Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Trabajo Fin de Grado

Ingeniería en Electrónica Industrial y Automática

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

Documentos:

1. Memoria
2. Planos
3. Pliego de condiciones
4. Presupuesto
5. Anexos:
    - 5.1: Documentación código
    - 5.2: Hojas de características

Autor:

D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

Resumen

Con una presencia de la tecnología cada vez mayor en nuestras vidas, la siguiente frontera es conectar también los objetos cotidianos, desde el frigorífico hasta los semáforos de la ciudad. Este campo es conocido como internet de las cosas y es uno de los ámbitos con mayor futuro en los próximos años, esperando que cada vez más objetos envíen los datos que recolectan a internet para su monitorización, ayudando a optimizar gran cantidad de procesos que actualmente se realizan sin obtener ningún tipo de retroalimentación, esto tiene el potencial de crear sistemas más eficientes, reduciendo el consumo de agua para el riego de los parques públicos o ayudando a detectar un defecto cardiaco antes de que se produzca un problema de salud grave, entre muchas aplicaciones.

Con este trabajo se pretende monitorizar las condiciones ambientales que pueden darse en un campo de cultivo, en este caso un campo de caquis, para tratar de acercar el internet de las cosas al mundo de la agricultura, una industria de gran peso en la economía y que está metida de lleno en un proceso de modernización y tecnificación.

En el proyecto se desarrolla un prototipo de dispositivo capaz de medir varias magnitudes del entorno del cultivo y accionar el riego, todo esto controlado de manera inalámbrica y accesible desde un ordenador o móvil.

Palabras clave: Sensor, LoRaWAN, The Things Network, IoT, Agricultura, Riego, Automatización.

Abstract

With an increasing presence of technology in our lives, the next frontier is connecting everyday objects, from the fridge to the traffic lights. This field is known as the Internet of things and is one of the fields with the brightest future in the coming years, expecting that more and more objects will send the collected data to the internet for monitoring, helping to optimize many processes that are currently being carried out without getting any feedback. This has the potential to create more efficient systems, reducing water consumption for irrigation of public parks or to detect a cardiac defect before a serious health problem happened, among many other applications.

This work aims to monitor the environmental conditions that can occur in a crop field, in this case, a field of persimmons, to help to bring the Internet of things to the world of agriculture, an industry of huge economical weight which is fully involved in a process of modernization.

In this project, a prototype of a device capable of measuring various magnitudes of the crop's environment and triggering the irrigation system is developed, all of which is controlled wirelessly and accessible from a computer or smartphone.

Keywords: Sensor, LoRaWAN, The Things Network, IoT, Agriculture, Irrigation, Automation.

Resum

Amb una presència de la tecnologia cada vegada major en les nostres vides, la següent frontera és connectar també als objectes quotidians, des del frigorífic fins als semàfors de la ciutat. Aquest àmbit és conegut com la internet de les coses i és un dels camps amb major futur en els propers anys, esperant que cada vegada més objectes envien les dades que recol·lecten a internet per al seu monitoratge, ajudant a optimitzar gran quantitat de processos que actualment es realitzen sense obtenir cap mena de retroalimentació, això té el potencial de crear sistemes més eficients, reduint el consum d'aigua per al reg dels parcs públics o per a detectar un defecte cardíac abans que es produïsca un problema de salut greu, entre moltes aplicacions.

Aquest treball pretén monitorar les condicions ambientals que poden donar-se en un camp de cultiu, en aquest cas un camp de caquis, per a tractar d'acostar la internet de les coses al món de l'agricultura, una indústria de gran pes en l'economia i que està ficada de ple en un procés de modernització i tecnificació.

En el projecte es desenvolupa un prototip de dispositiu capaç de mesurar diverses magnituds de l'entorn del cultiu i accionar el reg, tot això controlat sense fil i accessible des d'un ordinador o mòbil.

Paraules claus: Sensor, LoRaWAN, The Things Network, IoT, Agricultura, Reg, Automatització.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

1. Memoria

Autor:

D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

ÍNDICE DEL PROYECTO

ÍNDICE DE FIGURAS

GLOSARIO DE SIGLAS Y ABREVIATURAS

ADC	Conversor de analógico a digital
API	Interfaz de programación de aplicaciones
DC	Corriente continua
HAL	Capa de abstracción de hardware
I2C	Circuito inter-integrado
IDE	Entorno de desarrollo integrado
IoT	Internet de las cosas
LDR	Fotorresistencia
LPWAN	Red de área amplia de baja potencia
LoRa	Largo alcance
MCU	Microcontrolador
MEMS	Sistemas microelectromecánicos
MPU	Microprocesador
NC	Normalmente cerrado
NO	Normalmente abierto
PCB	Placa de circuito impreso
RAM	Memoria de acceso aleatorio
ROM	Memoria de solo lectura
SCL	Reloj en serie
SDA	Datos en serie
SPI	Interfaz de periféricos en serie
STM	STMicroelectronics
TTN	The Things Network

1. OBJETO DEL PROYECTO

El objetivo de este proyecto es el desarrollo del primer prototipo de un sistema para facilitar la producción del caqui mediante la automatización parcial del proceso. El sistema recolecta datos de las condiciones ambientales del cultivo para optimizar el regado de este y detectar posibles anomalías en el proceso.

Para lograr una correcta monitorización de las condiciones ambientales se pretende desarrollar un dispositivo IoT con diversos sensores, actuadores y conectividad inalámbrica. El dispositivo recolectará la información de los sensores y la enviará mediante una red de comunicación inalámbrica de bajo consumo, o LPWAN por sus siglas en inglés (Low Power Wide Area Network), a una plataforma online, desde donde el usuario podrá visualizarla. Basándose en la información recibida y las preferencias del usuario, el sistema activará las válvulas de riego de forma remota, utilizando el mismo dispositivo que recolectó la información u otro específico para ello, en cualquier caso, siempre ocurrirá haciendo uso de la misma red de comunicaciones.

El uso de una LPWAN permite tener sensores a distancias de cientos de metros o incluso kilómetros de la puerta de enlace (Gateway) haciendo uso de un consumo energético muy reducido, extendiendo la autonomía del dispositivo.

2. ANTECEDENTES

Desde la aparición de la agricultura, esta ha ido avanzando conforme lo hacían las sociedades en las que se encontraba. Actualmente uno de los conceptos tecnológicos que más está creciendo es el Internet de las cosas, también conocido como IoT por sus siglas en inglés (Internet of Things).

La aparición de la agricultura data alrededor del 9500 a.C. y se localiza en el Levante mediterráneo (Zona donde se localizan las actuales naciones de Siria, Líbano, Israel, Palestina y Jordania) [1]. Su aparición supuso un cambio drástico en la forma de vida de las personas, al pasar de ser comunidades nómadas de cazadores y recolectores a transformarse en sociedades sedentarias, este proceso es conocido como Revolución Neolítica.

A lo largo de los siguientes milenios, la agricultura fue extendiéndose por todo Oriente medio, a la vez que apareció de manera independiente en el Este de Asia. Durante este periodo se desarrollaron las primeras técnicas de cultivo a gran escala y se comenzó a utilizar animales (Principalmente el Uro, antecedente de las vacas actuales) para el trabajo agrícola.

Las técnicas de cultivo fueron mejorando con el paso del tiempo, permitiendo grandes avances en la sociedad y el crecimiento de estas, como ocurrió en la antigua Roma con la creación del arado romano y el sistema de barbecho.

En la Edad Media, los árabes desarrollaron numerosos avances en los procesos de producción agrícola, principalmente en lo relativo al regadío. Estos cambios junto al uso de la rotación trienal permitieron mejorar la producción de vegetales.

En los siglos posteriores, la agricultura ha ido cambiando debido a las sociedades de las distintas épocas, a la vez que influía en estas, provocando reestructuraciones en la organización social, demográficos y grandes migraciones humanas.

Con este pequeño contexto histórico se puede ver como los cambios en la producción agraria y el uso de nuevas técnicas permiten el avance de la sociedad.

El concepto de IoT se basa en la idea de conectar y sensorizar todo tipo de objetos cotidianos y dispositivos para poder controlarlos remotamente, o incluso sin intervención humana aprovechando la información que registran a través de sensores. La tecnología IoT es cada vez más popular debido a la disminución de los precios de los sensores y microcontroladores y la aparición de protocolos de comunicación inalámbricos orientados para este tipo de soluciones. El IoT permite obtener información en tiempo real e interactuar con dispositivos que pueden encontrarse a miles de kilómetros o, incluso que los propios dispositivos interactúen entre ellos sin necesidad de intervención humana.

La población humana es mucho mayor a lo que era en los siglos anteriores, esto provoca que para poder alimentar a todo el mundo sea necesario optimizar el terreno agrario para conseguir el máximo rendimiento posible. Como respuesta a este problema aparece la idea de la agricultura inteligente o *Smart*. La agricultura inteligente consiste en aplicar las nuevas tecnologías como drones, Big Data o IoT para hacer más eficientes las explotaciones agrícolas minimizando la necesidad de intervención humana en el proceso, reduciendo el consumo de agua y abonos, lo que ayuda a minimizar el impacto ambiental del proceso.

3. ESTUDIO DE NECESIDADES, FACTORES A CONSIDERAR: LIMITACIONES Y CONDICIONANTES

En este capítulo se pretende establecer cuáles son las necesidades que el proyecto intenta abarcar. Limitar las dimensiones de este permite tener un punto de partida claro sobre el que empezar a trabajar y unos objetivos a alcanzar

claramente definidos, esto facilita el desarrollo del proyecto y ayuda a identificar qué partes pueden requerir de más tiempo y esfuerzo.

En los subapartados siguientes se exponen los factores a analizar que se han considerado más relevantes para asegurar la viabilidad del proyecto.

3.1 Magnitudes a medir

Este es uno de los puntos más críticos en el proyecto, pues la utilidad real del sistema dependerá de cuáles sean las magnitudes medida, si estas no resultan ser las más relevantes para el análisis de las características del entorno, la funcionalidad del dispositivo será nula, independientemente del resto de características que pueda tener.

3.1.1 Temperatura

La temperatura juega un papel crucial para el correcto desarrollo de los árboles y sus frutos [2]. Si se alcanzan temperaturas excesivamente altas o bajas respecto a la ideal durante el periodo de maduración del caqui, en torno a 22 °C, el desarrollo del fruto puede verse afectado, dando como resultado caquis de menor tamaño y peso, lo que se traduce en una mayor dificultad a la hora de dar salida a la producción y una disminución en los beneficios obtenidos por su venta para el agricultor.

Aunque monitorizar la temperatura no supone un beneficio *per se,* puede ayudar a predecir cómo va a ser la cosecha y así suplementar adecuadamente los árboles para poder obtener los resultados deseados.

3.1.2 Humedad

La humedad es otro de los parámetros claves para el correcto desarrollo del árbol y del fruto. Tanto la humedad del aire como la del suelo resultan críticas. Si la humedad del suelo es demasiado baja, la planta será incapaz de absorber los minerales y fertilizantes necesarios para su correcto desarrollo [3]. Por el contrario, si la humedad del suelo es excesiva, la concentración de estos será demasiado baja para nutrir adecuadamente a la planta. En lo relativo a la humedad del aire la situación es similar, valores excesivamente altos o bajos provocarán problemas en el correcto desarrollo de la planta. La humedad demasiado baja puede producir el marchitamiento de las hojas e impedir el correcto desarrollo del fruto. La humedad excesivamente alta, deriva en deficiencias nutricionales que conducen a la obtención de frutos menor concentración de fructosa y sabor. Ambos escenarios (tanto exceso como defecto de humedad) favorecen la aparición de enfermedades en el cultivo.

3.1.3 Iluminancia

Los caquis requieren de una buena iluminancia para tener un desarrollo idóneo [4]. Para conseguir frutos de calidad es necesario asegurar una buena cantidad

de horas de sol al árbol, ya que si esto no ocurre el crecimiento y calidad de los caquis se verá mermado.

3.1.4 Presión atmosférica

La presión atmosférica no resulta directamente determinante para el desarrollo del caqui, pero es un parámetro que resulta útil en el ámbito de la meteorología [5] a la hora de predecir si se van a dar las condiciones apropiadas para que se produzca lluvia próximamente. Esta información resulta interesante a la hora de decidir si se va a regar el campo o no.

3.2 Comunicación y redes

Los sensores deben estar colocados en el propio campo donde se encuentren los árboles y, al mismo tiempo, ser capaces de enviar la información recolectada a la plataforma web desde la que se visualiza, para que esto se lleve a cabo es necesario disponer de una gateway que conecte el sistema a internet. La gateway no tiene por qué estar necesariamente en la misma localización física que la placa que contenga los sensores, siempre y cuando se pueda establecer un sistema de comunicación fiable entre ellos.

El sistema de comunicación a utilizar tendrá que ser inalámbrico, pues un sistema cableado resultaría totalmente inviable por la distancia a cubrir entre los sensores y la puerta de enlace a internet. Será necesario que la tecnología de comunicaciones elegida pueda ser capaz de cubrir grandes distancias, del orden de cientos de metros o incluso kilómetros. Además, tendrá que ser capaz de soportar la existencia de cuerpos que puedan interferir en la comunicación *Sensor-Gateway* como pueden ser todos los árboles que haya entre medias.

Puesto que la cantidad de información a transmitir es pequeña y no crítica, no es necesario que se utilice una tecnología con un gran ancho de banda o especialmente rápida.

Como conclusión, en esta aplicación resulta más importante que la comunicación pueda ocurrir entre grandes distancias de manera fiable que la disponibilidad de un gran ancho de banda o una gran velocidad de transmisión de datos.

3.3 Sistema de riego

El sistema de riego que se utiliza en gran parte de los campos de frutales de producción extensiva es el riego por goteo. Este sistema resulta especialmente bueno por la poca cantidad de agua que desperdicia. Otra ventaja que tiene es la facilidad que presenta a la hora de automatizarlo. El paso del agua por las tuberías de riego por goteo se puede controlar con una llave de paso manual o se puede controlar mediante electroválvula. Las electroválvulas resultan altamente convenientes al poder ser activadas remotamente y no suponer una dificultad de instalación mayor a la de las llaves de paso manuales.

Es común en los sistemas de riego que se utilicen estos no solo para la dosificación de agua en el campo, sino también para la administración de ciertos fertilizantes solubles en agua, por esta razón es habitual disponer de varias electroválvulas para poder elegir qué fuente se utiliza en cada momento. Dependiendo del tamaño del campo, es posible tener varios sectores de riego independientes unos de otros, lo cual aumenta todavía más el número de electroválvulas necesario.

3.4 Alimentación y autonomía

El sistema tiene una clara vocación inalámbrica al estar pensado para colocarse en campos de cultivo, lugares que de forma habitual se encuentras separados de los núcleos urbanos y no suelen tener la posibilidad de conectarse a la red eléctrica.

La placa que contenga los distintos sensores debe poder ser capaz de mantener estos funcionando durante periodos prolongados de tiempo. No parece práctico que haya que remplazar la batería de estos dispositivos cada pocos días, pues esto resultaría en un gran inconveniente para el agricultor, especialmente si existen varias de estas placas distribuidas a lo largo del campo.

Otro factor a considerar para la alimentación del sistema son las electroválvulas, las cuales necesitan ser alimentadas para su uso y, de nuevo, resultaría poco práctico tener que sustituir la batería que utilicen cada poco tiempo.

Idealmente el sistema debe tener una autonomía de varios meses, preferiblemente por encima de un año.

La gateway al poder colocarse en un lugar distinto a los sensores o las válvulas, existe la posibilidad de utilizar una localización que disponga de conexión a la red eléctrica para poder despreocuparse de su autonomía.

4. PLANTEAMIENTO DE SOLUCIONES ALTERNATIVAS Y JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA

A la hora de afrontar el proyecto a completar hay que tener en cuenta que existe una gran variedad de opciones y caminos a tomar, siendo algunos más pedregosos que otros. En este apartado tiene como objetivo plantear las distintas opciones que se han tenido en cada aspecto del proyecto y justificar el porqué de la solución adoptada, la cual se explicará en mayor detalle en el apartado 5. Recalcar que en varios casos hay una variedad de opciones disponibles mucho mayor a la expuesta, simplemente no se han presentado todas ellas porque resultaría inviable tener que exponerlas todas en esta memoria.

4.1. Sensores y actuadores

Un sensor, por definición, es un dispositivo cuya función es medir algún tipo de magnitud física. Por norma general, la información obtenida de la medida será procesada, en mayor o menor medida, para utilizarla posteriormente. La complejidad de este proceso y su uso depende enormemente del sensor y el objetivo que haya detrás de la decisión de medir la magnitud correspondiente. Un termómetro de mercurio sirve para medir la temperatura, al igual que una cámara termográfica, sin embargo, ambos objetos son tremendamente distintos. Esto se debe a que cuando se va a realizar una medición hay que tener en cuenta más factores aparte de la magnitud a medir, es necesario conocer el entorno donde ocurrirá, ver cuál es el nivel de sensibilidad requerido, de qué manera se va a procesar la información y cuál es el rango de valores a medir; estos son solo algunos de los factores a considerar.

Al seleccionar un sensor hay que prestar atención al tipo de salida que ofrece. En primer lugar, hay que considerar si esta va a ser digital o analógica. Una salida analógica nos proporcionará una señal de salida continua y proporcional a la magnitud mediad, esta señal tendrá que ser convertida en digital para que el microcontrolador pueda hacer uso de ella [6]. Por el contrario, una señal de tipo digital discretizará la señal a medir, la precisión esta respecto a la analógica dependerá de la resolución del sensor. La salida digital se compondrá de una serie de bits (normalmente formando uno o más bytes) que el microcontrolador podrá procesar directamente.

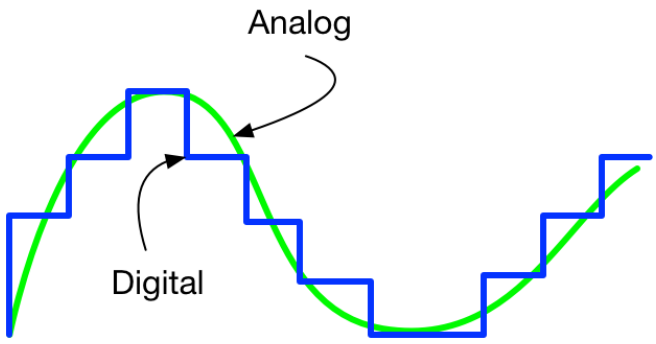


Figura 1. *Comparación entre la misma señal digital y analógica*

Realmente, los sensores digitales son la unión dentro de un mismo encapsulado de un sensor analógico y un ADC (Analog to Digital Converter) cuya salida utilizará la interfaz de comunicación que haya decidido el fabricante. En nuestro caso, como queremos que el microcontrolador sea capaz de procesar las señales de los sensores, lo más lógico es elegir sensores digitales.

Para hacer una criba en la elección de los sensores, un sistema rápido y efectivo es acceder a alguno de los principales distribuidores de componentes

electrónicos (mouser.es o digikey.es, por ejemplo) y utilizar los filtros que tienen habilitados para ver únicamente los que cumplen con las especificaciones mínimas que necesitemos. Entre estos hacer una pequeña búsqueda por internet para ver cuáles son más utilizados y disponen de más información sobre ellos online. Esto nos facilitará el trabajo posteriormente, pues si existen librerías ya creadas para el sensor, nos llevará menos trabajo adaptarlas a nuestro uso que si tuviésemos que crearlas desde cero. Además, al haber un mayor número de personas utilizándolo resulta más sencillo encontrar soluciones en caso de tener algún problema con el sensor.

4.1.1. Bus de comunicación

La elección del bus de comunicación va ligado al uso que vaya a tener el sistema a desarrollar. No hay buses de comunicación necesariamente mejores, su idoneidad dependerá de la información que se quiera transmitir y del entorno donde vaya a ocurrir. En nuestro caso los datos a transmitir serán pequeñas secuencias que proporcionen los valores que los sensores están leyendo. Por tanto, no es necesario un gran ancho de banda, y como no se van a utilizar en tiempo real (en ciertas aplicaciones, como puede ser un coche autónomo, los datos deben utilizarse al instante, pues de no ser así las consecuencias pueden ser terribles. Pero en nuestro caso no es necesaria esta inmediatez) no necesitamos grandes velocidades de transmisión ni un bus especialmente robusto. Además, hay que tener en cuenta que, el bus a elegir tiene que estar presente tanto en el microcontrolador como en los sensores que se elijan.

Basándonos en los motivos expuestos en el párrafo anterior, hay dos buses que cumplen con los requisitos:

- **I2C:** Inter-Integrated Circuit o I2C es un bus de comunicaciones serie ampliamente utilizado para la comunicación entre un microcontrolador y toda clase de sensores y actuadores. Únicamente requiere de dos cables para funcionar, SCL o Serial Clock para el reloj de pulsos y SDA o Serial Data para transmitir la información. El microcontrolador ejerce de máster (Es el que controla el reloj) y los sensores como esclavos (Solo transmiten información cuando el máster se la pide). En un mismo bus es posible tener 112 dispositivos conectados, esto ocurre gracias a que cada dispositivo debe tener una dirección I2C compuesta por 7 bits (Hay 16 direcciones no disponibles por estar reservadas para funciones especiales), esta dirección debe ser única respecto al resto de dispositivos del bus [7]. La velocidad máxima de funcionamiento es de 400 Kbps y con comunicación *half-duplex* (No es posible la transmisión y recepción de información simultáneamente). Existen actualizaciones del bus que permiten utilizar un mayor número de dispositivos y velocidades más altas, pero son menos comunes.

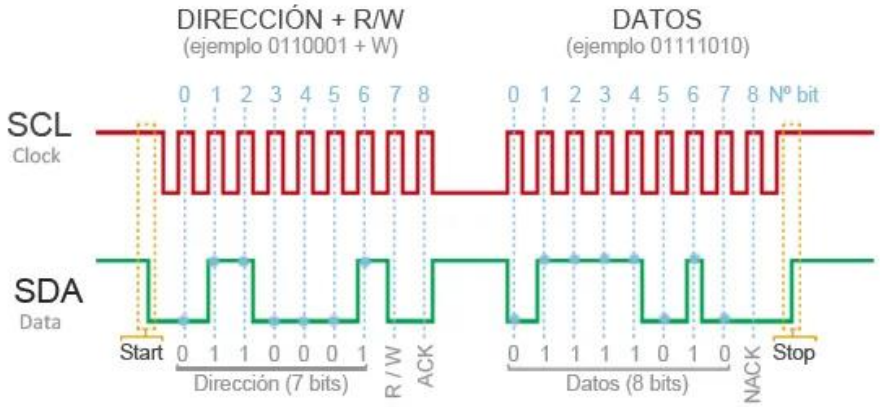


Figura 2. *Representación de una transmisión en I2C*

- **SPI:** Seria Periphal Interface o SPI es un bus de comunicaciones ampliamente utilizado en la industria [8]. Está compuesto por cuatro cables, SCLK (Serial Clock) para el reloj de pulsos, MOSI (Master Output Slave Input) y MISO (Master Input Slave Output) para la transmisión de información y SS (Slave Select) para elegir con que slave se comunica. A diferencia de I2C, en SPI no existen direcciones para los slaves, sino que cada uno de ellos debe conectarse a uno de los pines SS de los que disponga el máster (el número máximo de slaves dependerá de la cantidad de pines SS que tenga disponible el máster). La otra principal diferencia respecto a I2C es la posibilidad de comunicarse en *full-duplex* (Se transmite y recibe información de manera simultánea) gracias a la existencia de dos cables dedicados a la transmisión de información, además esto puede ocurrir a velocidades de hasta 10 Mbps, superior a I2C.

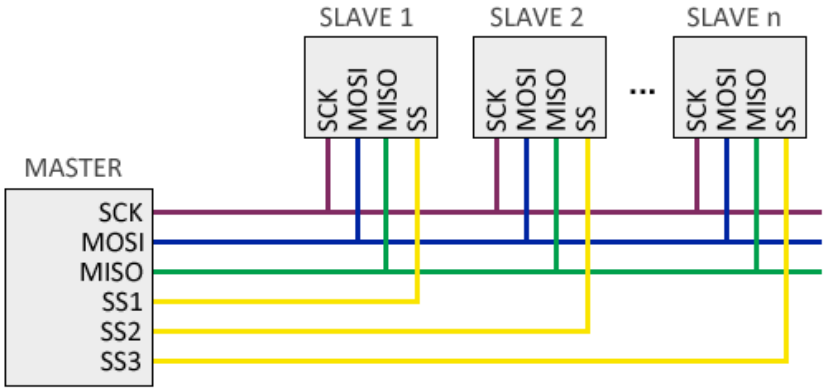


Figura 3. *Esquema de conexión en un bus SPI*

Una vez expuestos y analizados ambos buses de comunicación, se ha decidido hacer uso de I2C para el proyecto. La razón de esta decisión se basa en que, aunque SPI dispone de ventaja a la hora de transmitir datos, esta no iba a ser relevante para nuestro sistema. Además, I2C tiene la gran ventaja de simplificar

el cableado al utilizar solo 2 cables, independientemente de cuantos dispositivos queramos conectar.

4.1.2. Sensor de temperatura y humedad del aire

Es común encontrar encapsulados que dispongan simultáneamente de sensor de temperatura y sensor de humedad del aire, por ello ambos sensores comparten un mismo apartado en lugar de estar en dos apartados diferentes.

Para elegir este sensor habrá que mirar tres características principales: rango de medida, resolución y consumo energético.

El rango de medida deberá ser suficiente para que el sensor mida sin acercarse a saturación temperaturas inferiores a 0 °C y superiores a 40 °C y de 0-100% en humedad relativa. La precisión en temperatura deberá ser como mínimo de 0.3 °C y 3% para la humedad. Se buscará que el consumo sea bajo, especialmente en reposo, pues es como estará la mayor parte del tiempo; el consumo deberá encontrarse por debajo de 200 nA.

Entre los que cumplían las especificaciones se ha decidido utilizar el HDC2080 [9] de Texas Instruments por cumplir con los requisitos mínimos de precisión y rango y tener un consumo en reposo realmente bajo. Sus principales características son:

- Rango de humedad relativa: 0% a 100%
- Precisión humedad: ±2%
- Rango de temperatura: -40 °C a 85 °C
- Precisión temperatura: ±0.2 °C
- Consumo durante la medición: 550 uA
- Consumo en reposo: 50 nA

Figura 4. *HDC2080 en una placa para desarrollo*

Este modelo es común dentro de la comunidad “maker” por lo que hay una gran cantidad de información y librerías disponibles para su uso.

Existen otros modelos como el SHT31 de Sensirion o el HDC1080 de Texas Instruments que, aunque cumplían con las especificaciones de medida, tenían un consumo en reposo superior al HDC2080, razón por la que este ha sido finalmente el elegido.

4.1.3. Sensor de humedad del suelo

Los sensores para la humedad del suelo a de manera indirecta. Existen diversos sistemas para lograr esta medición, dos de los más populares y en los que nos centraremos son los que la miden mediante celda galvánica y los que lo hacen por capacitancia.

Los de celda galvánica tienen como principio de funcionamiento la celda galvánica de Luigi Galvani. Su principio de funcionamiento es una reacción de reducción-oxidación, concretamente electrólisis, entre dos metales separados entre sí dentro de un medio conductor, cuando se aplica electricidad a los metales (en la celda original Zinc en el ánodo y Cobre en el cátodo) existirá una transmisión de electrones del cátodo al ánodo, cuanto mejor conductor sea el medio en el que se encuentren los electrodos, mayor será la diferencia de potencial medida.

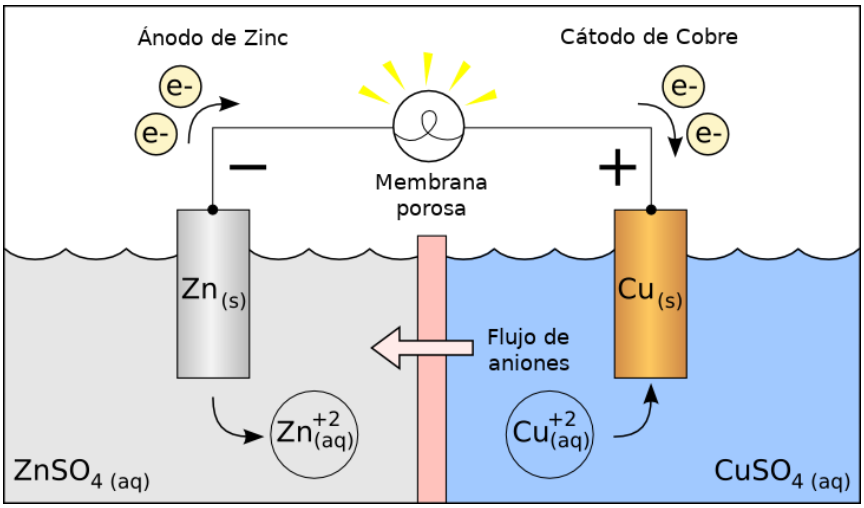


Figura 5. *Representación de una celda galvánica*

El principal problema de este tipo de sensores es que deben tener los electrodos expuestos al medio para funcionar correctamente, esto favorece la oxidación y el deterioro de estos. Asimismo, la propia naturaleza del funcionamiento de este tipo de sensores provoca que dejen de funcionar una vez existe un nivel de corrosión suficiente mente alto en el ánodo.

Figura 6. *Sensor dañado por la corrosión, como se puede apreciar, el cátodo ha perdido toda la deposición de cobre*

Los sensores de tipo capacitivo, aunque visualmente son similares a los galvánicos, su principio de funcionamiento es completamente distinto. Estos sensores miden la humedad del agua a partir del nivel capacitivo que detectan. Básicamente son un condensador en el que sus electrodos se encuentran en la sonda y el dieléctrico es el suelo, un cambio en el nivel de humedad del suelo se traduce en la resistencia del dieléctrico y, por tanto, en la capacitancia del sensor. Esto es lo que se medirá y desde donde se puede inferir el nivel de humedad del suelo.

Debido al principio de funcionamiento de los sensores capacitivos, no es necesario que los electrodos estén expuestos al medio, por lo que no sufren los problemas relacionados con ello que sí que tienen los de célula galvánica. Por este motivo se ha decidido hacer uso de un sensor de tipo capacitivo para el proyecto.

Como ejemplos de sondas capacitivas que podrían utilizarse para el proyecto nos encontramos con modelos profesionales como el sensor 10HS de ECHO DECAGON o la WET-2 de Delta T.

Figura 7. *Fotografía de las sondas 10HS (izquierda) y WET-2 (derecha)*

Este tipo de sondas tienen una gran durabilidad y han sido testeadas de forma que aseguran una nivel alto de fiabilidad en sus medidas, debido a esto tienen un alto precio (cientos de euros). Por otra parte, estas sondas necesitan de un circuito de adaptación para poder utilizarlas con el protocolo I2C, pues su salida es analógica en la 10HS y TTL serial en la WET-2.

Existe otro tipo de sondas capacitivas más económicas, en las que todo el sistema de medida ha sido producido sobre una PCB (Placa de circuito impreso, por sus siglas en ingles). Estas sondas no ofrecen las garantías de las profesionales, pero son mucho más económicas (menos de 10€) y capaces de ofrecer un gran rendimiento para hacer pruebas en las primeras fases de un proyecto o para uso doméstico. Como ejemplo de este tipo de sondas tenemos el “Capacitive Soil Moisture Sensor v1.2” el cual es un diseño genérico que es vendido por una gran cantidad de distribuidores bajo su marca propia. De nuevo, nos encontramos ante un sensor con salida analógica.

Figura 8. *Fotografía del sensor genérico que una gran cantidad de marcas venden*

La otra sonda capacitiva económica más popular es la diseñada y vendida por el fabricante de componentes electrónicos, ampliamente reconocido en la comunidad “maker” y educativa, Adafruit. Su sensor es el STEMMA soil sensor, este sensor es similar al popular diseño genérico, pero presentando una mayor calidad de fabricación, una mayor cantidad de información técnica proporcionada por el fabricante y una salida digital I2C.

Figura 9. *Fotografía del sensor STEMMA de Adafruit*

Como el proyecto que se está realizando tiene un eminente carácter prototípico, descartaremos el uso de sondas de corte profesional. Entre las de carácter no profesional, se ha decidido utilizar la sonda STEMMA [10] por las ventajas que supone respecto al otro modelo y que se explicaron en el párrafo anterior.

4.1.4. Sensor de iluminancia

La iluminancia es el parámetro que mide la cantidad de luz que incide sobre un área, esta se mide en luxs (lx), que equivale a lm/m². Estos sensores permiten saber si las plantas están recibiendo una cantidad de luz suficiente.

El sensor de iluminancia más sencillo que podemos encontrar son las LDR (Light-dependent resistor) o fotorresistencias. Su principio de funcionamiento es sencillo, su valor como resistencia depende de la cantidad de luz que reciba, cuanta más luz reciba, más baja será su resistencia. Este tipo de sensores al ser simplemente una resistencia variable no tienen ningún tipo de salida, simplemente hay que medir la corriente que circula por ellos para saber qué cantidad de luz están recibiendo, por tanto, son sensores analógicos.

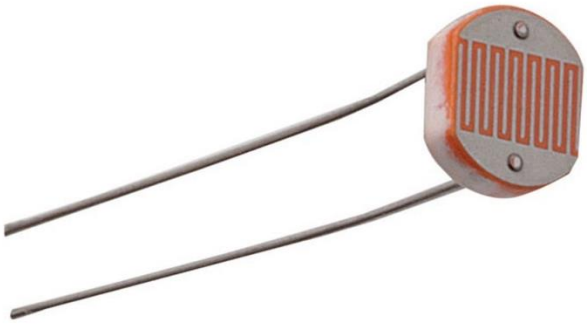


Figura 10. *Fotografía de un LDR convencional*

El otro tipo de sensores de iluminancia que podemos encontrar son los que hacen uso de un fotodiodo. Este tipo de sensores producen una corriente eléctrica cuando son expuestos a la luz, cuanta más luz, mayor será la corriente producida. Es común encontrarlos en un encapsulado que incluya el fotodiodo y un conversor ADC. Dos modelos comunes de este tipo de sensores son el TSL2591 de AMS y el VEML7700 de Vishay.

El TSL2591 cuenta con un conversor digital de alta sensibilidad y es capaz de proporcionar una salida directamente a la interfaz I2C. Este sensor dispone de 2 fotodiodos, uno para medir la luz ambiental y otro para la infrarroja.

El VMEL7700 es capaz de detectar valores de iluminancia de hasta 120 klx, incluye un filtro paso bajo para rechazar el ruido a 100 y 120 Hz y lleva integrado un sensor de temperatura para compensar los valores de las mediciones. Al igual que el TSL2591, dispone de un conversor de alta sensibilidad y su salida es I2C.

Ambos sensores son modelos populares y de uso extendido, resulta sencillo encontrar librerías desarrolladas para ellos e información sobre su uso.

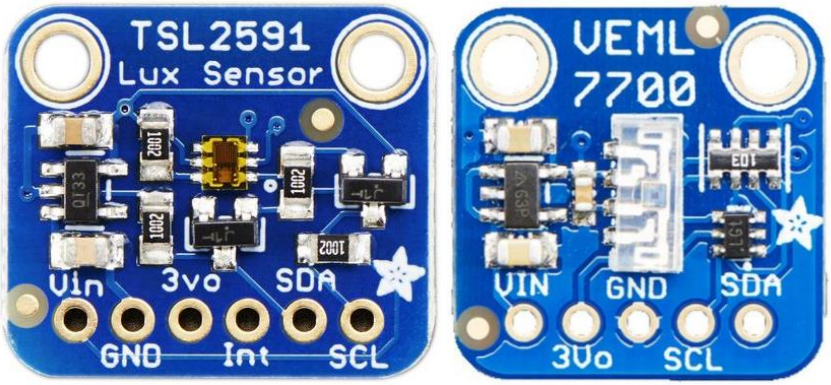


Figura 11. *Placas de Adafruit con los sensores en el centro de ellas*

Ambos sensores resultan válidos para este proyecto, pero se ha decidido hacer uso del TSL2591 por disponer de la capacidad de medir la luz infrarroja [11].

4.1.5. Sensor de presión atmosférica

Los sensores de presión atmosféricas son del tipo MEMS. El término MEMS hace referencia a *microelectromechanical systems,* son dispositivos de tamaño microscópico donde se combinan la tecnología mecánica y la eléctrica. En el caso de este tipo de sensores, existe una lámina piezorresistiva muy sensible a los cambios de presión, en la lámina existe una diferencia de potencial y dependiendo de la presión atmosférica el valore del piezorresistivo variará, a partir de estos dos valores se calculará la corriente producida. Esta corriente, al depender del valor del piezorresistivo se podrá utilizar para conocer la presión atmosférica. Es habitual que los sensores barométricos incluyan un sensor de temperatura para poder compensar los cambios resistivos debidos a la temperatura del componente piezorresistivo.

Algunos modelos populares de este tipo de sensores son:

- **Bosch BME280:** este sensor tiene un rango de medida de 300 a 1100 mbar, incluye sensor de temperatura y de humedad relativa, su consumo en reposo es de 0.2 uA y su salida es mediante I2C.
- **Infineon DPS310:** su rango de medición es de 300 a 1200 mbar, incluye un sensor de temperatura para compensación, además, cuenta con una memoria interna para almacenar hasta 32 mediciones y los coeficientes para calibrar la medición. Su consumo en reposo es de 0.5 uA. Se puede utilizar tanto para I2C como con SPI.
- **TE MS5637:** similar al DPS310, comparte el mismo rango de medición y también tiene memoria interna y sensor de temperatura. Por el contrario, su consumo en reposo es mucho mayor, 0.01 uA. Es compatible con I2C.

Hemos descartado el BME280 por no necesitar la medida de humedad relativa al tener otro sensor para ello, además no dispone de memoria interna con los

parámetros de calibración para poder tener medidas más precisas. Finalmente, el modelo seleccionado ha sido el MS5637 [12] por tener un consumo en reposo, que es el modo en el que estará el sensor la mayor parte del tiempo, mucho menor que el DPS310.

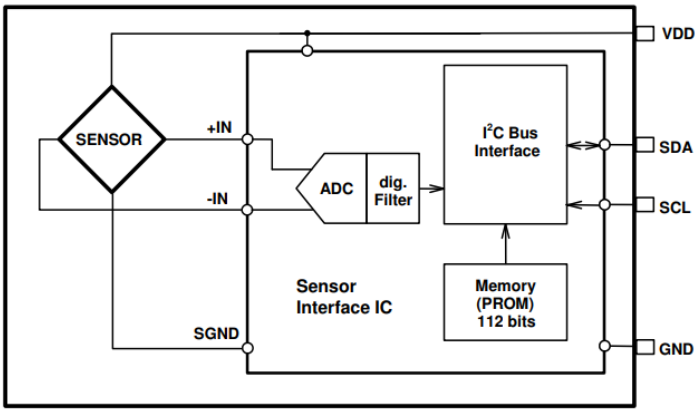


Figura 12. *Esquema del sensor MS5637*

4.1.6. Electroválvula de riego

Los solenoides de riego o electroválvulas permiten la apertura y cierre de las líneas humanas de manera automatizada. Este tipo de actuadores está compuesto por un solenoide y un muelle. El muelle sirve para forzar y mantener la válvula en la posición de reposo, dependiendo del actuador, esta puede ser abierta o cerrada. El solenoide crea un campo magnético cuando se le suministra una corriente eléctrica, este campo atraerá el cilindro ferromagnético que se encuentra dentro del solenoide modificando la posición en la que se encuentre y permitiendo o bloqueando el paso del fluido, dependiendo de cuál sea su posición de reposo habitual [13]. Las válvulas NC (Normalmente Cerradas) no permiten el paso del fluido en su posición de reposo, solo una vez son activadas. Las de tipo NO (Normalmente Abiertas) bloquean el paso del fluido al ser activadas.

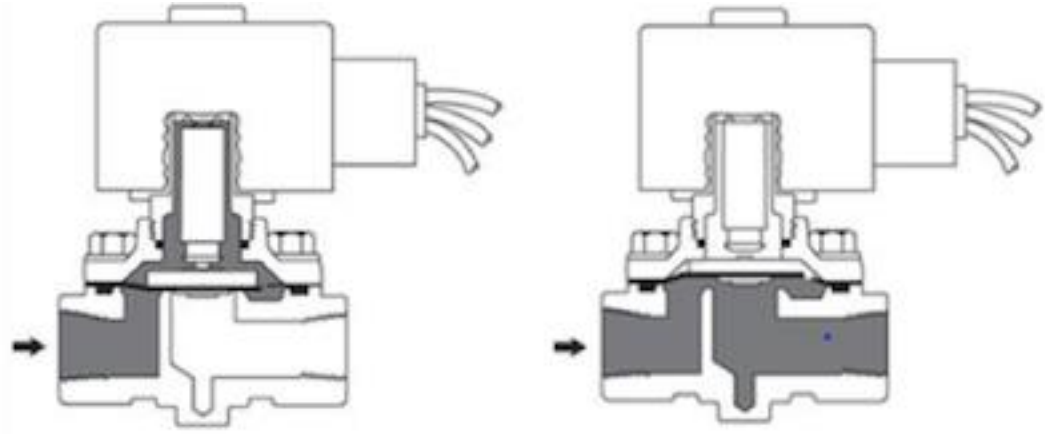


Figura 13. *Vista de la sección de una electroválvula NC, a la izquierda desactivada y a la derecha activada*

Existen electroválvulas tanto de corriente continua como de alterna y hay varios voltajes disponibles. Las configuraciones más comunes para las válvulas de riego son 24 V AC y 9 V DC. Las de 24 V emiten una señal para abrir la válvula y la mantienen, para cerrarla, dejan de emitir la señal. Están pensadas para localizaciones donde dispongan de conexión a la red eléctrica. Las de 9 V emiten la señal para abrir la válvula y vuelven a emitir para cerrarla, no la mantienen durante el tiempo de riego. Esto ocurre porque están pensadas para funcionar con pilas de 9 V en lugares sin acceso a la red eléctrica, por lo que el consumo es un factor vital a tener en cuenta.

Para nuestro proyecto, basándonos en la explicación previa, lo lógico resulta elegir una electroválvula de 9 V. Las otras especificaciones a tener en cuenta para elegir la electroválvula correcta son el caudal que ha de circular y el diámetro de la rosca de la tubería. El caudal máximo de riego que va a circular por las tuberías no excederá en ningún momento los 100 l/min y las tuberías que hay en la instalación son de una pulgada de diámetro. Toda válvula que cumpla con estas especificaciones debería ser válida para este uso, para mayor seguridad conviene utilizar modelos de marcas de cierto renombre que nos aseguren un mínimo de durabilidad y confianza como pueden ser RainBird, Orbit, Hunter o HidroRain. Finalmente, el modelo a utilar será una electroválvula de marca Orbit por ser la que existe en la instalación actual sobre la que se aplicará el prototipo.

4.2. Plataforma de desarrollo y microcontrolador

La plataforma donde se desarrolla el proyecto resulta un punto crítico de este, cada plataforma tiene sus particularidades, por lo que es necesario buscar la que mejor se adapte al uso que se persigue. Para ello es necesario comprender el concepto de microcontrolador.

Un microcontrolador es un circuito integrado formado por uno o más microprocesadores, memoria RAM, ROM y periféricos de entrada y salida. Los microcontroladores son capaces de ejecutar las ordenes que se encuentren almacenadas en su memoria.

Los microcontroladores (MCU) se diferencian de los microprocesadores (MPU) en que los segundos necesita de componentes externos, como la memoria, para poder funcionar mientras que los MCU disponen por sí mismos de todos los componentes esenciales [14].

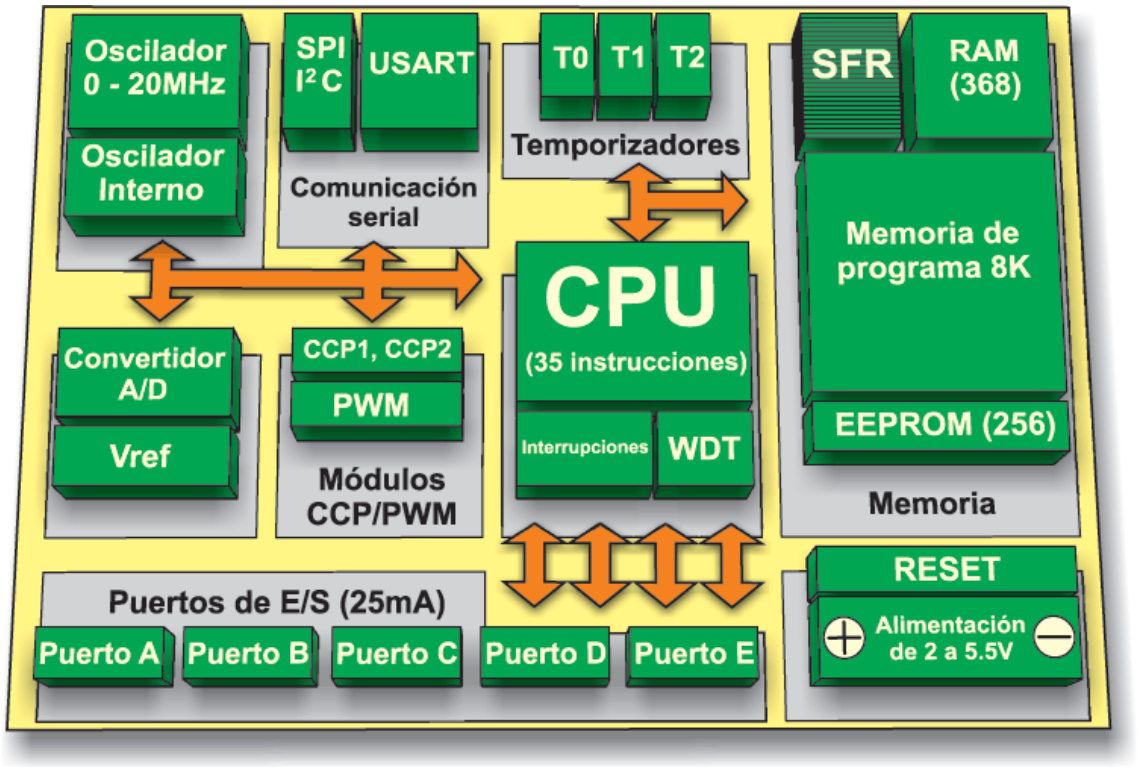


Figura 14. *Diagrama de bloques interno de un microcontrolador*

Los microcontroladores pueden clasificarse, según el tamaño de su bus de datos, en 8, 16 y 32 bits. Cuanto mayor sea el bus de datos, más complejas serán las operaciones que podrá realizar el microcontrolador.

Los microcontroladores, antiguamente eran programados en lenguaje ensamblador, el cual podía entender el MPU directamente. Actualmente, lo más habitual es programar el lenguajes de alto nivel, estos resultan mucho más intuitivos para el programador [15]. Cuando el programador ha terminado de escribir el código, este se compila. Al compilar lo que se hace es traducir el código del lenguaje utilizado por la persona a código máquina que pueda ser entendido por el MCU. Si el compilador no ha encontrado ningún error en el proceso, el programa puede ser grabado en la memoria interna del MCU. Al grabarlo, primero se eliminará la información que hubiese guardada previamente en la memoria del micro y a continuación se escribirán las nuevas “instrucciones” que utilizará el MCU.

Existe una gran variedad de microcontroladores disponibles, dependiendo de la función que se necesite, será mejor opción optar por un fabricante u otro. Dependiendo del fabricante y familia elegidos dispondremos de unos recursos de programación distintos. Esto es, aunque dos microcontroladores de dos fabricantes distintos estén diseñados para una misma función y puedan programarse en el mismo lenguaje, esto no significa que sean compatibles entre sí. Cada familia de microcontroladores tendrá sus propias librerías, compiladores

y herramientas de desarrollo y estás normalmente no servirán para otras familias de MCUs. Por ello es importante elegir una plataforma de desarrollo que disponga de una gran variedad de microcontroladores disponibles y librerías para poder elegir el modelo específico que más se ajuste a nuestras necesidades.

A continuación, presentamos dos de las plataformas más populares que hay en el mercado y de las que se han hecho uso durante la carrera.

4.2.1. Arduino

El entorno Arduino es con diferencia el más popular a la hora de desarrollar pequeños proyectos para el entorno doméstico [16]. Son tremendamente populares dentro de los aficionados a la electrónica e informática, pues tienen una gran comunidad detrás, lo que facilita enormemente la curva de aprendizaje para gente sin grandes conocimientos técnicos. Disponen de una gran cantidad de librerías para poder desarrollar cualquier proyecto.

La placa de desarrollo más popular es la Arduino Uno, esta placa utiliza un microcontrolador Microchip ATmega328P de 8 bits, dispone de varios puertos de entrada y salida, tanto analógicos como digitales. Resulta muy útil para realizar prototipos sencillos, pues únicamente dispone de 32 KB de memoria flash y 2 KB de RAM, por lo que no es capaz de almacenar aplicaciones muy largas.

Figura 15. *Placa Arduino Uno*

Otra placa popular en Arduino es la Zero, esta placa es una extensión de la Arduino Uno. Utiliza un micro SAMD21 de arquitectura de 32 bits, su memoria flash es de 256 KB y su RAM 32 KB, esto provoca que esta placa sea mucho más capaz que la Uno.

4.2.2. STM32

La familia de microcontroladores STM32 de STMicroelectronics está compuesta por MCUs de 32 bits basados en los microprocesadores Cortex-M de ARM. Existe una gran variedad de modelos disponibles y su orientación es más

profesional que la de los Arduino. Dentro de esta familia existen diversos grupos dependiendo de cuál sea el uso que se quiera hacer de ellos, desde el alto rendimiento hasta diseños centrados en un bajo consumo energético [17].

Existe una gran cantidad de herramientas de desarrollo para esta plataforma, además de librerías oficiales para la creación de proyectos en ella. A diferencia de en Arduino, aquí todas las soluciones disponibles tienen una orientación mucho más profesional y requieren un nivel técnico más elevado, esto dificulta más el desarrollo de proyectos, pero permite controlar y optimizar mucho mejor las aplicaciones desarrolladas.

La comunidad detrás de STM32, aun grande, no llega al tamaño de la de Arduino, pero no resulta complicado encontrar expertos sobre esta plataforma capaces de resolver dudas de un nivel técnico elevado, ya que muchos de ellos son profesionales que se dedican a desarrollar proyectos en estos microcontroladores como trabajo.

Debido a este carácter mucho más profesional de STM32 y a la existencia de una inmesa variedad de microcontroladores disponibles que permiten elegir el que mejor se ajuste a nuestras necesidades, se ha decidido hacer uso de esta plataforma por encima de Arduino.

4.3. Protocolo de comunicación

Para comunicar la información de los sensores a la plataforma online desde donde lo podrá visualizar el usuario, estos tienen que ser enviados de manera inalámbrica mediante algún protocolo de comunicación.

El sistema de comunicación a utilizar deberá ser capaz de alcanzar largas distancias de transmisión a la vez que tiene un consumo energético bajo, esto se corresponde con las llamadas LPWANs, acrónimo del término inglés “Low Power Wide Area Network”, su traducción al castellano, red de área amplia de baja potencia, ya nos da una primera definición de la función que tienen. Son redes que están pensadas para transmitir información entre puntos que se encuentren a una distancia considerable, su principal campo de aplicación es el entorno IoT. Los dispositivos que hacen uso de este tipo de redes no necesitan de un gran ancho de banda, generalmente la información enviada se corresponde pequeños paquetes para comunicar los datos recolectados por un sensor o activar algún actuador a distancia.

A diferencia de otras redes de transmisión de información como puedan ser el Wifi o bluetooth, las LPWAN suelen tener un uso más específico y no encontrarse de forma tan habitual en dispositivos de consumo como puedan ser los smartphones u ordenadores. El siguiente gráfico muestra la posición de las LPWAN respecto a otros tipos de comunicación inalámbrica:

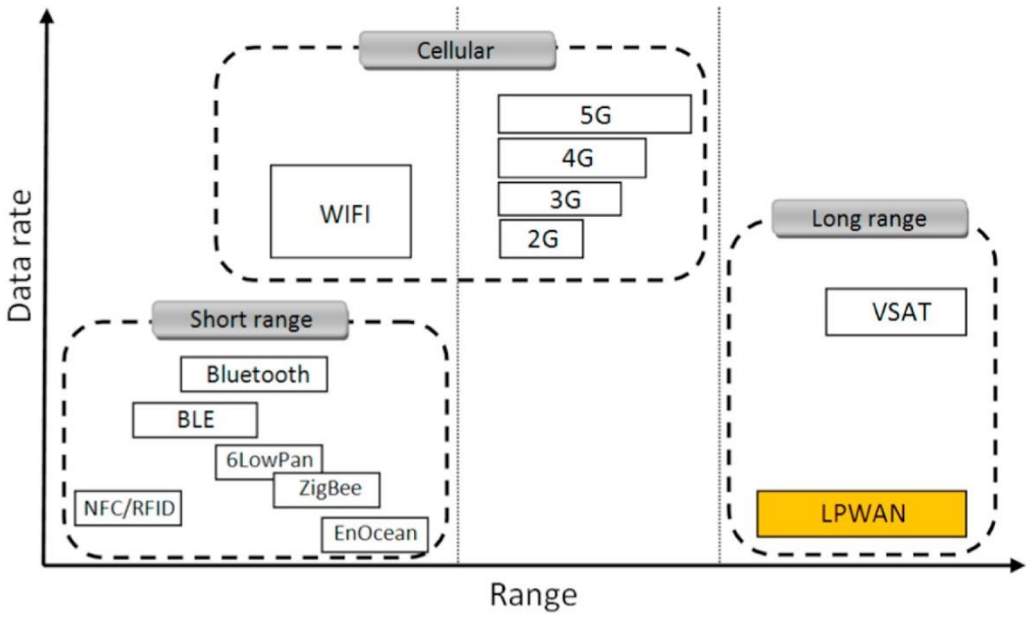


Figura 16. *El eje X indica el rango de alcance y el Y ancho de banda*

Como se puede apreciar en la gráfica, las LPWAN están destinadas a un sector en el que las otras tecnologías de comunicación no pueden desempeñarse adecuadamente por no funcionar con una distancia de transmisión suficientemente elevada. Otro inconveniente del resto de tecnologías que no aparece reflejado en la tabla es su mayor consumo energético.

Dentro del mundo de las LPWAN existen distintos enfoques, que han provocado que existan varias tecnologías similares para solucionar un mismo problema [18]. A continuación, expondremos tres de las principales tecnologías disponibles y las compararemos para ver cual resulta más idónea para las necesidades de nuestro proyecto.

4.3.1. LoRaWAN

LoRaWAN es un protocolo estándar de red LPWAN pensado para IoT. La tecnología sobre la que funciona LoRaWAN es LoRa, acrónimo de Long Range (Largo Alcance, en castellano) [19]. LoRa es un tipo de modulación de radiofrecuencia que fue desarrollado por la compañía estadounidense Semtech. Las principales virtudes de LoRaWAN son:

- Alta tolerancia a las interferencias
- Bajo consumo energético, hasta 10 años de autonomía
- Capacidad de penetrar en zonas urbanas densas o interiores
- Geolocalización sin necesidad de GPS
- Mantiene la comunicación con dispositivos en movimiento

Puede alcanzar los 50 km de distancia de transmisión. En situaciones idóneas de transmisión, sin ningún tipo de obstáculo entre emisor y receptor, se han alcanzado los 832 km de distancia [20].

Dependiendo del lugar en el que nos encontremos la frecuencia que se utilizará para LoRa será distinta, en América se utilizan los 915 MHz, en Europa los 868 MHz y en Asia, 433 MHz. En los tres casos se trata de licencias de uso libre, por lo que no es necesario ningún tipo de contrato o permiso para su uso.

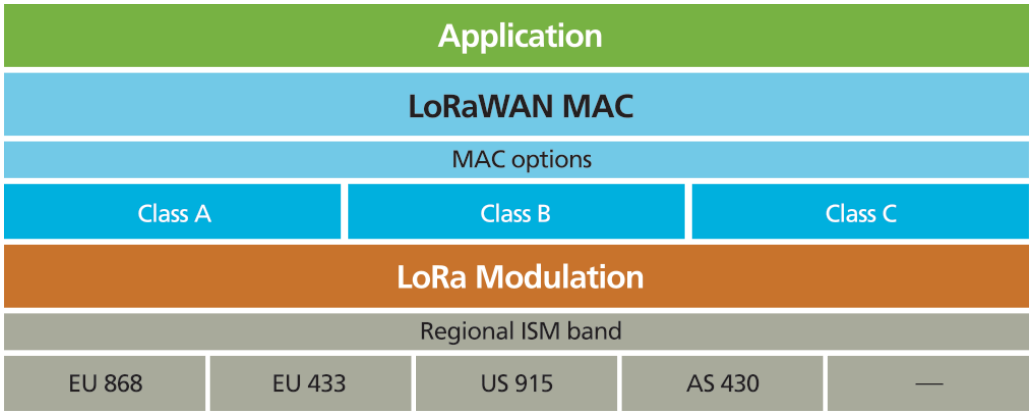


Figura 17. *Esta representación del stack de una aplicación IoT permite apreciar como LoRa funciona en un nivel de abstracción más bajo que LoRaWAN*

Las redes LoRaWAN tienen topología de estrella, esto significa que todos los nodos están conectados a un punto central, en primer lugar, a la gateway, y estas al servidor de red. Las comunicaciones LoRaWAN pueden encriptarse mediante AES-128 para mayor seguridad.

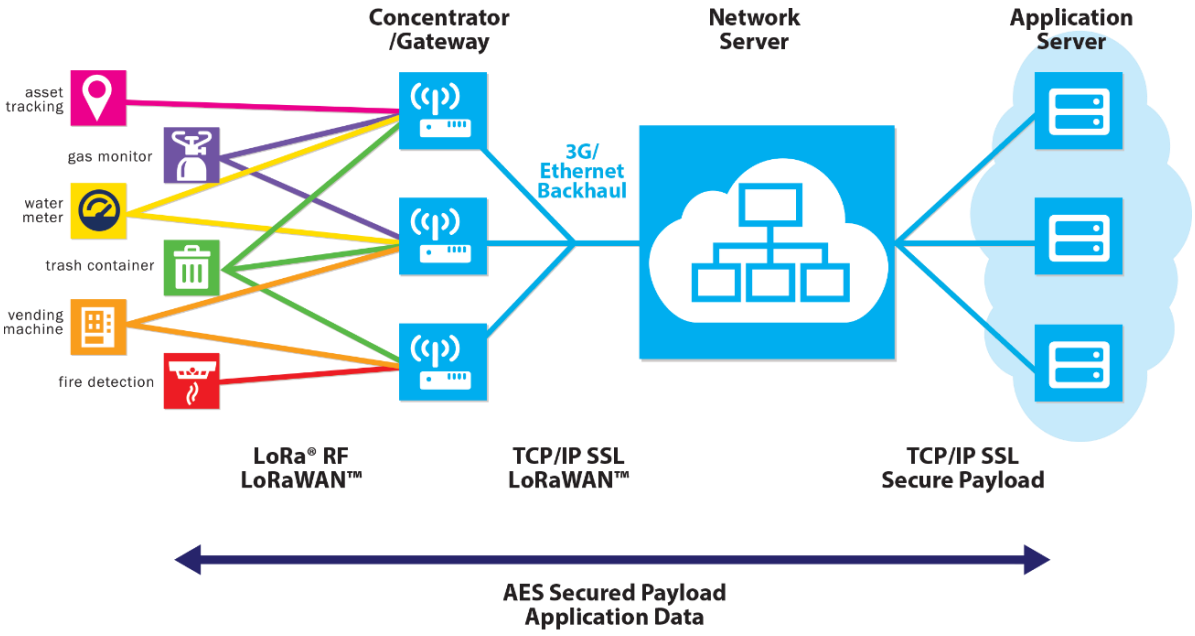


Figura 18. *Estructura de una red LoRaWAN*

El protocolo LoRaWAN establece tres clases de nodos:

- **Clase A**: es la clase más común, el nodo está en reposo y solo entra en modo escucha tras enviar información hacia la gateway.
- **Clase B**: en esta clase el módulo entra en modo escucha de forma cíclica siguiendo los periodos de tiempo que hayan sido determinados por el usuario
- **Clase C**: este modo fuerza al nodo a estar siempre en modo escucha, no es recomendado para dispositivos con batería, pues tiene un consumo energético mucho más elevado que los anteriores.

4.3.2. NarrowBand IoT

La tecnología NarrowBand IoT (abreviado NB-IoT) es un estándar abierto basado en la red LTE [21]. A diferencia del LTE, NB-IoT solo utiliza un ancho de banda de 200 kHz. NB-IoT tiene una velocidad de transmisión inferior a 100 kbps y una latencia de 1.5 a 10 segundos, estos valores no son muy altos, pero son suficientes para el objetivo de esta tecnología, transmisión de datos con baja periodicidad desde sensores en entornos urbanos y lograr un consumo energético reducido (hasta 10 años de autonomía). Gracias a su cobertura de hasta 20 dB de ganancia sobre GSM es utilizable tanto en interiores como en el exterior, aunque principalmente para dispositivos estáticos, pues no está diseñada para hacer seguimiento en tiempo real.

La principal diferencia de esta tecnología respecto a LoRaWAN es la red de comunicaciones que utiliza. LoRaWAN utiliza una frecuencia de uso libre (868 MHz en Europa) por lo que cualquiera puede instalar una Gateway propia para uso personal. Por el contrario, NB-IoT utiliza la banda de frecuencia del espectro licenciado de 800 MHz. Esta banda necesita de una licencia para poder instalar sistemas de comunicación sobre ella, por esta razón en NB-IoT no es el usuario final el que crea su propia red, sino que se utiliza la red de comunicaciones de las propias empresas de telecomunicación existentes. El proceso para lograr conectar los dispositivos NB-IoT a la red es bastante similar al necesario para conectar cualquier smartphone, el usuario necesitará una tarjeta sim que le proporcionará la compañía de telecomunicaciones y deberá pagar una cuota mensual para hacer uso de la red. En España la principal compañía que ofrece este servicio es Vodafone, que actualmente solo lo oferta para clientes profesionales, no para particulares. El hacer uso de las redes de telefonía móvil existentes permite tener garantizada la cobertura en casi cualquier punto del país y desentenderte de todo el proceso de instalación y mantenimiento de la red.

4.3.3. SigFox

La tercera solución planteada como red LPWAN es Sigfox. Sigfox es una compañía francesa que ha desarrollado su propio protocolo y red de telecomunicaciones [22]. Sigfox es, en cierta manera, el punto medio entre LoRaWAN y NB-IoT. Hace uso de una red de telecomunicaciones propia, pero a diferencia de NB-IoT lo hace a la misma frecuencia que LoRaWAN, 868 MHz en Europa. La cobertura de su red es bastante extensa a nivel nacional, teniendo cubiertas las principales ciudades del país.

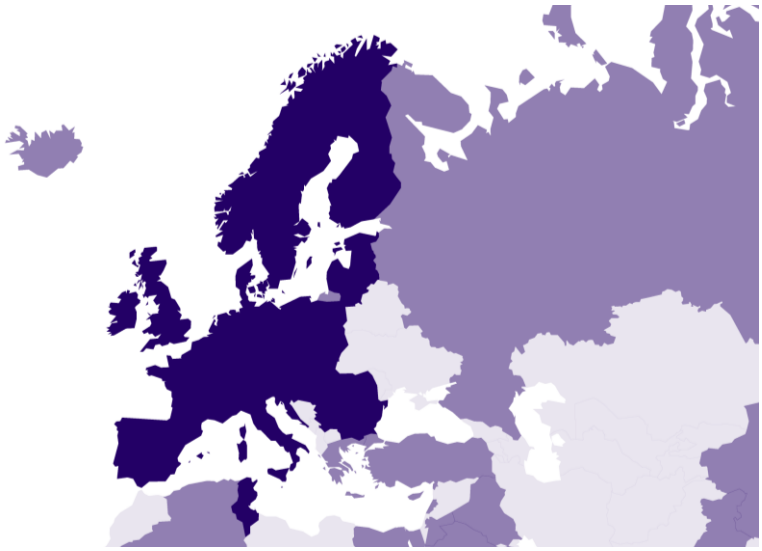


Figura 19. *Cobertura de Sigfox en azul oscuro las zonas cubiertas actualmente (junio 2020) y en morado las que serán cubiertas próximamente*

Sigfox dispone de toda la infraestructura necesaria para montar una red IoT, desde las gateways hasta los servidores en red, por lo que el usuario final únicamente necesita los dispositivos que desee conectar a esta. Para poder hacer uso de estos servicios el usuario debe pagar una subscripción anual a la compañía.

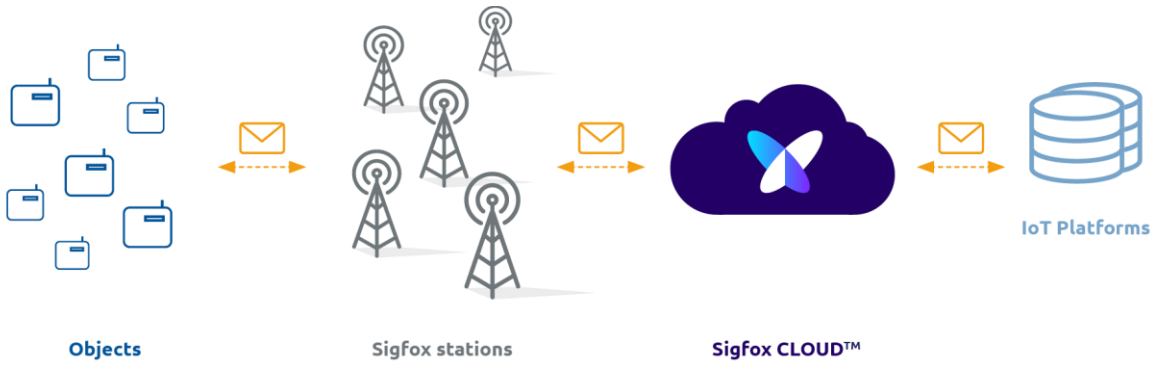


Figura 20. *Esquema de red de Sigfox, se puede apreciar que la topología de red es muy similar a la de LoRaWAN*

Sigfox permite al usuario enviar hasta 140 mensajes por día y dispositivo, los mensajes desde el nodo hacia la gateway tienen un "*payload"* (información que se transmite) máximo de 12 bytes, para mensajes en sentido opuesto el límite son 8 bytes.

El campo de aplicación de Sigfox es equivalente al de LoRaWAN, contando también con capacidad de usarse para la geolocalización de dispositivos. Como desventaja presenta la falta de encriptación y la imposibilidad de tener una red privada. Su alcance es aproximadamente 10 km en zonas urbanas y 40 km en zonas rurales.

Al compartir la frecuencia y ancho de banda con LoRa, es posible encontrar dispositivos capaces de utilizar ambas tecnologías, esto ayuda a reducir el precio de los módulos de comunicación por economía de escala, ayudando al usuario final a encontrar dispositivos a un precio más competitivo.

Como vemos, las tres tecnologías tienen una funcionalidad similar, la principal diferencia entre ellas es la manera en la que estructuran la red de comunicaciones. Para elegir que tecnología usar este apartado ha sido el más importante ya que las tres cumplen con las necesidades del sistema en cuanto a transmisión de mensajes.

En primer lugar, se ha descartado a NB-IoT por ser la menos madura de las tres, su uso todavía no está tan extendido como las otras dos y, aunque cuenta con la mayor red de todas, está orientada a un público profesional y a grandes empresas, por lo que resulta mucho más complicado conseguir una tarjeta SIM para hacer uso de ella.

Entre Sigfox y LoRaWAN, hemos optado por utilizar LoRaWAN. Aunque la red de Sigfox es extensa, no cubre por completo la zona más montañosa del interior de la provincia de Valencia, que es donde se encuentran los campos de caquis donde se quiere implantar el sistema. Además, al estar toda la infraestructura de red centralizada y controlada por Sigfox, esto reduce la capacidad de personalizar y controlar nuestra aplicación, obligándonos a utilizar una red pública y pagar una subscripción anual.

LoRaWAN nos permitirá crear la solución más personalizada de las tres opciones que había, siendo nosotros los que decidamos la gateway y el servidor de red de la aplicación, pudiendo crear una red privada que, aunque requiera un mayor desembolso inicial, nos permita no estar atados a subscripciones y poder ahorrar al largo plazo.

4.4. Módulo de comunicación del nodo

Una vez decidido que se va a utilizar LoRaWAN como tecnología de comunicación hay que decidir qué módulo de comunicaciones utilizar. Aunque el protocolo LoRaWAN sea "*open-source*", no ocurre lo mismo con la tecnología LoRa, que es propiedad de Semtech [23]. Ellos son los únicos fabricantes de transceptores para LoRa, por lo que la totalidad de módulos de comunicación para LoRaWAN utilizan uno de sus transceptores, aunque el fabricante de estos sea otra compañía distinta.

Los modelos de transceptor más comunes para nodo son los de las gamas SX126x y SX127x, las x representan el último digito, y diferencia los modelos por la frecuencia a la que funcionan. Como en Europa la frecuencia de uso es 868 MHz, nos centraremos en los dos principales modelos para ello, el SX1262 y el SX1276.

El SX1276 fue lanzado en 2013, tiene un factor de dispersión que puede ir desde 6 hasta 12, el ancho de banda es configurable desde 7.8 kHz hasta 500 kHz, el máximo nivel de transmisión es de 22 dBm, aunque la legislación europea no permite utilizar niveles superiores a 14 dBm. Es el modelo más común actualmente y, por tanto, el que más soporte y compatibilidad tiene. La comunicación con el microcontrolador se realiza mediante SPI.

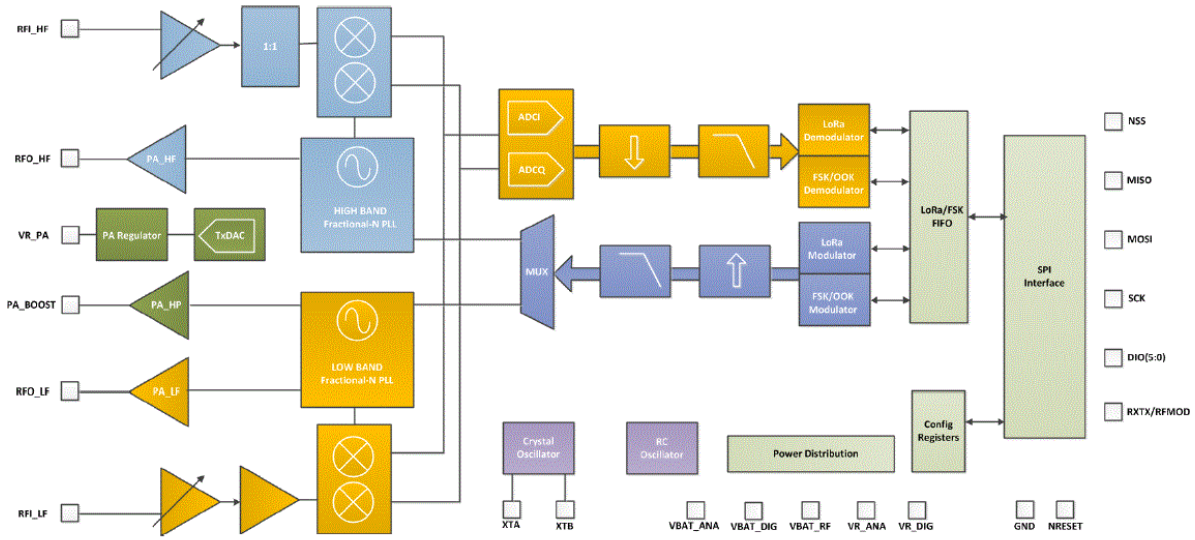


Figura 21. *Diagrama de bloques del SX1276*

El SX1262 fue lanzado en 2018 y es una versión que incluye ciertas mejoras respecto al SX1276. Las principales diferencias son la inclusión de un convertidor DC/DC de alta eficiencia que permite reducir de manera notable el consumo durante la recepción de paquetes hasta los 4.2 mA, mientras que el mínimo para el SX1276 es de 10.8 mA; el encapsulado de este módulo se ve reducido de los 6x6 mm hasta los 4x4 mm. Es esperable que se convierta en el transceptor más

popular en los próximos años, pero por ahora todavía domina el mercado el SX1276.

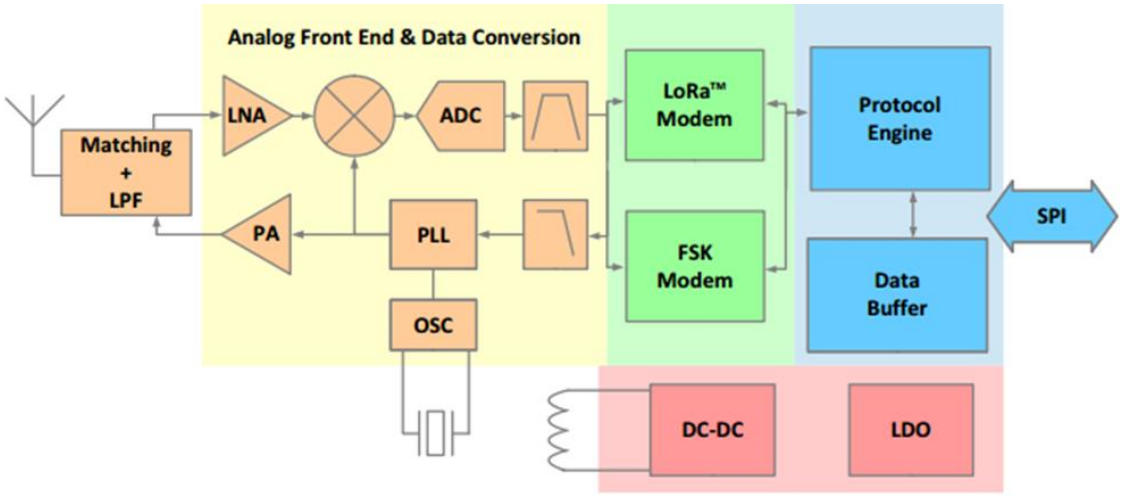


Figura 22. *Diagrama de bloques del SX1262*

4.4.1. CMWX1ZZABZ-078

Este módulo ha sido fabricado por Murata [24], en su interior de su encapsulado (12.5x11.6 mm) incluye tanto el transceptor SX1276 como un microcontrolador STM32L082 de bajo consumo, es capaz de funcionar tanto para la frecuencia europea como par la americana. Dispone de múltiples conexiones y puertos de entrada y salida, entre ellos destacan la presencia del ADC, DAC, interfaces UART, SPI, I2C; hasta 18 GPIO y USB. El MCU interno dispone de 192 kB de memoria flash y 20 kB de RAM.

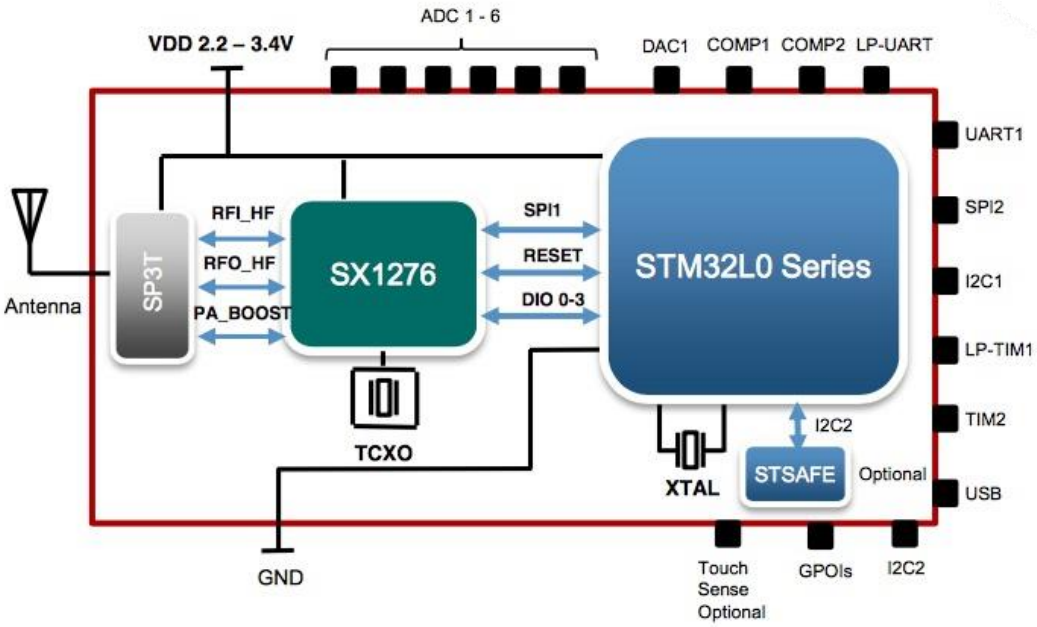


Figura 23. *Diagrama de bloques del CMWX1ZZABZ-078*

4.4.2. RFM95W

Este módulo de la compañía HopeRF está basado en el SX1276 [25], a diferencia del anterior, no incluye MCU, esta se conecta al transceptor mediante SPI. Incluye un sensor de temperatura e indicador de nivel de batería bajo. A nivel de especificaciones para la comunicación resulta muy similar al desarrollado por Murata. El módulo está compuesto por una PCB con *castellated holes* para poder soldarla otra donde se encuentren el MCU, los condensadores de desacoplo y el resto de los componentes de la placa desarrollada.

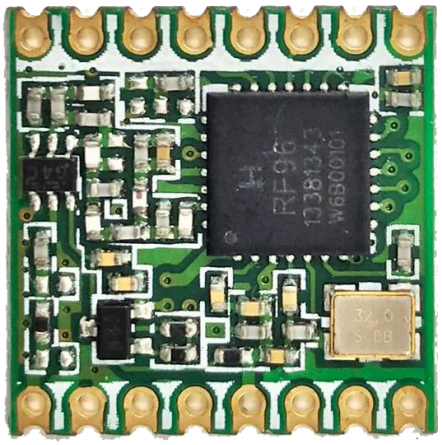


Figura 24. *Vista del módulo RFM95W*

La principal diferencia entre elegir un transceptor u otro es que, con el de Murata no será necesario añadir ningún MCU, mientras que en el de HopeRF sí.

Finalmente se ha decidido hacer uso del CMWX1ZZABZ-078 por incluir todo en un encapsulado menor y hace uso de un procesador de la familia STM32L0, la sección de STM32 orientada a la eficiencia y el bajo consumo, lo cual resulta idóneo para nuestro proyecto. Además, a pesar del pequeño tamaño incluye una gran variedad de interfaces de conexión, especialmente de I2C para poder conectar todos los sensores elegidos en el apartado 4.1. El punto más relevante que ha decantado la balanza a favor de Murata es la existencia de un kit de desarrollo para LoRaWAN de STMicroelectronics que utiliza este transceptor. La existencia de un kit de este tipo desarrollado por una compañía como ST nos da la seguridad de poder realizar todo el prototipo sin tener que preocuparnos por problemas en un apartado tan sensible como puede ser el hardware, especialmente cuando se trata de señales de alta frecuencia y comunicaciones, que siempre resultan más sensibles a las interferencias. El kit es el B-L072Z-LRWAN1[26], dispone de conectores compatibles con Arduino para poder añadir *shields* y ampliar las funcionalidades, se conecta al ordenador mediante USB e incluye un ST-Link para simplificar todo el proceso de carga *debug* del código sobre la propia placa. La placa también dispone de conector UFL para la

conexión de la antena y de un portapilas con espacio para 3 pilas AA, su precio es de unos 50 € dependiendo del vendedor.

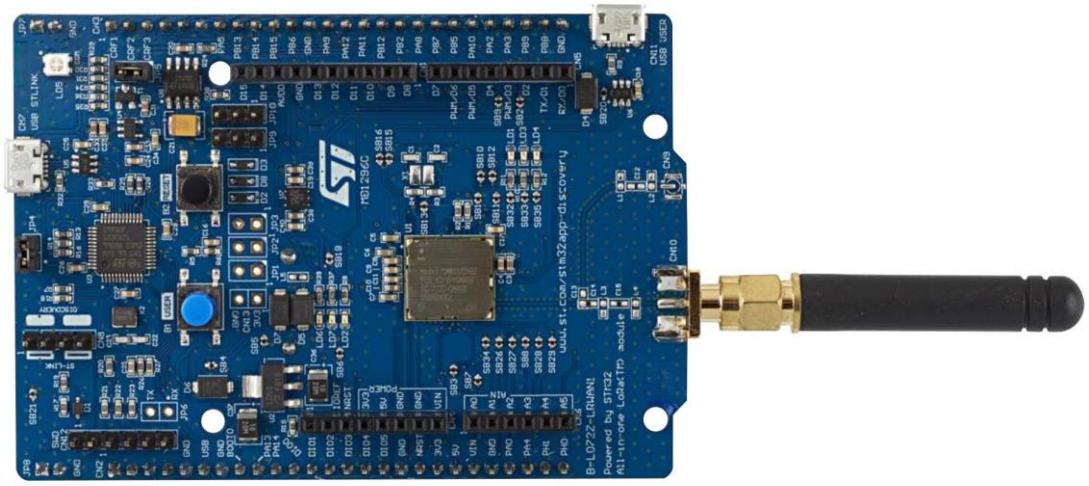


Figura 25. *Kit de desarrollo B-L072Z-LRWAN1*

4.5. Gateway

Una vez elegido el kit que actuara como nodo de nuestra aplicación es hora de elegir el dispositivo que ejercerá de gateway. El primer paso será decidir si se quiere utilizar una gateway de un solo canal o de varios. Si la gateway solo tiene un canal resultará más barata, pero es mucho más sencillo saturar el canal en de la gateway si varias aplicaciones con múltiples nodos tratan de conectarse a ella, debido a esto la propia LoRa Alliance no recomienda su uso, aunque pueden servir para aficionados con un conocimiento técnico limitado como una primera aproximación a LoRaWAN.

Siguiendo las recomendaciones de la LoRa Alliance vamos a utilizar una gateway con múltiples canales.

4.5.1. RAK2245

El fabricante chino RAK ha desarrollado esta placa con el módulo de Semtech SX1301. Consiste en una PCB que se conecta a la Raspberry Pi 3B+ o 4. Soporta hasta 8 canales de subida y 1 de bajada. Cumple con el protocolo LoRaWAN 1.0.2 y está disponible en distintas versiones dependiendo de la frecuencia de comunicaciones que se vaya a utilizar. Dispone de un módulo GPS integrado y un disipador para prevenir el sobrecalentamiento y el ruido térmico, esto permite alcanzar una potencia de transmisión de hasta 27 dBm.

Esta gateway es la sucesora del popular modelo RAK831, añadiendo como mejoras la capacidad de comunicar el módulo GPS con la Raspberry Pi mediante I2C, haciendo uso de una memoria EEPROM conforme la especificación Pi HAT

para la conexión entre la Raspberry y otras placas que le aporten funciones adicionales, por último, también incluye el disipador térmico.

El hacer uso de una Raspberry Pi como base sobre la que se integra el RAK2245 permite lograr una gateway con una alta capacidad de personalización y la posibilidad de integrar varias funciones en un solo dispositivo, pudiendo ejercer la Raspberry simultáneamente de gateway y servidor de red, por ejemplo. Actualmente la combinación de Raspberry más RAK2245 resulta la combinación más popular para particulares que quieren lograr una gateway con varios canales y una gran relación calidad/precio, una Raspberry 3B+ cuesta en torno a 40 € y el módulo de RAK 120 €. Este precio resulta para el usuario mucho más asumible que los precios habituales de otras gateway de orientación más profesional y con una capacidad de personalización mucho más cerrada. Existe una gran comunidad detrás de esta placa por lo que es sencillo resolver cualquier duda o problema que se sufra durante la configuración y utilización de la gateway.

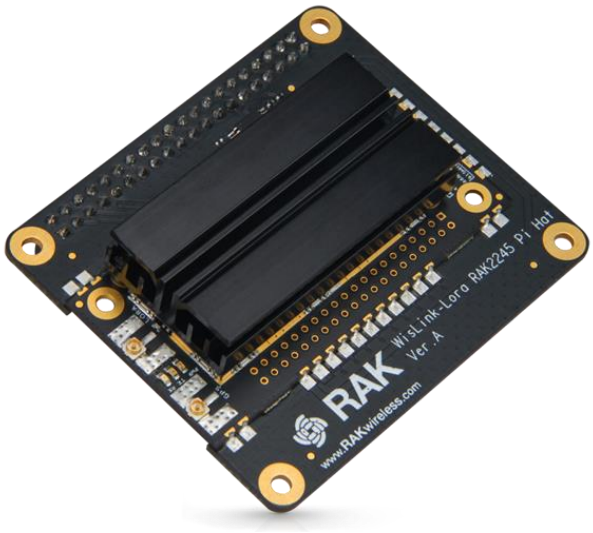


Figura 26. *RAK2245*

La RAK2245 en la mayoría de casos no es la gateway que se utilizaría para desarrollar un proyecto que busque una gran capacidad de transmisión, con la gateway por tanto situada en el exterior, a grandes distancias o un nivel alto de fiabilidad como haría falta en un proyecto que pueda desarrollar una empresa para desarrollar una gran red inalámbrica.

Sin embargo, permite desarrollar de manera asequible redes con varios sensores para pequeñas soluciones IoT y adquirir conocimiento sobre la configuración de la gateway, LoRaWAN y telecomunicaciones en general a estudiantes o usuarios con conocimientos medios que busquen ampliarlos o dedicarse a ello profesionalmente en un futuro. RAK vende una carcasa de aluminio para poder tener una solución completa de apariencia profesional e

instalarla en interiores, pues no garantiza la protección respecto a las inclemencias meteorológica.

Figura 27. *Carcasa de aluminio para la RAK2245 y la Raspberry Pi*

4.5.2. The Things Indoor Gateway

Esta gateway ha sido fabricada por The Things Industries, empresa creada por los fundadores del servidor The Things Network. Esta gateway dispone de 8 canales y conexión wifi de 2.4 GHz. Puede alimentarse desde la red eléctrica o a través de un puerto USB tipo C. Lleva las antenas integradas por lo que tiene un alcance muy limitado de apenas unos cientos de metros. Debido a su alcance no sería posible utilizarla para una aplicación real, pero puede servir para uso en laboratorio cuando lo que se busca es simplemente probar que se realiza correctamente la conexión entre los nodos y el servidor de red. Aunque la gateway es capaz de funcionar con cualquier servidor de red, está principalmente diseñada para su uso con The Things Network, pues su configuración para ellos es mínima. Esta gateway es el modelo de 8 canales más económico del mercado con un precio de 85 €

Existe una variedad de gateways mucho mayor a partir de los 300 €, pero ese precio resultaba inasumible por nuestra parte, por lo que la decisión lógica era elegir la RAK2245, puesto que al contar actualmente con una Raspberry Pi 3B+, el desembolso total sería de 120 €.

4.6. Servidor de red

El servidor de red es un ordenador o grupo de ordenadores orientado a transmitir la información recibida de manera segura y fiable a lo largo de toda la red. Un servidor red puede estar físicamente disponible para el usuario o puede hacerse uso de servidores de terceros que, aunque no se disponga de acceso físico a ellos, se pueden configurar y controlar mediante internet de manera remota.

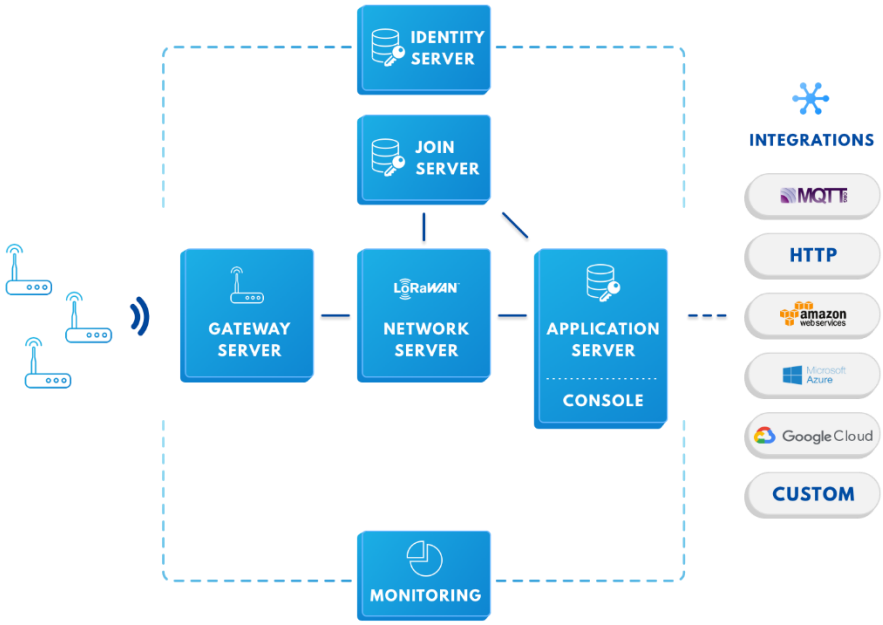


Figura 28. *Representación de una red, el servidor de red (Network Server) es el punto central de conexión dentro del stack de LoRaWAN*

4.6.1. The Things Network

The Things Network (TTN) es posiblemente el servidor red más popular en LoRaWAN, es un proyecto de código abierto lanzado en los Países Bajos en 2015. Su objetivo es crear una red abierta y descentralizada para LoRaWAN [28] donde los usuarios pueden conectarse a cualquiera de las gateways que otro usuario de la red haya configurado, con esto se logra crear una gran red en la que particulares que no dispongan de una gateway puedan acceder a la red de manera gratuita. Existe una gran cantidad de gateways distribuidas por todo el mundo y el número sigue aumentando conforme aumenta el tamaño de la comunidad.

Desde la consola online se realizan todas las operaciones necesarias, configurar y controlar las gateways, si se dispone de una propia; configurar las aplicaciones propias, registrar dispositivos, realizar la integración con las API y ver el tráfico de datos.

El foro de The Things Network es uno de los más grandes de internet en lo respectivo a LoRaWAN, por lo que el soporte disponible es casi ilimitado.

Al ser TTN un proyecto abierto y gratuito, cualquiera puede hacer uso de sus gateways. Esto puede provocar que una de ellas se sature si hay demasiados nodos intentándose conectar a ella de manera simultánea, por lo que no es recomendado para uso profesional. Para solucionar este problema, The Things Industries ofrece un servicio equivalente a TTN, pero orientado a su uso profesional, donde no existe una red abierta, sino que cada cliente dispone de la

suya propia, diversas opciones de alojamiento para el servidor y soporte técnico profesional. La idea es que un usuario pueda realizar los primeros pasos del proyecto de manera gratuita en la red abierta de TTN y posteriormente migrarlo todo automáticamente a la red privada de The Things Industries.

4.6.2. ChirpStack

ChirpStack, anteriormente conocido como LoRaServer, es un servidor para LoRaWAN de código abierto pensado para que cualquier usuario pueda descargar el software e instalarlo en su servidor propio para crear una LPWAN. Disponen también de un sistema operativo para Raspberry Pi, llamado Gateway OS, que permite convertir a esta en un servidor de red, por lo que junto a una RAK2245 se podría tener una gateway y un servidor de red en un solo dispositivo, de manera local y accesible al usuario.

ChirpStack dispone de un nivel de configuración mayor a TTN, permitiendo al usuario obtener un mayor nivel de personalización en el sistema, esto es una gran ventaja para usuarios avanzados, pero puede ser un inconveniente si no se tiene cierto grado de experiencia con el desarrollo de redes LPWAN.

Al ser este un proyecto de código abierto, cualquiera puede hacer una copia para modificar a su gusto y utilizarla para la solución específica que necesite, por lo que es útil para proyectos profesionales donde no se requiera de un soporte técnico externo.

Ambas soluciones para el servidor de red son válidas, pero para un correcto uso ChirpStack es necesario tener una mayor experiencia con redes LPWAN que para TTN, como actualmente no disponemos de dicha experiencia, hemos decidido utilizar TTN, para poder centrarnos en el desarrollo de la propia aplicación en sí y no tener que dedicar tanto tiempo a una parte menor del proyecto como sería toda la configuración e instalación de ChirpStack, pues este no es el objeto de este trabajo.

4.7. API

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de definiciones, protocolos, subrutinas y procedimientos. Funcionan como capa de abstracción para comunicar distintos subsistemas o aplicaciones sin necesidad de saber cómo están implementados cada uno de ellos. Las APIs ayudan a simplificar tareas complejas al añadir capas de abstracción, esto también trae como consecuencia negativa la perdida de flexibilidad al tener que limitarse al entorno ofrecido por la API.

Las APIs resultan muy útiles a la hora de crear interfaces gráficas que resulten amigables para el usuario. En nuestro caso, se busca que desde la interfaz puedan verse los distintos valores captados por los sensores, activar el sistema

de riego y seleccionar las preferencias del usuario sobre que notificaciones quiere recibir.

A la hora de elegir una API para nuestro proyecto, vamos a buscar una que disponga de integración directa con The Things Network para simplificar el proceso y asegurarnos una compatibilidad total.

4.7.1. Cayenne myDevices

Cayenne permite crear soluciones completas para IoT, desde la recepción de los datos hasta la interfaz del usuario, pasando por la monitorización de las medidas de los sensores, envío de notificaciones o creando acciones para responder automáticamente conforme el usuario haya decidido [30].

La integración de Cayenne con TTN es muy sencilla, solo necesitaremos registrar nuestra aplicación en la plataforma myDevices y enviar los datos desde el nodo con el formato de *payload* de Cayenne. La plataforma sabrá automáticamente que clase de datos está recibiendo y creará por si misma los *widgets* la interfaz gráfica o *dashboard* desde donde se verán los últimos valores recibidos y se podrá acceder a los gráficos con el registro histórico. Desde la interfaz también se pueden crear objetos para enviar información desde la plataforma hasta uno de los nodos (por ejemplo, para activar una electroválvula de riego determinada) o hacer que si recibe cierto valor en un parámetro concreto automáticamente responda a ello conforme el usuario haya decidido previamente.

La plataforma de Cayenne resulta muy intuitiva y al ser completamente online puede accederse a ella desde móvil y ordenador.

El uso de Cayenne es gratuito, pero si se quieren realizar soluciones profesionales con una aplicación para terceros será necesario optar por las opciones de pago de la plataforma myDevices en la que se encuentra integrada Cayenne.

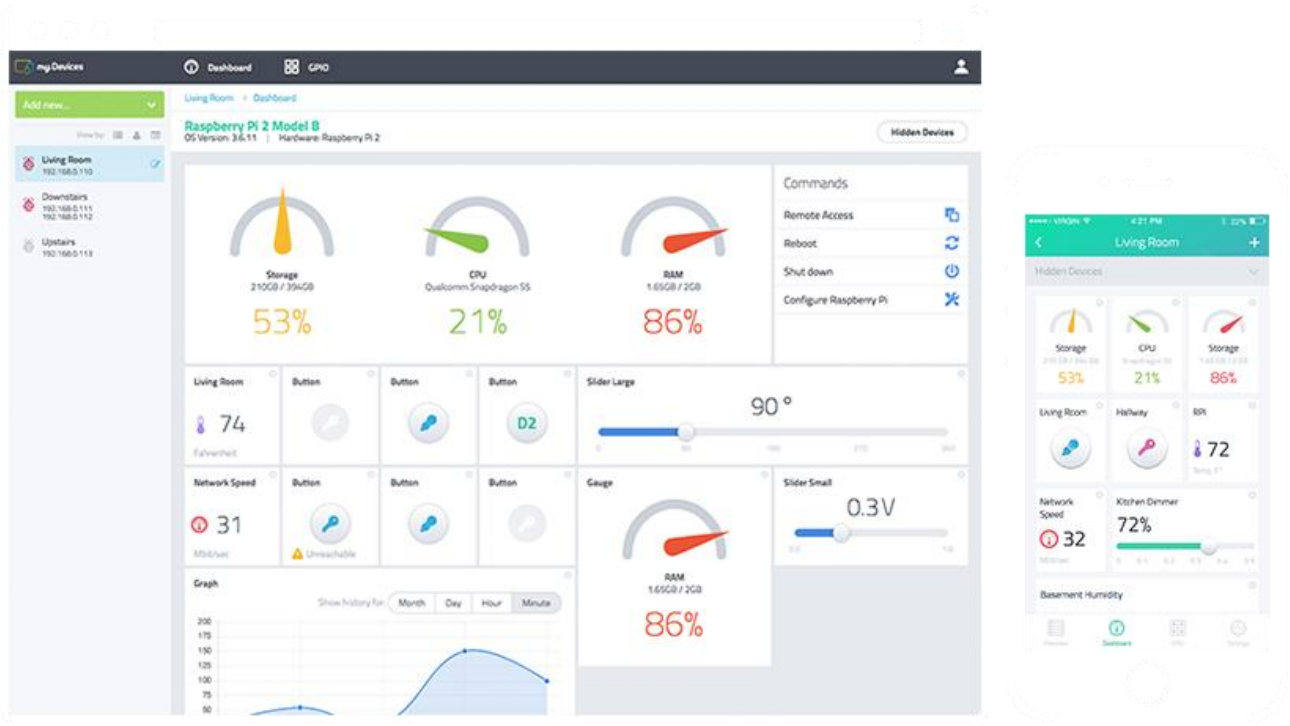


Figura 29. *Interfaz de Cayenne para la misma aplicación en PC y móvil*

4.7.2. TagoIO

TagoIO es una plataforma que ofrece todas las herramientas para gestionar los dispositivos, almacenar los datos y analizarlos, todo combinado en una aplicación de uso sencillo [31].

Al igual que Cayenne, TagoIO tiene integración directa con TTN, por lo que su configuración resulta sencialla.

Una ventaja de TagoIO sobre Cayenne es la posibilidad de añadir tus propios *scripts* para analizar los datos en tiempo real. Otra opción interesante de la plataforma es el poder simular los dispositivos para situaciones en la que no sea posible conectar los nodos, esto ayuda a reducir los retrasos en el desarrollo del proyecto y evitar que un problema en un apartado lo bloquee por completo.

Al contrario que en Cayenne, aquí los *widgets* no se generan automáticamente en el *dashboard,* hay que añadirlos manualmente, esto ralentiza el tener un *dashboard* operativo, pero permite personalizarlo mucho más. Además, es posible utilizar *dashboards* creados y compartidos por otros usuarios, por lo que no hay que empezar desde cero cada vez que haya que hacer la interfaz para una aplicación.

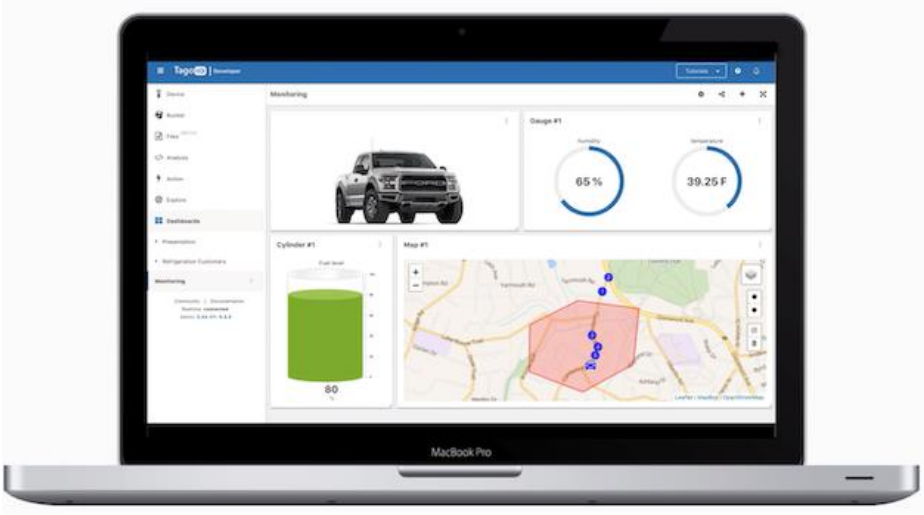


Figura 30. *Vista del dashboard en ordenador*

TagoIO ofrece tres planes de uso, un plan gratuito en el que se limita el número de dispositivos a 5, se reducen los *widgets* disponibles y la capacidad de analizar los datos en tiempo real se ve mermada. Los otros dos planes están pensados para desarrolladores, no tienen las limitaciones del plan gratuito. La mayor diferencia entre ellos es que uno está limitado a 100 dispositivos y el otro no tiene límite de dispositivos.

Una vez analizadas ambas opciones para la API, se ha decidido optar por Cayenne. Las dos razones tras esta decisión son, la generación automática de los *widgets* y que, aunque TagoIO dispone de ventajas en el análisis de datos en tiempo real, esto queda actualmente fuera de nuestro conocimiento técnico y sería necesaria una suscripción de pago para poder sacarle provecho.

4.8. IDE

El entorno de desarrollo integrado, abreviado IDE [32], es la aplicación informática desde la que el programador puede llevar a cabo el desarrollo del software. Es habitual que al IDE lo conformen los siguientes elementos:

- **Editor de código**: editor de texto al que se le han añadido funciones para facilitar la escritura de código, resalta la sintaxis de manera visual, dispone de función para autocompletar e indicar si se produce algún error de formato mientras se escribe el código
- **Compilador**: se encarga del compilado del código, puede haber más de uno disponible dentro de un mismo IDE, puede elegirse el nivel de optimización, así como configurar ciertos parámetros para adaptar la compilación al estándar del lenguaje que se esté utilizando
- **Depurador**: sirve para revisar el código ya compilado y su ejecución en el MCU, da la opción al programador para establecer puntos del código donde se detiene la ejecución y así comprobar los registros de memoria

para asegurarse que la ejecución está siendo la esperada. En IDEs muy básicos puede no estar disponible la opción de depurar

Una vez entendida la función del IDE, habrá que elegir el que mejor se adapte a nuestras necesidades. El código que vamos a escribir será en lenguaje C y C++ así que se buscará un IDE orientado a ello.

4.8.1. Keil uVision 5

Uno de los IDEs más utilizados en entornos profesionales para la programación de microcontroladores. Esto se debe a ser la primera compañía en implementar un compilador para C, disponer de un potente depurador integrado y permitir un alto nivel de personalización al usuario. En 2005 Keil fue adquirido por ARM.

Keil uVision 5 dispone de una versión gratuita en la que se limita el tamaño de los proyectos a compilar a los 32 KB. Si se quiere utilizar para proyectos de mayor tamaño, resulta necesario adquirir una licencia de uso. Estas licencias tienen un coste anual de varios miles de euros. Por suerte, Keil ofrece licencias gratuitas para proyectos que vayan a utilizarse en las familias STM32F0, STM32G0 o STM32L0, esto resulta para nuestra aplicación, que utiliza un micro de la familia STM32l0.

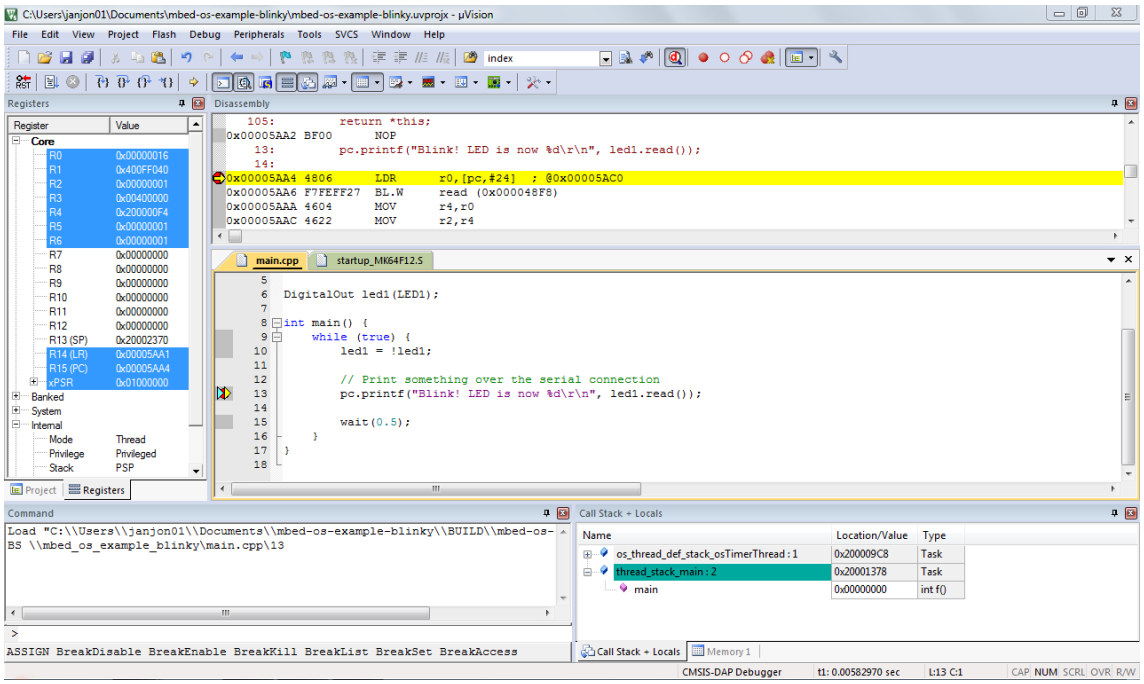


Figura 31. *Interfaz de Keil uVision 5 durante el depurado del código*

4.8.2. Mbed IDE

Mbed es una plataforma online de código abierto desarrollada como proyecto colaborativo y dirigida por ARM, fue lanzada en 2009, está orientada a la programación de microcontroladores de 32 bits con arquitectura ARM Cortex-M [35].

Este IDE es online, se acede a él a través de un navegador web, esto supone que el compilado no se realice utilizando el propio ordenador del usuario, sino el servidor dónde se encuentra alojado el IDE. La plataforma online incluye un forja o *forge* para alojar los repositorios con el código de los proyectos elaborados por los usuarios, esto facilita la importación de librerías de terceros a los proyectos propios.

Al ser una plataforma online, puede utilizarse en un ordenador para desarrollar una parte del código y continuar en otro distinto sin tener que copiar ningún archivo, únicamente hará falta iniciar sesión en el nuevo ordenador con la cuenta del usuario.

El mayor inconveniente que tiene Mbed es que no es capaz de depurar el código compilado en la plataforma, es cierto que permite exportar los proyectos a multitud de IDEs que disponen de esta función, pero esto obliga al usuario a tener que utilizar un segundo programa, cuando la mayoría de IDEs te permiten realizar el proceso completo de desarrollo del código sin tener que utilizar software adicional.

Mbed ha lanzado también Mbed OS, un sistema operativo diseñado para procesadores ARM y pensado para utilizarse en dispositivos IoT.

En junio de 2020 se lanzó la versión 1.0 de Mbed Studio, un IDE basado en Mbed OS que, a diferencia de la versión original de Mbed, está pensado para ser descargado y ejecutado de manera local por el usuario.

4.8.3. STM32CubeIDE

Este es el IDE más reciente de los tres, lanzado en 2019 por STMicroelectronics, nace de la fusión de Atollic TrueStudio con STM32CubeMX [35].

TrueStudio era un popular IDE basado en el *framework* Eclipse/CDT y escrito en Java que fue adquirido por STMicroelectronics. Incluye un depurador profesional basado en GNU gdb *debugger* capaz de depurar sistemas con varios procesadores. TrueStudio permite integrar clientes para la gestión de versiones de manera sencilla.

STM32CubeMX es el software STMicroelectronics para realizar toda la configuración base de un MCU de manera visual y guiada paso a paso. Permite realizar toda la configuración de los periféricos, reloj interno, iniciación de las interfaces de comunicación y calcular el consumo del MCU. El proyecto generado se importa al IDE elegido para trabajar sobre él.

STM32CubeIDE es como TrueStudio con la ventaja de tener STM32CubeMX integrado para poder crear el proyecto base desde el que trabajar desde el propio IDE. Este IDE esta obviamente orientado al desarrollo de software para los microprocesadores de STMicroelectronics al ser un software propio de la

compañía. Su objetivo es ayudar a simplificar todo el proceso de creación de sistemas embebidos basados en STM32, es completamente gratuito.

El proyecto de Mbed resulta ambicioso y recibe actualizaciones con mejoras sustanciales de manera continua, pero el no contar con un depurador es un problema excesivamente grande como para optar por él. Posiblemente en un futuro Mbed Studio resulte una opción a tener en cuenta, pero actualmente se encuentra en un estado muy prematuro, llevando apenas unos meses en el mercado.

Si el microcontrolador a utilizar fuese de cualquier familia distinta a la STM32L0, F0 o G0, la opción lógica sería optar por STM32CubeIDE que, aunque es un IDE reciente, realmente lleva años siendo popular entre los desarrolladores, solo que lo hacía bajo otro nombre. Como el procesador a utilizar es de la familia STM32L0, podemos utilizar una versión completa de manera gratuita, además a lo largo de la carrera se hemos hecho uso de este IDE en varias asignaturas, por lo que ya estamos familiarizados con él. En caso de no tener experiencia con ninguno de los dos IDEs, STM32CubeIDE podría resultar una buena opción para el proyecto, pero dado que se cuenta con experiencia previa con Keil, la elección de este resulta la opción más lógica.

5. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

Este apartado tiene como objetivo exponer el proceso completo que ha supuesto el desarrollo del prototipo, explicando los pasos seguidos y el razonamiento de las decisiones que se han ido tomando durante el proceso. Se ha dividido el apartado en dos subsecciones, la primera consiste en el proceso del desarrollo del nodo y la segunda en el de la configuración de la plataforma web.

5.1. Placa de sensores para el nodo

5.1.1. Desarrollo del hardware

Para desarrollar cualquier placa el primer paso es decidir los componentes que la conformarán, este proceso es posiblemente el más largo de todos, pues implica estudiar las diferentes opciones que cada fabricante presenta para cada tipo de componente y compararlos entre ellos para ver que opción resulta más idónea, teniendo en cuenta multitud de factores.

5.1.1.1. Creación de los esquemáticos

Los esquemáticos son el principal objeto de consulta, junto al *datasheet,* a la hora de intentar entender el funcionamiento de un circuito, por ello es importante tratar de hacerlos de manera clara y concisa.

En nuestro caso todo el proceso de desarrollo del circuito se ha realizado utilizando el software de acceso libre KiCad.

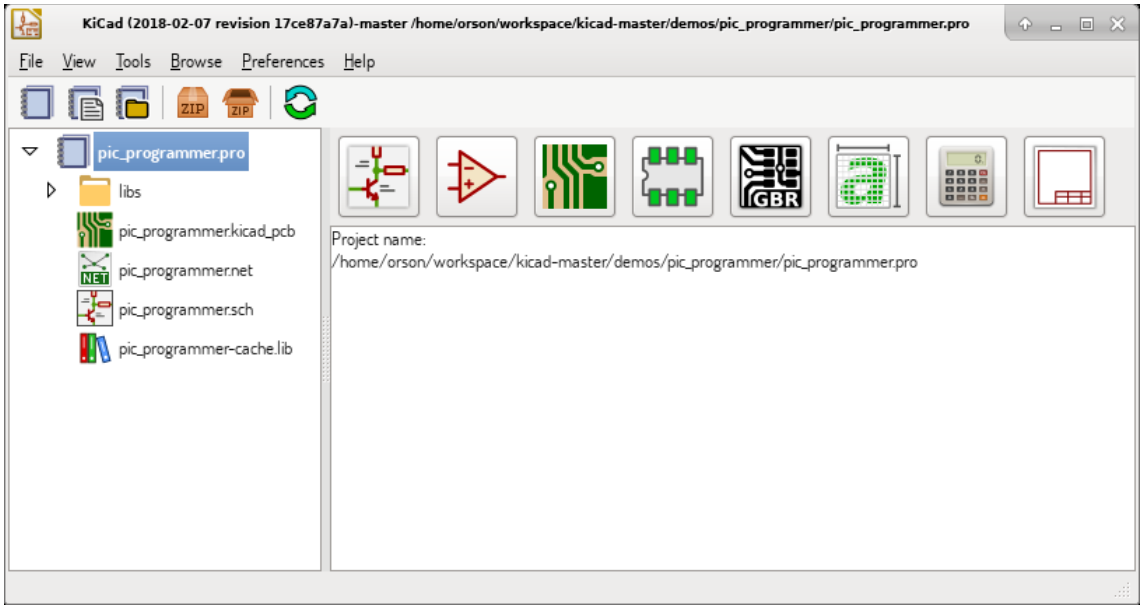


Figura 32. *Ventana principal de KiCad*

Una vez se han descargado o creado todos los modelos de los componentes que se vayan a utilizar (recalcar que el concepto modelo hace referencia a la representación gráfica que tendrá el componente en el esquemático), deberemos ir organizándolos y uniéndolos para lograr el circuito deseado. Este proceso se va a explicar en mayor detalle para cada parte del circuito en los apartados siguientes. Los esquemáticos completos del circuito se encuentran disponibles dentro del documento *2. Planos* de este proyecto, dentro de los planos hay cuatro hojas distintas, la primera indica la conexión entre las distintas partes del circuito, la segunda nos muestra el conexionado del módulo principal, CMWX1ZZABZ; el tercero los conectores presentes en el diseño y el cuarto los sensores utilizados.

5.1.1.2. I2C

El bus I2C es uno de los más sencillos desde el punto de vista eléctrico, es de baja velocidad por lo que las interferencias que existen con señales de alta velocidad no son un problema relevante y únicamente necesita de dos resistencias de *pull-up* para mantener en estado alto por defecto a SDA y SCL.

El I2C funciona a un voltaje concreto para todo el bus, en este caso a los 3,3 V que proporciona el máster, esto provoca que en caso de conectar algún dispositivo I2C externo que funcione a 5 V, este no lo hará de manera correcta. Para solucionar este problema se ha conectado un *Level Shifter,* este componente se encarga de adaptar a un bus I2C otros dispositivos que necesitan de valores de bus distintos, crea un bus equivalente al que contiene al máster y

"traduce" la información que se comunica de uno a otro. El modelo a utilizar es el popular modelo de Texas Instruments, PCA9306.

5.1.1.3. Sensores y actuadores

Los tres sensores que se han utilizado en este desarrollo son bastante sencillos en cuanto a conexiones eléctricas, se ha decidido no hacer uso de los pines para interrupciones que tiene disponibles en los encapsulados. En el caso del HDC2080 también se ha tenido en cuenta de que dependiendo de cómo se conecte el pin ADDR la dirección I2C del sensor cambiará.

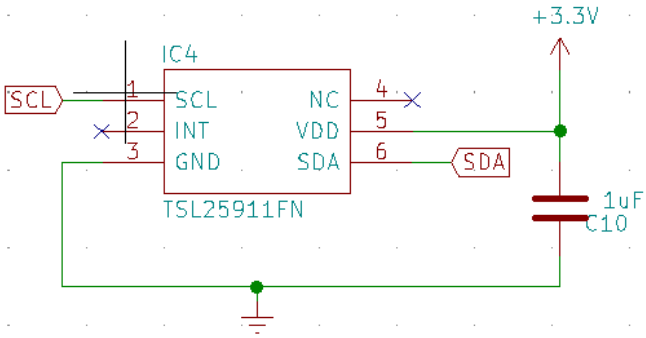


Figura 33. *Vista de las conexiones del TSL2591*

Para activar la electroválvula se decidió hacer uso de un pequeño circuito externo diseñado específicamente para la válvula que se decidió utilizar, por lo que de cara al diseño de la placa únicamente será necesario recordar que alguno de los pines esté libre para poder conectar la válvula.

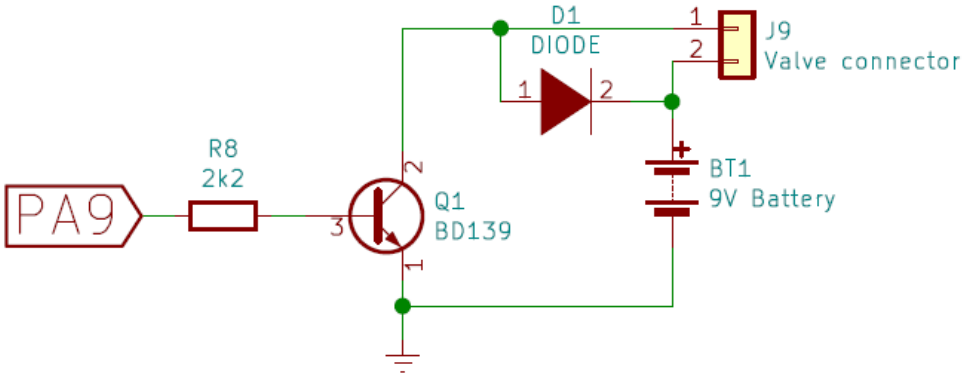


Figura 34. *Circuito de la electroválvula*

5.1.1.4. MCU

La MCU es, con holgada diferencia, el componente con mayor cantidad de pines. Para entender qué pines necesitan ser conectados y a qué, se ha tenido estudiar el *datasheet* y basar el diseño que se ha realizado en el propio diseño recomendado por el fabricante.

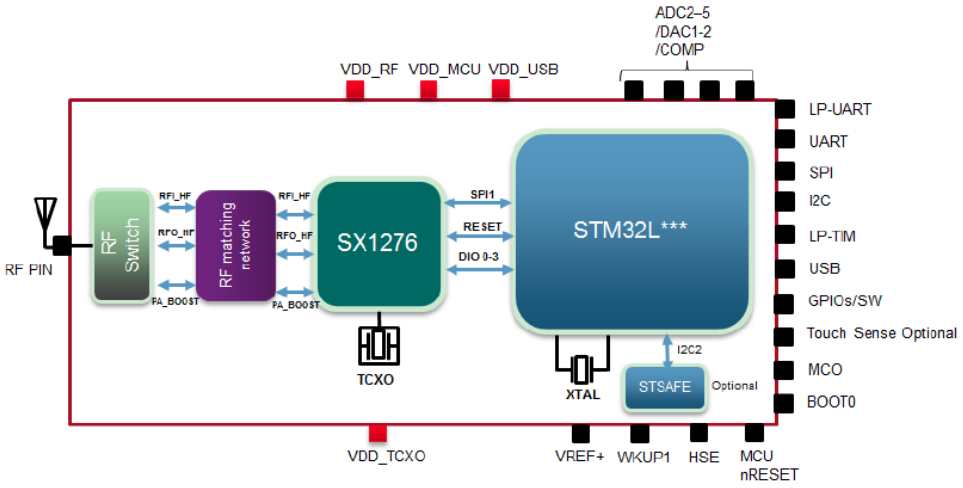


Figura 35. *Diagrama de bloques del CMWX1ZZABZ-078 en el que se aprecian algunos de los pines de los que dispone*

A la hora de hacer el circuito del MCU se decidió incluir un pulsador para poder reiniciar la placa manualmente, así como conexiones para antena tanto SMA como UFL para que se puedan conectar distintos tipos de antenas.

Figura 36. *Conectores SMA y UFL utilizados para antenas*

Para cualquier diseño de un MCU es importante seguir las recomendaciones del fabricante en cuanto a qué condensadores de desacoplo incluir, esto ayudará a tener un voltaje de alimentación más estable y nos evitará problemas posteriores en los buses de comunicación que puedan deberse a no alcanzar los valores adecuados de alimentación.

Aunque realmente el MCU que se ha utilizado mayormente en el proyecto es el que se encuentra en el B-L072Z, se decidió incluirlo en la placa con vistas a, en un futuro, desarrollar el software específicamente para esta placa y así prescindir del kit de STMicroelectronics. La razón de no excluir el B-L072Z desde el principio es que es una placa desarrollada por uno de los gigantes de la industria que nos da la seguridad de no tener ningún problema en el aspecto del hardware y que dispone de las librerías para LoRaWAN, mientras que si usásemos solo

nuestra placa hubiese sido necesario realizar muchas más pruebas para asegurarnos de que la comunicación inalámbrica funciona correctamente y hubiésemos tenido que desarrollar las librerías específicamente para nuestra placa en lugar de utilizar las proporcionadas por STM. Todo este proceso hubiese supuesto un desarrollo que es mucho mayor del esperado para un trabajo final de grado.

5.1.1.5. Desarrollo de la placa

Una vez se acabó de desarrollar los esquemáticos se contino con el diseño de la PCB. Este paso consiste en representar las conexiones y los propios componentes en la placa de cobre y dieléctrico que se va a utilizar. La primera tarea para llevar este proceso a cabo es crear los *footprints* o huellas de los componentes [36]. Un *footprint* es la proyección de los pads que se unirán mediante estaño a la placa. El correcto diseño de estas huellas resulta crucial, si no son de los tamaños adecuados es posible que no se pueda soldar a la placa el componente o que, si lo hace, este no funcione correctamente. Debido a esto, es habitual que el propio fabricante proporcione las especificaciones del *footprint* del componente en el datasheet.

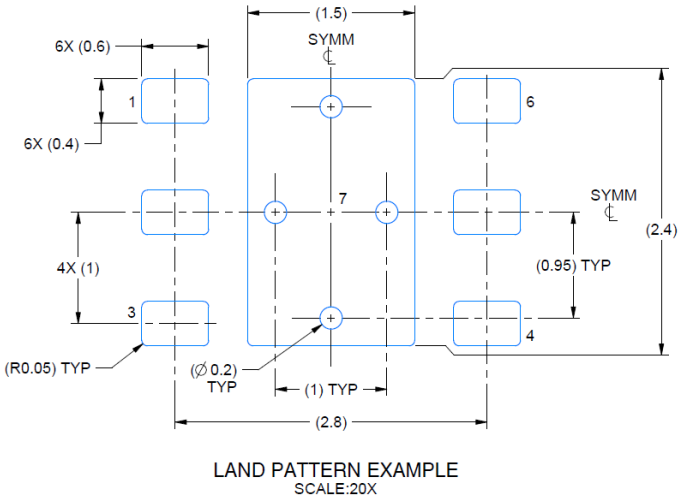


Figura 37. *Footprint del HDC2080. Footprint del HDC2080*

Con todas las huellas ya creadas pasamos a definir las dimensiones de la placa y colocar los componentes sobre ella, en nuestro caso esto resultó sencillo debido a que había espacio más que suficiente para todos los componentes. Con los componentes ya colocados comenzamos el *ruteado.* El *ruteado* es un proceso que consiste en conectar los distintos pines de los componentes conforme al esquemático, es importante tratar de realizar las conexiones más directas posibles, así como intentar que las pistas (lo que sería el “cable” que conecta dos pines en la PCB) no vayan cambiando todo el rato de una capa a otra, esto puede producir ruidos en la señal que transporten y que afecten al desempeño del circuito.

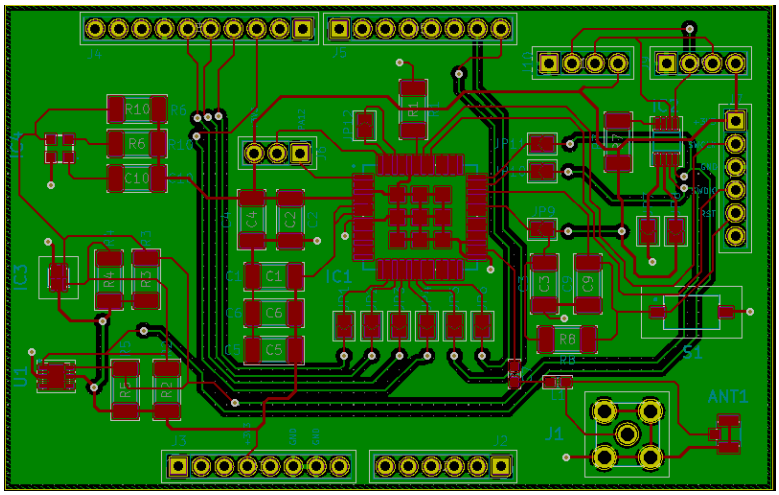


Figura 38. *Vista de la placa parcialmente ruteada*

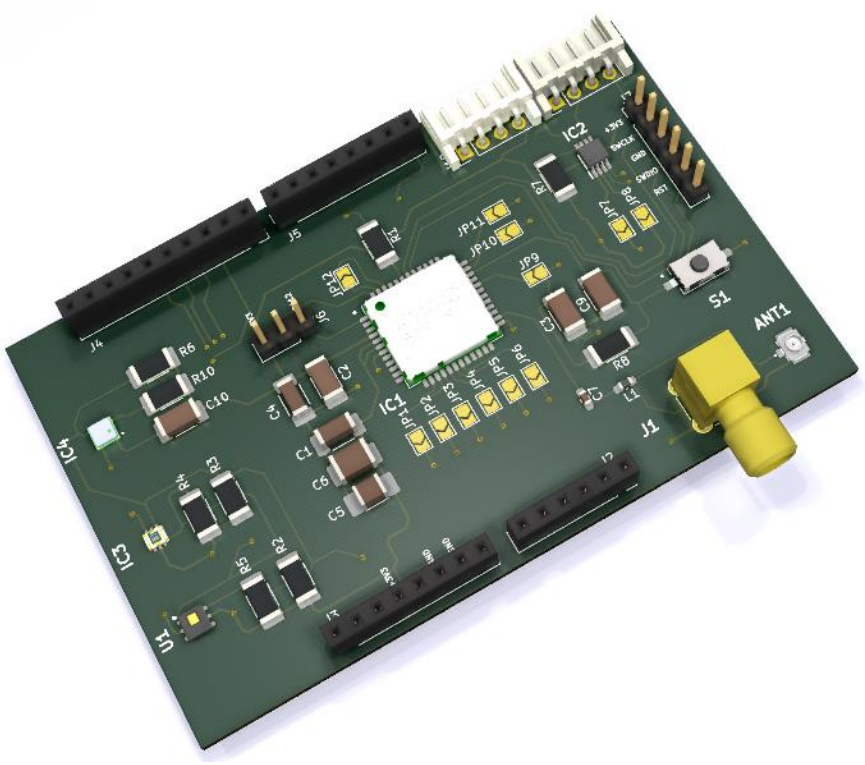


Figura 39*. Renderizado 3D de la placa*

A la hora de desarrollar una placa hay que intentar que la conexión a masa resulte lo más estable posible, por esto es una buena práctica tener una capa entera si es posible dedicada únicamente a masa [37]. En nuestro caso, al estar diseñando una placa de solo dos capas no fue posible dedicar una en exclusiva a masa, pero sí que se trató de minimizar el número de pistas que hubiese en ella.

Seguidamente, en este apartado se quiere explicar el proceso de diseño de la pista de la conexión de la antena de la MCU. Toda pista dispone de una

impedancia propia, en la mayoría de los casos esta es de un valor despreciable que no afecta al correcto funcionamiento del sistema, pero las antenas son señales de alta frecuencia, por lo que pueden tener grandes impedancias asociadas a las pistas por donde circule. Para que una antena alcance su punto óptimo de funcionamiento la impedancia que debe tener la pista es de 50 Ω. Para lograr el valor de impedancia deseado existen numerosas calculadoras en internet donde introducimos las características físicas de la placa y nos informa de que ancho de pista es necesario para obtener la impedancia buscada.

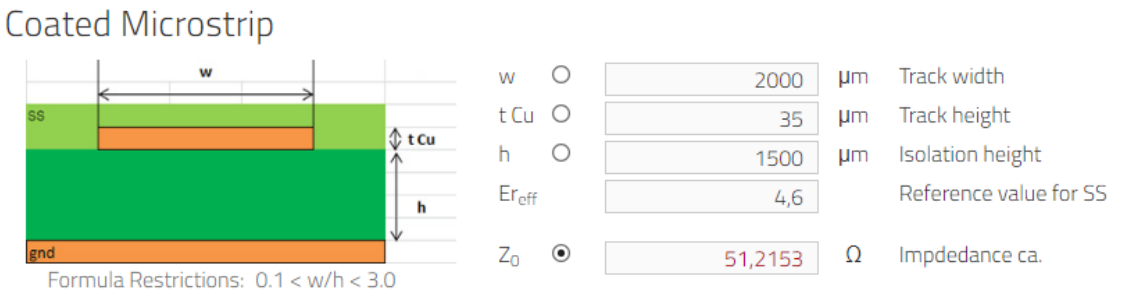


Figura 40. *Calculadora de impedancia con os valores de nuestra placa [38]*

Por último, la placa se conectó con el B-L072Z mediante los conectores de Arduino presentes. Esto significó que había que colocar la misma distribución de pines en nuestra placa para que pudiesen encajar y que las señales asociadas a cada pin tenían que ser las mismas.

5.1.1.6. Fabricación

Una vez que se ha completado el diseño de la PCB solo queda mandarla a fabricar. Para esto fue necesario mandar los gerber a la compañía que se eligió para fabricar la placa, JLC PCB. Los gerber son simplemente la representación de cada una de las capas que componen una placa (se pueden encontrar en el apartado de planos), con ellos el fabricante es capaz de crear la PCB (en el pliego de condiciones asociado a esta memoria se puede encontrar información más técnica a cerca de las especificaciones de fabricación de esta PCB).

5.1.1.7. Montaje y testeo

Cuando recibimos las placas y los componentes los revisamos visualmente para cerciorarnos de que estaban en buen estado y sin ningún defecto apreciable.

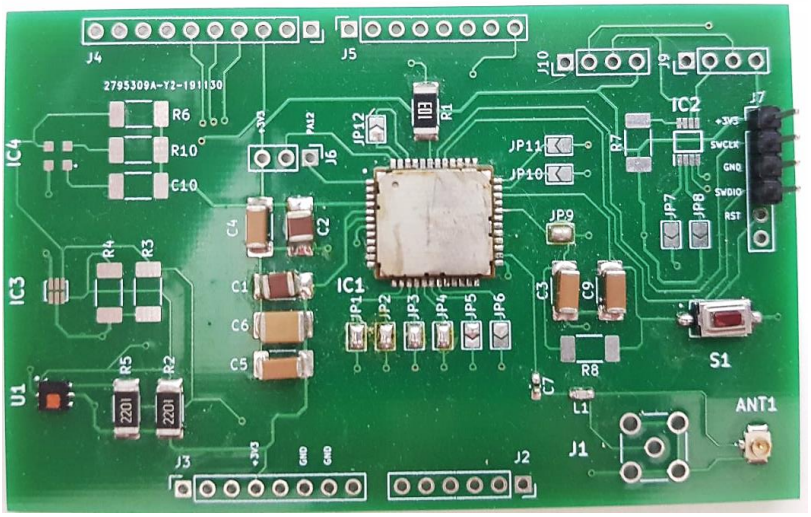


Figura 41. *Vista de la placa una vez recibida con algunos componentes ya soldados*

Una vez revisados se soldaron los componentes, se conectó la placa con el B-L072Z y ya se pasó al desarrollo del software para la placa.

5.1.2. Desarrollo del software

Con el hardware ya listo pasamos a desarrollar el software. El programa que se desarrolló consistía en una serie de funciones que llamaban a cada uno de los sensores para conocer su estado actual y enviar esos valores a la gateway.

Antes de nada, se probó el correcto desempeño de los sensores conectando la placa a un Arduino Uno y utilizando los ejemplos de código que proporcionan los fabricantes para asegurarnos de que el nodo funciona correctamente. Tras revisar algunas soldaduras un poco defectuosas, el nodo leyó todos los valores de manera satisfactoria.

5.1.2.1. Librerías de los sensores

Las librerías de los sensores se han diseñado adaptando las que se usaron anteriormente para comprobar el correcto funcionamiento de la placa, han sido desarrolladas por Adafruit para Arduino en C++ y son de dominio público. La librería de la electroválvula se ha diseñado desde cero debido a la sencillez de esta.

La plataforma Arduino utiliza código C y C++, al igual que los micros de STM, por lo que las principales modificaciones son las relacionadas con la comunicación I2C, teniendo que adaptarla desde la librería *Wire.h* de Arduino [39] a la *stm32l0xx_hal.h* (conocida como HAL) que se utiliza en los STM32L0 [40].

La librería Wire.h de Arduino tiene seis comandos principales:

- **begin()**: inicia la librería, normalmente solo se llama una vez.
- **beginTransmission(address)**: comienza la transmisión del máster hacia el esclavo del que se haya proporcionado la dirección I2C (parámetro address).
- **write()**: transmite la información que el master haya solicitado al esclavo o almacena los datos que el máster transmitirá al esclavo.
- **endTransmission()**: termina la transmisión que se haya iniciado con beginTransmission() y envía los datos de write().
- **requestFrom()**: usado por el máster para solicitar datos del esclavo.
- **read()**: lee del esclavo el byte que se solicitó con requestFrom().

La librería HAL utiliza otra serie de comandos para lograr el mismo objetivo:

- **HAL_I2C_Mem_Write(&Handler, address << 1, register, register_size, *data, size, timeout)**: este comando sirve para escribir en un registro particular a una dirección determinada, vemos que tiene varios parámetros que vamos a intentar explicar. &Handler es un puntero a los parámetros con los que se ha configurado el bus I2C utilizado. El segundo valor es la dirección del esclavo al que se quiere transmitir la información, esta desplazada una posición hacia la izquierda porque el bit menos relevante se utiliza para indicar al esclavo si la conexión es de lectura o escritura, en este caso se pondrá a 0 para escribir. Los dos parámetros siguientes sirven para decirle al esclavo que registro queremos leer y el tamaño de este. El quinto es un puntero al buffer del que se quiere transmitir la información o en el que se desea almacenarla, size indica el número de bytes a utilizar del buffer. Por último, timeout indica el tiempo máximo de espera de la transmisión.
- **HAL_I2C_Mem_Read(&Handler, address << 1, register, register_size, *data, size, timeout)**: es equivalente al comando anterior pero para leer un regirstro.
- **HAL_I2C_Master_Transmit(&Handler, address << 1, data, size, timeout)**: Sirve para transmitir una cadena de bytes a una dirección, comparte parámetros con las funciones anteriores. Se transmitirá la información que contenga la dirección a la que apunte data.
- **HAL_I2C_Master_Receive(&Handler, address << 1, data, size, timeout)**: equivalente a Transmit pero para leer del esclavo, primero será necesario utilizar Transmit para decirle al esclavo qué registro se quiere leer.

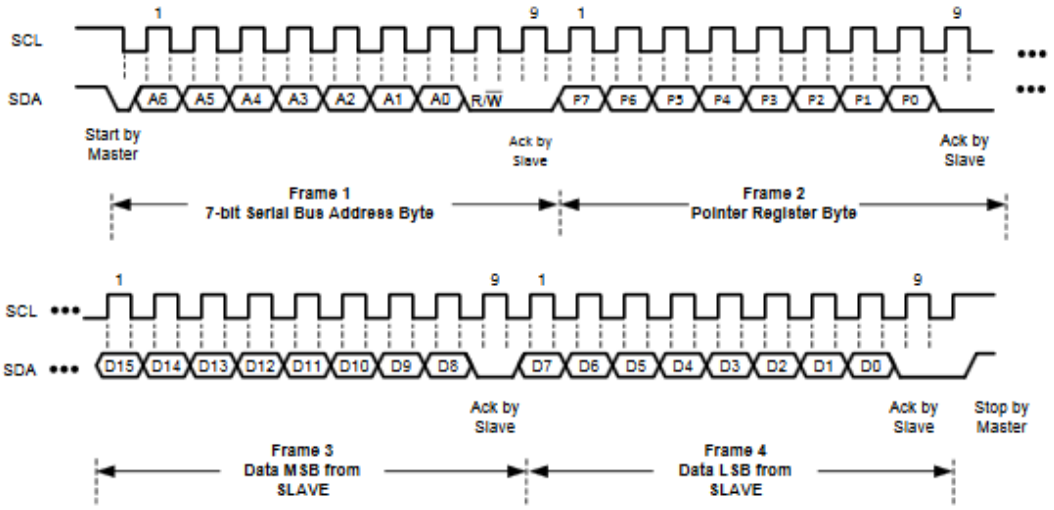


Figura 42. *Ejemplo de escritura de 2 bytes desde el máster hacia el esclavo*

Con esta información clara se pasó a modificar las funciones de escritura y lectura de registros en las librerías de los sensores.

Este es el aspecto del código para escribir un registro en Arduino en el sensor HDC2080:

```
void HDC2080::writeReg(uint8_t reg, uint8_t data)
{
  Wire.beginTransmission(_addr);                // Open Device
  Wire.write(reg);                              // Point to register
  Wire.write(data);                             // Write data to register
  Wire.endTransmission();                       // Relinquish bus control
}
```

Y esta sería su equivalencia utilizando las librerías HAL:

```
void HDC2080::writeReg(uint8_t reg, uint8_t data)
{
    HAL_I2C_Mem_Write(&_i2cHandler, _i2caddr << 1, reg,
I2C_MEMADD_SIZE_8BIT, &data, 1, 100);
}
```

Este proceso se llevó a cabo con el resto de librerías de los sensores.

Para la librería de la electroválvula únicamente fue necesario crear una función que configurase al pin A9 como salida digital y otra para decidir si el pin se coloca en estado alto o bajo:

```
void valve_Set(TValveState state)
{
if(state==VALVE_CLOSE)
        {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
        }
else if(state==VALVE_OPEN)
        {
         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        }
}
```

Todas las librerías se encuentran de manera completa en el anexo 1 del presente documento.

5.1.2.2. Aplicación principal

Con las librerías ya configuradas se pasó a modificar la plantilla base que STMicroelectronics ofrece para configuración de la comunicación como nodo de en sus dispositivos. El primer paso para que el nodo funcione correctamente es configurar el archivo commissioning.h, este archivo contiene los parámetros necesarios para conectarse a LoRaWAN a través de la gateway conforme a los valores que se hayan establecido. Los valors que hay que introducir son:

- **OVER_THE_AIR_ACTIVATION**: Poner a 1 para utilizar la activación Over-the-Air en lugar de una personalizada
- **STATIC_DEVICE_EUI**: Poner a 1 para utilizar el Dev Eui creado en TTN, si está en 0 utilizará el ID propio de la placa
- **LORAWAN_DEVICE_EUI**
- **LORAWAN_APP_EUI**
- **LORAWAN_APP_KEY**
- **LORAWAN_NWK_S_KEY**
- **LORAWAN_APP_S_KEY**

El resto de parámetros deberán ser introducidos tal cual aparecen en TTN al crear la aplicación y el nodo, esto se explicará en el apartado 5.2.2.

El funcionamiento simplificado de la aplicación principal consiste en leer los valores de los diferentes sensores, crear el *payload* a enviar con ellos conforme a la codificación Cayenne LPP, transmitirlos a la gateway abrir las ventanas para la recepción de información desde la gateway y ponerse en suspensión hasta que comience el siguiente ciclo.

Estos son los valores que se pasan en el payload, cada variable lee el valor correspondiente de los sensores utilizando la función designada para ello en la librería y lo almacena.

```
temperature = (int16_t)(hdc2080.readTemp() * 10);
humidity    = (uint16_t)(hdc2080.readHumidity() * 2);
pressure    = (uint16_t)(ms5637.readPressure() * 100 / 10);
illuminance = (uint16_t) tsl2591.getLux();
moisture    = (uint16_t)soilMoist.moistureRead() * 10;
```

A continuación, se crea el payload conforme a las especificaciones de la documentación de Cayenne LPP.

```
uint32_t i = 0;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_ILLUMINANCE;
AppData.Buff[i++] = (illuminance >> 8) & 0xFF;
AppData.Buff[i++] = illuminance & 0xFF;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_TEMPERATURE;
AppData.Buff[i++] = (temperature >> 8) & 0xFF;
AppData.Buff[i++] = temperature & 0xFF;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_HUMIDITY;
AppData.Buff[i++] = humidity & 0xFF;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_ANALOG_INPUT;
AppData.Buff[i++] = (moisture >> 8) & 0xFF;
AppData.Buff[i++] = moisture & 0xFF;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_BAROMETER;
AppData.Buff[i++] = (pressure >> 8) & 0xFF;
AppData.Buff[i++] = pressure & 0xFF;
AppData.Buff[i++] = cchannel++;
AppData.Buff[i++] = LPP_DATATYPE_DIGITAL_OUTPUT;
AppData.Buff[i++] = ElectroValveState;
```

Es importante realizar correctamente la configuración del *payload* para que la API nos genere los widgets correctamente.

Con este proceso terminado se puede dar por concluida la configuración del nodo.

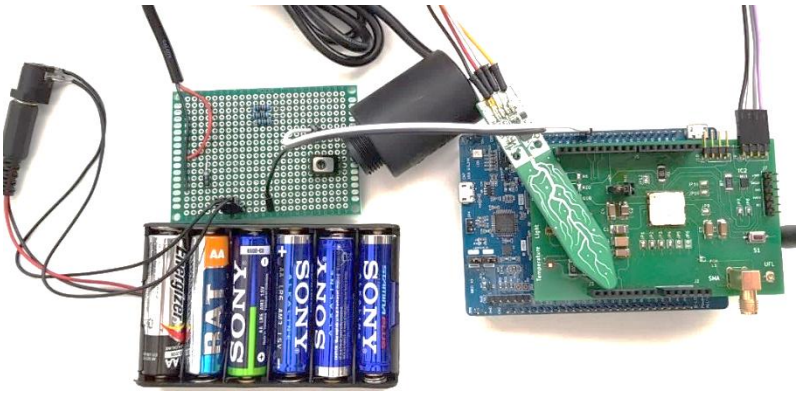


Figura 43. *Vista del nodo completo*

5.2. Configuración de la plataforma web

El nodo está listo para funcionar, pero para ello es necesario realizar toda la configuración de la plataforma web paso a paso.

5.2.1. Configuración de la gateway

El primer paso de este proceso fue configurar la gateway. En principio, la gateway a utilizar era la RAK2245, tal y como se explicó en el apartado 4.5, pero debido a la situación provocada por la COVID-19 no se ha podido utilizar esta gateway. RAK es una compañía China y a causa de los problemas con los envíos entre China y Europa no fue recibir la gateway con tiempo suficiente como para configurarla y hacer uso de ella en este proyecto. Como solución se decidió optar por la gateway de The Things Industries, las razones por las que se optó por esta solución fueron su reducido precio (pues ya se había hecho un desembolso importante por la otra gateway) y, sobre todo, que el distribuidor de este modelo es la compañía RS Componentes, con stock del producto en su almacén de Madrid, esto nos aseguraba recibir el producto en un par de días sin tener que sufrir más retrasos debidos a la congestión del tráfico de mercancías entre países.

Figura 44. *The Things Industries Indoor Gateway*

Una vez expuesto el problema y la solución respecto a la gateway, vamos a explicar los pasos que se llevaron a cabo para configurarla.

Este proceso fue muy sencillo de llevar a cabo, al ser un modelo desarrollado por la misma compañía que la plataforma de The Things Network, su conexión es casi instantánea.

La gateway se conecta a la electricidad, en este momento creará una red wifi a la que habrá que conectarse con un teléfono móvil y acceder a la página 192.168.4.1, en esta página elegiremos la conexión a internet que queremos que utilice la gateway y escribiremos la contraseña. Pulsamos en *Save & Reboot* y la

gateway debería reiniciarse y conectarse automáticamente a la red, si esto ocurre correctamente el led se quedará estático y de color verde.

El siguiente y último paso es conectar la gateway a TTN (para realizar esto será necesario haber creado antes la cuenta en TTN), para esto se registrará el dispositivo en la consola de TTN con la opción *I'm using the Legacy Packet Forwarder.* Se introduce el EUI del dispositivo y la configuración estará finalizada. El EUI puede localizarse al final de la página web del paso anterior, no hay que confundirla con la dirección MAC.

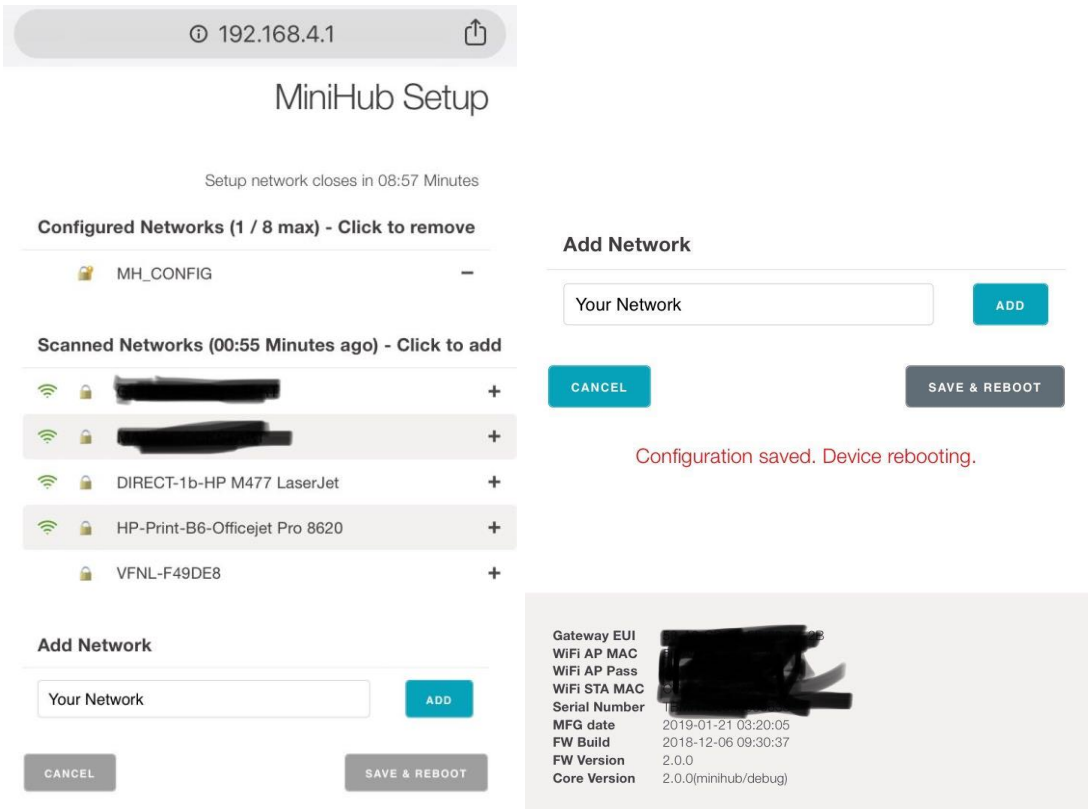


Figura 45. *Vista de la web 192.168.4.1 desde un smartphone*

5.2.2. Configuración del servidor de red

La configuración de The Things Network resultó sencilla e intuitiva. El primer paso es crear una cuenta para poder utilizar la aplicación. Con la cuenta ya creada, veremos que en la pantalla principal de la consola hay dos opciones, *Applications* y *Gateways.* Se selecciona *Gateways* y se añade una nueva, recordando que es necesario marcar la casilla que indica que se utiliza un *legacy packet forwarder* y se introduce el EUI de la gateway. Tras esto la gateway quedará registrada. En este punto se dispone de una gateway, pero todavía falta crear una aplicación desde la que se controlan los distintos nodos. El proceso de creación de la aplicación es análogo al de la gateway, se crea un Application ID para identificarla y el resto de parámetros los genera TTN de manera automática.

En este punto se tiene una aplicación vacía, por lo que hay que registrar algún dispositivo para que actúe como nodo. En la ventana de registrar dispositivo nos pide que introduzcamos un identificador único para el dispositivo, este número corresponde con el Device ID que se haya escrito en el commissioning.h. Ahora que se ha completado el registro del dispositivo, simplemente copiamos todos los valores que nos aparecen en la ventana acerca del dispositivo registrado y los pegamos en el commissioning.h, una vez realizado esto el nodo debería ser capaz de conectarse a TTN y comenzar a transmitir información.

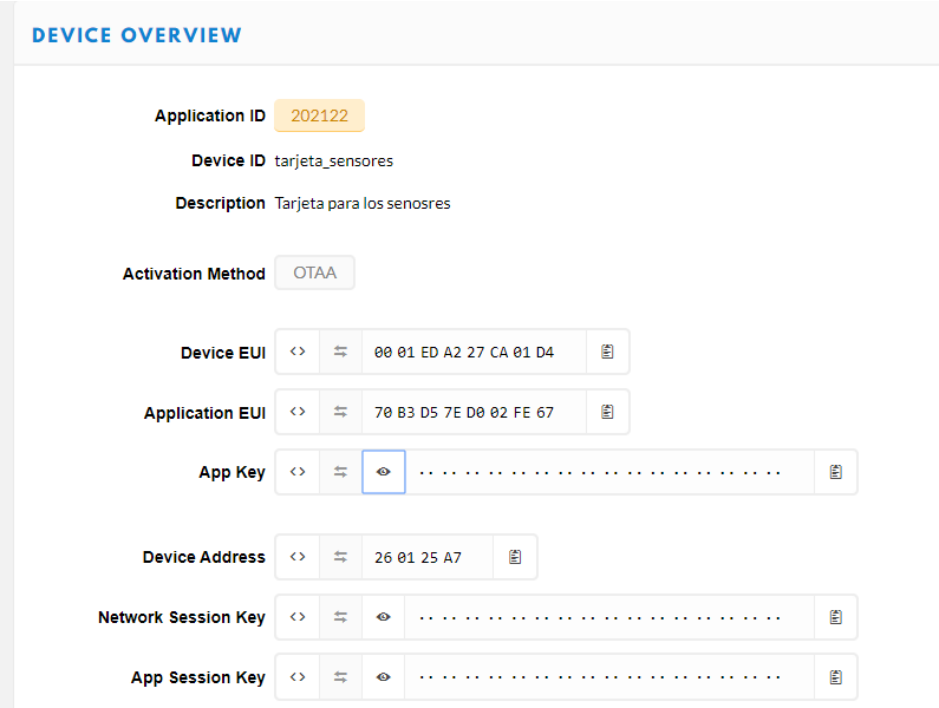


Figura 46. *Ventana de TTN donde se pueden ver los parámetros necesarios para la configuración del nodo*

5.2.3. Configuración de la API

El paso final para la configuración de la plataforma web es hacer que los valore que recibe TTN pasen a la API para poder visualizarlos. Este proceso es sumamente sencillo para la plataforma Cayenne My Devices, nos registramos en su web y añadimos un dispositivo del tipo Cayenne LPP de la lista desplegable, únicamente hay que introducir el DevEUI.

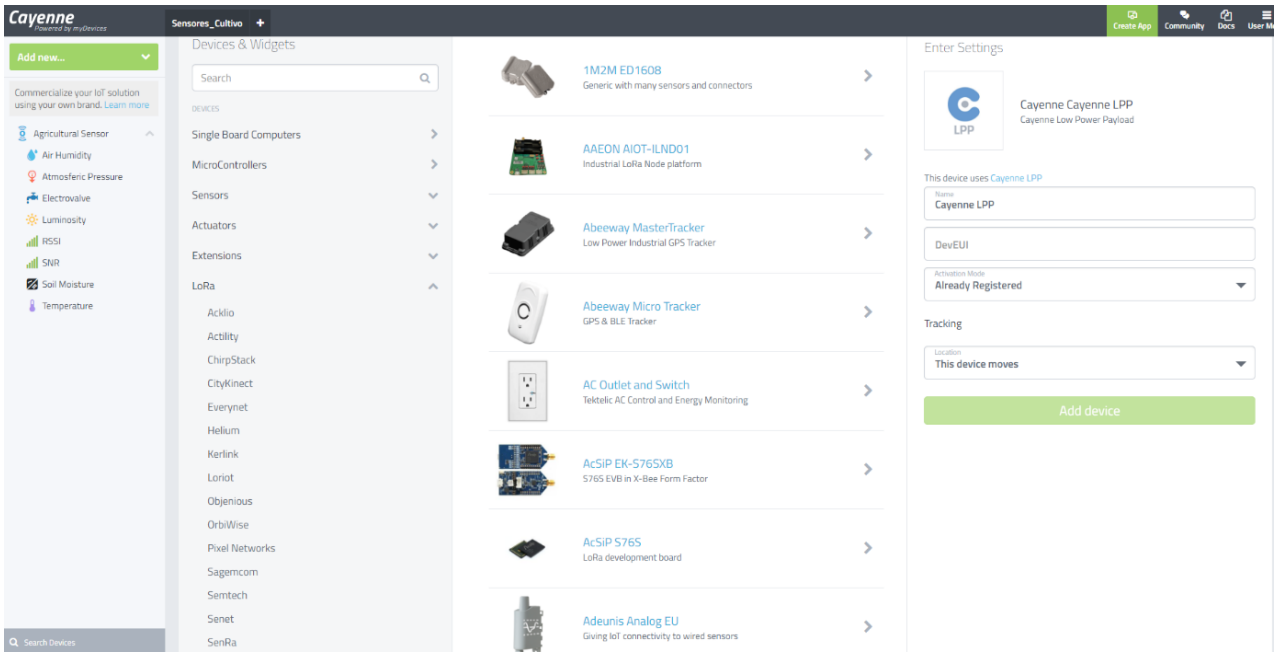


Figura 47. *Ventana para añadir dispositivos a Cayenne*

El paso final será volver a TTN y desde la pestaña de *integrations* añadir la integración a Cayenne, esta nos pedirá como parámetro el Process ID, esto es simplemente la secuencia alfanumérica que aparece en la url de la ventana principal de nuestro dispositivo creado en Cayenne. A partir de este punto debería comenzar a verse las tramas recibidas en TTN también en Cayenne y si el *payload* es correcto se verán los widgets con los valores leídos de los sensores.

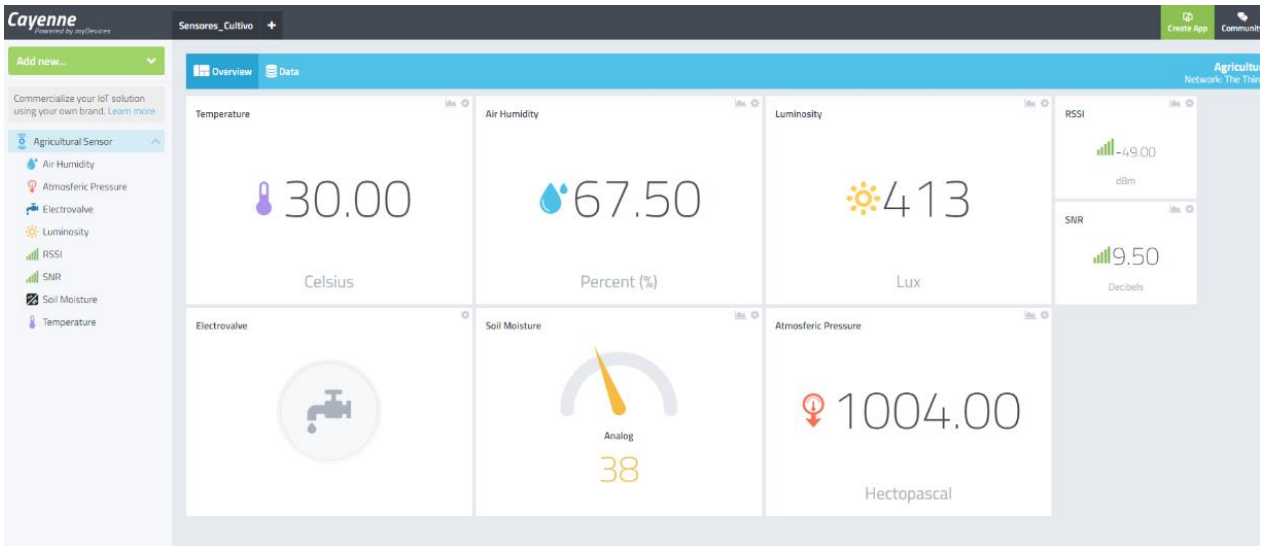


Figura 48. *Vista del dashboard con los widgets de cada sensor*

5.2.4. Prueba del sistema completo

Para realizar las pruebas finales se ha creado un proyecto en Cayenne. Esto consiste en una ventana similar al *dashboard* del dispositivo, pero desde donde podemos elegir ciertos comportamientos del sistema. Por ejemplo, se pueden crear alertas para que recibamos un SMS o un correo cuando cierto parámetro supere el umbral que hayamos definido, también hay un calendario desde el que podemos crear eventos para que se ejecuten de manera automática, como el proceso de regado.

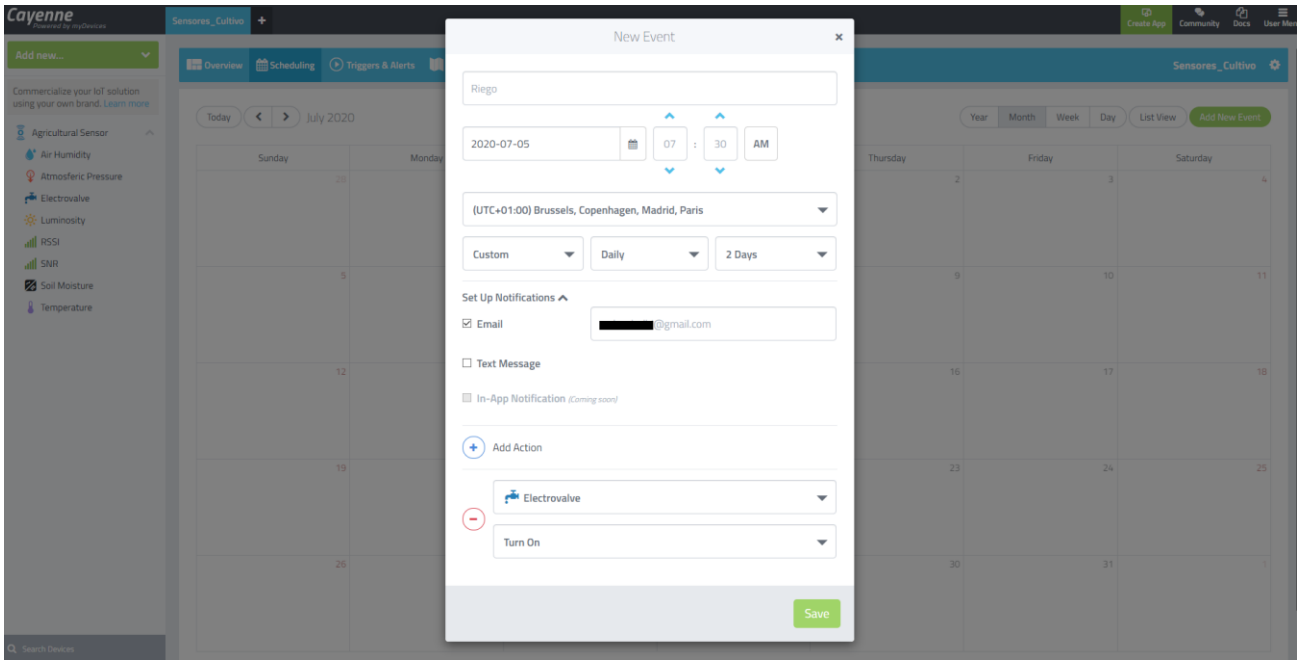
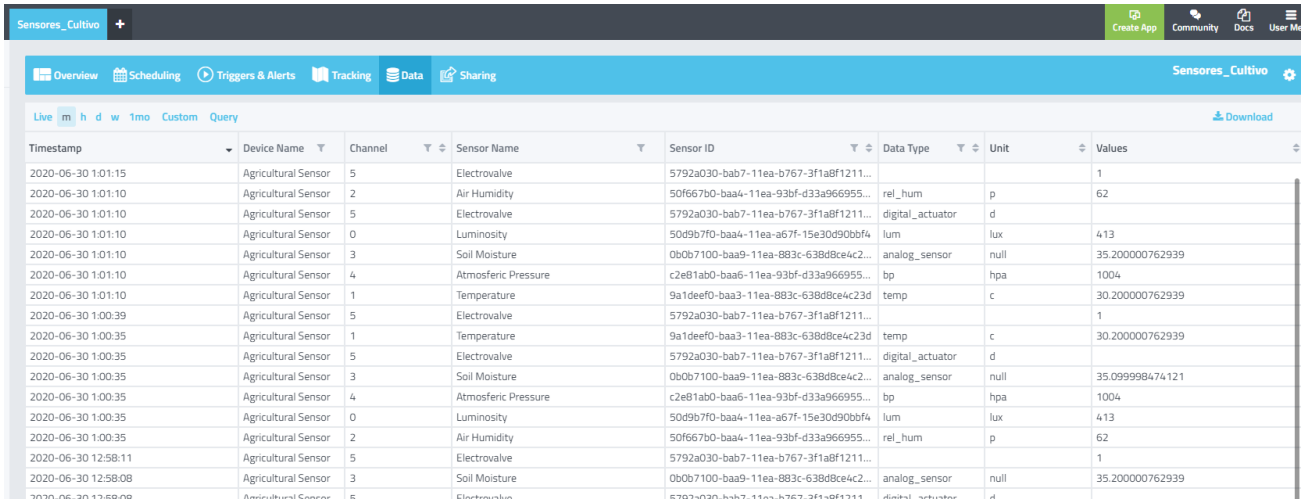


Figura 49. *Ventana desde la que configuramos eventos para el calendario*



Timestamp	Device Name	Channel	Sensor Name	Sensor ID	Data Type	Unit	Values
2020-06-30 1:01:15	Agricultural Sensor	5	Electrovalve	5792a030-bab7-11ea-b767-3f1a8f1211...			1
2020-06-30 1:01:10	Agricultural Sensor	2	Air Humidity	50f667b0-baa4-11ea-93bf-d33a966955...	rel_hum	p	62
2020-06-30 1:01:10	Agricultural Sensor	5	Electrovalve	5792a030-bab7-11ea-b767-3f1a8f1211...	digital_actuator	d	
2020-06-30 1:01:10	Agricultural Sensor	0	Luminosity	50d9b7f0-baa4-11ea-a67f-15e30d90bbf4	lum	lux	413
2020-06-30 1:01:10	Agricultural Sensor	3	Soil Moisture	0b0b7100-baa9-11ea-883c-638d8ce4c2...	analog_sensor	null	35.200000762939
2020-06-30 1:01:10	Agricultural Sensor	4	Atmosferic Pressure	c2e81ab0-baa6-11ea-93bf-d33a966955...	bp	hpa	1004
2020-06-30 1:01:10	Agricultural Sensor	1	Temperature	9a1deef0-baa3-11ea-883c-638d8ce4c23d	temp	c	30.200000762939
2020-06-30 1:00:39	Agricultural Sensor	5	Electrovalve	5792a030-bab7-11ea-b767-3f1a8f1211...			1
2020-06-30 1:00:35	Agricultural Sensor	1	Temperature	9a1deef0-baa3-11ea-883c-638d8ce4c23d	temp	c	30.200000762939
2020-06-30 1:00:35	Agricultural Sensor	5	Electrovalve	5792a030-bab7-11ea-b767-3f1a8f1211...	digital_actuator	d	
2020-06-30 1:00:35	Agricultural Sensor	3	Soil Moisture	0b0b7100-baa9-11ea-883c-638d8ce4c2...	analog_sensor	null	35.099998474121
2020-06-30 1:00:35	Agricultural Sensor	4	Atmosferic Pressure	c2e81ab0-baa6-11ea-93bf-d33a966955...	bp	hpa	1004
2020-06-30 1:00:35	Agricultural Sensor	0	Luminosity	50d9b7f0-baa4-11ea-a67f-15e30d90bbf4	lum	lux	413
2020-06-30 1:00:35	Agricultural Sensor	2	Air Humidity	50f667b0-baa4-11ea-93bf-d33a966955...	rel_hum	p	62
2020-06-30 12:58:11	Agricultural Sensor	5	Electrovalve	5792a030-bab7-11ea-b767-3f1a8f1211...			1
2020-06-30 12:58:08	Agricultural Sensor	3	Soil Moisture	0b0b7100-baa9-11ea-883c-638d8ce4c2...	analog_sensor	null	35.200000762939

Figura 50. *Datos recibidos desde el nodo en Cayenne*

Para poder controlar la aplicación remotamente se puede acceder a la página web cutt.ly/agro-monitor, no se puede garantizar que la web vaya a seguir

funcionando en el futuro, pero si se ha comprobado su correcto funcionamiento en el momento de realización del proyecto, siendo capaces de ver los valores de los sensores en la web y activar la válvula desde el teléfono móvil.

6. CONCLUSIONES

6.1. Acerca del desarrollo actual

El proyecto realizado ha sido una buena manera de integrar varios de los campos de la ingeniería vistos a lo largo de los cuatro años de carrera. Comenzando con una parte de electrónica más pura como es el desarrollo de esquemáticos y creación de un circuito integrado o una parte eminentemente informática como la programación de sistemas embebidos o el trabajo con redes de telecomunicación.

Para realizar este proyecto ha resultado especialmente útil el haber cursado las menciones de electrónica y la de informática industrial, así como la realización de prácticas de empresa orientadas al trabajo con sistemas embebidos. Todo esto ha contribuido a tener una buena base para investigar y comprender la variedad de tecnologías que se utilizan en IoT.

El resultado final obtenido con el proyecto lo valoramos positivamente, se ha logrado el objetivo que se planteó en el objeto, se dispone de un dispositivo capaz de recolectar los datos del entorno y activar una electroválvula de manera remota desde una página web. Desgraciadamente, la situación provocada por el Coronavirus ha dificultado el poder profundizar más en ciertas partes del trabajo. Al no haber recibido la gateway capaz de transmitir a grandes distancias no ha sido posible comprobar cuál sería la distancia máxima que se podría obtener entre el nodo y la gateway. Esto ha provocado que el trabajo se haya quedado un poco más limitado a lo que podría considerarse un trabajo de laboratorio y menos “de campo” de lo que en un primer momento se pretendía.

6.2. Continuación y mejoras futuras

Aunque como se ha dicho anteriormente, se valora positivamente el trabajo obtenido, esto ni mucho menos significa que no haya multitud de cosas por pulir y mejorar.

Se podría considerar que el siguiente paso sería desarrollar un segundo prototipo que sirva como continuación y mejora del que se dispone actualmente. Para este segundo prototipo la idea sería crear dos PCBs distintas, una orientada únicamente a la recolección de datos que haga uso de una sonda de humedad de carácter más profesional y otra que se centre únicamente en el poder controlar varias válvulas. En esta segunda versión se trataría de estudiar en

mayor profundidad el apartado de la alimentación del sistema para ver cómo optimizar el consumo al máximo y contemplar la posibilidad de que se alimentase el circuito con placas solares. Además, para esta versión resultaría necesario fabricar una carcasa que contuviese el dispositivo y lo protegiese de la lluvia. Por último, se llevaría a cabo las pruebas de campo con la RAK2245 que se pretendían hacer desde el principio.

7. BIBLIOGRAFÍA

[1] Wikipedia (junio 2020), *History of agriculture,* Recuperado de https://en.wikipedia.org/wiki/History_of_agriculture

[2] J. Hatfield y J. Prueger. (2015). Temperature extremes: Effect on plant growth and development. *Weather and Climate Extremes*, 10, 4-10, doi: https://doi.org/10.1016/j.wace.2015.08.001

[3] I. Griñán et al (2019). Leaf water relations in Diospyros kaki during a mild water deficit exposure. *Agricultural Water Management*, 217, 391-398 doi: https://doi.org/10.1016/j.agwat.2019.03.008

[4] J. C. Lopez. (2018). *La influencia de la luz en el crecimiento del cultivo*. Recuperado de https://www.pthorticulture.com/es/centro-de-formacion/la-influencia-de-la-luz-en-el-crecimiento-del-cultivo/

[5] C. D. Ahrens. (2017). *Essentials of Meteorology*. Boston: Cengage Learning

[6] R. L. Boylestad. (2015). *Introductory Circuit Analysis*. Pearson

[7] I2C-bus specification and user manual (2014) Recuperado de https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[8] Serial Peripheral Interface (SPI) (junio 2020). Recuperado de https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

[9] Texas Instruments (mayo 2018). *HDC2080 Datasheet.* Recuperado de https://www.ti.com/lit/ds/symlink/hdc2080.pdf

[10] Adafruit (junio 2020). *STEMMA soil sensor Datasheet*. Recuperado de https://www.mouser.es/pdfdocs/adafruit-stemma-soil-sensor-i2c-capacitive-moisture-sensor.pdf

[11] AMS (abril 2013). *TSL2591 Datasheet.* Recuperado de https://ams.com/documents/20143/36005/TSL2591_DS000338_6-00.pdf

[12] TE Connectivity (junio 2017). *MS5637 Datasheet.* Recuperado de https://www.te.com/commerce/DocumentDelivery/DDEController

[13] Tameson (junio 2020). *Solenoid Valve – How They Work.* Recuperado de https://tameson.com/solenoid-valve-types.html

[14] Microchip (2013). *Microprocessor (MPU) or Microcontroller (MCU)?.* Recuperado de ww1.microchip.com/downloads/en/DeviceDoc/MCU_vs_MPU_Article.pdf

[15] *High and Low Level Languages* (junio 2020). Recuperado de https://www.computerscience.gcse.guru/theory/high-low-level-languages

[16] Sparkfun (diciembre 2015). *The Arduino popularity contest*. Recuperado de https://www.sparkfun.com/news/1982

[17] STM (junio 2020). *STM32 32-but Arm Cortex MCUs.* Recuperado de https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[18] K. Kekki. (2019). A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express,* 5, 1-7, doi: https://doi.org/10.1016/j.icte.2017.12.005
[19] *LoRa Alliance*. (junio 2020). Recuperado de https://lora-alliance.org
[20] The Things Network. (abril 2020). *LoRa World Record Broken: 832km using 25mW.* Recuperado de https://www.thethingsnetwork.org/article/lorawan-world-record-broken-twice-in-single-experiment-1
[21] Vodafone. (diciembre 2016). *Narrowband-IoT.* Recuperado de https://www.vodafone.es/c/statics/narrowband-iot.pdf
[22] *Sigfox España*. (junio 2020). Recuperado de https://www.sigfox.es
[23] Semtech. (junio 2020). *LoRa*. Recuperado de https://www.semtech.com/lora
[24] Murata. (octubre 2018). *CMWXZZ1ABZ-078 Datasheet*. Recuperado de https://wireless.murata.com/pub/RFM/data/type_abz.pdf
[25] HopeRF. (junio 2020). *RFM95W*. Recuperado de https://www.hoperf.com/modules/lora/RFM95.html
[26] ST. (junio 2020). *STM32Lo Discovery kit LoRa.* Recuperado de https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html
[27] RAK. (junio 2020). *RAK Pi Hat Edition.* Recuperado de https://doc.rakwireless.com/rak2245-pi-hat-edition-lorawan-gateway-concentrator-module
[28] *The Things Network.* (junio 2020). Recuperado de https://www.thethingsnetwork.org
[29] *ChirpStack.* (junio 2020). Recuperado de https://www.chirpstack.io
[30] MyDevices. (junio 2020). *Cayenne Docs*. Recuperado de https://developers.mydevices.com/cayenne/docs/intro/
[31] TagoIO. (junio 2020). *TagoIO Blog*. Recuperado de https://tago.io/blog/
[32] RedHat. (junio 2020). *El concepto de IDE*. Recuperado de https://www.redhat.com/es/topics/middleware/what-is-ide
[33] ARM Keil. (junio 2020). *uVision IDE.* Recuperado de http://www2.keil.com/mdk5/uvision/
[34] ARM, (junio 2020). *Mbed*. Recuperado de https://os.mbed.com
[35] ST. (junio 2020). *Integrated Development Environment for STM32*. Recuperado de https://www.st.com/en/development-tools/stm32cubeide.html
[36] D. L. Jones. (junio 2004). *PCB Design Tutorial*. Recuperado de http://www.elec.canterbury.ac.nz/intranet/dsl/p70-pcb/p10-pcb/pcb_design_tutorial.pdf
[37] Altium. (octubre 2018). *Altium Designer Best Practices*. Recuperado de https://resources.altium.com/p/altium-designer-best-practices

[38] Multi-cb. (junio 2020). *Impedance Calculator.* Recuperado de https://www.multi-circuit-boards.eu/en/pcb-design-aid/impedance-calculation.html
[39] Arduino. (junio 2020). *Wire Library.* Recuperado de https://www.arduino.cc/en/reference/wire
[40] ST. (diciembre 2019). *UM1749 User Manual*. Recuperado de https://www.st.com/resource/en/user_manual/dm00113898-description-of-stm32l0-hal-and-low-layer-drivers-stmicroelectronics.pdf
[41] ST. (junio 2020). *LoRa*. Recuperado de https://www.st.com/en/applications/connectivity/lora.html

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

2. Planos

Autor:

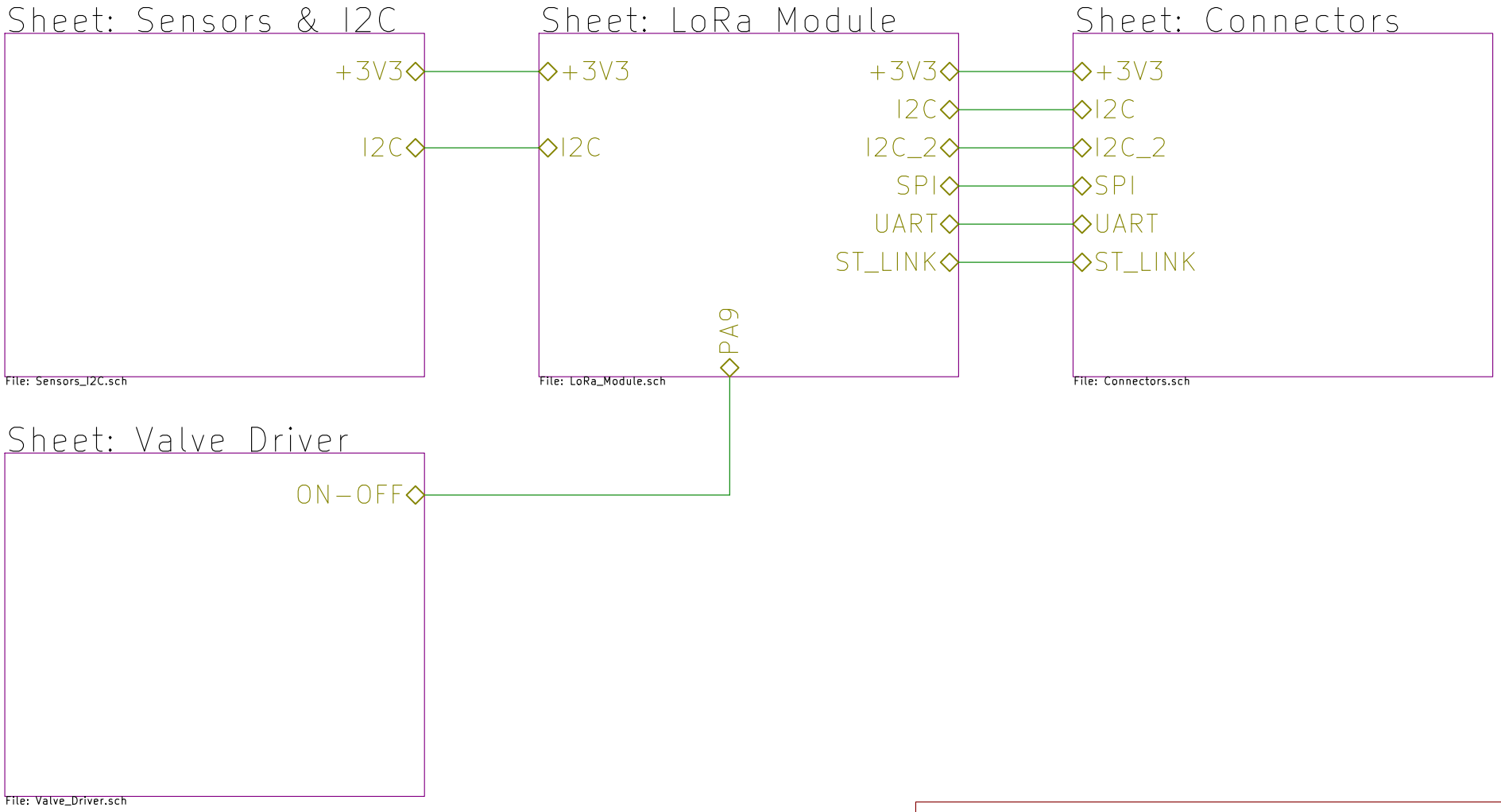
D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

ÍNDICE DE LOS PLANOS



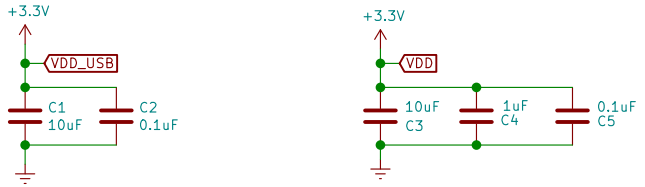
Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet: /
File: Agriculture_IoT.sch

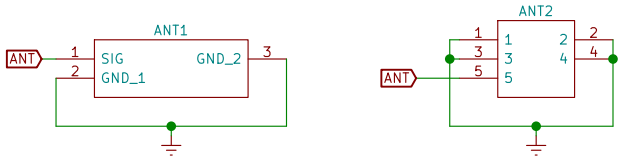
Title: Agricultural IoT Shield

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. eeschema (5.1.4)-1		Id: 1/5

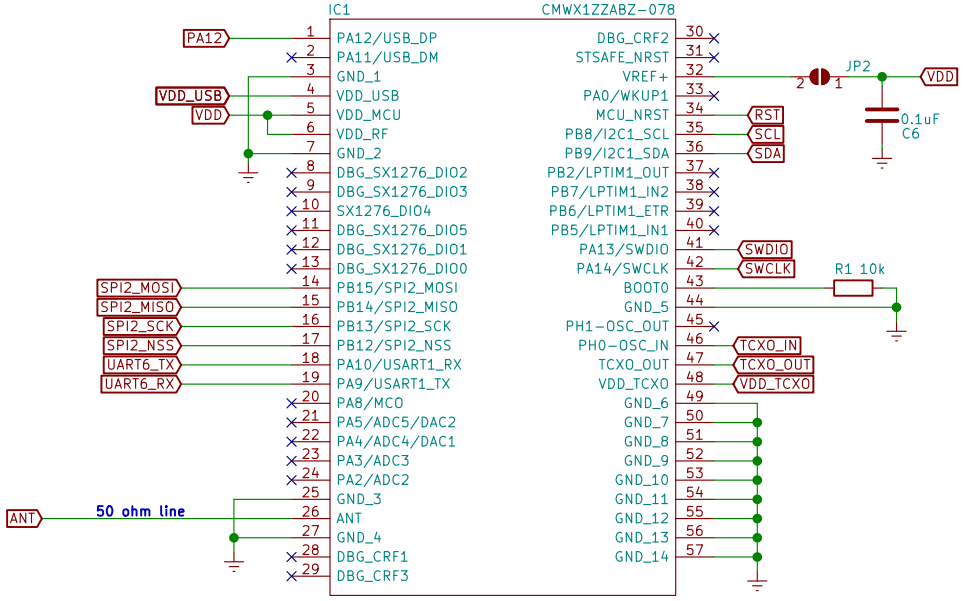
Power Supply



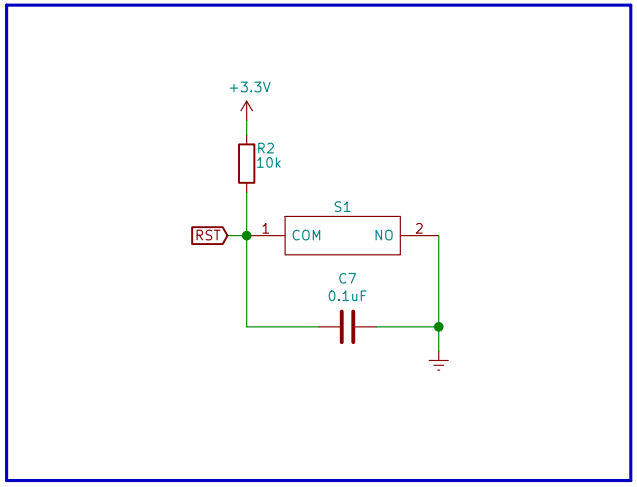
UFL & SMA Antennas



LoRa + MCU



Reset Switch



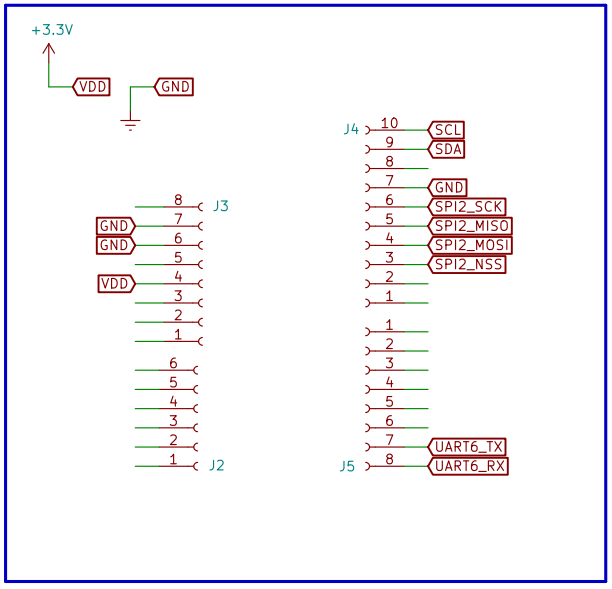
Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet: /LoRa Module/
File: LoRa_Module.sch

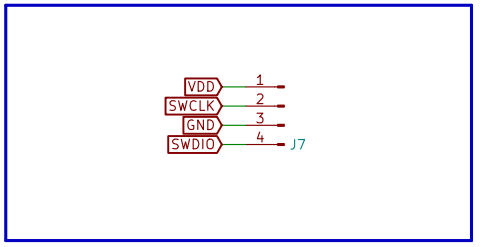
Title: LoRa Module

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. eeschema (5.1.4)-1		Id: 2/5

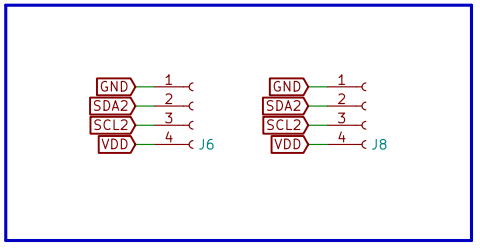
Arduino Connectors



ST Link



I2C Shifter



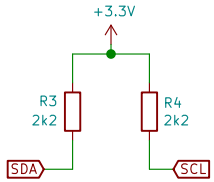
Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet: /Connectors/
File: Connectors.sch

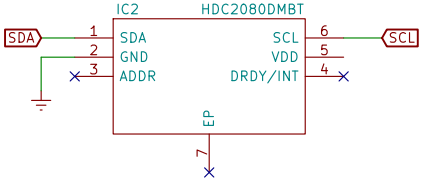
Title: Connectors

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. eeschema (5.1.4)-1		Id: 3/5

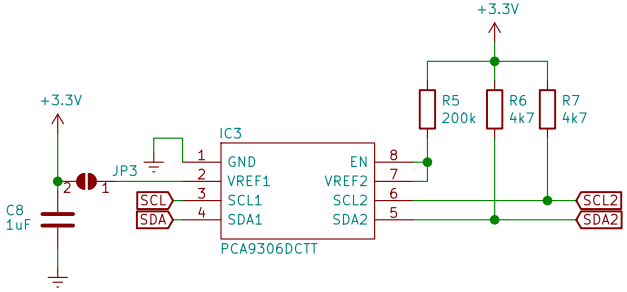
Pull-up Resistors



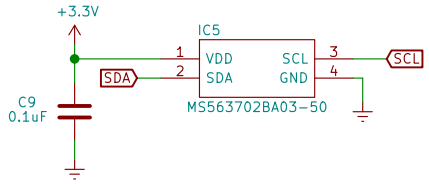
HDC2080



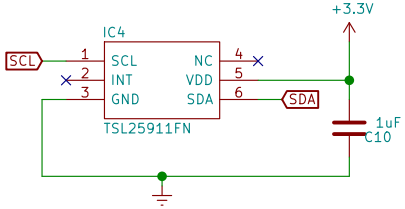
I2C Dual Bus



MS5637



TSL2591



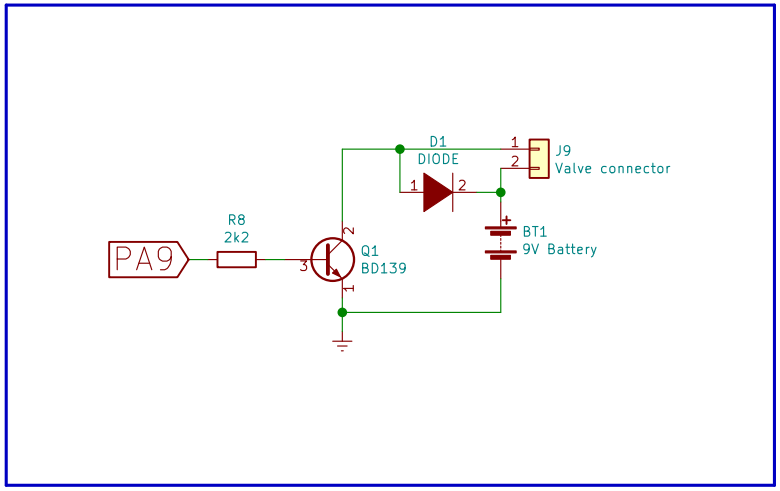
Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet: /Sensors & I2C/
File: Sensors_I2C.sch

Title: Sensors

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. eeschema (5.1.4)-1		Id: 4/5

Electrovalve Driver



Project: Agriculture_IoT
Author: C. Chulbi Perales

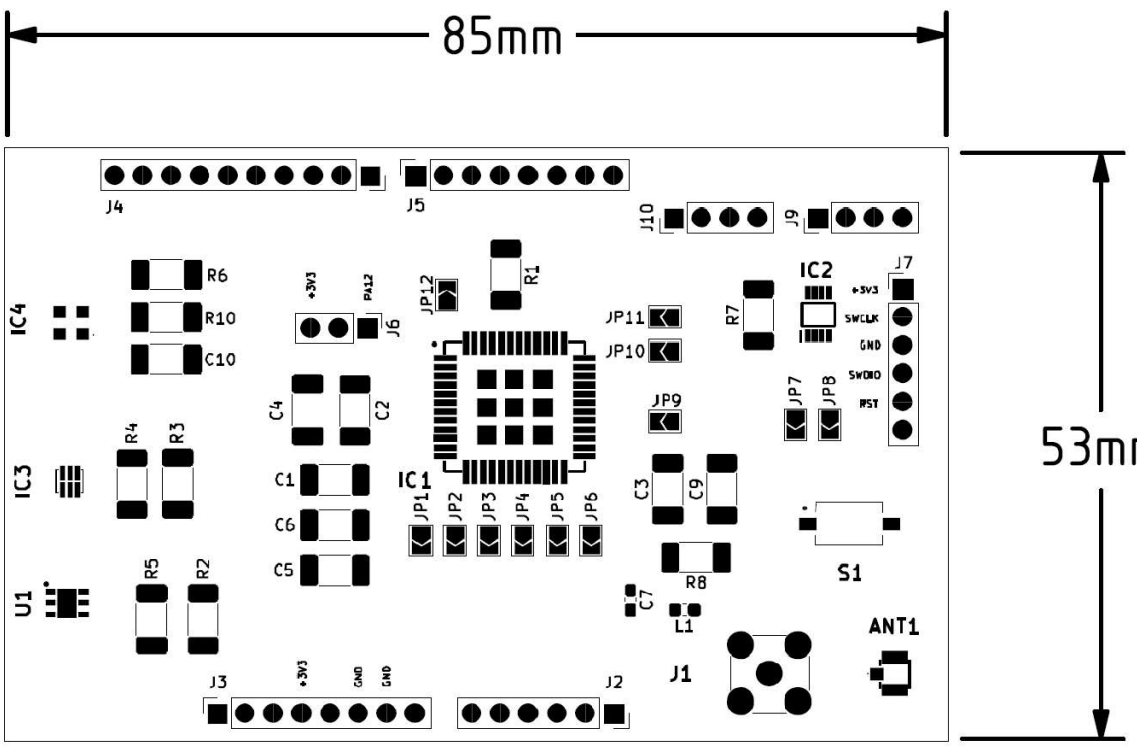
Sheet: /Valve Driver/
File: Valve_Driver.sch

Title: Electrovalve

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. eeschema (5.1.4)-1		Id: 5/5

2 Layers **1.6mm (0.062 inch) Green for 1 Oz**

Layer stack up	Layer	Thickness (mm)
	Top Solder Mask	0.01
	Top Layer	0.035
	Core	1.5
	Bottom Layer	0.035
	Bottom Solder Mask	0.01

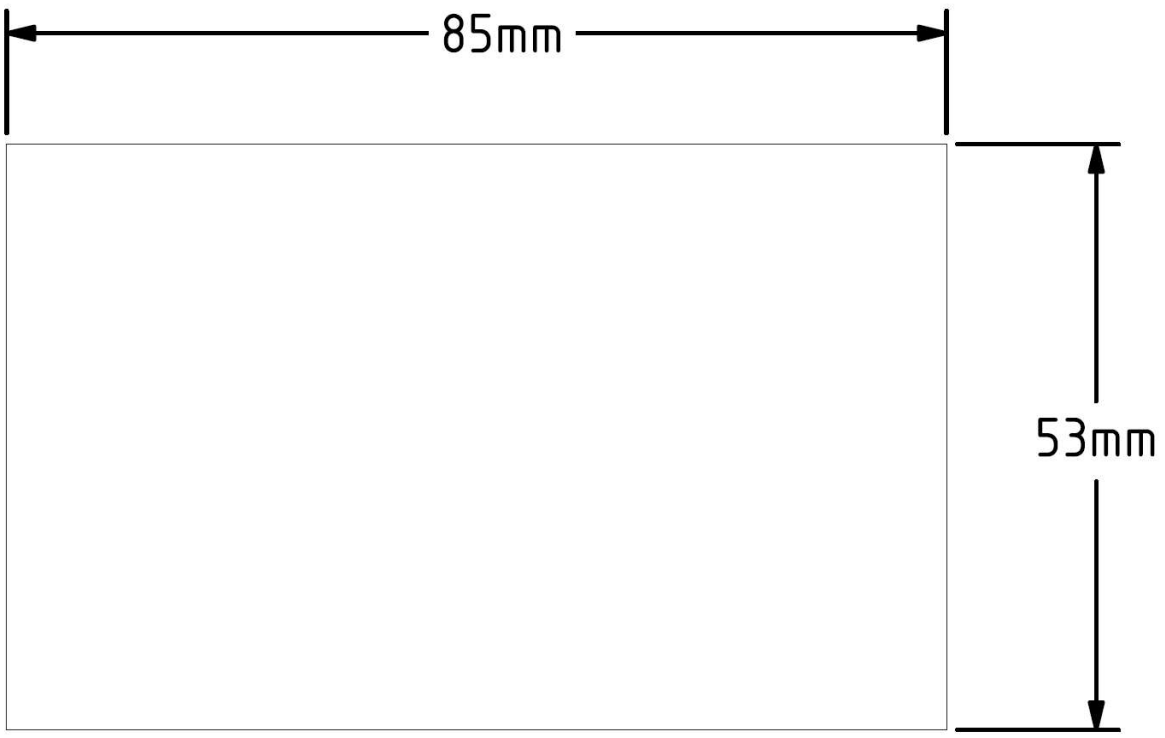


Project: Agriculture_IoT
Author: C. Chulbi Perales

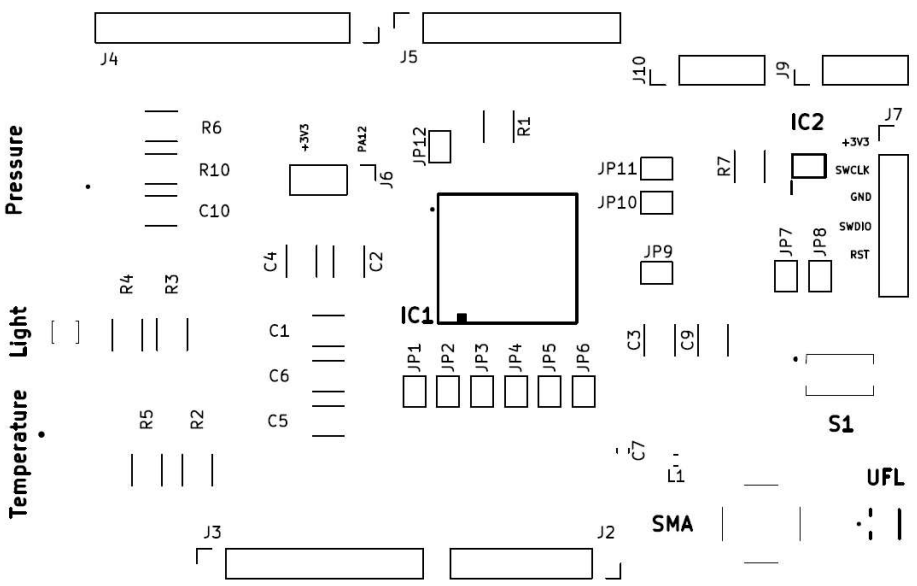
Sheet:
File:

Title: Stackup & Front View

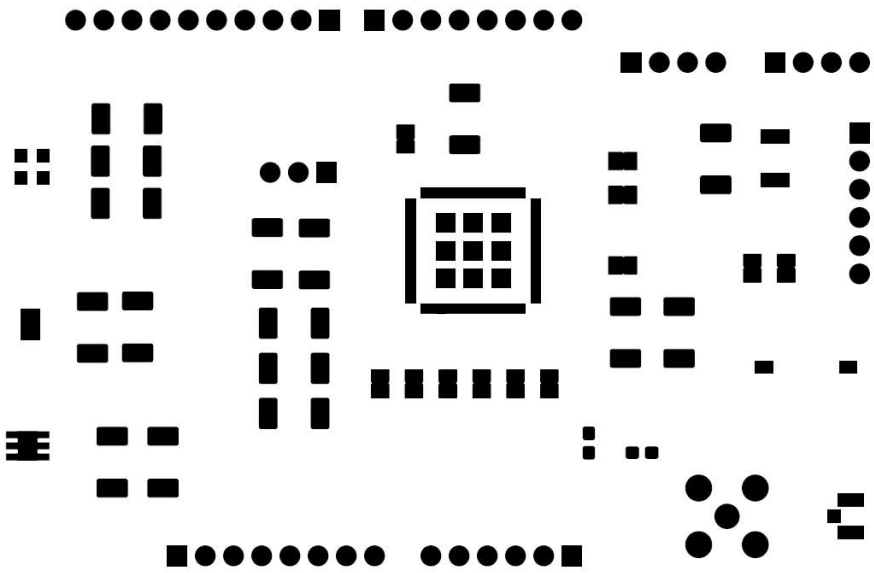
Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		



Project: Agriculture_IoT Author: C. Chulbi Perales		
Sheet: File:		
Title: Dimensions		
Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		



Project: Agriculture_IoT Author: C. Chulbi Perales		
Sheet: File:		
Title: Top Overlay		
Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

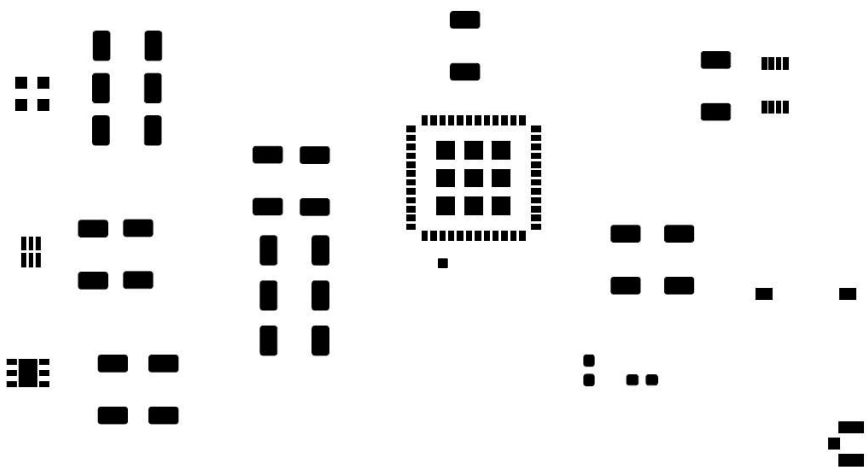


Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Top Solder Mask

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

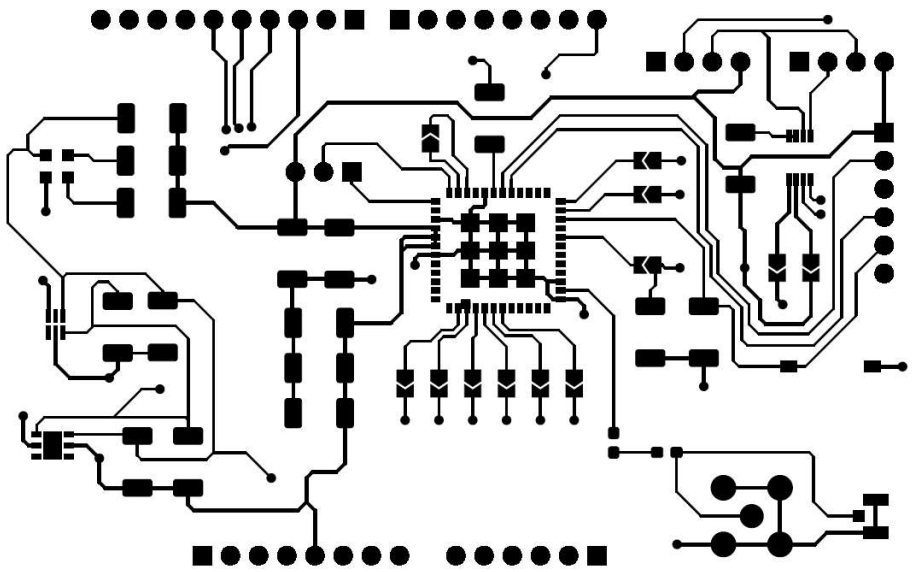


Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Top Paste Mask

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

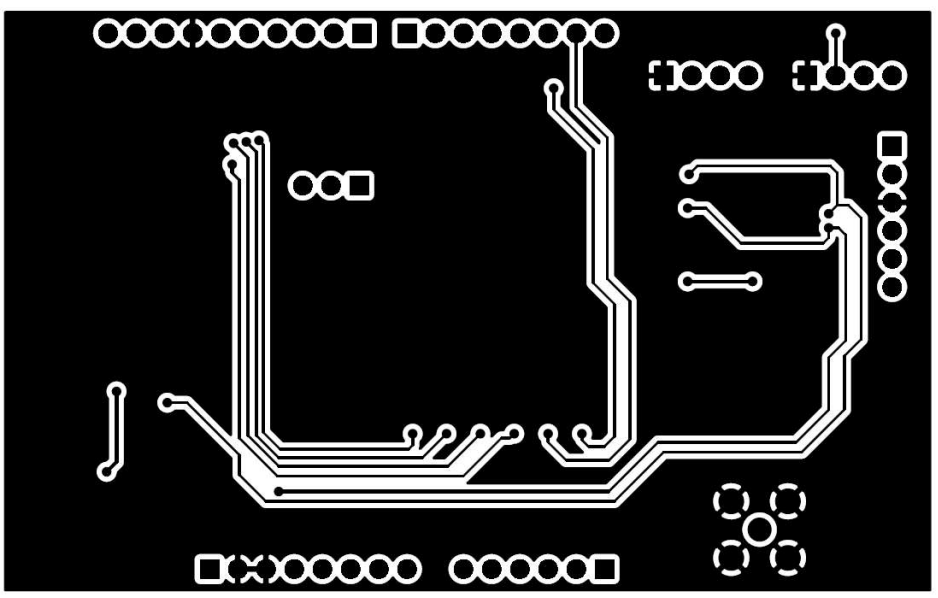


Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Top Copper Layer

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

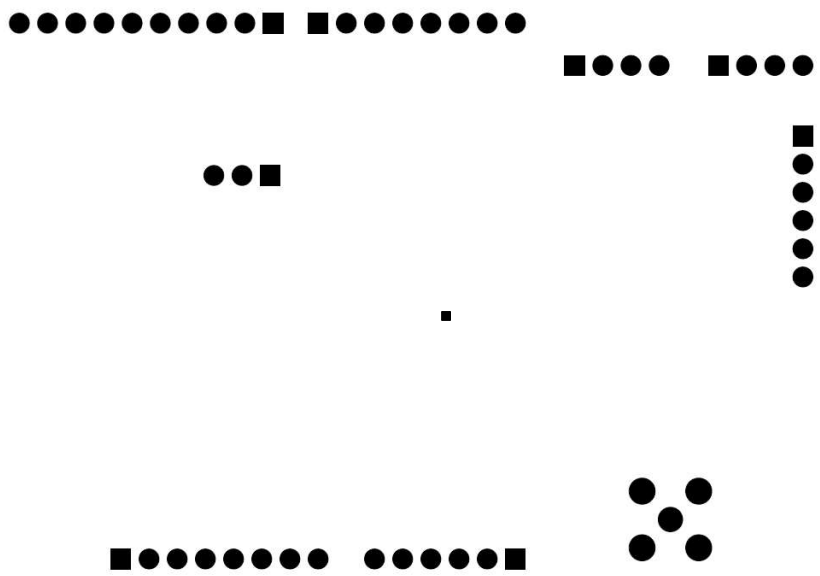


Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Bottom Copper Layer

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

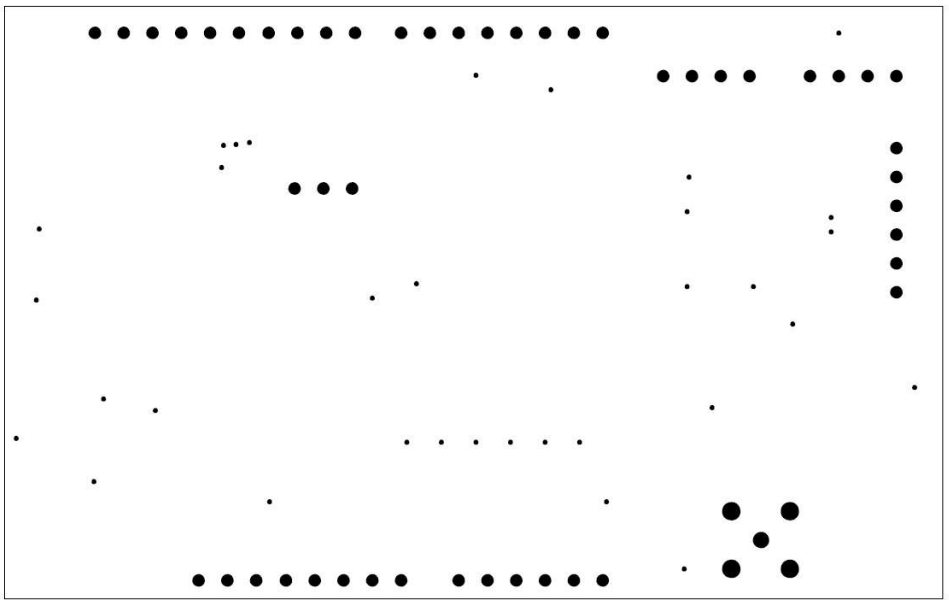


Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Bottom Solder Mask

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		



Project: Agriculture_IoT
Author: C. Chulbi Perales

Sheet:
File:

Title: Drill Holes

Size: A4	Date: 01/07/2020	**Rev: v1.0**
KiCad E.D.A. pl_editor (5.1.4)-1		

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

3. Pliego de condiciones

Autor:

D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

ÍNDICE DEL PLIEGO DE CONDICIONES

1. Definición y alcance del pliego

En el presente proyecto se realiza el diseño de un dispositivo para la adquisición de datos ambientales en entorno agrícola, específicamente en el cultivo del caqui. El dispositivo también es capaz de abrir la electroválvula del sistema de riego por goteo de la instalación. Toda la interacción entre el usuario y el dispositivo ocurre de manera remota, el dispositivo se conecta a internet utilizando como puente una red LoRaWAN y el usuario controla el riego y tiene acceso a los datos recolectados por los sensores mediante una plataforma web.

Este pliego incluye las instrucciones y directrices necesarias para poder fabricar, configurar y utilizar el dispositivo anteriormente descrito y cuya información pormenorizada se encuentra presente a lo largo del resto de documentos constituyentes de este proyecto.

2. Condiciones de carácter general

El dispositivo ha sido desarrollado para funcionar bajo unas condiciones concretas, no se puede garantizar su uso para fines distintos al original.

2.1. Instalación

El dispositivo se trata de un prototipo, por lo que, al no ser todavía una versión final, esta deberá ser instalada por alguien con unos conocimientos básicos en electrónica para evitar problemas en el proceso.

Hay que tratar de colocar el dispositivo lejos de fuentes de calor o espacios sombreados para lograr las mediciones más similares a la realidad.

Si el dispositivo se coloca a la intemperie, será necesario protegerlo de la lluvia y el polvo, de lo contrario su funcionamiento puede no ser correcto.

Para alimentar el dispositivo hay que colocar tres pilas de tipo AAA en su parte posterior, si se quiere hacer uso de la electroválvula, habrá que añadir una pila de 9V para ello.

La instalación de la electroválvula deberá ser realizada por personal familiarizado con las instalaciones de regadío y siempre cerciorándose antes de comenzar de que el paso del agua esta debidamente cerrado. Hay que asegurarse que el dispositivo se conecta por encima del nivel al que se encuentren las válvulas y tuberías de riego para evitar cualquier tipo de goteo o humedad excesiva que puedan dañar al dispositivo.

2.2. Seguridad

El dispositivo se considera seguro desde el punto de vista eléctrico, pues su fuente de alimentación son baterías de bajo voltaje (1,5V) y no debería resultar peligroso para el usuario si se han respetado las directrices de instalación del apartado anterior. El dispositivo cumple con normativa sobre aparatos electrónicos conforme al Real Decreto 110/2015.

Desde el punto de vista químico también puede considerarse el dispositivo como seguro al cumplir con la normativa europea de restricción de ciertas sustancias peligrosas RoHS.

Por último, el dispositivo se encuentra bajo la Directiva 93/68/CEE y su módulo de comunicaciones cumple con la directiva acerca de dispositivos RFID en la banda de 865-868 MHz (BOE-A-2008-11868), además está certificado para LoRaWAN por la LoRa Alliance.

2.3. Utilización

El dispositivo es del tipo *Plug and Play,* esto significa que no necesita de ningún tipo de configuración para su uso, comienza a funcionar en cuanto se conecta. Si hay alguna red LoRaWAN disponible el nodo se conectará a ella automáticamente para comenzar la transmisión de datos en intervalos de 30 minutos. Cada vez que se reinicia el dispositivo, la válvula se cierra automáticamente por seguridad.

La conexión a la red LoRa no puede garantizarse al hacer uso de redes públicas que podrían verse saturadas si muchos dispositivos tratan de conectarse a una misma gateway.

Se ha comprobado el correcto funcionamiento del nodo en un entorno de laboratorio y para periodos breves de tiempo, y aunque en principio no parece tener problemas a la hora de transmitir o recibir mensajes, no puede asegurarse que vaya a tener un funcionamiento indefinido en un entorno real debido a la falta de pruebas para ello.

2.4. Mantenimiento

El mantenimiento requerido por el nodo es bajo, hay que se cambiar las pilas una vez estas se encuentren bajas de carga. Si estas son no recargables estas deberán ser recicladas conforme al Real Decreto 710/2015. Las pilas que se instalen deberán ser nuevas, no habiendo sido utilizadas previamente para evitar problemas al tener conectadas a la vez pilas con distintas cargas.

Para asegurarse que el dispositivo dura la mayor cantidad de tiempo posible será necesario revisar su estado físico de manera periódica, para comprobar que no existen daños derivados de la humedad o de la exposición a la luz solar directa.

En caso de que fuese necesario realizar una actualización del firmware del dispositivo, el proceso deberá ser llevado a cabo por un usuario familiarizado con la plataforma STM32 y haciendo uso de un ordenador desde el que se pueda comprobar que el proceso ha sido satisfactorio.

3. Condiciones particulares de fabricación

3.1. Diseño de la PCB

El diseño de la PCB debe cumplir con la normativa actual de la IPC aplicable a este proyecto, las normas generales y las de especificaciones de diseño.

Las normas generales a cumplir son:

- IPC-T-50: términos y definiciones
- IPC-2615: dimensiones y tolerancias
- IPC-D-325: documentación requerida para los circuitos impresos

Las normas de especificaciones de diseño a cumplir son:

- IPC-2612: requerimientos para la documentación y diagramas electrónicos
- IPC-2221: estándar genérico para diseñó de PCBs
- IPC-7351B: requerimientos genéricos para diseños con componentes SMD

3.2. Fabricación de la PCB

La fabricación de la PCB deberá realizarse por una compañía con la certificación IPC-A-600 para asegurar unos estándares mínimos de calidad.

Las placas se fabricarán con los siguientes parámetros:

- Debe ser una PCB de doble capa
- En el momento de la entrega las PCBs deben estar separadas de manera individual, no unidas como un panel
- El grosor de la placa debe ser de 1,6 mm
- El color externo de la placa puede ser verde u otro, siempre que permita la correcta lectura de la serigrafía
- El acabado de la superficie puede ser HASL o ENIG, siempre y cuando cumplan sean libres de plomo y cumplan con la directiva RoHS
- El grosor de las capas de cobre debe ser de 0.35mm o 1 onza de peso
- El material de fabricación del dieléctrico debe ser FR-4 con una temperatura de transmisión vítrea mínima de 130 °C
- El tamaño mimo de los agujeros de la placa debe ser de 0,3 mm
- Las vías pueden ser tanto cubiertas como descubiertas

- Se debe realizar un test eléctrico para asegurar la correcta fabricación de la placa

Si la placa fabricada no cumple con todos los parámetros anteriores no se podrá asegurar su correcto funcionamiento

3.3. Ensamblaje de la PCB

El ensamblaje de los componentes en la placa se realizará preferiblemente en un horno de soldadura para PCBs.

Si se dispone de un *stencil* del circuito se recomienda su uso para facilitar el proceso de colocación del estaño en los pads, hay que asegurarse que la pasta cubre el hueco del pad sin dejar oquedades ni sobrepasar la altura del *stencil*, si este proceso se realiza manualmente hay que prestar especial atención a no poner una cantidad excesiva de pasta en los pads para evitar que se produzcan puentes entre ellos. En cualquier caso, la pasta de soldadura que se utilice deberá ser libre de plomo para cumplir con la normativa RoHS y haber sido almacenada conforme a las recomendaciones del fabricante para asegurar un correcto funcionamiento.

El proceso de soldadura deberá realizarse utilizando el perfil de soldadura de Murata para el CMWX1ZZABZ-078 y sus recomendaciones para este proceso.

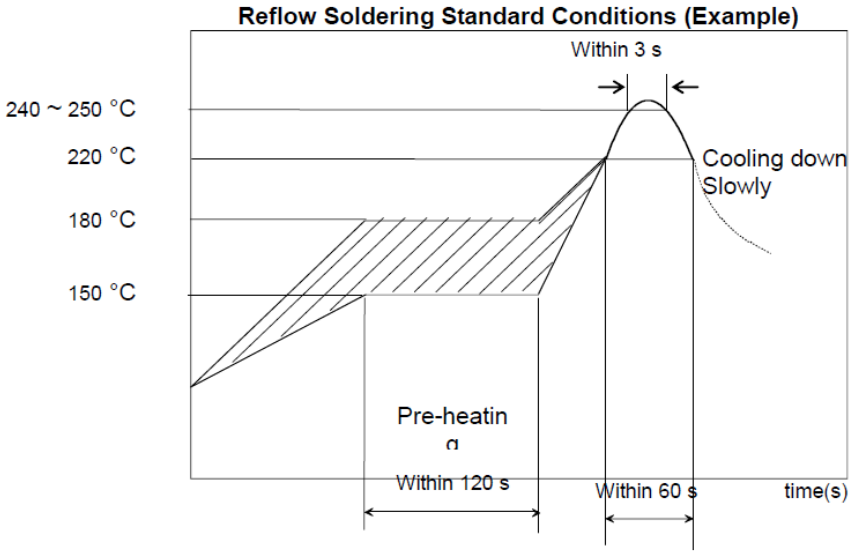


Figura 1. Perfil de soldadura recomendado por Murata

En caso de no disponer de un horno de soldadura es posible realizar el proceso con una pistola de aire caliente, en este caso se deberá intentar seguir el perfil de soldadura de la misma manera que se haría para un horno, pero en caso de que fuese necesario podrían calentarse los componentes durante un tiempo ligeramente mayor al horno para asegurarse de que se han soldado correctamente. La pistola de calor

deberá estar moviéndose sobre la placa de manera continua para evitar sobrecalentar ninguna zona de esta, antes de comenzar el proceso se debe comprobar el flujo de aire de la pistola para asegurarse que no sea tan alto como para poder desplazar fuera de los pads los componentes, ni tan bajo que resulte incapaz de manera homogénea la placa.

Una vez se complete el proceso de soldadura se deberán limpiar los componentes conforme a las recomendaciones establecidas por los fabricantes y a continuación medir con un multímetro el correcto funcionamiento de la placa.

3.4. Instalación de la electroválvula

La instalación de la electroválvula deberá realizarse una vez se haya comprobado que el paso del agua está cerrado. El primer paso será vaciar la tubería de los restos de agua que pudiesen quedar en su interior. A continuación, se retirará la válvula manual girando la rosca de la parte superior en sentido antihorario y la de la parte inferior en sentido horario. Seguidamente se colocará la electroválvula en el hueco dejado libre por la válvula manual y se apretarán las roscas. Una vez estén apretadas las roscas abriremos el paso del agua ligeramente para comprobar que no haya fugas por las juntas, si las hay, será necesario apretar más las roscas. Finalmente conectaremos la electroválvula al nodo y abriremos el paso del agua por completo.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

4. Presupuesto

Autor:

D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

ÍNDICE DEL PRESUPUESTO

1. Sobre el presupuesto

En el presupuesto se han recogido todos los gastos asociados al prototipo. En referencia al hardware, se incluye la compra de todos los componentes a ensamblar y la fabricación de la placa, así como el tiempo invertido en la elección de componentes, desarrollo de los esquemáticos, diseño de la PCB, montaje, pruebas para asegurar el correcto funcionamiento y desarrollo del software necesario. Relativo al servidor web, los gastos a tener en cuenta son el punto de acceso y otros componentes relacionados, necesarios para el correcto funcionamiento del sistema; así como las horas de trabajo invertidas para su puesta a punto. Aunque el servidor web y la aplicación de usuario son de uso gratuito, se han tenido en cuenta las horas de mano de obra necesarias para su configuración.

Todos los precios que recoge el presente documento en relación a los componentes eléctricos, así como al resto de elementos especificados a lo largo del presupuesto, datan de junio de 2020. Los componentes necesarios para la fabricación del nodo se han cotizado en el distribuidor autorizado Mouser.es, en el mercado de la electrónica existe una gran fluctuación en los precios de los componentes, así como una constante aparición de nuevos modelos o actualizaciones de los ya existentes. El presupuesto presenta una garantía y validez de un año.

El presupuesto contempla únicamente lo descrito en la propia memoria asociada a este, cualquier tipo de cambio o variación no reflejado en ella podrá tener un coste añadido respecto al presupuesto actual.

2. Materiales

2.1. Componentes electrónicos

En la siguiente tabla se desglosan los costes relacionados con los componentes electrónicos necesarios para la fabricación de una de las placas de sensores del nodo, el kit de desarrollo STMicroelectronics B-L072Z se encuentra especificado en el apartado 2.3 del presente documento.

Designator	Cantidad	Descripción	Unidad	Fabricante	Referencia Fabricante	Precio Unitario	Precio
IC1	1	Modulo LoRa + MCU	EA	Murata	CMWX1ZZABZ-078	14.18 €	14.18 €
J3,J5	2	Cabecera 8 pines THT	EA	Harwin	M20-7820842	0.99 €	1.98 €
U1	1	Sensor de temp. y hum.	EA	Texas Instruments	HDC2080DMBT	3.56 €	3.56 €
IC3	1	Sensor de luz ambiental	EA	AMS	TSL25911FN	1.97 €	1.97 €
IC4	1	Sensor de presion atm.	EA	TE Connectivity	MS563702BA03-50	2.51 €	2.51 €
J1	1	Conector SMA 50 ohm	EA	Linx Technologies	CONREVSMA001-G	3.11 €	3.11 €
J2	1	Cabecera 6 pines THT	EA	Harwin	M20-7820642	0.89 €	0.89 €
J4	1	Cabecera 10 pines THT	EA	Harwin	M20-7821042	1.25 €	1.25 €
J9,J10	2	Cabecera 4 pines THT	EA	Harwin	M20-7820442	0.73 €	1.46 €
ANT1	1	Conector UFL 50 ohm	EA	Linx Technologies	CONUFL001-SMD	0.62 €	0.62 €
C1,C2	2	Cap. 10uF 10V 1206	EA	Johanson Delectrics	100R18X106MV4E	0.69 €	1.38 €
C3,C4,C5,C9,C10	5	Cap. 0.1uF 10V 1210	EA	Johanson Delectrics	101S41W104MV4E	0.18 €	0.90 €
C6	1	Cap 1uF 25V 1210	EA	Taiyo Yuden	CGA2B1X7T0G105M050BC	0.43 €	0.43 €
IC2	1	I2C Level Shifter	EA	Texas Instruments	PCA9306DCTT	0.91 €	0.91 €
R1,R2,R3,R4,R5, R6,R8, R10	8	Res. 10k 1W 2010	EA	Bourns	CMP2010-FX-1002ELF	0.14 €	1.12 €
R7	1	Res. 200k 2W 2010	EA	TE Connectivity	3502200KFT	0.58 €	0.58 €
S1	1	Interruptor táctil	EA	C&K	PTS636_SK25J_SMTR_LFS	0.12 €	0.12 €
J6	1	Conector 3 pines THT	EA	TE Connectivity	826936-3	0.23 €	0.23 €
J7	1	Conector 6 pines THT	EA	TE Connectivity	826936-6	0.41 €	0.41 €
C7	1	Cap. 5.1 pF 100V 0201	EA	Murata	GRM0335C2A5R1CA01J	0.09 €	0.09 €
L1	1	Bobina 15 nH 0402	EA	Murata	LQG15WH15NJ02D	0.15 €	0.15 €
D1	1	Diodo 100V	EA	Fairchild	1N914	0.09 €	0.09 €
Q1	1	Transistor BJT NPN	EA	STMicroelectronics	BD139-10	0.41 €	0.41 €
Total							**38.35 €**

Tabla 1. Presupuesto componentes electrónicos

El coste total de este apartado asciende a **38,35 €**.

2.2. Fabricación del circuito impreso

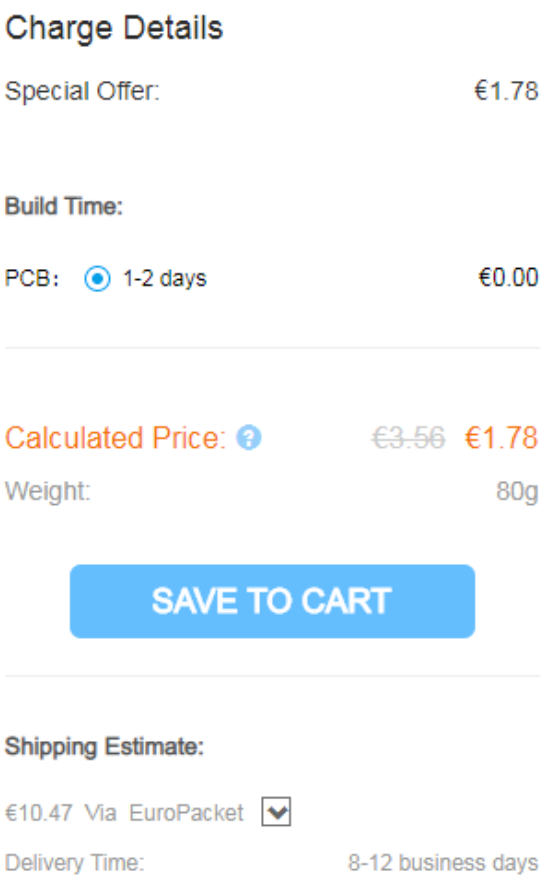
El circuito impreso resulta imprescindible para la fabricación del nodo, es necesario que el circuito fabricado cumpla con la normativa de fabricación IPC-600. El fabricante elegido es la compañía china JLCPCB.

Desde la propia web de la empresa es posible realizar todas las configuraciones necesarias para la fabricación de la PCB, una vez terminado este proceso se nos mostrará el precio de los circuitos impresos y los gastos de envío asociados.

Características	Opción
N° Capas	2
Dimensiones	83×55mm
Cantidad (5 min.)	5
Grosor de PCB	1.6 mm
Color	Verde
Acabado superficie	HASL
Grosor del cobre	1 oz
Dieléctrico	FR4 Tg 130-140

Tabla 2. Configuración de la PCB

El coste de fabricación y envío para 5 unidades son 12.25 € (2.45 € por unidad), 1.78 € por la fabricación y 10.47 € el concepto de envío.



El coste total de este apartado asciende a **12,25 €**.

2.3. Gateway y otros componentes

En este apartado se incluyen los costes de la Gateway y otros componentes imprescindibles para el correcto funcionamiento del proyecto, estos son el kit de STMicroelectronics B-L072, que forma parte del nodo, la sonda de humedad del suelo STEMMA, de Adafruit; la antena para el nodo y el programador ST-Link que se utilizó en parte del desarrollo del nodo. La Gateway fue adquirida del distribuidor oficial del producto, RS Componentes, el resto en Mouser.es.

Cantidad	Descripción	Unidad	Fabricante	Referencia Fabricante	Precio Un	Precio
1	Gateway LoRa	EA	The Things Industries	TTIG-868	72.90 €	72.90 €
1	Sensor humedad del suelo	EA	Adafruit	4026	6.75 €	6.75 €
1	Antena 868 MHz SMA 2dBi	EA	Abracon	AEACAC053010-S868	5.99 €	5.99 €
1	Programador ST-Link	EA	Adafruit	2548	11.25 €	11.25 €
Total						**96.89 €**

Tabla 3. Presupuesto de la Gateway y otros componentes

El coste total de este apartado asciende a **96,89 €**.

3. Mano de obra

Este apartado supone el gasto principal debido al gran porcentaje respecto al total del proyecto que supone el desarrollo del circuito impreso y de todo el software. El coste por hora se ha estimado en 13.69 € basándose en las tablas salariales para un ingeniero técnico en la industria del metal (Sector en el que se engloba la fabricación de productos electrónicos) para la provincia de Valencia conforme a la resolución del Boletín Oficial de la provincia de Valencia, número 30 con fecha del 12 de febrero del 2019. El salario percibido puede ser mayor que el presente en las tablas, pero como en este caso se trataría de un recién graduado, se ha decidido utilizar el salario base actual.

Como costes de mano de obra se incluyen cuatro apartados diferentes. El primer apartado es el diseño eléctrico, donde se circunscribe el diseño de los esquemáticos y del *layout* del circuito impreso, así como la preparación de los archivos a enviar al fabricante.

El segundo apartado consiste en el ensamblaje, limpieza y testeo del prototipo.

El tercer apartado consiste en el desarrollo completo de todo el software del proyecto, esto es, desarrollar las librerías de los sensores y el código de la aplicación principal, depurar y comprobar el correcto funcionamiento de este y, finalmente, configurar la Gateway y la API.

El apartado cuatro consiste en la redacción y generación de toda la documentación necesaria para el proyecto.

Unidad	Descripción	Parcial	Cantidad	Precio	Total
h	Diseño eléctrico		81	13.69 €	1,108.89 €
	1 - Diseño del esquema electrónico en KiCad	50			
	2 - Layout del circuito y diseño PCB	30			
	3 - Preparación documentación de fabricaciór	1			
h	Ensamblaje del prototipo		4	13.69 €	54.76 €
	1 - Montaje de la PCB. Soldadura manual	3			
	2 - Inspección visual	0.25			
	3 - Limpieza de restos de flux	0.25			
	4 - Test eléctrico de continuidad y polaridad	0.5			
h	Desarrollo de la programación y depuración		160	13.69 €	2,190.40 €
	1 - Desarrollo librerias	60			
	2 - Desarrollo código fuente aplicación final	60			
	3 - Depuración y programación prototipo	30			
h	4 - Configuración de la plataforma online	10			
	Redacción y generación de documentación		58	13.69 €	794.02 €
	1 - Redacción memoria de proyecto	50			
	2 - Generación de planos	3			
	3 - Generación de presupuestos	5			
Total					**4,148.07 €**

Tabla 4. Presupuesto de la mano de obra

El coste total de este apartado asciende a **4148,07 €**.

4. Amortización de equipos

Para la correcta realización del proyecto se precisa de diversos dispositivos, algunos de ellos con un alto coste, por ello se debe calcular la amortización de los mismos respecto al número de horas que han sido utilizados en el proyecto y la disminución proporcional de su vida útil.

Para calcular la vida útil de un dispositivo se multiplicará el número de horas de uso diarias por los 365 días del año y por la vida estimada en años del producto.

El valor obtenido dividirá el precio de cada producto para obtener el precio por hora de cada dispositivo. Este valor multiplicado por el número de horas que haya sido utilizado en el proyecto nos dará como resultado el coste en el proyecto que deberá suponer cada dispositivo para poder amortizar su uso.

Unidad	Descripción	Coste Equipo	Vida Útil	Cantidad	Precio	Total
h	Ordenador Portatil Lenovo T480	799.00 €	15000	300	0.053 €	15.980 €
h	Multímetro UNI-T 139C	39.95 €	10950	5	0.004 €	0.018 €
h	Aduino Uno	19.99 €	16000	4	0.001 €	0.005 €
h	Soldador de estaño	19.99 €	1500	2	0.013 €	0.027 €
h	Estacion de aire caliente	39.99 €	2500	1	0.016 €	0.016 €
Total						**16.05 €**

Tabla 5. Presupuesto de amortización de los dispositivos utilizados

El coste total de este apartado asciende a **16,05 €**.

5. Resumen

En este apartado se exponen los gastos calculados a lo largo de los apartados anteriores a fecha de julio de 2020.

Sección	Descripción	Parcial	Total
2	Materliales		147.49 €
2.1	Componentes electrónicos	38.35 €	
2.2	Fabricación circuito impreso	12.25 €	
2.3	Gateway y otros componentes	96.89 €	
3	Mano de obra		4,148.07 €
4	Equipos		16.05 €
Subtotal			**4,311.61 €**

Gastos adicionales	10.00%	431.16 €
Beneficio industrial	6.00%	258.70 €
Total sin IVA		**5,001.47 €**

IVA	21.00%	1,050.31 €
TOTAL COSTE PROYECTO		**6,051.78 €**

Tabla 5. Resumen de gastos del proyecto

La elaboración completa del proyecto supone un gasto total de SEIS MIL CINCUENTA Y UN EUROS CON SETENTA Y OCHO CÉNTIMOS (**6051,78 €**).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DEL CULTIVO DEL CAQUI MEDIANTE TECNOLOGÍA IOT

5. Anexos

Autor:

D. Carlos Chulbi Perales

Tutor:

D. Ángel Perles Ivars

Valencia, julio de 2020

ÍNDICE DEL ANEXO 1

1. Librerías de los sensores y actuadores

1.1. Adafruit_SoilMoisture.h

```
1. /**************************************************************************
   ******/
2. /*!
3.     @file     Adafruit_TSL2591.h
4.     @author   KT0WN (adafruit.com)
5.
6.     This is a library for the Adafruit TSL2591 breakout board
7.     This library works with the Adafruit TSL2591 breakout
8.     ----> https://www.adafruit.com/products/1980
9.
10.      Check out the links above for our tutorials and wiring diagrams
11.      These chips use I2C to communicate
12.
13.      Adafruit invests time and resources providing this open source
   code,
14.      please support Adafruit and open-source hardware by purchasing
15.      products from Adafruit!
16.  */
17.  /**************************************************************************
   ********/
18.
19.  #ifndef _TSL2591_H_
20.  #define _TSL2591_H_
21.
22.  #include <Adafruit_Sensor.h>
23.  #include <Arduino.h>
24.  #include <Wire.h>
25.
26.  #define TSL2591_VISIBLE (2)       ///< (channel 0) - (channel 1)
27.  #define TSL2591_INFRARED (1)      ///< channel 1
28.  #define TSL2591_FULLSPECTRUM (0) ///< channel 0
29.
30.  #define TSL2591_ADDR (0x29) ///< Default I2C address
31.
32.  #define TSL2591_COMMAND_BIT
   \
33.    (0xA0) ///< 1010 0000: bits 7 and 5 for 'command normal'
34.
35.  ///! Special Function Command for "Clear ALS and no persist ALS
   interrupt"
36.  #define TSL2591_CLEAR_INT (0xE7)
37.  ///! Special Function Command for "Interrupt set - forces an
   interrupt"
38.  #define TSL2591_TEST_INT (0xE4)
39.
40.  #define TSL2591_WORD_BIT (0x20)  ///< 1 = read/write word (rather
   than byte)
41.  #define TSL2591_BLOCK_BIT (0x10) ///< 1 = using block read/write
42.
43.  #define TSL2591_ENABLE_POWEROFF (0x00) ///< Flag for ENABLE
   register to disable
```

```
44.  #define TSL2591_ENABLE_POWERON (0x01)  ///< Flag for ENABLE
   register to enable
45.  #define TSL2591_ENABLE_AEN
   \
46.    (0x02) ///< ALS Enable. This field activates ALS function.
   Writing a one
47.           ///< activates the ALS. Writing a zero disables the ALS.
48.  #define TSL2591_ENABLE_AIEN
   \
49.    (0x10) ///< ALS Interrupt Enable. When asserted permits ALS
   interrupts to be
50.           ///< generated, subject to the persist filter.
51.  #define TSL2591_ENABLE_NPIEN
   \
52.    (0x80) ///< No Persist Interrupt Enable. When asserted NP
   Threshold conditions
53.           ///< will generate an interrupt, bypassing the persist
   filter
54.
55.  #define TSL2591_LUX_DF (408.0F)   ///< Lux cooefficient
56.  #define TSL2591_LUX_COEFB (1.64F) ///< CH0 coefficient
57.  #define TSL2591_LUX_COEFC (0.59F) ///< CH1 coefficient A
58.  #define TSL2591_LUX_COEFD (0.86F) ///< CH2 coefficient B
59.
60.  /// TSL2591 Register map
61.  enum {
62.    TSL2591_REGISTER_ENABLE = 0x00,          // Enable register
63.    TSL2591_REGISTER_CONTROL = 0x01,         // Control register
64.    TSL2591_REGISTER_THRESHOLD_AILTL = 0x04, // ALS low threshold
   lower byte
65.    TSL2591_REGISTER_THRESHOLD_AILTH = 0x05, // ALS low threshold
   upper byte
66.    TSL2591_REGISTER_THRESHOLD_AIHTL = 0x06, // ALS high threshold
   lower byte
67.    TSL2591_REGISTER_THRESHOLD_AIHTH = 0x07, // ALS high threshold
   upper byte
68.    TSL2591_REGISTER_THRESHOLD_NPAILTL =
69.        0x08, // No Persist ALS low threshold lower byte
70.    TSL2591_REGISTER_THRESHOLD_NPAILTH =
71.        0x09, // No Persist ALS low threshold higher byte
72.    TSL2591_REGISTER_THRESHOLD_NPAIHTL =
73.        0x0A, // No Persist ALS high threshold lower byte
74.    TSL2591_REGISTER_THRESHOLD_NPAIHTH =
75.        0x0B, // No Persist ALS high threshold higher byte
76.    TSL2591_REGISTER_PERSIST_FILTER = 0x0C, // Interrupt persistence
   filter
77.    TSL2591_REGISTER_PACKAGE_PID = 0x11,    // Package Identification
78.    TSL2591_REGISTER_DEVICE_ID = 0x12,      // Device Identification
79.    TSL2591_REGISTER_DEVICE_STATUS = 0x13,  // Internal Status
80.    TSL2591_REGISTER_CHAN0_LOW = 0x14,      // Channel 0 data, low
   byte
81.    TSL2591_REGISTER_CHAN0_HIGH = 0x15,     // Channel 0 data, high
   byte
82.    TSL2591_REGISTER_CHAN1_LOW = 0x16,      // Channel 1 data, low
   byte
83.    TSL2591_REGISTER_CHAN1_HIGH = 0x17,     // Channel 1 data, high
   byte
```

```
84.  };
85.
86.  /// Enumeration for the sensor integration timing
87.  typedef enum {
88.    TSL2591_INTEGRATIONTIME_100MS = 0x00, // 100 millis
89.    TSL2591_INTEGRATIONTIME_200MS = 0x01, // 200 millis
90.    TSL2591_INTEGRATIONTIME_300MS = 0x02, // 300 millis
91.    TSL2591_INTEGRATIONTIME_400MS = 0x03, // 400 millis
92.    TSL2591_INTEGRATIONTIME_500MS = 0x04, // 500 millis
93.    TSL2591_INTEGRATIONTIME_600MS = 0x05, // 600 millis
94.  } tsl2591IntegrationTime_t;
95.
96.  /// Enumeration for the persistance filter (for interrupts)
97.  typedef enum {
98.    //  bit 7:4: 0
99.    TSL2591_PERSIST_EVERY = 0x00, // Every ALS cycle generates an
   interrupt
100.   TSL2591_PERSIST_ANY = 0x01,   // Any value outside of threshold
   range
101.   TSL2591_PERSIST_2 = 0x02,     // 2 consecutive values out of
   range
102.   TSL2591_PERSIST_3 = 0x03,     // 3 consecutive values out of
   range
103.   TSL2591_PERSIST_5 = 0x04,     // 5 consecutive values out of
   range
104.   TSL2591_PERSIST_10 = 0x05,    // 10 consecutive values out of
   range
105.   TSL2591_PERSIST_15 = 0x06,    // 15 consecutive values out of
   range
106.   TSL2591_PERSIST_20 = 0x07,    // 20 consecutive values out of
   range
107.   TSL2591_PERSIST_25 = 0x08,    // 25 consecutive values out of
   range
108.   TSL2591_PERSIST_30 = 0x09,    // 30 consecutive values out of
   range
109.   TSL2591_PERSIST_35 = 0x0A,    // 35 consecutive values out of
   range
110.   TSL2591_PERSIST_40 = 0x0B,    // 40 consecutive values out of
   range
111.   TSL2591_PERSIST_45 = 0x0C,    // 45 consecutive values out of
   range
112.   TSL2591_PERSIST_50 = 0x0D,    // 50 consecutive values out of
   range
113.   TSL2591_PERSIST_55 = 0x0E,    // 55 consecutive values out of
   range
114.   TSL2591_PERSIST_60 = 0x0F,    // 60 consecutive values out of
   range
115. } tsl2591Persist_t;
116.
117. /// Enumeration for the sensor gain
118. typedef enum {
119.   TSL2591_GAIN_LOW = 0x00,  /// low gain (1x)
120.   TSL2591_GAIN_MED = 0x10,  /// medium gain (25x)
121.   TSL2591_GAIN_HIGH = 0x20, /// medium gain (428x)
122.   TSL2591_GAIN_MAX = 0x30,  /// max gain (9876x)
123. } tsl2591Gain_t;
124.
```

```
125. /**************************************************************************
   ********/
126. /*!
127.     @brief  Class that stores state and functions for interacting
   with TSL2591
128.    Light Sensor
129. */
130. /**************************************************************************
   ********/
131. class Adafruit_TSL2591 : public Adafruit_Sensor {
132. public:
133.   Adafruit_TSL2591(int32_t sensorID = -1);
134.
135.   boolean begin(TwoWire *theWire);
136.   boolean begin();
137.   void enable(void);
138.   void disable(void);
139.
140.   float calculateLux(uint16_t ch0, uint16_t ch1);
141.   void setGain(tsl2591Gain_t gain);
142.   void setTiming(tsl2591IntegrationTime_t integration);
143.   uint16_t getLuminosity(uint8_t channel);
144.   uint32_t getFullLuminosity();
145.
146.   tsl2591IntegrationTime_t getTiming();
147.   tsl2591Gain_t getGain();
148.
149.   // Interrupt
150.   void clearInterrupt(void);
151.   void registerInterrupt(uint16_t lowerThreshold, uint16_t
   upperThreshold,
152.                          tsl2591Persist_t persist);
153.   uint8_t getStatus();
154.
155.   /* Unified Sensor API Functions */
156.   bool getEvent(sensors_event_t *);
157.   void getSensor(sensor_t *);
158.
159. private:
160.   TwoWire *_i2c;
161.
162.   void write8(uint8_t r);
163.   void write8(uint8_t r, uint8_t v);
164.   uint16_t read16(uint8_t reg);
165.   uint8_t read8(uint8_t reg);
166.
167.   tsl2591IntegrationTime_t _integration;
168.   tsl2591Gain_t _gain;
169.   int32_t _sensorID;
170.
171.   boolean _initialized;
172. };
173. #endif
```

1.2. Adafruit_SoilMoisture.cpp

```
1. /*!
2.  * @file Adafruit_SoilMoisture.cpp
3.  *
4.  *
5.  */
6.
7. #include "Adafruit_SoilMoisture.h"
8. #include "util_console.h"
9. /*!
10.
   ********************************************************************
   ********************
11.   *  @brief      Start the seesaw
12.   *
13.   *                              This should be called when your
   sketch is
14.   *connecting to the seesaw
15.   *
16.   *  @param      &i2cHandler
17.   *  @param      i2caddr the I2C address of the seesaw
18.   *
19.   *  @return     true if we could connect to the seesaw, false
   otherwise
20.
   ********************************************************************
   ******************/
21.  void Adafruit_SoilMoisture::begin(I2C_HandleTypeDef &i2cHandler,
   uint8_t i2caddr)
22.  {
23.          _i2cHandler = i2cHandler;
24.          _i2caddr = i2caddr;
25.  }
26.
27.  /*!
28.
   ********************************************************************
   *********
29.   *  @brief      read the analog value on an capacitive touch-
   enabled pin.
30.   *
31.   *
32.   *  @return     the analog value. This is an integer between 0 and
   1023
33.
   ********************************************************************
   *******/
34.  uint16_t Adafruit_SoilMoisture::moistureRead(void) {
35.    uint8_t buf[2],reg[2];
36.    uint16_t return_value;
37.
38.          reg[0] = SEESAW_TOUCH_BASE;
39.          reg[1] = SEESAW_TOUCH_CHANNEL_OFFSET;
40.
41.    HAL_Delay(10);
```

```
42.          HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1, reg,
   2, 100);
43.          HAL_Delay(10);
44.          HAL_I2C_Master_Receive(&_i2cHandler, _i2caddr << 1, buf, 2,
   100);
45.
46.    return_value = (buf[0] << 8) | buf[1];
47.    return return_value;
48.  }
```

1.3. Electrovalve.h

```
1. /**
2. @file electrovalve .h
3. */
4.
5. # ifndef VALVE_H
6. # define VALVE_H
7.
8. #include "stm32l0xx_hal.h"
9.
10.  typedef enum { VALVE_OPEN , VALVE_CLOSE } TValveState ;
11.  void valve_Init ( void );
12.  void valve_Set ( TValveState state );
13.
14.  # endif
```

1.4. Electrovalve.cpp

```
1. /**
2.    @file electrovalve.c
3.    @brief Basic LED handling for libraries example
4.    @author Carlos Chulbi
5.
6. */
7.
8. #include "stm32l0xx_hal.h"      // cabeceras proporcionadas por St
   para simplificar el uso de los perifericos
9. #include "electrovalve.h"
10.
11.  /*****************************************************************
   ******************************/
12.  /**
13.    * @brief Preparing pin corresponding to PA9
14.    * @return none
15.    */
16.  void valve_Init(void) {
17.
```

```
18.     GPIO_InitTypeDef  GPIO_InitStruct;  // estructura donde se pone
   la configuracion deseada
19.
20.     __HAL_RCC_GPIOA_CLK_ENABLE();                       // darle
   reloj al periferico, AHORA VIVE!
21.
22.         /*Configure  GPIO pin Output Level */
23.         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
24.
25.    /* Configure the GPIO_LED pin */
26.    GPIO_InitStruct.Pin = GPIO_PIN_9;
            // pin que desamos configurar
27.    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; // lo vamos a usar
   como salida en push-pull
28.    GPIO_InitStruct.Pull = GPIO_NOPULL;         // desactivar pulls
29.    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
     // actualizacion pin
30.    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);             // hacer
   efectiva configuracion puerto
31.
32.  }
33.
34.  void valve_Set(TValveState state)
35.  {
36.  if(state==VALVE_CLOSE)
37.          {
38.          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
39.          }
40.  else if(state==VALVE_OPEN)
41.          {
42.           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
43.
44.          }
45.  }
46.
47.  /*** End of file
   ****************************************************************************
   ************/
```

1.5. HDC2080.h

```
1. /*
2.   HDC2080.h
3.   HDC2010.h originally created by: Brandon Fisher, August 1st 2017
4.
5.   This code is release AS-IS into the public domain, no guarantee or
   warranty is given.
6.
7.   Description: This header file accompanies HDC2080.cpp, and declares
   all methods, fields,
8.   and constants used in the source code.
9. */
10.
```

```
11.  #ifndef HDC2080_H
12.  #define HDC2080_h
13.
14.  #ifdef __cplusplus
15.  extern "C" {
16.  #endif
17.
18.  #include "stm32l0xx_hal.h"
19.  #include "stm32l0xx_hal_i2c.h"
20.  #include "util_console.h"
21.
22.  //  Constants for setting measurement resolution
23.  #define FOURTEEN_BIT 0
24.  #define ELEVEN_BIT 1
25.  #define NINE_BIT  2
26.
27.  //  Constants for setting sensor mode
28.  #define TEMP_AND_HUMID 0
29.  #define TEMP_ONLY         1
30.  #define HUMID_ONLY        2
31.  #define ACTIVE_LOW        0
32.  #define ACTIVE_HIGH       1
33.  #define LEVEL_MODE             0
34.  #define COMPARATOR_MODE 1
35.
36.  //  Constants for setting sample rate
37.  #define MANUAL                 0
38.  #define TWO_MINS               1
39.  #define ONE_MINS               2
40.  #define TEN_SECONDS            3
41.  #define FIVE_SECONDS    4
42.  #define ONE_HZ                 5
43.  #define TWO_HZ                 6
44.  #define FIVE_HZ                7
45.
46.  //Define HDC2080 I2C address
47.  #define HDC2080_addr 0x40
48.
49.  class HDC2080
50.  {
51.          public:
52.
53.                  void begin(I2C_HandleTypeDef &i2cHandler, uint8_t
     i2caddr);                                  // Join I2C bus
54.                  float readTemp(void);
     // Returns the temperature in degrees C
55.                  float readHumidity(void);
     // Returns the relative humidity
56.                  void enableHeater(void);
     // Enables the heating element
57.                  void disableHeater(void);
     // Disables the heating element
58.                  void setLowTemp(float temp);                  //
     Sets low threshold temperature (in c)
59.                  void setHighTemp(float temp);                 //
     Sets high threshold temperature (in c)
```

```
60.                 void setHighHumidity(float humid);            //
   Sets high Humiditiy threshold
61.                 void setLowHumidity(float humid);             //
   Sets low Humidity threshold
62.                 float readLowHumidityThreshold(void); // Returns
   contents of low humidity threshold register
63.                 float readHighHumidityThreshold(void);// Returns
   contents of high humidity threshold register
64.                 float readLowTempThreshold(void);             //
   Returns contents of low temperature threshold register (in C)
65.                 float readHighTempThreshold(void);            //
   Returns contents of high temperature threshold register (in C)
66.                 void triggerMeasurement(void);                //
   Triggers a manual temperature/humidity reading
67.                 void reset(void);
           // Triggers a software reset
68.                 void enableInterrupt(void);
    // Enables the interrupt/DRDY pin
69.                 void disableInterrupt(void);                  //
   Disables the interrupt/DRDY pin (High Z)
70.                 uint8_t readInterruptStatus(void);            //
   Reads the status of the interrupt register
71.                 void clearMaxTemp(void);
    // Clears the Maximum temperature register
72.                 void clearMaxHumidity(void);                  //
   Clears the Maximum humidity register
73.                 float readMaxTemp(void);
    // Reads the maximum temperature register
74.                 float readMaxHumidity(void);                  //
   Reads the maximum humidity register
75.                 void enableThresholdInterrupt(void);  // Enables
   high and low temperature/humidity interrupts
76.                 void disableThresholdInterrupt(void); // Disables
   high and low temperature/humidity interrupts
77.                 void enableDRDYInterrupt(void);
    // Enables data ready interrupt
78.                 void disableDRDYInterrupt(void);              //
   Disables data ready interrupt
79.
80.
81.
82.                 /* Sets Temperature & Humidity Resolution, 3 options
83.                    0 - 14 bit
84.                    1 - 11 bit
85.                    2 - 9 bit
86.                    default - 14 bit
                    */
87.                 void setTempRes(int resolution);
88.                 void setHumidRes(int resolution);
89.
90.                 /* Sets measurement mode, 3 options
91.                    0 - Temperature and Humidity
92.                    1 - Temperature only
93.                    2 - Humidity only
94.                    default - Temperature & Humidity
    */
95.                 void setMeasurementMode(int mode);
```

```
96.
97.                 /* Sets reading rate, 8 options
98.                    0 - Manual
99.                    1 - reading every 2 minutes
100.                   2 - reading every minute
101.                   3 - reading every ten seconds
102.                   4 - reading every 5 seconds
103.                   5 - reading every second
104.                   6 - reading at 2Hz
105.                   7 - reading at 5Hz
106.                   default - Manual            */
107.                void setRate(int rate);
108.
109.                /* Sets Interrupt polarity, 2 options
110.                   0 - Active Low
111.                   1 - Active High
112.                   default - Active Low                        */

113.                void setInterruptPolarity(int polarity);
114.
115.                /* Sets Interrupt mode, 2 options
116.                   0 - Level sensitive
117.                   1 - Comparator mode
118.                   default - Level sensitive   */
119.                void setInterruptMode(int polarity);
120.
121.        private:
122.                uint8_t _i2caddr;
                                    // Address of sensor
123.                I2C_HandleTypeDef _i2cHandler;
124.
125.                uint8_t readReg(uint8_t reg);
    // Reads a given register, returns 1 byte
126.                void writeReg(uint8_t reg, uint8_t data);     //
   Writes a byte of data to one register
127.
128.
129. };
130.
131. #ifdef __cplusplus
132. }
133. #endif
134.
135. #endif
```

1.6. HDC2080.cpp

```
1. /*
2.   HDC2080.cpp
3.   HDC2010.cpp originally created by: Brandon Fisher, August 1st 2017
4.
5.   This code is release AS-IS into the public domain, no guarantee or
   warranty is given.
```

```
6.
7.   Description: This library facilitates communication with, and
   configuration of,
8.   the HDC2080 Temperature and Humidity Sensor. It makes extensive use
   of the
9.   Wire.H library, and should be useable with both Arduino and
   Energia.
10.  */
11.
12.
13.  #include <HDC2080.h>
14.
15.  //Define Register Map
16.          #define TEMP_LOW 0x00
17.          #define TEMP_HIGH 0x01
18.          #define HUMID_LOW 0x02
19.          #define HUMID_HIGH 0x03
20.          #define INTERRUPT_DRDY 0x04
21.          #define TEMP_MAX 0x05
22.          #define HUMID_MAX 0x06
23.          #define INTERRUPT_CONFIG 0x07
24.          #define TEMP_OFFSET_ADJUST 0x08
25.          #define HUM_OFFSET_ADJUST 0x09
26.          #define TEMP_THR_L 0x0A
27.          #define TEMP_THR_H 0x0B
28.          #define HUMID_THR_L 0x0C
29.          #define HUMID_THR_H 0x0D
30.          #define CONFIG 0x0E
31.          #define MEASUREMENT_CONFIG 0x0F
32.          #define MID_L 0xFC
33.          #define MID_H 0xFD
34.          #define DEVICE_ID_L 0xFE
35.          #define DEVICE_ID_H 0xFF
36.
37.
38.  void HDC2080::begin(I2C_HandleTypeDef &i2cHandler, uint8_t i2caddr)
39.  {
40.          _i2cHandler = i2cHandler;
41.          _i2caddr = i2caddr;
42.
43.          reset();
44.
45.          setMeasurementMode(TEMP_AND_HUMID);
46.          setRate(ONE_HZ);
47.          setTempRes(FOURTEEN_BIT);
48.          setHumidRes(FOURTEEN_BIT);
49.  }
50.
51.  float HDC2080::readTemp(void)
52.  {
53.          triggerMeasurement();
54.
55.          uint8_t byte[2];
56.          uint16_t temperature_raw;
57.          float temperature_return;
58.
59.          byte[0] = readReg(TEMP_LOW);
```

```
60.          byte[1] = readReg(TEMP_HIGH);
61.
62.          temperature_raw = (unsigned int)byte[1] << 8 | byte[0];
63.
64.          temperature_return = (float)(temperature_raw) * 165 / 65536
   - 40;
65.
66.          return temperature_return;
67.
68.  }
69.
70.  float HDC2080::readHumidity(void)
71.  {
72.          triggerMeasurement();
73.
74.          uint8_t byte[2];
75.          uint16_t humidity_raw;
76.          float humidity_return;
77.
78.          byte[0] = readReg(HUMID_LOW);
79.          byte[1] = readReg(HUMID_HIGH);
80.
81.          humidity_raw = (unsigned int)byte[1] << 8 | byte[0];
82.
83.          humidity_return = (float)(humidity_raw)/( 65536 )* 100;
84.
85.          return (float)(humidity_raw)/( 65536 )* 100;
86.
87.  }
88.
89.  void HDC2080::enableHeater(void)
90.  {
91.          uint8_t configContents;        //Stores current contents of
   config register
92.
93.          configContents = readReg(CONFIG);
94.
95.          //set bit 3 to 1 to enable heater
96.          configContents = (configContents | 0x08);
97.
98.          writeReg(CONFIG, configContents);
99.
100. }
101.
102. void HDC2080::disableHeater(void)
103. {
104.         uint8_t configContents;        //Stores current contents of
   config register
105.
106.         configContents = readReg(CONFIG);
107.
108.         //set bit 3 to 0 to disable heater (all other bits 1)
109.         configContents = (configContents & 0xF7);
110.         writeReg(CONFIG, configContents);
111.
112. }
113.
```

```
114. uint8_t HDC2080::readReg(uint8_t reg)
115. {
116.     uint8_t return_value;
117.
118.     /* Check the communication status */
119.     HAL_I2C_Mem_Read(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, &return_value, 1, 100);
120.
121.     return return_value;
122. }
123.
124. void HDC2080::writeReg(uint8_t reg, uint8_t data)
125. {
126.     /* Check the communication status */
127.     HAL_I2C_Mem_Write(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, &data, 1, 100);
128.
129. }
130.
131. void HDC2080::setLowTemp(float temp)
132. {
133.         uint8_t temp_thresh_low;
134.
135.         // Verify user is not trying to set value outside bounds
136.         if (temp < -40)
137.         {
138.                 temp = -40;
139.         }
140.         else if (temp > 125)
141.         {
142.                 temp = 125;
143.         }
144.
145.         // Calculate value to load into register
146.         temp_thresh_low = (uint8_t)(256 * (temp + 40)/165);
147.
148.         writeReg(TEMP_THR_L, temp_thresh_low);
149.
150. }
151.
152. void HDC2080::setHighTemp(float temp)
153. {
154.         uint8_t temp_thresh_high;
155.
156.         // Verify user is not trying to set value outside bounds
157.         if (temp < -40)
158.         {
159.                 temp = -40;
160.         }
161.         else if (temp > 125)
162.         {
163.                 temp = 125;
164.         }
165.
166.         // Calculate value to load into register
167.         temp_thresh_high = (uint8_t)(256 * (temp + 40)/165);
168.
```

```
169.        writeReg(TEMP_THR_H, temp_thresh_high);
170.
171. }
172.
173. void HDC2080::setHighHumidity(float humid)
174. {
175.        uint8_t humid_thresh;
176.
177.        // Verify user is not trying to set value outside bounds
178.        if (humid < 0)
179.        {
180.                humid = 0;
181.        }
182.        else if (humid > 100)
183.        {
184.                humid = 100;
185.        }
186.
187.        // Calculate value to load into register
188.        humid_thresh = (uint8_t)(256 * (humid)/100);
189.
190.        writeReg(HUMID_THR_H, humid_thresh);
191.
192. }
193.
194. void HDC2080::setLowHumidity(float humid)
195. {
196.        uint8_t humid_thresh;
197.
198.        // Verify user is not trying to set value outside bounds
199.        if (humid < 0)
200.        {
201.                humid = 0;
202.        }
203.        else if (humid > 100)
204.        {
205.                humid = 100;
206.        }
207.
208.        // Calculate value to load into register
209.        humid_thresh = (uint8_t)(256 * (humid)/100);
210.
211.        writeReg(HUMID_THR_L, humid_thresh);
212.
213. }
214.
215. //  Return humidity from the low threshold register
216. float HDC2080::readLowHumidityThreshold(void)
217. {
218.        uint8_t regContents;
219.
220.        regContents = readReg(HUMID_THR_L);
221.
222.        return (float)regContents * 100/256;
223.
224. }
225.
```

```
226. //  Return humidity from the high threshold register
227. float HDC2080::readHighHumidityThreshold(void)
228. {
229.         uint8_t regContents;
230.
231.         regContents = readReg(HUMID_THR_H);
232.
233.         return (float)regContents * 100/256;
234.
235. }
236.
237. //  Return temperature from the low threshold register
238. float HDC2080::readLowTempThreshold(void)
239. {
240.         uint8_t regContents;
241.
242.         regContents = readReg(TEMP_THR_L);
243.
244.         return (float)regContents * 165/256 - 40;
245.
246. }
247.
248. //  Return temperature from the high threshold register
249. float HDC2080::readHighTempThreshold(void)
250. {
251.         uint8_t regContents;
252.
253.         regContents = readReg(TEMP_THR_H);
254.
255.         return (float)regContents * 165/256 - 40;
256.
257. }
258.
259.
260. /* Upper two bits of the MEASUREMENT_CONFIG register controls
261.    the temperature resolution*/
262. void HDC2080::setTempRes(int resolution)
263. {
264.         uint8_t configContents;
265.         configContents = readReg(MEASUREMENT_CONFIG);
266.
267.         switch(resolution)
268.         {
269.                 case FOURTEEN_BIT:
270.                         configContents = (configContents & 0x3F);
271.                         break;
272.
273.                 case ELEVEN_BIT:
274.                         configContents = (configContents & 0x7F);
275.                         configContents = (configContents | 0x40);
276.                         break;
277.
278.                 case NINE_BIT:
279.                         configContents = (configContents & 0xBF);
280.                         configContents = (configContents | 0x80);
281.                         break;
282.
```

```
283.                default:
284.                        configContents = (configContents & 0x3F);
285.        }
286.
287.        writeReg(MEASUREMENT_CONFIG, configContents);
288.
289. }
290. /*  Bits 5 and 6 of the MEASUREMENT_CONFIG register controls
291.     the humidity resolution*/
292. void HDC2080::setHumidRes(int resolution)
293. {
294.        uint8_t configContents;
295.        configContents = readReg(MEASUREMENT_CONFIG);
296.
297.        switch(resolution)
298.        {
299.                case FOURTEEN_BIT:
300.                        configContents = (configContents & 0xCF);
301.                        break;
302.
303.                case ELEVEN_BIT:
304.                        configContents = (configContents & 0xDF);
305.                        configContents = (configContents | 0x10);
306.                        break;
307.
308.                case NINE_BIT:
309.                        configContents = (configContents & 0xEF);
310.                        configContents = (configContents | 0x20);
311.                        break;
312.
313.                default:
314.                        configContents = (configContents & 0xCF);
315.        }
316.
317.        writeReg(MEASUREMENT_CONFIG, configContents);
318. }
319.
320. /*  Bits 2 and 1 of the MEASUREMENT_CONFIG register controls
321.     the measurement mode  */
322. void HDC2080::setMeasurementMode(int mode)
323. {
324.        uint8_t configContents;
325.        configContents = readReg(MEASUREMENT_CONFIG);
326.
327.        switch(mode)
328.        {
329.                case TEMP_AND_HUMID:
330.                        configContents = (configContents & 0xF9);
331.                        break;
332.
333.                case TEMP_ONLY:
334.                        configContents = (configContents & 0xFC);
335.                        configContents = (configContents | 0x02);
336.                        break;
337.
338.                case HUMID_ONLY:
339.                        configContents = (configContents & 0xFD);
```

```
340.                        configContents = (configContents | 0x04);
341.                        break;
342.
343.                default:
344.                        configContents = (configContents & 0xF9);
345.        }
346.
347.        writeReg(MEASUREMENT_CONFIG, configContents);
348. }
349.
350. /*  Bit 0 of the MEASUREMENT_CONFIG register can be used
351.     to trigger measurements  */
352. void HDC2080::triggerMeasurement(void)
353. {
354.        uint8_t configContents;
355.        configContents = readReg(MEASUREMENT_CONFIG);
356.
357.        configContents = (configContents | 0x01);
358.        writeReg(MEASUREMENT_CONFIG, configContents);
359. }
360.
361. /*  Bit 7 of the CONFIG register can be used to trigger a
362.     soft reset  */
363. void HDC2080::reset(void)
364. {
365.        uint8_t configContents;
366.        configContents = readReg(CONFIG);
367.
368.        configContents = (configContents | 0x80);
369.        writeReg(CONFIG, configContents);
370.        HAL_Delay(50);
371. }
372.
373. /*  Bit 2 of the CONFIG register can be used to enable/disable
374.     the interrupt pin  */
375. void HDC2080::enableInterrupt(void)
376. {
377.        uint8_t configContents;
378.        configContents = readReg(CONFIG);
379.
380.        configContents = (configContents | 0x04);
381.        writeReg(CONFIG, configContents);
382. }
383.
384. /*  Bit 2 of the CONFIG register can be used to enable/disable
385.     the interrupt pin  */
386. void HDC2080::disableInterrupt(void)
387. {
388.        uint8_t configContents;
389.        configContents = readReg(CONFIG);
390.
391.        configContents = (configContents & 0xFB);
392.        writeReg(CONFIG, configContents);
393. }
394.
395.
396. /*  Bits 6-4  of the CONFIG register controls the measurement
```

```
397.     rate  */
398. void HDC2080::setRate(int rate)
399. {
400.         uint8_t configContents;
401.         configContents = readReg(CONFIG);
402.
403.         switch(rate)
404.         {
405.                 case MANUAL:
406.                         configContents = (configContents & 0x8F);
407.                         break;
408.
409.                 case TWO_MINS:
410.                         configContents = (configContents & 0x9F);
411.                         configContents = (configContents | 0x10);
412.                         break;
413.
414.                 case ONE_MINS:
415.                         configContents = (configContents & 0xAF);
416.                         configContents = (configContents | 0x20);
417.                         break;
418.
419.                 case TEN_SECONDS:
420.                         configContents = (configContents & 0xBF);
421.                         configContents = (configContents | 0x30);
422.                         break;
423.
424.                 case FIVE_SECONDS:
425.                         configContents = (configContents & 0xCF);
426.                         configContents = (configContents | 0x40);
427.                         break;
428.
429.                 case ONE_HZ:
430.                         configContents = (configContents & 0xDF);
431.                         configContents = (configContents | 0x50);
432.                         break;
433.
434.                 case TWO_HZ:
435.                         configContents = (configContents & 0xEF);
436.                         configContents = (configContents | 0x60);
437.                         break;
438.
439.                 case FIVE_HZ:
440.                         configContents = (configContents | 0x70);
441.                         break;
442.
443.                 default:
444.                         configContents = (configContents & 0x8F);
445.         }
446.
447.         writeReg(CONFIG, configContents);
448. }
449.
450. /*  Bit 1 of the CONFIG register can be used to control the
451.     the interrupt pins polarity */
452. void HDC2080::setInterruptPolarity(int polarity)
453. {
```

```
454.        uint8_t configContents;
455.        configContents = readReg(CONFIG);
456.
457.        switch(polarity)
458.        {
459.                case ACTIVE_LOW:
460.                        configContents = (configContents & 0xFD);
461.                        break;
462.
463.                case ACTIVE_HIGH:
464.                        configContents = (configContents | 0x02);
465.                        break;
466.
467.                default:
468.                        configContents = (configContents & 0xFD);
469.        }
470.
471.        writeReg(CONFIG, configContents);
472. }
473.
474. /*  Bit 0 of the CONFIG register can be used to control the
475.     the interrupt pin's mode */
476. void HDC2080::setInterruptMode(int mode)
477. {
478.        uint8_t configContents;
479.        configContents = readReg(CONFIG);
480.
481.        switch(mode)
482.        {
483.                case LEVEL_MODE:
484.                        configContents = (configContents & 0xFE);
485.                        break;
486.
487.                case COMPARATOR_MODE:
488.                        configContents = (configContents | 0x01);
489.                        break;
490.
491.                default:
492.                        configContents = (configContents & 0xFE);
493.        }
494.
495.        writeReg(CONFIG, configContents);
496. }
497.
498.
499. uint8_t HDC2080::readInterruptStatus(void)
500. {
501.        uint8_t regContents;
502.        regContents = readReg(INTERRUPT_DRDY);
503.        return regContents;
504.
505. }
506.
507. //  Clears the maximum temperature register
508. void HDC2080::clearMaxTemp(void)
509. {
510.        writeReg(TEMP_MAX, 0x00);
```

```
511. }
512.
513. //  Clears the maximum humidity register
514. void HDC2080::clearMaxHumidity(void)
515. {
516.         writeReg(HUMID_MAX, 0x00);
517. }
518.
519. //  Reads the maximum temperature register
520. float HDC2080::readMaxTemp(void)
521. {
522.         uint8_t regContents;
523.
524.         regContents = readReg(TEMP_MAX);
525.
526.         return (float)regContents * 165/256 - 40;
527.
528. }
529.
530. //  Reads the maximum humidity register
531. float HDC2080::readMaxHumidity(void)
532. {
533.         uint8_t regContents;
534.
535.         regContents = readReg(HUMID_MAX);
536.
537.         return (float)regContents /256 * 100;
538.
539. }
540.
541.
542. // Enables the interrupt pin for comfort zone operation
543. void HDC2080::enableThresholdInterrupt(void)
544. {
545.
546.         uint8_t regContents;
547.         regContents = readReg(INTERRUPT_CONFIG);
548.
549.         regContents = (regContents | 0x78);
550.
551.         writeReg(INTERRUPT_CONFIG, regContents);
552. }
553.
554. // Disables the interrupt pin for comfort zone operation
555. void HDC2080::disableThresholdInterrupt(void)
556. {
557.         uint8_t regContents;
558.         regContents = readReg(INTERRUPT_CONFIG);
559.
560.         regContents = (regContents & 0x87);
561.
562.         writeReg(INTERRUPT_CONFIG, regContents);
563. }
564.
565. // enables the interrupt pin for DRDY operation
566. void HDC2080::enableDRDYInterrupt(void)
567. {
```

```
568.        uint8_t regContents;
569.        regContents = readReg(INTERRUPT_CONFIG);
570.
571.        regContents = (regContents | 0x80);
572.
573.        writeReg(INTERRUPT_CONFIG, regContents);
574. }
575.
576. // disables the interrupt pin for DRDY operation
577. void HDC2080::disableDRDYInterrupt(void)
578. {
579.        uint8_t regContents;
580.        regContents = readReg(INTERRUPT_CONFIG);
581.
582.        regContents = (regContents & 0x7F);
583.
584.        writeReg(INTERRUPT_CONFIG, regContents);
585. }
```

1.7. MS5637.h

```
1. /*
2.   This is a library written for the MS5637. Originally written by
   TEConnectivity
3.   with an MIT license. Library updated and brought to fit Arduino
   Library standards
4.   by Nathan Seidle @ SparkFun Electronics, April 13th, 2018
5.
6.   MEMs based barometric pressure sensors are quite common these days.
   The
7.   MS5637 shines by being the most sensitive barometric pressure
   sensor we have
8.   come across. It is capable of detecting the difference in 13cm of
   air! On top
9.   of that the MS5637 is low cost and easy to use to boot!
10.
11.    This library handles the initialization of the MS5637 and is able
   to
12.    query the sensor for different readings.
13.
14.    https://github.com/sparkfun/SparkFun_MS5637_Arduino_Library
15.
16.    Development environment specifics:
17.    Arduino IDE 1.8.5
18.
19.    MIT License
20.
21.    Copyright (c) 2016 TE Connectivity
22.
23.    Permission is hereby granted, free of charge, to any person
   obtaining a copy
24.    of this software and associated documentation files (the
   "Software"), to deal
```

```
25.    in the Software without restriction, including without limitation
   the rights
26.    to use, copy, modify, merge, publish, distribute, sublicense,
   and/or sell
27.    copies of the Software, and to permit persons to whom the
   Software is
28.    furnished to do so, subject to the following conditions:
29.
30.    The above copyright notice and this permission notice shall be
   included in all
31.    copies or substantial portions of the Software.
32.
33.    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
   EXPRESS OR
34.    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   MERCHANTABILITY,
35.    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
   SHALL THE
36.    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
   OTHER
37.    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
   ARISING FROM,
38.    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
   DEALINGS IN THE
39.    SOFTWARE.
40.
41.    SparkFun labored with love to create this code. Feel like
   supporting open
42.    source hardware? Buy a board from SparkFun!
43.    https://www.sparkfun.com/products/14688
44.  */
45.
46.  #ifndef SPARKFUN_MS5637_ARDUINO_LIBRARY_H
47.  #define SPARKFUN_MS5637_ARDUINO_LIBRARY_H
48.
49.  #include "stm32l0xx_hal.h"
50.  #include "stm32l0xx_hal_i2c.h"
51.  #include "util_console.h"
52.
53.  #define MS5637_COEFFICIENT_COUNT 7
54.
55.  #define MS5637_CONVERSION_TIME_OSR_256 0x00
56.  #define MS5637_CONVERSION_TIME_OSR_512 0x02
57.  #define MS5637_CONVERSION_TIME_OSR_1024 0x04
58.  #define MS5637_CONVERSION_TIME_OSR_2048 0x06
59.  #define MS5637_CONVERSION_TIME_OSR_4096 0x08
60.  #define MS5637_CONVERSION_TIME_OSR_8192 0x0A
61.
62.  #define MS5637_addr 0x76
63.
64.
65.  // Functions
66.  class MS5637 {
67.
68.    public:
69.
70.      void begin(I2C_HandleTypeDef &i2cHandler, uint8_t i2caddr);
```

```
71.      void reset(void);
72.      void setResolution(uint8_t OSR);
73.      uint32_t readPressure(void);
74.
75.    private:
76.      void writeReg(uint8_t reg, uint8_t data);
77.                  uint16_t readReg16(uint8_t reg);
78.                  uint32_t readReg24(uint8_t reg);
79.
80.      I2C_HandleTypeDef _i2cHandler; //The generic connection to
   user's chosen I2C hardware
81.                  uint8_t _i2caddr;
82.                  uint8_t OSR_value;
83.      float globalPressure;
84.      float globalTemperature;
85.
86.  };
87.  #endif
```

1.8. MS5637.cpp

```
1. #include "MS5637.h"
2.
3.
4. // MS5637 device commands
5. #define MS5637_RESET  0x1E
6. #define MS5637_START_PRESSURE_ADC_CONVERSION 0x40
7. #define MS5637_START_TEMPERATURE_ADC_CONVERSION 0x50
8. #define MS5637_READ_ADC 0x00
9.
10.  #define MS5637_CONVERSION_OSR_MASK 0x0F
11.
12.  // MS5637 commands
13.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_0 0xA0
14.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_1 0xA2
15.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_2 0xA4
16.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_3 0xA6
17.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_4 0xA8
18.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_5 0xAA
19.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_6 0xAC
20.  #define MS5637_PROM_ADDRESS_READ_ADDRESS_7 0xAE
21.
22.  // Coefficients indexes for temperature and pressure computation
23.  #define MS5637_CRC_INDEX 0
24.  #define MS5637_PRESSURE_SENSITIVITY_INDEX 1
25.  #define MS5637_PRESSURE_OFFSET_INDEX 2
26.  #define MS5637_TEMP_COEFF_OF_PRESSURE_SENSITIVITY_INDEX 3
27.  #define MS5637_TEMP_COEFF_OF_PRESSURE_OFFSET_INDEX 4
28.  #define MS5637_REFERENCE_TEMPERATURE_INDEX 5
29.  #define MS5637_TEMP_COEFF_OF_TEMPERATURE_INDEX 6
30.
31.
32.  /**
```

```
33.   * \brief Perform initial configuration. Has to be called once.
34.   */
35.  void MS5637::begin(I2C_HandleTypeDef &i2cHandler, uint8_t i2caddr)
   {
36.
37.    _i2cHandler = i2cHandler; //Grab which port the user wants us to
   use
38.          _i2caddr = i2caddr;
39.
40.          reset();
41.
42.          /* Hacer funcion para leer los 8 registros y asignarselos a
   las contantes
43.          uint32_t PROM_variables[2];
44.          HAL_I2C_Mem_Read(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, PROM_variables, 2, 100);
45.          */
46.
47.    //Set resolution to the highest level (17 ms per reading)
48.          setResolution(MS5637_CONVERSION_TIME_OSR_8192);
49.
50.  }
51.
52.
53.  void MS5637::writeReg(uint8_t reg, uint8_t data) {
54.    HAL_I2C_Mem_Write(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, &data, 1, 100);
55.  }
56.
57.  uint16_t MS5637::readReg16(uint8_t reg){
58.          uint8_t buffer[2];
59.          uint16_t return_value;
60.
61.          HAL_I2C_Mem_Read(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, buffer, 2, 100);
62.
63.          return_value = (buffer[0] << 8) | buffer[1];
64.          return return_value;
65.  }
66.
67.  uint32_t MS5637::readReg24(uint8_t reg){
68.          uint8_t buffer[3];
69.          uint32_t return_value;
70.
71.          HAL_I2C_Mem_Read(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, buffer, 3, 100);
72.
73.          return_value = buffer[0] << 16 | buffer[1] << 8 |
   buffer[2];
74.          return return_value;
75.  }
76.
77.  void MS5637::setResolution(uint8_t OSR) {
78.  OSR_value = OSR;
79.  }
80.
81.  void MS5637::reset(void) {
```

```
82.  uint8_t RST = 0x1E;
83.          HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1, &RST,
   I2C_MEMADD_SIZE_8BIT, 100);
84.  }
85.
86.  uint32_t MS5637::readPressure(void){
87.          uint32_t pressure;
88.          uint8_t buffer[3];
89.
90.          uint8_t readD1 = 0x4A, readADC = 0x00;
91.
92.          HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1,
   &readD1, I2C_MEMADD_SIZE_8BIT, 100);
93.          HAL_Delay(500);
94.
95.          HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1,
   &readADC, I2C_MEMADD_SIZE_8BIT, 100);
96.          HAL_I2C_Master_Receive(&_i2cHandler, _i2caddr, buffer,
   sizeof(buffer), 100);
97.
98.          pressure = buffer[0] << 16 | buffer[1] << 8 | buffer[2];
99.
100.         return pressure;
101. }
```

1.9. TSL2591.h

```
1. /*************************************************************************
   ******/
2. /*!
3.     @file     Adafruit_TSL2591.h
4.     @author   KT0WN (adafruit.com)
5.
6.     This is a library for the Adafruit TSL2591 breakout board
7.     This library works with the Adafruit TSL2591 breakout
8.     ----> https://www.adafruit.com/products/1980
9.
10.      Check out the links above for our tutorials and wiring diagrams
11.      These chips use I2C to communicate
12.
13.      Adafruit invests time and resources providing this open source
   code,
14.      please support Adafruit and open-source hardware by purchasing
15.      products from Adafruit!
16.  */
17.  /*************************************************************************
   ********/
18.
19.  #include "stm32l0xx_hal.h"
20.
21.  #define TSL2591_VISIBLE (2) // channel 0 - channel 1
22.  #define TSL2591_INFRARED (1) // channel 1
23.  #define TSL2591_FULLSPECTRUM (0) // channel 0
```

```
24.
25.  #define TSL2591_DEFAULT_ADDR 0x29
26.
27.  #define TSL2591_READBIT (0x01)
28.
29.  #define TSL2591_COMMAND_BIT (0xA0) // bits 7 and 5 for 'command
   normal'
30.  #define TSL2591_CLEAR_BIT (0x40) // Clears any pending interrupt
   (write 1 to clear)
31.  #define TSL2591_WORD_BIT (0x20) // 1 = read/write word (rather than
   byte)
32.  #define TSL2591_BLOCK_BIT (0x10) // 1 = using block read/write
33.
34.  #define TSL2591_ENABLE_POWERON (0x01)
35.  #define TSL2591_ENABLE_POWEROFF (0x00)
36.  #define TSL2591_ENABLE_AEN (0x02)
37.  #define TSL2591_ENABLE_AIEN (0x10)
38.  #define TSL2591_ENABLE_SAI (0x40)
39.  #define TSL2591_ENABLE_NPIEN (0x80)
40.
41.  #define TSL2591_LUX_DF (408.0F)
42.  #define TSL2591_LUX_COEFB (1.64F) // CH0 coefficient
43.  #define TSL2591_LUX_COEFC (0.59F) // CH1 coefficient A
44.  #define TSL2591_LUX_COEFD (0.86F) // CH2 coefficient B
45.
46.
47.  #define TSL2591_REGISTER_ENABLE 0x00
48.  #define TSL2591_REGISTER_CONTROL 0x01
49.  #define TSL2591_REGISTER_THRESHHOLDL_LOW 0x04
50.  #define TSL2591_REGISTER_THRESHHOLDL_HIGH 0x05
51.  #define TSL2591_REGISTER_THRESHHOLDH_LOW 0x06
52.  #define TSL2591_REGISTER_THRESHHOLDH_HIGH 0x07
53.  #define TSL2591_REGISTER_PID 0x11
54.  #define TSL2591_REGISTER_ID 0x12
55.  #define TSL2591_REGISTER_STATUS 0x13
56.  #define TSL2591_REGISTER_CHAN0_LOW 0x14
57.  #define TSL2591_REGISTER_CHAN0_HIGH 0x15
58.  #define TSL2591_REGISTER_CHAN1_LOW 0x16
59.  #define TSL2591_REGISTER_CHAN1_HIG  0x17
60.
61.
62.  typedef enum
63.  {
64.          TSL2591_INTEGRATIONTIME_100MS = 0x00,
65.          TSL2591_INTEGRATIONTIME_200MS = 0x01,
66.          TSL2591_INTEGRATIONTIME_300MS = 0x02,
67.          TSL2591_INTEGRATIONTIME_400MS = 0x03,
68.          TSL2591_INTEGRATIONTIME_500MS = 0x04,
69.          TSL2591_INTEGRATIONTIME_600MS = 0x05,
70.  }
71.  tsl2591IntegrationTime_t;
72.
73.  typedef enum
74.  {
75.          TSL2591_GAIN_LOW = 0x00, // low gain (1x)
76.          TSL2591_GAIN_MED = 0x10, // medium gain (25x)
77.          TSL2591_GAIN_HIGH = 0x20, // medium gain (428x)
```

```
78.         TSL2591_GAIN_MAX = 0x30, // max gain (9876x)
79.  }
80.  tsl2591Gain_t;
81.
82.
83.  class TSL2591 {
84.         public:
85.
86.                 //Funciones de usuario
87.                 void begin( I2C_HandleTypeDef &i2cHandler, uint8_t
   i2caddr = TSL2591_DEFAULT_ADDR );
88.
89.                 float calculateLux(uint16_t ch0, uint16_t ch1);
90.                 void setGain(tsl2591Gain_t gain);
91.                 void setTiming(tsl2591IntegrationTime_t
   integration);
92.                 uint16_t getLuminosity(uint8_t channel);
93.                 uint32_t getFullLuminosity();
94.
95.                 //added functions
96.                 uint32_t getLux ();
97.
98.                 tsl2591IntegrationTime_t getTiming();
99.                 tsl2591Gain_t getGain();
100.
101.                uint8_t checkId ();
102.                uint8_t checkStatus ();
103.                uint8_t checkRegister ( uint8_t requested_register
   );
104.
105.
106.
107.        private:
108.                //Funciones no acccesibles al usuario
109.                void enable(void);
110.                void disable(void);
111.                void write8(uint8_t r, uint8_t v);
112.                                void write8(uint8_t r);
113.                uint16_t read16(uint8_t reg);
114.                uint8_t read8(uint8_t reg);
115.
116.                //Private variables
117.                uint8_t _i2caddr;
118.                I2C_HandleTypeDef _i2cHandler;
119.
120.                tsl2591IntegrationTime_t _integration;
121.                tsl2591Gain_t _gain;
122.                bool _initialized;
123. };
```

1.10. TSL2591.cpp

```
/**************************************************************************/
/*!
    @file     Adafruit_TSL2591.cpp
    @author   KT0WN (adafruit.com)

    This is a library for the Adafruit TSL2591 breakout board
    This library works with the Adafruit TSL2591 breakout
    ----> https://www.adafruit.com/products/1980

      Check out the links above for our tutorials and wiring diagrams
      These chips use I2C to communicate

      Adafruit invests time and resources providing this open source code,
      please support Adafruit and open-source hardware by purchasing
      products from Adafruit!

      @section LICENSE

      Software License Agreement (BSD License)

      Copyright (c) 2014 Adafruit Industries
      All rights reserved.

      Redistribution and use in source and binary forms, with or without
      modification, are permitted provided that the following conditions are met:
      1. Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
      2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
      3. Neither the name of the copyright holders nor the
      names of its contributors may be used to endorse or promote products
      derived from this software without specific prior written permission.

      THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS'' AND ANY
      EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
      WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
      DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY
      DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
```

```
40.      (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
   OR SERVICES;
41.      LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
   HOWEVER CAUSED AND
42.      ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
   LIABILITY, OR TORT
43.      (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
   THE USE OF
44.     THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
   DAMAGE.
45.  */
46.  /**************************************************************************
   ********/
47.
48.  #include "tsl2591.h"
49.  #include "util_console.h"
50.
51.  void TSL2591::begin( I2C_HandleTypeDef &i2cHandler, uint8_t
   i2caddr) {
52.          _i2caddr = i2caddr;
53.          i2cHandler = i2cHandler;
54.
55.          //Set default configuration options
56.          _integration = TSL2591_INTEGRATIONTIME_100MS;
57.          _gain = TSL2591_GAIN_MED;
58.
59.          enable();
60.
61.          // Set default integration time and gain
62.          setTiming(_integration);
63.          setGain(_gain);
64.
65.
66.          disable();
67.  }
68.
69.  void TSL2591::enable(void)
70.  {
71.          // Enable the device by setting the control bit to 0x01
72.          write8( (TSL2591_COMMAND_BIT | TSL2591_REGISTER_ENABLE) ,
   (TSL2591_ENABLE_POWERON | TSL2591_ENABLE_AEN) );
73.  }
74.
75.  void TSL2591::disable(void)
76.  {
77.          // Disable the device by setting the control bit to 0x00
78.          write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_ENABLE,
   TSL2591_ENABLE_POWEROFF);
79.  }
80.
81.  void TSL2591::setGain(tsl2591Gain_t gain)
82.  {
83.          enable();
84.          _gain = gain;
85.          write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CONTROL,
   _integration | _gain);
86.          disable();
```

```
87.  }
88.
89.  tsl2591Gain_t TSL2591::getGain()
90.  {
91.          return _gain;
92.  }
93.
94.  void TSL2591::setTiming(tsl2591IntegrationTime_t integration)
95.  {
96.          enable();
97.          _integration = integration;
98.          write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CONTROL,
   _integration | _gain);
99.          disable();
100. }
101.
102. tsl2591IntegrationTime_t TSL2591::getTiming()
103. {
104.         return _integration;
105. }
106.
107. float TSL2591::calculateLux(uint16_t ch0, uint16_t ch1)
108. {
109.         uint16_t atime, again;
110.         float cpl, lux1, lux2, lux;
111.         uint32_t chan0, chan1;
112.
113.         // Check for overflow conditions first
114.         if ((ch0 == 0xFFFF) | (ch1 == 0xFFFF))
115.         {
116.                 // Signal an overflow
117.                 return -1;
118.         }
119.
120.         // Note: This algorithm is based on preliminary
   coefficients
121.         // provided by AMS and may need to be updated in the future
122.
123.         switch (_integration)
124.         {
125.                 case TSL2591_INTEGRATIONTIME_100MS:
126.                         atime = 100.0F;
127.                         break;
128.                 case TSL2591_INTEGRATIONTIME_200MS:
129.                         atime = 200.0F;
130.                         break;
131.                 case TSL2591_INTEGRATIONTIME_300MS:
132.                         atime = 300.0F;
133.                         break;
134.                 case TSL2591_INTEGRATIONTIME_400MS:
135.                         atime = 400.0F;
136.                         break;
137.                 case TSL2591_INTEGRATIONTIME_500MS:
138.                         atime = 500.0F;
139.                         break;
140.                 case TSL2591_INTEGRATIONTIME_600MS:
141.                         atime = 600.0F;
```

```
142.                        break;
143.                default: // 100ms
144.                        atime = 100.0F;
145.                        break;
146.        }
147.
148.        switch (_gain)
149.        {
150.                case TSL2591_GAIN_LOW:
151.                        again = 1.0F;
152.                        break;
153.                case TSL2591_GAIN_MED:
154.                        again = 25.0F;
155.                        break;
156.                case TSL2591_GAIN_HIGH:
157.                        again = 428.0F;
158.                        break;
159.                case TSL2591_GAIN_MAX:
160.                        again = 9876.0F;
161.                        break;
162.                default:
163.                        again = 1.0F;
164.                        break;
165.        }
166.
167.        // cpl = (ATIME * AGAIN) / DF
168.        cpl = (atime * again) / TSL2591_LUX_DF;
169.
170.   // Original lux calculation (for reference sake)
171.        //lux1 = ((float)ch0 - (TSL2591_LUX_COEFB * (float)ch1)) /
   cpl;
172.        //lux2 = ((TSL2591_LUX_COEFC * (float)ch0) -
   (TSL2591_LUX_COEFD * (float)ch1)) / cpl;
173.   // (float)ch1 ) ) / cpl; lux = lux1 > lux2 ? lux1 : lux2;
174.
175.
176.   // Alternate lux calculation 1
177.   // See:
   https://github.com/adafruit/Adafruit_TSL2591_Library/issues/14
178.        lux = (((float)ch0 - (float)ch1)) * (1.0F - ((float)ch1 /
   (float)ch0)) / cpl;
179.
180.   // Alternate lux calculation 2
181.   // lux = ( (float)ch0 - ( 1.7F * (float)ch1 ) ) / cpl;
182.
183.        // Signal I2C had no errors
184.        return lux;
185. }
186.
187. uint32_t TSL2591::getFullLuminosity(void)
188. {
189.        enable();
190.
191.        // Wait x ms for ADC to complete
192.        for (uint8_t d = 0; d <= _integration; d++)
193.        {
194.                HAL_Delay(200);
```

```
195.         }
196.
197.    // CHAN0 must be read before CHAN1
198.    // See: https://forums.adafruit.com/viewtopic.php?f=19&t=124176
199.         uint32_t return_value;
200.    uint16_t x,y;
201.
202.         y = read16(TSL2591_COMMAND_BIT |
    TSL2591_REGISTER_CHAN0_LOW);
203.    x = read16(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CHAN1_LOW);
204.
205.    return_value = (x << 16) | y;
206.
207.         disable();
208.
209.         return return_value;
210. }
211.
212. uint16_t TSL2591::getLuminosity(uint8_t channel)
213. {
214.         uint32_t x = getFullLuminosity();
215.
216.         if (channel == TSL2591_FULLSPECTRUM)
217.         {
218.                 // Reads two byte value from channel 0 (visible +
    infrared)
219.                 return (x & 0xFFFF);
220.         }       else if (channel == TSL2591_INFRARED)
221.         {
222.                 // Reads two byte value from channel 1 (infrared)
223.                 return (x >> 16);
224.         }       else if (channel == TSL2591_VISIBLE)
225.         {
226.                 // Reads all and subtracts out just the visible!
227.                 return ((x & 0xFFFF) - (x >> 16));
228.         }
229.
230.         // unknown channel!
231.         return 0;
232. }
233.
234. uint8_t TSL2591::checkId( )
235. {
236.         //enable();
237.         //Read ID to check if communicates and everything OK. Needs
    to return 0x50 if OK
238.         uint8_t id = read8(TSL2591_COMMAND_BIT |
    TSL2591_REGISTER_ID);
239.         //disable();
240.         return id;
241. }
242.
243. uint8_t TSL2591::checkStatus( )
244. {
245.         //enable();
246.         //Read ID to check if communicates and everything OK. Needs
    to return 0x50 if OK
```

```
247.         uint8_t status = read8(TSL2591_COMMAND_BIT |
   TSL2591_REGISTER_STATUS);
248.         //disable();
249.         return status;
250. }
251.
252. uint8_t TSL2591::checkRegister( uint8_t requested_register)
253. {
254.         //Read a register selected by the user
255.         return read8( requested_register );
256. }
257.
258. uint32_t TSL2591::getLux()
259. {
260.         uint32_t lum = getFullLuminosity();
261.         uint16_t ir, full;
262.
263.         ir = lum >> 16;
264.         full = lum & 0xFFFF;
265.
266.         return (uint32_t) calculateLux(full, ir);
267. }
268.
269. uint8_t TSL2591::read8(uint8_t reg)
270. {
271.         uint8_t buffer = reg;
272.         uint8_t return_value;
273.
274.         HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1,
   &buffer, 1, 100);
275.         HAL_I2C_Master_Receive(&_i2cHandler, _i2caddr << 1,
   &return_value, 1, 100);
276.
277.         return return_value;
278. }
279.
280. uint16_t TSL2591::read16(uint8_t reg)
281. {
282.         uint8_t buf[2];
283.         uint16_t return_value;
284.
285.         HAL_I2C_Mem_Read(&_i2cHandler, _i2caddr << 1, reg,
   sizeof(reg), buf, 2, 100);
286.
287.         return_value = (buf[0] << 8) | buf[1];
288.
289.         return return_value;
290. }
291.
292. void TSL2591::write8(uint8_t reg, uint8_t value)
293. {
294.         HAL_I2C_Mem_Write(&_i2cHandler, _i2caddr << 1, reg,
   I2C_MEMADD_SIZE_8BIT, &value, 1, 100);
295. }
296.
297. void TSL2591::write8(uint8_t reg)
298. {
```

```
299.         HAL_I2C_Master_Transmit(&_i2cHandler, _i2caddr << 1, ®, 1,
   100);
300. }
```

2. Configuración de LoRaWAN

2.1. commissioning.h

```
1. /*!
2.  * \file      Commissioning.h
3.  *
4.  * \brief     End device commissioning parameters
5.  *
6.  * \copyright Revised BSD License, see section \ref LICENSE.
7.  *
8.  * \code
9.  *                ______                              _
10.  *               / _____)             _              | |
11.  *              ( (____  _____ ____ _| |_ _____  ____| |__
12.  *               \____ \| ___ |    (_   _) ___ |/ ___)  _ \
13.  *               _____) ) ____| | | || |_| ____( (___| | | |
14.  *              (______/|_____)_|_|_| \__)_____)\____)_| |_|
15.  *              (C)2013-2017 Semtech
16.  *
17.  * \endcode
18.  *
19.  * \author    Miguel Luis ( Semtech )
20.  *
21.  * \author    Gregory Cristian ( Semtech )
22.  */
23.  /**
24.
   ******************************************************************************
   *********
25.    * @file    commissioning.h
26.    * @author  MCD Application Team
27.    * @brief   End device commissioning parameters
28.
   ******************************************************************************
   *********
29.    * @attention
30.    *
31.    * <h2><center>© Copyright (c) 2018 STMicroelectronics.
32.    * All rights reserved.</center></h2>
33.    *
34.    * This software component is licensed by ST under Ultimate
   Liberty license
35.    * SLA0044, the "License"; You may not use this file except in
   compliance with
36.    * the License. You may obtain a copy of the License at:
37.    *                              www.st.com/SLA0044
38.    *
```

```
39. ******************************************************************************
    *********
40.    */
41.
42.  /* Define to prevent recursive inclusion -------------------------
    -----------*/
43.  #ifndef __LORA_COMMISSIONING_H__
44.  #define __LORA_COMMISSIONING_H__
45.
46.  #ifdef __cplusplus
47.  extern "C" {
48.  #endif
49.  /*!
50. ******************************************************************************
    *********
51.   ********************************** WARNING
    **********************************
52. ******************************************************************************
    *********
53.    The crypto-element implementation supports both 1.0.x and 1.1.x
    LoRaWAN
54.    versions of the specification.
55.    Thus it has been decided to use the 1.1.x keys and EUI name
    definitions.
56.    The below table shows the names equivalence between versions:
57.                  +---------------------+--------------------------+
58.                  |       1.0.x         |          1.1.x           |
59.                  +=====================+==========================+
60.                  | LORAWAN_DEVICE_EUI  | LORAWAN_DEVICE_EUI       |
61.                  +---------------------+--------------------------+
62.                  | LORAWAN_APP_EUI     | LORAWAN_JOIN_EUI         |
63.                  +---------------------+--------------------------+
64.                  | LORAWAN_GEN_APP_KEY | LORAWAN_APP_KEY          |
65.                  +---------------------+--------------------------+
66.                  | LORAWAN_APP_KEY     | LORAWAN_NWK_KEY          |
67.                  +---------------------+--------------------------+
68.                  | LORAWAN_NWK_S_KEY   | LORAWAN_F_NWK_S_INT_KEY  |
69.                  +---------------------+--------------------------+
70.                  | LORAWAN_NWK_S_KEY   | LORAWAN_S_NWK_S_INT_KEY  |
71.                  +---------------------+--------------------------+
72.                  | LORAWAN_NWK_S_KEY   | LORAWAN_NWK_S_ENC_KEY    |
73.                  +---------------------+--------------------------+
74.                  | LORAWAN_APP_S_KEY   | LORAWAN_APP_S_KEY        |
75.                  +---------------------+--------------------------+
76. ******************************************************************************
    *********
77. ******************************************************************************
    *********
78. ******************************************************************************
    *********
79.   */
```

```
80.
81.  /*!
82.   * When set to 1 the application uses the Over-the-Air activation
   procedure
83.   * When set to 0 the application uses the Personalization
   activation procedure
84.   */
85.  #define OVER_THE_AIR_ACTIVATION                                 1
86.
87.  /*!
88.   * When using ABP activation the MAC layer must know in advance to
   which server
89.   * version it will be connected.
90.   */
91.  //#define ABP_ACTIVATION_LRWAN_VERSION_V10x
   0x01000300 // 1.0.3.0
92.
93.  //#define ABP_ACTIVATION_LRWAN_VERSION
   ABP_ACTIVATION_LRWAN_VERSION_V10x
94.
95.  /*!
96.   * Indicates if the end-device is to be connected to a private or
   public network
97.   */
98.  #define LORAWAN_PUBLIC_NETWORK                                  true
99.
100. /*!
101.  * When set to 1 DevEui is LORAWAN_DEVICE_EUI
102.  * When set to 0 DevEui is automatically generated by calling
103.  *         BoardGetUniqueId function
104.  */
105. #define STATIC_DEVICE_EUI                                       1
106.
107. /*!
108.  * \remark see STATIC_DEVICE_EUI comments
109.  */
110. #define LORAWAN_DEVICE_EUI                                      { 0x00,
   0x01, 0xED, 0xA2, 0x27, 0xCA, 0x01, 0xD4 }
111.
112. /*!
113.  * App/Join server IEEE EUI (big endian)
114.  */
115. #define LORAWAN_JOIN_EUI                                        { 0x70,
   0xB3, 0xD5, 0x7E, 0xD0, 0x02, 0xFE, 0x67 }
116.
117. /*!
118.  * Application root key
119.  * WARNING: NOT USED FOR 1.0.x DEVICES
120.  */
121. #define LORAWAN_APP_KEY                                         { 0x13,
   0x04, 0x55, 0x96, 0x8D, 0x94, 0x24, 0x35, 0xCD, 0x95, 0x42, 0x82,
   0x7B, 0x8B, 0x5E, 0x86 }
122.
123. /*!
124.  * Application root key - Used to derive Multicast keys on 1.0.x
   devices.
125.  * WARNING: USED only FOR 1.0.x DEVICES
```

```
126.  */
127. #define LORAWAN_GEN_APP_KEY                                 { 0x13,
   0x04, 0x55, 0x96, 0x8D, 0x94, 0x24, 0x35, 0xCD, 0x95, 0x42, 0x82,
   0x7B, 0x8B, 0x5E, 0x86 }
128.
129. /*!
130.  * Network root key
131.  * WARNING: FOR 1.0.x DEVICES IT IS THE \ref LORAWAN_APP_KEY
132.  */
133. #define LORAWAN_NWK_KEY                                     { 0x13,
   0x04, 0x55, 0x96, 0x8D, 0x94, 0x24, 0x35, 0xCD, 0x95, 0x42, 0x82,
   0x7B, 0x8B, 0x5E, 0x86 }
134.
135. /*!
136.  * Current network ID
137.  */
138. #define LORAWAN_NETWORK_ID                                  (
   uint32_t )0
139.
140. /*!
141.  * When set to 1 DevAdd is LORAWAN_DEVICE_ADDRESS
142.  * When set to 0 DevAdd is automatically generated using
143.  *         a pseudo random generator seeded with a value derived
   from
144.  *         BoardUniqueId value
145.  */
146. #define STATIC_DEVICE_ADDRESS                               0
147.
148. /*!
149.  * Device address on the network (big endian)
150.  *
151.  * \remark see STATIC_DEVICE_ADDRESS comments
152.  */
153. #define LORAWAN_DEVICE_ADDRESS                              (
   uint32_t ){ 0x26, 0x01, 0x2F, 0xB2 }
154.
155. /*!
156.  * Forwarding Network session integrity key
157.  * WARNING: NWK_S_KEY FOR 1.0.x DEVICES
158.  */
159. //#define LORAWAN_F_NWK_S_INT_KEY                             {
   0x07, 0x81, 0x7C, 0xEA, 0x03, 0x39, 0xB3, 0x42, 0xC8, 0x67, 0x6A,
   0x03, 0x56, 0xA1, 0x37, 0x00 }
160.
161. /*!
162.  * Serving Network session integrity key
163.  * WARNING: NOT USED FOR 1.0.x DEVICES. MUST BE THE SAME AS \ref
   LORAWAN_F_NWK_S_INT_KEY
164.  */
165. //#define LORAWAN_S_NWK_S_INT_KEY                             {
   0x07, 0x81, 0x7C, 0xEA, 0x03, 0x39, 0xB3, 0x42, 0xC8, 0x67, 0x6A,
   0x03, 0x56, 0xA1, 0x37, 0x00 }
166.
167. /*!
168.  * Network session encryption key
169.  * WARNING: NOT USED FOR 1.0.x DEVICES. MUST BE THE SAME AS \ref
   LORAWAN_F_NWK_S_INT_KEY
```

```
170.  */
171. //#define LORAWAN_NWK_S_ENC_KEY                                   {
   0x07, 0x81, 0x7C, 0xEA, 0x03, 0x39, 0xB3, 0x42, 0xC8, 0x67, 0x6A,
   0x03, 0x56, 0xA1, 0x37, 0x00 }
172.
173. /*!
174.  * Application session key
175.  */
176. //#define LORAWAN_APP_S_KEY                                       {
   0x8B, 0x58, 0x4F, 0x84, 0x17, 0x66, 0x66, 0x2F, 0x96, 0xC0, 0x48,
   0x1E, 0x02, 0xE1, 0x04, 0x09 }
177.
178. #ifdef __cplusplus
179. }
180. #endif
181.
182. #endif /* __LORA_COMMISSIONING_H__ */
```

3. Aplicación principal

3.1. main.c

```
1. /**
2.
   ******************************************************************************
   *********
3.   * @file    main.c
4.   * @author  MCD Application Team
5.   * @brief   this is the main!
6.
   ******************************************************************************
   *********
7.   * @attention
8.   *
9.   * <h2><center>© Copyright (c) 2018 STMicroelectronics.
10.    * All rights reserved.</center></h2>
11.    *
12.    * This software component is licensed by ST under Ultimate
   Liberty license
13.    * SLA0044, the "License"; You may not use this file except in
   compliance with
14.    * the License. You may obtain a copy of the License at:
15.    *                              www.st.com/SLA0044
16.    *
17.
   ******************************************************************************
   *********
18.    */
19.
20.  /* Includes ------------------------------------------------------
   -----------*/
21.  #include "stm32l0xx_hal.h"
22.  #include "hw.h"
```

```
23.  #include "low_power_manager.h"
24.  #include "lora.h"
25.  //#include "bsp.h"
26.  #include "timeServer.h"
27.  #include "vcom.h"
28.  #include "version.h"
29.  #include "electrovalve.h"
30.  #include "i2c.h"
31.  #include "HDC2080.h"
32.  #include "TSL2591.h"
33.  #include "MS5637.h"
34.  #include "Adafruit_SoilMoisture.h"
35.
36.
37.  /* Private typedef -----------------------------------------------
     -----------*/
38.  /* Private define ------------------------------------------------
     -----------*/
39.
40.  #define LORAWAN_MAX_BAT   254
41.
42.  /*!
43.   * CAYENNE_LPP is myDevices Application server.
44.   */
45.  //#define CAYENNE_LPP
46.  #define LPP_DATATYPE_DIGITAL_INPUT  0x00
47.  #define LPP_DATATYPE_DIGITAL_OUTPUT 0x01
48.  #define LPP_DATATYPE_ANALOG_INPUT 0x02
49.
50.  #define LPP_DATATYPE_HUMIDITY       0x68
51.  #define LPP_DATATYPE_TEMPERATURE    0x67
52.  #define LPP_DATATYPE_BAROMETER      0x73
53.  #define LPP_DATATYPE_ILLUMINANCE    0x65
54.  #define LPP_APP_PORT 99
55.
56.
57.  /*!
58.   * Defines the application data transmission duty cycle. 30 min,
     value in [ms].
59.   */
60.  #define APP_TX_DUTYCYCLE                            1800000
61.  /*!
62.   * LoRaWAN Adaptive Data Rate
63.   * @note Please note that when ADR is enabled the end-device should
     be static
64.   */
65.  #define LORAWAN_ADR_STATE LORAWAN_ADR_ON
66.  /*!
67.   * LoRaWAN Default data Rate Data Rate
68.   * @note Please note that LORAWAN_DEFAULT_DATA_RATE is used only
     when ADR is disabled
69.   */
70.  #define LORAWAN_DEFAULT_DATA_RATE DR_0
71.  /*!
72.   * LoRaWAN application port
73.   * @note do not use 224. It is reserved for certification
74.   */
```

```
75.  #define LORAWAN_APP_PORT                            2
76.  /*!
77.   * LoRaWAN default endNode class port
78.   */
79.  #define LORAWAN_DEFAULT_CLASS                       CLASS_A
80.  /*!
81.   * LoRaWAN default confirm state
82.   */
83.  #define LORAWAN_DEFAULT_CONFIRM_MSG_STATE
     LORAWAN_UNCONFIRMED_MSG
84.  /*!
85.   * User application data buffer size
86.   */
87.  #define LORAWAN_APP_DATA_BUFF_SIZE                          64
88.  /*!
89.   * User application data
90.   */
91.  static uint8_t AppDataBuff[LORAWAN_APP_DATA_BUFF_SIZE];
92.
93.  /*!
94.   * User application data structure
95.   */
96.  //static lora_AppData_t AppData={ AppDataBuff,  0 ,0 };
97.  lora_AppData_t AppData = { AppDataBuff,  0, 0 };
98.
99.  /* Private macro -------------------------------------------------
     -----------*/
100. /* Private function prototypes -----------------------------------
     -----------*/
101.
102. /* call back when LoRa endNode has received a frame*/
103. static void LORA_RxData(lora_AppData_t *AppData);
104.
105. /* call back when LoRa endNode has just joined*/
106. static void LORA_HasJoined(void);
107.
108. /* call back when LoRa endNode has just switch the class*/
109. static void LORA_ConfirmClass(DeviceClass_t Class);
110.
111. /* call back when server needs endNode to send a frame*/
112. static void LORA_TxNeeded(void);
113.
114. /* callback to get the battery level in % of full charge (254 full
     charge, 0 no charge)*/
115. static uint8_t LORA_GetBatteryLevel(void);
116.
117. /* LoRa endNode send request*/
118. static void Send(void *context);
119.
120. /* start the tx process*/
121. static void LoraStartTx(TxEventType_t EventType);
122.
123. /* tx timer callback function*/
124. static void OnTxTimerEvent(void *context);
125.
126. /* tx timer callback function*/
127. static void LoraMacProcessNotify(void);
```

```
128.
129. /* Private variables ---------------------------------------------
   -----------*/
130. /* load Main call backs structure*/
131. static LoRaMainCallback_t LoRaMainCallbacks = {
   LORA_GetBatteryLevel,
132.
   HW_GetTemperatureLevel,
133.                                                HW_GetUniqueId,
134.                                                HW_GetRandomSeed,
135.                                                LORA_RxData,
136.                                                LORA_HasJoined,
137.                                                LORA_ConfirmClass,
138.                                                LORA_TxNeeded,
139.
   LoraMacProcessNotify
140.                                              };
141. LoraFlagStatus LoraMacProcessRequest = LORA_RESET;
142. LoraFlagStatus AppProcessRequest = LORA_RESET;
143. /*!
144.  * Specifies the state of the application LED
145.  */
146. static uint8_t AppLedStateOn = RESET;
147.
148. static TimerEvent_t TxTimer;
149.
150. #ifdef USE_B_L072Z_LRWAN1
151. /*!
152.  * Timer to handle the application Tx Led to toggle
153.  */
154. static TimerEvent_t TxLedTimer;
155. static void OnTimerLedEvent(void *context);
156. #endif
157. /* !
158.  *Initialises the Lora Parameters
159.  */
160. static  LoRaParam_t LoRaParamInit = {LORAWAN_ADR_STATE,
161.                                      LORAWAN_DEFAULT_DATA_RATE,
162.                                      LORAWAN_PUBLIC_NETWORK
163.                                     };
164.
165. /* Private functions ---------------------------------------------
   -----------*/
166.
167.
168. HDC2080 hdc2080;
169. TSL2591 tsl2591;
170. MS5637 ms5637;
171. Adafruit_SoilMoisture soilMoist;


172.
173. bool ElectroValveState = 0;


174. /**
175.   * @brief  Main program
```

```
176.   * @param  None
177.   * @retval None
178.   */
179. int main(void)
180. {
181.   /* STM32 HAL library initialization*/
182.   HAL_Init();
183.
184.   /* Configure the system clock*/
185.   SystemClock_Config();
186.
187.   /* Configure the debug mode*/
188.   DBG_Init();
189.
190.   /* Configure the hardware*/
191.   HW_Init();
192.
193.   /* USER CODE BEGIN 1 */
194.         MX_I2C1_Init();
195.
196.         valve_Init();
197.
198.         hdc2080.begin(hi2c1, HDC2080_addr);
199.         tsl2591.begin(hi2c1, TSL2591_DEFAULT_ADDR);
200.         ms5637.begin(hi2c1, MS5637_addr);
201.         soilMoist.begin(hi2c1, SoilMoist_ADDRESS);
202.
203.   /* USER CODE END 1 */
204.
205.   /*Disbale Stand-by mode*/
206.   LPM_SetOffMode(LPM_APPLI_Id, LPM_Disable);
207.
208.   PRINTF("APP_VERSION= %02X.%02X.%02X.%02X\r\n",
   (uint8_t)(__APP_VERSION >> 24), (uint8_t)(__APP_VERSION >> 16),
   (uint8_t)(__APP_VERSION >> 8), (uint8_t)__APP_VERSION);
209.   PRINTF("MAC_VERSION= %02X.%02X.%02X.%02X\r\n",
   (uint8_t)(__LORA_MAC_VERSION >> 24), (uint8_t)(__LORA_MAC_VERSION >>
   16), (uint8_t)(__LORA_MAC_VERSION >> 8),
   (uint8_t)__LORA_MAC_VERSION);
210.
211.   /* Configure the Lora Stack*/
212.   LORA_Init(&LoRaMainCallbacks, &LoRaParamInit);
213.
214.   LORA_Join();
215.
216.   LoraStartTx(TX_ON_TIMER) ;
217.
218.
219.
220.
221.   while (1)
222.   {
223.
224.     if (AppProcessRequest == LORA_SET)
225.     {
226.       //reset notification flag
227.       AppProcessRequest = LORA_RESET;
```

```
228.       //Send
229.       Send(NULL);
230.     }
231.     if (LoraMacProcessRequest == LORA_SET)
232.     {
233.       //reset notification flag
234.       LoraMacProcessRequest = LORA_RESET;
235.       LoRaMacProcess();
236.     }
237.     //If a flag is set at this point, mcu must not enter low power
   and must loop
238.     DISABLE_IRQ();
239.
240.     // if an interrupt has occurred after DISABLE_IRQ, it is kept
   pending
241.     // and cortex will not enter low power anyway
242.     if ((LoraMacProcessRequest != LORA_SET) && (AppProcessRequest
   != LORA_SET))
243.     {
244. #ifndef LOW_POWER_DISABLE
245.       LPM_EnterLowPower();
246. #endif
247.     }
248.
249.     ENABLE_IRQ();
250.
251.   }
252. }
253.
254.
255. void LoraMacProcessNotify(void)
256. {
257.   LoraMacProcessRequest = LORA_SET;
258. }
259.
260.
261. static void LORA_HasJoined(void)
262. {
263. #if( OVER_THE_AIR_ACTIVATION != 0 )
264.   PRINTF("JOINED\n\r");
265. #endif
266.   LORA_RequestClass(LORAWAN_DEFAULT_CLASS);
267. }
268.
269. static void Send(void *context)
270. {
271.   /* USER CODE BEGIN 3 */
272.   uint16_t pressure = 0;
273.   int16_t temperature = 0;
274.   uint16_t humidity = 0;
275.         uint16_t illuminance = 0;
276.         uint16_t moisture = 0;
277.   uint8_t batteryLevel;
278.
279.   if (LORA_JoinStatus() != LORA_SET)
280.   {
281.     /*Not joined, try again later*/
```

```
282.     LORA_Join();
283.     return;
284.   }
285.
286.   TVL1(PRINTF("SEND REQUEST\n\r");)
287. #ifndef CAYENNE_LPP
288.   int32_t latitude, longitude = 0;
289.   uint16_t altitudeGps = 0;
290. #endif
291.
292. #ifdef USE_B_L072Z_LRWAN1
293.   TimerInit(&TxLedTimer, OnTimerLedEvent);
294.
295.   TimerSetValue(&TxLedTimer, 200);
296.
297.   LED_On(LED_RED1) ;
298.
299.   TimerStart(&TxLedTimer);
300. #endif
301.
302.   //BSP_sensor_Read(&sensor_data);
303.
304.         #ifdef CAYENNE_LPP
305.   uint8_t cchannel = 0;
306.         temperature = (int16_t)(hdc2080.readTemp() * 10);
   /* in °C * 10 */
307.   humidity    = (uint16_t)(hdc2080.readHumidity() * 2);      /* in
   hPa / 10 */
308.   pressure    = (uint16_t)(ms5637.readPressure() * 100 / 10);
   /* in %*2     */
309.         illuminance = (uint16_t) tsl2591.getLux();
310.         moisture    = (uint16_t)soilMoist.moistureRead() * 10;
311.
312.   uint32_t i = 0;
313.
314.   batteryLevel = LORA_GetBatteryLevel();                     /* 1
   (very low) to 254 (fully charged) */
315.
316.   AppData.Port = LPP_APP_PORT;
317.
318.         AppData.Buff[i++] = cchannel++;
319.         AppData.Buff[i++] = LPP_DATATYPE_ILLUMINANCE;
320.   AppData.Buff[i++] = (illuminance >> 8) & 0xFF;
321.   AppData.Buff[i++] = illuminance & 0xFF;
322.   AppData.Buff[i++] = cchannel++;
323.   AppData.Buff[i++] = LPP_DATATYPE_TEMPERATURE;
324.   AppData.Buff[i++] = (temperature >> 8) & 0xFF;
325.   AppData.Buff[i++] = temperature & 0xFF;
326.   AppData.Buff[i++] = cchannel++;
327.   AppData.Buff[i++] = LPP_DATATYPE_HUMIDITY;
328.   AppData.Buff[i++] = humidity & 0xFF;
329. #if defined( REGION_US915 ) || defined ( REGION_AU915 ) || defined
   ( REGION_AS923 )
330.   /* The maximum payload size does not allow to send more data for
   lowest DRs */
331. #else
332.   AppData.Buff[i++] = cchannel++;
```

```
333.   AppData.Buff[i++] = LPP_DATATYPE_ANALOG_INPUT;
334.         //AppData.Buff[i++] = batteryLevel*100/254;
335.
336.   AppData.Buff[i++] = (moisture >> 8) & 0xFF;
337.         AppData.Buff[i++] = moisture & 0xFF;
338.         AppData.Buff[i++] = cchannel++;
339.   AppData.Buff[i++] = LPP_DATATYPE_BAROMETER;
340.   AppData.Buff[i++] = (pressure >> 8) & 0xFF;
341.   AppData.Buff[i++] = pressure & 0xFF;
342.   AppData.Buff[i++] = cchannel++;
343.   AppData.Buff[i++] = LPP_DATATYPE_DIGITAL_OUTPUT;
344.   AppData.Buff[i++] = ElectroValveState;
345. #endif  /* REGION_XX915 */
346. #else  /* not CAYENNE_LPP */
347.
348.   temperature = (int16_t)(sensor_data.temperature * 100);
   /* in °C * 100 */
349.   pressure    = (uint16_t)(sensor_data.pressure * 100 / 10);
   /* in hPa / 10 */
350.   humidity    = (uint16_t)(sensor_data.humidity * 10);
   /* in %*10     */
351.   latitude = sensor_data.latitude;
352.   longitude = sensor_data.longitude;
353.   uint32_t i = 0;
354.
355.   batteryLevel = LORA_GetBatteryLevel();                     /* 1
   (very low) to 254 (fully charged) */
356.
357.   AppData.Port = LORAWAN_APP_PORT;
358.
359. #if defined( REGION_US915 ) || defined ( REGION_AU915 ) || defined
   ( REGION_AS923 )
360.   AppData.Buff[i++] = AppLedStateOn;
361.   AppData.Buff[i++] = (pressure >> 8) & 0xFF;
362.   AppData.Buff[i++] = pressure & 0xFF;
363.   AppData.Buff[i++] = (temperature >> 8) & 0xFF;
364.   AppData.Buff[i++] = temperature & 0xFF;
365.   AppData.Buff[i++] = (humidity >> 8) & 0xFF;
366.   AppData.Buff[i++] = humidity & 0xFF;
367.   AppData.Buff[i++] = batteryLevel;
368.   AppData.Buff[i++] = 0;
369.   AppData.Buff[i++] = 0;
370.   AppData.Buff[i++] = 0;
371. #else  /* not REGION_XX915 */
372.   AppData.Buff[i++] = AppLedStateOn;
373.   AppData.Buff[i++] = (pressure >> 8) & 0xFF;
374.   AppData.Buff[i++] = pressure & 0xFF;
375.   AppData.Buff[i++] = (temperature >> 8) & 0xFF;
376.   AppData.Buff[i++] = temperature & 0xFF;
377.   AppData.Buff[i++] = (humidity >> 8) & 0xFF;
378.   AppData.Buff[i++] = humidity & 0xFF;
379.   AppData.Buff[i++] = batteryLevel;
380.   AppData.Buff[i++] = (latitude >> 16) & 0xFF;
381.   AppData.Buff[i++] = (latitude >> 8) & 0xFF;
382.   AppData.Buff[i++] = latitude & 0xFF;
383.   AppData.Buff[i++] = (longitude >> 16) & 0xFF;
384.   AppData.Buff[i++] = (longitude >> 8) & 0xFF;
```

```
385.   AppData.Buff[i++] = longitude & 0xFF;
386.   AppData.Buff[i++] = (altitudeGps >> 8) & 0xFF;
387.   AppData.Buff[i++] = altitudeGps & 0xFF;
388. #endif  /* REGION_XX915 */
389. #endif  /* CAYENNE_LPP */
390.   AppData.BuffSize = i;
391.
392.   LORA_send(&AppData, LORAWAN_DEFAULT_CONFIRM_MSG_STATE);
393.
394.   /* USER CODE END 3 */
395. }
396.
397.
398. static void LORA_RxData(lora_AppData_t *AppData)
399. {
400.   /* USER CODE BEGIN 4 */
401.   PRINTF("PACKET RECEIVED ON PORT %d\n\r", AppData->Port);
402.
403.   switch (AppData->Port)
404.   {
405.    case 3:
406.       /*this port switches the class*/
407.       if (AppData->BuffSize == 1)
408.       {
409.         switch (AppData->Buff[0])
410.         {
411.           case 0:
412.           {
413.             LORA_RequestClass(CLASS_A);
414.             break;
415.           }
416.           case 1:
417.           {
418.             LORA_RequestClass(CLASS_B);
419.             break;
420.           }
421.           case 2:
422.           {
423.             LORA_RequestClass(CLASS_C);
424.             break;
425.           }
426.           default:
427.             break;
428.         }
429.       }
430.       break;
431.     case LPP_APP_PORT:
432.     {
433.       ElectroValveState = (AppData->Buff[2] == 0x64) ?  0x01 :
   0x00;
434.       if (ElectroValveState == RESET)
435.       {
436.         PRINTF("VALVE CLOSED\n\r");
437.         valve_Set(VALVE_CLOSE) ;
438.
439.       }
440.       else
```

```
441.       {
442.         PRINTF("VALVE OPEN\n\r");
443.         valve_Set(VALVE_OPEN) ;
444.       }
445.       break;
446.     }
447.     default:
448.       break;
449.   }
450.                   /* USER CODE END 4 */
451. }
452.
453. static void OnTxTimerEvent(void *context)
454. {
455.   /*Wait for next tx slot*/
456.   TimerStart(&TxTimer);
457.
458.   AppProcessRequest = LORA_SET;
459. }
460.
461. static void LoraStartTx(TxEventType_t EventType)
462. {
463.   if (EventType == TX_ON_TIMER)
464.   {
465.     /* send everytime timer elapses */
466.     TimerInit(&TxTimer, OnTxTimerEvent);
467.     TimerSetValue(&TxTimer,  APP_TX_DUTYCYCLE);
468.     OnTxTimerEvent(NULL);
469.   }
470.   else
471.   {
472.     /* send everytime button is pushed */
473.     GPIO_InitTypeDef initStruct = {0};
474.
475.     initStruct.Mode = GPIO_MODE_IT_RISING;
476.     initStruct.Pull = GPIO_PULLUP;
477.     initStruct.Speed = GPIO_SPEED_HIGH;
478.
479.     HW_GPIO_Init(USER_BUTTON_GPIO_PORT, USER_BUTTON_PIN,
   &initStruct);
480.     HW_GPIO_SetIrq(USER_BUTTON_GPIO_PORT, USER_BUTTON_PIN, 0,
   Send);
481.   }
482. }
483.
484. static void LORA_ConfirmClass(DeviceClass_t Class)
485. {
486.   PRINTF("switch to class %c done\n\r", "ABC"[Class]);
487.
488.   /*Optionnal*/
489.   /*informs the server that switch has occurred ASAP*/
490.   AppData.BuffSize = 0;
491.   AppData.Port = LORAWAN_APP_PORT;
492.
493.   LORA_send(&AppData, LORAWAN_UNCONFIRMED_MSG);
494. }
495.
```

```
496. static void LORA_TxNeeded(void)
497. {
498.   AppData.BuffSize = 0;
499.   AppData.Port = LORAWAN_APP_PORT;
500.
501.   LORA_send(&AppData, LORAWAN_UNCONFIRMED_MSG);
502. }
503.
504. /**
505.   * @brief This function return the battery level
506.   * @param none
507.   * @retval the battery level  1 (very low) to 254 (fully charged)
508.   */
509. uint8_t LORA_GetBatteryLevel(void)
510. {
511.   uint16_t batteryLevelmV;
512.   uint8_t batteryLevel = 0;
513.
514.   batteryLevelmV = HW_GetBatteryLevel();
515.
516.
517.   /* Convert batterey level from mV to linea scale: 1 (very low) to
   254 (fully charged) */
518.   if (batteryLevelmV > VDD_BAT)
519.   {
520.     batteryLevel = LORAWAN_MAX_BAT;
521.   }
522.   else if (batteryLevelmV < VDD_MIN)
523.   {
524.     batteryLevel = 0;
525.   }
526.   else
527.   {
528.     batteryLevel = (((uint32_t)(batteryLevelmV - VDD_MIN) *
   LORAWAN_MAX_BAT) / (VDD_BAT - VDD_MIN));
529.   }
530.
531.   return batteryLevel;
532. }
533.
534. #ifdef USE_B_L072Z_LRWAN1
535. static void OnTimerLedEvent(void *context)
536. {
537.   LED_Off(LED_RED1) ;
538. }
539. #endif
540. /************************ (C) COPYRIGHT STMicroelectronics *****END
   OF FILE****/
```

ÍNDICE DEL ANEXO 2. Hojas de características

En este anexo se encuentran las partes más relevantes de las hojas de características de los componentes, las versiones completas de estas pueden descargarse desde las webs de los fabricantes y se encuentran referenciadas en el apartado de bibliografía de la memoria de este documento.

1. Hoja de características de CMWX1ZZABZ-078

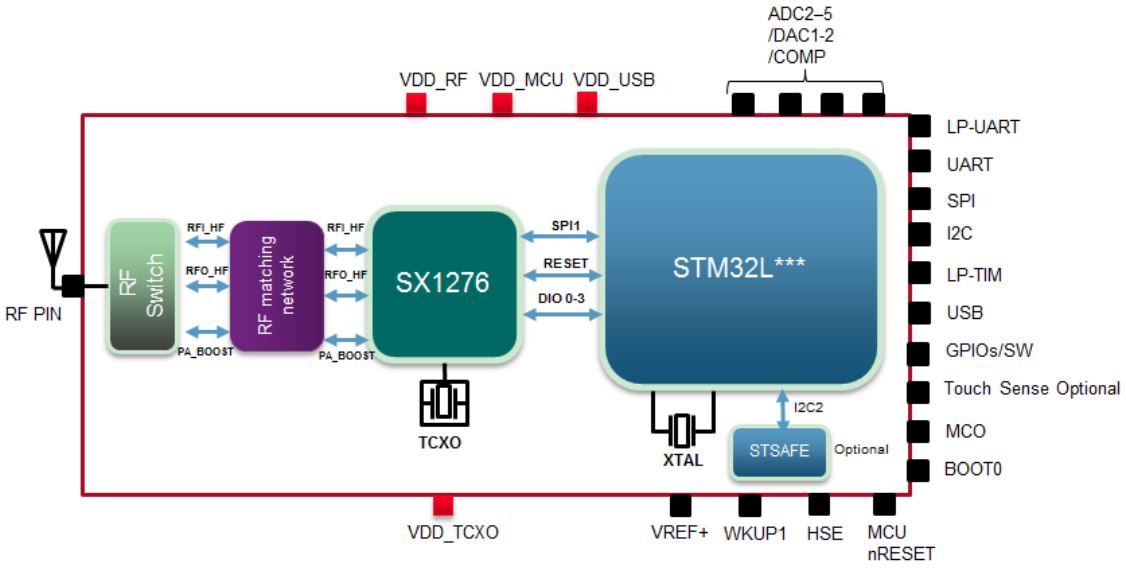
1. Features

Interfaces	: I2C, UART, USB, SPI
Main ICs	: STM32L, SX1276
Reference Clocks	: Integrated 32MHz clock (TCXO with frequency error=±2 ppm) and 32.768KHz clock (frequency error=±20 ppm)
Supported Frequencies	: 868 MHz, 915 MHz
Module Size	: 12.5 mm x 11.6 mm x 1.76 mm (Max)
Weight	: 0.48g (Typ)
Package	: Metal Shield can
RoHS	: This module is compliant with the RoHS directive

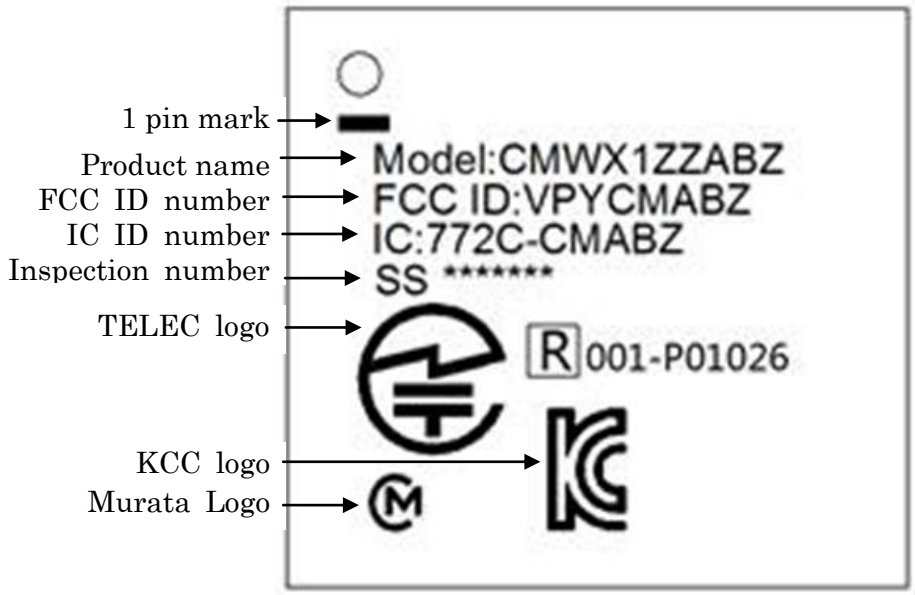
2. Part Number

Ordering Part Number	MCU	Secure element	Description
CMWX1ZZABZ-**TEMP**	STM32L**082**	NA	Engineering sample
CMWX1ZZABZ-**EVK**	STM32L**082**	NA	Evaluation board
CMWX1ZZABZ-**078**	STM32L**082**	NA	MP P/N
CMWX1ZZABZ-**TEMP-1**	STM32L**072**	NA	Engineering sample
CMWX1ZZABZ-**EVK-1**	STM32L**072**	NA	Evaluation board
CMWX1ZZABZ-**091**	STM32L**072**	NA	MP P/N

3. Block Diagram



4. Label Information



5. Absolute Maximum Ratings

Table 3 Maximum ratings

Parameters		Min	Typ	Max	Unit
Storage Temperature		-40	25	+90	degC
Input RF Level		-	-	10	dBm
Supply Voltage	VDD_USB	-0.3	-	3.9	V
	VDD_MCU, VDD_RF, VDD_TCXO	-0.3	-	3.9	V
	VREF+	-0.3	-	V_{DD_MCU}+0.4	V

6. Operating Condition

Table 4 Operating specification

Parameters		Min	Typ	Max	Unit
Operating Temperature		-40	25	+85	degC
Supply Voltage	VDD_USB (USB peripheral used) (1)	3.0	-	3.6	V
	VDD_USB(USB peripheral not used) (1)	$V_{DD\_MCU\_min}$	V_{DD_MCU}	$V_{DD\_MCU\_max}$	V
	VDD_MCU,VDD_RF,VDD_TCXO	2.2(3)	-	3.6	V
	VREF+(2)	1.8	-	V_{DD_MCU}	V

(1) VDD_USB must respect the following conditions:
- When VDD_MCU is powered on (VDD_MCU < VDD_MCU_min), VDD_USB should be always lower than VDD_MCU.
- When VDD_MCU is powered down (VDD_MCU < VDD_MCU_min), VDD_USB should be always lower than VDD_MCU.
- In operating mode, VDD_USB could be lower or higher than VDD_MCU.
- If the USB is not used, VDD_USB must be tied to VDD_MCU to be able to use PA11 and PA12 as standard I/Os.

(2) VREF+ is used to ensure a better accuracy on low-voltage inputs and outputs of ADC and DAC. Detailed information is on the STM32L082*** datasheet and user guider.

(3) When module is on +20dBm operation, the supply of the voltage should be set from 2.4V to 3.6V.

7. Electrical Characteristics

7.1. FSK/OOK Transceiver Specification

Conditions:

Supply voltage VDD=3.3 V, temperature = 25 °C, FXOSC = 32 MHz, FRF =868/915 MHz , 2-level FSK modulation without pre-filtering, FDA = 5 kHz, Bit Rate = 4.8 kb/s and terminated in a matched 50 Ohm impedance, shared Rx and Tx path matching, unless otherwise specified.

FSK/OOK Receiver Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
RFS_F_HF	LnaBoost is turned on	FDA = 5 kHz, BR = 4.8 kb/s		-117.5		dBm
IDDR (*)	Supply current in Receive mode	LnaBoost Off, band 1		22		mA
		LnaBoost On, band 1		23		mA

FSK/OOK Transmitter Specification

Symbol	Description	Conditions		Min.	Typ	Max	Unit
RF_OP	RF output power in 50 ohms on RFO pin (High efficiency PA)	Programmable with steps	Max		14		dBm
			Min		-5		dBm
RF_OPH	RF output power in 50 ohms on PA_BOOST pin(Regulated PA)	Programmable with 1dB steps	Max		18.5		dBm
			Min		2		dBm
ΔRF_ OPH_V	RF output power stability on PA_BOOST pin versus voltage supply.	VDD = 2.2 V to 3.6 V			+/-1		dB
ΔRF_T	RF output power stability versus temperature on PA_BOOST pin.	From T = -40 °C to +85 °C			+/-1.5		dB
IDDT (*)	Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST			128		mA
		RFOP = +17 dBm, on PA_BOOST			106		mA
		RFOP = +14 dBm, on RFO_HF pin			47		mA
		RFOP = + 7 dBm, on RFO_HF pin			34		mA

(*) IDDR and IDDT are total current consumption including MCU in active.

7.2. LoRa Transceiver Specification

Conditions:

The table below gives the electrical specifications for the transceiver operating with LoRaTM modulation. Following conditions apply unless otherwise specified: Supply voltage = 3.3 V, Temperature = 25° C, FXOSC = 32 MHz, Error Correction Code (EC) = 4/5, Packet Error Rate (PER)= 1%, CRC on payload enabled, Payload length = 10 bytes. With matched impedances

LoRa Receiver Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
IDDR_L (*)	Supply current in receiver LoRa mode, LnaBoost off	Band 1, BW = 125 kHz		21.5		mA
		Band 1, BW = 250 kHz		22.2		mA
		Band 1, BW = 500 kHz		23.6		mA
RFS_L125_HF	RF sensitivity, Long-Range Mode, highest LNA gain, LnaBoost for Band1, using split Rx/Tx path 125 kHz bandwidth	SF = 6		-117.5		dBm
		SF = 7		-122.5		dBm
		SF = 8		-125.5		dBm
		SF = 9		-128.5		dBm
		SF = 10		-131.0		dBm
		SF = 11		-133.5		dBm
		SF = 12		-135.5		dBm
RFS_L250_HF	RF sensitivity, Long-Range Mode, highest LNA gain, LnaBoost for Band1, using split Rx/Tx path	SF = 6		-114.0		dBm
		SF = 7		-119.0		dBm
		SF = 8		-122.0		dBm
		SF = 9		-125.0		dBm

	250 kHz bandwidth	SF = 10		-127.5		dBm
		SF = 11		-130.0		dBm
		SF = 12		-133.0		dBm

LoRa Transmitter Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
IDDT_L (*)	Supply current in transmitter mode	RFOP setting = 14 dBm		47		mA
		RFOP setting = 10 dBm		36		mA
IDDT_H_L (*)	Supply current in transmitter mode	Using PA_BOOST pin RFOP setting = 20 dBm		128		mA

(*) IDDR_L, IDDT_L and IDDT_H_L are total current consumption including MCU in active.

7.3. SIGFOX Transceiver Specification

Conditions:
The table below gives the electrical specifications for the transceiver operating with SIGFOX modulation. Following conditions apply unless otherwise specified: Supply voltage = 3.3 V, Temperature = 25° C. With matched impedances.

Notes: To operate as SIGFOX mode, the following configuration is required.
- TCXO_OUT (Pin 47) must be connected to PH0-OSC_IN (Pin46).
- PA12 (Pin 1) must be connected to TXCO_VCC (Pin48).
- SX1276_DIO4 (Pin10) must be connected to PA5 (Pin21).

SIGFOX Receiver Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
RFS_F_HF		AT$SB=x,1, AT$SF=x,1, AT$SR PER<0.1		-122		dBm
IDDR_S	Supply current in Receive mode	AT$TM=3,10		23		mA

SIGFOX Transmitter Specification

Symbol	Description	Conditions		Min.	Typ	Max	Unit
RF_OP_S	RF output power in 50 ohms on RF pin	Programmable with steps AT$SF	Max		18.5		dBm
			Min		4.5		dBm
IDDT_S	Supply current in Transmit mode with impedance matching	Output power setting 20 dBm AT$SF			128		mA
		Output power setting 14 dBm AT$SF			44		mA

7.4. Low power mode current

Conditions:
Power supply: 3.3V, Temp: Room, TCXO_VDD (pin 48 of the module) is connected to PA12 (Pin1 of the module)

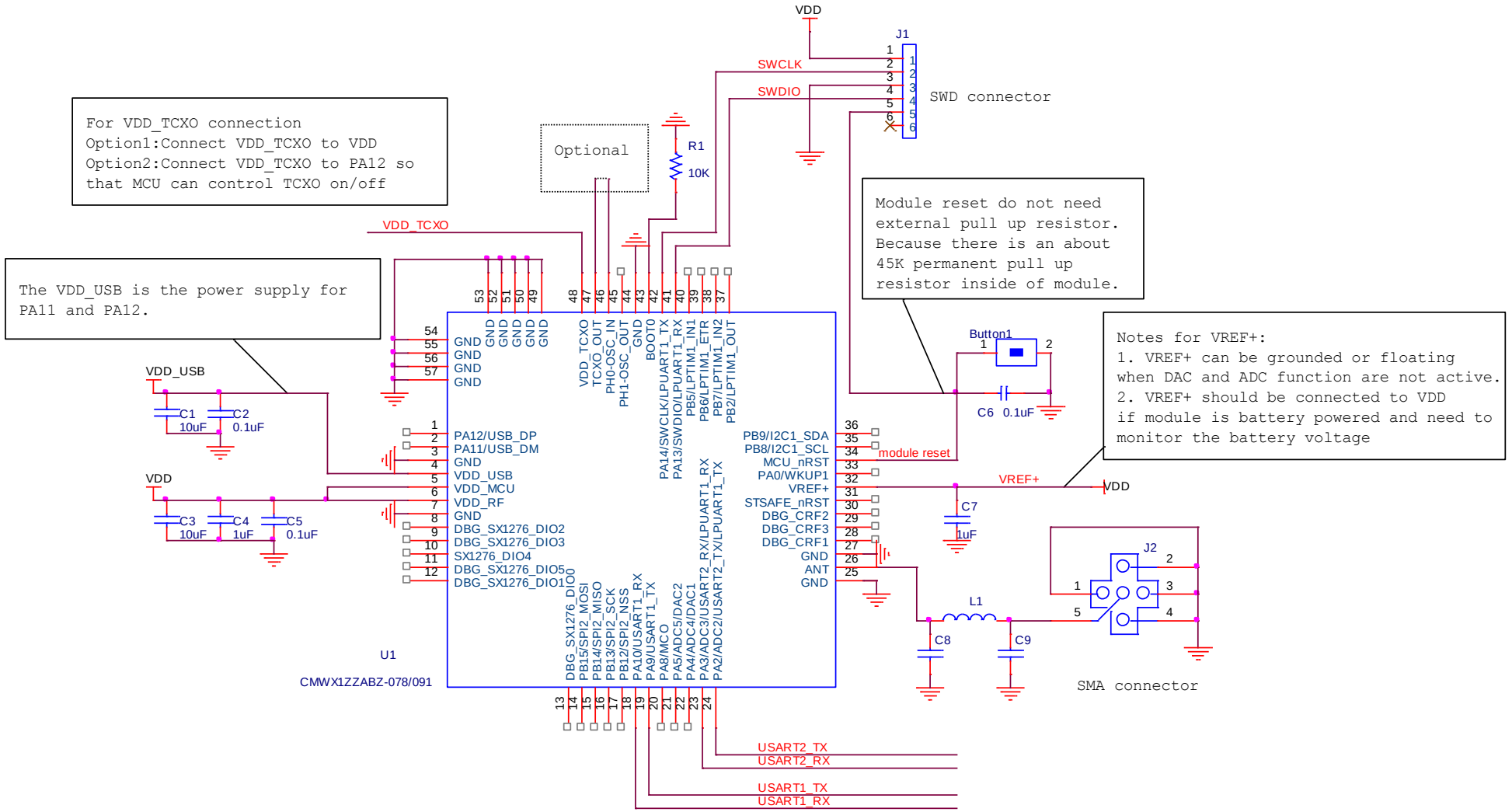
Mode	Description	Min.	Typ	Max	Unit
Mode0	STM32L0 in Stop mode with RTC (Real Time Clock) (*1) (*3) SX1276 in Sleep mode		1.65		uA
Mode1	STM32L0 in Standby mode with RTC (Real Time Clock) (*2) SX1276 in Sleep mode		1.40		uA

(*1) The Stop mode achieves the lowest power consumption while retaining the RAM and register contents and real time clock. All clocks in the V_{CORE} domain are stopped, the PLL, MSI RC, HSE crystal and HSI RC oscillators are disabled. The LSE or LSI is still running. The voltage regulator is in the low-power mode.
Some peripherals featuring wakeup capability can enable the HSI RC during Stop mode to detect their wakeup condition. The device can be woken up from Stop mode by any of the EXTI line, in 3.5us, the processor can serve the interrupt or resume the code. The EXTI line source can be any GPIO. It can be the PVD output, the comparator 1 event or comparator 2 event (if internal reference voltage is on), it can be the RTC alarm/tamper/timestamp/wakeup events, the USB/USART/I2C/LPUART/LPTIMER wakeup events.

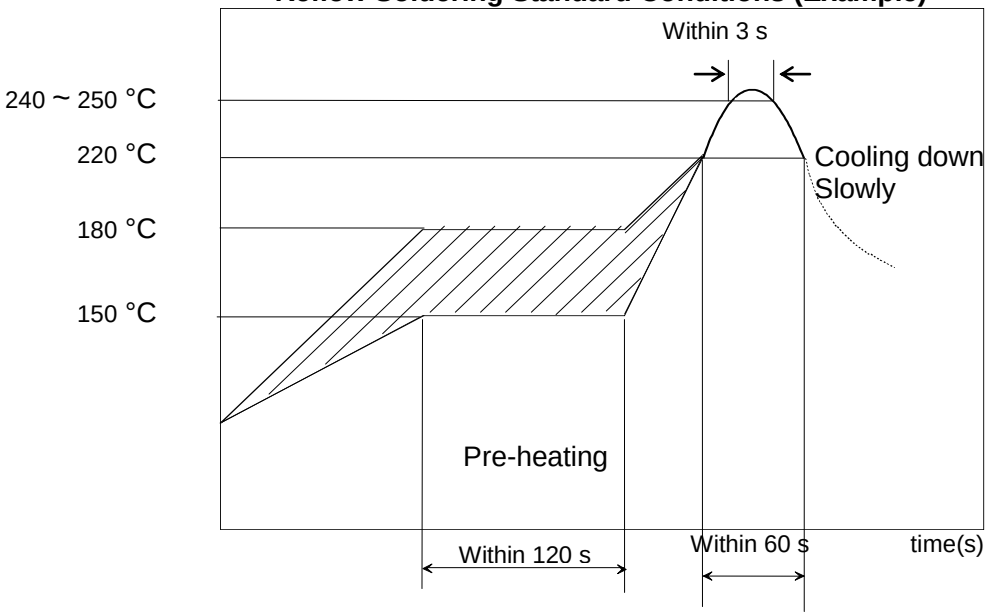
(*2) The Standby mode is used to achieve the lowest power consumption and real time clock. The internal voltage regulator is switched off so that the entire VCORE domain is powered off. The PLL, MSI RC, HSE crystal and HSI RC oscillators are also switched off. The LSE or LSI is still running. After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE Crystal 32 KHz oscillator, RCC_CSR register). The device exits Standby mode in 60 µs when an external reset (NRST pin), an

9. Reference circuit

In case of using the module as LoRa



Reflow Soldering Standard Conditions (Example)



Please use the reflow within 2 times.
Use rosin type flux or weakly active flux with a chlorine content of 0.2 wt % or less.

11.6 Cleaning :
Since this Product is Moisture Sensitive, any cleaning is not permitted.

11.7 Operational Environment Conditions :
Products are designed to work for electronic products under normal environmental conditions (ambient temperature, humidity and pressure). Therefore, products have no problems to be used under the similar conditions to the above-mentioned. However, if products are used under the following circumstances, it may damage products and leakage of electricity and abnormal temperature may occur.

- In an atmosphere containing corrosive gas (Cl2, NH3, SOx, NOx etc.).
- In an atmosphere containing combustible and volatile gases.
- Dusty place.
- Direct sunlight place.
- Water splashing place.
- Humid place where water condenses.
- Freezing place.

If there are possibilities for products to be used under the preceding clause, consult with Murata before actual use.

As it might be a cause of degradation or destruction to apply static electricity to products, do not apply static electricity or excessive voltage while assembling and measuring.

11.8 Input Power Capacity :
Products shall be used in the input power capacity as specified in this specifications.
Inform Murata beforehand, in case that the components are used beyond such input power capacity range.

2. Hoja de características de B-L072Z

UM2115
User manual

Discovery kit for LoRaWAN™, Sigfox™, and LPWAN protocols with STM32L0

Introduction

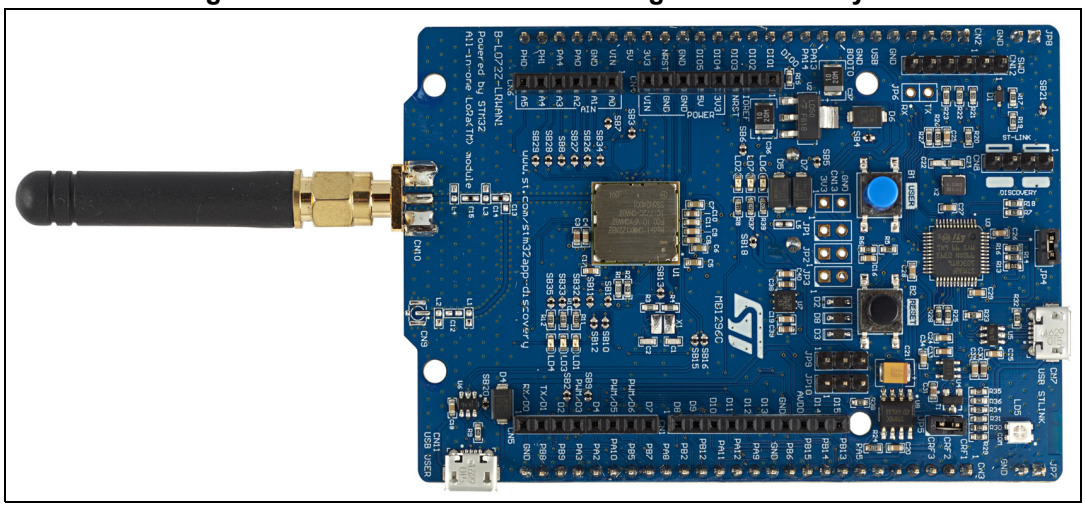
The B-L072Z-LRWAN1 Discovery kit embeds the CMWX1ZZABZ-091 LoRa®/Sigfox™ module (Murata). This Discovery kit allows users to develop easily applications with the STM32L072CZ and the LoRa®/Sigfox™ RF connectivity in one single module.
The B-L072Z-LRWAN1 Discovery kit has the full set of features available in the STM32L0 Series and offers ultra-low-power and LoRa®/Sigfox™ RF features. The B-L072Z-LRWAN1 Discovery kit is a low-cost and easy-to-use development kit to quickly evaluate and start a development with an STM32L072CZ microcontroller.

The B-L072Z-LRWAN1 Discovery kit includes LoRa®/Sigfox™ RF interface, LEDs, push-buttons, antenna, Arduino™ Uno V3 connectors, USB 2.0 FS connector in Micro-B format. The integrated ST-LINK/V2-1 provides an embedded in-circuit debugger and programmer for the STM32L0 MCUs.

The LoRaWAN™ stack is certified class A and C compliant. It is available inside the I-CUBE-LRWAN firmware package.The Sigfox™ stack is RCZ1, RCZ2, and RCZ4 certified. It is available inside the X-CUBE-SFOX expansion package.

To help users setting up a complete node (LoRaWAN™, Sigfox™, or both), the B-L072Z-LRWAN1 Discovery kit comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube package, as well as a direct access to the Arm® Mbed Enabled™ resources at the http://mbed.org website.

Figure 1. B-L072Z-LRWAN1 LoRa®/Sigfox™ Discovery kit



Picture is not contractual.

1 Features

The B-L072Z-LRWAN1 Discovery kit offers the following features:

- CMWX1ZZABZ-091 LoRa®/Sigfox™ module (Murata)
  - Embedded ultra-low-power STM32L072CZ Series MCUs, based on Arm® Cortex® -M0+ core, with 192 Kbytes of Flash memory, 20 Kbytes of RAM, 20 Kbytes of EEPROM
  - Frequency range: 860 MHz - 930 MHz
  - Frequency MHz (min): 860 MHz
  - Frequency MHz (max): 930 MHz
  - USB 2.0 FS
  - 4-channel,12-bit ADC, 2xDAC
  - 6-bit timers, LP-UART, I^2C and SPI
  - Embedded SX1276 transceiver
  - LoRa®, FSK, GFSK, MSK, GMSK and OOK modulations (+ Sigfox™ compatibility)
  - +14 dBm or +20 dBm selectable output power
  - 157 dB maximum link budget
  - Programmable bit rate up to 300 kbit/s
  - High sensitivity: down to -137 dBm
  - Bullet-proof front end: IIP3 = -12.5 dBm
  - 89 dB blocking immunity
  - Low RX current of 10 mA, 200 nA register retention
  - Fully integrated synthesizer with a resolution of 61 Hz
  - Built-in bit synchronizer for clock recovery
  - Sync word recognition
  - Preamble detection
  - 127 dB+ dynamic range RSSI
- Including 50 ohm SMA RF antenna
- 1 user and reset push-buttons
- Board connectors:
  - USB FS connector
  - SMA and U.FL RF
- Board expansion connectors:
  - Arduino™ Uno V3
- 7 LEDs:
  - 4 general-purpose LEDs
  - 5 V-power LED
  - ST-LINK-communication LED
  - Fault-power LED
- Flexible power-supply options: ST-LINK USB V_{BUS} or external sources

8 Hardware layout and configuration

The B-L072Z-LRWAN1 Discovery kit has been designed around the Murata LoRa®/Sigfox™ module including the STM32L072CZ microcontroller in a 49-pin WLCSP package.

Figure 2 illustrates the connection between the Murata LoRa®/Sigfox™ module and the peripherals (ST-LINK/V2, RF Antenna, LEDs, push-buttons, USB 2.0 FS Micro-B connector, 3xAAA battery holder).

Figure 3 and *Figure 4* help users to locate these features on the STM32L072 Discovery kit.

Figure 2. Hardware block diagram

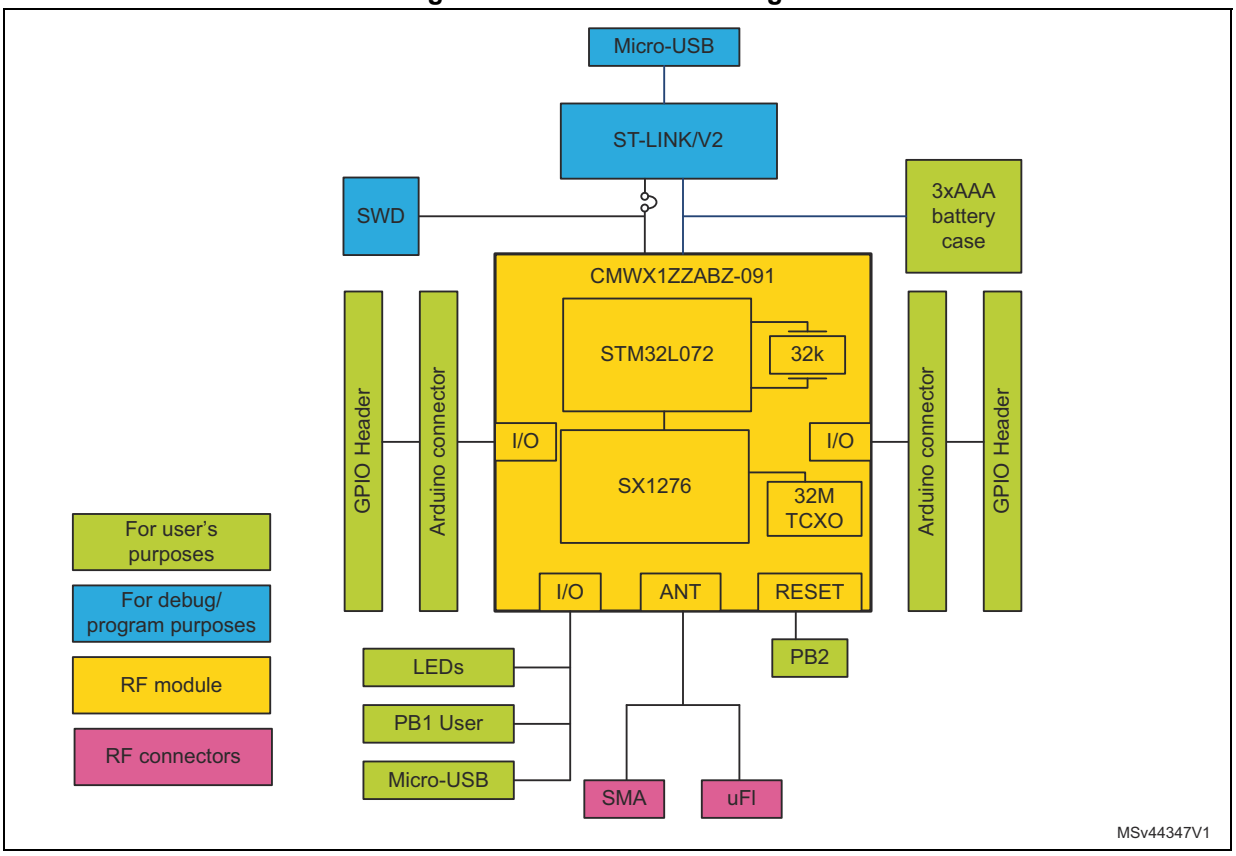


Figure 3. B-L072Z-LRWAN1 top layout

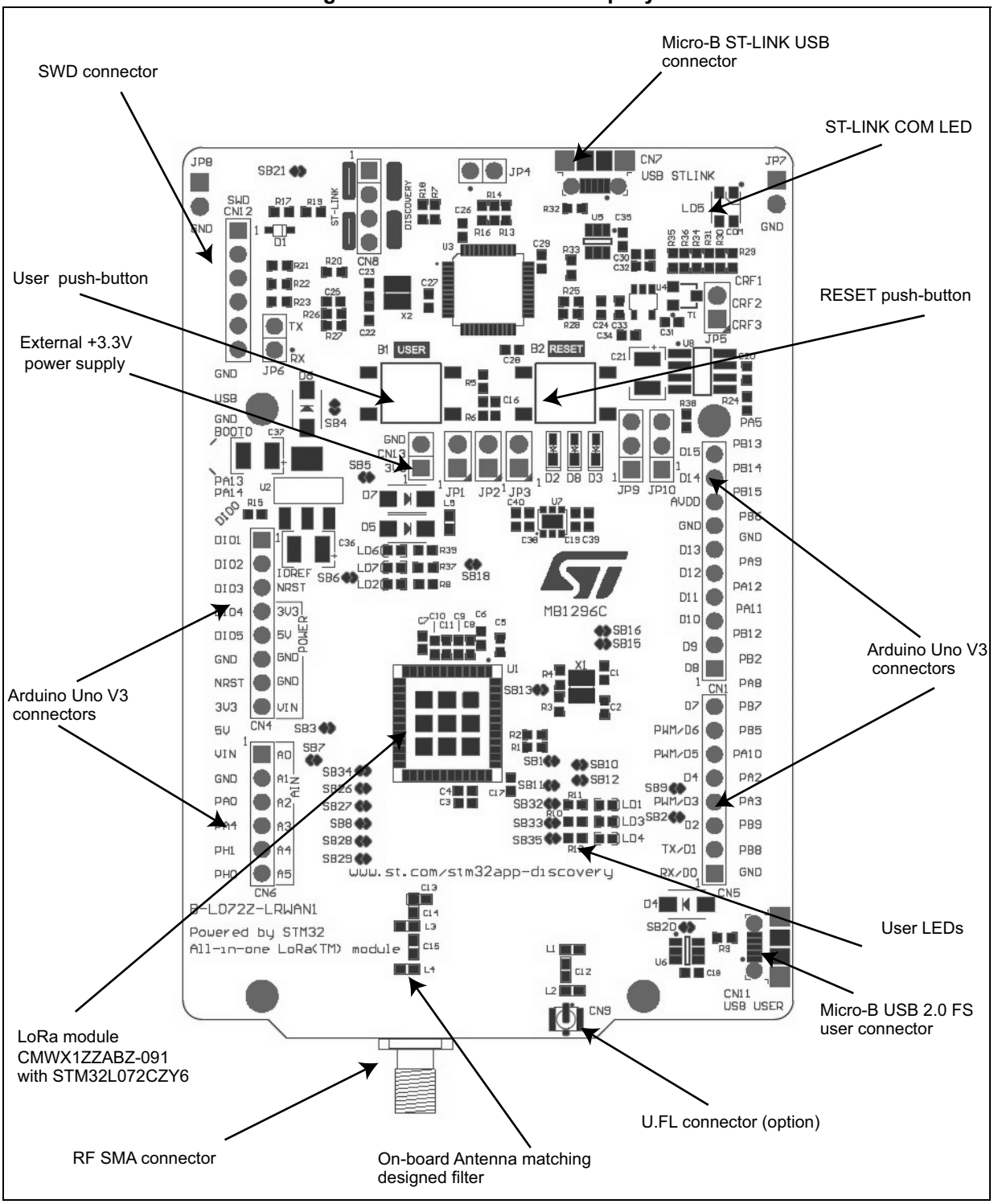
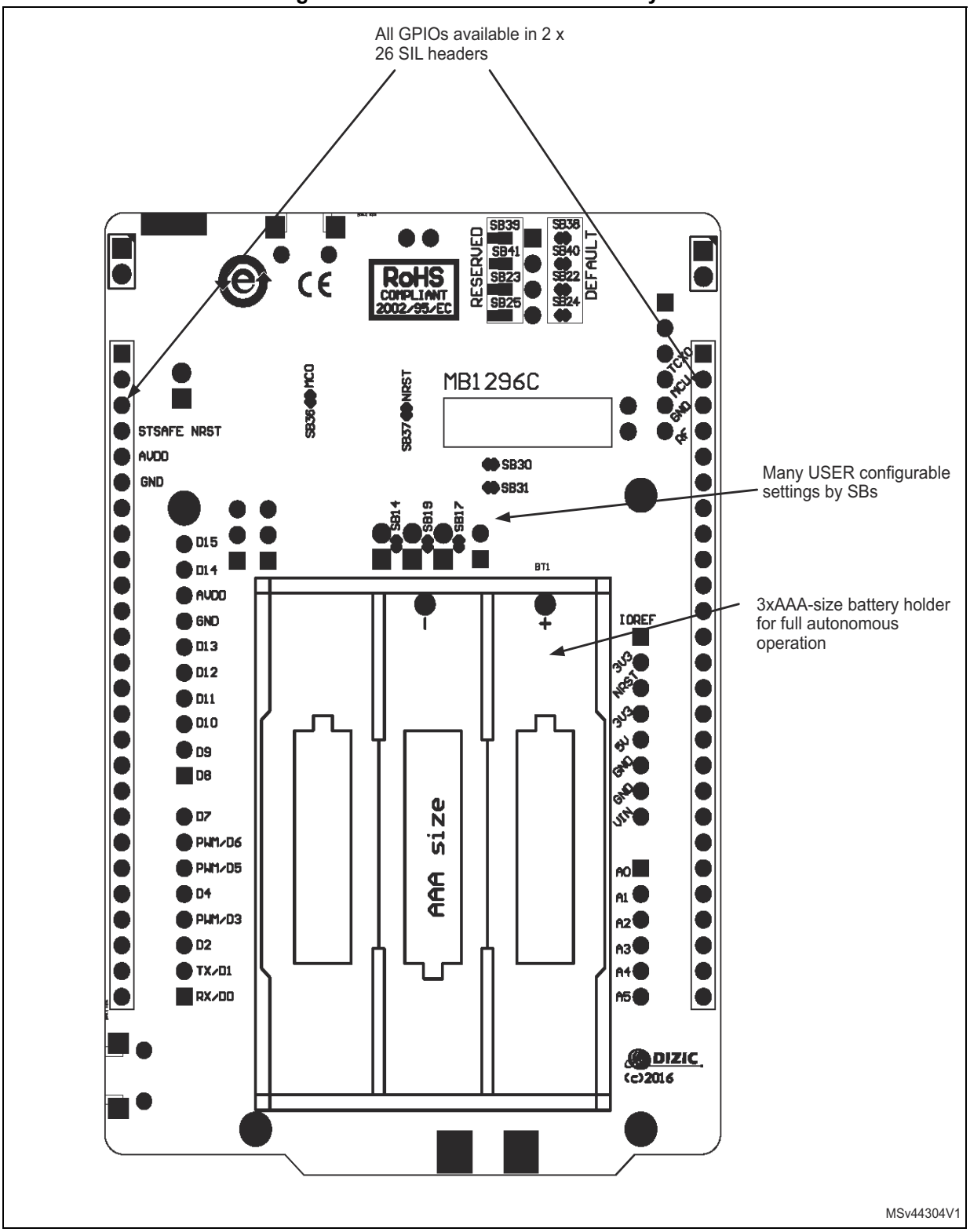


Figure 4. B-L072Z-LRWAN1 bottom layout



8.3 Power supply

The B-L072Z-LRWAN1 Discovery kit is designed to be powered in various ways. It can be simply plugged on a USB PC port with a Micro-B USB cable. In this mode, the board is programmed and debugged via the ST-LINK/V2-1 USB port on CN7. It is possible to use one of the different following sources:

- External +3.3 V connected to CN13 (+3.3 V and GND pins of CN13 must be connected).
- 7-12 V DC power supply plugged on Arduino™ Uno V3 connectors: VIN on pin 8 and GND on pin 7 of CN4 (VIN and GND pins must be both connected)
- USB 2.0 FS Micro-B on CN11 connector (Device mode). The power supply is provided by the USB port connected to CN11.
- On-board 3xAAA-sized battery holder BT1 located on the bottom side of the Discovery kit (batteries are not delivered inside the Discovery kit package). Respect the battery polarities mentioned in the battery case.
- 5V_ST_LINK DC power with limitation from ST-LINK USB connector. The USB type Micro-B connector CN7 of ST-LINK/V2-1. If the USB enumeration succeeds (as explained below), the ST-LINK 5 V link power is enabled, by asserting the PWR_ENn signal. This pin is connected to a power switch ST890, which powers the board. This power switch features also a current limitation to protect the PC in case of a short-circuit on board (more than 625mA). The Discovery kit can be powered from the ST-LINK USB connector, but only the ST-LINK circuit has the power before USB enumeration, because the host PC only provides 100 mA to the board at that time. During the USB enumeration, the Discovery kit requires 300 mA power from the host PC. If the host is able to provide the required power, the enumeration finishes by a "SetConfiguration" command and then, the power transistor ST890 is switched ON, the red LED LD7 is turned ON, thus the Discovery kit can consume maximum 300 mA current, but no more. If the host is not able to provide the requested current, the enumeration fails. Therefore the ST890 remains OFF and the STM32 part including the extension board is not powered. As a consequence the red LED LD7 remains turned OFF. In this case it is mandatory to use an external power supply.

To further decrease the current consumption of the board, the LED7 must be disconnected by opening SB18.

Users do not have to manage the different configurations with jumpers or switches. The power supplies are internally managed by a set of diodes on the respective power supply branches.

If the board is supplied by CN13, by battery or by CN11, SB37 must be removed to release the RESET pin managed by ST-LINK. In that case the ST-LINK is no more powered.

The red LED LD7 (+5 V power supply) is turned on (with SB18 ON) as soon as one of the power sources listed above is present.

Note: The Discovery kit must be powered by a power supply unit or by an auxiliary equipment complying with the standard EN-60950-1: 2006+A11/2009, and must be Safety Extra Low Voltage (SELV) with limited power capability.

9 Connectors

9.1 Arduino Uno V3 connectors

Table 5. Arduino Uno V3 connectors

Connector	Pin	Pin name	STM32 Pin	Function
CN1	10	D15	PB8	I2C1_SCL
	9	D14	PB9	I2C1_SDA
	8	AVDD	VREF+	VREF+
	7	GND	GND	Ground
	6	D13	PA5 or PB13	SPI1_SCK or SPI2_SCK
	5	D12	PB14	SPI2_MISO
	4	D11	PB15	SPI2_MOSI
	3	D10	PB6	LPTIM1_ETR
	2	D9	PB12	SPI2_NSS
	1	D8	PA9	USART1_TX
CN4	1	NC	-	-
	2	IOREF	-	+3.3 V Ref
	3	RESET	NRST	MCU_nRST
	4	+3.3 V	-	+3.3 V input/output
	5	+5 V	-	5 V output
	6	GND	-	Ground
	7	GND	-	Ground
	8	VIN	-	Power input
CN5	8	D7	PA8	MCO
	7	D6	PB2	LPTIM1_OUT
	6	D5	PB7	LPTIM1_IN2
	5	D4	PB5	LPTIM1_IN1
	4	D3	PB13 or NC	TIM21_CH1 or NC
	3	D2	PA10	USART1_RX
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

Table 5. Arduino Uno V3 connectors (continued)

Connector	Pin	Pin name	STM32 Pin	Function
CN6	1	A0	PA0	ADC_IN0
	2	A1	NC or PA0	NC or ADC_IN0
	3	A2	PA4	ADC_IN4
	4	A3	NC or PA4	NC or ADC_IN4
	5	A4	PH1 or PB9	OSC_IN or I2C1_SDA
	6	A5	PH0 or PB8	OSC_OUT or I2C1_SCL

9.2 B-L072Z-LRWAN1 Discovery kit CN2 and CN3 connectors

Table 6. Connector CN2

Connector	Pin	Pin name	STM32 Pin	Function
CN2	1	TCXO_VCC	-	LoRa®/Sigfox™ module TCXO power
	2	VDD_MCU_LRA	-	MCU section power supply
	3	GND	-	Ground
	4	VDD_RF_LRA	-	MCU section power supply
	5	GND	-	Ground
	6	VDD_USB_LRA	-	MCU section power supply
	7	GND	-	Ground
	8	BOOT0	BOOT0	BOOT0
	9	PA13	PA13	SWDIO
	10	PA14	PA14	SWCLK
	11	SX1276_DIO0	-	LoRa®/Sigfox™module debug pin
	12	SX1276_DIO1	-	LoRa®/Sigfox™module debug pin
	13	SX1276_DIO2	-	LoRa®/Sigfox™module debug pin
	14	SX1276_DIO3	-	LoRa®/Sigfox™module debug pin
	15	SX1276_DIO4	-	LoRa®/Sigfox™module debug pin
	16	SX1276_DIO5	-	LoRa®/Sigfox™ module debug pin
	17	GND	-	Ground
	18	MCU_nRST	NRST	RESET
	19	+3.3 V	-	+3.3 V power supply input/output
	20	+5 V	-	+5 V power supply input
	21	VIN	-	VIN power supply input (7-12Vdc)
	22	GND	-	Ground
	23	PA0	PA0	ADC_IN0
	24	PA4	PA4	ADC_IN4
	25	PH1	PH1	OSC_OUT
	26	PH0	PH0	OSC_IN

Table 7. Connector CN3

Connector	Pin	Pin name	STM32 Pin	Function
CN3	1	CRF1	PA1	LoRa®/Sigfox™ module dedicated pin
	2	CRF2	PC2	LoRa®/Sigfox™ module dedicated pin
	3	CRF3	PC1	LoRa®/Sigfox™ module dedicated pin
	4	STSAFE_nRST	-	STSAFE security IC reset pin
	5	AVDD	VREF+	VREF+
	6	GND	-	Ground
	7	PA5	PA5	ADC_IN5
	8	PB13	PB13	SPI2_SCK
	9	PB14	PB14	SPI2_MISO
	10	PB15	PB15	SPI2_MOSI
	11	PB6	PB6	LPTIM1_ETR
	12	GND	-	Ground
	13	PA9	PA9	USART1_TX
	14	PA12	PA12	USB_DP
	15	PA11	PA11	USB_DM
	16	PB12	PB12	SPI2_NSS
	17	PB2	PB2	LPTIM1_OUT
	18	PA8	PA8	MCO
	19	PB7	PB7	LPTIM1_IN2
	20	PB5	PB5	LPTIM1_IN1
	21	PA10	PA10	USART1_RX
	22	PA2	PA2	ADC_IN2
	23	PA3	PA3	ADC_IN3
	24	PB9	PB9	I2C1_SDA
	25	PB8	PB8	I2C1_SCL
	26	GND	-	Ground

9.3 Other connectors

9.3.1 Debug connector SWD

Table 8. Debug connector SWD (CN12)

Connector	Pin	Pin name	Function
CN12	1	VDD_TARGET	VDD from application
	2	SWCLK	SWD clock
	3	GND	Ground
	4	SWDIO	SWD data input/output
	5	NRST	RESET of target MCU
	6	SWO	Reserved

9.3.2 SWD Interface

It is very easy to use ST-LINK/V2-1 to program an STM32 microcontroller on an external application. Simply remove the two jumpers from CN8 and connect the application to the CN12 debug connector according to *Table 8*.

9.3.3 External +3.3 V

Table 9. External +3.3 V (CN13)

Connector	Pin	Pin name	Function
CN13	1	+3.3 V external	External +3.3 V power supply input
	2	GND	Ground

Caution: When using the external +3.3 V power supply input, SB6 must be OFF.

9.4 Description of the jumpers

Table 10. Description of the jumpers

Jumper	Pin number	Designation	Default state	Function
JP1	2	VDD_RF_LRA	OFF	Allows IDD VDD_RF_LRA measurement
JP2	2	VDD_USB_LRA	OFF	Allows IDD VDD_USB_LRA measurement
JP3	2	VDD_MCU_LRA	OFF	Allows IDD VDD_MCU_LRA measurement
JP5	2	USB charger	OFF	USB charger
JP7, JP8	2	GND	ON	Ground
JP6	2	ST-LINK TX/RX	OFF	ST-LINK TX/RX signals

Table 10. Description of the jumpers (continued)

Jumper	Pin number	Designation	Default state	Function
JP9	3	TCXO selection	2-3	Selection TCXO to VDD or external TCXO power
JP10	3	Reset source selection	1-2	Reset source selection between STSAFE or PA11

9.5 Configuration of the solder bridges

Refer to *Figure 8* and *Figure 9* to locate the solder bridges.

Table 11. Configuration of the solder bridges

Solder bridges	Designation	Default state	Function
SB19	Short VDD_USB_LRA	ON	Short VDD_USB_LRA connection
SB14	Short VDD_MCU_LRA	ON	Short VDD_MCU_LRA connection
SB17	Short VDD_RF_LRA	ON	Short VDD_RF_LRA connection
SB20	Short D4	OFF	D4 bypass
SB4	Short D6	OFF	D6 bypass
SB5	Short D7	OFF	D7 bypass
SB18	+5 V LED	ON	+5 V power supply ON
SB6	+3.3 V regulator output	ON	Used to disconnect internal +3.3 V regulator when external source applied on External 3.3 V pin
SB38,SB40, SB22, SB24	ST-LINK default	ON	Reserved
SB39, SB41, SB23, SB25	ST-LINK reserved	OFF	Reserved
SB37	ST-LINK RESET	ON	Connection between ST-LINK reset signal and LoRa®/Sigfox™ module reset
SB36	ST-LINK MCO	OFF	Optional ST-LINK MCO redirected to LoRa®/Sigfox™ module input clock OSC_IN
SB6	ST-LINK +5 V power	ON	Optional ST-LINK regulator disconnected from +5 V
SB21	ST-LINK force RESET	OFF	Reserved
SB26	PA5 to DIO4	OFF	Reserved to LoRa®/Sigfox™ module debug
SB27	PA4 to DIO5	OFF	Reserved to LoRa®/Sigfox™ module debug
SB28	ST-LINK TX	ON	Virtual COM port TX
SB29	ST-LINK RX	ON	Virtual COM port RX
SB15	LRA_USB_DP	OFF	Optional USB_DP connection
SB16	LRA_USB_DM	OFF	Optional USB_DM connection

Table 11. Configuration of the solder bridges (continued)

Solder bridges	Designation	Default state	Function
SB13	TCXO_OUT to OSC_IN	OFF	Allows connection of TXCO output to STM32L072CZY6 OSC_IN input
SB31	USER button PB2	ON	User push-button connected to PB2
SB30	USER button PA0	OFF	User push-button connected to PA0
SB32	LED LD1	ON	LD1 ON
SB33	LED LD3	ON	LD3 ON
SB35	LED LD4	ON	LD4 ON
SB34	LED LD2	ON	LD2 ON
SB10	PH1 Arduino	OFF	Connection A5(CN6) Arduino to PH1
SB3	PA5 Arduino	ON	Connection D13(CN1) Arduino to PA5
SB9	PB13 Arduino	ON	Connection D3(CN5) Arduino to PB13
SB2	PB13 Arduino	OFF	Connection D13(CN5) Arduino to PB13
SB7	PA0 alias Arduino	OFF	Connection A1(CN6) Arduino to PA0
SB8	PA4 alias Arduino	OFF	Connection A3(CN6) Arduino to PA4
SB11	PB9 Arduino	OFF	Connection A4(CN6) Arduino to PB9
SB12	PB8 Arduino	OFF	Connection A54(CN6) Arduino to PB8
SB1	PH0 Arduino	OFF	Connection A4(CN6) Arduino to PH0

Figure 10. B-L072Z-LRWAN1 Discovery kit, Top view

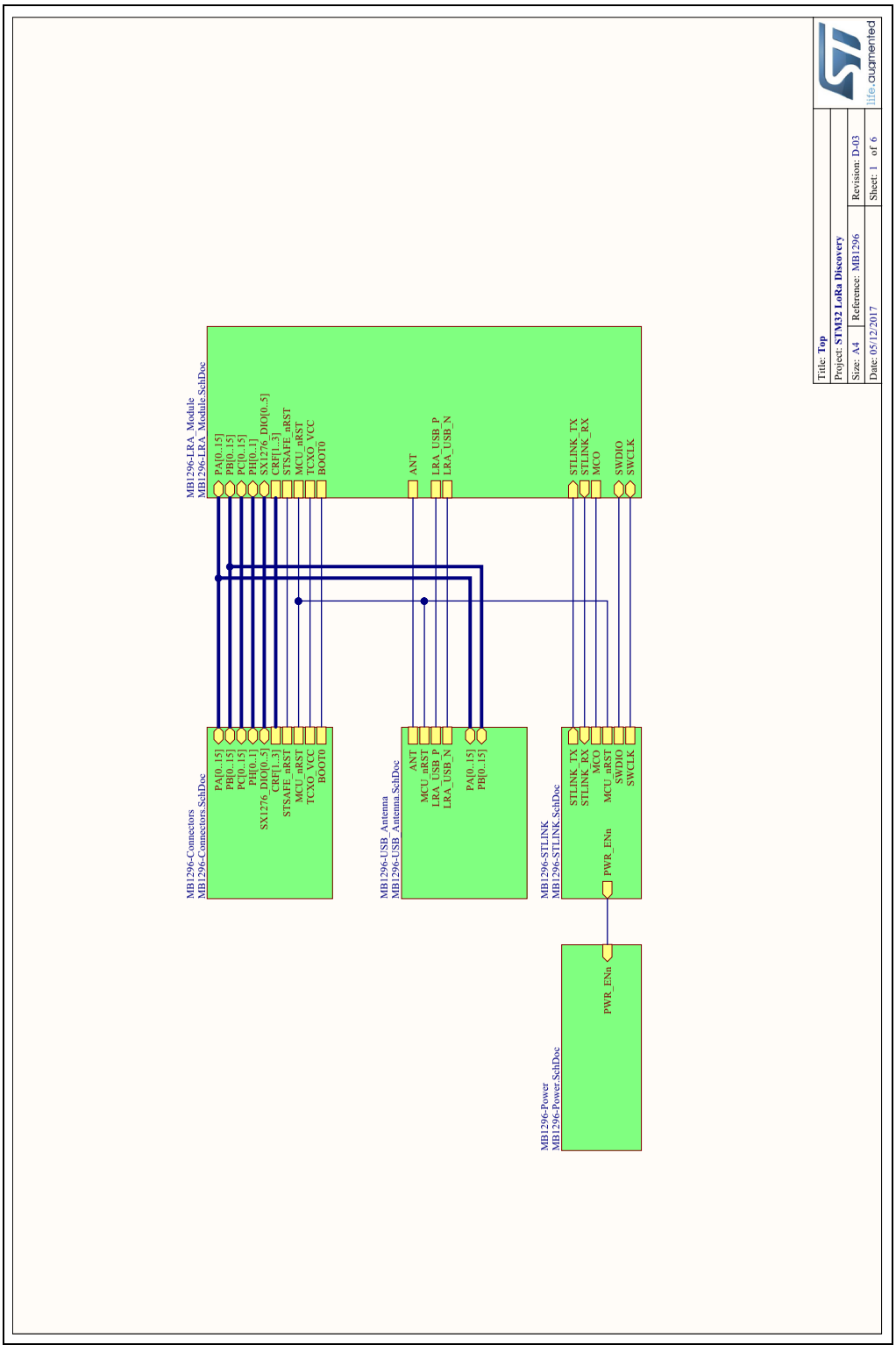


Figure 11. Power

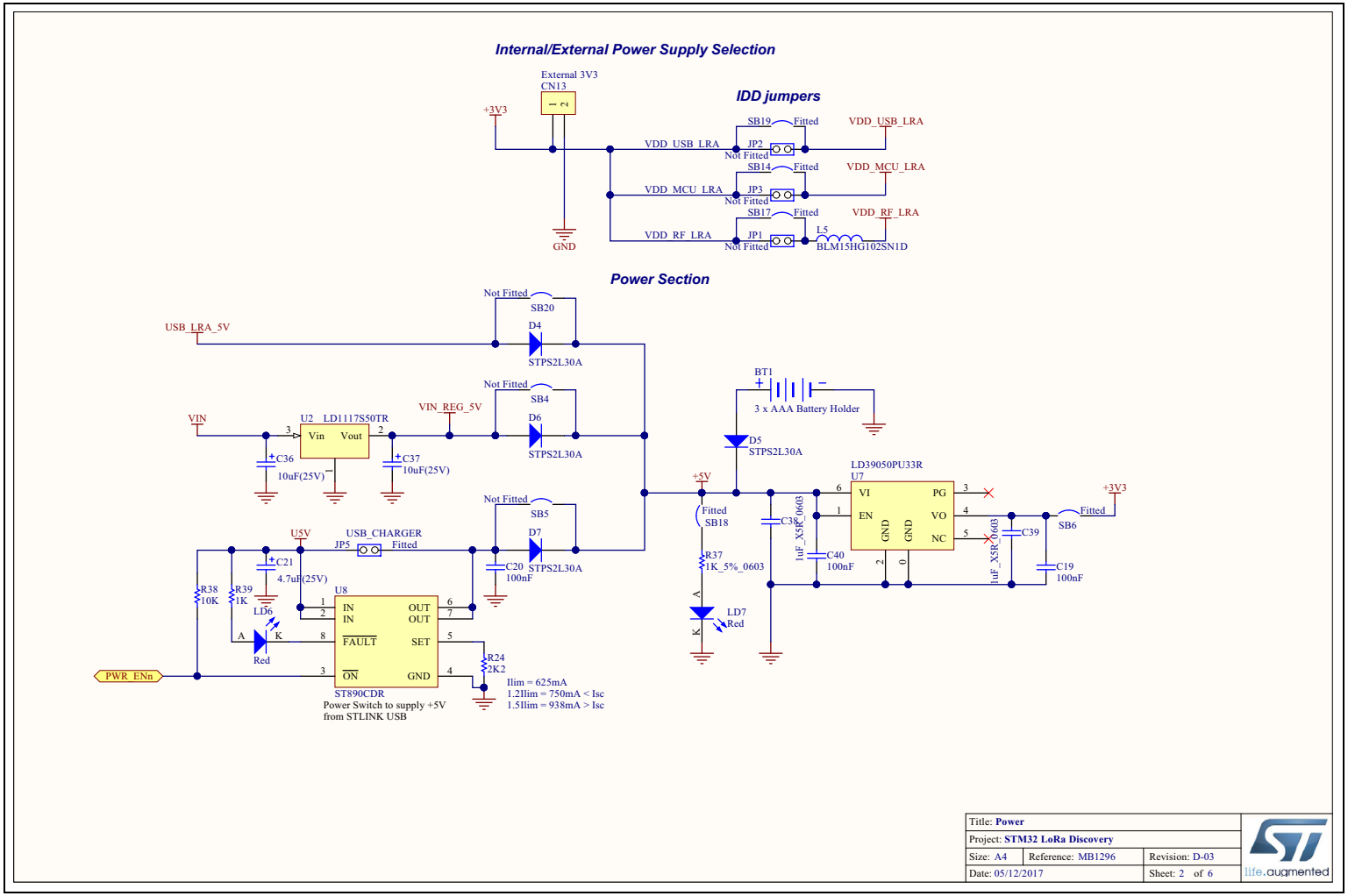


Figure 12. ST-LINK/V2-1

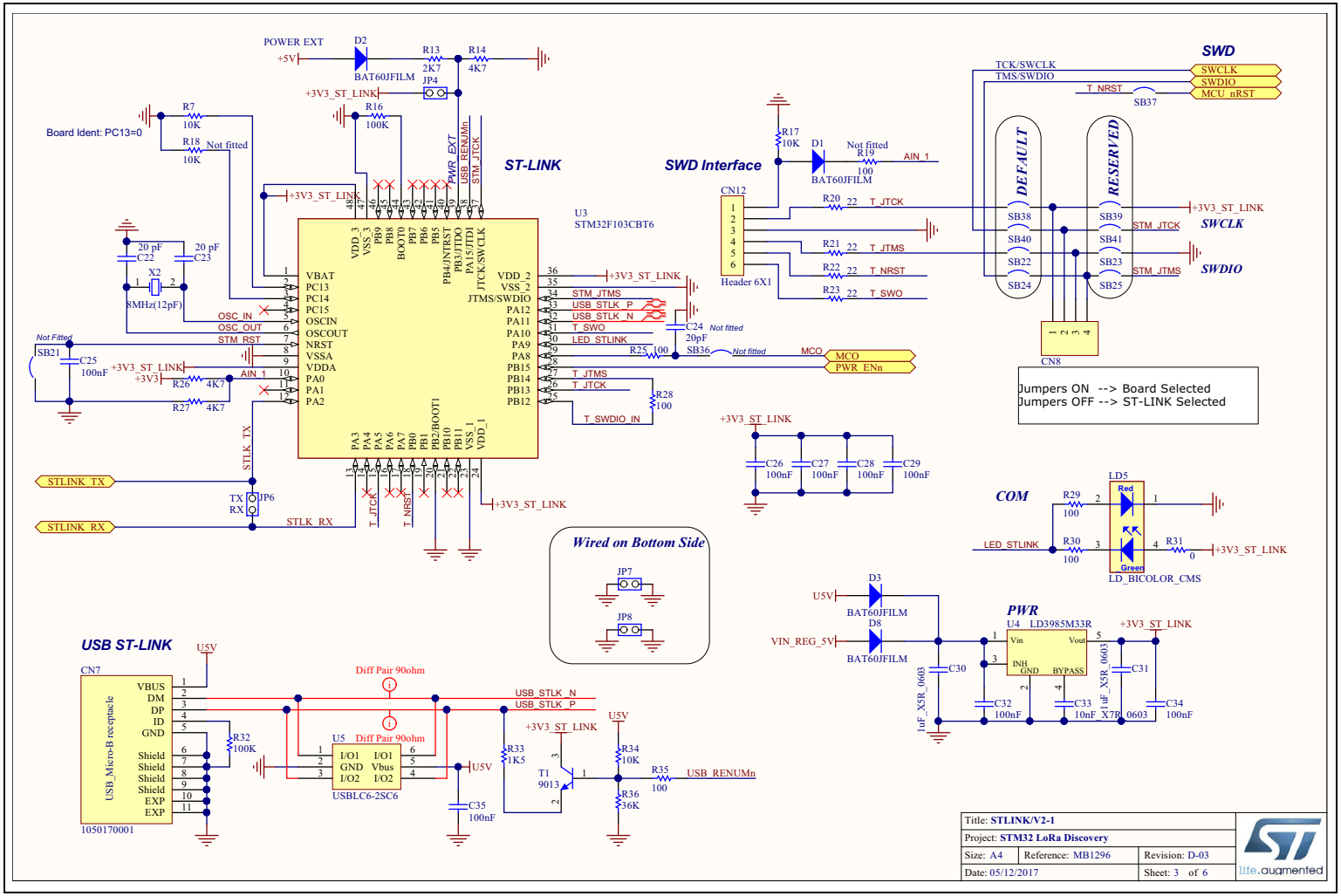


Figure 13. LoRa®/Sigfox™ module

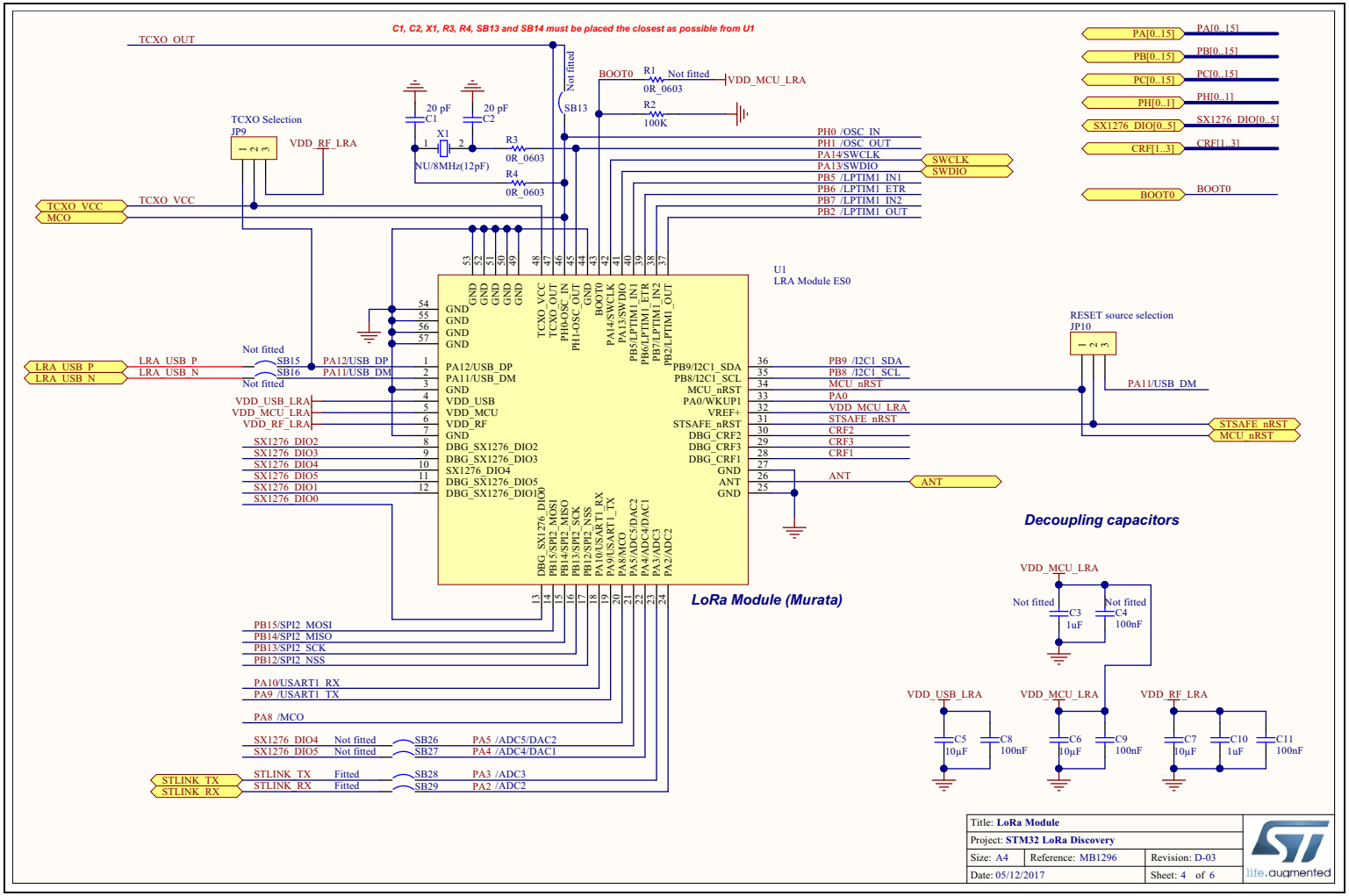


Figure 14. USB 2.0 FS and antenna

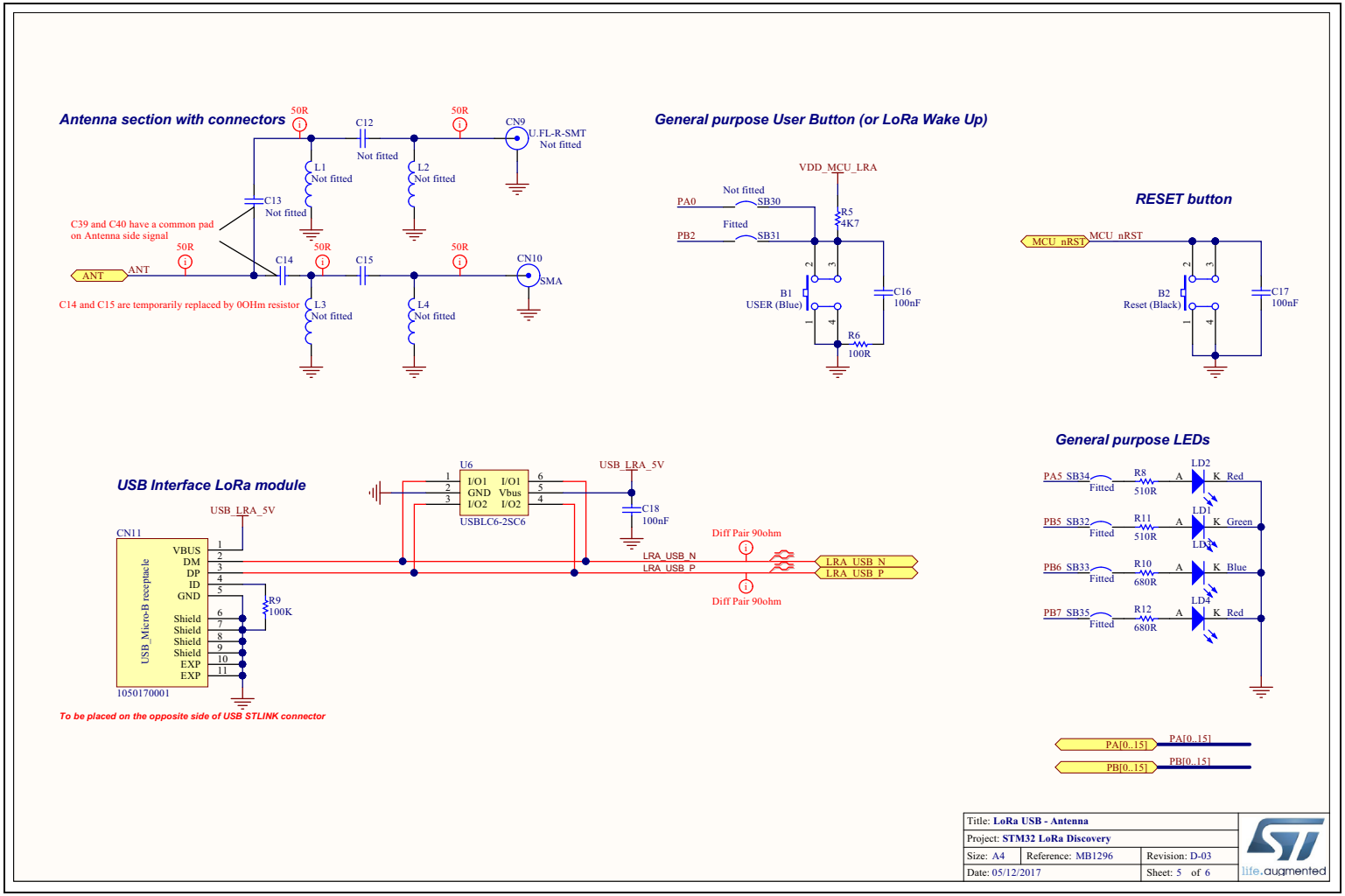
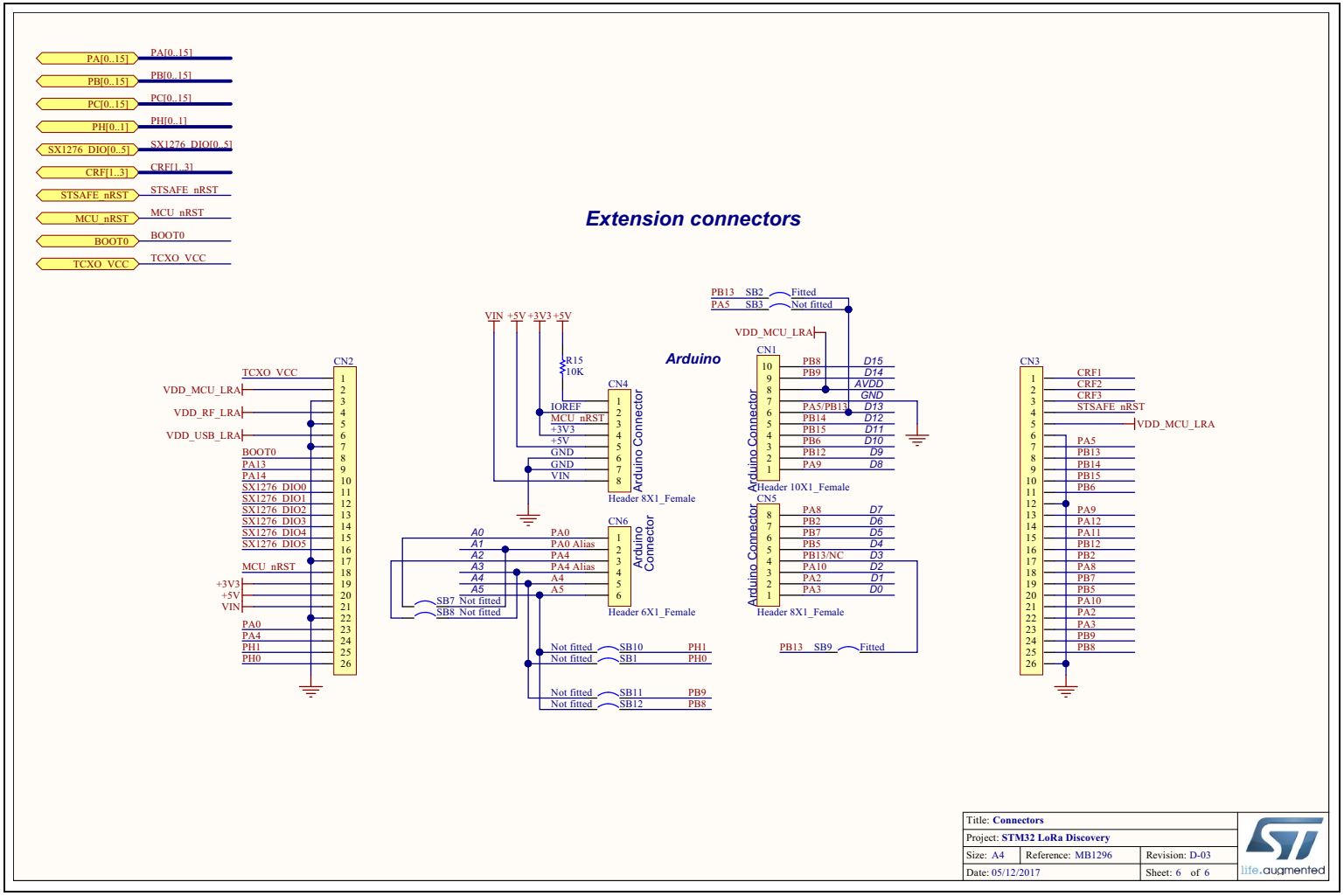


Figure 15. Connectors



3. Hoja de características de HDC2080

HDC2080 Low-Power Humidity and Temperature Digital Sensor

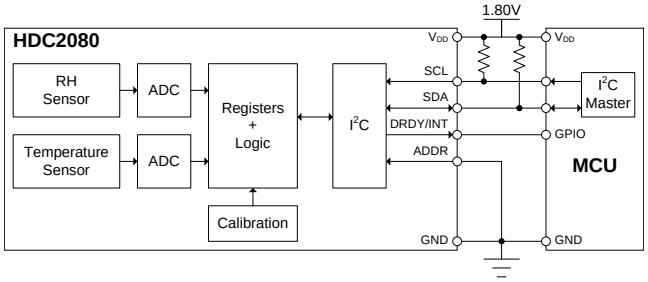
1 Features

- Relative humidity range: 0% to 100%
- Humidity accuracy: ±2% (typical), ±3% (maximum)
- Temperature accuracy: ±0.2°C (typical), ±0.4°C (maximum)
- Sleep mode current: 50 nA (typical), 100 nA (maximum)
- Average supply current (1 measurement/second)
  - 300 nA: RH% only (11 bit)
  - 550 nA: RH% (11 bit) + temperature (11 bit)
- Temperature range:
  - Operating: –40°C to 85°C
  - Functional: –40°C to 125°C
- Supply voltage range: 1.62 V to 3.6 V
- Available auto measurement mode
- I^2C interface compatibility

2 Applications

- Smart thermostats
- Smart home assistants
- Washer/dryers
- HVAC systems
- Inkjet printers

Typical Application



3 Description

The HDC2080 device is an integrated humidity and temperature sensor that provides high accuracy measurements with very low power consumption in a small DFN package. The capacitive-based sensor includes new integrated digital features and a heating element to dissipate condensation and moisture. The HDC2080 digital features include programmable interrupt thresholds to provide alerts and system wake-ups without requiring a microcontroller to be continuously monitoring the system. Combined with programmable sampling intervals, a low power consumption, and a support for a 1.8-V supply voltage, the HDC2080 is designed for battery-operated systems.

The HDC2080 provides high accuracy measurement capability for a wide range of environmental monitoring and Internet of Things (IoT) applications such as smart thermostats and smart home assistants. For designs where printed-circuit board (PCB) area is critical, a smaller CSP package option is available thru the HDC2010 with complete software compatibility with the HDC2080.

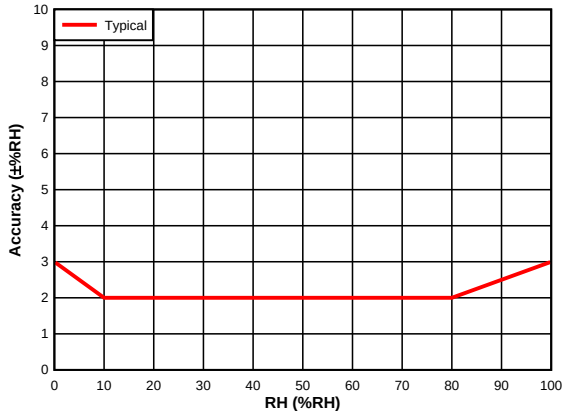
For applications with strict power-budget restrictions, Auto Measurement Mode enables the HDC2080 to automatically initiate temperature and humidity measurements. This feature allows users to configure a microcontroller into deep sleep mode because the HDC2080 is no longer dependent upon the microcontroller to initiate a measurement.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
HDC2080	WSON (6)	3.00 mm × 3.00 mm

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

RH Accuracy (T_A = 30°C)



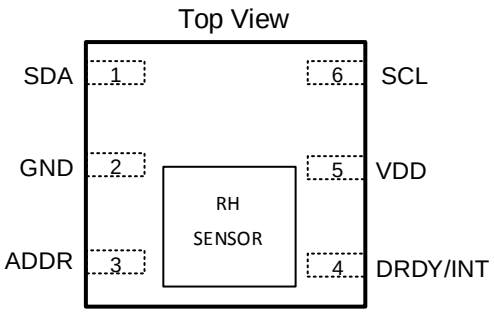
5 Description (continued)

Programable temperature and humidity thresholds in the HDC2080 allow the device to send a hardware interrupt to wake up the microcontroller when necessary. In addition, the power consumption of the HDC2080 is significantly reduced, which helps to minimize self-heating and improve measurement accuracy.

The HDC2080 is factory-calibrated to 0.2°C temperature accuracy and 2% relative humidity accuracy.

6 Pin Configuration and Functions

DMB Package
6-Pin PWSON
Top View



Pin Functions

PIN		I/O	DESCRIPTION
NAME	NO.		
SDA	1	I/O	Serial data line for I^2C, open-drain; requires a pullup resistor to V_{DD}
GND	2	G	Ground
ADDR	3	I	Address select pin – leave unconnected or hardwired to V_{DD} or GND. Unconnected slave address: 1000000 GND: slave address: 1000000 V_{DD}: slave address: 1000001
DRDY/INT	4	O	Data ready/Interrupt. Push-pull output
V_{DD}	5	P	Positive Supply Voltage
SCL	6	I	Serial clock line for I^2C, open-drain; requires a pullup resistor to V_{DD}

7 Specifications

7.1 Absolute Maximum Ratings(1)

		MIN	MAX	UNIT
VDD	Input Voltage	-0.3	3.9	V
GND	Input Voltage	-0.3	3.9	V
ADDR	Input Voltage	-0.3	3.9	V
SCL	Input Voltage	-0.3	3.9	V
SDA	Input Voltage	-0.3	3.9	V
T_{stg}	Storage temperature	-65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Rating* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Condition*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

7.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human body model (HBM), per ANSI/ESDA/JEDEC JS-001, all pins(1)	±2000	V
		Charged device model (CDM), per JEDEC specification JESD22-C101, all pins(2)	±500	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

7.3 Recommended Operating Conditions

over operating range (unless otherwise noted)

		MIN	NOM	MAX	UNIT
V_{DD}	Voltage Supply	1.62		3.6	V
T_{TEMP}	Temperature Sensor - Operating free-air temperature	-40		125	°C
T_{RH}	Relative Humidity Sensor - Operating free-air temperature	-20		70	°C
T_{HEATER}	Integrated Heater - Operating free-air temperature	-40		85	°C

7.4 Thermal Information

THERMAL METRIC(1)		HDC2080 PWSON (DMB) 6 PINS	UNIT
$R_{\theta JA}$	Junction-to-ambient thermal resistance	56.4	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	73.6	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	24.0	°C/W
Ψ_{JT}	Junction-to-top characterization parameter	3.8	°C/W
Ψ_{JB}	Junction-to-board characterization parameter	24.0	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	13.0	°C/W

(1) For more information about traditional and new thermal metrics, see the Semiconductor and IC Package Thermal Metrics application report.

7.5 Electrical Characteristics

at T_A = 30°C, V_{DD} = 1.8 V, 20% ≤ RH ≤ 80% (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
ELECTRICAL SPECIFICATION						
V_{DD}	Supply Voltage	Operating Range	1.62		3.6	V
I_{DD}	Supply current	RH measurement(1)		650	890	µA

(1) I2C read/write communication and pull up resistors current through SCL, SDA not included.

Electrical Characteristics (continued)

at T_A = 30°C, V_{DD} = 1.8 V, 20% ≤ RH ≤ 80% (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD}	Supply current	Temperature measurement(1)		550	730	µA
I_{DD}	Supply current	Sleep Mode		0.05	0.1	µA
I_{DD}	Supply current	Average at 1 measurement/second, RH or temperature only(1) (2)		0.3		µA
I_{DD}	Supply current	Average at 1 measurement/second, RH (11 bit)+temperature (11 bit)(1) (2)		0.55		µA
I_{DD}	Supply current	Average at 1 measurement every 2 seconds, RH (11 bit) + temperature (11 bit)(1) (2)		0.3		µA
I_{DD}	Supply current	Average @ 1 measurement every 10 seconds, RH (11 bit)+temperature (11 bit)		0.105		µA
I_{DD}	Supply current	Startup (average on startup time)		80		µA
I_{DDHEAT}	Integrated Heater (when enabled)(3)	V_{DD} = 3.3 V and T_A = -40°C to 85°C		90		mA
RELATIVE HUMIDITY SENSOR						
RH_{ACC}	Accuracy(4) (5) (6)			±2	±3	%RH
RH_{REP}	Repeatability(7)	14 bit resolution		±0.1		%RH
RH_{HYS}	Hysteresis(8)			±1		%RH
RH_{RT}	Response Time(9)	$t_{63\%}$ step(10)		8		sec
RH_{CT}	Conversion-time(7)	9 bit accuracy		275		µs
RH_{CT}	Conversion-time(7)	11 bit accuracy		400		µs
RH_{CT}	Conversion-time(7)	14 bit accurcay		660		µs
RH_{OR}	Operating range	Non-condensing(11)	0		100	%RH
RH_{LTD}	Long-term Drift(12)			±0.25		%RH/yr
TEMPERATURE SENSOR						
$TEMP_{OR}$	Operating range		-40		125	°C
$TEMP_{ACC}$	Accuracy(7)	5°C < T_A < 60°C		±0.2	±0.4	°C
$TEMP_{REP}$	Repeatability(7)	14 bit resolution		±0.1		°C
$TEMP_{CT}$	Conversion-time(7)	9 bit accuracy		225		µs
$TEMP_{CT}$	Conversion-time(7)	11 bit accuracy		350		µs
$TEMP_{CT}$	Conversion-time(7)	14 bit accurcay		610		µs

(2) Average current consumption while conversion is in progress.
(3) Heater operating range: – 40°C to 85°C.
(4) Excludes hysteresis and long-term drift.
(5) Excludes the impact of dust, gas phase solvents and other contaminants such as vapors from packaging materials, adhesives, or tapes, etc.
(6) Limits apply over the humidity operating range 20 to 80% RH (non-condensing) from 0 to 60°C.
(7) This parameter is specified by design and/or characterization and is not tested in production.
(8) The hysteresis value is the difference between an RH measurement in a rising and falling RH environment, at a specific RH point.
(9) Actual response times will vary dependent on system thermal mass and air-flow.
(10) Time for the RH output to change by 63% of the total RH change after a step change in environmental humidity.
(11) Recommended humidity operating range is 20 to 80% RH (non-condensing) over 0 to 60°C. Prolonged operation beyond these ranges may result in a shift of sensor reading, with slow recovery time.
(12) Drift due to aging effects at typical conditions (30°C and 20% to 50% RH). This value may be impacted by dust, vaporized solvents, outgassing tapes, adhesives, packaging materials, etc.

7.6 I^2C Interface Electrical Characteristics

At T_A = 30°C, V_{DD} = 3.3 V (unless otherwise noted).

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{IH}	Input High Voltage		0.7 x V_{DD}			V
V_{IL}	Input Low Voltage				0.3 x V_{DD}	V

I²C Interface Electrical Characteristics (continued)

At T_A = 30°C, V_{DD} = 3.3 V (unless otherwise noted).

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OL}	Output Low Voltage	I_{OL} = 3 mA			0.4	V
HYS	Hysteresis		0.1 x V_{DD}			V
C_{IN}	Input Capacitance on all digital pins(1)			0.5		pF

(1) This parameter is specified by design and/or characterization and it is not tested in production.

7.7 I²C Interface Timing Requirements

At T_A = 30°C, V_{DD} = 1.8 V (unless otherwise noted); values are based on statistical analysis of samples tested during initial release

		MIN	TYP	MAX	UNIT
f_{SCL}	Clock Frequency(1)	10		400	kHz
t_{LOW}	Clock Low Time(1)	1.3			µs
t_{HIGH}	Clock High Time(1)	0.6			µs

(1) This parameter is specified by design and/or characterization and it is not tested in production.

7.8 Timing Diagram

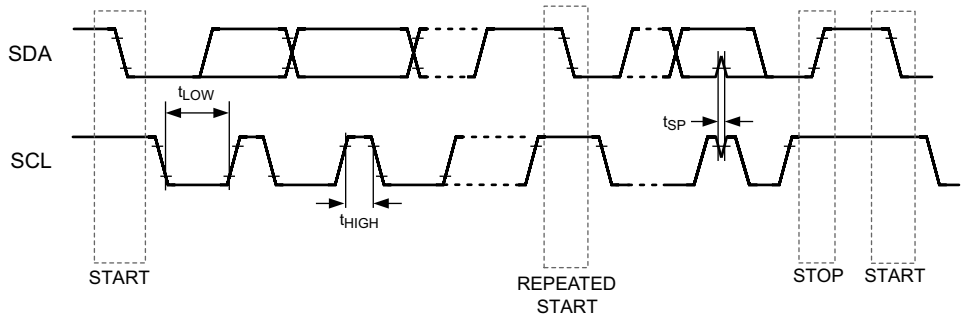


Figure 1. I2C Timing

7.9 Typical Characteristics

Unless otherwise noted. T_A = 30°C, V_{DD} = 1.80 V.

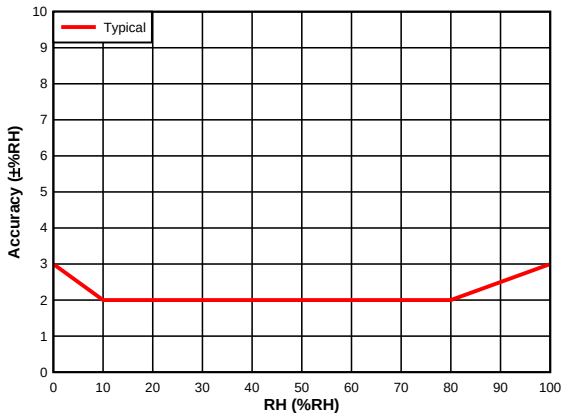


Figure 2. RH Accuracy vs. RH Set Point

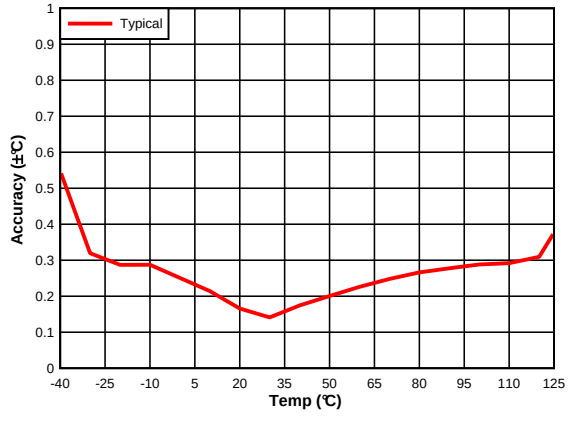


Figure 3. Temperature Accuracy vs. Temperature Set Point

Typical Characteristics (continued)

Unless otherwise noted. T_A = 30°C, V_{DD} = 1.80 V.

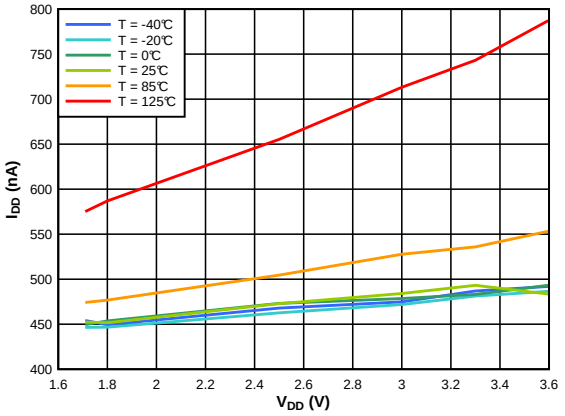


Figure 4. Supply Current vs. Supply Voltage, Average at 1 Measurement/Second, RH (11 Bit) and Temperature (11 Bit)

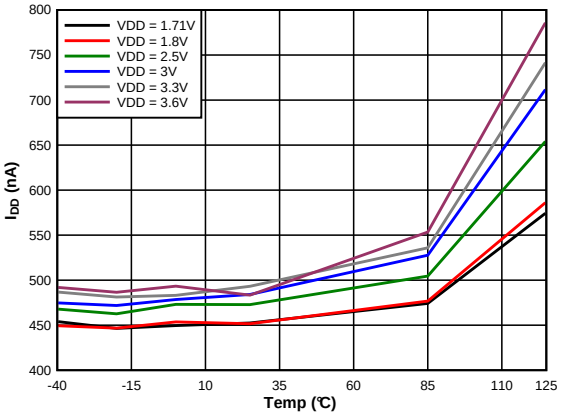


Figure 5. Supply Current vs. Temperature, Average at 1 Measurement/Second, RH (11 Bit) and Temperature (11 Bit)

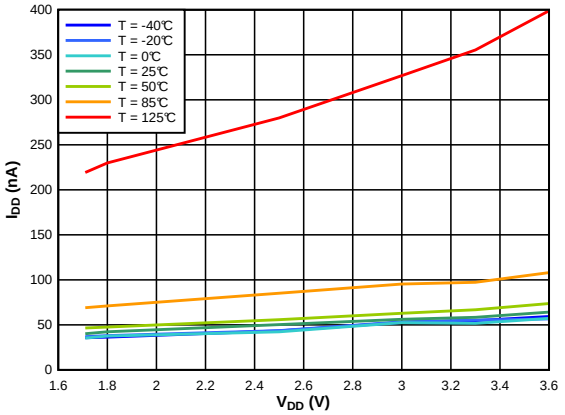


Figure 6. Supply Current vs. Supply Voltage, Sleep Mode

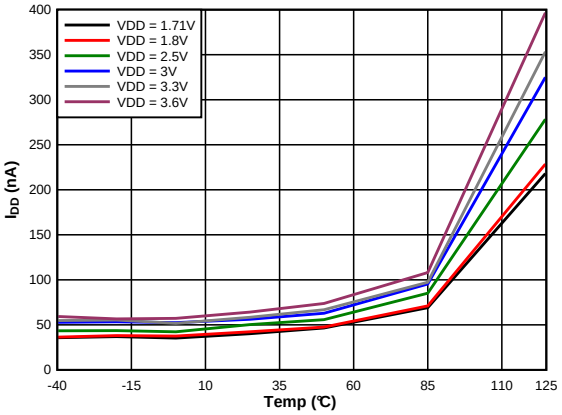


Figure 7. Supply Current vs. Temperature, Sleep Mode

8 Detailed Description

8.1 Overview

The HDC2080 is a highly integrated digital humidity and temperature sensor that incorporates both humidity-sensing and temperature-sensing elements, an analog-to-digital converter, calibration memory, and an I^2C interface that are all contained in a 3.00-mm × 3.00-mm 6-pin WSON package. The HDC2080 provides excellent measurement accuracy with very low power consumption and features programmable resolution for both humidity and temperature:

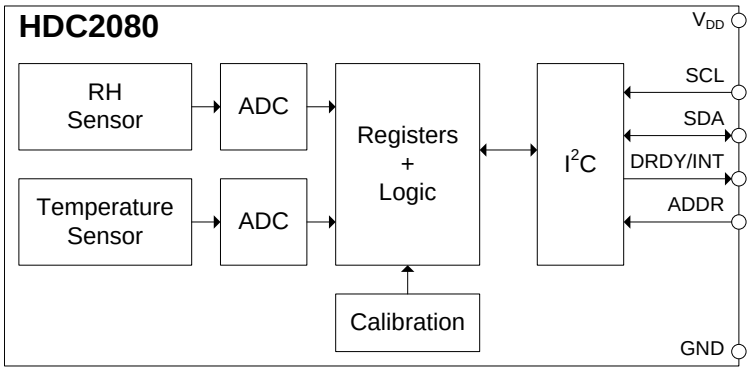
- Temperature resolution [9, 11, 14]
- Humidity resolution [9, 11, 14]

The conversion time during measurements is dependent upon the configured resolution for humidity and temperature, which can be configured for optimal power consumption.

The HDC2080 device incorporates a state-of-the-art polymer dielectric to provide capacitive-sensing measurements. As with most relative humidity sensors that include this type of technology, the user must meet certain application requirements to ensure optimal device performance for the sensing element. The user must:

- Follow the correct storage and handling procedures during board assembly. See *Humidity Sensor: Storage And Handling Guidelines* (SNIA025) for these guidelines.
- Protect the sensor from contaminants during board assembly and operation.
- Reduce prolonged exposure to both high temperature and humidity extremes that may impact sensor accuracy.
- Follow the correct layout guidelines for best performance. See *Optimizing Placement and Routing for Humidity Sensors* (SNAA297) for these guidelines.

8.2 Functional Block Diagram



8.3 Feature Description

8.3.1 Sleep Mode Power Consumption

One key feature of the HDC2080 is the low power consumption of the device, which makes the HDC2080 suitable in battery-powered or energy-harvesting applications. In these applications, the HDC2080 spends most of the time in sleep mode that has a typical current consumption of 50 nA. This minimizes the average power consumption and self-heating.

8.3.2 Measurement Modes: Trigger on Demand vs. Auto Measurement

Two types of measurement modes are available on the HDC2080: Trigger on Demand and Auto Mode.

Trigger on Demand is when each measurement reading are initiated through an I^2C command on an as-needed basis. After the measurement is converted, the device remains in sleep mode until another I^2C command is received.

8.4 Device Functional Modes

The HDC2080 has two modes of operation: Sleep Mode and Measurement Mode.

8.4.1 Sleep Mode vs. Measurement Mode

After power up, the HDC2080 defaults to Sleep Mode and waits for an I^2C instruction to set programmable conversion times, trigger a measurement or conversion, or read or write valid data. When a measurement is triggered, the HDC2080 switches to Measurement Mode that converts temperature or humidity values from integrated sensors through an internal ADC and stores the information in their respective data registers. The DRDY/INT pin can be monitored to verify if data is ready after measurement conversion. The DRDY/INT pin polarity and interrupt mode are set according to the configuration of the Interrupt Enable and DRDY/INT Configuration registers. After completing the conversion, the HDC2080 returns to Sleep Mode.

8.5 Programming

8.5.1 I^2C Serial Bus Address Configuration

To communicate with the HDC2080, the master must first address slave devices through a slave address byte. The slave address byte consists of seven address bits and a direction bit that indicates the intent to execute a read or write operation. The HDC2080 features an address pin to allow up to 2 devices to be addressed on a single bus. Table 1 describes the pin logic levels used to connect up to two devices. ADDR should be set before any activity on the interface occurs and remain constant while the device is powered up.

Table 1. HDC2080 I^2C Slave Address

ADDR	ADDRESS (7-BIT ADDRESS)
GND	1000000
VDD	1000001

8.5.2 I^2C Interface

The HDC2080 operates only as a slave device on the I^2C bus interface. It is not allowed to have multiple devices on the same I^2C bus with the same address. Connection to the bus is made through the open-drain I/O lines, SDA, and SCL. The SDA and SCL pins feature integrated spike-suppression filters and Schmitt triggers to minimize the effects of input spikes and bus noise. After power-up, the sensor needs at most 3 ms, to be ready to start RH and temperature measurement. After power-up the sensor is in sleep mode until a communication or measurement is performed. All data bytes are transmitted MSB first.

8.5.3 Serial Bus Address

To communicate with the HDC2080, the master must first address slave devices through a slave address byte. The slave address byte consists of seven address bits, and a direction bit that indicates the intent to execute a read or write operation.

8.5.4 Read and Write Operations

Address registers, which hold data pertaining to the status of the device, can be accessed through a pointer mechanism and can be accessed and modified with the following write and read procedures. The register address value is the first byte transferred after the device slave address byte with the R/W bit low. Every write operation to the HDC2080 requires a value for the register address (refer to Table 2).

When reading from the HDC2080, the current pointer location is used to determine which register is read by a read operation -- the pointer location points to the last written register address. To change the address for a read operation, a new value must be written to the pointer. This transaction is accomplished by issuing the slave address byte with the R/W bit set to '0', followed by the pointer byte. No additional data is required (refer to Table 4).

8.6 Register Maps

The HDC2080 contains data registers that hold configuration information, temperature and humidity measurement results, and status information.

Table 6. Register Map

ADDRESS (HEX)	NAME	RESET VALUE	DESCRIPTION
0x00	TEMPERATURE LOW	00000000	Temperature [7:0]
0x01	TEMPERATURE HIGH	00000000	Temperature [15:8]
0x02	HUMIDITY LOW	00000000	Humidity [7:0]
0x03	HUMIDITY HIGH	00000000	Humidity [15:8]
0x04	INTERRUPT/DRDY	00000000	DataReady and interrupt configuration
0x05	TEMPERATURE MAX	00000000	Maximum measured temperature (Not supported in Auto Measurement Mode)
0x06	HUMIDITY MAX	00000000	Maximum measured humidity (Not supported in Auto Measurement Mode)
0x07	INTERRUPT ENABLE	00000000	Interrupt Enable
0x08	TEMP_OFFSET_ADJUST	00000000	Temperature offset adjustment
0x09	HUM_OFFSET_ADJUST	00000000	Humidity offset adjustment
0x0A	TEMP_THR_L	00000000	Temperature Threshold Low
0x0B	TEMP_THR_H	11111111	Temperature Threshold High
0x0C	RH_THR_L	00000000	Humidity threshold Low
0x0D	RH_THR_H	11111111	Humidity threshold High
0x0E	RESET&DRDY/INT CONF	00000000	Soft Reset and Interrupt Configuration
0x0F	MEASUREMENT CONFIGURATION	00000000	Measurement configuration
0xFC	MANUFACTURER ID LOW	01001001	Manufacturer ID Low
0xFD	MANUFACTURER ID HIGH	01010100	Manufacturer ID High
0xFE	DEVICE ID LOW	11010000	Device ID Low
0xFF	DEVICE ID HIGH	00000111	Device ID High

8.6.1 Address 0x00 Temperature LSB

Table 7. Address 0x00 Temperature LSB Register

7	6	5	4	3	2	1	0
TEMP[7:0]							

Table 8. Address 0x00 Temperature LSB Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	TEMPERATURE [7:0]	R	00000000	Temperature LSB

8.6.2 Address 0x01 Temperature MSB

The temperature register is a 16-bit result register in binary format (the 2 LSBs D1 and D0 are always 0). The result of the acquisition is always a 14-bit value, while the resolution is related to one selected in Measurement Configuration register. The temperature must be read LSB first.

Table 9. Address 0x01 Temperature MSB Register

7	6	5	4	3	2	1	0
TEMP[15:8]							

Table 10. Address 0x01 Temperature MSB Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[15:8]	TEMPERATURE [15:8]	R	00000000	Temperature MSB

The temperature can be calculated from the output data with Equation 1:

$$\text{Temperature (°C)} = \left(\frac{\text{TEMPERATURE [15 : 0]}}{2^{16}} \right) \times 165 - 40 \quad (1)$$

8.6.3 Address 0x02 Humidity LSB

Table 11. Address 0x02 Humidity LSB Register

7	6	5	4	3	2	1	0
HUMIDITY[7:0]							

Table 12. Address 0x02 Humidity LSB Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	HUMIDITY [7:0]	R	00000000	Humidity LSB

8.6.4 Address 0x03 Humidity MSB

The humidity register is a 16-bit result register in binary format (the 2 LSBs D1 and D0 are always 0). The result of the acquisition is always a 14 bit value, while the resolution is related to one selected in Measurement Configuration register. The humidity measurement must be read LSB first.

Table 13. Address 0x03 Humidity MSB Register

7	6	5	4	3	2	1	0
HUMIDITY[15:8]							

Table 14. Address 0x03 Temperature MSB Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[15:8]	HUMIDITY[15:8]	R	00000000	Humidity MSB

The humidity can be calculated from the output data with Equation 2:

$$\text{Humidity (\%RH)} = \left(\frac{\text{HUMIDITY [15 : 0]}}{2^{16}} \right) \times 100 \quad (2)$$

8.6.9 Address 0x08 Temperature Offset Adjustment

Table 23. Address 0x08 Temperature Offset Adjustment Register

7	6	5	4	3	2	1	0
TEMP_OFFSET_ADJUST[7:0]							

Table 24. Address 0x08 Temperature Offset Adjustment Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	TEMP_OFFSET_ADJUST [7:0]	R/W	00000000	Temperature offset adjustment. Added to the converted Temperature value

The temperature can be adjusted adding the following values that are enable settings the equivalents bits:

7	6	5	4	3	2	1	0
–20.62°C	+10.32°C	+5.16°C	+2.58°C	+1.28°C	+0.64°C	+0.32°C	+0.16°C

The value is added to the converted temperature value for offset adjustment as shown in Figure 14

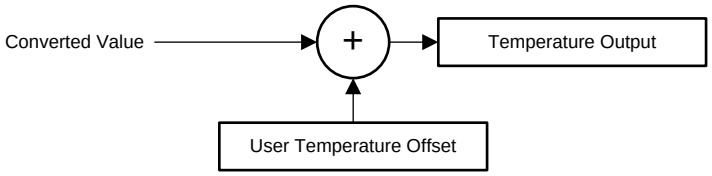


Figure 14. Temperature Output Calculation

The resulting temperature offset is a summation of the register bits that have been enabled (that is, programmed to 1). Some examples:

1. Programming TEMP_OFFSET_ADJUST to 00000001 adjusts the reported temperature by +0.16°C.
2. Programming TEMP_OFFSET_ADJUST to 00000111 adjusts the reported temperature by +1.12°C.
3. Programming TEMP_OFFSET_ADJUST to 00001101 adjusts the reported temperature by +2.08°C.
4. Programming TEMP_OFFSET_ADJUST to 11111111 adjusts the reported temperature by -0.16°C.
5. Programming TEMP_OFFSET_ADJUST to 11111001 adjusts the reported temperature by -1.12°C.
6. Programming TEMP_OFFSET_ADJUST to 11110011 adjusts the reported temperature by -2.08°C.

8.6.10 Address 0x09 Humidity Offset Adjustment

Table 25. Address 0x09 Humidity Offset Adjustment Register

7	6	5	4	3	2	1	0
HUM_OFFSET_ADJUST [7:0]							

Table 26. Address 0x09 Humidity Offset Adjustment Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	HUM_OFFSET_ADJUST [7:0]	R/W	00000000	Humidity offset adjustment. Added to the converted Humidity value

The humidity can be adjusted adding the following values that are enable settings the equivalents bits:

7	6	5	4	3	2	1	0
–25%RH	+12.5%RH	+6.3%RH	+3.1%RH	+1.6%RH	+0.8%RH	+0.4%RH	+0.2%RH

The value is added to the converted temperature value for offset adjustment as shown in Figure 15

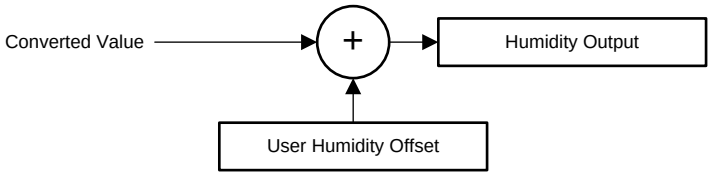


Figure 15. Humidity Output Calculation

The resulting humidity offset is a summation of the register bits that have been enabled (i.e. programmed to 1). Some examples:

1. Programming HUM_OFFSET_ADJUST to 00000001 adjusts the reported humidity by +0.20%RH.
2. Programming HUM_OFFSET_ADJUST to 00000101 adjusts the reported humidity by +1.00%RH.
3. Programming HUM_OFFSET_ADJUST to 00001010 adjusts the reported humidity by +2.00%RH.
4. Programming HUM_OFFSET_ADJUST to 11111111 adjusts the reported humidity by -0.10%RH.
5. Programming HUM_OFFSET_ADJUST to 11111011 adjusts the reported humidity by -0.90%RH.
6. Programming HUM_OFFSET_ADJUST to 11110101 adjusts the reported humidity by -2.10%RH.

8.6.11 Address 0x0A Temperature Threshold LOW

Table 27. Address 0x0A Temperature Threshold LOW Register

7	6	5	4	3	2	1	0
TEMP_THRES_LOW[7:0]							

Table 28. Address 0x0A Temperature Threshold LOW Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	TEMP_THRES_LOW[7:0]	R/W	00000000	Temperature threshold LOW value

The Temperature Threshold LOW can be calculated from the output data with Equation 5:

$$\text{Temperature threshold low (°C)} = \left(\frac{\text{TEMP\_THRES\_LOW [7 : 0]}}{2^8} \right) \times 165 - 40 \quad (5)$$

8.6.12 Address 0x0B Temperature Threshold HIGH

Table 29. Address 0x0B Temperature Threshold HIGH Register

7	6	5	4	3	2	1	0
TEMP_THRES_HIGH[7:0]							

Table 30. Address 0x0B Temperature Threshold HIGH Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	TEMP_THRES_HIGH[7:0]	R/W	11111111	Temperature threshold HIGH value

The Temperature Threshold HIGH can be calculated from the output data with Equation 6:

$$\text{Temperature threshold high (°C)} = \left(\frac{\text{TEMP\_THRES\_HIGH [7 : 0]}}{2^8} \right) \times 165 - 40 \quad (6)$$

8.6.13 Address 0x0C Humidity Threshold LOW

Table 31. Address 0x0C Humidity Threshold LOW Register

7	6	5	4	3	2	1	0
HUMI_THRES_LOW[7:0]							

Table 32. Address 0x0C Humidity Threshold LOW Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	HUMI_THRES_LOW[7:0]	R/W	00000000	Humidity threshold LOW value

The Humidity Threshold LOW can be calculated from the output data with Equation 7:

$$\text{Humidity threshold low (\%RH)} = \left(\frac{\text{HUMI\_THRES\_LOW}[7 : 0]}{2^8} \right) \times 100 \quad (7)$$

8.6.14 Address 0x0D Humidity Threshold HIGH

Table 33. Address 0x0D Humidity Threshold HIGH Register

7	6	5	4	3	2	1	0
HUMI_THRES_HIGH[7:0]							

Table 34. Address 0x0D Humidity Threshold HIGH Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	HUMI_THRES_HIGH[7:0]	R/W	11111111	Humidity threshold HIGH value

The Humidity Threshold HIGH can be calculated from the output data with Equation 8:

$$\text{Humidity threshold high (\%RH)} = \left(\frac{\text{HUMI\_THRES\_HIGH [7 : 0]}}{2^8} \right) \times 100 \quad (8)$$

8.6.15 Address 0x0E Reset and DRDY/INT Configuration Register

Table 35. Address 0x0E Configuration Register

7	6	5	4	3	2	1	0
SOFT_RES	AMM[2]	AMM[1]	AMM[0]	HEAT_EN	DRDY/INT_EN	INT_POL	INT_MODE

Table 36. Address 0x0E Configuration Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
7	SOFT_RES	R/W	0	0 = Normal Operation mode, this bit is self-clear 1 = Soft Reset EEPROM value reload and registers reset
[6:4]	AMM[2:0]	R/W	000	Auto Measurement Mode (AMM) 000 = Disabled. Initiate measurement via I^2C 001 = 1/120Hz (1 samples every 2 minutes) 010 = 1/60Hz (1 samples every minute) 011 = 0.1Hz (1 samples every 10 seconds) 100 = 0.2 Hz (1 samples every 5 second) 101 = 1Hz (1 samples every second) 110 = 2Hz (2 samples every second) 111 = 5Hz (5 samples every second)
3	HEAT_EN	R/W	0	0 = Heater off 1 = Heater on
2	DRDY/INT_EN	R/W	0	DRDY/INT_EN pin configuration 0 = High Z 1 = Enable
1	INT_POL	R/W	0	Interrupt polarity 0 = Active Low 1 = Active High
0	INT_MODE	R/W	0	Interrupt mode 0 = Level sensitive 1 = Comparator mode

8.6.16 Address 0x0F Measurement Configuration

Table 37. Address 0x0F Measurement Configuration Register

7	6	5	4	3	2	1	0
TRES[1]	TRES[0]	HRES[1]	HRES[0]	RES	MEAS_CONF[1]	MEAS_CONF[0]	MEAS_TRIG

Table 38. Address 0x0F Measurement Configuration Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
7:6	TRES[1:0]	R/W	00	Temperature resolution 00: 14 bit 01: 11 bit 10: 9 bit 11: NA
5:4	HRES[1:0]	R/W	00	Humidity resolution 00: 14 bit 01: 11 bit 10: 9 bit 11: NA
3	RES	R/W	0	Reserved
2:1	MEAS_CONF[1:0]	R/W	00	Measurement configuration 00: Humidity + Temperature 01: Temperature only 10: NA 11: NA
0	MEAS_TRIG	R/W	0	Measurement trigger 0: no action 1: Start measurement Self-clearing bit when measurement completed

8.6.17 Manufacturer ID Low

Table 39. Manufacturer ID Low Register

7	6	5	4	3	2	1	0
MANUFACTURER ID[7:0]							

Table 40. Address 0xFC Manufacturer ID Low Field Descriptions

BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	MANUFACTURER ID [7:0]	R	01001001	Manufacturer ID LOW value

8.6.18 Manufacturer ID High

These registers contain a factory-programmable identification value that identifies this device as being manufactured by Texas Instruments. These registers distinguish this device from other devices that are on the same I^2C bus. The manufacturer ID reads 0x4954

Table 41. Manufacturer ID High Register

7	6	5	4	3	2	1	0
MANUFACTURER ID[15:8]							

Table 42. Address 0xFD Manufacturer ID High Field Descriptions

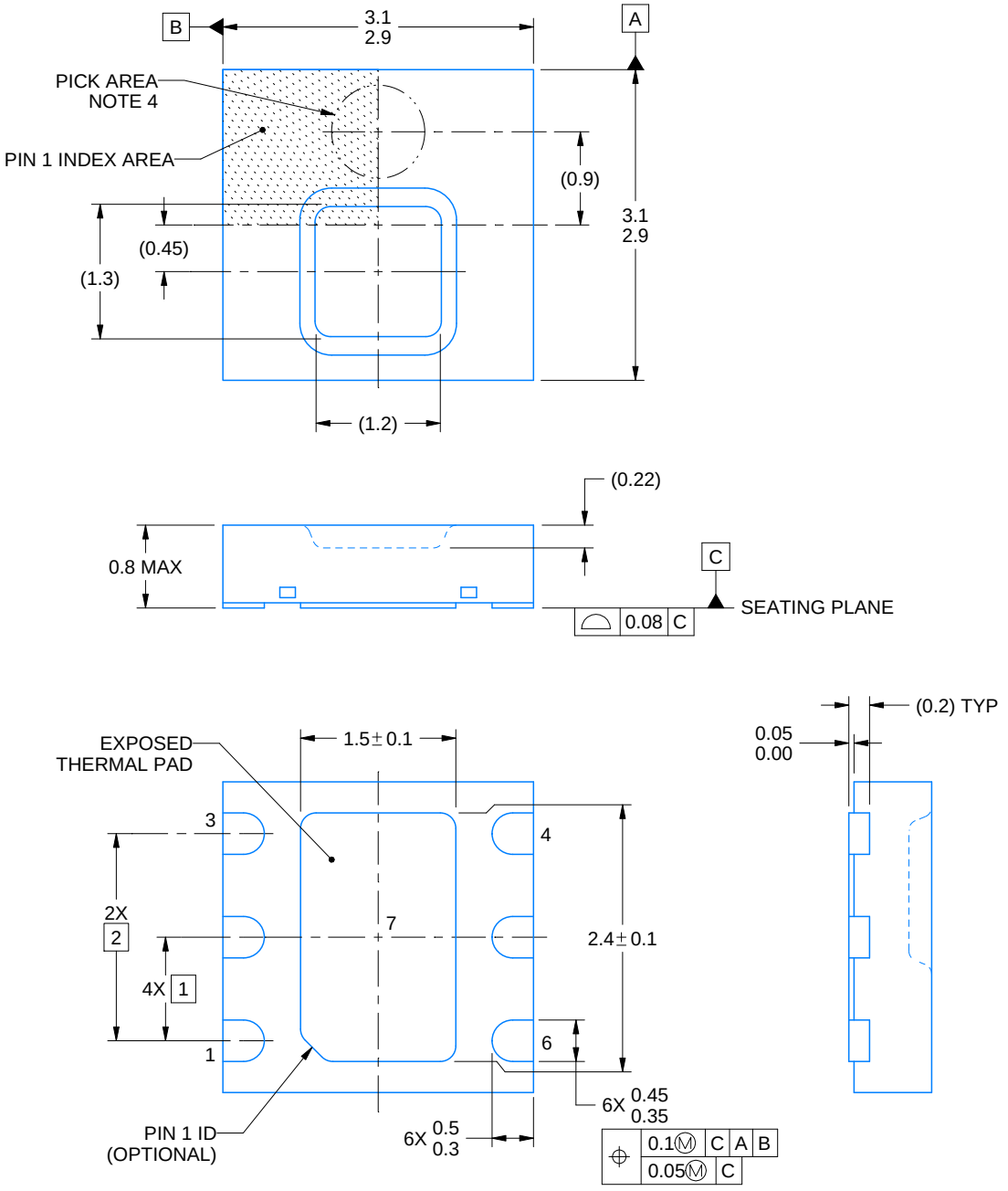
BIT	FIELD	TYPE	RESET	DESCRIPTION
[7:0]	MANUFACTURER ID [15:8]	R	01010100	Manufacturer ID HIGH value

PACKAGE OUTLINE

DMB0006A

WSON - 0.8 mm max height

PLASTIC SMALL OUTLINE - NO LEAD



4221225/C 12/2018

NOTES:

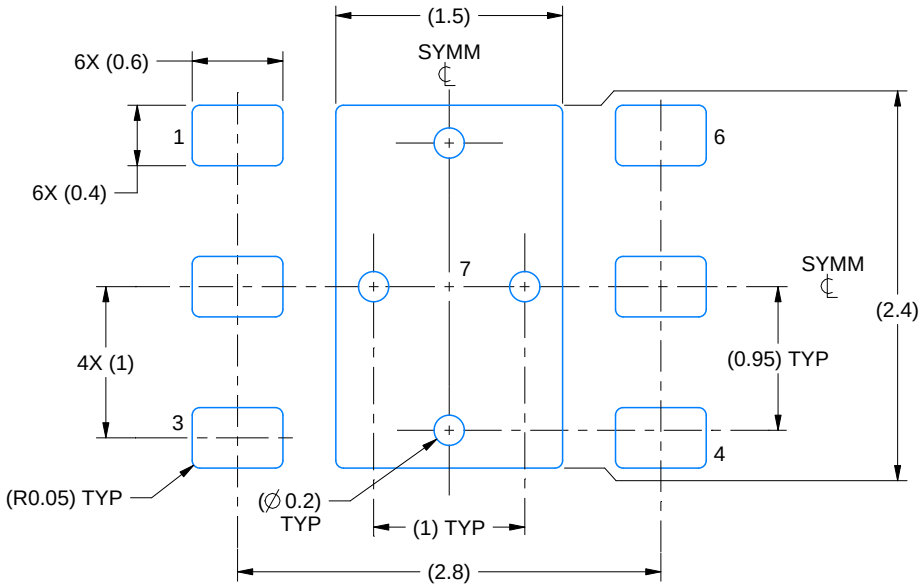
1. All linear dimensions are in millimeters. Any dimensions in parenthesis are for reference only. Dimensioning and tolerancing per ASME Y14.5M.
2. This drawing is subject to change without notice.
3. The package thermal pad must be soldered to the printed circuit board for thermal and mechanical performance.
4. Pick and place nozzle ∅ 0.9 mm or smaller recommended.

EXAMPLE BOARD LAYOUT

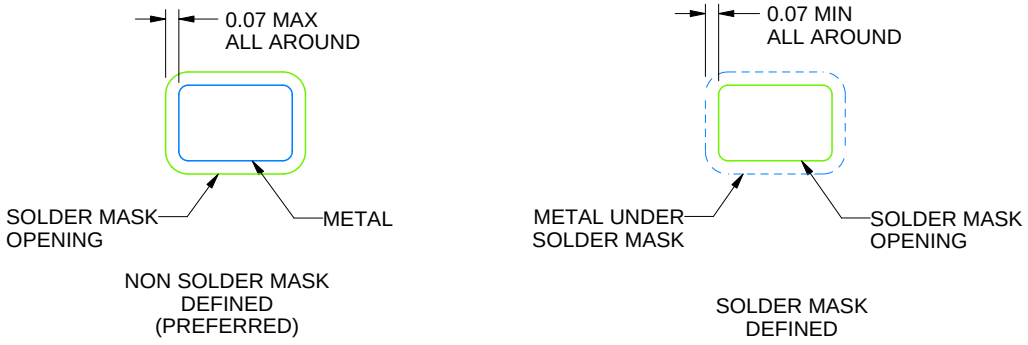
DMB0006A

WSON - 0.8 mm max height

PLASTIC SMALL OUTLINE - NO LEAD



LAND PATTERN EXAMPLE
SCALE:20X



SOLDER MASK DETAILS

4221225/C 12/2018

NOTES: (continued)

5. This package is designed to be soldered to a thermal pad on the board. For more information, see Texas Instruments literature number SLUA271 (www.ti.com/lit/slua271).
6. Vias are optional depending on application, refer to device data sheet. If any vias are implemented, refer to their locations shown on this view. It is recommended that vias under paste be filled, plugged or tented.

4. Hoja de características de TSL2591

TSL2591

Light-to-Digital Converter

General Description

The TSL2591 is a very-high sensitivity light-to-digital converter that transforms light intensity into a digital signal output capable of direct I^2C interface. The device combines one broadband photodiode (visible plus infrared) and one infrared-responding photodiode on a single CMOS integrated circuit. Two integrating ADCs convert the photodiode currents into a digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human eye response. The TSL2591 supports a traditional level style interrupt that remains asserted until the firmware clears it.

Ordering Information and Content Guide appear at end of datasheet.

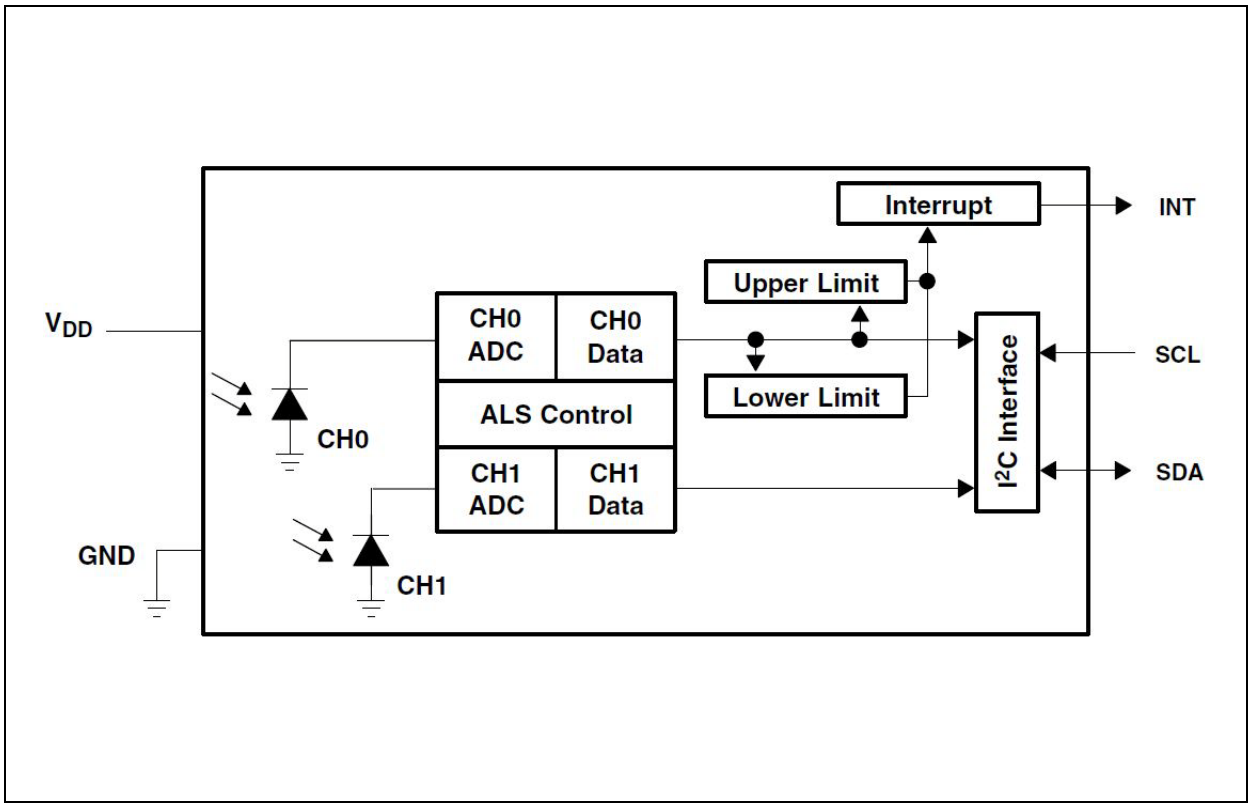
Figure 1:
Added Value of Using TSL2591

Benefits	Features
• Approximates Human Eye Response	• Dual Diode
• Flexible Operation	• Programmable Analog Gain and Integration Time
• Suited for Operation Behind Dark Glass	• 600M:1 Dynamic Range
• Low Operating Overhead	• Two Internal Interrupt Sources • Programmable Upper and Lower Thresholds • One Interrupt Includes Programmable Persistence Filter
• Low Power 3.0 µA Sleep State	• User Selectable Sleep Mode
• I^2C Fast Mode Compatible Interface	• Data Rates up to 400 kbit/s • Input Voltage Levels Compatible with 3.0V Bus

Block Diagram

The functional blocks of this device are shown below:

Figure 2:
Block Diagram



Pin Assignment

The TSL2591 pin assignments are described below.

Figure 3:
Pin Diagram

Package FN Dual Flat No-Lead (Top View): Package drawing is not to scale.

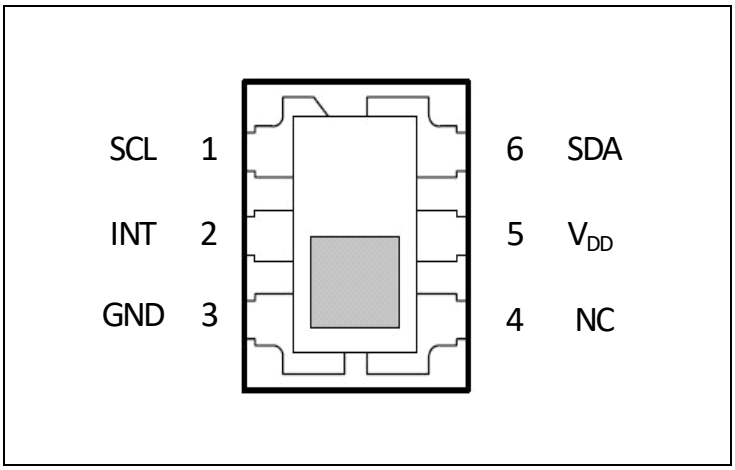


Figure 4:
Pin Description

Pin Number	Pin Name	Description
1	SCL	I^2C serial clock input terminal
2	INT	Interrupt — open drain output (active low).
3	GND	Power supply ground. All voltages are referenced to GND.
4	NC	No connect — do not connect.
5	V_{DD}	Supply voltage
6	SDA	I^2C serial data I/O terminal

Electrical Characteristics

All limits are guaranteed. The parameters with min and max values are guaranteed with production tests or SQC (Statistical Quality Control) methods. Device parameters are guaranteed at T_A = 25°C unless otherwise noted.

Figure 6:
Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Units
V_{DD}	Supply voltage	2.7	3	3.6	V
T_A	Operating free-air temperature	-30		70	°C

Figure 7:
Operating Characteristics, V_{DD}=3V, T_A=25°C (unless otherwise noted)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
I_{DD}	Supply current	Active Sleep state - no I^2C activity		275 2.3	325 4	µA
V_{OL}	INT, SDA output low voltage	3mA sink current 6mA sink current	0 0		0.4 0.6	V
I_{LEAK}	Leakage current, SDA, SCL, INT pins		-5		5	µA
V_{IH}	SCL, SDA input high voltage	TSL25911 ($V_{bus} = V_{DD}$)	0.7 V_{DD}			V
		TSL25913 (V_{bus} = 1.8)	1.26			
V_{IL}	SCL, SDA input low voltage	TSL25911 ($V_{bus} = V_{DD}$)			0.3 V_{DD}	V
		TSL25913 (V_{bus} = 1.8)			0.54	

Figure 8:
ALS Characteristics, V_{DD}=3V, T_A=25°C, AGAIN = High, AEN=1, (unless otherwise noted) (1) (2) (3)

Parameter	Conditions	Channel	Min	Typ	Max	Units
Dark ADC count value	E_e = 0, AGAIN = Max, ATIME=000b (100ms)	CH0 CH1	0 0		20 20	counts
ADC integration time step size	ATIME = 000b (100ms)		95	100	105	ms
ADC number of integration steps (4)			1		6	steps
Max ADC count	ATIME = 000b (100ms)		0		36863	counts
Max ADC count	ATIME = 001b (200ms), 010b (300ms), 011b (400ms), 100b (500ms), 101b (600ms)		0		65535	counts
ADC count value	White light (2) E_e = 4.98 µW/cm^2 ATIME = 000b (100 ms)	CH0 CH1	1120	1315 174	1510	counts
	λ_p = 850 nm (3) E_e = 5.62 µW/cm^2, ATIME = 000b (100 ms)	CH0 CH1	1230	1447 866	1665	counts
ADC count value ratio: CH1/CH0	White light (2)		0.092	0.132	0.172	
	λ_p = 850 nm (3)		0.558	0.598	0.638	
R_e Irradiance responsivity	White light (2) ATIME = 000b (100 ms)	CH0 CH1		264.1 34.9		counts/ (µW/cm^2)
	λ_p = 850 nm (3) ATIME = 000b (100 ms)	CH0 CH1		257.5 154.1		
Noise (4)	White light (2) E_e = 4.98 µW/cm^2 ATIME = 000b (100 ms)	CH0		1	2	1 standard deviation

Register Description

The device is controlled and monitored by registers accessed through the I²C serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Figure 15.

Figure 15:
Register Description

Address	Register Name	R/W	Register Function	Reset Value
--	COMMAND	W	Specifies Register Address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	CONFIG	R/W	ALS gain and integration time configuration	0x00
0x04	AILTL	R/W	ALS interrupt low threshold low byte	0x00
0x05	AILTH	R/W	ALS interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	ALS interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	ALS interrupt high threshold high byte	0x00
0x08	NPAILTL	R/W	No Persist ALS interrupt low threshold low byte	0x00
0x09	NPAILTH	R/W	No Persist ALS interrupt low threshold high byte	0x00
0x0A	NPAIHTL	R/W	No Persist ALS interrupt high threshold low byte	0x00
0x0B	NPAIHTH	R/W	No Persist ALS interrupt high threshold high byte	0x00
0x0C	PERSIST	R/W	Interrupt persistence filter	0x00
0x11	PID	R	Package ID	--
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	C0DATAL	R	CH0 ADC low data byte	0x00
0x15	C0DATAH	R	CH0 ADC high data byte	0x00
0x16	C1DATAL	R	CH1 ADC low data byte	0x00
0x17	C1DATAH	R	CH1 ADC high data byte	0x00

Note(s):

1. Devices with a primary I²C address of 0x29 also have a secondary I²C address of 0x28 that can be used for read only registers to quickly read in a single block I²C transaction.

Command Register

The COMMAND register specifies the address of the target register for future read and write operations, as well as issues special function commands.

7	6	5	4	3	2	1	0
CMD	TRANSACTION		ADDR/SF				

Fields	Bits	Description	
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.	
TRANSACTION	6:5	Select type of transaction to follow in subsequent data transfers	
		FIELD VALUE	**DESCRIPTION**
		00	Reserved - Do not use
		01	Normal Operation
		10	Reserved – Do not use
		11	Special Function – See description below
ADDR/SF	4:0	Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below apply only to special function commands.	
		FIELD VALUE	**DESCRIPTION**
		00100	Interrupt set – forces an interrupt
		00110	Clears ALS interrupt
		00111	Clears ALS and no persist ALS interrupt
		01010	Clears no persist ALS interrupt
		other	Reserved – Do not write
		The interrupt set special function command sets the interrupt bits in the status register (0x13). For the interrupt to be visible on the INT pin, one of the interrupt enable bits in the enable register (0x00) must be asserted. The interrupt set special function must be cleared with an interrupt clear special function. The ALS interrupt clear special functions clear any pending interrupt(s) and are self-clearing.	

Enable Register (0x00)

The ENABLE register is used to power the device on/off, enable functions and interrupts.

7	6	5	4	3	2	1	0
NPIEN	SAI	Reserved	AIEN	Reserved		AEN	PON

Fields	Bits	Description
NPIEN	7	No Persist Interrupt Enable. When asserted NP Threshold conditions will generate an interrupt, bypassing the persist filter.
SAI	6	Sleep after interrupt. When asserted, the device will power down at the end of an ALS cycle if an interrupt has been generated.
Reserved	5	Reserved. Write as 0.
AIEN	4	ALS Interrupt Enable. When asserted permits ALS interrupts to be generated, subject to the persist filter.
Reserved	3:2	Reserved. Write as 0.
AEN	1	ALS Enable. This field activates ALS function. Writing a one activates the ALS. Writing a zero disables the ALS.
PON	0	Power ON. This field activates the internal oscillator to permit the timers and ADC channels to operate. Writing a one activates the oscillator. Writing a zero disables the oscillator.

Control Register (0x01)

The CONTROL register is used to configure the ALS gain and integration time. In addition, a system reset is provided. Upon power up, the CONTROL register resets to 0x00.

7	6	5	4	3	2	1	0
SRESET	Reserved	AGAIN		Reserved	ATIME		

Fields	Bits	Description		
SRESET	7	System reset. When asserted, the device will reset equivalent to a power-on reset. SRESET is self-clearing.		
Reserved	6	Reserved. Write as 0.		
AGAIN	5:4	ALS gain sets the gain of the internal integration amplifiers for both photodiode channels.		
		FIELD VALUE	**DESCRIPTION**	
		00	Low gain mode	
		01	Medium gain mode	
		10	High gain mode	
		11	Maximum gain mode	
Reserved	3	Reserved. Write as 0.		
ATIME	2:0	ALS time sets the internal ADC integration time for both photodiode channels.		
		FIELD VALUE	**INTEGRATION TIME**	**MAX COUNT**
		000	100 ms	36863
		001	200 ms	65535
		010	300 ms	65535
		011	400 ms	65535
		100	500 ms	65535
		101	600 ms	65535

ALS Interrupt Threshold Register (0x04 – 0x0B)

The ALS interrupt threshold registers provide the values to be used as the high and low trigger points for the comparison function for interrupt generation. If C0DATA crosses below the low threshold specified, or above the higher threshold, an interrupt is asserted on the interrupt pin.

If the C0DATA exceeds the persist thresholds (registers: 0x04 – 0x07) for the number of persist cycles configured in the PERSIST register an interrupt will be triggered. If the C0DATA exceeds the no-persist thresholds (registers: 0x08 – 0x0B) an interrupt will be triggered immediately following the end of the current integration.

Note that while the interrupt is observable in the STATUS register (0x13), it is visible only on the INT pin when AIEN or NPIEN are enabled in the ENABLE register (0x00).

Upon power up, the interrupt threshold registers default to 0x00.

Register	Address	Bits	Description
AILTL	0x04	7:0	ALS low threshold lower byte
AILTH	0x05	7:0	ALS low threshold upper byte
AIHTL	0x06	7:0	ALS high threshold lower byte
AIHTH	0x07	7:0	ALS high threshold upper byte
NPAILTL	0x08	7:0	No Persist ALS low threshold lower byte
NPAILTH	0x09	7:0	No Persist ALS low threshold upper byte
NPAIHTL	0x0A	7:0	No Persist ALS high threshold lower byte
NPAIHTH	0x0B	7:0	No Persist ALS high threshold upper byte

ALS Data Register (0x14 - 0x17)

ALS data is stored as two 16-bit values; one for each channel. When the lower byte of either channel is read, the upper byte of the same channel is latched into a shadow register. The shadow register ensures that both bytes are the result of the same ALS integration cycle, even if additional integration cycles occur between the lower byte and upper byte register readings.

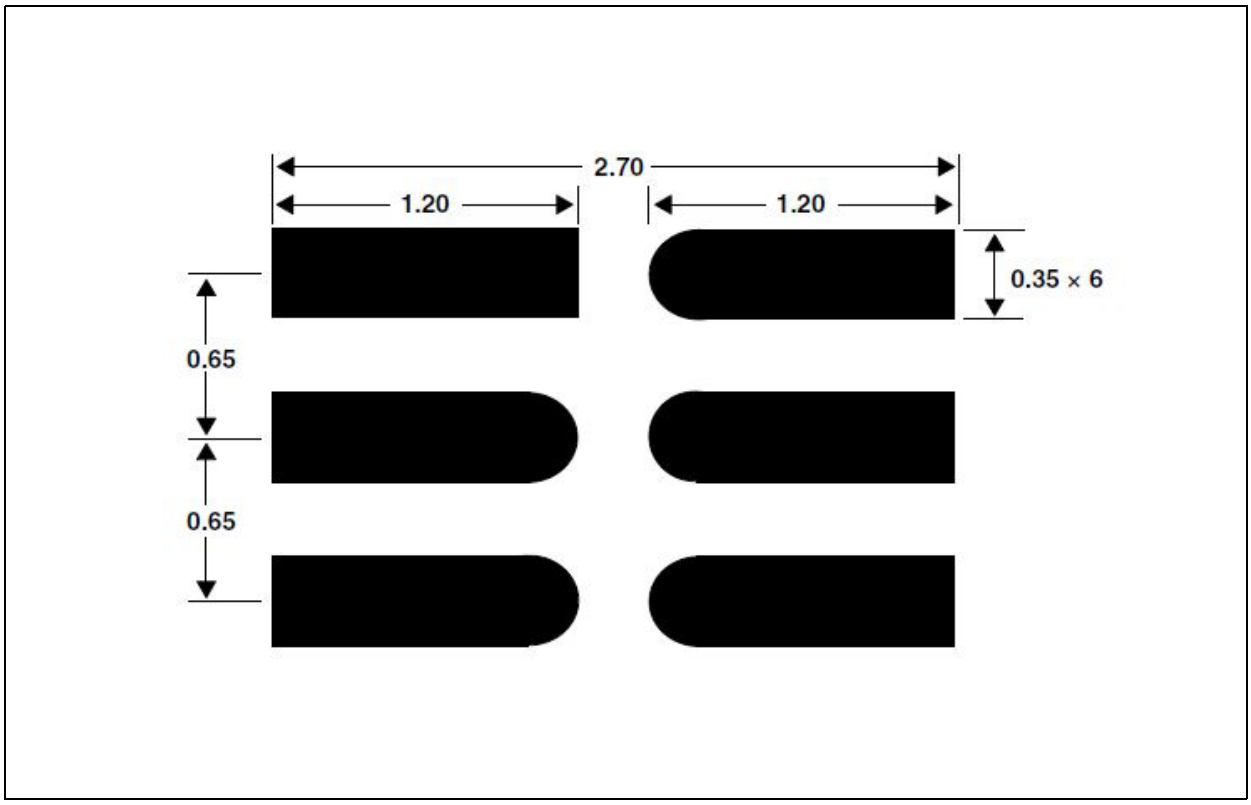
Each channel independently operates the upper byte shadow register. So to minimize the potential for skew between CH0 and CH1 data, it is recommended to read all four ADC bytes in sequence.

Register	Address	Bits	Description
C0DATAL	0x14	7:0	ALS CH0 data low byte
C0DATAH	0x15	7:0	ALS CH0 data high byte
C1DATAL	0x16	7:0	ALS CH1 data low byte
C1DATAH	0x17	7:0	ALS CH1 data high byte

PCB Pad Layout

Suggested land pattern based on the IPC−7351B Generic Requirements for Surface Mount Design and Land Pattern Standard (2010) for the small outline no-lead (SON) package is shown in Figure 17.

Figure 17:
Suggested FN Package PCB Layout (Top View)

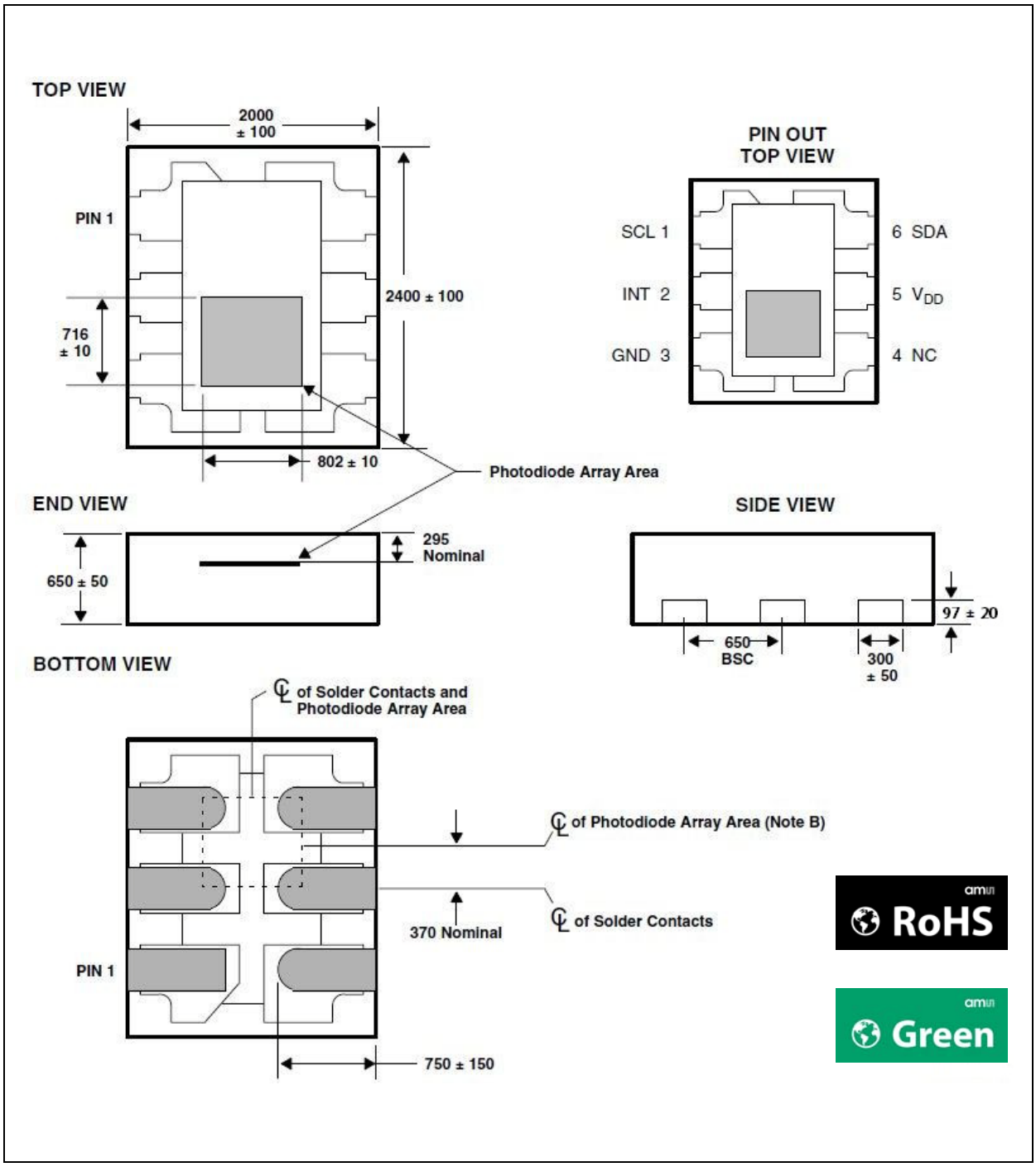


Note(s):

1. All linear dimensions are in millimeters.
2. This drawing is subject to change without notice.

Package Drawings & Markings

Figure 18:
FN Package – Dual Flat No-Lead Packaging Configuration



Note(s):

1. All linear dimensions are in micrometers.
2. The die is centered within the package within a tolerence of ±75 µm.
3. Package top surface is molded with an electrically non-conductive clear plastic compound having an index of refraction of 1.55.
4. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
5. This package contains no lead (Pb).
6. This drawing is subject to change without notice.

5. Hoja de características de MS5637

FEATURES

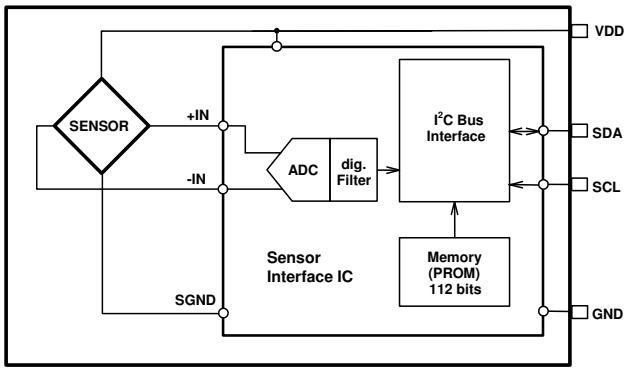
FIELD OF APPLICATION

Smart-phones
Tablet PCs
Personal navigation devices

TECHNICAL DATA

Sensor Performances (V_{DD} = 3 V)				
Pressure	**Min**	**Typ**	**Max**	**Unit**
Maximum Range	10		2000	mbar
ADC		24		bit
Resolution (1)		0.11 / 0.062/ 0.039 / 0.028 / 0.021 / 0.016		mbar
Error band at 25°C, 300 to 1200 mbar	-2		+2	mbar
Error band, -20°C to + 85°C 300 to 1200 mbar (2)	-4		+4	mbar
Response time (1)		0.5 / 1.1 / 2.1 / 4.1 / 8.22 / 16.44		ms
Long term stability		±1		mbar/yr
Temperature	**Min**	**Typ**	**Max**	**Unit**
Range	-40		+85	°C
Resolution		<0.01		°C
Accuracy at 25°C	-1		+1	°C
Notes: (1) Oversampling Ratio: 256 / 512 / 1024 / 2048 / 4096 / 8192 (2) With auto-zero at one pressure point				

FUNCTIONAL BLOCK DIAGRAM



PERFORMANCE SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Supply voltage	V_{DD}		-0.3		+3.6	V
Storage temperature	T_S		-20		+85	°C
Overpressure	P_{max}			6		bar
Maximum Soldering Temperature	T_{max}	40 sec max			250	°C
ESD rating		Human Body Model	-2		+2	kV
Latch up		JEDEC standard No 78	-100		+100	mA

ELECTRICAL CHARACTERISTICS

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Operating Supply voltage	V_{DD}		1.5	3.0	3.6	V
Operating Temperature	T		-40	+25	+85	°C
Supply current (1 sample per sec.)	I_{DD}	OSR 8192 4096 2048 1024 512 256		20.09 10.05 5.02 2.51 1.26 0.63		µA
Peak supply current		during conversion		1.25		mA
Standby supply current		at 25°C (V_{DD} = 3.0 V)		0.01	0.1	µA
VDD Capacitor		from VDD to GND	100	470		nF

ANALOG DIGITAL CONVERTER (ADC)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Output Word				24		bit
Conversion time	t_c	OSR 8192 4096 2048 1024 512 256		16.44 8.22 4.13 2.08 1.06 0.54		ms

PERFORMANCE SPECIFICATIONS (CONTINUED)

PRESSURE OUTPUT CHARACTERISTICS (V_{DD} = 3.0 V, T = 25 °C UNLESS OTHERWISE NOTED)

Parameter	Conditions		Min.	Typ.	Max.	Unit
Operating Pressure Range	P_{range}		300		1200	mbar
Extended Pressure Range	P_{ext}	Linear Range of ADC	10		2000	mbar
Relative Accuracy, autozero at one pressure point (1)	700…1000 mbar at 25°C			±0.1		mbar
Absolute Accuracy, no autozero	300..1200 mbar at 25°C 300..1200mbar, -20..85°C		-2 -4		+2 +4	mbar
Resolution RMS	OSR	8192 4096 2048 1024 512 256		0.016 0.021 0.028 0.039 0.062 0.11		mbar
Maximum error with supply voltage	V_{DD} = 1.5 V … 3.6 V			±0.5		mbar
Long-term stability				±1		mbar/yr
Reflow soldering impact	IPC/JEDEC J-STD-020C (See application note AN808 on http://meas-spec.com)			-1		mbar
Recovering time after reflow (2)				3		days

(1) Characterized value performed on qualification devices
(2) Recovering time at least 66% of the reflow impact

TEMPERATURE OUTPUT CHARACTERISTICS (V_{DD} = 3 V, T = 25°C UNLESS OTHERWISE NOTED)

Parameter	Conditions		Min.	Typ.	Max.	Unit
Absolute Accuracy	at 25°C -20..85°C		-1 -2		+1 +2	°C
Maximum error with supply voltage	V_{DD} = 1.5 V … 3.6 V			±0.3		°C
Resolution RMS	OSR	8192 4096 2048 1024 512 256		0.002 0.003 0.004 0.006 0.009 0.012		°C

PERFORMANCE SPECIFICATIONS (CONTINUED)

DIGITAL INPUTS (SDA, SCL)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Serial data clock	SCL				400	kHz
Input high voltage	V_{IH}		80% V_{DD}		100% V_{DD}	V
Input low voltage	V_{IL}		0% V_{DD}		20% V_{DD}	V
Input leakage current	I_{leak}	T = 25 °C			0.1	µA
Input capacitance	C_{IN}			6		pF

DIGITAL OUTPUTS (SDA)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Output high voltage	V_{OH}	I_{source} = 1 mA	80% V_{DD}		100% V_{DD}	V
Output low voltage	V_{OL}	I_{sink} = 1 mA	0% V_{DD}		20% V_{DD}	V
Load capacitance	C_{LOAD}			16		pF

PRESSURE AND TEMPERATURE CALCULATION

Start
Maximum values for calculation results:
P_{MIN} = 10mbar P_{MAX} = 2000mbar
T_{MIN} = -40°C T_{MAX} = 85°C T_{REF} = 20°C

Read calibration data (factory calibrated) from PROM

Variable	Description | Equation	Recommended variable type	Size [1] [bit]	Value min	Value max	Example / Typical
C1	Pressure sensitivity | $SENS_{T1}$	unsigned int 16	16	0	65535	46372
C2	Pressure offset | OFF_{T1}	unsigned int 16	16	0	65535	43981
C3	Temperature coefficient of pressure sensitivity | TCS	unsigned int 16	16	0	65535	29059
C4	Temperature coefficient of pressure offset | TCO	unsigned int 16	16	0	65535	27842
C5	Reference temperature | T_{REF}	unsigned int 16	16	0	65535	31553
C6	Temperature coefficient of the temperature | TEMPSENS	unsigned int 16	16	0	65535	28165

Read digital pressure and temperature data

D1	Digital pressure value	unsigned int 32	24	0	16777216	6465444
D2	Digital temperature value	unsigned int 32	24	0	16777216	8077636

Calculate temperature

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$	signed int 32	25	-16776960	16777216	68
TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$	signed int 32	41	-4000	8500	2000 = 20.00 °C

Calculate temperature compensated pressure

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{17} + (C4 * dT) / 2^6$	signed int 64	41	-17179344900	25769410560	5764707214
SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{16} + (C3 * dT) / 2^7$	signed int 64	41	-8589672450	12884705280	3039050829
P	Temperature compensated pressure (10…1200mbar with 0.01mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{15}$	signed int 32	58	1000	120000	110002 = 1100.02 mbar

Display pressure and temperature value

Notes
[1] Maximal size of intermediate result during evaluation of variable
[2] min and max have to be defined
[3] min and max have to be defined
[4] min and max have to be defined

Figure 2: Flow chart for pressure and temperature reading and software compensation.

SECOND ORDER TEMPERATURE COMPENSATION

In order to obtain best accuracy over temperature range, particularly at low temperature, it is recommended to compensate the non-linearity over the temperature. This can be achieved by correcting the calculated temperature, offset and sensitivity by a second-order correction factor. The second-order factors are calculated as follows:

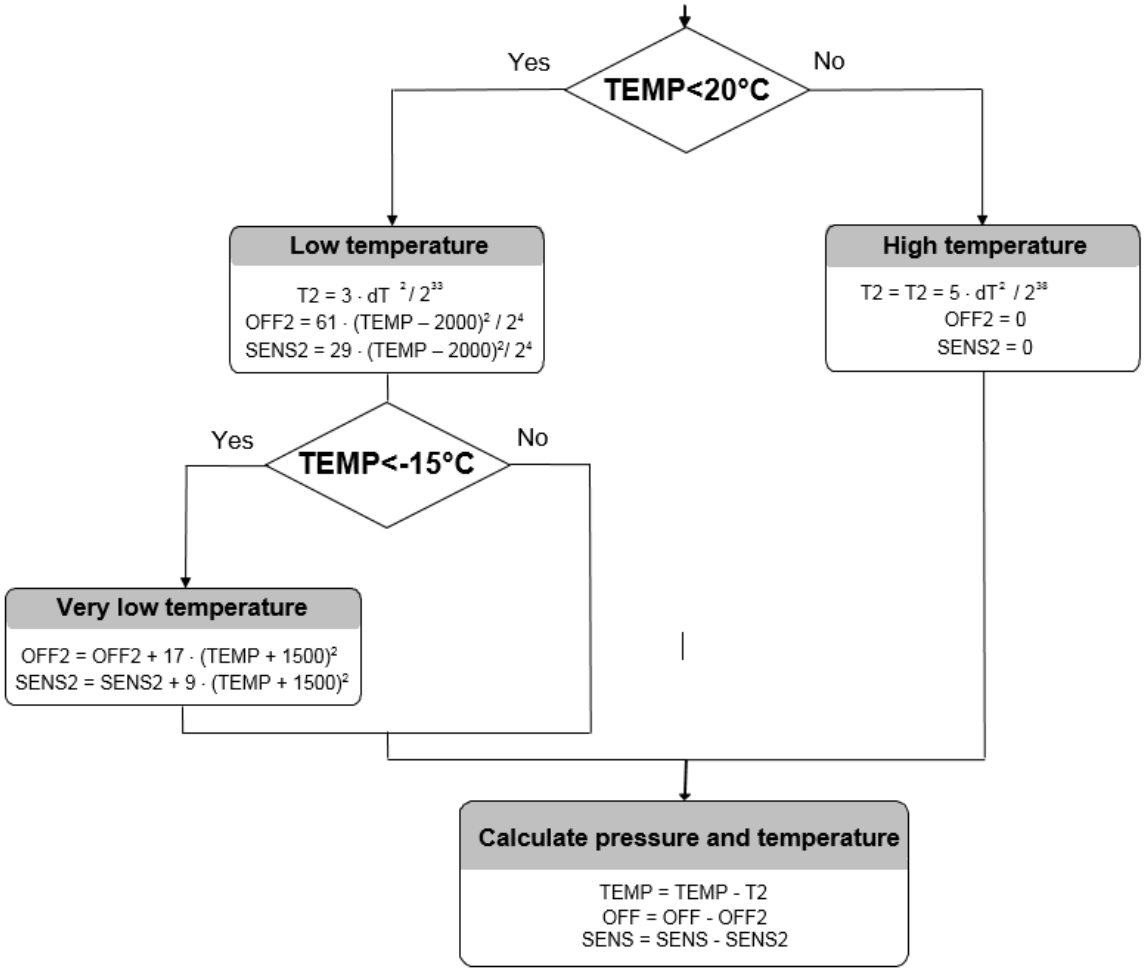


Figure 3: Flow chart for pressure and temperature to the optimum accuracy.

I²C INTERFACE

COMMANDS

The MS5637 has only five basic commands:

1. Reset
2. Read PROM (112 bit of calibration words)
3. D1 conversion
4. D2 conversion
5. Read ADC result (24 bit pressure / temperature)

Each I²C communication message starts with the start condition and it is ended with the stop condition. The MS5637 address is 1110110x (write : x=0, read : x=1).

Size of each command is 1 byte (8 bits) as described in the table below. After ADC read commands, the device will return 24 bit result and after the PROM read 16 bit results. The address of the PROM is embedded inside of the PROM read command using the a2, a1 and a0 bits.

	Command byte								**hex value**
Bit number	0	1	2	3	4	5	6	7	
Bit name	PROM	CONV	-	Typ	Ad2/ Os2	Ad1/ Os1	Ad0/ Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D1 (OSR=8192)	0	1	0	0	1	0	1	0	0x4A
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
Convert D2 (OSR=8192)	0	1	0	1	1	0	1	0	0x5A
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

Figure 4: Command structure

RESET SEQUENCE

The Reset sequence shall be sent once after power-on to make sure that the calibration PROM gets loaded into the internal register. It can be also used to reset the device PROM from an unknown condition.
The reset can be sent at any time. In the event that there is not a successful power on reset this may be caused by the SDA being blocked by the module in the acknowledge state. The only way to get the MS5637 to function is to send several SCLs followed by a reset sequence or to repeat power on reset.

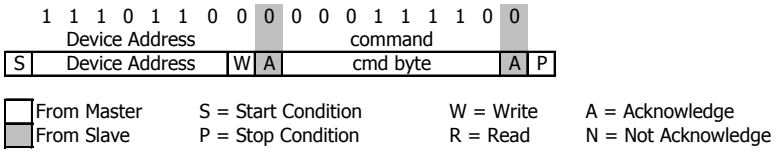


Figure 5: I²C Reset Command

PROM READ SEQUENCE

The read command for PROM shall be executed once after reset by the user to read the content of the calibration PROM and to calculate the calibration coefficients. There are in total 7 addresses resulting in a total memory of 112 bit. Addresses contains factory data and the setup, calibration coefficients, the serial code and CRC. The command sequence is 8 bits long with a 16 bit result which is clocked with the MSB first. The PROM Read command consists of two parts. First command sets up the system into PROM read mode. The second part gets the data from the system.

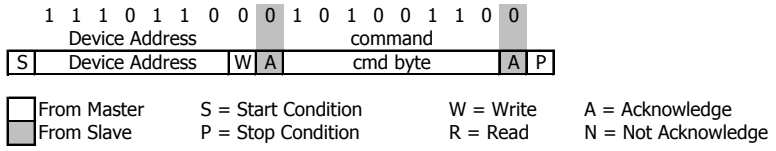


Figure 6: I²C Command to read memory address= 011

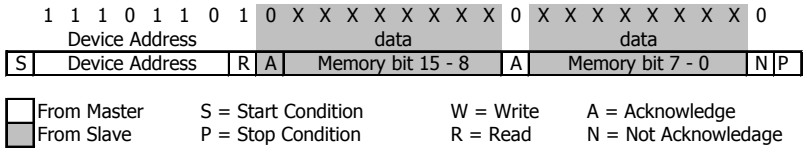


Figure 7: I²C answer from MS5637

CONVERSION SEQUENCE

The conversion command is used to initiate uncompensated pressure (D1) or uncompensated temperature (D2) conversion. After the conversion, using ADC read command the result is clocked out with the MSB first. If the conversion is not executed before the ADC read command, or the ADC read command is repeated, it will give 0 as the output result. If the ADC read command is sent during conversion the result will be 0, the conversion will not stop and the final result will be wrong. Conversion sequence sent during the already started conversion process will yield incorrect result as well. A conversion can be started by sending the command to MS5637. When command is sent to the system it stays busy until conversion is done. When conversion is finished the data can be accessed by sending a Read command, when an acknowledge is sent from the MS5637, 24 SCL cycles may be sent to receive all result bits. Every 8 bits the system waits for an acknowledge signal.

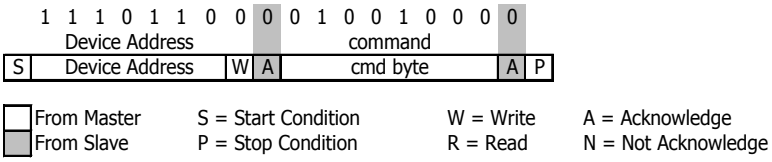


Figure 8: I²C command to initiate a pressure conversion (OSR=4096, typ=D1)

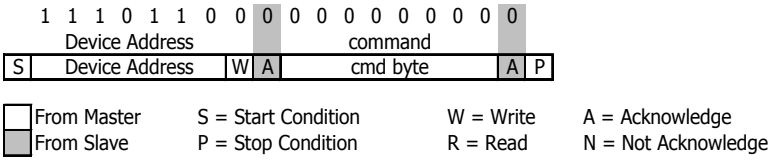


Figure 9: I²C ADC read sequence

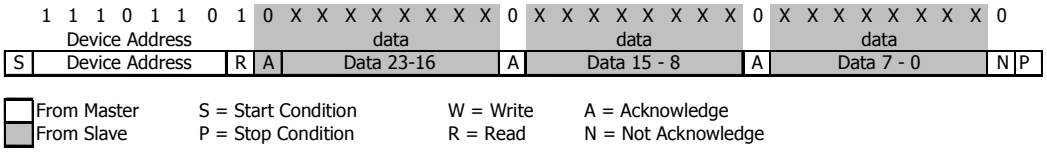


Figure 10: I²C answer from MS5637

CYCLIC REDUNDANCY CHECK (CRC)

MS5637 contains a PROM memory with 112-Bit. A 4-bit CRC has been implemented to check the data validity in memory. The C code example below describes the CRC calculation which is stored on DB12 to DB15 in the first PROM word.

Add	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	CRC				Factory defined											
1	C1															
2	C2															
3	C3															
4	C4															
5	C5															
6	C6															

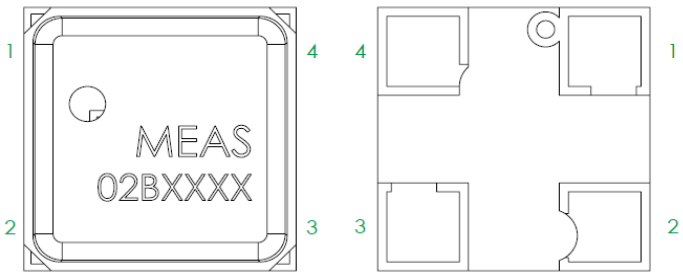
Figure 11: Memory PROM mapping

C Code example for CRC-4 calculation:

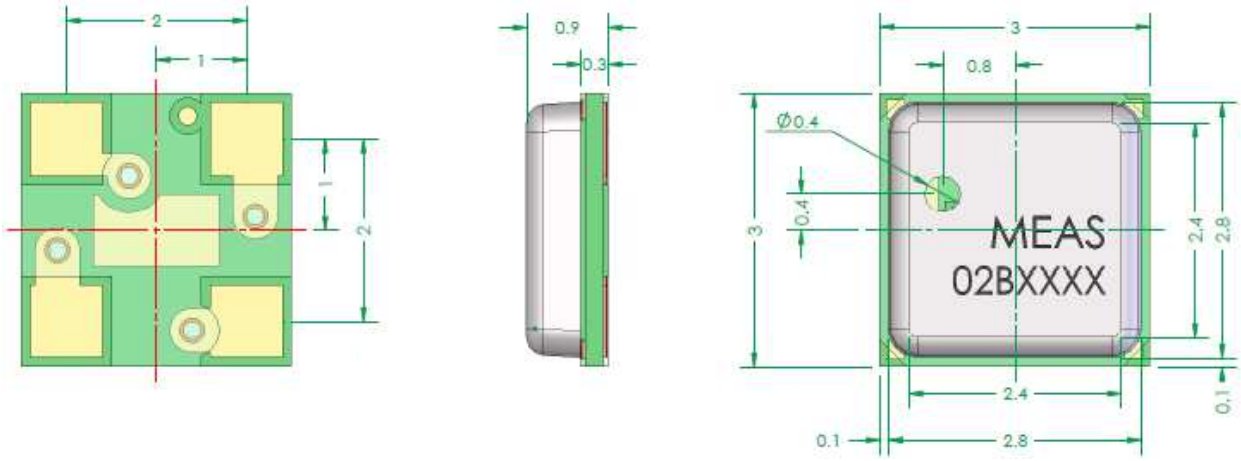
```
unsigned char crc4(unsigned int n_prom[])                    // n_prom defined as 8x unsigned int (n_prom[8])
{
int cnt;                                                     // simple counter
unsigned int n_rem=0;                                        // crc reminder
unsigned char n_bit;

        n_prom[0]=((n_prom[0]) & 0x0FFF);                    // CRC byte is replaced by 0
        n_prom[7]=0;                                         // Subsidiary value, set to 0
        for (cnt = 0; cnt < 16; cnt++)                       // operation is performed on bytes
                {                                            // choose LSB or MSB
                if (cnt%2==1)       n_rem ^= (unsigned short) ((n_prom[cnt>>1]) & 0x00FF);
                else                n_rem ^= (unsigned short) (n_prom[cnt>>1]>>8);
                for (n_bit = 8; n_bit > 0; n_bit--)
                        {
                        if (n_rem & (0x8000))       n_rem = (n_rem << 1) ^ 0x3000;
                        else                        n_rem = (n_rem << 1);
                        }
                }
        n_rem= ((n_rem >> 12) & 0x000F);                     // final 4-bit reminder is CRC code
        return (n_rem ^ 0x00);
}
```

PIN CONFIGURATION

Pin	Name	Type	Function
1	VDD	P	Positive supply voltage
2	SDA	I/O	I²C data
3	SCL	I	I²C clock
4	GND	I	Ground



DEVICE PACKAGE OUTLINE



Notes: (1) Dimensions in mm
(2) General tolerance: ±0.1

Figure 13: MS5637 package outline

RECOMMENDED PAD LAYOUT

Pad layout for bottom side of the MS5637 soldered onto printed circuit board.

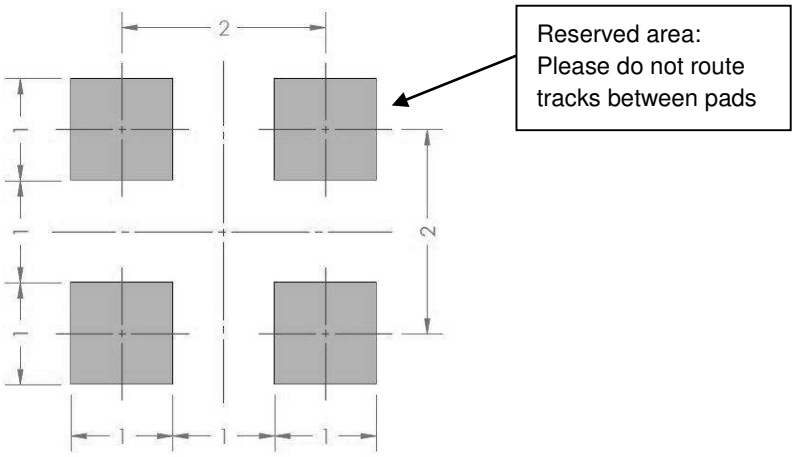
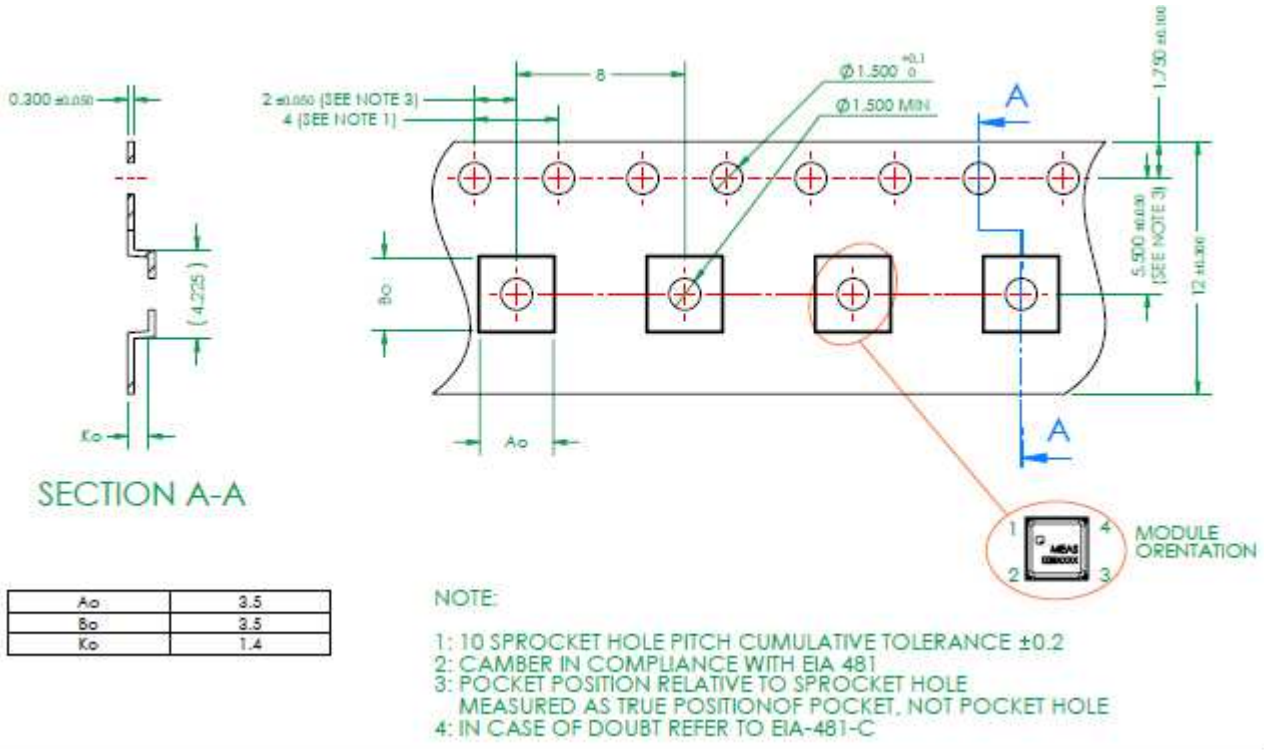


Figure 14: MS5637 pad layout

SHIPPING PACKAGE



Ao	3.5
Bo	3.5
Ko	1.4

NOTE:

1: 10 SPROCKET HOLE PITCH CUMULATIVE TOLERANCE ±0.2
2: CAMBER IN COMPLIANCE WITH EIA 481
3: POCKET POSITION RELATIVE TO SPROCKET HOLE
MEASURED AS TRUE POSITIONOF POCKET, NOT POCKET HOLE
4: IN CASE OF DOUBT REFER TO EIA-481-C