Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Medida de Cobertura y Envío de Datos de Sensores Ambientales a través de LoRa en Contexto Internet de las Cosas

Autor: Pedro Rivero Lobo

Tutor: Juan Antonio Becerra González

Dpto. Teoría de la señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Medida de Cobertura y Envío de Datos de Sensores Ambientales a través de LoRa en Contexto Internet de las Cosas

Autor:
Pedro Rivero Lobo

Tutor:
Juan Antonio Becerra González
Profesor Ayudante Doctor

Dpto. Teoría de la señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Medida de Cobertura y Envío de Datos de Sensores Ambientales a través de LoRa en Contexto Internet de las Cosas

Autor: Pedro Rivero Lobo
Tutor: Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Este trabajo jamás habría visto la luz si no fuera por un gran número de personas que han estado en mi vida desde que empecé la carrera.

En primer lugar quería dar las gracias a mi familia, pero, sobre todo, a mi padre José Pedro y a mi madre María José, quienes se han privado de todo para que tanto mi hermana María como yo tuviéramos estudios universitarios. Muchas gracias papá, muchas gracias mamá, la educación que me habéis dado es lo más valioso que tengo.

A mi abuela Mari y a mi abuela Loli, quienes me han ayudado en los exámenes encendiendo una vela y se han alegrado más que yo por los aprobados.

Gracias a todos mis amigos, amigas, compañeros y compañeras de clase, con quienes he reído, llorado y sufrido durante exámenes, con quienes he compartido horas de estudio en la biblioteca y con quienes he celebrado tanto mis aprobados como los suyos. Hago mención especial a Mercedes, con quien además de compartir muchísimas horas de biblioteca y lágrimas, he conservado una bonita amistad desde prácticamente el primer año a pesar de estudiar diferentes especialidades. También, agradecerle a mi gran amiga y eterna compañera de clase Esther, su amistad, ayuda y profesionalidad en todos los trabajos y prácticas que hemos estado juntos, además de su ayuda en los exámenes que nos hemos preparado juntos.

Agradecerle a mi pareja, Paula, el estar conmigo en mis peores momentos. Ha sido un apoyo fundamental en el transcurso del peor año de mi vida respecto a dificultad académica, siempre ha estado cuando lo he necesitado.

También agradecerle a mi tutor Juan Antonio Becerra su tiempo, paciencia, dedicación, comprensión, flexibilidad y rapidez, ya que, sin él, habría resultado imposible realizar este trabajo.

Y, por último, quería dedicar este proyecto a la memoria de mi abuelo Juan, quien por un año no pudo verme terminar la carrera.

Pedro Rivero Lobo
Sevilla, 2021

Resumen

Este proyecto consiste en la implementación de un sensor IoT (internet de las cosas, del inglés, Internet of Things) de temperatura y humedad relativa ambientales. Para ello se conectarán dos placas STM32WL55JC mediante LoRa, un tipo de modulación basada en la modulación CSS (modulación de espectro ensanchado chirp), con mucho alcance y bajo consumo de energía. Las placas son del fabricante STMicroelectronics y se caracterizan principalmente por su módulo integrado que utilizaremos para usar la modulación LoRa. Además estas placas necesitan una baja potencia para su funcionamiento.

Una de estas placas, actuando como transmisor, estará conectada a un sensor SHT20, el cual permite obtener medidas ambientales a través de las variables de humedad ambiental y temperatura. Este sensor tomará las medidas oportunas para enviarlas a la otra placa, que actúa como receptor, el cual estará conectado a un puerto UART de una computadora donde podremos recibir toda la información procedente del sensor. Una vez interconectadas las placas, se harán dos tipos de experimentos; en uno se tomarán medidas de la temperatura y humedad relativa cada 5 minutos durante 24 horas; y en el otro se determinará la cobertura de la conexión de las placas mediante los parámetros de RSSI y SNR, que son el indicador de intensidad recibida y la relación señal-ruido, respectivamente. Con esto, se comprobará que al aumentar el parámetro Spreading Factor de la modulación, se obtiene una mayor cobertura.

Abstract

This project consists of the implementation of an IoT (Internet of Things) sensor for environmental temperature and relative humidity. Two STM32WL55JC boards will be connected using LoRa, a type of modulation based on CSS modulation (chirp spread spectrum modulation), with long range and low power consumption. The boards are from the manufacturer STMicroelectronics and are mainly characterised by their integrated module which we will use to use LoRa modulation. In addition, these boards require low power for operation.

One of these boards, acting as a transmitter, will be connected to a SHT20 sensor, which allows environmental measurements to be obtained through the environmental humidity and temperature variables. This sensor will take the appropriate measurements to send them to the other board, acting as a receiver, which will be connected to a UART port of a computer where we will be able to receive all the information coming from the sensor. Once the boards are interconnected, two types of experiments will be carried out; in one, temperature and relative humidity measurements will be taken every 5 minutes for 24 hours; and in the other, the coverage of the connection of the boards will be determined by means of the RSSI and SNR parameters, which are the received intensity indicator and the signal-to-noise ratio, respectively. This will show that by increasing the Spreading Factor parameter of the modulation, a higher coverage will be obtained.

... -translation by DeeplL-

Índice Abreviado

Índice

1 Introducción

A día de hoy, el internet de las cosas cada vez evoluciona más rápido y cada día aumenta el número de dispositivos conectados a internet.

El tipo de dispositivo que ocupa un mayor porcentaje de la totalidad del internet de las cosas son los dispositivos inteligentes como los smartphones, tablets o notebooks, pero la idea es que con el paso de los años todo esté conectado a internet, incluso nuestro propio cuerpo.

En este documento veremos un ejemplo de aplicación del concepto del internet de las cosas, donde monitorizaremos en tiempo real las variables físicas de la temperatura y la humedad relativa del sitio donde coloquemos nuestro sensor, tal y como se verá en el capítulo 7.

En esta ocasión, usaremos la modulación LoRa® para comunicar nuestro transmisor y nuestro receptor, de manera que donde queramos leer las variables físicas mencionadas, será donde coloquemos nuestro transmisor.

A continuación se definen los objetivos, el alcance y la estructura de este trabajo.

1.1 Objetivos.

El objetivo principal de este trabajo es desarrollar un dispositivo IoT que medirá la temperatura y la humedad relativa del ambiente en el lugar donde esté situado y enviará dicha información a través de LoRa.

Como objetivo adicional, se elaborará una memoria teórica que incluya los conceptos básicos de teoría que preceden al trabajo técnico realizado.

1.2 Alcance.

Este trabajo recoge todos los pasos necesarios y el procedimiento requerido para montar sensor IoT donde se obtengan los valores de la humedad y la temperatura.

Para ello, se usarán dos placas de desarrollo STM32WL55JC, las cuales serán programadas para transferirse entre ellas la información correspondiente a la temperatura y la humedad del entorno donde se coloque la placa transmisora.

Posteriormente se harán dos experimentos donde se ponga a prueba las especificaciones del trabajo realizado.

1.3 Estructura.

Tras la introducción pasaremos a una sección teórica, compuesta por los capítulos 2, 3 y 4, donde se dará una noción básica de la teoría necesaria para entender la parte experimental y donde se verán algunos ejemplos y utilidades del IoT y LoRa.

Después pasaremos al capítulo 5, donde se dará paso a un capítulo de materiales en el que hablaremos de todos los dispositivos y herramientas relevantes de los que hemos hecho uso para conformar este proyecto.

Tras terminar el capítulo de materiales se pasará al capítulo 6, donde se desarrolla el proyecto y el trabajo más práctico, que consistirá en el montaje del circuito, desarrollo del código y explicación de las diferentes versiones por las que ha pasado el proyecto.

Posteriormente, en el capítulo 7, veremos un par de experimentos donde pondremos en práctica el proyecto desarrollado y también veremos algunos parámetros de la comunicación, como la cobertura que tendrá dicho proyecto.

A continuación del capítulo de experimentos, nos encontramos con el capítulo 8, donde hablaremos de las conclusiones finales y algunas posibles líneas futuras que podría tomar el proyecto, tales como algunas modificaciones o incorporaciones.

En la última parte del documento se encuentra apéndice A, donde encontramos la información referida a los códigos usados; y el apéndice B, con los datasheets de los componentes utilizados.

2 Internet de las cosas: IoT

Hoy día, son muchos los objetos electrónicos que están conectados a internet, tales como teléfonos o computadoras. IoT (del inglés, Internet of things/internet de las cosas) hace referencia a la interconexión de todos los tipos de objetos electrónicos.

Se trata de una arquitectura emergente basada en el Internet global. Esta da pie al intercambio de bienes y servicios entre redes de suministro. Además, posee un gran papel en la seguridad y privacidad de los actores involucrados [1].

En 1999 Kevin Ashton fue quien usó por primera vez el término de IoT, pero se considera que el IoT nació entre 2008 y 2009 [2].

El IoT introducirá beneficios en la gestión y el seguimiento en tiempo real de los activos y productos, ya que nos permitirá aumentar la cantidad de datos y la mejora de equipos y el uso de recursos, lo que podría ahorrar costes.

Esta arquitectura nos traerá un gran cambio en la vida cotidiana gracias a la posibilidad de acceso a servicios de educación, asistencia sanitaria, datos o transporte. Por otro lado, las empresas podrían aumentar su productividad gracias a los nuevos servicios personalizados de sus dispositivos inteligentes.

2.1 Aplicaciones.

En este apartado se hablará de las principales aplicaciones que puede tener el mundo del IoT. En resumidas cuentas, se podría decir que IoT está formada por un conjunto de sensores y actuadores que proporcionan y reciben información digital que se coloca en la red, de manera que se puedan transmitir los datos para su procesamiento [4]. A un mismo dispositivo se le pueden unir varios sensores, pudiendo así medir una gran cantidad de variables físicas para luego transmitirlas a la nube.

Hoy en día, todo tipo de dispositivos electrónicos tales como televisores o neveras tienen capacidad para la comunicación y la detección, las cuales irán creciendo con la incorporación de nuevas herramientas de detección. Los sistemas IoT tienen una arquitectura que se puede seccionar en cuatro capas [5]:

- **Detección de objetos**: El IoT incorpora esta capa reduciendo así los requisitos de la capacidad de los dispositivos, lo que permite su interconexión.
- **Intercambio de datos**: Esta capa es la responsable de la transmisión transparente de datos a través de las redes de comunicación.
- **Integración de la información**: Esta capa es la responsable de la comunicación y las transacciones. Gracias a ella los sensores se intercomunican con otros o propietarios de estos.

- **Servicios de aplicaciones**: En esta capa se da servicio de contenido a todos los usuarios de la red.

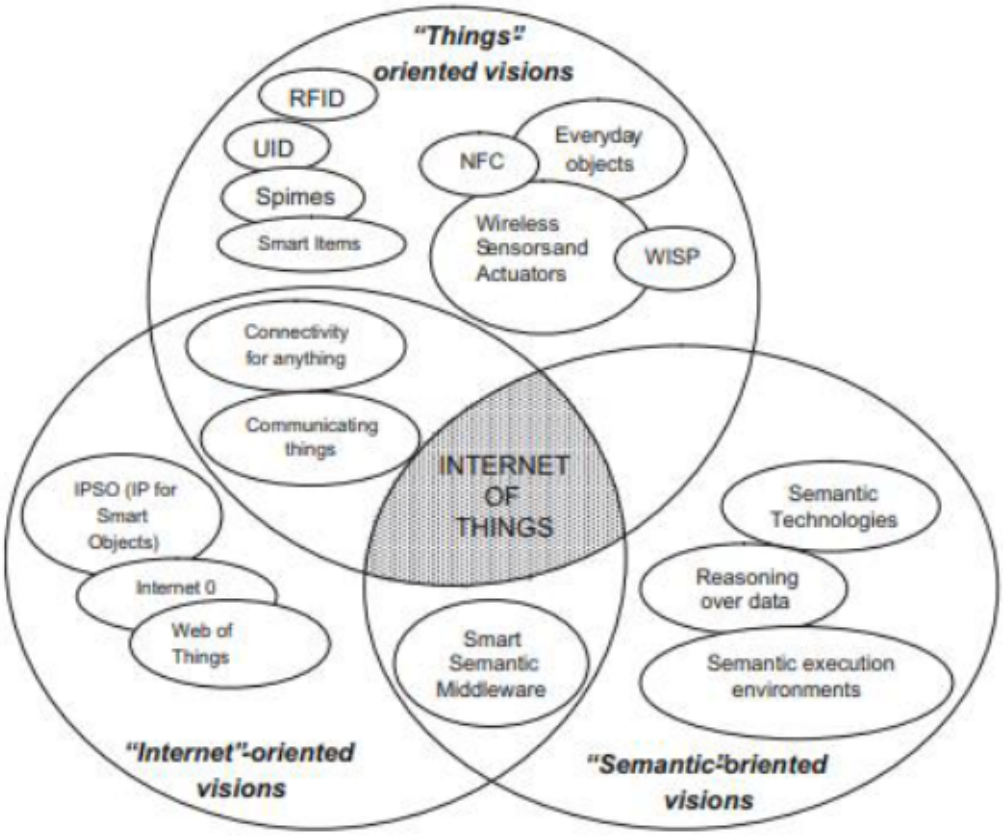


Figura 2.1 Perspectivas del internet de las cosas [11].

A modo de ejemplo, tenemos algunas aplicaciones y servicios que tiene el IoT en las tablas 2.1, 2.2 y 2.3. Estos ejemplos se han obtenido directamente de [12].

Tabla 2.1 Aplicaciones y servicios de ejemplo de IoT (PARTE 1/3)[12].

APLICACIÓN/SERVICIO	EJEMPLOS
Ciudades y transportes inteligentes	Integración de los servicios de seguridad. Optimización del transporte Sensores de aparcamiento Gestión de aparcamientos y tráfico en tiempo real Gestión de semáforos Localización de los coches Las redes energéticas inteligentes Seguridad Administración del Agua Riego de parques y jardines Contenedores de basura inteligentes Controles de contaminación y movilidad Sistemas de Votación Monitoreo de accidentes
Edificios inteligentes	Las mejoras en la eficiencia Mejoras de seguridad (sensores y alarmas) Aplicaciones domóticas Los servicios de salud y educación en el hogar Control remoto de los tratamientos para los pacientes Servicios de cable / satélite Sistemas de almacenamiento / generación de energía Apagado automático de la electrónica cuando no esté en uso Termostatos inteligentes Los detectores de humo y alarmas Aplicaciones de control de acceso Cerraduras inteligentes Seguridad para todos los miembros de la familia.
Electrónica de consumo	Teléfonos inteligentes Televisión inteligente Laptops, computadoras y tabletas Refrigeradores, lavadoras y secadoras inteligentes Sistemas de cine en casa inteligentes Aparatos inteligentes Sensores para el collar del animal doméstico Personalización de la experiencia del usuario El funcionamiento del producto autónomo Localizadores personales Gafas inteligentes
Salud	Monitoreo de las enfermedades crónicas Mejora de la atención y calidad de vida de los pacientes Trackers de Actividad Diagnóstico remoto Pulseras conectadas Cinturones interactivos Deporte y monitoreo de actividades de fitness Etiquetas inteligentes para fármacos Seguimiento del uso de drogas Los biochips Interfaces cerebro-ordenador Monitoreo de los hábitos alimenticios

Tabla 2.2 Aplicaciones y servicios de ejemplo de IoT (PARTE 2/3)[12].

APLICACIÓN/SERVICIO	EJEMPLOS
Educación	Vinculación de aulas virtuales y físicas Servicios de acceso a bibliotecas Intercambio de informes y resultados en tiempo real El aprendizaje permanente Aprendizaje de idiomas extranjeros Gestión de la asistencia
Automoción	Control de tráfico Avanzar en la información sobre lo que está roto Monitoreo inalámbrico de los neumáticos del coche La gestión inteligente de la energía y el control Auto diagnóstico Los acelerómetros Sensores de posición, de presencia y de proximidad Análisis de la mejor manera de ir en tiempo real a un sitio Localización por GPS Control de la velocidad del vehículo Vehículos autónomos que utilizan los servicios de del IoT.
Agricultura y medio ambiente	Medición y control de la contaminación Pronosticar cambios climáticos Las etiquetas RFID pasivas en productos agrícolas Sensores en palets de productos Gestión de residuos Cálculos de Nutrición.
Servicios de energía	Datos precisos sobre el consumo de energía La medición inteligente Redes inteligentes Análisis de consumo energético Pronosticar las tendencias futuras de energía Redes de sensores inalámbricos La producción de energía y el reciclaje
Fabricación	Gas y sensores de flujo Sensores inteligentes de variables físicas Visión de máquinas Detección acústica y de vibraciones Aplicaciones compuestas Control inteligente de robots Control y optimización de fabricación Reconocimiento de patrones Aprendizaje automático El análisis predictivo Logística móvil Gestión de almacenes Prevenir la sobreproducción Logística eficiente

Tabla 2.3 Aplicaciones y servicios de ejemplo de IoT (PARTE 3/3)[12].

APLICACIÓN/SERVICIO	EJEMPLOS
Conectividad inteligente	Gestión de datos y prestación de servicios Medios de comunicación social y las redes sociales La comunicación de grupo interactiva En streaming en tiempo real Juegos interactivos Realidad aumentada Supervisión de la seguridad de la red Interfaces de usuario disponibles La computación afectiva Métodos de autenticación biométrica Telemática de consumo Servicios de comunicación M2M Análisis de grandes datos Realidad virtual Servicios de computación en nube Computación ubicua Visión por computador Antenas inteligentes
Compras	Compras inteligentes RFID y otras etiquetas electrónicas y lectores Los códigos de barras en el comercio minorista Inventarios Control de la procedencia geográfica de productos Control de calidad de los alimentos y de la seguridad

2.2 Nuevas tecnologías.

IoT es posible gracias al desarrollo de las nuevas tecnologías. A continuación veremos las más relevantes [6] [7].

2.2.1 Energía.

Necesitamos tecnología que pueda generar energía de alta densidad de potencia que, al usarse en la nanoelectrónica de baja potencia, nos permita fabricar un dispositivo que se autoalimente basado en sensores inteligentes inalámbricos.

2.2.2 Sensores.

Los sensores son una parte fundamental del IoT. Estos pueden ser de cualquier tipo y se pueden colocar prácticamente en cualquier parte, tales como debajo de la piel, carteras, etc. Pueden ser tan pequeños del orden de milímetros y enviar información a kilómetros de distancia.

Los sensores son muy utilizados en la industria, desde la construcción hasta la salud y se pueden anticipar a las necesidades humanas basándose en la información recogida.

Siguiendo la Ley de Moore, los factores de forma de los chips de silicio son diseñados cada vez más pequeños y con mejor eficiencia y rendimiento.

Por otro lado, también baja el coste del ancho de banda, al igual que el coste de procesado, permitiendo así que cada vez haya más dispositivos que estén conectados y que sean lo suficientemente inteligentes como para actuar en función de la información recogida.

2.2.3 Comunicación.

Para que los dispositivos dentro de una red IoT puedan comunicarse se necesitarán nuevas antenas inteligentes multi-banda, las cuales irán integradas en el propio dispositivo.

Estas antenas deberían estar optimizadas en tamaño, coste y eficiencia. Podrán ser del tipo que sea más conveniente, como por ejemplo una bobina, una antena impresa, embebida o múltiple.

Por otro lado, las plataformas web del IoT donde se encuentran todos los objetos deberían tener protocolos de comunicación diseñados expresamente para estas arquitecturas.

2.2.4 Integración.

Todos los posibles dispositivos inteligentes mencionados deberán ir integrados en envases o productos, pudiendo así ahorrar costes.

Para ello, se hará uso de sustratos no convencionales como por ejemplo tejidos textiles o el papel e incluso se desarrollarán nuevos sustratos, conductores y materiales de unión adecuados a ambientes hostiles donde sea más propensa la degradación de dispositivos.

Existe una tecnología llamada System-in-Package (SiP) que permite la integración flexible y 3D de todo tipo de componentes, tales como antenas o sensores activos y componentes pasivos en envases.

Para conectar el chip del circuito integrado se usarán incrustaciones RFID con una estructura de acoplamiento por tiras. Gracias a esto se podrá producir una gran variedad de formas y tamaños de etiquetas.

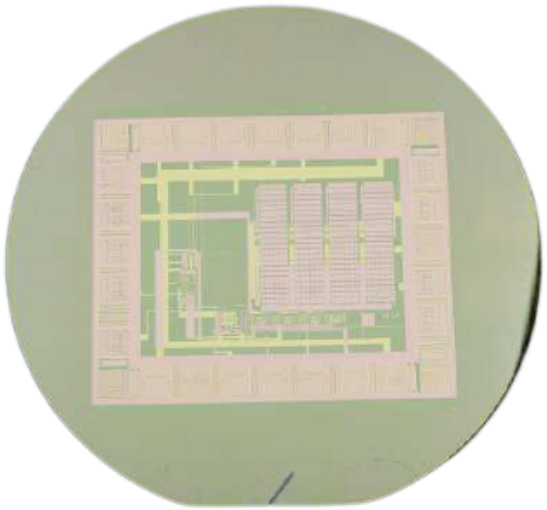


Figura 2.2 Circuito impreso en una etiqueta electrónica [20].

2.3 Desafíos y retos para el desarrollo y despliegue de IoT.

En los siguientes puntos vamos a analizar todos los retos a los que se enfrentarán el desarrollo y despliegue del IoT [8].

2.3.1 Fiabilidad.

Cuando hablamos de fiabilidad nos referimos al correcto funcionamiento del sistema basado en la especificación que ha de cumplir. La parte más crítica del IoT es la comunicación, es por eso por lo que tiene que ser resistente a errores, garantizando una correcta distribución de información. Se debe implementar fiabilidad en todas las capas vistas, tanto en software como en hardware. Una recepción no fiable podría dar como resultado errores, retrasos o pérdida de datos, decisiones erróneas y todo ello puede hacer el IoT no fiable.

2.3.2 Rendimiento.

Denominamos el rendimiento de una red de telefonía u ordenadores percibido por los usuarios de la red como la calidad de servicio, también conocido como QoS (Quality of Service). En su mayor parte, el rendimiento de los servicios del IoT depende directamente del rendimiento de sus componentes y de las nuevas tecnologías. Además, con el fin de mejorar el rendimiento del IoT, los dispositivos deben ser revisados y evaluados periódicamente.

2.3.3 Interoperabilidad.

La interoperabilidad es la capacidad de los dispositivos y de sus procedimientos de compartir información y facilitar el intercambio de información y conocimiento entre ellos mismos.

Esto supone otro desafío para el IoT, ya que se quiere manejar una gran variedad de dispositivos pertenecientes a diferentes plataformas.

Por ello, los diseñadores del IoT deberán construir aplicaciones en las que se puedan introducir nuevas funciones sin interferir en las funciones existentes mientras se mantiene la integración de las tecnologías de comunicación existentes.

En conclusión, las etiquetas del futuro tendrán que integrar todos los estándares de comunicación y los protocolos que trabajen en diferentes frecuencias, además de permitir arquitecturas diferentes y ser capaces de comunicarse con otras redes. Todo esto si no aparece un nuevo estándar global.

Figura 2.3 Visión del internet de las cosas (IoT) [10].

2.3.4 Seguridad y privacidad.

No es fácil garantizar la seguridad y privacidad de los usuarios debido a la heterogenealidad del IoT.

Como sabemos, la funcionalidad principal del IoT está basada en el intercambio de datos entre diferentes dispositivos conectados a internet. Uno de los problemas no considerados en los estándares es la distribución de las claves entre los diferentes dispositivos.

Como posible solución, tenemos la agrupación de los dispositivos integrados en una serie de redes virtuales a las cuales sólo puedan entrar determinados dispositivos de cada red virtual.

2.3.5 Gestión.

El hecho de que haya millones de dispositivos interconectados supone que haya mucha dificultad en el aspecto de gestionar fallos, configuraciones, contabilidad, rendimiento, y seguridad. Para gestionar todo, existen diferentes opciones:

- Open Mobile Alliance® ha desarrollado un estándar Light-weight M2M ("Machine to machine", intercambio de información de máquina a máquina) para proporcionar una interfaz entre todos los dispositivos M2M y los servidores. Así, ofrece aplicaciones M2M capaces de administrar de manera remota los dispositivos, aplicaciones y servicios M2M. Esto es muy importante para asegurar la conectividad bajo demanda en todo momento. Además, OMA especifica una serie de protocolos y mecanismos para la gestión de dispositivos y servicios móviles en entornos con pocos recursos
- Internet Engineering Task Force® (IETF) ha creado un protocolo (NETCONF light) para gestionar dispositivos restringidos, el cual proporciona lo necesario para poder instalar, eliminar y manipular la configuración de dispositivos en la red
- MASH IoT es una plataforma desarrollada de forma independiente que facilita el seguimiento, control y configuración de los activos del IoT, además de mejorar la velocidad de la conexión de dispositivos, garantizando así la prestación de servicios.

3 Long Range (LoRa)

Los transceptores son dispositivos que tienen capacidad de transmitir y recibir de manera inalámbrica mediante radiocomunicación, por ejemplo. Este es el caso de los dispositivos que usaremos en este proyecto. En nuestro caso, usaremos la modulación LoRa®, ya que puede enviar datos a largas distancias. Tras abordar esta sección, se tendrán unas nociones básicas de LoRa, LoRaWAN y la arquitectura de LoRaWAN.

3.1 Definición de LoRa.

LoRa (abreviatura del inglés Long Range, largo alcance) es un tipo de modulación patentada y desarrollada por Semtech®, la cual está basada en la modulación de espectro ensanchado chirp (Chirp Spread Spectrum, CSS). LoRa proporciona un largo alcance, un bajo consumo de energía a una baja velocidad de datos y una transmisión segura de datos.

Para transmitir información en una señal, esta debe modificarse de alguna manera. Hay varias formas de hacerlo y dos de los métodos más usados son la variación de la amplitud y la variación de la frecuencia.

3.1.1 Modulación de la amplitud (AM).

En la variación de la amplitud (AM), la intensidad de la señal de la portadora varía en proporción de la señal del mensaje que se transmite.

En la figura 3.1 podemos ver cómo la señal de información se transforma en la señal modulada. La señal de información se mezcla con la señal portadora usando un mezclador.

Por otro lado, vemos que la señal portadora, generada por un oscilador, tiene una frecuencia y una amplitud constantes.

Sin embargo, las señales AM son más vulnerables al ruido y ofrecen una peor calidad de sonido respecto a la modulación de frecuencia.

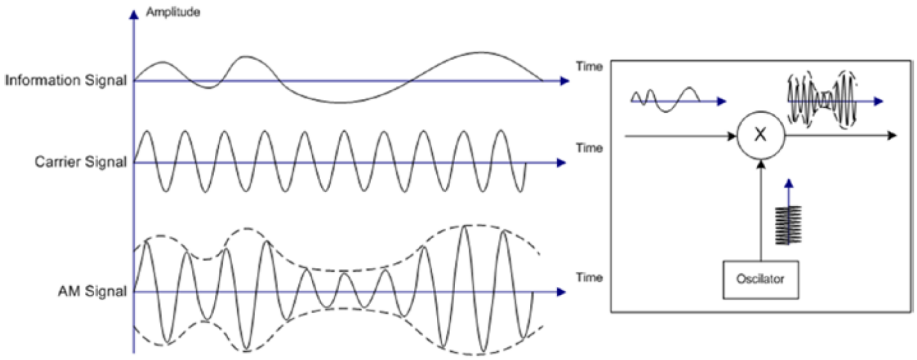


Figura 3.1 Modulación de la amplitud (AM) [13].

3.1.2 Modulación de frecuencia.

La modulación de frecuencia (FM) se utiliza sobre todo para la transmisión de radio FM. En este tipo de modulación, la frecuencia de la onda portadora se cambia de acuerdo con la intensidad de la señal mientras que la amplitud y la fase permanecen constantes.

En la figura 3.2 se muestra la técnica de FM. La señal de información se mezcla con la portadora usando un mezclador, al igual que ocurría con AM.

Cuando el voltaje de la señal de información es 0 la frecuencia de la portadora no cambia; cuando la señal de información se acerca a sus picos positivos la frecuencia de la portadora aumenta al máximo; y cuando hay un pico negativo, la frecuencia de la portadora se reduce al mínimo. De esta manera, obtenemos la señal modulada resultante con una amplitud constante y diferentes frecuencias.

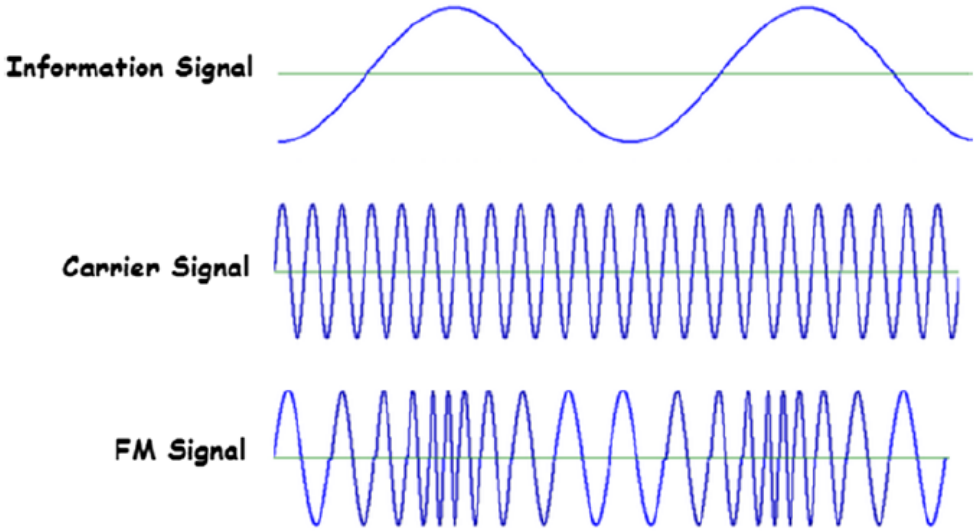


Figura 3.2 Modulación de la frecuencia (FM) [13].

Las señales FM son más resistentes al ruido y tienen una mejor calidad de sonido respecto a las señales AM. Sin embargo, no pueden viajar largas distancias y pueden ser bloqueadas por edificios altos o montañas.

3.1.3 Modulación por desplazamiento de frecuencia (FSK).

Una señal FSK representa una señal digital con dos frecuencias. Una frecuencia representaría el 1 digital y otra representaría el 0 digital. En la figura 3.3 se puede ver cómo una señal digital se transforma en una analógica utilizando la modulación FSK.

Se utiliza una señal portadora y dos frecuencias diferentes para representar los estados digitales de alto y bajo. La señal digital de datos se mezcla con la portadora y se codifica en una señal analógica modulada.

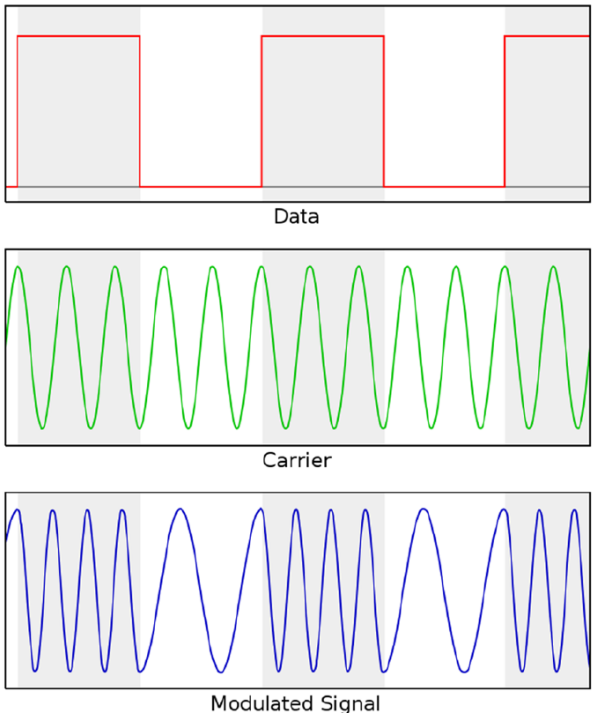


Figura 3.3 Modulación por desplazamiento de frecuencia (FSK) [13].

3.1.4 Chirp Spread Spectrum (CSS).

La modulación de espectro ensanchado chirp (CSS) mantiene las mismas características de baja potencia que la modulación FSK y es una técnica de espectro ensanchado que utiliza los pulsos (chirps) modulados en frecuencia lineal de banda ancha para codificar la información.

CSS se desarrolló para aplicaciones de radar en 1940 y se ha utilizado en comunicaciones militares y espaciales durante décadas debido a sus largas distancias de comunicación, sus bajos requisitos de potencia y su menor vulnerabilidad a las interferencias.

3.1.5 Modulación de espectro ensanchado LoRa.

Ya hemos comentado que la modulación LoRa está basada en CSS para codificar datos. Cada bit se distribuye por un factor de chipping. El número de chirps por bit lo denominaremos factor de difusión (spreading factor, SF).

La modulación LoRa es más compleja y resistente al ruido de fondo respecto a otras modulaciones. En vez de usar las dos frecuencias de FSK, lo que hace es barrer entre las dos frecuencias, como se puede ver en la figura 3.4. En la parte inferior, podemos ver los barridos de frecuencia de forma descendente. En la parte superior, se muestra los barridos de forma ascendente.

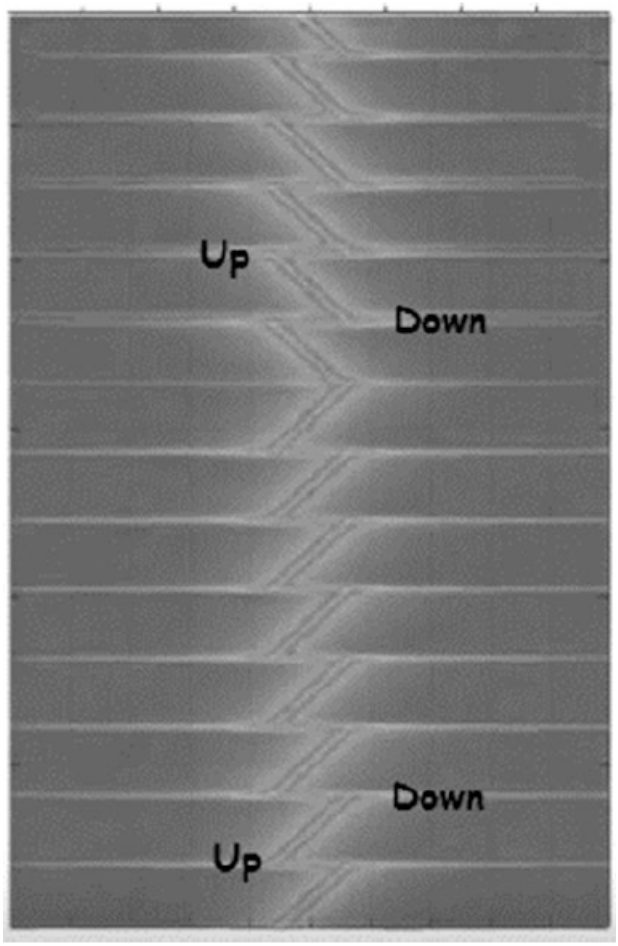


Figura 3.4 Barrido entre las dos frecuencias (descendente y ascendente en frecuencia) [13].

Spreading Factor (SF) [17]

El **Spreading Factor** es el factor de difusión de la modulación LoRa y es un parámetro clave en la transmisión.Encontrar el SF adecuado es crucial para lograr un buen rendimiento a largo plazo. Este nos indica cuántos chirps por segundo se envían desde el transmisor. Se clasifica entre 7 y 12 en función de las condiciones ambientales entre el transmisor y el receptor. Esto supondrá que la función ADR (Adaptive Data Rate) esté activada, lo que se recomienda para los dispositivos que estarán en un lugar fijo.

Cuanto menor sea el SF más chirps se enviarán por segundo, lo que se traduce en codificar más datos por segundo. Por otro lado, cuanto mayor sea el SF menor será el número de chirps por segundo, siendo menor el número de datos a codificar.

Para una misma cantidad de datos a enviar, cuanto mayor sea el SF mayor será el tiempo de transmisión, conocido como tiempo aire. Esto se traduce en un mayor tiempo de funcionamiento y por tanto un mayor consumo de energía.

Se podría pensar que es conveniente tener un SF bajo, pero el hecho de tener un SF alto tiene sus ventajas. Un SF alto supone un mayor tiempo de aire, lo que da más tiempo al receptor para muestrear la potencia de la señal, lo que se traduce en una mejor sensibilidad.

Una mejor sensibilidad se traduce en que se puede recibir la señal desde una distancia mayor, lo que implica una mejor cobertura. En teoría, por cada unidad de incremento en el SF se duplica el tiempo de transmisión para una misma cantidad de información.

Coding Rate (CR) [18]

El Coding Rate (CR) es la tasa de codificación que se refiere a la proporción de bits transmitidos que realmente transportan información.Esto se hace como técnica para agregar bits redundantes (de paridad) a la transmisión para que los errores se puedan recuperar en la recepción. Puede tomar los valores de 6/8, 4/6, etc. De manera que, si CR = 4/8, estamos transmitiendo el doble de bits que los que contienen información.

Ancho de banda (Bw) [19]

Ya hemos comentado que LoRa utiliza una modulación de espectro ensanchado. Este codifica la señal base con una señal de alta frecuencia que propaga la señal a través de un mayor ancho de banda, reduce el consumo energético y además aumenta las interferencias electromagnéticas.

También se ha comentado que el SF es variable. Para un ancho de banda disponible, mientras haya un mayor SF menor será la tasa de bits, de manera que aumentará el tiempo de transmisión y reducirá la batería del dispositivo transmisor.

La tasa de bits depende del ancho de banda y el spreading factor, siguiendo la ecuación:

$$Tasadebit = SF * \frac{BW}{2^{SF}} \tag{3.1}$$

LoRa permite 3 anchos de banda diferentes, 125KHz, 250KHz y 500KHz, los cuales están establecidos por las agencias reguladoras, que también regulan el SF.

3.2 Aplicaciones LoRa.

LoRa es adecuado para construir canales de comunicación de largo alcance que requieran bajas velocidades de datos. Las redes de sensores inalámbricos LoRa se pueden utilizar para crear una amplia gama de aplicaciones. Algunos ejemplos son [13]:

- Procesamiento agrícola
- Seguimiento de la contaminación del aire
- Seguimiento de activos
- Seguimiento de ganado
- Gestión energética y sostenibilidad
- Detección de caídas
- Detección de fuego
- Gestión de flotas
- Seguimiento de flotas
- Seguridad de casa
- Gestión de la calidad del aire interior
- Gestión de la temperatura industrial
- Detección de presencia de líquidos
- Localización de vehículos y carga robados
- Supervisión de refrigeradores médicos
- Gestión de aparcamientos
- Agricultura de precisión
- Mantenimiento predictivo
- Detección de fugas de radiación
- Calidad del envío
- Seguimiento de activos domésticos inteligentes

- Riego inteligente
- Iluminación inteligente
- Estacionamiento inteligente
- Monitoreo de flujo de tanques
- Gestión de residuos
- Monitoreo de flujo de agua
- Gestión y protección del agua
- Monitorización inalámbrica del nivel de gas

3.3 Cobertura.

Una sola puerta de enlace puede cubrir toda la cobertura de una ciudad entera o muchos kilómetros cuadrados. La cobertura depende en gran medida de los obstáculos que pueda haber tales como edificios o árboles, el medio ambiente y de factores técnicos como las interferencias provocadas por antenas de radio.

Veamos un ejemplo del uso de LoRa y el ahorro que supondría en costes: supongamos que tenemos una flota de vehículos con un sistema de seguimiento mediante rastreadores GPS. Cada vehículo transmite su ubicación periódicamente a un servidor GPS a través de la red móvil. Cada rastreador tiene un plan de datos.

Si tuviéramos 100 vehículos, tendríamos que contratar 100 planes de datos móviles. Pero, si reemplazamos cada rastreador GPS con un nodo de sensor LoRa y algunas puertas de enlace LoRa, solo necesitaremos planes de datos móviles para las puertas de enlace, que podrían ser 10 por ejemplo (dependiendo de la expansión geográfica que queramos alcanzar). Por lo que con esta implementación se reduciría el costo de datos móviles.

3.4 Arquitectura de una red LoRa.

La arquitectura de la red está basada en:

- **Nodos**: dispositivos finales que envían y reciben datos. Se les pueden conectar sensores, etc.
- **Gateway (puerta de enlace)**: dispositivo intermedio que actúa como plataforma entre la red LoRa e internet, permitiendo conectar los nodos con el Network Server. Se pueden considerar como convertidores o puntos de acceso. Cuentan con una dirección IP para enviar los datos a la Network.
- **LoRaWAN Network Server**: gestiona y administra los distintos dispositivos, así como las comunicaciones y permite la integración con los terceros.

En la figura 3.5 podemos ver un ejemplo de arquitectura de una red LoRa.

3.5 Redes de área amplia de bajo consumo: LPWAN.

Las redes LoRa se consideran redes de área amplia y de baja potencia (LPWAN). Los nodos pueden funcionar con batería cuya vida útil ronda los 10 años. Los nodos transmiten los datos en pequeñas cantidades a largas distancias y en algunas ocasiones varias veces por hora.

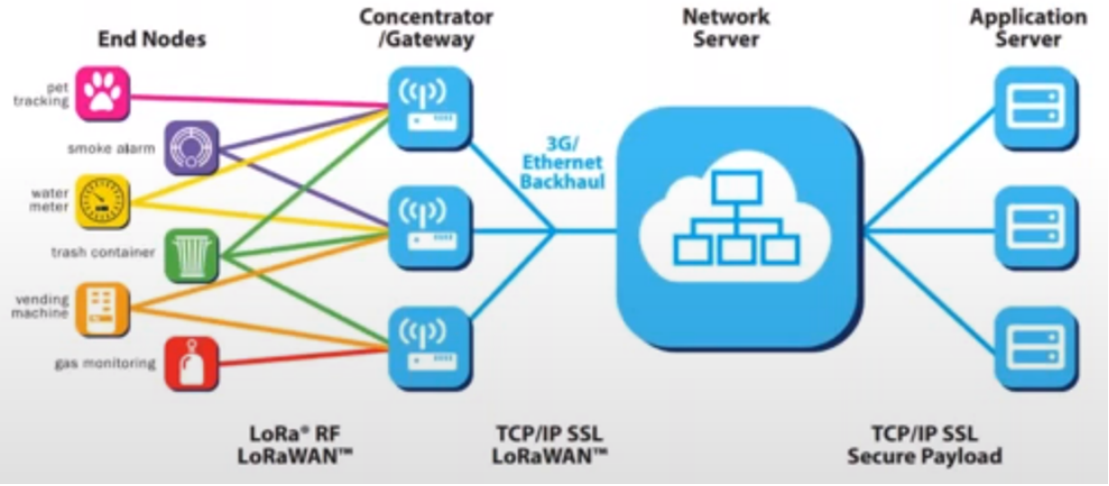


Figura 3.5 Esquema de la arquitectura de una red LoRa [19].

LPWAN (Low Power Wide Area: gran rango y bajo consumo) es una tecnología de red de área amplia inalámbrica que está especializada para dispositivos de interconexión con conectividad de baja tasa de datos, centrándose en el alcance y la eficacia energética. Por características de penetración de las ondas electromagnéticas mayormente se trabaja en el espectro subGHz: 433, 868 y 915 MHz para LoRa.

LoRa es una tecnología inalámbrica LPWAN con gran alcance geográfico (kilómetros) y con un bajo consumo eléctrico. La ventaja es que consume mucho menos y tiene una gran distancia.

LoRaWAN es un protocolo de red que utiliza LoRa en las redes LPWAN. Su función es la de realizar comunicación y administración a los dispositivos LoRa.

En la figura 3.6 podemos ver un esquema de lo que serían las capas de la arquitectura de una red LoRa.

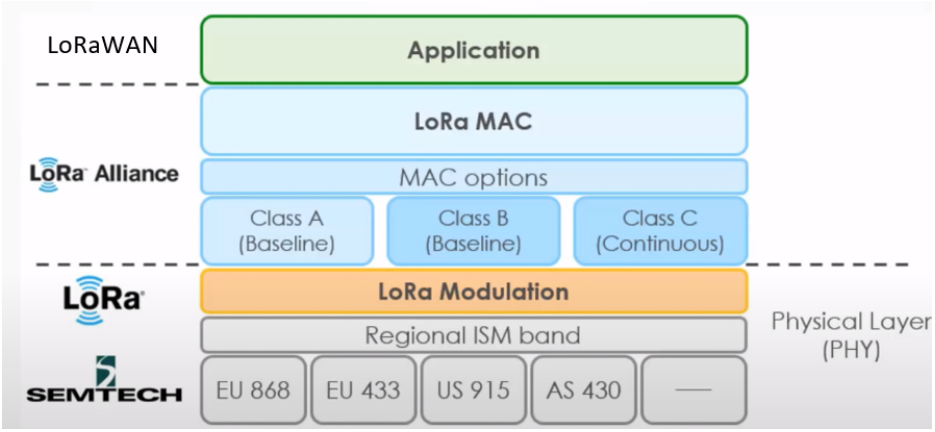


Figura 3.6 Esquema de las capas de la arquitectura de una red LoRa [19].

3.6 LoRaWAN.

Long Range Wide Area Network (LoRaWAN) es el protocolo de comunicación y la arquitectura del sistema para la red, mientras que la capa física LoRa habilita el enlace de comunicación de largo alcance. LoRaWAN influye en lo siguiente:

- Vida útil de la batería del nodo
- Capacidad de la red
- Calidad de servicio
- Seguridad
- Aplicaciones atendidas por la red

LoRaWAN consta de dispositivos finales, gateways, un servidor de red y servidores de aplicaciones. En una red LoRaWAN los datos transmitidos por un nodo final generalmente son recibidos por múltiples puertas de enlace. Cuando se reciben los datos, el Gateway reenviará el paquete recibido al servidor de red a través de la red móvil, Ethernet, Wi-Fi o satélite.

El software que se ejecuta en los Gateway es responsable de enviar cualquier paquete de datos entrante al servidor de red. Este software se conoce como reenviador de paquetes (packet forwarder).

Posteriormente, el servidor de red envía y recibe mensajes LoRaWAN desde y hasta dispositivos y se comunica con los servidores de aplicaciones descendentes.

El servidor de aplicaciones es el destino de los datos de la aplicación del dispositivo enviados como carga útil en los mensajes LoRaWAN.

3.6.1 Packet forwarder (reenviador de paquetes).

El packet forwarder es un software que se ejecuta en la puerta de enlace (gateway) LoRa. Permite que el concentrador LoRa transmita y reciba paquetes LoRa para enlaces ascendentes y descendentes desde los nodos finales a los servidores de red y desde los servidores de red a los nodos finales (bidireccional).

Existen packets forwarders de un solo canal o multicanal, siendo estos últimos los únicos compatibles con LoRaWAN.

Un packet forwarder se encargará de reenviar los paquetes LoRa recibidos por el módulo LoRa al servidor de red a través del enlace IP/UDP y emite paquetes LoRa que envía el servidor de red.

3.7 Ejemplo de conexión.

El sensor va conectado normalmente a un host (Arduino, STM32, PIC, etc) y después a través de una conexión UART/SPI se conecta al dispositivo radio el cual hace la transmisión de la información.

Se envía la información y se conecta al host que hace de gateway. LoRa permite que se conecten dispositivos entre sí sin necesidad de gateway (el gateway será necesario para conectarlo a internet).

En la figura 3.7 tenemos un ejemplo de conexión.

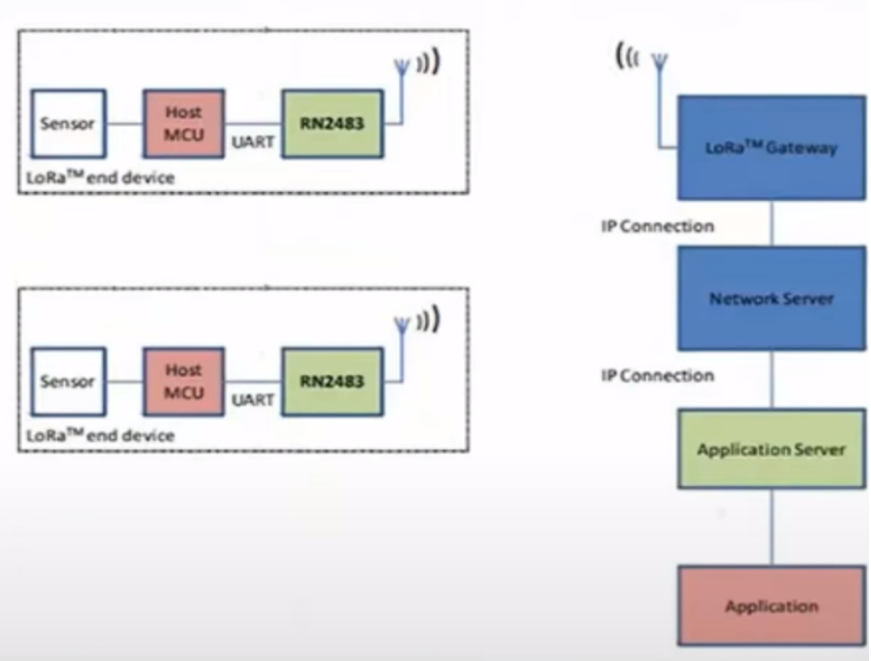


Figura 3.7 Ejemplo de conexión de la arquitectura de una red LoRa [19].

3.8 Atributos de LoRa/LoRaWAN

Algunos atributos que tiene LoRaWAN son:

- La comunicación entre nodos LoRa y gateway utiliza una modulación CSS.
- La red es de largo alcance. Generalmente es de unos 10 Km, pero se pueden hacer distancias más largas.
- El rendimiento se adapta al alcance, ancho de banda y velocidad de datos.
- La velocidad rondará de 0.25 kbps a 11kbps, afectando al alcance y al tamaño de los paquetes.
- La velocidad está relacionada con el spreading factor (SF), que como ya hemos dicho influye en la cantidad de datos redundantes existentes.
- Los dispositivos solo se comunican cuando hay datos para ser enviados. Esto hace que LoRaWAN sea de bajo consumo.
- LoRa puede operar su propia red de radio LoRaWAN sin necesidad de pagar espacio aéreo, por lo que hay un bajo costo.
- LoRaWAN es un protocolo seguro. La capa de seguridad de la red garantiza la autenticidad del nodo en la red. La seguridad de la capa de aplicación maneja el cifrado de datos entre los nodos y el servidor de la aplicación para que los mensajes no puedan leerse o interferirse en tránsito.
- Este protocolo no sirve para la navegación web, está diseñado para pequeñas cantidades de datos de dispositivos simples como sensores, luego solo se transmitirá una baja tasa de datos.

3.9 Clases de dispositivos (nodos).

En LoRaWAN diferenciamos 3 tipos de nodos dependiendo de su funcionamiento:

- **Clase A**: es el tipo más soportado por casi todos los dispositivos. Entra en modo escucha solo después de enviar un dato al gateway, por lo que es el que tiene un mayor ahorro de energía. Gracias a esto, es ideal para dispositivos que usan una batería.

- **Clase B**: en esta clase, los tramos de recepción son establecidos en unos tiempos determinados.
- **Clase C**: esta clase de dispositivos tienen su radio encendida la mayor parte del tiempo, de manera que puede recibir información siempre menos cuando está transmitiendo un mensaje. Este tipo de dispositivos está diseñado para dispositivos alimentados por la red eléctrica.

3.10 Frecuencias LoRaWAN.

LoRa Alliance define perfiles de frecuencia regionales para operar LoRaWAN para diferentes regiones regulatorias en todo el mundo. En la tabla 3.1 se enumeran las diferentes frecuencias permitidas en algunos países.

Tabla 3.1 Frecuencias LoRa por país [13].

País	Banda/Canal	Plan de canales
EEUU	902-928 MHz	US902-928, AU915928
Reino Unido	433.05-434.79 MHz 863-873 MHz 918-921 MHz	EU433 EU863-870 otro
Canadá	902-928 MHz	US902-928, AU915-928
Australia	915-928 MHz	AU915-928, AS923
India	865-867 MHz	IN765-867
Francia	433.05-434.79 MHz 863-873 MHz	EU433 EU863-870
Sri Lanka	433.05-434.79 MHz	EU433
España	863-873 MHz	EU863-870

4 Inter Integrated Circuits (I2C)

En este proyecto, para conectarnos al sensor de temperatura y humedad, tendremos que hacer uso del protocolo I2C. El bus Inter Integrated Circuits (I2C) es usado para la interconexión de dispositivos mediante un bus serie. Fue desarrollado por Philips® a principios de los 80 para conectar periféricos con una CPU. Se pretendía simplificar las conexiones entre los periféricos y aumentar la protección frente al ruido [14].

4.1 Características del I2C.

Las características principales del I2C son:

- Se trata de un bus de comunicaciones síncrono el cual está formado por 2 hilos: Serial Data Line (SDA) para los datos y Serial Clock Line (SCL) para el reloj de sincronización.
- La velocidad de transmisión *standard* es de hasta 100kbits/s, pero, en modo *fast*, puede alcanzar los 400Kbits/s. Si fuera necesario, también existe el modo *high-speed* que alcanza una velocidad de hasta 3.4Mbits/s
- Cada dispositivo que esté conectado al bus I2C tiene una dirección única.
- La distancia y el número de dispositivos que se pueden conectar está limitada por la capacidad del bus.
- El protocolo seguido para acceder al bus es el Master-Slave, pudiendo ser multimaestro. Si se diera la situación de que hubiera varios maestros en el bus, se hará un arbitraje que asegura que en cada instante sólo hay un dominante.

4.2 Conexión a nivel físico.

Para una correcta conexión todos los dispositivos tienen que estar conectados a las mismas líneas SDA y SCL.

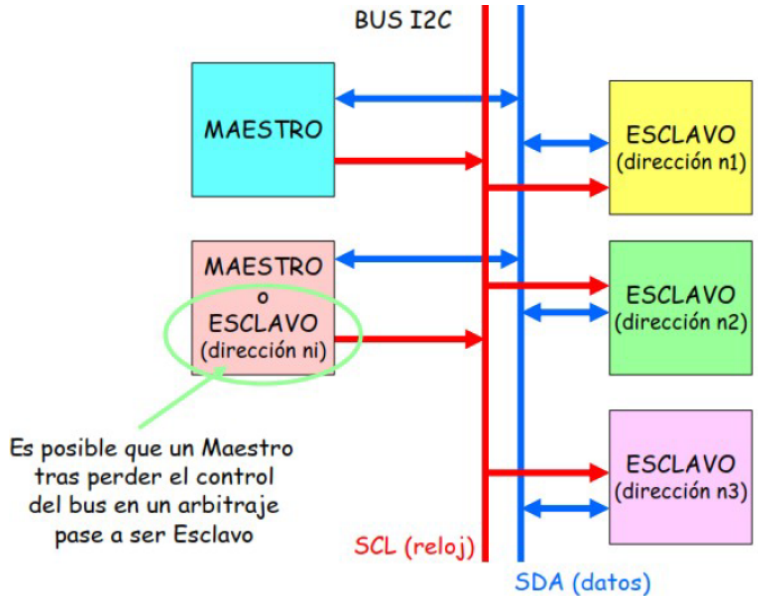


Figura 4.1 Esquema de conexión física [15].

4.3 Nivel de enlace.

Para el intercambio de información, a nivel de enlace usaremos el protocolo de acceso al medio maestro-esclavo. En este protocolo, el maestro controla toda la comunicación, de manera que genera la señal de reloj del bus SCL, inicia y termina la comunicación, da direcciones a los esclavos y establece en qué sentido será la comunicación.

En este protocolo, se necesita que por cada byte enviado se confirme la recepción por parte del destinatario.

Los datos irán por la línea SDA, de manera que por cada bit de datos será necesario un ciclo de SCL. Cabe destacar que los datos sólo pueden cambiar cuando SCL está a nivel bajo.

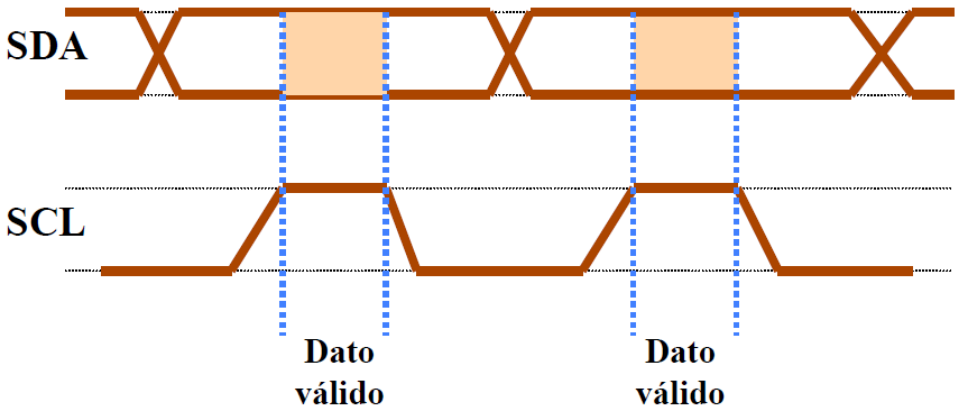


Figura 4.2 Ejemplo de comunicación I2C [14].

La unidad básica de transmisión será el byte, de modo que por cada byte enviado necesitaremos una confirmación (ACK) por parte del destinatario.

Para el ACK, el destinatario mantiene la línea SDA a nivel bajo durante el tiempo equivalente a 1 bit (1 ciclo de reloj). En el caso de que no fuera así, sería un NACK (no recibido).

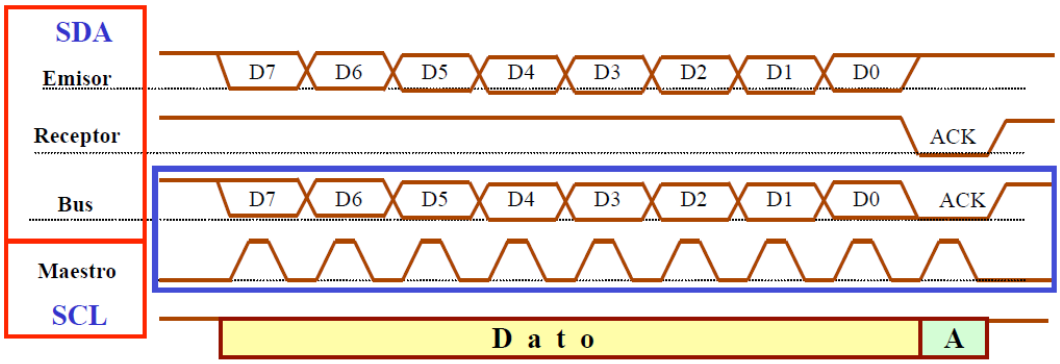


Figura 4.3 Ejemplo de envío de datos en I2C [14].

Para iniciar la transmisión, el maestro dará un flanco de bajada en SDA con SCL a nivel alto, tal y como se muestra en la figura 4.4.

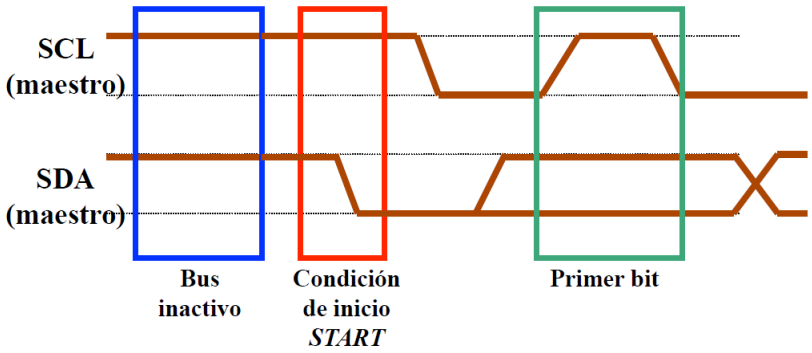


Figura 4.4 Ejemplo del inicio de la comunicación en I2C [14].

Y, para finalizar la transmisión, el maestro da un flanco de subida en SDA con SCL a nivel alto, como podemos ver en la figura 4.5.

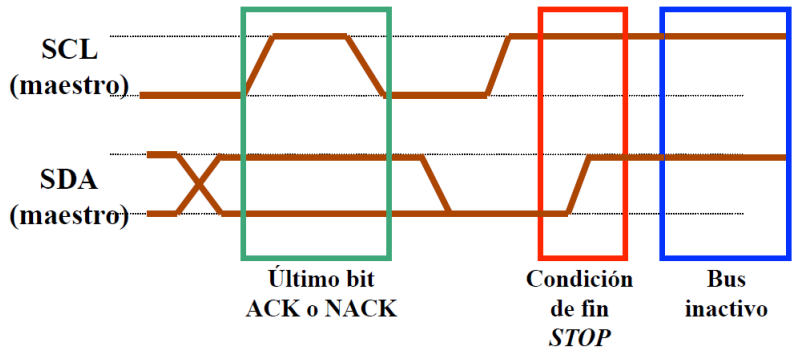


Figura 4.5 Ejemplo del fin de la comunicación en I2C [14].

Para realizar el intercambio de datos, lo primero que tiene que hacer el maestro tras dar el inicio de la transmisión es indicar la dirección de quién va a mandar información por el bus SDA, luego los 7 primeros bits serán la dirección del esclavo y un bit donde se indicará si será lectura (bit a nivel alto) o escritura (bit a nivel bajo).

En el momento que el maestro manda la dirección, todos los dispositivos conectados a la línea SDA escuchan para determinar si la dirección colocada en el bus es la suya.

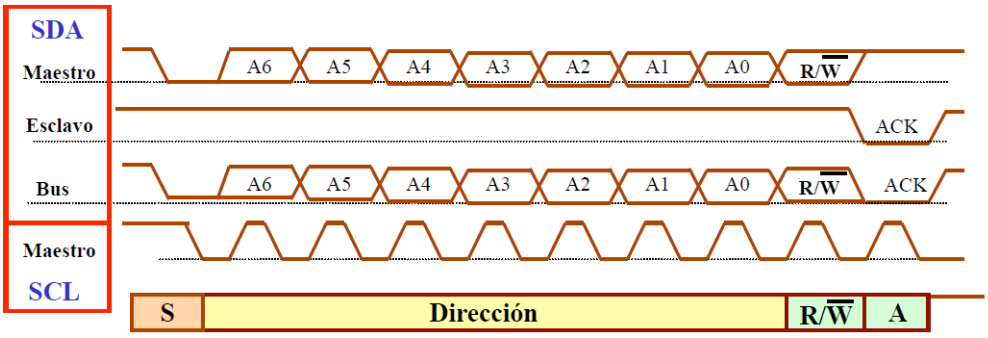


Figura 4.6 Ejemplo del envío de dirección en I2C [14].

Por último, en la figura 4.7 tenemos un esquema en el que el máster envía datos a un esclavo y en la figura 4.8 un esquema donde el máster es el que recibe datos del esclavo.

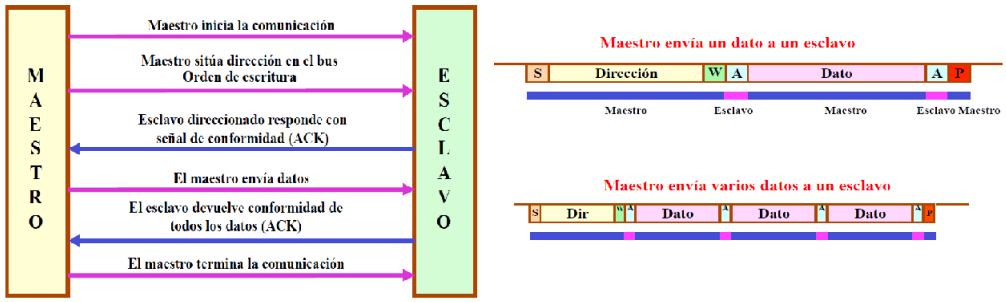


Figura 4.7 Esquema de envío de datos de máster a esclavo en I2C [14].

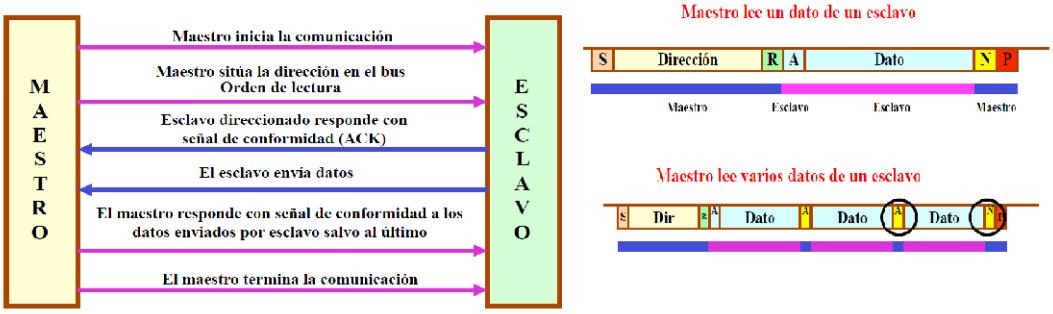


Figura 4.8 Esquema de envío de datos del esclavo al maestro en I2C [14].

5 Materiales

En este capítulo vamos a dar a conocer los diferentes materiales que hemos utilizado para la implementación del sensor IoT. Estos materiales los podemos clasificar en hardware y software.

5.1 Hardware.

Como hardware hemos usado 2 placas STM32WL55JC de **STMicroelectronics®**, además de un sensor SHT20.

5.1.1 Placa de desarrollo STM32WL55JC [16].

Las placas de desarrollo de las que hacemos uso son fabricadas por STMicroelectronics®, una importante empresa holandesa que desarrolla, fabrica y comercializa una gran parte de los circuitos integrados usados mundialmente.

Hemos elegido los dispositivos STM32WL55JC debido a su capacidad de comunicación mediante LoRa, ya que disponen de un módulo integrado que además de poder usar la modulación mencionada, también dispone de otros tipos de modulaciones como: (G)FSK, (G)MSK, y BPSK.

Estos dispositivos están diseñados para funcionar a una muy baja potencia con un gran rendimiento. Además, posee dos núcleos ARM® Cortex ®:

- **M4**: núcleo de 32 bits que funciona a una frecuencia máxima de 48 MHz y hace uso de un conjunto de instrucciones DSP, luego está optimizado para aplicaciones que necesiten operaciones numéricas de muy alta velocidad.
- **M0**:sirve para complementar al M4.

Estos dispositivos poseen varios mecanismos de protección para la memoria FLASH incorporada y SRAM, entre ellos tiene protección de lectura y escritura.

Las placas tienen un convertidor analógico digital (ADC) de 12 bits, un convertidor digital analógico (DAC) de 12 bits de baja potencia y dos comparadores de baja potencia asociados con un generador de voltaje de referencia de una alta precisión.

Además, estos dispositivos incorporan un Real Time Clock (RTC) de baja potencia con un contador de activación de 32 bits, un temporizador monocanal de 16 bits, dos temporizadores de cuatro canales de 16 bits, un temporizador de 32 bits de cuatro canales y tres temporizadores de 16 bits de ultra baja potencia.

Estos dispositivos también incorporan dos controladores DMA (7 canales cada uno) que permiten cualquier combinación de transferencia entre la memoria (memoria Flash, SRAM1 y SRAM2) y el

periférico, utilizando el DMAMUX1 para un mapeo flexible de canales DMA.

También cuentan con las interfaces de comunicación estándar y avanzada que se enumeran en las tablas 5.1 y 5.2.

Tabla 5.1 Conjunto de características más importantes de las placas STM32WL55JC agrupadas según su ámbito (TABLA 1/2) [16].

Radio	- Rango de frecuencias entre 150MHz y 960MHz - Tipo de modulaciones: LoRa®, (G)FSK, (G)MSK y BPSK - Sensibilidad de recepción: –123 dBm para 2-FSK (a 1,2 Kbit / s), –148 dBm para LoRa (a 10,4 kHz, para un factor de dispersión 12) - Transmisor de alta potencia de salida, programable hasta +22dBm - Transmisor de baja potencia de salida, programable hasta +15 dBm - Cumple con las siguientes regulaciones de radiofrecuencia como ETSI EN 300220, EN 300113, EN 301166, FCC CFR 47 Part 15, 24, 90, 101 y ARIB STD-T30, T-67, T-108 de Japón - Compatible con los protocolos normalizados o propietarios tales como LoRaWAN, Sigfox, W-MBus y más
Plataforma de ultrabajo consumo	- Fuente de alimentación de 1.8 V a 3.6 V - Rango de temperatura de –40 ° C a +105 ° C - Modo de apagado: 31 nA (V DD = 3 V) - Modo de espera (+ RTC): 360 nA (V DD = 3 V) - Modo Stop2 (+ RTC): 1.07 µA (V DD = 3 V) - MCU en modo activo: <72 µA / MHz (CoreMark®) - RX en modo activo: 4,82 mA - TX en modo activo: 15 mA a 10 dBm y 87 mA a 20 dBm
Núcleo (Core)	- CPU Arm®Cortex®-M4 de 32 bits - Arm®Cortex®-M0 + CPU de 32 bits
Seguridad e identificación	- Cifrado de hardware AES de 256 bits - Generador de números aleatorios verdaderos (RNG) - Protección del sector contra operaciones de lectura - Unidad de cálculo CRC - Identificador de dispositivo único - Identificador de matriz único de 96 bits - Acelerador de clave pública de hardware (PKA) - Servicios de gestión de claves - Capa MAC segura de sub-GHz - Actualización segura de firmware (SFU) - Instalación segura de firmware (SFI)

Tabla 5.2 Conjunto de características más importantes de las placas STM32WL55JC agrupadas según su ámbito (TABLA 2/2)[16].

Gestión del suministro y reposición	- Convertidor reductor SMPS integrado de alta eficiencia - Conmutador inteligente SMPS a LDO - BOR (reinicio de apagón) ultraseguro y de bajo consumo con 5 umbrales seleccionables - POR / PDR de ultrabajo consumo - Detector de voltaje programable (PVD) - Modo V BAT con RTC y registros de respaldo de 20x32 bytes
Fuentes de reloj	Oscilador de cristal de 32 MHz - Soporte TCXO: tensión de alimentación programable - Oscilador de 32 kHz para RTC con calibración - RC interno de alta velocidad de 16 MHz recortado en fábrica - RC interno de baja potencia de 32 kHz - RC interno de varias velocidades de baja potencia de 100 kHz a 48 MHz - PLL para CPU, ADC y relojes de audio
Núcleo (Core)	- CPU Arm®Cortex®-M4 de 32 bits - Arm®Cortex®-M0 + CPU de 32 bits
Memorias	- 256-Kbyte de memoria flash - 64-Kbyte RAM - 20x32-bit registro de respaldo - Cargador de arranque compatible con interfaces USART y SPI - Actualización de firmware OTA (over-the-air) - Protección del sector contra operaciones de lectura/escritura
Periféricos analógicos	- ADC de 12 bits a 2,5 Msps, hasta 16 bits con sobremuestreo de hardware, rango de conversión de hasta 3,6 V - DAC de 12 bits, muestreo y retención de baja potencia - 2x comparadores de potencia ultrabaja
Periféricos del sistema	- Buzón y semáforos para la comunicación entre Cortex®-M4 y Cortex®-M0 + firmware
Controladores	- 2x controlador DMA compatible con ADC, DAC, SPI, I2C, LPUART, USART, AES y temporizadores - 2x USART (ISO 7816, IrDA, SPI) - 1x LPUART (bajo consumo) - 2x SPI 16 Mbit/s - 3x I2C (SMBus / PMBus) - 2x temporizador de 1 canal de 16 bits - 1 temporizador de 4 canales de 16 bits - 1x temporizador de 32 bits y 4 canales - 3x temporizador de 16 bits de ultra bajo consumo - 1x RTC con contador de activación de subsegundos de 32 bits - 1x SysTick independiente - 1x perro guardián independiente - 1x perro guardián de la ventana
Otros	- Hasta 43 E/S, la mayoría tolera 5 V + Apoyo al desarrollo + Depuración de cable serie (SWD), JTAG - Capacidades de disparo cruzado de CPU dual - Todos los paquetes cumplen con ECOPACK2

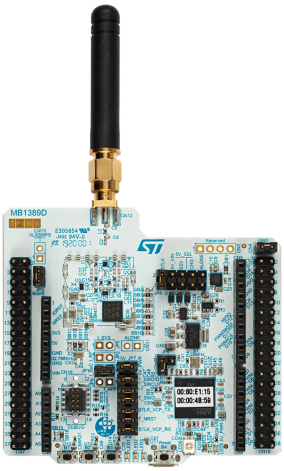


Figura 5.1 Imagen física de la placa STM32WL55JC 16].

La comunicación con las placas se hace mediante USB, que mediante el entorno de desarrollo STM32CubeIDE se cargarán los programas en las placas.

Para la comunicación con la consola del ordenador usaremos UART y para la comunicación con el sensor de temperatura y humedad usaremos I2C.

5.1.2 Sensor de temperatura y humedad SHT20.

Como sensor de temperatura y humedad usaremos el sensor SHT20 de sensirion®. Es un sensor estándar ampliamente utilizado.

Figura 5.2 Vista física del sensor SHT20.

La humedad se mide con un sensor capacitivo y la temperatura se mide con un sensor de banda de rechazo.

De toda la familia de sensores SHT2X, el SHT20 es el de menor precisión y coste, pero para nuestra aplicación es más que suficiente, de manera que tiene un margen de error de 3ºC en la temperatura medida y un 3 % en la medida de la humedad relativa.

Como ya hemos comentado en el apartado anterior, el sensor SHT20 se comunicará con la placa a través de I2C, de manera que lo conectaremos con los correspondientes pines a la placa que hará de transmisor.

Por otro lado, cabe destacar que su rango de temperaturas de funcionamiento es de -40ºC a +125ºC y su resolución es de 0.01ºC en la temperatura y 0.04 % en la humedad relativa.

Para la utilización y adaptación del mismo a la placa, lo hemos integrado a una placa con conexiones y hemos hecho uso de un recubrimiento de plástico, para protegerlo de agentes externos.

Figura 5.3 Módulo utilizado para incorporar el sensor SHT20 a la placa.

5.2 Software.

En este apartado, destacaremos el uso de 2 programas, STM32CubeIDE y CoolTerm.

5.2.1 Entorno de desarrollo integrado para STM32: STM32CubeIDE.

Como bien dice en la página de STMicroelectronics para descargar dicho entorno, STM32CubeIDE (a partir de ahora Cube por abreviar) es una plataforma de desarrollo C/C++ basada en Eclipse.

Además, gracias a la incorporación de STM32CubeMx podemos configurar las funcionalidades de creación de proyectos de una manera rápida y sencilla, instalando así lo necesario para empezar a programar la placa seleccionada. En Cube tenemos la posibilidad de configurar periféricos, generar código, generar código de compilación y funciones de depuración para microcontroladores y microprocesadores de la familia STM32. Además, está basado en GDB para la depuración.

Como extra, al seleccionar la placa que tenemos para empezar el proyecto se instalan una serie de ejemplos que puedes usar para ver las funcionalidades disponibles en la placa.

En nuestro caso, partiremos de uno de los ejemplos dados para elaborar nuestro proyecto.

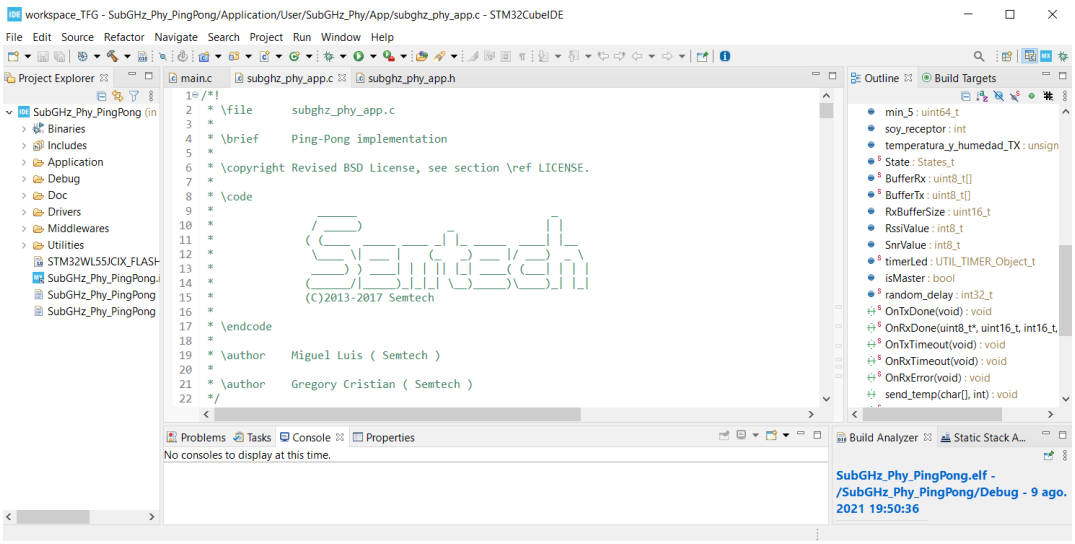


Figura 5.4 Vista general del entorno de desarrollo integrado STM32CubeIDE.

5.2.2 CoolTerm.

CoolTerm es el software que hemos utilizado como terminal de puerto serie. Se utiliza principalmente para proyectos donde se necesita intercambiar datos con hardware conectado a puertos serie, como es nuestro caso.

Permite guardar configuraciones de los puertos para la posterior inicialización, permitiendo así una conexión más rápida. También permite la captura de contenido, de este modo podemos guardar todo lo que va sucediendo en un fichero. Haremos uso de esto para guardar un registro de temperaturas y humedades relativas de un mismo día, pudiendo además indicar la fecha y la hora del momento.

En nuestro proyecto nos ha permitido estar controlando en todo momento qué sucedía en nuestras placas cuando se estaban comunicando, pudiendo corregir errores y depurar código eficientemente.

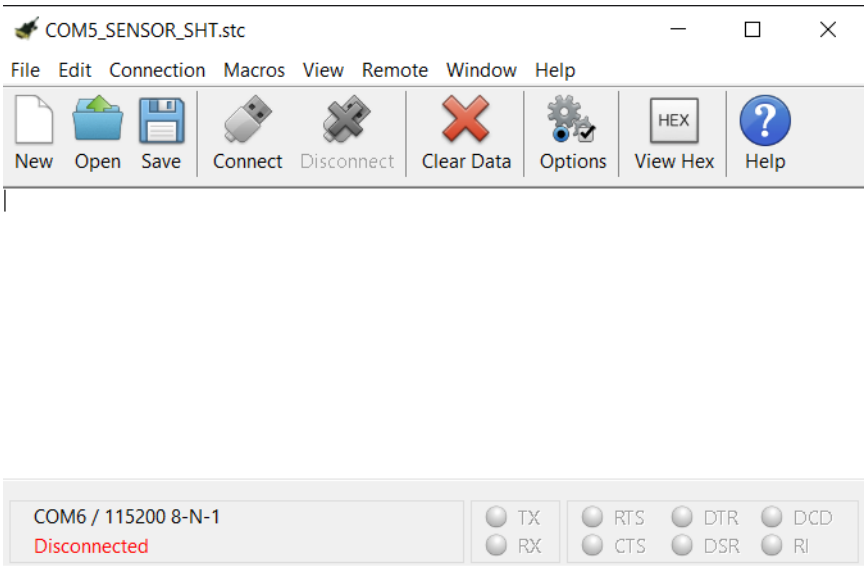


Figura 5.5 Ventana de inicio del software CoolTerm.

6 Metodología y Desarrollo

Este capítulo recoge todo el proceso que se ha llevado a cabo para la elaboración de nuestro proyecto, desde el estudio previo hasta llegar a la versión final en pleno funcionamiento.

Hablaremos de toda la metodología utilizada, las decisiones tomadas y haremos pequeñas menciones a código (en el apéndice A se incluirá archivos de código más completos para su posible consulta).

Se ha seguido un método iterativo, de manera que cada versión trataba de agregar una nueva funcionalidad al proyecto hasta alcanzar finalmente lo deseado.

Tras la definición del objetivo final, hemos dividido el problema en pequeñas modificaciones previas hasta cumplir el objetivo.

Los pasos intermedios (versiones) del proyecto, han sido:

- **Versión 0**: Estudio y entendimiento del ejemplo proporcionado por STMCubeMX.
- **Versión 1**: Conexión UART.
- **Versión 2**: Conexión con SHT20 (sensor de temperatura y humedad relativa).
- **Versión 3**: Implementación de retardos.
- **Versión 4**: Conexión entre placas.
- **Versión 5**: Diferenciación de códigos entre placas.

Por otro lado, dentro de cada versión, hemos seguido los siguientes pasos:

- Paso 1: Planteamiento de la modificación del proyecto. Nos hemos basado en las versiones previas que hemos elegido.
- Paso 2: Estudio y búsqueda de información. Hemos realizado búsquedas por la web para encontrar información relevante sobre cada modificación para poder llevarla a cabo.
- Paso 3: Aplicación y adaptación de la información disponible. Tras cada búsqueda hemos incorporado las consecuentes modificaciones a nuestro proyecto.
- Paso 4: Comprobación y depuración de errores. Cada vez que hemos incorporado/modificado código hemos tenido que depurarlo y confirmar el correcto funcionamiento.
- Paso 5: Puesta a punto. Una vez conseguido la modificación, hemos comprobado su correcto funcionamiento y hecho los correspondientes comentarios para su explicación en el código.

6.1 Versión 0: Estudio y entendimiento del ejemplo proporcionado por STM-CubeMX.

Como se mencionó en el capítulo de materiales, tras elegir la placa en STMCubeMX se descargan una serie de drivers y proyectos de ejemplo.

En nuestro caso, tras seleccionar la placa STM32WL55JC se nos descargan una series de proyectos ejemplo, entre ellos destaca el proyecto **SubGHz_Phy_PingPong**.

A grandes rasgos, este proyecto interconecta las dos placas que posean el código gracias a su módulo de antena. Lo hace bien a través de una modulación LoRa o mediante una modulación FSK.

Viene configurada la modulación LoRa de manera predeterminada, por lo que no tendremos que tocar el proyecto en ese aspecto.

Su funcionamiento está basado en tres funciones que se invocan desde la función principal "main" (apéndice A). Las veremos a continuación.

6.1.1 Función `void MX_SubGHz_Phy_Process(void)`

Esta función está dentro del bucle infinito `while(1)` y no tiene nada en su interior que nos pueda interesar para llegar a nuestro objetivo.

6.1.2 Función `void MX_SubGHz_Phy_Init(void)`

Esta función es invocada junto con las demás funciones para iniciar los periféricos de la placa.

Si nos introducimos dentro de ella, podremos encontrar que se hace nuevamente uso de dos funciones. Estas son `SystemApp_Init()` y `SubghzApp_Init()`, de las cuales sólo nos interesa la segunda, ya que en su interior se hace uso de la función `PingPong_Process()`, función que será la que modificaremos posteriormente para adaptarla a nuestro objetivo.

`PingPong_Process()` es una función situada en el archivo subghz_phy_app.c (apéndice A). En ella, inicialmente se reparte la función de maestro (Master) y esclavo (Slave) entre las placas.

El Master será quien envíe la cadena de caracteres "PING" y recibirá la cadena de caracteres "PONG". Una vez recibida la cadena "PONG" volverá a mandar la cadena "PING" y así de manera indefinida. De la misma manera pero al contrario, el Slave en cuanto reciba la cadena "PING", enviará la cadena "PONG" y así de manera indefinida.

Nada más arrancar el programa se asume por defecto el papel de master y se genera un retraso aleatorio denominado `random_delay`. Tras una serie de iteraciones del programa, cada placa va alternando su función de Slave y de Master, pero cada placa con un retraso diferente. Gracias a esto, llegará un punto en el que una placa sea Master y mande la cadena "PING" y la otra sea Slave y reciba dicha cadena. En ese preciso instante se realizará la conexión y se entrará de manera indefinida en el bucle de envío y recepción de "PING" y de "PONG", donde una placa hace la función de Master y otra la función de Slave.

6.2 Versión 1: Conexión UART.

Esta versión surge de la necesidad de poder analizar la información que se transmite y depurar el código desde una consola, en este caso en CoolTerm. Para ello, haremos uso del UART.

UART son las siglas de Universal Asynchronous Receiver-Transmitter y es el componente que permite la comunicación serie de la placa con el ordenador.

Antes de empezar y para facilitar las modificaciones, comentaremos las funciones implicadas con la conexión LoRa hasta la versión 4, de esta manera nada procedente con la conexión interferirá en lo que añadamos.

Una vez hecho esto, modificaremos la configuración conveniente en el elemento " **SubGHz_Phy_PingPong.ioc**" del proyecto dentro del STM32CubeIDE y posteriormente generaremos el código. De esta manera, se crea la variable `UART_HandleTypeDef huart2` y se inicializa todo lo necesario con la función `MX_USART2_UART_Init(void)` .

Antes de dar por finalizar la versión, hacemos una serie de pruebas de su correcto funcionamiento.

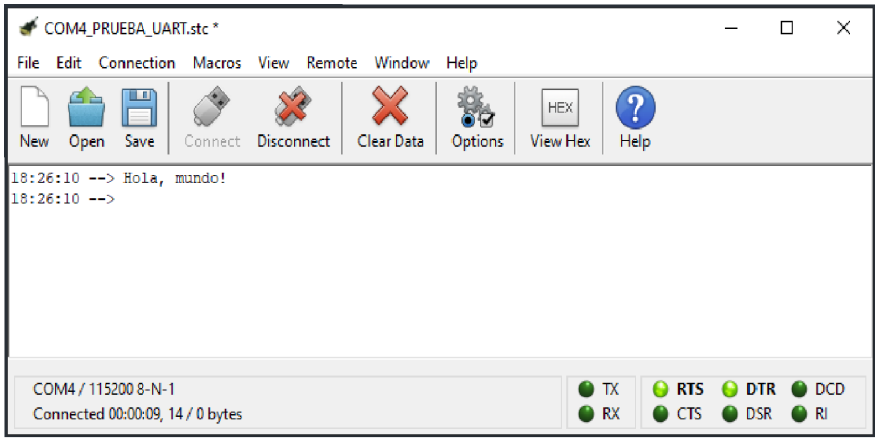


Figura 6.1 Muestra de la salida por consola del UART: Ejemplo "Hola, mundo!". Se hace uso del software CoolTerm.

A partir de ahora, con solo usar la función `HAL_UART_Transmit` pasándole los parámetros adecuados (entre ellos `huart2`) podremos transmitir información por el UART.

6.3 Versión 2: Conexión con SHT20.

A continuación se comenzó con la integración del sensor de temperatura y humedad SHT20 de sensirion®. Si observamos el datasheet de dicho sensor [B], podemos ver que utiliza una comunicación I2C.

Lo primero que hacemos es configurar los pines necesarios para poder usar dicha comunicación. Nuevamente, nos vamos al elemento " **SubGHz_Phy_PingPong.ioc**" y en conectividad configuramos los pines necesarios.

En esta ocasión, asignamos el pin PA10 a I2C1_SDA y PA9 a I2C1_SCL, de manera que, como su propio nombre indica, las líneas SDA y SCL de la comunicación I2C irán por el pin PA10 y por el pin PA9, respectivamente.

Generamos el código y obtenemos (de manera similar a como pasaba con el UART) la declaración de `I2C_HandleTypeDef hi2c1` y la inicialización del periférico I2C de la placa mediante `MX_I2C1_Init()`. Ya está lista nuestra placa para el uso del sensor, a continuación se presenta el uso de dichas funciones.

Tras una búsqueda por repositorios web, hemos encontrado unas librerías [A] que resultan útiles para el uso de dicho sensor, de manera que solamente tendremos que hacer una adaptación a lo que nos interesa.

En dichas librerías, trataremos los datos recibidos por el sensor en función de las indicaciones del datasheet y los transformaremos en datos legibles para nosotros. Destacaremos el uso de las siguientes funciones:

- `uint16_t SHT2x_GetRaw(uint8_t cmd)`: Encargada de enviar el comando adecuado al sensor y leer un valor bruto de 16 bits del mismo.
- `float SHT2x_GetTemperature(uint8_t hold)`: La cual nos devuelve la temperatura en grados centígrados (hace uso de `SHT2x_GetRaw`).
- `float SHT2x_GetRelativeHumidity(uint8_t hold)` : Nos devolverá la humedad relativa en tanto por ciento (también haciendo uso de `SHT2x_GetRaw`).

Con todo lo anterior, tras probarlo y conectar el sensor a sus correspondientes pines tal y como se muestra en la figura 6.2, podemos obtener el valor de la temperatura y la humedad relativa correctamente. Solo falta comunicarlo por el UART y podremos verlo en consola.

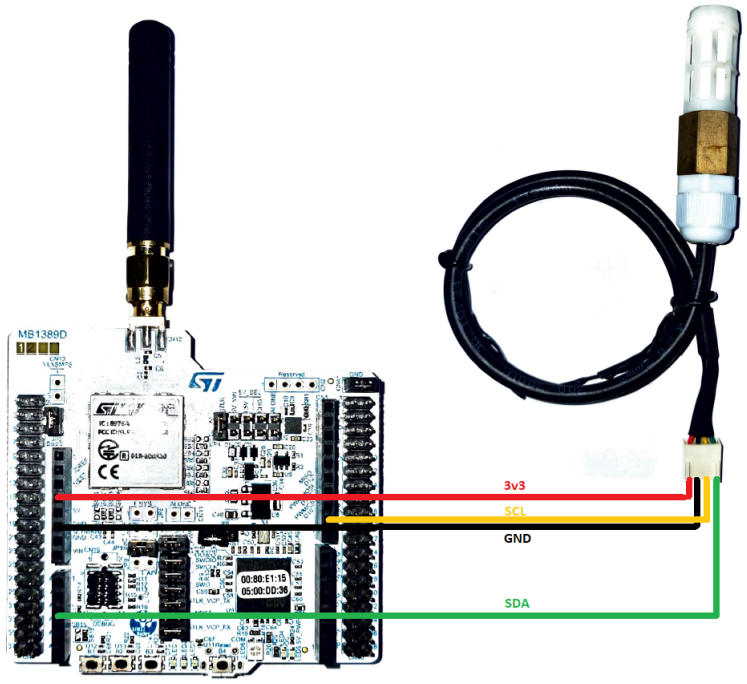


Figura 6.2 Esquema de conexiones de la placa con el sensor SHT20.

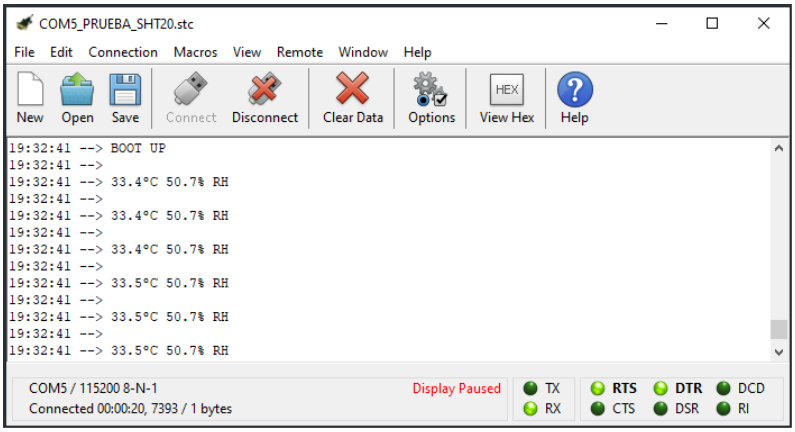


Figura 6.3 Salida del UART que muestra la temperatura y la RH en tiempo real. Se hace uso del software CoolTerm.

Como podemos observar en la figura 6.3, el sensor está tomando una temperatura de 33.5ºC y una humedad relativa de 50.7 %.

6.4 Versión 3: Implementación de retardos.

Una vez que tenemos funcionando correctamente los sensores, nuestra intención es hacer una medición en intervalos de tiempo, de manera que sólo se envíe la información pasado el intervalo de tiempo que hayamos programado.

Sin hacer cambios, ahora mismo la temperatura y la humedad relativa se transmiten en tiempo real y nuestro objetivo es que se vayan transmitiendo cada cinco minutos. Para eso bastará con hacer uso de la función `HAL_Delay(tiempo*1000)`, de esta manera bastará con pasarle como parámetro el tiempo (en segundos) que queramos que esté pausado el programa.

Aunque lo dejemos así, no funcionará, ya que el proyecto SubGHz_Phy_PingPong trae por defecto la función `HAL_Delay( )`sobrescrita en el fichero "sys_app.c".

Para solucionarlo, basta con comentar dicha función en ese fichero y ya automáticamente toma su funcionamiento habitual de la librería "HAL".

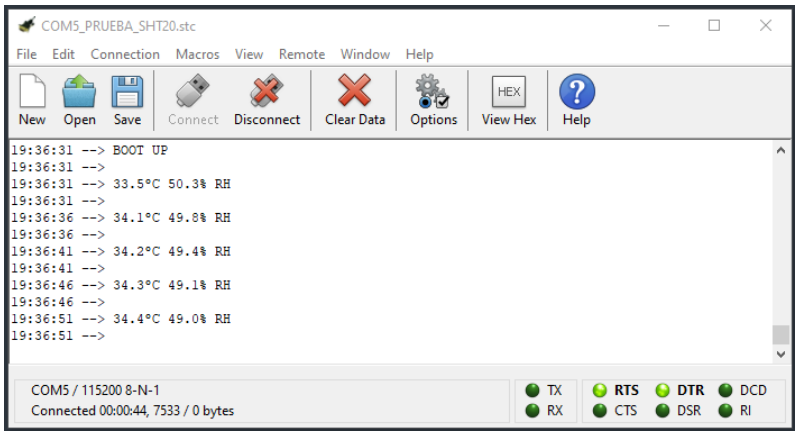


Figura 6.4 Salida del UART que muestra la temperatura y la RH para `tiempo` igual a 5 segundos. Se hace uso del software CoolTerm.

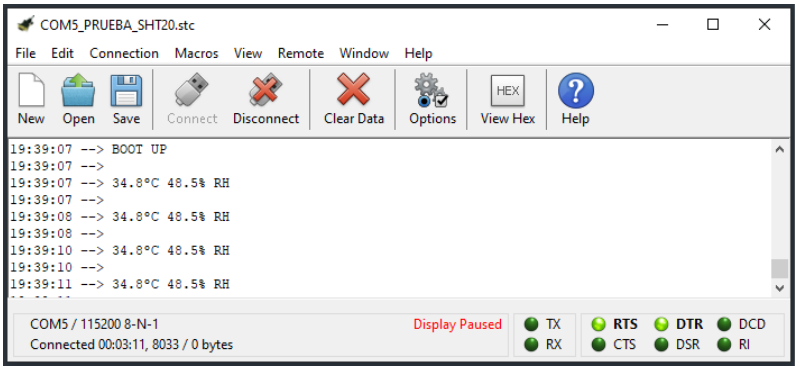


Figura 6.5 Salida del UART que muestra la temperatura y la RH para `tiempo` igual a 1 segundo. Se hace uso del software CoolTerm.

Como podemos ver en las figuras 6.4, 6.5 y 6.6, tras modificar varias veces el intervalo de tiempo, comprobamos que los retardos funcionan correctamente.

6.5 Versión 4: Conexión entre placas.

Lo siguiente que tenemos que hacer es interconectar las placas mediante LoRa. Para ello, en principio bastaría con quitar los comentarios puestos en las funciones que comentamos en la versión 1 (Punto 6.2), es decir, `void MX_SubGHz_Phy_Init(void)`y `void MX_SubGHz_Phy_Init(void)`.

Tras realizar las pruebas de conexión y comprobar un correcto funcionamiento del envío de las cadenas "PING" y "PONG" del proyecto, podemos confirmar que nuestras modificaciones no han hecho efecto en el funcionamiento del proyecto inicial.

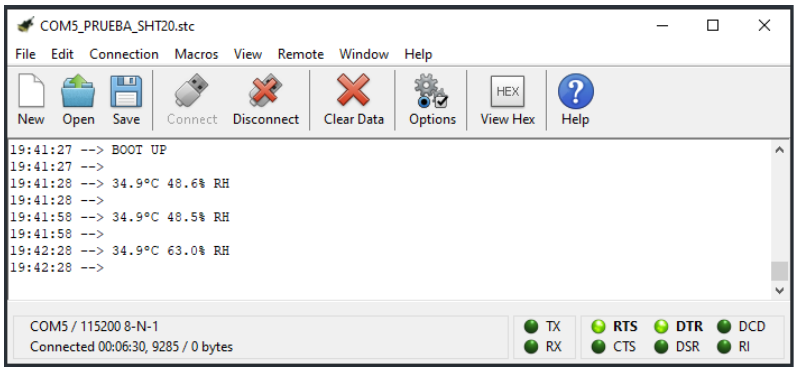


Figura 6.6 Salida del UART que muestra la temperatura y la RH para `tiempo` igual a 30 segundos. Se hace uso del software CoolTerm.

Ahora es el turno de cambiar el mensaje a enviar. Para ello , creamos una función propia denominada `void send_temp(char temp_y_hum[100], int size)` en el directorio subghz_phy_app.c donde pasaremos la cadena de caracteres con información de la temperatura y la humedad relativa. Cada vez que la llamemos desde el main.c, se introducirá la información en la variable que posteriormente se enviará por LoRa. A partir de ahora, cada vez que recibamos la palabra "PING", si estamos en la primera interacción enviaremos "PONG" a modo de confirmación de conexión y en las siguientes interacciones en vez de enviar "PONG" responderemos con la temperatura y la humedad relativa.

Se experimentaron una serie de errores con la información que se recibía del sensor en la placa transmisora, de manera que no se recibía ninguna temperatura y humedad relativa. Esto se debía a que no se inicializaba correctamente los periféricos relacionados con el sensor, por lo que hemos incluido en el `while(1)` del main.c las funciones de inicialización necesarias para el sensor, solucionando así los problemas y recibiendo perfectamente la temperatura y la humedad relativa en la placa receptora.

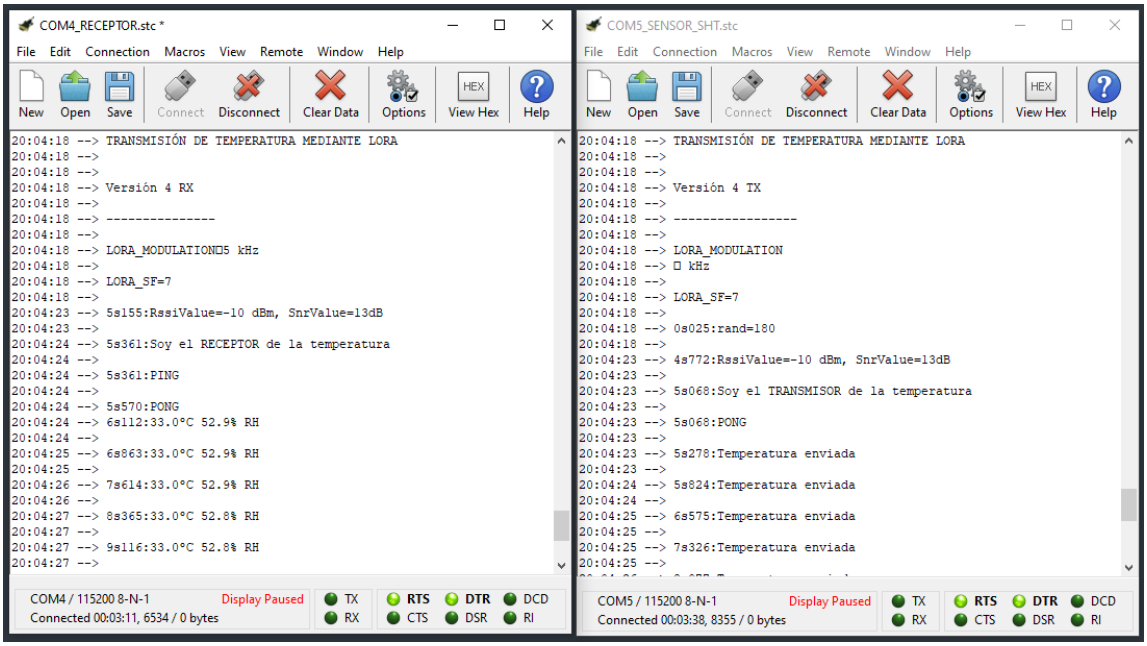


Figura 6.7 Salida del UART que muestra la temperatura y la RH recibida en el receptor y los mensajes de temperatura enviada en el transmisor. Se hace uso del software CoolTerm.

6.6 Versión 5: Diferenciación de placas.

Como se indicaba en la versión 0 (capítulo 6.5), se elige quién es el transmisor y quién es el receptor de manera aleatoria al arrancar las placas, de manera que no podíamos elegir quién era cada cual.

Para solucionar esto, hemos hecho una diferenciación de códigos, de manera que en uno de los códigos, la placa que posea dicho código, siempre será Master y no habrá posibilidad de que sea Slave (Transmisor).

Como el Slave es el transmisor, será el que tenga conectado el sensor SHT20 y, por tanto, no tocaremos el código respecto a la versión anterior, solamente haremos que siempre sea Slave y que no haya posibilidad de cambio a Master.

En el caso del Master, debido a que será el receptor de la temperatura, no tendrá conectado el sensor y solamente será el encargado de recibir la información, luego nos sobra todo lo relacionado con las funciones para la lectura del sensor. Además, modificaremos el código para que siempre sea Master y no exista la posibilidad de que se le asigne el papel de Slave.

Una vez hecha estas modificaciones, introducimos el código correspondiente a la placa que hará de receptor (Master) y el que hará de transmisor (Slave).

Con todo esto, el proyecto estaría en pleno funcionamiento a falta de modificar a nuestro interés el intervalo de tiempo de las medidas. En la siguiente figura podemos ver que el intervalo de tiempo es de 5 minutos.

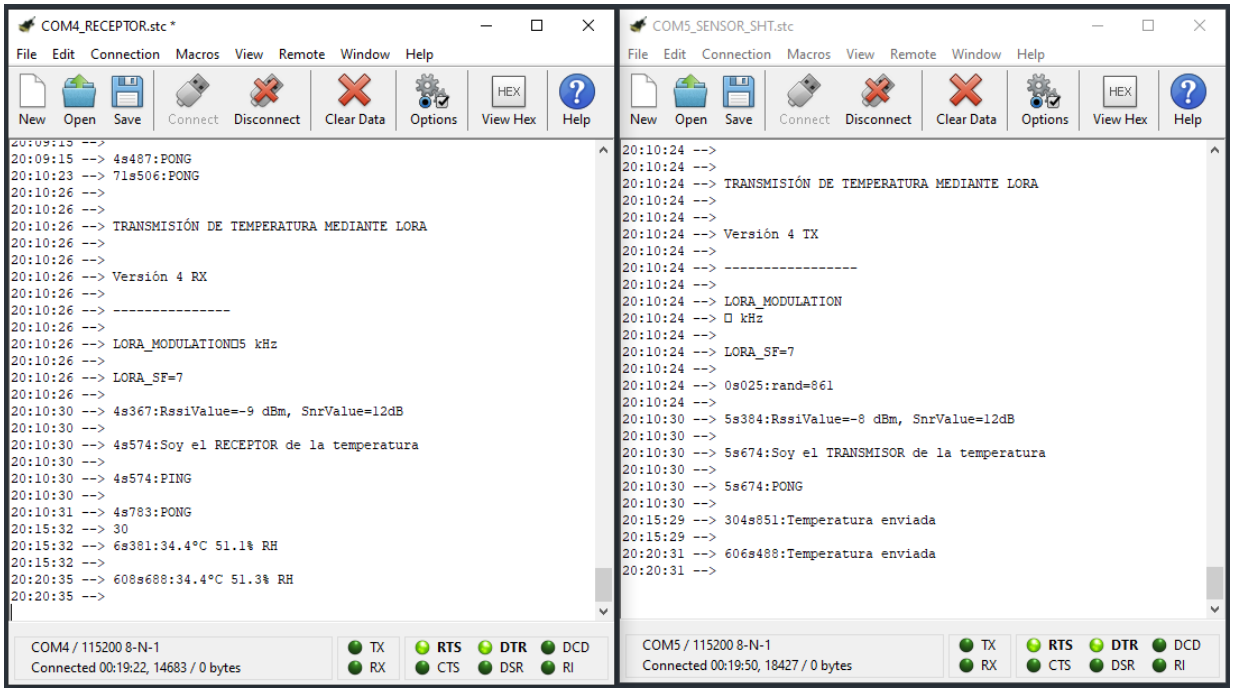


Figura 6.8 Salida del UART que muestra la temperatura y la RH recibida en el receptor y los mensajes de temperatura enviada en el transmisor cada 5 minutos. Se hace uso del software CoolTerm.

7 Experimentos y resultados

Una vez completado el proyecto, empezaremos con unas series de experimentos, con los que queremos poner a prueba su capacidad de funcionamiento, como por ejemplo indicar qué alcance tiene.

7.1 Toma de temperatura y humedad relativa en el tiempo.

Este experimento consistirá en realizar medidas en la temperatura y la humedad relativa de manera periódica durante un intervalo de tiempo.

El objetivo de este experimento es comprobar un correcto funcionamiento del proyecto durante un tiempo prolongado de tiempo. Para ello, hemos configurado nuestro transmisor para que transmita la temperatura y la humedad relativa cada 5 minutos, durante 24 horas seguidas, obteniendo así 288 muestras para un mismo día. Este experimento se ha realizado en dos escenarios. Uno de ellos en Sevilla capital entre el 9 y 10 de Agosto de 2021 y otro en Guadalcanal (Sevilla) entre el 14 y 15 de Agosto de 2021.

7.1.1 Toma de temperatura y humedad relativa en Sevilla.

Sevilla es considerada la capital de provincia más cálida de la península ibérica, de manera que en los meses de Julio y Agosto se suele superar los 33-40ºC. Respecto a la humedad, la provincia de Sevilla tiene registrado un clima árido o seco, siendo la capital donde se registran los índices de humedad anual más altos.

Sabiendo esto, hemos colocado el sensor en el barrio de la Macarena, en la cocina de un tercer piso y hemos registrado 288 muestras de temperatura y humedad relativa en 24 horas, obteniendo una gráfica como la de la figura 7.1.

7.1.2 Toma de temperatura y humedad relativa en Guadalcanal.

Guadalcanal está situada al norte de la provincia de Sevilla, concretamente en la Sierra Norte, a una altura de 662 metros sobre el nivel del mar. En Guadalcanal, la temperatura máxima promedio diaria es más de 28ºC desde el 16 de junio al 11 de septiembre. Respecto a la humedad, Guadalcanal suele tener una humedad relativa menor que la de Sevilla capital.

Teniendo en cuenta que las medidas fueron tomadas en plena ola de calor, se tomó una muestra cada 5 minutos durante 24 horas con el sensor colocado en plena calle, donde el sol podía darle de manera directa. La gráfica resultante es la mostrada en la figura 7.2.

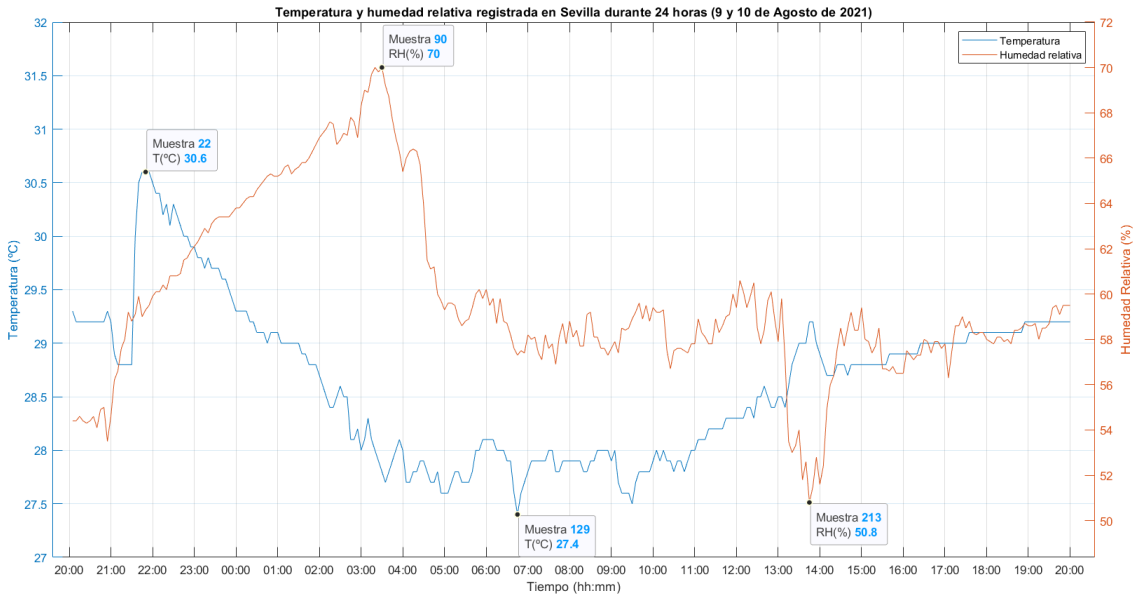


Figura 7.1 Temperatura y humedad relativa en Sevilla. Gráfica elaborada con el software Matlab® .

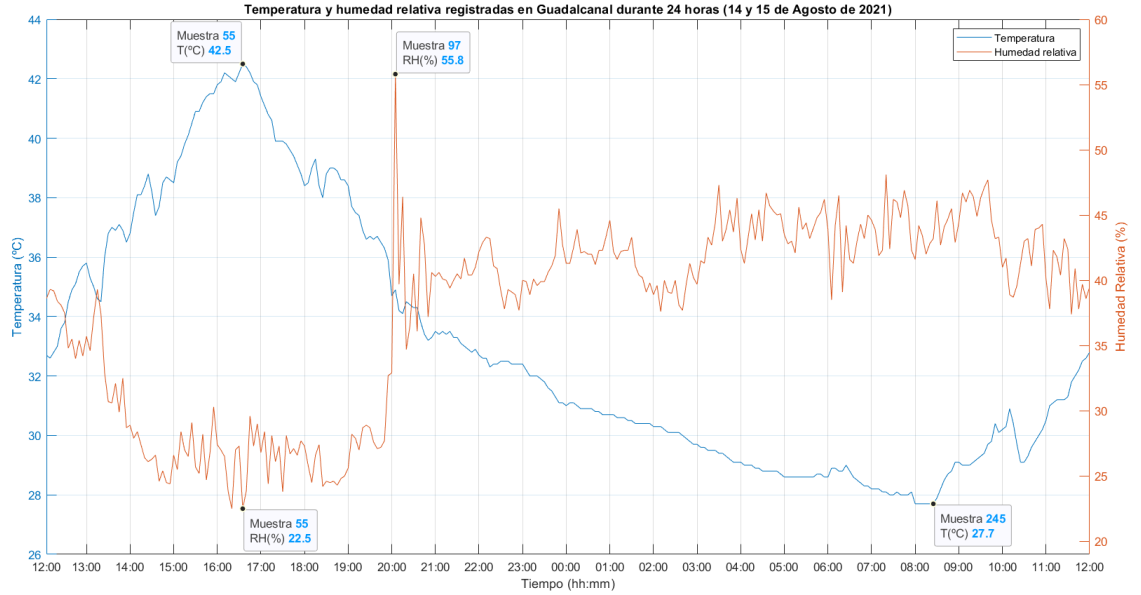


Figura 7.2 Temperatura y humedad relativa en Guadalcanal. Gráfica elaborada con el software Matlab® .

7.1.3 Conclusiones del experimento.

Los principales fallos en proyectos donde se implementa código aparecen cuando se está ejecutando en un intervalo de tiempo prolongado, tales como horas, días, semanas, etc. Por ello, estos errores son difíciles de predecir, ya que hasta pasado un tiempo después de su arranque no podremos ver dichos errores.

Una vez realizado este experimento en diferentes ubicaciones durante 24 horas cada uno, podemos concluir que el proyecto funciona sin errores, pudiéndose tener en pleno funcionamiento durante un tiempo prolongado.

7.2 Prueba de cobertura.

En este experimento vamos a poner a prueba la capacidad de conexión que tienen las placas en la distancia. Para ello, dejaremos la placa receptora conectada al ordenador, de manera que iremos capturando la información tales como el RSSI y la SNR. Por otro lado, nos moveremos por los alrededores con la placa transmisora, registrando las ubicaciones límite donde se producen las pérdidas de conexión.

RSSI son las siglas del inglés *Received Signal Strength Indicator*. Se utiliza para medir la potencia de las señales que se envían entre dispositivos en redes inalámbricas, por lo que es un indicador de intensidad recibida. Por otro lado, la SNR es la relación señal-ruido (del inglés *signal-to-noise ratio*) y es la relación existente entre la potencia de la señal transmitida y la potencia del ruido que pueda interferir.

Conociendo estos parámetros, hemos salido dos veces con nuestra placa, una con el spreading factor de la transmisión a 7 y otra con el spreading factor a 12, siendo estos el valor mínimo y máximo que puede tomar el SF respectivamente. Para ambos experimentos se ha dejado la placa receptora en el mismo origen para que se pueda ver la diferencia de un resultado respecto a otro.

7.2.1 Spreading Factor = 7.

Para un spreading factor igual a 7, el resultado es el mostrado en la figura 7.3.

Figura 7.3 Cobertura en el barrio de la Macarena de Sevilla (SF = 7). Mapa creado con ayuda de Google MyMaps® .

La superficie ocupada por la cobertura es de 1.25 Km^2 y, tal como se puede observar, la conexión se pierde para valores cercanos a -100dB en la RSSI, por lo que podemos afirmar que a partir de una RSSI menor a -100dB, se pierde la conexión.

Se debe puntualizar que los puntos A, B y D tienen más obstáculos o posibles interferencias tales como edificios. Por otro lado, el punto C es el más lejano y es el que tenía una vista directa con el receptor sin ningún edificio de por medio.

7.2.2 Spreading Factor = 12.

Tal y como hemos comentado en el capítulo 3.1.5, a un mayor SF tendremos un mayor tiempo de aire y por tanto una mayor cobertura. Como en esta ocasión hemos aumentado el spreading factor a 12, cabe esperar que podremos llegar más lejos con el transmisor respecto al anterior experimento, sin llegar a perder la conexión.

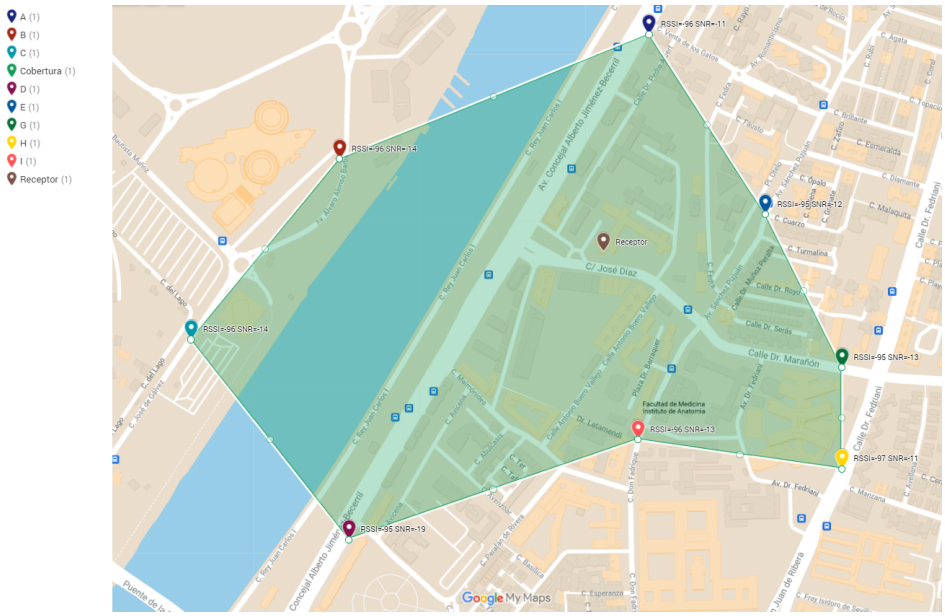


Figura 7.4 Cobertura en el barrio de la Macarena de Sevilla (SF = 12). Mapa creado con ayuda de Google MyMaps® .

Tal y como vemos en la figura 7.4, la cobertura de la conexión ha aumentado respecto al anterior experimento. El aumento del SF de 7 a 12 ha supuesto un aumento en la superficie de conexión de 1.25 Km^2 a 2.66 Km^2.

Igual que en el experimento anterior, en campo abierto se consigue una mayor cobertura, siendo el punto C el más apartado (617 metros en línea recta), con un RSSI = -96dB y un SNR = -14dB. Con este experimento podemos confirmar que al aumentar el SF, la cobertura de la conexión aumenta.

7.2.3 Conclusiones del experimento.

Tal y como adelantábamos en el capítulo 7, tras aumentar el SF aumentamos la distancia en la que se mantiene la conexión sin perder información. En nuestro caso, ha bastado con modificar el SF, ya que solo queremos transmitir la temperatura y la humedad relativa. Pero, si quisiéramos transmitir más datos, tendríamos que estudiar el SF adecuado, puesto que para un mayor SF, se reduce el número de datos a codificar.

También hemos podido experimentar que para situaciones en las que no hay obstáculos, o hay un menor número de obstáculos en la línea directa de conexión, la conexión se mantiene activa a una mayor distancia. Las pérdidas de conexión que se han registrado han tenido lugar por la existencia de edificios, vehículos y árboles que se encontraban en el momento y lugar del experimento, los cuales han interferido en la conexión de manera directa. También podría interferir el propio medio ambiente, pero este no ha sido el caso.

8 Conclusiones y líneas futuras

A lo largo del trabajo hemos documentado la importancia que tiene el IoT en el presente, las infinitas posibilidades existentes y lo mucho que puede avanzar en un futuro no muy lejano. Por otro lado, hemos entendido LoRa y el por qué es un tipo de modulación muy interesante a la hora de realizar los diferentes sensores IoT, esto es gracias a que requiere un bajo consumo y un ahorro económico respecto a las redes móviles, siempre que no haya que transmitir una tasa de datos elevada.

Hemos podido comprobar la dificultad que tiene programar hardware, puesto que existen diversos factores a tener en cuenta que pueden dar lugar a fallos importantes, los cuales no permiten el correcto funcionamiento de nuestro dispositivo. De esta manera, si no se divide el problema principal en secciones, se dificulta mucho la resolución del mismo. Nos hemos llevado más tiempo corrigiendo errores y comprobando el correcto funcionamiento que programando en sí.

Como hemos visto, existen diferentes experimentos con los que poner a prueba los proyectos, siendo el más difícil aquel que conlleva tener nuestro dispositivo en funcionamiento durante un largo periodo de tiempo. En nuestro caso, el experimento de tener el dispositivo 24 horas en funcionamiento (capítulo 7) ha sido el más difícil, puesto que ha sido el que más tiempo ha llevado, además de que el tenerlo en funcionamiento durante un elevado tiempo no nos permitía avanzar en nada más respecto al trabajo, ya que la conexión de las placas era lo primordial. Pero, gracias a esto, se ha podido comprobar el correcto funcionamiento de nuestro proyecto.

Con el experimento de la cobertura, hemos visto la importancia que tiene el spreading factor en las modulaciones LoRa, de manera que para un mayor SF tenemos una mayor cobertura. Sabiendo esto, tenemos que ser capaces de elegir un SF adecuado, teniendo en cuenta que si bien al aumentar el SF aumenta la cobertura, tendremos que reducir la tasa de datos a enviar, ocurriendo lo contrario al bajar el SF. En nuestra situación, debido a que la tasa de datos a enviar era baja, no hemos percibido problema al aumentar o disminuir el SF.

8.1 Líneas futuras.

En el ámbito de la programación, se pueden realizar ciertas modificaciones en un futuro. Por un lado, sería interesante la posibilidad de modificar el intervalo de tiempo en el que la placa transmisora está haciendo la toma de los parámetros ambientales. Para ello, deberíamos modificar tanto el código del transmisor como del receptor: en el lado del receptor, configuraríamos los botones de usuario de la placa para que, dependiendo del botón que se pulse, se envíe un número como respuesta al transmisor después de haber recibido la temperatura que se ha medido; y en el lado del transmisor, deberíamos identificar el número recibido e interpretarlo para modificar el intervalo en el que tenemos que tomar las medidas de los parámetros ambientales.

Por otro lado, se podría añadir un nuevo periférico, concretamente una pantalla de cristal líquido (lyquid-crystal-display: LCD). Se podría colocar una en el receptor o en el transmisor, o en ambos. Su función sería mostrar en la pantalla la temperatura y la humedad relativa captada y, si se ha hecho la modificación anterior propuesta, mostrar también el intervalo de tiempo en el que se están tomando las temperaturas. De esta manera, si te encuentras en el lado del transmisor, se podría ver la temperatura y la humedad relativa en tiempo real, además del intervalo de tiempo de envío de información.

Fuera del apartado de programación, se podría realizar el diseño electrónico de un circuito impreso, donde hacer uso del núcleo utilizado en la placa de desarrollo STM32WL55JC, la cual es la utilizada en este proyecto. Este circuito impreso deberíamos diseñarlo de forma específica para las funciones que hemos dado a nuestro proyecto, incluyendo periféricos tales como pines, para la conexión con el sensor, o el módulo de antena para la transmisión mediante LoRa. Para esto último, sería necesario un estudio intensivo del datasheet de la placa de desarrollo y el uso de una herramienta de diseño de circuitos electrónicos, como es el ejemplo de Eagle®.

En el apartado experimental, sería conveniente realizar experimentos con pequeñas variaciones en los parámetros de la modulación LoRa y anotar los resultados obtenidos. Algunos parámetros interesantes para modificar son la potencia de transmisión, el coding rate (CR) y el ancho de banda (Bw), de manera que la tasa de bits y la cobertura se verían afectadas.

Apéndice A
Código utilizado para la programación de la placa

En este apéndice se verán los códigos utilizados en el trabajo para poner en funcionamiento el proyecto.

A.1 Transmisor de la temperatura.

Las secciones siguientes irán dirigidas a la placa que se encargará de transmitir la temperatura y humedad relativa medida.

A.1.1 Función main.

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_SubGHz_Phy_Init();
  /* USER CODE BEGIN 2 */
  //PARA EL I2C DEL SENSOR
  __HAL_RCC_GPIOB_CLK_ENABLE();//Tanto esta línea como la siguiente se
     usan para activar los puertos que vamos a utilizar,
  __HAL_RCC_GPIOA_CLK_ENABLE();//Inicialmente iban dentro de MX_GPIO_
     Init() pero por razones que desconozco, no me deja usarla
                           // aún instanciándola yo, así que las he
                              puesto tal cual.

  MX_USART2_UART_Init(); //Para la comunicación UART
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  while (1)
  {
    /* -------- APLICACIÓN DE LECTURA DEL SENSOR SHT --------*/

    /* Retraso que añadimos */
    //HAL_Delay(seg_1);

   /* USER CODE END WHILE */

    MX_SubGHz_Phy_Process();
    /* USER CODE BEGIN 3 */
    /* Inicializamos en cada ejecución las librerías del sensor y de la
       comunicación I2C*/
    MX_I2C1_Init();
    SHT2x_Init(&hi2c1);
    SHT2x_SetResolution(RES_14_12);
  float cel = SHT2x_GetTemperature(1);
  float rh = SHT2x_GetRelativeHumidity(1);

  char temperatura_y_humedad[100] = { 0 };
  sprintf(temperatura_y_humedad,
        "%d.%dºC %d.%d%% RH\n\r",
        SHT2x_GetInteger(cel), SHT2x_GetDecimal(cel, 1),
        SHT2x_GetInteger(rh), SHT2x_GetDecimal(rh, 1));

  send_temp(temperatura_y_humedad, sizeof(temperatura_y_humedad));
  }
  /* USER CODE END 3 */
}
```

A.1.2 Librería para el sensor SHT20.

```
/* An STM32 HAL library written for the SHT2x temperature/humidity
   sensor series. */
/* Libraries by @eepj www.github.com/eepj */
#include "sht2x_for_stm32_hal.h"
#include "main.h"
#ifdef __cplusplus
extern "C"{
#endif

I2C_HandleTypeDef *_sht2x_ui2c;
```

```
/**
 * @brief Initializes the SHT2x temperature/humidity sensor.
 * @param hi2c User I2C handle pointer.
 */
void SHT2x_Init(I2C_HandleTypeDef *hi2c) {
  _sht2x_ui2c = hi2c;
}

/**
 *  @brief Performs a soft reset.
 */
void SHT2x_SoftReset(void){
  uint8_t cmd = SHT2x_SOFT_RESET;
  HAL_I2C_Master_Transmit(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, &cmd, 1, SHT
      2x_TIMEOUT);
}

/**
 * @brief Gets the value stored in user register.
 * @return 8-bit value stored in user register, 0 to 255.
 */
uint8_t SHT2x_ReadUserReg(void) {
  uint8_t val;
  uint8_t cmd = SHT2x_READ_REG;
  HAL_I2C_Master_Transmit(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, &cmd, 1, SHT
      2x_TIMEOUT);
  HAL_I2C_Master_Receive(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, &val, 1, SHT2
      x_TIMEOUT);
  return val;
}

/**
 * @brief Sends the designated command to sensor and read a 16-bit raw
     value.
 * @param cmd Command to send to sensor.
 * @return 16-bit raw value, 0 to 65535.
 */
uint16_t SHT2x_GetRaw(uint8_t cmd) {
  uint8_t val[3] = { 0 };
  HAL_I2C_Master_Transmit(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, &cmd, 1, SHT
      2x_TIMEOUT);
  HAL_I2C_Master_Receive(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, val, 3, SHT2x
      _TIMEOUT);
  return val[0] << 8 | val[1];
}
```

```
/**
 * @brief Measures and gets the current temperature.
 * @param hold Holding mode, 0 for no hold master, 1 for hold master.
 * @return Floating point temperature value.
 */
float SHT2x_GetTemperature(uint8_t hold) {
  uint8_t cmd = (hold ? SHT2x_READ_TEMP_HOLD : SHT2x_READ_TEMP_NOHOLD);
  return -46.85 + 175.72 * (SHT2x_GetRaw(cmd) / 65536.0);
}

/**
 * @brief Measures and gets the current relative humidity.
 * @param hold Holding mode, 0 for no hold master, 1 for hold master.
 * @return Floating point relative humidity value.
 */
float SHT2x_GetRelativeHumidity(uint8_t hold) {
  uint8_t cmd = (hold ? SHT2x_READ_RH_HOLD : SHT2x_READ_RH_NOHOLD);
  return -6 + 125.00 * (SHT2x_GetRaw(cmd) / 65536.0);
}

/**
 * @brief Sets the measurement resolution.
 * @param res Enum resolution.
 * @note Available resolutions: RES_14_12, RES_12_8, RES_13_10, RES
    _11_11.
 * @note RES_14_12 = 14-bit temperature and 12-bit RH resolution, etc.
 */
void SHT2x_SetResolution(SHT2x_Resolution res) {
  uint8_t val = SHT2x_ReadUserReg();
  val = (val & 0x7e) | res;
  uint8_t temp[2] = { SHT2x_WRITE_REG, val };
  HAL_I2C_Master_Transmit(_sht2x_ui2c, SHT2x_I2C_ADDR << 1, temp, 2, SHT
     2x_TIMEOUT);
}
/**
 * @brief Converts degrees Celsius to degrees Fahrenheit.
 * @param celsius Floating point temperature in degrees Celsius.
 * @return Floating point temperature in degrees Fahrenheit.
 */
float SHT2x_CelsiusToFahrenheit(float celsius) {
  return (9.0 / 5.0) * celsius + 32;
}
/**
 * @brief Converts degrees Celsius to Kelvin.
 * @param celsius Floating point temperature in degrees Celsius.
 * @return Floating point temperature in Kelvin.
 */
float SHT2x_CelsiusToKelvin(float celsius) {
  return celsius + 273;
}
```

```
/**
 * @brief Gets the integer part of a floating point number.
 * @note Avoids the use of sprinf floating point formatting.
 * @param num Floating point number.
 * @return Integer part of floating point number.
 */
int32_t SHT2x_GetInteger(float num) {
  return num / 1;
}

/**
 * @brief Gets the decimal part of a floating point number.
 * @note Avoids the use of sprinf floating point formatting.
 * @param num Floating point number.
 * @return Decimal part of floating point number.
 */
uint32_t SHT2x_GetDecimal(float num, int digits) {
  float postDec = num - SHT2x_GetInteger(num);
  return postDec * SHT2x_Ipow(10, digits);
}

/**
 * @brief Integer equivalent of pow() in math.h.
 * @param base Base.
 * @param power Power.
 * @return
 */
uint32_t SHT2x_Ipow(uint32_t base, uint32_t power) {
  uint32_t temp = base;
  for (uint32_t i = 1; i < power; i++)
    temp *= base;
  return temp;
}

#ifdef __cplusplus
}
#endif
```

A.1.3 Funciones de subghz_phy_app.c creadas/modificadas.

```
/* USER CODE BEGIN PrFD */
/*Función que introducirá la temperatura y humedad en el buffer de envío
    .
 * Será llamada desde el main.c, donde habremos leído previamente la
     temperatura y la humedad*/
void send_temp(char temp_y_hum[100], int size)
{
  for(int j=0;j<size;j++)
    {
      temperatura_y_humedad_TX[j] = temp_y_hum[j];
    }
}

static void PingPong_Process(void)
{
  Radio.Sleep();
  switch (State)
  {
    case RX:
      if (RxBufferSize > 0)
      {
      if (inicio == 1) //Si es la primera ejecución del código
      {
        if (strncmp((const char *)BufferRx, PING, sizeof(PING) - 1) == 0)
            //Si hemos recibido un ping
        {
        inicio = 0;
        UTIL_TIMER_Stop(&timerLed);
        /* switch off red led */
        BSP_LED_Off(LED_RED);
        /* slave toggles green led */
        BSP_LED_Toggle(LED_GREEN);
        /* Add delay between RX and TX */
        HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN);
        /*Mensaje inicial para indicar que somos el transmisor de la
            temperatura y la humedad */
        APP_LOG(TS_ON, VLEVEL_L, "Soy el TRANSMISOR de la temperatura\n\r
            ");
        /*slave sends PONG*/
        APP_LOG(TS_ON, VLEVEL_L,
            "PONG"
            "\n\r");
        memcpy(BufferTx, PONG, sizeof(PONG) - 1); //Aquí mandamos el
            mensaje PONG
        Radio.Send(BufferTx, PAYLOAD_LEN);
        }
```

```
      else /* valid reception but not a PING as expected */
      {
      /* Set device as master and start again */
      //isMaster = true;
      Radio.Rx(RX_TIMEOUT_VALUE);
      }
    }
    if (inicio == 0)
    {
        if (strncmp((const char *)BufferRx, PING, sizeof(PING) - 1) ==
            0)//Si somos el transmisor
        {
        UTIL_TIMER_Stop(&timerLed);
        /* switch off red led */
        BSP_LED_Off(LED_RED);
        /* slave toggles green led */
        BSP_LED_Toggle(LED_GREEN);
        /* Add delay between RX and TX */
        HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN);

        char temperatura_y_humedad__transmitida[100] = {0}; //Variable
            donde guardaremos la temperatura y la humedad medida

        /* Igualamos las tablas recibidas y la que vamos a transmitir
            */
        for(int j=0;j<sizeof(temperatura_y_humedad_TX);j++)
          {
            temperatura_y_humedad__transmitida[j] = temperatura_y_
                humedad_TX[j];
          }
        /* Copiamos en la variable BufferTx el valor de la temperatura
            y la humedad medida */
        memcpy(BufferTx, temperatura_y_humedad__transmitida, sizeof(
            temperatura_y_humedad__transmitida) - 1);
        Radio.Send(BufferTx, PAYLOAD_LEN); //Enviamos la temperatura
            por el canal
        APP_LOG(TS_ON, VLEVEL_L,"Temperatura enviada""\n\r");
        }
        else /* valid reception but not a PING as expected */
        {

        Radio.Rx(RX_TIMEOUT_VALUE);
        }
    }
    }
  break;
  case TX:
    Radio.Rx(RX_TIMEOUT_VALUE);
    break;
```

```
    case RX_TIMEOUT:
    case RX_ERROR:
      if (isMaster == true)
      {
        HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN + random_delay);

        /* master sends PING*/
        memcpy(BufferTx, PING, sizeof(PING) - 1);
        Radio.Send(BufferTx, PAYLOAD_LEN);
      }
      else
      {
        Radio.Rx(RX_TIMEOUT_VALUE);
      }
      break;
    case TX_TIMEOUT:
      Radio.Rx(RX_TIMEOUT_VALUE);
      break;
    default:
      break;
  }
}

/* USER CODE END PrFD */

/******** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

A.2 Receptor de la temperatura.

Las secciones siguientes irán dirigidas a la placa que se encargará de recibir la temperatura y la humedad relativa medida.

A.2.1 Función main.

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_SubGHz_Phy_Init();
  /* USER CODE BEGIN 2 */
  MX_USART2_UART_Init(); //Para la comunicación UART
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```

```
  while (1)
  {
    /* USER CODE END WHILE */
    MX_SubGHz_Phy_Process();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

A.2.2 Función de subghz_phy_app.c modificada.

```
static void PingPong_Process(void)
{
 Radio.Sleep();

 switch (State)
 {
   case RX:
     if (RxBufferSize > 0)
     {
     if (inicio == 1) //Primera vez que se ejecuta el código
     {
       if (strncmp((const char *)BufferRx, PONG, sizeof(PONG) - 1) == 0)
            //Si lo que hemos recibido es un PONG
       {
       inicio = 0;
       UTIL_TIMER_Stop(&timerLed);
       /* switch off green led */
       BSP_LED_Off(LED_GREEN);

       /* master toggles red led */
       BSP_LED_Toggle(LED_RED);

       /* Add delay between RX and TX */
       HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN);

       /*Mensaje inicial para indicar que somos el receptor de la
          temperatura*/
       APP_LOG(TS_ON, VLEVEL_L, "Soy el RECEPTOR de la temperatura\n\r")
          ; //Modificación para la versión 3

       /* master sends PING*/
       APP_LOG(TS_ON, VLEVEL_L,
           "PING"
           "\n\r");
       memcpy(BufferTx, PING, sizeof(PING) - 1); //Aquí mandamos el
          mensaje PING inicial
       Radio.Send(BufferTx, PAYLOAD_LEN);
       }
```

```
       else /* valid reception but neither a PING or a PONG message */
       {
       /* Set device as master and start again */
       isMaster = true;
       Radio.Rx(RX_TIMEOUT_VALUE);
       }
      }
      if (inicio == 0) //ahora pasamos a la comunicación de información
      {
       UTIL_TIMER_Stop(&timerLed);
       /* switch off green led */
       BSP_LED_Off(LED_GREEN);
       /* master toggles red led */
       BSP_LED_Toggle(LED_RED);
       /* Add delay between RX and TX */
       HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN);
              APP_LOG(TS_ON, VLEVEL_L,(const char *)BufferRx); //Aquí
                  mostramos por consola el valor recibido de la
                  temperatura
       memcpy(BufferTx, PING, sizeof(PING) - 1);
       Radio.Send(BufferTx, PAYLOAD_LEN);
      }
     }
    }
  break;
    case TX:
      Radio.Rx(RX_TIMEOUT_VALUE);
      break;
    case RX_TIMEOUT:
    case RX_ERROR:
        HAL_Delay(Radio.GetWakeupTime() + RX_TIME_MARGIN + random_delay);

        /* master sends PING*/
        memcpy(BufferTx, PING, sizeof(PING) - 1);
        Radio.Send(BufferTx, PAYLOAD_LEN);

      else
      {
        Radio.Rx(RX_TIMEOUT_VALUE);
      }
      break;
    case TX_TIMEOUT:
      Radio.Rx(RX_TIMEOUT_VALUE);
      break;
    default:
      break;
  }
}
```

Apéndice B
Secciones de datasheets utilizadas

3 Interface Specifications

Pin	Name	Comment
1	SDA	Serial Data, bidirectional
2	VSS	Ground
5	VDD	Supply Voltage
6	SCL	Serial Clock, bidirectional
3,4	NC	Not Connected

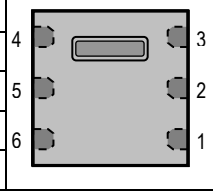


Table 2 SHT2x pin assignment, NC remain floating (top view)

3.1 Power Pins (VDD, VSS)

The supply voltage of SHT2x must be in the range of 2.1 – 3.6V, recommended supply voltage is 3.0V. Power supply pins Supply Voltage (VDD) and Ground (VSS) must be decoupled with a 100nF capacitor, that shall be placed as close to the sensor as possible – see Figure 11.

3.2 Serial clock (SCL)

SCL is used to synchronize the communication between microcontroller (MCU) and the sensor. Since the interface consists of fully static logic there is no minimum SCL frequency.

3.3 Serial SDA (SDA)

The SDA pin is used to transfer data in and out of the sensor. For sending a command to the sensor, SDA is valid on the rising edge of SCL and must remain stable while SCL is high. After the falling edge of SCL the SDA value may be changed. For safe communication SDA shall be valid t_{SU} and t_{HD} before the rising and after the falling edge of SCL, respectively – see Figure 12. For reading data from the sensor, SDA is valid t_{VD} after SCL has gone low and remains valid until the next falling edge of SCL.

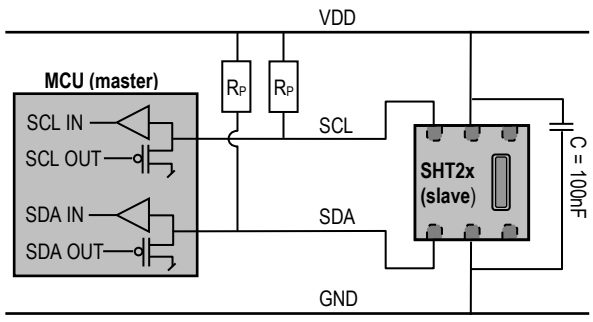


Figure 11 Typical application circuit, including pull-up resistors R_P and decoupling of VDD and VSS by a capacitor.

To avoid signal contention the micro-controller unit (MCU) must only drive SDA and SCL low. External pull-up resistors (e.g. 10kΩ), are required to pull the signal high. For the choice of resistor size please take bus capacity requirements into account (compare Table 5). It should be noted that pull-up resistors may be included in I/O circuits of MCUs. See Table 4 and Table 5 for detailed I/O characteristic of the sensor.

4 Electrical Characteristics

4.1 Absolute Maximum Ratings

The electrical characteristics of SHT2x are defined in Table 1. The absolute maximum ratings as given in Table 3 are stress ratings only and give additional information. Functional operation of the device at these conditions is not implied. Exposure to absolute maximum rating conditions for extended periods may affect the sensor reliability (e.g. hot carrier degradation, oxide breakdown).

Parameter	min	max	Units
VDD to VSS	-0.3	5	V
Digital I/O Pins (SDA, SCL) to VSS	-0.3	VDD + 0.3	V
Input Current on any Pin	-100	100	mA

Table 3 Electrical absolute maximum ratings

ESD immunity is qualified according to JEDEC JESD22-A114 method (Human Body Model at ±4kV), JEDEC JESD22-A115 method (Machine Model ±200V) and ESDA ESD-STM5.3.1-1999 and AEC-Q100-011 (Charged Device Model, 750V corner pins, 500V other pins). Latch-up immunity is provided at a force current of ±100mA with T_{amb} = 125°C according to JEDEC JESD78. For exposure beyond named limits the sensor needs additional protection circuit.

4.2 Input / Output Characteristics

The electrical characteristics such as power consumption, low and high level input and output voltages depend on the supply voltage. For proper communication with the sensor it is essential to make sure that signal design is strictly within the limits given in Table 4 & 5 and Figure 12.

Parameter	Conditions	min	typ	max	Units
Output Low Voltage, VOL	VDD = 3.0 V, -4 mA < IOL < 0mA	0	-	0.4	V
Output Sink Current, IOL		-	-	-4	mA
Input Low Voltage, VIL		0	-	30% VDD	V
Input High Voltage, VIH		70% VDD	-	VDD	V
Input Current	VDD = 3.6 V, VIN = 0 V to 3.6 V	-	-	±1	uA

Table 4 DC characteristics of digital input/output pads. VDD = 2.1V to 3.6V, T = -40°C to 125°C, unless otherwise noted.

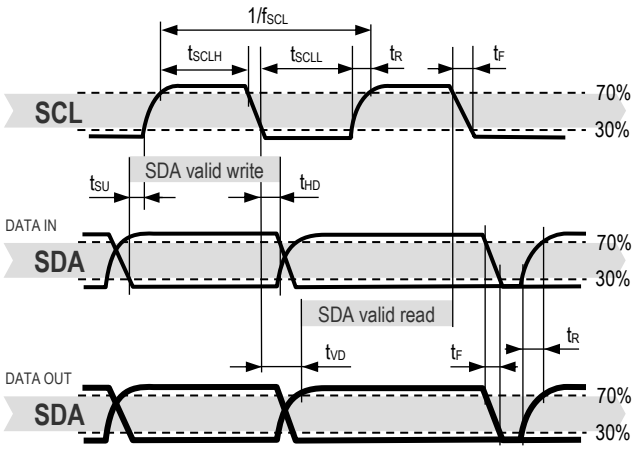


Figure 12 Timing Diagram for Digital Input/Output Pads, abbreviations are explained in Table 5. SDA directions are seen from the sensor. Bold SDA line is controlled by the sensor, plain SDA line is controlled by the micro-controller. Note that SDA valid read time is triggered by falling edge of anterior toggle.

Parameter	min	typ	max	Units
SCL frequency, f_{SCL}	0	-	0.4	MHz
SCL High Time, t_{SCLH}	0.6	-	-	µs
SCL Low Time, t_{SCLL}	1.3	-	-	µs
SDA Set-Up Time, t_{SU}	100	-	-	ns
SDA Hold Time, t_{HD}	0	-	900	ns
SDA Valid Time, t_{VD}	0	-	400	ns
SCL/SDA Fall Time, t_F	0	-	100	ns
SCL/SDA Rise Time, t_R	0	-	300	ns
Capacitive Load on Bus Line, C_B	0	-	400	pF

Table 5 Timing specifications of digital input/output pads for I²C fast mode. Entities are displayed in Figure 12. VDD = 2.1V to 3.6V, T = -40°C to 125°C, unless otherwise noted. For further information regarding timing, please refer to http://www.standardics.nxp.com/support/i2c/.

5 Communication with Sensor

SHT20 communicates with I²C protocol. For information on I²C beyond the information in the following Sections please refer to the following website:

http://www.standardics.nxp.com/support/i2c/.

Please note that all sensors are set to the same I²C address, as defined in Section 5.3.

Furthermore, please note, that Sensirion provides an exemplary sample code on its home page – compare www.sensirion.com/SHT20.

Please note that in case VDD is set to 0 V (GND), e.g. in case of a power off of the SHT2x, the SCL and SDA pads are also pulled to GND. Consequently, the I2C bus is blocked while VDD of the SHT2x is set to 0 V.

5.1 Start Up Sensor

As a first step, the sensor is powered up to the chosen supply voltage VDD (between 2.1V and 3.6V). After power-up, the sensor needs at most 15ms, while SCL is high, for reaching idle state, i.e. to be ready accepting commands from the master (MCU). Current consumption during start up is 350µA maximum. Whenever the sensor is powered up, but not performing a measurement or communicating, it is automatically in idle state (sleep mode).

5.2 Start / Stop Sequence

Each transmission sequence begins with Start condition (S) and ends with Stop condition (P) as displayed in Figure 13 and Figure 14.

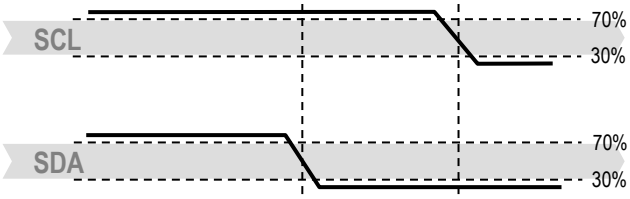


Figure 13 Transmission Start condition (S) - a high to low transition on the SDA line while SCL is high. The Start condition is a unique state on the bus created by the master, indicating to the slaves the beginning of a transmission sequence (bus is considered busy after a Start).

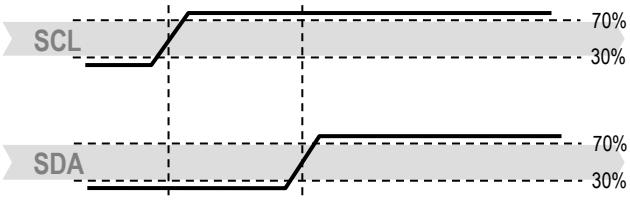


Figure 14 Transmission Stop condition (P) - a low to high transition on the SDA line while SCL is high. The Stop condition is a unique state on the bus created by the master, indicating to the slaves the end of a transmission sequence (bus is considered free after a Stop).

5.3 Sending a Command

After sending the Start condition, the subsequent I²C header consists of the 7-bit I²C device address '1000'000' and an SDA direction bit (Read R: '1', Write W: '0'). The sensor indicates the proper reception of a byte by pulling the SDA pin low (ACK bit) after the falling edge of the 8th SCL clock. After the issue of a measurement command ('1110'0011' for temperature, '1110'0101' for relative humidity'), the MCU must wait for the measurement to complete. The basic commands are summarized in Table 6.

Command	Comment	Code
Trigger T measurement	hold master	1110'0011
Trigger RH measurement	hold master	1110'0101
Trigger T measurement	no hold master	1111'0011
Trigger RH measurement	no hold master	1111'0101
Write user register		1110'0110
Read user register		1110'0111
Soft reset		1111'1110

Table 6 Basic command set, RH stands for relative humidity, and T stands for temperature

Hold master or *no hold master* modes are explained in next Section.

5.4 Hold / No Hold Master Mode

There are two different operation modes to communicate with the sensor: *Hold Master* mode or *No Hold Master* mode. In the first case the SCL line is blocked (controlled by sensor) during measurement process while in the latter case the SCL line remains open for other communication while the sensor is processing the measurement. No hold master mode allows for processing other I²C communication tasks on a bus while the sensor is measuring. A communication sequence of the two modes is displayed in Figure 15 and Figure 16, respectively.

In the *hold master mode*, the SHT2x pulls down the SCL line while measuring to force the master into a wait state. By releasing the SCL line the sensor indicates that internal processing is terminated and that transmission may be continued.

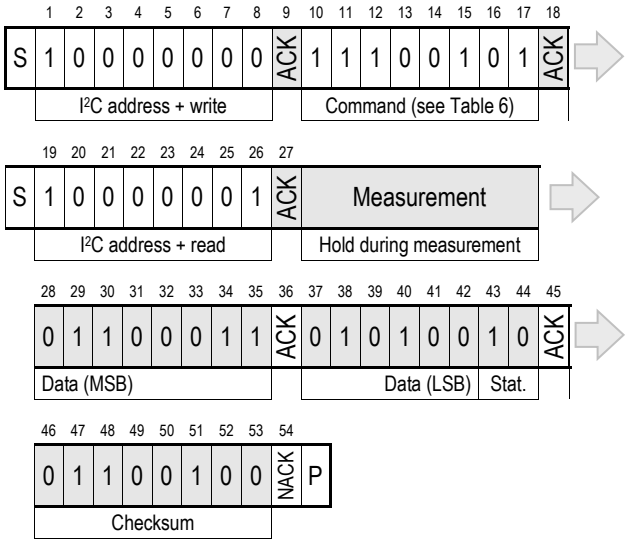


Figure 15 *Hold master* communication sequence – grey blocks are controlled by SHT2x. Bit 45 may be changed to NACK followed by Stop condition (P) to omit checksum transmission.

In *no hold master* mode, the MCU has to poll for the termination of the internal processing of the sensor. This is done by sending a Start condition followed by the I²C header (1000'0001) as shown in Figure 16. If the internal processing is finished, the sensor acknowledges the poll of the MCU and data can be read by the MCU. If the measurement processing is not finished the sensor answers no ACK bit and the Start condition must be issued once more.

When using the *no hold master* mode it is recommended to include a wait period of 20 µs after the reception of the sensor's ACK bit (bit 18 in Figure 16) and before the Stop condition.

For both modes, since the maximum resolution of a measurement is 14 bit, the two last least significant bits (LSBs, bits 43 and 44) are used for transmitting status information. Bit 1 of the two LSBs indicates the measurement type ('0': temperature, '1' humidity). Bit 0 is currently not assigned.

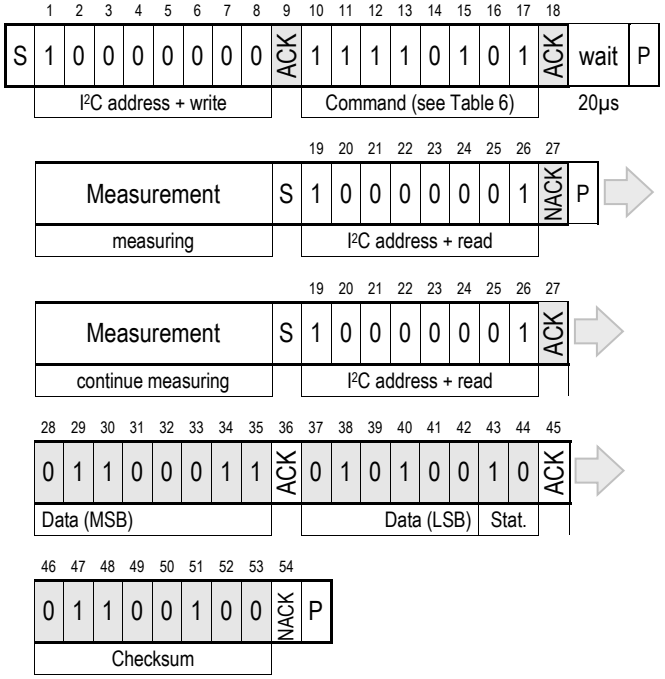


Figure 16 *No Hold master* communication sequence – grey blocks are controlled by SHT2x. If measurement is not completed upon "read" command, sensor does not provide ACK on bit 27 (more of these iterations are possible). If bit 45 is changed to NACK followed by Stop condition (P) checksum transmission is omitted.

In the examples given in Figure 15 and Figure 16 the sensor output is S_{RH} = '0110'0011'0101'0000'. For the calculation of physical values Status Bits must be set to '0' – see Chapter 6.

The maximum duration for measurements depends on the type of measurement and resolution chosen – values are displayed in Table 7. Maximum values shall be chosen for the communication planning of the MCU.

Resolution	RH typ	RH max	T typ	T max	Units
14 bit			66	85	ms
13 bit			33	43	ms
12 Bit	22	29	17	22	ms
11 bit	12	15	9	11	ms
10 bit	7	9			ms
8 bit	3	4			ms

Table 7 Measurement times for RH and T measurements at different resolutions. Typical values are recommended for calculating energy consumption while maximum values shall be applied for calculating waiting times in communication.

Please note: I²C communication allows for repeated Start conditions (S) without closing prior sequence with Stop condition (P) – compare Figures 15, 16 and 18. Still, any sequence with adjacent Start condition may alternatively be closed with a Stop condition.

5.5 Soft Reset

This command (see Table 6) is used for rebooting the sensor system without switching the power off and on again. Upon reception of this command, the sensor system reinitializes and starts operation according to the default settings – with the exception of the heater bit in the user register (see Sect. 5.6). The soft reset takes less than 15ms.

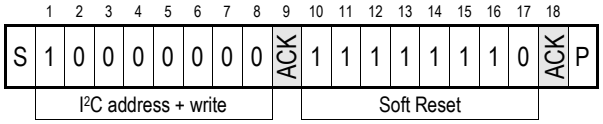


Figure 17 *Soft Reset* – grey blocks are controlled by SHT2x.

5.6 User Register

The content of User Register is described in Table 8. Please note that reserved bits must not be changed and default values of respective reserved bits may change over time without prior notice. Therefore, for any writing to the User Register, default values of reserved bits must be read first. Thereafter, the full User Register string is composed of respective default values of reserved bits and the remainder of accessible bits optionally with default or non-default values.

The *end of battery* alert is activated when the battery power falls below 2.25V.

The *heater* is intended to be used for functionality diagnosis – relative humidity drops upon rising temperature. The heater consumes about 5.5mW and provides a temperature increase of about 0.5 – 1.5°C.

OTP Reload is a safety feature and loads the entire OTP settings to the register, with the exception of the heater bit, before every measurement. This feature is disabled per default and is not recommended for use. Please use Soft Reset instead – it contains OTP Reload.

Bit	# Bits	Description / Coding	Default
7, 0	2	Measurement resolution RH / T '00': 12 bit / 14 bit '01': 8 bit / 12 bit '10': 10 bit / 13 bit '11': 11 bit / 11 bit	'00'
6	1	Status: End of battery[13] '0': VDD > 2.25V '1': VDD < 2.25V	'0'
3, 4, 5	3	Reserved	
2	1	Enable on-chip heater	'0'
1	1	Disable OTP Reload	'1'

Table 8 User Register. Cut-off value for End of Battery signal may vary by ±0.1V. Reserved bits must not be changed. "OTP reload" = '0' loads default settings after each time a measurement command is issued.

An example for I²C communication reading and writing the User Register is given in Figure 18.

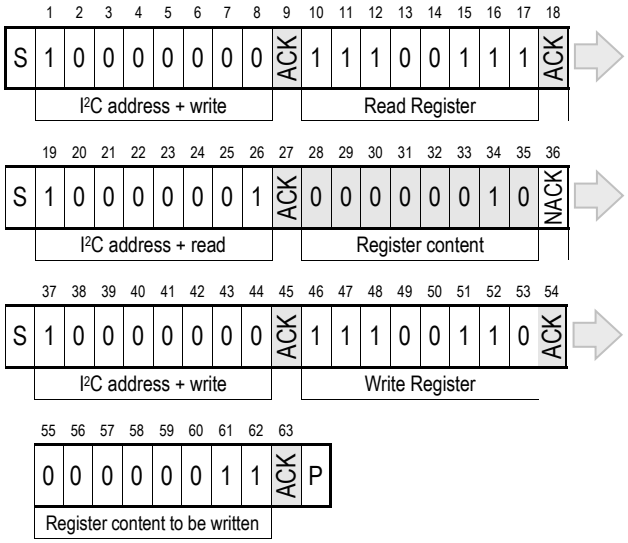


Figure 18 *Read and write register sequence* – grey blocks are controlled by SHT2x. In this example, the resolution is set to 8bit / 12bit.

5.7 CRC Checksum

SHT2x provides a CRC-8 checksum for error detection. The polynomial used is $x^8 + x^5 + x^4 + 1$. For more details and implementation please refer to the application note "CRC Checksum Calculation for SHT2x".

[13] This status bit is updated after each measurement

5.8 Serial Number

SHT20 provides an electronic identification code. For instructions on how to read the identification code please refer to the Application Note "Electronic Identification Code" – to be downloaded from the web page www.sensirion.com/SHT20.

6 Conversion of Signal Output

Default resolution is set to 12 bit relative humidity and 14 bit temperature reading. Measured data are transferred in two byte packages, i.e. in frames of 8 bit length where the most significant bit (MSB) is transferred first (left aligned). Each byte is followed by an acknowledge bit. The two status bits, the last bits of LSB, must be set to '0' before calculating physical values. In the example of Figure 15 and Figure 16, the transferred 16 bit relative humidity data is '0110'0011'0101'0000' = 25424.

6.1 Relative Humidity Conversion

With the relative humidity signal output S_{RH} the relative humidity *RH* is obtained by the following formula (result in %RH), no matter which resolution is chosen:

$$RH = -6 + 125 \cdot \frac{S_{RH}}{2^{16}}$$

In the example given in Figure 15 and Figure 16 the relative humidity results to be 42.5%RH.

The physical value RH given above corresponds to the relative humidity above liquid water according to World Meteorological Organization (WMO). For relative humidity above ice RH_i the values need to be transformed from relative humidity above water RH_w at temperature *t*. The equation is given in the following, compare also Application Note "Introduction to Humidity":

$$RH_i = RH_w \cdot \exp\left(\frac{\beta_w \cdot t}{\lambda_w + t}\right) \Big/ \exp\left(\frac{\beta_i \cdot t}{\lambda_i + t}\right)$$

Units are %RH for relative humidity and °C for temperature. The corresponding coefficients are defined as follows: β_w = 17.62, λ_w = 243.12°C, β_i = 22.46, λ_i = 272.62°C.

6.2 Temperature Conversion

The temperature *T* is calculated by inserting temperature signal output S_T into the following formula (result in °C), no matter which resolution is chosen:

$$T = -46.85 + 175.72 \cdot \frac{S_T}{2^{16}}$$

7 Environmental Stability

The SHT2x sensor series were tested based on AEC-Q100 Rev. G qualification test method where applicable. Sensor specifications are tested to prevail under the AEC-Q100 temperature grade 1 test conditions listed in Table 9[14].

Environment	Standard	Results[15]
HTOL	125°C, 408 hours	Pass
TC	-50°C - 125°C, 1000 cycles	Pass
UHST	130°C / 85%RH / ≈2.3bar, 96h	Pass
THB	85°C / 85%RH, 1000h	Pass
HTSL	150°C, 1000h	Pass
ELFR	125°C, 48h	Pass
ESD immunity	HBM ±4kV, MM ±200V, CDM 750V/500V (corner/other pins)	Pass
Latch-up	force current of ±100mA with T_{amb} = 125°C	Pass

Table 9: Performed qualification test series. HTOL = High Temperature Operating Lifetime, TC = Temperature Cycles, UHST = Unbiased Highly accelerated Stress Test, THB = Temperature Humidity Biased. For details on ESD see Sect. 4.1.

Sensor performance under other test conditions cannot be guaranteed and is not part of the sensor specifications. Especially, no guarantee can be given for sensor performance in the field or for customer's specific application.

If sensors are qualified for reliability and behavior in extreme conditions, please make sure that they experience same conditions as the reference sensor. It should be taken into account that response times in assemblies may be longer, hence enough dwell time for the measurement shall be granted. For detailed information please consult Application Note "Testing Guide".

8 Packaging

8.1 Packaging Type

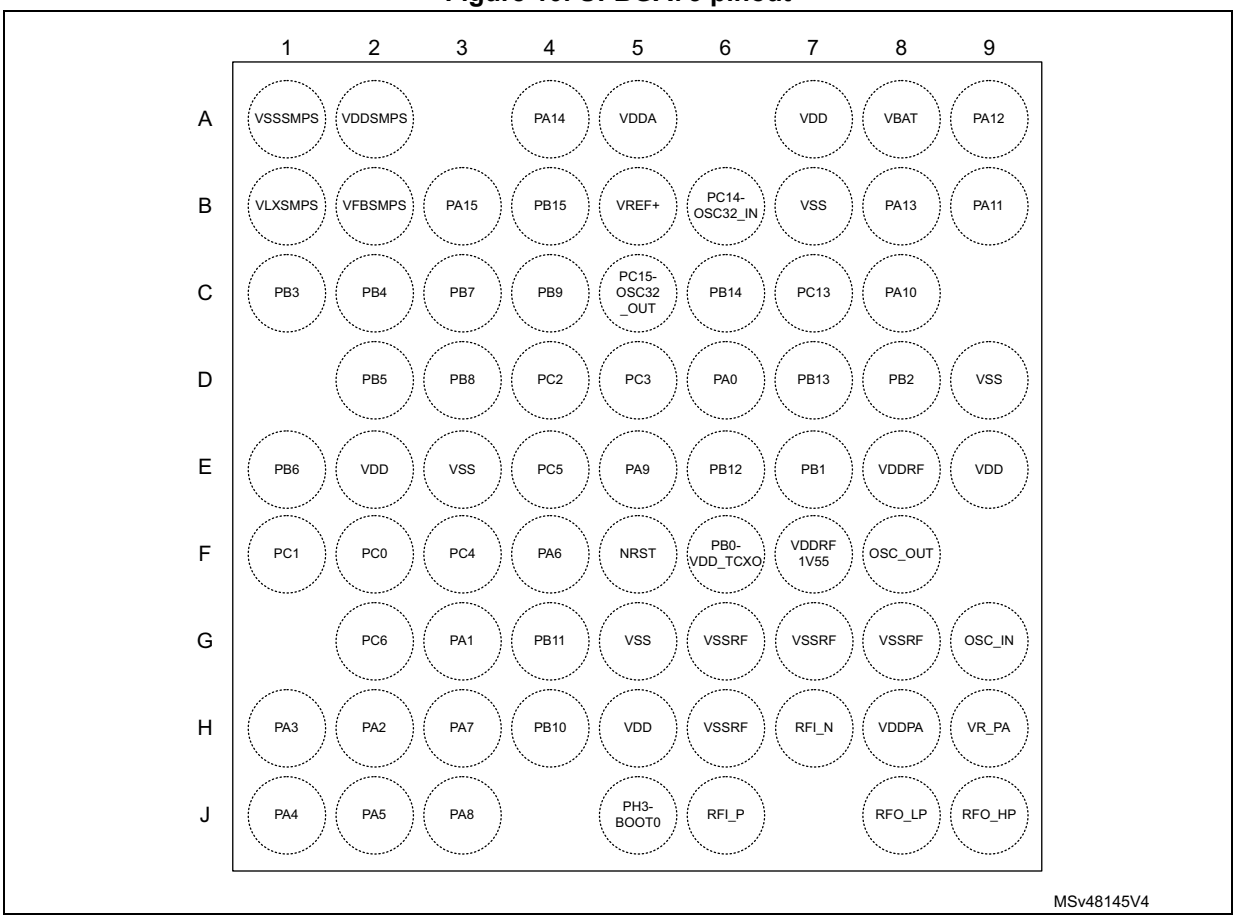
SHT2x sensors are provided in DFN packaging (in analogy with QFN packaging). DFN stands for Dual Flat No leads.

The sensor chip is mounted to a lead frame made of Cu and plated with Ni/Pd/Au. Chip and lead frame are over molded by green epoxy-based mold compound. Please note that side walls of sensors are diced and hence lead frame at diced edge is not covered with respective protective coating. The total weight of the sensor is 25mg.

[14] Temperature range is -40 to 125°C (AEC-Q100 temperature grade 1).

[15] According to accuracy and long term drift specification given on Page 2.

Figure 10. UFBGA73 pinout



1. The above figure shows the package top view.

Table 18. Legend/abbreviations used in the pinout table

Name	Abbreviation	Definition
Pin name	Unless otherwise specified in brackets below the pin name, the pin function during and after reset is the same as the actual pin name	
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input / output pin
	O	Output only pin
I/O structure	FT	5 V tolerant I/O
	RF	Radio RF pin
	TT	3 V tolerant I/O
	Option for FT I/Os	
	_f	I/O, Fm+ capable
	_a	I/O, with Analog switch function supplied by V_{DDA}

Table 18. Legend/abbreviations used in the pinout table (continued)

Name		Abbreviation	Definition
Notes		Unless otherwise specified by a note, all I/Os are set as analog inputs during and after reset.	
Pin functions	Alternate functions	Functions selected through GPIOx_AFR registers	
	Additional functions	Functions directly selected/enabled through peripheral registers	

Table 19. STM32WL55/54xx pin definition

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
-	E10	-	VSS	S	-	-	-	-
1	D11	C1	PB3	I/O	FT_a	-	JTDO/TRACESWO, TIM2_CH2, SPI1_SCK, RF_IRQ0, USART1_RTS, DEBUG_RF_DTB1, CM4_EVENTOUT	COMP1_INM, COMP2_INM, ADC_IN2, TAMP_IN3/WKUP3
2	D9	C2	PB4	I/O	FT_fa	-	NJTRST, I2C3_SDA, SPI1_MISO, USART1_CTS, DEBUG_RF_LDORDY, TIM17_BKIN, CM4_EVENTOUT	COMP1_INP, COMP2_INP, ADC_IN3
3	-	D2	PB5	I/O	FT_a	-	LPTIM1_IN1, I2C1_SMBA, SPI1_MOSI, RF_IRQ1, USART1_CK, COMP2_OUT, TIM16_BKIN, CM4_EVENTOUT	-
-	F7	E3	VSS	S	-	-	-	-
-	F11	E2	VDD	S	-	-	-	-
4	-	E1	PB6	I/O	FT_f	-	LPTIM1_ETR, I2C1_SCL, USART1_TX, TIM16_CH1N, CM4_EVENTOUT	-
5	-	C3	PB7	I/O	FT_f	-	LPTIM1_IN2, TIM1_BKIN, I2C1_SDA, USART1_RX, TIM17_CH1N, CM4_EVENTOUT	-
6	-	D3	PB8	I/O	FT_f	-	TIM1_CH2N, I2C1_SCL, RF_IRQ2, TIM16_CH1, CM4_EVENTOUT	-

Table 19. STM32WL55/54xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
-	-	C4	PB9	I/O	FT_f	-	TIM1_CH3N, I2C1_SDA, SPI2_NSS/I2S2_WS, IR_OUT, TIM17_CH1, CM4_EVENTOUT	-
-	-	F2	PC0	I/O	FT_f	-	LPTIM1_IN1, I2C3_SCL, LPUART1_RX, LPTIM2_IN1, CM4_EVENTOUT	-
-	-	F1	PC1	I/O	FT_f	-	LPTIM1_OUT, SPI2_MOSI/I2S2_SD, I2C3_SDA, LPUART1_TX, CM4_EVENTOUT	-
-	-	D4	PC2	I/O	FT	-	LPTIM1_IN2, SPI2_MISO, CM4_EVENTOUT	-
-	-	D5	PC3	I/O	FT	-	LPTIM1_ETR, SPI2_MOSI/I2S2_SD, LPTIM2_ETR, CM4_EVENTOUT	-
-	-	F3	PC4	I/O	FT	-	CM4_EVENTOUT	-
-	-	E4	PC5	I/O	FT	-	CM4_EVENTOUT	-
-	-	G2	PC6	I/O	FT	-	I2S2_MCK, CM4_EVENTOUT	-
7	H11	D6	PA0	I/O	FT_a	-	TIM2_CH1, I2C3_SMBA, I2S_CKIN, USART2_CTS, COMP1_OUT, DEBUG_PWR_REGLP1S, TIM2_ETR, CM4_EVENTOUT	TAMP_IN2/WKUP1
8	G10	G3	PA1	I/O	FT_a	-	TIM2_CH2, LPTIM3_OUT, I2C1_SMBA, SPI1_SCK, USART2_RTS, LPUART1_RTS, DEBUG_PWR_REGLP2S, CM4_EVENTOUT	-
9	F9	H2	PA2	I/O	FT_a	-	LSCO, TIM2_CH3, USART2_TX, LPUART1_TX, COMP2_OUT, DEBUG_PWR_LDORDY, CM4_EVENTOUT	LSCO
10	C8	H1	PA3	I/O	FT_a	-	TIM2_CH4, I2S2_MCK, USART2_RX, LPUART1_RX, CM4_EVENTOUT	-
-	E6	G5	VSS	S	-	-	-	-

Table 19. STM32WL55/54xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
11	K11	H5	VDD	S	-	-	-	-
12	J10	J1	PA4	I/O	FT	-	RTC_OUT2, LPTIM1_OUT, SPI1_NSS, USART2_CK, DEBUG_SUBGHZSPI_ NSSOUT, LPTIM2_OUT, CM4_EVENTOUT	-
13	H9	J2	PA5	I/O	FT	-	TIM2_CH1, TIM2_ETR, SPI2_MISO, SPI1_SCK, DEBUG_SUBGHZSPI_ SCKOUT, LPTIM2_ETR, CM4_EVENTOUT	-
14	G8	F4	PA6	I/O	FT	-	TIM1_BKIN, I2C2_SMBA, SPI1_MISO, LPUART1_CTS, DEBUG_SUBGHZSPI_ MISOOUT, TIM16_CH1, CM4_EVENTOUT	-
15	E8	H3	PA7	I/O	FT_fa	-	TIM1_CH1N, I2C3_SCL, SPI1_MOSI, COMP2_OUT, DEBUG_SUBGHZSPI_ MOSIOUT, TIM17_CH1, CM4_EVENTOUT	-
16	L10	J3	PA8	I/O	FT_a	-	MCO, TIM1_CH1, SPI2_SCK/I2S2_CK, USART1_CK, LPTIM2_OUT, CM4_EVENTOUT	-
17	K9	E5	PA9	I/O	FT_fa	-	TIM1_CH2, SPI2_NSS/I2S2_WS, I2C1_SCL, SPI2_SCK/I2S2_CK, USART1_TX, CM4_EVENTOUT	-
-	-	H4	PB10	I/O	FT_f	-	TIM2_CH3, I2C3_SCL, SPI2_SCK/I2S2_CK, LPUART1_RX, COMP1_OUT, CM4_EVENTOUT	-
-	-	G4	PB11	I/O	FT_f	-	TIM2_CH4, I2C3_SDA, LPUART1_TX, COMP2_OUT, CM4_EVENTOUT	-
18	J8	F5	NRST	I/O	FT	-	-	-
19	H7	J5	PH3-BOOT0	I/O	FT	-	CM4_EVENTOUT	BOOT0
-	L8	-	VDD	S	-	-	-	-

Table 19. STM32WL55/54xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
-	K7	-	VSS	S	-	-	-	-
-	J6	H6	VSSRF	S	-	-	-	-
-	H5	G6	VSSRF	S	-	-	-	-
20	L6	J6	RFI_P	I	RF	-	-	-
21	K5	H7	RFI_N	I	RF	-	-	-
-	G4	G7	VSSRF	S	-	-	-	-
-	J4	-	VSSRF	S	-	-	-	-
22	L4	J8	RFO_LP	O	RF	-	-	-
-	-	G8	VSSRF	S	-	-	-	-
23	K3	J9	RFO_HP	O	RF	-	-	-
-	H3	-	VSSRF	S	-	-	-	-
24	L2	H9	VR_PA	S	-	-	-	-
25	H1	H8	VDDPA	S	-	-	-	-
-	K1	-	VSSRF	S	-	-	-	-
26	G2	G9	OSC_IN	I	RF	-	-	-
27	F1	F8	OSC_OUT	O	RF	-	-	-
-	F3	-	VSSRF	S	-	-	-	-
28	E2	E8	VDDRF	S	-	-	-	-
29	D1	F7	VDDRF1V55	S	-	-	-	-
-	F5	D9	VSS	S	-	-	-	-
-	-	E9	VDD	S	-	-	-	-
30	B1	F6	PB0-VDD_TCXO	I/O	TT	-	COMP1_OUT, CM4_EVENTOUT	-
-	-	E7	PB1	I/O	FT_a	-	LPUART1_RTS_DE, LPTIM2_IN1, CM4_EVENTOUT	COMP2_INP, ADC_IN5
31	-	D8	PB2	I/O	FT_a	-	LPTIM1_OUT, I2C3_SMBA, SPI1_NSS, DEBUG_RF_SMPSRDY, CM4_EVENTOUT	COMP1_INP, COMP2_INM, ADC_IN4
32	-	E6	PB12	I/O	FT	-	TIM1_BKIN, I2C3_SMBA, SPI2_NSS/I2S2_WS, LPUART1_RTS, CM4_EVENTOUT	-

Table 19. STM32WL55/54xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
-	-	D7	PB13	I/O	FT_fa	-	TIM1_CH1N, I2C3_SCL, SPI2_SCK/I2S2_CK, LPUART1_CTS, CM4_EVENTOUT	ADC_IN0
-	-	C6	PB14	I/O	FT_fa	-	TIM1_CH2N, I2S2_MCK, I2C3_SDA, SPI2_MISO, CM4_EVENTOUT	ADC_IN1
33	D3	C8	PA10	I/O	FT_fa	-	RTC_REFIN, TIM1_CH3, I2C1_SDA, SPI2_MOSI/I2S2_SD, USART1_RX, DEBUG_RF_HSE32RDY, TIM17_BKIN, CM4_EVENTOUT	COMP1_INM, COMP2_INM, DAC_OUT1, ADC_IN6
34	E4	B9	PA11	I/O	FT_fa	-	TIM1_CH4, TIM1_BKIN2, LPTIM3_ETR, I2C2_SDA, SPI1_MISO, USART1_CTS, DEBUG_RF_NRESET, CM4_EVENTOUT	COMP1_INM, COMP2_INM, ADC_IN7
35	D5	A9	PA12	I/O	FT_fa	-	TIM1_ETR, LPTIM3_IN1, I2C2_SCL, SPI1_MOSI, RF_BUSY, USART1_RTS, CM4_EVENTOUT	ADC_IN8
36	D7	B8	PA13	I/O	FT_a	-	JTMS-SWDIO, I2C2_SMBA, IR_OUT, CM4_EVENTOUT	ADC_IN9
-	C2	B7	VSS	S	-	-	-	-
-	A2	A7	VDD	S	-	-	-	-
37	-	A8	VBAT	S	-	-	-	-
38	-	C7	PC13	I/O	FT	-	CM4_EVENTOUT	TAMP_IN1/ RTC_OUT1/RTC_TS/ WKUP2
39	B3	B6	PC14-OSC32_IN	I/O	FT	-	CM4_EVENTOUT	OSC32_IN
40	A4	C5	PC15-OSC32_OUT	I/O	FT	-	CM4_EVENTOUT	OSC32_OUT
-	-	B5	VREF+	S	-	-	-	-
41	B5	A5	VDDA	S	-	-	-	-
-	C4	-	VSS	S	-	-	-	-

Table 19. STM32WL55/54xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
42	C6	A4	PA14	I/O	FT_a	-	JTCK-SWCLK, LPTIM1_OUT, I2C1_SMBA, CM4_EVENTOUT	ADC_IN10
43	A8	B3	PA15	I/O	FT_fa	-	JTDI, TIM2_CH1, TIM2_ETR, I2C2_SDA, SPI1_NSS, CM4_EVENTOUT	COMP1_INM, COMP2_INP, ADC_IN11
-	-	B4	PB15	I/O	FT_f		TIM1_CH3N, I2C2_SCL, SPI2_MOSI/I2S2_SD, CM4_EVENTOUT	-
44	A6	-	VDD	S	-	-	-	-
-	B7	-	VSS	S	-	-	-	-
49(1)	G6	-	VSS	S	-	-	-	-
45	B9	B2	VFBSMPS	S	-	-	-	-
46	A10	A2	VDDSMPS	S	-	-	-	-
47	B11	B1	VLXSMPS	S	-	-	-	-
48	C10	A1	VSSSMPS	S	-	-	-	-

1. Pin 49 is an exposed pad that must be connected to V_{SS}.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Weber R., *Internet of Things - New security and privacy challenges. Computer Law and Security Review*, 2010.

[2] Dave Evans., *How the Next Evolution of the Internet Is Changing Everything. Cisco Internet of Things White Paper*, 2011.

[3] Stephen E. Deering and Robert M. Hinden., *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification*, 1998.

[4] Charith Perera et. al., *Sensing as a Service Model for Smart Cities Supported by Internet of Things. Transactions on Emerging Telecommunications Technology 25 (1): 81–93*, 2014.

[5] Ma HD, *"Internet of things: Objectives and scientific challenges". Journal of computer science and technology 26 (6): 919-924*, 2011.

[6] In Lee and Kyoochun Lee., *"The Internet of Things (IoT): Applications, investments, and challenges for enterprises, Business Horizons, 58, 431-440*,2015 .

[7] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M., *Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660.*, 2013.

[8] Ala Al-Fuqaha et al., *"Internet of Things: A survey on enabling technologies, protocols and applications", IEEE Communications Surveys & Tutorials. DOI 10.1109/COMST.2015.2444095*, 2015.

[9] The European Technology Platform on Smart Systems Integration., *"Internet of Things in 2020: A Roadmap for the future"*, 2008.

[10] Z. Shelby and C. Bormann, 6LoWPAN., *the wireless embedded internet, Wiley, pp. 1-25*, 2011.

[11] Atzori, L., Iera, A., & Morabito, G., *The Internet of Things: A survey. Computer Networks, 54(15), 2787–2805. ttps://doi.org/10.1016/j.comnet.2010.05.010*, 2010.

[12] Silvestre, S. y Salazar, J., *Internet de las cosas*.

[13] Seneviratne P., *Beginning LoRa Radio Networks with Arduino*, 2019.

[14] Pastor, J. y Revenga P., *Curso de Sensores en plataforma Arduino*, 2013.

[15] Pérez, G. A., *Microcontroladores ARM: Advanced Risc Machine*.

[16] Página oficial de STMicroelectronics., *https://www.st.com/content/st_com/en.html*.

[17] How Spreading Factor affects LoRaWAN device battery life., *https://www.thethingsnetwork.org/article/howspreadingfactoraffectslorawandevicebatterylife*.

[18] Pietrosemoli, E., *LoRa details*.

[19] Digi-Key Electronics, *Desarrollar con LoRa para aplicaciones IoT de baja tasa y largo alcance*.

[20] The New Now, *Etiquetas electrónicas para internet de las cosas*.