Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MÁSTER

"SISTEMA DE CAPTURA Y MONITORIZACIÓN DE VARIABLES FISIOLÓGICAS BASADO EN TECNOLOGÍAS IOT Y PLATAFORMAS DE BAJO COSTE"

Estudiante *PONS, VILLANUEVA, SERGIO*
Director/Directora *IRIGOYEN, GORDO, ELOY*
Departamento GICI
Curso académico *2020-2021*

Bilbao, 2, SEPTIEMBRE, 2021>

AGRADECIMIENTOS

En primer lugar, quiero agradecerle a mi tutor, Dr. Eloy Irigoyen Gordo, por su ayuda y disposición durante el desarrollo de este trabajo final de máster.

También quiero agradecer al GICI por brindarme todos los recursos y herramientas que fueron necesarios para llevar a cabo el proceso de investigación.

y en especial;

A mis padres, Ana y Ramón, por haber sido un apoyo constante a lo largo de mi vida.

RESUMEN

Resumen

En este Proyecto Fin de Máster se plantea y resuelve la implementación de un sistema de captura de señales fisiológicas en una persona, para poder enviarlas a un recurso remoto donde procesarlas, analizarlas y extraer información.

Se ha seguido una estructuración de cuatro niveles basada en el modelo OSI, todo ello construido desde un planteamiento de uso de plataformas de bajo coste y software de libre acceso (*open source*).

En primer lugar, se ha realizado un estudio minucioso de las distintas alternativas existentes en relación a la captura de señales fisiológicas con dispositivos tipo Arduino Nano, dónde el Arduino Nano 33 IoT fue seleccionado. Posteriormente, se han analizado los distintos protocolos de comunicación y *middlewares* existentes, descartando las opciones no válidas y seleccionando WiFi y MQTT como protocolos de comunicación y Eclipse Mosquitto como *middleware* MQTT (servidor/*broker*). Se ha seleccionado InfluxDB como base de datos y Node-RED como herramienta puente entre el *broker* y la base de datos. Acto seguido, se ha utilizado Grafana como software para la monitorización de los datos recopilados.

Por último, se han realizado unos primeros estudios de la gestión de avisos y alertas mediante Node-RED y se han realizado pruebas iniciales con otro dispositivo basado en el ESP32 de Espressif.

Como resultado, se ha obtenido un sistema IoT capaz de almacenar y monitorizar en tiempo real señales biológicas.

Palabras clave: IoT, MQTT, Arduino Nano, Node-RED, Open-source, Bioseñales

INDICE DE CONTENIDOS

Índice de contenido

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1 INTRODUCCIÓN

Hace años que vivimos en la era de la información, donde todo está conectado a Internet. Desde ciudades inteligentes (SmartCities) hasta relojes inteligentes (SmartWatches), se pretende dotar de una denominada "inteligencia" a los dispositivos para obtener información y que puedan comunicarse entre ellos para trabajar de forma conjunta.

En 2009, surgió el **Internet de las Cosas o IoT** (Internet of Things) cuando Kevin Ashton, profesor del Instituto Tecnológico de Massachusetts (MIT), utilizó el término por primera vez, aunque el concepto ya se utilizó en 1990.[1] El IoT se basa en "*una red de objetos físicos (cosas) que incorporan sensores, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet*". [1]

Aunque actualmente se fabrican objetos físicos que incorporan sensores capaces de comunicarse mediante WiFi y/o Bluetooth, como toda novedad en el mercado, tienen un coste elevado en comparación a los que carecen de esas capacidades. Es por ello que en los últimos años ha habido un auge de la creación de **microcontroladores** o **plataformas de desarrollo de bajo coste** para realizar proyectos caseros o DIY, por sus siglas en inglés (*Do It Yourself*), y de esta forma incorporar elementos que en un principio carecían de las capacidades IoT a la red de Internet.

Hoy en día el IoT se aplica en diversos campos como la industria, la domótica, la agricultura o **la salud**. En base al campo de aplicación y teniendo en cuenta los dispositivos de los que se disponen, se han generado multitud de plataformas digitales dedicadas a la generación de nuevas funcionalidades, herramientas y proyectos que impulsen aún más su uso.

Soluciones orientadas al uso sanitario que utilizan estos dispositivos están proliferando tanto en el mercado como en el mundo académico. Uno de los principales desafíos que presentan es el tratamiento de la información recopilada del usuario/paciente. El **protocolo de comunicación** escogido debe ser seguro, estable y robusto. Al trabajar con información sensible, se debe garantizar en todo momento la confidencialidad y seguridad de los datos manejados.

Con el fin de lograr desarrollar una solución económica y accesible, tanto en el ámbito socio-sanitario, como en el de atención a peronas dependientes, incidiendo principalmente en la comunicación entre dispositivos y el servidor, este TFM se ha estructurado en los siguientes capítulos:

- **Capítulo 1: Introducción**. En este capítulo se realiza una presentación del proyecto, donde se incluye el contexto.
- **Capítulo 2: Objetivos y alcance.**
- **Capítulo 3: Beneficios.**
- **Capítulo 4: Estado del arte.** Aquí se detalla el estado de los últimos años en relación a la captura de señales fisiológicas disponibles, tanto propuestas, como soluciones comercializadas.
- **Capítulo 5: Análisis de alternativas.** Este capítulo estudia en detalle las propuestas que mayor relevancia tienen para este trabajo, permitiendo seleccionar posteriormente para cada elemento que conforma el escenario de trabajo, aquella que cumpla con los requisitos fundamentales de una comunicación estable, segura y robusta. Para finalizar este capítulo, se indican los elementos finales con los que se ha realizado el trabajo.
- **Capítulo 6**: **Desarrollo de trabajo.** Este capítulo se centra en la implementación de los elementos seleccionados en el capítulo anterior. Se han realizado las pruebas de

funcionamiento empezando por lo más básico y se ha ido incrementando la complejidad conforme se incorporaban los elementos para desplegar la solución final buscando realizar el objetivo principal. También se han estudiado las formas de exprimir capacidades de medida y gestión del dispositivo AN33-IoT para obtener la información deseada lo más rápido y eficientemente posible durante la realización. Por último, se han realizado pruebas iniciales de desarrollos futuros planteados.

- **Capítulo 7: Análisis de resultados y discusión final**. Esta sección detalla los resultados y conclusiones obtenidas a partir de las pruebas realizadas en el capítulo anterior.
- **Capítulo 8**: **Plan de proyecto y planificación.** Este capítulo contiene la planificación realizada para este proyecto representada en un diagrama de Gantt.
- **Capítulo 9: Conclusiones.** Para finalizar, este capítulo resume todo lo realizado destacando los aspectos más relevantes.
- **Capítulo 10: Referencias bibliográficas.**
- **Capítulo 11: Anexos**.

1.1 Contexto actual.

La situación actual en la que se encuentra nuestra sociedad pone de manifiesto la debilidad que existe en cuestiones de monitorización y seguimiento de enfermedades y patologías en personas sensibles. Prueba evidente de todo ello es lo acontecido con el COVID-19, que ha cogido a todos los sistemas de salud del mundo totalmente a contrapié.

Debido al rápido crecimiento de la población y de los costes médicos, ofrecer una buena **atención médica** es cada vez una tarea más complicada. La cantidad de personas que necesitan de un sistema capaz de monitorizar la salud y de detectar alteraciones de ella en tiempo real para actuar rápidamente y tomar las medidas necesarias, es cada vez más elevado. Su ámbito de aplicación va desde pacientes de hospital, personas mayores, personas con necesidades especiales o deportistas. Esta situación combinada con el IoT ha generado conceptos nuevos como **eSalud (eHealth)** o el **Internet de las Cosas Médicas (IoMT).**

Si bien en la comunidad científica se estudian y proponen soluciones para hacer más accesible por parte de entornos médicos y/o familiares la información del estado de las personas bajo cuidado, todavía hay mucho camino que andar, sobre todo pensando en los sectores más desfavorecidos.

Específicamente lo que en este trabajo se pretende resolver es la creación de una solución capaz de capturar y monitorizar variables fisiológicas y que esté basado en tecnologías IoT y plataformas de bajo coste.

CAPÍTULO 2

OBJETIVOS Y ALCANCE

2 OBJETIVOS Y ALCANCE

2.1 Objetivos

En este TFM se plantea dar continuidad a una línea de trabajo en el Grupo de Investigación de Control Inteligente (GICI), llegando a implementar un sistema integral que pueda ofrecer resultados de aplicación real, a través de un primer prototipo.

El objetivo principal es la validación de una solución capaz de capturar y monitorizar variables fisiológicas basado en tecnologías IoT y plataformas de bajo coste.

Los objetivos parciales a abordar en este TFM son los siguientes:

- Estudiar cuáles son las alternativas existentes en relación a la captura de señales fisiológicas con plataformas tipo Arduino IDE y que capacidades de comunicación muestran. Específicamente se trabajará con Arduino Nano 33 IoT.

- Estudiar las capacidades de medida y gestión de los datos del dispositivo IoT seleccionado para lograr obtener la información deseada lo más rápida y eficientemente posible de forma que se pueda exprimir todo su potencial. Para ello se estudiarán las características intrínsecas del dispositivo y se realizarán pruebas para determinar dónde se localizan las limitaciones físico-temporales del mismo.

- Seleccionar las herramientas necesarias para obtener una solución de comunicación que garantice la transferencia de datos de forma segura, estable y robusta, probando para ello diferentes tecnologías. Se deberá garantizar la confidencialidad y seguridad de los datos manejados.

- Estudiar las soluciones ya propuestas desde el GICI (DISA), en la etapa de análisis de datos (señales fisiológicas), para dar una continuación a las mismas con este trabajo. Dejar disponibles los datos adquiridos en el servidor, garantizando que los datos se entregan ordenadamente y cumpliendo con las restricciones temporales oportunas.

Objetivos parciales secundarios:

- Estudiar cuál es la posibilidad de conectar varios dispositivos al servidor comprobando su correcto funcionamiento. Determinar el número máximo de dispositivos posibles a conectar al servidor.

- Aprender a usar dispositivos alternativos a Arduino, en concreto el modelo ESP32 de Espressif. Comprobar la posibilidad de su funcionamiento utilizando los dos núcleos.

- Realizar las primeras pruebas de una implementación que tras analizar los datos ofrezca un resultado en cuanto a la identificación de eventos patológicamente anómalos o peligrosos. Complementarlo con un sistema generador de alarmas y comunicación, tanto al usuario, como a las personas de su entorno médico y social.

2.2 **Alcance**

En este TFM se propone el desarrollo de un dispositivo de adquisición de señales fisiológicas, que sea económico, vestible/portable fácilmente, fiable y que mantenga todas las componentes necesarias para garantizar la correcta utilización de los datos/señales tomadas de los usuarios. Esta innovadora solución tecnológica quiere dar respuesta a los problemas de atención y control, que aparecen en colectivos altamente sensibles ante situaciones como la de la presente pandemia.

La presente propuesta plantea diseñar y validar un dispositivo que, basado en tecnologías IoT, permita recoger señales fisiológicas del usuario, para que de forma continua sean enviadas de modo seguro a un servidor que realizará las funciones de analizar el estado del paciente, determinar si se identifica algún tipo de cambio que reproduzca alguna patología concreta y permita comunicar dicha situación a él mismo, o a aquellas personas que estén al cuidado o tutorización del mismo. Esta transmisión de datos se realizará por canales inalámbricos (WiFi), incorporando protocolos de comunicación seguros, que permitan preservar la confidencialidad e intimidad del usuario.

Adicionalmente, esta comunicación se podrá apoyar en redes de telefonía, a través de tarjetas SIM. Tecnológicamente, este proyecto se implementará para recoger señales fisiológicas como son el ECG, la PPG, nivel GSR y temperatura, entre otras. Además, se desarrollará con plataformas de bajo coste, para que la solución final pueda llegar al mayor número de usuarios posible. Se plantea, además, trabajar sobre LINUX para poder gestionar mejor las funciones y características de PC en la comunicación.

CAPÍTULO 3

BENEFICIOS

3 BENEFICIOS

3.1 Beneficios sociales

Disponer de un dispositivo equiparable a los que se pueden obtener comercialmente pero que al mismo tiempo sea asequible para la gran mayoría de la población supone una mejora notable en la calidad de vida de la gente [2][3][4]. Gracias a la tecnología basada en IoT obtener este tipo de dispositivos de bajo coste es ya una realidad.

Los médicos dispondrán de un sistema de monitorización y alertas de sus pacientes en tiempo real que permitirán una rápida actuación por parte de los sanitarios o la detección precoz de problemas de salud.

3.2 Beneficios económicos

En el caso de la medicina, los dispositivos comerciales están sujetos a los términos y condiciones de la empresa asociada al producto, pudiendo obligar al comprador a tener contratado otros productos y/o servicios encareciendo el gasto final realizado por el consumidor.

Dotar a aquellas herramientas de la capacidad de comunicarse da la posibilidad de obtener un mejor seguimiento del estado de los procesos y de esta forma mejorar su administración y control del proceso productivo, generando así beneficios económicos para la empresa.

Disponer de un dispositivo *open-source* (código abierto) brinda la posibilidad al cliente de no depender de terceros para su implementación. Además de ser un dispositivo de bajo coste, no se incumple ninguna garantía en caso de querer realizar modificaciones en el mismo.

3.3 Beneficios técnicos

Este TFM utiliza una metodológica que permite determinar la viabilidad de los dispositivos de bajo coste, en concreto el Arduino Nano IoT 33, en función de los requisitos y escenario determinados.

De esta forma, se pretende comprobar si el dispositivo seleccionado para la realización del proyecto de GICI es viable. Obtener una valoración positiva como resultado del estudio permitirá la continuación del proyecto en el cual se aplicarán técnicas inteligentes relacionadas con el *Machine Learning* y *Deep Learning*.

CAPÍTULO 4

ESTADO DEL ARTE

4 ESTADO DEL ARTE

La realización de un proyecto IoT tiene que tener en cuenta una serie de elementos hardware y software para su correcto funcionamiento.

Un estudio exhaustivo del marco tecnológico del IoT es realizado por D. Yacchirema Vargas [4] en su tesis doctoral. Esta detalla las características, componentes fundamentales de construcción, arquitecturas de referencia, desafíos y aplicaciones potenciales del IoT.

Siguiendo la siguiente arquitectura de alto nivel de cuatro etapas o niveles [4] se ha realizado un estudio del arte de los diferentes elementos que las componen.

Ilustración 1 - Arquitectura de alto nivel IoT de cuatro niveles.

En el caso de este TFM, los elementos que forman parte de su arquitectura son:

1. **Nivel de percepción**:
    a. **Dispositivo**: Es el hardware principal donde se implementará el control a nivel de *Edge Computing*. También puede ser software en caso de usarse una máquina virtual. La elección del dispositivo restringe la opción de la plataforma o marco de desarrollo con la que se puede trabajar.
    b. **Sensor**: Son los elementos hardware encargados de obtener las variables fisiológicas para posteriormente trabajar con ellas.
2. **Nivel de conectividad:**
    a. Protocolo de comunicación: Determina las normas de comunicación entre los distintos elementos que conforman el proyecto. Estos protocolos engloban las capas física/enlace, red/transporte y aplicación del modelo OSI.
3. **Nivel de servicio:**
    a. Bróker de mensajería: Software intermediario entre dispositivo de bajo coste y el servidor de datos. Se encarga de la distribución de datos.
    b. Servidor de datos: Elemento software y hardware donde se almacenan los datos recopilados por el sensor tras haberse establecido la conexión a través del *broker*.
4. **Nivel de aplicación:**
    a. Software complementario: Software auxiliar que ha servido de ayuda y/o mejora para la implementación de la solución creada.

A continuación, se describe información relevante reciente para cada uno de ellos.

4.1 Nivel de percepción

Es la primera etapa de la arquitectura donde se interactúa con el entorno. Aquí se realiza la identificación de los dispositivos físicos IoT, se recopila la información mediante los sensores realizando posteriormente su conversión a señales digitales.[4]

4.1.1 Dispositivos

En la última década el IoT aplicado a la medicina ha tomado un incremento exponencial, tanto es así que ya se habla directamente del IoMT [5]. Ha habido un auge de creación de dispositivos de medida de señales fisiológicas y comunicación inalámbrica tanto propuestas como comercializadas.

Algunos de estos ejemplos pertenecen a empresas privadas que facilitan el dispositivo como un producto más dentro de sus soluciones en venta. Existen algunas de ellas que junto al elemento que captura y envía la información, incorporan otros elementos, formando así un sistema más elaborado. Este sistema brinda mayores capacidades de las que se dispondrían si únicamente si dispusiera del dispositivo de bajo coste.

Protección Senior [6] es una solución que ofrece la empresa Securitas Direct, la cual además de un reloj que captura señales fisiológicas, incorpora otro elemento físico, la Unidad Central, que contiene altavoces y micrófono. El paquete ofrece atención inmediata 24h, en caso de pulsarse el botón SOS, desde el reloj o la Unidad Central, se atiende al cliente en menos de 18 segundos. Los servicios que ofrece no se centran únicamente en emergencias, también incluye otros servicios para mejorar la calidad de vida del cliente, como del de Mayordomo o Telefarmacia 24h. La asistencia médica telefónica es otro de sus servicios, siendo posible obtener un diagnóstico médico por vídeo consulta. Por último, incluye también una aplicación telefónica desde la que, aquellos a los que se le de acceso, puedan monitorizar puedan monitorizar el estado del cliente.

Multitud de empresas pueden encontrarse hoy en día ofreciendo servicios similares, como iHealth [7], Vodafone con su servicio V-SOS [8], Neki [9] o Sensovida [10]. Sin embargo, disponer de todos estos servicios supone un coste extra en los hogares impidiendo que esté al alcance de muchas personas.

La sociedad capitalista en la que vivimos hoy en día está provocando un avance tecnológico sin precedentes. Cada año sale al mercado un nuevo producto que deja obsoleto al anterior. Es por ello que grandes marcas crean productos que incorporan nuevas funcionalidades o servicios. Actualmente existen en el mercado una gran variedad de relojes inteligentes que no solo miden el ritmo cardiaco o la oxigenación en sangre, también ofrecen otras opciones.

Empresas relacionadas con la telefonía están aprovechando las capacidades de relojes inteligentes (*smartwatch*), creando dispositivos más comercializados. Todos ellos tienen en común su capacidad de conexión con el *smartphone*, permitiendo realizar llamadas o enviar mensajes. El Xiaomi Mi Band 6 [11] ofrece además de las dos capacidades mencionadas, otros programas como la monitorización de la calidad del sueño, visualización de imágenes, supervisión del estrés, ejercicios de respiración, modo deportivo o seguimiento de la salud femenina.

Otras empresas que también disponen de estos dispositivos con características equivalentes o iguales son Apple [12] , Samsung [13] o Huawei [14].

Por otro lado, hay empresas fuera del mercado de telefonía y enfocadas más en el ámbito deportivo que también han desarrollado su propio smartwatch. Estos ofrecen características similares a los productos mencionados y también poseen la capacidad de comunicarse con el móvil a través de una app propia que se puede obtener desde la respectiva tienda de aplicaciones

del sistema operativo (SO) que posea el teléfono móvil. Una comparativa de todos estos productos puede encontrarse en [15].

Sin embargo, un aspecto negativo de estos productos comercializados es que no se dispone de acceso al software, abstrayendo totalmente al consumidor de cómo están programados, de forma que únicamente se centre en su uso. Al no ser de libre acceso, no se puede ver cómo han hecho el proceso, ni obtener la documentación para entender totalmente su funcionamiento.

Empresas dedicadas a la creación de dispositivos de bajo coste, normalmente placas de microcontrolador, permiten el desarrollo de proyectos *OpenSource*. El hecho de tener acceso a toda la documentación y no estar sujeto a las restricciones de confidencialidad de una empresa, permite que se creen plataformas digitales en las que compartir el trabajo realizado y ayudarse mutuamente.

Una de las empresas de desarrollo de software y/o hardware más conocidas es Arduino [16], la cual posee una larga lista de placas de microcontrolador con distintas funcionalidades. Dentro de los modelos disponibles, destaca el Arduino Nano 33 IoT [17], que como el nombre indica, tiene como objetivo su uso en proyectos relacionados con el Internet de las Cosas. Al mismo tiempo, Arduino posee una plataforma de desarrollo propia, Arduino IDE [18], y su propio foro, Arduino Forum [19].

Como con todo gran éxito, existen dispositivos que le hacen la competencia, el trabajo realizado por R. Requena recoge algunos de ellos [20], como Adafruit [21] y Sparkfun [22]. Ambas crean sus propias placas de microcontrolador, pero hacen uso del entorno Arduino IDE para su programación en la mayoría de ellas.

Otras empresas, como Bitalino [23], crean soluciones más orientadas específicamente al ámbito de la salud. Dispone entre sus productos, placas *all in one* (todo en uno) que incorporan de todo lo necesario para trabajar con señales biológicas como el microcontrolador, los sensores y los actuadores. Sin embargo, también obliga al consumidor a utilizar su propio software para la visualización de datos en tiempo real.

Por otro lado se encuentra Espressif Systems [24] que se dedica más a la fabricación de chips y módulos de comunicación. Dos de sus modelos más destacados son el ESP8266 [25] y su versión más reciente, el ESP32 [26]. A partir de ellos, otras empresas fabrican sus propias placas de desarrollo con las que se ha realizado los trabajos académicos más recientes. La empresa ofrece su propio *Development Framework* (marco de desarrollo), ESP-IDF [27], no obstante también es posible utilizar Arduino IDE realizando una serie de pasos para importar las librerías necesarias.[28]

La empresa Raspberry Pi [29] también dispone de dispositivos de bajo coste, sin embargo, a diferencia de los anteriores, que son microprocesadores, estos son microcontroladores. Una comparativa entre microcontroladores y microprocesadores es realizada por el autor E. Crespo [30].

Existen libros dedicados a la realización de proyectos IoT con Arduino, como "*IoT: Building Arduino-Based Projects*" [31], "*Beginning Arduino Nano 33 IoT*" [32], "*Building Arduino Projects for the Internet of Things*" [33]. Al igual que con Arduino, los modelos de Espressif también cuentan con libros dedicados a su uso en proyectos. Algunos ejemplos son "*Internet of Things Projects with ESP32*" [34] o "*Interget of Things with ESP8266*" [35].

Si bien no son libros especializados en el IoMT, sirven de gran utilidad para adentrarse en esta temática del IoT y permiten realizar los primeros pasos que, aunque a simple vista parezcan sencillos, son vitales para el desarrollo de trabajos de mayor complejidad.

También hay una gran variedad artículos de investigación que utilizan dispositivos Arduino para proyectos IoT relacionados con la monitorización de pacientes [36].

Por otro lado, también existen investigaciones y trabajos realizados con los modelos basados en los chips de Espressif [37][38][39]. O páginas web dedicadas a los modelos ESP32 y ESP8266 que contienen toda la información para realizar proyectos [40][41].

El trabajo final de master realizado por A. Martín Muñoz [42] presenta un análisis del funcionamiento de distintos tipos de redes formadas por dispositivos ESP8266. Aunque el estudio de topologías no entra dentro del ámbito de estudio de este proyecto, es necesario tener en cuenta las capacidades que presenta el dispositivo para futuras aplicaciones y desarrollos.

4.1.1.1 Máquina virtual

Una máquina virtual es un software que permite cargar un sistema operativo diferente al ordenador físico (huésped o *host*) donde se realiza la emulación.

Se pueden usar dos tipos de máquinas virtuales dependiendo de su funcionalidad: las de sistema y las de proceso.

Mientras que las de proceso únicamente permite ejecutar un proceso en concreto, como una aplicación, las de sistema emulan a un ordenador completo, lo que le permite hacerse pasar por otro dispositivo, como un PC, y de este modo instalarle otro sistema operativo.

Las máquinas de sistema poseen, aunque todos emulados, sus propios elementos hardware como discos duros, tarjeta de red o memoria. Esto elementos hardware virtuales son recursos reservados que provienen del ordenador huésped.

El sistema operativo simulado funciona con normalidad, *"sin que sepa que en verdad está metido dentro de una burbuja dentro de otro sistema operativo"* [43].

Algunas de las máquinas virtuales más conocidas para Windows son VirtualBox de Oracle [44] o VMware Workstation [45].

4.1.2 Sensores

Existe una gran variedad de sensores, sin embargo, los que aquí conciernen son aquellos utilizados para la obtención de bioseñales. Estos biosensores permiten monitorizar parámetros del cuerpo humano y realizar pruebas tales como la electromiografía (EMG), electrocardiografía (ECG), actividad electrodermal (EDA), electroencefalografía (EEG), fotopletismografía (PPG) o respiración (PZT).

La combinación del IoT con estos sensores crean la posibilidad de realizar aplicaciones que incluyen diagnósticos clínicos y monitorizaciones de diversas actividades como pueden ser el ritmo cardiaco, salud femenina, glucosa o el sueño. [3].

Un estado del arte de los sensores para el IoMT es realizado por P. Ray et al. [3]. Este trabajo posee tablas comparativas donde se muestran características importantes para su selección como el nombre del producto, su fiabilidad para el testeo, si tiene soporte IoT, su propósito de uso, la usabilidad de los datos recolectados, la energía consumida y el coste comparado con el resto de sensores pertenecientes a la misma tabla.

En Bitalino se pueden encontrar kits [46] que, en función del modelo, incorporan unos sensores u otros. Sin embargo, no se especifica el modelo en ninguna fuente de información accesible, al menos, sin realizar la compra del producto que, debido a su alto coste, no están al alcance de todo el mundo.

El trabajo realizado por F.Medina y A. Paternina [47] se realiza el diseño y construcción de un oxímetro de pulso inalámbrico con un sensor MAX30100.

4.2 Nivel de conectividad

La capa de Conectividad es la encargada de soportar y realizar las conexiones entre los distintos dispositivos físicos, servidores y dispositivos de red englobando las capas física/enlace, red/transporte y aplicación del modelo OSI (*Open Systems Interconnection*).

La tesis doctoral realizada por D. Yacchirema Vargas [4] sobre la interoperabilidad de los dispositivos IoT explica el estándar RFC 7452 definido por el *Internet Architecture Board (IAB)*, el cual describe tres patrones de comunicación para los dispositivos IoT: dispositivo a dispositivo (D2D), dispositivo a la *cloud* (D2C) y dispositivo a *gateway* (D2G). Realiza también una comparativa de algunos de los protocolos de conectividad más adoptados, separándolos en función de las capas del modelo OSI.

El trabajo realizado por R. Requena [20] menciona también la importancia del uso de especificaciones de estándares y protocolos para hacer frente a los problemas de heterogeneidad entre las distintas plataformas. Identifica tres niveles de incompatibilidad: de la tecnología del dispositivo, de comunicación y de datos.

Las similitudes entre los distintos protocolos de comunicación o *frameworks* disponibles para el intercambio de información IoT sugieren el potencial de la interoperabilidad. Debido a que los dispositivos IoT tienen limitaciones a nivel de poder de procesamiento de recursos computacionales es necesario que el protocolo de comunicación escogido debe ser fiable, escalable, ligero, interoperable, extensible y seguro [48].

4.2.1 Protocolos de la capa física y de enlace

Con el objetivo en mente de obtener mejores aplicaciones IoT, se están diseñando nuevas tecnologías y rediseñando otras existentes para satisfacer las necesidades que tienen los dispositivos inalámbricos IoT. Algunos de estos requisitos son: rango de cobertura amplio, costos operativos y de implementación o el bajo consumo de energía [4].

En la capa física y de enlace, destacan el uso de Wi-Fi, WiFi HaLow, Bluetooth, ZigBee, LoRaWAN entre otros. Son los protocolos y mecanismos de comunicación tradicionales para los sensores que más se usan para la conectividad de dispositivos IoT, cada una con sus pros y sus contras.

4.2.2 Protocolos de la capa de red y transporte

D. Yacchirema Vargas [4] aclara que aunque la conectividad IP de extremo a extremo esté disponible en los puntos finales mediante los protocolos como TCP o UDP, existen situaciones en la que los dispositivos son incapaces de ejecutar IP. Como solución propone una capa de adaptación de IPv6 sobre LowPAN (6LowPAN).

4.2.3 Protocolos de la capa de aplicación

El protocolo de comunicación seleccionado es un elemento clave para el intercambio de mensajes entre el dispositivo y el sistema de almacenamiento. Es por ello que se han realizado múltiples estudios detallando sus características, comparando sus ventajas y desventajas, para seleccionar aquel que se adecue más al proyecto en concreto.

En el TFM realizado por G. Gil Inchaurza [49] se realiza un estudio de las alternativas existentes para la adquisición de datos más extendidos actualmente. Son los protocolos XMPP (Extensible Messaging and Presence Protocol), AMQP (Advanced Message Queuing Protocol) y MQTT

(Message Queing Telemetry Transport). Al final, tras la comparativa, el autor concluye que MQTT es el protocolo más indicado para proyectos IoT.

Existen libros que realizan comparativas más exhaustivas como en "*IoT: Building Arduino-Based Projects*" [31] que además de comparar los tres protocolos mencionados en el párrafo anterior, también se incluyen los protocolos HTTP (Hypertext Transfer Protocol), UPnP (Universal Plug and Play) y CoAP (Constrained Application Protocol). U otros como "*Building Arduino Projects for the Internet of Things*." [33] que directamente se centran en las dos posibilidades más factibles, HTTP y MQTT, en función del objetivo que tenga el proyecto.

Dada la importancia del protocolo MQTT para las soluciones IoT, hay libros dedicados exclusivamente a dicho protocolo. El libro "MQTT Essentials – A Lightweight IoT Protocol" [50] permite al lector adentrarse en la utilización de este protocolo y conocer todas las posibilidades que ofrece.

Múltiples trabajos de investigación también seleccionan MQTT como protocolo de comunicación para proyectos IoT relacionados con la monitorización en tiempo real de la salud [37][39][51].

El trabajo realizado por I. Ibiricu Elizondo [39] muestra la importancia de disponer de servidor DHCP (*Dynamic Host Configuration Protocol*) en el caso de tener que administrar múltiples direcciones IP. Como su nombre indica, es un protocolo de configuración dinámica del host basado en un modelo cliente/servidor que proporciona automáticamente a un *host* IP parámetros relacionados con la configuración de red como dirección IP, puerta de enlace predeterminada o máscara de subred. Dado que no puede haber dos dispositivos con los mismos parámetros mencionados, su utilización simplifica el proceso y evita posibles errores humanos en el caso de introducirlos manualmente.

4.3 Nivel de Servicio

La capa de Servicio o capa de *middleware* se encarga del almacenamiento y gestión de los datos (base de datos) así como del procesamiento de dicha información.

"*Un **middleware** es un software de aplicación que abstrae la complejidad y heterogeneidad de las redes de comunicación subyacentes, hardware, sistemas operativos, lenguajes de programación, etc., para permitir a una entidad interactuar o comunicarse con otras entidades, que no fueron diseñaos originalmente para conectarse*"[4]

El trabajo de R. Requena [20] destaca la importancia de los *middlewares* a la hora de facilitar el desarrollo de la solución, puesto que le permite al desarrollador centrarse directamente en su aplicación.

Artículos académicos recientes muestran la utilización también de microprocesadores, como Raspberry Pi [29], como servidores físicos [5] donde instalar estas herramientas.

4.3.1 Brokers de mensajería

Es un servidor que gestiona los canales eventos (*topics*) en las arquitecturas publicista/subscriptor. Se encarga de la distribución y gestión de los datos entre los clientes.

G. Gil Inchaurza [49] realiza un estudio comparativo brokers de mensajería, a partir de ahora nos referiremos a ellos como "*broker"* o *"brokers*", más utilizados actualmente. Entre ellos se encuentran RabbitMQ, Mosquitto, EMQ o HiveMQ. Realiza una comparativa entre ellos teniendo en cuenta sus características de fiabilidad, disponibilidad y seguridad, las cuales

considera requisitos imprescindibles, siendo motivo justificado para descartar un *broker* como opción en el caso de no cumplir alguno de ellos.

Por otro lado, R. Roberto [20] toma como protocolo de comunicación MQTT, por lo que se centra únicamente en realizar una comparativa de las características que ofrecen los *brokers* MQTT. En esta comparativa, además de los *brokers* mencionados en el párrafo anterior, incluye otros como RSMB, MQTT.js o Mosca.

El libro "*MQTT Essentials – A Lightweight IoT Protocol*"[50] ofrece una serie de proyectos realizados con el *broker* Mosquitto en varios sistemas operativos, Linux, macOS y Windows.

4.3.2 Servidores de almacenamiento de datos

El trabajo realizado por I. Ibiricu [39], en los últimos año ha crecido exponencialmente el uso de bases de datos temporales (*Time Series Databases*) debido su uso en aplicaciones IoT o Big Data . Muchas de ellas son de código abierto, lo que permite su uso gratuito.

Lo que se pretende con ellas es almacenar grandes cantidades de datos, normalmente generados en tiempo real, en el histórico para posteriormente visualizarlos con otra herramienta o trabajar con ellos [52].

W. Nazco [52] realiza un estudio detallado sobre las diferentes bases de datos temporales disponibles. Algunas de las más utilizadas son InfluxDB [53], Graphite [54], OpenTSDB [55] o Prometheus [56]. Tras realizar la comparativa de sus características, la base de datos seleccionada por el autor es InfluxDB.

4.4 Nivel de aplicación

Esta capa, a la que se le ha denominado de aplicación, hace referencia a las aplicaciones software. No se está refiriendo a los protocolos de comunicación de la capa de aplicación del modelo OSI.

4.4.1 Herramientas adicionales

Existen hoy en día herramientas adicionales que facilitan la implementación de sistemas IoT, siendo Node-RED [57] una de las más exitosas y utilizadas como puente entre el *broker* y la base de datos. Diversos proyectos muestran su potencial en la creación de proyectos IoT [58][59][60].

4.4.2 Software de Control

Cuando se requiere controlar un gran número de dispositivos una forma de optimizar el tiempo a la hora de programarlos y controlarlos es mediante la utilización de software de control. Mediante ellos es posible tener un control local total de los dispositivos y automatizar el proceso de configuración y actualización vía OTA (*Over-The-Air*)[39].

Algunos de los sistemas más utilizados son ESPHome [61] por su utilización con los dispositivos de Espressif, o Tasmota [62] por su relación con dispositivos SonOff [63].

Ambos sistemas de control se pueden integrar del sistema operativo Home Assistant [64]. Este sistema operativo es un software gratuito y de código abierto. Aunque fue creado originalmente para ser el sistema de control central para dispositivos IoT en el ámbito del hogar, también podría utilizarse en otros, como la industria o la medicina.

El trabajo realizado por I. Ibiricu Elizondo [39] identifica ESPHome como la mejor opción, ya que mediante él es posible controlar y configurar múltiples dispositivos desde la configuración de un archivo YALM.

ESPHome soporta múltiples sensores, y posee de un listado de sensores típicos para la automatización del hogar para poder configurarlos. Lo que lo hace interesante es que también tiene una opción de "*custom components*", permitiendo, mediante código C++, escribir la programación necesaria para aquellos sensores que no estén listados.

4.4.3 Visualización de datos

En ocasiones es necesario ver en persona los valores que se están tomando con los sensores. Existen múltiples herramientas de visualización de datos que nos permiten monitorizarlos y de forma que la información esté plasmada de forma más clara.

Son representaciones gráficas de los datos almacenados en la base de datos como diagramas de barras, mapas, diagramas de quesos, etc.

El trabajo realizado por W. Nazco Torres [52] realiza una comparativa entre algunas de estas herramientas que permiten la visualización de datos mediante la producción de cuadros de mando y basadas en series temporales, en concreto Grafana [65] y Chronograf.

4.5 Conclusión

En este capítulo se ha visto como el despliegue de una solución IoT es necesario tener conocimientos de varias disciplinas, como redes de telecomunicaciones, computación en los distintos niveles (*Edge, Fog, Cloud*) o seguridad informática. Son muchos los elementos a tener en cuenta, pudiéndose realizar estudios dedicados a la realización de comparativas de cada uno de ellos.

En el siguiente capítulo se realizará un estudio comparativo de las diferentes alternativas de cada elemento tenidas en cuenta para realización de este proyecto.

CAPÍTULO 5

ANÁLISIS DE ALTERNATIVAS

5 ANÁLISIS DE ALTERNATIVAS

Siguiendo la misma estructura que la establecida en el estado del arte, en este capítulo se estudian las distintas posibilidades de selección en cada elemento del escenario de trabajo, comparando sus características, para formar parte de la solución final y realizar el desarrollo de este TFM.

El domino de aplicación de este proyecto se centra en el campo de la salud y medicina. Teniendo esto en cuenta y considerando la información recopilada en el Estado del Arte (capítulo 4), únicamente se han realizado comparativas de las distintas alternativas que pueden llegar a ser válidas para el ser seleccionadas como solución final.

Puesto que este trabajo es una continuación de lo realizado en el GICI [66], algunos de los elementos que conforman la selección final han sido determinados previamente. Aun así, se ha considerado relevante mostrar una comparativa de los mismos con otras opciones competitivas. Esto permite verificar si los elementos seleccionados son válidos o si sería necesario modificarlos en trabajos futuros.

En función de los requerimientos establecidos a partir de los objetivos definidos al inicio para este proyecto, a continuación, se procede a la realización de la comparativa de los distintos elementos que conformarán la solución final propuesta.

NOTA: Información más detallada de cada apartado puede encontrarse en los Anexos.

5.1 Dispositivos

El dispositivo seleccionado debe tener determinadas características orientadas a la realización de proyectos IoT. Una de sus características esenciales es la capacidad de comunicación de forma inalámbrica, ya sea vía WiFi o Bluetooth.

Además, otras características que también influirán en su elección son: microcontrolador, tamaño, precio, número de pines, tipo de comunicación aceptada, velocidad del reloj, memoria o alimentación.

Se pretende que el dispositivo escogido tenga un microprocesador que le permita realizar procesamientos con gran velocidad y que sean capaces de almacenar datos en su memoria hasta que puedan enviarlos a un sistema de almacenamiento de datos de forma inalámbrica.

Aunque el dispositivo preseleccionado con el que ha desarrollado este TFM es un Arduino Nano IoT 33[17] se ha considerado oportuno realizar una comparativa con algunos modelos de la misma familia Nano. También se ha realizado una comparativa con los modelos basados en los chips de Espressif dado su gran uso en la realización de proyectos tanto DIY como en el mundo académico.

5.1.1 Arduino

Arduino es "*una plataforma de creación de prototipos electrónicos de código abierto que permite a los usuarios crear objetos electrónicos interactivos*" [16].

Dentro de las posibilidades que ofrece para el desarrollo de aplicaciones orientadas al IoT, las placas a considerar como posible solución para este proyecto son las siguientes:

- Arduino Nano 33 IoT
- Arduino Nano RP2040 Connect

Arduino Nano 33 IoT

El AN-33IoT salió al mercado en 2019 como la versión reducida de la placa MKR WiFi 1010 [67]. Para lograr reducir sus dimensiones y precio, este modelo no posee cargador de batería ni compatibilidad con escudos [68].

Es considerado por la propia empresa Arduino como "*el punto de entrada más fácil y barato para mejorar los dispositivos existentes (y crear otros nuevos) para formar parte del IoT*"[17].

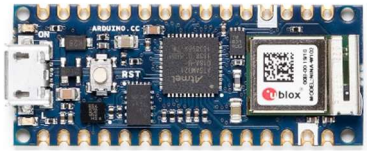


Ilustración 2 - Arduino Nano 33 IoT.[17]

Por otro lado, dentro de la familia Nano 33, también existen las placas Arduino Nano 33 BLE [69] y su versión con más sensores incluidos, Arduino Nano 33 BLE Sense [70]. Poseen, entre otras cosas, un procesador diferente al del AN-33IoT, pero la principal característica que los descarta como opción válida para este proyecto es su ausencia de capacidad de comunicación WiFi sin el uso de escudos. En la propia página de Arduino se pueden encontrar comparativas detalladas entre estos tres dispositivos [71][72].

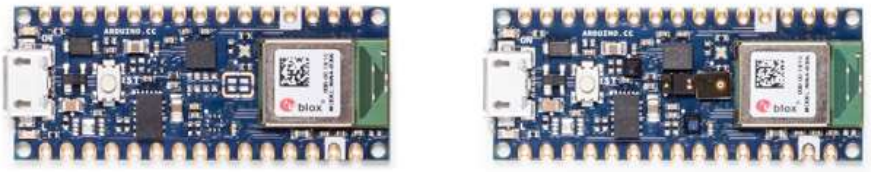


Ilustración 3 - Arduino Nano 33 BLE (izq.) y Arduino Nano 33 BLE Sense (dcha.).[69][70]

El AN-33IoT tiene implementado un Arm® Cortex®-M0 32-bit SAMD21, alejándose de la línea tradicional de Arduino del uso de microcontroladores ATMega. Mediante este nuevo diseño se obtiene una mejora significativa de memoria Flash, SRAM y velocidad de reloj. Aunque el microcontrolador no permita el uso de EEPROM, es posible emularlo mediante la memoria Flash reprogramable.

Su característica más destacable comparándolo con los Nano BLE, es su capacidad de comunicarse vía WiFi 802.11b / g /ny, y Bluetooth 4.2., haciéndolo un candidato idóneo para el desarrollo de soluciones IoT. Además, posee un microchip de encriptación ECC608A, característica muy importante cuando se quiere garantizar la seguridad en la comunicación[71].

Tiene dos procesadores, haciendo posible modificar el módulo de WiFiNINA para utilizar al mismo tiempo el WiFi y el Bluetooth al mismo tiempo. Otra posibilidad es la de introducir una versión superligera de Linux en un módulo y que el microcontrolador principal controle otros dispositivos. Sin embargo, realizar este tipo de modificaciones puede hacer que se rompa e inhabilita la garantía de la placa [17].

5.1.1.1 Arduino Nano RP2040 Connect

Se trata del modelo más reciente de la familia Nano de Arduino que salió al mercado a principios del 2021.

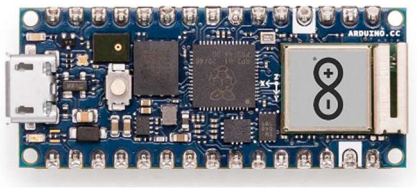


Ilustración 4 - Arduino Nano RP2040 Connect [73].

Su cerebro es el RaspberryPi RP2040 Silicon, un microcontrolador ARM Cortex-M0+ de doble núcleo con una velocidad de reloj de 133 MHz. Al igual que el modelo AN-33IoT, puede comunicarse vía WiFi 802.11b / g /ny y Bluetooth 4.2. Es posible operar con él y programarlo desde un navegador como si estuviera conectado al ordenador por USB. Tiene compatibilidad completa con servicios de Arduino como Arduino Cloud y Arduino IoT Remote [74].

Posee la potencia suficiente como para implementar modelos de aprendizaje automático con *frameworks* como TensorflowLite. Tiene incorporado un micrófono que le permite realizar la activación de sonido, el control de audio o reconocimiento de voz mediante IA. Su sensor de movimiento inteligente (IMU-6 ejes) le indica en qué dirección se mueve, le permite detectar caídas y activarse mediante doble toque [73].

Es compatible con el ecosistema de software RP2040 y también tiene soporte completo para MicroPython.

5.1.2 Espressif

Los productos desarrollados por la marca Espressif tienen dos posibles formas: chip o módulo. Haciendo uso de ellos, se pueden encontrar en el mercado distintas placas creadas por diversas compañías como Adafruit o Sparkfun, incluida la propia Espressif. Un listado de las placas originales se puede encontrar en la propia página de productos de Espressif para las dos series [75].

El ESP32 es el sucesor del ESP8266, lo que le permite hacer como mínimo todo lo que hacía su versión antigua. Por este motivo se descarta el uso de la versión ESP8266 como posible solución, sin embargo, toda la documentación relacionada con proyectos que lo utilizan es válida para el ESP32.

El ESP32, al igual que su versión anterior, es un chip/módulo de bajo coste con WiFi y Bluetooth que le permite realizar aplicaciones IoT [76] .

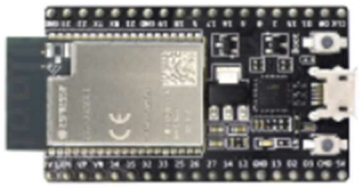


Ilustración 5 – Espressif ESP32 DevKit [75]

Dentro de las distintas posibilidades a la hora de estudiar las características de una placa para cada familia, se ha optado por las distribuidas por la empresa Az-Delivery [77]. Obtener los modelos originales no era factible por motivos de presupuesto y disponibilidad. Todas las placas

basadas en los modelos originales poseen las mismas capacidades de rendimiento y funcionalidades, aunque la distribución de los componentes pueda ser diferentes [78] , por lo que teóricamente no presenta ningún problema elegir una empresa u otra.

El modelo que se ha considerado en este TFM es el ESP32-WROOM-32 NodeMCU basado en el modelo ESP32 DevKit de Espressif.

Ilustración 6 - AZDelivery ESP32 ESP-WROOM-32 NodeMCU [77]**.**

El modelo de la ilustración superior tiene un procesador ESP32 con una CPU de dos núcleos Tensilica LX6, con una frecuencia de reloj de hasta 240 MHz, 448KB de ROM y 512 KB de SRAM. La frecuencia CPU del microcontrolador puede ser modificada desde ESP-IDF o el Arduino IDE, pero para este último se requiere realizar la instalación manual de las librerías necesarias para su uso [78].

Posee un sensor de efecto Hall para detectar cambio en el campo magnético de su entorno y un sensor de temperatura con un rango de 40ºC a 125ºC [78].

5.1.3 Tabla comparativa

La siguiente tabla muestra una comparación de características entre los dispositivos mencionados anteriormente. Toda la información está recopilada de los distribuidores de los respectivos dispositivos**.**

Parámetros	*Arduino Nano 33 IoT*	*Arduino Nano BLE*	*Arduino Nano BLE Sense*	*Arduino Nano RP2040 Connect*	*AZDelivery ESP32*
Microcontrolador	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	nRF52840	nRF52840	Raspberry Pi RP2040 (doble núcleo)	Tensilica Xtensa LX6 (doble núcleo) SoC (ESP-WROOM 32)
Conectividad	Nina W102 uBlox (WiFi and Bluetooth)	nRF52840 (Bluetooth)	nRF52840 (Bluetooth)	Nina W102 uBlox (WiFi and Bluetooth)	Bluetooth y WiFi
Elemento seguro	ATECC6080A	NO	NO-	ATECC608A-MAHDA-T Crypto IC	NO
Tensión operacional	3.3 V	3.3 V	3.3 V	3.3 V	3.3 V
Límite de tensión de entrada	21 V	21 V	21 V	21 V	5 V
Pin I/O de corriente DC	7 mA	15 mA	15 mA	4 mA	15 mA
Velocidad de reloj	48 MHz	64 MHz	64 MHz	133 MHz	80 a 240 MHz
Memoria Flash CPU	256 KB	1 MB	1 MB	16 MB	4MB (externa)
SRAM	32 KB	256 KB	256 KB	520 KB	520 KB
EEPROM	NO	NO	NO	448 KB	448 KB
Pines Digitales I/O	14	14	14	20	34
Pines PWM	11	Todos los digitales	Todos los digitales	20	Todos los de salida
UART	1	1	1	1	1
SPI	1	1	1	1	1
I2C	1	1	1	1	1
Pines Analógicos de entrada	8 (ADC 8/10/12 bit)	8 (ADC 12 bit 200 ksamples)	8 (ADC 12 bit 200 ksamples)	8	18 (ADC 12 bit)
Pines Analógicos de salida	1 (DAC 10bit)	Solo a través de PWM (no DAC)	Solo a través de PWM (no DAC)		2 (DAC 8 bit)
Interrupción externa	Todos los pines digitales	Todos los pines digitales	Todos los pines digitales		
USB	Nativo en el procesador SAMD21	Nativo en el procesador nRF52840	Nativo en el procesador nRF52840	Micro USB	MicroUSB
Sensores	IMU (LSM4DS3)	IMU (LSM9DS1)	IMU (LSM9DS1) Micrófono (MP34DT05) Gestos, luz y proximidad (APDS9960) Presión barométrica (LPS22HB) Temperatura y humedad (HTS221)	IMU (LSM6DSOXTR) Micrófono (MP34DT05)	Efecto Hall Temperatura Capacitivos
Dimensiones	45 x 18 mm	45 x 18 mm	45 x 18 mm	45 x18 mm	56x28 mm
Peso	*5 g*	*5 g*	*5 g*	*6 g*	*10 g*
Precio	16.00€	19.50€	29.00€	22.00€	10€

Tabla 1 - Comparación de dispositivos IoT.

5.2 Sensores

A la hora de realizar la selección de los sensores es necesarios tener en cuenta características fundamentales como su funcionalidad, material del que están hechos y aplicación.

El presente trabajo no abarca la elección de los sensores puesto que es una continuación del trabajo realizado en el GICI.

Los sensores preseleccionados han sido el MAX30100 [79] y el sistema de medición Grove GSR que utiliza un LM324 [80]. Mientras que con el primero se mide la oxigenación en sangre y ritmo cardíaco, el otro mide la respuesta galvánica de la piel.

Ilustración 7 - Sensor GY-MAX30100 [79]

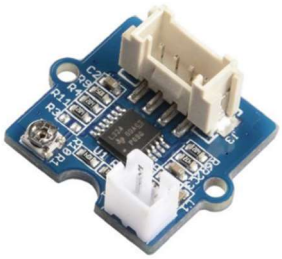


Ilustración 8 - Sensor LM324 [80]

5.3 Protocolos de comunicación

En este apartado se realiza una comparativa para los distintos protocolos de comunicación IoT para cada capa del modelo OSI.

5.3.1 Capa física y de enlace

Wi-Fi y Wi-Fi HaLow

Wi-Fi (Wireless Fidelity) es un protocolo que surgió con el objetivo de proporcionar una conectividad inalámbrica de corto alcance, fácil de usar y de implementar. Sin embargo, debido a su consumo energético, se recomienda que los dispositivos IoT solo usen especificaciones 802.11 b/g [4].

Como solución a los problemas de energía y alcance de este protocolo se creó el WiFi HaLow (802.11ah) [4]. Como se puede observar en la tabla 2, esta nueva versión tiene mayor alcance y trabaja a una banda inferior (1 GHz), es decir, le permite atravesar paredes con mayor facilidad [81].

Bluetooth

Bluetooth se define como "*una tecnología de comunicación inalámbrica de bajo consumo y bajo costo adecuada para la transmisión entre dispositivos móviles en un rango corto*. [4]" Su versión para dispositivos de baja potencia es el Bluetooth LE (BLE), su versión Bluetooth 4.0 en adelante. Esta última versión se diseñó con la idea de transmitir pequeñas cantidades de datos a velocidades bajas en espacios reducidos. Respecto a su versión 4.2, su versión 5 multiplica por dos su velocidad, por cuatro su rango y por ocho la cantidad de datos manteniendo el mismo bajo consumo de energía [82].

LoRaWAN

LoRaWAN es una especificación de una red LPWAN (*Low Power Wide Area Network*). Fue creada con la idea de comunicar dispositivos de bajo coste y bajo consumo alimentados por baterías [83]. Se ha optimizado de forma que se pueda aplicar a grandes redes con cantidades masivas de dispositivos (del orden de millones) de forma que se pueda aplicar tanto en el entorno rural como en el urbano.

Zigbee

Zigbee es considerada como una tecnología inalámbrica dedicada al campo de la domótica y el campo industrial. Está orientada a aplicaciones con comunicaciones de baja tasa de envío de datos en espacios limitados, como edificios o casas. Dentro de sus ventajas se encuentran su bajo consumo energético, robustez, nivel de seguridad, escalabilidad o su capacidad para albergar un gran número de nodos [83]. Por estos motivos es una buena candidata para aplicaciones IoT.

Tabla comparativa – Protocolo capa física/enlace

La siguiente tabla muestra una comparativa de las principales características de las tecnologías de comunicación inalámbricas más usadas.

Características	***WiFi***	***WiFi HaLow***	***Bluetooth BLE v4.2***	***LoRaWAN***	***ZigBee***
Estándar	IEE 802.11 a/b/g/n	IEE 802.11 ah	IEE 802.15.1	LoRa WAN R1.0	IEEE 802.15.4
Bandas de Frecuencia (MHz)	*2.400 / 5.000	<1.000	2.400	433/868/915	868/915/2.400
Rango (m)	Entorno: - Interior: 30 - Exterior: 100	<1.000	50 a 150	Entornos: - Urbanos: 2.500 - Suburbanos: 15.000 - rural 45.000	10
Velocidad de transferencia (Kbps)	a: 54.000 máx. teórica / 20.000 de promedio b: 11.000 máx. teórica / 5.900 sobre TCP 7.100 sobre UDP g: 54.000 máx. teórica / 20.000 de promedio n: 600.000 máx. teórica	150-7.800	1.000	0,3 a 50	250
Tamaño de la dirección (bits)	48	48	48	32	16/64
Tipo de red	WLAN	WLAN	WLAN	WWAN	WPAN
Topología de la red	Estrella	Estrella	Estrella	Estrella	Estrella, malla y árbol
Pila de protocolos	Física/Enlace	Física/Enlace	Física a Aplicación	Física /Enlace	Red/ Aplicación
Organización de estandarización	IEEE	IEEE	Bluetooth SIG	LoRa Alliance	ZigBee Alliance

*NOTA: Dependiendo de la versión del estándar tiene una, otra o las dos.

Tabla 2 - Análisis comparativo de los protocolos de capa física y de enlace.[4][84][83]

5.3.2 Capa de aplicación

El protocolo de comunicación seleccionado es un elemento clave para el intercambio de mensajes entre el dispositivo y el sistema de almacenamiento.

Debe poseer múltiples características, como ser capaz de implementarse en dispositivos con recursos limitados, fiable, fácil de desplegarse, que asegure la confidencialidad de los datos al realizar las comunicaciones, debe estar disponible en todo momento o que debe tener un control de acceso [48][49].

Únicamente se describe las características del protocolo seleccionado, MQTT, pero se puede observar una comparativa de las opciones descartadas en la tabla 3.

MQTT

El protocolo MQTT (*Message Queing Telemetry Transport*) se basa en una arquitectura de mensajería asíncrona publicista/subscriptor (Pub/Sub), permitiéndole adoptar distintos tipos de distribuciones (uno a uno, muchos a muchos) para realizar intercambios de mensaje en tiempo

casi en tiempo real. Trabaja sobre TCP/IP, lo que asegura que los mensajes se reciban en el mismo orden y sin errores, y soporta comunicaciones Máquina-a-Máquina (M2M) [49][50].

Fue ideado para ser ligero, simple, abierto y fácil de implementar de forma que se pudiera usar en redes poco fiables y poco ancho de banda con dispositivos de recursos limitados de procesamiento y memoria, lo que lo hace ideal para aplicaciones IoT [20].

Destacan dos entidades o actores dentro de la red: el *broker* y el cliente (pueden ser múltiples clientes). El *broker* es un nodo central que se encarga de la recepción de los mensajes enviados por los clientes asignándoles un nombre de evento (*topic*) para después redireccionarlos a aquellos otros clientes que estén subscritos a dichos *topics.* Por otro lado, un cliente puede ser cualquier elemento que pueda interactuar con el *broker* para enviar (publicar) o recibir (subscribirse) mensajes [42]. Casi todas las plataformas importantes basadas en la Nube (*Cloud*) lo soportan [48].

Es posible establecer relaciones padre-hijo gracias a su estructura jerárquica de *topics.* Cada *topic* puede ser separado en varios niveles, como un sistema de carpetas. Se puede acceder a múltiples *topics* al mismo tiempo mediante el uso de comodines (# o +). Es fiable, es decir, garantiza la confiabilidad de sus transmisiones entre el *broker* y el cliente pub/subs. Dependiendo del nivel de calidad de servicio QoS (*Quality of Service*) [4].

Tiene mecanismos de notificación antes desconexiones inesperadas, como el último deseo y testamento (*last will and testament*). Soporta que el almacenamiento persistente de mensajes por parte del *broker*. Y la conexión TCP puede cifrarse mediante SSL/TSL (*Secure Sockets Layer / Transport Layer Security*) [20].

Su principal desventaja es que es un protocolo centralizado. A costa de aumentar la complejidad del sistema, es posible desplegar réplicas como solución a este problema [49].

La siguiente imagen es una representación del funcionamiento del protocolo MQTT [4]:

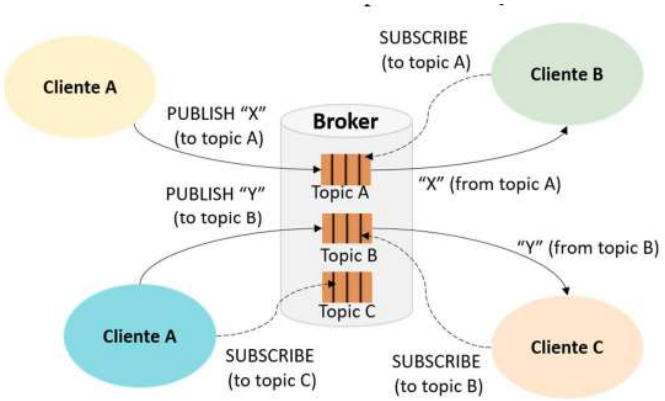


Ilustración 9 - Funcionamiento del protocolo MQTT [4].

5.3.2.1 Tabla comparativa – Protocolos capa de aplicación

La siguiente tabla es una comparativa entre los protocolos más usados con aplicaciones IoT:

Características	*MQTT*	*XMPP*	*AMQP*	*CoAP*	*HTTP*
Patrón de mensaje	Publicista/Suscriptor	Petición/Respuesta Publicista/Suscriptor	Petición/Respuesta Publicista/Suscriptor	Petición/Respuesta Publicista/Suscriptor (Observar)	Petición/Respuesta
Arquitectura	Árbol	Cliente/Servidor	Estrella	Árbol	Cliente/Servidor
Distribución de datos	1-a-N; N-a-N	1-a-1; N-a-N	1-a-N; N-a-N	1-a-N; N-a-1	1-a-1
Transporte	TCP (MQTT-S: UDP)	TCP	TCP	UDP	TCP
Capa de red	IPv4 o IPv6	IPv4 o IPv6	IPv4 o IPv6	IPv6	IPv4 o IPv6
Direccionamiento	Solo eventos *(topics)*	Identificación de Jabber	Cola, Evento (topic)/Clave de enrutamiento (routing key)	URI	URI
Mensajería Asíncrona	SI	SI	SI	SI	NO
QoS (Quality of Service)	3 niveles	NO	3 niveles	2 niveles	(Basado en TCP)
Filtrado	Evento (topic)	{user:to, from}, type, iq, presence packets	Cola	Identificador de recursos	Identificador de recursos
Soporte de transacciones	NO	NO	SI	NO	NO
Extensibilidad	NO	SI	SI	NO	NO
Comunicación M2M (Machine to Machine)	SI	SI	SI	SI	NO
Priorización de datos	NO	NO	SI	NO	NO
Caché de mensajes	SI	SI	SI	SI	SI
Alcance de la comunicación	D2C	D2C, C2C	D2D, D2C, C2C	D2D	D2C, C2C
Seguridad	SSL/TLS	SASL/TLS	SASL/TLS	DTLS, IPSec	HTTPS sobre TLS
Interoperabilidad	Fundamental	Estructural	Estructural	Semántica	Semántica
Tamaño de la cabecera	2 byte	Variable	8 byte	4 byte	No definido
Codificación	Binaria	Texto (datos binarios fuera de los límites)	Binaria	Binaria	Texto
Bajo consumo y pérdidas	Bueno	Aceptable	Bueno	Excelente	Aceptable
Formato del payload	Indeterminado	XML	Indeterminado	JSON, XML	Indeterminado
Tamaño máx. del mensaje	256MB	Indeterminado 64 KB (tamaño de la estrofa)	Indeterminado (RabbitMQ: 512 MB)	64 KB (UDP)	Indeterminado
Lenguajes de implementación [Broker/Servidor]	C, C++, Java, JavaScript, Erlang, Go, Lua, Python	C, C++, C#, Go, Erlang, Java, Python, Swift	C, C++, C#, Java, Python, Ruby	C, C++, C#, Java, Erlang, Go, Python, JavaScript, Ruby, Rust, Swift	C, C++, C#, PHP, Elixir/Erlang, Java, Sc Lisp, Lua, Node.js, Dart, Rust, Swift, Ru Go, Python, Haskell, Perl
Entidad gobernante	*OASIS*	IETF	OASIS	IETF	IETF, W3C

Tabla 3 - Análisis comparativo de los protocolos de la capa de aplicación [4][48].

5.4 Brokers

Como se especificará más adelante, el protocolo de comunicación escogido ha sido MQTT. Es por ello que el análisis de los *brokers* se centrará en aquellos que soporten MQTT, lo que no implica que únicamente soporten este protocolo.

Este apartado no entra en dealle sobre las características de todos los *brokers* disponibles, pero sí que muestra una tabla comparativa de algunos de los requisitos más importantes que deben de tener. Haciendo uso del trabajo realizado por D. Yacchirema [4] se ha tomado la decisión de utilizar Mosquitto como *broker* y considerar EMQ para futuras aplicaciones y desarrollos.

Los parámetros más importantes a comparar son aquellos que proporcionen información más relevante del rendimiento del *middleware*: fiabilidad, disponibilidad y seguridad [49]

5.4.1 Mosquitto

Eclipse Mosquitto se define como "*un broker de mensajes de código abierto (con licencia EPL/EDL) que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT*" [85]. Se trata de un *broker* ligero y que se puede utilizar un una gran variedad de dispositivos, desde ordenadores de placa única de baja potencia como Raspberry Pi, hasta servidores más complejos [85].

Es portable, fácil de instalar y está disponible para una amplia gama de plataformas. Además permite trabajar con MQTT simple, MQTT sobre TLS, MQTT sobre TLS (con certificado de cliente), MQTT sobre *WebSockets* y MQTT sobre *WebSockets* con TLS [85].

5.4.2 EMQ

Este software es uno de los últimos servidores/*broker* que han salido al mercado y se presenta como un buen candidato para el despliegue de múltiples dispositivos. Permite crear

réplicas del *broker*, para garantizar que habrá siempre uno funcionando en caso de que la principal sufra una caída.

Sin embargo, debido a su novedad no existen tantos trabajos en los que basarse para realizar este proyecto. Dadas las necesidades de este proyecto no se requiere de la característica extra que presenta frente a Mosquitto.

Teniendo en cuenta esto, y dada la complejidad que presenta su utilización frente a Mosquitto, no se ha seleccionado como solución final.

5.4.3 Tabla comparativa - Brokers

A continuación, se muestra una tabla comparativa de los *brokers* más usados:

Características	*Mosquitto*	*HiveMQ*	*RabbitMQ*	*EMQ*
Personalizado y económico	✓	×	✓	✓
Fiabilidad	✓	✓	✓	✓
Disponibilidad	×	✓	✓	✓
Control de acceso	✓	✓	✓	✓
Confidencialidad	✓	✓	✓	✓
QoS 0	✓	✓	✓	✓
QoS 1	✓	✓	✓	✓
QoS 2	✓	✓	×	✓
Autenticación	✓	✓	✓	✓
Puente	✓	×	×	✓
SSL	✓	✓	✓	✓
Topics dinámicos	✓	✓	✓	✓
Websockets	×	✓	?	✓

Clave: ✓ Soportado; × No soportado; ? Desconocido.

Tabla 4 - Comparativa entre brokers MQTT [20][86].

5.5 Servidores de almacenamiento de datos

Los datos recogidos mediante los dispositivos se tienen que almacenar en un lugar externo cuando se trabaja cantidades elevadas de información.

La decisión de que servidor de datos utilizar ha venido condicionada por las elecciones previas. La base de datos de series temporales seleccionada deberá ser compatible en este caso con MQTT.

Al igual que en el apartado anterior, no se entra en detalle sobre las características de todos los servidores de datos disponibles, pero sí que muestra una tabla comparativa de algunos de los requisitos más importantes que deben de tener. En este caso, tras observar el trabajo realizado por W. Nazco se ha tomado la decisión de utilizar InfluxDB como base de datos.

5.5.1 InfluxDB

InfluxDB es una plataforma para construir y operar aplicaciones de series temporales [53]. Sus principales usos están relacionados con la monitorización de servidores o datos recopilados con sensores IoT [52].

Aunque para este proyecto se ha seleccionado su versión gratuita, tiene una versión de pago denominada InfluxDB Enterprise que convierte cualquier instancia de InfluxData en un clúster listo para la producción que puede funcionar en cualquier lugar [53].

Su versión InfluxDB Cloud permite a los usuarios centrarse en la creación de software con una plataforma de series temporales sin servidor, fácil de usar y escalable, disponible en AWS, Azure y Google Cloud [53].

5.5.2 Tabla comparativa – Bases de datos

A continuación, se muestra una tabla comparativa de algunos de las bases de datos disponibles actualmente [52]:

Características	*InfluxDB*	*Graphite*	*OpenTSDB*	*Prometheus*
Versión inicial	2013	2006	2011	2015
Código Abierto	Si	Si	Si	Si
Basado en la nube	No	No	No	No
Lenguaje de implementación	Go	Python	Java	Go
S.O: de servidor	Linux	Linux	Linux	Linux
	OS X	Unix	Windows	OS X
APIs y otros métodos de acceso	HTTP API	HTTP API	HTTP API	RESTful
	JSON sobre UDP	sockets	Telnet API	HTTP/JSON API
Lenguajes de programación soportados	Net, Clojure, Erlang, Go, Haskell, Java, JavaScript, Lisp, Perl, PHP, Python, R, Ruby, Rust, Scala	JavaScript (Node.js), Python	Erlang, Go, Java, Python, R, Ruby	.Net, C++, Go, Haskell, Java, JavaScript (Node.js), Python, Ruby
Esquema de datos	Esquema libre	Si	Esquema libre	Si
Mecanografía	Datos numéricos y cadenas	Datos numéricos	Datos numéricos para métricas, cadenas para etiquetas	Datos numéricos únicamente
Scripts del lado del servidor	No	No	No	No
Desencadenantes	No	No	No	No
Soporte XML	No	No	No	No
Índices secundarios	No	No	No	No
SQL	SQL-like query language	No	No	No
Métodos de particionamientos	Sharding	Ninguno	Sharding	Sharding
Métodos de replicación	Factor de replicación seleccionable	Ninguno	Factor de replicación seleccionable	Si

Tabla 5 – Comparativa de bases de datos de series temporales [52] **.**

5.6 Aplicaciones

En este apartado no se ha realizado ninguna comparativa de programas de visualización debido a que no era objetivo de este proyecto. Únicamente se describirán aquellos programas seleccionados teniendo en cuenta su compatibilidad con las elecciones previas realizadas (protocolos, dispositivos, bases de datos, etc.) tras haber observado los resultados de otros proyectos [39][52][87].

5.6.1 Node-RED

Es un editor de flujo, donde mediante el uso de nodos permite simplificar la comunicación hardware, APIs y servicios de internet de forma rápida y sencilla mediante programación visual.

"Se ha consolidado como framework open-source para la gestión y transformación de datos en tiempo real en entornos de Industry 4.0., IoT, Marketing digital o sistemas de IA entre otros"[88].

Ilustración 10 - Dashboard de Node-RED [88].

Su principal motivo de uso es por que simplifica el uso de tecnologías que en un principio se podrían considerar como complejas.

Su estructura mínima se denomina nodo, y mediante el conexionado de ellos en la interfaz gráfica, se crean flujos o agrupaciones de nodos mediante los que se realizan tareas con una programación mínima.

Algunas de estas tareas pueden ser la subscripción o creación de un *topic*, envío de mensajes al correo o Twitter o la generación de documentos como Excel (.csv) o de texto plano (.txt).

5.6.2 Visualización de datos - Grafana

Grafana es una aplicación *open source* mediante la cual es posible generar gráficos interactivos (mediciones basadas en series de tiempo) o usar un sistema de alarmas, donde todo ello es configurable desde la interfaz gráfica del usuario.

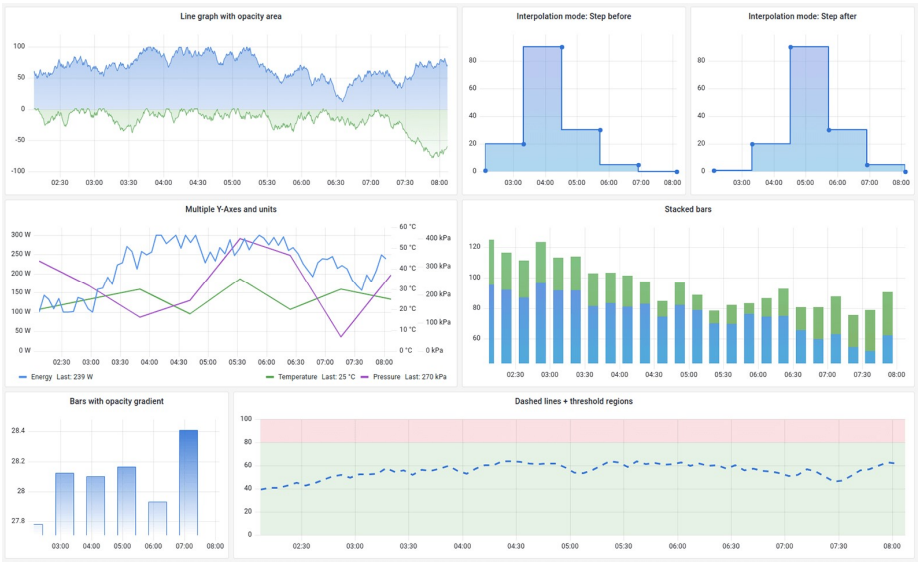


Ilustración 11 - Dashboard Grafana [65].

Permite, mediante el uso de *plugins*, conectar las fuentes de datos existentes a través de las API y presentan los datos en tiempo real sin necesidad de migrar o ingerir los datos [65]. Dentro de las muchas opciones disponibles, se encuentra la base de datos InfluxDB seleccionada para el desarrollo de este proyecto.

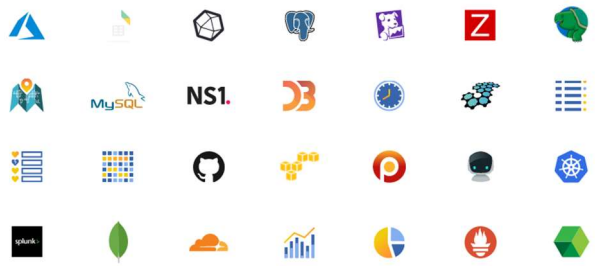


Ilustración 12 - Plugins Grafana [65].

Desde el punto de vista del desarrollado y administrador del sistema, presenta el inconveniente de que no se puede realizar la gestión y administración de la base de datos, únicamente visualiza los datos obtenidos de la base de datos. Viéndolo desde el punto de vista positivo, de esta forma es imposible comprometer los datos, permitiendo de esta forma conceder permisos a terceros mediante cuentas de usuario.

5.7 Selección final

Basándose en los requisitos descritos en los respectivos análisis de los elementos en los apartados anteriores, en este apartado se ha procedido a la selección de aquellos que conformarán la solución final de este proyecto.

Observando la Tabla 1 del apartado 5.1.3 y dado el alcance de este proyecto, el dispositivo **Arduino Nano IoT 33** (AN-33IoT) se presenta como una de las mejores opciones junto al dispositivo basado en el ESP32.

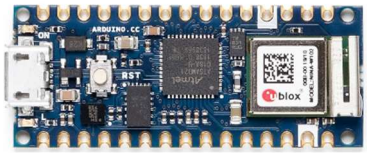


Ilustración 13 - Arduino Nano 33 IoT.[17]

Ambos presentan características presentadas como objetivo para este proyecto, como sus capacidades de conexión inalámbricas (WiFi y Bluetooth) y son de bajo coste (10 euros el AzDelivery ESP32 y 16 euros el AN-IoT).

El *framework* utilizado para la programación del dispositivo ha sido **Arduino IDE**. Esta plataforma permite su utilización de forma intuitiva y permite la incorporación de librerías externas en el caso de ser necesarias, como para el ESP32.

Ilustración 14 - Imagen de arranque de Arduino IDE [16].

Con el objetivo de reducir al mínimo el coste de la solución desarrollada, y teniendo en cuenta las capacidades del dispositivo AN-33IoT, se ha elegido el protocolo **Wi-Fi** como medio de

conectividad. No obstante, hay que tener en cuenta la existencia Wi-Fi HaLow para futuros desarrollos con dispositivos que permitan su uso.

El protocolo de comunicación seleccionado ha sido **MQTT** debido a que cumple con las características descritas en apartados anteriores. Además, dispone de una amplia selección de *middlewares* que se pueden desplegar y configurar fácilmente.

La elección del *broker* ha estado rivalizada entre Mosquitto y EMQ. Ambos cumplen todos los requisitos para ser seleccionados como elemento final. La única diferencia teórica es que EMQ garantiza la disponibilidad del sistema en todo momento mediante la creación de réplicas del *broker*. Aunque es una gran cualidad para su despliegue, implica un aumento de la complejidad de implementación de la solución final. Teniendo en cuenta el alcance de este proyecto, **Mosquitto** ha sido finalmente la opción seleccionada.

Ilustración 15 – Logo de Mosquitto .

Los señores utilizados son los preseleccionados por GICI. El **GY-MAX30100** para obtener los valores de oxígeno en sangre y el ritmo cardíaco; y el **Grove GSR** con un (LM324) para obtener la respuesta galvánica de la piel.

Es importante que el GY-MAX30100 sea la versión morada, puesto que la verde tiene un diseño defectuoso

Ilustración 16 – Sensor GY-MAX30100 [79]

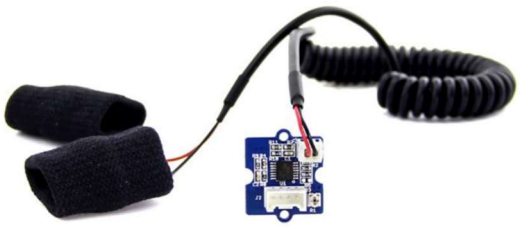


Ilustración 17 - Grove GSR [80].

El ordenador utilizado como HW donde se instala la máquina virtual es un portátil Asus ROG Strix G531GV con un procesador i7-9750H CPU 2.60GHz, 16 GB de memoria RAM, 1 TB de disco duro SSD, una tarjeta gráfica dedicada NVIDIA GeForce RTX 2060 con Windows 10 64 Bits como sistema operativo.[89]

Ilustración 18 - Portátil Asus ROG Strix G531Gv.[89]

Aunque en una primera instancia se seleccionó una Raspberry Pi como hardware donde instalar el servidor, posteriormente fue descartado como opción válida y sustituido por una máquina virtual de VMware v16.

Ilustración 19 - Máquina virtual VMware v16.[45]

Está máquina virtual, aunque es de pago, tiene una versión gratuita que se ha utilizado para la realización de este proyecto. El hecho que sea de pago no supone un problema para los objetivos de reducir el coste económico de la solución deseada dado que para aplicaciones desplegadas en el mundo real se utilizarían los ordenadores disponibles donde se instalaría el OS de Linux. Obviamente, en el caso de un despliegue desde cero, el coste del ordenador utilizado como hardware donde instalar el *broker* y servidor se tendrían que tener en cuenta en el cómputo final.

La máquina virtual tiene instalado un sistema operativo basado en Linux, concretamente Ubuntu 20.04.2 LTS, y se le han asignado 4 GB de memoria RAM, 2 procesadores, 40 GB de disco duro del portátil.

Ilustración 20 - Ubuntu 20.04.2 LTS [90]

No ha sido necesario la implementación de un servidor DHCP puesto que el número de nodos utilizados ha sido reducido. Queda descartado también la utilización de software de control como ESPHome o Home Automation por los mismos motivos. Sin embargo, son elementos a tener en cuenta para futuros desarrollos.

Se ha seleccionado **Node-RED** como herramienta auxiliar para la gestión de los datos debido a que facilita la implementación de la solución desarrollada y facilita la realización de otros objetivos como la generación de alertas.

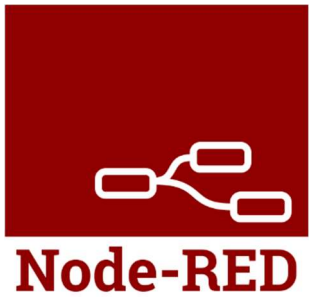


Ilustración 21 - Lodo de Node-RED [57].

Dentro de las múltiples bases de datos se ha seleccionado **IbfluxDB,** dado que posee las características necesarias para el desarrollo de este proyecto y puede implementarse fácilmente mediante el uso de Node-RED.

Ilustración 22 – Logo de InfluxDB [53].

Por último, aunque no es un requisito de este proyecto, para la visualización de datos se ha optado por **Grafana**, ya que es compatible con la base de datos InfluxDB.

Ilustración 23 – Logo de Grafana [65].

5.7.1 Resumen de la selección final

En este apartado se puede observar una tabla listando los elementos finales seleccionados y una figura a modo ilustrativo de dónde corresponden siguiendo la arquitectura por niveles establecida en el Capítulo 4.

Todo el despliegue se basa en el *Edge-Computing*, evitando el uso del *Cloud-Computing* debido a las instrucciones indicadas por el GICI, con la idea de tener un entorno más controlado.

Se destaca que todos los programas utilizados son *open source* o tiene versiones gratuitas con las características necesarias para la realización de este proyecto.

Elemento	***Nombre***	***Ubicación***
Dispositivo de bajo coste	Arduino Nano 33 IoT	Elemento físico
Framework	Arduino IDE v1.8.13	Portátil con WIN10
Sensores	GY-MAX30100 (morado) Grove Sensor (LM324)	Elementos físicos
Protocolo de comunicación	MQTT v3.1	Protocolo
Bróker	Mosquitto v1.6.9	VMware v16 con Ubuntu 20.04.2 LTS
Servidor de datos	InfluxDB v1.6.4	VMware v16 con Ubuntu 20.04.2 LTS
Gestión de datos	Node-RED v14.17.2	VMware v16 con Ubuntu 20.04.2 LTS
Visualización de datos	Grafana v8.0.6	VMware v16 con Ubuntu 20.04.2 LTS
Otros	Portátil Asus-ROG Strix G5321Gv con WIN10	Elemento físico

Ilustración 24 - Elementos de la solución final.

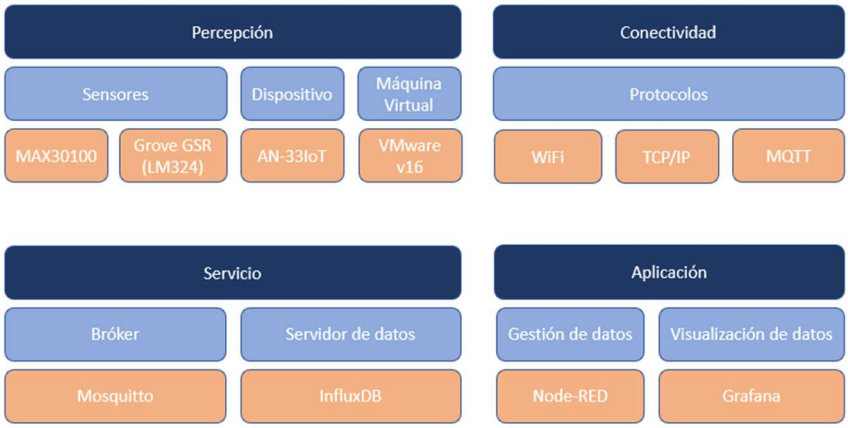


Ilustración 25 – Elementos de la solución final clasificados en sus niveles.

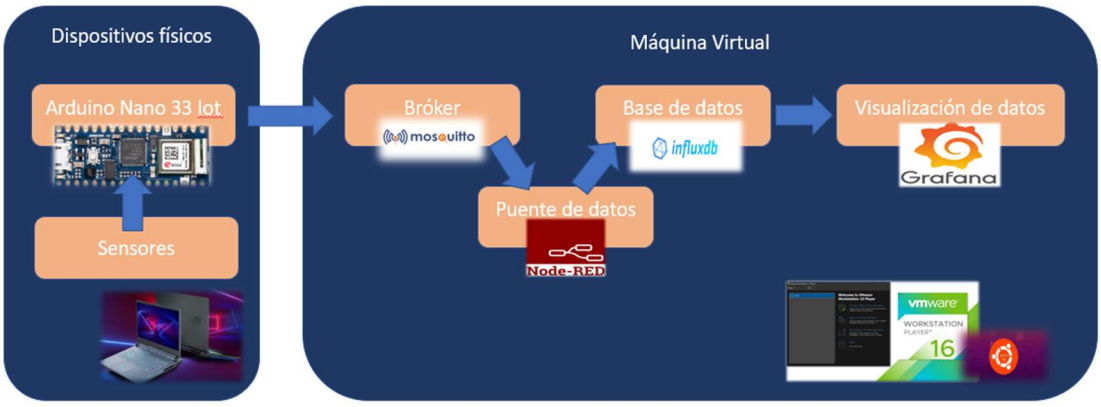


Ilustración 26 - Esquema solución final.

CAPÍTULO 6

DESARROLLO DEL TRABAJO

6 DESARROLLO DEL TRABAJO

Este capítulo está dividido en apartados en función de las tareas acometidas a lo largo del desarrollo de este TFM a partir de los materiales y herramientas seleccionadas en el capítulo anterior, incluyendo algunas pruebas y resultados parciales, para validar y valorar los hitos alcanzados.

Aunque en el capítulo anterior se describió los elementos finales de la solución desarrollada, tal y como se detallará en los siguientes apartados, el escenario de trabajo ha ido variando durante la realización de las pruebas de forma que se pudieran realizar las pruebas para cumplir los objetivos marcados para este proyecto.

Un análisis más detallado las conclusiones de cada apartado se mostrará en el siguiente capítulo, "análisis de resultados". Por otro lado, toda la implementación de los códigos utilizados se encontrará en los anexos.

6.1 Preparación del entorno de trabajo

La descripción de la instalación de los softwares necesarios para la solución implementada se detalla en el aptado de anexos. En ella se indican los posibles errores que pueden surgir y los pasos claves a realizar para solventarlos.

6.2 Pruebas iniciales realizadas. Escenario inicial.

En este apartado se realizarán unas pruebas para comprobar el funcionamiento del protocolo de comunicación WiFi y de los sensores. Para ello se realizarán pruebas independientes de algunas de las distintas partes en las que se ha estructurado el código final.

En el escenario inicial se parte de las soluciones propuestas por GICI, comprobando su estado de funcionamiento. Posteriormente se realizan pruebas de código propio, permitiendo así realizar comparaciones y realizar modificaciones en caso oportuno.

6.2.1 Soluciones propuestas por GICI

El GICI [66] proporcionó un código (*GICI.ino*) inicial e inacabado, con el que empezar a trabajar. A continuación, se muestra su diagrama de flujo a modo de representación gráfica:

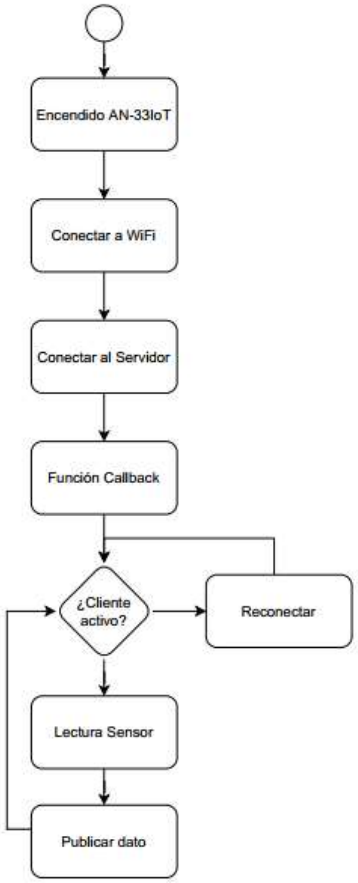


Ilustración 27 – Flujograma del código proporcionado por GICI (*GICI.ino*).

En el *setup* o función de inicialización, se realiza la conexión al WiFi, al servidor (*broker*) y se ejecuta la función *callback*. Posteriormente se ejecuta el *loop* o función bucle, donde se comprueba si la conexión entre el dispositivo y el servidor se mantiene activa.

En el caso de que esté activa, se produce la lectura de la bioseñal con el sensor y posteriormente se publica el valor en el servidor. En el caso de que la respuesta al estado del cliente sea que esté inactiva, se volverá a intentar establecer la conexión pasado un tiempo determinado.

Durante su ejecución se observó que había una serie de acciones que no tenían sentido teniendo en cuenta los objetivos de este proyecto.

Em primer lugar y muy importante, se observó que cada vez que el dispositivo obtiene una respuesta negativa al estado de conexión del cliente con el *broker* se le asigna un nuevo identificador generado aleatoriamente.

Esto supone un grave problema dado que el objetivo de este proyecto se basa en poder utilizar esta solución en el campo de la medicina y salud, donde un dispositivo representará a un individuo, almacenando sus bioseñales. Cambiar el valor del identificador sería como generar un nuevo perfil de usuario y guardaría los nuevos datos como si fuera otro individuo.

Por otro lado, la función *callback* devuelve el valor del *topic* seleccionado, haciendo desempeñar el rol de cliente subscriptor al dispositivo AN-33IoT. Aunque esta función se podría utilizar para futuros desarrollos, no tiene utilidad para el objetivo que aquí se propone.

6.2.2 Primeras conexiones a la red wifi

Establecer una arquitectura IoT implica implementar algún tipo de conexión inalámbrica, en este caso el dispositivo AN33-IoT se conectará a través del protocolo WiFi. Por lo tanto, una de las funciones que deberá implementar el código de la solución será la del establecimiento de conexión WiFi.

Tras incluir las librerías necesarias (*WiFiNINA.h*), es necesario especificar la red (SSID) con la que se desea establecer la conexión y su contraseña (*password*).

La primera acción a realizar será comprobar que la placa o dispositivo tiene el módulo necesario para realizar la comunicación vía WiFi. En el caso de no lo tenga, no ejecutara ninguna acción más. Se mostrará un mensaje por el terminal indicando el fallo y se requerirá apagar el dispositivo y realizar los ajustes necesarios en la red o el código.

En el caso de que la conexión se establezca se realizará un escaneado de las redes disponibles mediante una función, en este caso denominada *scanWifi()*. Mediante esta función se puede conocer los nombres de las redes disponibles, las fuerzas de las redes (dBm) y el tipo de encriptación que utilizan.

Por último, se procederá al intento reiterado de la conexión con la red indicada al inicio siempre y cuando el número de intentos realizados no supere al establecido, 15 intentos en este caso. Si se establece la conexión, se muestra el nombre de la red y su dirección IP.

En el caso de superar el límite de intentos, se mostrará por pantalla un mensaje de fallo en el establecimiento de la conexión, para posteriormente finalizar su ejecución.

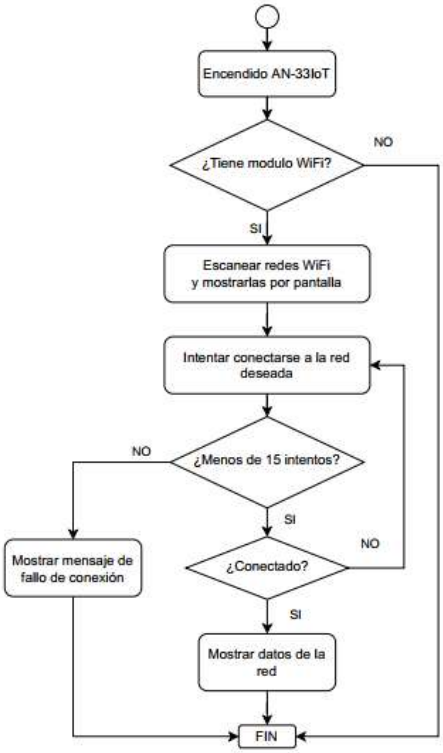


Ilustración 28 - Flujograma de la prueba de conexión WiFi (*WifiScan.ino*)

La realización de este código (*WifiScan.ino*) [32] ha supuesto una mejora respecto al proporcionado por GICI. Ahora se realiza una comprobación previa de si el dispositivo dispone de un módulo capaz de realizar la conexión y se ha establecido un número máximo de intentos

antes de finalizar su ejecución. Ya no se queda intentando establecer la conexión de forma infinita.

6.2.3 Primeras lecturas con los sensores

El objetivo de este proyecto es la adquisición de variables fisiológicas y para ello es necesario comprobar el correcto funcionamiento de los sensores que las capturan. En este caso se ha utilizado un sensor GY-MAX30100 para obtener los valores del pulso y el nivel de oxígeno en sangre y el Grove GSR con un (LM324) para obtener la respuesta galvánica de la piel.

Las pruebas para los sensores que a continuación se muestran se han realizado de forma independiente, es decir, una prueba para el sensor de pulso y oxigenación, y otra para el sensor GSR.

Pulso y oxigenación

Lo primero a tener en cuenta para este código (*Oximetro_y_pulsimetro.ino*) es que para el uso de este sensor es necesario incluir la librería "*MAX30100_PulseOximeter.h*". Esta librería permite instanciar un objeto de la clase pox (pulseoximeter) de diferentes formas:

- Muestras filtradas y umbral de detección de latidos.
- Valores muestreados procedentes del sensor, sin procesamiento.
- Valores muestreados después del filtro de eliminación de CC.

Acto seguido se establece la corriente con la que funcionara el led infrarrojo (IR) y se registra la función *callback* para la detección del latido.

En la función *callback* se implementa lo que se quiere que pase cuando se detecte un latido, en este caso, que ponga "Beat!" (¡Latido!). Su contenido se mostrará por pantalla durante la actualización de parámetros independientemente de si ha pasado el lapso de tiempo establecido.

Por último, haciendo uso de la función *millis()*, se controla el periodo de muestreo y monitorización de los valores obtenidos mediante las funciones propias de la clase pox para la obtención del ritmo cardiaco (*pox.getHeartRate()*)y la oxigenación en sangre (*pox.getSpO2()*).

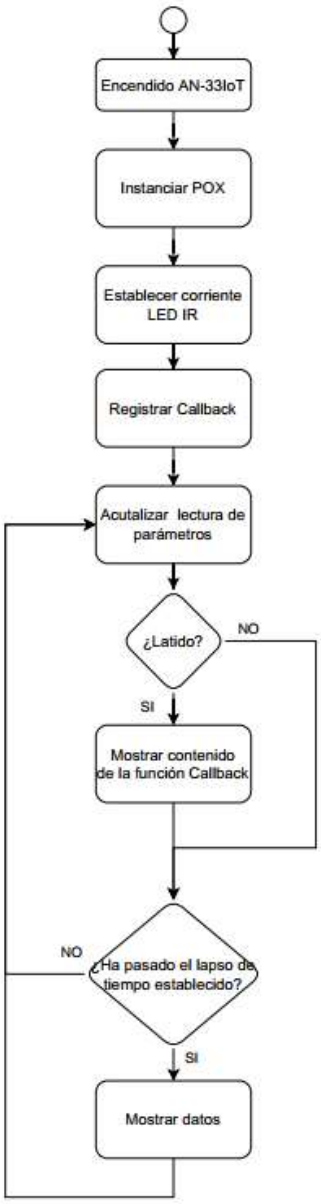


Ilustración 29 – Flujograma de la lectura del sensor MAX30100 (*Oximetro_y_pulsimetro.ino*).

A continuación, se muestran los resultados obtenidos y visualizados en el monitor serie de Arduino IDE y el esquema de conexionado utilizado. NOTA: el esquema no está realizado a escala.

```
Heart rate:65.43bpm / SpO2:98%
Beat!
Heart rate:67.14bpm / SpO2:98%
Beat!
Heart rate:65.29bpm / SpO2:98%
Beat!
Heart rate:64.57bpm / SpO2:98%
Beat!
Heart rate:61.26bpm / SpO2:98%
```

Ilustración 30 – Monitorización del ritmo cardíaco y oxigenación en sangre.

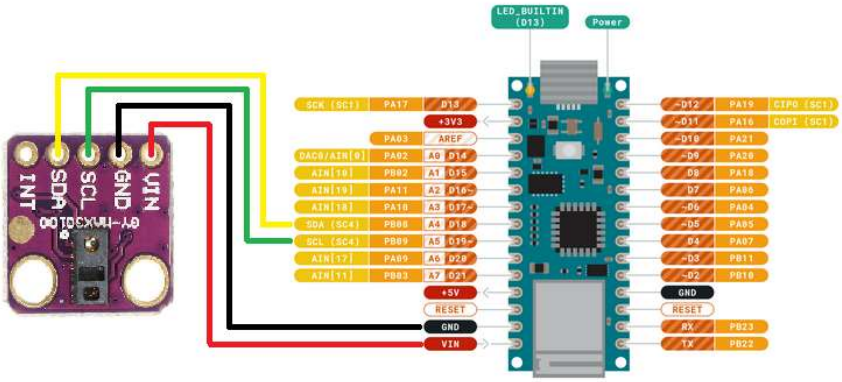


Ilustración 31 - Esquema de conexión del sensor MAX30100 [17][79]**.**

GSR

La implementación del código necesario (*GSR.ino*) para la captura de la respuesta galvánica de la piel mediante el Grove Sensor no requiere de ninguna librería adicional.

Únicamente ha sido necesario identificar el pin del AN-33IoT desde el que se realiza la lectura analógica y establecer el número de medidas que se desea capturar para posteriormente realizar la media y mostrarla por pantalla.

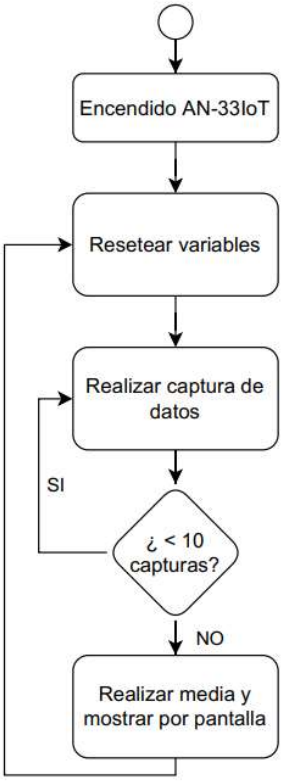


Ilustración 32 - Flujograma del sensor LM324 (GSR.ino).

A continuación, se muestran los resultados obtenidos y visualizados por la herramienta para hacer gráficos en serie de Arduino IDE.

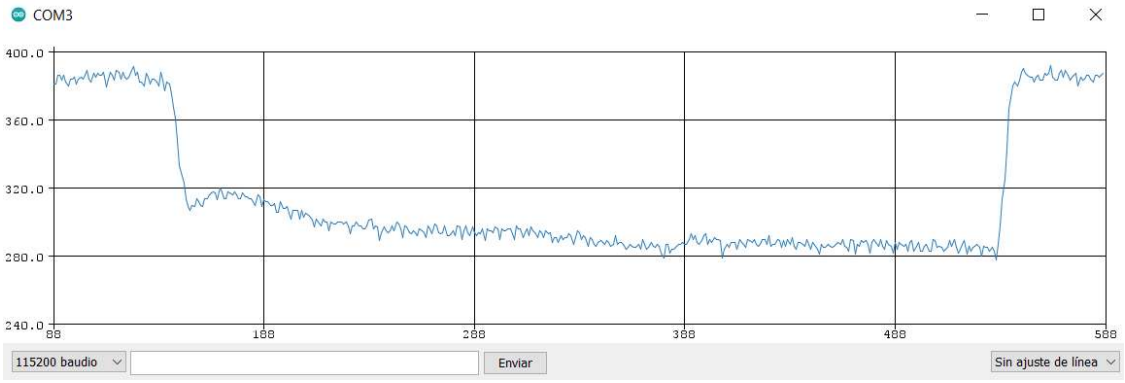


Ilustración 33 - Monitorización de la respuesta galvánica de la piel (GSR).

También se muestra el esquema no escalado de conexionado utilizado. La vista del sensor utilizada es la posterior para facilita la identificación del patillaje.

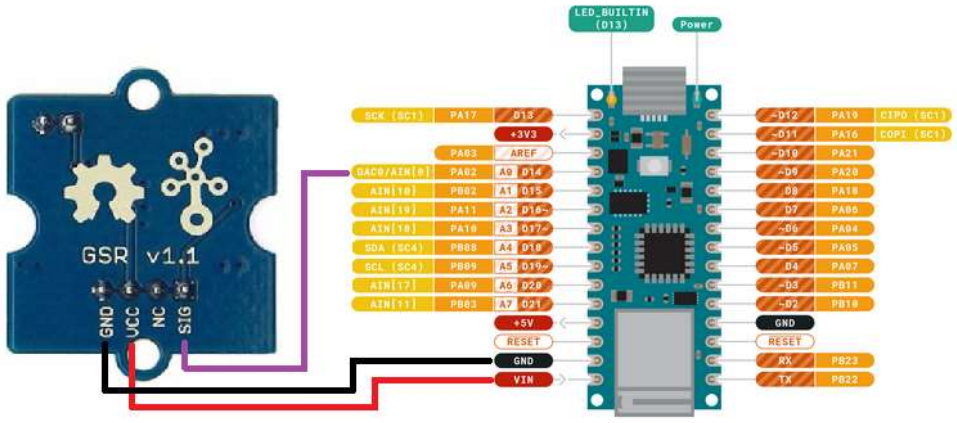


Ilustración 34 - Esquema de conexión del sensor LM324 (Grove GSR) [17][80].

Conclusiones – Pruebas iniciales

En estos códigos se realiza únicamente la lectura de los datos dejando postergando el envío de la información al *broker* para otros códigos.

Se han comprobado que ambos sensores funcionan correctamente.

Es posible establecer un método para controlar la frecuencia de muestreo.

6.3 Objetivo principal

En este apartado se realizarán unas pruebas para comprobar el funcionamiento del entorno final, desde la comprobación ente el dispositivo al *broker*, del *broker* a la base de datos pasando por el puente de datos (Node-RED) y finalmente de la base de datos al *software* de visualización (Grafana).

6.3.1 Primeros resultados – conexión con el broker

Antes de establecer la conexión con el *broker* instalado en la máquina virtual se realizó la comprobación del correcto funcionamiento con el servidor/*broker* MQTT que facilita Eclipse Mosquitto para realizar pruebas (*test.mosquitto.org*) [91].

Después se ha utilizado MQTTLens [92] para la visualización de los *topics* y sus datos. Es una aplicación de Google Chrome, que se conecta al *broker* MQTT y que es capaz de suscribirse y publicar en temas MQTT .

Publicación

Este código (*WiFiSimpleSender.ino*) [93] requiere de la instalación de la librería correspondiente para crear un cliente MQTT (*ArduinoMqttClient.h*) además de la otra correspondientes a la conexión WiFi descrita anteriormente. En este caso no se utiliza ningún sensor, por lo que no es necesario incorporar ninguna librería más.

Primero se establecen los parámetros de conexión WiFi y se instancian los clientes WiFi y MQTT. Después se establecen los parámetros del *broker* como su dirección IP y su puerto de enlace. Dependiendo de si queremos aplicar o no seguridad SSL/TSL la instanciación del cliente WiFi o el puerto de enlace del *broker* serán diferentes.

Una vez establecida la conexión WiFi se procederá a realizar la conexión con el *broker.* En el caso de que no se estableciera, se mostrará por pantalla un mensaje de error y su código de identificación para saber a qué se debe.

Si la conexión con el servidor/*broker* resulta exitosa se ejecutará en bucle una función que mantiene viva la conexión con el servidor (*poll()*) y en el caso de que se supere el intervalo de tiempo especificado se realizará la publicación del *topic* establecido. Después de cada publicación, se mostrará por pantalla el contenido del *topic* y se aumentará el contador que sirve para que el programador tenga un control visual del número de mensajes enviados.

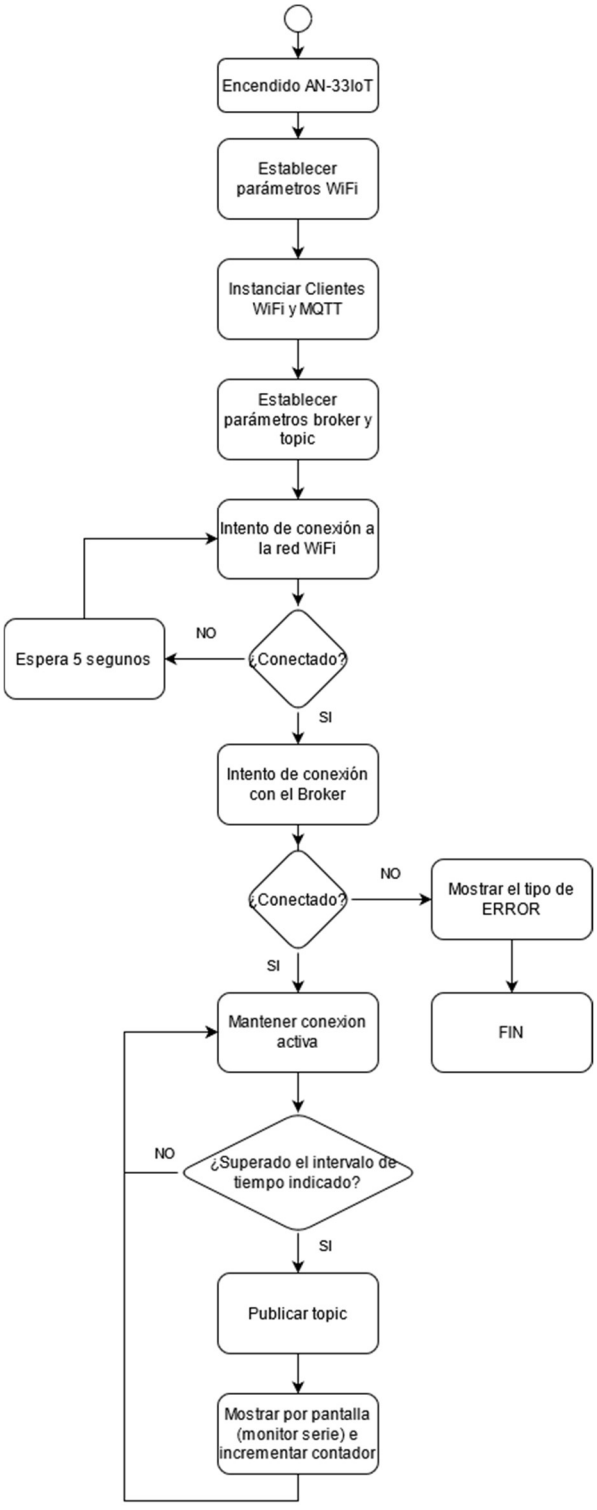


Ilustración 35 - Flujograma de publicación (WifiSimpleSender.ino).

A continuación, se muestran los resultados obtenidos y visualizados por el monitor serie de Arduino IDE. Además, también se muestran la información visualizada mediante la aplicación MQTTLens.

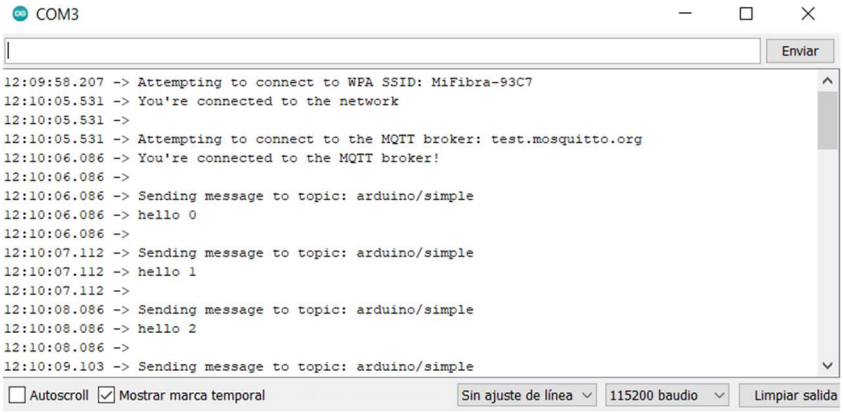


Ilustración 36 – Visualización de la ejecución del código de publicación.

En la ilustración superior se observa cómo hay mensajes que indican la confirmación de las conexiones con la red WiFi y el *broker*. También se ve la notificación de que se está publicando un mensaje, el nombre del *topic* donde se publica y el contenido del *topic*.

En la siguiente ilustración se observan los tres últimos mensajes enviados al *broker* con el *topic* "*arduino/simple*".

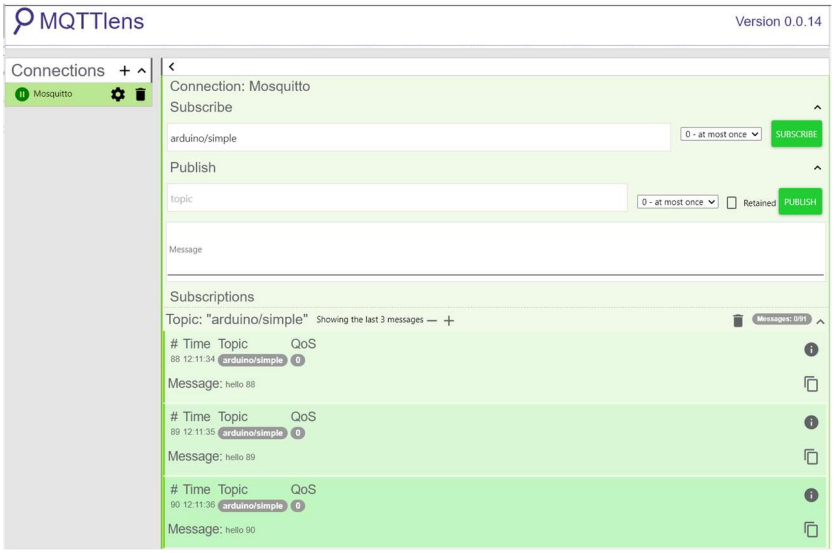


Ilustración 37 - Visualización de datos con MQTTLens.

Subscripción

Aunque no es necesario para este proyecto, también se ha realizado una prueba para comprobar si era posible subscribirse a un *topic*.

Este código (*WiFiSimpleReciver.ino*) [93], al igual que el anterior, requiere de la instalación de la librería correspondiente para crear un cliente MQTT además de la correspondientes a la conexión WiFi. Al no utilizar ningún sensor no es necesario incorporar ninguna librería más.

Al igual que con la publicación, primero se establecen los parámetros de conexión WiFi y se instancian los clientes WiFi y MQTT. Después se establecen los parámetros del *broker* como su dirección IP y su puerto de enlace. Se indica también el *topic* al que se quiere subscribir (*TFM/Sergio*).

Una vez establecida la conexión WiFi se procederá a realizar la conexión con el *broker*. En el caso de que no se estableciera, se mostrará por pantalla un mensaje de error y su código de identificación para saber a qué se debe.

Si la conexión con el servidor/*broker* resulta exitosa se realizará la subscripción al *topic* indicado previamente y se ejecutará en bucle una función que comprueba si hay algún mensaje nuevo publicado.

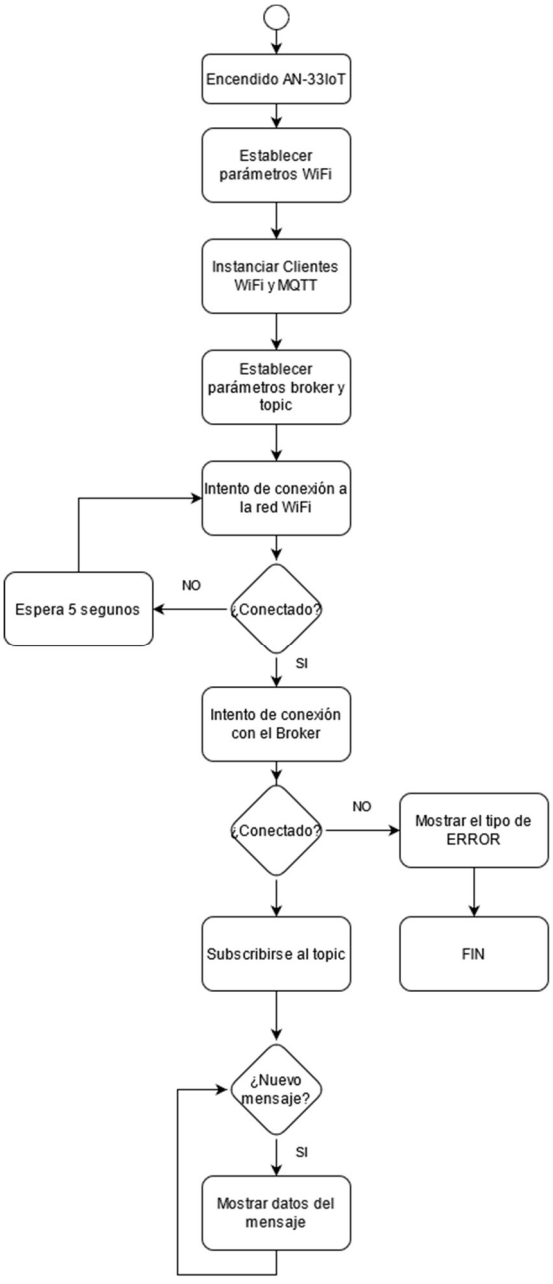


Ilustración 38 – Flujograma de subscripción (WiFiSimpleReciver.ino)

A continuación, se muestran los resultados obtenidos y visualizados por el monitor serie de Arduino IDE. Además, también se muestran la información visualizada mediante la aplicación MQTTLens.

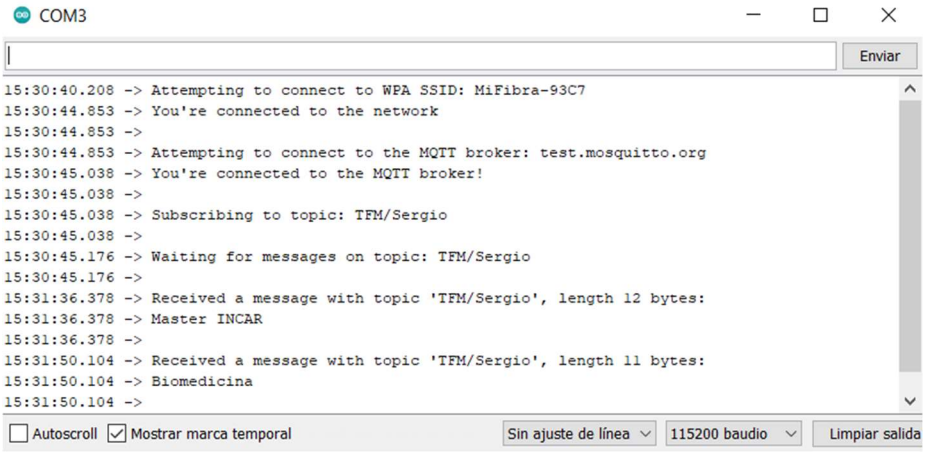


Ilustración 39 - Visualización de la ejecución del código de subscripción.

En la ilustración superior se observa cómo hay mensajes que indican la confirmación de las conexiones con la red WiFi y el *broker*. También se puede ver el *topic* al que se ha subscrito y posteriormente, cada vez que se publica un nuevo mensaje, el nombre del *topic* al que se ha subscrito, el tamaño del mensaje en *bytes* y su contenido.

La siguiente ilustración muestra como el último mensaje publicado en el *topic* "*TFM/Sergio*", el cual contenía "*Biomedicina*", coindice con el visualizado en el monitor serie de Arduino.

Ilustración 40 - Visualización de datos publicados con MQTTLens.

Conclusiones – Conexión con el broker

Es posible introducir seguridad en la comunicación mediante SSL/TSL entre el dispositivo y el *broker*.

Existen aplicaciones que permiten visualizar los *topics* como MQTTLens.

6.3.2 Primer escenario de trabajo con los elementos de la solución final

Hasta este punto se ha logrado comunicar de forma exitosa vía Wifi el dispositivo IoT con un servidor/*broker* de forma que fuera posible publicar datos a un *topic* o subscribirse a él para recibirlo, todo ello realizado en el portátil.

A partir de aquí, se realiza instalación del *broker* (Mosquitto), el puente de datos (Node-RED), la base de datos (InfluxDB) y la herramienta software de visualización (Grafana) en un dispositivo diferente al portátil.

En la introducción de este capítulo ya se dijo que el escenario de trabajo ha ido variando durante el desarrollo de la solución final. En un principio se utilizó Raspberry Pi como elemento físico donde instalar el *broker*, el servidor de datos y el software complementario (Node-RED y Grafana). Pero, tras observar la lentitud con la que funcionaba la RaspberryPi, se optó por utilizar una máquina virtual (descrita en la solución final en el capítulo 5) que actúa como si fuera otro portátil más en la red.

Para este primer escenario únicamente se utilizará el sensor GSR.

Primera prueba – Node-RED en RaspberryPi

Haciendo uso del código de publicar datos (*WiFiSimpleSender.ino*) y modificando las variables oportunas, en concreto la dirección IP del *broker,* se ha realizado la comprobación del funcionamiento de Node-RED.

Las primeras pruebas realizadas en la RaspberryPi mostraban como era posible guardar los datos enviados al *broker* en diversos formatos mediante la utilización de nodos.

El *broker* utilizado es el proporcionado por Eclipse Mosquitto para hacer pruebas "*test.mosquitto.org*".

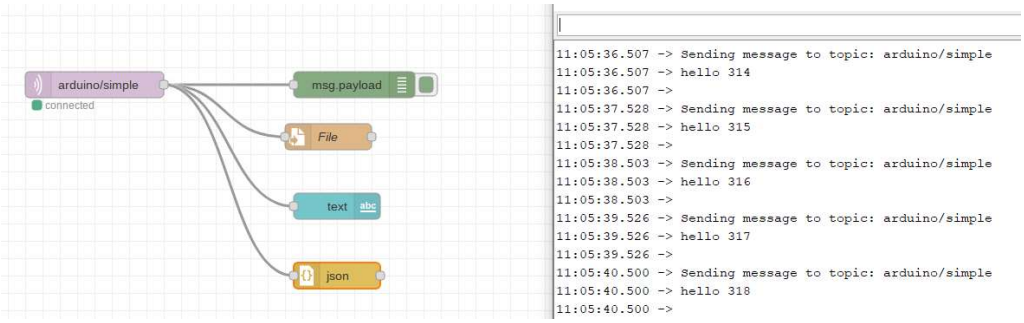


Ilustración 41 - Flujo de trabajo en Node-RED (izq.) y monitor serie Arduino (dcha.)

Ilustración 42 - Ficheros creados en RaspberryPi.

Segunda prueba – Node-RED en máquina virtual

Para esta prueba se generó un código nuevo (*TFM_v1.ino*) que incluía el envío de dos *topics*, uno para el valor GSR y otro el tiempo que tardaba en recopilar los datos. Al igual que con la prueba de la Raspberry Pi , se utiliza el *broker* de Eclipse Mosquitto para hacer pruebas (*test.mosquitto.org*).

Sin embargo, dónde se produce cambios interesantes dignos de mencionar es en su segunda versión (*TFM_v2.ino*). En esta nueva versión, que toma algunas ideas del proyecto de

SuperHouse Automation [94], se utiliza el *broker* Mosquitto instalado en la máquina virtual, por lo que se le ha podido incorporar un mecanismo de seguridad usuario/contraseña para hacer uso del *broker* que deberá de ser tenido en cuenta para acceder a él tanto desde el AN-33IoT como desde Node-RED.

Además, incluye un nuevo formato de *topic, JSON,* con en el que se puede combinar ambos *topics* de la primera versión y el cual permite una mejor estructuración de los datos para que sea más fácil procesarlos con Node-RED. También se han introducido variables para controlar cuánto tarda en realizar diversas acciones.

En las dos versiones se toma como referencia la estructura del flujo de diagrama de publicación de un *topic* descrita anteriormente.

En primer lugar, se definen los parámetros de configuración en un archivo aparte. Lo que antes se llamaba "*arduino_secrets.h*" pasa a llamarse "*configuration.h*". Este archivo contiene las claves de usuario/contraseña de la red WiFi y del *broker* además de la dirección IP y puerto del *broker*. También contiene otros parámetros extra como la velocidad de la consola serie USB o las definiciones de algunas variables *(#define name var*) que permiten controlar el tipo de mensaje que se enviará (variables por separado o en formato JSON).

Volviendo al *sketch* principal (*TFM_v2.ino*), y tras la incorporación de las librerías necesarias, se definen las variables globales. Algunas de las variables nuevas permiten controlar parámetros importantes como el tamaño del paquete de datos que se va a enviar (*g_topicSize*).

Se instancian los clientes WiFi y MQTT y a modo de tener una mejor organización y control del código, se declaran las funciones que se van a utilizar. Estas son:

- **mqttCallback():** Esta función se invoca cuando se recibe un mensaje MQTT. No es importante para el alcance de este proyecto porque no recibimos comandos vía MQTT. Pero puede ser de utilidad para desarrollos futuros para que el dispositivo actúe sobre los comandos (instrucciones o alertas) que le envíes.

- **initWiFi()**: Función basada en los desarrollos previos para realizar la conexión con la red WiFi. En el caso de no conectarse muestra mensaje de error.

- **reconnectMqtt():** Esta función se utiliza para reconectar con el *broker* MQTT, y publicar una notificación en el nuevo *topic* de estado denominado "*status_topic*". En caso de no realizarse la conexión muestra un mensaje con la identificación del tipo de error.

- **updateReadings():** Aquí se actualiza la lectura realizada por los sensores, muestra por pantalla (monitor serie) cuanto ha tardado en realizarlas, crea el paquete con los datos recopilados y los envía al mismo tiempo que se visualiza la información por el monitor serie. Para realizar estas acciones hace las llamadas a las funciones *showTimeNeeded(), reportToMqtt()* y *reportToSerial().* Finaliza vaciando el paquete.

- **reportToMqtt():** En esta función se informar de los últimos valores al *broker* MQTT. El tipo de *topic* enviado, valores por separado o JSON, depende de los parámetros configurados en el archivo auxiliar "*configuration.h*".

- **reportToSerial():** Mediante esta función se muestra por pantalla (terminal serie de Arduino IDE) información de las variables utilizadas. Como la identificación del paciente o usuario, el valor de los *topics* o el peso en bytes del *topic*. No es necesaria para el uso de la solución desarrollada, pero aporta información valiosa al programador.

- **NewclientID()**: Genera la identificación del nuevo cliente MQTT con las iniciales del usuario o paciente y lo muestra por pantalla. Para este proyecto no tiene mucha utilidad, los valores se introducen previamente a mano en el código, no obstante, podría utilizarse para desarrollos futuros.

- **showTimeNeeded():** Función auxiliar utilizada para saber cuánto tardad en realizar una acción. Calcula la diferencia entre dos variables que han hecho uso de la función *micros()*. Esta es la versión con más precisión de la función *millis()* utilizada en pruebas anteriores que "devuelve el número de milisegundos (microsegundos para la función *micros()*) desde que la placa Arduino empezó a ejecutar, luego de un reinicio o el encendido"[95] . No es necesaria para el uso de la solución desarrollada, pero aporta información valiosa al programador.

Ya en el *setup(),* primero se inicializa la conexión con el puerto USB y se establece la identificación del paciente/usuario. Después se configuran los *topics* para publicar las lecturas de los sensores insertando el ID único, el resultado tiene la forma "*arduino_1/GSR/valorID*". Por último, se configura el cliente MQTT los datos del archivo de configuración. Aquí también se realiza la modificación de tamaño del paquete que puede enviar por MQTT que por defecto es de 128 bytes.

A partir de aquí se realiza una ejecución en bucle. Si la conexión con la red WiFi es correcta, se comprueba el estado de conexión del cliente con el *broker*. En caso de desconexión, se intentará restablecer mediante la función *reconnectMqtt()*.

Independientemente del estado de la conexión WiFi y de la conexión con el *broker*, se ejecutará una función para procesar cualquier mensaje MQTT pendiente (*mqqttClient.loop()*) y se actualizarán las lecturas de los sensores mediante la función *updateReadings().*

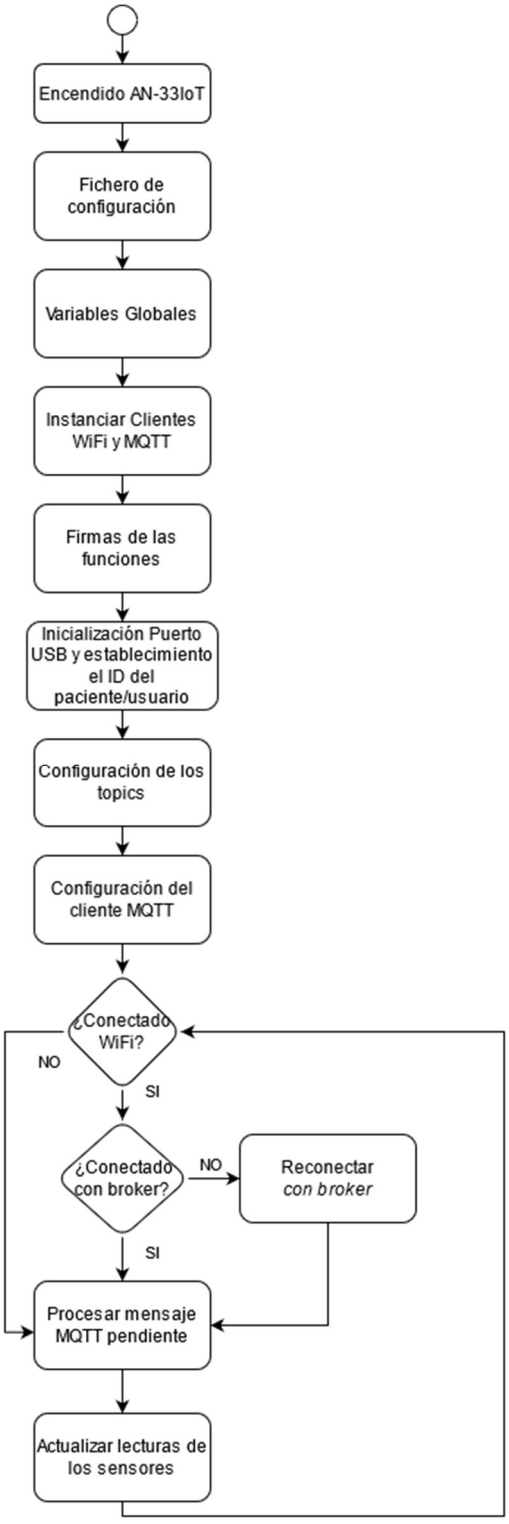


Ilustración 43 - Flujograma del código TFM_v2.ino

A continuación, se muestran los resultados obtenidos durante el proceso:

Las dos siguientes imágenes muestran desde el terminal de la máquina virtual los *topics* enviados al *broker* Mosquitto. Se han enviado cuatro *topics,* tres por separado y un JSON que contiene la información de los otros tres.

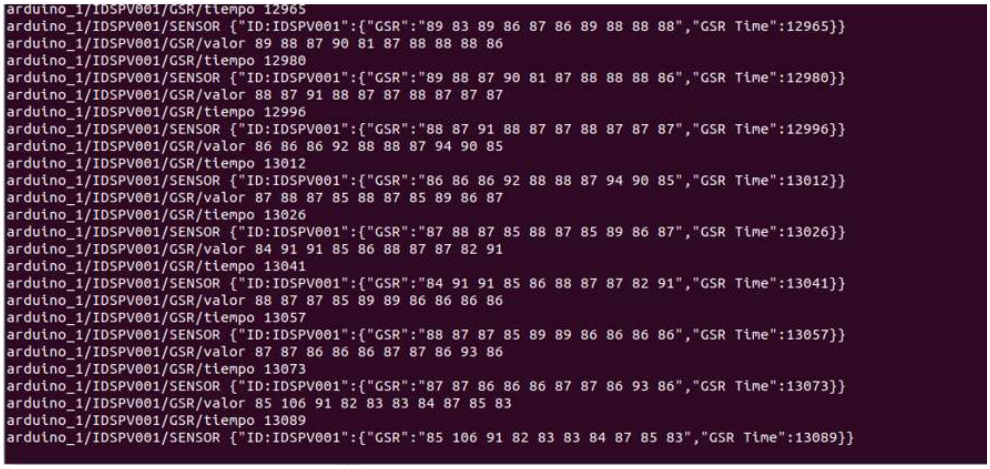


Ilustración 44 - Broker Mosquitto en máquina virtual.

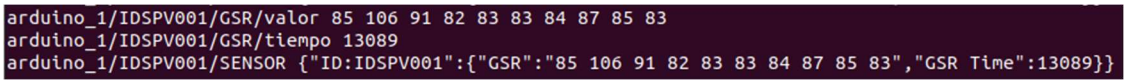


Ilustración 45 - Vista detallada de los topics enviados al broker Mosquitto en la máquina virtual.

Como se puede observar, la información contenida en los *topics* coincide.

Tercera prueba – Node-RED, InfluxDB y Grafana en máquina virtual

Una vez confirmada la conexión se ha procedido a almacenar los datos en la base de datos haciendo uso de Node-RED como herramienta puente entre el *broker* Mosquitto e InfluxDB.

El código utilizado es el mismo que en la prueba anterior (TFM_v2.ino). Las modificaciones se han realizado en Node-RED. Se han tenido que instalar los nodos oportunos para trabajar con IndluxDB.

A continuación, se muestra el flujo realizado en Node-RED:

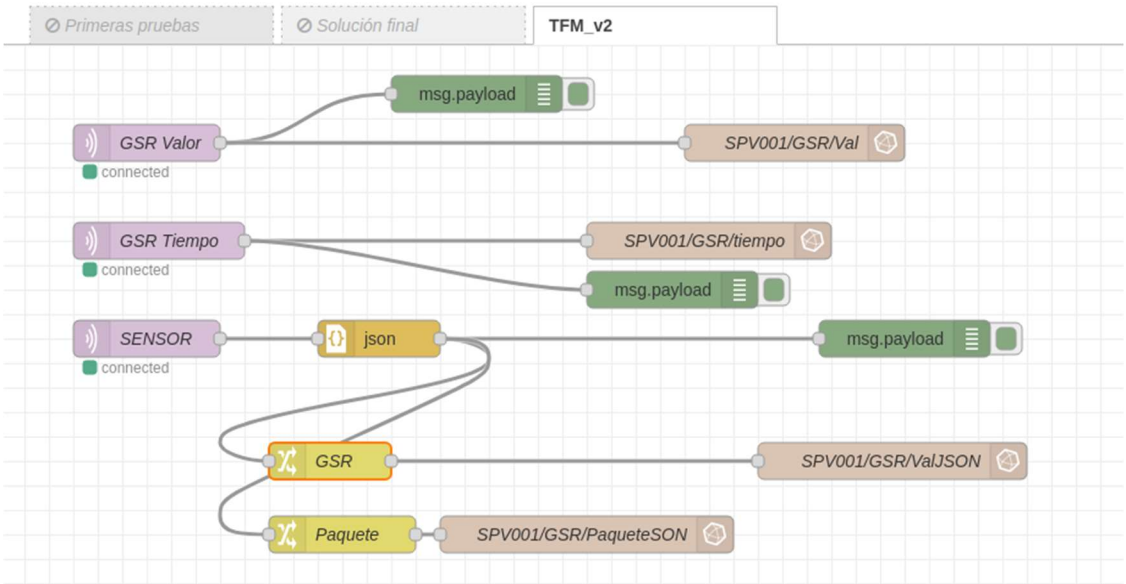


Ilustración 46 - Flujo realizado en Node-RED. Tercera prueba.

Mediante este flujo es posible extraer la información de los *topics* individuales y de *topic* JSON, para posteriormente guardarlos en la base de datos InfluxDB.

Desde el terminal de Linux se puede ejecutar InfluxDB y ver su contenido. Como se puede observar en la siguiente imagen, se han almacenado el contenido de los *topics* utilizados.

```
> select * from /.*/  limit 1
name: arduino_1/IDSPV001/GSR/tiempo
time                value
----                -----
1626727895813782810 7739

name: arduino_1/IDSPV001/GSR/valor
time                value
----                -----
1626727895813790921 89 88 90 87 88 88 87 87 91 87

name: arduino_1/IDSPV001/GSR/valorJSON
time                value
----                -----
1626727895818350209 89 88 90 87 88 88 87 87 91 87
```

Ilustración 47 - Datos almacenados en InfluxDB.

Luego mediante Grafana es posible visualizar estos datos en tiempo real. La siguiente imagen muestra la monitorización del *topic* "*arduino_1/IDSPV001/GSR/tiempo*" cuyo valor en el código se genera aleatoriamente a modo de prueba de visualización.

Ilustración 48 - Grafana. Visualización de datos.

Conclusiones – Primer escenario de trabajo con los elementos de la solución final

Node-RED es una herramienta muy eficaz y útil para el desarrollo de proyectos IoT.

Raspberry Pi 3 B+ es un dispositivo que aún carece de las características necesarias para manejar estos programas con fluidez.

Hay que tener en cuenta que a mayor seguridad más lenta será la comunicación.

6.3.3 Modificaciones para mejorar el rendimiento

Observando los resultados obtenidos en el primer escenario que contenía todos los elementos finales escogidos para formar parte de la solución final se observó que la frecuencia máxima estable (controlada) con la que se podía capturar y enviar lo datos era de 100Hz.

Aunque el periodo de muestreo es suficiente no se alcanzaba el objetivo establecido por GICI de 360 muestras por segundo. Es por ello que se decidió investigar si era posible mejorar el rendimiento del dispositivo AN-33IoT.

Las modificaciones necesarias para mejorar el rendimiento del dispositivo se han realizado cambiando algunos parámetros de la programación interna.

Prueba de mejora de rendimiento

Para realizar la comprobación de la eficacia de las modificaciones se ha decidido utilizar un código simple en vez del código final. En concreto se ha utilizado el código (*Velocidad_GSR.ino*) que realiza una función similar al descrito en apartados anteriores para la prueba de funcionamiento del sensor GSR.

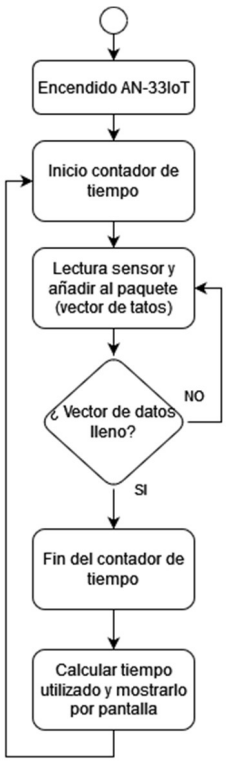


Ilustración 49 – Flujograma de Velocidad_GSR.ino

Sin la modificación de los parámetros internos, el tiempo necesario para la captura de 360 muestras es de 833.34 microsegundos por muestra, equivalente a una frecuencia de muestreo de 1200Hz.

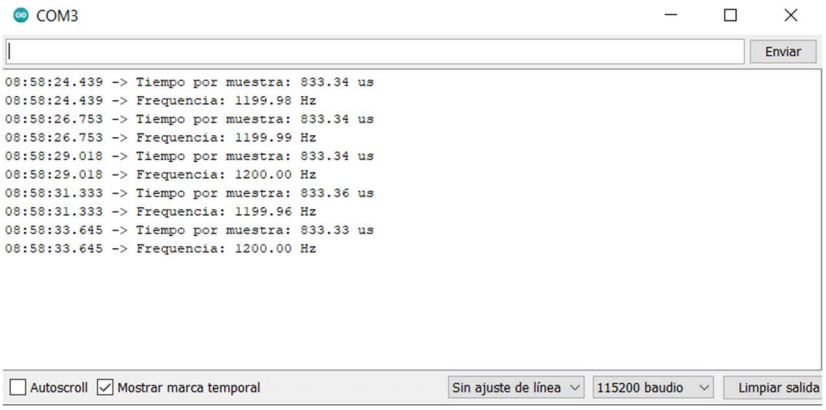


Ilustración 50 - Prueba de velocidad de adquisición. Archivo wiring.c por defecto.

Sin embargo, como se puede observar en el flujograma, esté código es una versión simple que no contempla el envío de los datos al broker, únicamente se centra en comprobar el efecto de las modificaciones en los parámetros internos del AN-33IoT.

Una de las opciones disponibles para mejorar el tiempo de muestro consiste en modificar el valor del parámetro *PRESCALER* (pre-escalador) en el archivo *wiring.c* ubicado en *"C:\Users\username\AppData\Local\Arduino15\packages\arduino\hardware\samd\1.8.11\cores\arduino"*.

Este parámetro permite modificar la velocidad con la que se realizar una conversión ADC. Alterando el retardo de propagación de una medición ADC esta función puede utilizarse para reducir/aumentar la frecuencia de reloj del sistema y su consumo energético [96][97].

Al cambiar la división del reloj de 512 a 64 se consigue mejorar la frecuencia de muestreo a casi 9000 Hz, 111.44 microsegundos por muestra. Esto es una velocidad de lectura de casi 7.5 veces más rápido.

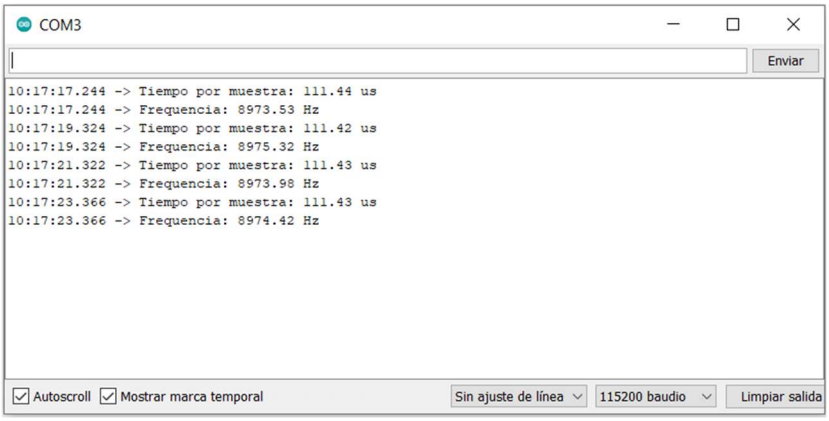


Ilustración 51 - Prueba de velocidad de adquisición. Archivo wiring.c modificado.

Conclusiones – Modificaciones para mejorar el rendimiento

Es posible, mediante la modificación de parámetros internos, alterar y mejorar el rendimiento de Arduino.

6.3.4 Escenario final

El escenario final hace uso del código *TFM_v3.ino*. Este código sigue la misma estructura de diagrama de flujo de su versión anterior con la diferencia de se han introducido las variables necesarias para indicar la frecuencia de muestreo deseada y las relacionadas con la nueva función para visualizar, a través del monitor serie, las frecuencias máxima y mínima de envío del paquete (*mostrarFrecuencias()*).

Se ha depurado el código, eliminando errores y partes innecesarias como comentarios, visualizaciones de parámetros (*reportToSerial()* ya no se ejecuta) o variables que, tras la modificación de la estructura del código, dejaban de ser necesarias.

Siguiendo el mismo procedimiento desarrollado en apartados anteriores, mediante Node-RED se seleccionarán los parámetros y se enviarán a la base de datos InfluxDB.

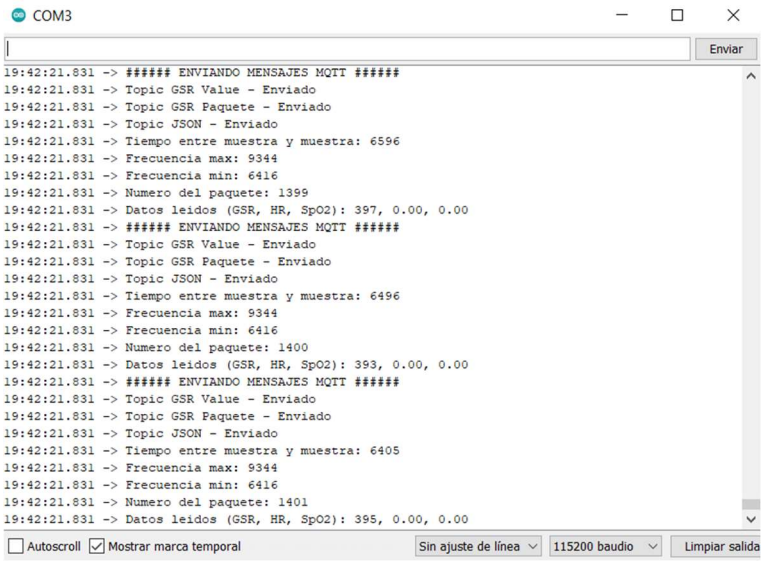


Ilustración 52 – Prueba de velocidad. Escenario final.

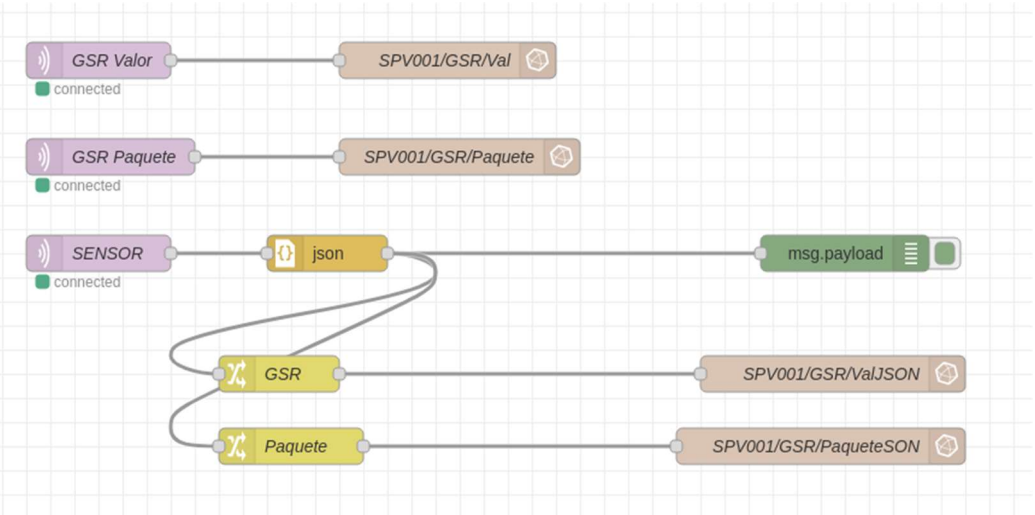


Ilustración 53 - Flujo Node-red. Escenario final.

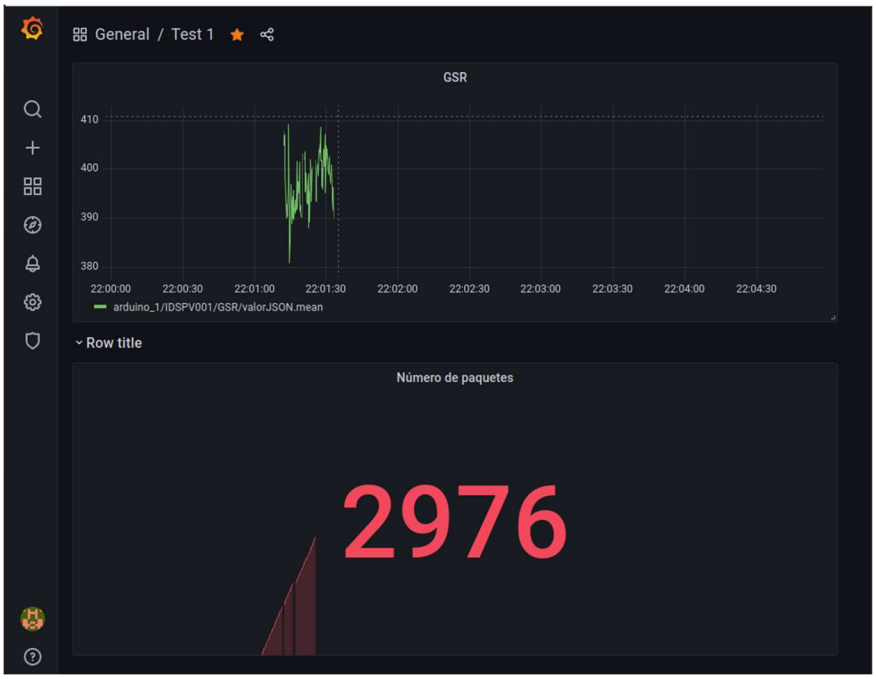


Ilustración 54 - Monitorización del GSR y el número de paquetes. Escenario final.

Modificaciones para mejorar el rendimiento – Escenario final

Utilizando el mismo código (*TFM_v3.ino*) se ha procedido a ver los efectos de modificar el archivo *wiring.c*. con el objetivo de mejorar la frecuencia de muestreo.

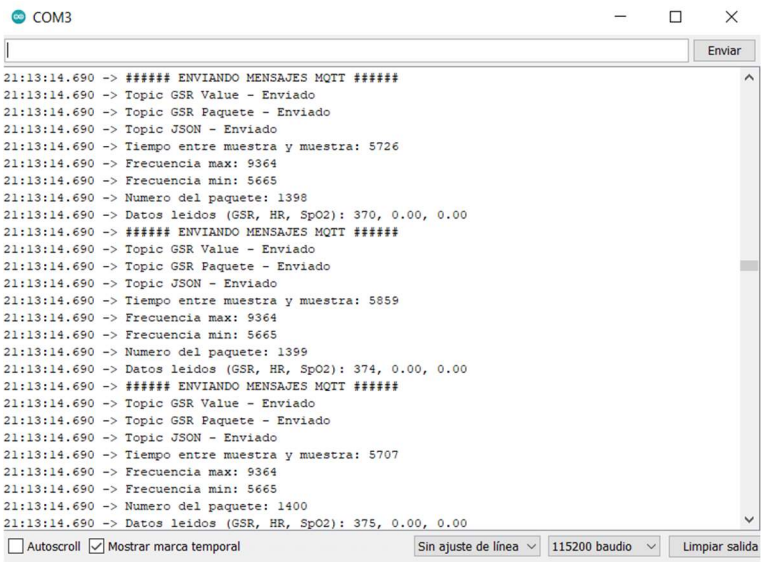


Ilustración 55 - Monitor serie. Escenario final. Modificaciones de rendimiento.

Conclusiones – Escenario final

La frecuencia de muestreo sin modificaciones ha tomado valores de 130Hz de media.

Tras la modificación del archivo *wiring.c* se ha conseguido mejorar la frecuencia de muestreo, tomando una media de valores de 141.75 Hz.

6.4 Objetivos secundarios

Aquí se incluyen las pruebas realizadas en relación a los objetivos secundarios: estos son conectar varios dispositivos al servidor al mismo tiempo, diseñar un sistema de envío de alertas y estudiar las capacidades de un dispositivo ESP32

6.4.1 Conexión de varios dispositivos

Haciendo uso del código final (TFM_v3.ino), este se implantó en otro dispositivo AN33-IoT cambiando únicamente el ID del cliente de 001 a 002.

Haciendo las modificaciones oportunas en los nodos de Node-RED para hacer referencia a los nuevos *topics* se ha conseguido que se almacenen datos de ambos dispositivos al mismo tiempo.

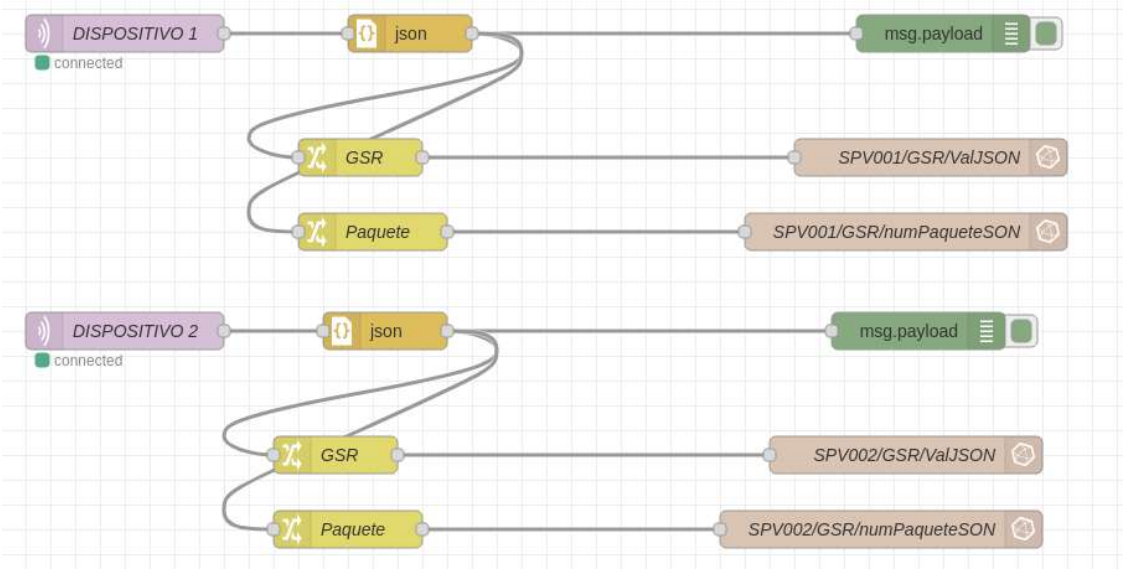


Ilustración 56 - Flujo Node-RED con dos dispositivos.

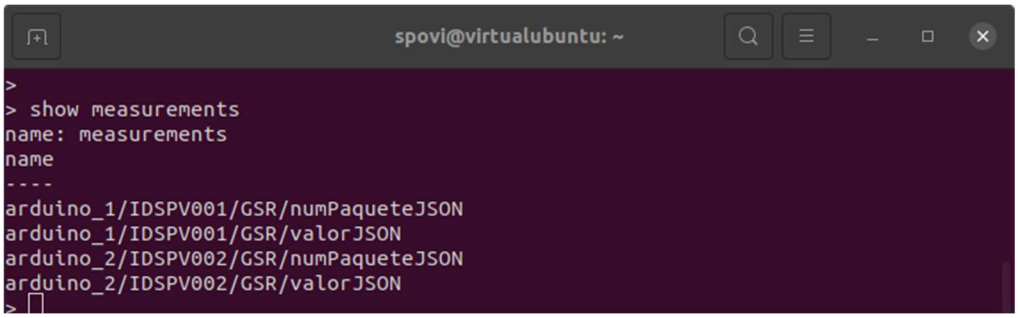


Ilustración 57 - Base de datos InfluxDB. Varios dispositivos.

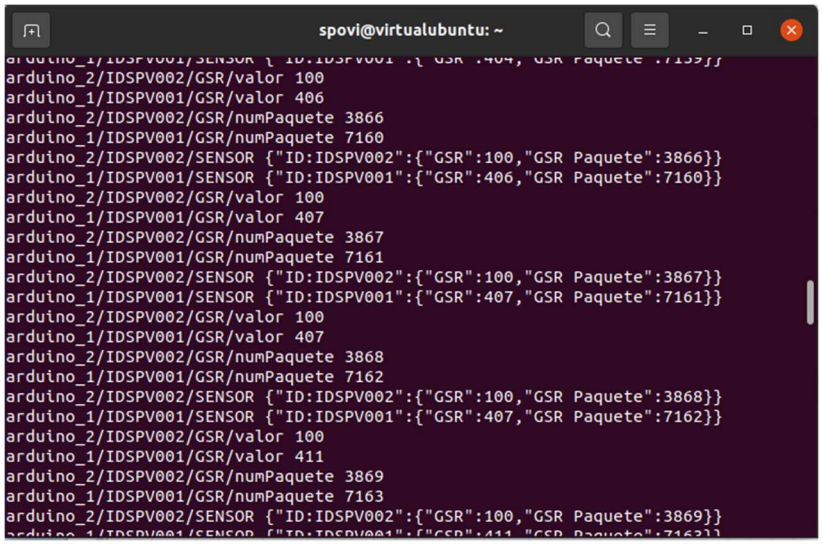


Ilustración 58 - Broker Mosquitto. Varios dispositivos.

6.4.2 Envío de alertas

Node-RED permite mediante sus nodos crear un flujo de trabajo con el que, introduciendo un código muy simple, es posible generar alertas y enviar el mensaje que se especifique a Twitter o el correo electrónico que se desee.

En este caso se va a utilizar una cuenta de correo Gmail para el envío de mensajes por correo. Para que Node-RED realice el envío del mensaje es necesario habilitar la opción de utilizar "apps menos seguras" desde las opciones de seguridad de Google Account.

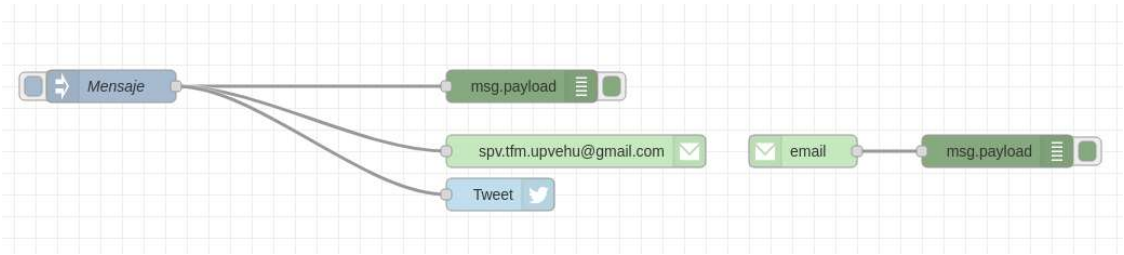


Ilustración 59 - Node-red. Envío de alertas.

Mediante la cuenta creada específicamente para este proyecto (*spv.tfm.upvehu@gmail.com*) se ha realizado el envío de un mensaje de alerta a través de Node-red.

A modo de ejemplo, se ha utilizado el nodo de "inyectar mensaje" el cual sustituye al flujo que controla cuando se produce el evento deseado para enviar el mensaje.

Se ha decidido simular la situación en la que, mediante un sensor de temperatura, integrado en el dispositivo IoT, se detectase que se ha sobrepasado un umbral crítico establecido en 39 ºC. El mensaje tiene como título "Alerta Paciente SPV001" haciendo referencia al paciente y su contenido detalla del problema.

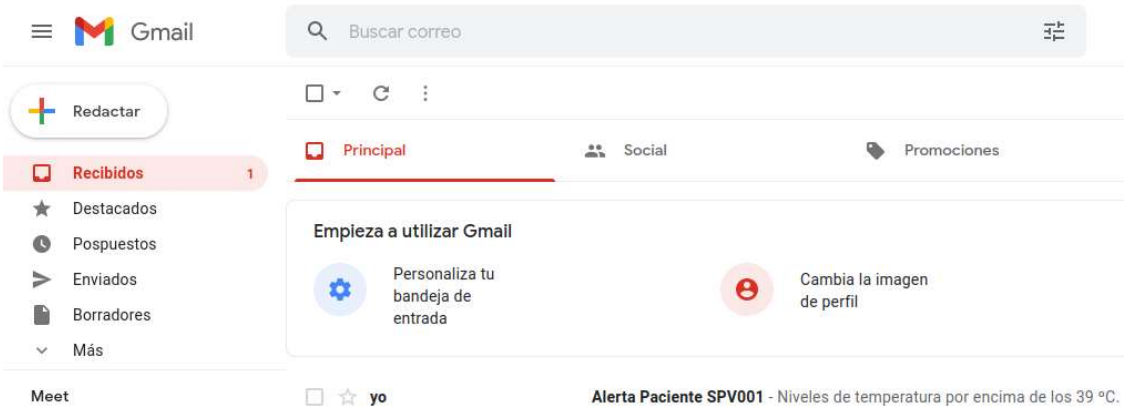


Ilustración 60 - Mensaje de alerta.

6.4.3 Primeras pruebas de funcionamiento con un ESP32

Con el objetivo de solucionar el inconveniente que plantea que el AN33-IoT sólo posea un núcleo y por lo tanto sólo pueda leer y ejecutar el código de forma secuencial, se ha optado por realizar unas pruebas con una placa con dos núcleos, en concreto un Az-Delivery ESP32-WROOM-32 NodeMCU.

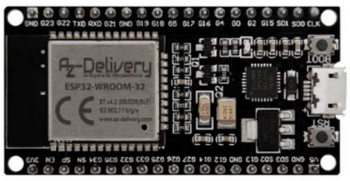


Ilustración 61 - Az-Delivery ESP32-WROOM-32 NodeMCU [77].

La idea que se persigue con la utilización de un dispositivo de dos núcleos es la de utilizar un núcleo para la lectura de los datos y su almacenamiento en un buffer de forma ininterrumpida, y con el otro ir sacando la información del *buffer* para enviarla al *broker* MQTT.

Arduino IDE soporta FreeRTOS (*Free Real Time Operating System*) para el ESP32, el cual permite manejar varias tareas en paralelo, las cuales se ejecutan de forma independiente.

Prueba de funcionamiento – Control de dos ledes

Lo que se pretende con esta prueba es entender como trabajar con dos núcleos. Para ello se dispone de dos ledes, uno rojo y otro verde, y lo que se pretende es que funcionen de manera independiente. El código utilizado ha sido *task_different_cores.ino*.

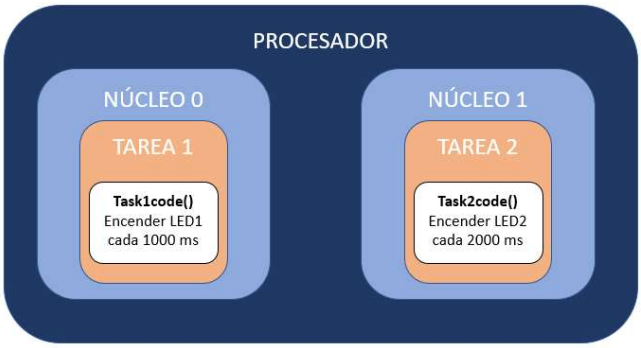


Ilustración 62 - Funcionamiento de los núcleos del ESP32.

A continuación, se muestra el esquema de conexionado (no realizado a escala).

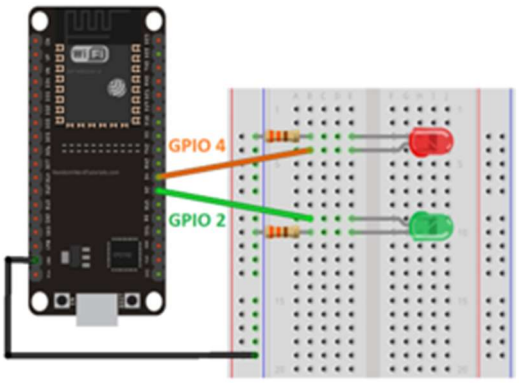


Ilustración 63 - Esquema de conexión ESP32 con dos ledes [98].

A continuación, se muestran los resultados obtenidos y visualizados en el monitor serie de Arduino IDE.

```
11:17:32.004 -> Led Core 0 - ON
11:17:32.004 -> Led Core 1 - ON
11:17:32.979 -> Led Core 0 - OFF
11:17:33.999 -> Led Core 0 - ON
11:17:33.999 -> Led Core 1 - OFF
11:17:34.970 -> Led Core 0 - OFF
```

Ilustración 64 – Monitorización del estado de los ledes.

Conclusiones - ESP32

Se puede observar en los resultados mostrados como los dos leds realizan acciones de encendido/apagado exactamente al mismo momento.

CAPÍTULO 8

ANÁLISIS DE RESULTADOS

7 ANÁLISIS DE RESULTADOS.

En este capítulo se realiza un análisis detallado de las conclusiones y resultados obtenidos durante el desarrollo del trabajo.

7.1 Análisis – Pruebas iniciales

En estos códigos se realiza únicamente la lectura de los datos postergando el envío de la información al *broker* para otros códigos.

Se han comprobado que ambos sensores funcionan correctamente, pero se ha observado que para el sensor LM324 del Grove GSR no sirven los pines de alimentación de 3.3V ni de 5V del AN-33IoT, es necesario utilizar el pin VIN.

Es posible establecer un método para el control de la frecuencia con la que se pueden capturar los datos. En los siguientes apartados se verá como existe una función con la misma funcionalidad, pero con mayor precisión denominada *micros()*.

7.2 Análisis – Conexión con el *broker*

Durante la realización de estos dos códigos se han incorporado una serie de mejoras respecto a los anteriores.

Entre ellas, se ha incluido un fichero extra (*arduino_secrets.h*) donde se introduce el SSID y PASS de la red a la que se quiere conectar, de forma que en el caso de que se comparta el código no figuren en él.

Se ha introducido en el *setup* el código necesario para que tras inicializar el puerto serie espere a que se abra antes de continuar ejecutando código. Antes de introducir esto, el monitor serie tardaba 0,2 segundos aproximadamente en comenzar a mostrar lo que se estaba transmitiendo, por lo que si había algo que estaba codificado en el *setup()* se ejecutaba tan rápido que no aparecía por el monitor serie.

Incluye como opción añadir seguridad a la comunicación mediante SSL/TLS entre el dispositivo y el *broker*. Esta opción está a modo de comentario en el código, por lo que habría que descomentarlo para que el *framework* lo pudiera ejecutar y realizar unas modificaciones mínimas (también indicadas en los comentarios).

Por otro lado, la herramienta MQTTLens ha permitido verificar el correcto funcionamiento de los dos códigos.

7.3 Análisis – Primer escenario de trabajo con los elementos de la solución final

Mediante Node-RED es posible guardar los datos de diversas formas dependiendo del nodo seleccionado. Entre algunas de las opciones disponibles se encuentran la generación de archivos .JSON, texto plano (.txt) o CSV que podría ser abierto posteriormente con Excel.

Como curiosidad Mosquitto permite implementar desde el terminal un comando para que se guarden en un texto plano los datos recibidos, sin embargo, no se tiene un control para guardarlos de forma ordenada.

Se descubrió un fallo importante del código en la versión implementada en la Raspberry Pi, este consistía en que no se vaciaba el paquete que contenía la información generando un error en el envío debido a que se superaba el tamaño permitido del paquete.

La biblioteca *PubSubClient* limita el tamaño de los mensajes MQTT a 128 bytes. Aunque es posible que un mensaje de formato largo funcione si el tamaño del búfer del mensaje se ha aumentado en la configuración.

Incorporar un mecanismo de seguridad usuario/contraseña para hacer uso del *broker* implica que deberá de ser tenido en cuenta para acceder a él tanto desde el AN-33IoT como desde Node-RED. Además, dentro de Node-RED se puede especificar el QoS con el que se recibe o envía un mensaje.

Obviamente, el parámetro de la velocidad de la consola serie USB se tiene que poner al máximo de la capacidad del AN-33IoT (115200 baudios) para que la comunicación entre el dispositivo y el portátil sea la más rápida posible.

7.4 Análisis – Modificaciones para mejorar el rendimiento

La modificación del parámetro PRESCALER es una buena opción para mejorar la velocidad de conversión ADC, y por tanto la velocidad de ejecución del código.

El archivo *wiring.c* es una librería en C con múltiples parámetros que pueden ser modificados para alterar el miento del dispositivo Arduino. Sin embargo, si se comete un error en su programación puede conllevar a un mal funcionamiento del dispositivo o incluso dejarlo inservible.

Existen otros parámetros dentro de esta librería que podrían mejorar el rendimiento del dispositivo. Algunos de esos parámetros permiten controlar la resolución de la conversión ADC, por defecto en 10 (*ADC_CTRLB_RESSEL_10BIT*) o establecer la longitud máxima del tiempo de muestreo, por defecto en 63 (*ADC->SAMPCTRL.reg = 0x3f*).

7.5 Análisis – Escenario final

La frecuencia de muestreo deseada ha sido introducida de forma manual (360Hz), sin embargo, para 1400 paquetes enviados, el mejor resultado ha sido 6,416 milisegundos (156 Hz aproximadamente) y el peor de 9,344 milisegundos (107 Hz aproximadamente).

Tras realizar la modificación, para 1400 paquetes enviados, el mejor resultado ha sido 5,665 milisegundos (176.5 Hz aproximadamente) y el peor de 9,364 milisegundos (107 Hz aproximadamente).

Comparando resultados, reduciendo la frecuencia de reloj para la conversión ADC mediante la modificación del al archivo *wiring.c* se ha conseguido mejorar la velocidad de media de muestreo en un 9,04% y en un 13,14% el envío más rápido.

Aunque estas frecuencias quedan lejos de la óptima deseada (360Hz) se encuentran dentro de los valores válidos para la monitorización de señales biológicas, siendo 100Hz un valor aceptable.

Se ha probado por separado diferentes funcionalidades de la solución final demostrando que son operativas y resta finalizar con la versión integrada de todos los componentes (SW y HW) del sistema.

La incorporación del sensor MAX30100 es una próxima línea futura que se en la que ya se está trabajando y en la que se continuará realizando en el seno de la línea de investigación del GICI.

Se ha visto que la lectura secuencial del código imposibilita realizar dos tareas al mismo tiempo. Por lo que se genera una nueva línea de investigación sobre posibilidad de utilizar otros dispositivos de dos núcleos como solución para realizar varias tareas al mismo tiempo (*multitasking*).

7.6 **Análisis – Gestión de alertas**

Node-RED permite implementar un sistema de alertas eficaz. Contiene múltiples nodos que permiten la gestión de avisos y alertas.

En este proyecto se ha visto cómo es posible enviar mensajes a una cuenta Gmail, pero también existe la posibilidad de comunicarse con otros medios como Twitter, Telegram o Whatsapp.

Queda pendiente para futuras implementaciones su integración con la solución final con varios sensores y en la que se incorpore el software de técnicas inteligentes desarrollado por el GICI.

7.7 **Análisis – ESP32**

Se puede observar en los resultados mostrados como los dos leds realizan acciones de encendido/apagado exactamente al mismo momento.

Para asignar partes de código a un núcleo específico es necesario la creación de tareas . Su sistema de prioridades empieza en 0, siendo esta la de menor prioridad. El procesador ejecutará las tareas con mayor prioridad primero.

Este dispositivo se podría usar para desarrollos futuros realizando las modificaciones oportunas en el código.

- El núcleo 0 podría encargarse de recoger las muestras (a frecuencia controlada) e ir guardándolas en el buffer o paquete.
- El núcleo 1 se encargaría de enviar el paquete al *broker*.

Sin embargo, sería necesario realizar un estudio más complejo. Una solución podría consistir en tener dos paquetes de forma que cuando se esté escribiendo en él no se pueda estar enviando la información que contiene.

Otra observación realizada con esta placa en concreto basada en el ESP32 de Espressif es que es necesario mantener pulsado el botón “BOOT/FLASH” para cargar un nuevo *sketch*. No es un problema grave, pero sí que puede resultar molesto para el programador.

Este dispositivo implementa una función que le permite asignar un ID único al dispositivo a diferencia de las placas Arduino.

CAPÍTULO 9

PLAN DE PROYECTO Y PLANIFICACIÓN

8 PLAN DE PROYECTO Y PLANIFICACIÓN

En este capítulo se reflejan los diferentes hitos alcanzados, mostrando los pasos que se han seguido para el desarrollo de este proyecto.

FASE 1 – BÚSQUEDA DE INFORMACIÓN:

- **Búsqueda bibliográfica y localización de fuentes de consulta**.
    - Buscar información básica en relación a dispositivos de medida de señales fisiológicas (ECG, GSR, RSP, PPG) y comunicación inalámbrica ya disponibles.
- **Centrarse en desarrollos con Arduino Nano 33 IoT (AN-33IoT)**
    - Reunir información del dispositivo y fuentes de consulta
- **Estudiar las propuestas de comunicación que existen para estas plataformas y seleccionar aquella que cumpla con los requisitos fundamentales de comunicación estable, segura y robusta**.
    - Estudiar el enfoque del Edge-Computing y como se ajusta a los objetivos de este TFM (estado del arte)
    - Preparar un listado de posibles soluciones a emplear.
    - Realizar una comparativa y seleccionar los protocolos de comunicación más idóneos, en base a las necesidades de este tipo de aplicación y a las características del dispositivo empleado.

FASE 2 – DESARROLLO DE PRUEBAS:

- **Preparación del entorno**
    - Instalación del SW necesario.
- **Pruebas iniciales**
    - Comprobar las soluciones propuestas por el GICI
    - Establecer conexión entre el AN33-IoT y la red WiFi.
    - Realizar mediciones con los sensores y el AN-33IoT.

FASE 3 – DESARROLLO DE LA SOLUCIÓN

- **Objetivo principal**
    - Establecer comunicación segura entre el AN-33IoT y el broker Mosquitto.
    - Utilización de Node-RED como herramienta puente entre el broker y la base de datos InfluxDB.
    - Visualización de los datos almacenados en InfluxDB mediante Grafana.
- **Modificaciones para la mejora del rendimiento**
    - Investigar los posibles cambios a realizar tanto en el HW como en el SW en el AN-33IoT
    - Realizar las pruebas con las modificaciones y comparar resultados.
- **Objetivos secundarios**
    - Probar a conectar varios dispositivos a la red.
    - Investigar como generar un sistema de alertas
    - Primeras pruebas de funcionamiento del ESP32.

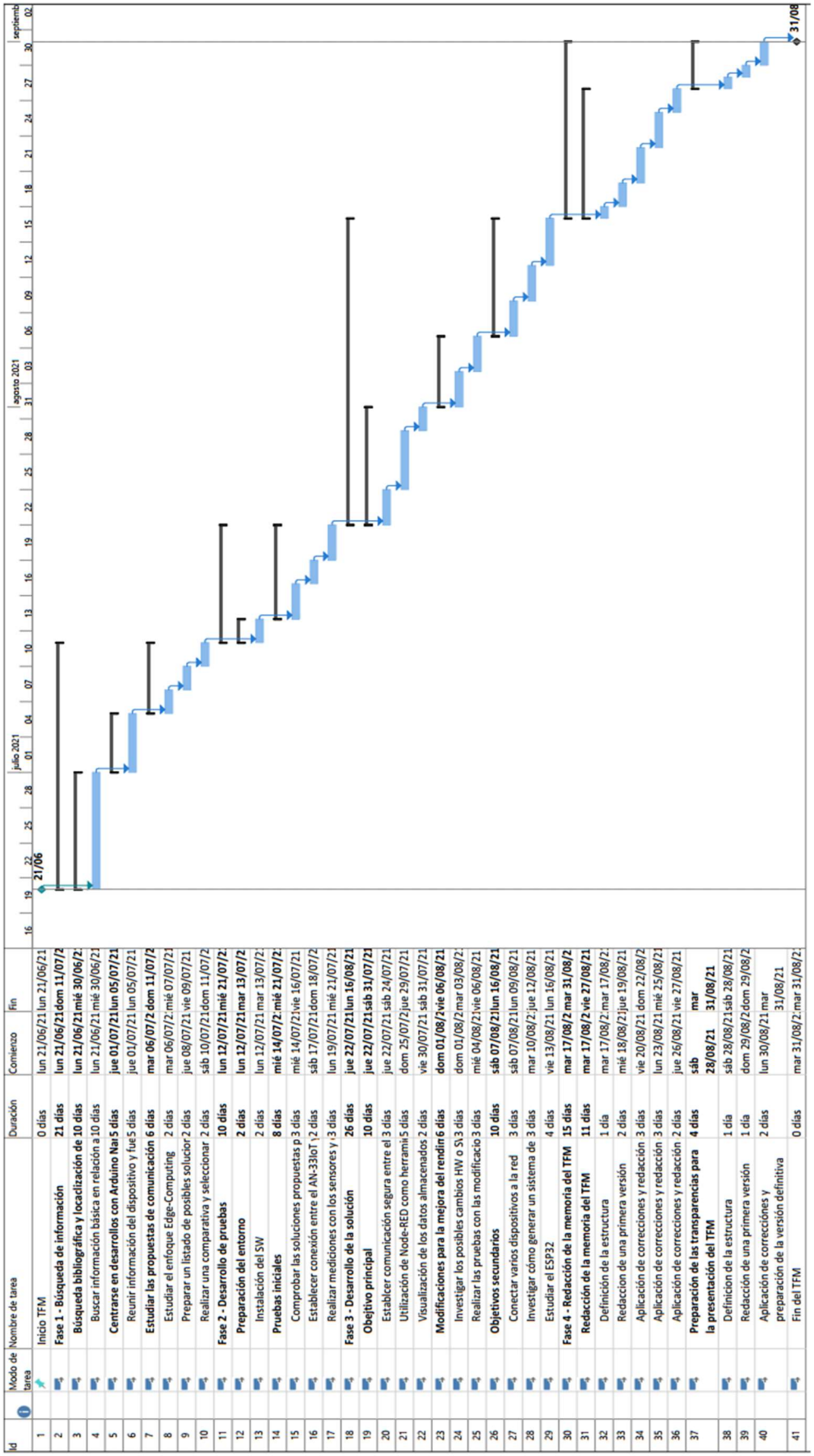
FASE 4 – REDACCIÓN DE LA MEMORIA DEL TFM

- **Redacción de la memoria del TFM.**
  - Definición de estructura: definir la estructura o índice de las secciones de la memoria del TFM, respetando los requisitos y reglas de redacción de la memoria.
  - Redacción de una primera versión: redactar una primera versión de la memoria que incluya la mayoría de los puntos o secciones de la memoria y entregarla para su corrección a los tutores.
  - Aplicación de correcciones y redacción de una segunda versión: aplicar las correcciones sugeridas por los tutores y mejorar la reacción. Redactar las secciones de la memoria faltantes y entregar la segunda versión de la memoria a los tutores.
  - Aplicación de correcciones y redacción de una tercera versión: aplicar las ultimas correcciones sugeridas por los tutores y mejorar la redacción. Redactar la memoria final y entregar la versión final de la memoria.

- **Preparar las transparencias para la presentación del TFM.**
  - Definición de estructura: definir la estructura o índice de las partes de la presentación del TFM, respetando los requisitos y tiempos máximos de la defensa.
  - Redacción de una primera versión: prepara una primera versión del PPT que incluya lo más relevante a presentar en la defensa y entregarla para su corrección a los tutores.

8.1 Diagrama de Gantt

En la siguiente hoja, se presenta el diagrama de Gantt donde queda reflejado la evolución del trabajo realizado.



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		Inicio TFM	0 días	lun 21/06/21	lun 21/06/21
2		**Fase 1 - Búsqueda de información**	**21 días**	**lun 21/06/21**	**dom 11/07/2**
3		**Búsqueda bibliográfica y localización de 10 días**		**lun 21/06/21**	**mié 30/06/2**
4		Buscar información básica en relación a 10 días		lun 21/06/21	mié 30/06/21
5		**Centrarse en desarrollos con Arduino Nan 5 días**		**jue 01/07/21**	**lun 05/07/21**
6		Reunir información del dispositivo y fue 5 días		jue 01/07/21	lun 05/07/21
7		**Estudiar las propuestas de comunicación 6 días**		**mar 06/07/2**	**dom 11/07/2**
8		Estudiar el enfoque Edge-Computing	2 días	mar 06/07/2	mié 07/07/21
9		Preparar un listado de posibles solucior 2 días		jue 08/07/21	vie 09/07/21
10		Realizar una comparativa y seleccionar	2 días	sáb 10/07/21	dom 11/07/2
11		**Fase 2 - Desarrollo de pruebas**	**10 días**	**lun 12/07/21**	**mié 21/07/2**
12		**Preparación del entorno**	**2 días**	**lun 12/07/21**	**mar 13/07/2**
13		Instalación del SW	2 días	lun 12/07/21	mar 13/07/2
14		**Pruebas iniciales**	**8 días**	**mié 14/07/2**	**mié 21/07/2**
15		Comprobar las soluciones propuestas p 3 días		mié 14/07/21	vie 16/07/21
16		Establecer conexión entre el AN-33IoT 2 días		sáb 17/07/21	dom 18/07/2
17		Realizar mediciones con los sensores y 3 días		lun 19/07/21	mié 21/07/21
18		**Fase 3 - Desarrollo de la solución**	**26 días**	**jue 22/07/21**	**lun 16/08/21**
19		**Obejtivo principal**	**10 días**	**jue 22/07/21**	**sáb 31/07/21**
20		Establcer comunicación segura entre el 3 días		jue 22/07/21	sáb 24/07/21
21		Utilización de Node-RED como herrami 5 días		dom 25/07/2	jue 29/07/21
22		Visualización de los datos almacenados 2 días		vie 30/07/21	sáb 31/07/21
23		**Modificaciones para la mejora del rendin 6 días**		**dom 01/08/2**	**vie 06/08/21**
24		Investigar los posibles cambios HW o SV 3 días		dom 01/08/2	mar 03/08/2
25		Realizar las pruebas con las modificacio 3 días		mié 04/08/2	vie 06/08/21
26		**Objetivos secundarios**	**10 días**	**sáb 07/08/21**	**lun 16/08/21**
27		Conectar varios dispositivos a la red	3 días	sáb 07/08/21	lun 09/08/21
28		Investigar cómo generar un sistema de	3 días	mar 10/08/2	jue 12/08/21
29		Estudiar el ESP32	4 días	vie 13/08/21	lun 16/08/21
30		**Fase 4 - Redacción de la memoria del TFM**	**15 días**	**mar 17/08/2**	**mar 31/08/2**
31		**Redacción de la memoria del TFM**	**11 días**	**mar 17/08/2**	**vie 27/08/21**
32		Definición de la estructura	1 día	mar 17/08/2	mar 17/08/2
33		Redaccion de una primera versión	2 días	mié 18/08/2	jue 19/08/21
34		Aplicación de correcciones y redacción	3 días	vie 20/08/21	dom 22/08/2
35		Aplicación de correcciones y redacción	3 días	lun 23/08/21	mié 25/08/21
36		Aplicación de correcciones y redacción	2 días	jue 26/08/21	vie 27/08/21
37		**Preparación de las transparencias para la presentación del TFM**	**4 días**	**sáb 28/08/21**	**mar 31/08/21**
38		Definicion de la estructura	1 día	sáb 28/08/21	sáb 28/08/21
39		Redacción de una primera versión	1 día	dom 29/08/2	dom 29/08/2
40		Aplicación de correcciónes y preparación de la versión definitiva	2 días	lun 30/08/21	mar 31/08/21
41		Fin del TFM	0 días	mar 31/08/2	mar 31/08/2

CAPÍTULO 10

CONCLUSIONES Y TRABAJOS FUTUROS

9 CONCLUSIONES Y TRABAJOS FUTUROS

9.1 **Conclusiones**

La elaboración de un proyecto IoT no es algo trivial. Aunque para la realización de proyecto sencillos es fácil realizar implementaciones funcionales, entender todo el proceso para obtener un resultado eficiente en todos sus aspectos conlleva tener un conocimiento elevado de las redes de comunicación.

Como resultado final, se ha conseguido presentar una solución para cada uno de los desafíos que presenta desarrollar un proyecto IoT de bajo coste (objetivo principal de este trabajo). Para la conectividad se ha establecido conexión vía WiFi , se ha presentado una estructura para la arquitectura del sistema; se ha conseguido comunicar todos los dispositivos y sensores que eran necesarios para la problemática de este TFM solucionando los problemas de interoperabilidad e integración; se ha utilizado una base de datos externa al dispositivo solventando el problema del almacenamiento; se han establecido y utilizado mecanismos de protección en las distintas etapas logrando satisfacer los requisitos de seguridad, confianza y privacidad.

Se válida la utilización del **Arduino Nano 33 IoT (AN-33IoT**) como dispositivo de bajo coste para obtener una solución a la problemática aquí planteada. Su microprocesador le permite realizar procesamientos a una velocidad aceptable para la aplicación deseada. Tiene suficiente memoria para almacenar esa información y enviarla posteriormente de forma inalámbrica al sistema de almacenamiento.

Aunque no es un objetivo imprescindible de este proyecto la visualización de los datos en tiempo real, conformándose como tiempos cuasi-rales, se ha detectado que existen retardos, inferiores a un segundo, entre el envío del mensaje por parte de AN-33IoT y su recepción en el servidor

La comunicación **WiFi** y el protocolo **MQTT** son factibles para establecer las comunicaciones entre los distintos dispositivos. Si bien es muy interesante las posibilidades que ofrece WiFi HaLow como futura alternativa en cuando saquen dispositivos de bajos costo compatibles.

Mosquitto MQTT se presenta como una buena solución para el alcance de este proyecto, sin embargo, para futuras aplicaciones es recomendable estudiar la posibilidad de implementar otro *broker* capaz de garantizar que habrá siempre uno funcionando en caso de que la principal sufra una caída. Por otra parte, EMQ queda pendiente como línea de investigación futura.

Tras realizar la modificación interna del AN-33IoT, las frecuencias de muestreo obtenidas se encontraban entre los 107 Hz y 176.5 Hz, valores por encima de los mínimos requeridos (100Hz).

Respecto a los objetivos secundarios, observando el trabajo realizado por los autores citados en el capítulo del Estado del Arte, queda demostrado que existen diferentes métodos para obtener una solución válida en el caso de querer conectar múltiples dispositivos. También se ha presentado una estructura en Node-RED de cómo se implementaría la recepción de datos por parte de dos dispositivos, quedando así solucionado el primer objetivo secundario.

Se ha logrado realizar unas primeras pruebas con un dispositivo basado en el modelo ESP32, cumpliendo así el segundo objetivo parcial secundario. Durante su experimentación se ha comprobado como gracias a su doble núcleo es capaz de realizar dos tareas al mismo tiempo. Esto puede llevar al desarrollo de una nueva estructuración en el código para reducir tiempos,

de forma que se pueda estar recopilando datos y almacenándolos temporalmente en un buffer de forma ininterrumpida, mientras con el otro núcleo se envía los datos del buffer al servidor de datos.

Mediante el uso de Node-RED se ha comprobado que es posible enviar alertas vía Twitter o correo electrónico en el caso de detectar eventos patológicamente anómalos o peligrosos. Se destaca que también ofrece la posibilidad de crear un *topic* de forma que el propio dispositivo implementado se subscriba al él. Por otro lado, aunque esta es una forma de realizar este último apartado parcial secundario, la búsqueda de información realizada a lo largo del desarrollo de este proyecto mostró que estas alertadas se pueden implementar de diversas formas dependiendo de los elementos seleccionados.

Para finalizar este apartado, se recuerda que es importante tener siempre presente que la tecnología cambia espacios de tiempo relativamente cortos, por lo que es vital estar al día de las novedades. Aun habiendo implementado una solución válida para este TFM, es fácil que una tecnología que a día de hoy se encuentre entre las más usadas, en unos años quede obsoleta.

9.2 Trabajos Futuros

Como se ha podido comprobar las posibilidades de desarrollar una solución IoT son muy amplias. El IoT está basado en multitud de tecnologías en constante evolución. Se trabaja a diario en la creación de nuevas versiones de protocolos de comunicación, *middlewares* posibilitando la implementación de nuevas soluciones.

Es por ello que, aun habiéndose cumplido los objetivos establecidos inicialmente, quedan pendientes de estudio algunas vías de investigación con las que se cree se mejoraría el presente trabajo:

1. **Implementación de otros sensores**. Como se describió anteriormente, queda pendiente la incorporación del sensor MAX30100. Es una próxima línea futura que se en la que ya se está trabajando y en la que se continuará realizando en el seno de la línea de investigación del GICI.

2. **Aplicación de software de análisis.** El GICI se encuentra actualmente desarrollando una solución programada con la que identificar eventos preestablecidos, como síntomas de alguna patología, a partir de las señales fisiológicas recibidas. Permitiría prevenir problemas más graves a través de una detección temprana del problema.

3. **Probar nuevos protocolos**: Existen hoy en día nuevos protocolos de comunicación, como el WiFi HaLow (<1GHz), cuyas características se presentan como una mejora significativa para aplicaciones IoT respecto al WiFi de bandas tradicionales (2.4 GHz y 5GHz). Sin embargo, faltan dispositivos de bajo coste en el mercado que sean capaces de soportarlo.

4. **Nuevos dispositivos de bajo coste**: Estudiar la posibilidad de desarrollar una implementación con dispositivos que ofrecen dos núcleos como posible solución al problema de la lectura secuencial del AN33-IoT. En este trabajo ya se ha comenzado a trabajar en esta línea de investigación con un ESP32, pero también hay que tener en cuenta las novedades del mercado. El dispositivo Arduino Nano RP2040 Connect se comenzó a comercializas a principios del 2021 y se presenta como un gran candidato para el desarrollo de nuevas soluciones basadas en IoT.

5. **Implementación de software de control:** Tal y como se explicó en la selección de la solución propuesta, el alcance de este proyecto no cubre el despliegue de un elevado número de dispositivos, por lo que no es necesario su implementación. Sin embargo, es una línea de investigación importante para futuros despliegues en los que sí que haya una gran cantidad de dispositivos.

6. **EMQ:** Durante el desarrollo de este proyecto se presentó EMQ como un posible *middleware* a seleccionar. Este software es uno de los últimos servidores/*broker* que han salido al mercado y se presenta como un buen candidato para el despliegue de múltiples dispositivos.

CAPÍTULO 11

REFERENCIAS BIBLIOGRÁFICAS

10 REFERENCIAS BIBLIOGRÁFICAS

[1] "El origen e historia del Internet de las Cosas (IoT)." https://www.bcendon.com/el-origen-del-iot/.

[2] C. Chakraborty, A. Banerjee, L. Garg, and J. J. P. C. Rodrigues, *Internet of Medical Things for Smart Healthcare*. Springer.

[3] P. P. Ray, D. Dash, and N. Kumar, "Sensors for internet of medical things: State-of-the-art, security and privacy issues, challenges and future directions," *Comput. Commun.*, vol. 160, pp. 111–131, Jul. 2020, doi: 10.1016/J.COMCOM.2020.05.029.

[4] D. C. Yacchirema Vargas, "Arquitectura de interoperabilidad de dispositivos físicos para le internet de las cosas (IoT)," Universidad Politécnica de Valencia, 2019.

[5] L. Greco, G. Percannella, P. Ritrovato, F. Tortorella, and M. Vento, "Trends in IoT based solutions for health care: Moving AI to the edge," *Pattern Recognit. Lett.*, vol. 135, pp. 346–353, 2020, doi: 10.1016/j.patrec.2020.05.016.

[6] "Protección Senior - Securitas Direct." http://proteccionsenior.com/.

[7] "iHealth Labs Europe - Connected Health." https://ihealthlabs.eu/es/ (accessed Aug. 16, 2021).

[8] "Cómo funciona V-SOS Band | Ayuda Vodafone Particulares." https://ayudacliente.vodafone.es/particulares/iot/v-sos-band/como-funciona-v-sos-band/.

[9] "Reloj Localizador Nock Senior - Neki." https://neki.es/reloj-localizador-gps-ancianos.html.

[10] "Sensovida, la teleasistencia más innovadora para personas mayores – Solidaridad Intergeneracional." https://solidaridadintergeneracional.es/wp/sensovida-la-teleasistencia-mas-innovadora-para-personas-mayores/.

[11] "Mi Smart Band 6 | Xiaomi España | Mi.com." https://www.mi.com/es/mi-smart-band-6/.

[12] "Watch - Apple (ES)." https://www.apple.com/es/watch/.

[13] "Relojes | Samsung España." https://www.samsung.com/es/watches/?product1=sm-r855fzdaeub&product2=sm-r855fzdaeub&product3=sm-r845fzkaeub.

[14] "Huawei Band 6 - HUAWEI España." https://consumer.huawei.com/es/wearables/band6/.

[15] "El mejor smartwatch 2021: guía de compra y comparativa de relojes inteligentes." https://www.xataka.com/analisis/mejor-smartwatch-2020-guia-compra-comparativa-1.

[16] "Arduino - Home." https://www.arduino.cc/.

[17] "Arduino Nano 33 IoT | Arduino Official Store." https://store.arduino.cc/arduino-nano-33-iot.

[18] "Software | Arduino." https://www.arduino.cc/en/software.

[19] "Arduino Forum." https://forum.arduino.cc/.

[20] R. Requena, “Infraestructura de eventos para la Internet de las cosas,” 2014.

[21] “Adafruit Industries, Unique & fun DIY electronics and kits.” https://www.adafruit.com/.

[22] “SparkFun Electronics.” https://www.sparkfun.com/.

[23] “Bitalino.” https://bitalino.com/.

[24] “ESP-SDK | Espressif Systems.” https://www.espressif.com/en/products/software/esp-sdk/overview.

[25] “ESP8266 Wi-Fi MCU I Espressif Systems.” https://www.espressif.com/en/products/socs/esp8266?6.

[26] “ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems.” https://www.espressif.com/en/products/socs/esp32?8.

[27] “IoT Development Framework I Espressif Systems.” https://www.espressif.com/en/products/sdks/esp-idf.

[28] “Instalando el ESP32 | Tienda y Tutoriales Arduino.” https://www.prometec.net/instalando-esp32/.

[29] “Teach, Learn, and Make with Raspberry Pi.” https://www.raspberrypi.org/.

[30] “Microcontrolador vs Microprocesador | Aprendiendo Arduino.” https://aprendiendoarduino.wordpress.com/2015/03/29/microcontrolador-vs-microprocesador/.

[31] P. Waher, *IoT : Building Arduino-Based Projects*. 2016.

[32] A. Kurniawan, *Beginning Arduino Nano 33 IoT*. 2021.

[33] A. Javed, *Building Arduino Projects for the Internet of Things*. 2016.

[34] A. Kurniawan, *Internet of Things Projects with ESP32*. Packt Publishing, 2019.

[35] M. Schawartz, *Internet of Things with ESP8266*. Packt Publishing, 2019.

[36] E. Moghadas, J. Rezazadeh, and R. Farahbakhsh, “An IoT patient monitoring based on fog computing and data mining: Cardiac arrhythmia usecase,” *Internet of Things*, vol. 11, p. 100251, 2020, doi: 10.1016/j.iot.2020.100251.

[37] B. S. Sarjerao, A. Prakasarao, B. S. Sarierao, and A. Prakasarao, “Smart Healthcare Monitoring System Using MQTT Protocol,” in *3rd International Conference for Convergence in Technology (I2CT)*, Nov. 2020, pp. 1–5, doi: 10.1109/I2CT.2018.8529764.

[38] A. Faro and D. Giordano, “ESP32 based Edge Devices to Bridge Smart Devices to MQTT Broker for Healthcare Purposes in the COVID Scenario,” in *10th International Conference on Pervasive and Parallel Computing, Communication and Sensors, PECCS 2020*, 2020, no. Peccs, pp. 978–989, doi: 10.5220/0010147800300041.

[39] I. Ibiricu Elizondo, “Sistema IoT para la monitorización y visualización de parámetros ambientales en entornos industriales Grado en Ingeniería en Tecnologías Industriales Trabajo Fin de Grado,” 2020.

[40] “140+ ESP32 Projects, Tutorials and Guides with Arduino IDE | Random Nerd Tutorials.” https://randomnerdtutorials.com/projects-esp32/.

[41] “Internet of Home Things » Blog Posts.” https://internetofhomethings.com/homethings/?page_id=207.

[42] A. Martín Muñoz, "Despliegue de una red de sensores basada en chips ESP-8266," Universidad Complutense de Madrid, 2020.

[43] "Máquinas virtuales: qué son, cómo funcionan y cómo utilizarlas." https://www.xataka.com/especiales/maquinas-virtuales-que-son-como-funcionan-y-como-utilizarlas.

[44] "Oracle VM VirtualBox." https://www.virtualbox.org/.

[45] "Máquina virtual Windows | Workstation Pro | VMware | ES." https://www.vmware.com/es/products/workstation-pro.html.

[46] "Bitalino." https://bitalino.com/products.

[47] F. M. Medina Núñez and A. F. Paternina De La Rosa, "Diseño y construcción de Oxímetro de pulso inalámbrico," pp. 1–4.

[48] E. Al-Masri *et al.*, "Investigating Messaging Protocols for the Internet of Things (IoT)," *IEEE Access*, vol. 8, pp. 94880–94911, 2020, doi: 10.1109/ACCESS.2020.2993363.

[49] G. Gil Inchaurza, "Comparativa teórica y práctica de middlewares MQTT," Euskal Herriko Unibertsitatea, 2018.

[50] Gastón C. Hillar, *MQTT Essentials – A Lightweight IoT Protocol*. Packt, 1967.

[51] A. Faro, D. Giordano, and M. Venticinque, "Deploying Wifi , RF and BLE sensors for pervasive monitoring and control," in *2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT*, 2020, pp. 605–610, doi: 10.1109/MetroInd4.0IoT48571.2020.9138187.

[52] W. Nazco Torres, "Almacenamiento y Visualización de Series Temporales," Universidad de la Laguna, 2018.

[53] "InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData." https://www.influxdata.com/.

[54] "GitHub - graphite-project/graphite-web: A highly scalable real-time graphing system." https://github.com/graphite-project/graphite-web.

[55] "OpenTSDB - A Distributed, Scalable Monitoring System." http://opentsdb.net/.

[56] "Prometheus - Monitoring system & time series database." https://prometheus.io/.

[57] "Node-RED." https://nodered.org/.

[58] "Medidor PZEM-004 + Arduino Nano Modbus RTU (RS232) & Plataforma IoT Node-RED - PDAControl." http://pdacontroles.com/medidor-pzem-004-arduino-nano-modbus-rtu-rs232-plataforma-iot-node-red/.

[59] "#41: Datalogging with MQTT, Node-RED, InfluxDB, and Grafana – SuperHouse Automation." https://www.superhouse.tv/41-datalogging-with-mqtt-node-red-influxdb-and-grafana/.

[60] D. L. Llinares, "Desarrollo de aplicación IoT Para la monitorizacion de consumos eléctricos de una vivienda," 2020.

[61] "ESPHome." https://esphome.io/.

[62] "News - Tasmota." https://tasmota.github.io/docs/.

[63] "SONOFF Official Homepage| Smart Home automation SONOFF Official." https://sonoff.tech/.

[64] "Home Assistant." https://www.home-assistant.io/.

[65] "Grafana: The open observability platform | Grafana Labs." https://grafana.com/.

[66] G. Goikoetxea Gutierrez, "Diseño y desarrollo de prenda sensorizada para medida de señales fisiológicas en aplicaciones socio-sanitarias.," Euskal Herriko Unibertsitatea, 2021.

[67] "Arduino MKR WiFi 1010 — Arduino Online Shop." https://store-usa.arduino.cc/products/arduino-mkr-wifi-1010.

[68] "Deep dive with Dario: A closer look at the new Arduino Nano 33 IoT | Arduino Blog." https://blog.arduino.cc/2019/05/24/getting-to-know-the-new-arduino-nano-33-iot/.

[69] "Arduino Nano 33 BLE — Arduino Official Store." https://store.arduino.cc/products/arduino-nano-33-ble.

[70] "Arduino Nano 33 BLE Sense — Arduino Online Shop." https://store-usa.arduino.cc/products/arduino-nano-33-ble-sense.

[71] "Arduino Every vs.Arduino 33: Comparación de placas Arduino." https://arduino.cl/arduino-every-vs-arduino-33-comparacion-de-placas-arduino/.

[72] "Arduino Nano 33 BLE v/s 33 BLE Sense v/s 33 IoT - Arduino Project Hub." https://create.arduino.cc/projecthub/thatiotguy/arduino-nano-33-ble-v-s-33-ble-sense-v-s-33-iot-07d2bf.

[73] "Arduino Nano RP2040 Connect with headers — Arduino Online Shop." https://store.arduino.cc/products/arduino-nano-rp2040-connect-with-headers?_gl=1%2A1jlk8ln%2A_ga%2ANzI1OTY4Mjc3LjE2MTUzOTM0NDc.%2A_ga_NEXN8H46L5%2AMTYzMDIyOTYzMC4zNy4xLjE2MzAyMjk2MzEuMA..

[74] "Al fin: Arduino Nano RP2040 Connect." https://solectroshop.com/es/blog/al-fin-arduino-nano-rp2040-connect--n86.

[75] "Development Boards | Espressif Systems." https://www.espressif.com/en/products/devkits.

[76] Az-Delivery, "AZ-Delivery ESP-32 Dev Kit C V2," *Az-Delivery*, vol. 20, no. 2. p. 2, 2017, doi: 10.25100/rc.v20i2.4609.

[77] "AZ-Delivery: su experto en microelectrónica." https://www.az-delivery.com/es.

[78] N. Cameron, *Electronics Projects with the ESP8266 and ESP32*. 2021.

[79] "Pulsímetro y oxímetro con Arduino y MAX30102." https://www.luisllamas.es/pulsimetro-y-oximetro-con-arduino-y-max30102/.

[80] "Grove - GSR Sensor - Seeed Wiki." https://wiki.seeedstudio.com/Grove-GSR_Sensor/.

[81] "La Wi-Fi para la Internet de las Cosas se llama 'Wi-Fi HaLow.'" https://cio.com.mx/la-wi-fi-para-la-internet-de-las-cosas-se-llama-wi-fi-halow/.

[82] "Bluetooth 5.0, novedades y diferencias con las anteriores versiones." https://hardzone.es/tutoriales/rendimiento/bluetooth-5-0-caracteristicas-novedades/.

[83] "Bluetooth | Aprendiendo Arduino." https://aprendiendoarduino.wordpress.com/category/bluetooth/.

[84] "Cómo saber la velocidad máxima que puede dar un router WiFi." https://www.xatakamovil.com/conectividad/como-saber-velocidad-maxima-que-puede-dar-router-wifi.

[85] "Eclipse Mosquitto." https://mosquitto.org/.

[86] "Features List | EMQ Docs."

https://docs.emqx.io/en/broker/v4.3/introduction/checklist.html.

[87] C. Ballesteros De Andrés, "Sistema IoT para la monitorización del estado de un centro de proceso de datos de grandes dimensiones," 2017.

[88] "Qué es Node-RED | Aprendiendo Arduino." https://aprendiendoarduino.wordpress.com/2020/03/05/que-es-node-red/.

[89] "ROG Strix G G531 | ROG Strix G G531 | Gaming Laptops | ROG - Republic of Gamers | ROG Global." https://rog.asus.com/laptops/rog-strix/rog-strix-g-g531-series/.

[90] "Nuevo Ubuntu 20.04.3 LTS, novedades y cómo descargar este Linux." https://www.softzone.es/noticias/open-source/ubuntu-20-04-3-lts-novedades-descargar-linux/.

[91] "test.mosquitto.org." http://test.mosquitto.org/.

[92] "MQTTLens - Chrome Web Store." https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm?hl=es.

[93] "ArduinoMqttClient/examples at master · arduino-libraries/ArduinoMqttClient." https://github.com/arduino-libraries/ArduinoMqttClient/tree/master/examples.

[94] "AQS/AirQualitySensorD1Mini.ino at main · SuperHouse/AQS." https://github.com/SuperHouse/AQS/blob/main/Firmware/AirQualitySensorD1Mini/AirQualitySensorD1Mini.ino (accessed Sep. 09, 2021).

[95] "Arduino: Usando la función millis() en lugar de delay() | Robots Didácticos." https://robots-argentina.com.ar/didactica/arduino-usando-la-funcion-millis-en-lugar-de-delay/.

[96] "speeding up analogread() at the Arduino Zero - Hardware / Arduino Zero - Arduino Forum." https://forum.arduino.cc/t/speeding-up-analogread-at-the-arduino-zero/426099.

[97] "Nano IOT 33 - analogRead() is Slower than expected - Nano Family / Nano 33 IoT - Arduino Forum." https://forum.arduino.cc/t/nano-iot-33-analogread-is-slower-than-expected/635209/2.

[98] "How to use ESP32 Dual Core with Arduino IDE." https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/.

[99] Miguel Ángel, "Configuración de una máquina VMware para internet y red privada por WIFI y LAN indistintamente | Espacio de Miguel Ángel." https://magomez4269.wordpress.com/2011/03/21/configuracion-de-una-maquina-vmware-para-internet-y-red-privada-por-wifi-y-lan-indistintamente/.

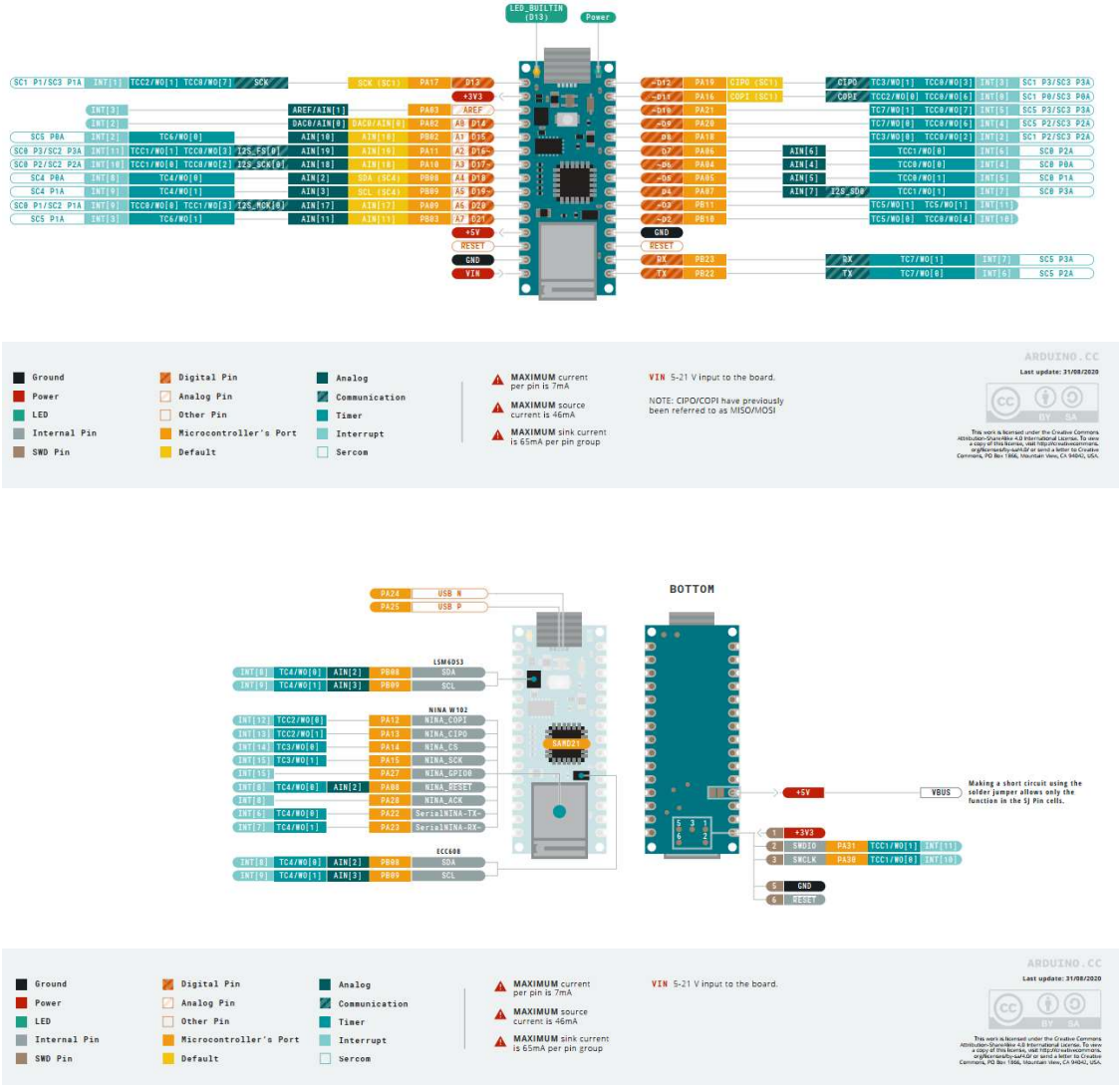
ANEXO I

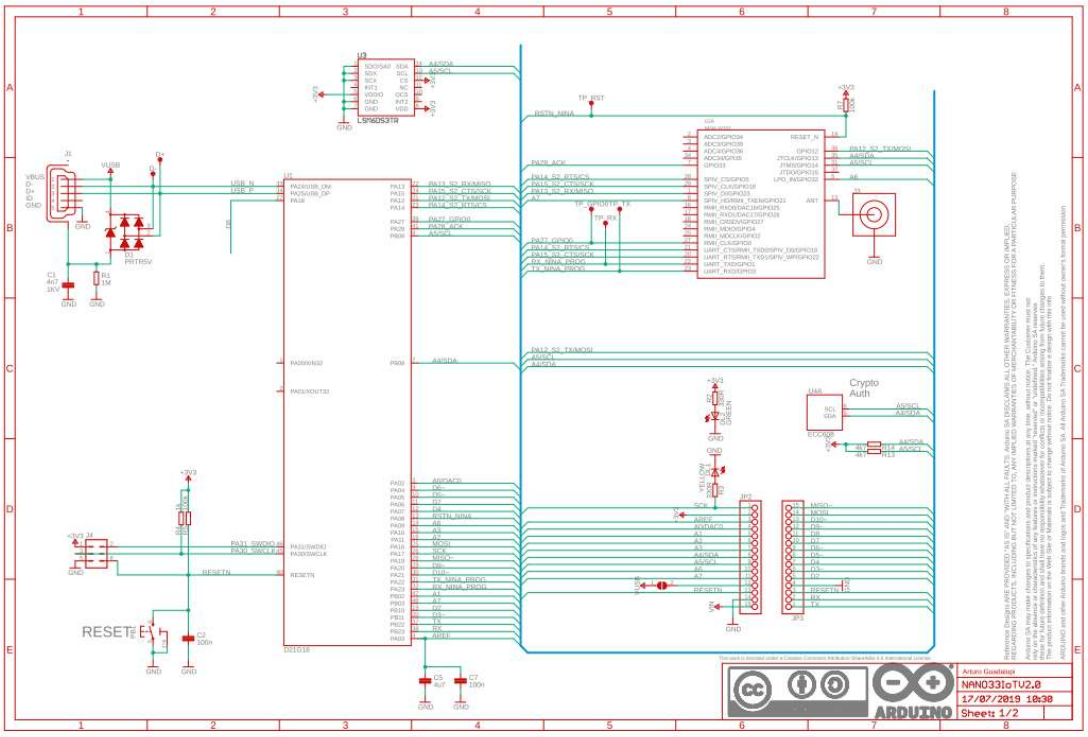
DOCUMENTACIÓN DE LOS COMPONENTES

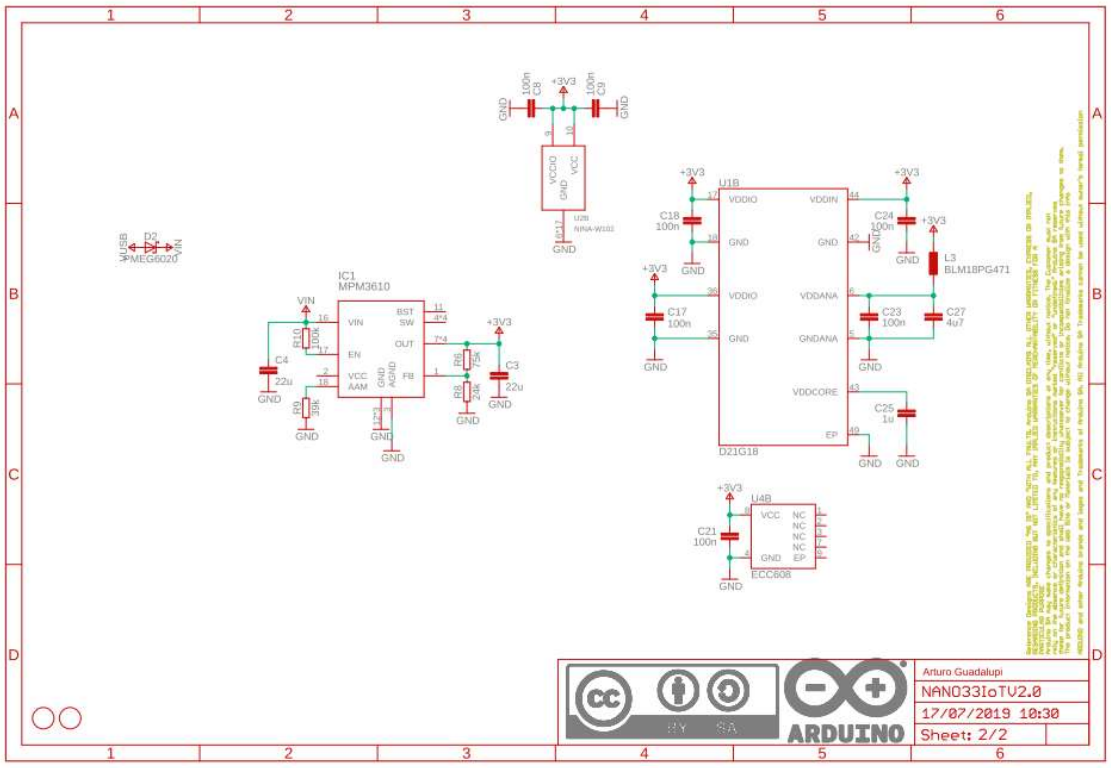
11 ANEXO I: DOCUMENTACIÓN DE LOS COMPONENTES

11.1 **Arduino Nano IoT 33**

Documentación del AN-33IoT obtenida en la tienda oficial de Arduino [17].

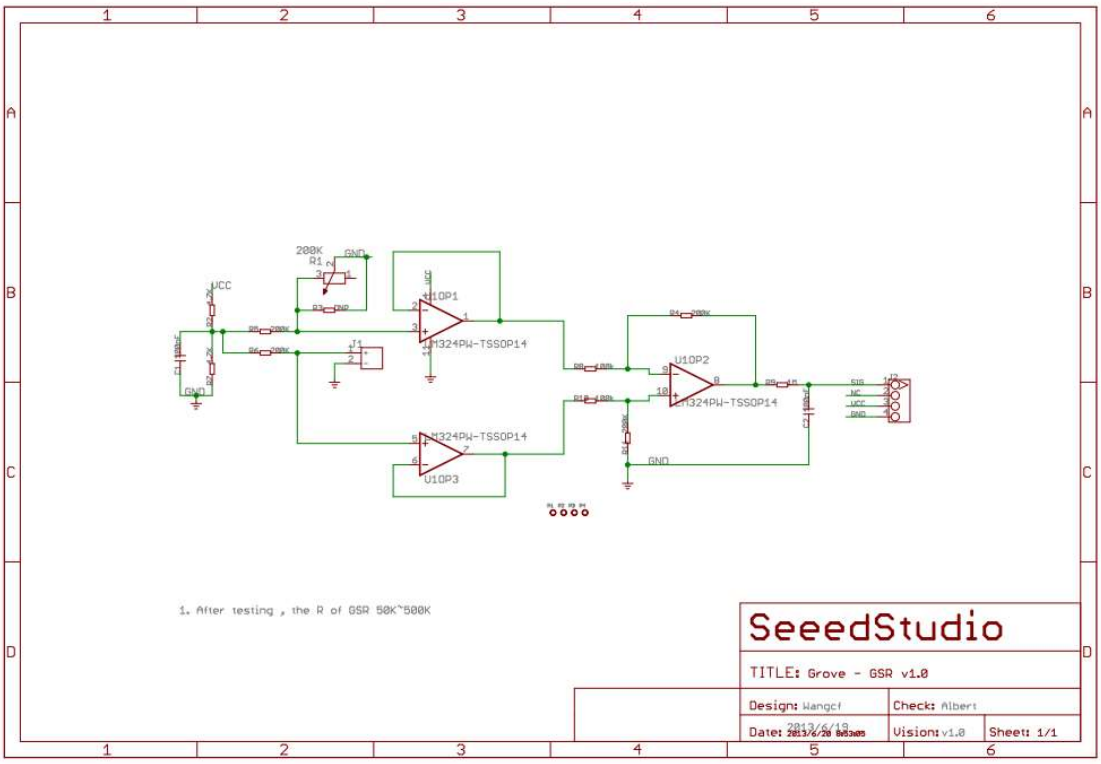



Crypto Auth
RESET
Arturo Guadalupi
NANO33IoTV2.0
17/07/2019 10:30
Sheet: 1/2
ARDUINO


IC1
MPM3610
D21G18
U4B
ECC608
Arturo Guadalupi
NANO33IoTV2.0
17/07/2019 10:30
Sheet: 2/2
ARDUINO

11.2 **Groove Sensor**

Documentación del sistema de medición Grove GSR obtenida en la tienda SeedStudio [80].



11.2.1 LM324

1 Features

- 2-kV ESD Protection for:
  - LM224K, LM224KA
  - LM324K, LM324KA
  - LM2902K, LM2902KV, LM2902KAV
- Wide Supply Ranges
  - Single Supply: 3 V to 32 V (26 V for LM2902)
  - Dual Supplies: ±1.5 V to ±16 V (±13 V for LM2902)
- Low Supply-Current Drain Independent of Supply Voltage: 0.8 mA Typical
- Common-Mode Input Voltage Range Includes Ground, Allowing Direct Sensing Near Ground
- Low Input Bias and Offset Parameters
  - Input Offset Voltage: 3 mV Typical
    A Versions: 2 mV Typical
  - Input Offset Current: 2 nA Typical
  - Input Bias Current: 20 nA Typical
    A Versions: 15 nA Typical
- Differential Input Voltage Range Equal to Maximum-Rated Supply Voltage: 32 V (26 V for LM2902)
- Open-Loop Differential Voltage Amplification: 100 V/mV Typical
- Internal Frequency Compensation
- On Products Compliant to MIL-PRF-38535, All Parameters are Tested Unless Otherwise Noted. On All Other Products, Production Processing Does Not Necessarily Include Testing of All Parameters.

2 Applications

- Blu-ray Players and Home Theaters
- Chemical and Gas Sensors
- DVD Recorders and Players
- Digital Multimeter: Bench and Systems
- Digital Multimeter: Handhelds
- Field Transmitter: Temperature Sensors
- Motor Control: AC Induction, Brushed DC, Brushless DC, High-Voltage, Low-Voltage, Permanent Magnet, and Stepper Motor
- Oscilloscopes
- TV: LCD and Digital
- Temperature Sensors or Controllers Using Modbus
- Weigh Scales

3 Description

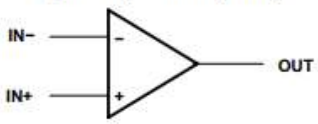
These devices consist of four independent high-gain frequency-compensated operational amplifiers that are designed specifically to operate from a single supply or split supply over a wide range of voltages.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LMx24, LMx24x, LMx24xx, LM2902, LM2902x, LM2902xx, LM2902xxx	SOIC (14)	8.65 mm × 3.91 mm
	CDIP (14)	19.56 mm × 6.67 mm
	PDIP (14)	19.30 mm × 6.35 mm
	CFP (14)	9.21 mm × 5.97 mm
	TSSOP (14)	5.00 mm × 4.40 mm
	SO (14)	9.20 mm × 5.30 mm
	SSOP (14)	6.20 mm × 5.30 mm
LM124, LM124A	LCCC (20)	8.90 mm × 8.90 mm

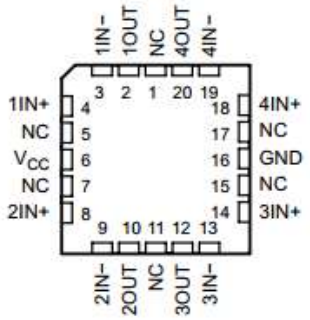
(1) For all available packages, see the orderable addendum at the end of the data sheet.

Symbol (Each Amplifier)

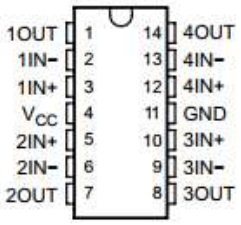


5 Pin Configuration and Functions

FK Package
20-Pin LCCC
(Top View)



D, DB, J, N, NS, PW, W
14-Pin SOIC, SSOP, CDIP, PDIP, SO, TSSOP, CFP
(Top View)



Pin Functions

PIN			I/O	DESCRIPTION
NAME	LCCC NO.	SOIC, SSOP, CDIP, PDIP, SO, TSSOP, CFP NO.		
1IN–	3	2	I	Negative input
1IN+	4	3	I	Positive input
1OUT	2	1	O	Output
2IN–	9	6	I	Negative input
2IN+	8	5	I	Positive input
2OUT	10	7	O	Output
3IN–	13	9	I	Negative input
3IN+	14	10	I	Positive input
3OUT	12	8	O	Output
4IN–	19	13	I	Negative input
4IN+	18	12	I	Positive input
4OUT	20	14	O	Output
GND	16	11	—	Ground
NC	1	—	—	Do not connect
	5			
	7			
	11			
	15			
	17			
V_{CC}	6	4	—	Power supply

11.3 **MAX30100**

MAX30100

Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health

General Description

The MAX30100 is an integrated pulse oximetry and heart-rate monitor sensor solution. It combines two LEDs, a photodetector, optimized optics, and low-noise analog signal processing to detect pulse oximetry and heart-rate signals.

The MAX30100 operates from 1.8V and 3.3V power supplies and can be powered down through software with negligible standby current, permitting the power supply to remain connected at all times.

Applications

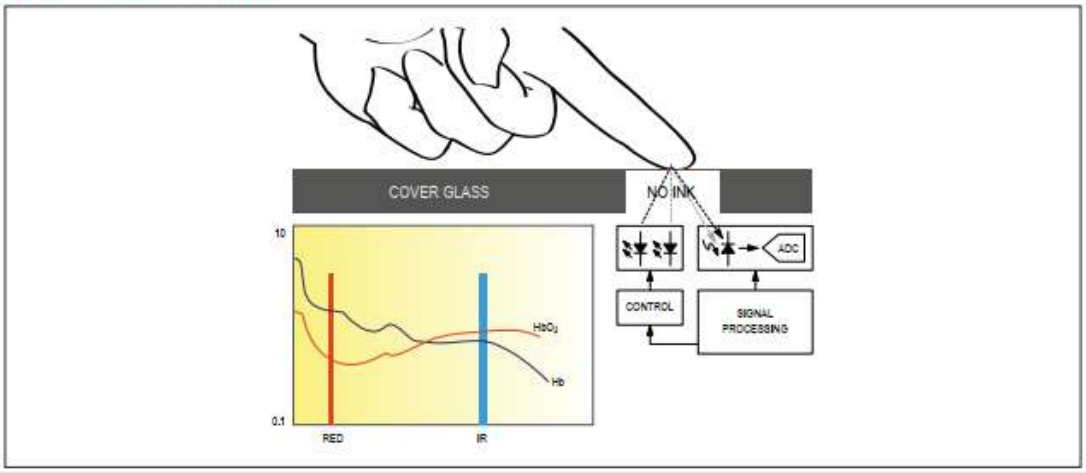
- Wearable Devices
- Fitness Assistant Devices
- Medical Monitoring Devices

Benefits and Features

- Complete Pulse Oximeter and Heart-Rate Sensor Solution Simplifies Design
  - Integrated LEDs, Photo Sensor, and High-Performance Analog Front -End
  - Tiny 5.6mm x 2.8mm x 1.2mm 14-Pin Optically Enhanced System-in-Package
- Ultra-Low-Power Operation Increases Battery Life for Wearable Devices
  - Programmable Sample Rate and LED Current for Power Savings
  - Ultra-Low Shutdown Current (0.7µA, typ)
- Advanced Functionality Improves Measurement Performance
  - High SNR Provides Robust Motion Artifact Resilience
  - Integrated Ambient Light Cancellation
  - High Sample Rate Capability
  - Fast Data Output Capability

Ordering Information ***appears at end of data sheet.***

System Block Diagram



Absolute Maximum Ratings

V_{DD} to GND .....-0.3V to +2.2V
GND to PGND.....-0.3V to +0.3V
x_DRV, x_LED+ to PGND.....-0.3V to +6.0V
All Other Pins to GND.....-0.3V to +6.0V
Output Short-Circuit Current Duration.....Continuous
Continuous Input Current into Any Terminal .....±20mA

Continuous Power Dissipation (T_A = +70°C)
OESIP (derate 5.8mW/°C above +70°C) 464mW
Operating Temperature Range..... -40°C to +85°C
Soldering Temperature (reflow).....+260°C
Storage Temperature Range ..... -40°C to +105°C

Package Thermal Characteristics (Note 1)

OESIP
Junction-to-Ambient Thermal Resistance (θ_{JA}) .....150°C/W
Junction-to-Case Thermal Resistance (θ_{JC}).....170°C/W

Note 1: Package thermal resistances were obtained using the method described in JEDEC specification JESD51-7, using a four-layer board. For detailed information on package thermal considerations, refer to www.maximintegrated.com/thermal-tutorial.

Electrical Characteristics

(V_{DD} = 1.8V, V_{IR_LED+} = V_{R_LED+} = 3.3V, T_A = +25°C, min/max are from T_A = -40°C to +85°C, unless otherwise noted.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
POWER SUPPLY							
Power-Supply Voltage	V_{DD}	Guaranteed by RED and IR count tolerance		1.7	1.8	2.0	V
LED Supply Voltage (R_LED+ or IR_LED+ to PGND)	V_{LED+}	Guaranteed by PSRR of LED Driver		3.1	3.3	5.0	V
Supply Current	I_{DD}	SpO_2 and heart rate modes, PW = 200µs, 50sps			600	1200	µA
		Heart rate only mode, PW = 200µs, 50sps			600	1200	
Supply Current in Shutdown	I_{SHDN}	T_A = +25°C, MODE = 0x80			0.7	10	µA
SENSOR CHARACTERISTICS							
ADC Resolution					14		bits
Red ADC Count (Note 3)	RED_C	Propriety ATE setup RED_PA = 0x05, LED_PW = 0x00, SPO2_SR = 0x07, T_A = +25°C		23,000	26,000	29,000	Counts
IR ADC Count (Note 3)	IR_C	Propriety ATE setup IR_PA = 0x09, LED_PW = 0x00, SPO2_SR = 0x07, T_A = +25°C		23,000	26,000	29,000	Counts
Dark Current Count	DC_C	RED_PA = IR_PA = 0x00, LED_PW = 0x03, SPO2_SR = 0x01			0	3	Counts
DC Ambient Light Rejection (Note 4)	ALR	Number of ADC counts with finger on sensor under direct sunlight (100K lux) LED_PW = 0x03, SPO2_SR = 0x01	RED LED		0		Counts
			IR LED		0		

Electrical Characteristics (continued)

(V_{DD} = 1.8V, V_{IR_LED+} = V_{R_LED+} = 3.3V, T_A = +25°C, min/max are from T_A = -40°C to +85°C, unless otherwise noted.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
DIGITAL CHARACTERISTICS (SDA, SDA, $\overline{INT}$)						
Output Low Voltage SDA, $\overline{INT}$	V_{OL}	I_{SINK} = 6mA			0.4	V
I²C Input Voltage Low	V_{IL_I2C}	SDA, SCL			0.4	V
I²C Input Voltage High	V_{IH_I2C}	SDA, SCL	1.4			V
Input Hysteresis	V_{HYS}	SDA, SCL		200		mV
Input Capacitance	C_{IN}	SDA, SCL		10		pF
Input Leakage Current	I_{IN}	V_{IN} = 0V, T_A = +25°C (SDA, SCL, INT)		0.01	1	µA
		V_{IN} = 5.5V, T_A = +25°C (SDA, SCL, INT)		0.01	1	µA
I²C TIMING CHARACTERISTICS (SDA, SDA, $\overline{INT}$)						
I²C Write Address				AE		Hex
I²C Read Address				AF		Hex
Serial Clock Frequency	f_{SCL}		0		400	kHz
Bus Free Time Between STOP and START Conditions	t_{BUF}		1.3			µs
Hold Time (Repeated) START Condition	$t_{HD,START}$		0.6			µs
SCL Pulse-Width Low	t_{LOW}		1.3			µs
SCL Pulse-Width High	t_{HIGH}		0.6			µs
Setup Time for a Repeated START Condition	$t_{SU,START}$		0.6			µs
Data Hold Time	$t_{HD,DAT}$		0		900	ns
Data Setup Time	$t_{SU,DAT}$		100			ns
Setup Time for STOP Condition	$t_{SU,STOP}$		0.6			µs
Pulse Width of Suppressed Spike	t_{SP}		0		50	ns
Bus Capacitance	C_B				400	pF
SDA and SCL Receiving Rise Time	t_R		$20 + 0.1C_B$		300	ns
SDA and SCL Receiving Fall Time	t_{RF}		$20 + 0.1C_B$		300	ns
SDA Transmitting Fall Time	t_{TF}		$20 + 0.1C_B$		300	ns

Note 2: All devices are 100% production tested at T_A = +25°C. Specifications over temperature limits are guaranteed by Maxim Integrated's bench or proprietary automated test equipment (ATE) characterization.

Note 3: Specifications are guaranteed by Maxim Integrated's bench characterization and by 100% production test using proprietary ATE setup and conditions.

Note 4: For design guidance only. Not production tested.

Electrical Characteristics (continued)

(V_{DD} = 1.8V, V_{IR_LED+} = V_{R_LED+} = 3.3V, T_A = +25°C, min/max are from T_A = -40°C to +85°C, unless otherwise noted.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
IR ADC Count—PSRR (V_{DD})	$PSRR_{VDD}$	Propriety ATE setup 1.7V < V_{DD} < 2.0V, LED_PW = 0x03, SPO2_SR = 0x01, IR_PA = 0x09, IR_PA = 0x05, T_A = +25°C		0.25	2	%
		Frequency = DC to 100kHz, 100mV_{P-P}		10		LSB
RED/IR ADC Count—PSRR (X_LED+)	$PSRR_{LED}$	Propriety ATE setup 3.1V < X_LED+ < 5V, LED_PW = 0x03, SPO2_SR = 0x01, IR_PA = 0x09, IR_PA = 0x05, T_A = +25°C		0.05	2	%
		Frequency = DC to 100kHz, 100mV_{P-P}		10		LSB
ADC Integration Time	INT	LED_PW = 0x00		200		µs
		LED_PW = 0x03		1600		µs
IR LED CHARACTERISTICS (Note 4)						
LED Peak Wavelength	λ_P	I_{LED} = 20mA, T_A = +25°C	870	880	900	nm
Full Width at Half Max	$\Delta\lambda$	I_{LED} = 20mA, T_A = +25°C		30		nm
Forward Voltage	V_F	I_{LED} = 20mA, T_A = +25°C		1.4		V
Radiant Power	P_O	I_{LED} = 20mA, T_A = +25°C		6.5		mW
RED LED CHARACTERISTICS (Note 4)						
LED Peak Wavelength	λ_P	I_{LED} = 20mA, T_A = +25°C	650	660	670	nm
Full Width at Half Max	$\Delta\lambda$	I_{LED} = 20mA, T_A = +25°C		20		nm
Forward Voltage	V_F	I_{LED} = 20mA, T_A = +25°C		2.1		V
Radiant Power	P_O	I_{LED} = 20mA, T_A = +25°C		9.8		mW
TEMPERATURE SENSOR						
Temperature ADC Acquisition Time	T_T	T_A = +25°C		29		ms
Temperature Sensor Accuracy	T_A	T_A = +25°C		±1		°C
Temperature Sensor Minimum Range	T_{MIN}			-40		°C
Temperature Sensor Maximum Range	T_{MAX}			85		°C

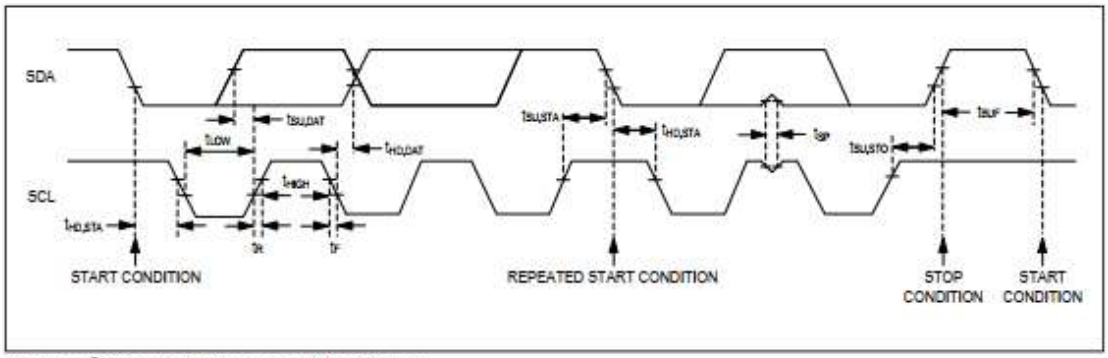


Figure 1. I²C-Compatible Interface Timing Diagram

ANEXO II

IMPLEMENTACIÓN DE LOS CÓDIGOS

12 ANEXO II: IMPLEMENTACIÓN DE LOS CÓDIGOS

12.1 **GICI.ino**

```
//Librerias para garantizar el funcionamiento del Wifi y el
funcionamiento del servidor MQTT
#include <WiFiNINA.h>
#include <PubSubClient.h>

WiFiClient wifiClient;
PubSubClient client(wifiClient);

//Definición de la red Wifi

const char* ssid = "Nombre de la red";
const char* password = "Contraseña";
const char* mqttServer = ""; // Dirección ip del servidor

//Definir las variables que vamos a utilizar.
int sensorGSR;
char v[10];//Vector para hacer el cambio de variable


// Establecemos la conexión Wifi
void setup_wifi()
{
   delay(10);

   Serial.println();
   Serial.print("Connecting to wifi ");
   Serial.println(ssid);

   WiFi.begin(ssid, password);

   while (WiFi.status() != WL_CONNECTED)
   {
     delay(500);
     Serial.print(".");
   }

   randomSeed(micros());

   Serial.println("");
   Serial.print("WiFi connected: ");
   Serial.println(ssid);
   Serial.println("IP address: ");
   Serial.println(WiFi.localIP());
}
void callback(char* topic, byte* payload, unsigned int length) {
   Serial.print("Message arrived [");
   Serial.print(topic);
   Serial.print("] ");
   for (int i = 0; i < length; i++) {
     Serial.print((char)payload[i]);
   }
   Serial.println();
}

// En caso de perder la conexion reconectar
void reconnect() {
// Bucle hasta que nos conectamos
   while (!client.connected()) {
     Serial.print("Attempting MQTT connection...");
     // Create a random client ID
     String clientId = "IdentificaciÓndelcliente";
     clientId += String(random(0xffff), HEX);
     // Intento de conexión
     if (client.connect(clientId.c_str())) {
       Serial.println("conectado");
       // Una vez conectado enviamos un mensaje de que nos hemos conectado a los 3 temas en cuestion
       client.publish("tema", "CONECTADO:");
       // ... and resubscribe
       client.subscribe("tema");//Tema al que nos vamos a subscribir para enviar la información
     } else {
       Serial.print("failed, rc=");
       Serial.print(client.state());
       Serial.println(" vuelve a intentar en 0.5s");
       // Esperar 0.5 segundos antes de volver a intentar conectar
       delay(500);
     }
   }
}
```

```
void setup() {
   // put your setup code here, to run once:
   //Velocidad de los baudios del puerto serial
   Serial.begin(velocidad deseada);

   //Inicializamos la conexion wifi y la conexion con el servidor
   setup_wifi();
   client.setServer(mqttServer, 1883);
   client.setCallback(callback);
}


void loop() {
   // put your main code here, to run repeatedly:
   // Comprobar la conexion al servidor. EN caso de no estar conectado
ir a la funcion reconnect y hasta que no se conecta no volver al
codigo principal
   if (!client.connected())
   {
     reconnect();
   }
   client.loop();

   //Leemos los valores del sensor GSR

    sensorGSR=analogRead(pin al que se ha conectado);

//Pasamos los numeros a un vector de char para poder enviarlos al
servidor para esto se usa el comando itoa();


client.publish("nombredeltema",vectordondeestanlosdatos);// Enviamos
el dato al tema en cuestión
```

12.2 **WiFiScan.ino**

```
/* SCANNING WIFI HOTSPOT
 *  We can access a WiFi hotspot if we know the WiFi SSID name.
 *  We'll scan an existing WiFi SSID and then print the list on the serial terminal
 *
 *  Obtenido del libro: "Beggining Arduino 33 IoT Projects"
 */


// Libraries
#include <SPI.h>
#include <WiFiNINA.h>
//#include <ESP8266WiFi.h>

//SSID and Password of your WiFi router
const char* ssid = "MiFibra-93C7";
const char* password = "gKxFRET6";

// Global var
int led = 13;

void setup() {
  Serial.begin(115200);
  pinMode(led, OUTPUT);

  // Check for the WiFi module
  if (WiFi.status() == WL_NO_MODULE){
    Serial.println("Communication with WiFi module failed !");
    // don't continue
    while(true);
  }

  // WiFi
  scanWiFi();
  connectToWiFi();

}

void loop() {

}

/*
 * PROCEDURES AND FUNCTIONS
 */

// PROCEDURE - SCANWIFI
void scanWiFi(){
  Serial.print("Scanning...");
  byte ssid = WiFi.scanNetworks();

  Serial.print("found ");
  Serial.println(ssid);

  for (int i = 0; i<ssid; i++){
    Serial.print(">> ");
    Serial.print(WiFi.SSID(i));       // Nombre de la red WiFi
    Serial.print("\t\tRSSI: ");
    Serial.print(WiFi.RSSI(i));       // Fuerza de la red
    Serial.print(" dBm");
    Serial.print("\tEncryption: ");   // Tipo de encriptacion
    Serial.println(WiFi.encryptionType(i));
  }

  Serial.println("");
  Serial.println("");
}

// PROCEDURE - CONNECT TO WIFI
void connectToWiFi(){
```

```
  // Connect to WiFi Network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to Wifi");
  Serial.println("...");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  int retries = 0;

  while ((WiFi.status() != WL_CONNECTED) && (retries < 15)){
    retries++;
    delay(500);
    Serial.print(".");
  }

  if (retries > 14){
    Serial.println(F("WiFi connection FAILED"));
  }

  if (WiFi.status() == WL_CONNECTED){
    Serial.println("WiFi connected!");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
  }
  Serial.println("Setup ready");
}
```

12.3 **Oxymetro_y_pulsimetro.ino**

```
//  Fuente: https://forum.arduino.cc/t/sensor-max30100-y-pantalla-tft-ili9486/693174

#include "MAX30100_PulseOximeter.h"

#define REPORTING_PERIOD_MS     1000

PulseOximeter pox;
uint32_t tsLastReport = 0;

float HR = 0.0;
int spo2 = 0;

void onBeatDetected()
{
    Serial.println("Beat!");
}

void setup()
{
    Serial.begin(115200);
    Serial.print("Initializing pulse oximeter..");

    // Initialize the PulseOximeter instance and register a beat-detected callback
    // The parameter passed to the begin() method changes the samples flow that
    // the library spews to the serial.
    // Options: (R: lecturas)
    //  * PULSEOXIMETER_DEBUGGINGMODE_PULSEDETECT : filtered samples and beat detection threshold
    //  * PULSEOXIMETER_DEBUGGINGMODE_RAW_VALUES : sampled values coming from the sensor, with no processing
    //  * PULSEOXIMETER_DEBUGGINGMODE_AC_VALUES : sampled values after the DC removal filter

    // Initialize the PulseOximeter instance
    // Failures are generally due to an improper I2C wiring, missing power supply
    // or wrong target chip
    if (!pox.begin()) {
        Serial.println("FAILED");
        for(;;);
    } else {
        Serial.println("SUCCESS");
    }

    // The default current for the IR LED is 50mA and it could be changed
    //   by uncommenting the following line. Check MAX30100_Registers.h for all the
    //   available options.
     pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);

    // Register a callback for the beat detection
    pox.setOnBeatDetectedCallback(onBeatDetected);
}

void loop()
{
    // Make sure to call update as fast as possible
    pox.update();
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        Serial.print("Heart rate:");
        HR = pox.getHeartRate();
        Serial.print(HR);
        Serial.print("bpm / SpO2:");
        spo2 = pox.getSpO2();
        Serial.print(spo2);
        Serial.println("%");


        tsLastReport = millis();
    }
}
```

12.4 **GSR.ino**

```
// https://wiki.seeedstudio.com/Grove-GSR_Sensor/


// Variables globales
const int GSR=A0;    // Pin con el que se realiza la medida
int sensorValue=0;   // Inicializacion de la variable a cero.
int gsr_average=0;   // Inicializacion de la variable a cero.
int imedidas = 10;   // Numero de medidas que hace para hacer la media

// Setup
void setup(){
  Serial.begin(9600);
}


// Loop
void loop(){

  long sum=0;  // Var local

  for(int i=0;i<imedidas;i++)           //Average the 10 measurements to remove the glitch
      {
      sensorValue=analogRead(GSR);
      sum += sensorValue;
      delay(5);
      }
   // Realizar media
   gsr_average = sum/imedidas;
   // Mostrar por pantalla
   Serial.println(gsr_average);

}
```

12.5 **WiFiSimpleSender.ino**

```
/*
  ArduinoMqttClient - WiFi Simple Sender

  This example connects to a MQTT broker and publishes a message to
  a topic once a second.

  The circuit:
  - Arduino MKR 1000, MKR 1010, nano 33 IoT or Uno WiFi Rev.2 board

  This example code is in the public domain.
*/

#include <ArduinoMqttClient.h>
#if defined(ARDUINO_SAMD_MKRWIFI1010) || defined(ARDUINO_SAMD_NANO_33_IOT) || defined(ARDUINO_AVR_UNO_WIFI_REV2)
  #include <WiFiNINA.h>
#elif defined(ARDUINO_SAMD_MKR1000)
  #include <WiFi101.h>
#elif defined(ARDUINO_ESP8266_ESP12)
  #include <ESP8266WiFi.h>
#endif

#include "arduino_secrets.h"
///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID;        // your network SSID (name)
char pass[] = SECRET_PASS;    // your network password (use for WPA, or use as key for WEP)

// To connect with SSL/TLS:
// 1) Change WiFiClient to WiFiSSLClient.
// 2) Change port value from 1883 to 8883.
// 3) Change broker value to a server with a known SSL/TLS root certificate
//    flashed in the WiFi module.

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

//const char broker[] = "test.mosquitto.org";
const char broker[] = "192.168.1.37";     // IP address of your MQTT broker
int        port     = 1883;
const char topic[]  = "arduino/simple";  // TOPIC

const long interval = 1000;
unsigned long previousMillis = 0;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);
  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    // failed, retry
    Serial.print(".");
    delay(5000);
  }

  Serial.println("You're connected to the network");
  Serial.println();

  // You can provide a unique client ID, if not set the library uses Arduino-millis()
  // Each client must have a unique client ID
  // mqttClient.setId("clientId");

  // You can provide a username and password for authentication
  // mqttClient.setUsernamePassword("username", "password");

  Serial.print("Attempting to connect to the MQTT broker: ");
  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {
    Serial.print("MQTT connection failed! Error code = ");
    Serial.println(mqttClient.connectError());

    while (1);
  }

  Serial.println("You're connected to the MQTT broker!");
  Serial.println();
}

void loop() {
  // call poll() regularly to allow the library to send MQTT keep alives which
  // avoids being disconnected by the broker
```

```
  mqttClient.poll();

  // avoid having delays in loop, we'll use the strategy from BlinkWithoutDelay
  // see: File -> Examples -> 02.Digital -> BlinkWithoutDelay for more info
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time a message was sent
    previousMillis = currentMillis;

    Serial.print("Sending message to topic: ");
    Serial.println(topic);
    Serial.print("hello ");
    Serial.println(count);

    // send message, the Print interface can be used to set the message contents
    mqttClient.beginMessage(topic);
    mqttClient.print("hello ");
    mqttClient.print(count);
    mqttClient.endMessage();

    Serial.println();

    count++;
  }
}
```

12.5.1 Arduino_secrets.h

```
#define SECRET_SSID "YOUR_SSID"
#define SECRET_PASS "YOUR_PASS"
```

12.6 **WiFiSimpleReciever.ino**

```
/*
  ArduinoMqttClient - WiFi Simple Receive

  This example connects to a MQTT broker and subscribes to a single topic.
  When a message is received it prints the message to the serial monitor.

  The circuit:
  - Arduino MKR 1000, MKR 1010 or Uno WiFi Rev.2 board

  This example code is in the public domain.
*/

#include <ArduinoMqttClient.h>
#if defined(ARDUINO_SAMD_MKRWIFI1010) || defined(ARDUINO_SAMD_NANO_33_IOT) || defined(ARDUINO_AVR_UNO_WIFI_REV2)
  #include <WiFiNINA.h>
#elif defined(ARDUINO_SAMD_MKR1000)
  #include <WiFi101.h>
#elif defined(ARDUINO_ESP8266_ESP12)
  #include <ESP8266WiFi.h>
#endif

#include "arduino_secrets.h"
///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID;        // your network SSID (name)
char pass[] = SECRET_PASS;    // your network password (use for WPA, or use as key for WEP)

// To connect with SSL/TLS:
// 1) Change WiFiClient to WiFiSSLClient.
// 2) Change port value from 1883 to 8883.
// 3) Change broker value to a server with a known SSL/TLS root certificate
//    flashed in the WiFi module.

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "test.mosquitto.org";
//const char broker[] = "192.168.1.33"; // ip RPi   BROKER
int        port     = 1883;
const char topic[]  = "TFM/Sergio";        // TOPIC

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


// attempt to connect to Wifi network:
Serial.print("Attempting to connect to WPA SSID: ");
Serial.println(ssid);
while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
  // failed, retry
  Serial.print(".");
  delay(5000);
}

Serial.println("You're connected to the network");
Serial.println();

// You can provide a unique client ID, if not set the library uses Arduino-millis()
// Each client must have a unique client ID
// mqttClient.setId("clientId");

// You can provide a username and password for authentication
// mqttClient.setUsernamePassword("username", "password");

Serial.print("Attempting to connect to the MQTT broker: ");
Serial.println(broker);

if (!mqttClient.connect(broker, port)) {
  Serial.print("MQTT connection failed! Error code = ");
  Serial.println(mqttClient.connectError());

  while (1);
}

Serial.println("You're connected to the MQTT broker!");
Serial.println();

Serial.print("Subscribing to topic: ");
Serial.println(topic);
Serial.println();

// subscribe to a topic
mqttClient.subscribe(topic);

// topics can be unsubscribed using:
// mqttClient.unsubscribe(topic);
```

```
  Serial.print("Waiting for messages on topic: ");
  Serial.println(topic);
  Serial.println();
}

void loop() {
  int messageSize = mqttClient.parseMessage();
  if (messageSize) {
    // we received a message, print out the topic and contents
    Serial.print("Received a message with topic '");
    Serial.print(mqttClient.messageTopic());
    Serial.print("', length ");
    Serial.print(messageSize);
    Serial.println(" bytes:");

    // use the Stream interface to print the contents
    while (mqttClient.available()) {
      Serial.print((char)mqttClient.read());
    }
    Serial.println();

    Serial.println();
  }
}
```

12.6.1 Arduino_secrets.h

```
#define SECRET_SSID "YOUR_SSID"
#define SECRET_PASS "YOUR_PASS"
```

12.7 **TFM_v2.ino**

```
/*
 * Algunas ideas tomadas de:
 * SUPERHOUSE AUTOMATION (http://www.superhouse.tv)
 * https://github.com/SuperHouse/AQS/blob/main/Firmware/AirQualitySensorD1Mini/AirQualitySensorD1Mini.ino
 *
 * Ultima fecha de modificacion: 21/07/2021
 */

 /*
  * TODO:
  *       Introducir libraria time.h .Da igual que el setTime no sea exacto, lo quieres para medir tiempos.
  *       Revisar si se envian todos los paquetes. Elevar a QoS 2.
  *
  *       Probar a cambiar a float las var que se quieren enviar a grafana. Sino cambiar mean to count.
  *
  *       Ver si hay alguna forma de convertir decima a HEX u otro codig que reduzca el peso del paquete.
  */

  /*
   * MODIFICADO:
   *
   *      Añadido g_topicSize
   */

   /*
    * OBSERVACIONES:
    *
    *     El TAMAÑO máximo de la cadena sera de 256 bytes (0-255). Esto implica que
    *     el tamaño de mysize maximo es de 255.
    */
#define VERSION "1.4"

/*-------------------- Configuration --------------------------*/
// Configuration should be done in the included file:
#include "configuration.h";

/*------------------- Libraries -----------------------------*/
#include <WiFiNINA.h>
#include <PubSubClient.h>
/*
 *  TODO : Buscar una forma de incluir libreria de tiempo.
 *  Necesita que lepases la hora. Utilizar datos subMqtt y  Node-red ??
 */

/*------------------ Global Variables ----------------------*/
// g_: for global

// SENSORS

#define g_mysize 30                         // Tamaño del vector de datos
byte g_data[g_mysize];                      // Vector de datos

bool g_GSR_readings_taken     = false;      // true/fasle: whether any readings have been taken
String g_GSR_value            = "";         // Value for sensor GSR
unsigned long g_GSR_time       = 0;         // Value for sensor GSR time
int g_pin_GSR                 = A0;         // Analog pin on NN33IoT
bool g_GSR_state_start        = 0;          // Timestamp when GSR state last changed


// MQTT
# define g_topicSize 255                    // Tamaño del string de los topics. Peso en bytes del paquete.
char g_mqtt_message_buffer[255];            // General purpose buffer for MQTT messages
char g_command_topic[g_topicSize];          // MQTT topic for receiving commands

#if REPORT_MQTT_SEPARATE
char g_GSR_mqtt_topic[g_topicSize];                 // MQTT topic for reporting GSR values
char g_time_mqtt_topic[g_topicSize];                // MQTT topic for reporting GRS time value
#endif

#if REPORT_MQTT_JSON
char g_mqtt_json_topic[g_topicSize];                // MQTT topic for reporting all values using JSON
#endif

// WiFi
#define WIFI_CONNECT_INTERVAL       500     // Wait 500ms intervals for wifi connection
#define WIFI_CONNECT_MAX_ATTEMPTS   10      // Number of attempts/intervals to wait

// Time
//time_t g_time;                              // Variable tiempo

//General
char* g_cliente_ID;                       // Simulate chipID

unsigned long g_tStart;                     // Initial time for measuring time spent
unsigned long g_tEnd;                       // End time for measuring time spent
```

```
/*------------------- Instantiate Global Objects ----------------*/
// MQTT
WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

/*--------------------- Function Signatures ------------------------*/ // Declaration
void mqttCallback(char* topic, byte* payload, unsigned int len);    // Done
bool initWiFi();                                                    // Done
void reconnectMqtt();                                               // Done
void updateReadings();                                              // Done
void reportToMqtt();                                                // Done
void reportToSerial();                                              // Done
String NewClientID();                                               // Done
void showTimeNeeded();                                              // Done

/* ------------------ Program --------------------------------------*/
/*
 * SETUP
 */

void setup()
{
  delay(1000); // quitar

  // Initialize connection
  Serial.begin(SERIAL_BAUD_RATE);
  Serial.println();
  Serial.print("Sensor GSR starting up, v");
  Serial.println(VERSION);

  // We need a unique device ID for our MQTT client connection
  // g_cliente_ID = NewClientID();
  g_cliente_ID = "IDSPV001";       // Get ID. Revisar TODO.


  /*------------------TOPICS--------------------------------------------------------------------------------*/
  // Set up the topics for publishing sensor readings. By inserting the unique ID,
  // the result is of the form: "arduino_1/GSR/valor" etc
  sprintf(g_command_topic, "cmnd/%s/COMMAND", g_cliente_ID);     // For receiving commands

  #if REPORT_MQTT_SEPARATE
    sprintf(g_GSR_mqtt_topic, "arduino_1/%s/GSR/valor", g_cliente_ID);    // Data form AN33IoT
    sprintf(g_time_mqtt_topic, "arduino_1/%s/GSR/tiempo", g_cliente_ID);    // Data form AN33IoT
  #endif

  #if REPORT_MQTT_JSON
    sprintf(g_mqtt_json_topic, "arduino_1/%s/SENSOR", g_cliente_ID);   // Data from AN33IoT
  #endif


 // Report the MQTT topics to the serial console
 Serial.println(g_command_topic);        // For receiving messages
 #if REPORT_MQTT_SEPARATE
   Serial.println("MQTT topics: ");
   Serial.println(g_GSR_mqtt_topic);      // From AN33IoT
   Serial.println(g_time_mqtt_topic);     // From AN33IoT
 #endif

 #if REPORT_MQTT_JSON
   Serial.println(g_mqtt_json_topic);    // From AN33IoT
 #endif

 // Connect to WiFi
 if (initWiFi())
 {
   Serial.print("WiFi connected");
   Serial.println(ssid);
   Serial.print("IP local address: ");
   Serial.println(WiFi.localIP());
 }
 else
 {
   Serial.print("WiFi FAILED");
 }
 delay(100);

 /* Set up the MQTT client */
 mqttClient.setServer(mqtt_broker, port);
 mqttClient.setCallback(mqttCallback);
 mqttClient.setBufferSize(255);              // Tamaño del paquete que puede enviar MQTT por defecto 128

 delay(1000); // quitar
}

/*
 * ------------------- MAIN LOOP ---------------------
 */
```

```
void loop() {
  if (WiFi.status() == WL_CONNECTED)
  {
    if (!mqttClient.connected())
    {
      reconnectMqtt();
    }
  }

  mqttClient.loop();           // Proces any outstanding MQTT messages

  updateReadings();


}

// FUNCTION - Generate client ID (chipID) ------------------------------------------------
/*
 *  TODO: Que las iniciales del paciente formen parte del ID. Que se intrduzca por teclado.
 *  Compatibilidad con base de datoso para la numeracion y que sea un ID único.
 *
 *  Idea: utilizar fecha y hora de creacion como parametro ID
 */
String NewClientID()
{
  String clientID = "SPV001";                  // Iniciales + XXX
  //clientID += String(random(0xffff), HEX);   // Generate random ID. No queremos un random.
  return clientID;

  // Mostrar por pantalla
  Serial.println();
  Serial.print("ID del cliente: ");
  Serial.println(clientID);
}

// FUNCTION - Connect to WiFi ------------------------------------------------------------------
bool initWiFi()
{
   delay(10);

   Serial.println();
   Serial.print("Connecting to wifi ");
   Serial.println(ssid);

   // Conenecto to WiFi
   WiFi.begin(ssid, password);
   randomSeed(micros());              // Semilla aleatoria para micros()


  // Wait for connection ser amount of intervals
  int num_attempts = 0;
  while (WiFi.status() != WL_CONNECTED && num_attempts <= WIFI_CONNECT_MAX_ATTEMPTS)
  {
   delay(WIFI_CONNECT_INTERVAL);
   Serial.print(".");
   num_attempts++;
  }

  if (WiFi.status() != WL_CONNECTED)
  {
    return false;
  }
  else
  {
    return true;
  }
}

// PROCEDURE - Reconnect to MQTT broker, and publish a notifiacion to the status topic ----------------------
void reconnectMqtt(){
  char mqtt_client_id[20];
  // Añade a mqtt_client_id donde %x lee valor dado g_cliente_ID
  sprintf(mqtt_client_id, "Paciente-%s", g_cliente_ID);
  // Loop until we're reconnected
  while(!mqttClient.connected())
  {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (mqttClient.connect(mqtt_client_id, mqtt_username, mqtt_password))
    {
      Serial.println("connected");
      // Once connected, publish announcement
      sprintf(g_mqtt_message_buffer, "Device %s starting up..", mqtt_client_id);
      mqttClient.publish(status_topic, g_mqtt_message_buffer);
      // Resubscribe
      // mqttclient.subscribe(g_command_topic);     // Activar cuando se requiera leer topic
    }
```

```
    else // Omitir Serial para acelerar código
    {
      Serial.print("failed, rc=");
      Serial.print(mqttClient.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrtying
      delay(5000);
    }
  }

}
// PROCEDURE - Callback : recibir mensaje de un topic ----------------------------------------------------
/*
 * This callback is invoked when a MQTT message is received. It's not important tright now
 * for  this project becasue we don't receive commands via MQTT. You can modify this function
 * to maje the device act on commands that you send it
 */
void mqttCallback(char* topic, byte* payload, unsigned int len) {
   Serial.print("Message arrived (topic)[");
   Serial.print(topic);
   Serial.print("] ");
   for (int i = 0; i < len; i++) {
     Serial.print((char)payload[i]);  // convierte a character el mensaje pasado como int
   }
   Serial.println();
}

// PROCEDURE - Update Readings ---------------------------------------------------------------------------
/*
 * TODO: Cambiar var tiempo a t_time
 */
void updateReadings(){
  unsigned long GSR_tStart;                     // Initial time for measuring time spent
  unsigned long GSR_tEnd;                      // Initial time for measuring time spent

   //Leemos los valores del sensor GSR
   GSR_tStart=micros();             // Inicio del tiempo para coger muestras. OJO MICROSEGUNDOS 10^-6
   for (int i=0; i<g_mysize;i++){
    g_data[i]=analogRead(g_pin_GSR);  // recogo 'mysize' numero muestras
     //time[i]=micros();
   GSR_tEnd=micros();              // Fin del tiempo para coger muestras
   }

  // Mostrar tiempo necesitado para leer los datos por pantalla
  showTimeNeeded(GSR_tStart,GSR_tEnd);
  g_GSR_readings_taken = true;


  // Creacion del paquete y Mostrar datos del vector enviados
  for (int i=0; i<g_mysize;i++){
    Serial.println(g_data[i],DEC);
    g_GSR_value.concat(g_data[i]);          // Creacion del paquete de datos
    g_GSR_value.concat(" ");                // Añade espacio
  }

  // Report the new values
   reportToMqtt();
   reportToSerial();

   // Vaciar paquete
   g_GSR_value = "";
}

// PROCEDURE - Report the latest values to MQTT ----------------------------------------------------------
void reportToMqtt()
{
  String message_string;


  #if REPORT_MQTT_SEPARATE
    if (true == g_GSR_readings_taken)
    {
      /* Report GSR value */
      message_string = String(g_GSR_value);
      message_string.toCharArray(g_mqtt_message_buffer, message_string.length() + 1);
      mqttClient.publish(g_GSR_mqtt_topic, g_mqtt_message_buffer);
      // TODO: Coger tiempo ???
      g_GSR_time = millis();

      /* Report GSR time value */
      message_string = String(g_GSR_time);
      message_string.toCharArray(g_mqtt_message_buffer, message_string.length() + 1);
      mqttClient.publish(g_time_mqtt_topic, g_mqtt_message_buffer);
    }
   #endif

   #if REPORT_MQTT_JSON
```

```
    /* Report all values combined into one JSON message */
    // This is an example message generated by Tasmota, to match the format:
    // {"Time":"2020-02-27T03:27:22","PMS5003":{"CF1":0,"CF2.5":1,"CF10":1,"PM1":0,
    //"PM2.5":1,"PM10":1,"PB0.3":0,"PB0.5":0,"PB1":0,"PB2.5":0,"PB5":0,"PB10":0}}
    //
    // This is the source code from Tasmota:
    //ResponseAppend_P(PSTR(",\"PMS5003\":{\"CF1\":%d,\"CF2.5\":%d,\"CF10\":%d,\"PM1\":%d,
    //\"PM2.5\":%d,\"PM10\":%d,\"PB0.3\":%d,\"PB0.5\":%d,\"PB1\":%d,\"PB2.5\":%d,\"PB5\":%d,\"PB10\":%d}"),
    //    pms_g_data.pm10_standard, pms_data.pm25_standard, pms_data.pm100_standard,
    //    pms_data.pm10_env, pms_data.pm25_env, pms_data.pm100_env,
    //    pms_data.particles_03um, pms_data.particles_05um, pms_data.particles_10um,
    //pms_data.particles_25um, pms_data.particles_50um, pms_data.particles_100um);

    // Note: The PubSubClient library limits MQTT message size to 128 bytes. The long format
    // message below only works because the message buffer size has been increased to 255 bytes
    // in setup.

      if (true == g_GSR_readings_taken)
      {
         // Convertir a array
         String str = g_GSR_value;
         char char_array[g_GSR_value.length()];
         // Eliminado el +1 para elimnar el ultimo caracter separador
         str.toCharArray(char_array, g_GSR_value.length());

         /* ------------------------------------------------------------------------------------------------

         Serial.println();
         Serial.print("VALOR DE CHAR ARRAY: ");
         Serial.println(char_array);
         Serial.print("VALOR DE g_GSR_value: ");
         Serial.println(g_GSR_value.length());
         Serial.println();

         delay(5000);

         //------------------------------------------------------------------------------------------------
         */
         // Es necesario poner %s entre "\%s"\ por ser un string o char
         sprintf(g_mqtt_message_buffer,  "{\"ID:%s\":{\"GSR\":\"%s\",\"GSR Time\":%i}}",
              g_cliente_ID, char_array, g_GSR_time);
      }
      mqttClient.publish(g_mqtt_json_topic, g_mqtt_message_buffer);
    #endif
}

// PROCEDURE - Mostar tiempos por pantalla ----------------------------------------------------------------
/*
 * TODO : Cambiar a tipo t_time
 */
void showTimeNeeded(unsigned long t_start, unsigned long t_end)
{
  Serial.println("---TIEMPO NECESITADO--");
  Serial.print("Inicio del tiempo (tStart) = ");
  Serial.println(t_start);  // Incio del tiempo
  Serial.print("Fin del tiempo (tEnd) = ");
  Serial.println(t_end);    // Fin del tiempo
  Serial.print("Tarda (tEnd-tStart) = ");
  Serial.println(t_end-t_start); // Lo que ha tardado en recoger la informacion
}

// PROCEDURE - Mostar por pantalla ------------------------------------------------------------------------
void reportToSerial()


{
  if (true == g_GSR_readings_taken)
  {

    int peso = 0;


    /* Paciente*/
    Serial.print("Paciente: ");
    Serial.println(g_cliente_ID);
    /* Report GSR value*/

    Serial.print(g_GSR_mqtt_topic);
    Serial.print(" topic is GSR: ");
    Serial.println(String(g_GSR_value));

    /* Report GSR time value*/
    Serial.print("Time: ");
    Serial.println(String(g_GSR_time));

    /* Peso del paquete*/
```

```
    Serial.print("Peso del paquete GSR value: ");
    peso = sizeof(g_GSR_mqtt_topic);
    Serial.print(peso);
    Serial.println(" bytes.");

    delay(5000);

  }
  Serial.println();
}
```

12.7.1 Configuration.h

```
/* ------------------- General config ------------------- */
// WiFi
const char* ssid          = "SIDD";      // Your WiFi SSID
const char* password      = "PASS";          // Your WiFi password

// MQTT
// char* mqtt_broker    = "test.mosquitto.org";         // IP address of your MQTT broker
const char* mqtt_broker    = "192.168.1.37";      // IP address of your MQTT broker
int port                   = 1883;                // Port connection
const char* mqtt_username = "mqtt_username";     // Your MQTT username
const char* mqtt_password = "mqtt_password";     // Your MQTT password
#define REPORT_MQTT_SEPARATE true                // Report each valoue to its own topic
#define REPORT_MQTT_JSON true                    // Report all values in a JSON message
const char* status_topic = "events";             // MQTT topic to report startup

// Serial
#define SERIAL_BAUD_RATE 115200                  // Speed for USB serial console
```

12.8 **Velocidad_GSR.ino**

```
/*
 * Nano IOT 33 - analogRead() is Slower than expected
 *
 * https://forum.arduino.cc/t/nano-iot-33-analogread-is-slower-than-expected/635209
 */

# define _SAMPLE_COUNT 360

void setup()
{
pinMode(A7, INPUT); //I've tried all of the audio input pins, btw
}

void loop()
{
float t0, t;

t0 = micros() ;
for( int i =0; i <_SAMPLE_COUNT; i++)
{
analogRead (A7);
}

t = micros() - t0;

Serial.print("Tiempo por muestra: ");
Serial.print((float)t/_SAMPLE_COUNT);
Serial.println(" us"); // microsegundos
Serial.print("Frequencia: ");
Serial.print((float)_SAMPLE_COUNT*1000000/t); // Multiplica por 10^6
Serial.print(" Hz");
Serial.println();

delay( 2000 ); // 2 segundos

}
```

12.9 **TFM_v3.ino**

```
/*
 * Autor: Sergio Pons Villanueva
 * Ultima fecha de modificacion: 20/08/2021
 */


/*------------------- Configuration --------------------------*/
// Configuration should be done in the included file:
#include "configuration.h";

/*------------------- Libraries ------------------------------*/
#include <WiFiNINA.h>
#include <PubSubClient.h>
/*
 *  TODO : Buscar una forma de incluir libreria de tiempo.
 *  Necesita que lepases la hora. Utilizar datos subMqtt y  Node-red ??
 */

/*------------------- Global Variables ----------------------*/
// g_: for global

// SENSORS

int g_data_buffer[100];                       // Vector de datos

bool g_GSR_readings_taken     = false;      // true/fasle: whether any readings have been taken
int g_GSR_value               = 0;          // Value for sensor GSR
int g_pin_GSR                 = A0;         // Analog pin on NN33IoT
float g_GSR_time_sep          = 0.0;        // Tiempo entre muestra y muestra


// MQTT
# define g_topicSize 100                    // Tamaño del string de los topics. Peso en bytes del paquete.
char g_mqtt_message_buffer[g_topicSize*5];        // General purpose buffer for MQTT messages
char g_command_topic[g_topicSize];                // MQTT topic for receiving commands

#if REPORT_MQTT_SEPARATE
char g_GSR_mqtt_topic[g_topicSize];               // MQTT topic for reporting GSR values
char g_npaquete_mqtt_topic[g_topicSize];          // MQTT topic for reporting GRS time value
#endif

#if REPORT_MQTT_JSON
char g_mqtt_json_topic[g_topicSize];              // MQTT topic for reporting all values using JSON
#endif

unsigned long g_npaquete       = 0;               // Value for number of pacakge sent
//int g_npaquete       = 0;               // Value for number of pacakge sent


// WiFi
#define WIFI_CONNECT_INTERVAL       500     // Wait 500ms intervals for wifi connection
#define WIFI_CONNECT_MAX_ATTEMPTS   10      // Number of attempts/intervals to wait

// Time
unsigned long g_segundo          = 0;        // Variable tiempo
unsigned long g_tiempo_inicio    = 0;
unsigned long g_tiempo_actual    = 0;


//General
char* g_cliente_ID;                         // Simulate chipID. *: NO CAMBIA

unsigned long g_tStart_loop  = 0;           // Initial time for measuring time spent
unsigned long g_tEnd_loop    = 0;           // End time for measuring time spent

// Contador
unsigned int g_contador = 0;                // Contar el numero de paquetes enviados (en la variable time)

// Nuevas var 11.08.2021
uint32_t g_tsLastReport       = 0;                  // ts: timeStamp?
#define MUESTRAS_SEGUNDO 360.0
float g_reporting_period      = 1000000 / MUESTRAS_SEGUNDO; // 10^6 por ser micros. Frec muestreo
int g_cantidad_datos_leidos   = 0;

// Nuevas var max y min frec muestro. Inicialiacion.
unsigned int g_maxfrec = 0;
unsigned int g_minfrec = 10000;
unsigned int g_auxfrec = 0;
/*------------------- Instantiate Global Objects ----------------*/
// MQTT
WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

/*-------------------- Function Signatures ----------------------*/
void mqttCallback(char* topic, byte* payload, unsigned int len);    // Done
bool initWiFi();                                                    // Done
```

```
void reconnectMqtt();                                                   // Done
void updateReadings();                                                  // Done
void reportToMqtt();                                                    // Done
void reportToSerial();                                                  // Done
String NewClientID();                                                   // Done
void calcularFrecuencias();                                             // Done
/* ------------------ Program -------------------------------------*/
/*
 * SETUP
 */


void setup()
{
  // Initialize connection
  Serial.begin(SERIAL_BAUD_RATE);

   while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.println();


  // We need a unique device ID for our MQTT client connection.
  // Normalmente utilizariamos el ID del dispositivo (los de arduino no los tienen)
  g_cliente_ID = "IDSPV001";       // Get ID. Revisar TODO.


  /*------------------TOPICS----------------------------------------------------------------------------------*/
  // Set up the topics for publishing sensor readings. By inserting the unique ID,
  // the result is of the form: "arduino_1/GSR/valor" etc
  sprintf(g_command_topic, "cmnd/%s/COMMAND", g_cliente_ID);                    // For receiving commands

  #if REPORT_MQTT_SEPARATE
    sprintf(g_GSR_mqtt_topic, "arduino_1/%s/GSR/valor", g_cliente_ID);              // Data form AN33IoT
    sprintf(g_npaquete_mqtt_topic, "arduino_1/%s/GSR/numPaquete", g_cliente_ID);    // Data form AN33IoT
  #endif

  #if REPORT_MQTT_JSON
    sprintf(g_mqtt_json_topic, "arduino_1/%s/SENSOR", g_cliente_ID);                // Data from AN33IoT
  #endif

  // Report the MQTT topics to the serial console
  Serial.println(g_command_topic);                  // For receiving messages

  #if REPORT_MQTT_SEPARATE
    Serial.println("MQTT topics: ");
    Serial.println(g_GSR_mqtt_topic);               // From AN33IoT
    Serial.println(g_npaquete_mqtt_topic);          // From AN33IoT
  #endif


  #if REPORT_MQTT_JSON
    Serial.println("MQTT topics en JSON: ");
    Serial.println(g_mqtt_json_topic);               // From AN33IoT
  #endif

  // Connect to WiFi
  if (initWiFi())
  {
    Serial.print("WiFi connected");
    Serial.println(ssid);
    Serial.print("IP local address: ");
    Serial.println(WiFi.localIP());


     /* Set up the MQTT client */
    mqttClient.setServer(mqtt_broker, port);
    mqttClient.setCallback(mqttCallback);
    // Tamaño del paquete que puede enviar MQTT por defecto 128 (recomendado 255)
    mqttClient.setBufferSize(2048);

  }
  else
  {
    Serial.println("WiFi FAILED");
    Serial.println("CHECK WIFI CONNEXION");
    while(true){


    }

  }
  Serial.println(g_reporting_period);
  g_tiempo_inicio = micros();
}
```

```
/*
 * ------------------- MAIN LOOP ----------------------------------------------------------------------------
 */

void loop() {


  if (WiFi.status() == WL_CONNECTED)
  {
    if (!mqttClient.connected())
    {
      reconnectMqtt();
    }
  }
  // Proces any outstanding MQTT messages. Tiene que ir después de la comprobación del Wifi.
  mqttClient.loop();

  Serial.print("Tiempo entre muestra y muestra: ");
  Serial.println(micros() - g_tsLastReport);

  // Calcular aproximadamente la frecuencia max y min
  //calcularFrecuencias();


  //g_reporting_period = 100000;
  // Control de frecuencia de muestreo
  if (micros() - g_tsLastReport >= g_reporting_period){
    calcularFrecuencias();

    g_tsLastReport = micros();

    updateReadings();
    reportToMqtt();

    // Vaciar paquete
    g_GSR_value = 0;


  }

    // Report the new values

   //reportToSerial();



}

// FUNCTION - Generate client ID (chipID) -----------------------------------------------
/*
 *  TODO: Que las iniciales del paciente formen parte del ID. Que se intrduzca por teclado.
 *  Compatibilidad con base de datoso para la numeracion y que sea un ID único.
 *
 *  Idea: utilizar fecha y hora de creacion como parametro ID
 */
String NewClientID()
{

  // Semilla aleatoria para valores random
  //randomSeed(micros());

  String clientID = "SPV001";                  // Iniciales + XXX
  //clientID += String(random(0xffff), HEX);   // Generate random ID. No queremos un random.
  return clientID;

  // Mostrar por pantalla
  Serial.println();
  Serial.print("ID del cliente: ");
  Serial.println(clientID);
}

// FUNCTION - Connect to WiFi -----------------------------------------------------------
bool initWiFi()
{
   Serial.println();
   Serial.print("Connecting to wifi ");
   Serial.println(ssid);

   // Conenecto to WiFi
   WiFi.begin(ssid, password);


   // Wait for connection ser amount of intervals
   int num_attempts = 0;
   while (WiFi.status() != WL_CONNECTED && num_attempts <= WIFI_CONNECT_MAX_ATTEMPTS)
   {
```

```
    delay(WIFI_CONNECT_INTERVAL);
    Serial.print(".");
    num_attempts++;
   }

   if (WiFi.status() != WL_CONNECTED)
   {
     Serial.print("Falied connexion to Wifi. Attempts: ");
     Serial.println(num_attempts);
     return false;
   }
   else
   {
     return true;
   }
}

// PROCEDURE - Reconnect to MQTT broker, and publish a notifiacion to the status topic ----------------------
void reconnectMqtt(){
  char mqtt_client_id[20];
   // Añade a mqtt_client_id donde %s lee valor dado g_cliente_ID
  sprintf(mqtt_client_id, "Paciente-%s", g_cliente_ID);

  // Loop until we're reconnected
  while(!mqttClient.connected())
  {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (mqttClient.connect(mqtt_client_id, mqtt_username, mqtt_password))
    {
      Serial.println("connected");
      // Once connected, publish announcement
      sprintf(g_mqtt_message_buffer, "Device %s starting up..", mqtt_client_id);
      mqttClient.publish(status_topic, g_mqtt_message_buffer);
      // Resubscribe
      // mqttclient.subscribe(g_command_topic);     // Activar cuando se requiera leer topic
    }
    else // Omitir Serial para acelerar código
    {
      Serial.print("failed, rc=");
      Serial.print(mqttClient.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrtying
      delay(5000);
    }
  }

}

// PROCEDURE - Callback : recibir mensaje de un topic ------------------------------------------------------
/*
 * This callback is invoked when a MQTT message is received. It's not important tright now
 * for  this project becasue we don't receive commands via MQTT. You can modify this function
 * to maje the device act on commands that you send it
 */
void mqttCallback(char* topic, byte* payload, unsigned int len) {
   Serial.print("Message arrived (topic)[");
   Serial.print(topic);
   Serial.print("] ");
   for (int i = 0; i < len; i++) {
     Serial.print((char)payload[i]);  // convierte a character el mensaje pasado como int
   }
   Serial.println();
}

// PROCEDURE - Update Readings -----------------------------------------------------------------------------
/*
 * TODO: Cambiar var tiempo a t_time
 */
void updateReadings(){

  g_GSR_value                 = analogRead(g_pin_GSR);
  g_GSR_readings_taken        = true;
  g_cantidad_datos_leidos++;
  //Serial.print("Datos leidos (GSR, HR, SpO2): ");
  Serial.print("Datos leidos (GSR): ");
  Serial.println(g_GSR_value);
  //Serial.print(", 0.00, 0.00");
}

// PROCEDURE - Report the latest values to MQTT ------------------------------------------------------------
void reportToMqtt()
{
  String message_string;
  Serial.println();
  Serial.println("###### ENVIANDO MENSAJES MQTT ######");
```

```
   #if REPORT_MQTT_SEPARATE
     if (true == g_GSR_readings_taken)
     {
       /* Report GSR value */
       message_string = String(g_GSR_value);
       message_string.toCharArray(g_mqtt_message_buffer, message_string.length() + 1);
       mqttClient.publish(g_GSR_mqtt_topic, g_mqtt_message_buffer);
       // TODO: Coger tiempo ???

       Serial.println("Topic GSR Value - Enviado");

       /* Report GSR time value */
       g_npaquete++;
       message_string = String(g_npaquete);
       message_string.toCharArray(g_mqtt_message_buffer, message_string.length() + 1);
       mqttClient.publish(g_npaquete_mqtt_topic, g_mqtt_message_buffer);
       Serial.println("Topic GSR Paquete - Enviado");
     }
    #endif

    #if REPORT_MQTT_JSON
     /* Report all values combined into one JSON message */

     // Note: The PubSubClient library limits MQTT message size to 128 bytes. The long format
     // message below only works because the message buffer size has been increased to 255 bytes
     // in setup.

       if (true == g_GSR_readings_taken)
       {

          // Es necesario poner %s entre "\%s"\ por ser un string o char
          // Si pones \" %i\" estas pasando a string un entero
          sprintf(g_mqtt_message_buffer,  "{\"ID:%s\":{\"GSR\":%i,\"GSR Paquete\":%i}}",
               g_cliente_ID, g_GSR_value, g_npaquete);
       }
       mqttClient.publish(g_mqtt_json_topic, g_mqtt_message_buffer);
       Serial.println("Topic JSON - Enviado");
     #endif
 }


// PROCEDURE - Mostar por pantalla ------------------------------------------------------------------------------
void reportToSerial()

{
  if (true == g_GSR_readings_taken)
  {

    int peso = 0;

    Serial.println("");
    Serial.println("###### MOSTRANDO DATOS ######");

    /* Paciente*/
    Serial.print("Paciente: ");
    Serial.println(g_cliente_ID);
    /* Report GSR value*/

    Serial.print(g_GSR_mqtt_topic);
    Serial.print(" topic is GSR: ");
    Serial.println(String(g_GSR_value));

    /* Report package number value*/
    Serial.print("Numero del paquete: ");
    Serial.println(String(g_npaquete));


     // Mostrar tamaño del buffer
    Serial.print("TAMAÑO DEL BUFFER: ");
    Serial.println(MQTT_MAX_PACKET_SIZE);

    /* Peso del paquete*/
    Serial.print("Peso del paquete GSR value: ");
    peso = sizeof(g_GSR_mqtt_topic);
    Serial.print(peso);
    Serial.println(" bytes.");

  }
}

// PROCEDURE - Calcular frecuencias max y min
void calcularFrecuencias(){
  g_auxfrec = micros() - g_tsLastReport;

  if (g_auxfrec > g_maxfrec && g_auxfrec < 20000)
```

```
  {
    g_maxfrec = g_auxfrec;
  }
  if (g_auxfrec < g_minfrec)
  {
    g_minfrec = g_auxfrec;
  }

   Serial.print("Frecuencia max: ");
   Serial.println(g_maxfrec);
   Serial.print("Frecuencia min: ");
   Serial.println(g_minfrec);

   Serial.print("Numero del paquete: ");
   Serial.println(String(g_npaquete));
}
```

12.9.1 Configuration.h

```
/* ------------------- General config ------------------- */
// WiFi
const char* ssid          = "SIDD";      // Your WiFi SSID
const char* password      = "PASS";          // Your WiFi password

// MQTT
// char* mqtt_broker    = "test.mosquitto.org";         // IP address of your MQTT broker
const char* mqtt_broker   = "192.168.1.37";      // IP address of your MQTT broker
int port                  = 1883;                // Port connection
const char* mqtt_username = "mqtt_username";     // Your MQTT username
const char* mqtt_password = "mqtt_password";     // Your MQTT password
#define REPORT_MQTT_SEPARATE true                // Report each valoue to its own topic
#define REPORT_MQTT_JSON true                    // Report all values in a JSON message
const char* status_topic = "events";             // MQTT topic to report startup

// Serial
#define SERIAL_BAUD_RATE 115200                  // Speed for USB serial console
```

12.10 **Task_different_cores.ino**

```
/*********
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
*********/

TaskHandle_t Task1;
TaskHandle_t Task2;

// LED pins
const int led1 = 12;
const int led2 = 14;

void setup() {
  Serial.begin(115200);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  delay(1000);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

  //create a task that will be executed in the Task1code() function, with priority 1 and executed on core 0
  xTaskCreatePinnedToCore(
                    Task1code,   /* Task function. */
                    "Task1",     /* name of task. */
                    10000,       /* Stack size of task */
                    NULL,        /* parameter of the task */
                    1,           /* priority of the task */
                    &Task1,      /* Task handle to keep track of created task */
                    0);          /* pin task to core 0 */
  //delay(500);

  //create a task that will be executed in the Task2code() function, with priority 1 and executed on core 1
  xTaskCreatePinnedToCore(
                    Task2code,   /* Task function. */
                    "Task2",     /* name of task. */
                    10000,       /* Stack size of task */
                    NULL,        /* parameter of the task */
                    1,           /* priority of the task */
                    &Task2,      /* Task handle to keep track of created task */
                    1);          /* pin task to core 1 */
    //delay(500);
}


//Task1code: blinks an LED every 1000 ms
void Task1code( void * pvParameters ){
  Serial.print("Task1 running on core ");
  Serial.println(xPortGetCoreID());

  for(;;){
    digitalWrite(led1, HIGH);
    Serial.println("Led Core 0 - ON");
    delay(1000);
    digitalWrite(led1, LOW);
    Serial.println("Led Core 0 - OFF");
    delay(1000);
  }
}

//Task2code: blinks an LED every 2000 ms
void Task2code( void * pvParameters ){
  Serial.print("Task2 running on core ");
  Serial.println(xPortGetCoreID());

  for(;;){
    digitalWrite(led2, HIGH);
    Serial.println("Led Core 1 - ON");
    delay(2000);
    digitalWrite(led2, LOW);
    Serial.println("Led Core 1 - OFF");
    delay(2000);
  }
}

void loop() {

}
```

ANEXO III

MANUALES DE INSTALACIÓN E INSTRUCCIONES

13 ANEXO III: MANUALES DE INSTALACIÓN E INSTRUCCIONES

13.1 Instalación de la máquina virtual WMware

La creación de la máquina virtual, a la que se le ha puesto el nombre de "virtualubuntu", no ha sido trivial, siendo el mayor problema encontrado la asignación de una IP que pudiera ser detectada por parte del dispositivo. En una primera instancia, aunque le fue asignada una IP diferente a la del portátil, no se lograba establecer la conexión entre el dispositivo y el broker instalado en la máquina virtual.

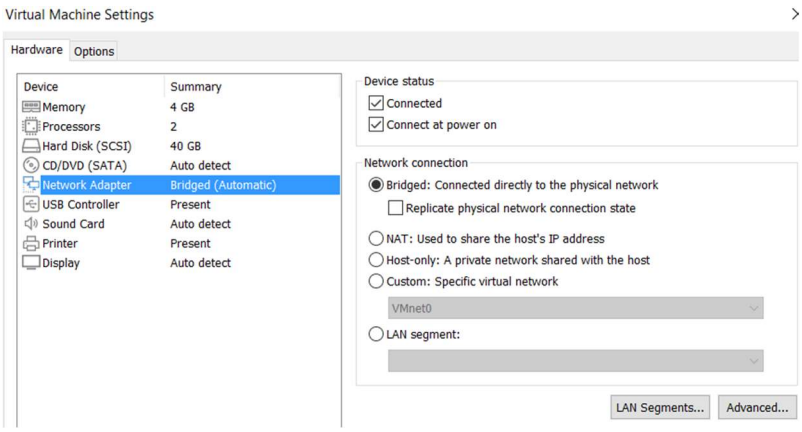
Se observó que los valores de configuración de red TCP/IP que se mostraba en el terminal de Ubuntu (introduciendo *ifconfig*) no coincidía con las direcciones asociadas a VMware que se mostraba (al introducir *ipconfig*) en terminal de Windows. Las siguientes ilustraciones muestran las diferentes IPs mencionadas:

```
spovi@virtualubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.37  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe                    of  prefixlen 64  scopeid 0x20<link>
        ether                     xqueuelen 1000  (Ethernet)
        RX packets 1589  bytes 1788147 (1.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 980  bytes 136241 (136.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
Adaptador de Ethernet VMware Network Adapter VMnet1:

   Sufijo DNS específico para la conexión. . :
   Vínculo: dirección IPv6 local. . . :
   Dirección IPv4. . . . . . . . . . . . : 192.168.237.1
   Máscara de subred . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . :

Adaptador de Ethernet VMware Network Adapter VMnet8:

   Sufijo DNS específico para la conexión. . :
   Vínculo: dirección IPv6 local. . . :
   Dirección IPv4. . . . . . . . . . . . : 192.168.85.1
   Máscara de subred . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . :
```

Tras una investigación más profunda, se encontró que el problema radicaba en que la máquina virtual estaba configurada por defecto en modo NAT, impidiendo así que se conectara de forma independiente a la red de casa. La solución al problema consistió en seleccionar el modo de funcionamiento *bridged* sobre la tarjeta inalámbrica [99].

Accediendo a las opciones de la máquina virtual (*Virtual Machine Settings*), se seleccionó como opción de conexión de la red de trabajo (Network connection), permitiendo de este modo una conexión directa a la red de trabajo física.



Una vez establecida, la máquina virtual apareció, tras su conexión a la red WiFi, como un dispositivo más conectado a la red de casa con la IP que tiene en Ubuntu.

13.2 **Servidor/broker Mosquitto**

A continuación, se muestran una serie de comandos que han sido de utilidad para manejar la el servidor/*broker* Mosquitto obtenidas de su página oficial [85].

SUBSCRIBIRSE en Raspberry pi

mosquitto_sub -d -t TOPIC

SUBSCRIBIRSE a otro broker y no al de Raspberry pi

mosquitto_sub -d -h IP_BROKER -t TOPIC

PUBLICAR un topic

mosquitto_pub -d -h BROKER -t TOPIC -m MESSAGE

mosquitto_pub -h 192.168.1.33 -t arduino/simple -m "hola mundo"

PUBLICAR un topic CON USUARIO Y CONTRASEÑA

mosquitto_pub -d -t TOPIC -m MESSAGE -u USER -P PASSWORD

mosquitto_sub -d -t TOPIC -u USER -P PASSWORD

GUARDAR DATOS directamente en el dispositivo con Mosquitto instalado

mosquito_sub -h BROKER -p PORT -t TOPIC -v > TEXT.txt

mosquito_sub -h localhost -p 1883 -t toppicName -v > docexport.txt

En el caso de que se quiera implementar seguridad mosquitto en un dispositivo Linux es necesario realizar los siguientes pasos que se describen a continuación.

Se recuerda que al realizar estas acciones en Moquitto implica introducir *user:pass* en el arduino y en el Node-red.

Crear usuario y contraseña y guardarlo en un archivo:

echo "mqtt_username:mqtt_password" > pwfile

cat pwfile

Encriptarlo

mosquitto_passwd -U pwfile

cat pwfile

Mover le archivo al directorio de configuración de mosquitto:

sudo mv pwfile /etc/mosquitto/

Editar la configuración de mosquitto:

sudo vim /etc/mosquitto/mosquitto.conf

Reiniciar mosquito para que cargue la nueva configuración

sudo /etc/init.d/mosquitto restart

Probar conexión sin introducir el user/pass e introduciéndolos para comprobar su funcionamiento

mosquitto_sub -v -t "#"

>> Connection Refused: not authorised

mosquitto_sub -v -u mqtt_username -P mqtt_password -t "#"

>> Connection Succed

13.3 **InfluxDB**

A continuación, se describen los pasos para realizar la instalación de la base de datos InfluxDB en un ordenador con sistema operativo basado en Linux. También se describen una serie de instrucciones que han sido de gran utilidad para el manejo y visualización de los datos.

13.3.1 Instalación InfluxDB

Desde el terminal linux se introduce:

wget https://repos.influxdata.com/influxdb.key

sudo apt-key add influxdb.key

Se selecciona la versión que nos corresponda (mirar)

lsb_realease -a

En el momento de la realización de este trabajo era la "buster"

echo "deb https://repos.influxdata.com/debian buster stable" | sudo tee /etc/apt/sources.list.d/influxdb.list

deb https://repos.influxdata.com/debian buster stable

Actualizar los paquetes disponibles

sudo apt-get update

sudo apt install influxdb

Decir al administrador de servicios que habilite InfluxDB al arrancar el sistema

sudo systemctl unmask influxdb (facilita el acceso externo)

sudo systemctl enable influxdb

sudo systemctl start influxdb

influx

Crear usuario/contraseña

CREATE USER admin WITH PASSWORD 'adminpassword' WITH ALL PRIVILEGES

Salimos

exit

Para editar la configuración

sudo nano /etc/influxdb/influxdb.conf

Usar CTRL+W para buscar la sección HTTP

auth-enabled = true

pprof-enabled = true

pprof-auth-enabled = true

ping-auth-enabled = true

Reiniciamos

sudo systemctl restart influxdb.service

Entrar con user/pass

influx -username 'admin' -password 'adminpassword'

Crear una base de datos

CREATE DATABASE sensors_db

13.3.2 Instrucciones de utilidad

Una vez todo conectado con Node-RED accedemos con la terminal a InfluxDB

Entrar con user/pass

influx -username 'admin' -password 'adminpassword'

Indicamos la base de datos a utilizar

use sensors_db

Mostramos las mediciones (topics)

show measurements

Mostrar datos

*select * from "TOPIC" limit NUMBER*

Borrar datos

DROP SERIES FROM /./*

Mostrar identificaciones

SHOW FIELD KEYS