SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE MORELIA
"José María Morelos y Pavón"

Instituto Tecnológico de Morelia
"José María Morelos y Pavón"

División de Estudios Profesionales
Departamento de Ingeniería Electrónica

tesis profesional

"Caracterización de paneles solares utilizando el internet de las cosas"

que para obtener el título de:

Ingeniero electrónico

presenta:

Jason Yair Corona Ventura

Director:

Gerardo Marx Chávez Campos

Morelia, Michoacán, México Marzo 2019

Jason Yair Corona Ventura: *Caracterización de paneles solares utilizando el internet de las cosas,* Caso de Estudio a Nivel Ingeniería, © Marzo 2019

Mesa de revisión:
Gerardo Marx Chávez Campos
Adriana del Carmen Téllez Anguiano
Miguelangel Fraga Aguilar
Rafael Lara Hernández

Localidad:
Morelia, Michoacán, México

Impresa:
Marzo 2019

Dedicado a mi madre y mis hermanos, mi familia...

Prólogo

El Internet de las cosas o "Internet of Things" es un tópico de gran relevancia en la actualidad, no solo por las ventajas de comunicación con sistemas vía remota, sino por su gran capacidad de producir datos y análisis; que posteriormente puede ser usado para minería de datos, entrenamiento de sistemas, modelado de sistemas, entre otros.

El presente trabajo muestra el desarrollo de un sistema que permite recolectar datos en un sistema fotovoltaico: corriente, voltaje, humedad, temperatura, irradiación solar, e información básica de un data logger como día, hora y mes. El sistema provee la capacidad de comunicación vía Wi-Fi, a través de una página web(html). La página permite la posibilidad de reconfigurar la forma en la que el sistema recolecta la información, datos de funcionamiento del sistema, mediciones en tiempo real, así como la opción de crear archivos con los datos recolectados por el sistema IoT.

A lo largo del trabajo de tesis se muestra el diseño y construcción del sistema, así como la integración física con el panel fotovoltaico, secciones de código de la operación básica, así como diagramas esquemáticos y circuitos impresos de una tarjeta-hija, de los sensores. La tarjeta de los sensores es adaptada para operar con la tarjeta de desarrollo de Texas Instruments del microcontrolador CC3200.

El sistema y los datos que arroja actualmente están siendo usados para el desarrollo del trabajo de tesis del alumno Ing. Oscar Lobato Nostroza, en el entrenamiento de una red neuronal que permita estimar la potencia que se produce en el panel solar, a partir de la información de los sensores y mediciones eléctricas.

Además este proyecto tiene la intención de contribuir y cooperar académicamente con otras universidades, a través de la reproducción del sistema para incrementar el número de datos e información que permite el desarrollo de modelos de redes neuronales. El primer acercamiento de esta intención se da con la estancia académica del alumno Ing. Oscar Lobato.

Toda la información de este proyecto está disponible en el **Open Science Framework** (OSF) bajo el nombre de `IoT System for Power Estimation on a Photovoltaic Panel` o en la liga: https://osf.io/a75cx/ del OSF.

Por último y no menos importante, me gustaría agradecer el empeño y esfuerzo realizado por el C. Jason Yair Corona Ventura. Alumno encargado del desarrollo del sistema y presente trabajo de tesis. Muchas gracias Jason y mucha suerte en tus próximas metas.

– Gerardo Marx Chávez Campos–

Resumen

El auge de la tecnología nos brinda servicios que facilitan la vida a costa de energía eléctrica, la demanda de la misma se traduce en aumentos al consumo de los combustibles fósiles, esto conlleva a su desgaste y contaminación del medio ambiente. Las energías renovables como alternativa a los combustibles fósiles a pesar de no ser contaminantes, tienen mayores tiempos de recuperación de la inversión y son mayormente utilizadas como un amortiguador del consumo global y no como fuente principal de energía. El motivo de la lenta aceptación a las energías renovables es su naturaleza variante que no otorga al usuario la seguridad de suministro constante que los combustibles fósiles proveen. La energía solar tiene potencial de convertirse en la fuente renovable más importante, es aprovechada mediante los paneles solares. Las principales problemáticas de los paneles solares son sus bajas eficiencias y la gran dependencia con la radiación solar incidente.Para amortiguar la incertidumbre sobre la generación eléctrica de un panel solar se desarrolló un sistema de monitoreo con capacidad de comunicación inalámbrica, almacenamiento de datos y bajo consumo energético que le otorgue autonomía y permita ser fácilmente adaptable sobre un panel solar, realizando mediciones con periodos controlados de tiempo que registren las variables de funcionamiento del panel solar y de comportamiento del medio en que se desempeña. Los datos registrados permiten que su comportamiento pueda ser observado y analizado dando a conocer las relaciones que existen entre las variables a las que está expuesto y cómo se ve afectado por ellas. Para casos donde la información sea suficientemente extensa, es posible llegar a análisis de predicción a corto plazo, permitiendo que el usuario pueda conocer la eficiencia del panel solar en un futuro, otorgando oportunidad para tomar precauciones o actuar con respecto a la información obtenida.

Índice general

Índice de figuras

Listado de secciones de código

Capítulo 1

Protocolo de investigación

1.1 Introducción

El acceso seguro a una fuente de energía constante es vital para el sustento de la sociedad moderna e incluso de la vida misma, esta dependencia de la energía eléctrica ocasiona un continuo uso de los combustibles fósiles ya que son fuente confiable de energía. La alta demanda de los combustibles fósiles conlleva múltiples desafíos y problemáticas como: agotamiento de reservas globales, calentamiento global y otros conflictos sociales que se generan por la escasez o el aumento de los precios. Las energías renovables representan la solución a los problemas generados por los combustibles fósiles, esto es gracias a que son en su mayoría de generación limpia de energía. La naturaleza inestable y variante de las fuentes de energía renovables ocasionan que su producción no sea constante, esto las limita a ser utilizadas como complemento al suministro energético, amortiguando la demanda total de combustibles fósiles.

La energía solar tiene potencial de convertirse en la fuente renovable más importante, es considerada prácticamente ilimitada [1]. La radiación solar está presente en todo el mundo, con mayor abundancia en aquellos países cercanos al ecuador de la tierra [2], sin embargo, no en todos los países existe infraestructura suficiente para aprovechar la energía potencial presente. EE.UU. a pesar que sus condiciones climatologías no son ideales para cosecha de energía solar, contando con niveles de radiación solar que van desde $3kWh/m^2$ hasta $5kWh/m^2$ promedio al día, tiene infraestructura para su aprovechamiento gracias al alto nivel de desarrollo tecnológico y su economía estable lo que resulta en crecimientos de hasta 119 % en la producción de energía solar, como se registró en el año 2016 [3]. La posición geográfica de México coloca al país en mejores condiciones para cosecha de energía solar contando con radiación solar promedio por día de $5kWh/m^2$, y regiones que llegan hasta valores de $6kWh/m^2$[4]. Contrario a la situación económica de EE.UU. en México la economía no permite el mismo aumento de la infraestructura para aprovechar la energía solar, sin embargo, iniciativas como "*Programa Nacional de Solidaridad, PRONALSOL*", que apoyan el desarrollo de infraestructuras de paneles solares en comunidades rurales, resultaron en un aumento de la producción de energía de fuente solar de 182 % en el periodo 2006-2012 [5], lo que se traduce en una generación aproximada de $5{,}55GWh/m^2$ en el sector residencial, donde el consumo es aproximadamente $208,333GWh/m^2$ al

año [6]. En el año 2013 se implementa en México la reforma energética, resultando en aumentos de capacidad energética instalada que fueron desde $3kW(2007)$ hasta $247{,}6MW(2016)$ a nivel nacional [7].

Actualmente, la eficiencia de los paneles solares oscila alrededor del 20 % [8]. Los paneles solares son fuertemente afectados por factores del medio [9, 10, 11] como la humedad con efectos negativos en la corriente entregada [12, 13], el ángulo de incidencia que varía la potencia de salida y en especial la temperatura reduciendo el voltaje generado [14, 15]. Para amortiguar el efecto ambiental constantemente se hacen cambios en los materiales semiconductores de los que se componen las celdas solares con objetivo de adaptar el comportamiento del panel y otorgar una mejor respuesta a la radiación solar, así como mejor resistencia a las variables del medio [16]. Las diferentes condiciones de trabajo de cada panel dentro de un arreglo de paneles resultan en diferencias en su comportamiento que al estar interconectados afecta a la producción total del sistema [17]. Las variaciones en comportamiento de los paneles solares representan incertidumbre en el consumidor al no ser una fuente constante y confiable de energía.

El rendimiento de un panel solar corresponde a la relación que existe entre la intensidad de la luz a la que está expuesto el panel y la energía que produce. Registrar las variables de comportamiento puede utilizarse para demostrar la relación que hay entre el medio y el rendimiento del sistema. A partir de dichos registros es posible estimar la potencia que se entrega. Para obtener los registros anteriores se requiere de un sistema de monitoreo que recolecte las variables y las almacene de manera estructurada para facilitar el acceso y manejo de información. Debido a que en algunos casos la ubicación al exterior donde se encuentra el panel solar dificulta el acceso constante para medición, se requiere que el sistema de recolección sea autónomo y capaz de mostrar en tiempo real las variables y/o el historial de sus mediciones.

1.2 Revisión del estado del arte

Los avances en las tecnologías de la información y comunicaciones mejoran la obtención y distribución de la información. Los sistemas de recolección se benefician obteniendo servicios que facilitan el monitoreo de procesos o variables resultando en información estructurada y fácilmente analizable. Un sistema de adquisición de datos convencional como el mostrado en el artículo [18] está compuesto por; hardware como sensores o transductores que se encarga de monitorear un proceso, comunicación mediante la cual se entrega la información y una computadora que la recibe. La etapa de recepción de la información puede dar lugar a aplicaciones como acciones de control o supervisión. Actualmente la tendencia de los sistemas de recolección es el Internet de las cosas (IoT, Internet of Things), esto debido a que transforma completamente la manera en que se obtiene, manipula y distribuye información remotamente. Los escenarios donde en un principio existía un operador observando el comportamiento del sistema y actuando con respecto a él, se han ido convirtiendo en sistemas autónomos que toman decisiones con base en el comportamiento en tiempo real de manera eficaz y rápida, restando responsabilidades al operador [19].

El monitoreo de signos vitales y variables relacionadas con la salud [20, 21, 22, 23] se ha visto ampliamente beneficiado del IoT, de igual manera el control y monitoreo remoto del hogar [24, 25, 26, 27] o ciudades [28, 29], esto es gracias a la comunicación que existe con dispositivos anteriormente aislados del exterior. Organizaciones como Dynatrace [30], Datadog [31], Amazon Web services [32], IBM Watson IoT [33], entre otros, venden servicios de almacenamiento en la nube para la información recolectada. Dichos servicios enfocados en software facilitan el monitoreo, la interpretación y el manejo de datos permitiendo que el hardware o variables específicas de un proceso se entregue a un tercero para su análisis, aumentando el alcance del monitoreo y sistema en general. Para necesidades más específicas

Hyperthings [34] ofrece servicios de control, monitoreo de rendimiento y estado relacionados con paneles solares, salud, energía, entre otros, teniendo sistemas diseñados para funcionar según se requiera, como se muestra en la Figura 1.1.

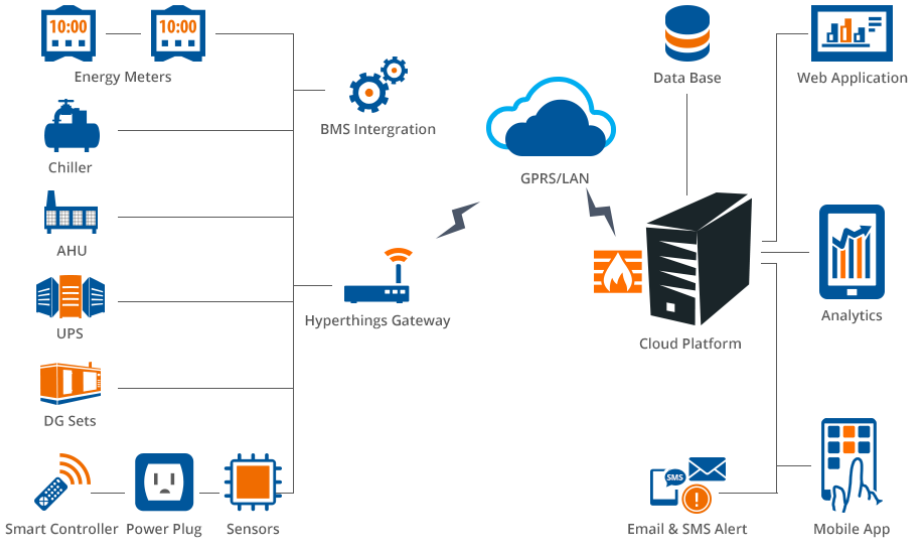


Figura 1.1: Sistema de monitoreo Hyperthings [34].

Los sistemas de monitoreo varían según cómo se requiera recolectar y manejar la información, así como la cantidad de sensores utilizados o el tipo de estos, aún así estos sistemas mantienen similitud en su estructura. Un sistema de monitoreo se compone por varias etapas, por ejemplo el mostrado en la Figura 1.2, siendo un sistema de monitoreo y control de un panel solar con IoT, se divide en 3 etapas de funcionamiento, cada una englobando aspectos clave; etapa de aplicación, de red y de sensado. La composición de estas etapas es en base a los requerimientos y alcances del sistema, para el caso de la etapa de sensado varían la cantidad o tipo de sensores y la comunicación entre ellos. Una vez obtenida la información por la etapa de sensado, la etapa de red se la distribuye, esto la hace accesible desde la etapa de aplicación. La información presente en la etapa de aplicación se manipula según las necesidades del usuario o las prestaciones que el sistema tenga; para este caso el sistema cuenta con comunicación bidireccional por lo que la información es interpretada y analizada con posibilidad de obtener una retroalimentación que modifique en tiempo real el comportamiento del sistema.

Un proceso o sistema que cuente con la recolección y entrega de sus variables, sin importar el método o la tecnología aplicada para conseguirlo, se convierte en un sistema inteligente capaz de comunicarse con su entorno. En el artículo [35] se muestra una problemática en cuanto a control energético industrial a la cual se le plantea solución con la implementación de un sistema de monitoreo basado en IoT. La implementación del sistema permite conocer el consumo energético en tiempo real, dando lugar a control y corrección del funcionamiento en base a la información, lo que reduce tiempos de detección de errores.

La variabilidad en la producción energética de un panel solar es un área de oportunidad sobre la cual se pueden implementar muchas posibles soluciones o mejoras que van desde cambios en la geometría del panel hasta sistemas de seguimiento de la luz solar. Es necesario conocer cómo reaccionan las variables del panel solar ante las variables del medio y en qué medida. Con esta información es posible dar al usuario la seguridad que espera, si bien su producción sigue siendo variable, en base a la información, es posible saber la magnitud de la variación.

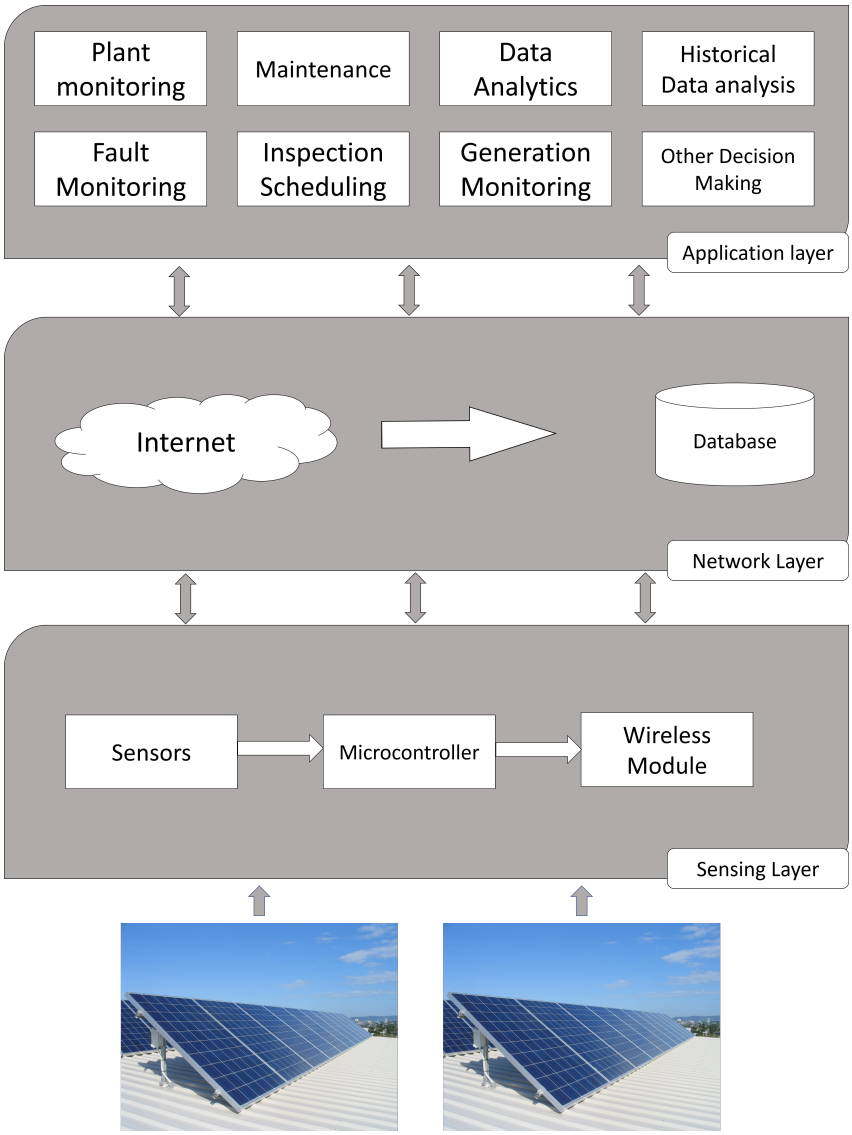


Figura 1.2: Sistema de monitoreo y control utilizando IoT [36].

1.3 Solución propuesta

Se propone un sistema de monitoreo remoto basado en IoT, utilizando un microcontrolador de bajo consumo energético controlado por un sistema operativo de tiempo real, integrando sensores digitales para medir las variables internas de un panel solar y las variables de interés del medio (luz, humedad y temperatura). Se ofrecen servicios de comunicación Wi-Fi para la entrega de datos o interacción con el usuario vía HTTP, capacidad de almacenamiento de datos de manera interna, batería integrada y un panel solar de baja potencia suficiente para mantener el sistema autónomo.

1.4 Objetivos

1.4.1 Objetivo general

Contribuir al estudio de las variables ambientales y sus efectos en los paneles solares, a través del desarrollo de un sistema basado en microcontrolador capaz de recolectar, almacenar de manera estructurada, y entregar datos de las mediciones, mediante servicios WEB. Para con los datos estudiar el comportamiento de un panel solar en relación al medio en que está expuesto.

1.4.2 Objetivos específicos

- Recolección periódica de variables disponibles en los sensores.
- Almacenamiento y estructurado de datos.
- Interacción del usuario con el sistema mediante servicios HTML.
- Control del consumo energético del sistema mediante modos de bajo consumo.
- Creación y adaptación del hardware para ser expuesto al medio.

1.5 Hipótesis

Un sistema de monitoreo remoto basado en IoT contribuirá a conocer la manera en que un sistema fotovoltaico y su entorno se relacionan entre si, a través del registro y almacenamiento de las variables significativas (luz, temperatura, humedad, corriente, voltaje y potencia) que pueden ser la base de un nuevo sistema para predecir su comportamiento.

1.6 Metodología

El desarrollo del sistema planteado requiere dividirse en múltiples etapas, para lograr el objetivo y validar la hipótesis se recolectan las variables significativas llevando registro de la fecha en que se realiza para ser acumuladas y presentadas como archivos vía una interfaz HTML como se muestra en la Figura 1.3.

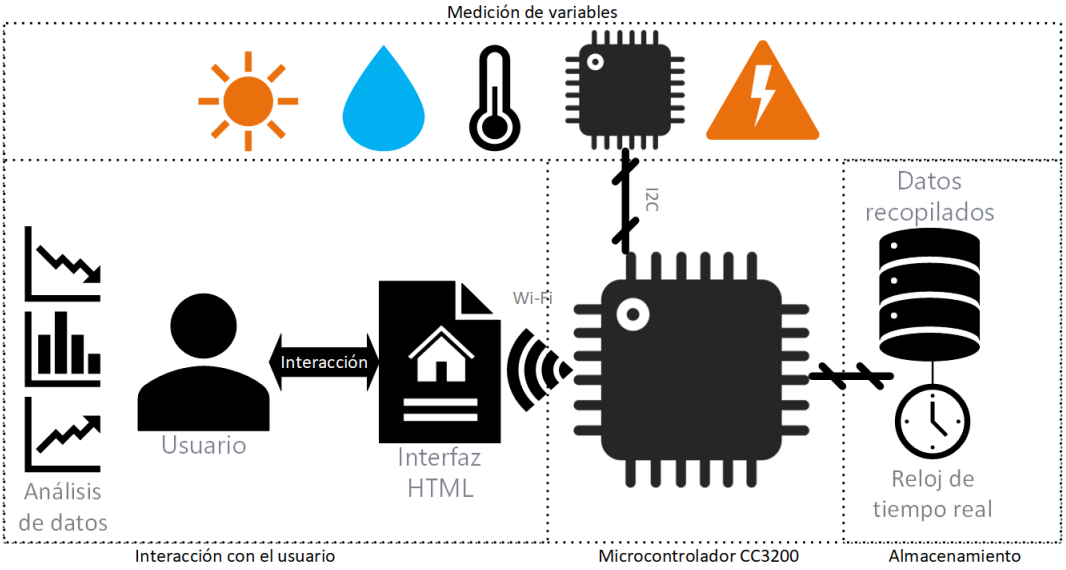


Figura 1.3: Metodología del sistema planteado.

Capítulo 2

Marco Teórico

2.1 Sistemas de recolección de variables

Un sistema de recolección se encarga de recoger información de un proceso, fenómeno o variable con el objetivo de obtener datos que reflejen su comportamiento general o específico, la información obtenida y su utilización dependen del ámbito de trabajo y si se requiere analizar o documentar. Los sistemas de recolección pueden ser tan sencillos como un simple muestreo de una temperatura dentro del hogar o tan complejos como un sistema de recolección y almacenamiento de variables remoto con múltiples sensores distribuidos dentro de una ciudad, todo según los requerimientos del usuario o las necesidades que se tengan.

Los sistemas de recolección más comunes suelen ser los sistemas de adquisición de datos (Data Acquisition Systems, DAQS por sus siglas en inglés). Estas tarjetas están implementadas de tal manera que sus entradas se encuentran acondicionadas para los sensores y transductores que se requieran usar según la variable que se desea medir, la recolección se hace directamente con una computadora personal (PC), mediante un bus de comunicación, como se ve en la Figura 2.1. Programas como Labview que cuentan con interfaces de diseño gráficas, facilitan la interacción con los sensores, ya que la extracción de valores se reduce a pasos como selección del tipo de sensor, el periodo de recolección y el tratado que se le da a los resultados.

Es importante resaltar que antes de implementar un sistema de adquisición de datos, las necesidades del sistema de recolección se tienen que tener bien establecidas ya que necesitan ser compatibles con las capacidades del sistema de adquisición, por ejemplo; si se requiere monitorear muchas variables en periodos de recolección pequeños se necesita alta velocidad de transmisión por lo que se requiere tarjetas con buses como PCI o Ethernet, mientras que si se necesita realizar mediciones de manera remota, es necesario considerar interfaces inalámbricas como Wi-Fi.

En general, con base en Figura 2.1 se puede decir, que un sistema de recolección digital se encuentra compuesto por sensores que de los cuales van a depender la necesidad o no por etapas de acondicionamiento y/o convertidores analógico-digital (Analogic-Digitl Converter, ADC por sus siglas en inglés),

Figura 2.1: Partes de un sistema DAQ.

seguidos por una comunicación para entrega de datos a la etapa de aplicación.

2.1.1 Sensores y transductores

Un sensor es un dispositivo capaz de detectar cambios de una variable o fenómeno físico reflejándola en un cambio en su comportamiento. Un transductor es un dispositivo capaz de percibir el cambio de una variable física y "traducirla" a otra [37]. Existe una basta variedad de sensores y transductores que para cuestiones de comparación se clasifican en dos tipos:

- Analógicos: entregan en sus terminales de salida valores que varían constantemente en el tiempo con relación proporcional al comportamiento de la variable de entrada que están midiendo, la magnitud medida puede encontrarse en la frecuencia de la señal de salida o en su voltaje, dependiendo del tipo de señal medida y del sensor. Estos sensores tienen como ventaja que suelen ser rápidos en comparación a los digitales, su principal desventaja es que normalmente requieren de instrumentación y acondicionamiento de señal.
- Digitales: un sensor digital cuenta con un transductor integrado que se encarga de transformar la variable y entregar un valor analógico que es recibido por un convertidor Analógico-Digital (ADC, por sus siglas en inglés) que convierte las variaciones analógicas a un valor binario que es desplegado entre sus terminales de salida por medio de un protocolo de comunicación serial. Tienen como desventaja los tiempos de conversión y su principal ventaja es no necesitar acondicionamiento extra.

2.1.2 Exactitud, precisión y sensibilidad en sensores

Para que un sistema de recolección sea confiable, es necesario que sus componentes también lo sean, por lo que los transductores y/o sensores utilizados al ser la interacción directa con el medio, deben entregar mediciones confiables que lo reflejen correctamente. Se busca entonces que los instrumentos utilizados, sean exactos, precisos y con sensibilidad suficiente para detectar la variable [38].

La exactitud en una medición es interpretada como la diferencia existente entre el valor real que se está midiendo y el valor que en realidad se está entregando, entre menor sea la diferencia más exacta será la medición. Esta característica está relacionada con la manera de obtención de la variable y su acondicionamiento, ya que es posible que un sensor a causa de su acondicionamiento se encuentre obteniendo mediciones exactas pero entregándola erróneamente a causa de su mala calibración.

Los dispositivos de medición deben ser capaces de repetir las mediciones bajo condiciones donde la variable medida no cambie, a esto se le llama precisión. En otras palabras, se refiere a la capacidad del sensor de entregar la misma medición para la misma entrada en diferentes momentos o condiciones. Esta característica es altamente requerida para variables de comportamiento lento como suele suceder con la temperatura.

La combinación de los conceptos anteriores vuelve a un sensor altamente confiable a la hora de registrar cambios de variable, sin embargo, es necesario conocer el tipo de variable que se mide y la utilidad que se le dará a la información para determinar la sensibilidad requerida en el sistema. La sensibilidad de un sensor, es el cambio mínimo en la variable que es detectable por el sensor. Para evitar que la variable a medir se encuentre alternando entre valores que el sensor no es capaz de detectar, es necesario conocer previo a la implementación, la sensibilidad del sensor con el acondicionamiento implementado. En aplicaciones digitales es común utilizar sensores que contienen todo el acondicionamiento en empaquetados pequeños, lo cual resulta práctico a la hora de implementar pero otorga sensibilidad fija que va relacionada con los bits de resolución del sensor.

2.1.3 Acondicionamiento de señal

Dependiendo de la naturaleza de las variables a medir, del o los sensores utilizados y del sistema se pueden presentar situaciones en las que sea necesario realizar ajustes ya sea en las señales de salida del sistema, las señales a medir o las señales que los sensores entregan. Estos ajustes a las señales se realizan para volver más accesible un resultado o entregarlo a otro dispositivo, siguiendo bloques de funcionamiento como el mostrado en la Figura 2.2.

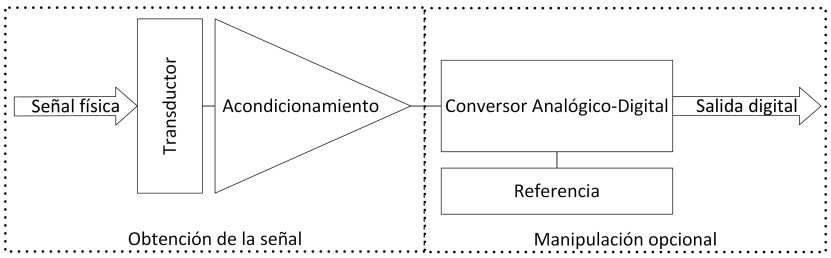


Figura 2.2: Diagrama a bloques del acondicionamiento de señal.

Los escenarios más comunes en que se pueden encontrar acondicionadores de señal son al utilizar sensores analógicos de tipo resistivos o de reactancia variable, a los cuales se les acondiciona la salida de manera que esta refleje linealmente la variable a medir con relación a su resistencia, estos circuitos se realizan comúnmente con amplificadores operacionales en diferentes arreglos y modos de operación que permiten tomar la señal sin alterarla gracias a su alta impedancia de entrada. En la Figura 2.3 se muestra un circuito amplificador inversor y su ecuación característica, al ser un amplificador, el circuito permite controlar el alcance de la señal de salida.

El ajuste de alcance anteriormente mencionado y el ajuste de cero permiten la calibración de la señal entregada por un sistema. El cero va en relación a la cantidad que se desea establecer como referencia o mínima de salida con base en una entrada, mientras que el alcance es la relación que hay entre la señal de entrada y la señal de salida, visto de otra forma, si se tratase de un sistema lineal, el ajuste de cero nos indica b y el alcance m dentro de la ecuación $y = mx + b$ de una linea recta.

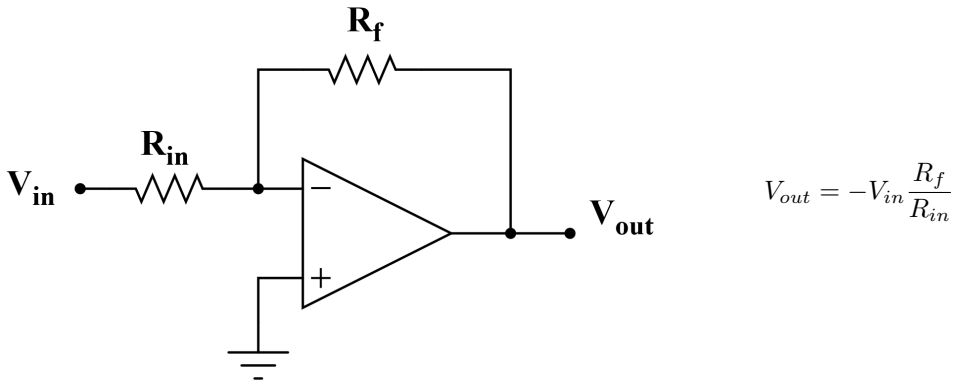


Figura 2.3: Amplificador inversor y su ecuación característica.

2.1.4 Protocolo de comunicación digital

Es un sistema de reglas que permite a dos o mas entidades intercambiar información por medio de la variación de alguna magnitud física. En sistemas digitales dependiendo las necesidades del sistema y sus características se pueden encontrar diferentes tipos de comunicación sobre las líneas de conexión; transmisión desde emisor-receptor (Simplex), transmisión emisor-receptor y viceversa (Half-Duplex) y transmisión emisor-receptor simultánea (Full-Duplex). También es posible agrupar los protocolos de comunicación en seriales/paralelas y síncronas/asíncronas [37].

- Paralela: generalmente utilizada en comunicación interna entre circuitos, representa la manera más rápida de transmisión de datos desde un emisor a un receptor debido a que se utiliza más de una sola línea de comunicación. En lugar de enviar bit a bit de manera serial, la comunicación paralela envía toda una trama con un mismo pulso de reloj, siendo una desventaja la utilización de más pines físicos para la conexión, por ejemplo un protocolo PCI utilizada para conectar componentes a las tarjetas madre de una PC tiene 32 o 64 pines.
- Serial: es la comunicación más utilizada en el ámbito digital, representa una solución óptima en la transmisión de datos. La trama de datos a enviar se agrupa en paquetes según requiera el protocolo para luego ser enviado uno tras otro de manera síncrona o asíncrona, con la ventaja de utilizar pocos pines ejemplo: UART e I^2C que utilizan 2 pines y SPI que utiliza 3 pines base y 1 más por cada esclavo.
- Síncrona: se transmiten tramas de datos fijas con sincronización entre el emisor y el receptor permitiendo control sobre el flujo de datos reduciendo errores en la transmisión gracias a la sincronía. Ejemplo: I^2C, entre sus 2 pines de comunicación se encuentra SCL que es la señal de reloj y SPI que utiliza SCLK como señal de reloj.
- Asíncrona: la transmisión de datos se realiza sin utilización de una señal externa de reloj, los datos pueden ser transmitidos según se requiera sin necesidad de un flujo constante, suelen caracterizarse por contar con bits de comienzo y parada. Ejemplo: UART que no utiliza pin de reloj, solamente cuenta con un pin para transmisión TX y otro para recepción RX.

Con los diferentes tipos de comunicación existente es necesario conocer las necesidades específicas del sistema antes de elegir un protocolo, para este caso, se necesita comunicación controlada y estable con múltiples dispositivos, por lo que tiene que ser síncrona y dado que hacer comunicación paralela implica más hardware se elige una comunicación serial. I^2C y SPI son ampliamente utilizados para

estas necesidades debido a que son de implementación sencilla y de bajo consumo energético. La mayor diferencia existe al comunicarse con múltiples esclavos, SPI requiere de 1 pin por cada esclavo presente en el bus mientras que I^2C hace el direccionamiento por software, esta es la razón por la cual se escoge I^2C [39].

Protocolo de comunicación I^2C

Originalmente creado por Philips en 1982 es uno de los protocolos de comunicación serial más utilizados para comunicación interna de un chip o externa chip a chip, con posibilidad de múltiples maestros utilizando 2 líneas de comunicación llamadas SDA y SCL. A diferencia de SPI no hay necesidad de un chip select por lo que en teoría pueden conectarse la cantidad de maestros y esclavos que se requiera. La comunicación utilizando este protocolo es definida por los siguientes aspectos clave:

- Direcciones de identificación de esclavos presentes en el bus de 7 bits, cada esclavo debe tener una dirección única.
- Tramas de datos enviadas divididas en 8 bits (1 byte).
- Bits de control de comunicación; comienzo, parada, dirección y reconocimiento (*acknowledgment*).

Las velocidades de transferencia más comunes para los buses de transmisión son $100\,\mathrm{kB\,s^{-1}}$, $400\,\mathrm{kB\,s^{-1}}$ y $3.4\,\mathrm{MB\,s^{-1}}$, llamadas modo estándar, modo rápido y modo de alta velocidad, respectivamente. Físicamente el bus de I^2C cuenta con dos líneas activas de drenador abierto y bidireccionales; SDA y SCL. Las líneas cuentan con resistencias pull-up normalmente de $4.7\,\mathrm{k\Omega}$, pudiendo ser menor o mayor el valor dependiendo la corriente que se necesite según la distancia, en la Figura 2.4 se muestra su esquema de conexión.

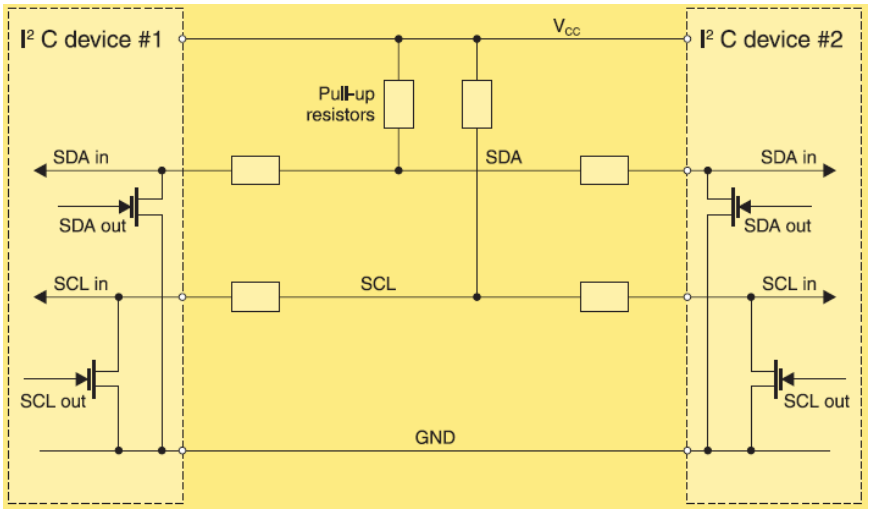
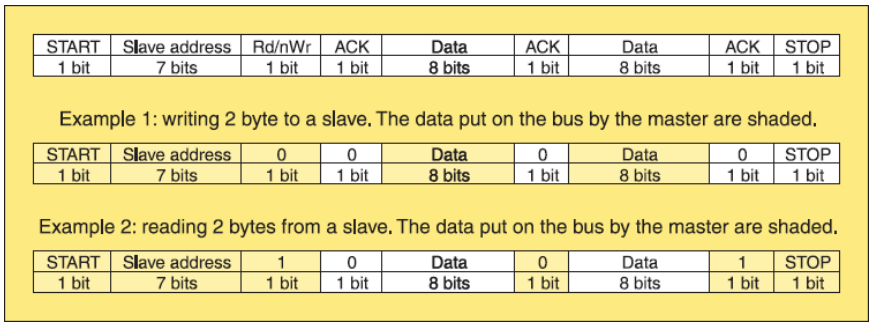


Figura 2.4: Esquema de conexión entre 2 dispositivos I^2C [40].

La comunicación entre dispositivos comienza con una señal de START que funciona como una llamada de atención para todos los dispositivos presentes en el bus, después, el maestro envía la dirección de 7 bits correspondiente al dispositivo con el que requiere comunicarse, se envía también una señal que indica si se pretende escribir o leer. Todos los dispositivos en el bus comparan la dirección recibida

con su dirección propia y si esta no coincide no realizan ninguna acción mientras que el dispositivo que si coincide responde con una señal conocida como ACKOWLEDGE. Con la comunicación establecida el maestro comienza a recibir o transmitir datos en tramas de 8 bits después de las cuales se necesita un bit de ACKOWLEDGE que confirma su recepción, al finalizar la transmisión de datos se envía por parte del maestro la condición de parada. La estructura de una señal transmitida sobre el bus de I^2C se muestra en la Figura 2.5 y las fases para las señales del bus se muestran en la Figura 2.6 [40].



START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 1: writing 2 byte to a slave. The data put on the bus by the master are shaded.

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 2: reading 2 bytes from a slave. The data put on the bus by the master are shaded.

START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Figura 2.5: Información transmitida por el bus I^2C [40].

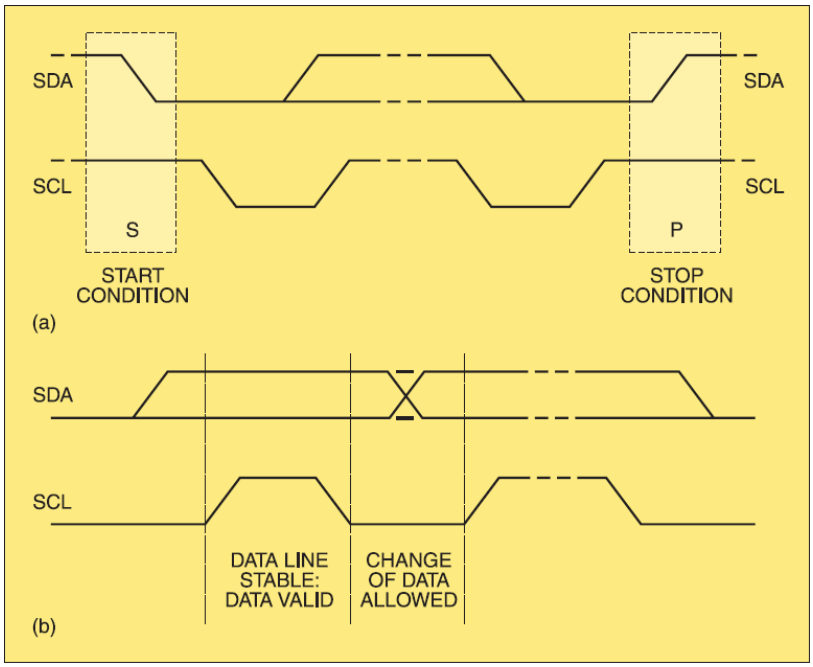


Figura 2.6: (a) las fases que indican las condiciones de START y STOP (b) las fases que indican que se puede leer el dato o que se puede cambiar el dato sobre el bus I^2C [40].

2.2 Internet de las cosas (IoT)

En 1999 el Instituto de Tecnología de Massachusetts fue el primero en introducir el concepto de IoT como un puente entre las interfaces de sensado digital interactuando con dispositivos inteligentes [41], desde entonces el concepto ha sido definido basado en el punto de vista de cada autor y la

aplicación, sin embargo, es posible extraer la esencia de cada definición. Considerando la Figura 2.7, en una ciudad donde una gran cantidad de objetos pueden sensar, comunicarse y compartir información interconectados en redes de Internet públicas o privadas, el comportamiento del medio se refleja en datos que dan lugar a infinidad de servicios o acciones a realizar en tiempo real. Por lo tanto IoT se vuelve una herramienta que convierte dispositivos antes inertes dentro del hogar o en la ciudad, en puntos de vista hacía la realidad de las cosas.

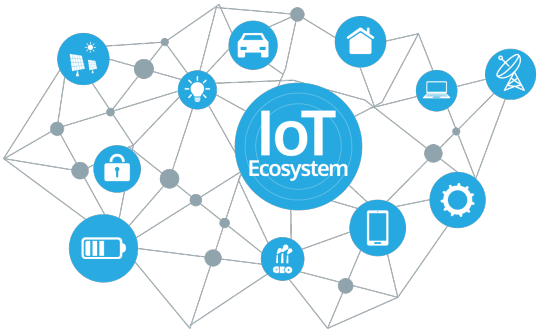


Figura 2.7: Esquema del ecosistema del Internet de las cosas.

En un principio, la información presente en Internet era ingresada manualmente por personas, ya sea escribiéndola directamente, grabando o tomando fotografías, todo centrado en la interacción del hombre con la maquina. Lo anterior mencionado tiene limitantes como el tiempo, precisión y atención de una persona, haciendo que los procesos de captura se vuelvan vulnerables ante errores humanos. IoT motiva la autonomía de los dispositivos, si la infraestructura de manejo de datos ya existe, es necesario alimentarla con información. Los dispositivos necesitan recolectar información propia o de su entorno y comunicarla sin ninguna interacción humana, permitiendo por ejemplo que, para casos industriales, se tenga conocimiento sobre el estado de la maquinaria con información capturada y entregada por esta misma.

Al igual que con redes de interconexión telefónica o de Internet, cuando se trata de sistemas IoT existen diferentes topologías que se pueden implementar con base en las necesidades que se busquen cubrir, como se muestra en la Figura 2.8, se pueden tener redes centralizadas en las cuales existan múltiples dispositivos que alimentan a un sólo concentrador, redes con concentradores en cadena que comparten conexiones y redes distribuidas donde todos comparten conexiones.

El elemento básico presente en sistemas IoT son los sensores, ya que estos reflejan la variable o el proceso monitorearado, seguidos por el medio a través del cual se va a transmitir de manera local la información para después encontrar el acceso a Internet mediante Routers. A partir de la transmisión de información a Internet, la implementación de los sistemas varía, ya sea que se utilicen servidores para guardado de información o que esta se distribuya de manera uniforme entre varios clientes. Al final, el objetivo de IoT y el punto común entre las diferentes topologías, es el análisis de la información recolectada y la entrega al usuario.

En la Figura 2.9 se muestra un ecosistema de aplicación IoT, que mediante monitoreo posibilita acciones de mantenimiento predictivo en maquinas de producción y chequeo de calidad previo a la salida al mercado del producto con base en las variables de funcionamiento medidas, una vez salido el producto al mercado, es posible seguir el producto con el monitoreo de variable externas.

El gran avance y la aceptación social de servicios de Internet brinda una oportunidad para el Internet de las cosas de ser implementado sobre una infraestructura presente globalmente. En su mayoría, salvo

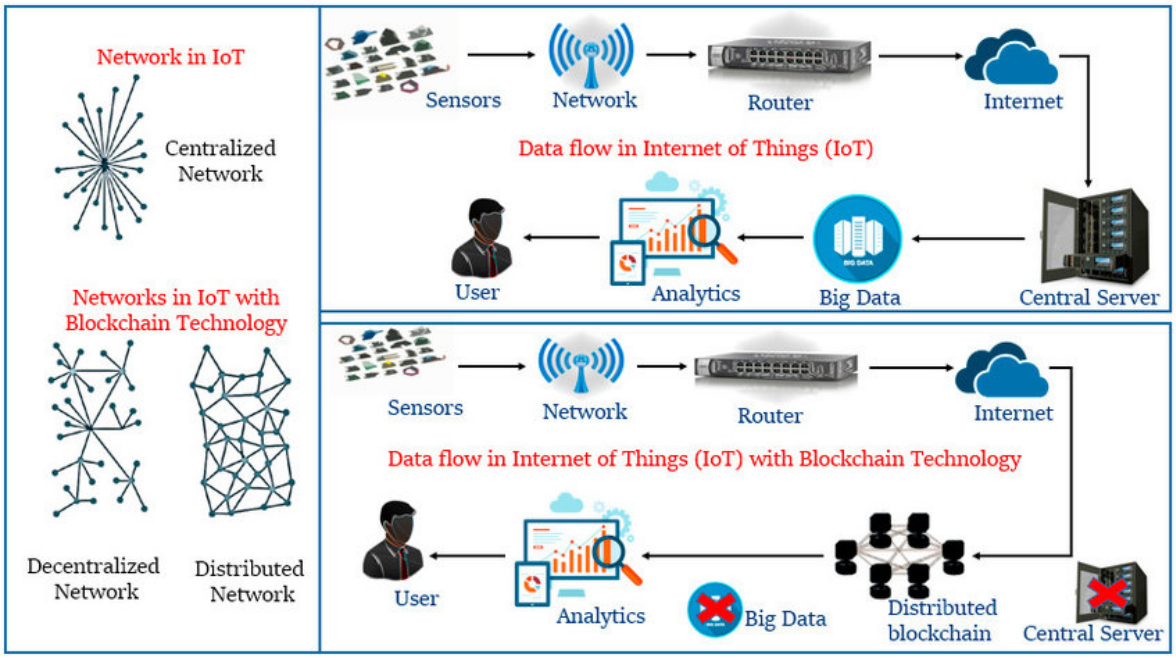


Figura 2.8: Topologías de conexión para sistemas IoT.

casos que se utilicen dentro de redes privadas para la comunicación a Internet, se utilizan protocolos de comunicación inalámbrica sobre dominios públicos, los protocolos de comunicación utilizados varían según la necesidad y posibilidades de la aplicación, en las siguientes secciones se muestran los más utilizados.

2.3 Protocolos de comunicación inalámbrica

La necesidad por automatización de procesos de la industria ha permitido a múltiples tecnologías desarrollarse más rápido, ejemplo de esto son las comunicaciones inalámbricas que brindan fácil interacción entre dispositivos y procesos dentro de un ambiente sin necesidad de contacto físico entre ellos, reduciendo cables e instrumentación. En este aspecto las comunicaciones que más predominan son Bluetooth, UWB(Ultra-Wideband), ZigBee y Wi-Fi que corresponden a los estándares 802.15.1, 802.15.3, 802.15.4 y 802.11a/b/g de la IEEE, respectivamente. Estos estandares definen las capas físicas para la comunicación inalámbrica `PHY` y `MAC` sobre un rango de acción de 10 a 100 m.

Bluetooth *IEEE 802.15.3*

Está basado en un sistema de radio de corto alcance diseñado para reemplazar cables en dispositivos sencillos como periféricos de una computadora, impresoras, entre otros. Trabaja normalmente en el rango de aplicaciones conocida como `WPAN` (Wireless Personal Area Network). Se definen dos topologías de conexión: Piconet en la cual todos las conexiones son controladas por un maestro o Scatternet que es una red dispersa sobre la cual pueden existir dos o más redes Piconet.

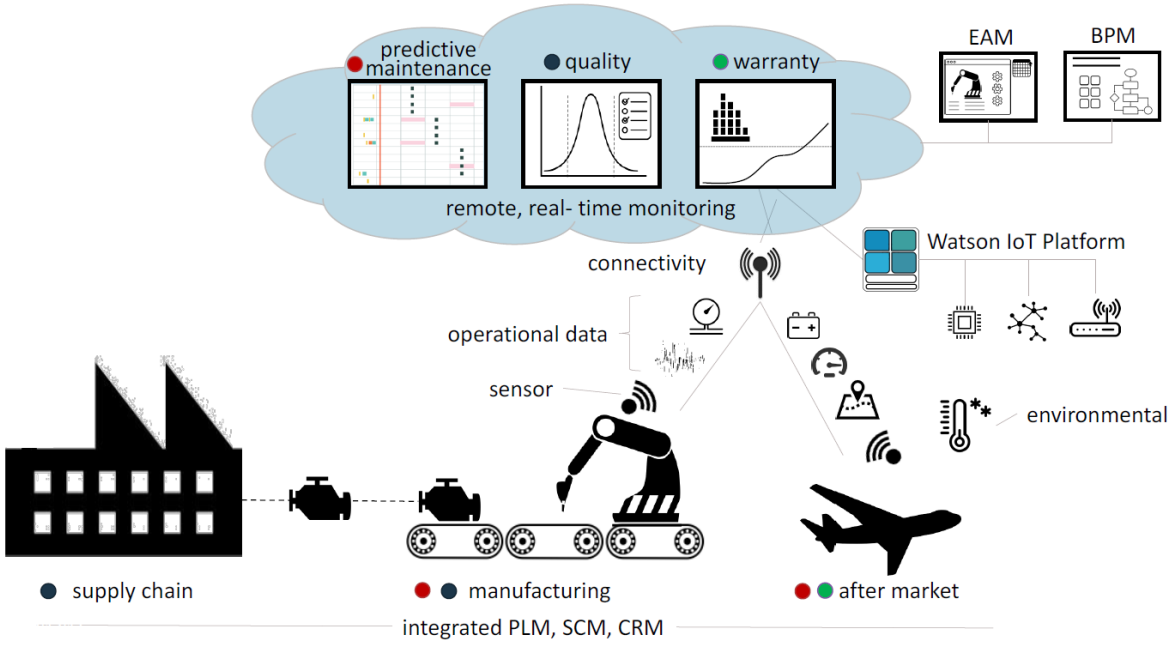


Figura 2.9: Aplicación de IoT en la industria.

UWB *IEEE 802.15.3*

Comunicación inalámbrica de corto alcance y velocidades relativamente altas de 110 Mbit s^{-1} hasta 480 Mbit s^{-1}. Las velocidades satisfacen la mayoría de las necesidades de transmisión de datos dentro del hogar como audio y vídeo, permitiéndose funcionar como remplazo de comunicaciones seriales como USB 2.0.

ZigBee *IEEE 802.15.4*

Al igual que Bluetooth, funciona en el rango de aplicaciones `WPAN` con dispositivos de consumo energético mínimo dentro de un rango de 10 m. La implementación de este protocolo otorga redes auto-organizables, de múltiples etapas y conexión confiable con larga duración de batería debido a su baja tasa de transmisión.

Wi-Fi *IEEE 802.11a/b/g*

Fidelidad inalámbrica (Wireles Fidelity) incluye estándares para redes inalámbricas de área local que permiten a los usuarios navegar por Internet con velocidades del ancho de banda (54 MB s^{-1}) si existe una conexión a Internet.

Para la elección del protocolo de comunicación inalámbrica a utilizar se hizo una comparación de los mencionados anteriormente apoyado por diversas fuentes, en estas fuentes se realizan estudios mostrando las ventajas y desventajas de cada uno, para qué se utilizan y dónde es más común encontrarles. Del estudio comparativo encontrado en [42] se extrae la Figura 2.10 que muestra el consumo energético

en diferentes dispositivos que cuentan con los protocolos de comunicación embebidos, se puede observar que los protocolos Wi-Fi y UWB son los que mayor consumo energético y mayor velocidad de transferencia tienen. Aunque el consumo energético de los protocolos varía según el microcontrolador utilizado, el artículo presenta una buena primera aproximación. Del mismo artículo la Figura 2.11 muestra la relación que existe entre el consumo de energía y la información transmitida, esto refleja la ventaja que conlleva transmitir información sobre Wi-Fi o UWB con respecto a los otros 2. Por lo tanto, teniendo en cuenta que el sistema de Wi-Fi utilizado para las pruebas del artículo tiene alto consumo de corriente y no se hace mención alguna sobre manejo de modos de bajo consumo; la potencia consumida por el sistema propuesto se verá reducida. Debido a que la implementación de cualquier protocolo de comunicación que no sea común llevaría a problemas de compatibilidad, se elige Wi-Fi para hacer más portable y compatible la comunicación, brindar consumo energético moderado y alta velocidad de transferencia de información.

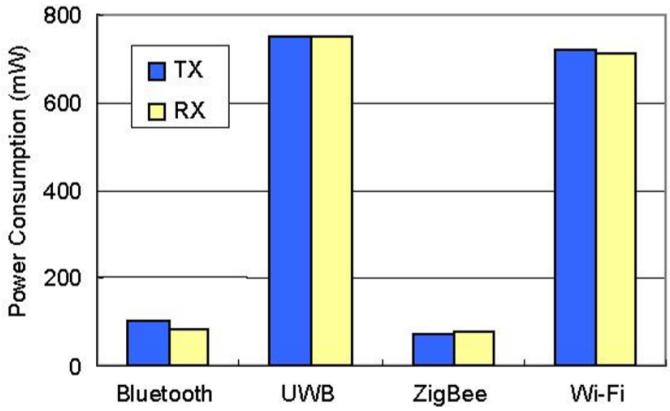


Figura 2.10: Comparación de la potencia consumida de cada protocolo [42].

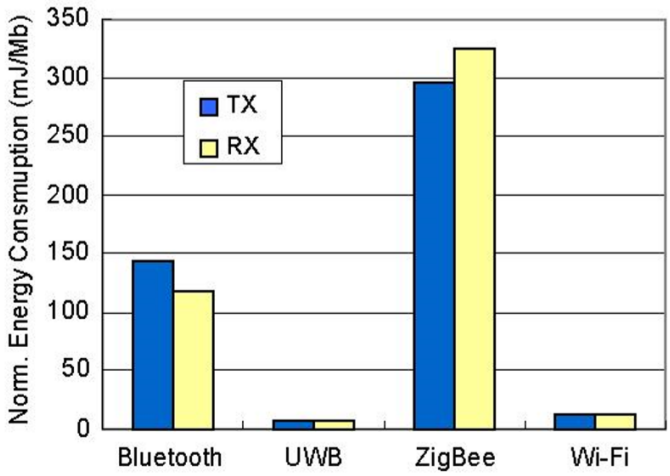


Figura 2.11: Comparación de la potencia consumida por Mb transmitido de cada protocolo [42].

2.3.1 Transmisión de datos sobre Wi-Fi

Los protocolos de comunicación mencionados anteriormente funcionan como el puente sobre el cual se transmiten los datos, esta etapa suele agruparse con el nombre de capa del medio, ya que incluye los instrumentos más primitivos esenciales para envío/recepción de bits. A partir de este punto, la manera en que los datos son empaquetados y tratados sobre la linea de transmisión varía dependiendo el protocolo de transmisión utilizado, siguiendo el mismo concepto de agrupación, con base en el modelo osi [35] esta etapa es conocida como la etapa de transporte, y dado que se elige utilizar Wi-Fi, la explicación se sentrará en los protocolos más utilizados sobre esta, que son TCP y UDP como capas de red y sobre las cuales se implementa la capa de aplicación HTTP.

TCP

El protocolo de control de transmisión (Transmission Control Protocol, TCP por sus siglas en inglés) es uno de los protocolos fundamentales en Internet, creado en 1974. El protocolo garantiza comunicación segura entre los puntos, vigilando la entrega de los datos sin errores y en el orden que estos fueron transmitidos, gracias a la implementación de checksum, cabeceras, reconocimiento de recepción y flujos de transmisión controlados.

Para poder llevar a cabo el intercambio de información entre cliente y servidor, se debe establecer una conexión que va asociada a un puerto y una dirección IP, esto se hace mediante un proceso de sincronización entre ambas partes, a esta conexión normalmente se le llama socket. TCP es un protocolo de comunicación ampliamente usado sobre el cual ya existen puertos designados para funciones especificas, como es el caso del puerto 80 utilizado para transmitir HTTP.

UDP

El protocolo de datagramas de usuario (User Datagram Protocol, UDP por sus siglas en inglés) permite el envío de datagramas o paquetes de datos sin que se haya establecido previamente una conexión ya que los datos de transmisión se encuentran en el mismo paquete. A diferencia del protocolo TCP, este no cuenta con control de flujo de datos ni el reconocimiento de su recepción por lo que la transmisión no es confiable pero si más rápida. Suele utilizarse en aplicaciones donde se requiera respuesta rápida sin sincronización entre el origen y el destino, es en cierta forma análoga a UART en las comunicaciones seriales.

Retomando lo anterior, ambos protocolos utilizan direcciones IP y puertos para direccionar sus mensajes; véase Figura 2.12. La principal diferencia entre TCP y UDP radica en la interacción presente entre transmisor/receptor, en TCP previo al intercambio de información es necesario establecer una conexión entre los puntos para asegurar la transmisión de datos, con cada trama de datos enviada, se tiene control de transmisión y verificación de recepción mientras que en UDP no se tiene el mismo enfoque a conexiones si no a transmisión de datos, por lo que se intenta enviarlos tan pronto estén disponibles sin verificar el proceso.

HTTP

El protocolo de transferencia de hipertexto (Hypertext Transfer Protocol, HTTP por sus siglas en inglés) es el protocolo que da lugar a los servicios WEB. Este sigue un esquema de petición-respuesta

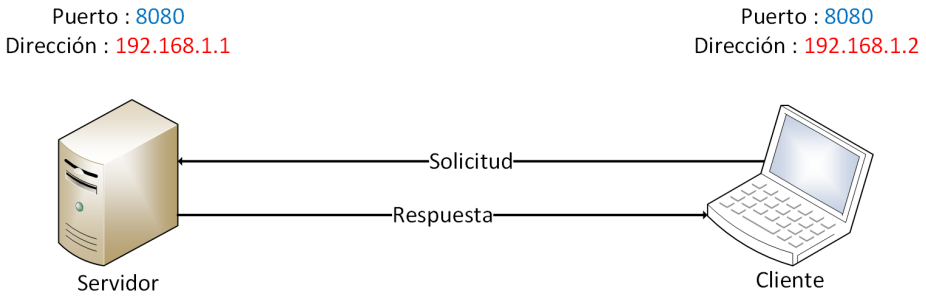


Figura 2.12: Intercambio de información entre cliente-servidor por TCP/UDP.

entre un cliente y un servidor, por ejemplo; la interacción entre los navegadores web y los servidores web que se lleva a cabo mediante mensajes de texto plano que contienen los métodos de peticiones o respuestas, las cabeceras y el cuerpo del mensaje.

Dentro del protocolo HTTP, se definen diferentes métodos de petición que le agregan funcionalidad y cambian según las versiones utilizadas, para este trabajo, los métodos de petición utilizados sobre los cuales es necesario centrarse serán los siguientes:

- GET Este método solicita representación de recurso, estas solicitudes son comúnmente utilizadas para recuperar datos sin provocar ningún otro efecto.
- POST Envía los datos que se le indique en el cuerpo de la petición, esto es comúnmente utilizado para actualizar recursos ya existentes o crear nuevos.

Al terminar una interacción con el mensaje se espera un código de respuesta con un significado especifico para cada método implementado, para este caso, al implementar métodos como GET y POST se esperan respuestas con formato “2XX” que indican que la petición ha sido procesada correctamente.

2.3.2 Lenguajes de programación WEB

HTTP ofrece las instrucciones sobre las cuales se da interacción entre cliente y servidor, pero hace falta la interfaz que facilite la interpretación del usuario sin necesidad de interacción directa con las instrucciones, aquí es donde lenguajes como HTML y JavaScript ofrecen solución sencilla y practica, la explicación se centrará sólo en esos 2 lenguajes ya que son los que fueron utilizados. La característica principal de estos lenguajes, es que a diferencia de otros, estos son ejecutados por el cliente, el servidor sólo provee las instrucciones o el “código” escrito correspondiente a cierta petición o dirección sin ejecutarlo ni compilarlo.

HTML

Desde los inicios del Internet, las demandas básicas de interacción con un usuario se solucionan con el lenguaje estático de marcas Hipertextuales (HyperText Markup Language, HTML por sus siglas en inglés), que como su nombre lo indica, es texto plano con etiquetas, lo que le otorga estructura y fácil interpretación gracias a sus etiquetas descriptivas; véase Figura 2.13. Los archivos de descripción de paginas o interfaces HTML suelen ser relativamente pequeños lo que los vuelve de despliegue rápido y

Sección de código 2.1: Código ejemplo de JavaScript

```
<html>
<head>
 <title>Hola Mundo</title>
</head>
<body>
 <script language="JavaScript">
 <!--
  document.write("<Hola Mundo en JavaScript!");
 -->
 </script>
</body>
</html>
```

son fácilmente legibles por todos los navegadores WEB con la desventaja de que cada navegador tiene variaciones en su interpretación.

Figura 2.13: Estructura básica de las etiquetas HTML.

JavaScript

Con estructura similar al lenguaje de programación orientado a objetos JAVA, este lenguaje no es orientado a objetos ni tampoco compilable, es un lenguaje interpretado que funciona mediante scripts que mejoran la interfaz del usuario permitiendo implementar páginas web dinámicas, lo que da solución a la naturaleza estática del texto plano presente en HTML. Dentro del código HTML de una página es común encontrar secciones con scripts que realizan una función especifica que normalmente no podría hacer HTML, en la Sección de código 2.1 se muestra un script de JavaScript.

Capítulo 3

Desarrollo del sistema de recolección de datos

3.1 Elaboración del hardware

Para obtener el funcionamiento remoto y la medición en sitio que se requiere, es necesario que el hardwre del sistema de recolección de variables propuesto sea compacto y de bajo consumo. Su hardware se compone por sensores digitales que no necesitan circuitos de acondicionamiento de señal y son de bajo consumo energético. Se utilizan en total 3 sensores que funcionan como esclavos sobre un bus I^2C con un microcontrolador con capacidad Wi-Fi que actúa como maestro.

Al comenzar el desarrollo del proyecto, con el hardware ya elegido se realizó una tarjeta con todos los sensores integrados, en la Figura A.1 en el Apéndice A se muestra el circuito impreso (PCB, por sus siglas en inglés) de la placa. La tarjeta se realiza para ser compatible con la versión integrada del CC3200; un SensorTag de diseño compacto con alimentación por batería que se acopla a la placa de sensores mediante un conector. La idea fue rechazada debido a que el datasheet del sensor INA260 advierte que el dispositivo se calienta a causa de la resistencia shunt integrada, lo que lleva a mediciones erróneas de temperatura ya que el sensor también está presente en la placa. El SensorTag perdió utilidad conforme el proyecto fue avanzando, a pesar de que su diseño compacto representa la solución buscada en tamaño no ofrece libertades a la hora de programar.

Para asegurar el rendimiento óptimo de los dispositivos se consultan sus respectivas hojas de datos que proporcionan las consideraciones necesarias y sus circuitos recomendados. En la Figura 3.1 se muestra el esquema del sistema físico implementado.

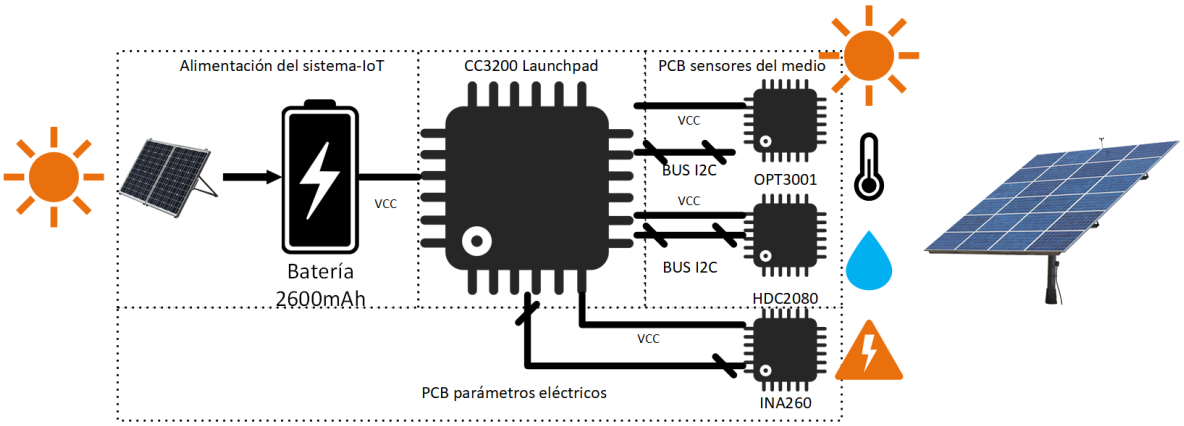


Figura 3.1: Esquema del sistema físico.

Microcontrolador CC3200

Es un microcontrolador de Texas Instruments con nucleo ARM Cortex de 80 MHz, certificado Wi-Fi y subsistema de control energético [43], son las características por las cuales se elige para el proyecto, empaquetado QFN 9×9 mm de 64 pines. Se utiliza el Launchpad con el microcontrolador integrado como entorno de desarrollo debido a que ofrece acceso rápido a los GPIO disponibles mediante pines y es fácilmente programable, en la Figura 3.2 se muestra la tarjeta de desarrollo.

Sensor HDC2080

Sensor digital de temperatura y humedad se elige por su precisión, con errores máximos de $\pm 0.2\,°C$ y $\pm 2\,\%RH$, su empaque compacto WSON 3×3 mm de 6 pines y su capacidad de comunicación I^2C [44].

Sensor OPT3001

Sensor digital de luz de alta precisión $\pm 0{,}2\,\%$ diseño de montaje superficial USON 2×2 mm de 6 pines y capacidad de comunicación I^2C [45].

Sensor INA260

Sensor digital de corriente, voltaje y potencia, cuenta con resistencia shunt integrada de $2\,m\Omega$, errores de lectura máximos del $0.15\,\%$ y un offset máximo de 5 mA, empaquetado compacto TSSOP 5×4.4 mm con 16 pines y capacidad de comunicación I^2C [46].

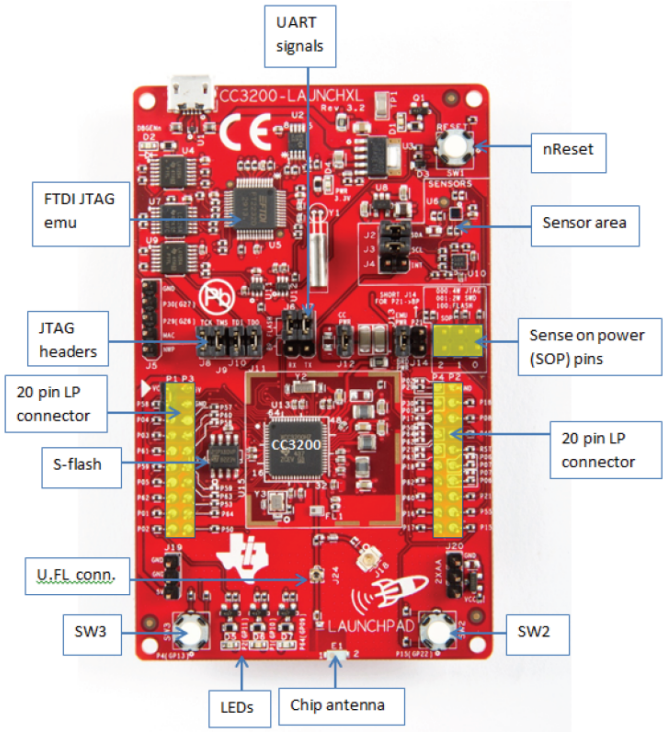


Figura 3.2: CC3200 Launchpad.

3.1.1 Esquemático y PCB

Los sensores utilizan comunicación I^2C, es necesario cuidar el ancho de las pistas del bus y la estabilidad de la comunicación. Se utilizan resistencias pull-up de 1 kΩ permitiendo que el bus de de comunicación tenga menos susceptibilidad a errores de nivel, interferencia o distancia a costa de un aumento en consumo de corriente. Con base en los requerimientos de los sensores y la naturaleza de las variables a medir, el sistema se divide en dos placas.

En la primera placa se incluye el sensor INA260 para la medición de los parámetros eléctricos del panel, se añade una batería de 2600 mA h con capacidad suficiente para alimentar el sistema durante varios días y un circuito de carga con un panel solar pequeño suficiente para cargar la batería durante la exposición al sol. En la Figura 3.3 se muestra el esquemático y en la Figura 3.4 el PCB.

La segunda placa es diseñada para la medición de parámetros del medio: humedad, temperatura y luz, con los sensores HDC2080 y OPT3001. Los sensores requieren estar expuestos al medio para leer correctamente las variables por lo que es necesario adaptar las conexiones y el sistema en general haciéndolo resistente a las condiciones climatologías, en la Figura 3.5 se muestra el esquemático y en la Figura 3.6 el PCB. Se cubre completamente el PCB con una capa de pintura antisoldante y se utiliza resina como protección en las conexiones de las terminales.

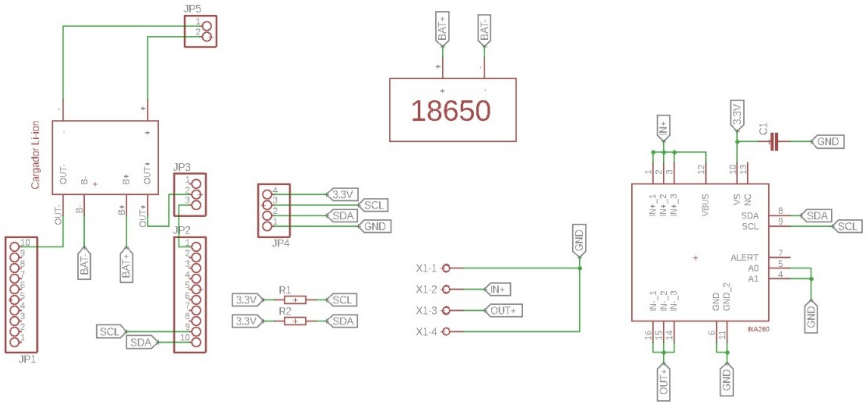


Figura 3.3: Esquemático placa de sensado eléctrico.

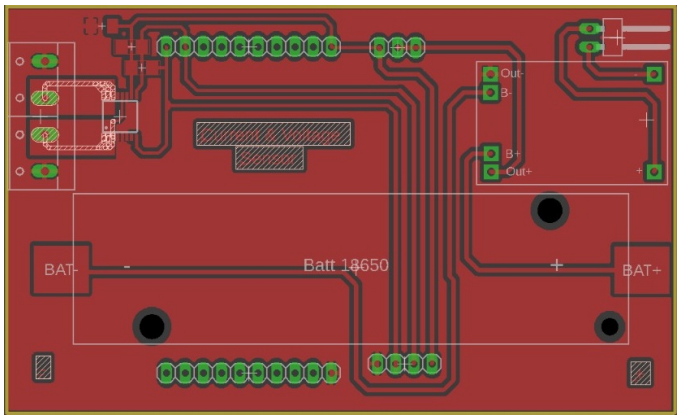


Figura 3.4: PCB placa de sensado eléctrico.

3.1.2 Aditamentos

Además de los diseños físicos del sistema electrónico se elabora, con ayuda de una impresora 3D, una base sobre la cual las placas se sujetan para colocarse sobre el panel solar como se muestra en la Figura 3.7.

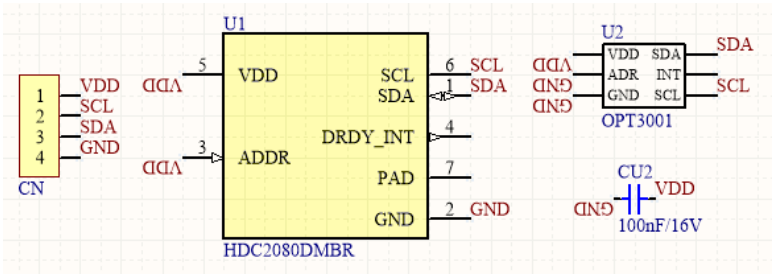


Figura 3.5: Esquemático placa de sensado del medio.

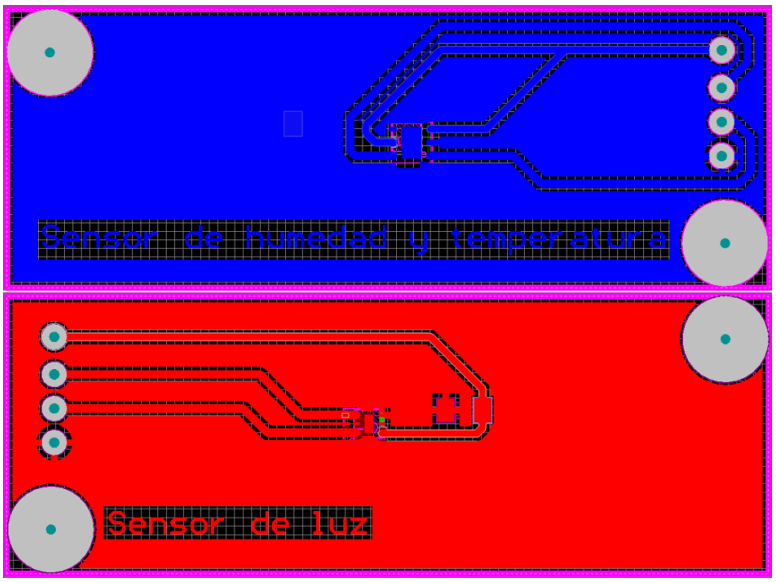


Figura 3.6: PCB placa de sensado del medio.

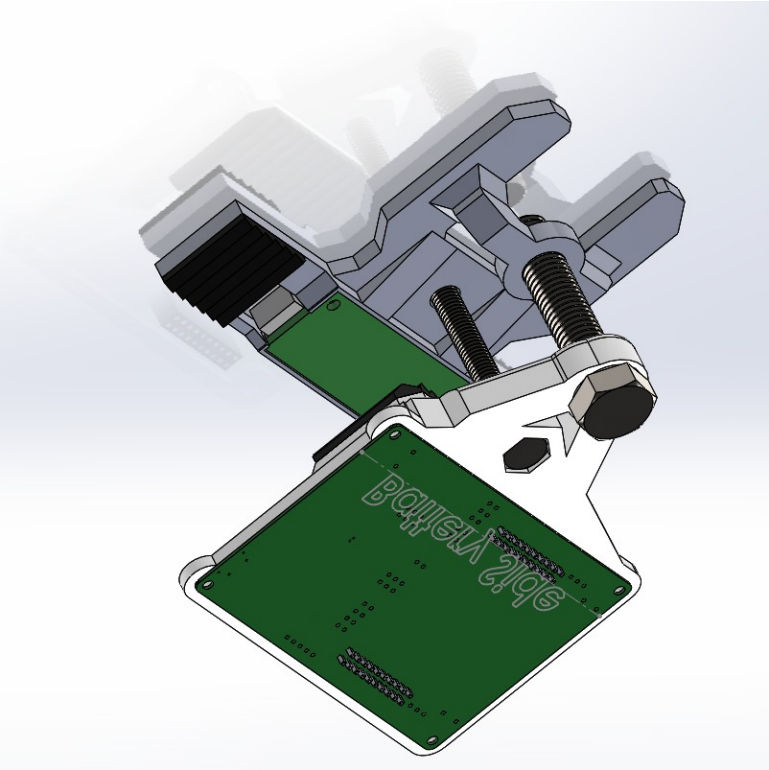


Figura 3.7: Base para montar el sistema.

3.1.3 Consideraciones de alimentación del sistema

El diseño de la alimentación del sistema tuvo varios cambios, en un principio se alimentó directamente desde la entrada USB de la tarjeta y de ahí, gracias a los conectores, pasaba a todos los componentes pero debido a que no se ocupa ninguno de los componentes para debuggear se aisló esa parte y se alimentó directamente desde la batería hasta los pines de alimentación de la placa. Esto condujo a la siguiente problemática, el voltaje entregado por la batería depende de su nivel de carga, en estado máximo entrega 4 V haciendo que los dispositivos estén cercanos o fuera de su rango de operación recomendado, por lo que en lugar de alimentar directamente se utiliza el convertidor lineal de voltaje integrado en el Launchpad para mantener su nivel en los rangos correctos.

3.2 Elaboración de software

Los requerimientos del sistema plantean diferentes situaciones y eventos; es necesario realizar mediciones periódicamente, comunicarse con los sensores vía I^2C, realizar y mantener conexiones Wi-Fi, atender peticiones HTML, estructurar y acumular las mediciones y finalmente la creación de archivos contenedores de mediciones diarias. Hay eventos síncronos dentro de los requerimientos del sistema que pueden ser controlados fácilmente dentro de una máquina de estados pero las peticiones HTML y las interrupciones por eventos Wi-Fi no son predecibles ni controlables por lo que una máquina de estados fija no ofrece una solución viable, es necesario implementar un manejo inteligente sobre la ejecución de tareas según la importancia de cada una, en vista de esta necesidad se implementa un sistema operativo de tiempo real (RTOS, por sus siglas en inglés). De esta forma se tienen las tareas funcionando independientemente con un núcleo que las controla como se ve en la Figura 3.8.

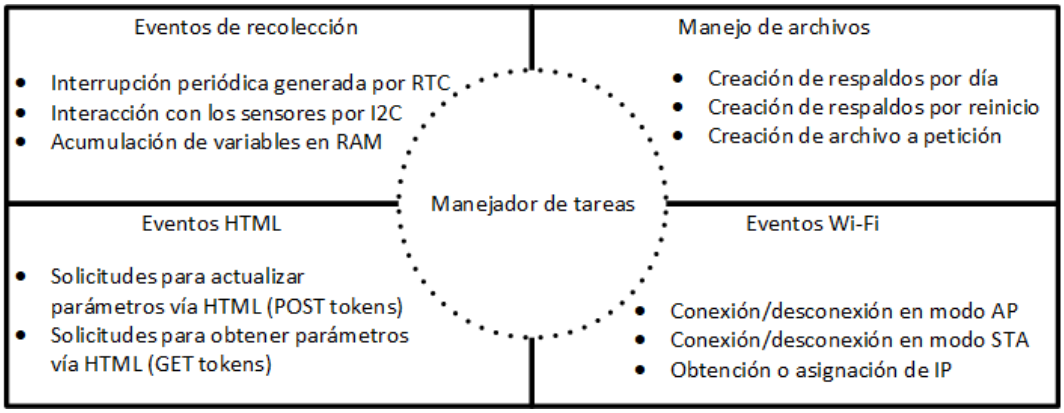


Figura 3.8: Tareas del sistema de recolección.

3.2.1 Real Time Operating System RTOS

La implementación de sistemas operativos en tiempo real permite a sistemas, en su mayoría de bajo nivel y bajo consumo, tener mejor soporte en manejo de recursos, sincronización, manejo preciso de tiempos, comunicación y mejor control sobre la interacción con los puertos de entrada/salida [47]. FreeRTOS es un sistema operativo en tiempo real completamente libre y ampliamente soportado por muchos dispositivos; por su sencillez y portabilidad se elige para implementarse en el sistema. Como

se ve en la Figura 3.9, el control de múltiples tareas se realiza seccionando el tiempo de ejecución para cada una con base en su actividad pendiente y a la prioridad que se le otorgue. Este control sobre los tiempos de ejecución aparenta la interacción "simultanea" de procesos, cuando en realidad existe un manejador de tareas (Scheduler) que se encarga del control de tiempos [48].

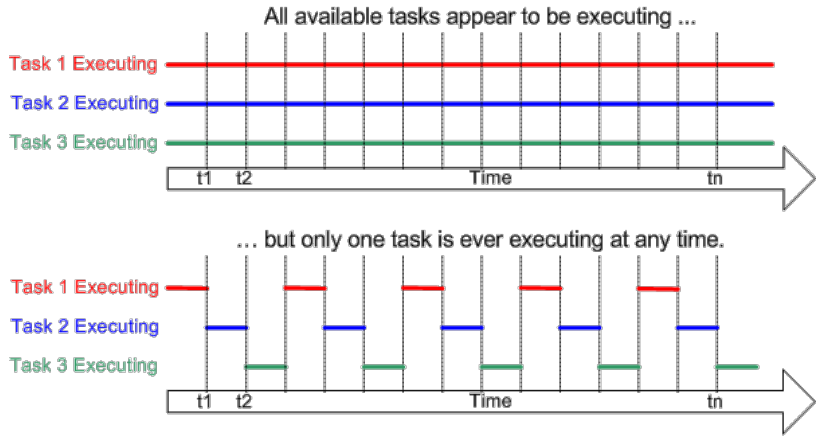


Figura 3.9: Manejo de tareas (*Scheduling*) del FreeRTOS.

3.2.2 Sistema de recolección de variables

El código del sistema de recolección está escrito en lenguaje C utilizando las API's, librerías y códigos ejemplo disponibles para el microcontrolador como base y apoyo para el desarrollo de aplicaciones que forman parte del kit de desarrollo de software (SDK, por sus siglas en inglés) versión 1.3.0. Se generan tareas para cubrir todos los eventos y necesidades que surjan, estableciendo un sistema de control de ejecución de tareas mediante mensajes internos controlados por el sistema operativo funcionando como sincronización o semáforo para las tareas. El diagrama de flujo de la Figura 3.10 muestra el proceso que sigue internamente el sistema mientras funciona. Los segmentos de funcionamiento, como se ve en la Figura 3.8, se explican en las secciones siguientes.

Eventos Wi-Fi

En esta sección se involucran todos los eventos relacionados con el protocolo Wi-Fi, estos eventos se controlan mediante 2 tareas de máxima prioridad que los agrupan de la siguiente manera:

- Tarea para eventos de conexión:

  Dependiendo del estado de funcionamiento del sistema son los eventos que suceden, si se encuentra en modo punto de acceso se pueden tener llamadas de atención cuando se establece o pierde conexión con el punto de acceso y si el sistema se encuentra en modo estación se tienen los eventos de la conexión y desconexión por parte del cliente. En la Sección de código B.10 en el Apéndice B se muestra la tarea que al ser llamada clasifica la fuente de la interrupción para actualizar las variables y el sistema en base al evento.

- Tarea para eventos de IP:

Los eventos IP que se esperan son 2; si se tiene el sistema en modo punto de acceso la función es llamada cuando se otorga IP al cliente, mientras que si se encuentra en modo estación sucede cuando se obtiene dirección IP por parte del punto de acceso. En la Sección de código B.11 en el Apéndice B se muestra la tarea que se encarga de atender estas interrupciones.

Al encenderse el sistema, dado que se busca interactuar con el usuario para recibir sus parámetros de conexión y tiempos de recolección, se sigue el proceso del código mostrado en la Sección de código B.4 en el Apéndice B en la cual se inicializan las variables necesarias y leen respaldos existentes. Se entra en un proceso cíclico que cambia el estado del Wi-Fi a modo punto de acceso con nombre "Sensor-Fi" y contraseña "sensores" utilizando la función mostrada en Sección de código B.5 en el Apéndice B, el proceso continúa al recibir una conexión, lo que activa el servidor WEB embebido habilitando la interacción HTML en espera del mensaje que indica la recepción de los datos. Cuando se reciben los parámetros de conexión, estos se utilizan para hacer el cambio a modo estación e intentar establecer conexión, este proceso es cuidado por el Watchdog para evitar que el sistema Wi-Fi se quede esperando IP. Si la conexión resulta exitosa se establecen las políticas de bajo consumo mientras que si no se logra conectar se activa un mensaje que permite a la función cíclica repetirse.

Eventos HTML

El control de todos los eventos de esta sección permite interactuar con el usuario mediante una interfaz HTML, la interacción se lleva a cabo por medio de símbolos definidos (tokens). Cuando se establece conexión con un cliente o se conecta a un punto de acceso se activan las rutinas que activan el servidor WEB embebido haciendo que al ingresar a la dirección IP predeterminada del sistema 192.168.1.1, o la que el punto de acceso le otorgue, se responda con una página WEB escrita en HTML y JavaScript.

La estructura de la página que se envía al cliente lleva secciones escritas que corresponden a los tokens que definidos dentro de un código en C, cada que se hace petición se dispara una interrupción que llama a la tarea HTML, esta tarea se encarga de recibir la interrupción y responder con base en los parametros de llamado. Existen 2 interrupciones, solicitudes de POST tokens y GET tokens.

- GET token

  Son de la forma "__SL_G_S" si son propios del sistema o "__SL_G_U" si son definidos por usuario, seguidos por un identificador. La función en C recibe la estructura del token que se solicita, para después otorgar una respuesta que reemplace el texto. Por ejemplo "__SL_G_UC", es un token definido por el usuario utilizado para mostrar en la página la cantidad de mediciones mientras que "__SL_G_S.A", es un token definido del sistema que se utiliza para obtener el tiempo que lleva en modo estación o punto de acceso sin generar interrupción.

- POST token

  Al igual que la estructura de los GET, son de la forma "__SL_P_S" o "__SL_P_U" dependiendo si es por parte del usuario o del sistema. Cada que se hace un POST se llama a la función en C que se encarga de segmentar la información si se trata de un token definido por el usuario mientras que si es uno propio del sistema el manejo es automático. La función es llamada con un puntero al mensaje en el cual se encuentra la estructura del POST y el resto de información que contiene. Los parámetros de conexión Wi-Fi y el tiempo de recolección se envían sobre una misma instrucción de POST con la estructura "__SL_P_ULD=DAT" seguida por los datos a enviar. La actualización de la hora interna del microcontrolador se hace con un POST definido por el sistema con estructura "__SL_P_S.J=" seguida por una cadena correspondiente a la fecha.

Eventos de recolección

Esta sección del comportamiento del sistema involucra la recepción del periodo de recolección y fecha, interrupciones periódicas de un reloj de tiempo real (RTC, por sus siglas en inglés), interacción con los sensores para nuevas mediciones y estructuramiento de datos.

Para que estos eventos de recolección comiencen es necesario que primero el sistema de eventos HTML dé lugar a la recepción de los parámetros de hora y periodo de recolección que el usuario introduce. El periodo de recolección se utiliza dentro de la configuración del reloj de tiempo real de 32 kHz propio del microcontrolador para generar interrupciones periódicas que saquen al sistema del modo de bajo consumo y funcionen como indicador para dar inicio al proceso de medición, la función que configura el timer se puede ver en la **??** en el Apéndice B. La función de atención a la interrupción del timer contiene una función de escritura sobre un mensaje llamado g_tRecolectar que es esperado dentro de la función mostrada en la Sección de código B.12 en el Apéndice B. Al desbloquearse el mensaje la función que lo lee termina y permite seguir la ejecución de la tarea.

Lo siguiente es la rutina de recolección de los sensores, el proceso se muestra en la Figura 3.11 y su código en la Sección de código B.13 en el Apéndice B. En primer lugar se registra el comienzo de la medición en la variable de tareas con su respectivo bit definido, después se inicializan las funciones y relojes necesarios para el protocolo I^2C, se repite el proceso de inicialización con cada medición debido a que al entrar en modo de bajo consumo este subsistema se desactiva. La función de inicialización del bus lo configura como maestro y en modo estándar con velocidad máxima de $100\,\mathrm{kB\,s^{-1}}$.

La comunicación a los sensores se realiza mediante fragmentación de las tramas de datos, al llamar a la función de escritura o lectura se indica la dirección correspondiente al sensor, después la dirección de memoria del arreglo que contiene la información a enviar o recibir y finalmente el número de bytes que se envían o esperan. Para el caso de escritura, el arreglo que se envía tiene que contener en su primera posición la dirección a la que se quiere realizar escritura, seguida por los datos que se quieran mandar. El orden en que se hace la lectura de sensores es el siguiente: OPT3001 con dirección 0x44, HDC2080 con dirección 0x41 e INA260 con dirección 0x40.

La hoja de datos del sensor de luz OPT3001 indica que al escribir 0xCA10 a la dirección de memoria 0x01 correspondiente a sus bits de configuración se dispara una medición y se configura un tiempo de conversión de 800 ms. El valor hexadecimal mencionado anteriormente se escribe cada que se quiere hacer una medición, se escribe después un byte para indicar un cambio en el puntero interno del sensor a la dirección 0x00 y después del tiempo de recolección se hace una petición de lectura esperando 2 bytes como respuesta. Si no se presenta ningún error de lectura se procede a interpretar el mensaje recibido, la respuesta se compone de 4 bits superiores correspondientes al exponente y 12 bits correspondientes al valor fraccional. Se utiliza una función Switch para evaluar el exponente y hacerle las operaciones correspondientes al valor fraccional siguiendo la ecuación 3.1, una vez terminadas las operaciones se asigna en la posición 0 del arreglo flotante de resultados el valor en luxes o 0 en caso de error.

$$lux = 0{,}01(2^{E[3:0]})R[11:0] \tag{3.1}$$

El sensor de humedad y temperatura HDC2080, a diferencia del sensor de luz, tiene como inicio predeterminado el funcionamiento que se requiere; 14 bits de resolución, mediciones activadas manualmente y de ambas variables con tiempos de conversión de 660 µs. El proceso de obtención de sus mediciones comienza escribiendo 0x1 en la dirección de memoria 0xF para indicar que comience una conversión, 20 ms después se escribe 0x0 para cambiar la dirección de su apuntador y hacer una petición de lectura esperando 4 bytes. El mensaje resultante contiene 32 bits de los cuales los 16 más altos corresponden a humedad y los más bajos a temperatura, se utilizan las ecuaciones 3.2 y 3.3 para

obtener los valores correspondientes de humedad y temperatura, respectivamente; si las operaciones de escritura/lectura resultan correctas se asignan a las posiciones 1 y 2 del arreglo flotante, en caso contrario se asigna 0.

$$humedad(\,\%) = (\frac{HUMEDAD[15:0]}{2^16}) * 100 \tag{3.2}$$

$$temperatura(^\circ C) = (\frac{TEMPERATURA[15:0]}{2^16}) * 165 - 40 \tag{3.3}$$

El proceso de lectura para el sensor INA260 requiere primero la configuración de su funcionamiento, se escribe 0x61FB a la dirección de memoria 0x0 correspondiente a su configuración. Esta escritura dispara una conversión de voltaje y corriente sin promediado con tiempo de conversión de 8.222 ms, el valor de potencia automáticamente se obtiene al finalizarse las conversiones. 20 ms después de disparar una conversión se hacen consecutivamente los cambios de puntero a las direcciones 0x1, 0x2 y 0x3 correspondientes a corriente, voltaje y potencia para recibir 2 bytes. El voltaje y la potencia son de conversión directa, se dividen entre 800 y 100 respectivamente. Para el caso de la corriente es necesario considerar que puede ser medida en ambos sentidos y que el valor obtenido se entrega en complemento a 2 por lo que si el bit 16 es 1 se utiliza la ecuación 3.4 y si es 0 se utiliza 3.5. Al igual que con los 2 sensores anteriores, se asignan los valores de corriente, voltaje y potencia en las posiciones 3, 4 y 5 del arreglo flotante si no ocurrió error en la lectura, en caso contrario se asigna 0.

$$corriente(A) = \frac{2^{16} - CORRIENTE[15:0] + 1}{800} \tag{3.4}$$

$$corriente(A) = \frac{CORRIENTE[15:0]}{800} \tag{3.5}$$

Al terminar la recolección de las variables se formatea a una cadena extrayendo los valores del arreglo de flotantes. El largo de la cadena depende de la magnitud de la variable, para el caso de la luz se le asignan 9 caracteres en total, 5 para la temperatura, humedad, corriente y voltaje, mientras que para la potencia se asignan 6, todas las variables se imprimen con 2 puntos decimales. El reloj de tiempo real interno del microcontrolador que se sincroniza mediante HTML se encarga de llevar el conteo del tiempo lo que permite extraer los datos de día, hora y minuto que son adjuntados con salto de línea al final a la cadena.

La variable contenedora de la medición más reciente es enviada a la función mostrada en la Sección de código B.14 en el Apéndice B. Esta función inicialmente recorre la nueva medición hasta encontrar el salto de línea o terminador de cadena que indica el tamaño que tiene, se utiliza este tamaño para hacer un desfase hacia la derecha a partir del último byte de la medición 287 y después se escribe la nueva medición en el inicio. Finalmente, es necesario verificar el estado de las mediciones, si el número coincide con 288 es necesario realizar un traslado de las mediciones a un archivo en flash y reiniciar el acumulador.

Manejo de archivos

Dado que el objetivo del sistema es ofrecer y almacenar información, se crean múltiples funciones para facilitar el proceso de almacenamiento y entrega de datos. Cuando se llega a la medición 288 la función que acumula las mediciones desbloquea un mensaje que inicia el proceso de

generación de archivos como se ve en la Sección de código B.15 en el Apéndice B. Se comienza por armar el nombre del archivo que para estar disponible desde HTML necesita comenzar con "www", la estructura final del nombre es "www/jason/diaX.csv", la X representa el número de día del archivo, este número se reinicia cada semana. Cada medición puede llegar a contener 55 B y se acumulan 288 por lo que su tamaño máximo sin contar las cabeceras es de 15 840 B, la hoja de datos del microcontrolador recomienda trabajar con tamaños potencia de dos por lo que se usa 16 384 B. Con el archivo creado y cerrado se abre nuevamente para escribir el contenido del arreglo acumulador añadiendo como cabecera *sep=,\n* para indicar "," como separador de columnas y *Luz(Lux),Temperatura(C),Humedad(%),Corriente(A),Voltaje(V),Potencia(W), Día,Hora,Minuto\n* como títulos, cada nueva fila se indica con "\n". Al terminar la escritura el acumulador de mediciones es reiniciado.

Para cuestiones de monitoreo se añade la creación de archivos a petición desde HTML, esto desbloquea un mensaje esperado dentro de la tarea mostrada en la Sección de código B.16 en el Apéndice B. La función crea un archivo con el nombre "www/jason/historial.csv", el archivo contiene las mediciones presentes en el acumulador con las mismas cabeceras que el respaldo por días.

Finalmente, si se presentan eventos que ocasionen al Watchdog reiniciar el sistema o se tiene una petición de reinicio, previo a esto se desbloquea un mensaje dentro una tarea que se encarga de respaldar las mediciones presentes en RAM en un archivo con nombre "bcp/med.txt" que al reiniciar el sistema es leído y cargado nuevamente.

3.2.3 Control de errores

Dentro de las tareas que se ejecutan en el sistema se detectaron eventos aleatorios que causan comportamientos erróneos o congelamiento haciendo imposible continuar su funcionamiento sin reiniciar físicamente la alimentación por lo que se implementa un sistema de control de errores con ayuda del Watchdog. Cada tarea, antes de comenzar un proceso que pueda ocasionar ciclos o esperas infinitas, activa dentro de una variable su bandera de proceso y al terminar ese proceso la misma tarea se encarga de bajar su bandera. Si durante el proceso ocurre algún problema y no es posible terminarla en 10 segundos, el Watchdog genera una interrupción para verificar el comportamiento del sistema en general, si no se le ha indicado un reinicio revisa la variable que contiene las banderas para ver qué tarea evita que el sistema entre en bajo consumo (reiniciando la cuenta del Watchdog) y con base en la bandera que está activa se toma la decisión correspondiente, en la Figura 3.12 se muestra el proceso de control que sigue el Watchdog para la atención de problemas y en la Sección de código B.17 en el Apéndice B se muestra el código.

La implementación del Watchdog como control general del sistema otorga mayor seguridad y autonomía, haciendo que errores en el bus o de conexión no generen ciclos infinitos o trabas dentro de los procesos, tomando acciones de reinicio de tareas, de procesos o de periféricos según sea el caso.

3.2.4 Interfaz de usuario

Para la configuración del sistema se necesitan los parámetros de conexión como el nombre del punto de acceso, la contraseña y el tipo de seguridad, de igual manera es necesario el periodo de recolección y el modo de inicio por lo que se requiere interacción con el usuario para la recepción de datos, también a la hora de entregar mediciones y monitorear el comportamiento es necesario tener un medio por el cual sea posible la comunicación bidireccional. Para cumplir estas necesidades se desarrollaron 2

interfaces sobre un servidor WEB embebido en el microcontrolador, el código fue escrito en HTML y JavaScript utilizando plantillas de CSS propias de los códigos ejemplos de Texas Instruments para el entorno visual. En la Figura 4.1 se puede ver la página WEB con la que el sistema de monitoreo responde al conectarse mediante Wi-Fi al AP generado e ingresando a la dirección IP predeterminada.

La interfaz HTML de configuración permite recopilar la información y enviarla mediante un POST token de usuario que genera interrupción en el microcontrolador para segmentar y utilizar la información. Se agrega un botón extra para sincronizar la hora del microcontrolador con la hora del dispositivo que lo está programando mediante un POST token propio del sistema no controlable por el usuario.

Al ingresar a la dirección IP que el Router le otorga al sistema se muestra la página de la Figura 4.2, desde esta página es posible visualizar mediante GET tokens cuánto tiempo lleva conectado el dispositivo a un Router, el número de mediciones que se lleva en RAM, la cantidad de respaldos por días que se han hecho, y la última medición realizada. Se ofrecen botones de generación automática para la descarga de los archivos de respaldo y generación de archivos a petición. Para situaciones que lo requieran, se agrega un botón que dispara el proceso de reinicio dentro del sistema.

3.2.5 Versiones y consideraciones

Las primeras pruebas de software se hicieron sobre el SensorTag que incluye muchos sensores y un diseño compacto del microcontrolador CC3200 que al ser aplicación comercial ofrece flexibilidad a la hora de manipular el código. Los métodos de acceso al SensorTag sólo permiten debuggear y ejecutar códigos sobre RAM más no sobreescribir en FLASH el código original. Las pruebas en esta plataforma terminaron con un sistema de recolección utilizando los sensores HDC1010 y OPT3001 presentes en la placa comunicandose por I^2C con entrega de mediciones por UART y mediante TCP como se muestra en la Figura A.3 y Figura A.2 en el Apéndice A.

Antes de implementar HTML se intentó utilizar los protocolos de comunicación TCP y WEB sockets que presentaron múltiples inconvenientes. Los problemas de eventos aleatorios en las conexiones TCP y WEB sockets se corrigieron con la implementación del sistema operativo en tiempo real, pero ambos protocolos requieren que previo a la conexión el cliente conozca la dirección IP y el puerto para establecer comunicación, haciendo al sistema menos versátil por lo que a pesar de representar comunicación rápida y segura, establecer puentes de comunicación limita las capacidades del sistema, entonces, se descartan y se reemplazan con los métodos POST y GET propios de HTML.

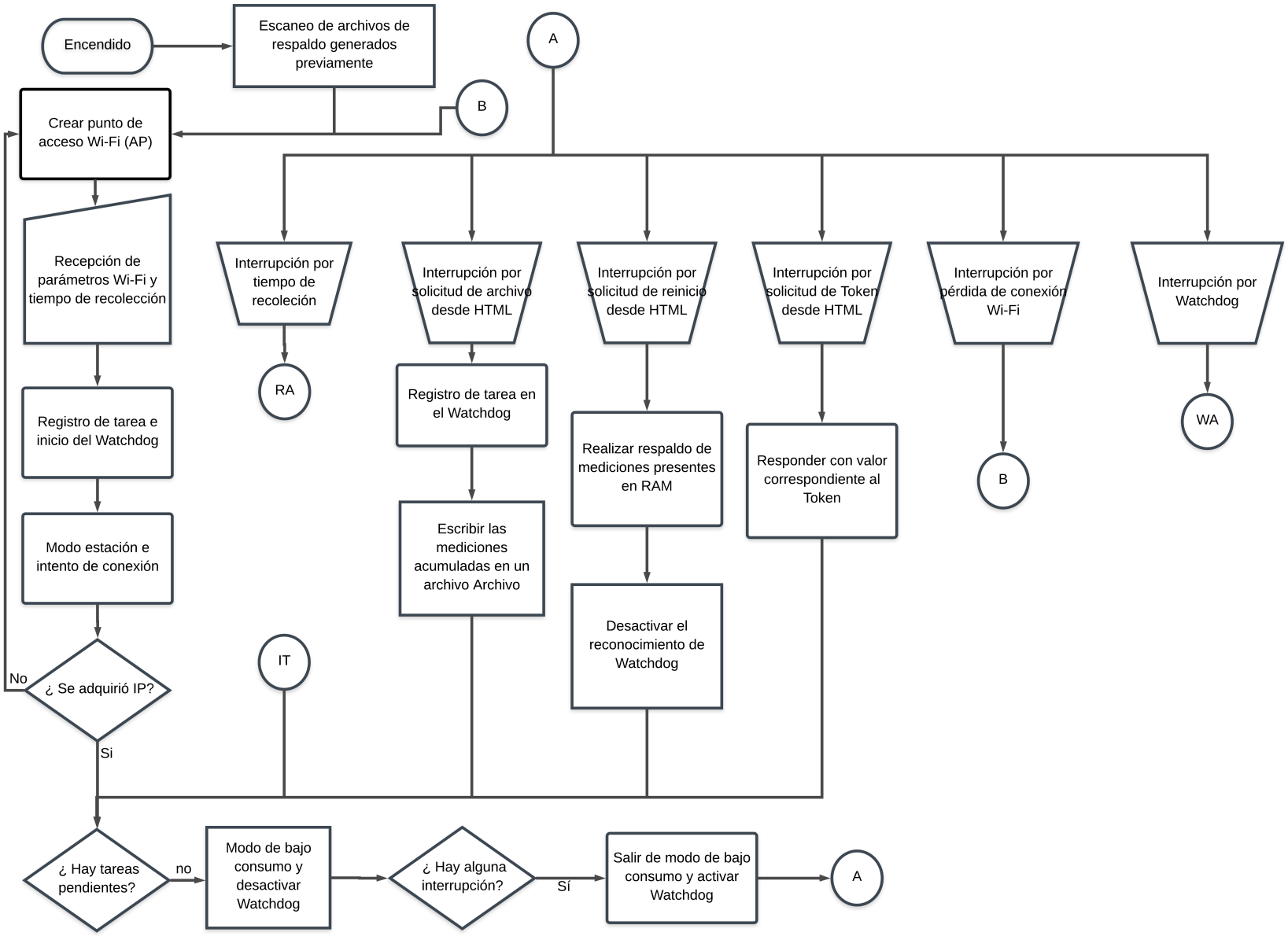


Figura 3.10: Diagrama de flujo del sistema de recolección.

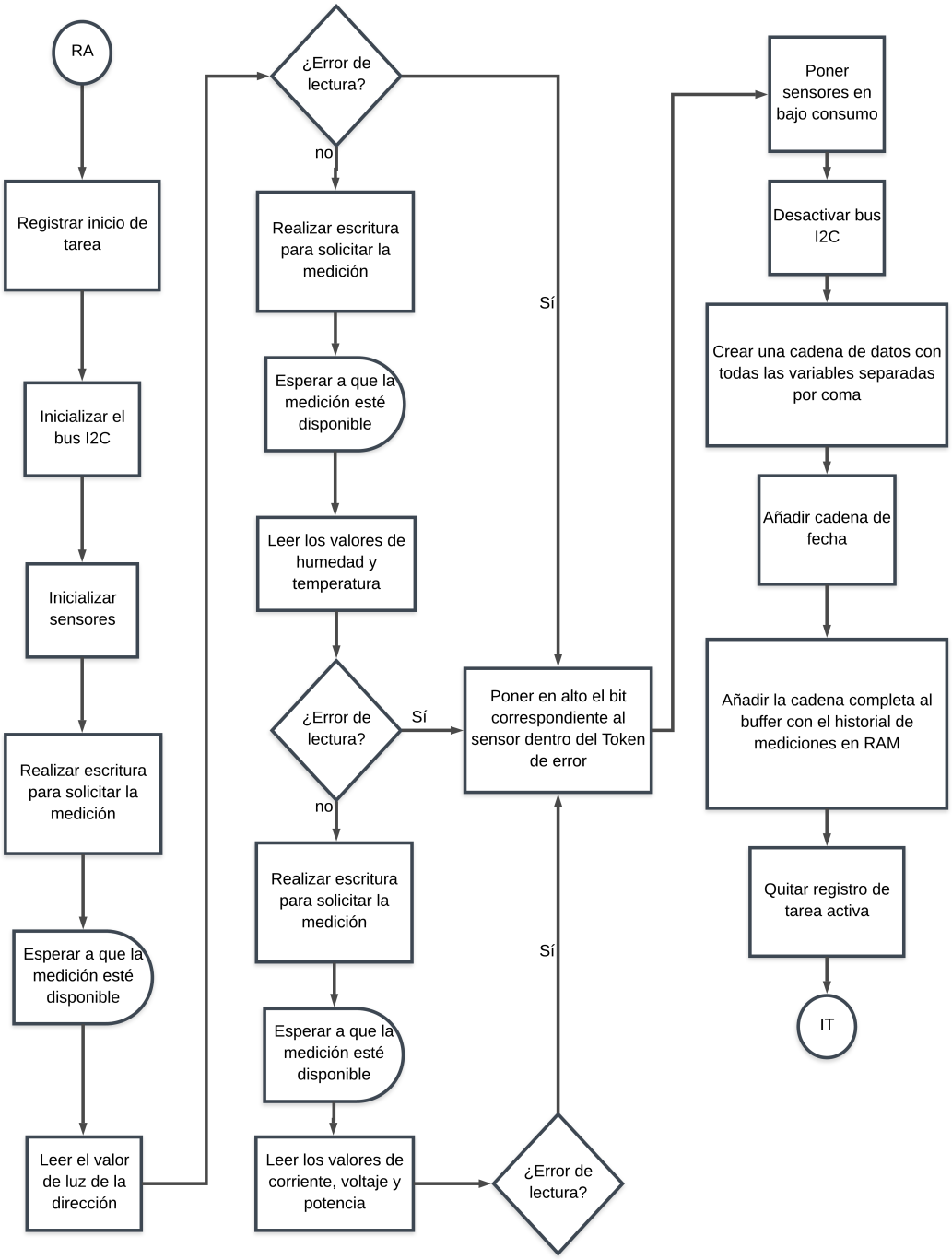


Figura 3.11: Diagrama de flujo de la tarea de recolección de variables.

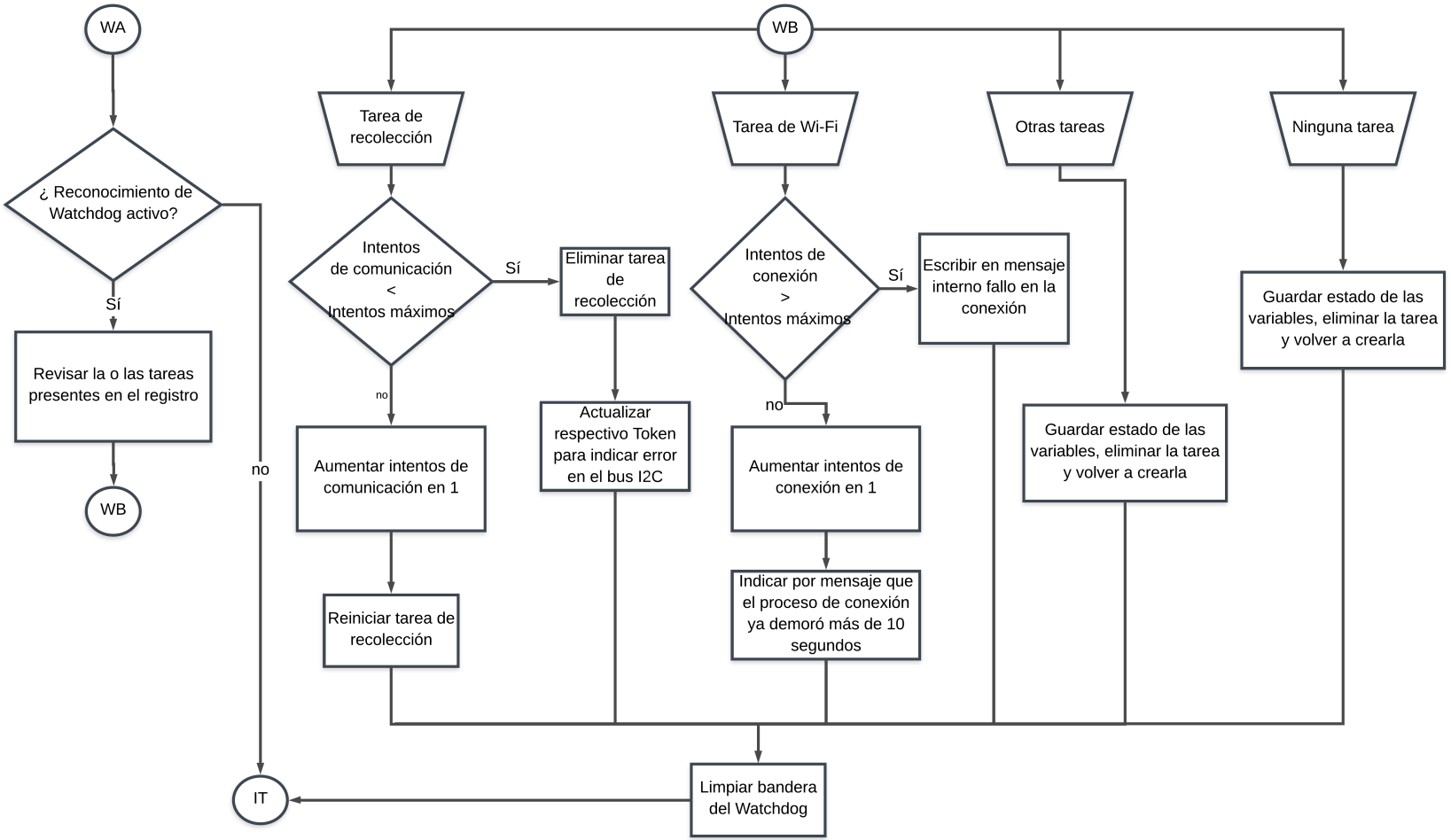


Figura 3.12: Diagrama de flujo del control de errores con Watchdog.

Capítulo 4

Resultados

En este capítulo se presenta el procedimiento para la configuración y uso del sistema, y la extracción de datos para un posterior análisis. Además, se muestra como está estructurada cada medición y a su vez un ejemplo de un archivo muestra descargado del sistema. Estos datos son validados al compararse con las mediciones de la estación meteorológica del instituto. sin embargo, debido a que el sistema y la estación están ubicados en áreas ligeramente diferentes, los datos son normalizados para analizar la tendencia general, se realizaron también mediciones comparativas con otros dispositivos para la validación de los sensores.

4.1 Uso y configuración de la interfaz

Al encender el sistema, se crea un punto de acceso de nombre Sensor-Fi, al cual sólo un dispositivo se puede conectar a la vez, cuando se establece conexión, si se ingresa a la dirección "192.168.1.1" desde cualquier navegador, se muestra la ventana de configuración del sistema en interfaz HTML, como se ve en la Figura 4.1. Antes de ingresar los datos requeridos, es necesario otorgarle la hora al sistema, para que se sincronice y la utilice para llevar el registro de fecha que se anexa a las variables.Para el caso del periodo de recolección, es necesario tener en cuenta que el mínimo aceptable es de 2 segundos, debido a que los tiempos de conversión de los sensores están fijados al máximo, lo que vuelve las lecturas lentas y limita el muestreo. Al terminar de escribir los datos del punto de acceso al que se busca conectar, se presiona el botón actualizar, esto llama a una instrucción `POST` dentro del código en C que recibe los parámetros para realizar la conexión Wi-Fi sin ninguna restricción conocida. Para las pruebas del sistema-IoT se elige 300 segundos en el periodo de muestreo, con la intención de coincidir con el intervalo de actualización de otras estaciones de monitoreo, de esta manera se puede evaluar los resultados con 288 mediciones generadas por día.

Si los parámetros recibidos por el sistema son correctos y la red se encuentra dentro del alcance de la antena, a aproximadamente 20 m sin obstáculos, se realiza la conexión con el punto de acceso. Al ingresar desde un navegador en una PC a la dirección IP que el punto de acceso otorga al sistema-IoT, este responde con la interfaz HTML de los sensores que se muestra en la Figura 4.2.

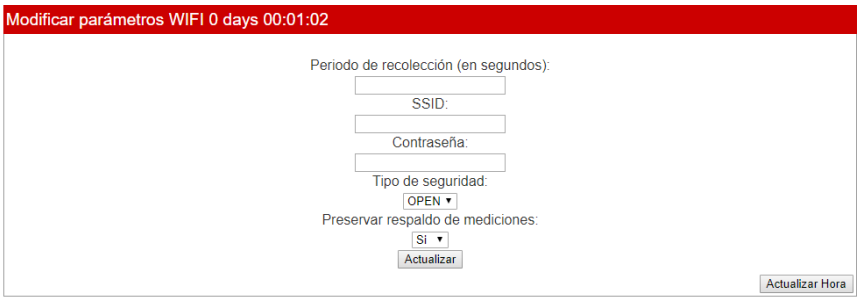


Figura 4.1: Ventana de configuración básica del sistema: Periodo de recolección, Red (SSID), contraseña, seguridad y respaldo.

Se tienen 2 maneras para conocer la IP del sistema, la manera más sencilla es mediante la aplicación "SensorTag" disponible para android y IOS que escanea la red a la que se conecta y busca los dispositivos que respondan con los tokens correspondientes al microcontrolador, como se ve en la Figura 4.3, el dispositivo se identifica como "IoT-SensorTag". La otra manera de conocer la dirección IP asignada es ingresando a la configuración del Router y buscar el nombre mencionado anteriormente en la lista de dispositivos conectados. Al conocer la IP desde, utilizando cualquier navegador, se llega a la ventana de sensores, desde esta es posible visualizar, mediante instrucciones `GET` de HTML, el estado del sistema-IoT; debido a que no hay conexión cerrada punto a punto como `TCP` o `UDP`, no hay restricciones en cuanto a conexiones simultáneas. Además de conocer el estado actual del sistema, la ventana de sensores permite la descarga de dos tipos de archivos: uno de ellos son respaldos de generación automática por día que realiza el propio sistema; mientras que la otra opción permite generar un archivo a petición con las mediciones que aún no han sido respaldadas del día actual.

Figura 4.2: Ventana de petición y monitoreo de datos.

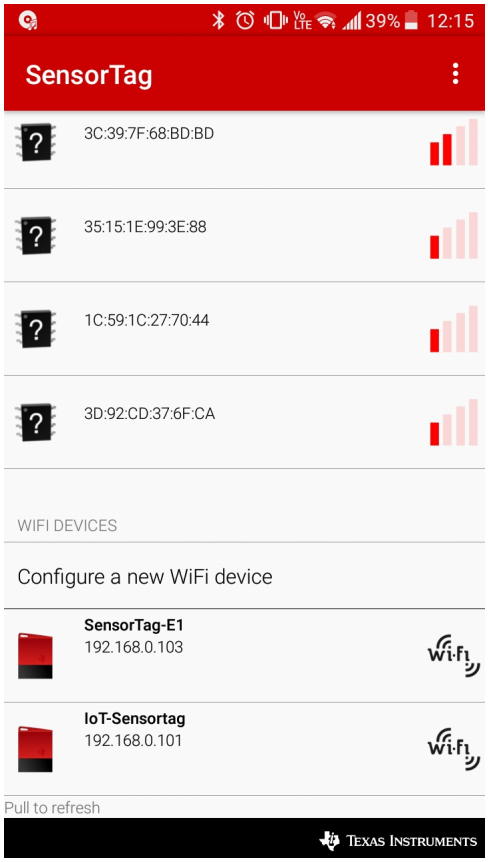


Figura 4.3: Obtención de la IP asignada al sistema por el Router desde la aplicación de Texas Instruments.

4.2 Validación de mediciones con otros sistemas

Con la intención de tener una primera validación de las mediciones del sistema-IoT, los datos recolectados fueron comparados con mediciones extraídas de la estación meteorológica del Instituto Tecnológico de Morelia. Los datos sin un procesamiento previo, permiten una impresión de la tendencia general del entorno de las mediciones, y cómo se ve reflejada en ambos.

Sin embargo, es importante resaltar que el sistema no está bajo las mismas condiciones que la estación meteorológica con la que se compara, por lo tanto posiblemente existirá variación entre las mediciones. Por tal motivo, además de las mediciones de la estación meteorológica, se propone realizar mediciones con instrumentos de medición específicos, para validar las mediciones obtenidas por el sistema. Por ejemplo, para el caso de las mediciones de intensidad de luz se utiliza un luxómetro, para la temperatura y la humedad se usa una tarjeta de desarrollo de Texas Instruments denominada "Sensortag", mientras que los parámetros eléctricos se validan con ayuda de una carga electrónica y una fuente de corriente.

Entonces, con los datos proporcionados por la estación meteorológica y las mediciones del sistema-IoT, dentro del mismo lapso de días se obtienen las gráficas mostradas en las Figura 4.4, Figura 4.5 y Figura 4.6, en las cuales se puede observar que la tendencia de los datos de intensidad de luz, temperatura y humedad, respectivamente. Es necesario mencionar que los datos están normalizados con base a la Ecuación 4.1 utilizada en [49], la cual divide la diferencia entre la muestra actual y la mínima sobre la diferencia entre el mínimo y máximo del rango de datos.

$$x_i^* = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

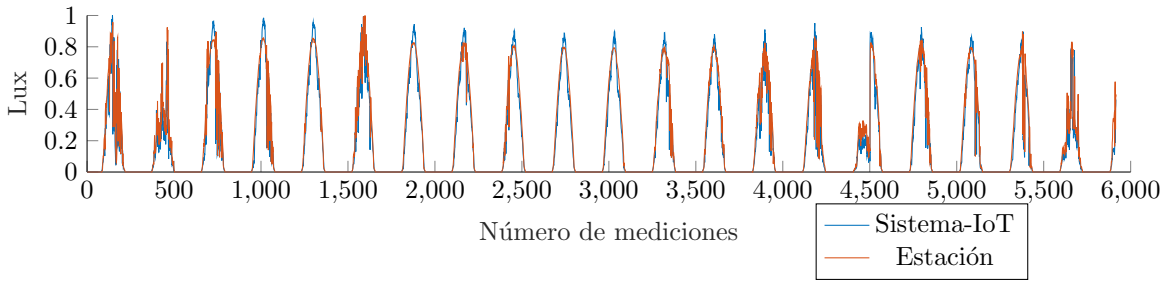


Figura 4.4: Gráfica comparativa entre mediciones normalizadas de luz de la estación meteorológica y el sistema propuesto

De las figuras anteriores se puede observar que los datos representan en promedio entre 21 o 22 días; 6,500 datos aproximadamente. Debido a la cantidad de mediciones, es poco práctico lograr observar el comportamiento a detalle de las mediciones, por lo que en Figura 4.7,Figura 4.8 y Figura 4.9 se muestran los datos de dos días de mediciones de luz, temperatura y humedad, respectivamente.

A diferencia de la humedad y la temperatura, la luz es la variable que más se asemeja en la comparación de la Figura 4.7, ya que los sistemas están expuestos prácticamente a la misma iluminación, con pequeñas variaciones en la posición y ángulo. Por otra parte, la temperatura Figura 4.8 y la humedad Figura 4.9 presentan diferencias notables en cuanto a los valores. Esto puede deberse en gran medida a que los sensores de la estación meteorológica no están bajo las mismas condiciones, por lo que no reflejan las variables bajo las cuales el panel solar opera si no a las del medio en general.

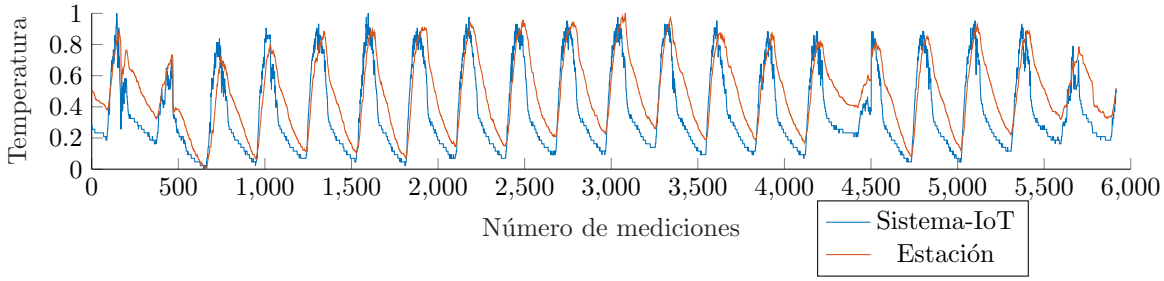


Figura 4.5: Gráfica comparativa entre mediciones normalizadas de temperatura de la estación meteorológica y el sistema propuesto

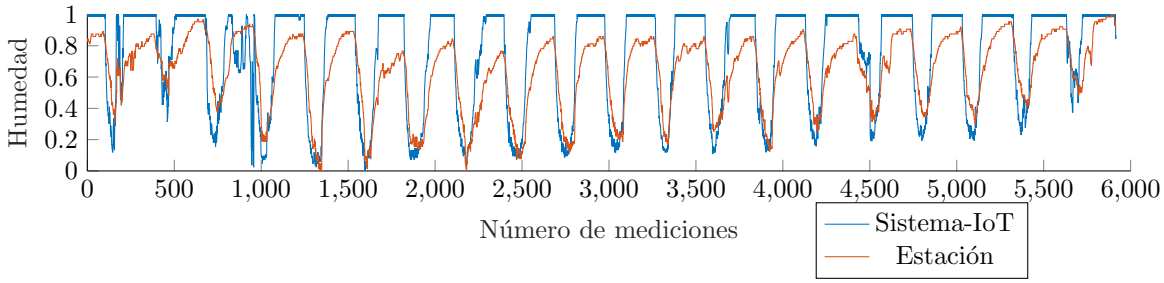


Figura 4.6: Gráfica comparativa entre mediciones normalizadas de humedad de la estación meteorológica y el sistema propuesto

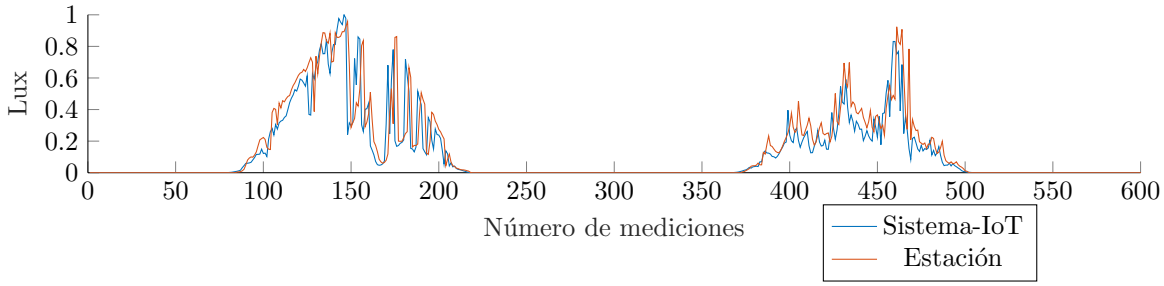


Figura 4.7: Fragmento de la gráfica comparativa entre mediciones normalizadas de luz

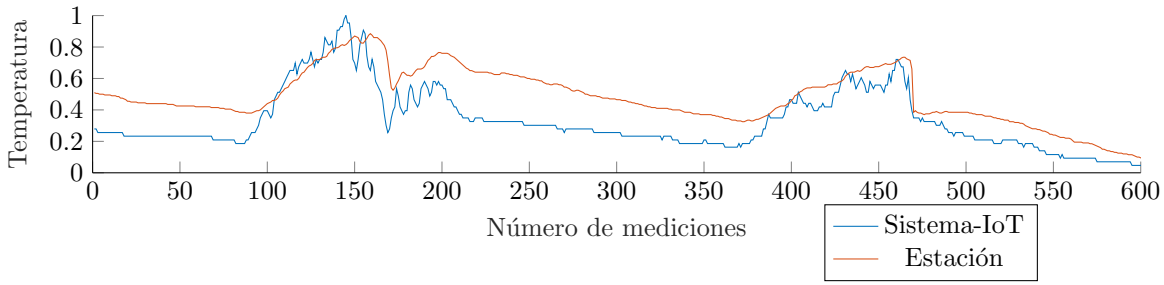


Figura 4.8: Fragmento de la gráfica comparativa entre mediciones normalizadas de temperatura

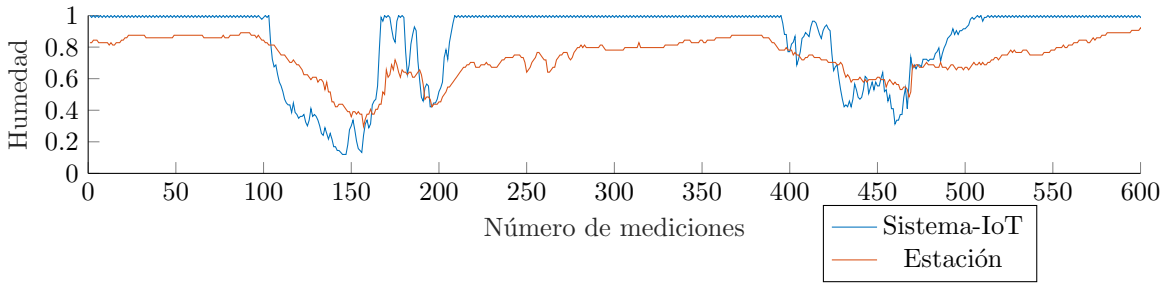


Figura 4.9: Fragmento de la gráfica comparativa entre mediciones normalizadas de humedad

4.3 Error de mediciones

La correlación existente entre las variables del entorno medidas por el sistema de monitoreo y la estación de monitoreo refleja cuanta dependencia existe entre los cambios de una y si se presentan en otra, siguiendo la formula de la Ecuación 4.2, con base en las gráficas se espera que el coeficiente de radiación sea más alto que el de las otras dos variables, y en efecto, los cálculos indican que la luz tiene una correlación de 0.963 indicando que aún si no están expuestos a exactamente la misma iluminación, siguen los mismos patrones, el siguiente coeficiente es el de la humedad, resultando en 0.8352 y finalmente la temperatura con 0.8185, estos 2 valores se encuentran relativamente diferentes y con más variaciones con base en la estación, y era de esperarse, ya que están expuestos directamente a la radiación solar y el viento que pueden calentar o enfriar de manera directa el panel solar.

$$\text{Correlación(X,Y)} = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{\sqrt{\sum(X-\bar{X})^2 \sum(Y-\bar{Y})^2}} \tag{4.2}$$

Además de la comparación realizada directamente con la estación de monitoreo, se hacen comparativas entre diferentes fuentes, para el caso de la humedad, en la Figura 4.10 se muestran valores obtenidos desde 4 fuentes en un lapso de tiempo de 2 horas. En esta Figura se observa una gran variación entre las diferentes fuentes. La estación de monitoreo y Google se esperaría que estuvieran más próximas entre ellas pero al estar en lugares diferentes tienen referencias distintas. Es importante resaltar que el dispositivo comercial de sensado utilizado (Sensortag) y el sistema-IoT usan la misma familia de sensor, y con base en los resultados el Sensortag al no estar expuesto a la temperatura provocada por la radiación directa tiene valores similares a los entregados por Google, mientras que el sistema-IoT al estar bajo las condiciones cercanas al panel tiene más tendencia hacia las mediciones de la estación de monitoreo que se encuentra cerca.

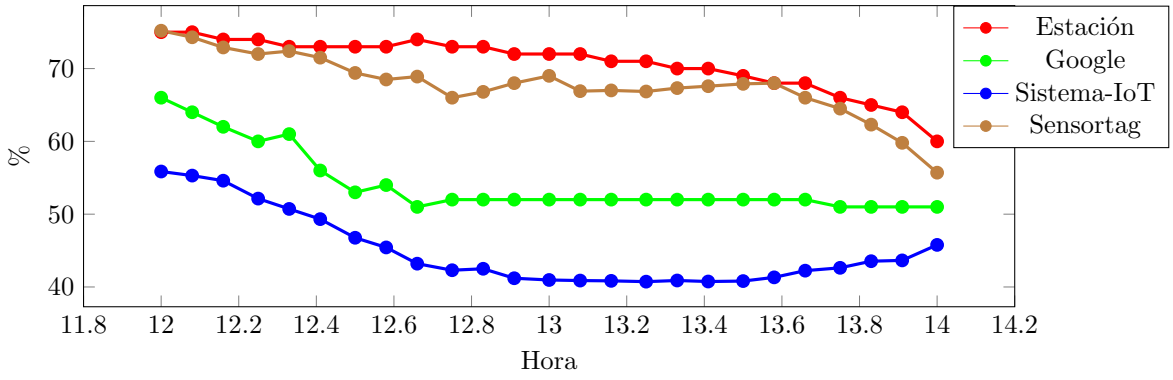


Figura 4.10: Comparación de mediciones de humedad desde diferentes fuentes.

En la Figura 4.11 se pueden observar mediciones de temperatura de 4 fuentes diferentes; la estación de monitoreo, Google, el sistema-IoT y un Sensortag que es colocado bajo las mismas condiciones que el sistema-IoT. La estación y Google reflejan casi la misma temperatura, mientras que el sistema-IoT y el Sensortag reflejan temperaturas aproximadamente 20 grados más altas. Como ya se ha mencionado en la subsección anterior, al estar expuestos directamente a la radiación solar, las temperaturas medidas son más altas y su variación será dependiente de la radiación solar, lo que se traduce en una mayor variabilidad comparado con Google y la estación. Las mediciones realizadas por el Sensortag varían considerablemente entre punto y punto, esto es debido a que el dispositivo hace mediciones a alta velocidad a costa de reducir los tiempos de conversión, esta reducción ocasiona perdida de precisión.

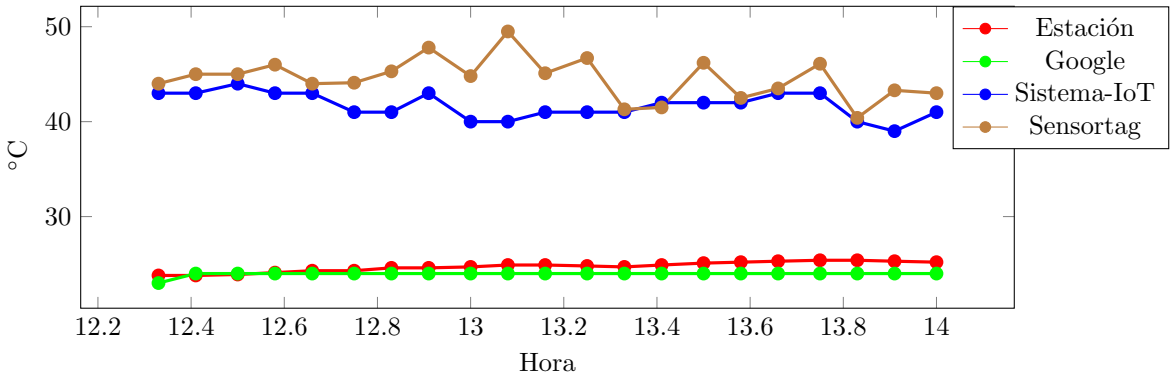


Figura 4.11: Comparación de mediciones de temperatura desde diferentes fuentes.

Las mediciones de luz se comparan con un luxómetro digital, en la Figura 4.12 se muestran los resultados de los valores obtenidos. En un principio se tiene una diferencia entre mediciones de aproximadamente 10k lx que conforme pasa el tiempo se ve reducida hasta 5k. Las variaciones entre mediciones, son debidas en su mayoría al notablemente mayor angulo de captación de luz del luxómetro ya que conforme la posición del sol provocaba una incidencia de luz más directa en el sensor, sus mediciones son más parecidas. Estos resultados no implican que las mediciones de luz estén erróneas, si bien existe diferencia entre los resultados, lo importante es medir correctamente la incidencia directa de luz con base en la inclinación del panel solar, ya que esta es la que presenta mayor generación de potencia y como se ve en las gráficas, al tener incidencia directa se tiende a medir mejor la variable real.

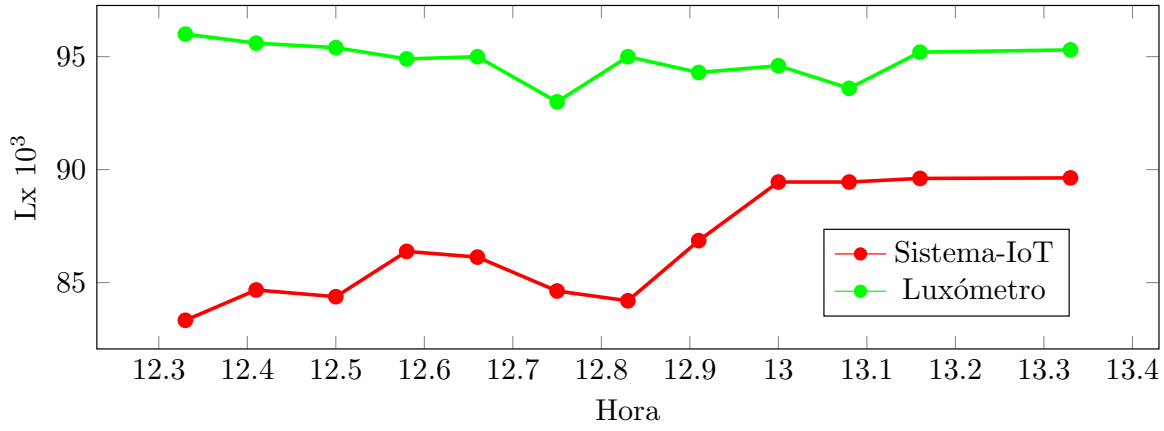


Figura 4.12: Comparación de mediciones de luz entre luxómetro y Sistema-IoT.

Las 3 restantes mediciones son las de corriente, voltaje y potencia, que en realidad son sólo 2 variables (voltaje y corriente) ya que la potencia es el resultado de la multiplicación de las 2 anteriores por lo que es necesario verificar que estén bien registradas, para esto se utiliza una fuente de corriente y una carga electrónica. En la Figura 4.13 y Figura 4.14 se observan las gráficas de los resultados obtenidos.

En ambas gráficas se limitan los puntos marcados, lo que permite observar de mejor manera los resultados; entre las mediciones obtenidas se encuentra una variación máxima de 6.25 mV en el rango de 0-24 y 2.5 mA en el rango de 0-1A por lo que se puede concluir que el sensor utilizado es de alta

precisión y refleja fielmente los valores.

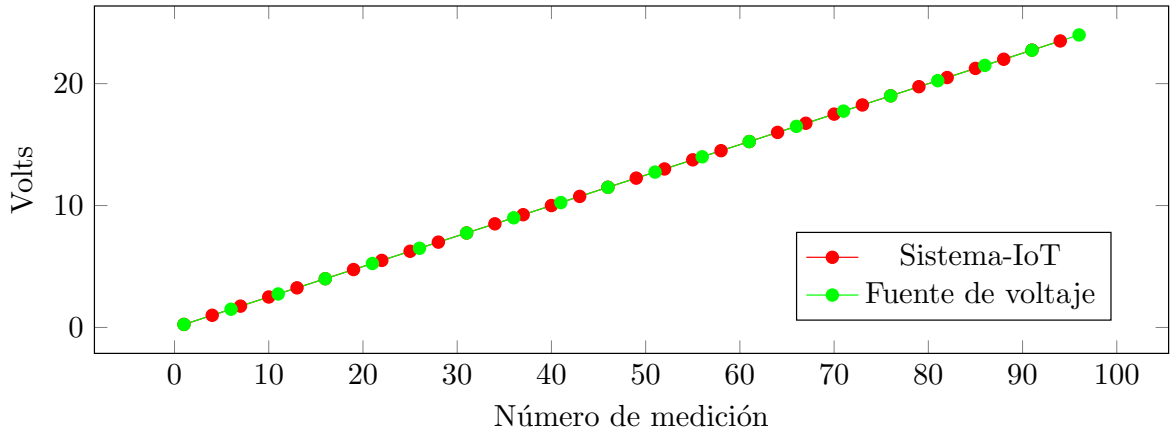


Figura 4.13: Comparación de mediciones de voltaje entre fuente de voltaje y Sistema-IoT.

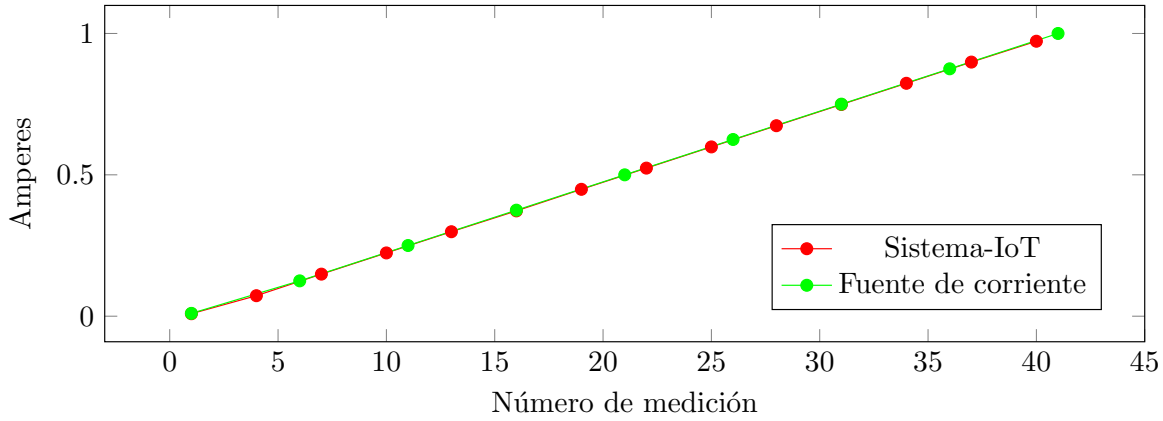


Figura 4.14: Comparación de mediciones de corriente entre fuente de corriente y Sistema-IoT.

4.4 Estructura de datos en el sistema

Las mediciones están estructuradas de manera que sean fácil de manipular y/o importar a tablas de cálculo, procesadores de texto o entornos de análisis de datos como MatLab, Python o R-Studio. El archivo tiene formato ".csv" y usa una cabecera propia, cada elemento es separado con una coma simple "," y cada cambio de línea, con saltos de línea. El archivo que se genera con 288 mediciones,pesa máximo 15.840 B, entonces cada archivo no representa gran porcentaje, en relación a la capacidad de 8 MB de la FLASH; siendo posible almacenar aproximadamente 115,200 mediciones. En la Tabla 4.1 se muestra un extracto de la estructura con la que se entregan las mediciones del archivo.

4.5 Consumo energético

El consumo de corriente inicial del sistema-IoT en modo punto de acceso es de 80 mA. Cuando el sistema entra en modo estación, su consumo se reduce a 25 mA mientras espera realizar una lectura. En ese momento que el sistema realiza una medición, 2 segundos aproximadamente, el sistema sale del modo de bajo consumo, activando los subsistemas que le permiten hacer las mediciones, este cambio en el sistema hace que el consumo energético suba a 40 mA. En ese sentido y para garantizar la autonomía del sistema se incorpora una batería de 2.600 mA, esta batería permite alimentar al sistema durante 4.3 días, sin recarga. No obstante la batería es recargada mediante un panel solar pequeño que genera un máximo de 300 mA, si esta generación de energía ocurre durante aproximadamente 3 horas al día, el funcionamiento del sistema podría operar indefinidamente.

4.6 Aspectos mecánicos e instalación

En la Figura 4.15 se muestra como el sistema se encuentra sujeto al panel solar, se elige esta opción para que la medición de variables a las que está expuesto el panel sean representativas. En la Figura 4.15 se ve la batería en (1) y el sensor de corriente en (2). Mientras que en la Figura 4.16 se muestra el panel solar que recarga la batería en (3), y el sensor de luz, humedad y temperatura en (4).

Es importante resaltar que la placa de parámetros eléctricos y el Launchpad (Figura 4.15) se les adaptó una caja aislada con silicón que permite cubrirlas completamente para poder exponer el sistema a una operación en campo representativa; véase Figura 4.17. A pesar de estas precauciones, se presentaron problemas por la exposición al entorno de las conexiones, esto se soluciono al aislar con resina las conexiones tanto de los conectores de comunicación, como las terminales de los sensores. Como se ve en la Figura 4.18, el sistema-IoT es diferente a la propuesta inicial, ya que en ella no se tenían considerados los factores del medio como humedad y lluvia que obligan a tomar medidas y diseños diferentes. En esta misma figura, se puede observar que sobre la placa de parámetros del medio se coloca un vidrio que funciona como atenuador de la radiación solar, debido a que el limite del sensor es de 83k lx y la radiación solar máxima medida con el luxómetro es de 100k lx aproximadamente, al caracterizar el vidrio utilizado como atenuador se encontró una atenuación del 45 % que se considera a la hora de extraer los valores del sensor.

Tabla 4.1: Extracto de datos entregados por el sistema-IoT.

Luz(Lux)	Temperatura(C)	Humedad(%)	Corriente(A)	Voltaje(V)	Potencia(W)	Día	Hora	Minuto
17213.44	31.00	41.00	00.34	14.84	005.12	9	15	7
19732.48	30.00	43.00	00.40	14.86	005.90	9	15	2
08158.72	28.00	44.00	00.17	14.73	002.47	9	14	57
14530.56	30.00	43.00	00.29	14.80	004.35	9	14	52
28026.88	32.00	41.00	00.55	14.95	008.20	9	14	47
18508.80	30.00	44.00	00.37	14.85	005.56	9	14	42
14504.96	29.00	44.00	00.29	14.81	004.33	9	14	37
13184.00	29.00	44.00	00.26	14.80	003.92	9	14	32
11407.36	28.00	46.00	00.23	14.77	003.38	9	14	27
10286.08	30.00	45.00	00.20	14.75	002.97	9	14	22
12328.96	32.00	37.00	00.23	14.78	003.47	9	14	17
17889.28	33.00	37.00	00.33	14.83	004.88	9	14	12
27013.12	34.00	37.00	00.48	14.91	007.12	9	14	7
23674.88	33.00	38.00	00.41	14.88	006.16	9	14	2
21831.68	34.00	35.00	00.37	14.85	005.49	9	13	57
35819.52	39.00	27.00	00.56	14.96	008.40	9	13	52
40304.64	37.00	29.00	00.61	14.98	009.10	9	13	47
39792.64	41.00	26.00	00.60	14.98	008.99	9	13	42
39792.64	41.00	25.00	00.61	14.98	009.06	9	13	37
40683.52	40.00	28.00	00.61	14.98	009.10	9	13	32
42250.24	43.00	24.00	00.61	14.99	009.10	9	13	27
43315.20	41.00	26.00	00.61	14.98	009.09	9	13	22
44482.56	41.00	26.00	00.62	14.99	009.25	9	13	17
44175.36	40.00	28.00	00.62	14.99	009.25	9	13	12
44482.56	40.00	31.00	00.62	14.99	009.25	9	13	7
45137.92	38.00	32.00	00.62	14.99	009.29	9	13	2
46551.04	36.00	32.00	00.62	14.99	009.36	9	12	57
46120.96	39.00	31.00	00.61	14.99	009.14	9	12	52
44789.76	38.00	35.00	00.62	14.99	009.24	9	12	47

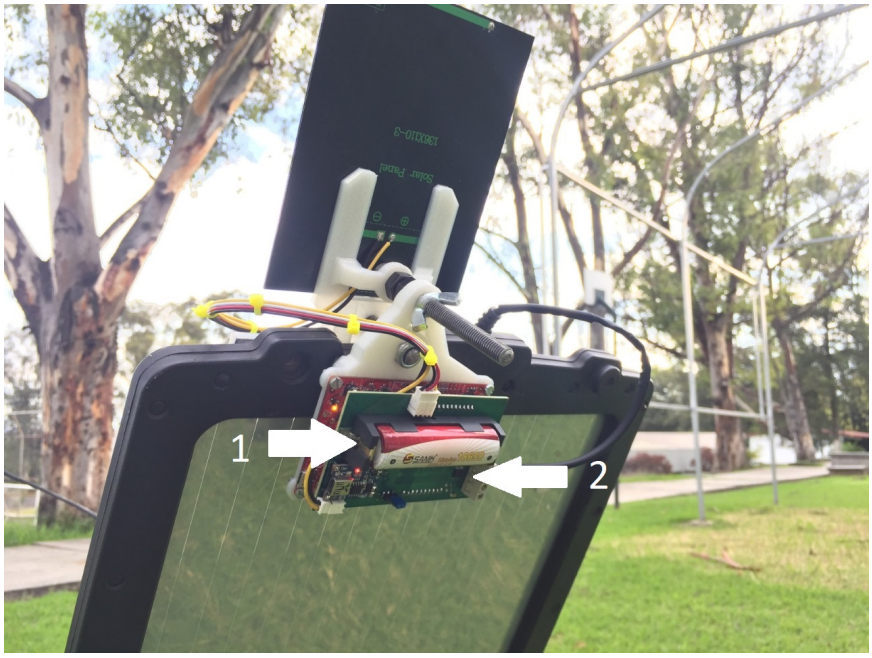


Figura 4.15: Sistema de medición montado sin cobertura (vista trasera) (1) batería del sistema-IoT, (2) sensor de parámetros eléctricos.

Figura 4.16: Sistema de medición montado sin cobertura (vista delantera) (3) panel solar para carga de bateria (4) sensor de luz.

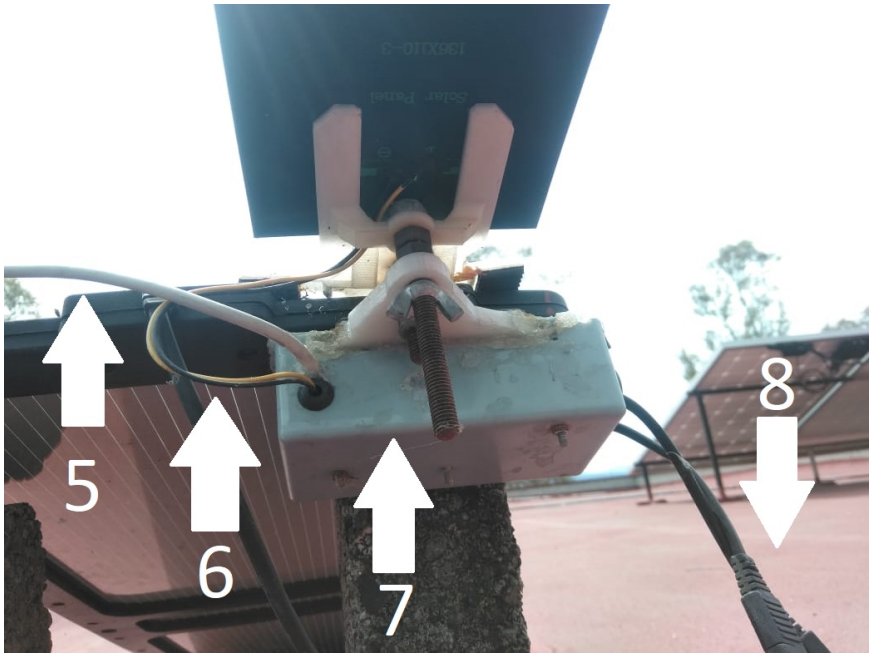


Figura 4.17: Sistema-IoT con cobertura expuesto al medio. (5) Alimentación y bus de comunicación (6) salida del panel del sistema (7) cobertura del launchpad y placa de sensor electrico (8) conexión del panel solar medido hacia la carga.

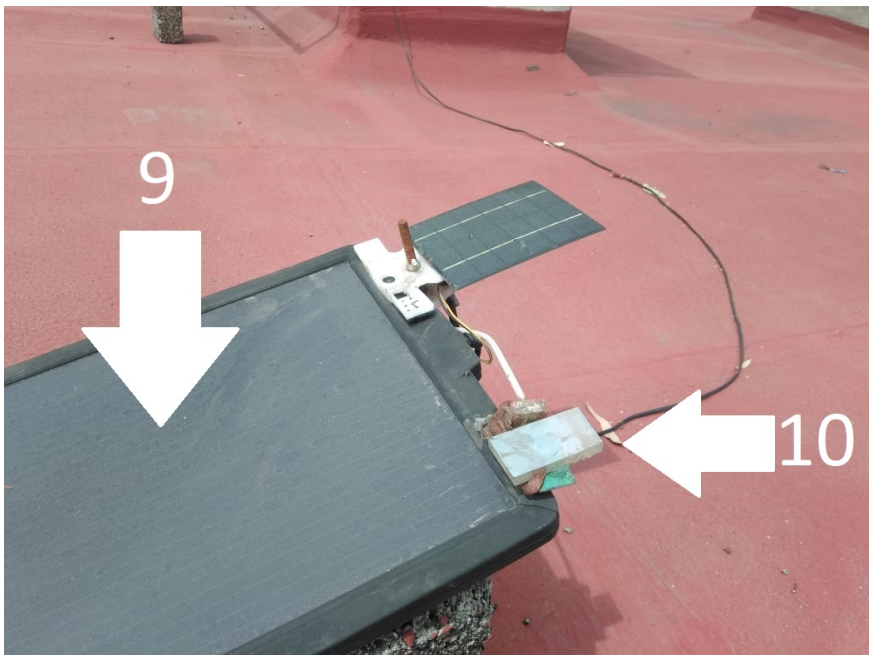


Figura 4.18: Implementación final del sistema-IoT. (9) Panel solar medido (10) placa de sensores del medio.

Capítulo 5

Conclusiones y trabajo futuro

Considerando el desempeño, capacidades de lectura y representación de los datos que han sido mostrados en capítulo previo, se puede indicar que el sistema IoT desarrollado es una herramienta versátil capaz de medir y almacenar la variables eléctricas y ambientales con una precisión comparable a equipos comerciales especializados, además permite lecturas en tiempo real y acceso a las mediciones desde el exterior inalambricamente.

El protocolo de comunicación serial I^2C, con las debidas consideraciones eléctricas, permite acoplar múltiples dispositivos de manera rápida y confiable, sin representar hardware extra. Además el sistema es versátil e interactivo, gracias a la comunicación inalámbrica Wi-Fi y los servicios HTML, que son una tecnología ampliamente utilizada actualmente.

La estructura de las mediciones obtenidas es clave para la compatibilidad y el alcance del proyecto, por lo que la estructura de los archivos lleva como objetivo ser reconocible por procesadores de texto o programas de entornos matemáticos. El sistema se mantiene en modos de operación que reducen la corriente consumida sin pérdida de rendimiento, debido a que se tiene control preciso sobre los componentes y tareas que se activan a cada momento, resultando en procesos correctamente controlados y eficientes. Con la implementación de su propio panel solar se obtiene un sistema completamente autónomo y funcional, capaz de monitorear el panel y con comunicación remota de alta compatibilidad con otros dispositivos y tecnologías. Los circuitos electrónicos son sensibles ante las variaciones del medio por lo que tintas antisoldantes, carcasas de protección y resina son necesarias a la hora de exponerlos al medio.

Con los parámetros del medio registrados junto a los valores eléctricos del panel se pueden llegar a establecer relaciones entre las variables de entrada y salida del panel solar, esto es parte del trabajo futuro que se pretende implementar en un proyecto de posgrado.

El diseño del sistema a pesar de ser funcional y cumplir los objetivos tiene áreas de oportunidad, por ejemplo; la implementación de escritura a servidores externos que disminuiría el consumo energético de manera drástica hasta en un 80 %, con el único inconveniente de perder la interacción por HTML, también es necesario, para diseños más compactos cambiar la presentación del microcontrolador CC3200 a un modulo con lo esencial o a un diseño propio no destinado a desarrollo. Los sensores también

representan limitantes como: la saturación del sensor de luz y la a veces inestable funcionalidad del sensor de humedad y temperatura, de hecho, es recomendable utilizar sensores por separado para la humedad y temperatura ya que en ocasiones, los efectos de alta humedad generan condensación dentro del mismo sensor que se solucionan calentando el propio dispositivo. Si fuese necesario recolectar mediciones de paneles solares que superen los 36 V o 15 A de generación en su salida, será necesario cambiar el sensor INA260 a otro con mayores capacidades.

Anexos

Apéndice A

Versiones anteriores del sistema

El sistema-IoT final que ha sido tratado en esta tesis, fue el resultado de la selección entre múltiples maneras de dar solución a las problemáticas presentadas, en este apartado, se muestran imágenes representativas de la primera versión de hardware y software que se fueron implementadas.

La primera versión del PCB integraba el sensor de humedad y temperatura HDC1010, el sensor de parámetros eléctricos INA260 y el sensor de luz OPT3001 en una misma tarjeta, véase Figura A.1. Este primer diseño se hizo compatible con el conector `SC` del SensorTag de Texas Instruments, por donde se comparten las lineas del bus I^2C y alimentación. Para la recolección de las variables se implementaron dos vías; comunicación serial UART como se muestra en la Figura A.2 y mediante comandos por TCP Figura A.3.

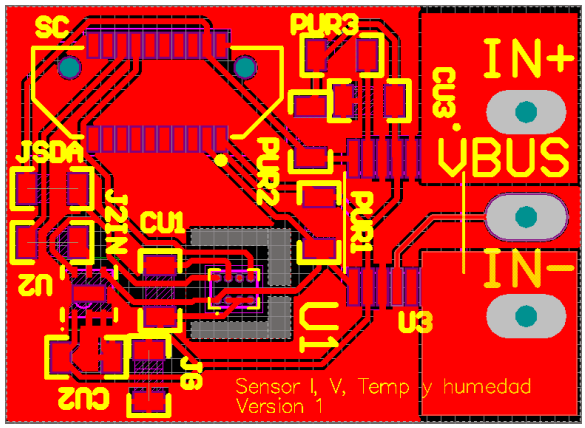


Figura A.1: Primer prototipo placa de recolección.

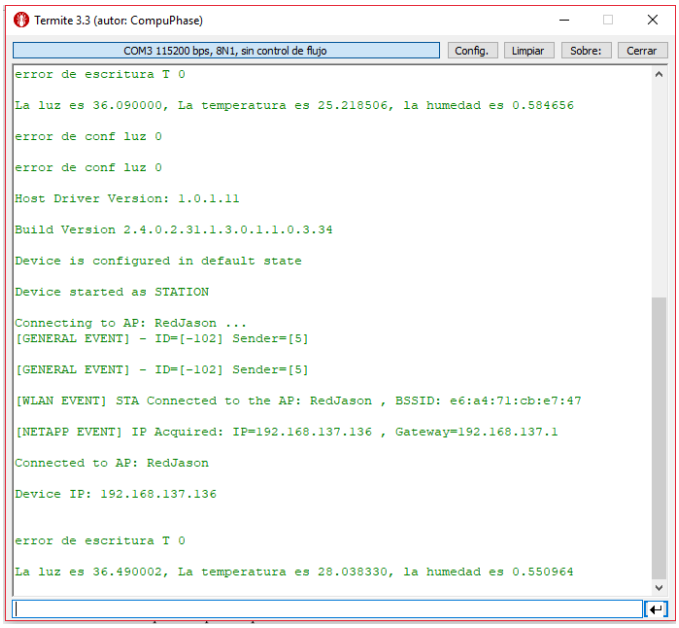


Figura A.2: Terminal UART conectada al sistema-IoT.

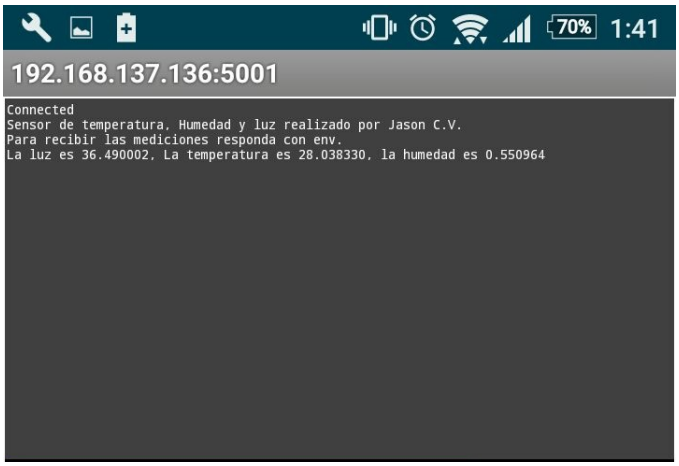


Figura A.3: Comunicación desde cliente TCP con la primera versión del sistema-IoT.

Esta primera versión del sistema tuvo mayor aplicación a la hora de desarrollo que de implementación, ya que gracias a la comunicación UART es posible monitorear las etapas de funcionamiento del sistema y en caso de encontrase errores se pueden detectar con mayor facilidad, la comunicación UART permaneció en el código aunque no fuera utilizado, permitiendo futuras mejoras. A diferencia de UART, la comunicación TCP implementada no prevaleció, ya que si bien representa comunicación rápida y segura, no es de fácil interacción ni tan ampliamente compatible como lo es una página en HTML que cambia comandos por interfaz visual, volviendo más amigable la interacción para el usuario.

Apéndice B

Código implementado

En las siguientes páginas, se explica el código implementado dentro del sistema-IoT con el que se logró obtener los resultados. La mayor parte de la funcionalidad del sistema-IoT se encuentra escrita en lenguaje C, utilizando la versión 1.3 del kit de desarrollo de software proporcionado por Texas Instruments (Software Development Kit, SDK por sus siglas en inglés) como apoyo. La interfaz para la interacción con el usuario, es controlada por HTML y JavaScript.

Cuando el microcontrolador se energiza, procede a ejecutar la función `main` presente en la Sección de código B.1, dentro de esta se hacen las funciones esenciales para la inicialización del sistema. Las primeras funciones en ejecutarse son: `BoardInit` y `PinMuxConfig`, que se explican en la Sección de código B.2 y Sección de código B.3, respectivamente. Después, se detecta el motivo de reinicio del sistema con la función `PRCMSysResetCauseGet` de la línea de código 8, este valor es utilizado para condicionar la inicialización. Si el apagado fue por cuestión del Watchdog significa que fue a petición por lo que es necesario realizar un apagado correcto e indicar que hay archivos que recuperar (línea 15). Con el inicio completado, se procede a declarar las tareas propias del microcontrolador para controlar los eventos Wi-Fi y HTTP con la función `VStartSimpleLinkSpawnTask` indicando qué prioridad se les va a otorgar a las tareas (línea 21). A partir de la línea 26 hasta la 70 se crean los mensajes globales que sirven como semáforos para la comunicación e interacción entre tareas y control de procesos con la función `osi_MsgQCreate`.

Finalmente, se crean las tareas con la función `osi_TaskCreate` la cual recibe; descripción, prioridad y el espacio de memoria asignado. Se otorga la prioridad más alta a la tarea `TimerGPIOTask` (Sección de código B.4) ya que es la que atiende directamente los eventos resultantes en caso de interrupción desde GPIO, conexión o petición por reinicio. Se declaran también las tareas `Recolectar` (Sección de código B.12), `FileW` (Sección de código B.16), `FileBackup` (Sección de código B.15), `Status` (Sección de código B.19), `IntHdlr` (Sección de código B.21) y `Respaldar` (Sección de código B.23). Una vez declaradas todas las tareas y mensajes a utilizar, se cede el control al sistema operativo con la función `osi_start` (línea 78).

Es importante resaltar que a lo largo del código implementado existen diversos reportes de errores, de estado o de variables por medio del bus serial UART con la función `UART_PRINT`. Estos reportes no

afectan al funcionamiento del sistema pero si representan una herramienta útil a la hora de corregir errores o implementar nuevas secciones.

Sección de código B.1: Función `main` que controla el inicio y la declaración de tareas y mensajes.

```
void main(void)
{
 unsigned long ulResetCause;
 int iRetVal;
 BoardInit();
 PinMuxConfig();
 ulResetCause = PRCMSysResetCauseGet();
 if( ulResetCause == PRCM_WDT_RESET ){
  HIBEntrePreamble();
  MAP_PRCMOCRRegisterWrite(0,1);
  MAP_PRCMHibernateWakeupSourceEnable(PRCM_HIB_SLOW_CLK_CTR);
  MAP_PRCMHibernateIntervalSet(330);
  MAP_PRCMHibernateEnter();
 }else if(ulResetCause==PRCM_HIB_EXIT){
  RecuperarB=1;
 }
 platform_init();
 g_tUartHndl = uart_open(PRCM_UARTA0);
 tGPIODbgHndl = cc_gpio_open(GPIO_09, GPIO_DIR_OUTPUT);
 cc_gpio_write(tGPIODbgHndl, GPIO_09, 1);
 iRetVal = VStartSimpleLinkSpawnTask(SPAWN_TASK_PRIORITY);
 if(iRetVal < 0){
  UART_PRINT("could not create simplelink task\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tAPGPIO, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tRecolectar, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_wake, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tFILEBackup, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tFILEW, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tHTTPD, NULL, sizeof( unsigned char ),1);
 if(iRetVal < 0){
  UART_PRINT("could not create msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = osi_MsgQCreate(&g_tWkupSignalQueue,NULL,sizeof( unsigned char ), 10);
 if (iRetVal < 0){
```

```
58   UART_PRINT("unable to create the msg queue\n\r");
59   LOOP_FOREVER();
60  }
61  iRetVal = osi_MsgQCreate(&g_Respaldo, NULL,sizeof( unsigned char ), 1);
62  if (iRetVal < 0){
63   UART_PRINT("unable to create the msg queue\n\r");
64   LOOP_FOREVER();
65  }
66  iRetVal =   osi_MsgQCreate(&g_Tarea, NULL, sizeof(unsigned char), 5);
67  if  (iRetVal < 0){
68   UART_PRINT("unable to create the msg queue\n\r");
69   LOOP_FOREVER();
70  }
71  osi_TaskCreate(TimerGPIOTask,(const signed char*)"Tarea para control de estados e
        inicializar el sistema", 2048, NULL, 1, NULL );
72  osi_TaskCreate(Recolectar,(const signed char*)"Tarea recoleccion", OSI_STACK_SIZE,
        NULL, 3, &TH_recolectar );
73  osi_TaskCreate(FileW,(signed char*)"Archivos",OSI_STACK_SIZE,NULL,2,NULL);
74  osi_TaskCreate(FileBackup,(signed char*)"Backup",OSI_STACK_SIZE,NULL,2,NULL);
75  osi_TaskCreate(Status,(signed char*)"Status",OSI_STACK_SIZE,NULL,4,NULL);
76  osi_TaskCreate(IntHdlr,(const signed char*)"Tarea interrupciones", OSI_STACK_SIZE,
        NULL, 2, NULL );
77  osi_TaskCreate(Respaldar,(signed char*)"Respaldar",OSI_STACK_SIZE,NULL,4,NULL);
78  osi_start();
79 }
```

En la Sección de código B.2 se muestra la función `BoardInit` que es llamada sólo una vez durante el proceso de inicio del sistema. La función declara la tabla de vectores (línea 3), lo que permite habilitar interrupciones (líneas 4 y 5) y terminar de iniciar el microcontrolador (línea 16).

Sección de código B.2: Función `BoardInit` que inicializa vectores de la tarjeta.

```
1 static void BoardInit(void){
2  // Set vector table base
3  IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
4  MAP_IntMasterEnable();
5  MAP_IntEnable(FAULT_SYSTICK);
6  PRCMCC3200MCUInit();
7 }
```

La función `PinMuxConfig` de la (Sección de código B.3) contiene toda la configuración necesaria para utilizar los periféricos como se requiere. Se comienza por habilitar los relojes a utilizar, como por ejemplo los de las comunicaciones seriales (UART e I^2C), después, se asignan los pines a sus respectivas funciones, ya sea que funcionen como salida, entrada o como parte de un pin de protocolo.

Sección de código B.3: Función `PinMuxConfig` que configura los pines y relojes a utilizar.

```
1  void PinMuxConfig(void){
2   // Enable Peripheral Clocks
3   MAP_PRCMPeripheralClkEnable(PRCM_UARTA0, PRCM_RUN_MODE_CLK);
4   MAP_PRCMPeripheralClkEnable(PRCM_GPIOA1, PRCM_RUN_MODE_CLK);
5   MAP_PRCMPeripheralClkEnable(PRCM_GPIOA2, PRCM_RUN_MODE_CLK);
6   MAP_PRCMPeripheralClkEnable(PRCM_I2CA0, PRCM_RUN_MODE_CLK);
7   // Configure PIN_55 for UART0 UART0_TX
8   MAP_PinTypeUART(PIN_55, PIN_MODE_3);
9   // Configure PIN_57 for UART0 UART0_RX
10  MAP_PinTypeUART(PIN_57, PIN_MODE_3);
```

```
11  // Configure PIN_64 for GPIOOutput
12  MAP_PinTypeGPIO(PIN_64, PIN_MODE_0, false);
13  MAP_GPIODirModeSet(GPIOA1_BASE, 0x2, GPIO_DIR_MODE_OUT);
14  // Configure PIN_04 for GPIOInput
15  MAP_PinTypeGPIO(PIN_04, PIN_MODE_0, false);
16  MAP_GPIODirModeSet(GPIOA1_BASE, 0x20, GPIO_DIR_MODE_IN);
17  // Configure PIN_08 for GPIOInput
18  MAP_PinTypeGPIO(PIN_08, PIN_MODE_0, false);
19  MAP_GPIODirModeSet(GPIOA2_BASE, 0x2, GPIO_DIR_MODE_IN);
20  // Configure PIN_01 for I2C0 I2C_SCL
21  MAP_PinTypeI2C(PIN_01, PIN_MODE_1);
22  // Configure PIN_02 for I2C0 I2C_SDA
23  MAP_PinTypeI2C(PIN_02, PIN_MODE_1);
24 }
```

El código mostrado en la Sección de código B.4, corresponde a la tarea `TimerGPIOTask` con más alta prioridad dentro del sistema, esta comienza su ejecución declarando variables para sincronizar mensajes, manejadores de archivos e indicadores de estado. En la línea de código 9, se crea un mensaje interno que sirve como bandera para la conexión Wi-Fi, seguido por la inicialización de los módulos internos del microcontrolador con la función `sl_Start` (línea 13). Se accede al sistema de archivos para obtener la información de los respaldos previamente creados presentes en la Flash (línea 18-45) o archivos a petición (línea 47) para después asignarle el valor encontrado a la variable global que se encarga de los nombres de los archivos (línea 46) o activar la bandera para indicar que hay un archivo por petición creado (línea 49). Las líneas 51-53 corresponden a la apertura, lectura y cerrado de archivos previamente respaldados con las mediciones presentes en RAM al momento de apagar el sistema, estos valores, se asignan dentro del arreglo global que contiene las mediciones para comenzar a acumular a partir de estas en caso de haber.

Es a partir de la línea 54 que se entra en un ciclo infinito con ejecución controlada por el mensaje de la línea 97, este se desbloquea cada que surge una petición para reiniciar desde HTML o se presiona un botón. El proceso cíclico comienza desactivando el servidor HTTP y cambiando el modo de funcionamiento Wi-Fi a punto de acceso (AP) con la función `SwitchToAPMode` (Sección de código B.5), una vez cambiado activa nuevamente el servidor HTTP y se queda a la espera de un mensaje generado por el usuario a la hora de enviar los datos de conexión (línea 58). Después, con base en el tipo de inicio requerido por el usuario se decide si se vacía o no, la variable global contenedora de mediciones (líneas 60-63), se limpian bits indicadores de estado, se vuelve a detener el servidor HTTP, se guardan los parámetros recibidos y se cambia el modo Wi-Fi a estación (STA) con la función `SwitchToStaMode` (Sección de código B.6). Con el Wi-Fi en modo estación (línea 69) se vuelve a activar el servidor HTTP, se establecen las políticas de energía para operación normal, se asigna el nombre de "IoT-Sensortag" al dispositivo para después proceder a intentar una conexión al punto de acceso otorgado por el usuario (línea 83) con la función `WlanConnect` (Sección de código B.7), es necesario mencionar que previo al proceso de intento de conexión, se levanta la bandera dentro de la variable global tareas (línea 82).

Con base en el resultado de la función que intenta conectar al punto de acceso, se puede proceder a entrar en modo de bajo consumo y continuar el funcionamiento normal (línea 89) o se desbloquea el mensaje (línea 97) con la función de la línea 85 para que se repita el proceso cíclico. Si es la primera vez que se realiza el ciclo, se declaran las 2 fuentes posibles para salir del modo de bajo consumo (además de las interrupciones HTTP) que son; la interrupción al presionar un botón (línea 92) y las interrupciones periódicas del timer que controla los tiempos para recolectar datos (línea 93) con las funciones `SetGPIOAsWkUp` (Sección de código B.8) y `SetTimerAsWkUp` (Sección de código B.9).

Sección de código B.4: Tarea principal TimerGPIOTask que inicializa el sistema

```
void TimerGPIOTask(void *pvParameters)
{
 long lFileHandle;
 SlFsFileInfo_t FsFileInfo;
 unsigned char ucSyncMsg;
 int iRetVal = 0,Status;
 unsigned int primeravez=1;
 iRetVal = osi_MsgQCreate(&g_tConnection, NULL, sizeof( unsigned char ), 3);
 if (iRetVal < 0){
  UART_PRINT("unable to create the msg queue\n\r");
  LOOP_FOREVER();
 }
 iRetVal = sl_Start(NULL, NULL, NULL);
 if (iRetVal < 0){
  UART_PRINT("Failed to start the device \n\r");
  LOOP_FOREVER();
 }
 Status = sl_FsGetInfo("www/jason/dia1.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=1;
 }
 Status = sl_FsGetInfo("www/jason/dia2.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=2;
 }
 Status = sl_FsGetInfo("www/jason/dia3.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=3;
 }
 Status = sl_FsGetInfo("www/jason/dia4.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=4;
 }
 Status = sl_FsGetInfo("www/jason/dia5.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=5;
 }
 Status = sl_FsGetInfo("www/jason/dia6.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=6;
 }
 Status = sl_FsGetInfo("www/jason/dia7.csv",0,&FsFileInfo);
 if(Status==0){
  BackupI=7;
 }
 itoa(BackupI,BackupIA);
 Status = sl_FsGetInfo("www/jason/historial.csv",0,&FsFileInfo);
 if(Status==0){
  FILESTATUS=1;
 }
 sl_FsOpen((unsigned char *)"bcp/med.txt",FS_MODE_OPEN_READ,NULL, &lFileHandle);
 sl_FsRead( lFileHandle, 0, (unsigned char *)AVal, sizeof(AVal));
 sl_FsClose(lFileHandle, NULL, NULL , 0);
 while(FOREVER){
  sl_NetAppStop(SL_NET_APP_HTTP_SERVER_ID);
  SwitchToAPMode(ROLE_AP);
  sl_NetAppStart(SL_NET_APP_HTTP_SERVER_ID);
  osi_MsgQRead(&g_tHTTPD, &ucSyncMsg, OSI_WAIT_FOREVER);
  UART_PRINT("data received");
  if(sc==1){
   memset(AVal,0,sizeof(AVal));
   sc=0;
  }
  CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
  CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED);
```

```
66    CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);
67    sl_NetAppStop(SL_NET_APP_HTTP_SERVER_ID);
68    GuardarParam();
69    SwitchToStaMode(iRetVal);
70    ROLE=0;
71    sl_NetAppStart(SL_NET_APP_HTTP_SERVER_ID);
72    // Set the power management policy of NWP
73    iRetVal = sl_WlanPolicySet(SL_POLICY_PM, SL_NORMAL_POLICY, NULL, 0);
74    if (iRetVal < 0){
75     UART_PRINT("unable to configure network power policy\n\r");
76     LOOP_FOREVER();
77    }
78    _u8 *my_device = "IoT-Sensortag";
79    sl_NetAppSet (SL_NET_APP_DEVICE_CONFIG_ID, NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN,
          strlen(my_device), (_u8 *) my_device);
80    // connecting to the Access Point
81    g_ucWdogCount = 0;
82    tareas|=TASK_WIFI;
83    if(-1 == WlanConnect()){
84     UART_PRINT("Connection to AP failed\n\r");
85     osi_MsgQWrite(&g_tAPGPIO, &ucSyncMsg,OSI_NO_WAIT);
86    }
87    else{
88     UART_PRINT("Connected to AP\n\r");
89     lp3p0_setup_power_policy(POWER_POLICY_STANDBY);
90    }
91    if(primeravez==1){
92     gGPIOHndl = SetGPIOAsWkUp();
93     gTimerHndl = SetTimerAsWkUp();
94     primeravez=0;
95    }
96    tareas&=~TASK_WIFI;
97    osi_MsgQRead(&g_tAPGPIO, &ucSyncMsg, OSI_WAIT_FOREVER);
98   }
99  }
```

La función `SwitchToAPMode` de Sección de código B.5 es llamada cada que se requiere cambiar el modo de funcionamiento Wi-Fi, esta llama a la función propia del microcontrolador que se encarga de cambiar el modo de operación (línea 6), se indica en una variable que ahora el sistema está en el rol 1 correspondiente a punto de acceso y después se configura el punto de acceso con el nombre y contraseña que se requiere (líneas 8-10), se detienen los procesos relacionados con Wi-Fi y después se reactivan (líneas 11 y 12 respectivamente), se espera hasta que se obtenga IP (192.168.1.1) y después se configura el funcionamiento de la red para terminar esperando la conexión de un cliente (línea 26).

Sección de código B.5: Función `SwitchToAPMode` utilizada para crear punto de acceso.

```
1  void SwitchToAPMode(int iMode){
2   unsigned char ucDHCP;
3   char  secType = SL_SEC_TYPE_WPA;
4   long lRetVal = -1;
5   if(iMode == ROLE_AP){
6    sl_WlanSetMode(ROLE_AP);
7    ROLE=1;
8    sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SSID, strlen("SensorFI"),(unsigned char
         *)"SensorFI");
9    sl_WlanSet(SL_WLAN_CFG_AP_ID,WLAN_AP_OPT_SECURITY_TYPE, 1,(unsigned char *)&
         secType);
10   sl_WlanSet(SL_WLAN_CFG_AP_ID,WLAN_AP_OPT_PASSWORD,strlen((const char *)"sensores")
         ,(unsigned char *)"sensores");
11   sl_Stop(10);
```

```
12    sl_Start(0,0,0);
13    while(!IS_IP_ACQUIRED(g_ulStatus)){
14     //looping till ip is acquired
15    }
16    unsigned char len = sizeof(SlNetCfgIpV4Args_t);
17    SlNetCfgIpV4Args_t ipV4 = {0};
18    // get network configuration
19    lRetVal = sl_NetCfgGet(SL_IPV4_AP_P2P_GO_GET_INFO,&ucDHCP,&len,(unsigned char *)&
          ipV4);
20    if (lRetVal < 0)
21    {
22     UART_PRINT("Failed to get network configuration \n\r");
23     LOOP_FOREVER();
24    }
25    UART_PRINT("Connect a client to Device\n\r");
26    while(!IS_IP_LEASED(g_ulStatus)){
27     //wating for the client to connect
28    }
29    UART_PRINT("Client is connected to Device\n\r");
30   }
31  }
```

La función `SwitchToStaMode` de la Sección de código B.6 es llamada cada que se requiere cambiar el modo de funcionamiento Wi-Fi. Se utiliza la función `sl_WlanSetMode` para cambiar el modo de operación, después se detienen y activan los servicios internos para la conexión con `sl_Stop` y `sl_Start`.

Sección de código B.6: Función `SwitchToStaMode` utilizada para cambiar el modo de operación Wi-Fi a estación.

```
1  void SwitchToStaMode(int iMode){
2   sl_WlanSetMode(ROLE_STA);
3   MAP_UtilsDelay(80000);
4   sl_Stop(10);
5   MAP_UtilsDelay(80000);
6   sl_Start(0,0,0);
7  }
```

Cuando el sistema-IoT se encuentra en modo estación, la función `WlanConnect` de la Sección de código B.7 es llamada para intentar una conexión a la red que ha sido proporcionada por el usuario desde HTML. Se comienza iniciando las variables a utilizar (líneas 2-9), para después, iniciar el Watchdog directamente con la función `WDT_IF_Init`. A partir de la línea 14, se entra en el proceso de conexión, se indica mediante UART que se va a intentar conectar al punto de acceso (líneas 15-17) para con la función `sl_WlanConnect` realizar la conexión. El proceso siguiente se queda a la espera de la activación de un mensaje (línea 22), utilizando las interrupciones del Watchdog como limite de tiempo para la conexión, cuando ocurre un evento relacionado con conexión Wi-Fi, las tareas mandan el indicador correspondiente a través del mensaje, se esperan 5 posibles resultados:

- `EVENT_CONNECTION` Si la conexión se logra, se indicia en la variable `iConnect` que será utilizado por los otros casos.
- `EVENT_IP_ACQUIRED` Cuando se entra dentro de este caso, la conexión ya se realizó y el punto de acceso a otorgado una dirección IP, por lo que el valor de retorno de la función debe ser 0 para indicar que el proceso resultó correcto.
- `WDOG_EXPIRED` Si se supera el tiempo de espera y hay una conexión presente se procede a desconectar ya que no se ha obtenido IP y a reiniciar el proceso de conexión.

- `EVENT_DISCONNECTION` Cuando se desconecta del punto de acceso es necesario indicarlo dentro de la variable local `iConnect`
- `CONNECTION_FAILED` Si los intentos de conexión se han cumplido y no se logra establecer conexión, el Watchdog envía este indicador, por lo que el valor de retorno de la función se vuelve -1.

Sección de código B.7: Función `WlanConnect` que controla el proceso de intento de conexión a un punto de acceso.

```
int WlanConnect(){
 int iRetVal = -1;
 int iConnect = 0;
 tBoolean iRetCode=0;
 unsigned char ucQueueMsg = 0;
 SlSecParams_t secParams;
 secParams.Key = (signed char *)SSIDKEYA;
 secParams.KeyLen = strlen(SSIDKEYA);
 secParams.Type = KEYTYPEA;
 WDT_IF_Init(WatchdogIntHandler, MILLISECONDS_TO_TICKS(WD_PERIOD_MS));
 MAP_PRCMPeripheralClkEnable(PRCM_WDT, PRCM_SLP_MODE_CLK);
 g_ucFeedWatchdog = 1;
 g_ucWdogCount = 0;
 while(!(ucQueueMsg & (EVENT_IP_ACQUIRED|CONNECTION_FAILED))){
  UART_PRINT("Trying to connect to AP: ");
  UART_PRINT((const char *)SSIDA);
  UART_PRINT("\n\r");
  sl_WlanConnect((signed char *)SSIDA,
  strlen((const char *)SSIDA), 0, &secParams, 0);
  iConnect = 0;
  do{
   osi_MsgQRead(&g_tConnection, &ucQueueMsg, OSI_WAIT_FOREVER);
   switch(ucQueueMsg){
    case EVENT_CONNECTION:
    iConnect = 1;
    break;
    case EVENT_IP_ACQUIRED:
    iRetVal = 0;
    break;
    case WDOG_EXPIRED:
    // disconnect from the Access Point
    if(iConnect){
     WlanDisconnect();
    }
    // stop the simplelink with reqd. timeout value (30 ms)
    sl_Stop(SL_STOP_TIMEOUT);
    UART_PRINT("sl stop\n\r");
    MAP_UtilsDelay(8000);
    // starting the simplelink
    sl_Start(NULL, NULL, NULL);
    UART_PRINT("sl start\n\r");
    break;
    case EVENT_DISCONNECTION:
    iConnect = 0;
    break;
    case CONNECTION_FAILED:
    iRetVal = -1;
    break;
    default:
    UART_PRINT("unexpected event\n\r");
    break;
   }
```

```
53    }while(ucQueueMsg == (unsigned char)EVENT_CONNECTION);
54   }
55   return(iRetVal);
56  }
```

En caso de ser necesario, se agrega la posibilidad de salir del modo de bajo consumo por interrupción generada al presionar un botón del Launchpad, para esto, se utiliza la función `SetGPIOAsWkUp` (Sección de código B.8). Esta función llama a la función interna `cc_gpio_open` que lo declara como entrada y fuente para salir del bajo consumo, después, habilita sus interrupciones utilizando `cc_gpio_enable_notification`.

Sección de código B.8: Función `SetGPIOAsWkUp` que declara interrupción de GPIO como fuente para salir del modo bajo consumo.

```
1  cc_hndl SetGPIOAsWkUp(){
2   cc_hndl tGPIOHndl;
3   // setting up GPIO as a wk up source and configuring other related parameters
4   tGPIOHndl = cc_gpio_open(GPIO_SRC_WKUP, GPIO_DIR_INPUT);
5   cc_gpio_enable_notification(tGPIOHndl, GPIO_SRC_WKUP, INT_FALLING_EDGE,
6   (GPIO_TYPE_NORMAL | GPIO_TYPE_WAKE_SOURCE));
7   return(tGPIOHndl);
8  }
```

Para la medición constante es necesario un contador que permanezca activo aún al entrar en modo de bajo consumo, por lo cual se utiliza el reloj de tiempo real (Real Time Clock, RTC por sus siglas en inglés) de 32 kHz. Con la función `SetTimerAsWkUp` (Sección de código B.9) se le declara como fuente para salir del bajo consumo. Dicha función, iniciliza el timer utilizando la función `cc_timer_start` enviando los parametros como la variable global `LPDS_DUR_SEC` que recibe los segundos del periodo de recolección enviado por el usuario, la función que se encarga de atender sus interrupciones y el conteo inicial.

Sección de código B.9: Función `SetTimerAsWkUp` que declara interrupción del Timer como fuente para salir del bajo consumo.

```
1  cc_hndl SetTimerAsWkUp(){
2   cc_hndl tTimerHndl;
3   struct cc_timer_cfg sRealTimeTimer;
4   struct u64_time sInitTime, sIntervalTimer;
5   // setting up Timer as a wk up source and other timer configurations
6   sInitTime.secs = 0;
7   sInitTime.nsec = 0;
8   cc_rtc_set(&sInitTime);
9   sRealTimeTimer.source = HW_REALTIME_CLK;
10  sRealTimeTimer.timeout_cb = TimerCallback;
11  sRealTimeTimer.cb_param = NULL;
12  tTimerHndl = cc_timer_create(&sRealTimeTimer);
13  sIntervalTimer.secs = LPDS_DUR_SEC;
14  sIntervalTimer.nsec = LPDS_DUR_NSEC;
15  cc_timer_start(tTimerHndl, &sIntervalTimer, OPT_TIMER_PERIODIC);
16  return(tTimerHndl);
17 }
```

La tarea `sl_WlanEvtHdlr` mostrada en Sección de código B.10 se encarga de responder ante los 4 eventos que puede generar la interacción Wi-Fi. En caso de que suceda una conexión (línea 4), se

desbloquea un mensaje para indicar que se realizó correctamente (línea 7). Si sucede un evento de desconexión, se limpian los bits de estado de conexión e IP adquirida, se avisa mediante el mensaje de conexión que ha ocurrido la desconexión y se prepara al sistema para volver a intentar la conexión desbloqueando el mensaje correspondiente a la tarea Sección de código B.4. Si el dispositivo se encuentra funcionando en modo punto de acceso y un cliente se conecta se indica en los bits respectivos (líneas 31-35), mientras que si un cliente se desconecta se limpian los bits correspondientes (líneas 39-40).

Sección de código B.10: Tarea `sl_WlanEvtHdlr` que atiende los eventos de conexión Wi-Fi.

```
1  void sl_WlanEvtHdlr(SlWlanEvent_t *pSlWlanEvent){
2   unsigned char ucQueueMsg = 0;
3   switch(pSlWlanEvent->Event){
4    case SL_WLAN_CONNECT_EVENT:
5     ucQueueMsg = (unsigned char)EVENT_CONNECTION;
6     if(g_tConnection != 0){
7     osi_MsgQWrite(&g_tConnection, &ucQueueMsg, OSI_WAIT_FOREVER);
8     }
9     else{
10     UART_PRINT("Error: sl_WlanEvtHdlr: Queue does not exist\n\r");
11     while(FOREVER);
12    }
13    UART_PRINT("C\n\r");
14   break;
15   case SL_WLAN_DISCONNECT_EVENT:
16   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
17   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);
18   if (ROLE==0){
19    ucQueueMsg = (unsigned char)EVENT_DISCONNECTION;
20    if(g_tConnection != 0){
21    osi_MsgQWrite(&g_tConnection, &ucQueueMsg, OSI_WAIT_FOREVER);
22    }
23    else{
24     UART_PRINT("Error: sl_WlanEvtHdlr: Queue does not exist\n\r");
25     while(FOREVER);
26    }
27    UART_PRINT("Disconnected switching to AP\n\r");
28    osi_MsgQWrite(&g_tAPGPIO, &ucQueueMsg,OSI_NO_WAIT);
29   }
30   break;
31   case SL_WLAN_STA_CONNECTED_EVENT:{
32   // when device is in AP mode and any client connects to device cc3xxx
33   SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
34   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION_FAILED);
35  }
36  break;
37  case SL_WLAN_STA_DISCONNECTED_EVENT:{
38   // when client disconnects from device (AP)
39   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
40   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED);
41  }
42  break;
43  default:
44  break;
45  }
46 }
```

Cuando se establece una conexión con un punto de acceso o un cliente se conecta, se generan eventos relacionados a IP que son atendidos por la tarea `sl_NetAppEvtHdlr` mostrada en Sección de código B.11. Cuando el punto de acceso le otorga una dirección IP al sistema (línea 7) se modifican los bits correspondientes y se procede a mandar dentro de un mensaje la notificación (línea 14). Con base en si la IP se otorga o se libera, se cambian los bits correspondientes como se ve en la línea 26 y 30 respectivamente.

Sección de código B.11: Tarea `sl_NetAppEvtHdlr` que atiende los eventos relacionados al manejo de IP

```
1  void sl_NetAppEvtHdlr(SlNetAppEvent_t *pNetAppEvent){
2   unsigned char ucQueueMsg = 0;
3   if(!pNetAppEvent){
4    return;
5   }
6   switch(pNetAppEvent->Event){
7    case SL_NETAPP_IPV4_IPACQUIRED_EVENT:
8     UART_PRINT("IP ADQUIRIDA");
9     SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);
10    g_ulIpAddr = pNetAppEvent->EventData.ipAcquiredV4.ip;
11    ucQueueMsg = (unsigned char)EVENT_IP_ACQUIRED;
12    if(ROLE==0){
13     if(g_tConnection != 0){
14      osi_MsgQWrite(&g_tConnection, &ucQueueMsg, OSI_WAIT_FOREVER);
15     }
16    else{
17     UART_PRINT("Error: sl_NetAppEvtHdlr: Queue does not exist\n\r");
18     while(FOREVER);
19    }
20   }
21   UART_PRINT("IP: ");
22   PrintIPAddr(g_ulIpAddr);
23   UART_PRINT("\n\r");
24   break;
25   case SL_NETAPP_IP_LEASED_EVENT:{
26    SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED);
27   }
28   break;
29   case SL_NETAPP_IP_RELEASED_EVENT:{
30   CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED);
31   }
32   break;
33   default:
34   break;
35  }
36 }
```

La tarea `Recolectar` mostrada en Sección de código B.12 se encarga de llamar a los subprocesos necesarios para realizar una medición, utilizando el arreglo flotante local de la línea 2 como contenedor. Esta tarea es cíclica con ejecución controlada por el mensaje de la línea 5, al inicio y fin de una medición se activa y desactiva la bandera propia de la tarea (líneas 6 y 14).

El proceso de recolección comienza activando el bus I^2C en modo maestro para después llamar a la función `recolecta` de la Sección de código B.13 que interactúa con los sensores (línea 8), al regresar de la función, se desactiva el bus I^2C y su contador de errores, después dentro de la variable global de medición más reciente `ValMR` se guardan los valores presentes en el arreglo flotante `acum` con formato de cadena y cada valor separado con ",", a estos valores se les anexan el día, hora y minuto en el que fueron realizadas (línea 11). Con las variables presentes en el arreglo `ValMR` se llama a la función `acumular` (línea 13) presente en la Sección de código B.14 que acumula la medición dentro del contenedor global de mediciones.

Sección de código B.12: Tarea `Recolectar` que controla el proceso de medición medición.

```
1  void Recolectar(void *pvParameters){
2   float acum[10];
3   unsigned char ucSyncMsg;
4   while(FOREVER){
5    osi_MsgQRead(&g_tRecolectar, &ucSyncMsg, OSI_WAIT_FOREVER);
6    tareas|=TASK_RECOLECT;
7    I2C_IF_Open(I2C_MASTER_MODE_STD);
8    recolecta(acum);
9    I2C_IF_Close();
10   g_ucI2CCOUNT=0;
11   sprintf(ValMR,"%09.2f,%05.2f,%05.2f,%05.2f,%05.2f,%06.2f,%d,%d,%d\n",acum[0],acum
         [1],acum[2],acum[3],acum[4],acum[5],dateTime.sl_tm_day,dateTime.sl_tm_hour,
         dateTime.sl_tm_min);
12   UART_PRINT(ValMR);
13   acumular(ValMR);
14   tareas&=~(TASK_RECOLECT);
15  }
16 }
```

La función `recolecta` mostrada en la Sección de código B.13 es llamada cada que se requiere recolectar las mediciones de los sensores, cuando se hace llamado a esta función ya está activado el bus de comunicación y se manda la dirección del arreglo flotante para guardar los valores resultantes.

Utilizando un buffer se envía la configuración `0xCA10` a la dirección `0x01` mediante la función de escritura de la línea 8. Este proceso se repite para cada sensor, la primera posición del arreglo corresponde al registro al que se busca acceder o escribir, lo que se coloca después corresponde al dato que se quiere escribir. De la línea 10 a la línea 63 se tiene la recolección de la variable de luz del sensor OPT3001 con dirección `0x44`. Se hace una escritura sin valor a la dirección `0x00` que ocasiona el inicio de conversión (línea 12), 900 ms después, se llama a la función de lectura esperando extraer 2 B, en caso de que no haya ningún error, se procede a realizar el tratado del resultado (lineas 16-59) mientras que si hubo error en la lectura, se asigna directamente 0.0 al resultado y se levanta la bandera de error correspondiente a esa variable (línea 61). El valor resultante es asignado a la primera posición del puntero.

Las líneas 64-92 corresponden a las lecturas de humedad y temperatura del sensor HDC2080, para comenzar la conversión se escribe un bit en la dirección `0x0F` del sensor, y 20 ms después se apunta a la dirección `0x00` (línea 71) esperando leer 4 B, para después hacer las debidas conversiones, terminando por asignar la temperatura en la segunda posición del puntero y la humedad en la tercera.

Las líneas 93-139 corresponden a las lecturas de corriente, voltaje y potencia del sensor INA260. Se comienza escribiendo la configuración `0x61FB` en el registro `0x00` del sensor (línea 97), después se apunta a la dirección `0x1` para disparar una medición, se esperan 20 ms antes de hacer la petición de lectura esperando 2 B (línea 101) como respuesta, si no se presenta ningún problema se asigna el valor de corriente en la cuarta posición del arreglo. El proceso se repite para el caso del voltaje y potencia que corresponden a los registros `0x2` y `0x3` del sensor, estos resultados son asignados a la quinta y sexta posición del arreglo, respectivamente.

Sección de código B.13: Función `recolecta` de interacción con sensores.

```
1  void recolecta(float *puntero){
2   unsigned int variableent=0,exponente=0;
3   int error=0;
4   float variablefloat=0.0;
5   aucDataBufS[1]=0xCA;
6   aucDataBufS[2]=0x10;
7   aucDataBufS[0]=0x01;
8   I2C_IF_Write(ADDRL,aucDataBufS,3,1);
9   I2Cerror=0;
10  //OPT3001 direccion 0x44
11  aucDataBufS[0]=0x00;
12  I2C_IF_Write(ADDRL,aucDataBufS,1,1);
13  osi_Sleep(900);
14  error=I2C_IF_Read(ADDRL,aucDataBufR,2);
15  if(error>=0){
16   variableent=(aucDataBufR[0]<<8)|(aucDataBufR[1]);
17   exponente=(variableent>>12);
18   variableent=variableent&(0xFFF);
19   switch(exponente){
20    case (0x0):
21     variablefloat=((float)variableent)/100;
22    break;
23    case (0x1):
24     variablefloat=((float)variableent)*2/100;
25    break;
26    case (0x2):
27     variablefloat=(float)variableent*4/100;
28    break;
29    case (0x3):
30     variablefloat=(float)variableent*8/100;
31    break;
32    case (0x4):
33     variablefloat=(float)variableent*16/100;
34    break;
35    case (0x5):
36     variablefloat=(float)variableent*32/100;
37    break;
38    case (0x6):
39     variablefloat=(float)variableent*64/100;
40    break;
41    case (0x7):
42     variablefloat=(float)variableent*128/100;
43    break;
44    case (0x8):
45     variablefloat=(float)variableent*256/100;
46    break;
47    case (0x9):
48     variablefloat=(float)variableent*512/100;
49    break;
50    case (0xA):
51     variablefloat=(float)variableent*1024/100;
```

```
    break;
    case (0xB):
     variablefloat=(float)variableent*2048/100;
    break;
    default:
    break;
   }
  }else{
   variablefloat=0.0;
   I2Cerror|=0x1;
  }
  *puntero=variablefloat;
  //Iniciar conversion
  aucDataBufS[0]=0x0F;
  aucDataBufS[1]=0x1;
  error=I2C_IF_Write(ADDRTH,aucDataBufS,2,1);
  osi_Sleep(20);
  //HDC2080 direccion 0x42
  aucDataBufS[0]=0x00;
  I2C_IF_Write(ADDRTH,aucDataBufS,1,0);
  error=I2C_IF_Read(ADDRTH,&aucDataBufR[0],4);
  if(error>=0){
   //--Temperatura 0x0 - Direccion
   variableent=(aucDataBufR[1]<<8)|(aucDataBufR[0]);
   variablefloat=(float)((variableent*165)/65536)-40.0;
   if ((variablefloat>100.0)||(variablefloat<0.0)){
    variablefloat=0.0;
   }
   *(puntero+1)=variablefloat;
   //--Humedad 0x1 - Direccion
   variableent=(aucDataBufR[3]<<8)|(aucDataBufR[2]);
   variablefloat=(float)((variableent*100)/65535);
   if((variablefloat>100.0)||(variablefloat<0.0)){
    variablefloat=0.0;
   }
   *(puntero+2)=variablefloat;
  }else{
   *(puntero+1)=0.0;
   *(puntero+2)=0.0;
   I2Cerror|=0x2;
  }
  //--Corriente dir 0x01
  aucDataBufS[2]=0xFB;
  aucDataBufS[1]=0x61;
  aucDataBufS[0]=0x00;
  I2C_IF_Write(ADDRIVP,aucDataBufS,3,1);
  aucDataBufS[0]=0x01;
  I2C_IF_Write(ADDRIVP,aucDataBufS,1,1);
  osi_Sleep(20);
  error=I2C_IF_Read(ADDRIVP,aucDataBufR,2);
  if(error>=0){
   variableent=(aucDataBufR[0]<<8)|(aucDataBufR[1]);
   if(aucDataBufR[0]&(0x80)){
    variablefloat=((float)(65535-variableent)/800);
   }else{
    variablefloat=(float)variableent/800;
   }
  }
  else{
   I2Cerror|=0x4;
   variablefloat=0.0;
  }
  *(puntero+3)=variablefloat;
  //--Voltaje 0x40 dir 0x02
  aucDataBufS[0]=0x02;
```

```
117  I2C_IF_Write(ADDRIVP,aucDataBufS,1,1);
118  error=I2C_IF_Read(ADDRIVP,aucDataBufR,2);
119  if(error>=0){
120   variableent=(aucDataBufR[0]<<8)|(aucDataBufR[1]);
121   variablefloat=(float)variableent/800;
122  }else{
123   I2Cerror|=0x8;
124   variablefloat=0.0;
125  }
126  *(puntero+4)=variablefloat;
127  //--Potencia 0x40 dir 0x03
128  aucDataBufS[0]=0x03;
129  I2C_IF_Write(ADDRIVP,aucDataBufS,1,1);
130  error=I2C_IF_Read(ADDRIVP,aucDataBufR,2);
131  if(error>=0){
132   variableent=(aucDataBufR[0]<<8)|(aucDataBufR[1]);
133   variablefloat=(float)variableent/100;
134  }else{
135   I2Cerror|=0x10;
136   variablefloat=0.0;
137  }
138  *(puntero+5)=variablefloat;
139 }
```

Cada que se hace una medición, la tarea encargada del proceso llama a la función `acumular` mostrada en la Sección de código B.14, esta trabaja con la variable contenedora de las mediciones acumuladas `AVal` y es llamada con la variable contenedora de la medición más reciente `ValMR`. La función comienza por analizar las mediciones presentes en el acumulador global de mediciones, escaneando en busca de saltos de linea que indican fin de medición o terminadores de cadena para saber que ya no hay más mediciones (líneas 5-18), estos indices son utilizados para hacer el debido corrimiento del acumulador global de mediciones y poder anexar las nuevas (líneas 19-27), se utiliza el indice `respaldomediciones` para llevar cuenta del numero de mediciones y en caso de que este sea mayor a 288, indicar que es necesario realizar un respaldo y reiniciar los indices (líneas 28-39). Se utiliza también la variable `BackupI` para tener control sobre el nombre del futuro archivo de respaldo por día, este es convertido a arreglo con la línea 37 y es utilizado por la tarea `FileBackup` presente en la Sección de código B.15.

Sección de código B.14: Función `acumular` que acumula las mediciones

```
1  void acumular(char * variable) {
2   unsigned int mediciones = 0;
3   unsigned int indN = 0, i = 0, indA = 0;
4   unsigned char ucQueueMsg = 0;
5   while ( * (variable + indN) != '\0') {
6    indN++;
7   }
8   indN = indN - 1;
9   while (AVal[indA] != '\0') {
10   if (AVal[indA] == '\n') {
11    mediciones = mediciones + 1;
12    if (mediciones == (Totalmediciones - 1)) {
13     break;
14    }
15   }
16   indA++;
17  }
18  respaldomediciones = mediciones + 1;
19  if (indA > 0) {
20   for (i = indA; i > 0; i--) {
21    AVal[i + indN] = AVal[i];
```

```
22    }
23    AVal[indN + indA + 1] = '\0';
24   }
25   for (i = 0; i <= indN; i++) {
26    AVal[i] = * variable++;
27   }
28   memset(contadormed, 0, sizeof(contadormed));
29   itoa(respaldomediciones, contadormed);
30   if (respaldomediciones >= 288) {
31    BackupI++;
32   if (BackupI > CANTIDAD_RESPALDOS) {
33    BorrarBackups();
34    BackupI = 1;
35   }
36   respaldomediciones = 0;
37   itoa(BackupI, BackupIA);
38   osi_MsgQWrite( & g_tFILEBackup, & ucQueueMsg, OSI_NO_WAIT);
39   }
40  }
```

La Sección de código B.15 muestra la tarea `FileBackup` asignada para la generación de archivos cada 288 mediciones. La tarea es de ejecución cíclica controlada por el mensaje de la línea 6 y tiene su bandera de identificación de tarea. Las líneas 8-11 muestran el nombre que llevará el archivo, utilizando el índice global `BackupIA` que indica el numero del 1-7 que le corresponde al archivo que se creará, este nombre es utilizado para con la línea 12 crear el archivo de máximo 16 kB. Al igual que con la escritura de archivos a petición, las líneas 23-31 corresponden a la escritura del archivo, primero las cabeceras (línea 23) para después escribir todo el arreglo contenedor (línea 27). Finalmente, con las mediciones de `AVal` presentes en el archivo creado se borra completamente el contenedor (línea 32) y se baja su bandera de tarea.

Sección de código B.15: Tarea `FileBackup` que genera archivos de respaldo por día.

```
1  void FileBackup(void *pvParameters){
2   long lFileHandle,lRetVal;
3   unsigned char  ucQueueMsg,nombre[50]="";
4   unsigned long ulToken;
5   while(1){
6    osi_MsgQRead(&g_tFILEBackup, &ucQueueMsg,OSI_WAIT_FOREVER);
7    tareas|=TASK_FILEBCKP;
8    strcpy(nombre,"www/jason/");
9    strcat(nombre,dia);
10   strcat(nombre,BackupIA);
11   strcat(nombre,".csv");
12   lRetVal = sl_FsOpen((unsigned char *)nombre,FS_MODE_OPEN_CREATE(16384,
         _FS_FILE_OPEN_FLAG_COMMIT|_FS_FILE_PUBLIC_WRITE|_FS_FILE_PUBLIC_READ),&ulToken
         ,&lFileHandle);
13   if(lRetVal < 0)
14   {// File may already be created
15    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
16   }else{// close the user file
17    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
18   }//  open a user file for writing
19   lRetVal = sl_FsOpen((unsigned char *)nombre,FS_MODE_OPEN_WRITE,&ulToken,&
         lFileHandle);
20   if(lRetVal < 0){
21    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
22   }// write to the file
23   lRetVal = sl_FsWrite(lFileHandle,0,&ValPrim[0], sizeof(ValPrim));
24   if (lRetVal < 0){
25    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
26   }// write to the file
27   lRetVal = sl_FsWrite(lFileHandle,strlen(ValPrim),(unsigned char *)AVal, sizeof(
```

```
      AVal));
28   if (lRetVal < 0){
29    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
30   }// close the user file
31   lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
32   memset(AVal,0,sizeof(AVal));
33   tareas&=~TASK_FILEBCKP;
34  }
35 }
```

La tarea `FileW` de la Sección de código B.16 es llamada cada que el usuario mediante la página HTML hace una petición para generar un archivo. Esta es una tarea cíclica con ejecución controlada por el mensaje de la línea 7, la tarea tambien cuenta con bit de registro para identificar su ejecución (línea 8). El proceso de creación es primero crear un archivo con ayuda de la función de la línea 9, con el nombre `www/jason/historial.csv` y tamaño máximo de 16 kB, con base en el resultado de la operación anterior se cierra el archivo (línea 14) pero si ocurre un error se llega a la línea 16 para indicarlo. Con el archivo creado, ahora se procede a abrir el archivo (línea 19) para con ayuda de las funciones de escritura (línea 20-32) se haga la escritura primero del arreglo que contiene los nombres de las columnas `ValPrim` (línea 24) para añadir todo el arreglo de mediciones acumuladas `AVal`. Finalmente, se cierra el archivo y se indica en la variable de estado del archivo (línea 33) que ya está creado.

Sección de código B.16: Tarea `FileW` encargada de la creación de archivos a petición.

```
1  void FileW(void *pvParameters){
2   long lFileHandle;
3   unsigned long ulToken;
4   unsigned char ucQueueMsg;
5   long lRetVal;
6   while(1){
7    osi_MsgQRead(&g_tFILEW, &ucQueueMsg,OSI_WAIT_FOREVER);
8    tareas|=TASK_FILEW;
9    lRetVal = sl_FsOpen((unsigned char *)USER_FILE_NAME,FS_MODE_OPEN_CREATE(16384,
         _FS_FILE_OPEN_FLAG_COMMIT|_FS_FILE_PUBLIC_WRITE|_FS_FILE_PUBLIC_READ),&ulToken
         ,&lFileHandle);
10   if(lRetVal < 0){
11    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
12   }
13   else{
14    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
15    if (SL_RET_CODE_OK != lRetVal){
16     ASSERT_ON_ERROR(FILE_CLOSE_ERROR);
17    }
18   }
19   lRetVal = sl_FsOpen((unsigned char *)USER_FILE_NAME,
20   FS_MODE_OPEN_WRITE,&ulToken,&lFileHandle);
21   if(lRetVal < 0){
22    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
23   }
24   lRetVal = sl_FsWrite(lFileHandle,0,&ValPrim[0], sizeof(ValPrim));
25   if (lRetVal < 0){
26    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
27   }
28   lRetVal = sl_FsWrite(lFileHandle,strlen(ValPrim),(unsigned char *)AVal, sizeof(
         AVal));
29   if (lRetVal < 0){
30    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
31   }
32   lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
```

```
33    FILESTATUS=1;
34    tareas&=~TASK_FILEW;
35   }
36  }
```

La función `WatchdogIntHandler` mostrada en la Sección de código B.17 es llamada cada que transcurre el tiempo de conteo asignado dentro del reloj del Watchdog. Al entrar a la función, primero se verifica que no se haya pedido dejar de bajar su bandera (línea 4) ya que si es así se retorna directamente de la función (línea 5) lo que provocaría que la siguiente ocasión en que se genere la interrupción, el sistema se reinicie. En caso contrarío, si no se ha indicado dejar de bajar la bandera se procede a revisar la variable global de tareas activas para verificar qué tarea ha ocasionado una espera de 10 segundos.

Para los casos en que la tarea de recolección genere espera mayor a 10 segundos (líneas 6-19), primero se aumenta el contador de intentos de comunicación por el bus I^2C, para después eliminar y volver a crear la tarea que se haya quedado congelada y activar su mensaje cíclico para que la ejecución sea inmediata (líneas 8-11), si sucede que ya se ha intentado hacer este mismo proceso varías veces y siguen existiendo tiempos de 10 segundos entonces se activa el mensaje de respaldo, se elimina la tarea de recolección y se indica a una variable global el código de error correspondiente a error en el bus de comunicación (líneas 13-15). Finalmente se limpia la bandera de la tarea y la bandera del Watchdog.

Dado que los problemas con la generación de archivos indican error interno en el sistema de archivos si las tareas activas coinciden con la escritura de archivos a petición (línea 20) o la generación automática de respaldos (línea 24), se indica que se deje de bajar la bandera del Watchdog para generar un reinicio.

Se utiliza el Watchdog como temporizador para la conexión Wi-Fi, por lo que si coincide con este proceso (línea 28), se verifica la cantidad de veces que se ha intentado conectar ya que si son más de las establecidas se indica que la conexión falló (línea 37) y si aún hay intentos se indica que la espera ya superó los 10 s. Si la conexión ya fue establecida se activa el mensaje para indicarlo y en cualquiera de los casos se limpia la bandera del Watchdog.

Sección de código B.17: Función `WatchdogIntHandler` que atiende de interrupciones del Watchdog.

```
1  void WatchdogIntHandler(void)
2  {
3   unsigned char ucQueueMsg = 0;
4   if(g_ucFeedWatchdog==0)
5   {return;}
6   if(tareas&TASK_RECOLECT){
7    if(g_ucI2CCOUNT < COMUNICATION_RETRIES){
8     g_ucI2CCOUNT++;
9     osi_TaskDelete(&TH_recolectar);
10    osi_TaskCreate(Recolectar,(const signed char*)"Tarea recoleccion", OSI_STACK_SIZE
          , NULL, 3,&TH_recolectar );
11    osi_MsgQWrite(&g_tRecolectar, &ucQueueMsg,OSI_WAIT_FOREVER);
12   }else{
13    osi_MsgQWrite(&g_Respaldo, &ucQueueMsg, OSI_NO_WAIT);
14    osi_TaskDelete(&TH_recolectar);
15    I2Cerror|=0x20;
16    tareas&=~TASK_RECOLECT;
17   }
18   WatchdogIntClear(WDT_BASE);
19  }
20  if(tareas&TASK_FILEW){
21   MAP_WatchdogIntClear(WDT_BASE);
```

```
22    g_ucFeedWatchdog=0;
23   }
24   if(tareas&TASK_FILEBCKP){
25    MAP_WatchdogIntClear(WDT_BASE);
26    g_ucFeedWatchdog=0;
27   }
28   if(tareas&TASK_WIFI){
29    if(g_ucWdogCount < CONNECTION_RETRIES){
30     g_ucWdogCount++;
31     ucQueueMsg = (unsigned char)WDOG_EXPIRED;
32     MAP_WatchdogIntClear(WDT_BASE);
33    }
34    else{
35     ucQueueMsg = (unsigned char)CONNECTION_FAILED;
36     MAP_WatchdogIntClear(WDT_BASE);
37    }
38    if(g_tConnection != 0){
39     osi_MsgQWrite(&g_tConnection, &ucQueueMsg, OSI_NO_WAIT);
40     MAP_WatchdogIntClear(WDT_BASE);
41    }
42    else{
43     UART_PRINT("Error: WatchdogIntHandler: Queue does not exist\n\r");
44    }
45   }
46   if(tareas==0){
47    MAP_WatchdogIntClear(WDT_BASE);
48   }
49  }
```

A diferencia de las funciones y tareas previas, la función `vApplicationIdleHook` de la Sección de código B.18, no forma parte del código principal del sistema ni es de llamado o ejecución controlable, esta función es llamada automáticamente por el manejador de tareas del FreeRTOS cada que todas las tareas se encuentran en ciclos de espera o no hay ningún proceso pendiente. Esta particularidad de ejecución la hace indicada para manejar la entrada a modos de bajo consumo (línea 3) si se cumplen las condiciones: estar en modo punto de acceso, ninguna bandera de tarea activa y no se ha pedido dejar de alimentar al Watchdog.

Sección de código B.18: Función `vApplicationIdleHook` propia del FreeRTOS que controla el modo de bajo consumo

```
1  void vApplicationIdleHook( void ){
2   if((ROLE==0)&&(tareas==0)&&(g_ucFeedWatchdog==1)){ //condiciones para no entrar a
        LPDS si se dejo de alimentar el WDT,
3    cc_idle_task_pm();
4   }
5  }
```

Cada que el sistema entra en modo bajo consumo, la mayoría de los subprocesos tales como relojes, buses de comunicación y el Watchdog son desactivados, por lo que cuando alguna de las fuentes para despertar explicadas anteriormente ocurre, es necesario reactivar los procesos que no se retoman automáticamente. La Sección de código B.19 muestra el código de la tarea `Status` de comportamiento cíclico con ejecución controlada por un mensaje que se desbloquea cuando el sistema sale del modo de bajo consumo, cuya única función es volver a iniciar el Watchdog (línea 5).

Sección de código B.19: Tarea `Status` que inicia el watchdog.

```
1 void Status(void *pvParameters){
2  unsigned char ucSyncMsg;
3  while(FOREVER){
4   osi_MsgQRead(&g_wake, &ucSyncMsg, OSI_WAIT_FOREVER);
5   start_wdt();
6  }
7 }
```

La función `start_wdt` de la Sección de código B.20 se encarga de iniciar el Watchdog, la inicialización se hace indicando el numero de milisegundos que cuenta antes de generar una interrupción, indicado por la definición global `WD_PERIOD_MS` que para este caso vale 10000, y también se le añade el nombre de la función que va a atender las interrupciones generadas.

Sección de código B.20: Función `start_wdt` que inicializa el watchdog.

```
1 void start_wdt(void){
2  // Set up the watchdog interrupt handler.
3  WDT_IF_Init(WatchdogIntHandler, MILLISECONDS_TO_TICKS(WD_PERIOD_MS));
4 }
```

Las interrupciones individuales tanto del GPIO como del Timer, desbloquean mensajes que son esperados en la tarea `IntHdlr` presente en la Sección de código B.21. Esta tarea es de ejecución cíclica controlada por el mensaje de la línea 5 que se desbloquea cada que se sale del modo de bajo consumo, se esperan 3 posibles motivos:

- Timer (línea 7): Cuando el reloj de tiempo real asignado para el control del periodo de medición llega a las cuentas correspondientes, se genera una interrupción que desbloquea un mensaje (línea 10) dando inicio al proceso de recolección de variables.
- GPIO (línea 12): Cuando se presiona un botón del Launchpad, su interrupción desbloquea el mensaje que da inicio al cambio de modo Wi-Fi de estación a punto de acceso sin interrumpir otros procesos.
- HTTP (línea 18): Para toda petición hecha por un cliente al acceder a la dirección IP del sistema-IoT, el servidor HTTP embebido genera una interrupción que es atendida por la tarea `sl_HttpServerCallback` Sección de código B.22

Sección de código B.21: Tarea `IntHdlr` que ejecuta procesos con base en interrupciones

```
1 void IntHdlr(void *pvParameters){
2  unsigned char ucQueueMsg = 0;
3  // waits for the message from the various interrupt handlers(GPIO,Timer)
4  while(FOREVER){
5   osi_MsgQRead(&g_tWkupSignalQueue, &ucQueueMsg, OSI_WAIT_FOREVER);
6   switch(ucQueueMsg){
7    case 1:
8    UART_PRINT("timer\n\r");
9    sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION,&configOpt, &configLen,(_u8 *)(&
         dateTime));
10   osi_MsgQWrite(&g_tRecolectar, &ucQueueMsg,OSI_WAIT_FOREVER);
11   break;
```

```
12      case 2:
13      UART_PRINT("GPIO\n\r");
14      if(ROLE==0){
15       osi_MsgQWrite(&g_tAPGPIO, &ucQueueMsg,OSI_WAIT_FOREVER);
16      }
17      break;
18      case 3:
19      UART_PRINT("host irq\n\r");
20      break;
21      default:
22      UART_PRINT("invalid msg\n\r");
23      break;
24    }
25   }
26  }
```

La tarea `sl_HttpServerCallback` de la Sección de código B.22, forma parte de las tareas propias del microcontrolador que son llamadas de manera automática cuando se presenta su condición. Cada que un cliente interactúa con el servidor HTTP la función es llamada con diferentes parámetros de entrada, se tienen dos diferentes tipos de llamada:

- Petición de información GET: Este parámetro de entrada se genera cuando en la página HTML hay tokens definidos por el usuario que previo a ser mostrados al cliente necesitan actualizarse con el valor real. La respuesta que se otorga para este tipo de entrada depende del token que se está buscando y corresponde a las líneas 11-77. El código para cada token se repite, en primer lugar se analiza el token buscado con los tokens definidos, después se entrega una respuesta a `pSlHttpServerResponse` y se termina la cadena para indicar que el valor está listo (línea 75).
- Recepción de información POST: Si el cliente o la página hacen una petición para introducir un dato dentro del sistema, se escanea el token POST con base en los que se tienen definidos para saber el tratado que se le dará a la información recibida. Se tienen 3 diferentes POST definidos; la creación de archivos a petición (líneas 93-96), reinicio de sistema (líneas 97-100) y la extracción de datos enviados por el cliente (líneas 102-104)

Sección de código B.22: Tarea `sl_HttpServerCallback` que atiende los eventos GET y POST de HTTP.

```
1  void sl_HttpServerCallback(SlHttpServerEvent_t *pSlHttpServerEvent,
2  SlHttpServerResponse_t *pSlHttpServerResponse){
3   tareas|=TASK_NETWORK;
4   unsigned char strLenVal = 0;
5   unsigned char strLenVal2 = 0;
6   unsigned char ucQueueMsg = 0;
7   if(!pSlHttpServerEvent || !pSlHttpServerResponse){
8    return;
9   }
10  switch (pSlHttpServerEvent->Event){
11   case SL_NETAPP_HTTPGETTOKENVALUE_EVENT:{
12    unsigned char *ptr;
13    ptr = pSlHttpServerResponse->ResponseData.token_value.data;
14    pSlHttpServerResponse->ResponseData.token_value.len = 0;
15    if(memcmp(pSlHttpServerEvent->EventData.httpTokenName.data, GET_tokenFile,
16    strlen((const char *)GET_tokenFile)) == 0){
17     strLenVal = strlen(FILE_STRING);
18     memcpy(ptr, FILE_STRING, strLenVal);
19     ptr += strLenVal;
20     pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
```

```
21      if(FILESTATUS & 0x01){
22       strLenVal = strlen(FILE_CREATED_STRING);
23       memcpy(ptr, FILE_CREATED_STRING, strLenVal);
24       ptr += strLenVal;
25       pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
26      }else{
27       strLenVal = strlen(FILE_NON_CREATED_STRING);
28       memcpy(ptr, FILE_NON_CREATED_STRING, strLenVal);
29       ptr += strLenVal;
30       pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
31      }
32     }else if(memcmp(pSlHttpServerEvent->EventData.httpTokenName.data, GET_tokenBackup
            ,
33     strlen((const char *)GET_tokenBackup)) == 0){
34      strLenVal = strlen(BackupIA);
35      memcpy(ptr, BackupIA, strLenVal);
36      ptr += strLenVal;
37      pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
38     }else if(memcmp(pSlHttpServerEvent->EventData.httpTokenName.data,
           GET_tokenContadorMed,
39     strlen((const char *)GET_tokenContadorMed)) == 0){
40      strLenVal=strlen(contadormed);
41      memcpy(ptr,contadormed,strLenVal);
42      ptr+= strLenVal;
43      pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
44     }else if(memcmp(pSlHttpServerEvent->EventData.httpTokenName.data,
           GET_tokenActualMed,
45     strlen((const char *)GET_tokenActualMed)) == 0){
46      strLenVal = strlen(ValMR)-1;
47      memcpy(ptr,ValMR,strLenVal);
48      ptr+=strLenVal;
49      pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
50
51     }else if(memcmp(pSlHttpServerEvent->EventData.httpTokenName.data,
           GET_tokenI2Cerror,
52     strlen((const char *)GET_tokenI2Cerror)) == 0){
53      I2CerrorA=I2Cerror;
54      if(I2CerrorA){
55       strLenVal = strlen("E");
56       memcpy(ptr,"E",strLenVal);
57       ptr+=strLenVal;
58       pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
59       while(I2CerrorA){
60        if(I2CerrorA&0x1){
61         strLenVal = strlen("1");
62         memcpy(ptr,"1",strLenVal);
63         ptr+=strLenVal;
64         pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
65        }else{
66         strLenVal = strlen("0");
67         memcpy(ptr,"0",strLenVal);
68         ptr+=strLenVal;
69         pSlHttpServerResponse->ResponseData.token_value.len += strLenVal;
70        }
71        I2CerrorA=I2CerrorA>>1;
72       }
73      }
74     }
75     *ptr = '\0';
76    }
77    break;
78    case SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT:{
79     unsigned char led;
80     unsigned char *ptr = pSlHttpServerEvent->EventData.httpPostData.token_name.data;
81     if(memcmp(ptr, POST_token, strlen((const char *)POST_token)) == 0){
```

```
82      ptr = pSlHttpServerEvent->EventData.httpPostData.token_value.data;
83      strLenVal = strlen(LED_STRING);
84      if(memcmp(ptr, LED_STRING, strLenVal) != 0){
85       break;
86      }
87      ptr += strLenVal;
88      led = *ptr;
89      strLenVal = strlen(LED_ON_STRING);
90      strLenVal2 = strlen(LED_OFF_STRING);
91      ptr += strLenVal;
92      if(led == '1'){
93       if(memcmp(ptr, LED_ON_STRING, strLenVal) == 0){
94        osi_MsgQWrite(&g_tFILEW, &ucQueueMsg,OSI_NO_WAIT);
95        //Write file
96       }
97       else if(memcmp(ptr,LED_OFF_STRING,strLenVal2) == 0){
98        osi_MsgQWrite(&g_Respaldo, &ucQueueMsg, OSI_NO_WAIT);
99        g_ucFeedWatchdog=0;
100      }
101     }
102     else if(led == '2'){
103      ExtraerDWIFI((ptr-1));
104     }
105    }
106   }
107   break;
108   default:
109   break;
110  }
111  tareas&=~TASK_NETWORK;
112 }
```

Finalmente, en casos dónde se presenten problemas de funcionamiento o simplemente se requiera reiniciar el sistema, previo al reinicio, la tarea `Respaldar` de la Sección de código B.23 se encarga crear un archivo (línea 9) de nombre `med.txt` al que se le va a escribir el contenido del acumulador global de mediciones `AVal` sin cabeceras (línea 31). De esta forma, si al próximo inicio se requiere retomar las mediciones que no completaron un día, estas estén disponibles.

Sección de código B.23: Tarea `Respaldar` que crea un archivo para respaldar mediciones.

```
1  void Respaldar(void *pvParameters){
2   long lFileHandle;
3   unsigned char ucSyncMsg =0;
4   unsigned long ulToken;
5   long lRetVal;
6   while(1){
7    osi_MsgQRead(&g_Respaldo, &ucSyncMsg, OSI_WAIT_FOREVER);
8    //Respaldar mediciones realizadas
9    lRetVal = sl_FsOpen((unsigned char *)"bcp/med.txt",FS_MODE_OPEN_CREATE(16384,
         _FS_FILE_OPEN_FLAG_COMMIT|_FS_FILE_PUBLIC_WRITE|_FS_FILE_PUBLIC_READ),&ulToken
         ,&lFileHandle);
10   if(lRetVal < 0){
11    // File may already be created
12    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
13    ASSERT_ON_ERROR(lRetVal);
14   }
15   else{
16    // close the user file
17    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
18    if (SL_RET_CODE_OK != lRetVal){
19     ASSERT_ON_ERROR(FILE_CLOSE_ERROR);
```

```
20      }
21    }
22    //  open a user file for writing
23    lRetVal = sl_FsOpen((unsigned char *)"bcp/med.txt",
24    FS_MODE_OPEN_WRITE,
25    &ulToken,
26    &lFileHandle);
27    if(lRetVal < 0){
28      lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
29    }
30    // write to the file
31    lRetVal = sl_FsWrite(lFileHandle,0,AVal, sizeof(AVal));
32    if (lRetVal < 0){
33      lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
34    }
35    // close the user file
36    lRetVal = sl_FsClose(lFileHandle, 0, 0, 0);
37    if (SL_RET_CODE_OK != lRetVal){
38      ASSERT_ON_ERROR(FILE_CLOSE_ERROR);
39    }
40  }
41 }
```

Bibliografía

[1] J. Twidell and T. Weir, *Renewable energy resources*. Routledge, 2015.

[2] C. Hargreaves, "The Worldwide Distribution of Solar Resources by Nation by Act on Climate," Tech. Rep., 2015. [Online]. Available: http://www.wrforum.org/wp-content/uploads/2015/10/SS1-Hargreaves.pdf

[3] A. Hobson and M. Munsell, "US Solar Market Set to Grow 119 % in 2016, Installations to Reach 16 GW — SEIA," p. 1, 2016. [Online]. Available: http://www.seia.org/news/us-solar-market-set-grow-119-2016-installations-reach-16-gw

[4] CONAGUA, "Irradiación Solar," 2012. [Online]. Available: http://www.conagua.gob.mx/CONAGUA07/Contenido/Documentos/presentacion1.pdf

[5] G. Salgado Escobar, "Implementación de la energía solar como factor de abastecimiento energético en el sector vivienda: México 2006-2012," Ph.D. dissertation, 2009.

[6] M. A. R. Fraustro, "Evaluación económica y ambiental de escenarios de la energía solar en el sector residencial de México 2030," Ph.D. dissertation, Universidad Nacional Autónoma de México, 2010.

[7] Secretaría de Energía, "Beneficios de la GLD y EE en Mexico," SENER, Mexico, Tech. Rep., 2017. [Online]. Available: http://www.gob.mx/cms/uploads/attachment/file/201875/Beneficios_de_la_GLD_y_EE_en_Mexico.pdf

[8] J. W. Jung, J. W. Jo, E. H. Jung, and W. H. Jo, "Recent progress in high efficiency polymer solar cells by rational design and energy level tuning of low bandgap copolymers with various electron-withdrawing units," *Organic Electronics: physics, materials, applications*, vol. 31, pp. 149–170, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.orgel.2016.01.034

[9] S. Mekhilef, R. Saidur, and M. Kamalisarvestani, "Effect of dust, humidity and air velocity on efficiency of photovoltaic cells," *Renewable and Sustainable Energy Reviews*, vol. 16, no. 5, pp. 2920 – 2925, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032112001050

[10] N. Sharma, P. Sharma, D. Irwin, and P. Shenoy, "Predicting solar generation from weather forecasts using machine learning," in *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*. IEEE, 2011, pp. 528–533.

[11] T. Verma, A. Tiwana, C. Reddy, V. Arora, and P. Devanand, "Data analysis to generate models based on neural network and regression for solar power generation forecasting," in *Intelligent Systems, Modelling and Simulation (ISMS), 2016 7th International Conference on*. IEEE, 2016, pp. 97–100.

[12] H. Sangrody, M. Sarailoo, N. Zhou, N. Tran, M. Motalleb, and E. Foruzan, "Weather forecasting error in solar energy forecasting," *IET Renewable Power Generation*, 2017.

[13] V. Omubo-Pepple, "Effects of temperature, solar flux and relative humidity on the efficient conversion of solar energy to electricity," *... Journal of Scientific ...*, vol. 35, no. 2, pp. 173–180, 2009. [Online]. Available: http://content.imamu.edu.sa/Scholars/it/VisualBasic/ejsr_35_2_02.pdf

[14] S. Dubey, J. N. Sarvaiya, and B. Seshadri, "Temperature dependent photovoltaic (PV) efficiency and its effect on PV production in the world - A review," *Energy Procedia*, vol. 33, pp. 311–321, 2013.

[15] V. J. Fesharaki, M. Dehghani, J. J. Fesharaki, and H. Tavasoli, "The Effect of Temperature on Photovoltaic Cell Efficiency," *Proceeding of the 1st International Conference on Emerging Trends in Energy Conservation-ETEC*, no. November, pp. 20–21, 2011.

[16] M. Habibi, F. Zabihi, M. R. Ahmadian-Yazdi, and M. Eslamian, "Progress in emerging solution-processed thin film solar cells - Part II: Perovskite solar cells," *Renewable and Sustainable Energy Reviews*, vol. 62, pp. 1012–1031, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.rser.2016.05.042

[17] M. R. Maghami, H. Hizam, C. Gomes, M. A. Radzi, M. I. Rezadad, and S. Hajighorbani, "Power loss due to soiling on solar panel: A review," *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 1307 – 1316, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032116000745

[18] M. Unterweger and L. D. Costrell, "Data Acquisition Systems," 2009. [Online]. Available: http://www.osti.gov/servlets/purl/958282/

[19] Spiceworks, "How the IoT is changing Network monitoring," 2018. [Online]. Available: https://www.spiceworks.com/it-articles/iot-changing-network-monitoring/

[20] P. A. Heidenreich, C. M. Ruggerio, and B. M. Massie, "Effect of a home monitoring system on hospitalization and resource use for patients with heart failure," *American Heart Journal*, vol. 138, no. 4, pp. 633–640, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0002870399701766

[21] A. J. Sixsmith, "An evaluation of an intelligent home monitoring system," *Journal of Telemedicine and Telecare*, vol. 6, no. 2, pp. 63–72, 2000. [Online]. Available: https://doi.org/10.1258/1357633001935059

[22] N. K. Suryadevara and S. C. Mukhopadhyay, "Wireless sensor network based home monitoring system for wellness determination of elderly," *IEEE Sensors Journal*, vol. 12, no. 6, pp. 1965–1972, 2012.

[23] S. Gyulay, D. Gould, B. Sawyer, D. Pond, A. Mant, and N. Saunders, "Evaluation of a microprocessor-based portable home monitoring system to measure breathing during sleep," *Sleep*, vol. 10, no. 2, pp. 130–142, 1987.

[24] M. Wang, G. Zhang, C. Zhang, J. Zhang, and C. Li, "An IoT-based appliance control system for smart homes," in *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, 2013, pp. 744–747.

[25] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, and C. Lung, "Smart Home: Integrating Internet of Things with Web Services and Cloud Computing," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science(CLOUDCOM)*, vol. 02, 2013, pp. 317–320. [Online]. Available: doi.ieeecomputersociety.org/10.1109/CloudCom.2013.155

[26] Y. Jie, J. Y. Pei, L. Jun, G. Yun, and X. Wei, "Smart Home System Based on IOT Technologies," in *2013 International Conference on Computational and Information Sciences*, 2013, pp. 1789–1791.

[27] F. K. Santoso and N. C. H. Vun, "Securing IoT for smart home system," in *2015 International Symposium on Consumer Electronics (ISCE)*, 2015, pp. 1–2.

[28] a. Zanella, N. Bui, a. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6740844

[29] H. Schaffers, A. Sallstrom, M. Pallot, J. M. Hernandez-Munoz, R. Santoro, B. Trousse, J. Domingue, A. Galis, A. Gavras, T. Zahariadis, D. Lambert, F. Cleary, P. Daras, S. Krco, H. Müller, M.-S. Li, and Others, *The Future Internet-Future Internet Assembly 2011: Achievements and Technological Promises*, 2011.

[30] Dynatrace, "What is a monitoring environment?" 2018. [Online]. Available: https://www.dynatrace.com/support/help/get-started/introduction/what-is-a-monitoring-environment/

[31] DataDog, "Datadog products," 2018. [Online]. Available: https://www.datadoghq.com/product/

[32] Amazon, "Amazon Web services," 2018. [Online]. Available: aws.amazon.com/iot

[33] IBM, "Watson Internet of Things," 2018. [Online]. Available: https://www.ibm.com/internet-of-things

[34] HyperThings, "Solar monitoring," 2018. [Online]. Available: http://www.hyperthings.in/index.php/iot-solutions/solar-monitoring

[35] Y. Li, P. Niu, and Z. Su, "Design of greenhouse monitoring and control system based on LED lighting," *2015 12th China International Forum on Solid State Lighting, SSLCHINA 2015*, pp. 123–126, 2015.

[36] S. Adhya, D. Saha, A. Das, J. Jana, and H. Saha, "An IoT based smart solar photovoltaic remote monitoring and control unit," *2016 2nd International Conference on Control, Instrumentation, Energy & Communication (CIEC)*, no. October, pp. 432–436, 2016. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7513793

[37] M. J. Usher and D. A. Keating, *Sensors and Transducers: Characteristics, Applications, Instrumentation, Interfacing*, ser. New Electronics Series. Macmillan Education UK, 1996. [Online]. Available: https://books.google.com.mx/books?id=6jhdDwAAQBAJ

[38] J. Taylor, *Introduction To Error Analysis: The Study of Uncertainties in Physical Measurements*, ser. A series of books in physics. University Science Books, 1997. [Online]. Available: https://books.google.com.mx/books?id=giFQcZub80oC

[39] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui, "FPGA implementation of i2C & SPI protocols: A comparative study," *2009 16th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2009*, pp. 507–510, 2009.

[40] F. Leens, "An introduction to I2C and SPI protocols," *IEEE Instrumentation and Measurement Magazine*, vol. 12, no. 1, pp. 8–13, 2009.

[41] IOTeu, "The Internet of Things — the internet of things." [Online]. Available: http://www.theinternetofthings.eu/what-is-the-internet-of-things

[42] J.-s. Lee, Y.-w. Su, and C.-c. Shen, "A Comparative Study of Wireless Protocols :," *IECON Proceedings (Industrial Electronics Conference)*, pp. 46–51, 2007.

[43] Texas Instruments, "CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU," p. 71, 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc3200.pdf

[44] Texas Instruments, "HDC2010 Low Power Humidity and Temperature Digital Sensors," 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/hdc2080.pdf

[45] Texas Instruments, "OPT3001 Ambient Light Sensor," *Texas Instruments*, 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/opt3001.pdf

[46] Texas Instruments, "Precision Digital Current and Power Monitor With Low-Drift , Precision Integrated Shunt," 2016. [Online]. Available: http://www.ti.com/lit/ds/symlink/ina260.pdf

[47] J. A. Stankovic and R. Rajkumar, "Real-time operating systems," *Real-Time Systems*, vol. 28, no. 2-3 SPEC. ISS., pp. 237–253, 2004.

[48] Real Time Engineers Ltd., "The Free RTOS™ Reference Manual," 2016. [Online]. Available: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf

[49] H. Zhou, Y. Zhang, L. Yang, and Q. Liu, "Short-Term Photovoltaic Power Forecasting Based on Stacking-SVM," *2018 9th International Conference on Information Technology in Medicine and Education (ITME)*, pp. 994–998, 2018.