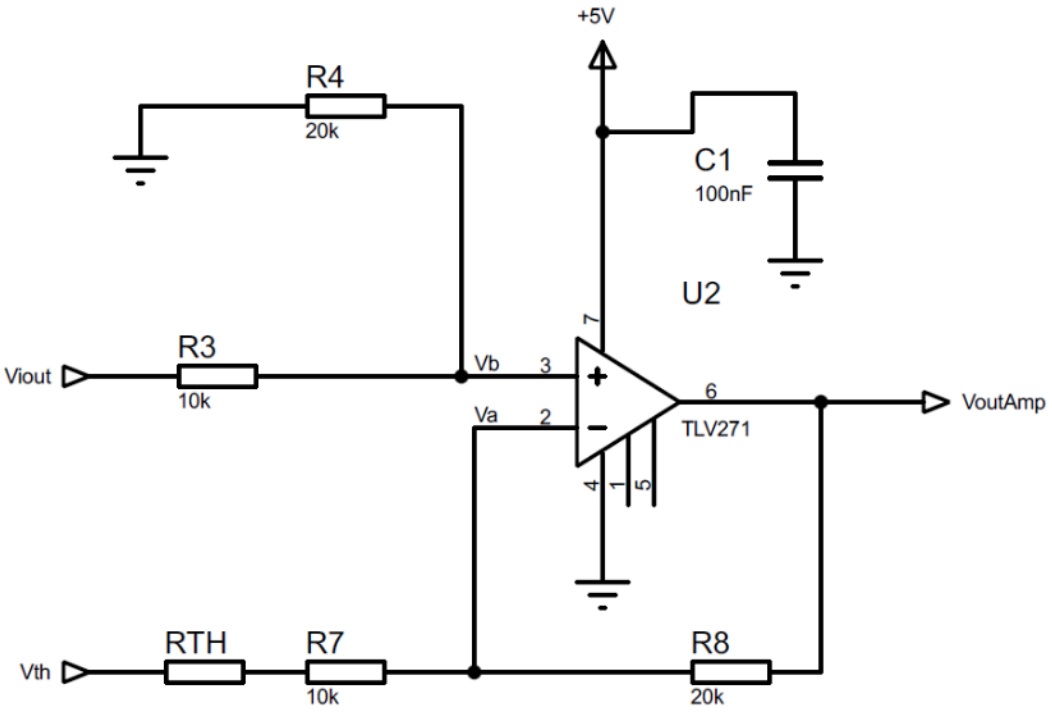


Figura 5. Circuito con amplificador operacional en modo restador

Una vez hecho este cambio, es posible apreciar con mayor claridad que el amplificador operacional se encuentra en modo restador, por lo que se utiliza el teorema de superposición para calcular por separado la contribución de cada una de las entradas a la salida (Mancini y Carter, 2009). Para esto es necesario primero establecer las siguientes relaciones:

$$I_{th} = \frac{V_{th} - V_a}{R_{th} + R7} \tag{1.6}$$

$$I_{outAmp} = \frac{V_a - V_{outAmp}}{R8} \tag{1.7}$$

$$V_b = \frac{R4}{R3 + R4} V_{iout} \tag{1.8}$$

Para el primer caso, si V_{iout} es igual a cero, V_b y V_a también lo son. Al no existir flujo de corriente por las entradas del amplificador, I_{th} es igual a I_{outAmp}, por lo que

se puede establecer la siguiente relación de igualdad con las ecuaciones 1.6 y 1.7:

$$\frac{V_{th}}{R_{th}+R7}=\frac{-V_{outAmp1}}{R8} \tag{1.9}$$

Posteriormente se obtiene la salida parcial V_{out1} en función de V_{th} y de las resistencias a partir de la ecuación 1.9:

$$V_{outAmp1}=-\frac{R8}{R_{th}+R7}V_{th} \tag{1.10}$$

Para el segundo caso, cuando V_{th} es igual a cero, se debe calcular primero la salida del divisor de tensión que resulta en V_a, de la siguiente forma:

$$V_a=\frac{R_{th}+R7}{R_{th}+R7+R8}V_{outAmp2} \tag{1.11}$$

Bajo el supuesto que V_a y V_b son iguales, se sustituye el término V_b en la ecuación 1.8 por la relación obtenida en la ecuación 1.11 para V_a, y se despeja la salida parcial $V_{outAmp2}$:

$$V_{outAmp2}=\frac{R4}{R3+R4}\cdot\frac{R_{th}+R7+R8}{R_{th}+R7}V_{iout} \tag{1.12}$$

Finalmente se suman las dos salidas parciales definidas en las ecuaciones 1.10 y 1.12, y se obtiene la relación de salida total V_{outAmp}:

$$V_{outAmp}=\frac{R4}{R3+R4}\cdot\frac{R_{th}+R7+R8}{R_{th}+R7}V_{iout}-\frac{R8}{R_{th}+R7}V_{th} \tag{1.13}$$

Al sustituir las variables de la ecuación 1.13 por sus valores correspondientes, resulta en la siguiente fórmula:

$$V_{outAmp}=\frac{122}{63}V_{iout}-\frac{100}{21} \tag{1.14}$$

Cabe resaltar que la ecuación 1.14 difiere un poco de la relación ideal que se estaba buscando:

$$V_{outAmp}=2V_{iout}-5 \tag{1.15}$$

Sin embargo, no presenta problema alguno ya que es una cuestión que se puede resolver en la programación del microcontrolador.

Una vez obtenida la relación de entrada y salida del circuito de acondicionamiento, es necesario adaptar la función definida previamente para el cálculo de la intensidad de corriente del motor, correspondiente a la ecuación 1.3, a fin de que se encuentre en términos de la salida del acondicionamiento. Para esto, primero debemos despejar a V_{iout} de la ecuación 1.14, con el objetivo de establecer la relación con la cual se pueda calcular la salida del sensor en términos de la salida del amplificador operacional V_{outAmp}:

$$V_{iout} = \frac{63}{122} V_{outAmp} + \frac{100}{21} \tag{1.16}$$

La salida del sensor cuando no hay corriente fluyendo sobre sus terminales, V_{offset}, se calcula también con la ecuación 1.16. La salida del amplificador operacional que resulta de aplicar el voltaje V_{offset} como entrada la representamos con el término V_{outAmp_offset}.

Una vez reemplazadas V_{iout} y V_{offset} en la ecuación 1.3 por sus relaciones equivalentes planteadas en términos de la salida del amplificador, se obtiene la siguiente ecuación, que es la que se utilizó finalmente para el cálculo de la intensidad de corriente en el programa ejecutado por el microcontrolador:

$$I_p = \frac{63}{122} \cdot \frac{(V_{outAmp} - V_{outAmp\_offset})}{sensibilidad} \tag{1.17}$$

Por último, es necesario recalcar que la señal que resulta del acondicionamiento pasa por otro amplificador operacional configurado como seguidor (Franco, 2015), para favorecer el acoplamiento de impedancias con la entrada del convertidor analógico-digital, permitiendo de esta forma una conversión más precisa.

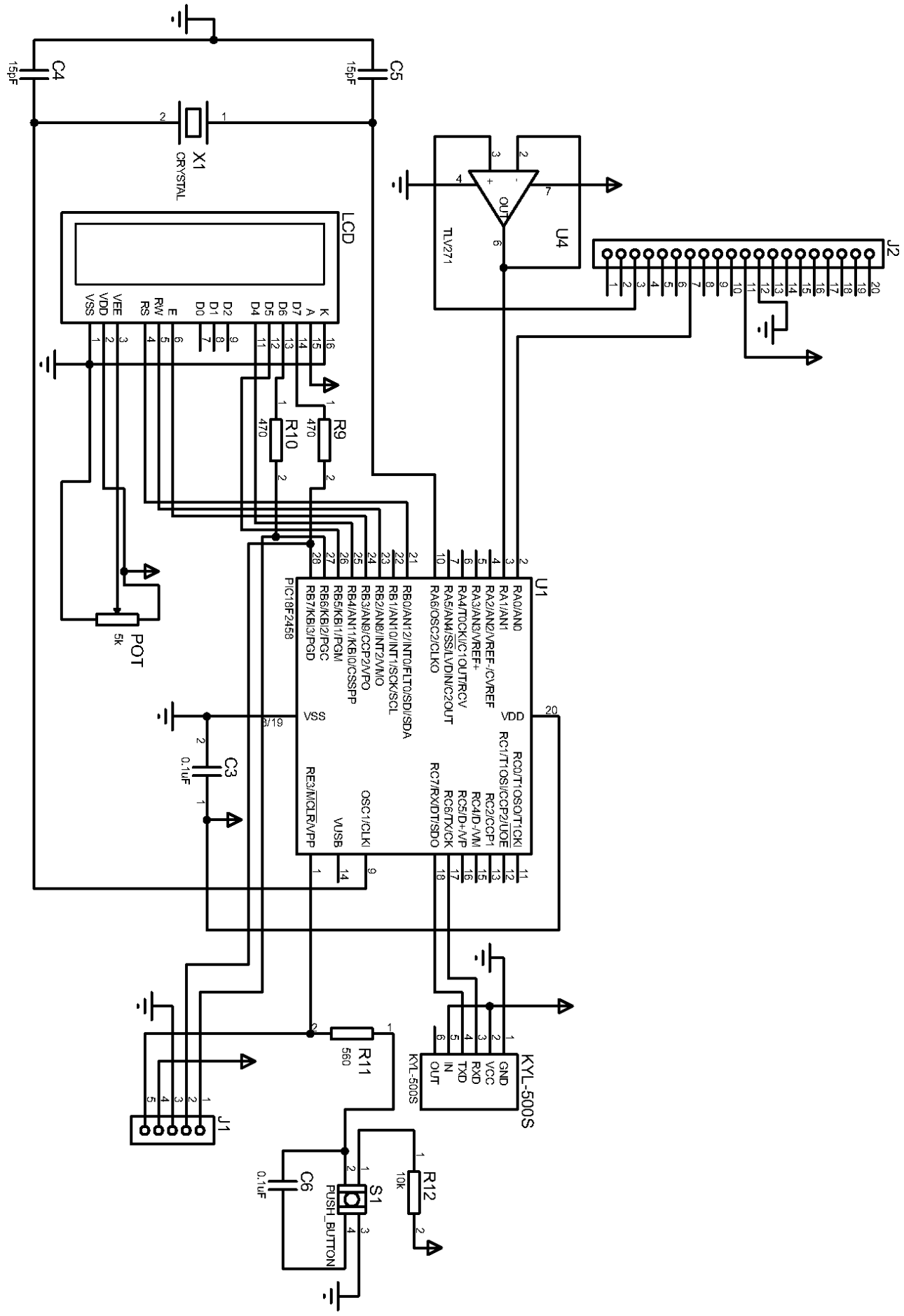


Figura 6. Circuito del sistema de medición

1.2 Circuito digital

El circuito digital correspondiente al sistema de medición se puede apreciar en la figura 6. El microcontrolador U1 es el que se encarga de hacer todo el procesamiento de las mediciones, de enviar los datos al módulo de transmisión KYL-500S y de mostrar los resultados en la pantalla LCD QC1602A. El conector J1 se utiliza para reprogramar el microcontrolador en caso de que se necesite, mientras que del conector J2 llega la polarización del circuito y las señales analógicas correspondientes al voltaje y corriente del motor. En el caso de la medición de corriente se utiliza el amplificador operacional U4 para el acoplamiento de impedancias, como ya se había mencionado anteriormente.

Se procederá ahora con una explicación más detallada sobre los elementos principales del circuito.

Microcontrolador

El microcontrolador utilizado es un PIC18F2458 que contiene un módulo de conversión analógico-digital de 12 bits y un módulo de comunicación serie asíncrona UART, entre otros periféricos.

Para establecer la frecuencia de operación del microcontrolador en 20 [MHz], lo suficientemente rápido para los procesos de conversión analógica-digital que se quieren realizar, como se muestra posteriormente, se puede utilizar un cristal que opere a esa frecuencia y configurar los bits FOSC3:FOSC0 del registro de configuración CONFIG1H del microcontrolador para utilizar este cristal como oscilador primario en modo de alta velocidad (HS). Es necesario utilizar dos capacitores cerámicos de 15 [pF] para asegurar la forma cuadrada de los pulsos producidos por el oscilador, y el postescalador dentro del microcontrolador debe ser configurado de forma que no afecte la frecuencia de oscilación, por medio de los bits CPUDIV del registro de configuración CONFIG1L. Por último, para seleccionar al reloj del sistema como oscilador primario, se deben manipular los bits SCS1:SCS0 del registro OSCCON.

Transmisión de datos

El módulo KYL-500S es un transceptor de radiofrecuencia miniatura que opera en la banda de los 400 a 470 [MHz], tiene un intervalo de alcance de

aproximadamente un kilómetro y cuenta con un módulo UART integrado con el cual es posible comunicarse mediante un microcontrolador (Shenzen KYL Communication Equipment Co.,). El módulo consiste en sí, del transceptor CC1020 de Texas Instruments que funciona a baja potencia, el cual se conecta con un microcontrolador P89LPC921 que implementa la funcionalidad de comunicación serie asíncrona con dispositivos externos. En el caso de esta tesis, se configuró el módulo para que transmitiera a una frecuencia de 434.325 [MHz] sin ninguna razón en particular más que para probar su funcionamiento.

En el caso del PIC18F2458, este microcontrolador contiene un módulo transceptor síncrono-asíncrono EUSART que para comunicarse con el módulo de radiofrecuencia anterior se configura para trabajar únicamente de forma asíncrona. Los registros que utilizamos en el microcontrolador para esto son TXSTA, para habilitar la opción de transmisión asíncrona a 8 bits y en modo de alta velocidad, y BAUDCON, para habilitar el registro de 16 bits SPBRGH:SPBRG, que sirve para especificar la velocidad de transmisión.

La velocidad establecida para la comunicación entre el módulo KYL-500S y el microcontrolador fue de 9600 bauds. En el caso de este último dispositivo, fue necesario calcular un valor derivado de esta velocidad con el fin de insertarlo en el registro SPBRGH:SPBRG, valor que se obtiene mediante la siguiente fórmula definida por el fabricante:

$$SPBRGH{:}SPBRG = \frac{Fosc/(Tasa\ de\ Bauds\ deseada)}{4} - 1 \qquad (1.18)$$

Donde F_{osc} es la frecuencia de oscilación a la que opera el microcontrolador (20 MHz). Esto resulta en un valor de aproximadamente 520.

Recepción y retransmisión de datos

El circuito de la estación retransmisora se muestra en la figura 7. Éste tiene el propósito de funcionar como intermediario entre los dispositivos que procesan las mediciones y aquellos para visualizar los datos. Para esto, la estación recibe los datos del sistema de medición a través de un módulo de comunicación por radiofrecuencia KYL-500S, y por medio de un Arduino Mega 2560 (representados sus pines en la figura 7 por el conector J1), estos datos son reenviados a un módulo Bluetooth y a la computadora que en ese momento se

encuentre conectada al Arduino. Toda la comunicación que se hace directamente con el Arduino en esta estación es de forma serial y asíncrona.

El módulo Bluetooth utilizado es el HC-05, que además de tener un módulo UART integrado, tiene la capacidad de actuar ya sea como esclavo o como maestro en una conexión inalámbrica, y en caso de necesitar configurar alguno de sus parámetros, permite utilizar comandos AT.

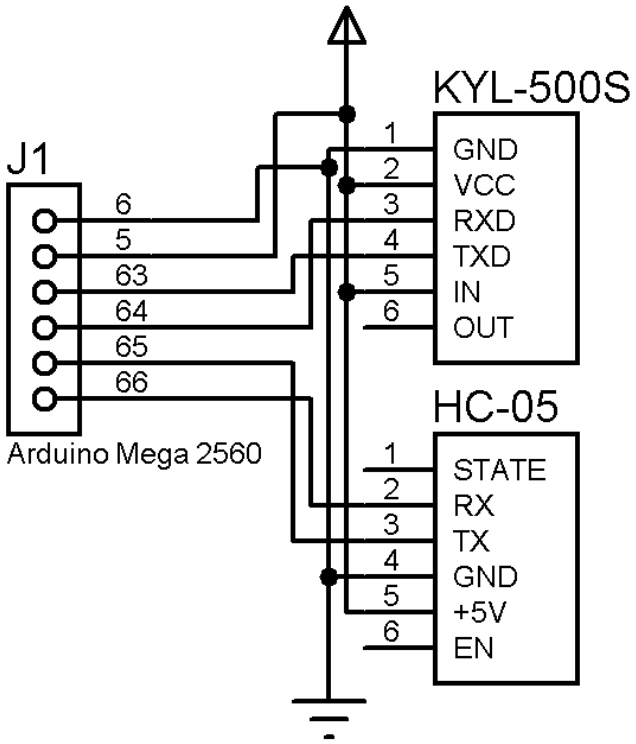


Figura 7. Circuito de la estación retransmisora

1.3 Conversión analógica digital

El microcontrolador utiliza un convertidor analógico-digital con una resolución de 12 bits, con el cual se obtiene un rango de valores del 0 al 4095. De acuerdo con lo que se configuró en la práctica, la resolución de voltaje, o voltaje del bit menos significativo V_{LSB}, se calculó con la siguiente fórmula:

$$V_{LSB} = \frac{V_{Hi} - V_{Low}}{2^n} \tag{1.19}$$

Donde V_{Hi} es el límite superior del intervalo de voltaje que puede ser convertido y V_{Low} el inferior, que en el caso de esta tesis correspondieron a 5 [V] y 0 [V] respectivamente, y *n* es la resolución del convertidor. El resultado fue un valor de 1.22 [mV].

Tiempo de conversión

En el microcontrolador, se utilizan dos canales del convertidor, AN0 y AN1, para obtener las muestras correspondientes a la corriente y el voltaje de forma alternada (la estructura interna del convertidor puede ser apreciada en la figura 8). El tiempo mínimo necesario para obtener la representación digital de un instante de la señal entrante se puede dividir en el tiempo usado para la adquisición de la muestra y en aquél usado para la conversión.

El tiempo mínimo de adquisición se obtiene mediante la suma de tres parámetros: tiempo de asentamiento del amplificador, tiempo de carga del capacitor y el coeficiente de temperatura. De acuerdo con lo expuesto por el fabricante del microcontrolador, el tiempo de asentamiento del amplificador T_{AMP} es igual a 0.2 [µs] y el coeficiente de temperatura T_{COFF} para una temperatura del sistema igual a 85 [°C], la máxima que puede alcanzar, se logra en 1.2 [µs].

Para calcular el tiempo de carga del capacitor, se consideran los siguientes parámetros: el valor del capacitor C_{HOLD}, la resistencia de interconexión R_{IC}, la resistencia del interruptor de muestreo R_{SS} y la impedancia de salida R_S. El valor del capacitor es de 25 [pF], para la resistencia de interconexión se asume su valor máximo de 1 [kΩ], y la resistencia del interruptor se determina en unos 2 [kΩ], donde este último valor se deriva de la gráfica de voltaje contra resistencia mostrada en la figura 8 utilizando un V_{DD} de 5 [V]. La impedancia de salida es de prácticamente 0 en el caso del circuito con el sensor de corriente, debido al amplificador operacional que se usa en modo seguidor, mientras que para el circuito divisor de voltaje, con el cual se obtienen las muestras de tensión del motor, la impedancia de salida es de 1.386 [kΩ]. El tiempo de carga del capacitor T_C se calcula de la siguiente forma:

$$T_C = -(C_{HOLD})(R_{IC} + R_{SS} + R_S)\ln\left(\frac{1}{4096}\right) \qquad (1.20)$$

Lo que resulta en un valor de 0.623 [µs] para el caso de la corriente y en 0.912 [µs] para el del voltaje.

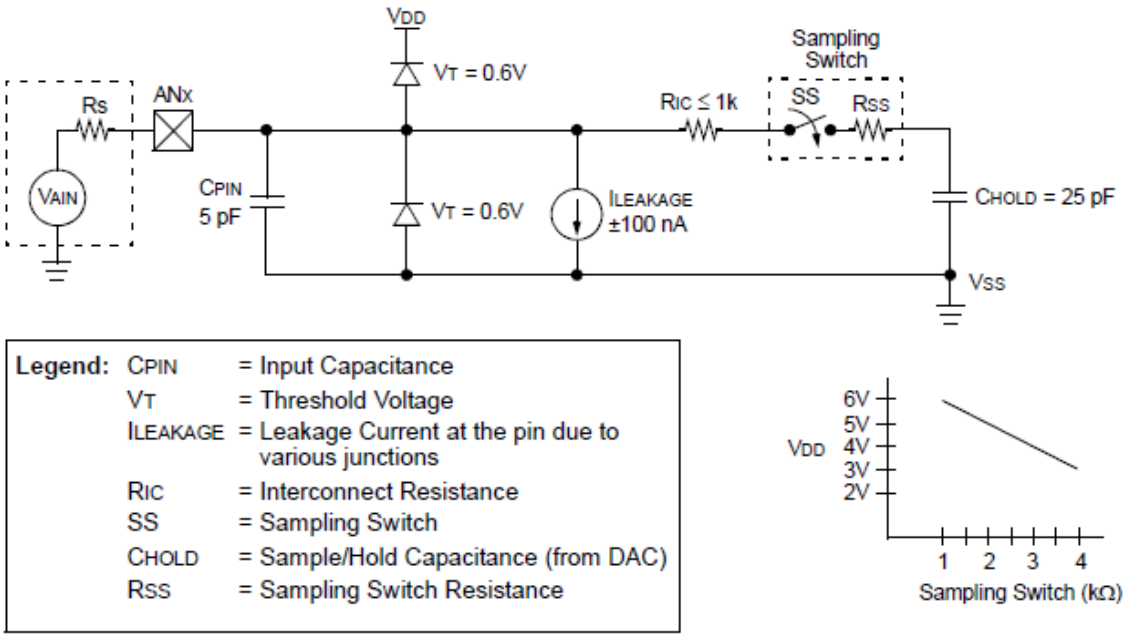


Figura 8. Modelo analógico de la entrada del convertidor

Fuente: Microchip Technology Inc. (2009) ***PIC18F2458/2553/4458/4553: Data Sheet****. EE. UU., p. 25.*

El tiempo mínimo de adquisición se obtiene, entonces, mediante la suma de los tres parámetros calculados anteriormente, resultando en un total de 2.023 [µs] para el caso de la corriente y en uno de 2.312 [µs] para el del voltaje. Para configurar el tiempo de adquisición dentro del microcontrolador se utiliza como referencia el tiempo de conversión por bit mínimo T_{AD}, que el fabricante lo establece en 0.8 [µs]. Tomando en cuenta esto último, el tiempo de adquisición real T_{ACQT} se definió como 20 veces el tiempo de conversión por bit mínimo con el fin de mantener cierto margen frente al tiempo mínimo de adquisición calculado previamente, resultando en un valor total de 16 [µs].

Para calcular el tiempo utilizado tanto para la adquisición como para la conversión de una muestra es necesario sumar, entonces, el tiempo de adquisición calculado anteriormente, 13 veces el tiempo de conversión por bit mínimo usado para la conversión (12 tiempos correspondientes a cada bit del resultado y uno de espera), y el tiempo necesario para descargar el capacitor que retiene la muestra, que equivale a una vez el tiempo de conversión por bit.

Esta suma de tiempos para establecer la duración de la conversión analógica-digital resulta en un total de 27.2 [µs], y se puede apreciar de forma desglosada en la figura 9.

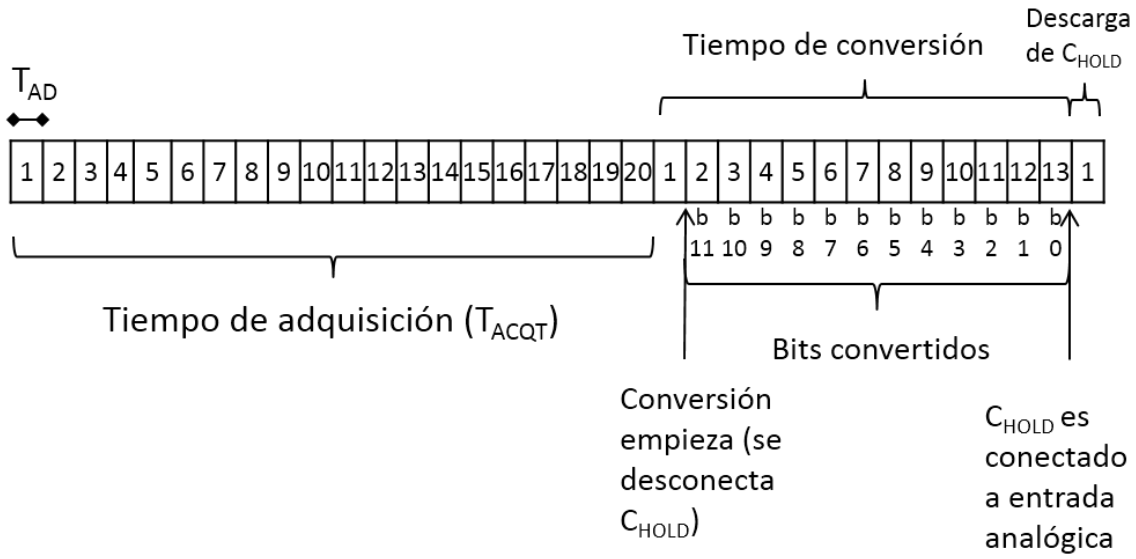


Figura 9. Conversión analógica-digital en función del tiempo de conversión por bit mínimo T_{AD}.

Con el fin de obtener valores certeros del proceso de conversión se quisieron tomar 10 muestras para cada medición de voltaje y de corriente, con el objetivo de promediarlas posteriormente. Esto resultó en un total de 20 muestras por cada ciclo de muestreo, cada uno de estos ciclos con una duración de 544 [µs]. Debido a que queríamos que el tiempo utilizado para la obtención de las mediciones estuviera dentro de un intervalo de 1 [ms], se insertó un retardo de 9 [µs] dentro de la rutina de interrupción usada para aquel proceso, tomando en consideración que el tiempo gastado en la ejecución del código dentro de esta rutina era de 276 [µs].

Finalmente se estableció que el cálculo de la energía consumida fuera para un periodo de 100 [ms], por lo que se introdujo un retardo de 35 [ms] en la rutina principal, tomando en cuenta que el tiempo de ejecución para esta rutina era de unos 64 [ms].

Para corroborar los cálculos se utilizó un osciloscopio y se midió el tiempo de ejecución de las rutinas de interés. Esto se logró configurando un puerto de

salida del microcontrolador para generar un voltaje alto y bajo al iniciar y acabar estas rutinas, obteniendo los resultados observados en las figuras 10 y 11.

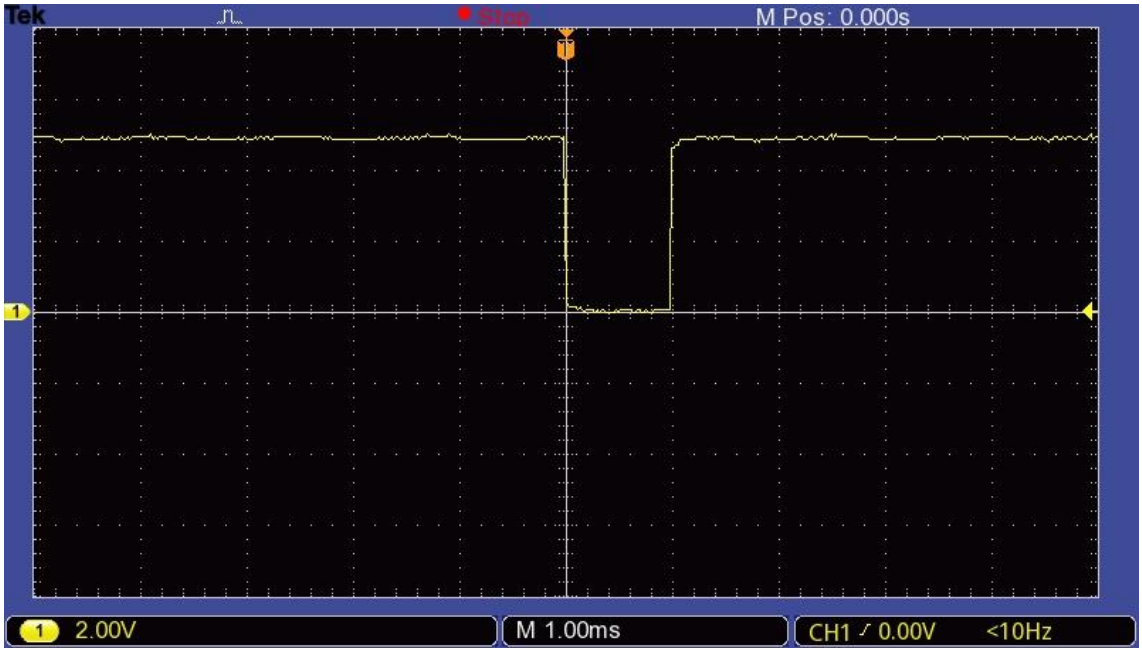


Figura 10. El periodo de la rutina de interrupción es de aproximadamente 1 [ms]

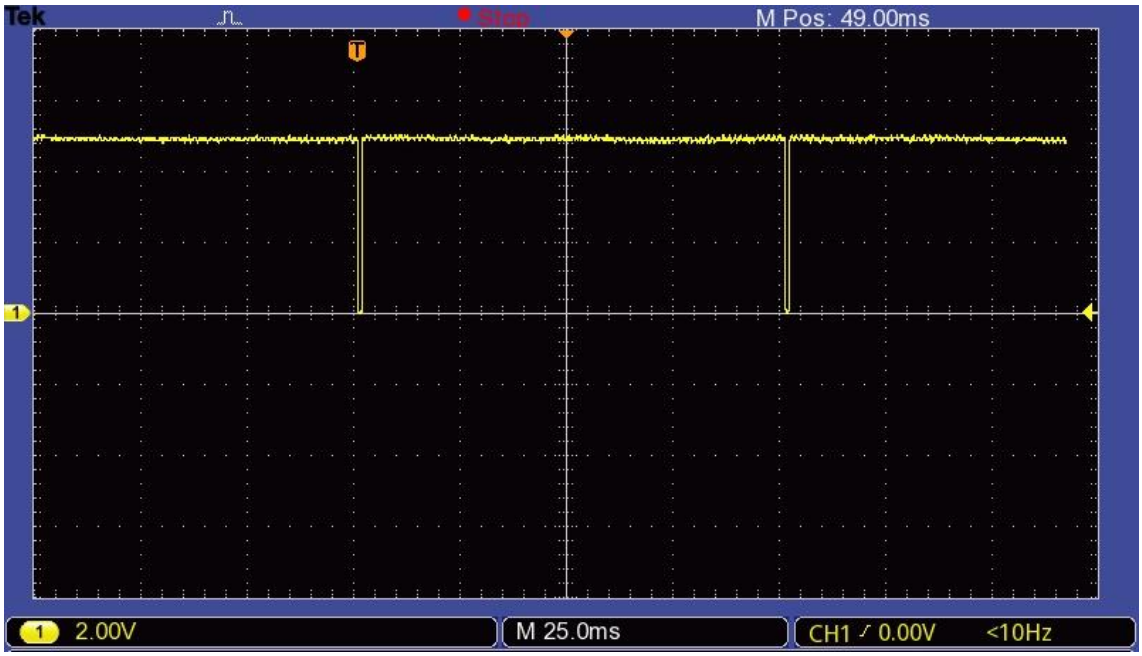


Figura 11. El periodo del programa es de aproximadamente 100 [ms]

Tanto el tiempo de adquisición como el reloj utilizado para la conversión del convertidor analógico-digital se configura por medio del registro ADCON2, el establecimiento de puertos como analógicos se realiza en el registro ADCON1,

y la habilitación y la selección del canal a utilizar para la conversión se realiza con el registro ADCON0. Por último, para habilitar la interrupción asociada con la conversión, se debe poner en cero el bit ADIF del registro PIR1 y en uno los bits ADIE y GIE de los registros PIE1 e INTCON respectivamente. El resultado de la conversión se almacena en el registro ADRESH:ADRESL.

2 Software

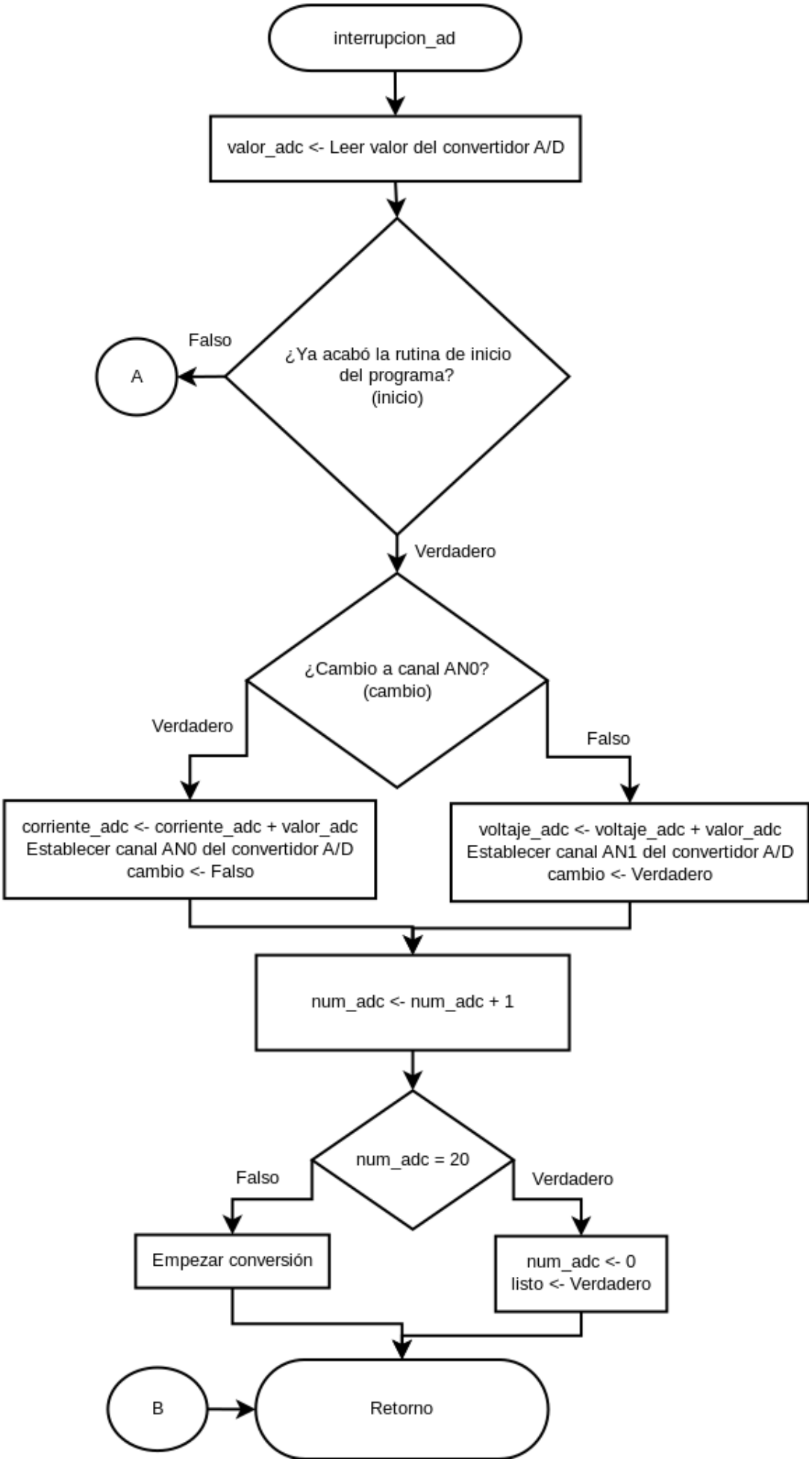
2.1 Programación del microcontrolador

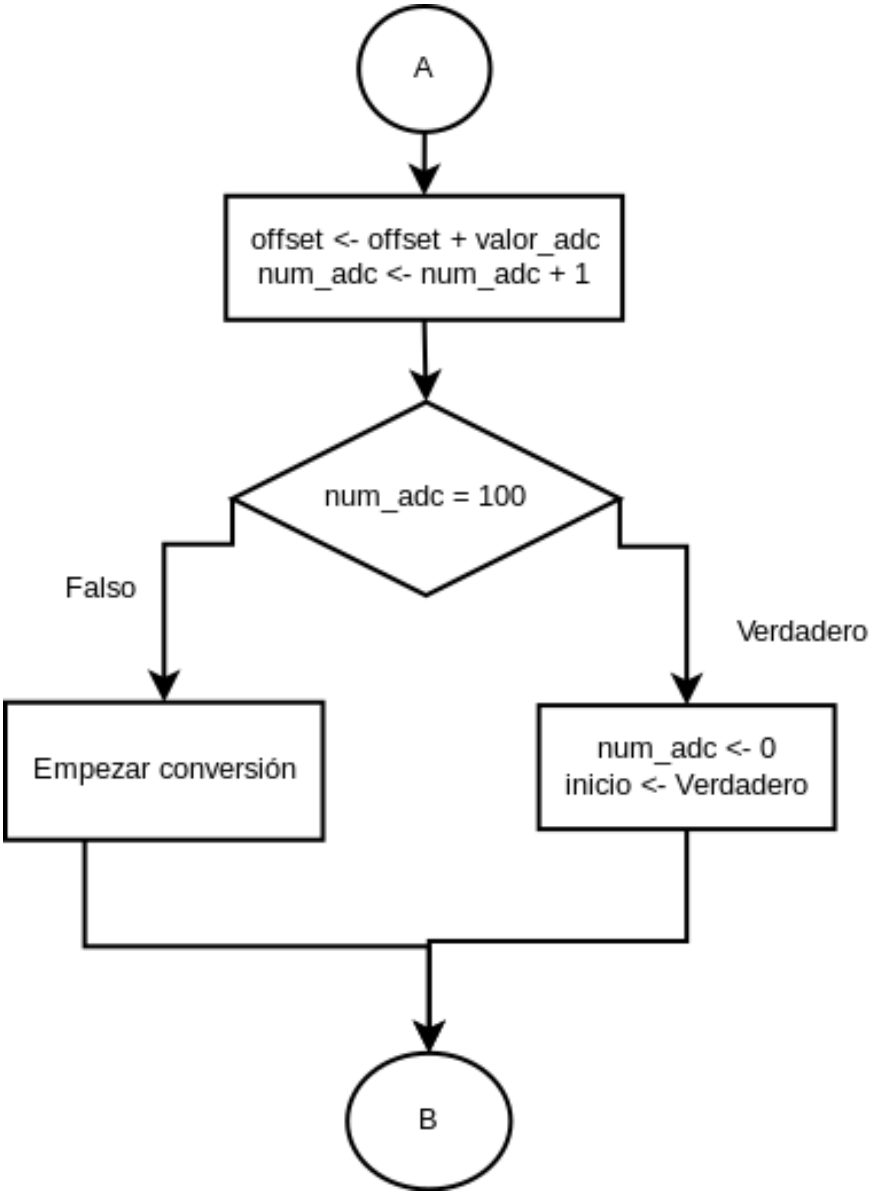
El programa del microcontrolador se hizo con el lenguaje C, utilizando las bibliotecas y el compilador de CCS para PICs. El código se muestra en su totalidad en el apéndice C y a continuación mostraremos sus elementos principales.

Interrupciones

El PIC18F2458 cuenta con 19 fuentes de interrupción configurables, una de las cuales está asociada con la terminación del proceso de conversión del convertidor analógico digital. Para el caso de esta tesis, se implementó una rutina de interrupción con el identificador *interrupcion_ad* para obtener las muestras de corriente y voltaje con las cuales calcular la potencia instantánea en el proceso principal. Para asegurar la consistencia en los resultados, se toman varias muestras durante 1 [ms] y posteriormente se promedian.

Al inicio del programa, el microcontrolador debe obtener un valor de voltaje que equivalga a aquél devuelto por el sensor ante una corriente de 0 [A] (representado en la ecuación 1.17 como V_{outAmp_offset}), que es usado para el posterior cálculo de otros valores de corriente, por lo que la rutina de interrupción obtiene 100 muestras de este valor y las guarda en una variable *offset* para después calcular su promedio en el proceso principal. Una vez realizado esto, la rutina de interrupción procede a obtener de forma continua 10 muestras por cada canal de forma alternada durante el intervalo de tiempo determinado. El diagrama de flujo que describe este proceso se muestra a continuación:


interrupcion_ad
valor_adc <- Leer valor del convertidor A/D
¿Ya acabó la rutina de inicio del programa? (inicio)
Falso
A
Verdadero
¿Cambio a canal AN0? (cambio)
Verdadero
Falso
corriente_adc <- corriente_adc + valor_adc
Establecer canal AN0 del convertidor A/D
cambio <- Falso
voltaje_adc <- voltaje_adc + valor_adc
Establecer canal AN1 del convertidor A/D
cambio <- Verdadero
num_adc <- num_adc + 1
num_adc = 20
Falso
Verdadero
Empezar conversión
num_adc <- 0
listo <- Verdadero
B
Retorno



En esta rutina de interrupción, *valor_adc* es una variable local y las demás son globales o estáticas debido a que estas últimas deben conservar su valor para invocaciones posteriores de la misma rutina y para su procesamiento en la rutina principal; todas las variables son inicializadas en cero antes de cada ciclo de obtención de muestras. La variable *listo* se utiliza como bandera para indicar a la rutina principal cuando ésta ya puede procesar las muestras obtenidas.

Cálculo de potencia

Una vez obtenidas las muestras suficientes, se utilizan dos rutinas para calcular el valor real del voltaje y la corriente; estas rutinas reciben como argumento el

promedio de los valores devueltos por el convertidor para cada medición en el intervalo de tiempo establecido.

Para obtener el valor real de voltaje *V*, se multiplica el valor promedio devuelto por el convertidor $V_{A/D}$, por el voltaje del bit menos significativo V_{LSB} definido en la ecuación 1.19. A este resultado se le suma la mitad del V_{LSB}, debido a la relación que plantea la función de entrada/salida del convertidor mostrada en la figura 12, para posteriormente sumarle el error absoluto e_{abs} calculado en la sección de acondicionamiento de señales y multiplicar el resultado por la constante *K* definida en la ecuación 1.2 que nos relaciona los niveles de voltaje:

$$V = ((V_{A/D} + 0.5)V_{LSB} + e_{abs}) \cdot K \qquad (2.1)$$

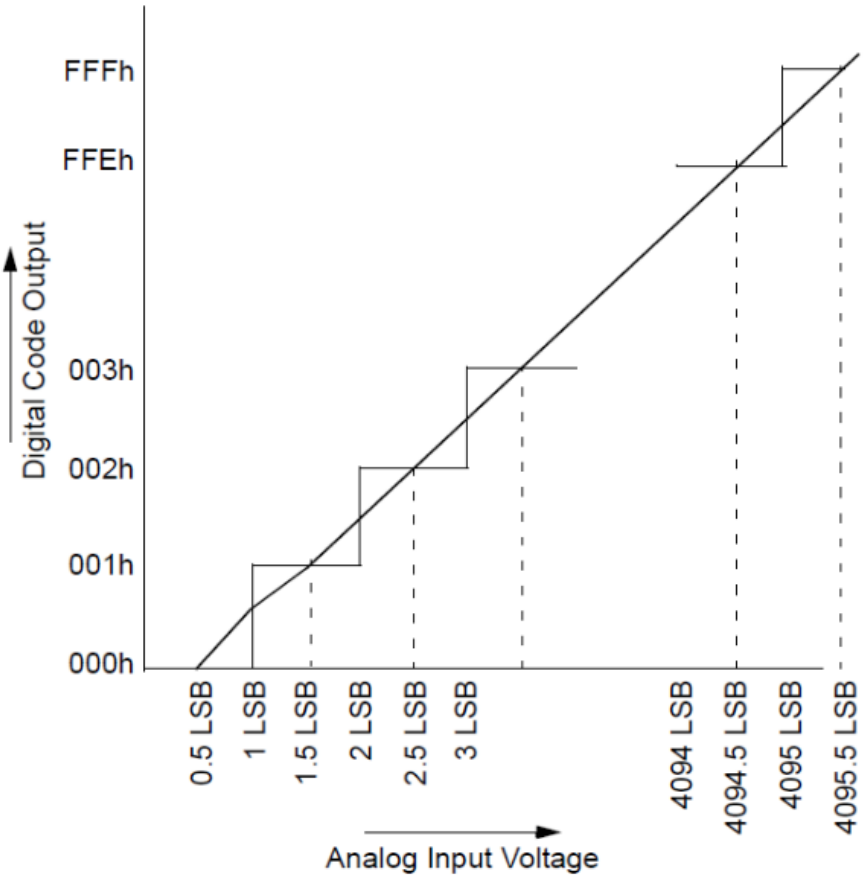


Figura 12. Función de entrada/salida del convertidor A/D

Fuente: Microchip Technology Inc. (2009) ***PIC18F2458/2553/4458/4553: Data Sheet****. EE. UU., p. 25.*

Para obtener el valor real de intensidad de corriente *I*, se utiliza una fórmula derivada de la ecuación 1.17 y parecida, hasta cierto punto, a la definida

anteriormente para el voltaje. Es de resaltar que se le debe de restar al valor devuelto por el convertidor $V_{A/D}$ aquél correspondiente a cuando el sensor registra una corriente de 0 [A], V_{A/D_offset}, por las razones que se mencionaron en la sección de acondicionamiento de señales. La fórmula es como sigue:

$$I = \frac{63}{122 \cdot sensibilidad}(V_{A/D} - V_{A/D\_offset})V_{LSB} \tag{2.2}$$

Posteriormente, se debe calcular la potencia instantánea, multiplicando el valor de corriente por el de voltaje, con el fin de obtener después la energía consumida, al que se llega multiplicando la potencia resultante por el periodo de tiempo al que corresponde la medición $T_{medición}$. Por último, se convierte el valor resultante de joules a kilojoules para evitar un posible error de sobreflujo aritmético que impida mantener el sistema en ejecución durante un largo periodo. La fórmula para la potencia instantánea P y la energía empleada en kilojoules E_k se muestran a continuación:

$$P = I \cdot V \tag{2.3}$$

$$E_k = \frac{P \cdot T_{medición}}{1000} \tag{2.4}$$

Protocolo de transmisión

Con el fin de que pueda establecerse una relación de los datos a través del tiempo, cada dato es enviado junto con un contador y el periodo de medición al que corresponde, que al multiplicarlos, ubica los datos en un momento específico en relación con el inicio del monitoreo de la energía consumida por el motor. El formato de los datos que se envían por los módulos de comunicación es el siguiente (donde el carácter '#' indica la presencia de un número):

. # P # . # V # . # I # T # C

En el formato anterior, el número racional que precede al carácter 'P' corresponde a la energía, aquél que precede a 'V', es el voltaje, y el número que precede a 'I' corresponde a la intensidad de corriente. El número entero que antecede al carácter 'T' es el periodo de tiempo durante el cual se hacen las mediciones mientras que aquél que precede a 'C' corresponde al contador de muestra.

2.2 Programación de aplicación para sistema operativo Android

Android es un sistema operativo para smartphones que se distingue por su código abierto y por la gran variedad de dispositivos que soporta. Se desarrolló una aplicación en Java para este sistema operativo con el fin de poder visualizar los resultados de la medición de energía en un smartphone y no estar sujetos a los límites en la portabilidad del hardware que presentan las computadoras tradicionales, obteniendo, para esto, la información correspondiente por medio del módulo Bluetooth instalado en la estación retransmisora.

La aplicación para Android primero revisa si hay dispositivos Bluetooth previamente acoplados y los despliega en una lista, en caso de no encontrar al dispositivo encargado de enviar los datos de energía, se puede proceder a escanear el área y desplegar una nueva lista. Una vez conectado al dispositivo Bluetooth emisor, se reciben los datos y se muestra una gráfica que va actualizándose conforme se obtiene más información. La aplicación consiste de dos actividades, las cuales serán explicadas a continuación, el código en su totalidad se encuentra en el anexo E.

Actividades

La actividad con la que inicia la aplicación se llama *MainActivity*, la cual se encarga de manejar la interfaz mostrada en la figura 13 que permite al usuario conectarse con un dispositivo Bluetooth, obteniendo previamente los permisos necesarios para esta acción. Se puede desglosar en los siguientes elementos interactivos:

Elemento	Descripción
Despliegue de dispositivos Bluetooth encontrados 	La sección debajo del texto "Dispositivos encontrados:", en la pantalla de inicio, corresponde a un elemento de tipo Spinner, también llamado control de número, al cual nos referimos con el identificador *dispositivos*. Al tocar este elemento se muestra en una lista el nombre y dirección de los dispositivos Bluetooth encontrados, de los cuales, aquél seleccionado por el usuario es el que termina siendo desplegado en la pantalla de la forma como se muestra en la imagen de la izquierda. Los datos de los dispositivos Bluetooth se guardan en una lista *bld* de tipo BluetoothDevice, la cual es asociada al Spinner *dispositivos* por medio de un adaptador, *adapter*, que maneja elementos de tipo cadena de caracteres.
Botón para realizar escaneo 	Al seleccionar este botón con el identificador *bt_escaneo*, se invoca una rutina handler que cancela cualquier proceso de descubrimiento que se haya estado ejecutando anteriormente por el adaptador Bluetooth, *mBluetoothAdapter*, limpia los datos presentes en el Spinner *dispositivos* e intenta iniciar un nuevo proceso de descubrimiento de dispositivos

	Bluetooth. El código de esta rutina se muestra a continuación:
	
	`bt_escaneo.setOnClickListener(new View.OnClickListener() {`
	
	`@Override`
	`public void onClick(View v) {`
	
	`mBluetoothAdapter.cancelDiscovery();`
	`adapter.clear();`
	`bld.clear();`
	`mBluetoothAdapter.startDiscovery();`
	`}`
	`});`

Botón para conectarse con dispositivo 	Al seleccionar este botón, *bt_conectar*, se invoca una rutina handler en la cual, primero, se obtiene la posición del dato escogido en el Spinner *dispositivos*, con el fin de usarlo de índice y obtener el dato correspondiente en la lista asociada *bld*; y segundo, se inicia la conexión con el dispositivo seleccionado, invocando una instancia de la clase *Conectar*, la cual se ejecuta de forma asíncrona al proceso principal. Esta instancia crea un BluetoothSocket con un identificador universal único (UUID), el cual se utiliza para conectarse al dispositivo elegido y crear un canal de comunicación a través del cual se pueden enviar y recibir datos; las instancias de la clase *Conectar* se encargan también de llamar a la siguiente actividad, *Graficar*,

	pasando como argumento el BluetoothSocket generado. La rutina handler del botón bt_conectar se muestra a continuación: *bt_conectar.setOnClickListener(new View.OnClickListener() {* *@Override* *public void onClick(View v) {* *if(adapter.isEmpty())* *return;* *int pos = dispositivos.getSelectedItemPosition();* *Conectar cnc = new Conectar(bld.get(pos));* *cnc.execute();* *}* *});*

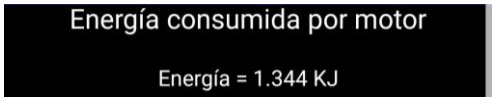
Figura 13. Interfaz de usuario de MainActivity

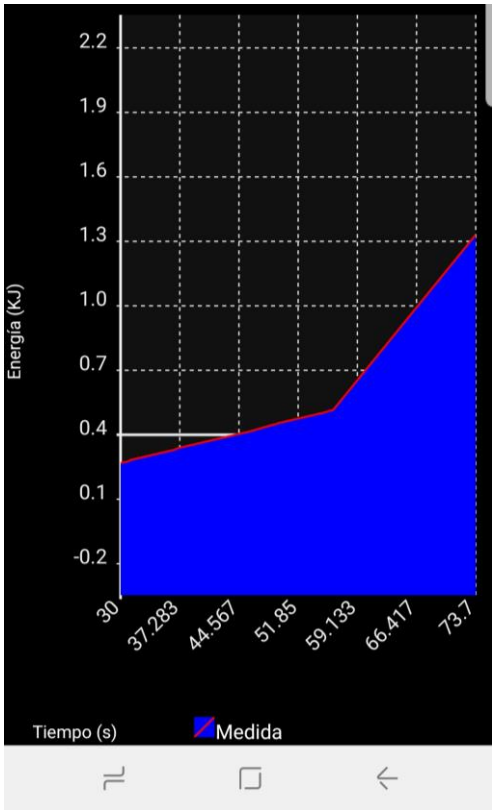
Para poder obtener la lista de dispositivos Bluetooth previamente detectados por el proceso asíncrono de descubrimiento, es necesario registrar un BroadcastReceiver, un tipo de handler que escucha a eventos de todo tipo, y limitar su invocación de forma que solo responda a la terminación de este proceso de descubrimiento (Android Open Source Project,). La rutina dentro del BroadcastReceiver se ejecuta por cada dispositivo encontrado, en la cual se extraen los datos del dispositivo y se insertan en la lista *bld* y el adaptador *adapter*.

La actividad que recibe datos por medio de Bluetooth y los despliega de forma gráfica se llama *Graficar*, su interfaz de usuario se muestra en la figura 14. Esta actividad presenta los siguientes elementos gráficos:

Elemento	Descripción

Barra de aplicación con botones
 
Lista con posibles tipos de mediciones a graficar
 
	En la barra de aplicación, se incluyen dos botones, uno con el nombre de "RESET" y otro que consiste de tres puntos acomodados de forma vertical. Al seleccionar el botón "RESET", se invoca una rutina handler *resetear* que únicamente vuelve a desplegar la gráfica con los elementos guardados anteriormente, pero de forma que ésta queda centrada en el eje *y* de acuerdo con el último valor obtenido, con lo cual se puede volver a visualizar la gráfica en su totalidad. El código de esta rutina se muestra a continuación:
	
	`public void resetear(){`
	`  int sz = serie1.size();`
	`  //se obtiene de la serie de datos el último valor correspondiente al eje y`
	`  double y = serie1.getY(sz - 1).doubleValue();`
	`  grafica.centerOnRangeOrigin(y);`
	`}`
	Al seleccionar el botón con los tres puntos, se despliega una lista con los tres posibles tipos de mediciones que se pueden graficar, por lo que de acuerdo con la que se elija, cambiará lo que se muestra en las secciones debajo de esta barra.

Vista que muestra el último dato recibido 	En la sección inmediatamente debajo de la barra de aplicación, se actualiza continuamente un elemento de texto que muestra el valor del último dato

	recibido, de acuerdo con el tipo de medición seleccionado.
Gráfica  	Para la sección con la gráfica se utilizó una biblioteca externa llamada “Androidplot” que permite crear fácilmente vistas dinámicas con este tipo de elementos y modificar sus parámetros, como tipo de línea, límites de los ejes, etiquetas desplegadas, colores utilizados, y comportamiento del zoom, entre otras cosas. En este caso se utilizó una gráfica de línea para hacer más sencilla la visualización de los datos en función del tiempo, con el eje *x* siendo el tiempo en segundos y el eje *y* una de las tres mediciones que se pueden elegir. Los datos que se muestran son solo una cantidad limitada de los más recientemente recibidos, para evitar agotar los recursos del smartphone.

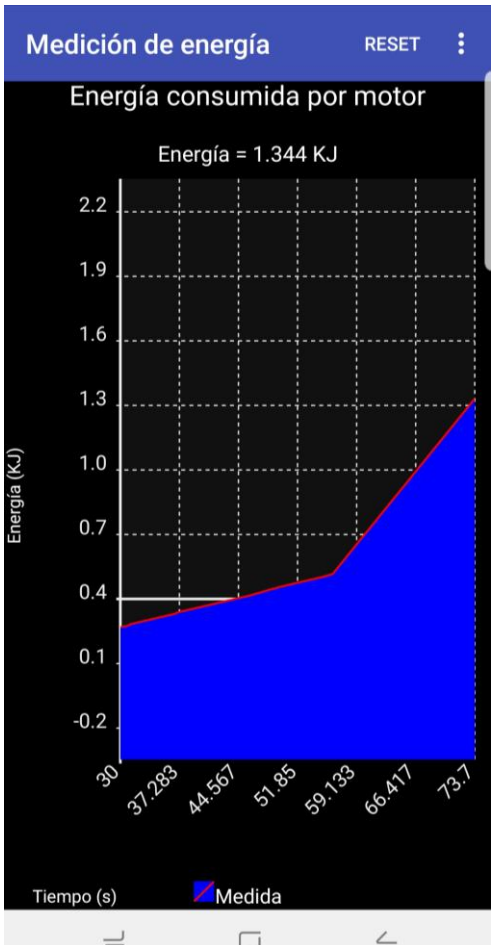


Figura 14. Interfaz de usuario de Graficar

Para desplegar los datos en la gráfica, es necesario contar con una instancia de una clase que se ejecute de forma asíncrona y se encargue de recibir los datos y de implementar los métodos para desplegar estos últimos. De acuerdo con la implementación de Androidplot, se requieren dos rutinas, *getX* y *getY*, para desplegar los datos recibidos. Cada una de estas rutinas debe revisar que el índice del dato que se pide no sea mayor a nuestro límite de datos. *getX* despliega los datos relacionados con el tiempo, almacenados en la lista *tiempo*, y *getY* aquellos almacenados en la lista *datos*, que de acuerdo con lo que el usuario haya elegido, corresponderán a una de las tres mediciones posibles:

```
public Number getX(int serie, int indice){
   if(indice >= TAM_MUESTRA+1)
      throw new IllegalArgumentException();
   return Double.parseDouble(tiempo.get(indice));
}

public Number getY(int serie, int indice){
   if(indice >= TAM_MUESTRA+1)
      throw new IllegalArgumentException();
   return Double.parseDouble(datos.get(indice));
}
```

Para entender exactamente cómo se hace el procesamiento de los datos, es necesario analizar el código ejecutado por las instancias de la clase con el identificador *Conectado*, tal como se hará a continuación.

Teniendo ya una conexión Bluetooth con la estación retransmisora, representada en el código por el identificador *mmInstream*, se obtiene del flujo de entrada una línea de caracteres *line* terminada por un salto de línea y se procede a revisar si no hubo algún error en la transmisión, haciendo para esto una prueba con una expresión regular tal como se muestra en el siguiente código:

```
if ((line = mmInStream.readLine()) != null
   && line.matches("^[0-9]+\\.[0-9]{3}P[0-9]+\\.[0-9]{3}V[0-9]+\\.[0-9]{3}I[0-
9]+T[0-9]+C"))
```

Posteriormente se extraen los datos de la línea de caracteres y se agregan a las listas correspondientes de *energía*, *voltaje*, *corriente*, *periodo* y *tiempo*. Es de notar que el valor del contador se debe multiplicar por el periodo y convertir a segundos para obtener el tiempo exacto al cual corresponde la medición. Debido a que el periodo es enviado en milisegundos, se debe dividir entre *BASE_TIEMPO*, que es equivalente a 1000, para convertirlo a segundos:

```
energia.add(line.substring(0, line.indexOf('P')));
voltaje.add(line.substring(line.indexOf('P') + 1, line.indexOf('V')));
corriente.add(line.substring(line.indexOf('V') + 1, line.indexOf('I')));
periodo.add(line.substring(line.indexOf('I') + 1, line.indexOf('T')));
tiempo.add(Double.toString(Double.parseDouble(line.substring(line.indexOf('T')
+ 1, line.indexOf('C'))) * Double.parseDouble(periodo.get(periodo.size() - 1)) /
BASE_TIEMPO));
```

Para evitar llenar la memoria del smartphone con los datos recibidos, se establece un contador para el número de datos almacenados en las listas, *TAM_MUESTRA*, y un límite para este contador, *LIMITE_TAM_MUESTRA*, de forma que una vez que se empieza a rebasar, se procede eliminando los datos más viejos de las listas, aquellos en la posición cero:

```
if (TAM_MUESTRA < LIMITE_TAM_MUESTRA)
   TAM_MUESTRA++;
else {
   tiempo.remove(0);
   energia.remove(0);
   voltaje.remove(0);
   corriente.remove(0);
   periodo.remove(0);
}
```

Para almacenar los datos recibidos, se deben pedir permisos para crear un archivo. Este archivo es de tipo CSV y lleva el nombre de “medicion_” concatenado con el día y tiempo en el que se crea, y se guarda en lo que Android considera el almacenamiento externo del smartphone. Entonces, si se obtienen los permisos necesarios, se escriben los valores recibidos en el archivo abierto, separados por una coma cada uno, de forma que se respeta el formato CSV:

```
if (permiso) //si tiene el permiso para escribir en un archivo
   writer.write(energia.get(energia.size() - 1) + "," + voltaje.get(voltaje.size() - 1)
+ "," + corriente.get(corriente.size() - 1) + "," + periodo.get(periodo.size() - 1) +
"," + tiempo.get(tiempo.size() - 1) + "\n");
```

En la misma rutina donde se ejecuta todo lo anterior, se coloca también el código que reacciona ante el cambio en la selección de la medición a trazar en una gráfica. Esta selección es representada por la variable *opción_gráfica*, que de acuerdo con su valor, cambia la lista a la que se refiere *datos* y una variable de tipo cadena de caracteres, *str*, que tiene el elemento de texto con la última medición recibida. Este último texto se despliega debajo de la barra de la aplicación, por lo que se debe enviar al proceso principal para que sea capturado por un handler y realice el cambio pertinente:

```
switch (opcion_grafica){
   case 1:
      datos = energia;
      str = "Energía = " + datos.get(datos.size() -1) + " KJ";
      break;
   case 2:
      datos = voltaje;
      str = "Voltaje = " + datos.get(datos.size() - 1) + " V";
      break;
   case 3:
      datos = corriente;
      str = "Corriente = " + datos.get(datos.size() -1) + " A";
      break;
}
mHandler.obtainMessage(MESSAGE_READ, str.length(), -1,
str).sendToTarget();
```

Por último, si cambia la medición a desplegarse, se debe también cambiar el texto y límites asociados al eje *y*, y redibujar la gráfica. Para saber si hubo un cambio de selección se utiliza la variable *opcion_ant*, que almacena el valor de la opción elegida anteriormente por el usuario, y se compara con la selección actual:

```
if(opcion_ant != opcion_grafica) {
   if(opcion_ant != 0) { //opcion_ant es cero cuando apenas inicia la actividad
      grafica.clear();
      TextLabelWidget txt = grafica.getRangeTitle();
      txt.setText(rangos[opcion_grafica - 1]); //en rangos están los textos
correspondientes a la opción elegida
      grafica.setRangeTitle(txt);
      grafica.addSeries(serie1, serie1Formato);
      grafica.redraw();
   }
   grafica.setRangeLowerBoundary(Double.parseDouble(datos.get(0))/2.0,
BoundaryMode.FIXED);
   grafica.setRangeUpperBoundary(Double.parseDouble(datos.get(datos.size()-
1))*2.0, BoundaryMode.FIXED);
   opcion_ant = opcion_grafica;
}
```

2.3 Programación de aplicación para Windows y GNU/Linux

Se realizó un programa para desplegar en una computadora una gráfica con los datos recibidos por medio de la comunicación serie con el Arduino de la estación

retransmisora, y guardar estos datos en un archivo con formato CSV. Al igual que con la aplicación para Android, se da la opción de visualizar la energía consumida, la diferencia de potencial y la intensidad de corriente. Se utilizó el lenguaje Python en su versión 3 para este programa con la finalidad de poder crear ejecutables tanto para Windows como para GNU/Linux sin necesidad de cambiar la estructura del mismo; para esto se utilizó el programa “pyinstaller”. Algunos de los módulos externos que se utilizaron fueron “matplotlib” para la gráfica, “pyserial” para obtener los datos por medio de comunicación serie con el Arduino y “tkinter” para el desplegado de los diálogos emergentes. El código se encuentra en su totalidad en el anexo D.

Este programa es bastante parecido al que se realizó para Android, pero contiene ciertos detalles en su implementación. La función que va actualizando la gráfica se llama *update*, y para que sea llamada de forma constante, se pasa como argumento a la función *FuncAnimation* de la biblioteca “matplotlib”. Se configura además para que la invocación de la rutina sea cada 1 [ms] y para que esta última reciba como argumentos las listas con los datos recibidos (*energía*, *voltaje*, *corriente*, *periodo*, *tiempo*):

```
ani = FuncAnimation(fig, update,
fargs=(energia,voltaje,corriente,periodo,tiempo), interval=1)
```

En la función *update*, primero se espera a recibir datos binarios por medio del puerto de comunicación serie, los cuales son transformados a cadenas de caracteres bajo el formato de codificación UTF-8:

```
try:
    ser_str = ser.readline().strip().decode('utf-8')
except:
    return
```

Posteriormente se revisa que cumplan con el formato establecido mediante la comparación con una expresión regular *pattern* igual a la definida en el código para el programa de Android. En caso exitoso, se utiliza la función *obtener_valor* para extraer los valores de las mediciones y ponerlos en las listas correspondientes. En el caso del tiempo, se debe hacer el cálculo al igual que en el programa de Android:

```
if(not pattern.fullmatch(ser_str)):
    return

ser_str = obtener_valor(ser_str, energia, "P")
ser_str = obtener_valor(ser_str, voltaje, "V")
ser_str = obtener_valor(ser_str, corriente, "I")
ser_str = obtener_valor(ser_str, periodo, "T")


ind = ser_str.find("C")
tiempo.append(float(ser_str[0:ind])*periodo[-1]/1000)
```

La función *obtener_valor* está definida como sigue:

```
def obtener_valor(ser_str, data, c):
    ind = ser_str.find(c)
    data.append(float(ser_str[0:ind]))
    return ser_str[ind+1:]
```

Los valores más recientes se escriben en el archivo CSV abierto anteriormente:

```
f.write(str(energia[-1]) + "," + str(voltaje[-1]) + "," + str(corriente[-1]) + "," +
str(periodo[-1]) + "," + str(tiempo[-1]) + "\n")
```

Las listas se limitan a contener solamente los 100 valores más recientes:

```
energia = energia[-100:]
tiempo = tiempo[-100:]
voltaje = voltaje[-100:]
corriente = corriente[-100:]
periodo = periodo[-100:]
```

Se dibuja la gráfica con los datos utilizando las funciones especializadas del módulo “matplotlib”, en donde cabe resaltar que para agilizar el proceso de visualización se dibujan solamente una quinta parte de los datos. De acuerdo con el valor que indique la variable *Index.opcion*, modificado por medio de los botones presentados en la interfaz de usuario, es el tipo de medición que se dibuja, por lo que se utiliza el diccionario *switch_func* para obtener la lista con los datos apropiados y *switch_axis* para obtener el texto asociado con el eje *y*. A este último texto se le concatena el valor del dato más reciente conforme a la medición elegida. Por último, de acuerdo con el tipo de medición que se grafica es la forma en la que se modifica el escalamiento en el eje *y*.

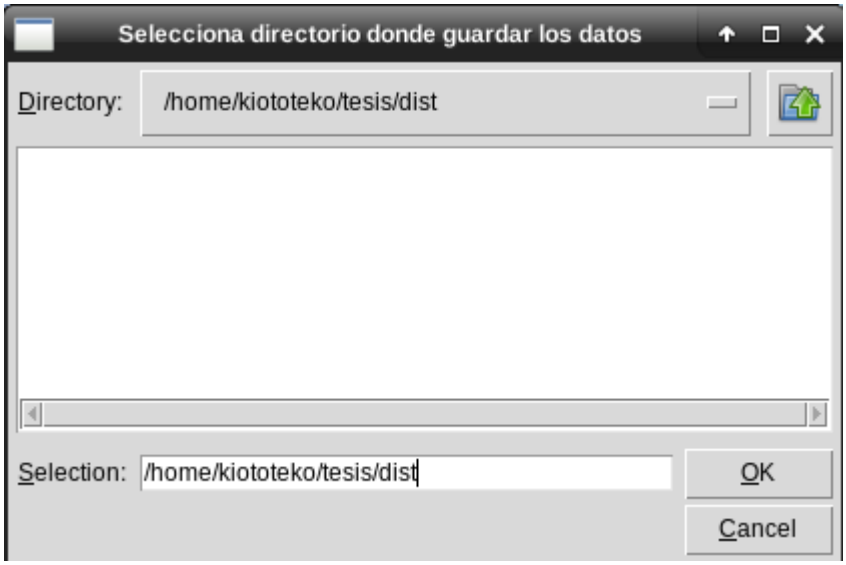
```
switch_axis = {
    1: "Energía (KJ)",
    2: "Voltaje (V)",
    3: "Intensidad de corriente (A)"
}

switch_func = {
    1: energia,
    2: voltaje,
    3: corriente
}

ax.clear()
xdata = switch_func.get(Index.opcion, 1)

ax.plot(tiempo[::10], xdata[::10])
ax.set_ylabel(switch_axis.get(Index.opcion, 1) + " [" + str(xdata[-1]) + "]")

ax.set_xlabel("Tiempo (s)")
ax.set_title("Consumo energético", loc='left')

if(Index.opcion > 1):
    ax.autoscale(axis='x')
    ax.set_ylim(min(xdata) - max(xdata), max(xdata)*2)
else:
    ax.autoscale(axis='both')

ln.set_data(tiempo, xdata)

return ln,
```

Interfaz gráfica

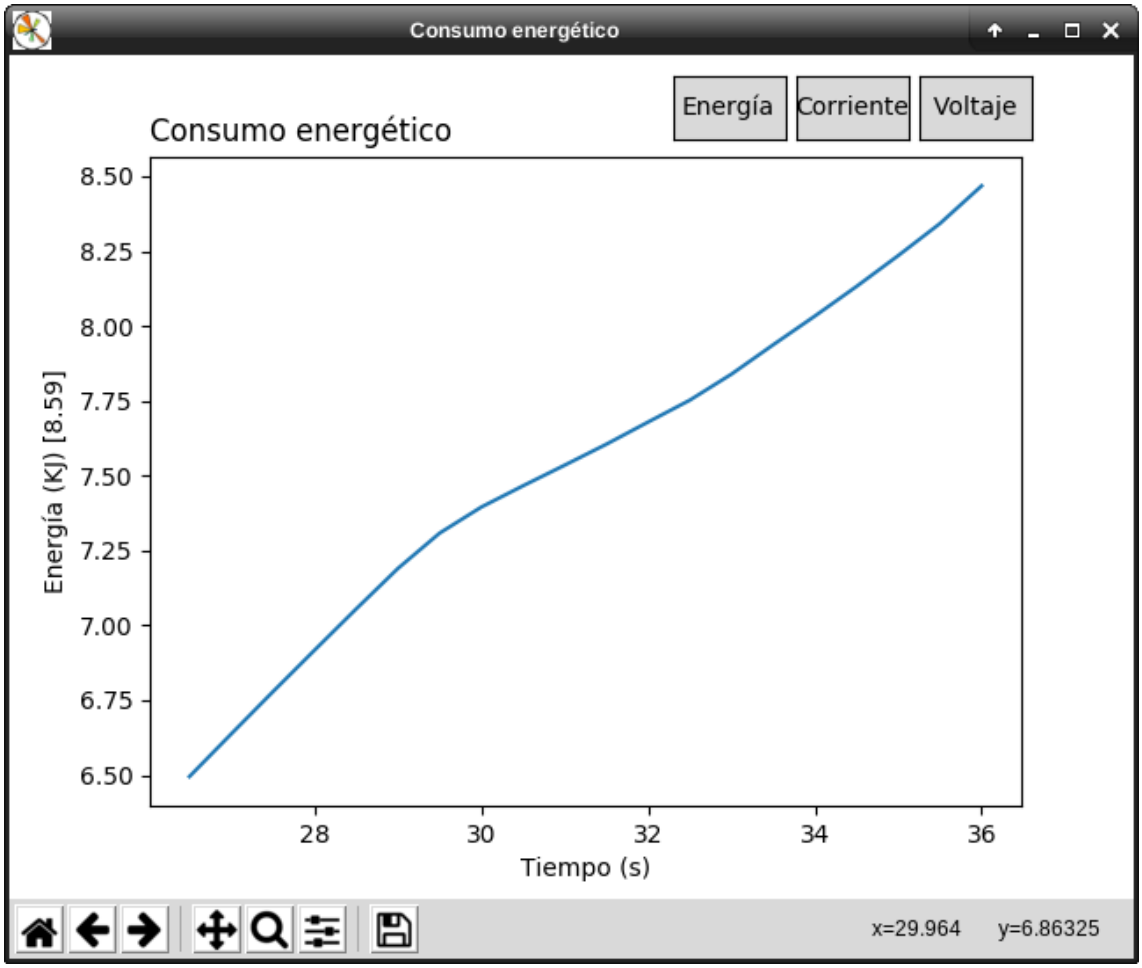
La interfaz gráfica del programa consiste de una secuencia de diálogos emergentes como se muestra a continuación.

La primera ventana que emerge da la opción de escoger el puerto serie del cual obtener los datos:

Al dar click en “Listo”, surge otro diálogo emergente que da la opción al usuario de elegir dónde guardar el archivo generado con los datos de las mediciones, que utiliza el mismo formato para el nombre como en la aplicación para Android:



Finalmente en una última ventana se dibuja la gráfica con los datos obtenidos, en donde por medio de los botones presentes en la parte superior de la ventana se puede cambiar el tipo de medición que se representa:



3 Pruebas y resultados

La tarjeta diseñada para el sistema de medición se muestra en la figura 15 y aquélla para el sistema de retransmisión se aprecia en la figura 16. Los circuitos de estas tarjetas fueron diseñados con el programa Proteus Design Suite de Labcenter Electronics, como se muestra en los apéndices A y B (Gallardo, 2015). Es de notar que debido a que el circuito de medición fue polarizado por un convertidor DC-DC, se tuvo que tener especial cuidado con su plano de tierra (Barrow, 2007). Tanto el sensor de corriente como el arreglo de resistencias divisor de voltaje, utilizado para la medición de la tensión eléctrica en el motor, se colocaron en la tarjeta con el circuito controlador del motor, y como ya se había mencionado anteriormente, el diseño de esta última tarjeta fue producto de otro trabajo.

Figura 15. Tarjeta impresa con el circuito de medición

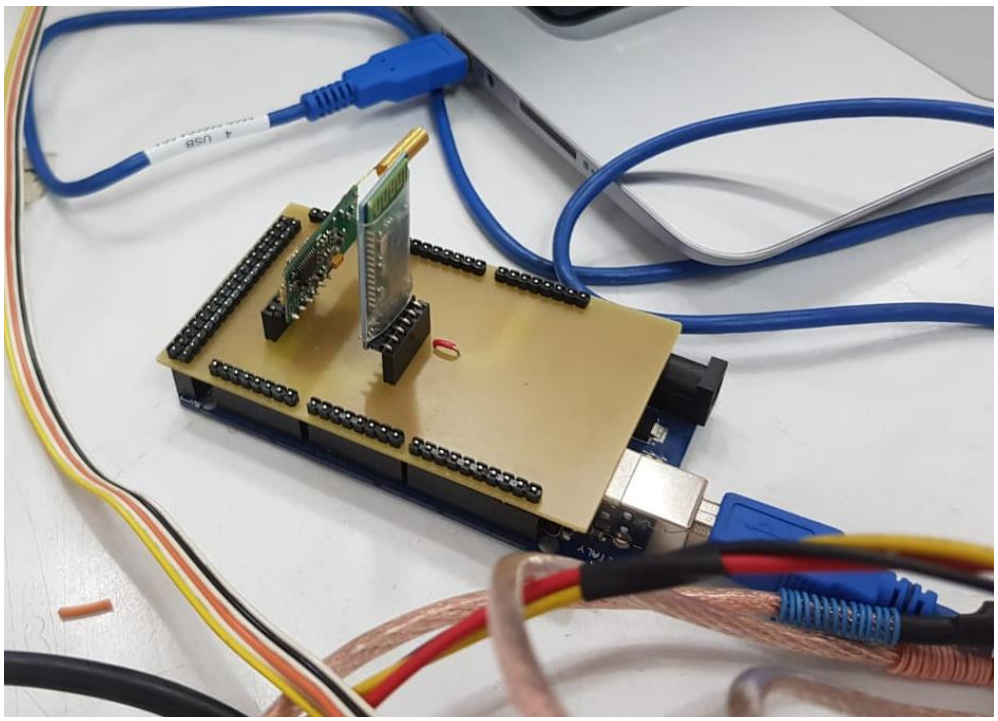


Figura 16. Tarjeta impresa con el circuito de retransmisión diseñada para ser usada como shield en un Arduino Mega 2560.

En la prueba de los dos sistemas, primero montamos la tarjeta con el circuito de medición en el circuito controlador del motor, siendo alimentado este último por unos bancos de baterías LiPo 2S. Posteriormente conectamos el Arduino Mega 2560 con el circuito de retransmisión a una computadora con sistema operativo GNU/Linux, con el fin de analizar los datos recibidos por medio del programa creado para este sistema. Al modular la corriente aplicada al motor a través de un dedal conectado al circuito controlador, tanto en la pantalla LCD como en la aplicación para la computadora se pudieron apreciar claramente los cambios en la energía consumida, como se muestra en la figura 17.

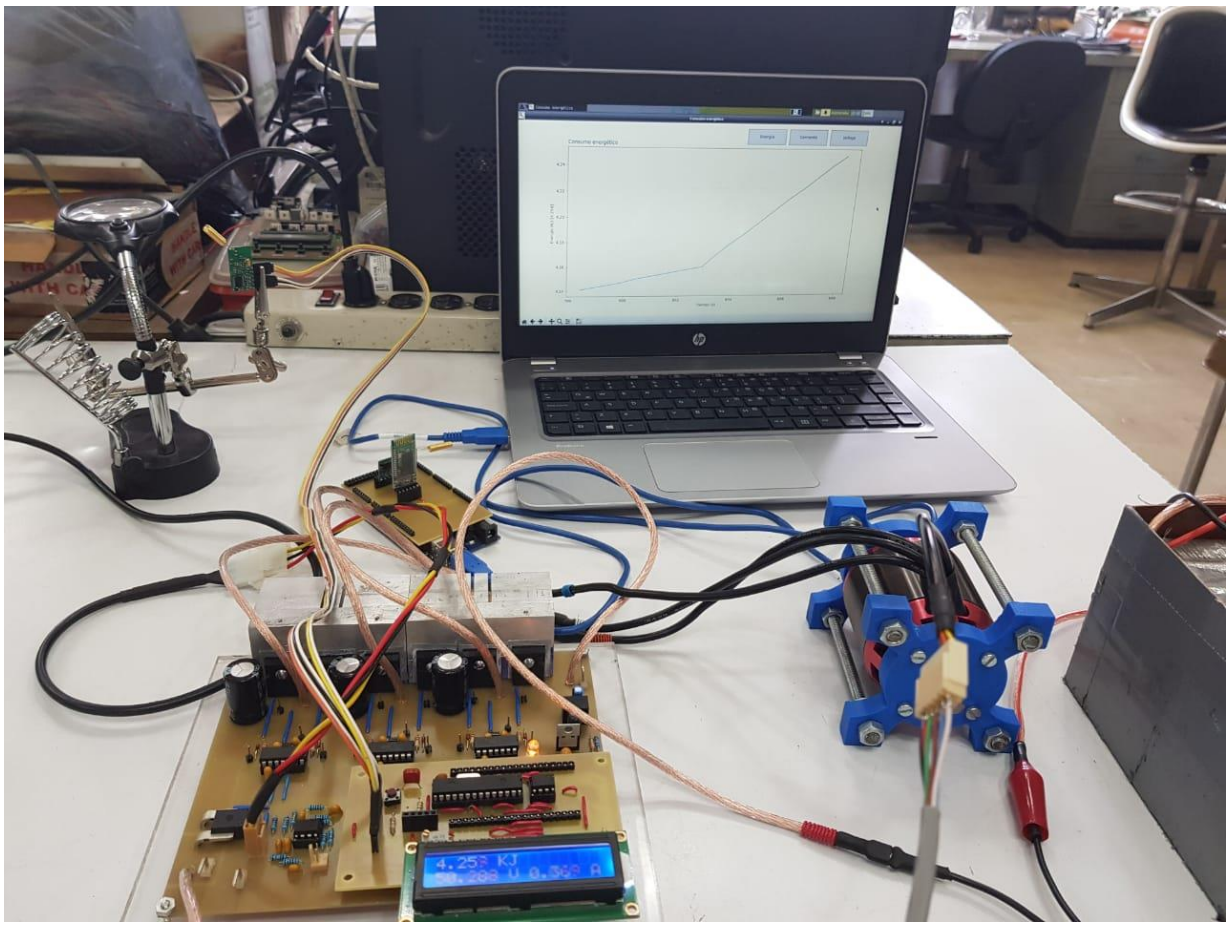


Figura 17. Prueba de los dos sistemas con el motor

Asimismo se probó la aplicación creada para Android en un smartphone Samsung Galaxy S8, y el programa para Windows, en una computadora con este sistema operativo, como se muestra en la figura 18 (las instrucciones para el uso de la aplicación creada para Android y para los sistemas operativos GNU/Linux y Windows se encuentran en los anexos F y G respectivamente).

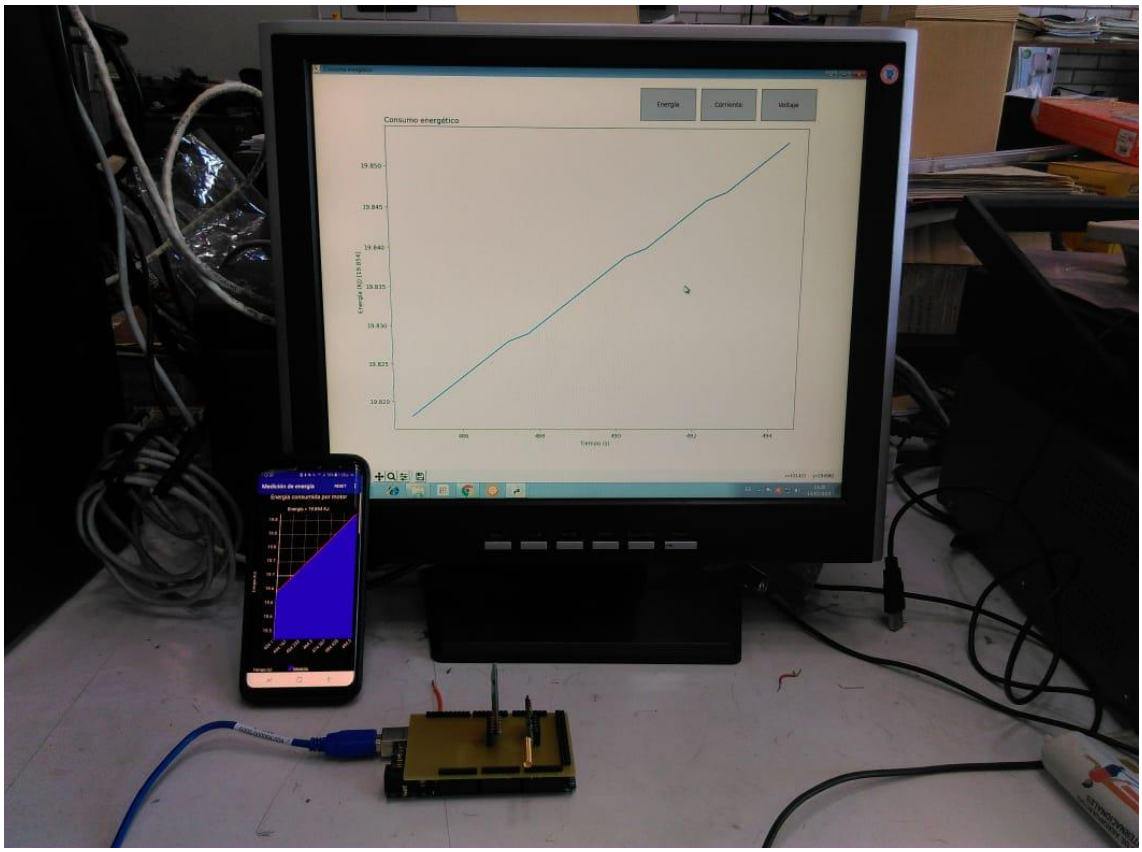


Figura 18. Prueba de los programas visualizadores de datos

Una vez hechas estas pruebas, se montó el circuito controlador junto con el circuito de medición en el auto de pruebas eléctrico como se muestra en la figura 19.

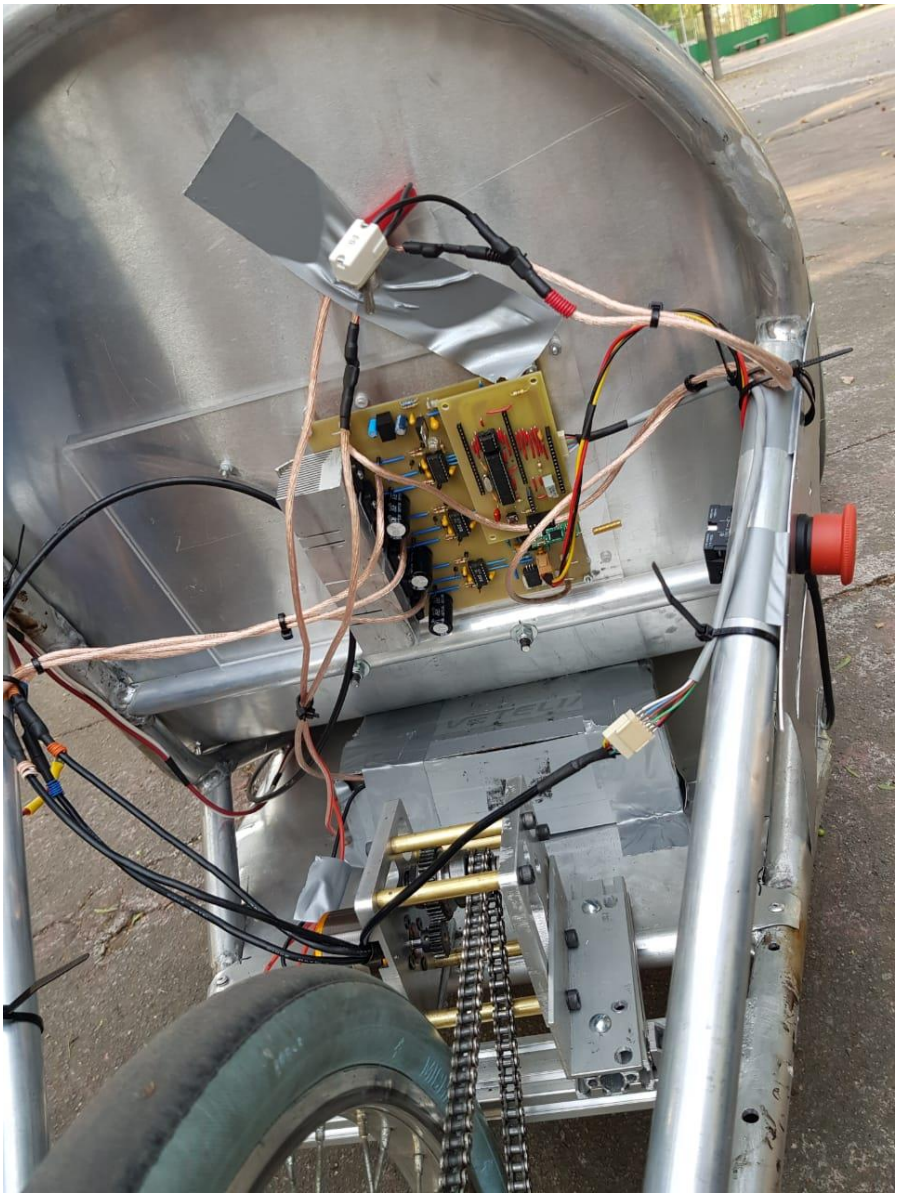


Figura 19. Parte trasera del auto de pruebas eléctrico

Por último, se probó el alcance de la transmisión de los datos. Para esto se contó con la ayuda de un conductor que alejaría el auto lo más posible del sistema de retransmisión, el cual estaba conectado a una computadora, con el fin de observar cualquier anomalía en la recepción de datos. Las pruebas se hicieron en un espacio que permitió al conductor alejarse hasta un máximo de 100 metros de la estación retransmisora sin que hubiera una pérdida en la transmisión de los datos más que cuando la antena del transmisor era bloqueada por la placa de metal del asiento del auto.

Figura 19. Auto de pruebas eléctrico junto con el sistema de retransmisión conectado a una computadora

Figura 20. Prueba de alcance con el auto de pruebas eléctrico

4 Conclusiones y trabajo futuro

En esta tesis se presentó el diseño e implementación de un sistema digital de medición de energía móvil basado en un microcontrolador, que permite transmitir los resultados de las mediciones a una estación retransmisora fija encargada a su vez de enviar estos datos a los dispositivos finales de los usuarios.

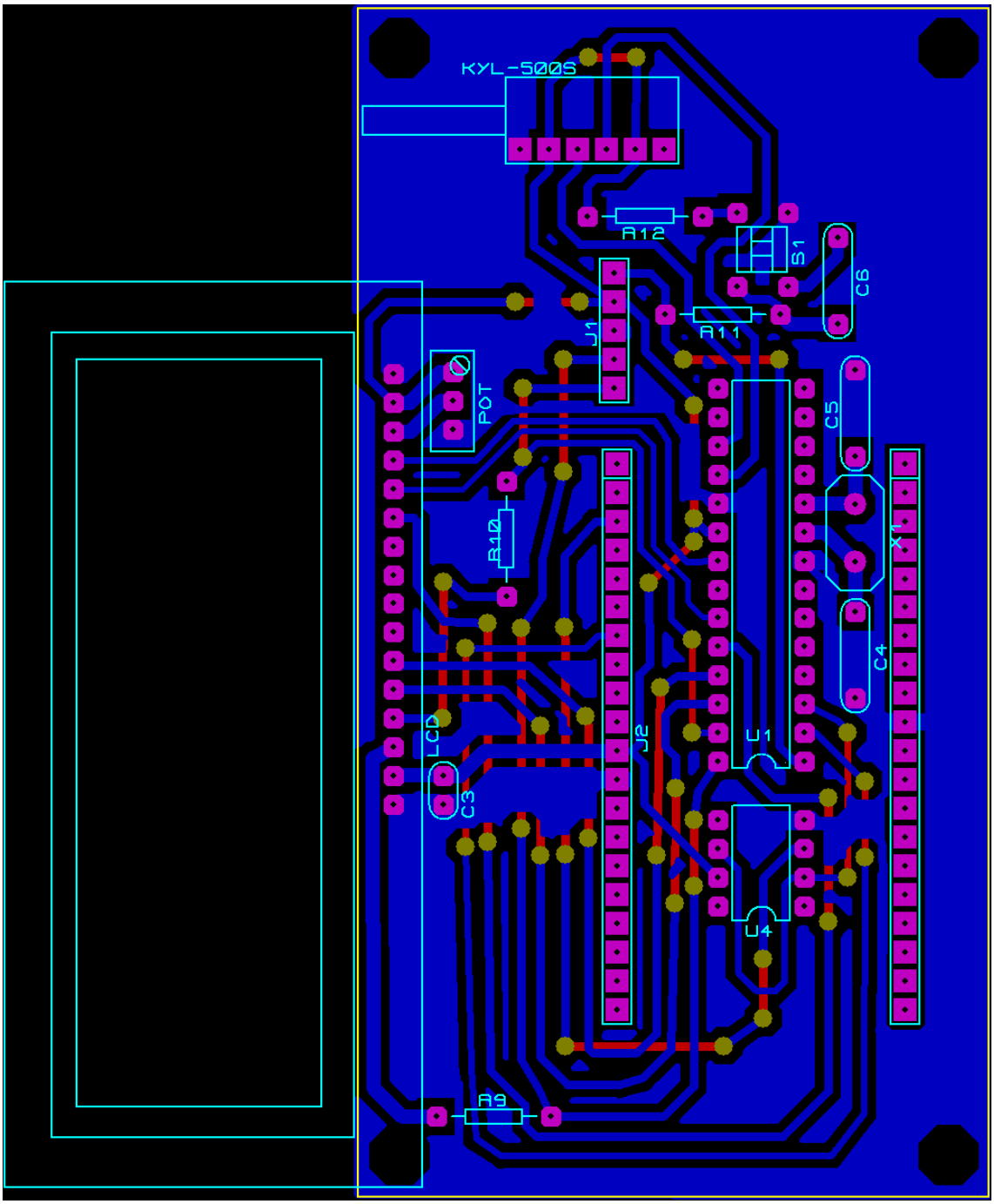
Para el cálculo de la energía, se detalló la forma en la que se tomaron las mediciones correspondientes de voltaje y corriente, desde el acondicionamiento de estas señales hasta su conversión a valores digitales, tomando en consideración para ello las relaciones de entrada/salida presentadas por los diferentes circuitos usados. Se tuvo un cuidado especial con el intervalo de tiempo al que correspondía el cálculo de la energía, haciendo un análisis minucioso del periodo utilizado en la ejecución de las distintas rutinas del programa. En cuanto a la transmisión de datos, se realizó una breve descripción de cada uno de los circuitos utilizados, haciendo énfasis en las tecnologías y protocolos en los cuales estaban basados. La presentación de los resultados del cálculo de la energía consumida se realizó de forma gráfica en cada una de las plataformas consideradas, generando unos programas especiales para tal objetivo.

El sistema implementado ha hecho posible realizar un monitoreo preciso y en tiempo real del consumo energético del motor de un auto eléctrico, lo cual ha dado lugar a que se desarrollen pruebas enfocadas a descubrir las condiciones que generan un consumo más eficiente y de esta forma mejorar el diseño del auto como consecuencia de los resultados de estas pruebas. El hecho de que las mediciones se transmitan a una estación fija mientras el auto está en movimiento hace del monitoreo un proceso sencillo y menos tedioso de lo que sería si se tuviera que estar siguiendo al auto como consecuencia de no tener un sistema inalámbrico de transmisión de gran alcance. Por último, el hecho de que las aplicaciones desarrolladas permitan visualizar los datos de forma gráfica y además almacenen los datos en un archivo, introduce un factor de flexibilidad en el procesamiento de los mismos que favorece a sus usuarios, siendo en este caso los académicos del grupo de electrónica del ICAT.

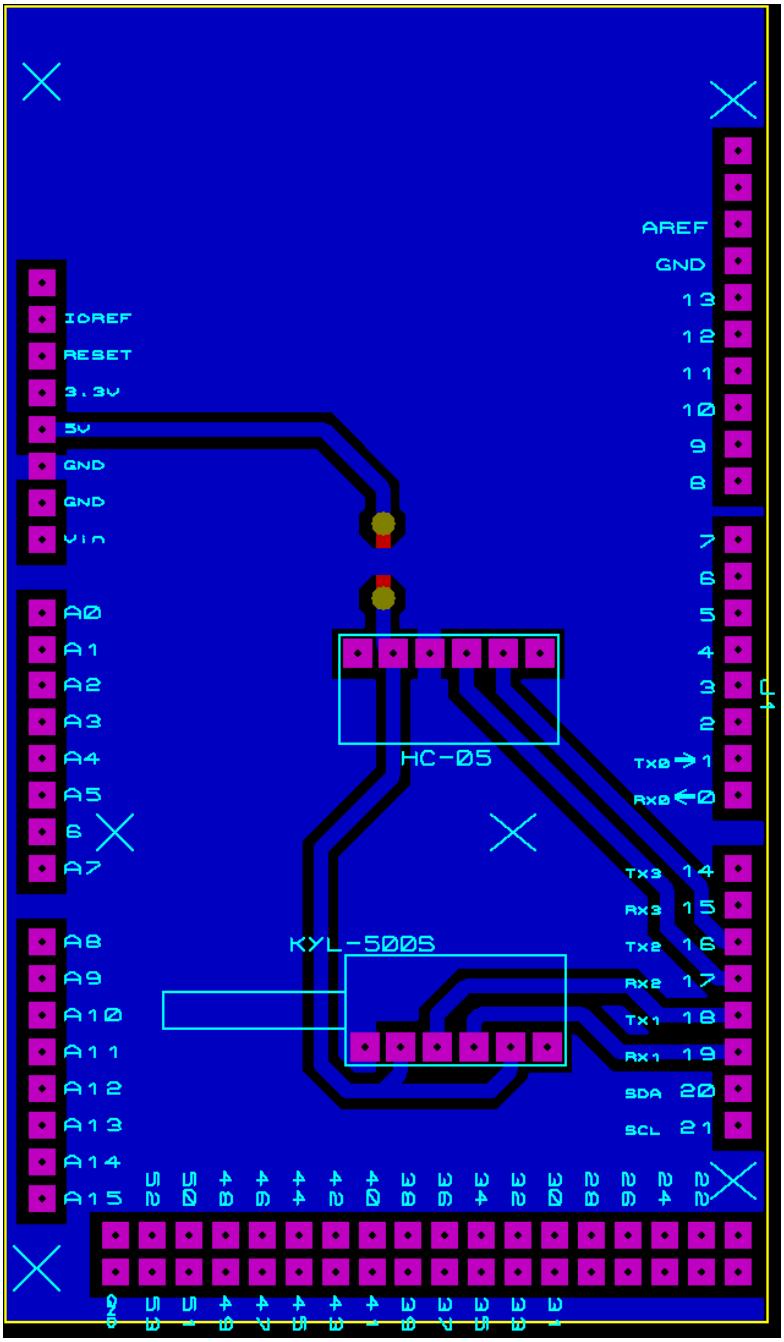
Para un trabajo futuro, la medición de energía podría hacerse más precisa mediante la reducción del tiempo utilizado para la conversión analógica-digital y la ejecución de las rutinas que procesan las mediciones, de forma que el cálculo de la potencia instantánea, y con ello el de la energía consumida, se haga de forma más frecuente y el resultado refleje más fielmente las variaciones que puedan existir. Podría hacerse también una mejor colocación del módulo transmisor en el auto de pruebas eléctrico de forma que los componentes del vehículo no obstruyan la transmisión de los datos.

5 Apéndices

Apéndice A. Circuito impreso del sistema de medición



Apéndice B. Circuito impreso de la estación retransmisora diseñado para servir como shield para un Arduino Mega 2560.



Apéndice C. Código utilizado en el programa del microcontrolador.

```
#include <18F2458.h>
#device ADC=12
#fuses HS, NOWDT, NOPROTECT, NOPUT, NOBROWNOUT, NOLVP,
NOCPD, CPUDIV1
#use delay(clock=20000000)
#use rs232(uart1, baud=9600)
#include <string.h>

#define LCD_RS_PIN      PIN_B0
#define LCD_RW_PIN      PIN_B2
#define LCD_ENABLE_PIN  PIN_B3
#define LCD_DATA4       PIN_B4
#define LCD_DATA5       PIN_B5
#define LCD_DATA6       PIN_B6
#define LCD_DATA7       PIN_B7

#include <lcd.c>

#define MAX_BUF 17
#define MAX_PAYLOAD 32
#define VREF_H 5
#define VREF_L 0

#define RES_INV 2441
#define RES_INV_2 1220
#define SENSIBILIDAD 129 //25*63/122
#define NUM_MUESTRAS_DIV 0.1
#define NUM_MUESTRAS_OFFSET 0.01
#define BASE_TIEMPO 0.1 //Tiempo durante el cual se hacen las mediciones
#define K_VOLTAJE 1313 //Constante de voltaje
#define E_ABS 30 //Error absoluto

#define FRACCION_NEG(x) if(fraccional_##x < 0) \
                        fraccional_##x *= -1

unsigned int16 corriente_adc = 0, voltaje_adc = 0;
int listo = 0, inicio = 0;
unsigned int32 offset = 0;


# INT_AD
void interrupcion_ad(void)
{
   static int num_adc = 0, cambio = 0;
   unsigned int16 valor_adc = READ_ADC(ADC_READ_ONLY);

//Dependiendo de si es la primera conversión (la de calibración) se ejecuta otro
código
```

```
   if(inicio){
      delay_us(9);
      if(!cambio){ //se alternan las mediciones de corriente y voltaje para
obtener un total de 20
         voltaje_adc += valor_adc;
         set_adc_channel(1);
      }
      else{
         corriente_adc += valor_adc;
         set_adc_channel(0);
      }
      cambio = cambio? 0 : 1;

      ++num_adc;


      if(num_adc == 20){
         listo = 1;
         num_adc = 0;
      }
      else{
         READ_ADC(ADC_START_ONLY);
      }
   }
   else{ //Código para la calibración
      offset += valor_adc;
      ++num_adc;


      if(num_adc == 100){
         num_adc = 0;
         inicio = 1;
      }
      else
         READ_ADC(ADC_START_ONLY);
   }
}


void init_adc(void)
{
   setup_adc_ports(AN0_TO_AN1 | VSS_VDD);
   setup_adc(ADC_CLOCK_DIV_16 | ADC_TAD_MUL_20);
   delay_us(10);
}

void print_lcd(char buf[MAX_BUF])
{
   //Se manda la cadena de caracteres al módulo LCD
   int i, fin = 0;
```

```
    for(i = 0; i < MAX_BUF-1; i++){
        if(! fin){
            if(buf[i])
                lcd_putc(buf[i]);
            else
                fin = 1;
        }
        else
            lcd_putc(0x20);
    }
}



int32 obtener_valor_voltaje (int32 val, int16 cte)
{
    int32 sb;

    sb = val * RES_INV + RES_INV _2;
    return (sb*(VREF_H - VREF_L)*0.0001 + E_ABS)*cte*0.01;
}

int32 obtener_valor_corriente (int32 val)
{
    char buf[MAX_BUF];
    signed int32 sb;

    if(val - offset > 0){
        sb = (val - offset) * RES_INV;
        return ((sb*(VREF_H - VREF_L)*0.0001))*SENSIBILIDAD*0.1;
    }
    else
        return 0;
}


void enviar_dato(signed int32 energia, signed int32 voltaje, signed int32
corriente)
{
    signed int32 fraccional_ene, fraccional_vol, fraccional_cor;
    signed int32 entera_ene, entera_vol, entera_cor;
    char buf[MAX_PAYLOAD] = {0};
    static int32 count = 0;
    static char kj[4] = " KJ"

    entera_ene = energia * 0.001;
    fraccional_ene = energia - entera_ene*1000;
    entera_vol = voltaje * 0.001;
    fraccional_vol = voltaje - entera_vol*1000;
    entera_cor = corriente * 0.001;
```

```
    fraccional_cor = corriente - entera_cor*1000;
    FRACCION_NEG(ene);
    FRACCION_NEG(vol);
    FRACCION_NEG(cor);

    sprintf(buf, "%Ld.%03Lu",  entera_ene, ((int32)fraccional_ene));

    printf("%sP%Ld.%03LuV%Ld.%03LuI%LuT%LuC\n", buf, entera_vol,
((int32)fraccional_vol),
          entera_cor, ((int32)fraccional_cor), (int32)(BASE_TIEMPO*1000.0),
count);

    strcat(buf, KJ);

    lcd_send_byte(0,0x2);
    print_lcd(buf);

    memset(buf, 0,  MAX_PAYLOAD);

    lcd_send_byte(0,0xc0);
    sprintf(buf, "%Ld.%03Lu V %Ld.%03Lu A", entera_vol,
((int32)fraccional_vol),
          entera_cor, ((int32)fraccional_cor));
    print_lcd(buf);

    count++;

}


void main()
{
    unsigned int32 corriente, voltaje;
    signed int32 potencia, energia = 0, energia_k = 0, entero;
    char buf[MAX_BUF];


    //Inicializamos el módulo LCD y el ADC
    lcd_init();
    init_adc();


    enable_interrupts(INT_AD);      //Activa la interrupción de ADC
    enable_interrupts(GLOBAL);      //Activa las interrupciones



    set_adc_channel(1);

    delay_ms(10);
```

```
   READ_ADC(ADC_START_ONLY);

   while(! inicio);

   offset *= NUM_MUESTRAS_OFFSET;

   set_adc_channel(0);
   READ_ADC(ADC_START_ONLY);


   while(true){
      if(listo){ //Cuando ya se han tomado las mediciones suficientes
         //output_high(PIN_C2);

         listo = 0;

         voltaje =
obtener_valor_voltaje(((int32)(voltaje_adc*NUM_MUESTRAS_DIV)),
K_VOLTAJE);
         corriente =
obtener_valor_corriente(((int32)(corriente_adc*NUM_MUESTRAS_DIV)));

         voltaje_adc = 0;
         corriente_adc = 0;

         potencia = ((signed int32)voltaje)*corriente*0.001;
         energia += potencia*BASE_TIEMPO;
         if(energia >= 1000){
            entero = energia*0.001;
            energia_k += entero;
            energia -= entero*1000;
         }

         enviar_dato(energia_k, voltaje, corriente);

         delay_ms(35); //para que sean 100 ms
         READ_ADC(ADC_START_ONLY);
         //output_low(PIN_C2);

      }
   }
}
```

Apéndice D. Código utilizado en el programa de Python para Windows y GNU/Linux.

```
import matplotlib.pyplot as plt
import serial
import serial.tools.list_ports
import time
from matplotlib.widgets import Button
from matplotlib.animation import FuncAnimation
from datetime import datetime
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox
from pathlib import Path
import sys
import re

INIT_LIM = 5

pattern = re.compile("^[0-9]+\.[0-9]{3}P[0-9]+\.[0-9]{3}V[0-9]+\.[0-9]{3}I[0-9]+T[0-9]+C")

fig, ax = plt.subplots()
tiempo, energia, voltaje, corriente, periodo = [], [], [], [], []
ln, = plt.plot([],[],'ro', animated=True)
axcorr = plt.axes([0.7, 0.9, 0.1, 0.075])
axvolt = plt.axes([0.81, 0.9, 0.1, 0.075])
axenerg = plt.axes([0.59, 0.9, 0.1, 0.075])
bcorr = Button(axcorr, 'Corriente')
bvolt = Button(axvolt, 'Voltaje')
benerg = Button(axenerg, 'Energía')

fig.canvas.set_window_title("Consumo energético")

dt = datetime.today()
fecha = dt.strftime("%d%m%y%H%M%S")

root = tk.Tk()
root.title("Puerto serial")
root.protocol("WM_DELETE_WINDOW", lambda: (root.destroy(),sys.exit()))
root.pack_propagate(0)
root.geometry("500x200")

ports = list(serial.tools.list_ports.comports())

devices = []

for p in ports:
    devices.append(p.device)
```

```
if(not devices):
    root.withdraw()
    messagebox.showerror("Error", "No hay puertos seriales disponibles")
    root.destroy()
    sys.exit()

tkvar = tk.StringVar(root)
tkvar.set(devices[0])

popupMenu = tk.OptionMenu(root, tkvar, *devices)
tk.Label(root,text="Escoge el puerto serial").pack(pady=10)

popupMenu.pack(padx=5, pady=20)
botonOK = tk.Button(root,text="Listo", width=40, command= lambda:
(root.quit(),root.withdraw()))

botonOK.pack(padx=5,pady=20, side=tk.BOTTOM)


root.mainloop()

serial_path = tkvar.get()

file_path = filedialog.askdirectory(title="Selecciona directorio donde guardar los
datos")
root.destroy()

if(not file_path):
        sys.exit()

data_folder = Path(file_path)
data_file = "medicion_" + fecha + ".csv"

f = open(data_folder / data_file, "w+")
f.write("Energía,Voltaje,Intensidad de corriente,Periodo,Tiempo\n")

class Index(object):
    opcion = 1

    def grafica_corr (self, event):
        Index.opcion = 3

    def grafica_volt (self, event):
        Index.opcion = 2

    def grafica_energ (self, event):
        Index.opcion = 1
```

```
callback = Index()
bcorr.on_clicked(callback.grafica_corr)
bvolt.on_clicked(callback.grafica_volt)
benerg.on_clicked(callback.grafica_energ)

switch = {
    'P': energia,
    'V': voltaje,
    'I': corriente,
    'T': periodo,
    'C': tiempo
}

try:
    ser = serial.Serial(serial_path, timeout=5)
except:
    messagebox.showerror("Error", "No se pudo abrir puerto serial")
    sys.exit()

def handle_close(event):
    f.close()

fig.canvas.mpl_connect('close_event', handle_close)

def obtener_valor(ser_str, data, c):
    ind = ser_str.find(c)
    data.append(float(ser_str[0:ind]))
    return ser_str[ind+1:]

def update(frames, energia,voltaje,corriente,periodo,tiempo):
    t = time.time()
    ser_str = ""

    switch_axis = {
        1: "Energía (KJ)",
        2: "Voltaje (V)",
        3: "Intensidad de corriente (A)"
    }

    try:
        ser_str = ser.readline().strip().decode('utf-8')
    except:
        return

    if(not pattern.fullmatch(ser_str)):
        return


    ser_str = obtener_valor(ser_str, energia, "P")
```

```
    ser_str = obtener_valor(ser_str, voltaje, "V")
    ser_str = obtener_valor(ser_str, corriente, "I")
    ser_str = obtener_valor(ser_str, periodo, "T")


    ind = ser_str.find("C")
    tiempo.append(float(ser_str[0:ind])*periodo[-1]/1000)

    f.write(str(energia[-1]) + "," + str(voltaje[-1]) + "," + str(corriente[-1]) + "," +
str(periodo[-1]) + "," + str(tiempo[-1]) + "\n")


    energia = energia[-100:]
    tiempo = tiempo[-100:]
    voltaje = voltaje[-100:]
    corriente = corriente[-100:]
    periodo = periodo[-100:]

    switch_func = {
        1: energia,
        2: voltaje,
        3: corriente
    }


    ax.clear()
    xdata = switch_func.get(Index.opcion, 1)

    ax.plot(tiempo[::5], xdata[::5])
    ax.set_ylabel(switch_axis.get(Index.opcion, 1) + " [" + str(xdata[-1]) + "]")
    ax.set_xlabel("Tiempo (s)")
    ax.set_title("Consumo energético", loc='left')

    if(Index.opcion > 1):
        ax.autoscale(axis='x')
        ax.set_ylim(min(xdata) - max(xdata), max(xdata)*2)
    else:
        ax.autoscale(axis='both')


    ln.set_data(tiempo, xdata)

    return ln,

ani = FuncAnimation(fig, update,
fargs=(energia,voltaje,corriente,periodo,tiempo), interval=1)
plt.show()
```

Apéndice E. Código utilizado en el programa para Android.

MainActivity.java

```
package com.icat.bluetooth_motor;

import android.Manifest;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.os.AsyncTask;
import android.os.Handler;
import android.os.Message;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_ENABLE_BT = 1,
REQUEST_COARSE_LOC = 2;
    BluetoothAdapter mBluetoothAdapter;
    Spinner dispositivos;
    final static UUID uuid = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

    List<String> spinnerArray =  new ArrayList<String>();
```

```
    List<BluetoothDevice> bld = new ArrayList<BluetoothDevice>();
    ArrayAdapter<String> adapter;
    private ProgressBar progressBar;



    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if(BluetoothDevice.ACTION_FOUND.equals(action)){
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                adapter.add(device.getName() + " " + device.getAddress());
                adapter.notifyDataSetChanged();
                bld.add(device);
            }
            else
if(BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
                progressBar.setVisibility(View.VISIBLE);
            }
            else
if(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)){
                progressBar.setVisibility(View.INVISIBLE);
            }

        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        adapter = new ArrayAdapter<String>(
                this, R.layout.spinner_layout, spinnerArray);


        getSupportActionBar().setTitle(R.string.main_title);

        progressBar = findViewById(R.id.progreso);
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if(mBluetoothAdapter == null) {
            Toast.makeText(this, R.string.no_blue, Toast.LENGTH_LONG).show();
            return;
        }
        if(!mBluetoothAdapter.isEnabled()){
            Intent BtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(BtIntent, REQUEST_ENABLE_BT);
```

```
        }
        else
            consulta_bluetooth();

        if(ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED){

            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_COARSE_LOCATION},
                    REQUEST_COARSE_LOC);

        }

        dispositivos = findViewById(R.id.dispos);
        Button bt_escaneo = findViewById(R.id.bt_escaneo);
        Button bt_conectar = findViewById(R.id.bt_conectar);


adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdow
n_item);
        dispositivos.setAdapter(adapter);

        bt_escaneo.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mBluetoothAdapter.cancelDiscovery();
                adapter.clear();
                bld.clear();
                mBluetoothAdapter.startDiscovery();
            }
        });
        bt_conectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(adapter.isEmpty())
                    return;
                int pos = dispositivos.getSelectedItemPosition();
                Conectar cnc = new Conectar(bld.get(pos));
                cnc.execute();
            }
        });



        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        registerReceiver(mReceiver, filter);
```

```
    }


    @Override
    protected void onDestroy (){
        super.onDestroy();
        unregisterReceiver(mReceiver);
    }


    @Override
    public void onRequestPermissionsResult(int requestCode, String
permissions[], int[] grantResults){

        switch(requestCode){
            case REQUEST_COARSE_LOC: {
                if(grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED);
                else
                    Toast.makeText(this, R.string.funcionalidad,
Toast.LENGTH_LONG).show();
            }
        }
    }


    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if(requestCode == REQUEST_ENABLE_BT){
            if(resultCode == RESULT_OK){
                consulta_bluetooth();
            }
        }
    }


    void consulta_bluetooth(){
        Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
        if(pairedDevices.size() > 0){
            for(BluetoothDevice device : pairedDevices){
                adapter.add(device.getName() + " " + device.getAddress());
                bld.add(device);
            }
            adapter.notifyDataSetChanged();
        }
    }
```

```
    private class Conectar extends AsyncTask<Void, Void, Integer>{
        private final BluetoothSocket btSocket;
        private final BluetoothDevice btDevice;

        public Conectar(BluetoothDevice device){
            BluetoothSocket tmp = null;
            btDevice = device;

            try {
                tmp = device.createRfcommSocketToServiceRecord(uuid);
            } catch(IOException e){}
            btSocket = tmp;
        }

        public Integer doInBackground(Void ... v){
            mBluetoothAdapter.cancelDiscovery();

            try{
                publishProgress();
                btSocket.connect();
            }catch (IOException e){
                try{
                    btSocket.close();
                } catch(IOException b){}
                return 1;
            }
            return 0;
        }

        public void onPostExecute(Integer result){
            progressBar.setVisibility(View.INVISIBLE);

            if(result == 0) {
                Intent intent = new Intent(MainActivity.this, Graficar.class);
                BtSocketHandler.setBtsocket(btSocket);
                startActivity(intent);

            }
            else
                Toast.makeText(MainActivity.this, R.string.no_conecta,
Toast.LENGTH_LONG).show();
        }


        public void onProgressUpdate(Void ... v){
            progressBar.setVisibility(View.VISIBLE);
        }
    }
```

}

Graficar.java

```
package com.icat.bluetooth_motor;

import android.Manifest;
import android.app.usage.ExternalStorageStats;
import android.bluetooth.BluetoothSocket;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.graphics.Color;
import android.graphics.DashPathEffect;
import android.graphics.Paint;
import android.graphics.PointF;
import android.os.AsyncTask;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.androidplot.Plot;
import com.androidplot.Region;
import com.androidplot.ui.Anchor;
import com.androidplot.ui.HorizontalPositioning;
import com.androidplot.ui.SeriesBundle;
import com.androidplot.ui.Size;
import com.androidplot.ui.SizeMode;
import com.androidplot.ui.TextOrientation;
import com.androidplot.ui.VerticalPositioning;
import com.androidplot.ui.widget.TextLabelWidget;
import com.androidplot.util.PixelUtils;
import com.androidplot.xy.BoundaryMode;
import com.androidplot.xy.CatmullRomInterpolator;
import com.androidplot.xy.LineAndPointFormatter;
```

```
import com.androidplot.xy.PanZoom;
import com.androidplot.xy.RectRegion;
import com.androidplot.xy.SimpleXYSeries;
import com.androidplot.xy.StepMode;
import com.androidplot.xy.XYCoords;
import com.androidplot.xy.XYGraphWidget;
import com.androidplot.xy.XYPlot;
import com.androidplot.xy.XYSeries;
import com.androidplot.xy.XYSeriesFormatter;

import org.w3c.dom.Text;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.DateFormat;
import java.text.DecimalFormat;
import java.text.FieldPosition;
import java.text.Format;
import java.text.ParsePosition;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Observable;
import java.util.Observer;

public class Graficar extends AppCompatActivity {

    private XYPlot grafica;
    private static final int MESSAGE_READ = 3;
    private ActualizaGrafica actGraf;
    private Conectado data;
    static private Thread hilo;
    private TextLabelWidget acumulado;
    static private int BOUNDARY_STEP_Y = 10;
    static private double BASE_TIEMPO = 1000.0; //milisegundos
    final private int
MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE = 1;
    static SampleDynamicSeries serie1;
    static LineAndPointFormatter serie1Formato;
    static private BufferedWriter writer = null;
    static private int opcion_grafica = 1;
```

```
    static String[] rangos = {"Energía (KJ)", "Voltaje (V)", "Intensidad de corriente
(A)"};
    static boolean permiso = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_graficar);


        getSupportActionBar().setTitle(R.string.main_title);



        if(ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED)
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE);
        else {
            permiso = openWriteFile();
        }


        grafica = findViewById(R.id.plot);

        acumulado = new TextLabelWidget(grafica.getLayoutManager(),
"Energía",
                new Size(
                        PixelUtils.dpToPix(100), SizeMode.ABSOLUTE,
                        PixelUtils.dpToPix(100), SizeMode.ABSOLUTE),
                TextOrientation.HORIZONTAL);

        acumulado.getLabelPaint().setTextSize(PixelUtils.dpToPix(16));

        Paint p = new Paint();
        p.setARGB(100,0,0,0);
        acumulado.setBackgroundPaint(p);

        acumulado.position(0, HorizontalPositioning.RELATIVE_TO_CENTER,
                PixelUtils.dpToPix(45), VerticalPositioning.ABSOLUTE_FROM_TOP,
                Anchor.TOP_MIDDLE);
        acumulado.pack();


        final Handler mHandle = new HandleManager(acumulado);


        actGraf = new ActualizaGrafica(grafica);
```

```
        data = new Conectado(BtSocketHandler.getBtsocket(), mHandle, grafica);

        serie1 = new SampleDynamicSeries(data, 0,
getResources().getString(R.string.medida));


        serie1Formato = new LineAndPointFormatter(Color.RED, null,
Color.BLUE, null);

        serie1Formato.setInterpolationParams(
                new CatmullRomInterpolator.Params(10,
CatmullRomInterpolator.Type.Centripetal));

        grafica.addSeries(serie1, serie1Formato);

        data.addObserver(actGraf);

        grafica.setDomainStepMode(StepMode.SUBDIVIDE);
        grafica.setDomainStepValue(7);
        grafica.setRangeStepMode(StepMode.SUBDIVIDE);
        grafica.setRangeStepValue(10);

        grafica.setRangeBoundaries(0, BOUNDARY_STEP_Y,
BoundaryMode.FIXED);

        DashPathEffect dashFx = new DashPathEffect(
                new float[] {PixelUtils.dpToPix(3), PixelUtils.dpToPix(3)}, 0);
        grafica.getGraph().getDomainGridLinePaint().setPathEffect(dashFx);
        grafica.getGraph().getRangeGridLinePaint().setPathEffect(dashFx);


grafica.getGraph().getLineLabelStyle(XYGraphWidget.Edge.BOTTOM).setForm
at(new DecimalFormat("####.###"));

        PanZoom.attach(grafica, PanZoom.Pan.BOTH,
PanZoom.Zoom.STRETCH_VERTICAL);

        hilo = new Thread(data);
        hilo.start();


    }



    @Override
    public void onRequestPermissionsResult(int requestCode, String
permissions[], int[] grantResults){
        switch(requestCode){
```

```
            case
MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE:
                if(grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
                    permiso = openWriteFile();
                }
                else {
                    Toast.makeText(getApplicationContext(), "No se podrán guardar
los datos", Toast.LENGTH_LONG).show();
                }

        }
    }


    public boolean isExternalStorageWritable() {
        String state = Environment.getExternalStorageState();
        if(Environment.MEDIA_MOUNTED.equals(state)){
            return true;
        }
        return false;
    }

    public boolean openWriteFile(){
        if(isExternalStorageWritable()){
            File directorio, archivo;
            directorio = getPublicAlbumStorageDir(getString(R.string.nombre_dir));
            DateFormat dateFormat = new
SimpleDateFormat("ddMMyyyyHHmmss", Locale.US);
            Date date = new Date();
            archivo = new File(directorio, getString(R.string.nombre_archivo) + "_" +
dateFormat.format(date) + ".csv");
            try {
                archivo.createNewFile();
                FileWriter fw = new FileWriter(archivo);
                writer = new BufferedWriter(fw);
                writer.write("Energía,Voltaje,Intensidad de
corriente,Periodo,Tiempo\n");
            } catch (Exception e){
                Log.e("archivo", e.getMessage());
                return false;
            }
            return true;
        }
        return false;
    }



    public File getPublicAlbumStorageDir(String albumName) {
```

```
        File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY
_DOCUMENTS), albumName);
        file.mkdir();
        return file;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu){
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_grafica, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.reset:
                resetear();
                return true;
            case R.id.energia:
                opcion_grafica = 1;
                return true;
            case R.id.voltaje:
                opcion_grafica = 2;
                return true;
            case R.id.corriente:
                opcion_grafica = 3;
                return true;

        }

        return super.onOptionsItemSelected(item);
    }

    public void resetear(){
        int sz = serie1.size();
        double x = serie1.getX(sz - 1).doubleValue(), y = serie1.getY(sz -
1).doubleValue();
        Log.i("serie1", "size=" + Integer.toString(sz) + " x=" + Double.toString(x) + "
y=" + Double.toString(y));

        grafica.centerOnRangeOrigin(y);

    }


    @Override
    public void onDestroy(){
        super.onDestroy();
```

```
        hilo.interrupt();
        BtSocketHandler.closeBtsocket();
    }

    @Override
    public void onStop() {
        super.onStop();
        try {
            writer.flush();
        } catch (Exception e){
            Log.e("flush", "no se pudo escribir en archivo");
        }

    }

    @Override
    public void onPause(){
        super.onPause();
    }

    @Override
    public void onStart() {
        super.onStart();
    }


    @Override
    public void onResume() {
        super.onResume();
    }

    private static class Conectado implements Runnable {
        private final BufferedReader mmInStream;
        private Handler mHandler;
        private int TAM_MUESTRA = 0, LIMITE_TAM_MUESTRA = 1000;
        private MyObservable notifier;
        private List<String> energia, tiempo, voltaje, corriente, periodo, datos;
        private XYPlot grafica;


        public Conectado(BluetoothSocket socket, Handler mHandler, XYPlot
grafica) {
            InputStream tmpIn = null;
            notifier = new MyObservable();
            this.grafica = grafica;

            try{
                tmpIn = socket.getInputStream();
            } catch (IOException e) {}
```

```
            mmInStream = new BufferedReader (new InputStreamReader(tmpIn));
            this.mHandler = mHandler;
            energia = new ArrayList<String>();
            tiempo = new ArrayList<String>();
            voltaje = new ArrayList<String>();
            periodo = new ArrayList<String>();
            corriente = new ArrayList<String>();
            datos = new ArrayList<String>();

        }

        class MyObservable extends Observable{
            @Override
            public void notifyObservers(){
                setChanged();
                super.notifyObservers();
            }
        }


        public void run(){

            String line;
            String str = "";
            int opcion_ant = 0;



            while(true) {
                if(hilo.isInterrupted()){
                    return;
                }
                try{
                    if ((line = mmInStream.readLine()) != null
                            && line.matches("^[0-9]+\\.[0-9]{3}P[0-9]+\\.[0-9]{3}V[0-
9]+\\.[0-9]{3}I[0-9]+T[0-9]+C")) {
                        energia.add(line.substring(0, line.indexOf('P')));
                        voltaje.add(line.substring(line.indexOf('P') + 1, line.indexOf('V')));
                        corriente.add(line.substring(line.indexOf('V') + 1,
line.indexOf('I')));
                        periodo.add(line.substring(line.indexOf('I') + 1, line.indexOf('T')));

tiempo.add(Double.toString(Double.parseDouble(line.substring(line.indexOf('T')
+ 1,
                                line.indexOf('C'))) *
Double.parseDouble(periodo.get(periodo.size() - 1)) / BASE_TIEMPO));


                        if (TAM_MUESTRA < LIMITE_TAM_MUESTRA)
                            TAM_MUESTRA++;
```

```
                    else {
                        tiempo.remove(0);
                        energia.remove(0);
                        voltaje.remove(0);
                        corriente.remove(0);
                        periodo.remove(0);
                    }

                    if (permiso)
                        writer.write(energia.get(energia.size() - 1) + "," +
voltaje.get(voltaje.size() - 1) +
                                "," + corriente.get(corriente.size() - 1) + "," +
periodo.get(periodo.size() - 1) + "," +
                                tiempo.get(tiempo.size() - 1) + "\n");

                    switch (opcion_grafica){
                        case 1:
                            datos = energia;
                            str = "Energía = " + datos.get(datos.size() -1) + " KJ";
                            break;
                        case 2:
                            datos = voltaje;
                            str = "Voltaje = " + datos.get(datos.size() - 1) + " V";
                            break;
                        case 3:
                            datos = corriente;
                            str = "Corriente = " + datos.get(datos.size() -1) + " A";
                            break;
                    }

                    if(opcion_ant != opcion_grafica) {
                        if(opcion_ant != 0) {
                            grafica.clear();
                            TextLabelWidget txt = grafica.getRangeTitle();
                            txt.setText(rangos[opcion_grafica - 1]);
                            grafica.setRangeTitle(txt);
                            grafica.addSeries(serie1, serie1Formato);
                            grafica.redraw();
                        }

grafica.setRangeLowerBoundary(Double.parseDouble(datos.get(0))/2.0,
BoundaryMode.FIXED);

grafica.setRangeUpperBoundary(Double.parseDouble(datos.get(datos.size()-
1))*2.0, BoundaryMode.FIXED);
                        opcion_ant = opcion_grafica;
                    }

                    mHandler.obtainMessage(MESSAGE_READ, str.length(), -1,
str).sendToTarget();
```

```
grafica.setDomainLowerBoundary(Double.parseDouble(tiempo.get(0)),
BoundaryMode.FIXED);
                    if(tiempo.size() > 1)

grafica.setDomainUpperBoundary(Double.parseDouble(tiempo.get(tiempo.size(
)-1)), BoundaryMode.FIXED);

                    notifier.notifyObservers();

                }

            } catch (IOException e) {
                Log.e("run", "error en ejecución: " + e.getMessage());
            }

        }

    }


    public int getItemCount(int serie){
        return datos.size();
    }

    public Number getX(int serie, int indice){
        if(indice >= TAM_MUESTRA+1)
            throw new IllegalArgumentException();
        return Double.parseDouble(tiempo.get(indice));
    }

    public Number getY(int serie, int indice){
        if(indice >= TAM_MUESTRA+1)
            throw new IllegalArgumentException();
        return Double.parseDouble(datos.get(indice));
    }

    public void addObserver(Observer observer){
        notifier.addObserver(observer);
    }

}

private static class HandleManager extends Handler{
    TextLabelWidget sel;

    HandleManager(TextLabelWidget sel){
        this.sel = sel;
```

```
        }

        @Override
        public void handleMessage(Message msg){
            sel.setText((String) msg.obj);
        }

    }


    private class ActualizaGrafica implements Observer {
        Plot grafica;

        public ActualizaGrafica(Plot grafica) {
            this.grafica = grafica;
        }

        @Override
        public void update(Observable o, Object arg){
            grafica.redraw();
        }
    }


    class SampleDynamicSeries implements XYSeries {
        private Conectado fuente;
        private int indiceSerie;
        private String titulo;

        public SampleDynamicSeries(Conectado fuente, int indiceSerie, String
titulo){
            this.fuente = fuente;
            this.indiceSerie = indiceSerie;
            this.titulo = titulo;
        }

        @Override
        public String getTitle(){
            return titulo;
        }

        @Override
        public int size(){
            return fuente.getItemCount(indiceSerie);
        }

        @Override
        public Number getX(int indice){
            return fuente.getX(indiceSerie, indice);
        }
```

```
        @Override
        public Number getY(int indice) {
            return fuente.getY(indiceSerie, indice);
        }
    }
}
```

BtSocketHandler.java

```
package com.icat.bluetooth_motor;

import android.bluetooth.BluetoothSocket;
import android.util.Log;

public class BtSocketHandler {

    private static BluetoothSocket btsocket;

    public static synchronized BluetoothSocket getBtsocket(){
        return btsocket;
    }

    public static synchronized void setBtsocket(BluetoothSocket btsocket){
        BtSocketHandler.btsocket = btsocket;
    }

    public static synchronized void closeBtsocket(){
        try{
            btsocket.close();
        }catch (Exception e) {
            Log.e("BtSocketHandler", e.getMessage());
        }

    }
}
```

Apéndice F. Instrucciones para el uso de la aplicación en Android.

Una vez instalada la aplicación en su smartphone, al iniciar su ejecución debe presentarse una pantalla igual a la de la figura 13 de la sección de *Programación de aplicación para sistema operativo Android* de este trabajo. Si en esa pantalla se toca el área debajo del texto “Dispositivos encontrados” se desplegará una lista con los dispositivos Bluetooth previamente sincronizados. En caso de que el dispositivo requerido no aparezca, es necesario tocar el botón de “ESCANEAR” y esperar un momento a que se muestre una nueva lista con dispositivos Bluetooth. En caso de que ya se tenga seleccionado el dispositivo que se quiere, puede proseguir tocando el botón “CONECTARSE”.

Aparecerá entonces una pantalla igual a la de la figura 14 de la misma sección referida anteriormente. En esta pantalla podrá observar la forma en la que se actualiza una gráfica de línea de acuerdo a los datos que recibe la aplicación. Si la línea de la gráfica supera el borde superior, puede tocar el botón “RESET” para ajustar la escala y volver a ver por completo la gráfica. En caso de que quiera cambiar el tipo de medición visualizado, solo debe tocar el botón con los tres puntos apilados de forma vertical y seleccionar el que desea de la lista desplegada. Los datos se guardarán en un archivo con el nombre de “medición_” concatenado con la fecha y hora de aquel momento, estos archivos se podrán encontrar en la subcarpeta de “Mediciones” dentro de la carpeta de “Documentos” en el sistema de archivos del smartphone.

Apéndice G. Instrucciones para el uso de la aplicación en Windows y GNU/Linux.

Al ejecutar la aplicación, deberá presentarse un diálogo que pregunta por el puerto serie al cual está conectado el Arduino, la obtención de esta información dependerá del sistema operativo que se use. Una vez seleccionado el puerto serie correcto, aparecerá otro diálogo que solicitará el nombre de la carpeta en la cual guardar el archivo con los datos recibidos. Al especificar esto último, la aplicación procederá a desplegar una ventana con la gráfica del consumo energético. Dentro de esta ventana se encuentran tres botones correspondientes a los tres tipos de mediciones que se pueden graficar, por lo que para visualizar otra medida se deberá dar click en el botón respectivo.

Apéndice H. Hoja de datos para sensor de corriente ACS756KCA-050B.

ACS756xCB

Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 3 kVRMS Voltage Isolation and a Low-Resistance Current Conductor

FEATURES AND BENEFITS

- Industry-leading noise performance through proprietary amplifier and filter design techniques
- Total output error 0.8% at T_A = 25°C
- Small package size, with easy mounting capability
- Monolithic Hall IC for high reliability
- Ultra-low power loss: 100 µΩ internal conductor resistance
- 3 kV_{RMS} minimum isolation voltage from pins 1-3 to pins 4-5
- 3.0 to 5.0 V, single supply operation
- 3 µs output rise time in response to step input current
- 20 or 40 mV/A output sensitivity
- Output voltage proportional to AC or DC currents
- Factory-trimmed for accuracy
- Extremely stable output offset voltage
- Nearly zero magnetic hysteresis

TÜV America Certificate Number: U8V 14 05 54214 028

UL Certified File No.: E316429

PACKAGE:

5-Pin CB Package (suffix PFF)

Additional leadforms available for qualifying volumes

DESCRIPTION

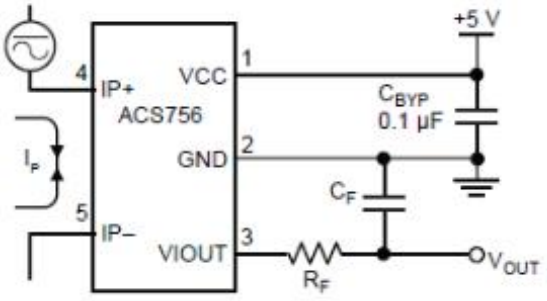
The Allegro ACS756 family of current sensor ICs provides economical and precise solutions for AC or DC current sensing in industrial, automotive, commercial, and communications systems. The device package allows for easy implementation by the customer. Typical applications include motor control, load detection and management, power supplies, and overcurrent fault protection.

The device consists of a precision, low-offset linear Hall circuit with a copper conduction path located near the die. Applied current flowing through this copper conduction path generates a magnetic field which the Hall IC converts into a proportional voltage. Device accuracy is optimized through the close proximity of the magnetic signal to the Hall transducer. A precise, proportional voltage is provided by the low-offset, chopper-stabilized BiCMOS Hall IC, which is programmed for accuracy at the factory.

The output of the device has a positive slope (>V_{CC}/2) when an increasing current flows through the primary copper conduction path (from terminal 4 to terminal 5), which is the path used for current sampling. The internal resistance of this conductive path is 100 µΩ typical, providing low power loss.

The thickness of the copper conductor allows survival of the device at up to 5× overcurrent conditions. The terminals of the conductive path are electrically isolated from the signal leads

Continued on the next page…



Typical Application

The ACS756 outputs an analog signal (V_{OUT}) that varies linearly with the uni- or bi-directional AC or DC primary sampled current, I_P, within the range specified. C_F is for optimal noise management, with values that depend on the application.

Apéndice I. Hoja de datos para módulo de transmisión KYL-500S.

Shenzhen KYL Communication Equipment Co., Ltd

KYL-500S Mini-size Wireless Data Transceiver Module

KYL-500S is a Mini-size RF transceiver. It is usually used for restricted space application. With TTL interface, it is widely used for micro-controller wireless communication and other TTL level port communication systems. It has high reliability and good performance.

I. Technical specification

PERFORMANCE	
Power Output:	50mW(Default), (10~100mW optional)
RF Line-of-sight Range:	1000m@1200bps; 600m@9600bps
RF Effective Rate:	1200/2400/4800/9600/19200bps
Space Channel:	1MHz(Default),(12.5/25KHz/other customization)
Bandwidth:	<25KHz
Receiver Sensitivity:	-123dBm@1200bps(1% BER)
NETWORKING	
Networking Topology:	Point-to-point, point-to-multipoint
COMPATIBILITY	
KYL-200 and KYL-300 series	
POWER	
Supply Voltage:	5V DC (default), 3.3-3.6V(optional)
Transmit Current:	<40mA

Apéndice J. Hoja de datos para el amplificador operacional TLV271.

TEXAS INSTRUMENTS

TLV271, TLV272, TLV274

SLOS351E – FEBRUARY 2004 – REVISED NOVEMBER 2015

TLV27x Family of 550-µA/Ch, 3-MHz, Rail-to-Rail Output Operational Amplifiers

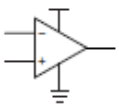
1 Features

- Rail-to-Rail Output
- Wide Bandwidth: 3 MHz
- High Slew Rate: 2.4 V/µs
- Supply Voltage Range: 2.7 V to 16 V
- Supply Current: 550 µA/Channel
- Input Noise Voltage: 39 nV/√Hz
- Input Bias Current: 1 pA
- Specified Temperature Range:
  - Commercial Grade: 0°C to 70°C
  - Industrial Grade: –40°C to 125°C
- Ultrasmall Packaging:
  - 5-Pin SOT-23 (TLV271)
  - 8-Pin MSOP (TLV272)
- Ideal Upgrade for TLC72x Family

2 Applications

- E-Bike
- Power Banks
- Smoke detectors
- Solar Inverters
- Low-Power Motor Controls
- Battery-Powered Instruments
- Building Automation

Operational Amplifier



3 Description

Operating from 2.7 V to 16 V over the extended industrial temperature range from -40°C to +125°C, the TLV27x is a low power, wide bandwidth operational amplifier (opamp) with rail to rail output. This makes it an ideal alternative to the TLC27x family for applications where rail-to-rail output swings are essential. The TLV27x provides 3-MHz bandwidth from only 550 µA.

Like the TLC27x, the TLV27x is fully specified for 5-V and ±5-V supplies. The maximum recommended supply voltage is 16 V, which allows the devices to be operated from a variety of rechargeable cells (±8 V supplies down to ±1.35 V).

The CMOS inputs enable use in high-impedance sensor interfaces, with the lower voltage operation making an attractive alternative for the TLC27x in battery-powered applications.

All members are available in PDIP and SOIC with the singles in the small SOT-23 package, duals in the MSOP, and quads in the TSSOP package.

The 2.7-V operation makes it compatible with Li-Ion powered systems and the operating supply voltage range of many micropower microcontrollers available today including TI's MSP430.

Device Information(1)

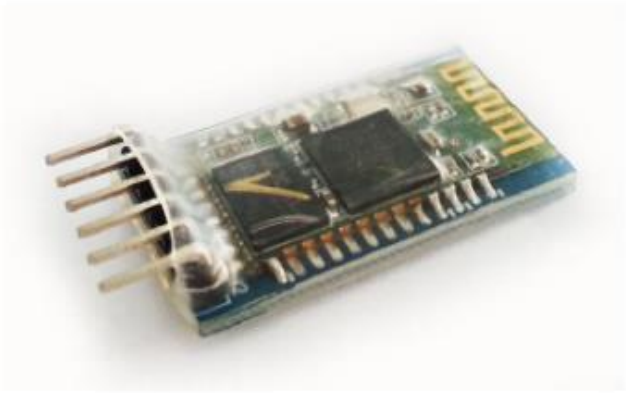
PART NUMBER	PACKAGE	BODY SIZE (NOM)
TLV271	SOIC (8)	4.98 mm × 3.91 mm
	SOT-23 (5)	2.90 mm × 1.60 mm
	PDIP (8)	9.81 mm × 6.35 mm
TLV272	SOIC (8)	4.98 mm × 3.91 mm
	PDIP (8)	9.81 mm × 6.35 mm
	VSSOP (8)	3.00 mm × 3.00 mm
TLV274	SOIC (14)	8.65 mm × 3.91 mm
	PDIP (14)	3.90 mm × 6.35 mm
	TSSOP (14)	5.00 mm × 4.40 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Apéndice K. Hoja de datos para el módulo Bluetooth HC-05.

Descargable desde <https://www.gme.cz/data/attachments/dsh.772-148.1.pdf>.

HC-05 Bluetooth Module



User's Manual V1.0

Apéndice L. Hoja de datos para el microcontrolador PIC18F2458.

MICROCHIP

PIC18F2458/2553/4458/4553

28/40/44-Pin High-Performance, Enhanced Flash, USB Microcontrollers with 12-Bit A/D and nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 Endpoints (16 bidirectional)
- 1-Kbyte Dual Access RAM for USB
- On-Chip USB Transceiver with On-Chip Voltage Regulator
- Interface for Off-Chip USB Transceiver
- Streaming Parallel Port (SPP) for USB Streaming Transfers (40/44-pin devices only)

Power-Managed Modes:

- Run: CPU On, Peripherals On
- Idle: CPU Off, Peripherals On
- Sleep: CPU Off, Peripherals Off
- Idle mode Currents Down to 5.8 μA Typical
- Sleep mode Currents Down to 0.1 μA Typical
- Timer1 Oscillator: 1.1 μA Typical, 32 kHz, 2V
- Watchdog Timer: 2.1 μA Typical
- Two-Speed Oscillator Start-up

Special Microcontroller Features:

- C Compiler Optimized Architecture with Optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash Program Memory Typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory Typical
- Flash/Data EEPROM Retention: > 40 Years
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 41 ms to 131s
- Programmable Code Protection
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via Two Pins
- In-Circuit Debug (ICD) via Two Pins
- Optional Dedicated ICD/ICSP Port (44-pin TQFP package only)
- Wide Operating Voltage Range (2.0V to 5.5V)

Flexible Oscillator Structure:

- Four Crystal modes, Including High-Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal Oscillator Block:
  - 8 user-selectable frequencies, from 31 kHz to 8 MHz
  - User-tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Dual Oscillator Options allow Microcontroller and USB module to Run at Different Clock Speeds
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if any clock stops

Peripheral Highlights:

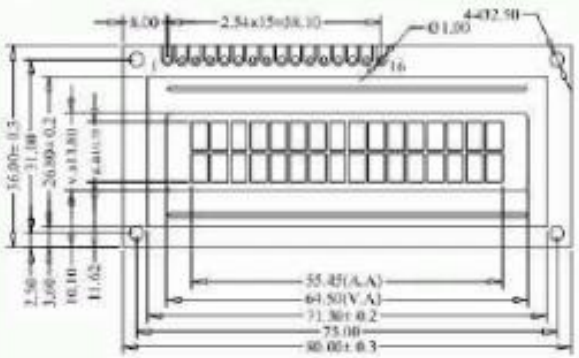
- High-Current Sink/Source: 25 mA/25 mA
- Three External Interrupts
- Four Timer modules (Timer0 to Timer3)
- Up to 2 Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 5.2 ns (TCY/16)
  - Compare is 16-bit, max. resolution 83.3 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bits
- Enhanced Capture/Compare/PWM (ECCP) module:
  - Multiple output modes
  - Selectable polarity
  - Programmable dead time
  - Auto-shutdown and auto-restart
- Enhanced USART module:
  - LIN bus support
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I^2C™ Master and Slave modes
- 12-Bit, up to 13-Channel Analog-to-Digital Converter module (A/D) with Programmable Acquisition Time
- Dual Analog Comparators with Input Multiplexing

Note: This document is supplemented by the *"PIC18F2455/2550/4455/4550 Data Sheet"* (DS39632). See **Section 1.0 "Device Overview"**.

Device	Program Memory		Data Memory		I/O	12-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comp.	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™			
PIC18F2458	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2553	32K	16384											
PIC18F4458	24K	12288			35	13	1/1	Yes					
PIC18F4553	32K	16384											

Apéndice M. Hoja de datos para la pantalla LCD QC1602A.

1.DIMENSION OUTLINE



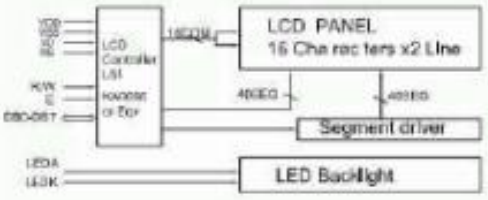
2.MECHANICAL SPECIFICATIONS

ITEM	SPECIFICATIONS	ITEM	REMARK
Module Size(L×W×H)	80.0×36.0×13.0	mm	Reference Dimensional Outline
View Area(W×H)	64.5×13.8	mm	
Effective V.Area	55.45×10.75	mm	
Number of Characters	16CH×2Lines	-	
Character Size(W×H)	2.95×5.15	mm	
Dot Size(W×H)	0.55×0.60	mm	
Weight(Reflective/Led)	-	g	

3.ABSOLUTE MAXIMUM RATINGS

ITEM	SYMBOL	CONDITION	STANDARD	
			MIN	MAX
Logic Voltage	V_{DD}	Ta=25℃	-0.3V	7V
LCD Voltage	V_{LCD}		-0.3V	13V
Input Voltage	V_I		-0.3V	V_{DD}+0.3V
Operation Temperature	T_{OP}	—	-20℃	70℃
Storage Temperature	T_{ST}	—	-30℃	80℃

4.BLOCK DIAGRAMMECHANICAL



5.LED BACKLIGHT SPECIFICATIONS

ITEM	SYMBOL	TYPE	MAX	UNIT
Ta=25℃				
Forward Voltage	V_F	4.1	4.3	V
Forward Current	I_F	120	—	mA
Emission Wave Length	λ_P	568	—	nm

6.INTERFACE PIN CONNECTIONS

ITEM	SYMBOL	LEVEL	FUNCTIONS
1	VSS	0V	Power Ground
2	VDD	+5V	Power supply for logic
3	VO	—	Contrast adjust
4	RS	H/L	H:data L:command
5	R/W	H/L	H:read L:write
6	E	H/H→L	Enable signal
7-14	DB0-DB7	H/L	Data Bus
15	LEDA	+5V	Power supply for LED Backlight
16	LEDK	0V	

7.ELECTRICAL CHARACTERISTICS

ITEM	SYMBOL	MIN	TYPE	MAX	UNIT
Ta=25℃					
Logic Power	V_{DD}	4.5	5	5.5	V
Input High Voltage	V_{IH}	2.2	—	V_{DD}	V
Input Low Voltage	V_{IL}	-0.3	—	0.6	V
Output High Voltage	V_{OH}	2.4	—	V_{DD}	V
Output Low Voltage	V_{OL}	0	—	0.4	V
Logic Current	I_{DD}	—	1.5	3.0	mA
Operation Voltage For LCD	V_{OP}=V_{DD}-V_O	—	5	—	V

Bibliografía

Allegro MicroSystems. (2018) **ACS756xCB** [Internet]. Disponible desde < http://www.allegromicro.com/~/media/files/datasheets/acs756-datasheet.ashx> [Acceso 10 de diciembre 2018].

Android Open Source Project. **Temas principales para desarrolladores** [Internet]. Disponible desde: <https://developer.android.com/guide/> [Acceso 10 de enero 2019].

Barrow, J. (2007) Reducing Ground Bounce in DC-to-DC Converters - Some Grounding Essentials. **Analogue Dialogue**. 41(6), junio, p.1-5.

Basile, B.; Schauer, S.; Venkat, K. (2014) **Implementation of a Single-phase Electronic Watt-hour Meter Using the Msp430f6736**. Texas Instruments.

Electrical4U. (2018) **Construction of AC Energy Meter** [Internet]. Disponible desde: <https://www.electrical4u.com/construction-of-ac-energy-meter/> [Acceso 4 de febrero 2019].

Electronics Tutorials. **Hall Effect Sensor** [Internet]. Disponible desde: <https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html> [Acceso 6 de febrero 2019].

Franco S. (2015) **Design with operational amplifiers and analog integrated circuits**. 4a ed. EE. UU., McGraw-Hill.

Castillo Hernández, J.; Kemper Valverde, N.; Sovero Anacheyta, G. (2013) Medición inalámbrica del consumo eléctrico para el ahorro de energía en edificios. En: Instituto Nacional de la Tecnología Industrial. **10th International Congress on Electrical Metrology, septiembre 2013, Buenos Aires, Argentina**.

Holman, J.P. (1986) **Métodos experimentales para ingenieros**. 2a. ed. México, McGraw-Hill.

Mancini, R.; Carter, B. (2009) **Op Amps for Everyone**. 3a. ed. EE. UU., Elsevier.

Gallardo Puertas, O. (2015) **Fabricación de placas de circuito impreso con Proteus**. Tesis de licenciatura, Universidad de Valladolid.

Raman, B. (2012) **Overview of Single Phase Induction Type Energy Meter**, [Internet], Electrical Engineering Portal. Disponible desde: <https://electrical-engineering-portal.com/overview-of-single-phase-induction-type-energy-meter> [Acceso 4 de febrero 2019].

Shenzen KYL Communication Equipment Co. **KYL-500S** [Internet]. Disponible desde <http://www.rf-data.com/en/product-23475-28813-93845.html> [Acceso 16 enero 2019].