ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

Variación del consumo energético en APIs WEB para IoT

Power consumption variability on IoT related APIs WEB

Rodrigo Morell, morell.r@gmail.com

Universidad Nacional de la Patagonia Austral
Av. Lisandro de la Torre 860 - Río Gallegos - Santa Cruz - Argentina

RESUMEN

En los últimos tiempos, ha habido un gran interés por las aplicaciones de IoT. A su vez, su desarrollo ha sido delineado por requerimientos específicos, como la eficiencia energética. Dado el contexto mundial de cambio climático, es importante que las nuevas tecnologías ayuden a reducir la huella de carbono. En este sentido, las interfaces de programación de aplicaciones WEB (APIs WEB) se presentan como una solución para la conectividad de los dispositivos IoT. Este estudio analiza como varía el consumo de energía de las baterías en las aplicaciones de IoT que utilizan servicios de APIs WEB, así como los factores que influyen en la reducción del consumo de energía.

Palabras clave: APIs WEB; IoT; RESTful; ESP32

ABSTRACT

In recent years, there has been a growing interest in implementing and developing IoT applications. However, this has led to several challenges that need to be addressed. For instance, during the design stages, critical provisions such as power consumption have been identified as important requirements. Additionally, the current global climate change context has heightened awareness of the carbon footprint of new technologies. To address these challenges, APIs WEB services have emerged as a potential solution for supporting IoT device connectivity. This paper aims to explore how energy consumption at the battery level will impact IoT applications supported by APIs WEB services, and the factors that contribute to its reduction.

Key words: APIs WEB; IoT; RESTful; ESP32

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

1. INTRODUCCIÓN

Debido al avance en la integración de dispositivos electrónicos, como microchips capaces de alojar sistemas operativos y procesar gran cantidad de datos a altas frecuencias, los SoC (system on a chip) han posibilitado el desarrollo de las tecnologías IoT (internet of the things) [17] que a su vez se han beneficiado no solo por el progreso en las técnicas de integración sino también por la mejora en la conectividad de internet [16]. Las aplicaciones móviles capaces de controlar dispositivos en forma remota, registrar datos, y/o tomar decisiones a distancia también se han beneficiado por el avance en la conectividad y expansión en las redes. La comunicación en las arquitecturas IoT puede estar soportada sobre diversos tipos de plataformas y protocolos como propone Hejazi et al.[14] y Dizdarević et al.[6]. Uno de los protocolos que posibilitan el procesamiento de la información en aplicaciones de IoT es el HTTP (Hypertext Transfer Protocol). Sobre el protocolo HTTP los desarrolladores pueden implementar sus aplicaciones a través de las APIs WEB. La utilización de APIs WEB ofrece una excelente opción para el desarrollo de aplicaciones IoT como indica Ferreira et al. [9] por su versatilidad para adaptarse a diferentes tipos de diseños y arquitecturas. Recientemente, ha surgido una extensa variedad de proveedores de *cloud computing* que ofrecen sus servicios para procesar información de IoT a través de sus APIs [31].

Con la implementación de control a distancia, muchos desafíos cobran importancia para el diseño de APIs WEB que se puedan emplear para aplicaciones de IoT [29] [30]. Algunas de las propiedades críticas que estas APIs WEB deben resolver son tamaño, seguridad y consumo de energía. El tamaño de las APIs WEB debe ser reducido, ya que los dispositivos o nodos generalmente se encuentran limitados en potencia de procesamiento de CPU y tamaño de memoria [12] [2] [26]. La seguridad es otro aspecto crítico, ya que los nodos en muchas ocasiones soportan aplicaciones en industrias críticas como centrales eléctricas o plantas químicas, y en caso de un acceso no permitido a la red IoT, dichas aplicaciones pondrían en riesgo a grandes poblaciones de usuarios [25] [11]. Por último, el consumo de energía cobra relevancia en la confiabilidad del sistema, ya que los nodos y/o dispositivos IoT deben permanecer operativos solo con la energía de baterías durante extensos períodos de meses o años. Por otra parte, dichos dispositivos suelen estar instalados en lugares remotos de difícil acceso para el recambio de sus baterías. Existen diversos enfoques respecto al uso eficiente de la energía para los dispositivos IoT [31] [28].

En este escrito se analiza experimentalmente qué factores afectan la variabilidad del consumo de energía en aplicaciones IoT soportadas por APIs WEBs. El objetivo es revisar si existen elementos de una API WEB que permitan optimizar el consumo energético. El consumo energético cobra relevancia en el diseño de estas APIs porque las aplicaciones móviles y dispositivos IoT deben tener la capacidad de funcionar durante extensos períodos de tiempo, sin conexión a la red eléctrica gracias al uso de baterías. Dichos dispositivos deben optimizar el consumo a fin de poder realizar la comunicación con el servidor y procesar los datos de entrada/salida de periféricos, solamente con la energía almacenada en sus baterías. La otra razón que impacta en las restricciones del consumo es el reducido tamaño de las baterías, ya que muy a menudo los dispositivos IoT deben poder alojarse en espacios limitados, o deben ser los suficientemente livianos para poder acoplarse a vehículos autónomos.

Este documento está organizado de la siguiente manera. En la sección 2 se mencionan los trabajos que están relacionados con el experimento. En la sección 3 se explica el marco conceptual y se proporciona una breve descripción sobre las APIs WEB. En la sección 4 se expone la metodología utilizada en el experimento. En la sección 5 se presentan los resultados, y en la sección 6 se discute acerca de las implicaciones del escrito y los interrogantes abiertos, para trabajos futuros.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

2. TRABAJOS RELACIONADOS

Existen varias propuestas para el análisis del consumo energético en aplicaciones móviles y del tipo IoT. En algunos casos se propone analizar la latencia en los protocolos de comunicación, como en el caso del tipo CoAP (Constrained Application Protocol) [24] [6] y su comparación con el protocolo HTTP. En trabajos como el de Myasnikov et al. [27] se hace una revisión de la literatura de los métodos para poder estimar el consumo energético en aplicaciones móviles, se describen tanto *frameworks* que permiten estimar el consumo según el tipo de código de software como también los desarrollos de *testbeds* de medición específicos para cada aplicación. Según el escrito mencionado se han realizado estudios que tratan de estimar el consumo de energía en aplicaciones móviles a partir del código que compone las aplicaciones [21] [1] [13] [22]. En otros estudios, como el de Cherifi et al. [3] se compara una simulación del consumo energético con los valores reales obtenidos a través de un *testbed* para una red de sensores inalámbricos basados en ContikiOS [4] y HTTP. Recientemente, se han publicados trabajos como el de Labib et al. [19] en donde se analiza el consumo de aplicaciones basadas en SoC de bajo costo utilizando distintos protocolos como BLE (Bluetooth Low Energy) [20] y ESP-NOW [7]. Cabe destacar que no se han encontrado trabajos que analicen el consumo para un nodo IoT basado en la arquitectura del SoC ESP32 [32] comunicado a través de una API WEB.

3. MARCO CONCEPTUAL

Existen diversas descripciones del significado de API (Application Programming Interface) como el caso de [8] y [34]. Según la literatura, la API actúa como un intermediario entre el usuario de la aplicación en el *frontend* y la lógica en el *backend*. Uno de los principales beneficios de una API es la encapsulación, es decir, que se encapsula la lógica de programación para que el usuario no pueda verla. Con este argumento, proveedores de aplicaciones de software pueden hacer disponibles sus servicios a usuarios externos sin necesidad de revelar o exponer la estructura lógica. Una API además actúa como una interface estándar para que distintas aplicaciones corriendo en diversos dispositivos puedan acceder a la lógica del *backend*. Una de las características de esta interface es que en todos los casos que desde el *backend* se realice la misma petición, el *frontend* provee el mismo resultado.

Una API WEB se emplea para que determinada funcionalidad de un servicio en la nube pueda ser utilizado por desarrolladores para sus propias aplicaciones. Estas aplicaciones pueden ser instaladas en distintos tipos de *hardware* tales como dispositivos móviles, *tablets*, *smart* TVs, etc. Los desarrolladores pueden programar sus propias aplicaciones con el empleo de funcionalidades externas aportadas por servicios. Por ejemplo, una de las funcionalidades de Facebook API WEB permite compartir información de sus usuarios, y en el caso de Google API WEB facilita la consulta de mapas. Las APIs mejoran la experiencia del uso de cualquier aplicación gracias al acceso a funcionalidades externas.

Durante el proceso de definición de una API WEB se establece un contrato, en el mismo se deben predeterminar aspectos como el tipo de funcionalidad que dará la API WEB, su *endpoint* dado por la URL en la que el usuario podrá consumir los datos, los parámetros de entrada-salida tales como formato, tipos y nombres de los datos. Así también el SLA (service-level agreement) que define la disponibilidad, tiempo de respuesta del servicio, el número de consultas por unidad de tiempo que se puedan realizar, requerimientos legales como licencias comerciales, marcas registradas, pagos por uso, documentación de uso y soporte, tutoriales, y especificaciones de cómo los usuarios deberían registrarse antes de consumir la API WEB.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

Debido a estas características, el contrato otorga más confiabilidad para el consumo de la API WEB ya que las actualizaciones en la interface están predefinidas. De esta manera se minimizará el impacto y los defectos en el uso de la aplicación debido a los cambios de versión en la interface de la API WEB.

3.1. RESTful API WEB

REST (Representational state transfer) es un estilo de arquitectura de software que define algunos requerimientos específicos. Para que una API sea considerada RESTful [10] [33] se deben cumplir las siguientes condiciones:

• Client-Server: Se necesita una interface entre el cliente y el servidor, los cuales se comunican a través de la misma y son independientes entre sí. El cliente y el servidor se pueden cambiar siempre que la interface permanezca sin cambios. Las peticiones o "request" siempre provienen del lado del cliente.
• Code on Demand: El cliente debe entender y poder ejecutar el código que descarga del servidor. Ejemplos de esto son Java applets, scripts, plug-ins, etc. Este requerimiento es opcional.
• Cache: Este requisito indica que puede haber un cache para almacenar la información que se pide desde el cliente recurrentemente. El cache puede estar localizado en el servidor o en el cliente.
• Layered System: Puede haber múltiples capas en la arquitectura con diferentes funciones tales como almacenamiento y encriptación.
• Uniform Interface: Las aplicaciones en el servidor y cliente se pueden desarrollar independientemente mientras se mantengan consistentes con la interface.
• Stateless: La comunicación entre cada petición del cliente al servidor se realiza sin tener en cuenta el estado de la petición anterior, es decir, que cada una de las mismas deben ser independientes.

Cuando un cliente consume datos de una API RESTful la misma trasfiere una representación del estado del recurso. Existen varios formatos para transmitir la información via HTTP. Los datos transmitidos pueden representarse en formatos tales como JSON (Javascript Object Notation) o XML (Extensible Markup Language).
En las aplicaciones RESTful se definen distintos métodos para que el cliente pueda acceder al consumo de la API en el servidor a través de HTTP. En el protocolo HTTP se pueden realizar los siguientes 5 métodos de peticiones:

• GET: Para la lectura de datos.
• POST: Para crear datos.
• PUT: Para actualizar o cambiar datos ya existentes.
• PATCH: Para actualizar o cambiar datos ya existentes, pero al mismo tiempo modificar atributos.
• DELETE: Para borrar datos.

Estas acciones son básicamente las mismas operaciones que se realizan con el ciclo de vida de los datos en una base de datos (CREATE, READ, UPDATE, y DELETE). Cuando se realiza la petición a la API WEB el protocolo HTTP retorna un código de estado en formato JSON o XML según el resultado de la operación. Los códigos de estado son los siguientes:

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

• 200 OK: Significa que la petición fue exitosa. Los métodos GET, PUT o PATCH podrían devolver este código en caso de éxito.
• 201 Created: Indica que el pedido de POST fue exitoso y el nuevo registro ha sido creado.
• 204 No Content: Se refiere a que una petición DELETE fue exitosa.
• 400 Bad Request: Significa que ha ocurrido un error en la petición del cliente al servidor. Por ejemplo, podría ser un error en la sintaxis del formato JSON.
• 401 Unauthorized: Se obtiene cuando en la petición del cliente al servidor falta, o no se pudo validar la autenticación.
• 403 Forbidden: Ocurre cuando la petición al recurso está prohibida.
• 404 Not Found: Indica que el recurso no existe.

4. METODOLOGÍA

Con el objetivo de analizar los factores que afectan el consumo en una aplicación IoT se propone replicar la experiencia de Labib et al. [19] a través de una implementación típica de IoT. En el trabajo Labib et al. se realizaron mediciones de consumo para un nodo basado en el ESP32 cuando se transmiten datos por bluetooth y por WIFI a través del protocolo ESP-NOW en la red local. En el experimento se propone presentar los valores de consumo medidos en el mismo SoC, pero empleando una API WEB para la conexión de la app IoT. La configuración propuesta se indica en la figura 1.

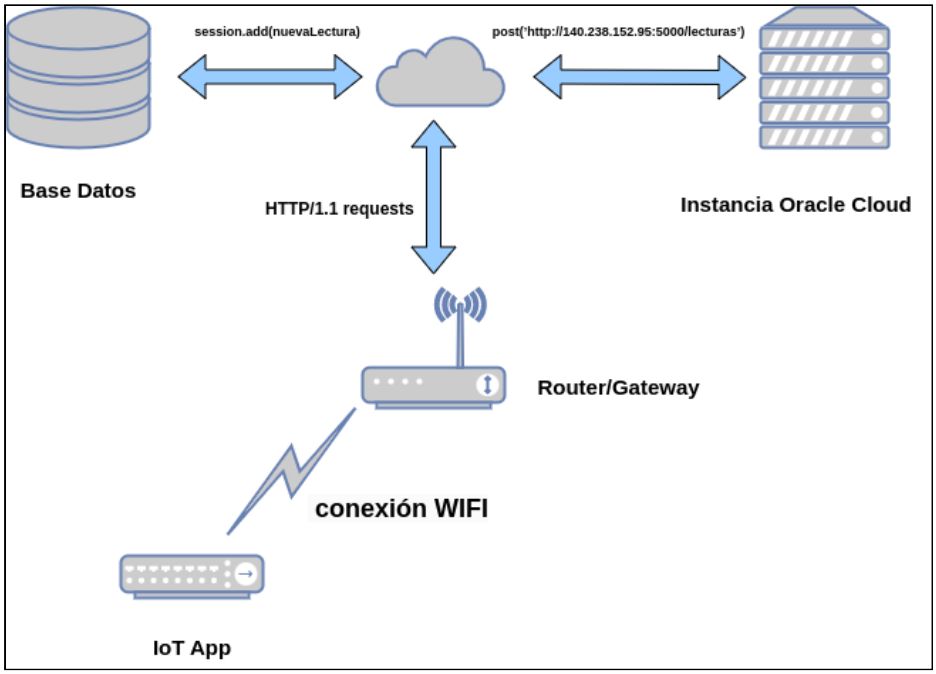

Figura 1 Diagrama de la implementación.

Para poder medir el consumo energético se programó una aplicación IoT de testeo que transmite la información a través de una red WIFI al *router* o *gateway*. Esta app almacena datos en una base de datos remota a través de una API WEB programada en una instancia de Oracle cloud [5]. El objetivo es poder registrar los datos de consumo de la app cada vez que la misma invoca a la API WEB.
La app de IoT se codificó en MicroPython sobre un módulo Esp32. El pseudocódigo de la app IoT se muestra en el cuadro 1 y el código completo se puede consultar en el apéndice.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

```
importar librerías
conectarse a la red WIFI
diccTemperatura = 0
while True do
        contar 3 segundos
        TempRandom = valor random entre 2 y 40
        Se hace un POST a la API con TempRandom
end
```

Cuadro 1 Pseudocódigo de la app IoT

La API que corre sobre el servidor de Oracle cloud se realizó en Python y Flask, su pseudocódigo se indica en el cuadro 2 (código completo en apéndice).

```
importar librerías
conexión a la base de datos
se declara clase Temp en la base de datos
se instancia una clase Flask
se declara el endpoint lecturas de la API
se define la función add para escritura en la base de datos
se escribe la temperatura en la base de datos
```

Cuadro 2 Pseudocódigo de la API WEB

Para tomar los datos de consumo se utilizó un sensor INA219 [15] conectado a la aplicación IoT a través de otro módulo ESP32 [32] (se utilizó un segundo módulo ESP32 con el objeto de no contaminar las mediciones de la app con el consumo propio del sensor). Este módulo registra y almacena los datos en un archivo de texto para su posterior análisis. La información del consumo es muestreada cada 0.2 segundos, es decir, 5 veces por segundo.
En la figura 2 se muestra el esquema de conexiones.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

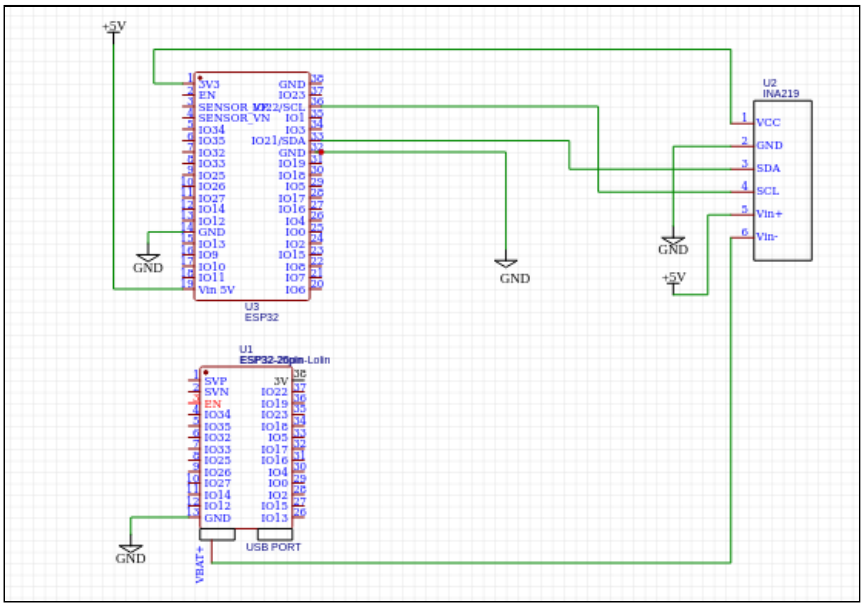

Figura 2 Esquema de las conexiones para la toma de mediciones de consumo

El pseudocódigo para registrar las mediciones se indica en el cuadro 3 (código completo en apéndice).

importar librerías
se inicializa sensor INA219
se define función para leer corriente
se define función para leer voltaje
while *True* **do**
    contar 0.2 segundos
    guardar datos de consumo en un archivo
end

Cuadro 3 Pseudocódigo de la medición del consumo

5. RESULTADOS

Durante la experiencia se registraron 327 mediciones de consumo de corriente y voltaje en la batería, los valores de potencia se obtuvieron del producto de la corriente por el voltaje.
En la figura 3 se muestran las mediciones tomadas durante un lapso de 3 minutos. Se indica la variación de corriente, voltaje y potencia consumida.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

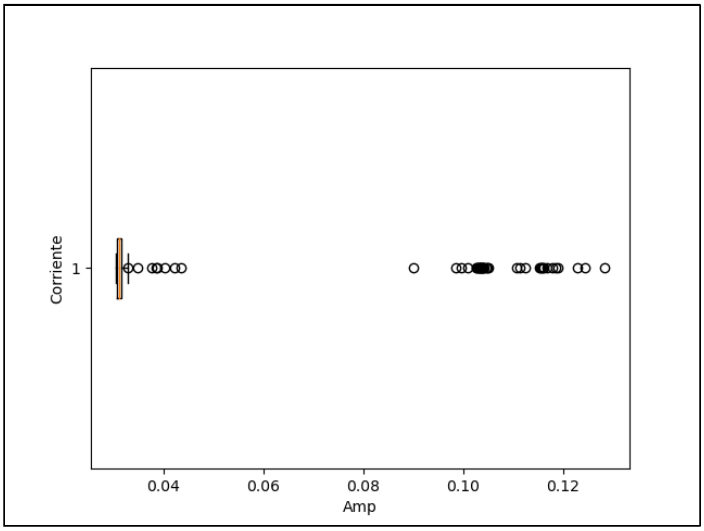

Figura 3a Variabilidad en la corriente de la batería

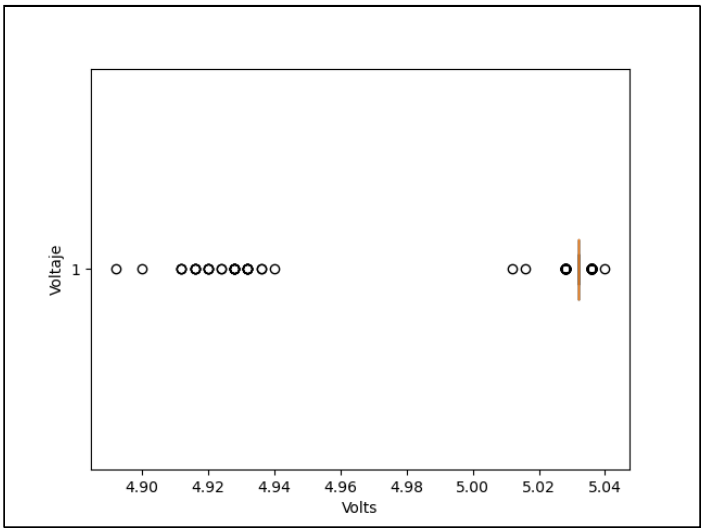

Figura 3b Variabilidad en el voltaje de la batería

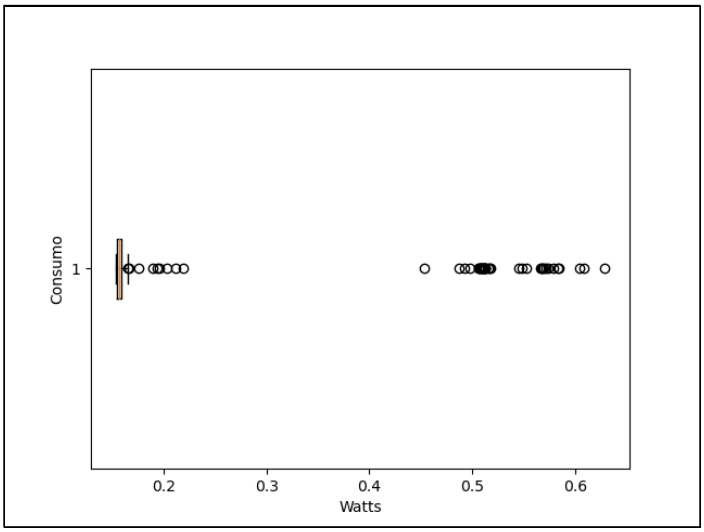

Figura 3c Variabilidad en el consumo de la batería

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

Según los datos registrados, en el momento de mayor consumo se produce un pico de corriente de 128.4 mA y la tensión de batería baja a 4.9 V lo que indica una potencia de 0.629 W. Por otra parte, durante los momentos de reposo el consumo se estabiliza en 30.4 mA, la tensión de la batería se eleva a 5.032 V con lo que el consumo baja a 0.152 W. Entre el máximo consumo de 0.629 W y el mínimo de 0.152 W existe un incremento de 4.13 veces.
La justificación estadística de las 327 mediciones tomadas se puede analizar a través de un diagrama de Pareto en donde los valores por rango están ordenados por frecuencias de ocurrencia. En la figura 4 se indican los Paretos de mediciones de corriente y voltaje respectivos. En la figura 4a los valores de corriente más frecuentes comprenden un rango de entre 30.4 mA a 42.4 mA, con un total de 290 mediciones, lo que correspondería a los periodos de menor consumo. Los siguientes valores en frecuencia corresponden al rango de 102.4 mA a 114.4 mA, estos valores se obtuvieron 20 veces. Finalmente, los mayores valores de corriente se obtuvieron 11 veces con un rango de entre 114.4 mA y 126.4 mA. El mismo análisis se puede realizar sobre el Pareto de valores de voltaje en la figura 4b. En este caso se aprecia que los valores de voltaje más frecuentes estuvieron en el rango de 5.028 V y 5.045 V con 250 mediciones, esto se verifica durante los periodos de menor consumo. En la 4ª barra de frecuencia se pueden observar 14 mediciones correspondientes al mayor consumo con un rango de voltajes de entre 4.909 V y 4.926 V. Los valores de la 5ª barra se pueden considerar *outliers* bajo este análisis debido a su baja probabilidad de ocurrencia.

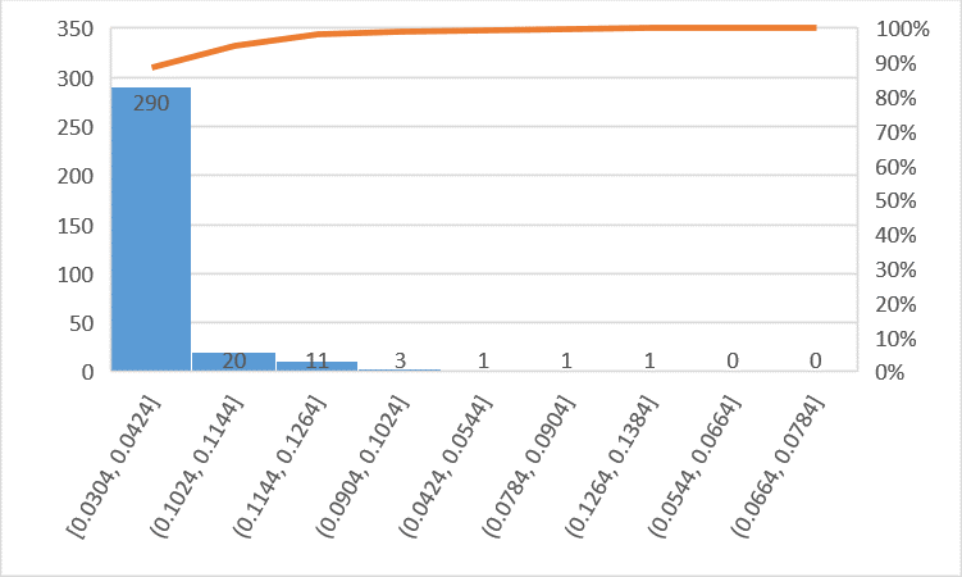

Figura 4a Pareto de mediciones de corriente

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

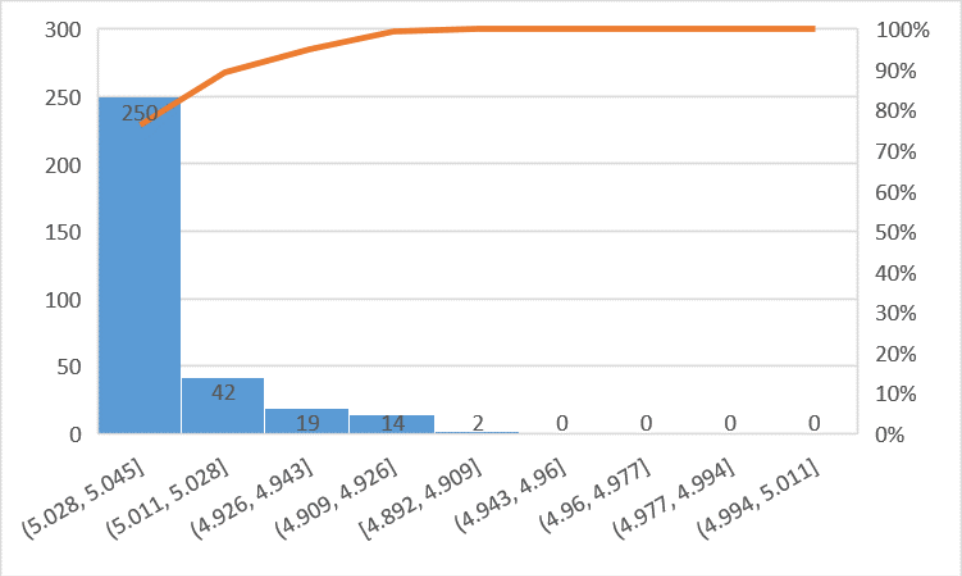

Figura 4b Pareto de mediciones de voltaje

6. DISCUSIÓN Y CONCLUSIONES

En este estudio se examinó como varía el consumo de energía en una aplicación IoT que utiliza una API WEB. Los resultados mostraron que el consumo máximo de energía ocurre cuando el hardware activa su transmisor de radiofrecuencia (RF) para conectarse al *router*. Este resultado ya se ha verificado en el escrito de Labib et al. [19] con diferentes métodos de comunicación y protocolos.

Los datos mostrados en el experimento indican que el pico de consumo se da cuando la aplicación llama a la API WEB para transferir los datos, esto en efecto concuerda con los datos obtenidos por otros trabajos en donde durante la llamada a la API WEB se pone en funcionamiento el modulador y el transmisor de RF para la comunicación con el *router*.

En propuestas anteriores, la estimación del consumo de aplicaciones móviles se realizó por análisis del código que compone el software [21] [1] [13], o utilizando un servicio corriendo en el dispositivo [22]. Otros trabajos efectuaron la medición del consumo utilizando hardware de diseño específico para nodos de IoT [18] [23].

Este escrito se diferencia de las propuestas nombradas, ya que se basa en la medición directa del consumo en el nodo IoT utilizando un sensor de bajo costo empleando el INA219 [15], en segundo lugar, este trabajo intenta medir el consumo durante el establecimiento de comunicación con una API WEB utilizando el protocolo HTTP sobre una red WIFI y no los protocolos Bluetooth o ESP-NOW como en el caso de Labib et al. [19]. En el estudio, solo se utilizó una llamada específica (POST) a la API WEB para transmitir datos de longitud fija, lo que significa que solo se probó un tipo de interacción con la API y no se exploraron otras posibles llamadas o comportamientos. En consecuencia, no se pudo determinar si existen otras formas de interactuar con la API que puedan producir resultados diferentes. Este experimento tiene aplicaciones prácticas en la medición del consumo en dispositivos IoT. Al adaptar el método a otros dispositivos IoT, se podría configurar el muestreo del consumo y registrar los datos para su posterior análisis. Esto permitiría a los usuarios monitorear el consumo de energía de sus dispositivos y tomar decisiones sobre su uso.

Después de llevar a cabo la experiencia, surgieron algunas cuestiones que necesitan ser exploradas con mayor detalle. Al parecer, el consumo de energía en los nodos IoT está directamente relacionado con el tiempo que tarda el dispositivo en establecer la comunicación con el *router* mediante el modulador de RF. Además, este tiempo está relacionado con el tipo

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

de *hardware* utilizado en el SoC, lo que implica que no es posible generalizar los resultados del consumo de energía en los nodos IoT basados en otro tipo de *hardware*.

El tiempo que se tarda en establecer la comunicación con el *router* está vinculado directamente con el protocolo de transmisión y la cantidad de datos (payload) que se envían. En otras palabras, si se desea reducir el consumo de energía, es necesario analizar las variables que intervienen en el protocolo para determinar qué aspectos se pueden optimizar.

En cuanto al diseño de la API WEB, existen varias capas de abstracción involucradas hasta llegar al control del hardware específico, lo que dificulta la calibración del tiempo de transmisión con los niveles de abstracción disponibles durante la programación de la API. Para calibrar u optimizar el tiempo de transmisión, se deben modificar parámetros del *hardware* involucrado con lenguajes de bajo nivel, como C, Rust o ensamblador. Esto permitiría ajustar el nivel de potencia que utiliza el transmisor de RF del *modem* a través de la librería del fabricante del SoC y reducir los tiempos de activación del modem en su conexión con la red WIFI.

En resumen, es necesario profundizar en el análisis de las variables que influyen en el consumo de energía de los nodos IoT para optimizar su rendimiento y eficiencia energética.

Por otro lado, se puede ahorrar energía de la batería mediante la activación del modo de hibernación o *deep sleep* del CPU durante los períodos de inactividad, lo que implica reducir la frecuencia de llamadas a la API. Es importante destacar que al despertar la CPU se produce un pico de consumo que debe ser integrado durante el funcionamiento para determinar si realmente hay un ahorro de energía en la batería. Algunos diseños optan por disminuir el consumo apagando el *modem* durante el proceso de cálculo y comunicación con los periféricos, encendiéndolo solamente antes de la transmisión de datos.

Es fundamental tener en cuenta el proceso de conexión a la red WIFI como un factor clave a considerar. Todo lo relacionado con la reducción del consumo depende en gran medida del hardware utilizado y de la sincronización y ajuste de las tareas que debe realizar el sistema IoT. Es importante tener en cuenta que la adopción de una arquitectura de bajo consumo no solo depende de la elección del hardware adecuado, sino también de la implementación eficiente de algoritmos y la adopción de técnicas de gestión de energía.

Otro posible factor que podría afectar el consumo eléctrico de los nodos IoT es el diseño de la interfaz de programación de aplicaciones (API) que se utiliza para su comunicación. En este sentido, sería interesante analizar si existen diferencias en el consumo al implementar diversas tecnologías para las APIs. En este estudio, se utilizó una API basada en Flask y Python, aunque existen otras alternativas disponibles. Por lo tanto, sería necesario llevar a cabo experimentos utilizando otros tipos de tecnologías para comparar los resultados en cuanto al consumo de energía.

Es posible que una API basada en HTTP no sea el método más eficiente para utilizar en una aplicación de IoT si lo que se busca es reducir el consumo eléctrico. En este sentido, podría ser conveniente emplear el protocolo MQTT, el cual es más efectivo en términos de consumo de energía en una red IoT. Como trabajo futuro sería importante analizar y comparar los resultados de consumo eléctrico en los nodos IoT utilizando diferentes tecnologías de APIs para determinar cuál es la más adecuada para su implementación en este tipo de aplicaciones. En principio, se podría comparar el consumo en una API basada en HTTP como la implementada en este escrito con una API en MQTT.

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

REFERENCIAS

1. AHMAD, R. W., NAVEED, A., RODRIGUES, J. J. P. C., GANI, A., MADANI, S. A., SHUJA, J., MAQSOOD, T., & SAEED, S. (2019). Enhancement and assessment of a code-analysis-based energy estimation framework. IEEE Systems Journal, 13(1), 1052–1059. https://doi.org/10.1109/jsyst.2018.2823733
2. BORMANN, C., ERSUE, M., & KERANEN, A. (2014). Terminology for constrained-node networks. RFC Editor.
3.CHERIFI, N., GRIMAUD, G., VANTROYS, T., & BOE, A. (2015). Energy consumption of networked embedded systems. 2015 3rd International Conference on Future Internet of Things and Cloud.
4. contiki: The official git repository for Contiki, the open source OS for the Internet of Things. (n.d.).
5. Discover the next Generation cloud platform. (n.d.). Oracle.com. Retrieved April 18, 2023, from https://www.oracle.com/cloud/
6. DIZDAREVIĆ, J., CARPIO, F., JUKAN, A., & MASIP-BRUIN, X. (2019). A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration. ACM Computing Surveys, 51(6), 1–29. https://doi.org/10.1145/3292674
7. Esp-now. (n.d). Espressif.com. Retrived April 15, 2023, from https://www.espressif.com/en/products/software/esp-now/overview
8. FAQ - OpenAPI Initiative. (2016, October 11). OpenAPI Initiative. https://www.openapis.org/faq
9. FERREIRA, H. G. C., DIAS CANEDO, E., & DE SOUSA, R. T. (2013). IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and arduino. 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob).
10. Fielding dissertation: CHAPTER 5: Representational state transfer (REST). (n.d.). Uci.edu. Retrieved April 15, 2023, from https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
11. GARG, H., & DAVE, M. (2019). Securing IoT devices and SecurelyConnecting the dots using REST API and middleware. 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU).
12. HAHM, O., BACCELLI, E., PETERSEN, H., & TSIFTES, N. (2016). Operating systems for low-end devices in the internet of things: A survey. IEEE Internet of Things Journal, 3(5), 720–734. https://doi.org/10.1109/jiot.2015.2505901
13. HAO, S., LI, D., HALFOND, W. G. J., & GOVINDAN, R. (2012). Estimating Android applications' CPU energy usage via bytecode profiling. 2012 First International Workshop on Green and Sustainable Software (GREENS).
14. HEJAZI, H., RAJAB, H., CINKLER, T., & LENGYEL, L. (2018). Survey of platforms for massive IoT. 2018 IEEE International Conference on Future IoT Technologies (Future IoT).
15. INA219. (n.d.). Www.ti.com. Retrieved April 15, 2023, from https://www.ti.com/product/INA219
16. Individuals using the Internet (% of population). (n.d.). World Bank Open Data. Retrieved April 15, 2023, from https://data.worldbank.org/indicator/IT.NET.USER.ZS
17. ITU-T recommendation database. (n.d.). ITU. Retrieved April 15, 2023,from https://handle.itu.int/11.1002/1000/11559

18. JIANG, X., DUTTA, P., CULLER, D., & STOICA, I. (2007). Micro power meter for energy monitoring of wireless sensor networks at scale. 2007 6th International Symposium on Information Processing in Sensor Networks.
19. LABIB, M. I., ELGAZZAR, M., GHALWASH, A., & ABDULKADER, S. N. (2021). An efficient networking solution for extending and controlling wireless sensor networks using low-energy technologies. PeerJ. Computer Science, 7(e780), e780. https://doi.org/10.7717/peerj-cs.780
20.Learn about Bluetooth. (n.d.). Bluetooth® Technology Website. Retrieved April 15, 2023, from https://www.bluetooth.com/learn-about-bluetooth/
21.LI, X., & GALLAGHER, J. P. (2016). An energy-aware programming approach for mobile application development guided by a fine-grained energy model. In arXiv [cs.SE]. http://arxiv.org/abs/1605.05234
22. LI, Y., CHEN, H., & SHI, W. (2014). Power behavior analysis of mobile applications using Bugu. Sustainable Computing Informatics and Systems, 4(3), 183–195. https://doi.org/10.1016/j.suscom.2014.07.002
23. LIM, R., FERRARI, F., ZIMMERLING, M., WALSER, C., SOMMER, P., & BEUTEL, J. (2013). FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. Proceedings of the 12th International Conference on Information Processing in Sensor Networks.
24. LUDOVICI, A., MORENO, P., & CALVERAS, A. (2013). TinyCoAP: A novel constrained application protocol (CoAP) implementation for embedding RESTful web services in wireless sensor networks based on TinyOS. Journal of Sensor and Actuator Networks, 2(2), 288–315. https://doi.org/10.3390/jsan2020288
25. MADDEN, N. (2021). API Security in Action. Manning Publications.
26. MUN, D.-H., DINH, M. L., & KWON, Y.-W. (2016). An assessment of internet of things protocols for resource-constrained applications. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).
27. MYASNIKOV, V.M., SARTASOV, S., SLESAREV, I., & GESSEN, P. (2020). Energy Consumption Measurement Frameworks for Android OS: A Systematic Literature Review.
28. RANA, B., SINGH, Y., & SINGH, P. K. (2021). A systematic survey on internet of things: Energy efficiency and interoperability perspective. Transactions on Emerging Telecommunications Technologies, 32(8). https://doi.org/10.1002/ett.4166
29. RAO, S., CHENDANDA, D., DESHPANDE, C., & LAKKUNDI, V. (2015). Implementing LWM2M in constrained IoT devices. 2015 IEEE Conference on Wireless Sensors (ICWiSe).
30. *Railway*. (n.d.). Railway. Retrieved April 22, 2023, from https://railway.app/
31. Ray, P. P. (2016). A survey of IoT cloud platforms. Future Computing and Informatics Journal, 1(1–2), 35–46. https://doi.org/10.1016/j.fcij.2017.02.001
30. SANCHEZ, J. L. C., BERNAL BERNABE, J., & SKARMETA, A. F. (2018). Integration of anonymous credential systems in IoT constrained environments. IEEE Access: Practical Innovations, Open Solutions, 6, 4767–4778. https://doi.org/10.1109/access.2017.2788464
31. SILVA, P. V. B. C. DA, TACONET, C., CHABRIDON, S., CONAN, D., CAVALCANTE, E., & BATISTA, T. (2023). Energy awareness and energy efficiency in internet of things middleware: a systematic literature review. Annals of Telecommunications - Annales Des Télécommunications, 78(1–2), 115–131. https://doi.org/10.1007/s12243-022-00936-5

32. The Internet of Things with ESP32. (n.d.). Esp32.net. Retrieved April 15, 2023, from http://esp32.net/
33. What is a REST API? (n.d.). Redhat.com. Retrieved April 15, 2023, from https://www.redhat.com/en/topics/api/what-is-a-rest-api
34. What Is An API (Application Programming Interface)? (n.d.). Amazon.com. Retrieved April 15, 2023, from https://aws.amazon.com/what-is/api/

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

APÉNDICE

Código API

##
##
Entre las líneas 1 y 5 se importan las librerías necesarias. En la línea 10 se inicializa un diccionario (datos) con un solo registro (temperatura) para capturar los datos. En la línea 14 se define una salida del microcontrolador para manejar un LED del mismo y verificar visualmente el funcionamiento del hardware. Entre las líneas 16 y 23 se define un bucle infinito que se repite cada 3 segundos. En la línea 13 se asigna al registro temperatura un valor aleatorio entre 2 y 40 con el objeto de simular los datos. En la línea 19 se llama a la API realizando un POST a la dirección http://140,238,152,95:5000/lecturas. En la línea 21 se muestra en la consola los valores en formato JSON que se enviaron a la API.
##
##
########## Código en MicropPython de la aplicación IoT ##########################
##
##
```
1 import random
2 import urequests
3 import connectWIFI
4 from machine import Pin
5 from time import sleep
6 connectWIFI.connect()
7
8 headers = {"Content-Type": "application/json; charset=utf-8"}
9
10 data = {
11 "temperatura": 00,
12 }
13
14 p22 = Pin(22, Pin.OUT)
15
16 while True:
17 randTemp = random.uniform(2, 40)
18 data["temperatura"] = randTemp
19 response = urequests.post('http://140.238.152.95:5000/lecturas', headers=headers,
20 json=data)
21 print(response.text)
22 p22.value(not p22.value())
23 sleep(3)
```
##

##
##
Entre las líneas 1 y 7 se importan las librerías utilizadas. Entre las líneas 9 y 12 se configura la llamada a la base de datos en el server de Railway (*Railway*, n.d.). En las líneas 14 a 18 se

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

declara la clase que define la tabla temp en la base de datos. En la línea 24 se instancia una app de la clase flask. En la línea 47 se declara el endpoint lecturas. En la línea 48 se declara la función add_lectura para almacenar los datos de temperatura en la base de datos. Entre las líneas 52 y 58 se busca en la base de datos cuál fue el último registro para poder seguir ingresando datos. Entre las líneas 60 y 65 se escriben los nuevos datos en la base de datos.

```
################################################################################
################################################################################
#################### Código en Python de la API WEB ############################
################################################################################
################################################################################
1 from flask import Flask, jsonify, request
2 from sqlalchemy import create_engine
3 from sqlalchemy import Column, String, Integer, Float, DateTime, func
4 from sqlalchemy.ext.declarative import declarative_base
5 from sqlalchemy.orm import sessionmaker
6 from datetime import datetime
7 import json
8
9 db_string = "postgresql://postgres:h2AFk9Iy9pRbbfDf8WC2@containers-us-west-
30.railway.app
10 :5747/railway"
11 db = create_engine(db_string)
12 base = declarative_base()
13
14 class Temp(base):
15 __tablename__ = 'temp'
16 id = Column(Integer, primary_key=True)
17 temperatura = Column(Float)
18 fecha = Column(String)
19
20 Session = sessionmaker(db)
21 session = Session()
22 base.metadata.create_all(db)
23
24 app = Flask(__name__)
25
26 """
27############## Estas son endpoints de testeo de la  ##########################
28 @app.route('/')
29 def hola():
30 return ("hola")
31
32 @app.route('/otro')
33 def otraRuta():
34 return ("este es otro mensaje")
35
36 @app.route('/valores')
37 def hello():
38 return jsonify(lecturas)
39
```

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

```
40 @app.route('/valores/<int:id>', methods=['GET'])
41 def get_lectura(id):
42 lectura = next((lectura for lectura in lecturas if lectura['id'] == id), None)
43 if lectura:
44 ##########################################################################
45 """
46
47 @app.route('/lecturas', methods=['POST'])
48 def add_lectura():
49 lectura = request.get_json()
50 lecturaStr = lectura["temperatura"]
51
52 # Read
53 temp = session.query(Temp)
54 max_id = 0
55 for t in temp:
56 if max_id <= t.id:
57 max_id = t.id
58 max_id = max_id + 1
59
60 # Create
61 dt = datetime.now()
62 dtStr = dt.strftime("%X, %d %b %Y")
63 nuevaLectura = Temp(id=max_id, temperatura=lecturaStr, fecha=dtStr)
64 session.add(nuevaLectura)
65 session.commit()
66
67 app.run(host='0.0.0.0')
##########################################################################
```

##
##

En las líneas 1 y 4 se importan las librerías necesarias. Entre las líneas 14 y 33 se inicializa el sensor INA219. En la línea 36 se define la función que captura los valores de consumo de corriente. En la línea 42 se define la función para leer los valores de voltaje en la batería. En las líneas 46 y 47 se configura el archivo de texto para poder registrar los datos de consumo. Entre las líneas 49 y 58 se define un bucle infinito que toma los datos de consumo y los guarda en el archivo de texto cada 0.2 segundos. Luego la API guarda los datos en una base de datos de Postgresql alojada en un server de Railway.

##
##
########## Código en MicroPython de la aplicación para medición de consumo ########
##
##

```
1 from machine import I2C, Pin
2 import time
3 from time import time, sleep, gmtime
4 import connectWIFI
```

```
5
6 connectWIFI.connect()
7
8 """
9 INA219 Power Monitor interface.
10 Copyright GPL3.0 sergei.nz.
11 https://www.ti.com/lit/ds/symlink/ina219.pdf
12 """
13
14 MAX_CURRENT = 3.2 # Amps
15 CURRENT_LSB = MAX_CURRENT/(2**15)
16 R_SHUNT = 0.1 # Ohms
17 CALIBRATION = int(0.04096 / (CURRENT_LSB * R_SHUNT))
18
19 CONF_R = 0x00
20 SHUNT_V_R = 0x01
21 BUS_V_R = 0x02
22 POWER_R = 0x03
23 CURRENT_R = 0x04
24 CALIBRATION_R = 0x05
25
26 ADDRESS = 0x40
27
28 SDA = Pin(21)
29 SCL = Pin(22)
30 FREQ = 400000
31
32 i2c = I2C(sda=SDA,scl=SCL,freq=FREQ)
33 i2c.writeto_mem(ADDRESS, CALIBRATION_R ,(CALIBRATION).to_bytes(2, 'big'))
34
35
36 def read_current():
37 raw_current = int.from_bytes(i2c.readfrom_mem(ADDRESS, SHUNT_V_R, 2), 'big')
38 if raw_current >> 15:
39 raw_current -= 2**16
40 return raw_current * CURRENT_LSB
41
42 def read_voltage():
43 return (int.from_bytes(i2c.readfrom_mem(ADDRESS, BUS_V_R, 2), 'big') >> 3) * 0.004
44
45
46 f = open('data.txt', 'w')
47 f.write("I[Amp]" + ", " + "V[Vol]" + ", " + "W[Watts]" + ", " + "TimeStamp" + "\n")
48
49 while True:
50 corriente = round(read_current(), 4)
51 voltage = round(read_voltage(), 4)
52 potencia = voltage * corriente
53 tiempo = gmtime()
54 print("I=",corriente, "V=",voltage)
```

ICT-UNPA-349-2023
ISSN: 1852-4516
Aprobado por Resolución N° 0582/23-R-UNPA
https://doi.org/10.22305/ict-unpa.v15.n3.991

```
55 f.write(str(corriente) + ", " + str(voltage) + ", " + str(potencia) + ",
56 " + str(tiempo) + "\n")
57 print(gmtime())
58 sleep(0.2)
##############################################################################
```