

TRABAJO ESPECIAL DE GRADO

**DISEÑO DE UN SISTEMA DE ADQUISICIÓN Y TRANSMISIÓN DE DATOS
ORIENTADO A MEDIDORES DE ENERGÍA ELÉCTRICA, UTILIZANDO
UNA RED MALLADA WIFI CON MICROCONTROLADORES ESP32**

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Marco Alejandro Rodríguez Ferrer
para optar al título de
Ingeniero Electricista.

Caracas, julio de 2020

TRABAJO ESPECIAL DE GRADO

**DISEÑO DE UN SISTEMA DE ADQUISICIÓN Y TRANSMISIÓN DE DATOS
ORIENTADO A MEDIDORES DE ENERGÍA ELÉCTRICA, UTILIZANDO
UNA RED MALLADA WIFI CON MICROCONTROLADORES ESP32**

TUTOR ACADÉMICO: Ing. José Alonso

Presentado ante la ilustre
Universidad Central de Venezuela
por el Br. Marco Alejandro Rodríguez Ferrer
para optar al título de
Ingeniero Electricista.

Caracas, julio de 2020

*A mis padres
Ayarlen Ferrer y Marco Rodríguez.*

*A la memoria de mis abuelos
Esperanza Merecuana y Francisco Ferrer.*

*Dónde quiera que se encuentren, por todo lo que hicieron por mi y mis hermanos.
Eternamente gracias.*

RECONOCIMIENTOS Y AGRADECIMIENTOS

Autor del Trabajo de Grado

Título del Trabajo de Grado

Tutor Académico: nombre del profesor. Tesis. Caracas, Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Mención Electrónica. Año 2020, xvii, 144 pp.

Palabras Claves: Palabras clave.

Resumen.- Escribe acá tu resumen

ÍNDICE GENERAL

RECONOCIMIENTOS Y AGRADECIMIENTOS	III
ÍNDICE GENERAL	VIII
LISTA DE FIGURAS	XII
LISTA DE TABLAS	XV
LISTA DE ACRÓNIMOS	XVI
INTRODUCCIÓN	1
CONCEPTUALIZACIÓN DEL PROYECTO	3
1.1. PLANTEAMIENTO DEL PROBLEMA	3
1.2. JUSTIFICACIÓN	4
1.3. OBJETIVO GENERAL	5
1.4. OBJETIVOS ESPECÍFICOS	5
MARCO REFERENCIAL	6
2.1. Fundamentos de medición consumo de energía eléctrica	6
2.1.1. Definición	6
2.1.2. Clasificación de los medidores	7
2.1.2.1. Según el principio de funcionamiento	7
2.1.2.2. Según su construcción:	8
2.1.3. Infraestructuras de medición de energía	9

2.1.3.1.	Red de área local (Home Area Network)	10
2.1.3.2.	Red de vecindad (Neighbourhood Area Network) . .	11
2.1.3.3.	Red de área amplia (Wide Area Network)	11
2.1.4.	Normativa aplicada a las redes de medición	12
2.1.4.1.	Modelo OSI	13
2.1.4.2.	GWAC	15
2.2.	Redes malladas	15
2.2.1.	Definición	15
2.2.2.	Características	17
2.2.2.1.	Sobre los protocolos de enrutamiento en las redes Malladas	17
2.2.2.2.	Direccionamiento por MAC	18
2.3.	La red mallada ESP-MESH	19
2.3.1.	Definición	19
2.3.2.	Terminología en la ESP-MESH	20
2.3.3.	Topología de árbol sobre la red ESP-MESH	21
2.3.4.	Tipos de nodos en la red ESP-MESH	23
2.3.5.	Señales de reconocimiento y límites de intensidad de señal (RSSI)	24
2.3.6.	Tablas de enrutamiento	25
2.4.	Sistema operativo en tiempo real (RTOS)	27
2.4.1.	FreeRTOS	28
2.4.1.1.	Tareas	28
2.4.1.2.	Colas	28
2.4.1.3.	Semáforos	28
2.4.1.4.	Grupos de eventos	29

2.5. Microcontrolador ESP32	29
2.5.1. Descripción	29
2.5.2. Características	30
2.5.2.1. WiFi	30
2.5.2.2. CPU y Memoria	30
2.5.2.3. Relojes y temporizadores	31
2.5.2.4. Periféricos	31
2.5.2.5. Seguridad	32
DEFINICIÓN Y DESCRIPCIÓN DEL HARDWARE	33
3.1. Definición del hardware	33
3.2. Descripción del hardware	34
3.2.1. Interfaz de comunicación inalámbrica	34
3.2.2. Interfaz serial con el medidor	35
3.2.2.1. Bus serial	35
3.2.2.2. Salida de calibración	37
3.2.3. Unidad controladora	38
3.2.4. Circuito impreso	39
DEFINICIÓN Y DESCRIPCIÓN DEL SOFTWARE	43
4.1. Definición del software	43
4.2. Descripción del software	46
4.2.1. Modo de configuración	47
4.2.1.1. Tablas de particiones en el ESP32	47
4.2.1.2. Interfaz Gráfica de usuario	49
4.2.2. Modo de operación	52

4.2.2.1.	La red mallada y el nodo central	53
4.2.2.2.	Nodo serial RS485	57
4.2.2.3.	Nodo de entrada por pulsos	59
4.2.2.4.	Nodo Repetidor	65
4.2.2.5.	Otras consideraciones	66
PRUEBAS Y RESULTADOS		67
5.1.	Resultados de la interfaz gráfica	67
5.2.	Prueba de funcionamiento del sistema	72
5.2.1.	Extracción de datos del nodo contador de pulsos	74
5.2.2.	Extracción de datos del nodo RS485	75
5.3.	Prueba de restablecimiento del sistema	76
5.3.1.	Para el nodo central	76
5.3.2.	Para cualquier otro nodo	77
CONCLUSIONES		79
RECOMENDACIONES		80
Código fuente		81
Datasheet MAX3485		102
REFERENCIAS		111

LISTA DE FIGURAS

2.1.	Infraestructura general de un sistema de medición avanzado	10
2.2.	Arquitectura de red en comunicaciones para medición inteligente. J. Ekanayake and K. Liyanage, SMART GRID TECHNOLOGY AND APPLICATIONS, First edit. New Delhi, 2013.	12
2.3.	E. Hossain, Z. Han, and V. Poor, Smart Grid Communications and Networking. New York: CAMBRIDGE UNIVERSITY PRESS, 2012. .	16
2.4.	Ilustración de una red Wi-Fi tradicional. Tomada desde [1]	19
2.5.	Ilustración de una red Wi-Fi ESP-MESH. Tomada desde [1]	20
2.6.	Topología de árbol observada en la red ESP-MESH luego de su establecimiento. Tomada de [1]	22
2.7.	Tipos de nodos en una red ESP-MESH. Tomada desde [1]	23
2.8.	Ejemplo ilustrativo sobre el umbral RSSI. Tomado de [1]	25
2.9.	Ilustración de una red mallada, las tablas y subtablas de enrutamiento. Tomado de [1]	26
2.10.	Interacción del OS con otras capas del sistema informático.	27
3.1.	Diagrama general de los componentes del Hardware.	33
3.2.	Microcontrolador ESP32 con su antena WiFi integrada.	34
3.3.	Diagrama de pines del encapsulado DIP8 para MAX3485.	35
3.4.	Configuración recomendada por el fabricante del chip.	36
3.5.	Configuración recomendada por Espressif para el manejo de un chip MAX-485.	36

3.6.	Círcuito impreso con un condensador y el chip MAX3485 utilizada para la comunicación mediante el bus serial en los medidores necesarios.	37
3.7.	Topología utilizada en la salida opto acoplada del medidor de energía.	38
3.8.	Unidad controladora del sistema, tarjeta de desarrollo ESP32.	38
3.9.	Esquemático del módulo de alimentación.	39
3.10.	Esquemáticos de interfaces con los medidores.	39
3.11.	Esquemático de la unidad controladora implementada con el ESP32.	40
3.12.	Imagen del diseño frontal de la PCB utilizando KiCad.	40
3.13.	Imagen del diseño posterior de la PCB utilizando KiCad.	41
3.14.	Imagen de la capa frontal de la PCB renderizada.	41
3.15.	Imagen de la capa posterior de la PCB renderizada.	42
4.1.	Diagrama de bloques genérico del sistema	44
4.2.	Diagrama general de funcionamiento ilustrado.	46
4.3.	Diagrama ilustrativo de la memoria flash para los modelos de tablas de partición. Columna 1: Aplicación de fábrica. Columna 2, 3, 4: Aplicaciones con arranques múltiples	48
4.4.	Diagrama de flujo de las tareas implementadas en el nodo central.	54
4.5.	Estructura de una trama Modbus TCP/IP.	57
4.6.	Diagrama de flujo de las tareas implementadas en el nodo RS485.	58
4.7.	Diagrama de flujo de las tareas implementadas en el nodo de pulsos.	60
5.1.	Log serial del microcontrolador en modo servidor HTTP cuando es enviada una de las pantallas.	67
5.2.	Vista de la pantalla de inicio de sesión del usuario.	68
5.3.	Mensaje recibido en el servidor cuando el usuario rellena el inicio de sesión y envía los datos.	68

5.4.	Vista de la pantalla del formulario de parámetros de la red mallada.	69
5.5.	Mensaje recibido en el servidor cuando el usuario rellena el formulario de red mallada y envía los datos.	69
5.6.	Vista de la pantalla de formulario de parámetros seriales.	70
5.7.	Mensaje recibido en el servidor cuando el usuario rellena el formulario de parámetros seriales y envía los datos.	70
5.8.	Vista de la pantalla de parámetros de red local	71
5.9.	Mensaje recibido en el servidor cuando el usuario rellena el formulario de red local y envía los datos.	71
5.10.	Vista de la pantalla para modificar los parámetros de acceso del usuario. .	72
5.11.	Mensaje recibido en el servidor cuando el usuario rellena el formulario de actualizar usuario y contraseña y envía los datos.	72
5.12.	Diagrama general de la red para pruebas.	73
5.13.	Ejemplo de la red restablecida luego de desconectar el nodo central si el nodo repetidor gana la votación.	76
5.14.	Ejemplo de la red restablecida luego de desconectar el nodo repetidor. .	77

LISTA DE TABLAS

3.1.	Lista de materiales utilizados en el diseño de la PCB	42
4.1.	ESP-IDF Tabla de Particiones por defecto (Aplicación única de fábrica, sin OTA)	48
4.2.	Tabla de particiones personalizada para la aplicación	49
4.3.	Tabla de particiones de la aplicación	62
5.1.	Registro de mensajes del maestro sobre la comunicación con el esclavo de pulsos.	74
5.2.	Registro de mensajes del nodo contador de pulsos sobre la comunica- ción con el maestro.	74
5.3.	Registro de mensajes del maestro sobre la comunicación con el nodo serial rs485.	75
5.4.	Registro de mensajes del nodo serial rs485 sobre la comunicación con el maestro.	75
5.5.	Tiempo de recuperación de la red cuando es desconectado el nodo central. .	76
5.6.	Tiempo de recuperación de la red cuando es desconectado un nodo, exceptuando el nodo central.	78

LISTA DE ACRÓNIMOS

IEC: International Electrotechnical Comission Comisión Electrotécnica Internacional

RTOS: Real Time Operative Sysyem, Sistema operativo en tiempo real

AMR: Automatic Meter Reading, Lectura de medición automática

HAN: Home Area Network, Red de área doméstica

NAN: Neighbourhood Area Network, Red de área de vecindario

WAN: Wide Area Network, Red de área amplia

ISO: International Organization for Standardization, Organización Internacional de Normalización

OSI: Open System Interconnection, Interconexión de Sistemas Abiertos

GUI: Grafic User Interface, Interfaz Gráfica de Usuario

NVS: Non volatile Storage, Memoria no Volátil

STA: STA, Estación (Utilizado para WiFi)

AP: Access Point, Punto de Acceso

INTRODUCCIÓN

Los primeros indicios del uso de la energía eléctrica sucedieron en el cuarto final del siglo XIX. La sustitución del gas y aceite por la electricidad además de ser un proceso técnico fue un verdadero cambio social que implicó modificaciones extraordinarias en la vida cotidiana de las personas, cambios que comenzaron por la sustitución del alumbrado público y posteriormente por varias clases de procesos industriales como motores, metalurgia, refrigeración y de último llegaron a las comunicaciones con la radio y la telefonía.

El siguiente cambio de paradigma en el que se vio involucrado la electricidad tuvo lugar a lo largo del siglo XX y surge desde la necesidad de facilitar las tareas realizadas a diario en casa. En ello los investigadores de la época vieron una solución adaptando equipos con energía eléctrica para su uso en el hogar. Las industrias replicaron el crecimiento tecnológico que tuvieron en sus productos, lo que trajo como consecuencia el desarrollo los electrodomésticos. La primera producción de aparatos en masa como refrigeradores, lavadoras, televisores y radios sucedieron en esta época y tuvieron una alta receptividad por parte de los compradores. La invención del transistor solo aceleró el reemplazo de aparatos dada su capacidad de minimizar los equipos.

La integración de la electrónica a la industria fomentó la creación de sistemas automatizados de adquisición de datos, supervisión y control también llamados sistemas *SCADA* por sus siglas en inglés. Estos sistemas manejan áreas críticas de las industrias y son parte de los procesos fundamentales de muchas de ellas por lo que necesitan ser diseñados con robustez, fiabilidad y seguridad. La aparición del Internet

y las comunicaciones modernas en estos sistemas permite a los usuarios, de manera inalámbrica incluso, monitorear y actuar sobre el sistema a distancia, sin presencia física en la planta.

Además de poder monitorear y realizar acciones sobre los sistemas, los instrumentos de medida de última tecnología se fabrican de modo que puedan ser compatibles con medios de comunicación inalámbricas lo que posibilita la transmisión de datos adquiridos sin necesidad de cable a la central del sistema *SCADA*. El presente trabajo de grado pretende realizar el diseño de un sistema de adquisición y transmisión de datos integrando microcontroladores ESP32 a medidores de energía eléctrica para formar una red mallada inalámbrica capaz de transmitir los datos recolectados a un punto central.

En este archivo debe escribir su introducción.

De acuerdo a Brea la transformada de Laplace debe estudiarse como una función definida en el campo de los números complejos [?].

Otro modo de referencial es [?]

El resto del reporte consta de: en el Capítulo ?? se describe...

En el trabajo se emplea el enfoque de [?]

De acuerdo a la ecuación

CAPÍTULO I

CONCEPTUALIZACIÓN DEL PROYECTO

1.1. PLANTEAMIENTO DEL PROBLEMA

La energía eléctrica es diferente de otras manifestaciones de la energía, debido a que no se puede almacenar por si sola como electricidad. Esto obliga a que la energía eléctrica consumida por un equipo u aparato tenga que generarse al momento en el cual se vaya a consumir. Los procesos para la generación de energía tienen costos altos de desarrollo e implementación a gran escala (países o estados) por lo que surtir de energía a las industrias y electrodomésticos tiene un costo que la empresa que genera la energía necesita recuperar. Como consecuencia se suele medir el consumo de cada uno de los usuarios por razones enteramente económicas.

En Venezuela se utiliza el mismo método de adquisición de datos desde que se instaló el sistema eléctrico. Este consiste en un operador que se acerca hasta el lugar donde se encuentra un medidor de energía y registra la lectura que marca el medidor, esto se hace de manera repetitiva para todos los sitios donde se quiera registrar el consumo. En ocasiones los medidores tienen una salida codificada donde comunica el valor del consumo por infrarrojo lo que permite al operador registrar el valor de ese consumo mediante un aparato compatible con este protocolo. Debido a esta problemática surge la necesidad de sustituir este sistema de adquisición de datos manual por uno que no requiera el traslado del operador hasta el sitio, que sea económico, confiable y eficiente.

Los principales equipos de medición de energía poseen en su diseño una salida por pulsos y soportan distintos protocolos de comunicación, lo que representa una ventaja al trabajar con microcontroladores, pues estos son adaptables a la mayoría de los protocolos mediante programación lo que facilita la adquisición de los datos a partir del medidor. Por otra parte trabajar con microcontroladores ofrece la posibilidad de realizar comunicaciones inalámbricas si se adapta un módulo WiFi como periférico. Interconectar estos módulos WiFi para formar una red mallada permitiría la transmisión de los datos captados a una mayor distancia que la lograda por un único módulo y permitiría su salida hacia alguna red externa deseada sin utilizar cables entre los medidores y la central de adquisición de datos, y sin intervención presencial del operador. Ilustradas las debilidades expuestas anteriormente y las ventajas que representaría un sistema de este tipo se evidencia la necesidad de realizar el diseño.

1.2. JUSTIFICACIÓN

Una red mallada WiFi utilizando microcontroladores permite adaptar a la red equipos que soportan distintos métodos de extracción de datos; otorga la posibilidad de interconectar dispositivos mediante comunicaciones inalámbricas, que no poseen dicha capacidad originalmente; además su desarrollo permitiría extender las variables a medir y los métodos de adquisición de datos del sistema; y por último, posee bajos costos de instalación al no requerir de cableado entre los elementos de la red.

Establecer la red mallada se requiere de nodos que posean la capacidad de enlazarse entre sí formando redes de comunicación, además los nodos deben ser capaces de enrutar los mensajes donde viaja la información. Para un sistema de adquisición de

datos es ideal que cada nodo de la red este conformado por un microcontrolador con un módulo WiFi integrado, esto permitiría que cada uno de los nodos extrajera los datos según el método que se requiera en cada fuente de datos y al formar parte de la red mallada se facilitaría la adquisición de datos para un sistema central. El ESP32 es una opción viable para esta aplicación debido a su módulo WiFi integrado y las librerías desarrolladas en comunicación vía WiFi, utilizar dicha tarjeta representa una ventaja económica y reduce los tiempos de desarrollo respecto a otros microcontroladores.

1.3. OBJETIVO GENERAL

Diseñar un sistema de adquisición y transmisión de datos orientado a medidores de energía eléctrica, utilizando una red mallada WiFi con microcontroladores ESP32.

1.4. OBJETIVOS ESPECÍFICOS

- Documentar los principales métodos de extracción de datos soportados por un medidor de energía, en particular, el protocolo Modbus por RS485 y la salida por pulsos.
- Diseñar el módulo de programa para los nodos que componen la red mallada, conformados por microcontroladores ESP32
- Adaptar un nodo para ser compatible con la salida por pulsos de un medidor de energía y almacenar el valor de la medida para su adquisición mediante la red.
- Adaptar un nodo para adquirir datos desde un medidor de energía que soporte protocolo Modbus RTU vía RS485.
- Validar el funcionamiento del sistema.

CAPÍTULO II

MARCO REFERENCIAL

En este capítulo se documentarán los fundamentos sobre Medidores de energía, Redes WiFi Malladas y el protocolo Modbus.

2.1. Fundamentos de medición consumo de energía eléctrica

2.1.1. Definición

Los medidores de consumo de energía eléctrica, también llamados contadores de energía debido a la tarea que desempeñan, son “Un instrumento destinado a medir la energía eléctrica integrando la potencia con respecto al tiempo” [2]. Las compañías de electricidad utilizan medidores de energía instalados en cada cliente con el propósito de monitorear y facturar el consumo. Debido a ello estos equipos suelen estar calibrados en unidades de facturación de energía, comúnmente se utiliza el kilovatio hora [kWh] y se lee el valor registrado en cada medidor una vez ha llegado el período de cobranza.

Cuando se desea ahorrar energía durante ciertos períodos de tiempo, algunos medidores pueden registrar la demanda de energía, es decir el máximo uso de potencia en un intervalo de tiempo. Aplicar esta metodología de registro de demanda brinda la capacidad de variar las tarifas de electricidad durante el día, permitiendo registrar el consumo de cada usuario durante períodos de tarifas altas (picos de demanda) o en

casos de tarifas bajas donde la demanda de energía en el sistema es baja y la energía menos costosa. Incluso en algunas áreas los medidores de energía poseen relés para desprendimiento de cargas por un periodo de tiempo en caso de picos muy altos de demanda.

2.1.2. Clasificación de los medidores

Tomando como referencia lo que [3] refleja en su trabajo, los medidores de consumo de energía pueden agruparse mediante las siguientes clasificaciones.

2.1.2.1. Segundo el principio de funcionamiento

Medidores electromecánicos: El tipo más común de medidor eléctrico, el registro de la energía se realiza mediante el conteo de las revoluciones de un disco metálico que es conductor eléctrico pero no conductor magnético. Este disco se hace rotar mediante la inducción electromagnética generada por la alimentación, a una velocidad proporcional a la potencia que pasa a través del medidor. El número de vueltas es entonces proporcional a el consumo de energía.

Medidores electrónicos: De forma general este medidor está compuesto por la alimentación, un circuito de medición, un circuito de procesamiento (usualmente microcontroladores) y comunicación además de otros módulos agregados como un RTC, una pantalla de cristal líquido, un módulo de comunicación mediante infrarrojo, entre otros.

El circuito de medición está compuesto por muestreadores y cuantificadores conectados a las entradas de corriente y tensión, así como a la referencia de tensión. Esto viene seguido por una sección de conversión analógica-digital para encontrar el equivalente digitalizado del valor de las entradas. Dichas entradas

en formato digital son tratadas utilizando un procesador digital de señal para calcular los distintos parámetros de medición.

Este tipo de medidor muestra la energía consumida en una pantalla LCD o LED, además de medir la energía consumida pueden también registrar otros parámetros de la carga y del suministro, como la tasa de demanda instantánea y máxima.

2.1.2.2. Segundo su construcción:

Medidor monofásico bifilar (una fase y neutro):

Está compuesto por una bobina de tensión y una de corriente. Su capacidad usualmente está entre 15 A y 60 A.

Medidor bifásico bifilar (dos fases):

Está compuesto por dos bobinas de tensión y dos bobinas de corriente. Usualmente utilizado para medir la energía eléctrica consumida por aparatos que funcionan con la tensión fase-fase residencial.

Medidor bifásico trifilar (dos fases y neutro):

Está compuesto por dos bobinas de tensión y dos bobinas de corriente. Se usa para medir la energía consumida por aparatos que requieran para su funcionamiento dos fases, con este medidor se puede medir la energía consumida por otros aparatos conectados a la misma instalación que funcionen con una sola fase.

Medidor trifásico tetrafilar (tres fases y neutro):

Está compuesto por tres bobinas de tensión y tres bobinas de corriente. Se utiliza para medir la energía consumida por aparatos que requieran funcionar con tres fases.

2.1.3. Infraestructuras de medición de energía

La infraestructura de cada sistema de medición es la característica que delimita sus particularidades, “El cambio de visión de la medición de energía eléctrica se realiza desde el año de 1970 con la incorporación del envío de datos, la comunicación en los años 70 era unidireccional, en una sola vía, desde el usuario hasta la empresa distribuidora. La primera etapa fue la medición AMR (Automatic Meter Reading) y duró alrededor de 30 años dando paso a la evolución, llamada medición inteligente o smart Metering. Smart metering requiere un alto grado de telecomunicaciones en ambas vías: usuario-empresa distribuidora.” [4].

La meta en cuanto a estos sistemas es llegar a una medición inteligente avanzada, es decir incorporar a toda la red eléctrica dispositivos con la capacidad de operar por medio de telecomunicaciones, permitiendo equilibrar la generación de energía eléctrica al consumo real, por medio de los datos enviados por todos los medidores inteligentes.

Existen ciertos requerimientos para realizar una medición inteligente: alta confiabilidad, vida útil, interoperabilidad, rentabilidad, seguridad, consumo mínimo de energía, bajos costos de instalación y mantenimiento. La tecnología desarrollada debe cumplir con estos requerimientos para ser realmente funcional en el campo de la medición automática de consumo.

Para realizar medición inteligente se pueden emplear diferentes tipos de tecnologías de telecomunicaciones de acuerdo al área de aplicación y al canal de transmisión. La figura 2.1 muestra los tipos de arquitecturas de comunicación que son empleadas en la medición inteligente. Los medios guiados para realizar telecomunicaciones incluyen la red telefónica pública conmutada (PSTN), PLC portadora en la línea de alimentación,

Ethernet y cable módems. Los medios no guiados para realizar telecomunicaciones incluyen WiFi, ZigBee, infrarrojos, RFID y GSM / GPRS / CDMA celular.

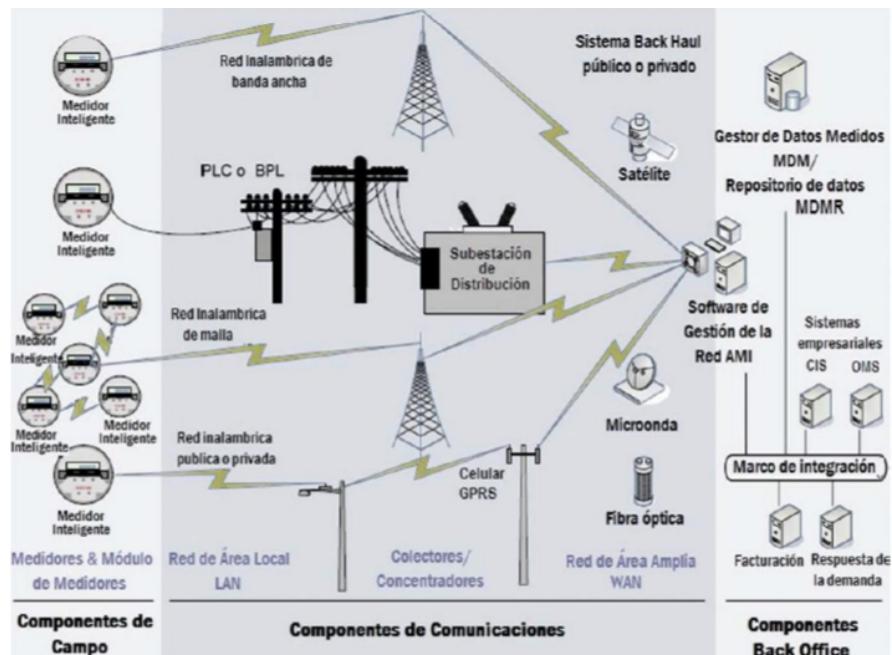


Figura 2.1. Infraestructura general de un sistema de medición avanzado
Fuente: Adaptado de [5]

La infraestructura necesaria para la medición inteligente se divide en tres zonas de acuerdo al tipo de comunicación [5], dichas zonas son las siguientes:

- Red de área local (HAN).
- Red de área de vecindad (NAN).
- Red de área amplia (WAN).

2.1.3.1. Red de área local (Home Area Network)

Es un sistema integrado dentro de los hogares que permite la comunicación entre diferentes dispositivos inteligentes, el dispositivo límite de esta red es el medidor de

energía eléctrica. Este tipo de redes puede utilizar comunicaciones cableadas como BPL (Broadband over Power Lines) o PLC (Power Line Carrier) pero también comunicaciones inalámbricas como redes privadas, una red de banda ancha o redes malladas para comunicarse con la red NAN.

2.1.3.2. Red de vecindad (Neighbourhood Area Network)

Este tipo de red permite la conexión entre múltiples HANs, es un sistema de interconexión entre redes de medidores inteligentes. El principal elemento que lo constituye es el concentrador. El concentrador es el dispositivo que actúa como un puente entre los contadores inteligentes y la puerta de enlace. El concentrador de datos detecta y gestiona los medidores inteligentes de forma automática, realiza lecturas de los consumos y transfiere la información a los centros de control, también facilitan los mensajes de diagnóstico, actualizaciones de firmware y supervisión de las condiciones de los medidores.

2.1.3.3. Red de área amplia (Wide Area Network)

Es la red encargada de conectar múltiples sistemas de distribución y actúa como un puente entre redes de menor tamaño y la red del cliente. Debe ofrecer una red de retorno para conectar los servicios de recolección de datos con las instalaciones del cliente. Dicha red de retorno puede adoptar una variedad de tecnologías (Ethernet, red celular, banda ancha) para transferir la información extraída de las redes NAN a las oficinas locales de servicios públicos. Una puerta de enlace perteneciente a la red WAN puede utilizar la conexión de banda ancha o una red basada en IP para proporcionar un acceso para que el cliente puede acceder a los datos requeridos. La privacidad, fiabilidad

y seguridad de la información son los principales aspectos que se evalúan en este tipo de redes.

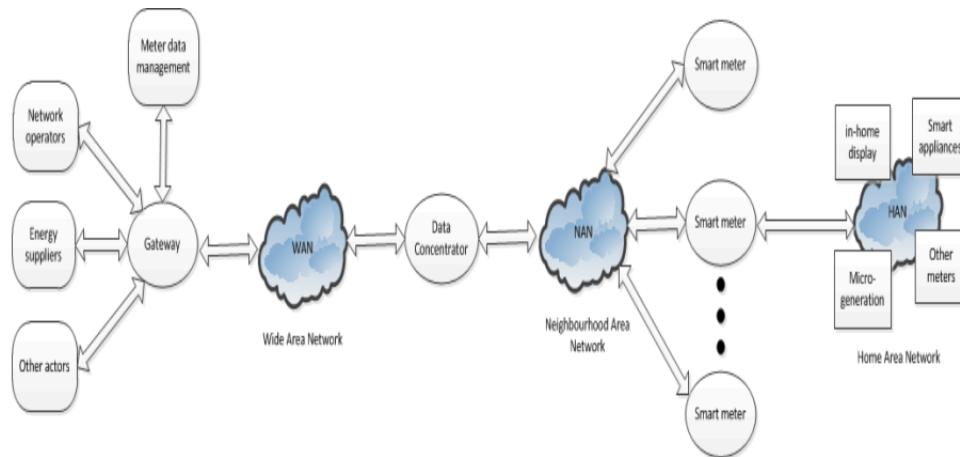


Figura 2.2. Arquitectura de red en comunicaciones para medición inteligente. J. Ekanayake and K. Liyanage, SMART GRID TECHNOLOGY AND APPLICATIONS, First edit. New Delhi, 2013.

2.1.4. Normativa aplicada a las redes de medición

Los problemas en redes de medición suelen ocurrir cuando falla la interoperabilidad entre los distintos sistemas que las componen, esto es causado por la transmisión de datos que se efectúa a través de diversas redes y sistemas de comunicación. Es por ello que la interoperabilidad se convierte en algo fundamental, permitiendo a la infraestructura y la información fluir correctamente en un sistema interoperable permitiendo el intercambio de datos sin necesidad de la intervención manual. El punto más relevante de la interoperabilidad es proporcionar un sistema con la capacidad de integrar componentes plug-and-play, esto se refiere a la capacidad de configurar los componentes y el sistema principal automáticamente comienza a operar con solo conectarlo. Aunque es un concepto sencillo en muchas situaciones puede llegar a ser complejo y poco práctico mostrar una interfaz estándar entre dos sistemas diferentes.

Estas consideraciones en la interoperabilidad no solo reducen los costos de instalación y de integración sino que también definen los requisitos que un componente nuevo debe tener para conectarse al sistema existente, lo que se refleja en la sustitución sencilla de componentes y en una alta escalabilidad del sistema en caso de una mayor demanda o implementación de tecnologías más eficientes.

2.1.4.1. Modelo OSI

Existen diferentes modelos y arquitecturas referenciales para el diseño de un sistema, el primer modelo creado por la ISO en 1980 es llamado OSI (Open System Interconnection) es un estándar que tiene por objetivo conseguir interconectar sistemas de procedencia distinta para que estos pudieran intercambiar información sin ningún tipo de impedimentos debido a los protocolos con los que estos operaban de forma propia según su fabricante. El modelo OSI está conformado por 7 capas o niveles de abstracción:

1. De aplicación
2. De presentación
3. De sesión
4. De transporte
5. De red
6. De enlace de datos
7. Capa física

Cada uno de estos niveles ejerce sus propias funciones para que en conjunto el sistema sea capaz de alcanzar su objetivo final. Es por esta abstracción que el modelo hace posible la intercomunicación en diferentes sistemas al delimitar las funciones que debe realizar cada capa de operación. Es importante aclarar que el modelo OSI no es la definición de una topología ni un modelo de red en sí mismo. Tampoco especifica ni define los protocolos que se utilizan en la comunicación, ya que estos están implementados de forma independiente a este modelo. Lo que realmente hace OSI es definir la funcionalidad de ellos para conseguir un estándar.

- **Capa de aplicación:** Ofrece la posibilidad de acceder a los servicios de las demás capas. Define los protocolos que utilizan las aplicaciones para intercambiar datos. Normalmente el usuario no interactúa con esta capa sino con programas que sirven de interfaz para ocultar el nivel de complejidad subyacente.
- **Capa de presentación:** Se encarga de representar la información funcionando como un traductor entre equipos. En esta capa se trabajan más los datos en sí que cómo se establece el enlace para que estos lleguen. Se tratan aspectos como la semántica (qué significan) y la sintaxis (formato de los datos) para garantizar la compatibilidad con los equipos del sistema. También es la encargada de cifrar los datos y comprimirlos de ser necesario.
- **Capa de sesión:** Esta capa es la que se encarga de mantener y controlar el enlace establecido entre dos computadores que están transmitiendo datos de cualquier índole. Por lo tanto, el servicio provisto por esta capa es la capacidad de asegurar que, dada una sesión establecida entre dos máquinas, la misma se pueda efectuar para las operaciones definidas de principio a fin, reanudándolas en caso de interrupción.¹¹ En muchos casos, los servicios de la capa de sesión son parcial o totalmente prescindibles.

2.1.4.2. GWAC

La integración de la automatización asociada con los recursos eléctricos es importante para respaldar una mayor eficiencia e incorporar recursos renovables, variables y vehículos eléctricos en el sistema eléctrico. Los problemas de integración que enfrenta esta comunidad son análogos a los que enfrenta la industria de la salud, los servicios de emergencia y otras comunidades complejas con muchas partes interesadas. Para resaltar este problema y fomentar la comunicación y el desarrollo de una comunidad de interoperabilidad de redes inteligentes, el GridWise Architecture Council (GWAC) creó un Marco de establecimiento de contexto de interoperabilidad. Este "modelo conceptual"ha sido útil para explicar la importancia de la alineación organizativa, además de las especificaciones de interfaz técnica e informativa para dispositivos y sistemas de redes inteligentes". Como siguiente paso para construir una comunidad sensible a la interoperabilidad, el GWAC ha estado desarrollando un Modelo de madurez de interoperabilidad de redes inteligentes (SG IMM) tomando prestado del trabajo realizado por otros para abordar circunstancias similares. El modelo proporciona un medio para medir el estado y el progreso, analizar las brechas y priorizar los esfuerzos para mejorar la situación. El objetivo es crear una herramienta, o un conjunto de herramientas, que fomente una cultura de interoperabilidad en la comunidad de redes inteligentes. En la siguiente ilustración se hace referencia a una comparación de OSI y GWAC con sus respectivos niveles o capas cada uno:

2.2. Redes malladas

2.2.1. Definición

Una red WiFi mallada se puede definir como una red que permite la comunicación entre nodos a través de múltiples saltos en una topología mallada [6]. Un nodo es la

OSI	GWAC	Función de las capas
7. Aplicación	3. Sintaxis de interoperabilidad Comprensión de los datos Estructura mensajes Intercambio entre sistemas	- Traducción de los caracteres de datos de un formato a otro. - Estructura del contenido del mensaje - Patrones de intercambio de mensajes
6. Presentación		
5. Sesión		- Traducción de direcciones lógicas y nombres a direcciones físicas.
4. Transporte	2. Interoperabilidad de las redes Intercambio de mensajes entre el sistema a través de una variedad de redes	- Transferencia transparente y fiable de datos entre sistemas. - La transferencia de datos entre el origen y el destino a través de intermediarios de la red. - Gestión de la orden de entrega de mensajes
3. Red		
2. Enlace de datos	1. Conectividad básica Mecanismo para establecer conexiones físicas y lógicas del sistema	- Acceso a los medios de hardware y conectividad eléctrica - Transferencia de datos entre nodos de red - Codificación de caracteres, transmisión, recepción, decodificación y corrección de errores
1. Física		

Figura 2.3. E. Hossain, Z. Han, and V. Poor, Smart Grid Communications and Networking. New York: CAMBRIDGE UNIVERSITY PRESS, 2012.

unidad mínima de la red, estas unidades son las encargadas de generar los enlaces entre los dispositivos usuarios que construyen la red mallada.

Por lo general estas redes poseen clientes, enrutadores y puertas de enlace. Los clientes son dispositivos electrónicos, sistemas embebidos o sensores que pueden comunicarse con otros en la red. El enrutador es un dispositivo electrónico que sirve como un intermediario entre dos o más redes para transportar los datos de una red a otra. Y las compuertas de enlace son dispositivos electrónicos que conectan la red con Internet.

[7]

Cuando un nodo no puede operar, el resto de los nodos en la red WiFi mallada aún pueden comunicarse con los otros, bien sea, directa o indirectamente a través de uno o más nodos intermediarios [8]

2.2.2. Características

La red de Internet como la conocemos es el ejemplo más conocido y similar a una red mallada, puesto que Internet es una red con gran cantidad de nodos en los que el mensaje es enviado desde un punto y es recibido en otro mediante un enrutado inherente a la red. Tomando este ejemplo como base se pueden delimitar algunas características de esta topología de red:

- Permite que el camino recorrido por el mensaje entre un punto y otro sea dinámico, es decir, que la ruta que toma el mensaje cambie si se requiere debido a la ocurrencia de algún evento que impidan la comunicación con un nodo intermedio específico.
- Ofrece una mayor cantidad de rutas posibles para el mensaje ya que idealmente cada nodo puede conectarse con cualquier otro directamente o a través de terceros.
- Cada nodo posee un identificador único en la red que lo diferencia de los demás nodos para que no haya errores en el direccionamiento de la información.

2.2.2.1. Sobre los protocolos de enrutamiento en las redes Malladas

Esta capacidad de elegir la ruta debe estar basada en algoritmos para determinar el camino óptimo que será recorrido por el mensaje. Este algoritmo de enrutamiento debe tomar en cuenta las distintas condiciones que puede presentar el medio de transmisión, las interferencias o ruido que pueda existir según la banda de transmisión, la posible colisión de datos en un nodo, etc, para así determinar hacia dónde debe ir el mensaje en cada nodo para llegar a su destino.

La implementación de topologías malladas en sistemas embebidos ha encontrado problemas en la necesidad de procedimientos adicionales relacionados con el enruteamiento. Hay algunos ejemplos de protocolos que soportan la red mallada sobre una red IP, por ejemplo el protocolo B.A.T.M.A.N (Better Approach To Mobile Adhoc Networking), Babel (Protocolo de distancia de vector para IPv6 y IPv4 con propiedades de convergencia rápida), HWMP (Protocolo Híbrido Inalámbrico Mallado), entre otros [7]. El problema surge en la implementación de la pila TCP/IP en estos sistemas puesto que un sistema embebido posee recursos de cómputo limitados. Es por ello que se debe seleccionar según la aplicación el poder de cómputo necesario y la topología de la red que se requiere para cumplir con cada solución. Se describirán algunas redes WiFi malladas, con funcionamiento caracterizado según la aplicación:

Para que el mensaje tenga un destinatario debe existir una manera de identificar cada nodo de una red mallada, en el caso de la red mallada WiFi se utiliza la dirección MAC para diferenciar un nodo de otro.

2.2.2.2. Direcccionamiento por MAC

En una red de computadoras, la dirección MAC es un valor único asociado a un adaptador de red. La dirección MAC también es conocida como dirección de hardware o dirección física del adaptador [7]. Las direcciones MAC se componen de doce números hexadecimales (48 bits de longitud). Por convención las direcciones MAC son escritas en uno de los dos formatos siguientes:

$$MM : MM : MM : SS : SS : SS \quad o \quad MM-MM-MM-SS-SS-SS \quad (2.1)$$

La primera mitad de la dirección MAC contiene el número identificador del fabri-

cante del adaptador (i.e. 00:A0:C9:14:C8:29). Dichos identificadores están regulados por un estándar de Internet. La segunda parte de la dirección MAC representa el número de serial asignado al adaptador por el fabricante. [7]

2.3. La red mallada ESP-MESH

2.3.1. Definición

Una red Wi-Fi de infraestructura tradicional es una red punto a multipunto donde un único nodo central conocido como punto de acceso (AP) está conectado directamente a todos los demás nodos conocidos como estaciones (STA). El AP es responsable de arbitrar y reenviar las transmisiones entre las estaciones. Algunos AP también retransmiten transmisiones hacia/desde una red IP externa a través de un enrutador. Las redes Wi-Fi de infraestructura tradicional sufren la desventaja de un área de cobertura limitada debido al requisito de que cada estación debe estar dentro del alcance para conectarse directamente con el AP. Además, las redes Wi-Fi tradicionales son susceptibles de sobrecarga ya que el número máximo de estaciones permitidas en la red está limitado por la capacidad del AP.

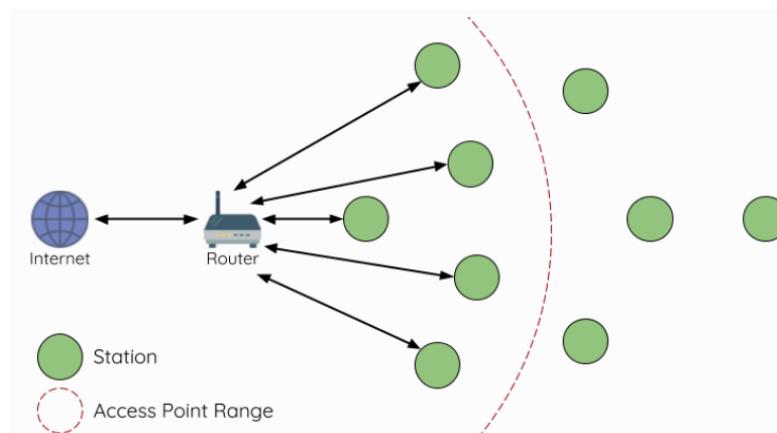


Figura 2.4. Ilustración de una red Wi-Fi tradicional. Tomada desde [1]

Proporcionada por espressif la red mallada ESP-MESH, se diferencia de las redes Wi-Fi de infraestructura tradicional en que no es necesario que los nodos se conecten a un nodo central. En cambio, los nodos pueden conectarse con nodos vecinos. Los nodos son mutuamente responsables de retransmitir las transmisiones de los demás. Esto permite que dicha red tenga un área de cobertura mucho mayor, ya que los nodos aún pueden lograr la interconectividad sin necesidad de estar dentro del alcance del nodo central. Asimismo, ESP-MESH también es menos susceptible a sobrecargas, ya que el número de nodos permitidos en la red ya no está limitado por un solo nodo central. [1]

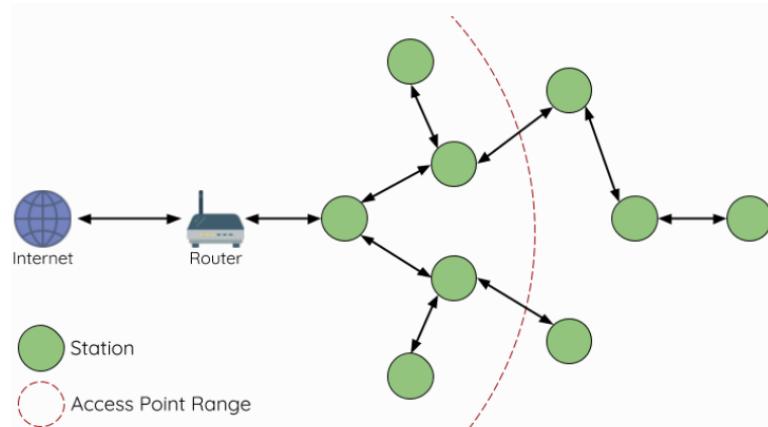


Figura 2.5. Ilustración de una red Wi-Fi ESP-MESH. Tomada desde [1]

2.3.2. Terminología en la ESP-MESH

- **Nodo:** Cualquier dispositivo que es o que puede ser parte de una red ESP-MESH.
- **Nodo raíz (root, root node):** Es el nodo cima de la red, este nodo se conecta a la puerta de enlace.
- **Nodo hijo (child, child node):** Es un nodo X que establece conexión con un nodo Y. Dicho nodo Y se encuentra más cerca del nodo raíz que el nodo X.
- **Nodo padre (padre, parent node):** Es un nodo Y que establece conexión con

un nodo X, el nodo X se encuentra más lejos del nodo raíz que el nodo Y.

- **Nodo descendiente (descendant node):** Cualquier nodo accesible mediante la repetición del proceso padre a hijo.
- **Nodos hermanos (sibling nodes):** Nodos que comparten el mismo nodo padre.
- **Conexión:** Una asociación Wi-Fi tradicional entre un AP y una estación. Un nodo en ESP-MESH utilizará su interfaz de estación para asociarse con la interfaz softAP de otro nodo, formando así una conexión. El proceso de conexión incluye los procesos de autenticación y asociación en Wi-Fi.
- **Salto inalámbrico (wireless hop):** La parte de la ruta entre el nodo de origen y de destino para un mensaje que corresponde a una única conexión inalámbrica. Un paquete de datos que atraviesa una sola conexión se conoce como salto único, mientras que atravesar varias conexiones se conoce como salto múltiple.
- **Subred:** Subdivisión de una red ESP-MESH que consta de un nodo y todos sus nodos descendientes. Por lo tanto, la subred del nodo raíz consta de todos los nodos de una red ESP-MESH.

2.3.3. Topología de árbol sobre la red ESP-MESH

ESP-MESH está construido sobre la infraestructura del protocolo Wi-Fi y se puede considerar como un protocolo de red que combina muchas redes Wi-Fi individuales en una sola WLAN. En Wi-Fi, las estaciones se limitan a una sola conexión con un AP (conexión ascendente) en cualquier momento, mientras que un AP se puede conectar simultáneamente a varias estaciones (conexiones descendentes). Sin embargo, ESP-MESH permite que los nodos actúen simultáneamente como una estación y un AP. Por lo tanto, un nodo en ESP-MESH puede tener múltiples conexiones descendentes usando su interfaz softAP, mientras que simultáneamente tiene una única conexión

ascendente usando su interfaz de estación. Esto, naturalmente, da como resultado una topología de red de árbol con una jerarquía de padres e hijos que consta de varias capas.

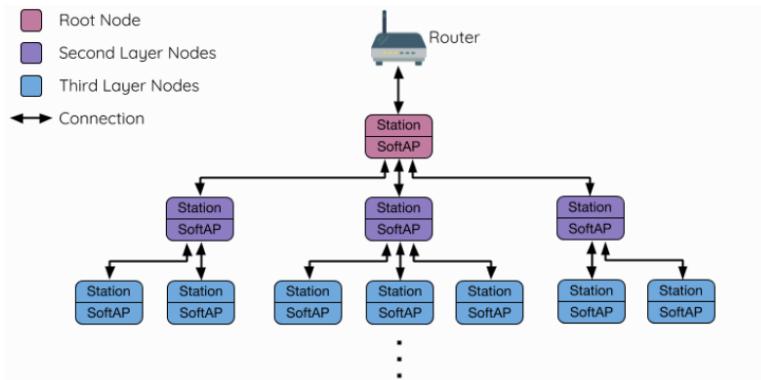


Figura 2.6. Topología de árbol observada en la red ESP-MESH luego de su establecimiento. Tomada de [1]

ESP-MESH es una red de múltiples saltos (multi-hop), lo que significa que los nodos pueden transmitir paquetes a otros nodos en la red a través de uno o más saltos inalámbricos. Por lo tanto, los nodos en ESP-MESH no solo transmiten sus propios paquetes, sino que también sirven como retransmisores para otros nodos. Siempre que exista una ruta entre dos nodos cualesquiera en la capa física (a través de uno o más saltos inalámbricos), cualquier par de nodos dentro de una red ESP-MESH puede comunicarse.

2.3.4. Tipos de nodos en la red ESP-MESH

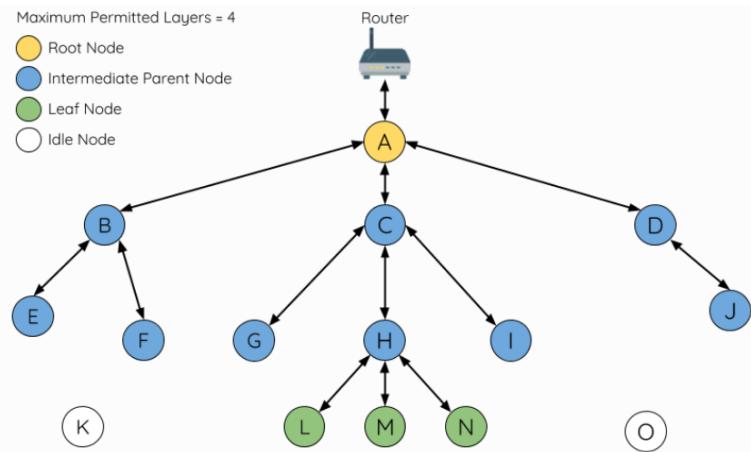


Figura 2.7. Tipos de nodos en una red ESP-MESH. Tomada desde [1]

- **Nodo raíz:** Es el nodo superior de la red y sirve como la única interfaz entre la red ESP-MESH y una red IP externa. El nodo raíz está conectado a un enrutador Wi-Fi convencional y transmite paquetes hacia / desde la red IP externa a los nodos dentro de la red ESP-MESH. Solo puede haber un nodo raíz dentro de una red ESP-MESH y la conexión ascendente del nodo raíz solo puede ser con el enrutador. Con referencia al diagrama anterior, el nodo A es el nodo raíz de la red.
- **Nodo hoja:** Es un nodo al que no se le permite tener nodos secundarios (sin conexiones descendentes). Un nodo hoja solo puede transmitir o recibir sus propios paquetes, pero no puede reenviar los paquetes de otros nodos. Si un nodo está situado en la capa máxima permitida de la red, se asignará como nodo hoja. Esto evita que el nodo forme conexiones descendentes, lo que garantiza que la red no agregue una capa adicional. Algunos nodos sin una interfaz softAP (solo estación) también se asignarán como nodos hoja debido al requisito de una interfaz softAP para cualquier conexión descendente. Con referencia al diagrama anterior, los nodos L / M / N están situados en la capa máxima permitida de la red, por lo que se han asignado como nodos hoja.

- **Nodos padres intermedios:** Los nodos conectados que no son ni el nodo raíz ni un nodo hoja son nodos padres intermedios. Un nodo intermedio debe tener una única conexión ascendente (un único nodo padre), pero puede tener de cero a varias conexiones descendentes (de cero a varios nodos hijos). Por lo tanto, un nodo padre intermedio puede transmitir y recibir paquetes, pero también reenviar paquetes recibidos desde sus conexiones ascendentes y descendentes. Haciendo referencia al diagrama anterior, los nodos B a J son nodos padres intermedios. Los nodos padres intermedios sin conexiones descendentes, como los nodos E / F / G / I / J, no son equivalentes a los nodos hoja, ya que todavía se les permite formar conexiones descendentes en el futuro.
- **Nodos ociosos:** Los nodos que aún no se han unido a la red se asignan como nodos inactivos. Los nodos inactivos intentarán formar una conexión ascendente con un nodo padre intermedio o intentarán convertirse en el nodo raíz en las circunstancias correctas. Refiriéndose al diagrama anterior, los nodos K y O son nodos inactivos.

2.3.5. Señales de reconocimiento y límites de intensidad de señal (RSSI)

Cada nodo en ESP-MESH que pueda formar conexiones descendentes (es decir, tiene una interfaz AP) enviará periódicamente tramas beacon Wi-Fi. Los nodos utilizan tramas beacon para permitir que otros nodos detecten su presencia y conozcan su estado. Los nodos inactivos escucharán las tramas beacon para generar una lista de nodos padres potenciales, desde dicha lista el nodo inactivo formará una conexión ascendente.

La intensidad de la señal de una posible conexión ascendente está representada por el RSSI (Indicación de intensidad de la señal recibida) contenido en las tramas beacon del posible nodo padre. Para evitar que los nodos formen una conexión ascendente dé-

bil, ESP-MESH implementa un mecanismo de umbral RSSI para las tramas beacon. Si un nodo detecta una trama beacon con un RSSI por debajo del umbral preconfigurado, el nodo transmisor no se tendrá en cuenta para formar una conexión ascendente.

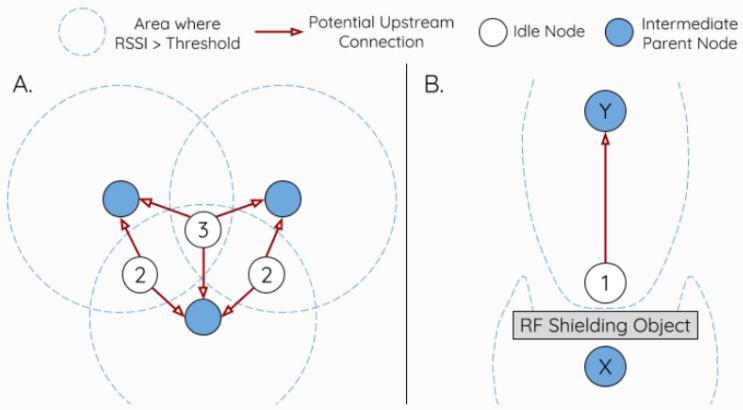


Figura 2.8. Ejemplo ilustrativo sobre el umbral RSSI. Tomado de [1]

En la imagen anterior 2.8 el recuadro A ilustra la cantidad de nodos padres posibles que tiene cada nodo según el RSSI que recibe. En el recuadro B de dicha imagen se observa cómo un objeto que reduce el alcance de las transmisiones del nodo X afecta en la elección del parent, pues al tener el nodo Y un mayor RSSI en el nodo acaba siendo este el elegido para la conexión ascendente a pesar de encontrarse más lejos físicamente.

2.3.6. Tablas de enrutamiento

Cada nodo dentro de una red ESP-MESH mantendrá su tabla de enrutamiento individual utilizada para enrutar correctamente los paquetes ESP-MESH al nodo de destino correcto. La tabla de enrutamiento de un nodo en particular constará de las direcciones MAC de todos los nodos dentro de la subred del nodo en particular (incluida la dirección MAC del propio nodo en particular). Cada tabla de enrutamiento está

dividida internamente en múltiples subtablas y cada subtabla corresponde a la subred de cada nodo hijo.

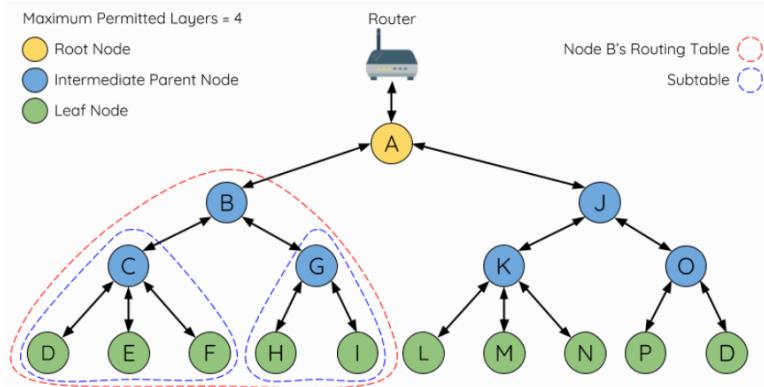


Figura 2.9. Ilustración de una red mallada, las tablas y subtablas de enruteamiento.
Tomado de [1]

Usando el diagrama anterior como ejemplo, la tabla de enruteamiento del nodo B consistiría en las direcciones MAC de los nodos B a I (es decir, equivalente a la subred del nodo B). La tabla de enruteamiento del nodo B está dividida internamente en dos subtablas que contienen los nodos C a F y los nodos G a I (es decir, equivalentes a las subredes de los nodos C y G respectivamente).

La red utiliza las tablas de enruteamiento para determinar si un paquete debe reenviarse en sentido ascendente o descendente según las siguientes reglas:

1. Si la dirección MAC de destino del paquete está dentro de la tabla de enruteamiento del nodo actual y no es el nodo actual, seleccione la subtabla que contiene la dirección MAC de destino y reenvíe el paquete de datos en sentido descendente al nodo secundario correspondiente a la subtabla.

- Si la dirección MAC de destino no se encuentra dentro de la tabla de enrutamiento del nodo actual, reenvíe el paquete de datos en sentido ascendente al nodo principal del nodo actual. Si lo hace repetidamente, el paquete llegará al nodo raíz, donde la tabla de enrutamiento debe contener todos los nodos dentro de la red.

2.4. Sistema operativo en tiempo real (RTOS)

Un sistema operativo es un conjunto de órdenes y programas de un sistema informático que gestiona los recursos de hardware y provee servicios que permiten el funcionamiento de otros programas.



Figura 2.10. Interacción del OS con otras capas del sistema informático.

La ventaja de utilizar sistemas operativos es que estos permiten gestionar los recursos de modo que se puedan ejecutar múltiples órdenes al mismo tiempo. Aunque en realidad cada procesador solo está ejecutando una orden a la vez, el sistema operativo es esa capa de abstracción que simula la ejecución en paralelo de varias tareas.

2.4.1. FreeRTOS

FreeRTOS es una implementación de un sistema RTOS que está diseñado para ser lo suficientemente liviano para correr en un microcontrolador. Este posee interfaces de programación que proveen planificadores en tiempo real, comunicación entre tareas, temporizadores y sincronización. Lo que quiere decir que una descripción más exacta sería tratarlo como un kernel en tiempo real. El planificador usado en el FreeRTOS, logra determinismo, proporcionando a los usuarios la capacidad de asignar prioridad a cada tarea de ejecución.

2.4.1.1. Tareas

Las tareas se implementan como funciones. Cada tarea es un pequeño programa en sí mismo. Tiene un punto de entrada, se ejecuta normalmente de forma continua en un bucle infinito y no se cierra.

2.4.1.2. Colas

Las colas proporcionan mecanismo de comunicación de una a otra tarea, de una tarea a una interrupción y de una interrupción a una tarea. La comunicación se basa en la transferencia en un dato ubicado en la cola.

2.4.1.3. Semáforos

Los semáforos son muy similares a variables booleanas con la diferencia de que estos son utilizados para bloquear o desbloquear tareas, esperar a que ocurran eventos o cambios de un valor lógico y pueden ser activados desde una interrupción.

2.4.1.4. Grupos de eventos

Los grupos de eventos son espacios de memoria de hasta 23 bits en los que cada bit representa una variable booleana que puede utilizarse como una bandera para sincronizar tareas, bloquearlas o desbloquearlas. También puede ser llamada desde interrupciones.

2.5. Microcontrolador ESP32

2.5.1. Descripción

El ESP-32 es un chip con integración WiFi y Bluetooth diseñado con la tecnología de ultra bajo consumo de 40 nm. Este microcontrolador esta diseñado para aplicaciones móviles, electrónicos personales, y de Internet de las cosas (IoT). Posee características de bajo consumo, incluyendo reloj de alta precisión, múltiples estados de energía, y escalamiento de consumo dinámico.

Además es una solución integrada, que ya posee WiFi, Bluetooth, junto con alrededor de 20 componentes externos. El chip incluye una interruptor de antena, acoplador de radio-frecuencia, amplificador de potencia, amplificador de recepción de bajo ruido, filtros, y módulos de administración de consumo.

Este microcontrolador posee dos núcleos CPU que pueden ser individualmente controlados, con un reloj ajustable entre 80 MHz y 120 MHz. Además, usa el sistema operativo en tiempo real freeRTOS y tensión de alimentación nominal de 3,3 V.

2.5.2. Características

2.5.2.1. WiFi

- 802.11 b/g/n.
- 802.11 n (hasta 150Mbps).
- WMM.
- TX/RX A-MPDU, RX A-MSDU.
- Bloque de ACK inmediato.
- Defragmentación.
- Monitorización de faro automático (Hardware TSF).
- 4 interfaces virtuales WiFi.
- Soporte simultáneo para estación, Punto de acceso y modo promiscuo.
- Diversidad de antena.

2.5.2.2. CPU y Memoria

- Doble núcleo Xtensa de 32 bits, hasta 600 MIPS.
- 448 kB ROM.
- 520 kB SRAM.
- 16 kB SRAM en RTC.
- Memoria flash embebida de hasta 16MB

2.5.2.3. Relojes y temporizadores

- Oscilador interno con calibración de 8MHz, RC
- Oscilador externo de cristal desde 2 a 60MHz.
- Dos grupos de temporizadores, incluyendo 2x64-bits con un perro guardián en cada uno.
- TX/RX A-MPDU, RX A-MSDU.
- Un temporizador y un perro guardián RTC.

2.5.2.4. Periféricos

- 34 GPIO mapeables.
- ADC de 12 bits con hasta 18 canales.
- Do convertidores de digital a analógico
- 10 sensores táctiles.
- 4 SPI, 2 I^2C , 3 UART
- 1 host (SD/eMMC/SDIO)
- Interfaz de MAC Ethernet con DMA dedicado y soporte IEEE 1588.
- CAN 2.0.
- Soporte simultaneo para estación, Punto de acceso y modo promiscuo.
- IR (TX/RX).
- Motor PWM.

- LED PWM hasta de 16 canales
- Sensor Hall.

2.5.2.5. Seguridad

- Modo Boot seguro.
- Encriptación de flash.
- 1024-bits OTP, hasta 768-bit clientes
- Aceleración de criptografía por hardware:
 - AES.
 - SHA-2
 - RSA
 - ECC
- 4 SPI, 2 I^2C , 3 UART
- 1 host (SD/eMMC/SDIO)
- Interfaz de MAC Ethernet con DMA dedicado y soporte IEEE 1588.
- CAN 2.0.
- Soporte simultaneo para estación, Punto de acceso y modo promiscuo.
- IR (TX/RX).
- Motor PWM.
- LED PWM hasta de 16 canales
- Sensor Hall.

CAPÍTULO III

DEFINICIÓN Y DESCRIPCIÓN DEL HARDWARE

3.1. Definición del hardware

Un sistema de adquisición y transmisión de datos orientado a medidores de energía, debe contar con varios elementos que, interconectados, lleven a un funcionamiento general.

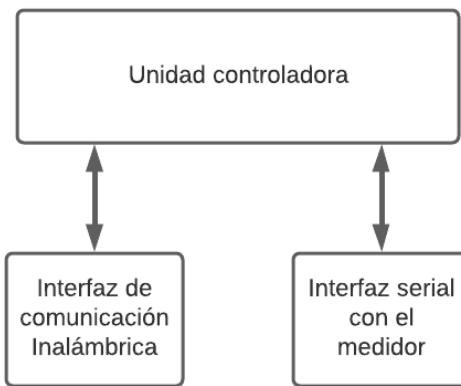


Figura 3.1. Diagrama general de los componentes del Hardware.

Dicho sistema requiere de una interfaz de comunicación inalámbrica debido a la necesidad de interconectar los distintos dispositivos que formarán los nodos de la red que representaría la parte de transmisión. Si se desea extraer el contador de energía del medidor también requiere de una interfaz serial para interactuar con este, esta sería la parte ,de adquisición de los datos.

Por último se requiere de una unidad controladora o cerebro para que se maneje el flujo de datos, el almacenamiento, tramas y demás rutinas necesarias para el correcto funcionamiento de las interfaces. Es de resaltar que tanto la interfaz de comunicación inalámbrica como la serial pueden funcionar como entradas y salidas de datos, para con la red y con el medidor, respectivamente.

3.2. Descripción del hardware

3.2.1. Interfaz de comunicación inalámbrica

La interfaz de comunicación inalámbrica es imprescindible para este sistema. Para construir una red mallada WiFi el equipo elegido como nodo de la red debe ser compatible con esta tecnología. El ESP32 tiene integrada una antena compatible con los protocolos 802.11 b/g/n y la circuitería necesaria para implementar y manejar este tipo de interfaz sin necesidad de otro dispositivo.



Figura 3.2. Microcontrolador ESP32 con su antena WiFi integrada.

3.2.2. Interfaz serial con el medidor

El sistema debe tener la capacidad de extraer los datos de un medidor de energía para luego ser compartidos con el resto de la red y con el cliente externo, esta interfaz se encarga de la extracción de los datos.

Para lograr esto se debe interactuar con el medidor conectado mediante comunicación serial, en la búsqueda de la mayor compatibilidad del sistema con los medidores existentes y nuevos se crearon dos

Buscando tener la mayor compatibilidad con todos los tipos de medidores se plantea utilizar dos maneras de comunicarse con el medidor: Un bus serial y mediante la salida de calibración, con estas se cubriría la mayoría de los equipos.

3.2.2.1. Bus serial

Los medidores de energía con los que se busca compatibilidad en este apartado son los que manejan un protocolo Modbus en un bus serial RS485. Por esto se plantea utilizar un chip MAX345 que funcione como intermediario entre la comunicación serial del UART del microcontrolador y el bus de campo.

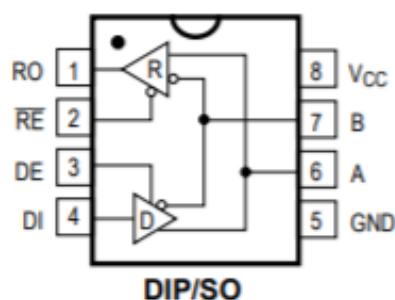


Figura 3.3. Diagrama de pines del encapsulado DIP8 para MAX345.

El fabricante Espressif ofrece algunas sugerencias a la hora de trabajar con chip seriales para RS485, de igual forma el fabricante del chip nos ofrece un ejemplo de aplicación. Como se observa en la siguientes imágenes:

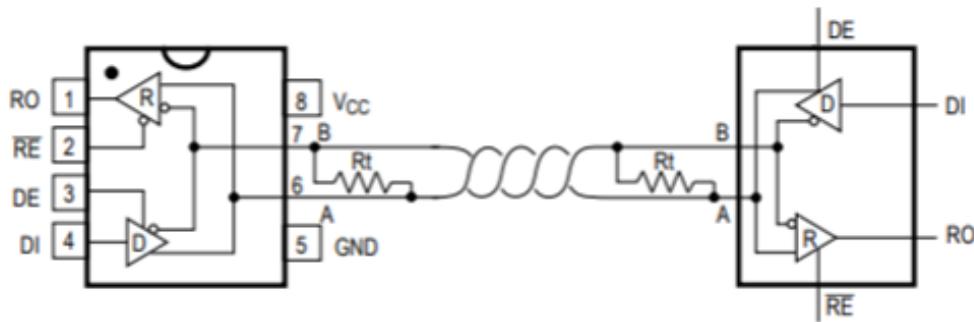


Figura 3.4. Configuración recomendada por el fabricante del chip.

RS485 example connection circuit schematic:

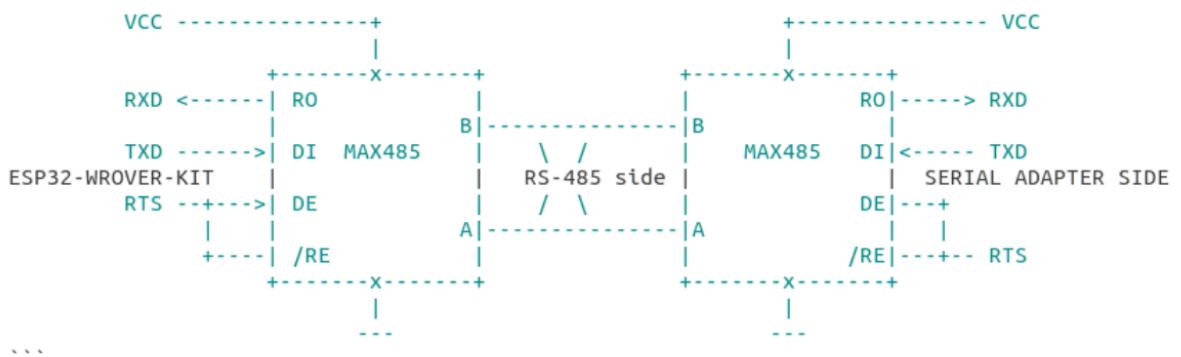


Figura 3.5. Configuración recomendada por Espressif para el manejo de un chip MAX-485.

La implementación final es semejante a la recomendada por Espressif en 3.5 pero colocando la resistencia RT de valor 150 Ohms como lo recomienda el fabricante del chip en 3.4.

Se colocó un condensador de 100 nF entre alimentación y tierra junto con el chip MAX3485 en un circuito impreso para poder ser utilizado en la implementación del prototipo del sistema, que se hará en una protoboard.

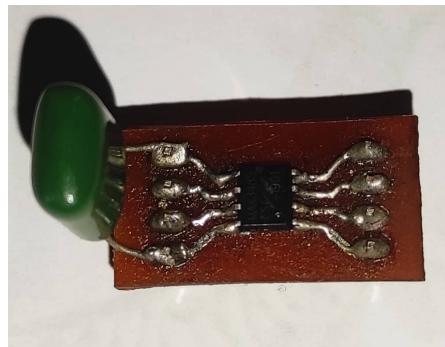


Figura 3.6. Circuito impreso con un condensador y el chip MAX3485 utilizada para la comunicación mediante el bus serial en los medidores necesarios.

3.2.2.2. Salida de calibración

La salida de calibración se utiliza en la fabricación de los medidores para verificar y corregir el parámetro de conversión de imp/kWh que tiene el equipo. Esta salida suele estar formada por un optoacoplador que envía un pulsos cada cierta fracción de kWh , estos pulsos se pueden captar desde la unidad controladora y aprovecharlos para extraer el conteo de energía.

Pero se deben tener en cuenta dos cosas: el medidor puede tener una condición inicial de energía que no se captará y, de ocurrir una falla de alimentación se perderá la cuenta que se lleva si no se guarda de manera permanente. Ambos problemas se tratarán mediante el software y se explicará la solución propuesta más adelante.

Para utilizar dicha salida se utiliza un transistor y una resistencia de pull-up para evitar dañar las entradas del microcontrolador con un voltaje o corriente mayor a la soportada.

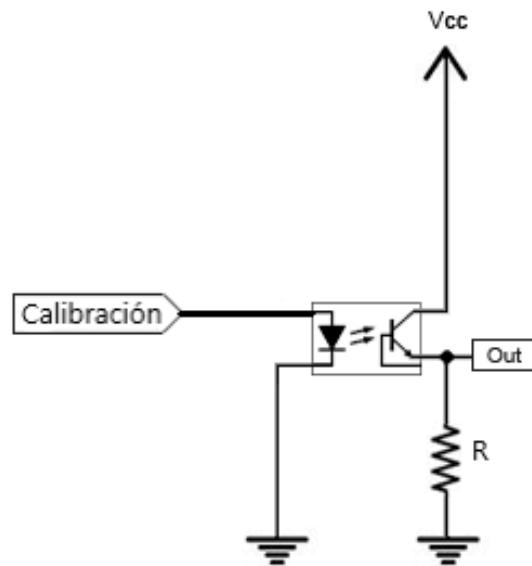


Figura 3.7. Topología utilizada en la salida opto acoplada del medidor de energía.

3.2.3. Unidad controladora

La unidad controladora es la encargada de: gestionar el estado del nodo en la red mallada, recibir y transmitir los datos, controlar los periféricos para enviar el mensaje necesario a través de la interfaz serial al medidor y además de comunicarse enviar al exterior de la red mallada el resultado de esta adquisición de los datos.

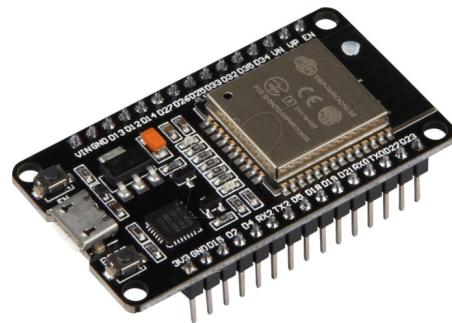


Figura 3.8. Unidad controladora del sistema, tarjeta de desarrollo ESP32.

El cerebro de cada nodo tiene muchas tareas pero no todas se ejecutan al mismo

tiempo, muchas dependen del nodo y de la configuración que se le da a este mediante el software. La forma de manejar de manera correcta las tareas en cada nodo será ilustrada en el siguiente capítulo.

3.2.4. Circuito impreso

Considerando las necesidades expuestas anteriormente se realizó el diseño de un circuito impreso. En este se agregó la circuitería necesaria para la salida de calibración con el optoacoplador, el módulo RS485, todo lo necesario para iniciar la unidad controladora ESP32 y un módulo de alimentación AC/DC con entrada en 120V y salida a 5V y luego un DC/DC de 5V a 3.3v ya que el ESP32 utiliza una lógica de 3.3v. Primero se presentan los esquemáticos:

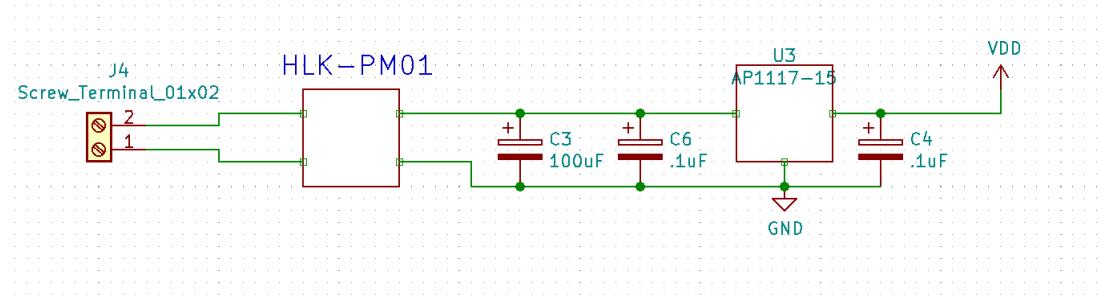


Figura 3.9. Esquemático del módulo de alimentación.

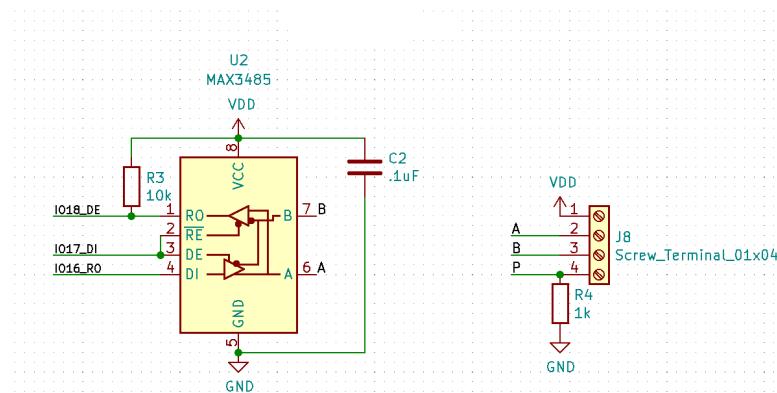


Figura 3.10. Esquemáticos de interfaces con los medidores.

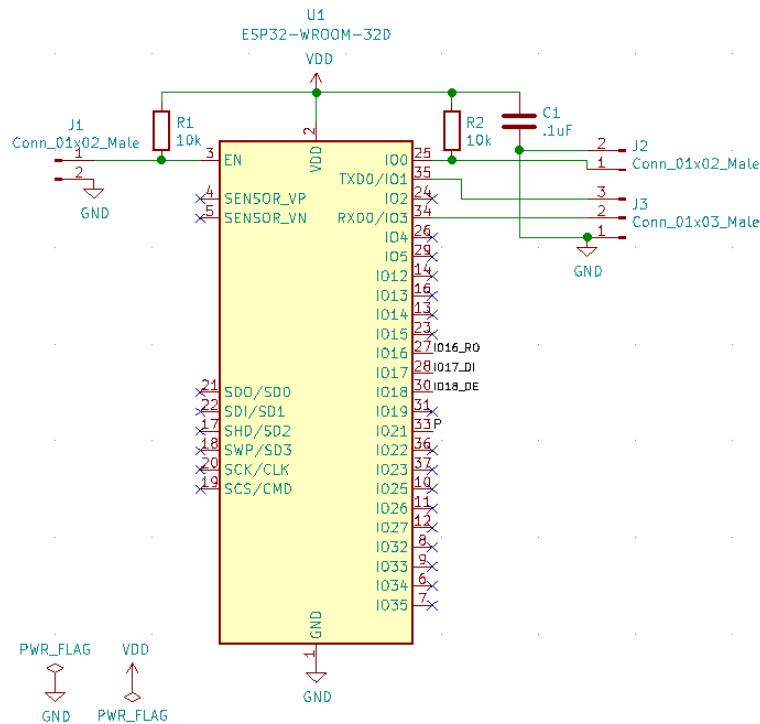


Figura 3.11. Esquemático de la unidad controladora implementada con el ESP32.

Es de resaltar que se buscó adaptar el tamaño de la tarjeta dentro de una contendor con cerramientos industriales, es por esta razón que tiene ese corte específico. Seguidamente, las visuales del diseño del circuito impreso:

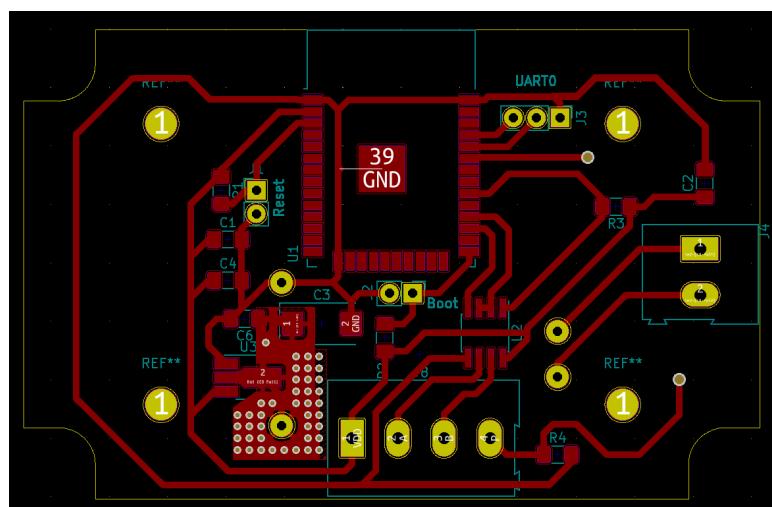


Figura 3.12. Imagen del diseño frontal de la PCB utilizando KiCad.

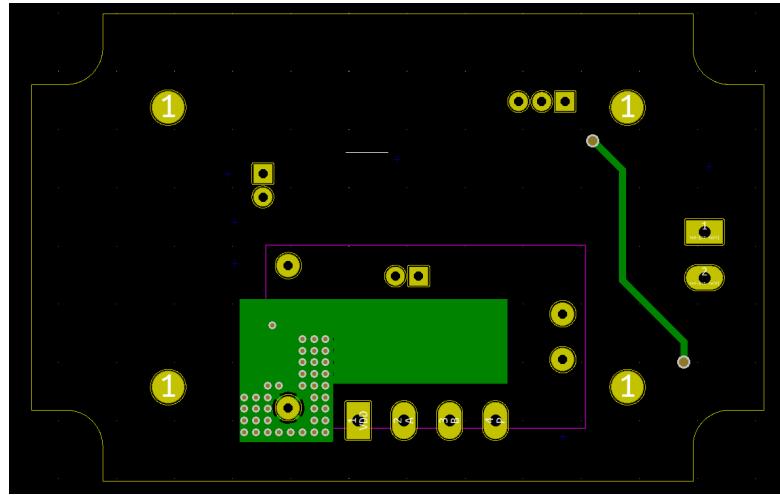


Figura 3.13. Imagen del diseño posterior de la PCB utilizando KiCad.

Y por último unas imágenes renderizadas de la tarjeta, junto con la lista de materiales en ella:

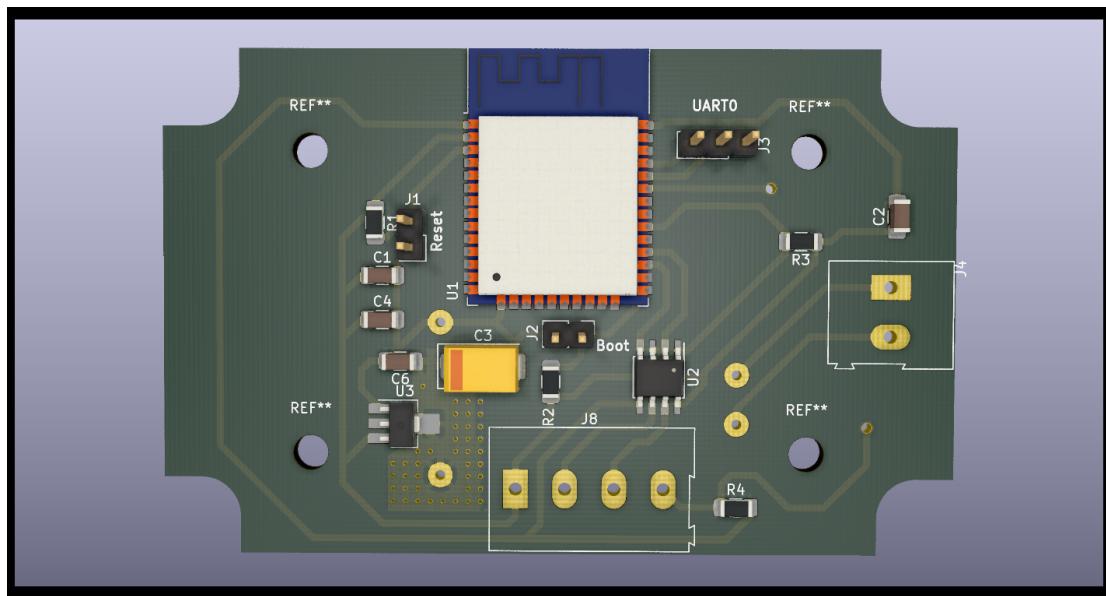


Figura 3.14. Imagen de la capa frontal de la PCB renderizada.

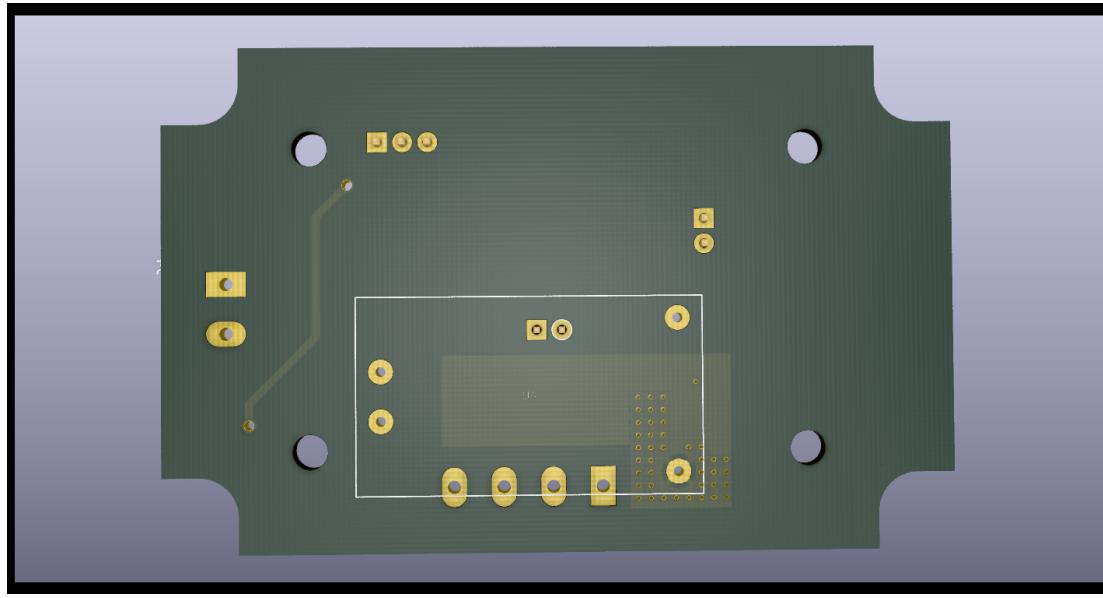


Figura 3.15. Imagen de la capa posterior de la PCB renderizada.

Tabla 3.1. Lista de materiales utilizados en el diseño de la PCB

Identificador	Descripción	Cantidad
J1,J2	Conn_01x02_Male	2
C1,C2,C4,C6	.1uF	4
U4	HLK-PM01	1
R4	1k	1
R3,R1,R2	10k	3
U1	ESP32-WROOM-32D	1
1,2,3,4	MountingHole_3.5mm	4
J3	Conn_01x03_Male	1
J4	Screw_Terminal_01x02	1
C3	100uF	1
J8	Screw_Terminal_01x04	1
U3	AP1117-15	1
U2	MAX3485	1

CAPÍTULO IV

DEFINICIÓN Y DESCRIPCIÓN DEL SOFTWARE

4.1. Definición del software

Para el objetivo planteado en este proyecto, se requiere el uso de un microcontrolador por lo que se debe implementar un programa o firmware para configurar y utilizar, de manera correcta, los periféricos correspondientes a las conexiones inalámbricas y el manejo de datos seriales; además es necesario un protocolo de conexión hacia un agente externo que permita enviar los datos recolectados por la infraestructura, una infraestructura que debe ser creada mediante este firmware y como método de configuración de los nodos una interfaz gráfica de usuario (GUI) que le permita a la red adaptarse de manera sencilla al lugar donde se instale.

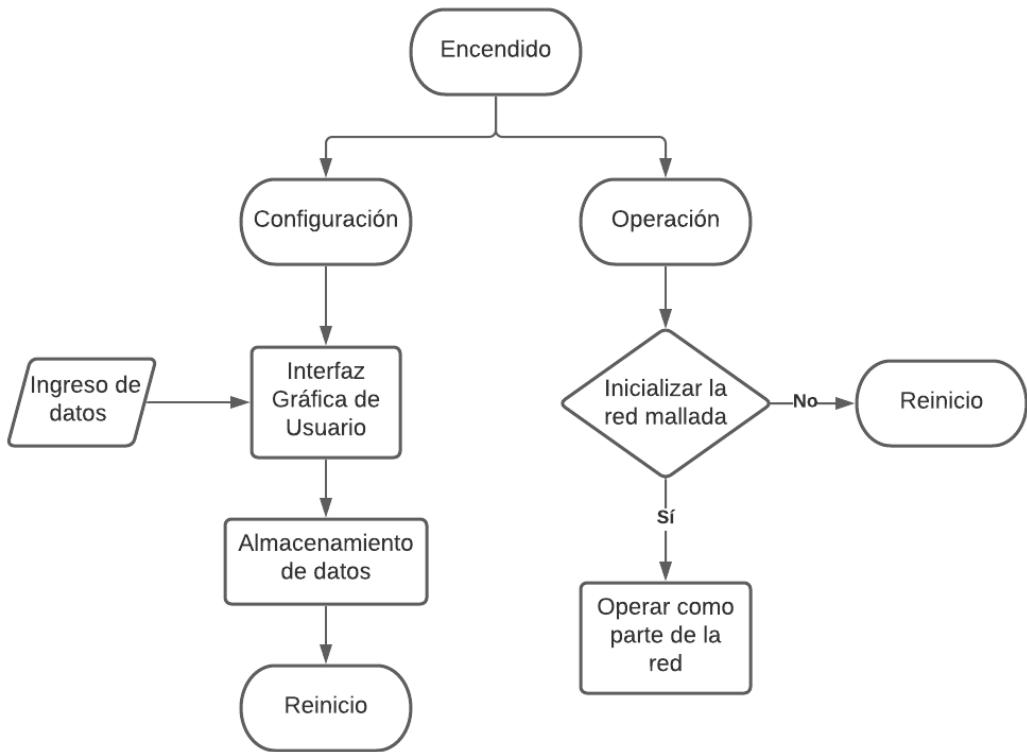


Figura 4.1. Diagrama de bloques genérico del sistema

El firmware en el microcontrolador (ESP32) debe ser capaz de iniciar en dos modos: el modo de operación y el modo de configuración. El modo de inicio debe ser controlado por el usuario.

La rutina de configuración debe ofrecer al usuario una interfaz gráfica donde se pueda modificar todos los parámetros que sean necesarios para establecer la conexión con el punto de acceso local, del mismo modo se debe conectar la red con el agente externo y realizar el envío de datos; los parámetros de funcionamiento de la parte serial de cada nodo para extraer desde los medidores la información y los necesarios en la red mallada para establecer la cantidad de conexiones y el registro de nodos en la red; además del usuario y contraseña de acceso a dicha página.

Por otra parte, la rutina de configuración debe detenerse en caso de que faltase algún parámetro fundamental para el funcionamiento, en caso contrario debe ir activando etapa por etapa los procesos para: configurar los periféricos necesarios, activar la conexión inalámbrica, conectarse con el punto de acceso y registrarse en la red mallada.

Las operaciones anteriores son compartidas en todos los nodos y son necesarias para entrar en la red. Las siguientes rutinas a ejecutarse dependen de la jerarquía que posee el nodo en la red mallada y de la función que se haya configurado para el nodo en la interfaz gráfica. La jerarquía que tiene un nodo dependerá de la intensidad de señal de este respecto al punto de acceso.

En caso de ser el nodo de mayor jerarquía, dicho nodo será el encargado de gestionar la comunicación de la red con el exterior, este nodo servirá como el enlace para enviar y recibir la información para toda la red. Además a partir de este nodo se formará el resto de la topología de red mallada.

Los demás nodos deben ser repetidores para que la señal pueda abarcar el espacio físico necesario o nodos conectados a medidores para la extracción de datos. Estas dos rutinas no deben ser excluyentes, los nodos deben tener la capacidad de ser repetidores y extraer datos al mismo tiempo.

Hay dos rutinas principales para la extracción de datos, una mediante la salida de calibración del medidor y otra que se conecta a un bus serial RS485. La rutina de extracción de datos debe estar configurada para que se utilice el método necesario según

el medidor. En caso de que los datos sean extraídos por la salida de calibración del medidor se debe diseñar una rutina para contar y almacenar los pulsos que esta genera considerando la cantidad inicial de kWh, esto por ser el registro que se desea transmitir debe estar almacenado de manera persistente y debe estar protegido contra pérdidas de alimentación. Por el contrario si en el nodo los datos son adquiridos mediante un bus serial, la persistencia de datos es realizada por el medidor, entonces la rutina en este caso solo debe encargarse de la trama serial. En ambos casos una vez adquiridos los datos se deben enviar a la red exterior cuando sean solicitados.

4.2. Descripción del software

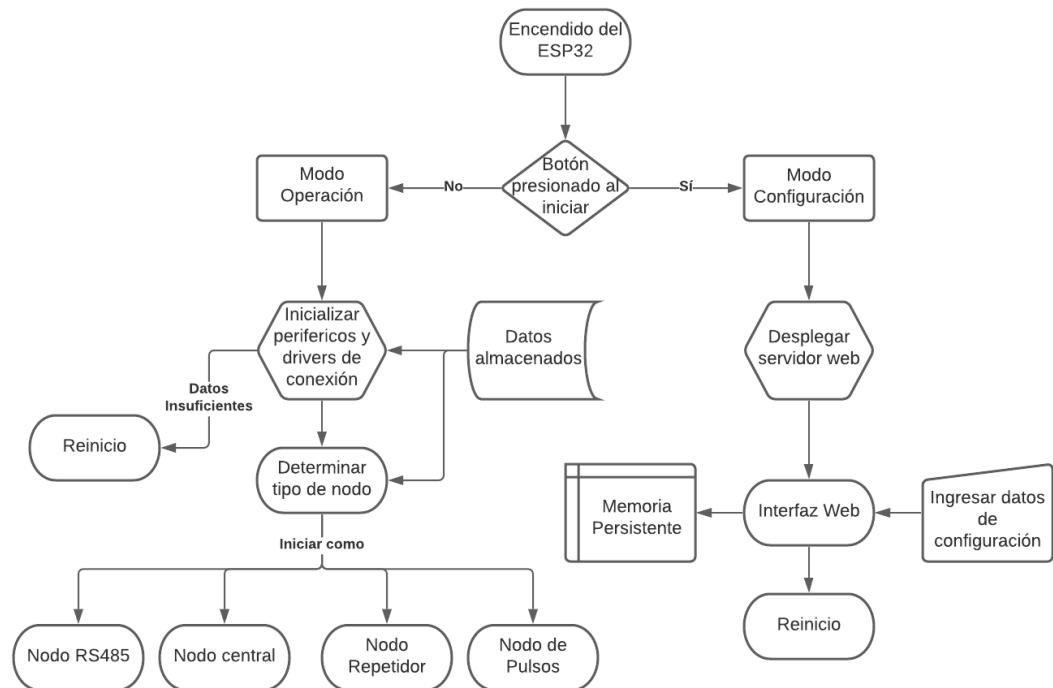


Figura 4.2. Diagrama general de funcionamiento ilustrado.

4.2.1. Modo de configuración

El modo de configuración utiliza un sistema de archivos para implementar un servidor web que soporta protocolos http. Este sistema de archivos en el microcontrolador se trabaja con la API de *SPIFFS* que ofrece el fabricante, esto permite grabar en la memoria flash del microcontrolador los diferentes archivos necesarios para implementar una página web como interfaz gráfica de usuario. Sin embargo para que esto ocurra se requiere en la configuración de inicio definir un espacio en una tabla de particiones para toda esta estructura.

4.2.1.1. Tablas de particiones en el ESP32

Un código cualquiera en el ESP32 puede contener múltiples aplicaciones, así como muchos tipos diferentes de datos (datos de calibración, sistemas de archivos, almacenamiento de parámetros, etc.) es por esto que las tablas de particiones se encuentran con un offset para cada código en cuestión en la memoria flash.

En cada grabado se utiliza un archivo de tabla de particiones por defecto que es definida en el menú de configuraciones también llamado *menuconfig*, hay dos tablas modelos para utilizar en este menú: *Aplicación única de fábrica* (4.1) y *Aplicación de dos definiciones*. La diferencia entre ambas tablas radica en la cantidad de particiones en la memoria flash y cómo son utilizadas: La aplicación única solo posee un código para arranque de tipo *app* en la tabla de particiones, mientras que la tabla de dos definiciones posee dos códigos para iniciar el micro y es el gestor de arranque mediante ciertas reglas el que decide cuál data de tipo *app* poner en marcha. Este último modelo de tabla es útil cuando se quiere incorporar actualizaciones inalámbricas del programa en el micro también llamadas *Over The Air updates (OTA)*.

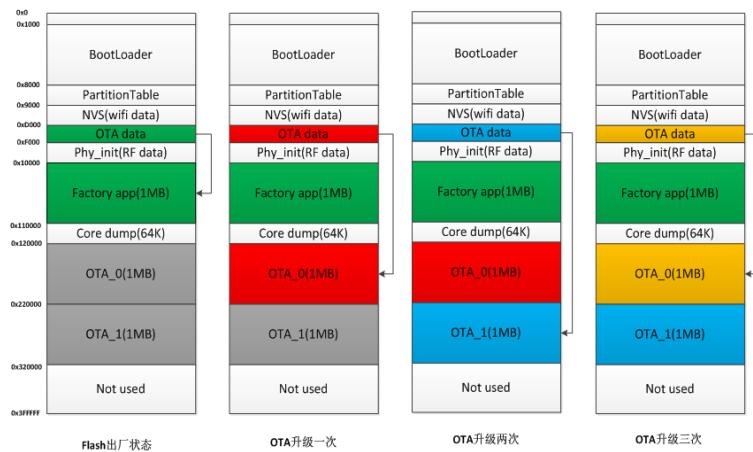


Figura 4.3. Diagrama ilustrativo de la memoria flash para los modelos de tablas de partición. Columna 1: Aplicación de fábrica. Columna 2, 3, 4: Aplicaciones con arranques múltiples

En este proyecto se utiliza una aplicación única sin actualizaciones inalámbricas por lo que se utilizó el primer modelo de tabla, cuyas estructuras en archivo CSV se especifican a continuación:

Tabla 4.1. ESP-IDF Tabla de Particiones por defecto (Aplicación única de fábrica, sin OTA)

Name	Type	SubType	Offset	Size	Flags
nvs	data	nvs	0x9000	0x6000	
phyinit	data	phy	0xf000	0x1000	
factory	app	factory	0x10000	1M	

Con un desplazamiento de 0x10000 (64KB) en la flash, se encuentra la aplicación llamada "fábrica". El gestor de arranque ejecutará esta aplicación de forma predefinida. También hay dos regiones de datos definidas en la tabla de particiones para almacenar la partición de la biblioteca NVS y los datos de inicio *PHY* (datos de calibración de periféricos).

Con la finalidad de adaptarse a la aplicación fue necesario modificar la tabla de particiones puesto que la interfaz gráfica de módulo de configuración y las diferentes particiones para datos requerían de un tamaño mayor al que posee la tabla de particiones original. Para crear o modificar los tamaños en una partición basta con configurar en el *menuconfig* del proyecto que se utilizará una tabla de particiones personalizada.

Tabla 4.2. Tabla de particiones personalizada para la aplicación

Name	Type	SubType	Offset	Size	Flags
nvs	data	nvs	0x9000	0x6000	
phyinit	data	phy	0xf000	0x1000	
factory	app	factory	0x10000	1M	
www	data	spiffs		1M	
.	.	.		.	
.	.	.		.	
.	.	.		.	

La columna *Name* y la fila *www* de este archivo separado por comas representado en la tabla 4.2 se define el espacio de memoria donde se guardaran todos los archivos necesarios para la página web, el *Type* es data y el *SubType* es spiffs debido a que así se debe definir en la memoria el tipo de datos para un sistema de archivos. Y el espacio en la memoria es de 1MB como lo dice en la columna *Size*.

El servidor *HTTP* está implementado por el fabricante en los ejemplos de aplicaciones, se tomo dicha API y se adaptó a las necesidades de esta aplicación.

4.2.1.2. Interfaz Gráfica de usuario

La interfaz gráfica consta de 5 pantallas, cada una con un formulario que son parámetros necesarios para conectarse a la red mallada, para su funcionamiento como algún tipo de nodo o para la conexión con el enrutador. A la interfaz se accede mediante

la dirección IP del nodo o mediante un DNS relacionado con la MAC. También se incluye la configuración de usuario y clave para proteger al acceso, de usuarios no deseados. Las pantallas por las que puede ingresar datos el usuario son:

1. Inicio de sesión

El inicio de sesión requiere al menos de un usuario y contraseña para autenticar el acceso a la configuración del sistema. Esta pantalla debe ofrecer el acceso al sistema de configuración si estos son introducidos correctamente y rechazarlos en caso de que no.

2. Parámetros de red mallada

La pantalla de ingreso de los parámetros para la red mallada requiere tener los siguientes parámetros:

- ID de la red: Que es el identificador de la red, es un parámetro parecido en formato a una dirección MAC que permite identificar la red mallada y diferenciarla de otras implementadas.
- Contraseña de la red: Es la clave requerida por los nodos para ingresar a la red. Sin este parámetro y el anterior no serán admitidos en la red mallada.
- Cantidad máxima de capas: Se refiere a la cantidad de capas aceptadas por la red, este parámetro es necesario para la configuración de la misma. El valor varía entre 1 y 25 pero el fabricante recomienda 10 para tiempos de retardos manejables.
- Cantidad máxima de estaciones conectadas: Este parámetro es el número máximo de estaciones WiFi conectadas que tendrá el nodo, en términos de la red mallada es la cantidad máxima de nodos hijo que puede tener un nodo. Se aceptan valores entre 1 y 10 pero recomienda que sea menor que 6 para un buen funcionamiento del punto de acceso.

- Puerto del socket: El número de puerto TCP/IP que será abierto por el microcontrolador para conexiones externas, esto será explicado en detalle en la rutina correspondiente a este proceso.

3. Parámetros seriales

La pantalla de ingreso de los parámetros para el funcionamiento serial debe depender del tipo de nodo que se busque configurar puesto que hay al menos 3 tipos de nodos en la red sin contar el nodo central. En general para cualquier tipo de nodo los parámetros necesarios son:

- Tasa de baudios: La tasa de comunicación con el medidor ó el bus serial.
- Factor de conversión [$\frac{imp}{kWh}$]: La cantidad de impusos por cada kWh de energía registrada por el medidor al que se conecte el nodo.
- Condición inicial de energía: Al soportar medidores ya instalados se deben tomar en cuenta las condiciones iniciales de energía consumida que marca el medidor para que el sistema sea coherente con el equipo.
- Identificador de esclavo: Necesario para identificar el tipo de nodo que emulará un bus serial.

4. Configuración de red local

Esta vista debe contemplar el ingreso de los parámetros de SSID y contraseña de la red local.

5. Configuración de inicio de sesión

Esta vista responde a la necesidad de cambiar el nombre y la contraseña por defecto para el acceso al sistema de configuración mediante la interfaz web

gráfica.

Además de las pantallas el micro debe recibir y procesar todos estos datos que le son enviados en formato JSON. En cada pantalla se tiene una pareja clave-valor por cada campo de los formularios, el servidor debe recibirlas y extraer los datos para que luego almacenarlos de forma permanente en la memoria y estos datos estén disponibles cuando se reinicie el dispositivo ya configurado.

4.2.2. Modo de operación

En este modo el primer periférico en configurarse en cualquiera de los nodos es la memoria no volátil (NVS), también llamada memoria flash, que se utiliza para almacenar información persistente entre reinicios del microcontrolador. El fabricante ofrece una interfaz de programación de aplicaciones () que permite configurarla y operar fácilmente con esta.

Esto se debe a que ahí se almacenan todos los datos que se transmiten al microcontrolador mientras este se encuentra en el modo de configuración.

Se toma desde la memoria flash la información sobre el tipo de nodo (serial RS485, entrada por pulsos o repetidor) y todos los parámetros que vienen asociados a este (tasa de baudios, ID serial, kWh iniciales, etc.), así como los parámetros necesarios para registrarse en la red mallada y poder comunicarse a través de la red.

Luego se debe configurar el WiFi en cada uno de los medidores para poder unirse a la red mallada. Este periférico en el ESP32 posee también una interfaz de programación de aplicaciones para programarlo de manera más sencilla (). Mediante esta interfaz, se calibra el WiFi, se asigna un lugar para almacenar sus parámetros de calibración y

se coloca el modo de operación. El ESP32 posee tres modos de operación para este periférico: estación (STA), punto de acceso (AP) y un modo híbrido (AP/STA). Este último es el que se suele utilizar en la red mallada. Una vez configurado el WiFi se procede con la configuración e inicio de la red mallada.

Cada nodo tiene una serie de tareas que son repartidas de manera equitativa entre ambos núcleos del ESP32, el objetivo es que estas se ejecuten en paralelo y el RTOS sea el encargado de esperar por los datos necesarios para su ejecución sincronizada y correcta.

4.2.2.1. La red mallada y el nodo central

La topología de red mallada utilizada en el proyecto está basada en la que provee *Espressif* para ser utilizada en conjunto con el ESP32. Dicha topología es llamada ESP-MESH y el fabricante proporciona una interfaz de programación de aplicaciones (API) para interactuar y modificar dicha topología con la intención de que sea adaptable a las soluciones que se deseen implementar. Según el tipo de medidor que haya sido configurado en la interfaz web serán activadas unas tareas u otras, puesto que cada medidor requiere de diferentes procesos para extraer y transmitir sus datos.

El objetivo es implementar una red mallada que transmita una información representativa de la cantidad de energía consumida, tomándola a partir de medidores de energía. Esta arquitectura de red requiere de un nodo particular para comunicarse hacia redes externas, llamado nodo raíz o nodo *root*. Dicho nodo se elige mediante parámetros de calidad de señal, específicamente se tiene en cuenta la intensidad de señal (RSSI) de cada nodo con el enrutador. Considerando esto se efectúa una votación en la cual se elige qué nodo será asignado como nodo raíz, este suele ser el nodo de mayor intensidad

de señal pues esto garantiza una comunicación estable.

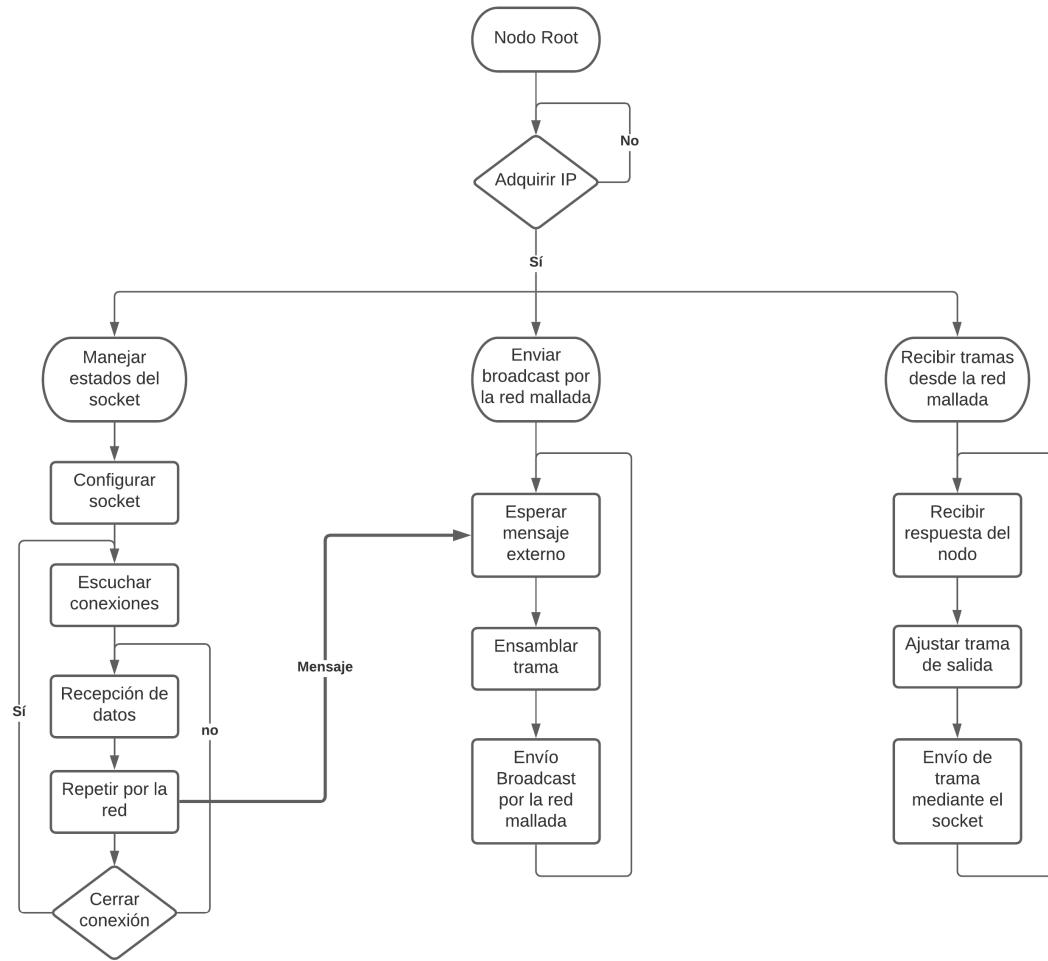


Figura 4.4. Diagrama de flujo de las tareas implementadas en el nodo central.

Una vez elegido este nodo comienza a ejecutarse el código con las tareas implementadas en la programación para él. Son tres tareas, sin embargo debe comenzar estableciendo la conexión WiFi con el enrutador y obteniendo la IP de parte del DHCP del enrutador. Esta dirección asignada es la que será utilizada para comunicarse con la red mallada desde una red exterior.

Una vez se obtiene la IP se crean en el RTOS las tareas necesarias para el nodo central:

- **Manejar estados del socket:** El sistema requiere un punto de conexión que sirva como entrada y salida para los datos requeridos. Un socket ofrece una manera de intercambiar un flujo de datos y permite enviar mensajes a otra dirección IP.

Dicho socket requiere de un proceso de configuración para el que se utiliza inicialmente un número de puerto. Este puerto será habilitado para permitir conexiones entrantes de clientes que soporten esta estructura de socket y el protocolo inherente a este bien sea TCP o UDP. Se utilizó el protocolo TCP ya que este protocolo de transporte garantiza la integridad de los datos.

Una vez establecidos el puerto y el protocolo a soportar, el socket queda en espera de conexiones entrantes. Cuando un cliente solicita conectarse y el socket no está ocupado por otro cliente, esta conexión es aceptada y se cambia el estado a *esperar mensaje*.

Al ser recibido un mensaje este es copiado a una estructura de datos llamada *cola* del sistema RTOS que funciona como un buzón de mensajes, en el que se puede acceder desde otra tarea que se encuentre esperando por dicha información, esta tarea toma la información y la retira del buzón y la utiliza para su ejecución.

El socket vuelve al estado de *esperar mensaje* pero no por un tiempo ilimitado, el socket para mantener la conexión requiere recibir mensajes cada cierto tiempo y si no se cumple con el tiempo máximo entre mensajes el socket acabará con la conexión establecida con ese cliente y se pondrá a la espera de clientes nuevos, esto para que un cliente no bloquee permanentemente el puerto.

- **Enviar mensajes broadcast por la red mallada:** Esta tarea se encuentra a la espera del mensaje que se recibe en el socket, una vez hecho esto la tarea toma de la *cola* del sistema RTOS la información depositada. Con estos datos se procede a repetir este mensaje mediante WiFi para todos los nodos conectados a este nodo central utilizando la función de envío de mensajes internos que ofrece el API de la red mallada.

Para que esto funcione el nodo central debe tomar la dirección MAC de cada nodo desde su tabla de enrutamiento, ya que en la tabla del nodo central se encuentran todos los nodos de la red. La API del fabricante ofrece dicha opción y los mensajes que son enviados mediante WiFi van dirigidos con la dirección MAC de cada nodo. Repetidos según las reglas de la red hasta llegar a su destino.

Una vez los nodos que conforman la red mallada reciben el mensaje deben interpretarlo pero para que la información contenida en este tenga sentido, debe haber un protocolo estándar para comunicarse. Se escogió el protocolo Modbus TCP/IP como lenguaje interno de la red mallada por la poca modificación que requiere la trama para comunicarse con los medidores que tienen interfaces Modbus RS485 y por la existencia de software que emula un maestro modbus mediante un socket TCP/IP.

El protocolo Modbus TCP/IP es un protocolo sincronizado por silencios, de un solo maestro y múltiples esclavos. En este caso se utilizan los mensajes enviados por el socket para simular el maestro Modbus. Por lo que la red diseñada se puede ver de manera análoga como un bus serial inalámbrico, que repite tal cual el mensaje hacia los demás nodos. En la trama se envía:

- **Identificador del mensaje:** Evita repeticiones de mensajes.
- **Identificador de protocolo:** Se refiere a la versión del protocolo Modbus.

- **Longitud de la trama:** Cantidad de bytes de la trama sin contar el mensaje.
- **Identificador del esclavo:** Numero en hexadecimal del esclavo al que se dirige el mensaje.
- **Código de función:** Las funciones están identificadas con códigos hexadecimales. Por ejemplo: 0x03 es la función para leer un registro estático y 0x04 es la encargada de leer un registro de entrada.
- **Datos:** Aquí se encuentran los números de registros y la información contenida en ellos.

ID de Transición 2 bytes	Protocolo ID 2 bytes	Longitud 2 bytes	Unit ID 1 byte	Cód. Función 1 byte	Datos n bytes
-----------------------------	-------------------------	---------------------	-------------------	------------------------	------------------

Figura 4.5. Estructura de una trama Modbus TCP/IP.

- **Recibir tramas desde la red mallada y enviarlas hacia una dirección externa:** Luego de que se envía la trama mediante la red WiFi esta tarea espera la respuesta del esclavo al que iba dirigido el mensaje. Cuando la respuesta llega desde la red mallada se utilizan las partes de la trama correspondientes a la longitud para enviar el mensaje mediante el socket TCP/IP hacia el cliente conectado. El tiempo que el mensaje toma en llegar desde el maestro al esclavo correspondiente y de vuelta debe ser menor al tiempo máximo de retardo del mensaje que establezca el maestro modbus para que no hayan errores de sincronización.

4.2.2.2. Nodo serial RS485

Este tipo de nodo se debe inicializar con los parámetros seriales mediante el modo de configuración en la interfaz gráfica para el usuario. El código en este nodo fue pensado para que este nodo funcione como una conversión de inalámbrico a serial, puesto que es el encargado de comunicarse con medidores que cumplen la norma

RS485 y soportan el protocolo Modbus. Pero también fue pensado para transformar la trama modbus TCP/IP que llega desde la red mallada en una trama modbus RTU que es la que soportan la mayoría de medidores con modbus.

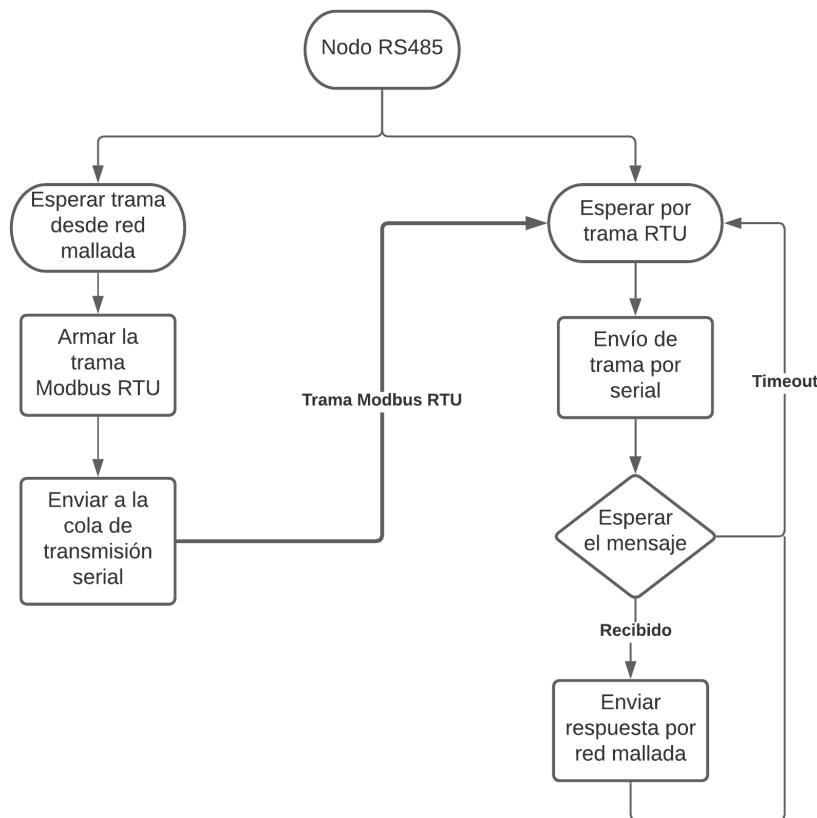


Figura 4.6. Diagrama de flujo de las tareas implementadas en el nodo RS485.

- **Esperar la trama desde la red mallada:** Esta tarea se encarga tomar el mensaje que llega y convertir el modbus TCP/IP en un Modbus RTU, esta conversión consiste en quitar los 6 primeros bytes de la trama modbus TCP/IP que serían los bytes de cabecera y luego colocar una comprobación de errores mediante un CRC de 2 bytes al final de la trama.

Una vez realizado este procedimiento, el resultado es puesto en una *cola* del

sistema RTOS para comunicarlo hacia la tarea encargada de manejar el bus serial del microcontrolador.

- **Esperar por trama RTU:** En esta tarea se maneja el bus de datos para la comunicación serial con los medidores. Se toman los datos recibidos en la *cola* del sistema y se envían mediante el UART del microcontrolador hacia el bus serial con las API del UART que ofrece el fabricante. A partir del envío comienza a correr un tiempo máximo de respuesta que tiene el dispositivo serial conectado al bus para devolver la información solicitada.

Todos los nodos configurados como este tipo de nodo repiten el mensaje independientemente de su contenido pero al ser protocolo modbus solo responderá el esclavo que corresponda con el id contenido en el mensaje. Si la respuesta serial llega, esta se transforma de modbus RTU a modbus TCP/IP para ser enviada al nodo central como respuesta y el nodo queda en espera del siguiente mensaje.

4.2.2.3. Nodo de entrada por pulsos

Este nodo se inicializa mediante la interfaz web gráfica para configuración. Este nodo se creó con la finalidad de adaptarse a los medidores que se encuentran ya colocados en la mayoría de conjuntos residenciales y comerciales. Estos medidores poseen una salida binaria que es la que se utiliza para calibrarlos, en esta salida se puede observar un cantidad de impulsos por cada kWh consumido que registra el medidor.

La idea de este nodo es tomar esos impulsos y almacenarlos para contar el total de los kWh consumidos, teniendo en consideración la condición inicial del medidor y para evitar que la cuenta se pierda por cualquier razón este nodo debe almacenar dicho

valor en una memoria que sea persistente.

Además el nodo debe tener la capacidad de responder a los mensajes enviados por la red mallada y en caso de que dicho mensaje contenga la trama indicada para solicitar el valor de energía, responder con dicho registro mediante la red mallada WiFi hacia el nodo central.

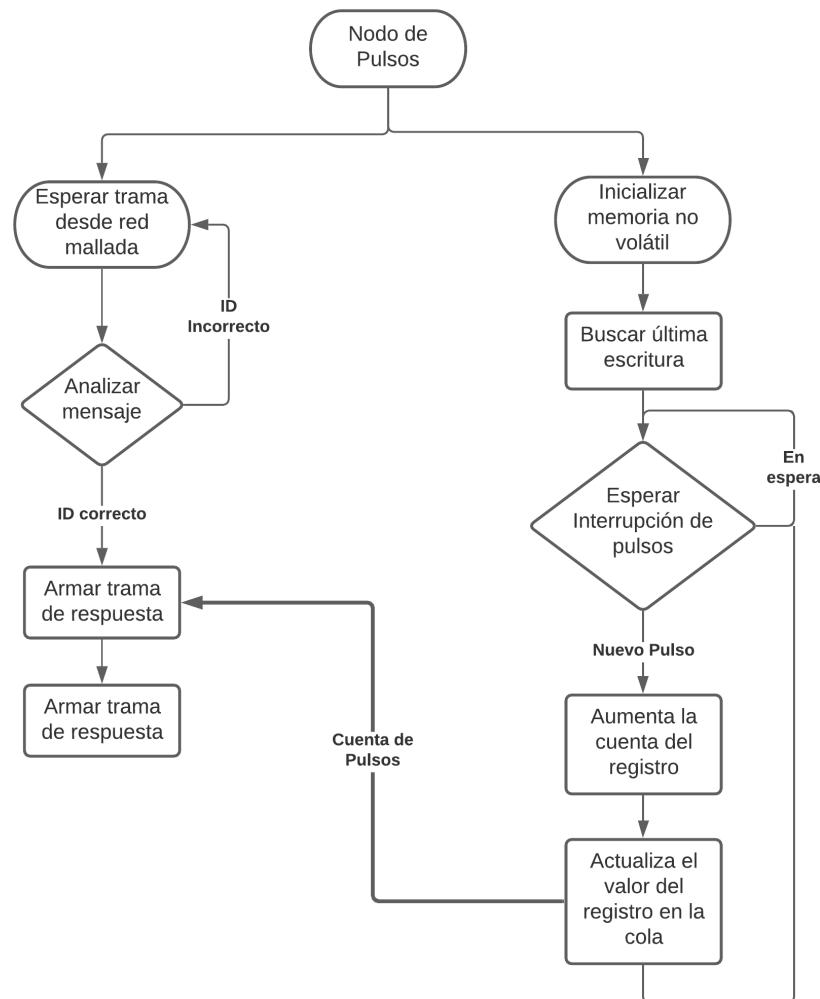


Figura 4.7. Diagrama de flujo de las tareas implementadas en el nodo de pulsos.

- **Esperar trama desde la red mallada:** El nodo debe poder comunicarse por la red mallada, por lo que se implementó en este la comunicación mediante la red utilizando la API del fabricante. Esta tarea estará en espera hasta que llegue el mensaje proveniente de la red mallada, una vez se tenga dicho mensaje se procede a comprobar los campos respectivos al identificador del esclavo, el código de función, el número de registro y el CRC. De comprobarse que todos estos campos son correctos se procede a armar la trama de respuesta con el valor actual de la cuenta de pulsos y se envía de vuelta al nodo central encapsulado en la trama modbus TCP/IP.
- **Inicializar memoria no volátil:** En el microcontrolador utilizado el fabricante utiliza una API llamada NVS (*Non volatile Storage*) para referirse a la memoria flash, esta interfaz de programación ofrece facilidades para tratar y almacenar datos en la memoria flash, así como la capacidad de manejar las particiones, páginas y registros que la comprenden.

Para inicializar la memoria no volátil se utiliza dicha interfaz ofrecida por el fabricante. Cada partición de la memoria no volátil requiere de un nombre, el nombre por defecto es *nvs* pero en este caso serán utilizadas varias particiones por lo que se les asignó el nombre de *app1* , *app2*, *app3*. Estos nombres son definidos en la tabla de particiones y serán utilizados en el código del programa.

Tabla 4.3. Tabla de particiones de la aplicación

Name	Type	SubType	Offset	Size	Flags
nvs	data	nvs	0x9000	0x6000	
phyinit	data	phy	0xf000	0x1000	
factory	app	factory	0x10000	1M	
www	data	spiffs		1M	
app1	data	nvs		64K	
app2	data	nvs		64K	
app3	data	nvs		64K	

- **Tamaños y tablas de particiones**

Las páginas de partición poseen 64kB de la memoria flash cada una, esto debido a que cuando se inicializa una partición con el API *NVS* dicha partición consume una cantidad de memoria *RAM* proporcional al tamaño de la partición inicializada, por lo que para no consumir memoria *RAM* excesiva se escogió este tamaño de partición.

La partición a su vez se divide en estructuras llamadas páginas, que también poseen nombres, estas páginas tienen un tamaño de 4kB (por defecto) y el nombre por defecto de la página inicial es *storage*, el resto de las 15 páginas fueron nombradas con los nombres *pv0 - pv14*.

Y las páginas están compuestas por registros, 126 para ser exactos, llamados por el fabricante como entradas. Estas llevan también un nombre y fueron asignadas como (*e0-e125*). Las entradas son capaces de almacenar diferentes tipos de datos: enteros con y sin signo, cadenas de caracteres y arreglos.

Es importante que se consideren los límites físicos de la memoria flash, un mismo espacio de memoria puede sobreescribirse hasta unas 100000 veces

antes de dañarse. Por lo que la cantidad de veces que se reescribe en una entrada en el algoritmo se fijó como 50000, para preservar el estado físico de la memoria. Para el algoritmo, una entrada se encuentra llena cuando alcanza ese número de veces escrita.

- **Buscar la última escritura en la memoria**

El algoritmo para almacenar los pulsos comienza en la partición *appl* y comprueba si esta partición se encuentra 'llena' mediante una bandera lógica en la pagina principal *storage*, de ser así cierra la partición actual y busca en la siguiente hasta encontrar una que no esté llena, de no encontrar una disponible se emite una alerta mediante el serial.

Una vez en la partición se busca la última página en la que se escribió mediante un entero almacenado en la página principal *storage*. En la última página escrita se cuenta la cantidad de entradas escritas, con este numero se puede saber la última entrada que se escribió.

Se toma el valor de esa última entrada y se le suman todas las entradas, páginas y particiones anteriores además de la condición inicial de kWh para saber la cantidad de pulsos que se llevan en la cuenta y se almacena en el registro para estar disponible en caso de que se deba enviar mediante la red WiFi.

- **Nuevo pulso detectado**

Por último cuando se produce un nuevo pulso y se detecta mediante una interrupción en el pin asignado, el programa aumenta la cuenta de pulsos

total que se encuentra en el registro para enviar por la red y también aumenta la cuenta en la entrada actual para que la información sea persistente.

En caso de llenarse una entrada se cambia a la siguiente hasta llegar a la entrada *e125*, luego se cambia a la siguiente página y se modifica el entero correspondiente a la última página escrita en *storage*, hasta llegar a la página *pv14*, en este caso se coloca una bandera lógica en la página *storage* para avisar que se encuentra llena la partición y se cambia a la siguiente. Luego vuelve al estado de espera de la siguiente interrupción por pulsos.

■ Cálculos de vida útil estimada según el uso de la memoria flash

Tomando la como referencia el medidor de una casa promedio en España, al año se alcanza la cifra de 9922 kWh consumidos por lo que haciendo una estimación para la duración de la memoria tendríamos:

$$52 \frac{\text{semana}}{\text{año}} \times 7 \frac{\text{dia}}{\text{semana}} \times 24 \frac{\text{horas}}{\text{dia}} = 8736 \frac{\text{horas}}{\text{año}} \quad (4.1)$$

$$\frac{9922 \frac{\text{kWh}}{\text{año}}}{8736 \frac{\text{horas}}{\text{año}}} = 1,1358 \frac{\text{kWh}}{\text{hora}} \quad (4.2)$$

La cantidad de pulsos emitidos por kWh depende enteramente del medidor, puesto que todos tienen valores diferentes. Para un medidor de 1800 $\frac{\text{imp}}{\text{kWh}}$ que es el que se utiliza para las pruebas de sistema, se tiene que:

$$1,1358 \frac{\text{kWh}}{\text{hora}} \times 1800 \frac{\text{impulsos}}{\text{kWh}} = 2045 \frac{\text{impulsos}}{\text{hora}} \quad (4.3)$$

En el cálculo de la vida útil es necesario tomar en cuenta la cantidad de impulsos que se puede almacenar en las particiones:

$$126 \text{ entradas} \times 15 \text{ paginas} \times 50k \frac{\text{impulsos}}{\text{entrada}} = 94,5 \text{ Mimpulsos} \quad (4.4)$$

$$94,5 \text{ Mimpulsos} \times 2045 \frac{\text{impulsos}}{\text{hora}} = 46210 \text{ horas o 5 anios} \quad (4.5)$$

Esto significa que cada partición duraría para el consumo de una casa promedio 5 años, al tener 3 particiones inicialmente se estarían garantizando según los cálculos unos 15 años de vida útil.

Esto sin considerar que el número de particiones de 64kB se puede aumentar y solo se ve limitado por el tamaño de la memoria flash del micro que en muchos casos llega a ser entre 2 y 16 MB.

4.2.2.4. Nodo Repetidor

El nodo repetidor es un nodo que ha sido configurado para ser parte de la red mallada sin tener conexiones físicas con alguna remota mediante un bus serial o con un medidor de energía mediante su salida de calibración.

Este tipo de nodos es útil para ampliar la cobertura de la red en caso de que sea necesario abarcar un mayor área o tener mayor densidad de nodos para mejorar la conexión entre estos en una zona específica. Este nodo no posee tareas implementadas en sus núcleos, solo posee el código necesario para ser parte de la red y repetir en ambos sentidos los mensajes que lleguen a este.

4.2.2.5. Otras consideraciones

Cabe destacar que cualquier tipo de nodo de la red puede y retransmite los mensajes que le llegan sin importar si es de pulsos, serial RS485 o repetidor. Esta capacidad de retransmisión viene incorporada en el esqueleto de la red implementado por el fabricante.

CAPÍTULO V

PRUEBAS Y RESULTADOS

Una vez establecido el hardware y realizado el diseño del software necesario para implementar la aplicación en el microcontrolador se realizó una serie de pruebas para constatar la correcta implementación del sistema.

5.1. Resultados de la interfaz gráfica

Para realizar las pruebas del sistema que se presentan a continuación se debió configurar cada uno de los nodos utilizando la interfaz gráfica implementada en el modo de configuración. A continuación se ilustra el procedimiento y las distintas pantallas implementadas en ello:

```
[...]
I (17971) REQUEST: req->uri: /
I (17981) esp-rest: File sending complete
I (18221) REQUEST: req->uri: /js/lib/jquery.min.js
I (19041) esp-rest: File sending complete
I (19041) REQUEST: req->uri: /css/bootstrap-lux.min.css
I (21141) esp-rest: File sending complete
I (21151) REQUEST: req->uri: /css/estilos.css
I (21161) esp-rest: File sending complete
I (21161) REQUEST: req->uri: /js/lib/popper.min.js
I (21341) esp-rest: File sending complete
I (21341) REQUEST: req->uri: /js/lib/bootstrap.min.js
I (21951) esp-rest: File sending complete
I (21961) REQUEST: req->uri: /js/login.js
I (22291) esp-rest: File sending complete
I (22421) REQUEST: req->uri: /img/fondo.png
I (23351) esp-rest: File sending complete
I (23651) REQUEST: req->uri: /favicon.ico
I (23851) esp-rest: File sending complete
```

Figura 5.1. Log serial del microcontrolador en modo servidor HTTP cuando es enviada una de las pantallas.

Cuando el usuario solicita cualquier página al servidor se produce recibe la petición y se procesa enviando posteriormente el archivo solicitado como se puede ver en la figura 5.1, esto es común para todas las pantallas y se puede ver afectado por la congestión de la red local, ya que el envío se realiza mediante de esta.

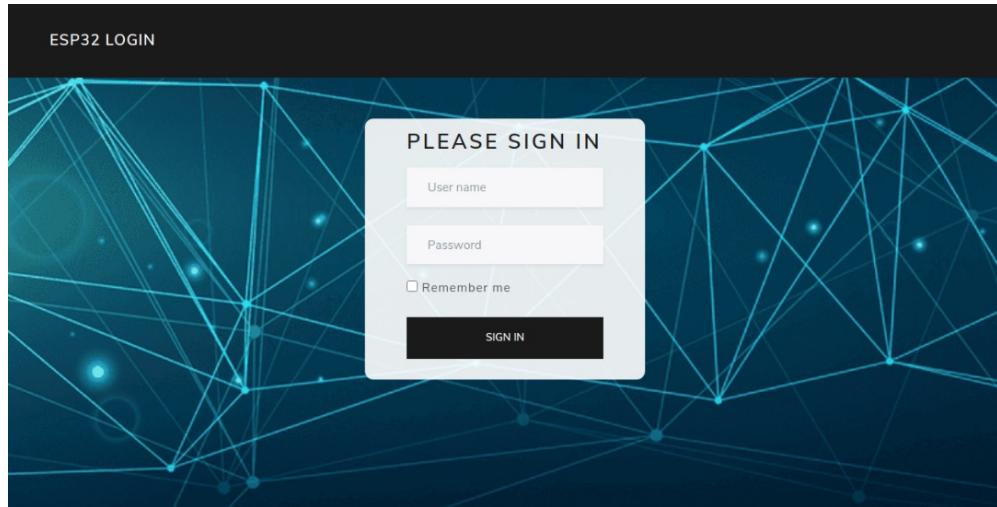


Figura 5.2. Vista de la pantalla de inicio de sesión del usuario.

Seguidamente, se puede observar la pantalla de inicio de sesión, figura 5.2, en la que el usuario debe introducir de manera correcta clave y contraseña para acceder a la configuración del equipo, considerando la seguridad del mismo.

```
I (85971) DEBUG: BUFFER: {"usuario":"admin","contrasena":"admin","remember":false}
```

Figura 5.3. Mensaje recibido en el servidor cuando el usuario rellena el inicio de sesión y envía los datos.

Una vez es iniciada la sesión se muestra el formulario de configuración de la red mallada, en el que se introducen los parámetros de configuración de dicha red.

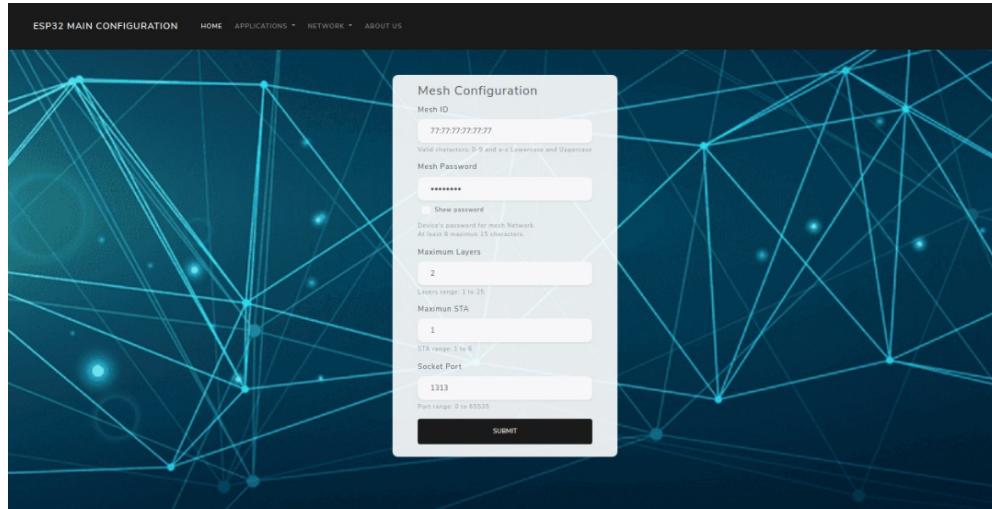


Figura 5.4. Vista de la pantalla del formulario de parámetros de la red mallada.

```
I (6881) REQUEST: req->uri: /net-user-log.html
I (6891) esp-rest: File sending complete
I (7261) REQUEST: req->uri: /css/bootstrap-lux.min.css
I (9991) esp-rest: File sending complete
I (10001) REQUEST: req->uri: /js/lib/jquery.min.js
I (10691) esp-rest: File sending complete
I (10691) REQUEST: req->uri: /js/lib/popper.min.js
I (11171) esp-rest: File sending complete
I (11181) REQUEST: req->uri: /css/estilos.css
I (11501) esp-rest: File sending complete
I (11501) REQUEST: req->uri: /js/lib/bootstrap.min.js
I (12581) esp-rest: File sending complete
I (12581) REQUEST: req->uri: /js/net-user-log.js
I (12871) esp-rest: File sending complete
I (13331) REQUEST: req->uri: /userlog
Error (ESP_ERR_NVS_NOT_FOUND) opening NVS handle!
I (13331) JSON USERLOG: {
    "user": "",
    "pass": ""
}
```

Figura 5.5. Mensaje recibido en el servidor cuando el usuario rellena el formulario de red mallada y envía los datos.

Con los parámetros en las figuras 5.4 y 5.5 se puede inicializar la red mallada. Se procede luego mediante el menú de la barra superior de la página para acceder al formulario de parámetros seriales.

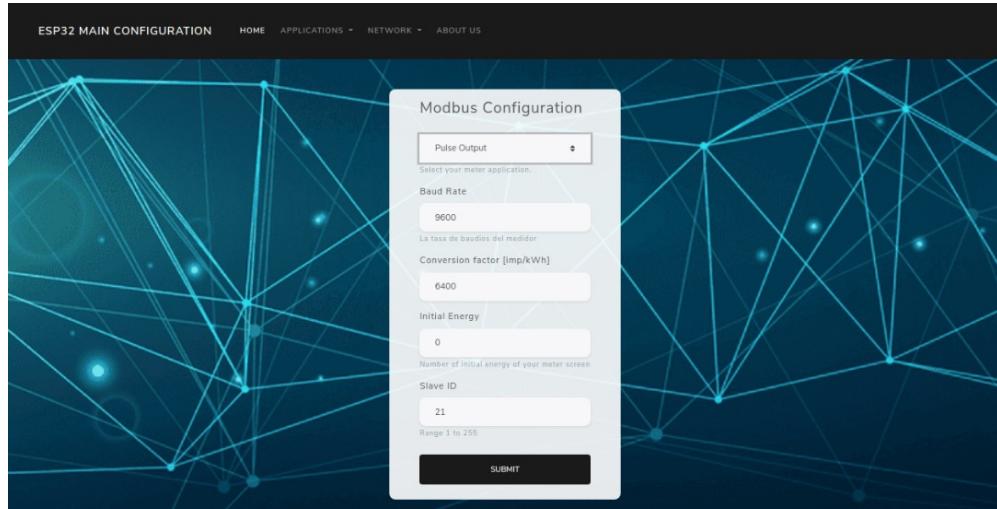


Figura 5.6. Vista de la pantalla de formulario de parámetros seriales.

```
I (628021) DEBUG: BUFFER: {"type": "pulsos", "baudrate": "9600", "convfac": "4200", "iniene": "0", "slaveid": "1"}
I (628021) FILL MODBUS: {
    "type": "pulsos",
    "baudrate": "9600",
    "convfac": "4200",
    "iniene": "0",
    "slaveid": "1"
}
W (628031) FILL MODBUS: Modbus meter type: pulsos
W (628041) FILL MODBUS: Baud rate: 9600
W (628041) FILL MODBUS: Conversion Factor: 4200
W (628051) FILL MODBUS: Energia inicial: 0
W (628051) FILL MODBUS: Slave ID: 1
I (628061) SET_MODBUS: pulsos
I (628071) SET_MODBUS: 1
I (628071) SET_MODBUS: 4200
I (628071) SET_MODBUS: 9600
I (628071) SET_MODBUS: 0
```

Figura 5.7. Mensaje recibido en el servidor cuando el usuario rellena el formulario de parámetros seriales y envía los datos.

Los parámetros de las figuras 5.6 y 5.7 son los parámetros requeridos para la extracción de la data desde el medidor. En la siguiente pantalla se debe llenar el formulario de la red local:

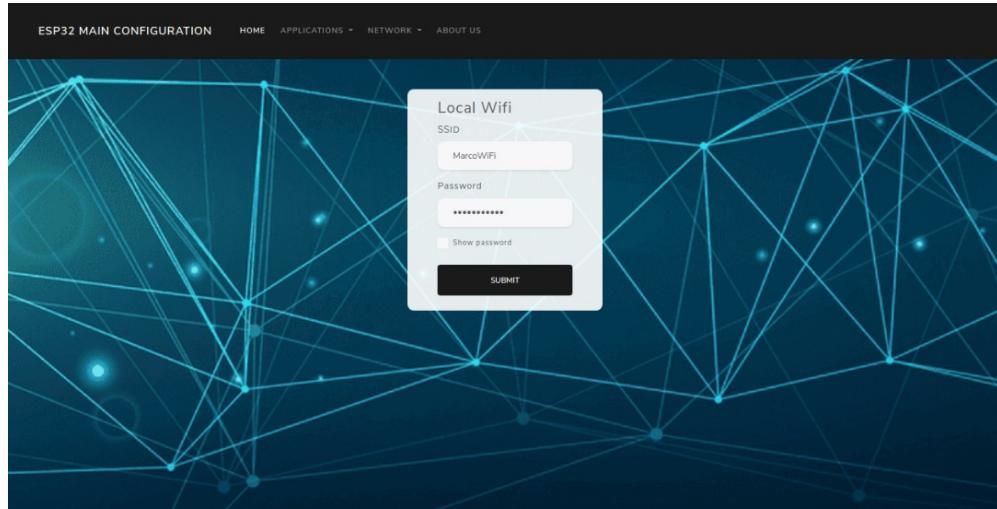


Figura 5.8. Vista de la pantalla de parámetros de red local

```
I (788421) DEBUG: BUFFER: {"ssid":"i", "pass": "a"}  
I (788421) FILL LOCWIFI: {  
    "ssid": "i",  
    "pass": "a"  
}  
W (788421) FILL LOCWIFI: Local WiFi SSID: C a  
W (788431) FILL LOCWIFI: Local WiFi password: a a 2  
I (788441) SET_LOCWIFI: .  
I (788441) SET_LOCWIFI: .
```

Figura 5.9. Mensaje recibido en el servidor cuando el usuario rellena el formulario de red local y envía los datos.

Los parámetros de las figuras 5.8 y 5.9 son utilizados para que el equipo pueda tener acceso al WiFi en caso de ser el nodo central. Con este paso se cumple el mínimo necesario para que el equipo funcione. Por último se realizó una pantalla para cambiar el usuario y la contraseña de inicio de sesión, en caso de que se quiera modificar esta.

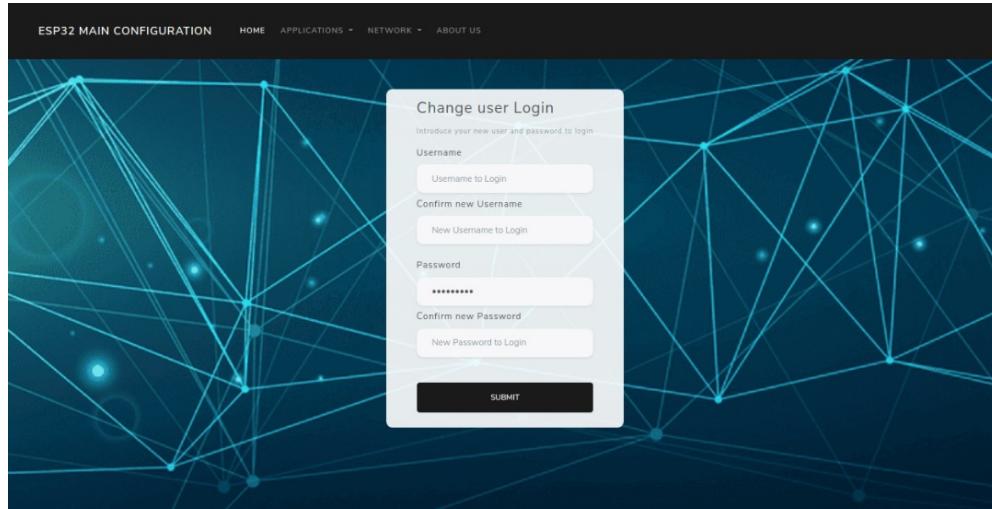


Figura 5.10. Vista de la pantalla para modificar los parámetros de acceso del usuario.

```

I (6081) REQUEST: req->uri: /net-user-log.html
I (6891) esp-rest: File sending complete
I (7261) REQUEST: req->uri: /css/bootstrap-lux.min.css
I (9991) esp-rest: File sending complete
I (10001) REQUEST: req->uri: /js/lib/jquery.min.js
I (10691) esp-rest: File sending complete
I (10691) REQUEST: req->uri: /js/lib/popper.min.js
I (11171) esp-rest: File sending complete
I (11181) REQUEST: req->uri: /css/estilos.css
I (11501) esp-rest: File sending complete
I (11501) REQUEST: req->uri: /js/lib/bootstrap.min.js
I (12581) esp-rest: File sending complete
I (12581) REQUEST: req->uri: /js/net-user-log.js
I (12871) esp-rest: File sending complete
I (13331) REQUEST: req->uri: /userlog
Error (ESP_ERR_NVS_NOT_FOUND) opening NVS handle!
I (13331) JSON USERLOG: (
    "user": "",
    "pass": ""
)

```

Figura 5.11. Mensaje recibido en el servidor cuando el usuario rellena el formulario de actualizar usuario y contraseña y envía los datos.

5.2. Prueba de funcionamiento del sistema

Una vez introducidos los datos de configuración en el modo anterior, se reinician los nodos para que la red comience a funcionar. Cada nodo se conecta al WiFi y entre sí, luego se realiza una votación en la que se elige un nodo central, estableciéndose así la primer capa de la red.

Posteriormente se establece la estructura de la red, en el caso de estas pruebas realizadas se colocó en los parámetros del nodo central un máximo de tres (3) capas de profundidad para la red y un máximo de dos (2) conexiones con nodos hijos para dicho nodo central.

En la siguiente capa de la red, la segunda capa, se encuentra un nodo repetidor que funcionará como demostración de la capacidad de la red de comunicarse con nodos en capas más profundas a través de otros nodos. El mismo fue configurado con la capacidad de tener para esta prueba un (1) solo nodo hijo conectado. En esta segunda capa también se encuentra uno de los nodos a medir, el nodo de pulsos, conectado directamente con el nodo central.

En la tercera y última capa para esta prueba y conectado al nodo enlace se encuentra el nodo de transmisión serial RS485. Un diagrama de la prueba quedaría de la siguiente manera:

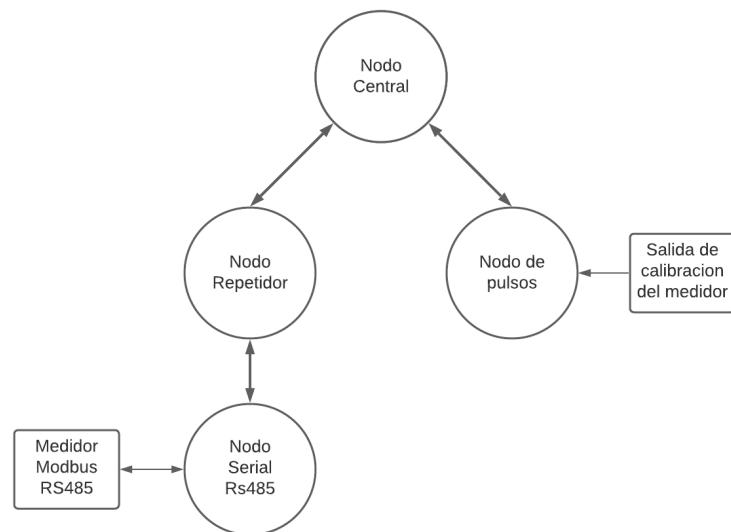


Figura 5.12. Diagrama general de la red para pruebas.

En esta red se realizaran pruebas de extracción de datos para comprobar el funcionamiento de los nodos y pruebas de restablecimiento del sistema en caso de que los nodos se desconecten de la red.

El tiempo entre envío de mensajes en la red se estableció en 1 segundo y el máximo tiempo de respuesta aceptado se estableció en 5s para las siguientes pruebas.

5.2.1. Extracción de datos del nodo contador de pulsos

Tabla 5.1. Registro de mensajes del maestro sobre la comunicación con el esclavo de pulsos.

Registro del maestro modbus					
Prueba N°	Enviados	Recibidos	Tiempo [s]	Errores	Error[%]
1	1500	1499	2223,801	13	0,87
2	1500	1499	2226,643	16	1,07
3	1500	1500	2226,970	15	1

Tabla 5.2. Registro de mensajes del nodo contador de pulsos sobre la comunicación con el maestro.

Nodo contador de pulsos					
Prueba N°	Recibidos	Enviados	Tiempo [s]	Error de procesamiento	Error[%]
1	1499	1499	2223,801	-	-
2	1499	1499	2226,643	-	-
3	1500	1500	2226,970	-	-

El nodo contador de pulsos culminó con una cantidad de errores máxima en todas las pruebas realizadas del 1,07 % en el registro del maestro modbus (tabla 5.1), lo que denota el correcto funcionamiento de la red con este nodo.

Se observa en la tabla 5.2 que la cantidad de mensajes recibidos y enviados es la misma por lo que no hubo errores en el procesamiento del nodo, esta cantidad además coincide casi en totalidad con el registro del maestro por lo que se infiere que los errores registrados son errores debido al timeout ó error de identificador en la trama enviada por el nodo.

5.2.2. Extracción de datos del nodo RS485

Tabla 5.3. Registro de mensajes del maestro sobre la comunicación con el nodo serial rs485.

Registro del maestro modbus					
Prueba N°	Enviados	Recibidos	Tiempo[s]	Errores	Error[%]
1	1500	1481	2761,411	67	4,46
2	1500	1481	2761,803	67	4,46
3	1500	1480	2758,267	65	4,33

Tabla 5.4. Registro de mensajes del nodo serial rs485 sobre la comunicación con el maestro.

Nodo serial RS485					
Prueba N°	Recibidos	Enviados	Tiempo[s]	Error de procesamiento	Error[%]
1	1498	1481	2761,411	17	1,13
2	1498	1481	2761,803	17	1,13
3	1498	1480	2758,267	18	1,20

La diferencia en la cantidad de mensajes enviados y recibidos evidencia perdidas de paquetes en la red y un funcionamiento menor que el nodo de pulsos, las tasas de error registradas por el maestro denotan un porcentaje máximo de 4,46 % como se observa en la tabla 5.3.

Con los datos de la tabla 5.4 se puede observar que hubo un máximo de 18 errores

en el procesamiento de los mensajes en una prueba, estos errores en el procesamiento del nodo pueden surgir debido al tiempo de espera máximo que se tiene programado en el bus serial de este tipo de nodos, puede deberse al tiempo de respuesta del medidor asociado al nodo que puede verse saturado y responder menos mensajes de los que se demandan.

5.3. Prueba de restablecimiento del sistema

5.3.1. Para el nodo central

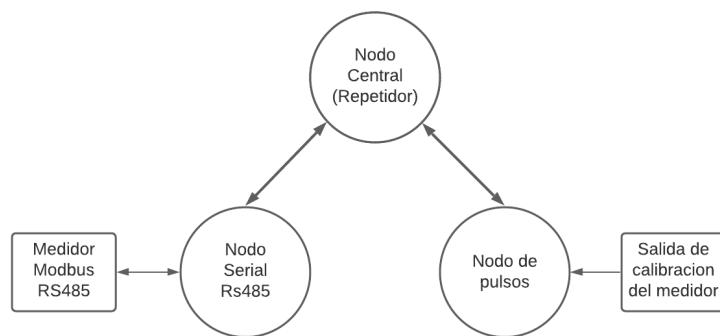


Figura 5.13. Ejemplo de la red restablecida luego de desconectar el nodo central si el nodo repetidor gana la votación.

Tabla 5.5. Tiempo de recuperación de la red cuando es desconectado el nodo central.

Prueba N°	Tiempo de recuperación [ms]
1	25541
2	24598
3	25720
4	25231
5	24896

De las pruebas y el tiempo en la tabla 5.5 se puede observar que el nodo central como nodo fundamental de la red tarda más de 25 segundos en recuperarse, a este tiempo le afectan varios factores: El tiempo que toma en darse cuenta la red que se perdió la conexión con dicho nodo central, luego el tiempo de votación para la elección del nuevo nodo central y por último el tiempo de reordenamiento de la red (puede verse afectado por la cantidad de nodos).

Al ser solo 3 nodos los restantes y tardar dicha cantidad de tiempo, este es un factor a considerar para su uso en aplicaciones críticas.

5.3.2. Para cualquier otro nodo

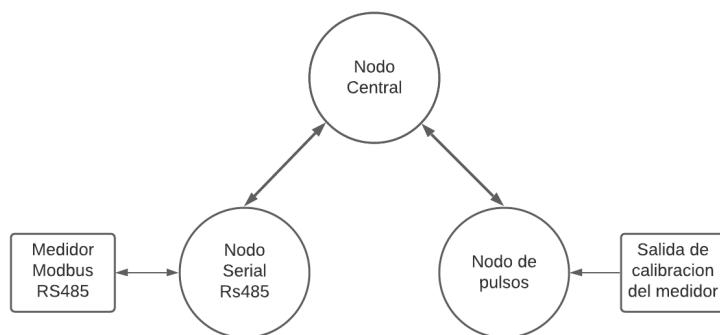


Figura 5.14. Ejemplo de la red restablecida luego de desconectar el nodo repetidor.

Tabla 5.6. Tiempo de recuperación de la red cuando es desconectado un nodo, exceptuando el nodo central.

Prueba N°	Tiempo de recuperación [ms]
1	17776
2	18163
3	17142
4	17964
5	18162
6	18254
7	18422
8	17968
9	17566
10	17001

En este caso (tabla 5.6) se observa que el tiempo es varios segundos menor respecto al tardado para el restablecimiento del nodo central, para un nodo cualquiera no se debe realizar una votación ni el restablecimiento de la totalidad de la red sino que puede ser parcialmente renovada, reduciendo así el tiempo que se tarda la red en sanar la pérdida.

CAPÍTULO VI

CONCLUSIONES

-
-
-
-
-

CAPÍTULO VII

RECOMENDACIONES

-
-
-
-
-

Apéndice I

Código fuente

El siguiente archivo es llamado `mesh.c`, en él se ilustran las tareas y la funcionalidad mallada de la red:

```
1 #include "mesh.h"
2 #include <lwip/netdb.h>
3 #include <sys/param.h>
4 #include "CRC.h"
5 #include "UART1.h"
6 #include "driver/gpio.h"
7 #include "driver/timer.h"
8 #include "esp_event.h"
9 #include "esp_system.h"
10 #include "esp_websocket_client.h"
11 #include "lwip/err.h"
12 #include "lwip/sockets.h"
13 #include "lwip/sys.h"
14 #include "math.h"
15 #include "meters_data.h"
16 #include "nvs_rotate.h"
17 #include "ram-heap.h"
18 #include "task_verify.h"
19 #include "tcpip_adapter.h"
20
21 static uint8_t tx_buf[TX_SIZE] = {
22     0,
23 };
24 static uint8_t rx_buf[RX_SIZE] = {
25     0,
26 };
27 static bool is_mesh_connected = false;
28 static mesh_addr_t mesh_parent_addr;
29 static int mesh_layer = -1;
30
31 static const char *MESH_TAG = "mesh_main";
32 static const char *TCP_TAG = "TCP_SERVER";
33
34 uint8_t SLAVE_ID;
```

```

35     uint16_t fconv;
36     uint16_t port;
37     uint64_t energy_ini;
38     SemaphoreHandle_t smfPulso = NULL;
39     SemaphoreHandle_t smfNVS = NULL;
40     bool creador = true;
41
42     tipo_de_medidor tipo;
43     uint32_t baud_rate;
44
45     /* ***** Tareas del Root *****/
46     void esp_mesh_tx_to_ext(void *arg)
47     {
48         /*
49         * Recibe la trama proveniente del nodo solicitado en la red mesh y
50         * Envía la trama mediante el socket tcp creado en tcp_server_task
51         *
52         */
53
54     mesh_addr_t from;
55     mesh_data_t data;
56     data.data = rx_buf;
57     data.size = sizeof(rx_buf);
58     data.proto = MESH_PROTO_BIN;
59     int flag = 0, sendControl;
60     esp_err_t error;
61     char trama[tamBUFFER];
62
63     INT_VAL len;
64     while (esp_mesh_is_root())
65     {
66         error = esp_mesh_recv(&from, &data, portMAX_DELAY, &flag, NULL,
67                               0);
68         if (error == ESP_OK)
69         {
70             len.Val = rx_buf[4] + rx_buf[5] + 6;
71             for (int i = 0; i < len.Val; i++)
72             {
73                 trama[i] = (char)rx_buf[i];
74             }
75             for (int i = 0; i < len.Val; i++)
76             {
77                 printf("trama[%d] = %02x\r\n", i, trama[i]);
78             }
79             sendControl = send(men, trama, len.Val, 0);
80             if (sendControl < 0 || sendControl != len.Val)
81             {
82                 ESP_LOGE("Tx to Ext", "Error in send");
83             }
84         }
85     }
86     ESP_LOGE(MESH_TAG, "Se elimino tarea TX EXT");

```

```

86     vTaskDelete(NULL);
87 }
88 static void tcp_server_task(void *pvParameters)
89 {
90     /*
91     * Tarea que maneja el servidor TCP, por aquí se realizan conexiones
92     * con
93     * redes externas
94     */
95     char rx_buffer[128];
96     char addr_str[128];
97     int addr_family;
98     int ip_protocol;
99     int sock = 0;
100    struct timeval timeout, timeout_listen;
101    timeout.tv_sec = 20;
102    timeout.tv_usec = 0;
103    timeout_listen.tv_sec = 60 * 5;
104    timeout_listen.tv_usec = 0;
105
106 #ifdef CONFIG_IPV4
107     struct sockaddr_in dest_addr;
108     dest_addr.sin_addr.s_addr = htonl(INADDR_ANY);
109     dest_addr.sin_family = AF_INET;
110     dest_addr.sin_port = htons(port);
111     addr_family = AF_INET;
112     ip_protocol = IPPROTO_IP;
113     inet_ntoa_r(dest_addr.sin_addr, addr_str, sizeof(addr_str) - 1);
114
115 #else // IPV6
116     struct sockaddr_in6 dest_addr;
117     bzero(&dest_addr.sin6_addr.un, sizeof(dest_addr.sin6_addr.un));
118     dest_addr.sin6_family = AF_INET6;
119     dest_addr.sin6_port = htons(port);
120     addr_family = AF_INET6;
121     ip_protocol = IPPROTO_IPV6;
122     inet6_ntoa_r(dest_addr.sin6_addr, addr_str, sizeof(addr_str) - 1);
123 #endif
124
125     int listen_sock =
126         socket(addr_family, SOCK_STREAM, ip_protocol); // Crea el socket
127     if (listen_sock < 0)
128     {
129         ESP_LOGE(TCP_TAG, "Unable to create socket: errno %d", errno);
130     }
131
132     if (setsockopt(listen_sock, SOL_SOCKET, SO_RCVTIMEO, (char *)&
133                     timeout_listen,
134                     sizeof(timeout_listen)) == 0)
135     {
136         printf("Tiempo Listening %d s\r\n", (uint32_t)timeout_listen.

```

```

136         tv_sec);
137     }
138
139     ESP_LOGI(TCP_TAG, "Socket created");
140
141     int err = bind(listen_sock, (struct sockaddr *)&dest_addr,
142                    sizeof(dest_addr)); // Asigna ip al socket
143     if (err != 0)
144     {
145         ESP_LOGE(TCP_TAG, "Socket unable to bind: errno %d", errno);
146     }
147     ESP_LOGI(TCP_TAG, "Socket bound, port %d", port);
148
149     while (esp_mesh_is_root())
150     {
151         err = listen(listen_sock, 1); // A la espera de las posibles
152                     conexiones
153         if (err != 0)
154         {
155             ESP_LOGE(TCP_TAG, "Error occurred during listen: errno %d",
156                      errno);
157             break;
158         }
159         ESP_LOGI(TCP_TAG, "Socket listening");
160
161         struct sockaddr_in6 source_addr; // Large enough for both IPv4 or
162                     IPv6
163         uint addr_len = sizeof(source_addr);
164         sock =
165         accept(listen_sock, (struct sockaddr *)&source_addr,
166                &addr_len); // Se acepta la conexión en caso de poder realizarse
167
168         if (setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout,
169                        sizeof(timeout)) == 0)
170         {
171             printf("Configurado Timeout %d s\r\n", (uint32_t)timeout.tv_sec
172                  );
173         }
174         men = sock;
175
176         if (sock < 0)
177         {
178             ESP_LOGE(TCP_TAG, "Unable to accept connection: errno %d",
179                      errno);
180             continue;
181         }
182         ESP_LOGI(TCP_TAG, "Socket accepted");
183
184         while (esp_mesh_is_root())
185         {
186             printf("Recibiendo TCP...\r\n");
187             int len = recv(sock, rx_buffer, sizeof(rx_buffer) - 1,

```

```

182     0); // Se queda esperando en el estado de recepción
183
184     // Error occurred during receiving
185     if (len < 0)
186     {
187         ESP_LOGE(TCP_TAG, "recv failed: errno %d", errno);
188         timer_pause(TIMER_GROUP_0, TIMER_0);
189
190         break;
191     }
192
193     // Connection closed
194     else if (len == 0)
195     {
196         ESP_LOGI(TCP_TAG, "Connection closed");
197         break;
198     }
199     // Data received
200     else
201     {
202         // Get the sender's ip address as string
203         if (source_addr.sin6_family == PF_INET)
204         {
205             inet_ntoa_r((( struct sockaddr_in *) & source_addr) -> sin_addr.
206                         s_addr,
207                         addr_str, sizeof(addr_str) - 1);
208         }
209         else if (source_addr.sin6_family == PF_INET6)
210         {
211             inet6_ntoa_r(source_addr.sin6_addr, addr_str, sizeof(
212                         addr_str) - 1);
213
214             rx_buffer[len] =
215             0; // Null-terminate whatever we received and treat like a
216                 // string
217             ESP_LOGI(TCP_TAG, "Received %d bytes from %s:", len, addr_str
218                     );
219             ESP_LOGI(TCP_TAG, "%s", rx_buffer);
220
221             xQueueSendToFront(RxSocket, rx_buffer, pdMS_TO_TICKS(1000));
222         }
223     }
224
225     if (sock != -1)
226     {
227         ESP_LOGE(TCP_TAG, "Shutting down socket and restarting...");  

228         shutdown(sock, 0);
229         close(sock);
230         // socket(addr_family, SOCK_STREAM, ip_protocol);
231     }
232 }
```

```

230     ESP_LOGE(MESH_TAG, "Se Elimino la tarea TCP\r\n");
231     close(sock);
232     vTaskDelete(NULL);
233 }
234 void esp_mesh_p2p_tx_main(void *Pa)
235 {
236     /*
237     * Tarea para repartir el mensaje recibido en el servidor por
238     * Broadcast de la
239     * red Mesh
240     */
241
242     esp_err_t err;
243     mesh_data_t data;
244     char mensaje2[tamBUFFER] = "";
245     data.proto = MESH_PROTO_BIN;
246     data.data = tx_buf;
247     data.size = sizeof(tx_buf);
248     data.tos = MESH_TOS_P2P;
249
250     int tamano, len = 0;
251     mesh_addr_t rt[CONFIG_MESH_ROUTE_TABLE_SIZE];
252
253     while (esp_mesh_is_root())
254     {
255         xQueueReceive(RxSocket, (char *)mensaje2, portMAX_DELAY);
256
257         tamano = 6 + (uint8_t)mensaje2[4] + (uint8_t)mensaje2[5];
258
259         for (int i = 0; i < tamano; i++)
260         {
261             tx_buf[i] = (uint8_t)mensaje2[i];
262         }
263         esp_mesh_get_routing_table((mesh_addr_t *)&rt,
264 CONFIG_MESH_ROUTE_TABLE_SIZE * 6, &len);
265
266         printf("Mandando a Nodo\r\n");
267
268         for (int i = 1; i < len; i++)
269         {
270             err = esp_mesh_send(&rt[i], &data, MESH_DATA_P2P, NULL, 0);
271             if (err != ESP_OK)
272             {
273                 ESP_LOGW(MESH_TAG, "\r\nMensaje no enviado\r\n");
274             }
275         }
276         ESP_LOGE(MESH_TAG, "Se elimino tarea Tx main\r\n");
277         vTaskDelete(NULL);
278     }
279
280 /* ***** Tareas de un Nodo ***** */

```

```

281  /* *** Bus RS485 Standard *** */
282  void esp_mesh_p2p_rx_main(void *arg)
283  {
284      INT_VAL len;
285      INT_VAL CRC;
286      esp_err_t err;
287      mesh_addr_t from;
288      mesh_data_t data;
289      int flag = 0;
290      data.data = rx_buf;
291      data.size = sizeof(rx_buf);
292      mesh_rx_pending_t pendientes, auxi = { .toSelf = 0 };
293      uint8_t trama[tamBUFFER];
294      const unsigned char *aux = &trama[4];
295
296      while (!esp_mesh_is_root())
297      {
298          printf("Esperando...\r\n");
299          err = esp_mesh_recv(&from, &data, portMAX_DELAY, &flag, NULL, 0);
300          esp_mesh_get_rx_pending(&pendientes);
301          if (pendientes.toSelf != auxi.toSelf)
302          {
303              printf("Pendientes = %d\r\n", pendientes.toSelf);
304              auxi.toSelf = pendientes.toSelf;
305          }
306          switch (err)
307          {
308              case ESP_OK:
309              {
310                  printf("Recibido por Mesh\r\n");
311
312                  if (!esp_mesh_is_root())
313                  {
314                      len.Val = rx_buf[4] + rx_buf[5];
315
316                      for (uint16_t i = 0; i < len.Val; i++)
317                      {
318                          trama[i + 4] = rx_buf[6 + i];
319                      }
320                      CRC.Val = CRC16(aux, len.Val);
321                      len.Val = len.Val + 2;
322                      trama[len.Val + 2] = CRC.byte.LB;
323                      trama[len.Val + 3] = CRC.byte.HB;
324                      trama[0] = rx_buf[0];
325                      trama[1] = rx_buf[1];
326                      trama[2] = len.byte.HB;
327                      trama[3] = len.byte.LB;
328                      xQueueSendToFront(TxRS485, &trama, portMAX_DELAY);
329                  }
330              }
331              break;
332              case ESP_ERR_MESH_NOT_START:

```

```

333     {
334         ESP_LOGE(MESH_TAG, "Aun no esta Iniciada la Mesh\r\n");
335     }
336     break;
337     case ESP_ERR_MESH_TIMEOUT:
338     {
339         ESP_LOGW(MESH_TAG, "Timeout Error\r\n");
340     }
341     break;
342     default:
343         ESP_ERROR_CHECK_WITHOUT_ABORT( err );
344         break;
345     }
346 }
347
348 ESP_LOGW(MESH_TAG, "Tarea Rx main eliminada");
349 vTaskDelete(NULL);
350 }
351 void bus_rs485(void *arg)
352 {
353     INT_VAL longitud;
354     mesh_data_t dataMesh;
355     BaseType_t ctrl_cola;
356
357     dataMesh.proto = MESH_PROTO_BIN;
358     dataMesh.size = tamBUFFER;
359     dataMesh.data = tx_buf;
360     dataMesh.tos = MESH_TOS_P2P;
361
362     uint16_t txlen, txctrl;
363
364     uint8_t dataTx[tamBUFFER];
365     uint8_t *aux = &dataTx[4];
366
367     while (!esp_mesh_is_root())
368     {
369         // Espera a recibir la trama RTU
370
371         xQueueReceive(TxRS485, (uint8_t *)dataTx, portMAX_DELAY);
372         printf("Recibido en cola...\r\n");
373         txlen = dataTx[2] + dataTx[3];
374
375         txctrl = uart_write_bytes(uart1, (char *)aux, txlen);
376
377         if (txctrl > 0)
378         {
379             ESP_LOGI("RS485", "Tx FIFO: %u", txctrl);
380             ctrl_cola =
381             xQueueReceive(RxRS485, (uint8_t *)tx_buf + 4, pdMS_TO_TICKS
382             (10000));
383             if (ctrl_cola == pdTRUE)
384             {

```

```

384     tx_buf[0] = dataTx[0];
385     tx_buf[1] = dataTx[1];
386     tx_buf[2] = 0;
387     tx_buf[3] = 0;
388     longitud.Val = tx_buf[4] + tx_buf[5] - 2;
389     tx_buf[4] = longitud.byte.HB;
390     tx_buf[5] = longitud.byte.LB;
391     tx_buf[4 + longitud.Val] = 0x00;
392     tx_buf[4 + longitud.Val + 1] = 0x00;
393     for (int i = 0; i < (longitud.Val + 4); i++)
394     {
395         printf("tx_buf[ %d ] = %02x\r\n", i, tx_buf[i]);
396     }
397     esp_err_t err = esp_mesh_send(NULL, &dataMesh, MESH_DATA_P2P,
398                                   NULL, 0);
399     if (err == ESP_OK)
400     {
401         printf("Mandado\r\n");
402     }
403     else
404     {
405         ESP_LOGW("RS485", "Queue timeout proceeding to UART1 Flush");
406         uart_flush(UART_NUM_1);
407     }
408 }
409 vTaskDelete(NULL);
410 }

411 /* **** Medidor de pulsos *****/
412 /* Interrupcion de entrada de pulso*/
413 void IRAM_ATTR INT_GPIO_PULSOS(void *arg)
414 {
415     xSemaphoreGiveFromISR(smfPulso, NULL);
416 }
417 void modbus_tcpip_pulsos(void *arg)
418 {
419     esp_err_t err;
420     int flag = 0;
421     mesh_addr_t from;
422     mesh_data_t data_rx, data_tx;
423     data_rx.data = rx_buf;
424     data_rx.size = RX_SIZE;
425     data_tx.data = tx_buf;
426     data_tx.size = TX_SIZE;
427
428     energype_t energia;
429
430     while (!esp_mesh_is_root())
431     {
432         ESP_LOGI("MODBUS", "Entré aquí");

```

```

435     err = esp_mesh_recv(&from, &data_rx, portMAX_DELAY, &flag, NULL,
436                         0);
437     ESP_LOGI("MODBUS", "Mensaje Recibido");
438     ESP_LOGI("MODBUS RX", " %x %x",
439              rx_buf[0],
440              rx_buf[1], rx_buf[2], rx_buf[3], rx_buf[4], rx_buf[5], rx_buf[6],
441              rx_buf[7], rx_buf[8], rx_buf[9], rx_buf[10], rx_buf[11]);
442
443     if ((err == ESP_OK) && (rx_buf[6] == SLAVE_ID) && (rx_buf[7] == 0
444             x03) &&
445             (rx_buf[8] == MODBUS_ENERGY_REG_INIT_POS_H) &&
446             (rx_buf[9] == MODBUS_ENERGY_REG_INIT_POS_L) &&
447             (rx_buf[11] == MODBUS_ENERGY_REG_LEN))
448     {
449         xQueuePeek(Cuenta_de_pulsos, &(energia.tot), portMAX_DELAY);
450         tx_buf[0] = rx_buf[0];
451         tx_buf[1] = rx_buf[1];
452         tx_buf[2] = 0x00;
453         tx_buf[3] = 0x00;
454         tx_buf[4] = 0x00; // len hb
455         tx_buf[5] = 0x0b; // len lb
456         tx_buf[6] = SLAVE_ID; // SlaveID
457         tx_buf[7] = 0x03;
458         tx_buf[8] = 0x08; // byte count
459         tx_buf[9] = energia.u8.1118;
460         tx_buf[10] = energia.u8.1118;
461         tx_buf[11] = energia.u8.18;
462         tx_buf[12] = energia.u8.118;
463         tx_buf[13] = energia.u8.hh8;
464         tx_buf[14] = energia.u8.h8;
465         tx_buf[15] = energia.u8.hhhh8;
466         tx_buf[16] = energia.u8.hhh8;
467         err = esp_mesh_send(NULL, &data_tx, MESH_DATA_P2P, NULL, 0);
468         if (err == ESP_OK)
469         {
470             ESP_LOGI("MODBUS", "Respuesta enviada");
471         }
472     }
473     vTaskDelete(NULL);
474 }
475 static void rotar_nvs(void *arg)
476 {
    /* Nomenclatura:
     *   pf: Página fija. Es la pagina que necesita acceder el micro
     *       para saber
     * en qué página variable se encuentra. en ella se encuentra un
     *       registro de
     * igual nombre (pf) que contiene el número de paginas variables
     *       llenas.
     */

```

```

481     *      pv: Página variable. En esta página se guarda la cuenta de cada
482     *      pulso
483     * registrada. Las páginas variables como su nombre lo indica van
484     * variando a
485     * medida que crece la cuenta de pulsos.
486     *
487     * partition: Espacio de memoria flash del micro en la que se
488     * encuentra
489     * actualmente el último pulso.
490     */
491
492     uint8_t partition_number; // Index de partition name (app#)
493
494     int32_t indexPv,           // Index del namespace (pv#)
495     indexEntradaActual, // Index de la entrada activa en la pv (e#)
496     cuentaPulsosPv;        // Cuenta de pulsos en la entrada activa
497
498     uint64_t total_pulsos, // Total de pulsos: suma de los pulsos en
499     // cada
500     // particion, página y entrada utilizada
501     inicial_pulsos;
502     char *pvActual,          // String para namespace de la página
503     // variable
504     *entradaActualpv, // String para el key de la entrada actual
505     *partition_name; // String para el nombre de la partición
506
507     inicial_pulsos = round((float)energy_ini * (float)fconv);
508     xQueueOverwrite(Cuenta_de_pulsos, &total_pulsos);
509
510     esp_err_t err = search_init_partition(&partition_number);
511
512     if (asprintf(&partition_name, "app%d", partition_number) < 0)
513     {
514         free(partition_name);
515         ESP_LOGE("ROTAR_NVS", "Nombre de particion no fue creado");
516     }
517
518     nvs_flash_init_partition(partition_name);
519
520     show_ram_status("Partición iniciada");
521
522     /*
523     * Leyendo el valor del contador de la pagina fija. Retorna el
524     * namespace y el
525     * número de pagina variable se encuentran los datos más recientes
526     */
527     err = leer_contador_pf(&partition_name, &pvActual, &indexPv);
528     if (err != ESP_OK)
529     printf("Error (%s) leyendo contador desde la pagina fija!\n",
530           esp_err_to_name(err));
531     ESP_LOGI("LPF", "pvActual: %s indexPv: %d", pvActual, indexPv);

```

```

527     ESP_LOGI("DEBUG", "Antes de leer pagina variable");
528
529     /*
530      * Revisar pagina variable actual y buscar el ultimo registro
531      * escrito
532      */
533     err = leer_pagina_variable(&partition_name, &pvActual, &
534                               indexEntradaActual,
535                               &entradaActualpv, &cuentaPulsosPv);
536     if (err != ESP_OK)
537         printf("Error (%s) buscando el registro escrito más reciente\n",
538                esp_err_to_name(err));
539     ESP_LOGI("LPV",
540             "pvActual: %s indexEntradaActual: %d Entrada actual: %s Cuenta "
541             "'pulsos: %d",
542             pvActual, indexEntradaActual, entradaActualpv, cuentaPulsosPv);
543
544     ESP_LOGI("DEBUG", "Antes del while");
545
546     while (!esp_mesh_is_root())
547     {
548         // Interrupción generada por pulsos
549         if (xSemaphoreTake(smfPulso, portMAX_DELAY) == pdTRUE)
550         {
551             /*Si se llegó al límite en una particion*/
552             if (indexPv == Limite_paginas_por_particion &&
553                 indexEntradaActual == Limite_entradas_por_pagina &&
554                 cuentaPulsosPv == Limite_pulsos_por_entrada)
555             {
556                 partition_number++;
557
558                 if (partition_number <= max_particiones)
559                 {
560                     levantar_bandera(partition_name);
561
562                     char *aux;
563                     nvs_handle_t my_handle;
564
565                     // Cerrando la particion anterior
566                     if (asprintf(&aux, "app%d", partition_number - 1) < 0)
567                     {
568                         free(aux);
569                         ESP_LOGE("ROTAR_NVS", "Nombre de particion no fue creado");
570                     }
571                     ESP_LOGI("DEINIT", "%s", aux);
572                     err = nvs_flash_deinit_partition(aux);
573                     if (err != ESP_OK)
574                         ESP_LOGE("CNP", "ERROR (%s) IN DEINIT", esp_err_to_name(err));
575                     else
576                         free(aux);
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
280
```

```

575
576     // Inicializando la nueva partición
577     if (asprintf(&partition_name, "app%u", partition_number) <
578         0)
579     {
580         free(partition_name);
581         ESP_LOGE("ROTAR_NVS", "Nombre de particion no fue creado");
582     }
583     ESP_LOGW("CNP", "Partition changed to %s", partition_name);
584     nvs_flash_init_partition(partition_name);
585
586     ESP_LOGI("PARTICION ACTUAL", "DEBUG 2 %s", partition_name);
587
588     // Llenando particion
589     esp_err_t err = nvs_open_from_partition(partition_name, "storage",
590                                             NVS_READWRITE, &my_handle);
591     if (err != ESP_OK)
592     ESP_LOGE("NVS", "ERROR IN NVS_OPEN");
593     //    free(*pname);
594
595     // Colocando la bandera de llenado en 0
596     err = nvs_set_u8(my_handle, "finished", 0);
597     if (err != ESP_OK)
598     ESP_LOGE("NVS", "ERROR IN SET");
599     err = nvs_commit(my_handle);
600     if (err != ESP_OK)
601     ESP_LOGE("NVS", "ERROR IN COMMIT");
602
603     // Get del número de la partición
604     err = nvs_get_u8(my_handle, "pnumber", &partition_number);
605     if (err != ESP_OK && err != ESP_ERR_NVS_NOT_FOUND)
606     ESP_LOGE("NVS", "ERROR IN GET");
607     else if (err == ESP_ERR_NVS_NOT_FOUND)
608     {
609         err = nvs_set_u8(my_handle, "pnumber", partition_number);
610         if (err != ESP_OK)
611             ESP_LOGE("NVS", "ERROR IN SET");
612         err = nvs_commit(my_handle);
613         if (err != ESP_OK)
614             ESP_LOGE("NVS", "ERROR IN COMMIT");
615     }
616     // Close
617     nvs_close(my_handle);
618 }
619
620 if (partition_number <= max_particiones)
621 {
622     err = contar_pulsos_nvs(&partition_name, &partition_number, &
623     indexPv,

```

```

623     &indexEntradaActual , &cuentaPulsosPv);
624
625     total_pulsos =
626         inicial_pulsos + // Aporte de los pulsos iniciales
627         (partition_number - 1) * Limite_paginas_por_particion *
628         Limite_entradas_por_pagina *
629         Limite_pulsos_por_entrada + // Aporte c/u de las particiones
630         (indexPv - 1) * Limite_entradas_por_pagina *
631         Limite_pulsos_por_entrada + // Aporte de las paginas
632         anteriores
633         (indexEntradaActual - 1) *
634         Limite_pulsos_por_entrada + // Aporte de la página actual
635         cuentaPulsosPv;           // Aporte de la entrada
636         actual
637
638     ESP_LOGI("PULSOS", "PARTICION: %d PV: %d ENT: %d Pulsos: %d",
639             partition_number, indexPv, indexEntradaActual, cuentaPulsosPv
640             );
641     ESP_LOGI("TOTAL PULSOS", "%llu", total_pulsos);
642
643     // Enviando los pulsos a otra tarea
644     xQueueOverwrite(Cuenta_de_pulsos , &total_pulsos);
645 }
646 else
647     ESP_LOGE("APP", "All partitions are full");
648     show_ram_status("Por pulsos");
649 }
650
651 free(pvActual);
652 free(entradaActualpv);
653 free(partition_name);
654 ESP_LOGE("RTOS", "La tarea NVS-Rotative ha sido eliminada");
655 vTaskDelete(NULL);
656
657 /* *** Manejadores de eventos *** */
658 void mesh_event_handler(void *arg, esp_event_base_t event_base,
659 int32_t event_id, void *event_data)
660 {
661     char rx_child[9] = "Rx_child";
662     char rx_rs485[9] = "Rx_RS485";
663     char modbus_pulse[11] = "Comun P2P";
664     char count_pulse[] = "Rotative NVS";
665     mesh_addr_t id = {
666         0,
667     };
668     static uint8_t last_layer = 0;
669
670     switch (event_id)
671     {
672         case MESH_EVENT_STARTED:

```

```

672 {
673     esp_mesh_get_id(&id);
674     ESP_LOGI(MESH_TAG, "<MESH_EVENT_MESH_STARTED>ID:" MACSTR "", 
675             MAC2STR(id .addr));
676     is_mesh_connected = false;
677     mesh_layer = esp_mesh_get_layer();
678 }
679 break;
680 case MESH_EVENT_STOPPED:
681 {
682     ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOPPED>");
683     is_mesh_connected = false;
684     mesh_layer = esp_mesh_get_layer();
685 }
686 break;
687 case MESH_EVENT_CHILD_CONNECTED:
688 {
689     mesh_event_child_connected_t *child_connected =
690     (mesh_event_child_connected_t *)event_data;
691     ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_CONNECTED>aid: %d, "
692             MACSTR "", 
693             child_connected->aid, MAC2STR(child_connected->mac));
694 }
695 break;
696 case MESH_EVENT_CHILD_DISCONNECTED:
697 {
698     mesh_event_child_disconnected_t *child_disconnected =
699     (mesh_event_child_disconnected_t *)event_data;
700     ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_DISCONNECTED>aid: %d, "
701             MACSTR "", 
702             child_disconnected->aid, MAC2STR(child_disconnected->mac));
703 }
704 break;
705 case MESH_EVENT_ROUTING_TABLE_ADD:
706 {
707     mesh_event_routing_table_change_t *routing_table =
708     (mesh_event_routing_table_change_t *)event_data;
709     ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_ADD>add %d, new: % 
710             d",
711             routing_table->rt_size_change , routing_table->rt_size_new );
712 }
713 break;
714 case MESH_EVENT_ROUTING_TABLE_REMOVE:
715 {
716     mesh_event_routing_table_change_t *routing_table =
717     (mesh_event_routing_table_change_t *)event_data;
718     ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_REMOVE>remove %d, 
719             new: %d",
720             routing_table->rt_size_change , routing_table->rt_size_new );
721 }
722 break;
723 case MESH_EVENT_NO_PARENT_FOUND:

```

```

720 {
721     mesh_event_no_parent_found_t *no_parent =
722     (mesh_event_no_parent_found_t *)event_data;
723     ESP_LOGI(MESH_TAG, "<MESH_EVENT_NO_PARENT_FOUND>scan times: %d",
724             no_parent->scan_times);
725 }
726 /* TODO handler for the failure */
727 break;
728 case MESH_EVENT_PARENT_CONNECTED:
729 {
730     gpio_set_level(LED_PAPA, 1);
731     mesh_event_connected_t *connected = (mesh_event_connected_t *)
732         event_data;
733     esp_mesh_get_id(&id);
734     mesh_layer = connected->self_layer;
735     memcpy(&mesh_parent_addr.addr, connected->connected.bssid, 6);
736     ESP_LOGI(
737         MESH_TAG,
738         "<MESH_EVENT_PARENT_CONNECTED>layer: %d-->%d, parent:" MACSTR
739         " %s, ID:" MACSTR "",
740         last_layer, mesh_layer, MAC2STR(mesh_parent_addr.addr),
741         esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>"
742         : "",
743         MAC2STR(id.addr));
744     last_layer = mesh_layer;
745     is_mesh_connected = true;
746     if (esp_mesh_is_root())
747     {
748         tcpip_adapter_dhcpc_start(TCPIP_ADAPTER_IF_STA);
749     }
750     else
751     {
752         switch (tipo)
753         {
754             case (rs485):
755                 creador = vTaskB(rx_child);
756                 if (creador)
757                 {
758                     xTaskCreatePinnedToCore(esp_mesh_p2p_rx_main, rx_child,
759                     3072 * 2,
760                     NULL, 5, NULL, 0);
761                 }
762                 creador = vTaskB(rx_rs485);
763                 if (creador)
764                 {
765                     xTaskCreatePinnedToCore(bus_rs485, rx_rs485, 3072 * 3,
766                     NULL, 5,
767                     NULL, 1);
768                     iniciarUART(tipo, baud_rate);
769                 }
770             break;
771             case (pulsos):

```

```

769     creador = vTaskB(modbus_pulse);
770     if (creador)
771     {
772         xTaskCreatePinnedToCore(modbus_tcpip_pulsos, modbus_pulse
773             ,
774             3072 * 2, NULL, 5, NULL, 0);
775     }
776     creador = vTaskB(count_pulse);
777     if (creador)
778         xTaskCreatePinnedToCore(rotar_nvs, count_pulse, 4 * 1024,
779             NULL, 5,
780             NULL, 1);
781     break;
782     case (chino):
783     creador = vTaskB(rx_child);
784     if (creador)
785     {
786         xTaskCreatePinnedToCore(esp_mesh_p2p_rx_main, rx_child,
787             3072 * 2,
788             NULL, 5, NULL, 0);
789     }
790     creador = vTaskB(rx_rs485);
791     if (creador)
792     {
793         xTaskCreatePinnedToCore(bus_rs485, rx_rs485, 3072 * 3,
794             NULL, 5,
795             NULL, 1);
796         iniciarUART(tipo, baud_rate);
797     }
798     break;
799     case (enlace):
800     break;
801     case MESH_EVENT_PARENT_DISCONNECTED:
802     {
803         gpio_set_level(LED_PAPA, 0);
804         mesh_event_disconnected_t *disconnected =
805             (mesh_event_disconnected_t *)event_data;
806         ESP_LOGI(MESH_TAG, "<MESH_EVENT_PARENT_DISCONNECTED>reason: %d",
807             disconnected->reason);
808         is_mesh_connected = false;
809         mesh_layer = esp_mesh_get_layer();
810     }
811     break;
812     case MESH_EVENT_LAYER_CHANGE:
813     {
814         mesh_event_layer_change_t *layer_change =
815             (mesh_event_layer_change_t *)event_data;
816         mesh_lay

```

```

817     } er = layer_change->new_layer;
818     ESP_LOGI(
819         MESH_TAG, "<MESH_EVENT_LAYER_CHANGE>layer: %d-->%d %s",
820         last_layer,
821         mesh_layer,
822         esp_mesh_is_root() ? "<ROOT>" : (mesh_layer == 2) ? "<layer2>" :
823         "";
824         last_layer = mesh_layer;
825     }
826     break;
827 case MESH_EVENT_ROOT_ADDRESS:
828 {
829     mesh_event_root_address_t *root_addr =
830     (mesh_event_root_address_t *)event_data;
831     ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_ADDRESS>root address:" MACSTR "", MAC2STR(root_addr->addr));
832     for (int i = 0; i < 6; i++)
833     {
834         root_address.addr[i] = root_addr->addr[i];
835     }
836     break;
837 case MESH_EVENT_VOTE_STARTED:
838 {
839     mesh_event_vote_started_t *vote_started =
840     (mesh_event_vote_started_t *)event_data;
841     ESP_LOGI(
842         MESH_TAG,
843         "<MESH_EVENT_VOTE_STARTED>attempts: %d, reason: %d, rc_addr:" MACSTR "", vote_started->attempts, vote_started->reason, MAC2STR(vote_started->rc_addr.addr));
844     }
845     break;
846 case MESH_EVENT_VOTE_STOPPED:
847 {
848     ESP_LOGI(MESH_TAG, "<MESH_EVENT_VOTE_STOPPED>");
849     break;
850 }
851 case MESH_EVENT_ROOT_SWITCH_REQ:
852 {
853     mesh_event_root_switch_req_t *switch_req =
854     (mesh_event_root_switch_req_t *)event_data;
855     ESP_LOGI(MESH_TAG,
856         "<MESH_EVENT_ROOT_SWITCH_REQ>reason: %d, rc_addr:" MACSTR "", switch_req->reason, MAC2STR(switch_req->rc_addr.addr));
857     }
858     break;
859 case MESH_EVENT_ROOT_SWITCH_ACK:
860 {
861     /* new root */
862     mesh_layer = esp_mesh_get_layer();
863 }

```

```

866     esp_mesh_get_parent_bssid(&mesh_parent_addr);
867     ESP_LOGI(MESH_TAG,
868             "<MESH_EVENT_ROOT_SWITCH_ACK>layer: %d, parent:" MACSTR "", 
869             mesh_layer, MAC2STR(mesh_parent_addr.addr));
870 }
871 break;
872 case MESH_EVENT_TODS_STATE:
873 {
874     mesh_event_tod_s_state_t *tods_state =
875         (mesh_event_tod_s_state_t *)event_data;
876     ESP_LOGI(MESH_TAG, "<MESH_EVENT_TODS_REACHABLE>state: %d", *
877             tod_s_state);
878 }
879 break;
880 case MESH_EVENT_ROOT_FIXED:
881 {
882     mesh_event_root_fixed_t *root_fixed =
883         (mesh_event_root_fixed_t *)event_data;
884     ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_FIXED>%s",
885             root_fixed->is_fixed ? "fixed" : "not fixed");
886 }
887 break;
888 case MESH_EVENT_ROOTASKED_YIELD:
889 {
890     mesh_event_root_conflict_t *root_conflict =
891         (mesh_event_root_conflict_t *)event_data;
892     ESP_LOGI(MESH_TAG,
893             "<MESH_EVENT_ROOTASKED_YIELD>" MACSTR ", rssi: %d, capacity: %d"
894             ,
895             MAC2STR(root_conflict->addr), root_conflict->rssi,
896             root_conflict->capacity);
897 }
898 break;
899 case MESH_EVENT_CHANNEL_SWITCH:
900 {
901     mesh_event_channel_switch_t *channel_switch =
902         (mesh_event_channel_switch_t *)event_data;
903     ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHANNEL_SWITCH>new channel: %d",
904             channel_switch->channel);
905 }
906 break;
907 case MESH_EVENT_SCAN_DONE:
908 {
909     mesh_event_scan_done_t *scan_done = (mesh_event_scan_done_t *)
910         event_data;
911     ESP_LOGI(MESH_TAG, "<MESH_EVENT_SCAN_DONE>number: %d", scan_done
912             ->number);
913 }
914 break;
915 case MESH_EVENT_NETWORK_STATE:
916 {
917     mesh_event_network_state_t *network_state =

```

```

914     ( mesh_event_network_state_t *)event_data ;
915     ESP_LOGI(MESH_TAG, "<MESH_EVENT_NETWORK_STATE>is_rootless: %d",
916               network_state ->is_rootless );
917 }
918 break;
919 case MESH_EVENT_STOP_RECONNECTION:
920 {
921     ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOP_RECONNECTION>");
922 }
923 break;
924 case MESH_EVENT_FIND_NETWORK:
925 {
926     mesh_event_find_network_t *find_network =
927     ( mesh_event_find_network_t *)event_data ;
928     ESP_LOGI(MESH_TAG,
929             "<MESH_EVENT_FIND_NETWORK>new channel: %d, router BSSID:" MACSTR
930             " " ,
931             find_network ->channel , MAC2STR( find_network ->router_bssid));
932 }
933 break;
934 case MESH_EVENT_ROUTER_SWITCH:
935 {
936     mesh_event_router_switch_t *router_switch =
937     ( mesh_event_router_switch_t *)event_data ;
938     ESP_LOGI(MESH_TAG,
939             "<MESH_EVENT_ROUTER_SWITCH>new router: %s, channel: %d, " MACSTR
940             " " ,
941             router_switch ->ssid , router_switch ->channel ,
942             MAC2STR( router_switch ->bssid));
943 }
944 break;
945 default:
946     ESP_LOGI(MESH_TAG, "unknown id: %d", event_id);
947     break;
948 }
949 }
950 void ip_event_handler( void *arg , esp_event_base_t event_base , int32_t
951 event_id ,
952 void *event_data )
{
953     char tcp_server[11] = "tcp_server";
954     char tx_root[3] = "TX";
955     char tx_ext[7] = "TX_EXT";
956     ip_event_got_ip_t *event = ( ip_event_got_ip_t *)event_data ;
957     ESP_LOGI(MESH_TAG, "<IP_EVENT_STA_GOT_IP>IP: %s",
958             ip4addr_ntoa(&event ->ip_info.ip));
959     creator = vTaskB(tcp_server);
960     if ( creator )
961     {
962         xTaskCreatePinnedToCore( tcp_server_task , "tcp_server" , 1024 * 4 ,
963             NULL, 1 ,
964             NULL, 1 );

```

```

964 }
965 creador = vTaskB(tx_root);
966 if (creador)
967 {
968     xTaskCreatePinnedToCore(esp_mesh_p2p_tx_main, "TX", 1024 * 3,
969     NULL, 5, NULL,
970     0);
971 }
972 creador = vTaskB(tx_ext);
973 if (creador)
974 {
975     xTaskCreatePinnedToCore(esp_mesh_tx_to_ext, "TX_EXT", 3072 * 2,
976     NULL, 5,
977     NULL, 0);
978 }
979 /* **** Configuracion e inicializacion ****/
980 /* **** Configuracion e inicializacion ****/
981 /* **** Configuracion e inicializacion ****/
982 void config_gpio_pulsos(tipo_de_medidor tipo)
983 {
984     if (tipo == pulsos)
985     {
986         ESP_LOGI(MESH_TAG, "Configurando GPIO para medidor tipo pulsos");
987         // Tipo de interrupcion
988         gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
989
990         gpio_pad_select_gpio(PULSOS);
991         gpio_set_direction(PULSOS, GPIO_MODE_DEF_INPUT);
992         gpio_isr_handler_add(PULSOS, INT_GPIO_PULSOS, NULL);
993         gpio_set_intr_type(PULSOS, GPIO_INTR_POSEDGE);
994     }
995     else
996     {
997         ESP_LOGI(MESH_TAG, "Configurando GPIO para medidor tipo RS485");
998         gpio_pad_select_gpio(RS485);
999         gpio_set_direction(RS485, GPIO_MODE_DEF_OUTPUT);
1000    }
1001    gpio_pad_select_gpio(LED_PAPA);
1002    gpio_set_direction(LED_PAPA, GPIO_MODE_DEF_OUTPUT);
1003 }
1004 /* Inicio Mesh*/
1005 void mesh_init(form_mesh fmesh, form_locwifi fwifi, form_modbus
    fmodbus)

```

Ahora compila usando `gcc`:

```
$ gcc -o hello hello.c
```

Apéndice II

Datasheet MAX3485

Click [here](#) for production status of specific part numbers.

**MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491**

3.3V-Powered, 10Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers

General Description

The MAX3483, MAX3485, MAX3486, MAX3488, MAX3490, and MAX3491 are 3.3V, low-power transceivers for RS-485 and RS-422 communication. Each part contains one driver and one receiver. The MAX3483 and MAX3488 feature slew-rate-limited drivers that minimize EMI and reduce reflections caused by improperly terminated cables, allowing error-free data transmission at data rates up to 250kbps. The partially slew-rate-limited MAX3486 transmits up to 2.5Mbps. The MAX3485, MAX3490, and MAX3491 transmit at up to 10Mbps.

Drivers are short-circuit current-limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if both inputs are open circuit.

The MAX3488, MAX3490, and MAX3491 feature full-duplex communication, while the MAX3483, MAX3485, and MAX3486 are designed for half-duplex communication.

Applications

- Low-Power RS-485/RS-422 Transceivers
- Telecommunications
- Transceivers for EMI-Sensitive Applications
- Industrial-Control Local Area Networks

Features

- Operate from a Single 3.3V Supply—No Charge Pump!
- Interoperable with +5V Logic
- 8ns Max Skew (MAX3485/MAX3490/MAX3491)
- Slew-Rate Limited for Errorless Data Transmission (MAX3483/MAX3488)
- 2nA Low-Current Shutdown Mode (MAX3483/MAX3485/MAX3486/MAX3491)
- -7V to +12V Common-Mode Input Voltage Range
- Allows up to 32 Transceivers on the Bus
- Full-Duplex and Half-Duplex Versions Available
- Industry Standard 75176 Pinout (MAX3483/MAX3485/MAX3486)
- Current-Limiting and Thermal Shutdown for Driver Overload Protection

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX3483CPA	0°C to +70°C	8 Plastic DIP
MAX3483CSA	0°C to +70°C	8 SO
MAX3483C/D	0°C to +70°C	Dice*
MAX3483EPA	-40°C to +85°C	8 Plastic DIP
MAX3483ESA	-40°C to +85°C	8 SO
MAX3485CPA	0°C to +70°C	8 Plastic DIP
MAX3485CSA	0°C to +70°C	8 SO
MAX3485C/D	0°C to +70°C	Dice*
MAX3485EPA	-40°C to +85°C	8 Plastic DIP
MAX3485ESA	-40°C to +85°C	8 SO

Ordering Information continued at end of data sheet.

* Contact factory for dice specifications.

Selection Table

PART NUMBER	GUARANTEED DATA RATE (Mbps)	SUPPLY VOLTAGE (V)	HALF/FULL DUPLEX	SLEW-RATE LIMITED	DRIVER/ RECEIVER ENABLE	SHUTDOWN CURRENT (nA)	PIN COUNT
MAX3483	0.25	3.0 to 3.6	Half	Yes	Yes	2	8
MAX3485	10		Half	No	No	2	8
MAX3486	2.5		Half	Yes	Yes	2	8
MAX3488	0.25		Half	Yes	Yes	—	8
MAX3490	10		Half	No	No	—	8
MAX3491	10		Half	No	Yes	2	14

**MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491**

**3.3V-Powered, 10Mbps and Slew-Rate-Limited
True RS-485/RS-422 Transceivers**

Absolute Maximum Ratings

Supply Voltage (VCC)	7V
Control Input Voltage (RE, DE)	-0.3V to 7V
Driver Input Voltage (DI).....	-0.3V to 7V
Driver Output Voltage (A, B, Y, Z)	-7.5V to 12.5V
Receiver Input Voltage (A, B).....	-7.5V to 12.5V
Receiver Output Voltage (RO)	-0.3V to (VCC + 0.3V)
Continuous Power Dissipation ($T_A = +70^\circ\text{C}$)	
8-Pin Plastic DIP (derate 9.09mW/°C above +70°C) ..	727mW
8-Pin SO (derate 5.88mW/°C above +70°C).....	471mW

14-Pin Plastic DIP (derate 10mW/°C above +70°C) ...	800mW
14-Pin SO (derate 8.33mW/°C above +70°C).....	667mW
Operating Temperature Ranges	
MAX34_C_	0°C to +70°C
MAX34_E_	-40°C to +85°C
Junction Temperature.....	+160°C
Storage Temperature Range	-65°C to +160°C
Lead Temperature (soldering, 10sec)	+300°C

Package Information

14 SOIC

PACKAGE CODE	S14+1
Outline Number	21-0041
Land Pattern Number	90-0112
Thermal Resistance, Single-Layer Board:	
Junction to Ambient (θ_{JA})	120
Junction to Case (θ_{JA})	37
Thermal Resistance, Four-Layer Board:	
Junction to Ambient (θ_{JA})	84
Junction to Case (θ_{JA})	34

For the latest package outline information and land patterns (footprints), go to [www.maximintegrated.com/packages](#). Note that a “+”, “#”, or “-” in the package code indicates RoHS status only. Package drawings may show a different suffix character, but the drawing pertains to the package regardless of RoHS status.

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](#).

MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491

3.3V-Powered, 10Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers

DC Electrical Characteristics

($V_{CC} = 3.3V \pm 0.3V$, $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted. Typical values are at $T_A = +25^\circ C$)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Differential Driver Output	V_{OD}	$R_L = 100\Omega$ (RS-422), Figure 4		2.0			V
		$R_L = 54\Omega$ (RS-485), Figure 4		1.5			
		$R_L = 60\Omega$ (RS-485), $V_{CC} = 3.3V$, Figure 5		1.5			
Change in Magnitude of Driver Differential Output Voltage for Complementary Output States (Note 1)	ΔV_{OD}	$R_L = 54\Omega$ or 100Ω , Figure 4			0.2		V
Driver Common-Mode Output Voltage	V_{OC}	$R_L = 54\Omega$ or 100Ω , Figure 4			3		V
Change in Magnitude of Common-Mode Output Voltage (Note 1)	ΔV_{OC}	$R_L = 54\Omega$ or 100Ω , Figure 4			0.2		V
Input High Voltage	V_{IH}	DE, DI, RE		2.0			V
Input Low Voltage	V_{IL}	DE, DI, RE			0.8		V
Logic Input Current	I_{IN1}	DE, DI, RE			± 2		μA
Input Current (A, B)	I_{IN2}	DE = 0V, $V_{CC} = 0V$ or 3.6V	$V_{IN} = 12V$	1.0			mA
			$V_{IN} = -7V$	-0.8			
Output Leakage (Y, Z)	I_O	DE = 0V, RE = 0V, $V_{CC} = 0V$ or 3.6V, MAX3491	$V_{OUT} = 12V$	20			μA
			$V_{OUT} = -7V$	-20			
Output Leakage (Y, Z) in Shutdown Mode	I_O	DE = 0V, RE = V_{CC} , $V_{CC} = 0V$ or 3.6V, MAX3491	$V_{OUT} = 12V$	1			μA
			$V_{OUT} = -7V$	-1			
Receiver Differential Threshold Voltage	V_{TH}	$-7V \leq V_{CM} \leq 12V$		-0.2	0.2		V
Receiver Input Hysteresis	ΔV_{TH}	$V_{CM} = 0V$			50		mV
Receiver Output High Voltage	V_{OH}	$I_{OUT} = -1.5mA$, $V_{ID} = 200mV$, Figure 6		$V_{CC} - 0.4$			V
Receiver Output Low Voltage	V_{OL}	$I_{OUT} = 2.5mA$, $V_{ID} = 200mV$, Figure 6			0.4		V
Three-State (High Impedance) Output Current at Receiver	I_{OZR}	$V_{CC} = 3.6V$, $0V \leq V_{OUT} \leq V_{CC}$			± 1		μA
Receiver Input Resistance	R_{IN}	$-7V \leq V_{CM} \leq 12V$		12			$k\Omega$
Supply Current	I_{CC}	No load, DI = 0V or V_{CC}	DE = V_{CC} , RE = 0V or V_{CC}	1.1	2.2		mA
			DE = 0V, RE = 0V	0.95	1.9		
Supply Current in Shutdown Mode	I_{SHDN}	DE = 0V, RE = V_{CC} , DI = V_{CC} or 0V		0.002	1		μA
Driver Short-Circuit Output Current	I_{OSD}	$V_{OUT} = -7V$			-250		mA
		$V_{OUT} = 12V$			250		
Receiver Short-Circuit Output Current	I_{OSR}	$0V \leq V_{RO} \leq V_{CC}$		± 8	± 60		mA

MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491

3.3V-Powered, 10Mbps and Slew-Rate-Limited
True RS-485/RS-422 Transceivers

Driver Switching Characteristics—MAX3485, MAX3490, and MAX3491

($V_{CC} = 3.3V$, $T_A = +25^\circ C$)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Differential Output Delay	t_{DD}	$R_L = 60\Omega$, Figure 7	1	22	35	ns
Driver Differential Output Transition Time	t_{TD}	$R_L = 60\Omega$, Figure 7	3	8	25	ns
Driver Propagation Delay, Low-to-High Level	t_{PLH}	$R_L = 27\Omega$, Figure 8	7	22	35	ns
Driver Propagation Delay, High-to-Low Level	t_{PHL}	$R_L = 27\Omega$, Figure 8	7	22	35	ns
$ t_{PLH} - t_{PHL} $ Driver Propagation Delay Skew (Note 2)	t_{PDS}	$R_L = 27\Omega$, Figure 8		8		ns
DRIVER OUTPUT ENABLE/DISABLE TIMES (MAX3485/MAX3491 only)						
Driver Output Enable Time to Low Level	t_{PZL}	$R_L = 110\Omega$, Figure 10	45	90		ns
Driver Output Enable Time to High Level	t_{PZH}	$R_L = 110\Omega$, Figure 9	45	90		ns
Driver Output Disable Time from High Level	t_{PHZ}	$R_L = 110\Omega$, Figure 9	40	80		ns
Driver Output Disable Time from Low Level	t_{PLZ}	$R_L = 110\Omega$, Figure 10	40	80		ns
Driver Output Enable Time from Shutdown to Low Level	t_{PSL}	$R_L = 110\Omega$, Figure 10	650	900		ns
Driver Output Enable Time from Shutdown to High Level	t_{PSH}	$R_L = 110\Omega$, Figure 9	650	900		ns

Driver Switching Characteristics—MAX3486

($V_{CC} = 3.3V$, $T_A = +25^\circ C$)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Differential Output Delay	t_{DD}	$R_L = 60\Omega$, Figure 7	24	48	70	ns
Driver Differential Output Transition Time	t_{TD}	$R_L = 60\Omega$, Figure 7	15	35	60	ns
Driver Propagation Delay, Low-to-High Level	t_{PLH}	$R_L = 27\Omega$, Figure 8	20	48	70	ns
Driver Propagation Delay, High-to-Low Level	t_{PHL}	$R_L = 27\Omega$, Figure 8	20	48	70	ns
$ t_{PLH} - t_{PHL} $ Driver Propagation Delay Skew (Note 2)	t_{PDS}	$R_L = 27\Omega$, Figure 8		11		ns
Driver Output Enable Time to Low Level	t_{PZL}	$R_L = 110\Omega$, Figure 10	55	100		ns
Driver Output Enable Time to High Level	t_{PZH}	$R_L = 110\Omega$, Figure 9	55	100		ns
Driver Output Disable Time from High Level	t_{PHZ}	$R_L = 110\Omega$, Figure 9	45	80		ns
Driver Output Disable Time from Low Level	t_{PLZ}	$R_L = 110\Omega$, Figure 10	45	80		ns
Driver Output Enable Time from Shutdown to Low Level	t_{PSL}	$R_L = 110\Omega$, Figure 10	700	1000		ns
Driver Output Enable Time from Shutdown to High Level	t_{PSH}	$R_L = 110\Omega$, Figure 9	700	1000		ns

MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491

3.3V-Powered, 10Mbps and Slew-Rate-Limited
True RS-485/RS-422 Transceivers

Driver Switching Characteristics—MAX3483 and MAX3488

($V_{CC} = 3.3V$, $T_A = +25^\circ C$)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Differential Output Delay	t_{DD}	$R_L = 60\Omega$, Figure 7	600	900	1400	ns
Driver Differential Output Transition Time	t_{TD}	$R_L = 60\Omega$, Figure 7	400	700	1200	ns
Driver Propagation Delay, Low-to-High Level	t_{PLH}	$R_L = 27\Omega$, Figure 8	700	1000	1500	ns
Driver Propagation Delay, High-to-Low Level	t_{PHL}	$R_L = 27\Omega$, Figure 8	700	1000	1500	ns
$ t_{PLH} - t_{PHL} $ Driver Propagation Delay Skew (Note 2)	t_{PDS}	$R_L = 27\Omega$, Figure 8		100		ns
DRIVER OUTPUT ENABLE/DISABLE TIMES (MAX3485/MAX3491 only)						
Driver Output Enable Time to Low Level	t_{PZL}	$R_L = 110\Omega$, Figure 10	900	1300		ns
Driver Output Enable Time to High Level	t_{PZH}	$R_L = 110\Omega$, Figure 9	600	800		ns
Driver Output Disable Time from High Level	t_{PHZ}	$R_L = 110\Omega$, Figure 9	50	80		ns
Driver Output Disable Time from Low Level	t_{PLZ}	$R_L = 110\Omega$, Figure 10	50	80		ns
Driver Output Enable Time from Shutdown to Low Level	t_{PSL}	$R_L = 110\Omega$, Figure 10	1.9	2.7		ns
Driver Output Enable Time from Shutdown to High Level	t_{PSH}	$R_L = 110\Omega$, Figure 9	2.2	3.0		ns

Receiver Switching Characteristics

($V_{CC} = 3.3V$, $T_A = +25^\circ C$)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Time to Shutdown	t_{SHDN}	MAX3483/MAX3485/MAX3486/MAX3491 only (Note 3)	80	190	300	ns
Receiver Propagation Delay, Low-to-High Level	t_{RPLH}	$V_{ID} = 0V$ to $3.0V$, $C_L = 15pF$, Figure 11	25	65	90	ns
		MAX3483/MAX3488	25	75	120	
Receiver Propagation Delay, High-to-Low Level	t_{RPHL}	$V_{ID} = 0V$ to $3.0V$, $C_L = 15pF$, Figure 11	25	65	90	ns
		MAX3483/MAX3488	25	75	120	
$ t_{PLH} - t_{PHL} $ Receiver Propagation Delay Skew	t_{RPDS}	$V_{ID} = 0V$ to $3.0V$, $C_L = 15pF$, Figure 11		10		ns
		MAX3483/MAX3488		20		
Receiver Output Enable Time to Low Level	t_{PRZL}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		25	50	ns
Receiver Output Enable Time to High Level	t_{PRZH}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		25	50	ns
Receiver Output Disable Time from High Level	t_{PRHZ}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		25	45	ns
Receiver Output Disable Time from Low Level	t_{PRLZ}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		25	45	ns
Receiver Output Enable Time from Shutdown to Low Level	t_{PRSL}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		720	1400	ns
Receiver Output Enable Time from Shutdown to High Level	t_{PRSH}	$C_L = 15pF$, Figure 12, MAX3483/MAX3485/MAX3486/MAX3491 only		720	1400	ns

Note 1: ΔV_{OD} and ΔV_{OC} are the changes in V_{OD} and V_{OC} , respectively, when the DI input changes state.

Note 2: Measured on $|t_{PLH}(Y) - t_{PHL}(Y)|$ and $|t_{PLH}(Z) - t_{PHL}(Z)|$.

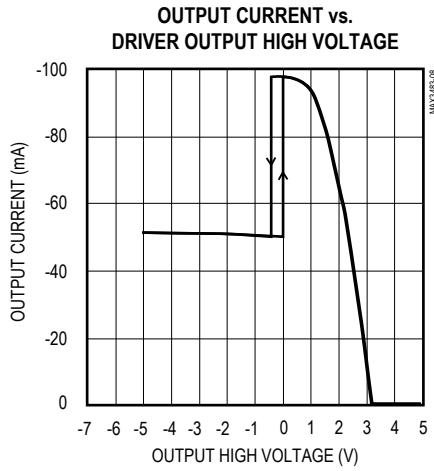
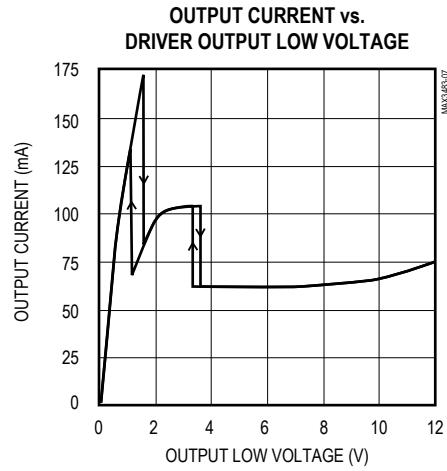
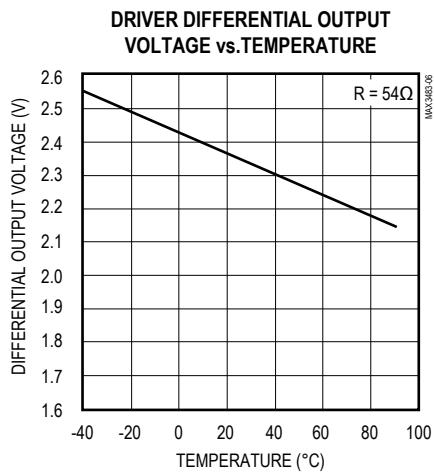
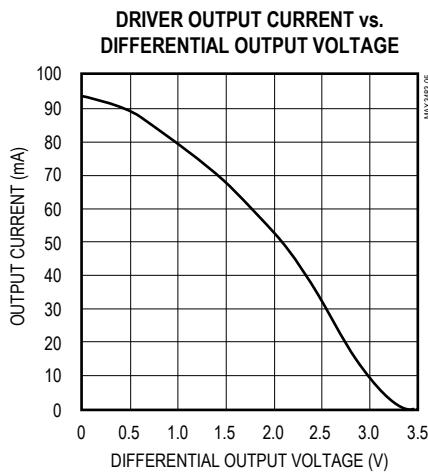
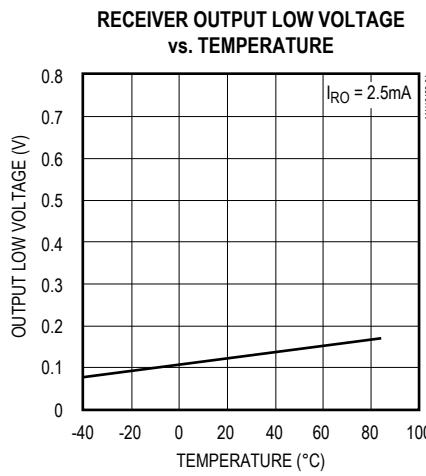
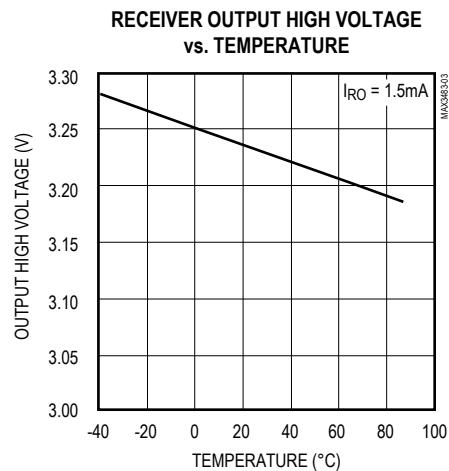
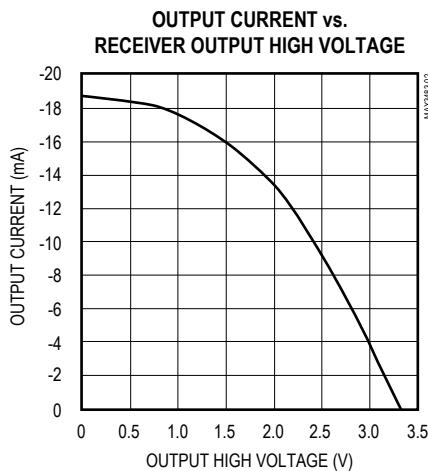
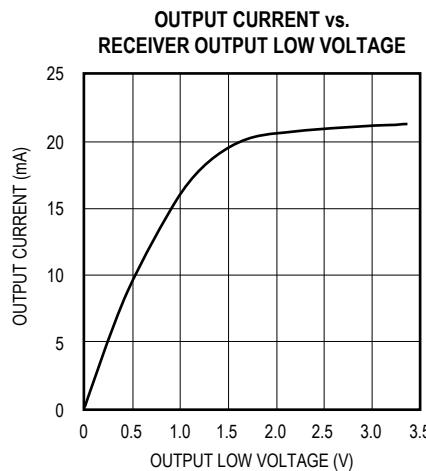
Note 3: The transceivers are put into shutdown by bringing RE high and DE low. If the inputs are in this state for less than 80ns, the parts are guaranteed not to enter shutdown. If the inputs are in this state for at least 300ns, the parts are guaranteed to have entered shutdown. See *Low-Power Shutdown Mode* section.

MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491

3.3V-Powered, 10Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers

Typical Operating Characteristics

($V_{CC} = 3.3V$, $T_A = +25^\circ C$, unless otherwise noted.)

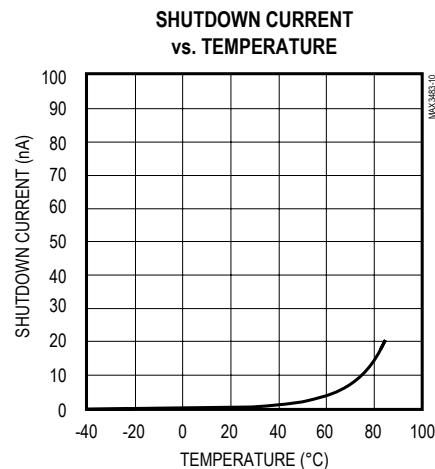
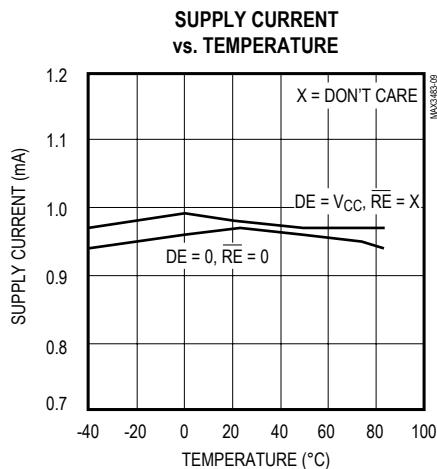


MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491

3.3V-Powered, 10Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers

Typical Operating Characteristics (continued)

($V_{CC} = 3.3V$, $T_A = +25^\circ C$, unless otherwise noted.)



Pin Description

PIN			NAME	FUNCTION
MAX3483/ MAX3485/ MAX3486	MAX3488/ MAX3490	MAX3491		
1	2	2	RO	Receiver Output. If A > B by 200mV, RO will be high; if A < B by 200mV, RO will be low.
2	—	3	\bar{RE}	Receiver Output Enable. RO is enabled when \bar{RE} is low; RO is high impedance when \bar{RE} is high. If \bar{RE} is high and DE is low, the device will enter a low-power shutdown mode.
3	—	4	DE	Driver Output Enable. The driver outputs are enabled by bringing DE high. They are high impedance when DE is low. If \bar{RE} is high and DE is low, the device will enter a low-power shutdown mode. If the driver outputs are enabled, the parts function as line drivers. While they are high impedance, they function as line receivers if \bar{RE} is low.
4	3	5	DI	Driver Input. A low on DI forces output Y low and output Z high. Similarly, a high on DI forces output Y high and output Z low.
5	4	6, 7	GND	Ground
—	5	9	Y	Noninverting Driver Output
—	6	10	Z	Inverting Driver Output
6	—	—	A	Noninverting Receiver Input and Noninverting Driver Output
—	8	12	A	Noninverting Receiver Input
7	—	—	B	Inverting Receiver Input and Inverting Driver Output
—	7	11	B	Inverting Receiver Input
8	1	13, 14	V_{CC}	Positive Supply: $3.0V \leq V_{CC} \leq 3.6V$
—	—	1, 8	N.C.	No Connect—not internally connected

**MAX3483/MAX3485/
MAX3486/MAX3488/
MAX3490/MAX3491**

**3.3V-Powered, 10Mbps and Slew-Rate-Limited
True RS-485/RS-422 Transceivers**

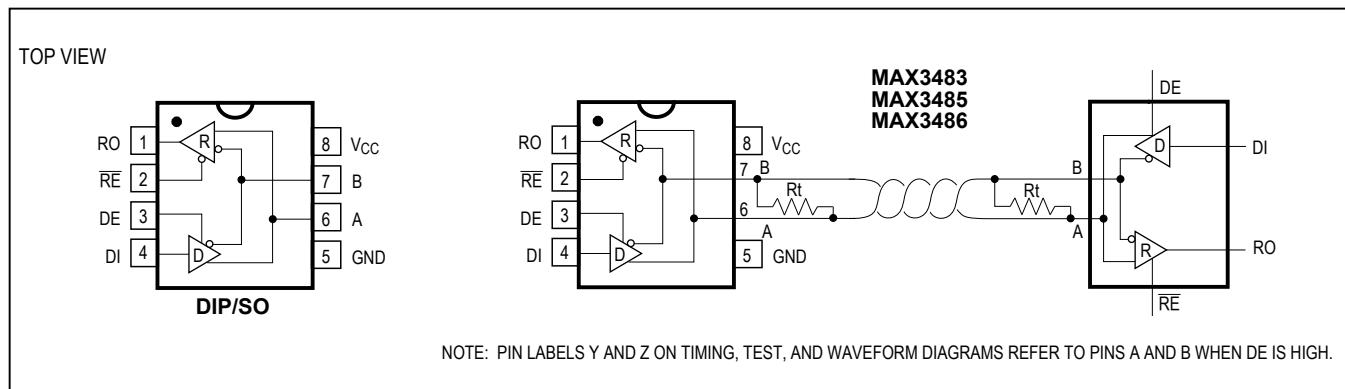


Figure 1. MAX3483/MAX3485/MAX3486 Pin Configuration and Typical Operating Circuit

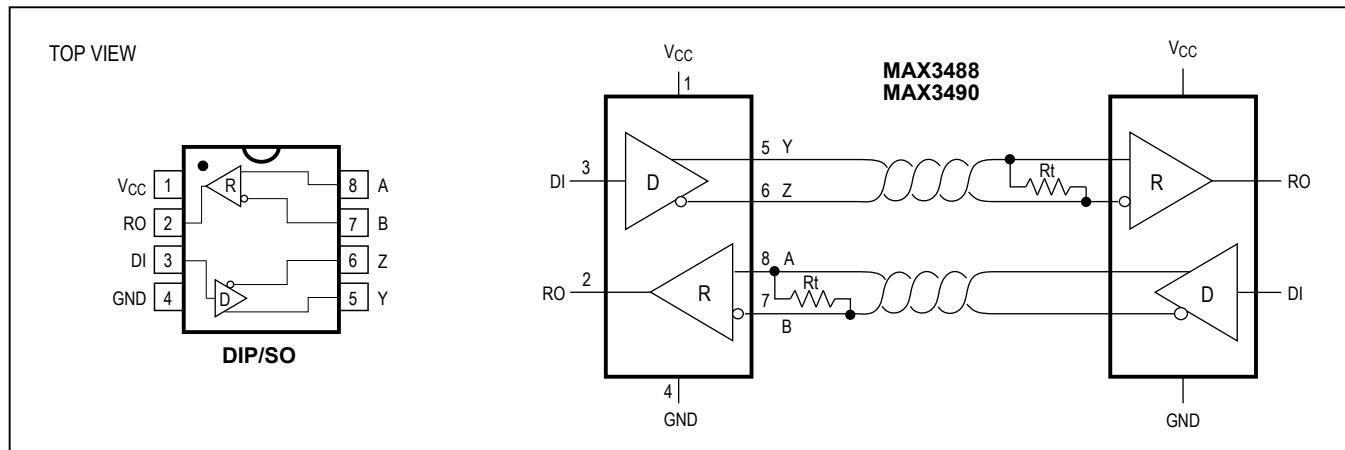


Figure 2. MAX3488/MAX3490 Pin Configuration and Typical Operating Circuit

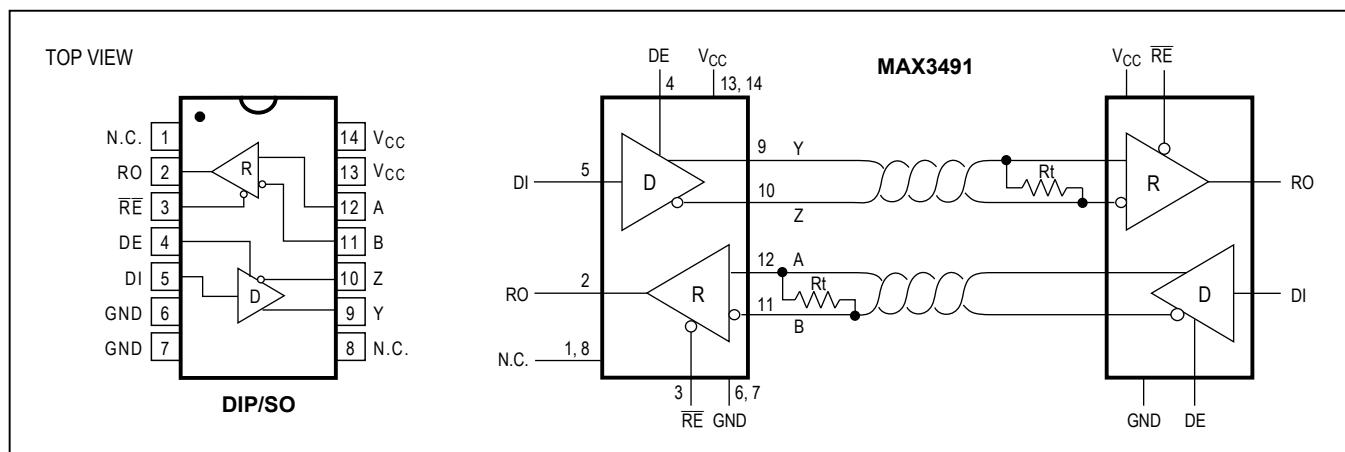


Figure 3. MAX3491 Pin Configuration and Typical Operating Circuit

REFERENCIAS

- [1] E. Systems. (2020) "fundamentos de esp-mesh". Consultado en diciembre 2020. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/mesh.html#mesh-introduction>
- [2] IEC. (2007) 313-01-35. Consultado en diciembre 2020. [Online]. Available: <http://www.electropedia.org/iev/iev.nsf/display?openform&ievref=313-01-35>
- [3] D. Londono, “Desarrollo de una guía enfocada a medidores de energía y conexiones de medidores,” 2013.
- [4] M. Ruiz, “Interoperabilidad entre medidores inteligentes de energía eléctrica residencial,” 2015, obtenido desde <https://dspace.ups.edu.ec/bitstream/123456789/8435/6/UPS-KT01062.pdf>, en agosto 2020.
- [5] M. Alvarado, “Servicios de medición avanzada (AMI) para redes inteligentes y su adaptabilidad en el marco de la legislación ecuatoriana,” 2011.
- [6] M. Bahr, “Update on the hybrid wireless mesh protocol of IEEE 802.11s,” *Siemens Corporate Technology, Information and Communications*, p. 1, 10 2016.
- [7] A. Vazquez, “Diseño e implementación de una red wifi mallada que soporte protocolo modbus para equipos de control industrial.” 2020, consultado en Octubre, 2020.
- [8] M. B. Rifki Muhendra, Aditya Rinaldi, “Development of WiFi mesh infrastructure for internet of things applications,” *Engineering Physics International Conference, EPIC 2016*, p. 331, 2016.