Diseño e Implementación de una Placa de Desarrollo para Aplicaciones de Nodos LoRaWAN

Alex Francisco Bustos Pinzón
Quién, para obtener el grado de
Ingeniero Electrónico

Presenta
Informe de Actividades de Investigación Desarrolladas
dentro del
Grupo de Investigación en Telecomunicaciones de la Universidad Distrital, GITUD

Dirigió el Ingeniero
José Noe Poveda Zafra

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍAS
PROYECTO CURRICULAR DE INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C.
2018

II

Índice general

Introducción ix

1. Descripción de Resultados 1

1.1. Hardware . . . 2

1.1.1. Breakouts . . . 2

1.1.2. Módulo Núcleo . . . 5

1.1.3. Módulo de Aplicación . . . 9

1.2. Librerías de Software . . . 12

1.2.1. Librería Serial . . . 13

1.2.2. Librerías I2C . . . 14

1.3. Documentación . . . 14

2. Análisis de Resultados 17

2.1. Placa de Desarrollo . . . 17

2.2. Librerías . . . 17

3. Exposición, Evaluación y Cumplimiento de Objetivos 21

3.1. Objetivo General . . . 21

3.2. Objetivos Específicos . . . 21

3.2.1. Documentación de información . . . 21

3.2.2. Librerías de Software . . . 22

3.2.3. Librería para control del módem LoRaWAN . . . 22

3.2.4. Diseño e Implementación de la Placa de Desarrollo . . . 22

3.2.5. Pruebas de Integración y Funcionamiento . . . 23

4. Conclusiones y Recomendaciones **25**
4.1. Criterios de diseño del módulo de Aplicación . . . 26
4.2. Trabajo Futuro . . . 27

A. Control de Versiones con Git **29**
A.1. Git: SCM de Código Abierto . . . 30
A.2. Estado de un Repositorio . . . 31
A.3. Creación/Clonado de un Repositorio . . . 32
A.4. El flujo de trabajo con Git . . . 32
A.5. Agregado y Registro de Cambios . . . 33
A.6. Ramas . . . 34
A.7. Actualización y Fusión . . . 34
A.8. Etiquetas . . . 34
A.9. Reemplazo de cambios locales . . . 35
A.10. Resumen . . . 35

B. Nodos LoRaWAN **37**
B.1. Introducción . . . 37
B.1.1. Activación en la red . . . 37
B.1.2. Intercambio de mensajes . . . 39
B.1.3. Configuración sugerida para un nodo de una WSN . . . 39
B.2. Librerías . . . 40
B.2.1. `pylatex.h` . . . 41
B.2.2. `WiMOD_LoRaWAN_API.h` . . . 41
B.2.3. `hci_stack.h` . . . 42
B.3. Ejemplos de uso de las librerías . . . 42
B.3.1. Envío de datos de sensores . . . 42
B.3.2. Envío directo de mensajes . . . 43

C. Librerías para Lectura de Sensores **45**
C.1. T6713: Sensor de CO2, de Amphenol Sensors . . . 45
C.2. BH1750FVI: Sensor de Iluminancia de ROHM Semiconductor . . . 46
C.3. iAQ Core: Sensor de CO2 y TVOC, de ams . . . 47
C.4. BMP280: Sensor de Presión y Temperatura de BOSCH . . . 48

Índice de figuras

1.1. Breakout para el PIC16F1769 con conexión ICSP 3
1.2. Características de la guía de ondas coplanar en un sustrato FR4 a 868 MHz 4
1.3. Breakout compatible con los módem iM880B e iM881A 4
1.4. Detalle de los caminos RF 5
1.5. Montajes punto a punto de encapsulados LCC 5
1.6. *Breakouts* de sensores 6
1.7. Ensamble de la PCB del *Núcleo* 7
1.8. Diagrama Esquemático del *Núcleo* 8
1.9. Representación 3D del *Núcleo*, revisada para producción 9
1.10. Nodo implementado en *protoboard* 10
1.11. Diagrama Esquemático del circuito del módulo de Aplicación 11
1.12. Modelo 3D del módulo de Aplicación, revisado para producción 12

2.1. *Mote* "mDot™ Box" de Multitech 18
2.2. "Mote II" de IMST 19
2.3. Tamaños del *núcleo* y el programador de Multitech 20

A.1. Página de descarga de Git 30
A.2. Apertura en contexto del terminal con soporte Git en Windows 31
A.3. Salida de `git status` sin Git 32
A.4. Salida de `git status` con Git 32

B.1. Interfaz del programa WiMOD LoRaWAN EndNode Studio de IMST 38
B.2. Perfil de Configuración por defecto para la Clase A. Tomado de [5]. 39

Índice de tablas

A.1. Resumen de comandos Git 35

B.1. Resumen de funciones de `pylatex.h` 41
B.2. Resumen de funciones de `WiMOD_LoRaWAN_API.h` 42
B.3. Resumen de funciones de `hci_stack.h` 42

C.1. Resumen de funciones de `bh1750fvi.h` 46

Introducción

El Internet de las Cosas, *Internet of Things* (IoT) es un concepto mencionado de manera recurrente que, a grandes rasgos, surge como una solución a la necesidad creciente de acceso a la información, pretendiendo interconectar elementos de diversa índole, para obtener información de ellos casi en tiempo real, en cualquier lugar en que se encuentre un usuario. Conectarlos a una Red de Área Local, *Local-Area Network* (LAN) sencilla, restringiría su visibilidad solamente a elementos cercanos [1]. Una Red de Área Amplia, *Wide-Area Network* (WAN) sería entonces el tipo adecuado de topología de red para esta labor.

Los distintos proveedores de telefonía celular brindan acceso a Internet, una WAN por excelencia, pero al operar en bandas de RadioFrecuencia (RF) cuyo uso requiere licencia, es imperativo el firmar un contrato con estos operadores para servirse de su infraestructura de telecomunicaciones y conectarlos a Internet. Las WAN de baja potencia, *Low-Power WAN* (LPWAN) se prestan como una alternativa a esto dado que operan en bandas RF para uso Industrial, Científico y Médico, *Industrial, Scientific and Medical* (ISM).

Alrededor del 2015, surgió un intento de varios fabricantes de integrar múltiples tecnologías LPWAN, resultando de ello la Alianza LoRaWAN [2]. La WAN de rango amplio, *Long-Range WAN* (LoRaWAN) es un protocolo para redes LPWAN, diseñado para conectar "cosas" energizadas por batería, a Internet en redes regionales, nacionales o globales, cuyo objetivo son los requerimientos del IoT como la comunicación bi-direcional, la seguridad de extremo a extremo y servicios de movilidad y localización [3]. Esta tecnología LPWAN tiene como base de funcionamiento a la modulación LoRa [4].

Componentes de un *nodo* LoRaWAN

En la jerga desarrollada dentro de la comunidad LoRaWAN, se concibe al *mote* o *nodo*, como al dispositivo final de la red que consume los recursos de la red LoRaWAN, e integra como mínimo a un microcontrolador (μC), un módem LoRaWAN, periféricos (sensores y actuadores) y una etapa de alimentación [5].

Microcontrolador

El μC es un Circuito Integrado, *Integrated Circuit* (IC) que incluye las unidades básicas de un computador [6, 7], pero con recursos reducidos dado que están dedicados a aplicaciones específicas.

En el *nodo* estas aplicaciones abarcarán, entre otras: desencadenar acciones a partir de eventos, como los mensajes entrantes de la red; agendar la lectura de sensores, realizar tareas de procesamiento de los datos y gestionar la energía disponible.

Módem LoRaWAN

El MOdulador/DEModulador (MODEM) LoRaWAN es el elemento de *Hardware* (hw.) que brinda una interfaz para el envío y recepción de mensajes sobre la red LoRaWAN, mediante un protocolo de comunicación entre periféricos, usualmente serializado, como el Transmisor/Receptor Universal Asíncrono, *Universal Asynchronous Receiver/Transmitter* (UART), la Interfaz de Puerto Serial, *Serial Port Interface* (SPI) o los Circuitos Inter-Integrados, *Inter-Integrated Circuits* (I^2C). Este elemento de hardware suele ser un dispositivo pasivo, esto es, que requiere de un anfitrión (en el caso del *nodo*, el *microcontrolador*) para darle utilidad alguna.

Periféricos

Son todos aquellos dispositivos auxiliares que permiten ingresar y extraer datos del sistema, como sensores, actuadores, pantallas, teclados, entre otros. Al ser el nodo la base de una Red Inalámbrica de Sensores, *Wireless Sensor Network* (WSN), los periféricos a considerar serán entradas al sistema, originadas por sensores de distintas variables.

Alimentación

Para que el nodo funcione la mayor cantidad de tiempo con la menor energía posible sin sobrecargas, debe existir un circuito que administre las distintas ofertas y demandas de cada uno de los elementos que conforman el nodo.

Prefacio

El Grupo de Investigación en Telecomunicaciones de la Universidad Distrital Francisco José de Caldas (UDFJC) (GITUD), en su compromiso con el proyecto de *Evaluación de una red IoT de rango amplio bajo el estándar LoRa*, apoyado por el Centro de Investigaciones y Desarrollo Científico de la UDFJC (CIDC); dispone de una cantidad variada de sensores y otros elementos destinados al proyecto, con los que se construyó una placa de desarrollo para esta tecnología, denominada *núcleo* (Capítulo 1), y unas librerías de software asociadas a la placa, que se pueden tomar como punto de partida para el desarrollo sobre esta. Tanto al *núcleo* como a las librerías se les hicieron algunos análisis (Capítulo 2), y se evaluaron en relación a los objetivos planteados en el Plan de Actividades de Investigación (Capítulo 3). Adicional a las conclusiones y recomendaciones finales (Capítulo 4), se incluyen unos apéndices relacionados con los aportes a un libro, actualmente en fase de redacción, que condensa los resultados del proyecto, y toda la información adicional se encuentra en el CD adjunto.

Capítulo 1

Descripción de Resultados

En el GITUD se disponen los siguientes elementos:

- Sensor HPMA115S0 [8]
- Sensor iAQ Core C [9]
- Sensor HDC1010 [10]
- Sensor BMP280 [11]
- Sensor BH1750FVI [12]
- Sensor T6713 [13]
- Módem LoRaWAN iM880B-L [14]
- μC PIC16F1769 en encapsulado *Small Outline IC* (SOIC) de 20 pines[15]
- μC PIC18F2550 en encapsulado *Plastic Dual In-Line* (PDIP) de 28 pines[16]
- GPS GNS601 [17]
- Regulador de Carga de celda de Litio, MCP73832 , 500 mA max.
- Regulador de Voltaje PAM2305AAB330, 3.3V, topología buck [18]
- Conectores SMA de borde de placa
- Conectores micro-USB
- Antenas con conector SMA

Algunas observaciones de los componentes anteriores, de los cuales la mayoría son de montaje superficial:

- El PIC18F2550 se puede trabajar directamente en una Placa de Prototipos, *breadboard* (*protoboard*) pero se dispone apenas de un par de unidades.
- La mayoría de sensores ocupan un área menor a 4 mm × 4 mm.
- El Módem LoRaWAN, el GPS GNS601 y el sensor iAQ Core; presentan encapsulado *Leadless Chip Carrier* (LCC) superficial, con la mayor separación entre pines de todos los elementos de montaje superficial listados, característica que se aprovechó para empezar a desarrollar en estos, empleando la técnica de soldado punto a punto para obtener un montaje rudimental, mostrado en la Figura 1.5.
- El sensor de material particulado HPMA115S0 no es de montaje superficial, pero requiere un conector especial[8] para poder acceder a sus pines.

Ante estos componentes y sus limitantes, el enfoque consistió en separar las varias labores a las que el desarrollador se tendrá que enfrentar.

1.1. Hardware

Se planteó inicialmente que la solución abordara en la misma Placa de Circuito Impreso, *Printed Circuit Board* (PCB), todos los componentes del *nodo*. Esto tiene el inconveniente de que el *nodo* estaría limitado a esa combinación particular de componentes y μC, lo que para el desarrollo, incrementaría el precio de la solución, si los componentes no se usan del todo. Para mitigar esto, el circuito del *nodo* se segmentó en dos módulos, denominados *núcleo* y *aplicación*. Los componentes de cada módulo se probaron mediante circuitos de prueba por componente. Las PCB se desarrollaron en el *software* (sw.) para la Automatización del Diseño Electrónico, *Electronic Design Automation* (EDA), *Proteus*, de *Labcenter Electronics*. Con este sw. se generaron los archivos en formato Gerber, empleados en la industria de manufactura de PCBs, tal fue el caso para el laboratorio de PCBs de la Facultad de Ingenierías de la UDFJC, en el que también se dispone de la licencia del sw. mencionado.

1.1.1. Breakouts

Cada *breakout* es un circuito que incluye un componente de hw. en particular, siendo apto para etapas de aprendizaje, desarrollo y evaluación del componente de hw. en cuestión. Un *breakout* se ensambla típicamente sobre una PCB, e incluye uno o más conectores aptos para ser insertados, típicamente en una *protoboard*.

Microcontrolador

El primer candidato a tener un *breakout* es precisamente el μC, dada su labor de dictar el comportamiento del *nodo*. Se muestra el *breakout* del PIC16F1769 en la Figura 1.1, que incluye un conector adicional para la interfaz de Programación Serial en Circuito, *In-Circuit Serial Programming* (ICSP), que puede ser empleada con el programador PicKit de Microchip, u otro programador, siempre y cuando respete tanto la disposición de pines para ICSP sugerida por Microchip [19], como los niveles de voltaje del μC [15].

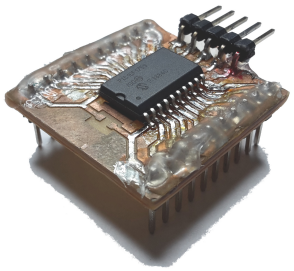


Figura 1.1: Breakout para el PIC16F1769 con conexión ICSP

Módem LoRaWAN

El *breakout* del módem LoRaWAN iM880B-L, Figura 1.3; presenta espacio para soldarle dos regletas de pines. En el extremo de una de ellas se encuentran los conectores mínimos para su control por interfaz UART, pero se pueden soldar también los demás terminales, de ser necesario.

La única consideración especial al momento de diseñar su *breakout* fue precisamente la relacionada con su aspecto RF: el trayecto que conecta al módem con el conector SMA, ha de tener una impedancia característica de 50 Ω [14, 20]. Para calcular las distintas dimensiones del trayecto se ingresaron los parámetros conocidos de la PCB en un sw. para el análisis y síntesis de líneas de transmisión, como los incluidas en Qucs o KiCad. Dichos parámetros se muestran en la Figura 1.2 a una frecuencia de 868 MHz, en la que operan los estación base (*gateway*) LoRaWAN de GITUD, y partiendo de estos parámetros se diseñaron las guías de onda coplanares, tanto en el *breakout* del módem LoRaWAN, como en el *núcleo*, como se detalla en la Figura 1.4.

Mientras se diseñaba y ensamblaba su *breakout*, se realizó un montaje rústico para acceder a la interfaz UART de uno de los módulos, soldando alambres calibre 22 AWG en los pines de esta interfaz y así poder trabajarlo de manera cómoda y compacta en *protoboard*, como en la Figura 1.5, que posteriormente se empleó en el nodo implementado en *protoboard* de la Figura 1.10.

Sensores

La circuitería requerida por los sensores mencionados al comienzo del capítulo es mínima, dado que su cadena de medida está integrada en el mismo encapsulado, e incluyen también una interfaz digital como I^2C o UART, para permitir la lectura externa de sus valores, por ejemplo desde un μC. La dificultad radicó en soldarlos a sus *breakouts*, Figura 1.6, y al módulo de *aplicación*, Figura 1.12: El HDC1010 presenta un encapsulado de Arreglo de Esferas en Malla, *Ball-Grid Array* (BGA), con separación entre esferas de 0.5 mm, y una separación recomendada de 0.237 mm [10] para las zonas de cobre en la PCB que dan soporte a los terminales, llamadas *pads*.

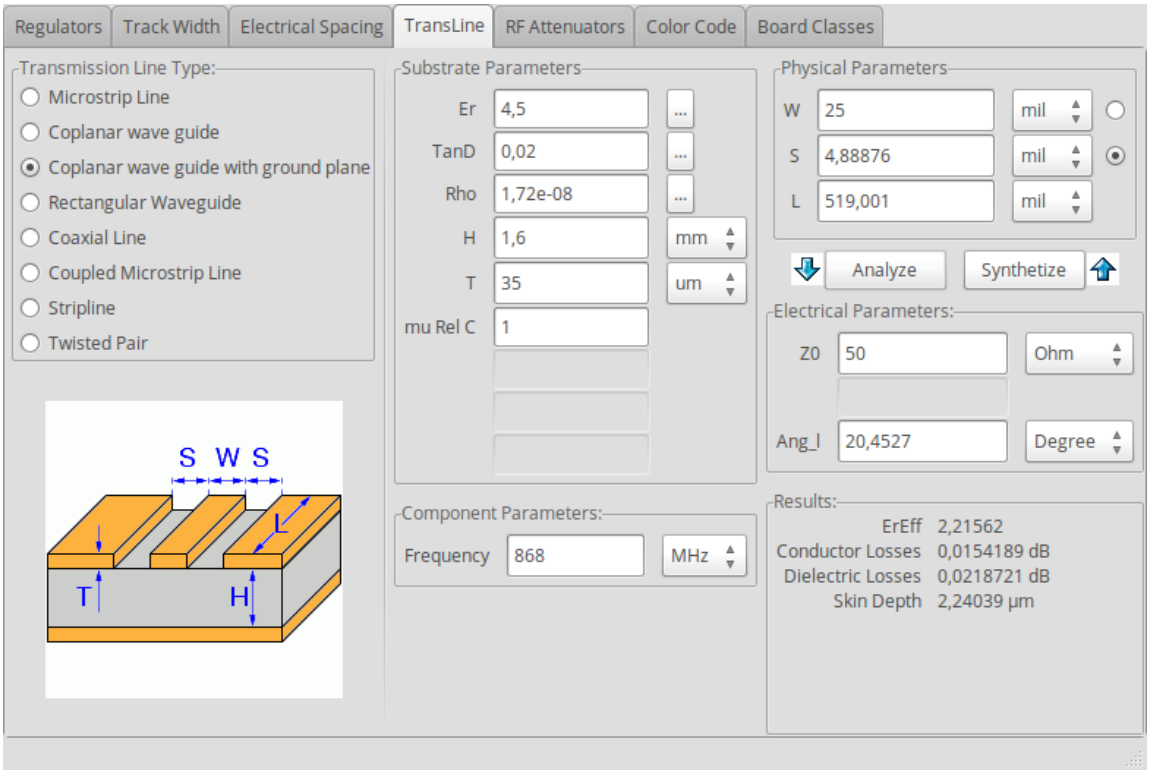


Figura 1.2: Características de la guía de ondas coplanar en un sustrato FR4 a 868 MHz

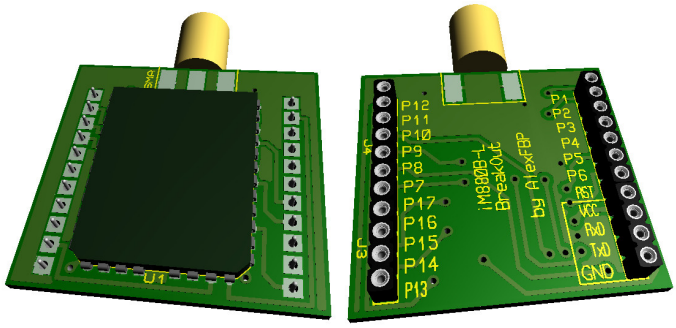


Figura 1.3: Breakout compatible con los módem iM880B e iM881A

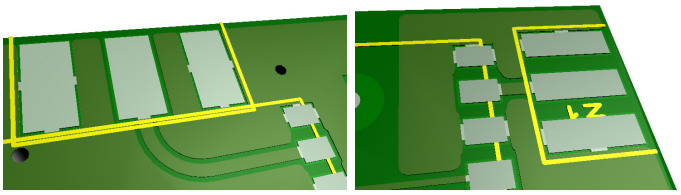


Figura 1.4: Detalle de los caminos RF creados. Izquierda: *breakout* del iM88xx, Derecha: *núcleo*

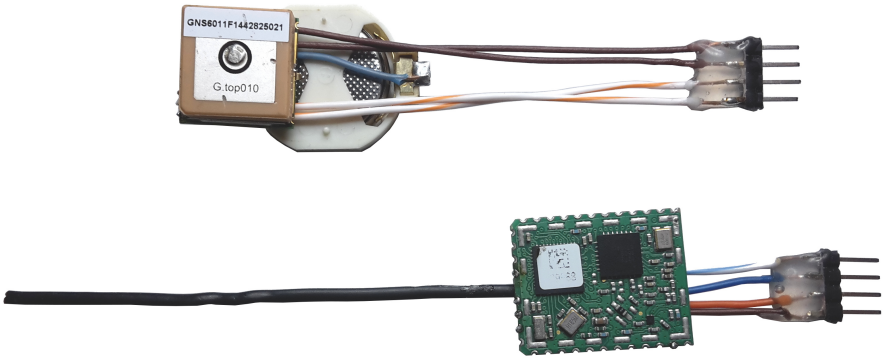


Figura 1.5: Montajes punto a punto de encapsulados LCC, con puntos de soldadura con la regleta de pines reforzados y aislados con silicona. Arriba: GPS GNS601. Abajo: módem LoRaWAN iM880B-L, del cuál, la antena es un alambre cortado a $\lambda/4_{868\ MHz} \simeq 8{,}64\ cm$, soldado directamente sobre el pin RF del módem.

1.1.2. Módulo Núcleo

El *núcleo*, mostrado en la Figura 1.7, es la placa de desarrollo principal, emplea al PIC16F1769 para controlar al módem iM880B-L mediante una interfaz UART 115200,8N1[1][14]. Se le reservó un espacio para soldar regletas de pines comerciales y así acceder a los terminales restantes tanto del μC como del módem.

El nodo implementado en *protoboard* de la Figura 1.10, fue el punto de partida para el *núcleo*, empezando con el PIC18F2550, mientras se fabricaban las PCB necesarias para el ensamble de los *breakouts*. Al ser los componentes mayormente de montaje superficial, obligó desde un comienzo a incluir el soporte a depuración y programación en circuito, dado que no tiene sentido someter al IC a estrés térmico y mecánico, asociados al proceso de desoldado, en caso de pensar cambiar su programación extrayendo al IC de la PCB. Así se consideró y verificó el espacio para ICSP, antes de conectar los demás elementos en la *protoboard*.

[1] Las tres letras en "8N1", en su orden, codifican la longitud de bits, el tipo de bit de paridad y la cantidad de bits de parada. En este caso, 8 bit de datos, sin bit de paridad (None) y un bit de parada, respectivamente.

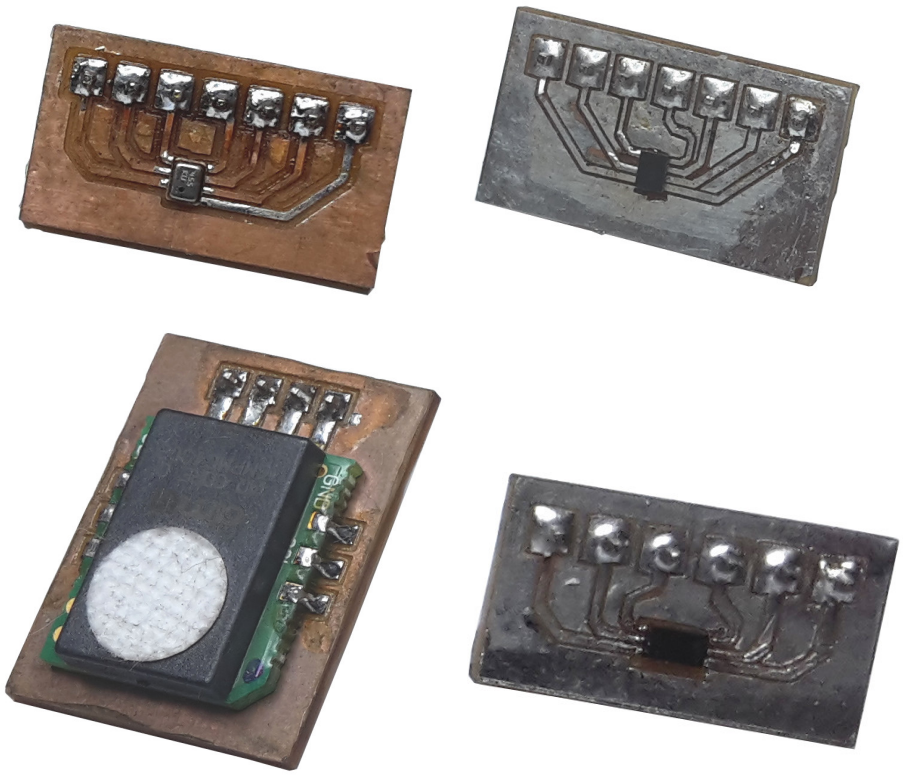


Figura 1.6: *Breakouts* de sensores. De izquierda a derecha y de arriba a abajo: BMP280, HDC1010, iAQ Core y BH1750FVI.

Identificados los componentes esenciales de la *protoboard* que podían quedar aislados, y después de finalmente ensamblado el *breakout* del PIC16F1769 y probado en la *protoboard*, se llegó al esquema eléctrico del *núcleo*, Figura 1.8, el cual fue el punto de partida para el diseño y ensamble del primer prototipo, mostrado en la Figura 1.7.

Posterior a la implementación de este prototipo, se le realizó una revisión minuciosa para prepararlo a un proceso de producción en masa, cuyo modelo revisado se ilustra en la Figura 1.9.

Selección de Periféricos por Puerto

La gran mayoría de μCs suele disponer de varios periféricos digitales y análogos en el mismo IC. Al no ser usual emplearlos todos o gran parte de ellos, de manera simultanea; es común encontrar que en un μC los pines se encuentran multiplexados, esto es, comparten sus Entradas y Salidas (E&S) como periférico, con las E&S de uno o más de los demás periféricos, que en caso de no ser utilizados, dejan al pin como una E&S de propósito general, *General Purpose Input-Output* (GPIO) [6, 7]. Pero en caso de demandarse demasiados periféricos, la multiplexación asignada por el fabricante puede

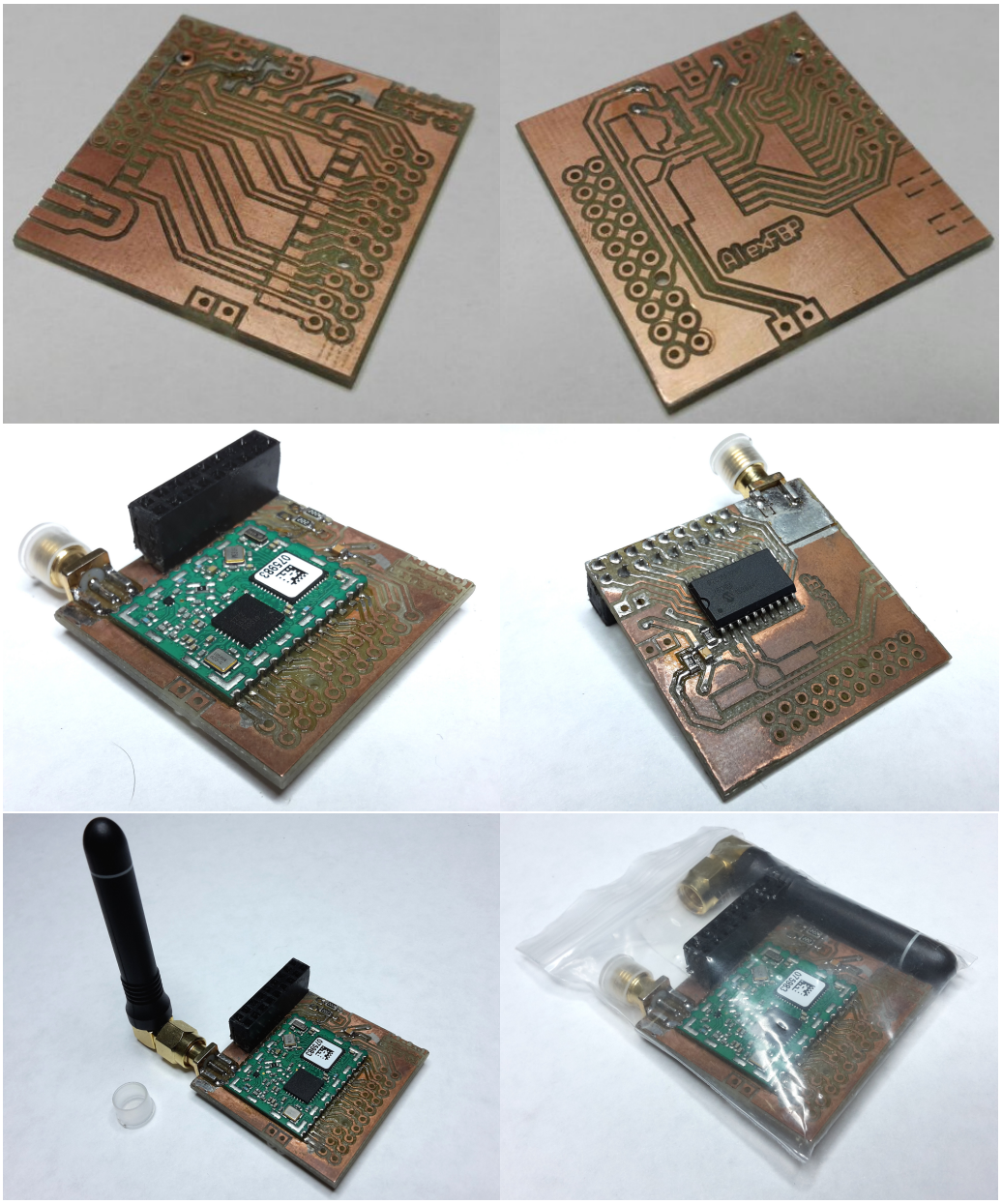


Figura 1.7: Ensamble de la PCB del *Núcleo*. Arriba: PCB realizada. Centro: *Núcleo* ensamblado. Abajo, izq.: Unión con antena. Abajo, der.: Ejemplo de empaque para distribución.

llegar a ser conflictiva, a pesar de que pareciera ser que en la mayoría de los casos se diseña de manera conveniente.

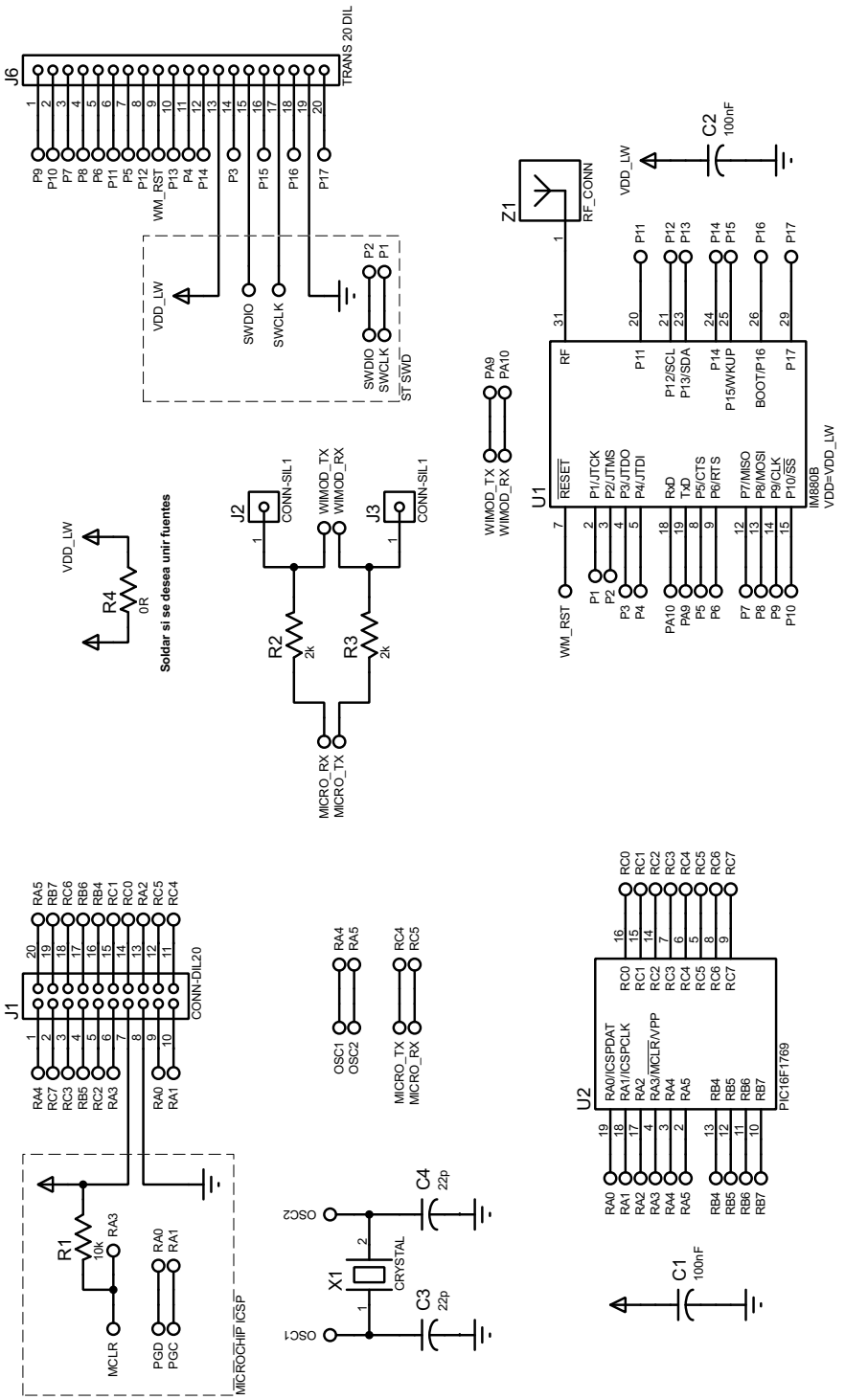


Figura 1.8: Diagrama Esquemático del *Núcleo*

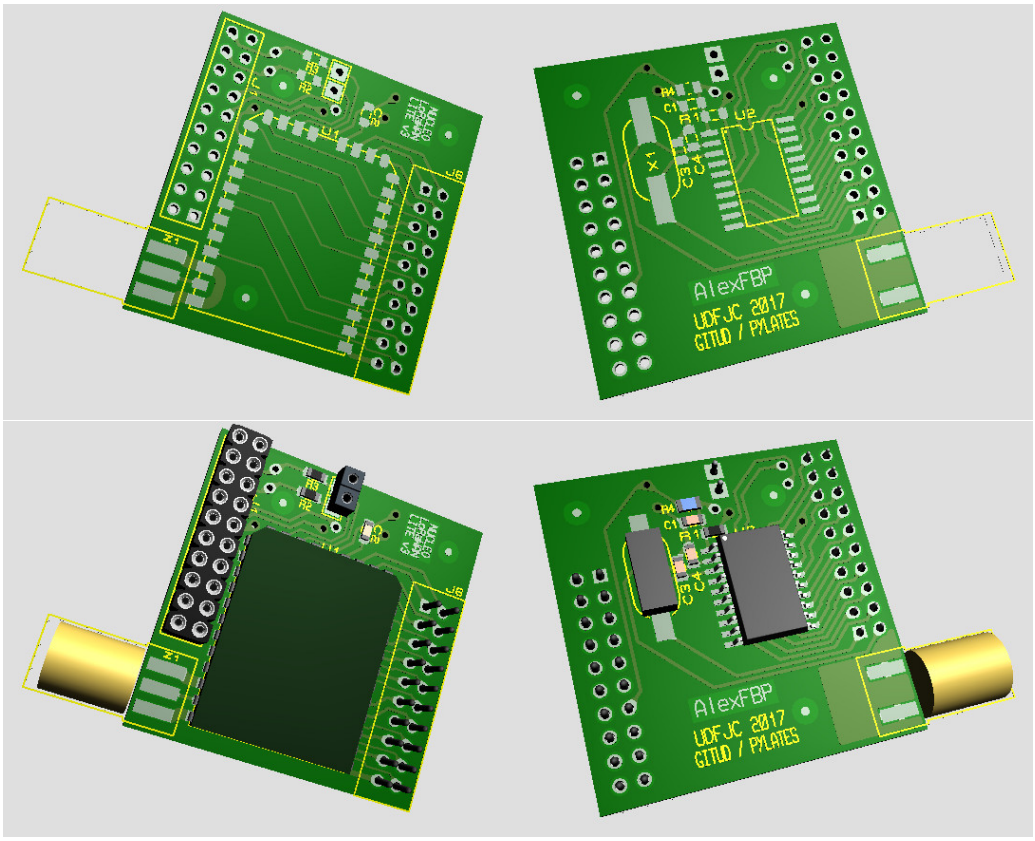


Figura 1.9: Representación 3D del *Núcleo*, revisada para producción

El PIC16F1769 dispone de *Peripheral Port Selection* (PPS), un módulo del que dependen los demás periféricos dado que configura el cómo se conectan las entradas y salidas de los periféricos con los pines del μC, generalizando así el concepto de multiplexación de pines a, al menos, los periféricos digitales. Sumando esto a la disponibilidad de todos sus pines en la PCB, brinda al desarrollador o desarrolladores, la libertad de seleccionar arbitrariamente la ubicación de los periféricos en caso que la configuración por defecto no sea conveniente, inclusive en tiempo de ejecución; característica limitada en la practica solamente por la cantidad de pines y periféricos disponibles en el dispositivo, aproximándose casi a un Dispositivo Lógico Programable, *Programmable Logic Device* (PLD).

1.1.3. Módulo de Aplicación

El módulo de aplicación, cuyo diagrama esquemático se ilustra en la Figura 1.11, presenta espacio para soldar un conector *Dual In-Line* (DIL), o regleta de pines de doble fila, de 20 pines; que empata con el conector J1 del circuito del *núcleo*, Figura 1.8, y brinda energía a este último desde el conector micro USB y/o desde una batería de Litio opcional que se puede conectar en el socket dispuesto para esta. Incluye los elementos excluidos del *núcleo* que originalmente conformaban la implementación del

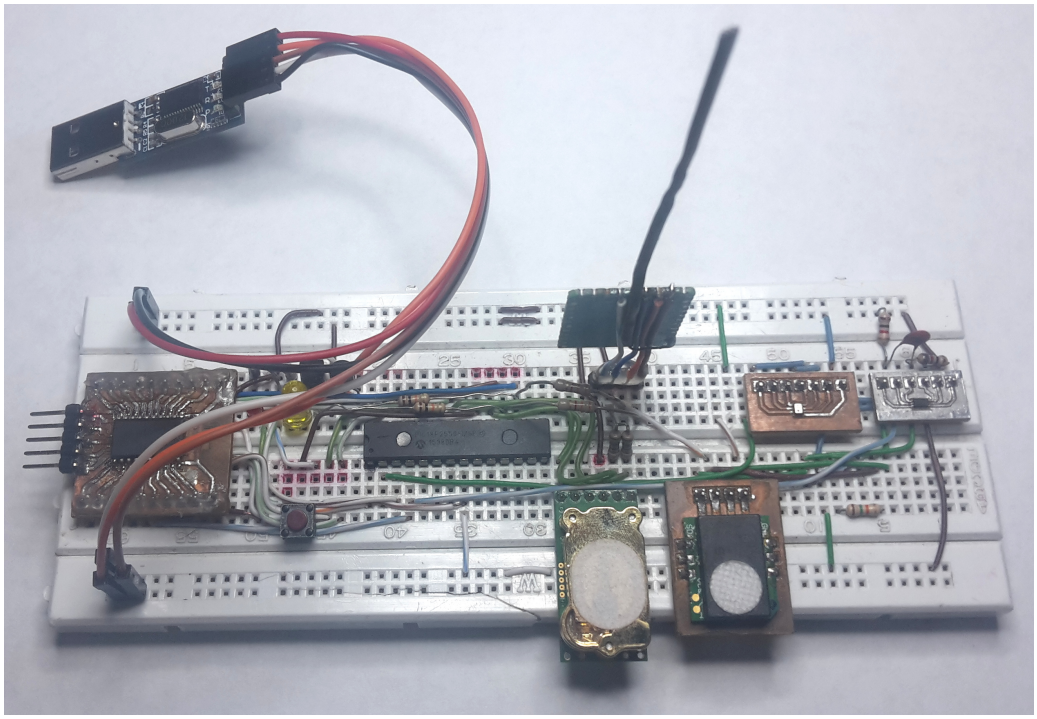


Figura 1.10: Nodo implementado en *protoboard*. Emplea un módulo serial-USB basado en el IC PL-2303 de Prolific[21]; para energizar al *nodo*, dado que se toman de este los voltajes de 3.3 V y 5 V disponibles, pero eventualmente se usa para labores de depuración. Para ilustrar el espacio reservado al PIC18F2550 y al *breakout* del PIC16F1769, se muestran ambos en el circuito, pero ¡solo debe estar uno de los dos!. Arriba del pulsador de botón rojo (para reinicio) está el espacio para ICSP, un LED rojo para labores de depuración y otro LED amarillo para indicar actividad en las líneas UART.

nodo en *protoboard* de la Figura 1.10, como los sensores para la aplicación WSN, un GPS GNS601 con batería de respaldo CR-2032 [17], para brindar geo-referenciación a los datos; y los circuitos asociados a la regulación de la carga de la batería de litio y regulación del voltaje del sistema.

Como los rangos de alimentación de los sensores disponibles varían, se necesitan al menos dos voltajes (5 V y 3.3 V) para energizar la totalidad de componentes en el mismo circuito. El conector micro USB, común actualmente en los teléfonos celulares; energiza al circuito de carga para la batería de Litio opcional, basado en el IC MCP73832; y también los elementos de 5 V, para así obligar a simplificar el consumo y diseño del circuito de aplicación, limitando el respaldo de batería a componentes de 3.3 V, los cuales se energizan mediante el regulador de voltaje en IC PAM2305AAB330, de tipo conmutado, con topología buck [24, 18].

En el módulo de aplicación se aprecian también dos elementos cuya interfaz de lectura es únicamente UART: El GPS GNS601 y el sensor HPMA115S0. [17, 8]. Al haber solo un módulo EUSART en el PIC16F1769, que ya se está usando como UART para la comunicación ya existente del *núcleo* con el módem, se aprovecha el PPS (Sección 1.1.2) para multiplexar internamente el módulo serial.

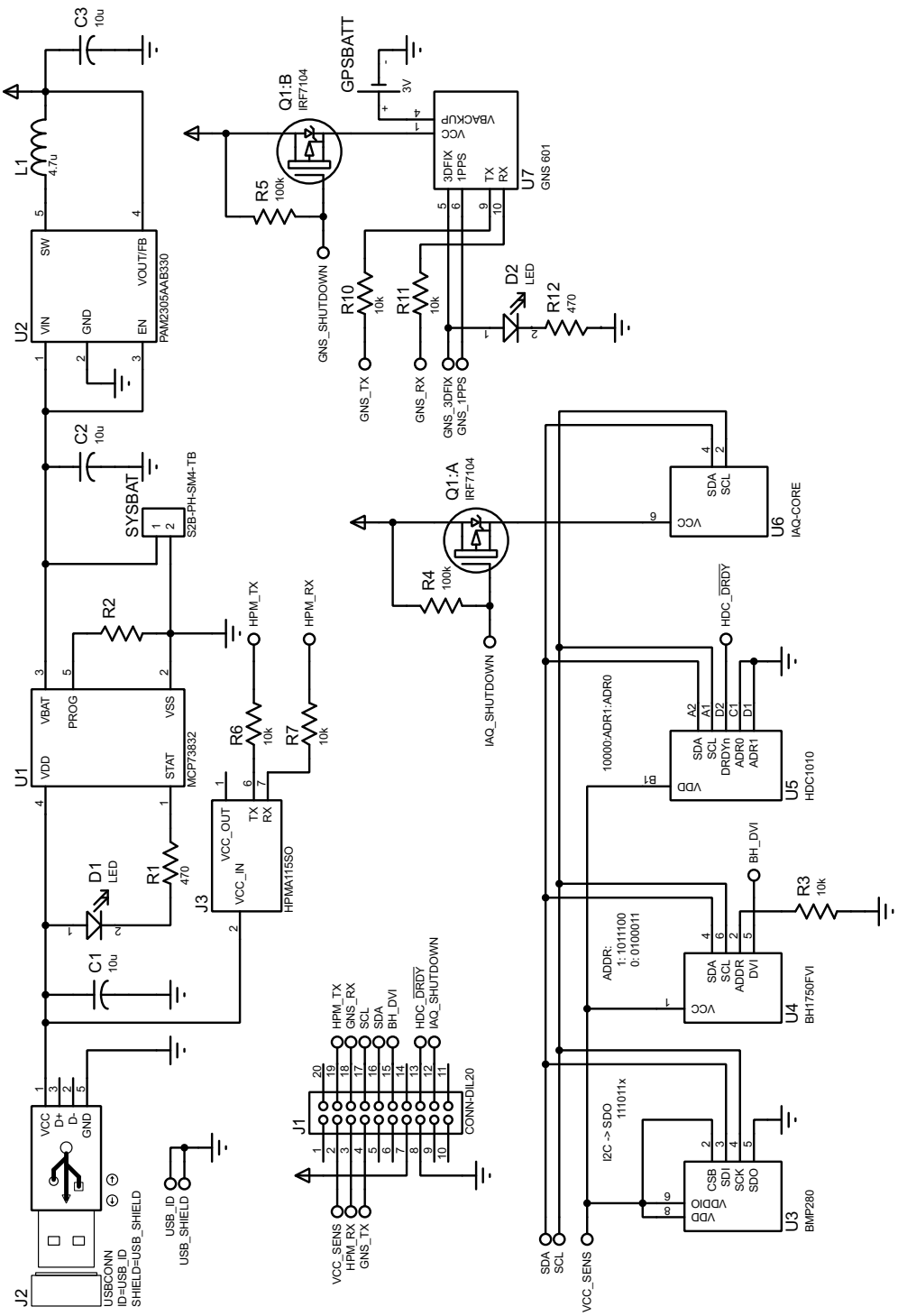


Figura 1.11: Diagrama Esquemático del circuito del módulo de Aplicación. R2 se debe calcular según [22, 23]

Este código es comparativamente menor en instrucciones de procesador y complejidad, que el que se destinaría en una implementación de software del UART.

Verificado el esquema de la Figura 1.11 para la aplicación, se tomó como base para ensamble sobre PCB, cuyo modelo 3D se muestra en la Figura 1.12. Este se logró gracias a los *breakouts* realizados para los sensores, Figura 1.6

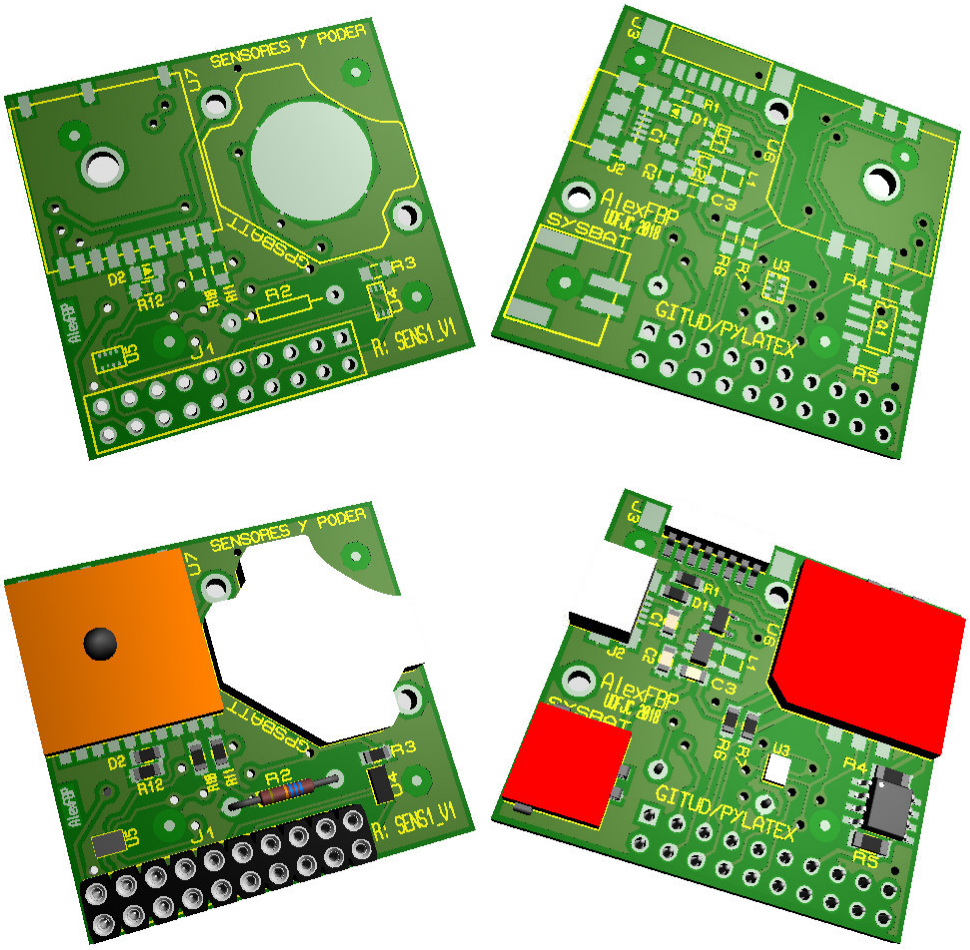


Figura 1.12: Modelo 3D del módulo de Aplicación, revisado para producción

1.2. Librerías de Software

Para agilizar el tiempo de desarrollo y a la vez brindar una solución portable de software, al menos entre μCs, se decidió programar en C, en específico con la implementación XC8 de Microchip [25] la

cual es conforme a ANSI C, estándar del Instituto Nacional Estadounidense de Estándares, *American National Standards Institute* (ANSI) para el lenguaje de programación C.

Como Entorno Integrado de Desarrollo, *Integrated Development Environment* (IDE), se empleó MPLAB X IDE, también de Microchip[26], basado en NetBeans IDE [27]; el cual, junto al programador PicKit 3 disponible en el almacén de laboratorios de la Facultad de Ingenierías de la UDFJC; permite depurar empleando el mismo puerto para ICSP ya disponible.

Dada la envergadura del proyecto de software, se decidió brindarlo de un Sistema de Control de Versiones, *Version Control System* (SCV). En específico se decidió por Git, el cual es un SCV distribuido, lo que facilita también el trabajo colaborativo, el manejo de errores, entre otros [28, 29]. El proyecto completo se puede *clonar* desde su *repositorio* en GitHub [30], siguiendo el procedimiento del Apéndice A. Como herencia de NetBeans, IDE del cual se basó MPLAB X, este tiene soporte nativo a Git.

Una *librería* en C se crea a partir de varios elementos, de código fuente escritos también en C, que pueden ser abiertos en un editor de texto plano como el Block de Notas de Windows, o nano, vim, emacs, en plataformas Linux, o soluciones multiplataforma como Atom y Visual Studio Code; o también depender de archivos binarios ya compilados, cuya apertura en estos editores de texto muestra casi en su totalidad, caracteres sin sentido. Entre otros [31, 32, 33]:

- Una "cabecera" o *header*, archivo de texto plano con extensión ".h". Este también se puede ver como una interfaz de programación de aplicaciones, *Application Programming Interface* (API) dado que condensa los prototipos de las funciónes disponibles de la librería
- Las implementaciones en C de estos archivos, sea en código fuente, con extensión ".c"
- Códigos objeto, archivos binarios terminados en ".o" para objetos locales o en ".so" para librerías compartidas del sistema, compilados acorde a las cabeceras.

Serán el compilador y el *linker* los encargados de unir todas las librerías al código principal, para así compilar el archivo .hex con el programa final [34], labor que suele ser gestionada de manera transparente por el mismo IDE.

1.2.1. Librería Serial

Tanto para labores de depuración mediante envío simple de cadenas ASCII a modo de verificación, por ejemplo, para visualizar los valores leídos de sensores, mientras se envían sus valores por UART, como de soporte a la pila HCI mencionada anteriormente; se implementó una librería para el control del Transmisor/Receptor Síncrono/Asíncrono Universal Mejorado, *Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter* (EUSART) del PIC, a modo de UART. Esta se basó en el código de ejemplo disponible en la página del fabricante del iM880 [14], en el código generado por el Configurador de Código de Microchip, *Microchip's Code Configurator* (MCC) [35], una vez instalado en MPLAB X, y en las características puntuales de los módulos EUSART de cada μC [15, 16].

Módem LoRaWAN iM880B de IMST

Los módem LoRaWAN xDot de Multitech, y RN2xxx de Microchip; se controlan por comandos ASCII, lo que facilita su interpretación e ingreso directo mediante casi cualquier cliente de terminal serial como *PuTTy*[36], *Hyperterminal*[37], *minicom*[38], u otra herramienta de sw. para acceder a los puertos de comunicación serial de un computador personal, *Personal Computer* (PC). Los módem iM880B y toda la familia de módems de IMST se controlan mediante una pila de protocolos, la Interfaz de Control del Anfitrion, *Host Controller Interface* (HCI), que sigue basándose en la interfaz UART. La pila HCI aligera los tiempos de procesamiento, de validación y de envío de comandos; dado que disminuye la cantidad de octetos necesarios para enviar un comando, teniendo como contrapartida que hace ilegible el mensaje mediante las herramientas mencionadas anteriormente dado que el formato no tiene correspondencia ASCII sino ha de interpretarse de manera binaria.

1.2.2. Librerías I2C

Los sensores de la placa de aplicación se leen mediante I^2C, aprovechando las librerías I^2C generadas por el MCC, que configuran a bajo nivel el Puerto Serial Síncrono Maestro, *Master Synchronous Serial Port* (MSSP) del PIC como I^2C maestro. Dicha librería se tomó como base para las librerías de los siguientes sensores:

- HDC1010, de Humedad y Temperatura
- iAQ Core, de CO_2 y TVOC
- BH1750FVI, de Iluminancia
- BMP280, de Presión Ambiental y Temperatura

Una explicación de las funciones básicas de la librería para cada sensor se encuentra en el Apéndice C.

1.3. Documentación

Para facilitar el desarrollo de soluciones enfocadas al IoT con los productos anteriores, se recopilaron los siguientes elementos en el CD adjunto al presente informe:

- Manuales de uso rápido de ambas placas
- Hojas de datos originales de los distintos componentes
- Una hoja de cálculo, a modo de base de datos; con la información destacada de los componentes disponibles.
- Una copia (al momento de grabación del CD) del repositorio del proyecto.

La implementación de las librerías de software (ficheros ".c") se escribió en lo posible de manera auto-documentada[39] y sus cabeceras (archivos ".h") (sección 1.2) incluyen la documentación de cada una de sus funciones y declaraciones, en formato *javadoc*[40, 41]. Por la primer parte, la auto-documentación; al asignar nombres para la labor de cada variable; evita la labor de agregar comentarios para explicar el código, y a su vez facilita la interpretación del código para quien quiera seguir trabajando sobre el PIC. Y la segunda parte, se presta para dar homogeneidad entre implementaciones: En caso que la implementación no sirva, por ejemplo, al cambiar de plataforma de desarrollo; se puede tomar la cabecera y sus descripciones como criterio para realizar la implementación particular de estas, que fue justamente lo que se hizo para la librería WiMOD LoRaWAN de IMST, disponible como un código de ejemplo [14] que solo compila bajo plataformas Windows.

Adicionalmente el repositorio se encuentra documentado en formato Markdown[42, 43], convención de redacción ligera que se puede interpretar leyendose directamente en un editor de texto plano y así mismo servir para generar HTML, como lo hacen varios servidores Git remotos (Apéndice A) como GitHub[2], BitBucket[3] y GitLab[4].

[2] https://github.com
[3] https://bitbucket.org
[4] https://gitlab.com

Capítulo 2

Análisis de Resultados

Se brinda a continuación un análisis tanto de los dos productos de hardware y software principales, como de los derivados:

2.1. Placa de Desarrollo

El *mote* de Multitech, 'MDOT-EVB-xxx', Figura 2.1, incluye algunos sensores, un módulo GPS y un módem MDOT-xxx-xx-UFL. Los precios del módem y el *mote* en Octopart rondan a la fecha del presente documento, los USD 35 y USD 122 respectivamente.

Competencia del *mote* de Multitech es el "Mote II" de IMST, Figura 2.2, también con otros sensores y un GPS, pero con el iM881A como su módem LoRaWAN[45, 20]. Este último y el *mote*, a la fecha cuestan USD 16 y USD 238 respectivamente, directamente en la tienda de IMST [46].

La dimensión promedio de ambos *motes* es de $15 \times 8 \times 3\ cm^3$. En la Figura 2.3 se hace una comparación de los tamaños del *núcleo* con el programador que aparece en la Figura 2.1. El volumen del *nodo* resultante, compuesto por las dos placas y sobre estas la batería de litio, es de aproximadamente $5 \times 4 \times 3\ cm^3$, siendo un 16 % más compacto que el del promedio de los dos *motes*.

2.2. Librerías

Las librerías creadas se entenderán como un producto intermedio, una herramienta para la producción de un software final. El estándar ISO/IEC 9126 define un modelo que clasifica la calidad de un software en un conjunto de características y sub-características, donde la traducción de sus términos base se presenta a continuación:

- Funcionalidad: *Conjunto de atributos que tienen que ver con la existencia de un conjunto de funciones y sus propiedades especificadas. Las funciones son aquellas que satisfacen necesidades declaradas o implícitas.*



Figura 2.1: *Mote* "mDot™ Box" de Multitech a la derecha, junto a su programador a la izquierda. Fuente: Página web del producto [44].

- Fiabilidad: *Conjunto de atributos que se basan en la capacidad del software para mantener su nivel de rendimiento en las condiciones establecidas durante un período de tiempo establecido.*
- Usabilidad: *Un conjunto de atributos que se relacionan con el esfuerzo necesario para su uso, y con la evaluación individual de dicho uso, por un conjunto de usuarios establecidos o implícitos.*
- Eficiencia: *Conjunto de atributos que tienen relación con el nivel de rendimiento del software y la cantidad de recursos utilizados, en las condiciones establecidas.*
- Mantenibilidad: *Un conjunto de atributos que se relacionan con el esfuerzo necesario para realizar modificaciones específicas.*
- Portabilidad: *Un conjunto de atributos que tienen que ver con la capacidad del software para ser transferido de un entorno a otro.*



Figura 2.2: "Mote II" de IMST. Su programador viene integrado en el mismo dispositivo. Fuente: Ítem en su tienda web oficial[46]

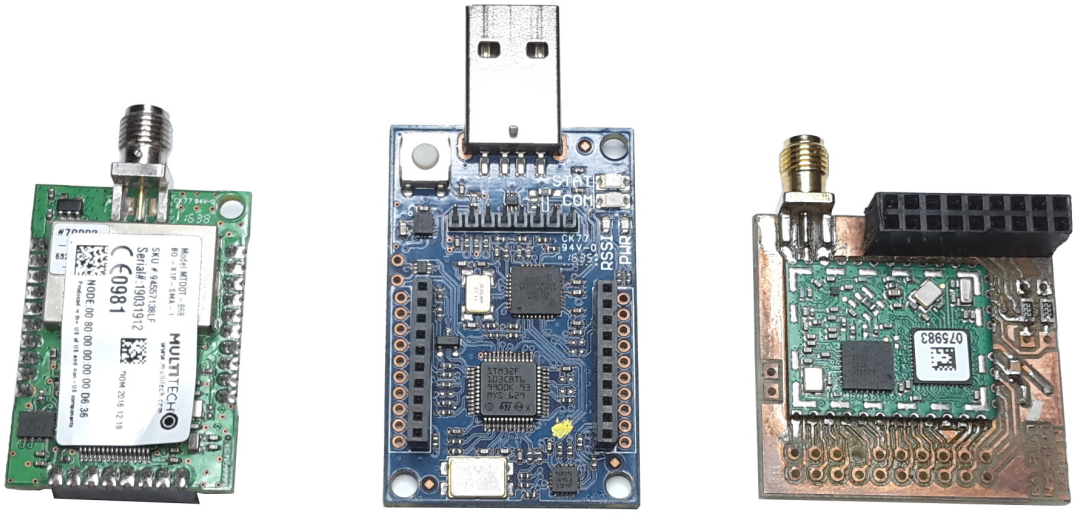


Figura 2.3: Tamaños del *núcleo* y el programador de Multitech. Izquierda: Módem LoRaWAN MDOT-868-SMA. Centro: Programador para la familia mDot de Multitech. Derecha: *Núcleo* ensamblado.

Capítulo 3

Exposición, Evaluación y Cumplimiento de Objetivos

Los objetivos planteados en el *Plan de Actividades de Investigación* fueron los siguientes:

3.1. Objetivo General

Diseñar e implementar una placa de desarrollo para nodos LoRaWAN, cuya labor será principalmente adquirir, procesar y transmitir la información de los agentes influyentes en la calidad del aire, adquirida por diversos sensores digitales disponibles.

3.2. Objetivos Específicos

Se presenta a continuación, la evaluación de cada uno de los objetivos específicos propuestos, para lograr el objetivo general citado anteriormente:

3.2.1. Documentación de información

Documentar la información relevante tanto de las generalidades de LoRa y LoRaWAN, como de las particularidades para los elementos disponibles, en relación al nodo.

Cumplido: Toda la información mencionada en la Sección 1.3 que no haya quedado en los Apéndices, se encuentra disponible en el CD anexo al presente informe.

3.2.2. Librerías de Software

Programar librerías de software para la adquisición de medidas de mínimo tres de los sensores disponibles.

De los sensores mencionados en la Sección 1, aquellos en que se verificó el funcionamiento esencial de sus librerías son los siguientes:

- BMP280 (`bmp280.h` y `bmp280.c`)
- BH1750FVI (`bh1750fvi.h` y `bh1750fvi.c`)
- iAQ Core (`iaq.h` y `iaq.c`)
- T6713 (`T67xx.h` y `T67xx.c`)

Adicional a las librerías anteriores, que también se encuentran en el mismo repositorio en GitHub del proyecto[30], y de las cuales se dejó un breve ejemplo de funcionamiento en el Apéndice C, cumpliendo así la cantidad mínima de librerías propuestas, también quedó adelantado gran parte del trabajo para llegar a las librerías de los demás sensores, quedaron estas pendientes de verificación de funcionalidad esencial.

3.2.3. Librería para control del módem LoRaWAN

Programar librerías que permitan el uso del módem LoRaWAN, para el envío de los datos adquiridos por los sensores.

Dado el uso de la pila HCI, en efecto se segmentó en varias librerías:

- Aplicación WSN (`pylatex.h` y `pylatex.c`).
- API LoRaWAN (`WiMOD_LoRaWAN_API.h` y `WiMOD_LoRaWAN_API.c`)
- Pila HCI (`hci_stack.h` y `hci_stack.c`)

La librería para una aplicación WSN (`pylatex.h`) se basa en la implementación de la API LoRaWAN (`WiMOD_LoRaWAN_API.h`) creada para los PIC (`WMLW_API.c`), la cual tiene funciones básicas para transmisión y recepción de octetos sobre UART, pero no hay funciones en esta misma API que permitan configurar directamente los parámetros de autenticación en la red, por lo cual es necesario haber realizado dicha configuración mediante uno de los programas brindados por IMST como el *EndNode Studio* o el *DevTool*[14]. El Apéndice B es evidencia del cumplimiento de este objetivo.

3.2.4. Diseño e Implementación de la Placa de Desarrollo

Diseñar e implementar la placa de desarrollo.

Cumplido: No solamente se limitó al diseño e implementación del prototipo de la Figura 1.7, sino que adicionalmente se siguió revisando hasta llegar a otro modelo apto para producción en masa, como el ilustrado en la Figura 1.9.

3.2.5. Pruebas de Integración y Funcionamiento

Realizar pruebas finales del dispositivo, particularmente en su integración con los sensores y su funcionamiento en la red LoRaWAN.

Cumplido: En conjunto al desarrollo del nodo, la infraestructura de red también se ha desarrollado en implementaciones[47, 48] del servidor de red LoRaWAN y un servidor para analítica de datos, como lo son LoRa Server [49] y Grafana [50].

Capítulo 4

Conclusiones y Recomendaciones

Una mayor densidad de integración, o *miniaturización*; implicará el uso de, tanto componentes superficiales (sección 1), como de más técnicas de diseño de PCBs, incrementando consecuentemente los costes de producción. Pero estos incrementos se pueden apaciguar, bien aumentando las distancias mínimas en las reglas de diseño, como aumentando la cantidad de PCBs a fabricar. Esto último es conveniente si se desea thru-hole, que suele realizarse bajo una cantidad mínima de varias unidades, cuyo precio unitario decrementa dramáticamente con la cantidad: Un proveedor determinado, para 4 unidades de cualquiera de las placas, realiza el trabajo a $23 USD por unidad, pero el mismo proveedor para 20 unidades entrega a $5 USD cada unidad, implicando esto una reducción de hasta el 78 % del coste unitario, haciendo viable la entrada del producto a una etapa de producción. Lo anterior se evidenció precisamente en la labor realizada en el laboratorio de fabricación de PCBs en la Facultad de Ingenierías de la UDFJC; para lograr el prototipo de la Figura 1.7, pues se requirieron varios intentos para llegar a la PCB final, dado que los espaciamientos entre pistas llevaron varias veces a la máquina a sus límites de resolución, obligando incluso a realizar pequeñas reparaciones locativas en las PCB.

Siempre que sean necesarios thru-hole en el diseño, conexiones entre ambas caras de cobre de una PCB; y más si estos son realizados de manera artesanal; es importante soldar las conexiones, sin importar si ya con un multímetro se presenta continuidad entre las pistas de ambas capas, y tras ello, realizar una revisión minuciosa tanto en pruebas de conductividad y aislamiento necesarias, previo al inicio del ensamble del circuito sobre la PCB, dado que si se considera agregar máscara anti-solder a la PCB, una conexión aparentemente firme puede quedar totalmente aislada al aplicarse.

Realizar los *breakouts* de la sección 1.1.1 a una sola capa de cobre es posible, pero se deben llevar con cuidado dado que al ser pensados para pruebas, es seguro su inserción y remoción frecuente de la *protoboard*, sometiendo a estrés mecánico sobretodo a los pines de extremos, como se nota ligeramente en el breakout del HDC1010, Figura 1.6. De esto se recomienda aprovechar producción en masa y realizarlos a dos caras de cobre, con thru-hole; y adicionalmente soldar conectores de tipo hembra, cuyo encapsulado distribuye cualquier esfuerzo mecánico hacia los pines cercanos, reduciendo aún más la posibilidad de que las pistas se levanten y se llegue a la necesidad de reforzarlas, como se hizo en la Figura 1.1.

Siempre al momento de importar componentes, y sobretodo si se piensan llevar a una solución

comercial, es conveniente hacer una revisión exhaustiva de sus requerimientos eléctricos, mecánicos y de accesorios adicionales requeridos: El sensor HPMA115S0 requería un conector específico, que si bien aparecía en su hoja de datos, al momento de su compra se omitió pensando se conseguiría fácilmente, pero fue necesario realizar otra compra adicional, dejando estancado el desarrollo de este sensor.

Para la frecuencia de 868 MHz, el haber empleado una línea microstrip, hubiese elevado considerablemente el grosor de la pista: seleccionandola en la interfaz de la Figura 1.2, el grosor de la pista se incrementó de 0.025 pulgadas de la guía de ondas coplanar, a 0.115 pulgadas, siendo esto aún mas grueso que el ancho de los *pads* recomendados por IMST para el iM880B-L [14].

4.1. Criterios de diseño del módulo de Aplicación

En el desarrollo de aplicaciones arbitrarias, orientadas principalmente al IoT, posiblemente se tenga la situación de administrar la energía de una cantidad diversa de sensores y actuadores, sin estar limitado por alguno de estos voltajes.

El *núcleo* se independiza del módulo de aplicación; para desligar su polarización de un nivel particular de voltaje, dejándolo al criterio del diseñador, siempre y cuándo se encuentre en su rango de voltajes de operación. En otras palabras, el núcleo no tendrá una fuente regulada de voltaje: La *aplicación*, o circuito(s) externo(s) al *núcleo*, ha(n) de cubrir las siguientes funciones:

- *Administración de energía*. Cada aplicación tendrá diversas necesidades (consumos continuos, pulsados, condicionales, distintos niveles de voltaje y corriente, entre otros) y disponibilidades de energía (baterías, celdas solares, generadores, fuentes DC no reguladas, generadores a gasolina, etc.), demandando así únicamente del diseñador electrónico, una fuente de voltaje entre 2.5 V y 3.6 V, rango de operación común tanto al PIC como al módem [15, 14], que al respetarse, puede ser usado como una medida indirecta del nivel de batería del sistema, dado que el iM880B-L incluye un comando para obtener su nivel de polarización en mV. Pero si es inminente polarizar al iM88xx de una fuente regulada, haciendo imposible esta opción, se puede aprovechar una interfaz digital, puede ser mediante el mismo I^2C; para brindar de alguna manera al sistema de una lectura del nivel de batería, u otras señales de energía, como la disponibilidad de las distintas fuentes de energía, entre otros, y también de la circuitería necesaria para que el *núcleo* pueda deducir esto, son algunas de las posibilidades.
- *Polarización del Sistema*. Integrar varios sensores en una misma placa de aplicación, requerirá disponer en esta todos los voltajes y elementos activos y pasivos necesarios para un buen funcionamiento de cada elemento, bajo un consumo mínimo. Cada fabricante suele brindar valores de capacitancias recomendados para el desacople del voltaje de la fuente de alimentación y de otras señales, usualmente bastante cerca al dispositivo, pero también serán necesarios estos y otros elementos pasivos y activos para completar las necesidades de la aplicación: Cadenas de medida completas para elementos sensores de salida análoga y/o por variación de alguno de sus parámetros físicos, circuitos de apagado y encendido para dispositivos sin opción de inhabilitación, entre otras consideraciones de cada elemento y por tanto, de la aplicación.
- *Acondicionamiento de Señales*. El separar la aplicación del núcleo tiene la consecuencia de que toda señal externa al nodo ha de ser llevada a un nivel de voltaje en el rango de 0 V y el voltaje

de polarización suministrado al nodo.

Con lo anterior en mente, es innecesaria una PCB en fase de desarrollo: Con un conector apto para soldar regletas de pines que permiten usar *jumpers*, cables para montaje en *protoboard*, del mismo modo que lo tienen otras placas de desarrollo como el Arduino, la Raspberry Pi, la Launchpad, entre otras; se puede empezar implementando el circuito en *protoboard* para irlo refinando sucesivamente hasta llegar a un prototipo de PCB, que puede refinarse aún más y llegar a un diseño final apto para fase de producción.

4.2. Trabajo Futuro

El *núcleo* integra dos componentes principales: un μC y el modem LoRaWAN. Pero este último presenta una interfaz de programación para su μC interno, de referencias STM32L151C8 y STM32L051C8 para el iM880B[14] e iM881A[20], respectivamente. Dicha interfaz de programación es para la conexión de un STLink, el programador de ST que emplea la interfaz SWD para programar esa familia de μC, análogo a la brindada para ser usada por el programador PicKit 3 de Microchip mediante la interfaz ICSP.

Estos módem vienen configurados de fábrica como tal, pero el desarrollar sobre su μC liberaría el espacio en memoria FLASH ocupado por el código asociado a la pila HCI, y por consecuencia, el módulo UART, permitiendo esto incluso emplear para comunicarse con el PIC embebido en la placa y usarlo como extensión de puertos y periféricos, convirtiéndose así en un dispositivo esclavo del iM88xx.

El enfoque mas cercano a esto en la actualidad es el esfuerzo hecho por StackForce para el firmware[51] del Mote II[45] de Semtech e IMST, que emplea al μC del iM881A ya embebido, pero tiene precisamente el inconveniente mencionado en la sección 1.1 de que este dispositivo quedó limitado a una aplicación en particular: mostrar las bondades de LoRaWAN. Se plantean dos alternativas para lograr un desarrollo que no implique partir desde cero para desarrollar sobre el mismo μC, llevando a una disminución de recursos:

- Conseguir un STLink y emplear los terminales para la conexión SWD en el breakout del iM880 para programarlo.
- Emplear el bootloader para actualizar el *firmware* (FW) sin depender del STlink, empleando el UART del iM881A.

Por otro lado, para la programación del dispositivo, se podría extender la funcionalidad del conector micro USB de la placa de aplicación (sección 1.1.3), para que sirva también como programador por USB del μC del *nodo*, o como interfáz de depuración/programación JTAG, y no depender del PicKit u otro programador, a costa de elevar la complejidad y costo de la placa de aplicación.

Aprovechando que las librerías desarrolladas poseen poca o nula dependencia de la plataforma, se pueden emplear para crear una interfáz gráfica de usuario, *Graphical User Interface* (GUI) de configuración y depuración, alternativa a los sw. *EndNode Studio* y *DevTool* de IMST, mencionados en el Apéndice A; independiente a la plataforma.

Apéndice A

Control de Versiones con Git

Como estudiante y en general, al realizar proyectos de software de cualquier tipo, se pueden presentar las siguientes situaciones:

- Se creó un programa funcional al que posteriormente se le hicieron modificaciones por cualquier razón, pero no se guardó copia de los archivos empleados para llegar a la versión inicial del programa.

- En caso que haya pasado lo anterior, quizás se hayan podido deshacer las acciones y así llegar a la versión original, pero ya el programa original no funciona porque de pronto falta algún archivo por revertir, y si funciona, entonces también puede darse que no se hayan guardado las modificaciones realizadas.

- En un proyecto grande de software aparecerá el trabajo en equipo, y después de haber separado labores, una persona de pronto llevó labores de otra persona, presentándose así conflictos al momento de combinar los dos o más trabajos.

- Se borró accidentalmente todo el código, debido a algún mantenimiento del sistema o a algún ataque.

- En caso que se haya intentado llevar un control de versiones básico, tal como guardar una copia de la carpeta, y nombrarla con “version 1”, “version 2”, “version sin X”, “version con Y”, “version 2018”, “version2018_2”, “esta si fue”, “final”, “final final” y cualquier otra idea similar de nombre; estos nombres no brindan una idea de qué fue lo que puntualmente cambió, para haber requerido guardar cada una de estas versiones, información que, combinada junto a la auto-documentación [39]; podría ser importante al momento de intentar retomar un proyecto abandonado.

Estos son algunos de los inconvenientes que se resuelven empleando un Sistema de Control de Versiones, *Version Control System* (SCV). A continuación se presenta una breve descripción del como usar Git como SCV para un proyecto [52].

A.1. Git: SCM de Código Abierto

Git es un SCV distribuido, Libre y de Código Abierto, diseñado para manejar cualquier cosa, de proyectos pequeños a grandes, privados y abiertos, con velocidad y eficiencia.[29].

Para empezar a llevar el control de versiones de un proyecto, o trabajar en un proyecto con control de versiones en Git ya existente, el único prerrequisito es tener instalado el cliente Git para la plataforma en que se encuentre trabajando, desde la página de descargas de Git-scm, Figura A.1, o para plataformas Linux, generalmente basta con descargarlo del gestor de paquetes del sistema.

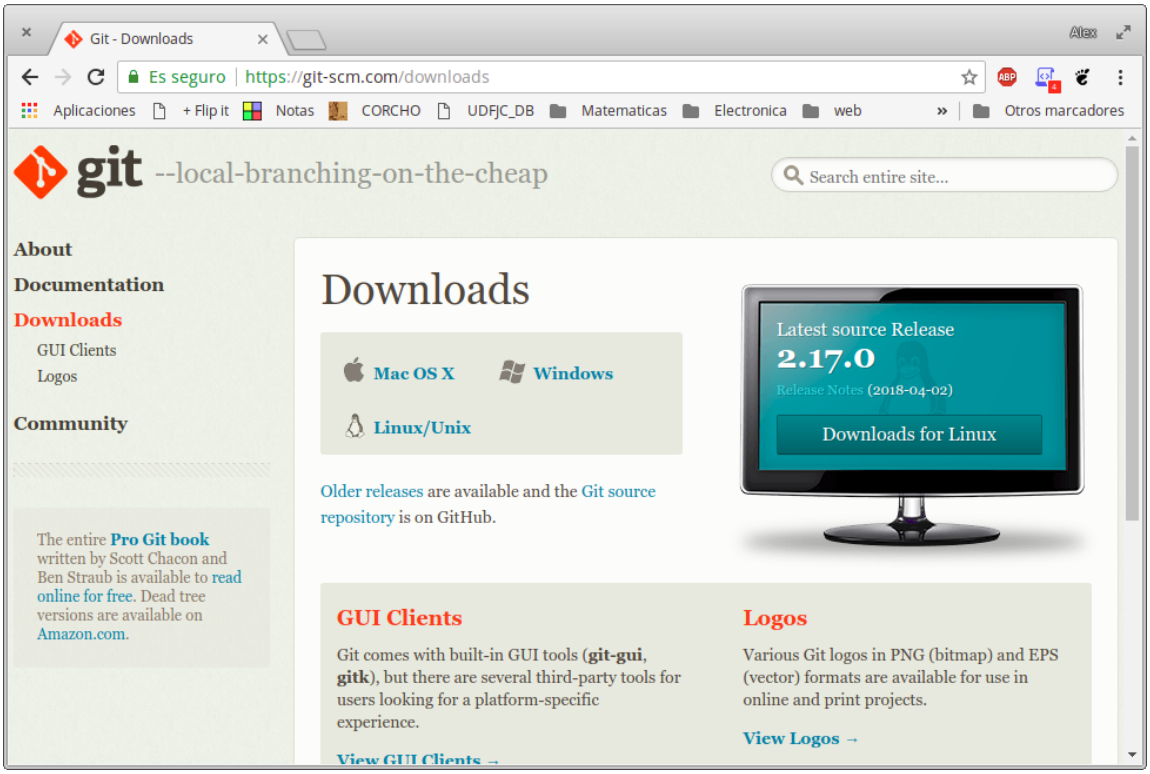


Figura A.1: Página de descarga de Git

El cliente de Git es un programa que se ejecuta en una consola, pero también se puede instalar una GUI adicional para facilitar la labor [1]. Algunos IDEs tienen soporte nativo para este SCV [2]. Para *Windows*, el cliente Git como terminal se puede lanzar en casi cualquier ubicación desde el explorador de *Windows*, con el menú contextual, Figura A.2. En sistemas base Linux, basta con haberlo instalado desde los repositorios oficiales y lanzar una terminal, también en el mismo lugar en que se desea iniciar o *clonar* un SCV ya existente.

[1] En `https://git-scm.com/downloads/guis` se encuentran varias

[2] `https://code.visualstudio.com/` y `https://atom.io/` son algunos IDEs multiplataforma con soporte Git

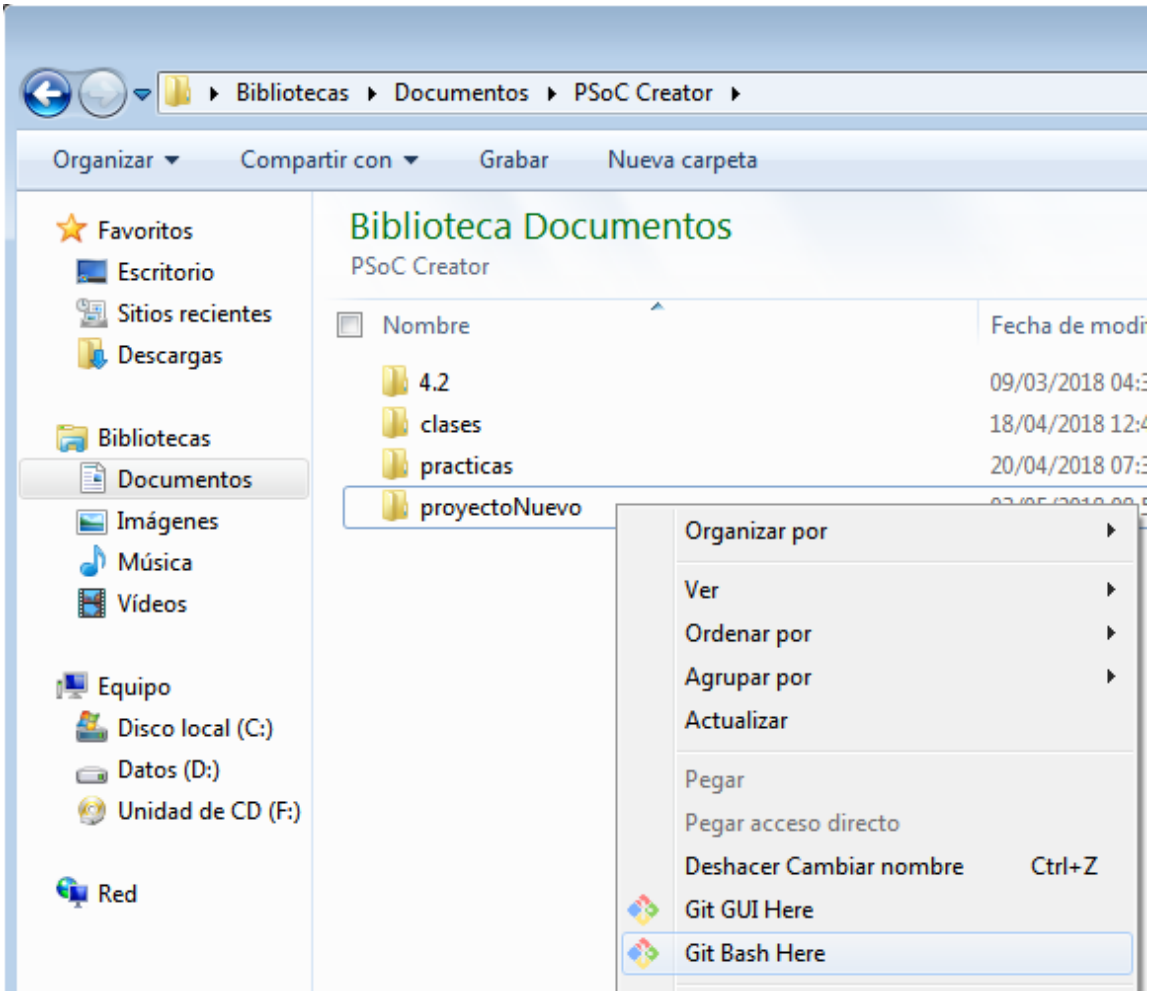


Figura A.2: Apertura en contexto del terminal con soporte Git en Windows

A.2. Estado de un Repositorio

Una vez abierto el terminal, dependiendo la ubicación en que se haya lanzado, puede que haya o no disponibilidad de Git (Figuras A.3 y A.4), y si lo hay, hay múltiples opciones y acciones asociadas al estado del repositorio local. Se puede dar luz de esto con:

```
1 git status
```

A pesar de que es el comando que menos hace, su función es justamente brindar información acerca del estado del repositorio y, adicional a esto, en cada estado sugiere algunas de las acciones que se pueden tomar. Se recomienda realizar antes y/o después de cualquiera de los comandos que se tratarán mas adelante.

```
alex@alex-X450CC:~$ git status
fatal: Not a git repository (or any parent up to mount point /home)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
alex@alex-X450CC:~$
```

Figura A.3: Salida de `git status` sin Git

```
alex@alex-X450CC:~/Documentos/PROYECTOS/pylates/SoftwareLibs$ git status
En la rama master
Su rama está actualizada con «origin/master».
nothing to commit, working directory clean
alex@alex-X450CC:~/Documentos/PROYECTOS/pylates/SoftwareLibs$
```

Figura A.4: Salida de `git status` con Git

A.3. Creación/Clonado de un Repositorio

Para crear un repositorio nuevo, en el terminal se ingresa este comando:

```
1 git init
```

Esto crea una carpeta oculta, ".git" en el lugar en que se haya lanzado este comando, que se recomienda mantener así, oculta, y no modificar manualmente, a no ser que se tenga absoluta certeza de lo que se esté haciendo, dado que esta lleva el historial de todas y cada una de las versiones de nuestro proyecto.

Si por otro lado, ya hay un proyecto existente con control de versiones o *repositorio* que interesa obtener, la acción a realizar es el *clonado* del repositorio, requiriéndose en este caso la URL del repositorio. Para clonar el proyecto de código fuente tratado en este documento[30]:

```
1 git clone https://github.com/pylatex/im880b.git
```

A.4. El flujo de trabajo con Git

El repositorio (local) se compone de tres "árboles" administrados por Git:

- El directorio de trabajo, que contiene los archivos
- El índice o *stage*, que actúa como una zona intermedia, como una "sala de espera"
- el HEAD, que referencia al último *commit* realizado.

Los *commit* son cada una de las versiones, ya almacenadas en el repositorio.

A.5. Agregado y Registro de Cambios

Para indicarle a Git que empiece a registrar cambios (llevarlos al *stage*) de archivos determinados:

```
1 git add ARCHIVO
```

La especificación del archivo soporta comodines. Por ejemplo, el comodín asterisco valida elementos con cualquier contenido en que se encuentre ubicado:

- Para agregar todos los códigos fuente, que para C su extensión suele ser .c:

```
1 git add *.c
```

- Para agregar todos los archivos que empiecen y terminen con algo específico:

```
1 git add imagen*.jpeg
```

Una vez agregados todos los archivos necesarios al *stage*, se crea una captura o *commit* de estos archivos del *stage* con:

```
1 git commit -m 'RESUMEN CONCISO QUE MOTIVO EL COMMIT'
```

Si un *commit* involucró gran cantidad de cambios, omitiendo la opción `-m` y el mensaje asociado a esta, se abre un editor de texto plano para ingresar, por convención en la primer línea, este mensaje del commit, y en las siguientes, un recuento tan detallado como se quiera. Pero el editor de texto no abre en blanco sino con un texto muy similar a la salida de `git status` como comentarios que empiezan con '# ' y no se agregan al *commit* precisamente por ser comentarios. El *commit* tendrá efecto una vez guardado el mensaje y cerrado el editor.

En este punto el archivo se encuentra ya en el HEAD, pero no en un repositorio remoto. Para llevarlos a un repositorio remoto:

```
1 git push origin master
```

Donde `origin` es el apodo por defecto para el repositorio remoto (en caso de haber uno, se pueden configurar varios servidores remotos) y `master` suele ser la *rama* principal, *maestra*, del proyecto; pero con otro nombre se pueden subir los cambios a otra rama, representando cada *rama* un punto en que el repositorio diverge, se bifurca.

Ahora, para un repositorio local no clonado, sino iniciado; primero habrá que indicarle el servidor en que se encuentra el repositorio remoto:

```
1 git remote add origin SERVIDOR
```

`SERVIDOR` es la URL entregada por un servidor como GitHub, BitBucket, GitLab, SourceForge, u otro servidor remoto. Si es válida, a partir de este punto se podrán realizar *push* al repositorio, como se había intentado anteriormente, en caso que haya generado algún error. La sintaxis anterior también indica que al servidor remoto, *remote*; se le denominará localmente como *origin*.

A.6. Ramas

Las ramas permiten desarrollar funcionalidades aisladas. Como se mencionó anteriormente, la rama *master* es la rama por defecto al inicializar un repositorio. Se pueden crear y cambiarse a ellas en un solo comando:

```
1 git checkout -b RAMA
```

Para volver a la rama principal, *master*:

```
1 git checkout master
```

Todos los cambios son locales, incluyendo *ramas*, *commit*, entre otros; a no ser que se publiquen a las demás personas con acceso al repositorio remoto, se suban (*push*) a este. Para subir una rama al repositorio:

```
1 git push origin RAMA
```

A.7. Actualización y Fusión

Para trabajo en conjunto a más personas, seguramente ellas ya habrán realizado y publicado sus cambios (*commit* y *push*). Para obtener esos nuevos cambios:

```
1 git pull
```

Si el cambio es local, por ejemplo, el realizado en una rama:

```
1 git merge RAMA
```

En cualquiera de los dos casos, *merge* o *pull*, Git intentará fusionar automáticamente los cambios, y de no poder hacerlo, se presentarán *conflictos* que han de ser arreglados manualmente antes de poder seguir trabajando. Una vez modificados, se ha de "desmarcar" su conflicto volviendo a agregarlos (*add*) al repositorio:

```
1 git add ARCHIVO_CONFLICTIVO_MODIFICADO
```

Estos cambios se pueden revisar usando

```
1 git diff RAMA_ORIGINAL RAMA_OBJETIVO
```

A.8. Etiquetas

Es recomendable asignar una etiqueta a cada versión publicada de un software (refiriéndose, por ejemplo, a una versión estable y lista para su distribución). Por ejemplo, para agregar la etiqueta, *tag* "1.0.0":

```
git tag 1.0.0 1b2e1d63ff
```

Donde "1b2e1d63ff" son los diez primeros carácteres del IDentificador (ID) del commit al cual se le quiere asignar la etiqueta. Para obtener este ID:

```
git log
```

Se pueden usar menos de estos caracteres, pero este valor ha de ser único entre todos los commit.

A.9. Reemplazo de cambios locales

Si llegase a presentarse la situación de haber realizado alguna equivocación, se pueden descartar los cambios de archivos locales con:

```
git checkout -- ARCHIVO
```

Esto reemplaza al archivo con su versión en HEAD. Los demás archivos, como los agregados al *stage* y también los nuevos, se mantendrán intactos.

Si se requiere algo más drástico, como descartar todos los cambios y *commits* locales, se puede traer la última versión del servidor remoto y hacer que el local apunte a esta:

```
git fetch origin
git reset --hard origin/master
```

A.10. Resumen

Tabla A.1: Resumen de comandos Git

Comando	Descripción
COMANDO --help	Muestra una ayuda de COMANDO
git status	Informa del estado del control de versiones.
git init	Inicializa un repositorio
git add	Agrega archivos al *stage*
git commit	Crea un commit con los archivos del *stage*
git branch	Crea una rama
git checkout	Se cambia a una rama o commit determinados
git merge	Toma los cambios de una rama indicada
git push	Publica cambios a un servidor remoto
git pull	Toma los últimos cambios del servidor remoto

Apéndice B

Nodos LoRaWAN

B.1. Introducción

En el campo de las comunicaciones móviles, al *dispositivo final*, o *end-device* de la red –nombre debido a que hace parte de la *interfaz inalámbrica* de la red, o de la *red de acceso*– se le denomina *nodo*. En redes de telefonía celular, el *nodo* de la red es el *teléfono celular* o simplemente *celular*; y en la jerga de la comunidad LoRaWAN al *nodo* se le conoce como *mote*. Ambas WAN mencionadas siguen un modelo de red de tipo cliente/servidor (C/S), en que los *nodos* son los clientes, sin necesidad de comunicación directa entre ellos [53], gracias a la existencia del servidor LoRaWAN. El servidor LoRaWAN se encuentra en Internet, implementado de manera monolítica o distribuida; y su labor principal es recibir y responder los mensajes de la red LoRaWAN a través de los *gateways* LoRaWAN, dispositivos que median entre la interfaz inalámbrica, que da soporte a LoRaWAN mediante las modulaciones LoRa o GFSK [54, 4]; y la interfaz IP.

B.1.1. Activación en la red

Análogo al concepto de *dirección IP* en el modelo TCP/IP [55], cada mensaje ha de estar asociado a un *nodo*, mediante una dirección LoRaWAN. La *activación* es el proceso en que se asigna esta dirección al *nodo*. Dicha activación puede ser manual mediante la activación por personalización, *Activation By Personalization* (ABP), o gestionada por el servidor LoRaWAN mediante la activación al aire, *Over The Air Activation* (OTAA). Una vez activo, podrá enviar mensajes LoRaWAN confirmados o no confirmados (análogos a TCP y UDP) [1].

Sin importar el método de activación, en el *nodo* han de estar almacenados los parámetros necesarios para lograr esta activación. Estos parámetros suelen obtenerse ingresando en alguna interfaz provista por el servidor LoRaWAN, el Identificador del Nodo, *End-User Identifier* (EUI) del *nodo*, asignado de fábrica, análogo al IMEI en los celulares. Para los módem iM88xx, IMST brinda el sw. *EndNode Studio* como interfaz de configuración brindada para su familia de productos WiMOD [14]. Una captura de este sw. en la Figura B.1, ilustra los parámetros necesarios según el modo de activación.

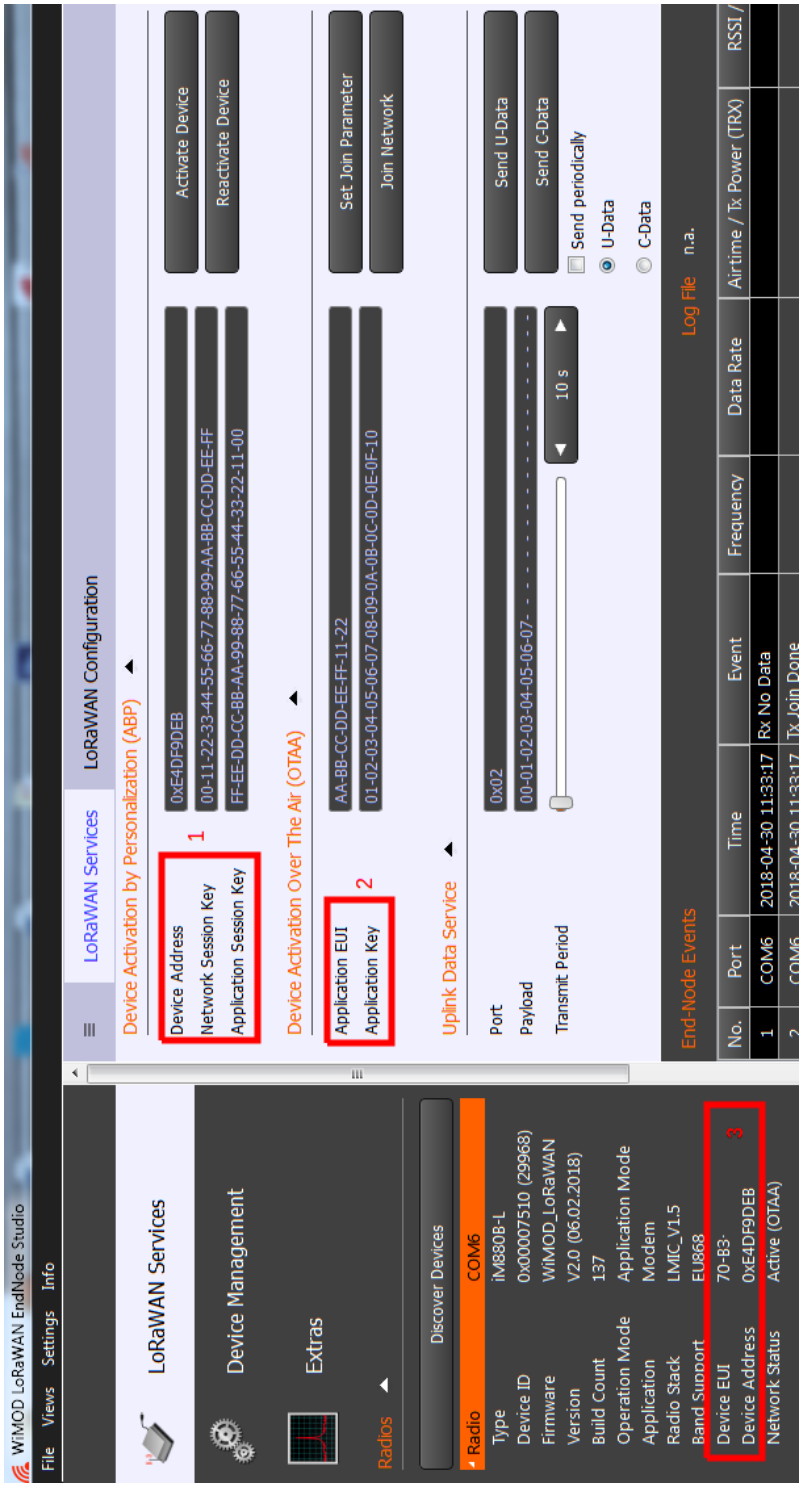


Figura B.1: Interfaz del programa WiMOD LoRaWAN EndNode Studio de IMST. 1. Parámetros por ABP. 2. Parámetros por OTAA. 3. EUI a dar para obtener los parámetros de activación. Notese que el nodo se encuentra activo por OTAA (*Network Status: Active (OTAA)*) pero la dirección del dispositivo (*Device Address: 0xE4...*) es la misma del cuadro 1.

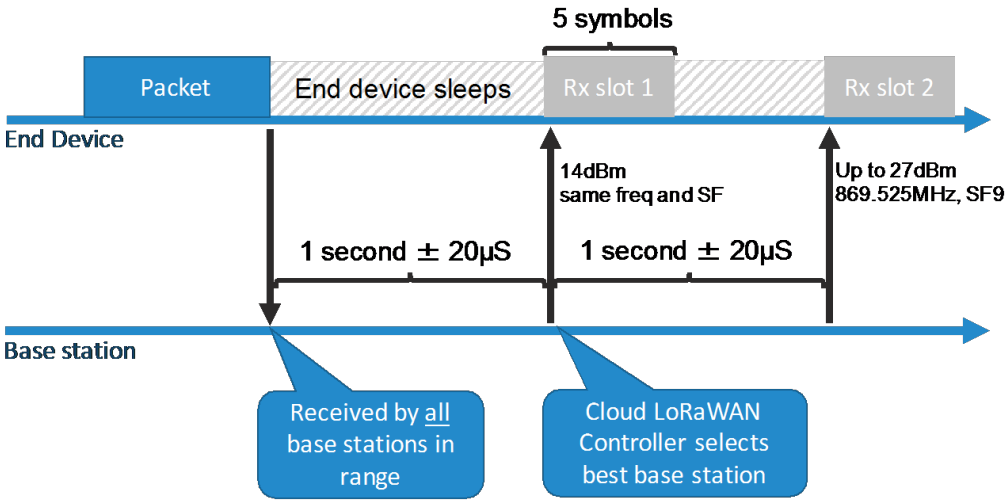


Figura B.2: Perfil de Configuración por defecto para la Clase A. Tomado de [5].

B.1.2. Intercambio de mensajes

En la Figura B.1, en la sección *"Uplink Data Service"*; se observa también el único parámetro requerido para enviar un mensaje sobre la red: el *puerto de destino* (del servidor), el cual es un número de 0 a 255, siendo el 0 reservado para mensajes de Control de Acceso al Medio, *Media Access Control* (MAC) [54]. Pero el *quién* puede enviar mensajes y el *cómo* se envían los mensajes; lo determina el modo o *clase* de operación, que puede ser A, B o C; de las cuales solo se menciona la Clase A, dado que es la de más bajo consumo, y por tanto, de implementación obligatoria por parte de un módem LoRaWAN [54].

En Clase A, el *nodo* siempre inicia la comunicación con un mensaje dirigido al servidor LoRaWAN, el cual a su vez solamente puede responder en una de las dos ventanas de tiempo que da el nodo, espaciadas entre sí aproximadamente un segundo, proceso que se ilustra en la Figura B.2. En caso de haber en el servidor algún mensaje pendiente para el nodo, se anexa al mensaje con un indicador adicional de más mensajes pendientes, o simplemente se confirma la recepción del mensaje, en caso de ser necesario. Por este comportamiento, la clase A es adecuada para el envío de datos sencillos como los de una WSN, pero inadecuada para la transmisión en tiempo real de vídeo, voz e inclusive imágenes.

B.1.3. Configuración sugerida para un nodo de una WSN

La siguiente es una serie de consideraciones adecuadas para una WSN, que podría ser tomada y/o adaptada a otras aplicaciones:

- Activar el dispositivo por OTAA, para darle certeza al nodo de que sus mensajes llegarán al servidor mediante una dirección válida en la red de la cual sus mensajes no serán descartados.

- Una vez activado, enviar la información de cada uno de los sensores mediante mensajes no confirmados, dado que la confirmación de un mensaje puede demorarse demasiado, sobretodo en las tasas de bit más bajas soportadas, obligando al dispositivo a descartar datos, mientras espera la confirmación de un mensaje arbitrario.

- Ocasionalmente, se puede enviar un mensaje confirmado para verificar la conexión con el servidor LoRaWAN.

B.2. Librerías

En la sección de descargas de los módem LoRaWAN iM88xx de IMST [14, 20], se encuentran los programas *EndNode Studio*, Figura B.1; y el *DevTool*. Ambos programas, creados para el sistema operativo *Windows*, de *Microsoft*; controlan la interfaz UART del PC, para comunicarse mediante mensajes HCI con un módem LoRaWAN iM88xx conectado al PC mediante un conversor serial-usb comercial. El *EndNode Studio* presenta una GUI para la configuración de los parámetros y uso básico de su funcionalidad como módem, y el *DevTool* es una herramienta de desarrollo que permite depurar las labores de codificación y decodificación de mensajes HCI.

Con el fin de reemplazar al PC con un μC, que pueda ser empleado en aplicaciones embebidas, se alternó la conexión UART del PC original, entre el módem y el μC, para obtener una serie de *librerías* que un desarrollador pueda emplear para controlar los módem iM88xx con un μC. Dado que queda fuera del alcance del libro la explicación de este y otros conceptos de programación en C o C++, esenciales para tener total claridad del contenido de las secciones siguientes, tales como los apuntadores, punteros a funciones, prototipos de funciones, estructuras, uniones, uso y llamado de librerías, entre otros; se recomienda encarecidamente al lector realizar una breve consulta de los conceptos mencionados, en caso de no tener claridad de estos, a pesar de que se tratará de hacer una breve introducción a estos, donde sea necesario.

Para facilitar el desarrollo y puesta en marcha de *nodos* LoRaWAN, se creó una serie de *librerías* escritas en lenguaje C, partiendo de una implementación inicial en μCs PIC de Microchip, programadas de tal manera que haya una mínima o nula dependencia en esta familia y fabricante de μCs. Tras haber clonado el repositorio con las librerías creadas (véase el Apéndice A), y luego definir la carpeta `src` del repositorio clonado, como "`PATH`" del proyecto; el desarrollador puede *incluir* al comienzo (generalmente) de sus códigos en C o C++, las funciones de la librería, con la siguiente sentencia de preprocesador:

```
1 #include "LIBRERIA.h"
```

En las librerías, los archivos `.h` son la API del usuario: exponen prototipos de funciones, definiciones de tipos, constantes, entre otros; a los archivos `.c` que los incluyan con la sentencia anterior. De hecho al abrir estos archivos `.h` en un editor de texto plano o en el IDE, se puede consultar la documentación asociada a cada prototipo de función en formato *javadoc* [41, 40], y declaraciones de constantes y definiciones de tipos asociadas. La implementación de estas funciones declaradas en los archivos `.h` puede venir, o bien con el código fuente original, en archivos de lenguaje C(`.c`) o C++(`.cpp`); o compiladas para la plataforma *objetivo* en archivos `.o`.

Tabla B.1: Resumen de funciones de `pylatex.h`

Comando	Descripción
initLW()	Inicializa la aplicación LoRaWAN para el envío de datos de sensores, y registra la función a usar para enviar un octeto por UART a 115200 8N1, hacia el iM88xx
AppendMeasure()	Anexa una medida a la cola de envío
SendMeasures()	Envía las medidas en la cola de envío y libera la cola.
short2charp()	Emplear como segundo argumento de la función anterior: Recibe un entero de 16 bit y lo separa en un arreglo de dos bytes, donde el elemento cero es el de mayor peso. Devuelve la dirección de este arreglo.
clearAppPayload()	Libera la cola. Llamado automáticamente por initLoraApp() y SendMeasures().
pylatexRx()	Colocar cada octeto recibido por UART en esta función.

B.2.1. `pylatex.h`

Es una librería de funciones pensadas para una WSN que, en vez de enviar los datos de sensores en formato ASCII y/o en múltiples mensajes, intenta empaquetar y codificar los datos de manera binaria en el mismo mensaje, de manera similar al formato de Carga Útil de Baja Potencia, *Low Power Payload* (LPP) de Cayenne [56]. Esta codificación es adecuada para la baja rata de mensajes de la Clase A. En la Tabla B.1 se listan sus funciones, cuya implementación depende a su vez de las librerías `"WiMOD_LoRaWAN_API.h"` y `"hci_stack.h"`, a ser tratadas posteriormente; y de la librería estándar `<string.h>`.

La librería es independiente a la plataforma, dado que no controla directamente al UART. Solo depende de registrar una función encargada de enviar un arreglo por UART, ingresándola como argumento de la función `initLoraApp()`. Por otro lado, cada octeto recibido por la misma interfaz UART ha de ser ingresado como argumento de la función `pylatexRx()`. El único detalle a cuidar es externo: la función a registrar debe estar asociada al UART conectado al módem bajo 115200,8N1 [14, 20].

B.2.2. `WiMOD_LoRaWAN_API.h`

Esta librería es el soporte de la anterior. Permite enviar un mensaje arbitrario sobre LoRaWAN, con o sin confirmación y se encarga de la activación en la red, por lo cual se puede emplear para una aplicación distinta a la de una WSN o para tomarla como librería base para otra aplicación. Adaptada del código de ejemplo de la sección de descargas del iM880B-L [14], del cual las funciones implementadas se resumen en la Tabla B.2.

La implementación de la librería `pylatex.h` le pasa directamente a `Init()` el argumento que recibe su función `initLW()`.

Tabla B.2: Resumen de funciones de `WiMOD_LoRaWAN_API.h`

Comando WiMOD_LoRaWAN_...	Descripción
Init()	Registra la función para enviar mensajes al módem e intenta autenticar al nodo en la red.
SendPing()	Para verificar la conexión entre el μC y el módem
JoinNetworkRequest()	Solicita activación en la red
SendURadioData()	Envía un mensaje sin confirmación
SendCRadioData()	Envía un mensaje con confirmación
GetNetworkStatus()	Obtiene el estado de activación en la red
Process()	Llamar con cada byte recibido por UART

Tabla B.3: Resumen de funciones de `hci_stack.h`

Función	Descripción
InitHCI()	Inicializa el motor de la pila HCI. Adicionalmente registra la función a llamar una vez recibido un mensaje HCI.
SendHCI()	Envía un mensaje HCI (CRC+SLIP+UART)
IncomingHCIpacker()	Llamar en cada byte entrante: Compone un mensaje a partir de los bytes recibidos, y lanza la función registrada, una vez hay un mensaje HCI completo.

B.2.3. `hci_stack.h`

A su vez, esta librería soporta a la anterior: los mensajes de la API LoRaWAN no se pueden enviar directamente hacia el módem iM88xx, hay que primero anexarles una Comprobación de Redundancia Cíclica (CRC) y colocarlos en un *envoltorio*, tal como el protocolo SLIP (*Serial Line Internet Protocol*) (SLIP), para asegurar una mínima integridad de los mensajes intercambiados entre el μC y el módem mediante UART. Estos tres procesos los lleva la *pila HCI*, la cual, tal como viene implementada en el código de ejemplo para PC, mencionado anteriormente; implementa cada capa en una librería separada, haciendo que cada mensaje ocupe hasta tres veces su espacio en memoria antes de ser transmitido, lo que es impensable para una implementación en μC, con recursos escasos comparados con los de un PC, plataforma a la cuál se orientó el código de ejemplo. La librería `hci_stack.h`, cuyas funciones se resumen en la Tabla B.3, compacta los tres procesos: CRC, SLIP y UART, en uno solo.

B.3. Ejemplos de uso de las librerías

B.3.1. Envío de datos de sensores

Suponiendo que la librería `"LIBRERIA_SENSOR_CO2.h"`, dispone de la función `LecturaCO2()`, que hace todo lo necesario para adquirir una medida de CO_2 en ppm. como un entero de 16 bit sin signo; el código a continuación configura un PIC como el controlador de un *nodo* en una WSN, por tanto

emplea las funciones de la librería `pylatex.h`, mencionadas anteriormente, para enviar una medida cada 5 segundos.

```
#include <xc.h> //Recursos del PIC, expuestos por XC8

#include "pylatex.h"        //Libreria para una WSN
#include "SerialDevice.h"   //Acceso al Serial del uC
#include "LIBRERIA_SENSOR_CO2.h"    //Literal.

void main(void) {

    //Inicializacion del Sistema
    systemSetup();      //Entradas, salidas, otros
    SerialDevice_Open("",115200,8,0); //UART 115200 8N1
    enableInterrupts(); //Habilitar Interrupciones

    //Registro de la funcion que envia un arreglo hacia el modem
    initLoraApp((serialTransmitHandler)SerialDevice_SendData);

    /* ... */ //Inicializacion de Sensores

    //Programa Principal
    while (true) {
        //Lectura del valor
        AppendMeasure(PY_CO2,short2charp(LecturaCO2()));
        //Envio de mensaje no confirmado
        SendMeasures(PY_UNCONFIRMED);
        //Espera 5 segundos entre medidas
        __delay_ms(5000);
    }
}

//Rutina de Interrupciones
void __interrupt ISR (void) {
    //Eventos de recepcion en el modulo EUSART
    if (RCIE && RCIF) {
        char rx_err=RCSTA;//Lectura del registro de error
        pylatexRx(RCREG);//Leer RCREG implica borrar RCIF
    } else ; //Alguna Interrupcion no capturada
}
```

B.3.2. Envío directo de mensajes

Para trabajar a más bajo nivel que el brindado por la aplicación de la sección anterior, para enviar mensajes arbitrarios sobre LoRaWAN; se pueden emplear las funciones expuestas por

WiMOD_LoRaWAN_API.h:

```
#include <string.h>
#include "WiMOD_LoRaWAN_API.h"

void main (void) {
    const char puerto = 5; //Arbitrariamente
    WiMOD_LoRaWAN_Init((serialTransmitHandler)SerialDevice_SendData)
        ;

    //Envio sin confirmacion
    const char msgU[]="Mensaje␣sin␣Confirmacion";
    WiMOD_LoRaWAN_SendURadioData(puerto,msgU,strlen(msgU));

    demora_segundos(5); //Retardo entre mensajes

    //Envio con confirmacion
    const char msgC[]="Mensaje␣con␣Confirmacion";
    WiMOD_LoRaWAN_SendCRadioData(puerto,msgC,strlen(msgC));

    while(1)
        sleep(); //Mantener suspendido el microcontrolador
}
```

Apéndice C

Librerías para Lectura de Sensores

Los ejemplos a continuación, realizan una lectura de los valores de sensores por I^2C, y a modo de depuración, sus valores son enviados por el UART del μC, de tal manera que se pueden interpretar fácilmente con un programa de acceso al terminal por UART sin importar el sistema operativo. Adicional a ello, esta depuración depende de incluir al comienzo del código las librerías estándar `<string.h>` y `<stdio.h>`. La función `enviaMsgSerie()`, que puede venir de una librería o ser implementada por el usuario; representa a la función encargada de enviar un arreglo por UART.

Adicionalmente, los sensores dependen internamente de la implementación de una librería `i2c.h`, que permite abstraer todos los mecanísmos internos del μC para el control del bus I^2C de este, en modo maestro. Para el caso, esta librería se generó mediante el MCC de MPLAB-X IDE.

C.1. T6713: Sensor de CO2, de Amphenol Sensors

Este sensor viene de fábrica soldado con conectores para inserción directa en *protoboard*, lo que junto a su función de calibración automática, motivó a ser el primero para el cual crear sus librerías. Una vez incluida la librería `T67xx.h` en el archivo, un código de ejemplo que emplea sus funciones para obtener un entero sin signo de 16 bit con la medida de CO_2 en ppm. es la siguiente:

```
1  //Reserva un entero de 16 bit sin signo
2  unsigned short rsp;
3
4  //Realiza la lectura
5  if (T67xx_CO2(&rsp)) {  //T67xx_CO2 retorno "true":
6      //Leer y enviar por UART el valor de "rsp"
7      phlen=(char)sprintf(buff,"CO2: %u PPM\r\n",rsp);
8      enviaMsgSerie(buff,phlen);
9  } else {    //T67xx_CO2 retorno "false":
10     enviaMsgSerie((char *)"No hubo lectura\r\n",0);
11 }
```

Tabla C.1: Resumen de funciones de `bh1750fvi.h`

Función	Descripción
BHinit()	Carga la dirección a ser usada con el sensor. Su único argumento es el estado del pin ADDR.
BHwrite()	Retorna si hubo éxito o no, al intentar enviar un comando al sensor, como una OR bit a bit entre una resolución y un modo.
BHread()	Retorna si hubo éxito o no, al intentar leer el valor del sensor, y escribirlo en el entero de 16bit apuntado por el argumento.

C.2. BH1750FVI: Sensor de Iluminancia de ROHM Semiconductor

Como no siempre será conveniente colocar este sensor a la intemperie, sino mas bien bajo una carcasa que tendrá un índice de refracción determinado, que puede alterar la medida de iluminancia; posee instrucciones para que el mismo sensor realice la compensación adecuada a este índice de refracción, aparte de la simple lectura de sus mediciones, las cuales pueden ser únicas, dejando al sensor en estado de bajo consumo de energía una vez tomada; o continuas. Se resumen las funciones empleadas para trabajar este sensor en la Tabla C.1.

El ejemplo a continuación, depende de la librería `bh1750fvi.h` una vez incluida al comienzo del código fuente.

```
//Declaracion de un entero de 16 bit sin signo
unsigned short light;

//Inicializacion del sensor. DVI controla directamente
//al pin DVI del sensor
DVI=false;  // 0 V en DVI
__delay_us(5);
DVI=true;   // 3.3 V en DVI
BHinit(false);
BHwrite(BH1750_RESET);  //Reinicio del sensor
BHwrite(BH1750_PWR_ON); //Encendido
//Modo continuo, baja resolucion
BHwrite(BH1750_CONTINOUS | BH1750_LR);

//Intento de lectura del valor, indicando la direccion de "light"
if (BHread(&light)) {   //BHread retorno "true"
    //Se puede leer e imprimir "light":
    phlen=(char) sprintf(buff,"Luz: %u\n\r",light);
    enviaMsgSerie(buff,phlen);
} else {    //BHread retorno "false"
    //posible error de comunicacion y/o configuracion
    enviaMsgSerie((char *)"No hubo lectura\n\r",0);
```

```
23 }
```

Para el modo en que se configuró en el ejemplo (línea 12), de acuerdo a la hoja de datos, el valor obtenido ha de dividirse sobre 1.2 para obtener la lectura en lux.

C.3. iAQ Core: Sensor de CO2 y TVOC, de ams

El más grande de los sensores que quedaron en la placa de aplicación sugerida, tras limitarlos solamente a aquellos que pueden funcionar con 3.3 V; actualiza su lectura cada segundo, por tanto no tiene sentido leerlo en un intervalo menor. No posee registros que se puedan direccionar, su interfaz I^2C siempre se debe acceder en modo lectura[9], tras lo cual devuelve todos los parámetros posibles tanto de las variables que mide, como de su estado. Su librería `iaq.h` contiene apenas una función: `iaq_read()`, la cual recibe un puntero a una estructura de tipo `IAQ_T` (declarado también en la librería), y retorna un booleano que indica si se pueden leer o no los miembros de esta estructura apuntada, debido a que el sensor se haya podido leer o no. Después de incluir la librería `iaq.h` al comienzo del archivo, un ejemplo que realiza la lectura de ambos valores es el siguiente:

```
1  const char e_ok[]= "OK";
2  const char e_run[]="RUNIN";
3  const char e_busy[]="BUSY";
4  const char e_rr[]="ERROR";
5  char const *rspsens;
6  IAQ_T iaq;
7
8  enviaMsgSerie((char *)"iAQ␣Core␣-␣",0);
9  if (iaq_read(&iaq)) {   //iaq_read() retorno "true":
10
11     switch (iaq.status) {
12         case IAQ_OK:
13             rspsens=e_ok;
14             break;
15         case IAQ_RUNIN:
16             rspsens=e_run;
17             break;
18         case IAQ_BUSY:
19             rspsens=e_busy;
20             break;
21         default:
22         case IAQ_ERROR:
23             rspsens=e_rr;
24             break;
25     }
26     //Imprimir interpretacion de la trama
27     phlen=(char)sprintf(buff,"CO2:␣%u␣PPM,␣",iaq.pred);
28     enviaMsgSerie(buff,phlen);
```

```
    phlen=(char)sprintf(buff,"Estado: %s, ",rspsens);
    enviaMsgSerie(buff,phlen);
    phlen=(char)sprintf(buff,"Resistencia: %lu, ",iaq.resistance);
    enviaMsgSerie(buff,phlen);
    phlen=(char)sprintf(buff,"TVOC: %u PPB\n\r",iaq.tvoc);
    enviaMsgSerie(buff,phlen);
} else {    //iaq_read() retorno "false":
    enviaMsgSerie((char *)"No hubo lectura\r\n",0);
}
```

C.4. BMP280: Sensor de Presión y Temperatura de BOSCH

Este sensor presenta una particularidad: en sus registros no se encuentra una lectura directa de las variables que mide. En su lugar, se hace necesario leer de este una serie de constantes de calibración y variables, con las cuales se debe seguir una serie de pasos para obtener los valores [11]. Estas operaciones son la mayoría de punto flotante, que implementadas en un μC pueden consumir una gran cantidad de recursos de procesamiento. Tras haber incluido al comienzo del código, la librería `bmp280.h`, un segmento de código que realiza la lectura de los valores, y los prepara a ser enviados para su procesamiento en remoto, es el siguiente:

```
//Declaracion de espacio para los 30 registros y una auxiliar
char mem[30],i=0;
unsigned short val;
unsigned long val2;
//Inicializacion, con el estado logico del pin SDO
BMP280init(false);
//Modo continuo, muestreo estandar para temperatura y presion
BMP280writeCtlMeas(BMPnormalMode | BMPostX1 | BMPospX1);
//Lectura de los bancos de constantes y variables (hacia "mem")
if (BMP280readTrimming(&mem[0]) && BMP280readValues(&mem[24])){
    //Ambas lecturas fueron exitosas: se puede leer "mem"
    enviaMsgSerie((char *)"TRIM VALUES (T1 T2 T3 P1 ... P9):\n\r",0)
        ;
    while (i<24) {
        val  = (unsigned short)mem[i++] << 8;
        val += mem[i++];
        phlen=(char)sprintf(buff,(i==2 || i==8)?"%u ":"%i ",val);
        enviaMsgSerie(buff,phlen);
    }
    enviaMsgSerie((char *)"\n\rSENS VALUES (PRESS TEMP):\n\r",0);
    while (i<30) {
        val2  = (unsigned long)mem[i++] << 12;
        val2 += (unsigned long)mem[i++] << 4;
        val2 += (unsigned long)mem[i++] >> 4;
        phlen = (char)sprintf(buff,"%lu ",val2);
```

```
25          enviaMsgSerie(buff,phlen);
26      }
27      enviaMsgSerie((char *)"\n\r",0);
28  } else {    //Una o las dos lecturas fallaron:
29      enviaMsgSerie((char *)"BMP␣-␣No␣hubo␣lectura\n\r",0);
30  }
```

En la línea 10 del código anterior, ya se dispone de la lectura de los datos en crudo de las posiciones de memoria en que se encuentran las constantes $T_1 ... T_3$ y $P_1 ... P_9$, y los valores U_T y U_P, todos de tipo entero con signo, exceptuando P_1 y T_1, de tipo entero sin signo. Todos estos se han de usar en la siguiente transcripción del procedimiento sugerido en la Hoja de Datos del sensor BMP280 [11], para obtener los valores de presión y temperatura:

$$V_1 \leftarrow \left(\frac{U_T}{2^{14}} - \frac{T_1}{2^{10}}\right) \times T_2 = \frac{T_2}{2^{10}}\left(\frac{U_T}{2^4} - T_1\right) \tag{C.1}$$

$$V_2 \leftarrow \left(\frac{U_T}{2^{17}} - \frac{T_1}{2^{13}}\right)^2 \times T_3 = \frac{T_3}{2^{26}}\left(\frac{U_T}{2^4} - T_1\right) \tag{C.2}$$

$$t_{fine} \leftarrow V_1 + V_2 \tag{C.3}$$

$$T \leftarrow \frac{t_{fine}}{(5120 = 2^{10} \times 5)} \tag{C.4}$$

$$V_1 \leftarrow \frac{t_{fine}}{2^1} - (64000 = 2^9 \times 5^3) \tag{C.5}$$

$$V_2 \leftarrow \frac{V_1^2 P_6}{2^{15}} \tag{C.6}$$

$$V_2 \leftarrow V_2 + 2V_1 P_5 \tag{C.7}$$

$$V_2 \leftarrow \frac{V_2}{2^2} + 2^{16} P_4 \tag{C.8}$$

$$V_1 \leftarrow \frac{\frac{V_1^2 P_3}{2^{19}} + V_1 P_2}{2^{19}} = \left(\frac{V_1}{2^{19}}\right)^2 P_3 + \left(\frac{V_1}{2^{19}}\right) P_2 \tag{C.9}$$

$$V_1 \leftarrow P_1\left(1 + \frac{V_1}{2^{15}}\right) \tag{C.10}$$

$$p \leftarrow 2^{20} - U_P \tag{C.11}$$

$$p \leftarrow \left(p - \frac{V_2}{2^{12}}\right) \times \frac{(6250 = 2 \times 5^5)}{V_1} \tag{C.12}$$

$$V_1 \leftarrow \frac{p^2 P_9}{2^{31}} \tag{C.13}$$

$$V_2 \leftarrow \frac{p P_8}{2^{15}} \tag{C.14}$$

$$p \leftarrow p + \frac{V_1 + V_2 + P_7}{2^4} \tag{C.15}$$

Las anteriores ecuaciones deben entenderse como asignaciones, esto es, $p \leftarrow p + 1$ significa que

a p se le debe asignar su valor actual, incrementado en 1, no hay inconsistencia matemática en esta expresión de ejemplo. Retomando las asignaciones anteriores, en (C.4) se tiene la temperatura en grados Celsius, y en (C.15) se tiene la presión en Pascales.

Este procedimiento puede ser implementado fácilmente en el servidor, con lenguajes de alto nivel como javascript(nodejs[1]) o php[2]. Tener en cuenta al momento de la implementación que todas las operaciones deben forzarse a flotante, para anular la promoción integral, por ejemplo, en la Ecuación ((C.1)) el 2^4 ha de escribirse como 16.0 para forzar una división entre flotantes.

[1] `https://nodejs.org/es/`
[2] `http://php.net/manual/es/preface.php`

Bibliografía

[1] A.S. Tanenbaum. *Redes de computadoras*. Editorial Alhambra S. A. (SP), 2003. ISBN: 9789702601623. URL: https://books.google.com.co/books?id=WWD-4oF9hjEC.

[2] *About LoRa Alliance™*. Inglés. LoRa Alliance™. URL: https://lora-alliance.org/about-lora-alliance.

[3] *What is the LoRaWAN™ Specification?* Inglés. LoRa Alliance™. URL: https://lora-alliance.org/about-lorawan.

[4] *AN1200.22: LoRa™ Modulation Basics*. Inglés. Semtech. URL: https://www.semtech.com/uploads/documents/an1200.22.pdf.

[5] *LoRa Device Developer Guide. Orange Connected Objects & Partnerships*. Inglés. Orange y Actility. 2017. URL: https://partner.orange.com/wp-content/uploads/2017/11/LoRa-Device-Developer-Guide-Orange.pdf.

[6] E.P. Municio, F.R. Domínguez y L.J.L. Pérez. *Microcontrolador PIC16F84: desarrollo de proyectos*. RA-MA, 2005. ISBN: 9788478976911. URL: https://books.google.com.co/books?id=-TIpAAAACAAJ.

[7] J.M.A. Usategui e I.A. Martínez. *Microcontroladores "PIC": diseño práctico de aplicaciones*. Microcontroladores PIC: diseño práctico de aplicaciones. Primera parte. El PIC16F84: lenguajes PBASIC y ensamblador v. 1. McGraw-Hill, 2003. ISBN: 9788448137885. URL: https://books.google.com.co/books?id=APpmPQAACAAJ.

[8] *HPM Series PM2.5 Particle Sensor with UART output*. Inglés. Honeywell Sensing. URL: https://sensing.honeywell.com/honeywell-sensing-hpm-series-particle-sensors-datasheet-32322550-d-en.pdf.

[9] *iAQ-core C Indoor air quality module, I2C interface, continuous operation mode*. Inglés. ams AG. URL: http://ams.com/eng/content/download/686523/1787697/348216.

[10] *HDC1010 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor*. Inglés. Texas Instruments. URL: http://www.ti.com/lit/gpn/hdc1010.

[11] *BMP280 Digital Pressure Sensor*. Inglés. BOSH Sensortec. URL: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-19.pdf.

[12] *Digital 16bit Serial Output Type Ambient Light Sensor IC*. Inglés. ROHM Semiconductor. URL: http://www.mouser.com/ds/2/348/bh1750fvi-e-186247.pdf.

[13] *Telaire T6700 Series | Miniature CO2 Sensor Modules*. Inglés. Amphenol Advanced Sensors. URL: https://amphenol-sensors.com/en/products/co2/co2-modules/3215-t6700.

[14] IMST GmbH. Inglés. IMST GmbH. URL: https://wireless-solutions.de/products/radiomodules/im880b-l.html.

[15] Microchip. *PIC16(L)F1764/5/8/9*. Inglés. Inf. téc. Microchip. 2017. URL: https://www.microchip.com/PIC16F1769.

[16] *PIC18F2455/2550/4455/4550*. Inglés. Microchip. URL: http://www.microchip.com/wwwproducts/en/pic18f2550.

[17] *GPS receiver GNS 601*. Inglés. Global Navigation Systems GmbH. 2012. URL: http://static6.arrow.com/aropdfconversion/a1fb5a37520cf896ef409d9825c753012b8cbec0/gns601_datasheet.pdf.

[18] D.W. Hart. *Electrónica de potencia*. Pearson Educación, 2001. ISBN: 9788420531793. URL: https://books.google.com.co/books?id=dIzMAAAACAAJ.

[19] *In-Circuit Serial Programming™ (ICSP™) Guide*. Inglés. Michochip. 2003. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/30277d.pdf.

[20] IMST GmbH. Inglés. IMST GmbH. URL: https://wireless-solutions.de/products/radiomodules/im880b-l.html.

[21] *PL2303 Family*. Inglés. Prolific. URL: http://www.prolific.com.tw/US/ShowProduct.aspx?pcid=41.

[22] *MCP73831/2 Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers*. Inglés. Microchip. URL: http://www.microchip.com/wwwproducts/en/MCP73832.

[23] *BU-409: Charging Lithium-ion*. Inglés. Battery University, Cadex. URL: http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries.

[24] *PAM2305: 1A Step-Down DC-DC Converter*. Inglés. Diodes Inc. URL: https://www.diodes.com/assets/Datasheets/PAM2305.pdf.

[25] *MPLAB® XC Compilers*. Inglés. Microchip. URL: http://www.microchip.com/mplab/compilers.

[26] *MPLAB® X Integrated Development Environment (IDE)*. Inglés. Microchip. URL: http://www.microchip.com/mplab/mplab-x-ide.

[27] *NetBeans IDE*. Inglés. URL: https://netbeans.org/.

[28] Ravishankar Somasundaram. *Git: Version control for everyone*. Inglés. Packt Publishing, 2012. ISBN: 9781849517522.

[29] *Página Oficial de Git*. Inglés. URL: https://git-scm.com/.

[30] Alex F. Bustos. *HCI Stack and Application for iM880B*. Inglés. 2018. URL: https://github.com/pylatex/im880b.

[31] *How to Build Your Own C Library*. Inglés. URL: http://www.cs.dartmouth.edu/~campbell/cs50/buildlib.html.

[32] Tia Newhall. *C Libraries*. Inglés. Swarthmore College. URL: https://cs.swarthmore.edu/~newhall/unixhelp/howto_C_libraries.html.

[33] Alfred Park. Inglés. randu.org. 1999. URL: https://randu.org/tutorials/c/libraries.php.

[34] *MPLAB® XC8 C Compiler User's Guide*. Inglés. Microchip. URL: ww1.microchip.com/downloads/en/DeviceDoc/50002053G.pdf.

[35] *MPLAB® Code Configurator*. Inglés. Microchip. URL: http://www.microchip.com/mplab/mplab-code-configurator.

[36] *PuTTY - a free SSH and telnet client for Windows*. Inglés. URL: https://www.putty.org/.

[37] *HyperTerminal Private Edition*. Inglés. URL: https://www.hilgraeve.com/hyperterminal/.

[38] *minicom(1) - Linux man page*. Inglés. URL: https://linux.die.net/man/1/minicom.

[39] J.L.Á. Jiménez. *UF2406 - El cliclo de vida del desarrollo de aplicaciones*. Editorial Elearning, S.L., 2016. URL: https://books.google.com.co/books?id=OVlWDwAAQBAJ.

[40] José A. Mañas. *javadoc: documentación de programas*. URL: https://web.dit.upm.es/~pepe/doc/fprg/javadoc.htm.

[41] *How to Write Doc Comments for the Javadoc Tool*. Inglés. Oracle. URL: http://www.oracle.com/technetwork/articles/java/index-137868.html.

[42] Javier Cristóbal. *Markdown - La guía definitiva en español*. 2018. URL: https://markdown.es/.

[43] Matt Cone. *Markdown Guide*. Inglés. 2018. URL: https://www.markdownguide.org/.

[44] *MultiConnect® mDot™ Box*. Inglés. Multitech. URL: https://www.multitech.com/brands/multiconnect-mdot-box.

[45] *Mote II - Long Range Radio*. Inglés. IMST. URL: https://wireless-solutions.de/products/starterkits/mote-ii.html.

[46] *Mote II*. Inglés. IMST GmbH. URL: https://shop.imst.de/wireless-modules/lora-products/25/mote-ii.

[47] *Acceso a la Implementación del Servidor LoRa*. Inglés. URL: https://lora.glud.org.

[48] *Acceso a la Implementación del Servidor Grafana*. Inglés. URL: https://grafana.glud.org/.

[49] *LoRa Server, open-source LoRaWAN network-server*. Inglés. URL: https://www.loraserver.io/.

[50] *Grafana: The open platform for beautiful analytics and monitoring*. Inglés. URL: https://grafana.com/.

[51] Inglés. Semtech y Stackforce. URL: https://github.com/Lora-net/LoRaMac-node.

[52] Roger Dudler. *git - la guía sencilla*. URL: http://rogerdudler.github.io/git-guide/index.es.html.

[53] I. Sommerville y M.I.A. Galipienso. *Ingeniería del software*. Pearson Educación, 2005. ISBN: 9788478290741. URL: https://books.google.com.co/books?id=gQWd49zSut4C.

[54] *LoRaWAN™ Specification v1.1*. Inglés. LoRa Alliance™. URL: https://lora-alliance.org/resource-hub/lorawantm-specification-v11.

[55] B.A. Forouzan. *TCP/IP Protocol Suite*. McGraw-Hill Forouzan networking series. McGraw-Hill Higher Education, 2009. ISBN: 9780070166783. URL: https://books.google.com.co/books?id=yNthPwAACAAJ.

[56] *Cayenne Low Power Payload*. Inglés. URL: https://mydevices.com/cayenne/docs/lora/#lora-cayenne-low-power-payload.