CONTROL REMOTO DEL ENCENDIDO Y APAGADO Y MONITORIZACIÓN DEL CONSUMO DE ENERGÍA ELÉCTRICA DE UN SISTEMA DE ILUMINACIÓN

RAILIN SANTIAGO TOLOZA VILLALOBOS
JULIÁN ANDRÉS PIÑERES RODRÍGUEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2017

CONTROL REMOTO DEL ENCENDIDO Y APAGADO Y MONITORIZACIÓN DEL CONSUMO DE ENERGÍA ELÉCTRICA DE UN SISTEMA DE ILUMINACIÓN.

RAILIN SANTIAGO TOLOZA VILLALOBOS
JULIÁN ANDRÉS PIÑERES RODRÍGUEZ

Trabajo de Grado para Optar el Título de
Ingeniero Electrónico

Director
JAIME GUILLERMO BARRERO PEREZ
Magister en Potencia eléctrica

Codirector
LUIS FERNANDO RUEDA VÁSQUEZ
Magister en Electrónica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2017

CONTENIDO

Pág.

LISTAS DE FIGURAS

Pag.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

TITULO: CONTROL REMOTO DEL ENCENDIDO Y APAGADO Y MONITORIZACION DEL CONSUMO DE ENERGIA ELECTRICA DE UN SISTEMA DE ILUMINACION*

AUTORES: JULIAN ANDRES PIÑERES RODRIGUEZ, RAILIN SANTIAGO TOLOZA VILLALOBOS**

PALABRAS CLAVE: control, medición, remoto, wifi, blynk, serial, halfduplex, protocolos, sistemas embebidos, comunicación, modbus.

DESCRIPCIÓN

Este trabajo de investigación propone la implementación de un sistema para el control del encendido y apagado y la monitorización de las variables eléctricas de una luminaria de forma remota, con ayuda de dispositivos electrónicos de actualidad con capacidad de conectarse a internet y una interfaz creada desde una aplicación, para el teléfono móvil.

El enfoque que tiene este proyecto, permite mediante un sistema compuesto por un hardware y un software, favorecer el uso eficiente y racional de energía a los usuarios del sistema eléctrico colombiano, de manera sencilla, económica y segura. Mejorando la interacción que tienen los usuarios con el sistema eléctrico en sus viviendas aplicado al sistema de iluminación, debido a que por descuido humano se puede desperdiciar energía al dejar la iluminación encendida cuando esta no se requiera. Para cumplir este objetivo, se usa un sistema embebido de bajo costo, dispositivos para la medición eléctrica con protocolo de comunicación industrial modbus, el cual permite la extracción de datos y el uso una plataforma para internet de las cosas para la gestión de los datos, la plataforma empleada, es una aplicación para el móvil que se puede descargar sin ningún costo; Además, se usa un sensor de proximidad para optimizar el sistema y el uso de un relé de estado sólido con el que se controla un circuito de corriente alterna con un circuito de corriente continua.

.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería Eléctica, Electrónica y de Telecomunicaciones. Director: Juan Guillermo Barrero Pérez, Magíster en Potencia Eléctrica. Codirector. Luis Fernando Rueda Vásquez, Magíster en Electrónica.

ABSTRACT

TITLE: REMOTE CONTROL OF POWER ON / OFF AND MONITORING THE ELECTRICAL ENERGY CONSUMPTION OF AN ILLUMINATION SYSTEM. *

AUTHORS: JULIAN ANDRES PIÑERES RODRIGUEZ, RAILIN SANTIAGO TOLOZA VILLALOBOS**

KEYWORDS: control, measurement, remote, wifi, blynk, serial, halfduplex, protocols, embedded systems, communication, modbus.

DESCRIPTION

This research project proposes the implementation of a system to control the switching on and off and the monitoring of the electrical variables of a luminaire remotely, with the help of current electronic devices with the ability to connect to the Internet and an interface created from a application, for the mobile phone.

The approach that this project has, allows through a system composed of hardware and software, to favor the efficient and rational use of energy to users of the Colombian electricity system, in a simple, economical and safe way. Improving the interaction that users have with the electrical system in their homes applied to the lighting system, because human carelessness can waste energy by leaving the lighting on when it is not required. To achieve this objective, an embedded low-cost system is used, devices for electrical measurement with modbus industrial communication protocol, which allows the extraction of data and use a platform for internet of things for data management, the platform used, is an application for the mobile that can be downloaded without any cost; In addition, a proximity sensor is used to optimize the system and the use of a solid state relay with which an alternating current circuit is controlled with a direct current circuit.

* Work degree.

** Faculty of Physical-Mechanical Engineering. School of Electrical Engineering, Electronics and Telecommunications. Director: Juan Guillermo Barrero Pérez, Master in Electrical Power. Co-director Luis Fernando Rueda Vásquez, Master in Electronics.

INTRODUCCION

Las políticas sobre uso racional y eficiente de la energía (URE), están orientadas a la disminución del consumo de la energía eléctrica, a propiciar el uso racional y eficiente de energía en los sectores de consumo y a la integración de Fuentes No Convencionales de Energía Renovable (FNCER). Reportes de la UPME (Unidad de Planeación Minero-Energética) muestran un crecimiento en la demanda de consumo eléctrico en todo el territorio nacional, razón por la que se está motivando al ahorro en nuestros hogares y a fomentar el uso de lámparas halógenas y de tecnología *LED*, que reducen el consumo de energía eléctrica.[1]

De la mano con el desarrollo tecnológico ha venido tomando fuerza un concepto llamado red inteligente (RI) que busca la integración de las Tecnologías de la Información y Comunicación (TIC'S) en la generación, distribución y consumo de un sistema eléctrico con el fin de obtener fiabilidad, calidad y eficiencia del servicio. Una RI necesita una infraestructura de medida junto con una red de comunicación adecuada que le proporcione la información necesaria para la toma de decisiones y los medios adecuados para el envío y recepción de los datos usados para el control y monitoreo de estos sistemas. En las Instituciones, universidades y centros de investigación se han emprendido proyectos innovadores en el sector de RI. Cabe destacar que empresas que brindan el servicio de energía eléctrica centran sus esfuerzos en la medición inteligente mediante dispositivos que permiten el envío de la información de forma bidireccional entre el usuario y la empresa prestadora del servicio. Estos dispositivos deberían incluir entre otras cosas información sobre precios, instrucciones de conexión o desconexión, alarmas e instrucciones cuando exista un malgasto de energía, actualizaciones de *software* de fecha y hora, así mismo,

[1] UPME. Demanda y eficiencia energética [en línea]. Citado en 4 de marzo de 2016. Disponible en: http://www1.upme.gov.co/Paginas/Demanda-y-Eficiencia-Energetica.aspx.

le permita al usuario la posibilidad de recibir información sobre el gasto energético de su hogar de forma remota. Esto le permitirá realizar cambios en sus modos de consumo así también, le permitirá tener control de todos los aparatos que conecta a la red eléctrica.

Actualmente las empresas prestadoras del servicio de energía eléctrica monitorean el consumo de todas las residencias mediante un esquema de lecturas periódicas del medidor que se encuentra ubicado en el exterior de inmueble. Los usuarios por su parte reciben una factura mensual donde se le informa de manera general la cantidad de energía consumida medida en KWh/mes. Por su parte el usuario no tiene información detallada de las cargas conectadas (bombillas, electrodomésticos, etc.) dificultando el control y monitoreo de esta energía por parte del usuario. En este proyecto se presenta una propuesta con la información detallada sobre la construcción de un sistema para el control y la medición de las variables eléctricas en una luminaria de forma local y remota mediante dispositivos electrónicos y plataformas TIC. Cabe enunciar que se utilizó una plataforma de desarrollo para internet de las cosas (*Internet Of Things* - IOT) y un medidor de energía eléctrica marca EASTRON con protocolo Modbus y comunicación RS485. El foco principal de este proyecto es desarrollar herramientas de fácil desarrollo y bajo costo que le permitan al usuario, cambiar hábitos de consumo en el usuario y fomente el URE, mediante su interacción con el sistema eléctrico de su inmueble.

1. OBJETIVOS

1.1 OBJETIVO GENERAL

Implementar un sistema que permita controlar de forma local/remota la iluminación de un recinto y monitorear su consumo de energía eléctrica.

1.2 OBJETIVOS ESPECÍFICOS

- Controlar el encendido y apagado de una luminaria de manera local, a partir de un sensor de detección de presencia.

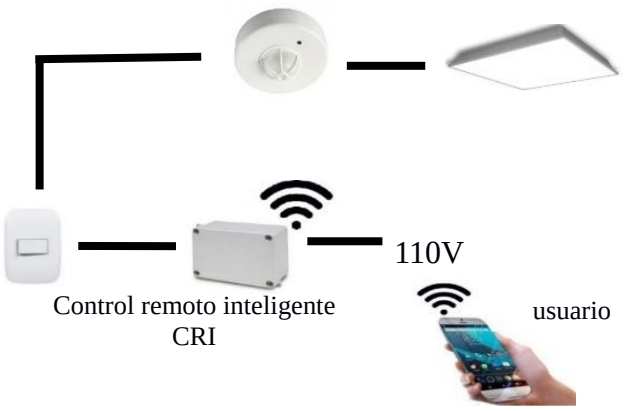
- Desarrollar una interfaz que permita realizar el encendido y apagado de una bombilla a través de un dispositivo móvil conectado a internet.

- Establecer una comunicación serial, entre el medidor de energía eléctrica y una tarjeta de desarrollo, exportar los datos del medidor y visualizarlos en una aplicación móvil.

2. PREAMBULO

El esquema tradicional de una instalación eléctrica residencial, ofrece herramientas para el control de las cargas mediante interruptores o algún sensor y contadores de energía para el monitoreo del consumo eléctrico, pero son poco eficientes a la hora de promover el ahorro y buen uso del recurso eléctrico. La propuesta que se planteó, fue pensada en fortalecer el sistema convencional mediante un sistema inteligente, que le permita al cliente mejorar sus hábitos de consumo de energía eléctrica, de manera cómoda mediante su teléfono móvil y sin tener que hacer mayores modificaciones de la estructura física del lugar. Esta autonomía con mínima intervención humana, es las que se busca adaptar en las construcciones sostenibles.[2].

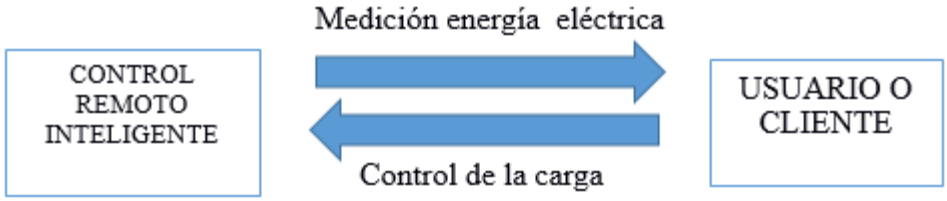
Figura 1. Control y medición remota



La figura 1, muestra un sistema para controlar una lampara de tecnología LED por medio de un dispositivo de CRI, accionado por un teléfono móvil, que va a trabajar articuladamente con un sensor de movimiento y el interruptor manual para el control local de este sistema.

[2] REPÚBLICA DE COLOMBIA. Ministerio de Vivienda. Construcción sostenible. [en línea]. Disponible en: http://www.minvivienda.gov.co/cambio-climatico/mitigacion/construccion-sostenible

Figura 2. Interacción entre el usuario y el control remoto inteligente.



La comunicación entre el control remoto inteligente y el usuario se evidencia en la Figura 2, donde se evidencia el flujo de datos que se realiza del sistema remoto, que envia el valor de la energía eléctrica, proveniente de un monitoreo periodico a la carga, esta informacion viaja por un canal de comunicación *wifi* hacia el usuario que recibe esta informacion en su movil; El usuario por su parte, envia de forma asincrona, una señal desde su celular que viaja por el mismo canal *wifi*, y ejerce una acción sobre el control remoto inteligente para la conmutación de la carga.

La metodología que se abordó para desarrollar el proyecto tiene dos pilares importantes, el control y la medición, definidos en los mismos objetivos. La primera tarea fue diseñar un esquema del sistema a partir de diagramas de bloques; una vez identificada cada una de las partes, se gestionó la compra de los dispositivos principales que componen la implementación, bajo los parámetros de utilidad, costo y tamaño. El diseño del sistema es un modelo para un circuito monofásico en serie, su modo de funcionamiento hace que el control local y el remoto sean dependientes entre sí, de esta manera, se tiene un interruptor manual en estado de encendido en serie con un conmutador inteligente, accionado por *wifi* y un sensor, que cerrará el circuito y conectará la carga cuando detecte presencia en el lugar. En la Figura 3 se encuentra más ampliado el sistema implementado en serie.

Figura 3. Esquema Serie.

El sensor empleado está diseñado para montaje en techo, su funcionamiento se basa en un sensor infrarrojo pasivo o PIR, que recibe los rayos infrarrojos ocasionados del movimiento humano; soporta una carga nominal de una lampara incandescente o una lampara ahorradora de energía; detecta movimiento a una distancia de máximo 6m, esta distancia es la resultante entre una línea perpendicular al medidor y un ángulo de apertura de 24 grados. El sensor está diseñado para tener un rango de detección de aproximadamente 360 grados, luz ambiente ajustable entre los 3-2000 LUX para una funcionalidad de día y noche; cuenta con una perrilla externa, para retardar el tiempo de funcionamiento mínimo 7 segundos o máximo 7 minutos. Este sensor es de los más económicos que existen en el mercado para esta tecnología de detección a distancia.[3].

El sistema requiere que de un *Smartphone* o celular inteligente que pueda conectarse a internet, además de un sistema operativo con soporte para

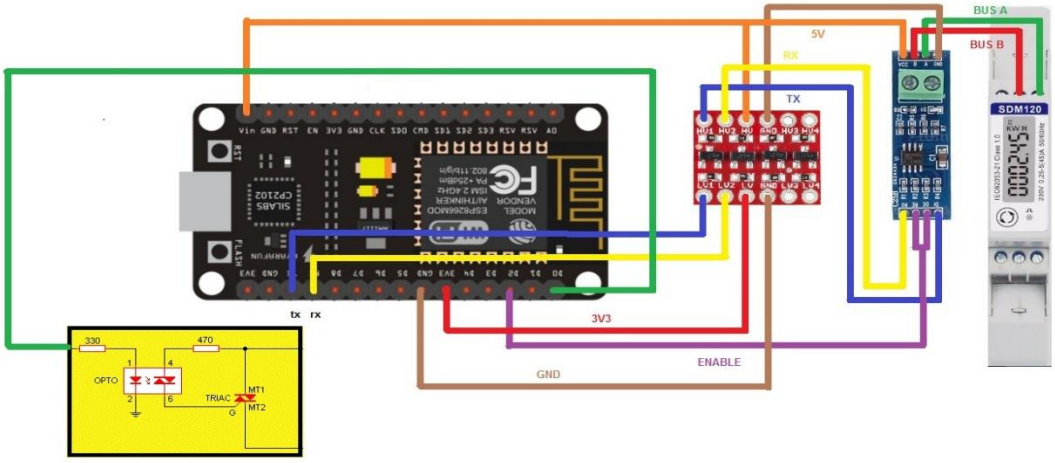
[3] ALIXPRESS. Automático Interruptor infrarrojo PIR Motion Sensor de Luz LED de Techo 037. [en línea]. Disponible en: https://es.aliexpress.com/store/product/Free-shipping-High-Sensitive-360degree-Motion-Sensor-220-240V-AC-Motion-Sensor-Automatic-Light-Switch-ET037/817842_829256468.html?spm=a219c.search0104.3.17.mFXkpA&ws_ab_test=searchweb0_0,searchweb201602_1_10152_1006.

aplicaciones de Android. Este dispositivo va a poder conectarse con un servidor, que almacena los datos enviados por la tarjeta de desarrollo. Como carga se usó de una Lampara LED en forma de panel de 24W, temperatura de color 6500K.

3. HARDWARE CONTROL REMOTO INTELIGENTE

El circuito mostrado en la Figura 4, corresponde al *hardware* del control remoto inteligente, que requiere la utilización de un medidor de energía monofásico para realizar un monitoreo periódico de la carga. Este tiene la funcionalidad de exportar datos de la medición por un puerto RS485 y protocolo *Modbus*, que establece una conexión *half dúplex* entre el medidor y una tarjeta de desarrollo con conectividad *wifi*. Esta última, recibe la información por el puerto serial UART, procesa los datos y mediante un enlazamiento con un *router* local, carga estos datos en el servidor de la aplicación móvil, para llegar finalmente al usuario. La aplicación por su parte, le permite al usuario, enviar una señal a la tarjeta de desarrollo para que active o desactive uno de los puertos de propósito general, para controlar el circuito que hace la conmutación de la carga. En el proceso La tensión de salida de estos puertos es de 3.3V, un valor suficiente para excitar el diodo emisor de luz de un relé de estado sólido que conectará la carga con la red.

Figura 4. Esquema del control remoto inteligente



En el diagrama, se muestra un convertidor de nivel, empleado para acondicionar el voltaje en la transmisión de 3.3V a 5V, requerido por el MAX485 y proteger el puerto en la recepción. También se aprecia que se usan los puertos Tx0 y Rx0 de la tarjeta para la implementación del protocolo *Modbus*, por lo que no va a ser posible que la tarjeta se comunique con el computador o se empleen estos puertos para otra tarea, mientras se encuentra haciendo esta función.

Se usó un cargador de celular conectado al mismo sistema para energizar la tarjeta, que cuenta con un pin de 5V para alimentar los conversores utilizados.

3.1 RELÉ DE ESTADO SOLIDO (SSR)

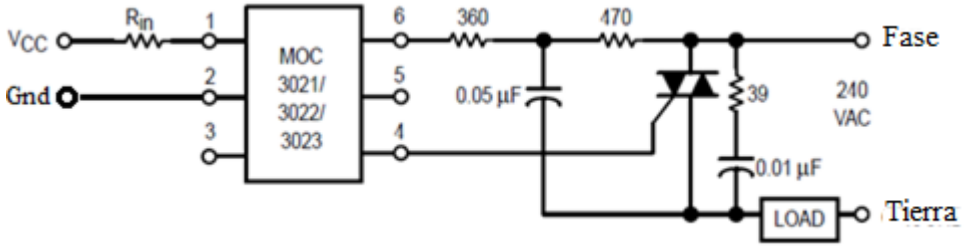
Un relé es un dispositivo electromecánico que funciona como un interruptor controlado por un circuito eléctrico; Este dispositivo consiste en una bobina, un terminal común, un terminal de normalmente cerrado y un terminal normalmente abierto. Cuando se energiza la bobina, el terminal común y el terminal normalmente abierto se unirán.

Existe un dispositivo electrónico creado a partir de componentes de estado sólido que emula el comportamiento de un relé electromecánico, y es a lo que hoy día se le conoce como relé de estado sólido o por sus siglas en inglés SSR *solid state relay*. Un relé de estado sólido (SSR), es un dispositivo usado para el control de cargas de potencia a partir de señales de control de bajo voltaje e intensidad de corriente; estos SSR tienen más ventajas con respecto a los relés electromecánicos, son más livianos, silenciosos, más rápidos, no sufren desgaste mecánico, son inmunes a los choques y vibraciones, generan muy pocas interferencias, conmutan altas corrientes y voltajes sin producir arcos, proporcionan varios kilovoltios de aislamiento entre la entrada y la salida. Una desventaja son dispositivos de una sola posición. Esto significa que un solo SSR

no puede conmutar al mismo tiempo varias cargas independientes como lo hacen los relés electromecánicos. (Scneider Electric, s.f.)

En la **¡Error! No se encuentra el origen de la referencia.**, se muestra el squema de un SSR a partir de un MOC3021, un *triac* y algunos elementos pasivos como resistencias y capacitores.

Figura 5. Circuito de aplicación típica

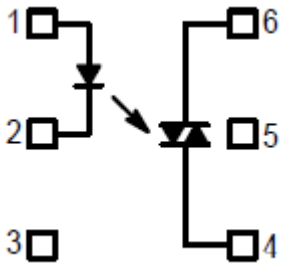


Fuente: Datasheet del Moc 3021

Finalmente, se va a implementar un circuito SSR, ya que elementos que lo componen son asequibles fácilmente en el mercado. En este circuito, se conmuta el lado de la fase y la carga queda conectada a tierra. La resistencia de 39 ohmios y el condensador de 0.01 uF constituyen la red de *snubber* para del *triac*, y la resistencia de 470 ohmios y el condensador de 0.05 uF son la red de *snubber* para acoplar con la carga. La red de *snubber* suprime los picos de tensión causados por la conmutación, pero puede o no ser necesaria, dependiendo del *triac* y la carga utilizada.

El Moc 3021 es un optocoplador, es la parte principal del circuito de disparo, se compone de un diodo led emisor de luz que se activa con una señal de voltaje de 3,3 V y en su salida por un fototransistor el cual recibe la luz emitida por el diodo led y activa o desactiva la compuerta del *triac* a emplear. El esquema de conexión del Moc 3021 se muestra en la figura, este modelo de optocoplador trabaja con una corriente de disparo máxima de 15 mA.

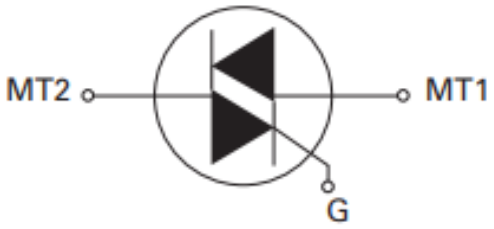
Figura 6. Diagrama de pines Moc 3021



Fuente: Datasheet del Moc 3021

La Figura 7, corresponde al Q6025L5, un *triac* alternistor bidireccional de 25 A, diseñado para conmutar cargas en AC, empleado para aplicaciones que requieren control de fase, tiene una capacidad para soportar un voltaje de subida de hasta 600V, no requiere red de *snubber.*
(Alldatasheet, s.f.)

Figura 7. Esquema del triac



Fuente: Datasheet Q6025L

3.2 MEDIDOR EASTRON SMD120C

En la Figura 8. se muestra el SDM120C, un analizador de energía monofásica bidireccional con LCD, diseñado para montaje en riel, grado de protección IP51, Soporte de carga hasta de 45 A, su consumo de energía es menor a 2W/10VA. El medidor puede estar provisto de un puerto de salida RS485 para la comunicación

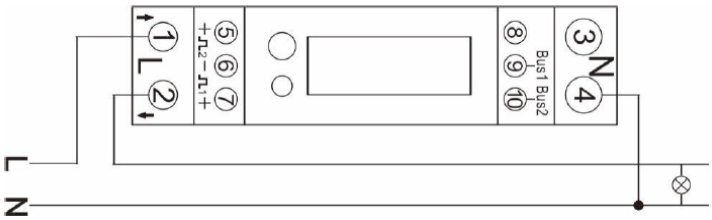
a distancia (J. Eastron Electronic Instruments, 2016). En la **¡Error! No se ncuentra el origen de la referencia.** del ANEXO A, se puede observar las especificaciones eléctricas del medidor al igual que los valores de exactitud para cada variable eléctrica medible.

Figura 8. Medidor EASTRON SDM120

Fuente: Datasheet del medidor

La medición de potencia se encuentra respaldada por la Comisión electrotécnica internacional (IEC) la cual es una organización mundial de normalización y IEC62053-21 hace referencia a equipos de medición. norma para contadores estáticos de energía activa destinados a uso residencial, comercial o de industria ligera Clase B EN50470-3.

Figura 9. Diagrama de conexión eléctrica del medidor.



Fuente: Datasheet del medidor

El Diagrama mostrado en la Figura 9, corresponde a las conexiones físicas de la medición de energía eléctrica en una carga, en el pin 1 y 2 para conexión de la

fase, 4 para conexión del neutro, por último, los pines 9 y 10 corresponden a los buses del protocolo RS485.

El medidor Eastron SDM120 soporta la función de comunicación *Modbus*, su protocolo estándar es EN 13757-3-2004, esta Norma Europea cubre los parámetros físicos y de capa de enlace de la comunicación en banda base sobre par trenzado para sistemas de comunicación y lectura remota de medidores. Las características para la comunicación serial del medidor son configurables y pueden variar de la siguiente manera:

Velocidad de transmisión: 300, 600, 1200, 2400 (predeterminado), 4800, 9600bps.

Paridad: Ninguno / par / impar Valor predeterminado: even.

Bit de parada: 1.

Dirección principal: 1 a 250.

En la tabla 1, se encuentran las direcciones de los registros que contienen la información de las variables eléctricas a monitorizar; Estas direcciones corresponden a los registros 3X de la memoria del medidor de energía. La representación de los parámetros que se van a consultar, se programa en el código en formato hexadecimal.

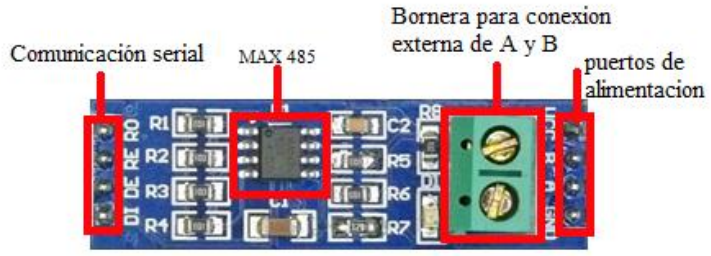
Tabla 1. Dirección de los registros de entrada

Dirección de los registros	Parámetro eléctrico	Unidades	Longitud (Bytes)	Protocolo Modbus	
				Hi	Lo
30001	Voltaje	V	4	00	00
30007	Corriente	A	4	00	06
30013	Potencia activa	W	4	00	0C
30019	Potencia aparente	VA	4	00	12
30031	Factor de potencia	ninguna	4	00	1E
30071	Frecuencia	Hz	4	00	46
30075	Importar energía activa	KWh	4	00	48

3.3 CONVERSOR RS485-TTL

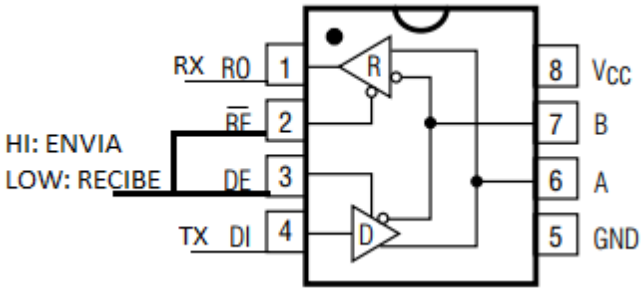
Es un módulo que convierte las señales TTL a RS-485 y viceversa; su componente principal es el chip MAX485, un transceptor que consume 830mW y exige una corriente de alimentación de 120 a 150uA en operación; El nivel de voltaje de las señales del bus de datos A y B oscila entre los +12V y los -7V [*datasheet* MAX485]. En la figura 10, se muestra que este dispositivo cuenta con pines externos para inserción en *protoboard* y señalizados para facilitar su conexión física.

Figura 10. Convertidor RS485-UART



El MAX 485 utiliza el protocolo de transmisión *half dúplex* para el envío de datos a una velocidad máxima de 2.5Mbps, este protocolo permite la conexión de hasta 32 dispositivos no consecutivos. En la Figura 11 se encuentra el esquema de conexión del MAX485 el cual maneja tres estados, transmisión (TX), recepción (RX) y una señal de habilitación cuya función es evitar la colisión en el canal de comunicación.

Figura 11. Esquema MAX485



Fuente: Datasheet MAX485

3.4 TARJETA DE DESARROLLO WIFI

Actualmente, en el mercado se encuentran tarjetas de desarrollo como la *Raspberry Pi*, la *Beagle Board*, la *Intel Galileo*, la *Intel Edison*, el omega plus y la NodeMCU entre otras, las cuales proporcionan la posibilidad de utilizar diferentes protocolos de comunicación como *Wi-Fi*, y *Bluetooth*, además de otros periféricos. En la Tabla 2, podemos ver una comparativa entre algunas tarjetas de desarrollo.

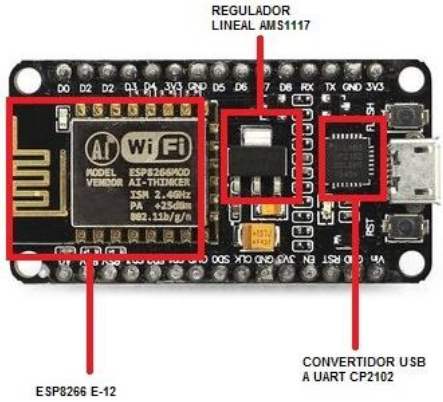
Tabla 2. Comparación entre tarjetas de desarrollo

	NodeMCU	Arduino Uno	Raspberry
Precio (USD)	5	16	43
Puertos GPIO	12	14	26
Puertos comunicación Serial (Tx, Rx)	1.5	1	2
Velocidad de procesamiento	80Mhz	20Mhz	700Mhz
Conectividad wifi	SI	NO	SI
Conexión bluetooth	NO	NO	SI

Para la selección de la tarjeta se tuvieron en cuenta dos factores importantes relacionados con la función que esta debe desempeñar. Se requiere puerto serial, para la comunicación *half dúplex* que no requiere una cantidad de puertos GPIO mayor a 5. La NodeMCU es la opción más viable teniendo en cuenta que es 8 veces más económica que la *Raspberry*.

La NodeMCU es una plataforma de código abierto; desarrollada para aplicaciones de internet de las cosas IoT, su *hardware* (Ver Figura 12) se encuentra formado por el módulo SoC ESP8266 E-12 y un *firmware* basado en Elua. Esta placa permite ser programada con un PC, ya que cuenta con el integrado CP2102 de Silicon Labs, que permite la conversión Serial a USB y un regulador lineal de 5V a 3.3V para proteger el módulo.

Figura 12. Hardware de la NodeMCU



Fuente: i2.linio.

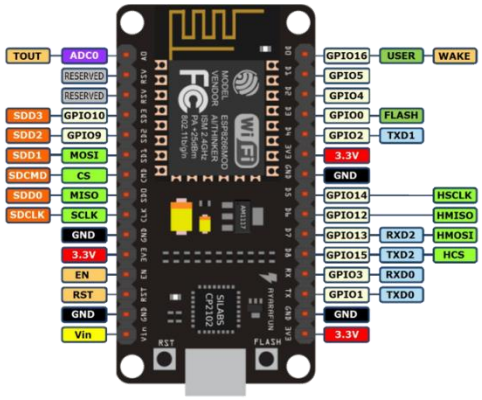
La NodeMCU ofrece al usuario todos los recursos integrados en la ESP8266-E12 tales como: un microcontrolador de 32 bits de extra bajo consumo de potencia y RSIC de 16 bits, con un reloj de la CPU de 80MHz, SRAM y ROM; se permite acceder a la memoria a través de interfaces iBus, dBus y AHB. el espacio SRAM que está disponible para los usuarios es alrededor de los 36kB; como la memoria ROM no es programable en el SoC, el programa del usuario debe ser almacenado en una memoria flash SPI externa con capacidad de 4MB; además cuenta con interfaz SPI / SDIO o I2C / UART y GPIO´S.

La NodeMCU y su familia de ESP son uno de los chips de *wifi* más completos de la industria; Integra los interruptores de antena, balum RF, Amplificador de potencia, amplificadores de recepción de bajo ruido, filtros, módulos de gestión de

energía, una antena 3DBi PCB-on-board que reduce circuitería externa. El módulo admite el estándar IEEE802.11 b/g/ n, pila de protocolo TCP/IP lo que permite la conexión a una red wifi. (Shenzhen Anxinke Technology CO;LTD)

Esta tarjeta permite ser programada mediante comandos AT; Si se quiere programar en lenguaje lua, la tarjeta debe ser flasheada con su *firmware*. Actualmente podemos encontrar un *core* de ARDUINO para programar la tarjeta en lenguaje C. (Vowstar, s.f.).

Figura 13. Esquema de pines NodeMcu



Fuente: IOTBYTES. NodeMCU ESP12 Dev Kit V1.0 Pin

En la Figura 13 se muestra el diagrama de pines de la tarjeta NodeMCU AMICA, algunos los pines cumplen más de una función; la tarjeta cuenta con un par serial TXD0, RXD0 y uno para transmisión TXD1, TXD2 y RXD2 son los pines para el control en el flujo de datos del serial0. Vin es una salida de 5V provenientes del puerto USB, este esquema nos orienta mejor acerca de las conexiones físicas de la tarjeta.

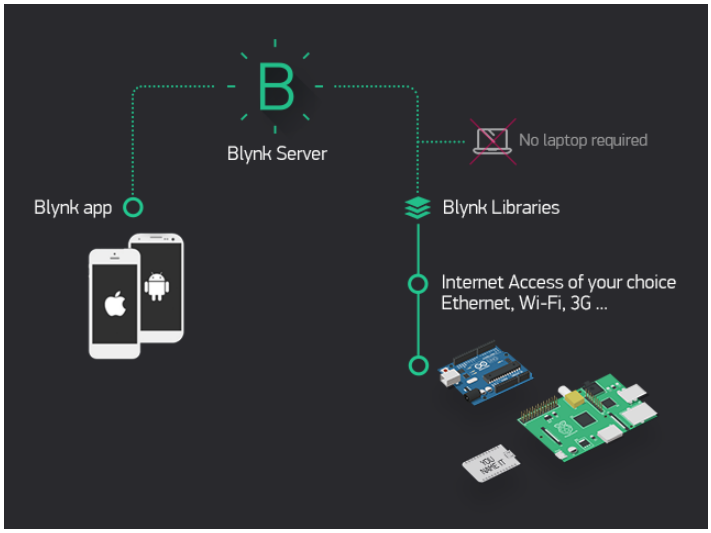
4. SOFTWARE CONTROL REMOTO INTELIGENTE

Existen diversas alternativas a la hora de usar una plataforma de desarrollo para la recepción de datos del medidor y el control del sistema, se puede optar por crear aplicaciones móviles nativas para Android, mediante diferentes plataformas para desarrollar *software* como Android studio, Basic 4 Android, App Inventor, HTML5, o usar aplicaciones de código abierto de la *playstore* para IoT como Cayenne, Node-RED, Eclipse Iot, OpenHUB, RIOT, Brillo, etcétera que permitirán interactuar con el *hardware* que tengas disponible.[4].

Para desarrollar el entorno de desarrollo, se escogió la aplicación Blynk, una Plataforma de iOS y aplicaciones Android, para el control de tarjetas con capacidad de conectarse a internet mediante, wifi, ethernet o el chip ESP8266. La aplicación maneja toda la autenticación y la comunicación a través de su servidor, que se ejecuta en Java y es de código abierto. Una vez instalado, se construye un interfaz gráfico, usando *widgets* o iconos propios, que se ajustan a las necesidades de cada sistema a construir. El servidor de la aplicación es responsable de la comunicación y el almacenamiento de datos entre el *Smartphone* y el *hardware*. La figura 14, nos da una idea más clara de cómo interactúa la aplicación móvil con el *hardware*, la aplicación cuenta con una librería desarrollada para Arduino.

[4] BBVAOPEN4U. (25 de Agosto de 2015). El Internet de las Cosas de código abierto: plataformas y aplicaciones para desarrolladores. [en línea]. Disponible en: BBVA API_MARKET: https://bbvaopen4u.com/es/actualidad/el-internet-de-las-cosas-de-codigo-abierto-plataformas-y-aplicaciones-para

Figura 14. Comunicación entre el móvil y la tarjeta



Fuente: Kickstarter. Aboxfullofwidgets (kickstarter, 2015)

La NodeMCU fue diseñada principalmente para trabajar con el lenguaje Lua, aunque también se pueden usar otros lenguajes de programación como Micropython o C. Las plataformas Micropython y Lua, son útiles, aunque trabajan con lenguajes interpretados. Conociendo la compatibilidad de la tarjeta con los comandos C, se desarrolló el código en el entorno de desarrollo integrado (IDE) de Arduino; Su lenguaje de programación propio similar a C++, La ventaja de este, es que presta soporte en tarjetas de terceros y se encuentran librerías y ejemplos de códigos por toda la internet. La tarjeta puede programarse siempre y cuando se le instale *firmware* para soporte de *software*. En el ANEXO C, se encuentra la información sobre el proceso de instalación del Firmware.

El código compilado es una adaptación de una librería para Arduino, que permite la comunicación de la NodeMCU en modo maestro con un medidor de energía modelo EASTRON SDM12O como esclavo[5]. La gestión de datos del medidor se realiza mediante la utilización del código de función para la lectura de sus registros

[5] GITHUB. Peninquen - Modbus-Energy-Monitor-Arduino. Disponible en: Modbus: https://github.com/peninquen/Modbus-Energy-Monitor-Arduino

de memoria, del protocolo *Modbus RTU*, y empleando un canal de comunicación RS485 *half-duplex*; La conexión al servidor de la aplicación se realiza con comandos de librerías para Arduino, para habilitar la comunicación.

La librería se compone de un archivo de cabecera llamado *ModbusSensor.h*, con las definiciones de las clases, el archivo *ModbusSensor.cpp*, donde están los métodos y las rutinas del código, y el archivo principal o *main* nombrado *codigo_node.ino*.

En el archivo principal empieza definiendo la librería *ESP8266WiFi.h* para programar la tarjeta con el IDE de arduino y las librerías para establecer la comunicación de la tarjeta con el servidor de la aplicación, mediante la librería de nombre *BlynkSimpleEsp8266.h* nombre y contraseña de la red a la que se va a conectar la tarjeta, el carácter *auth* corresponde a la clave de autorización del proyecto creado, ver ANEXO B.

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

char auth[] = "7b851565100c449187af92390e5b335e";
char ssid[] = "Redmi";
char pass[] = "31144269";
//char ssid[] = "JESUSGOMEZ";
//char pass[] = "29282930";
```

Posteriormente, se escriben los comandos para la inicialización y escritura en el puerto serial de la NodeMCU, la velocidad para la transmisión y recepción de datos del protocolo *Modbus* se establece a 2400 baudios, que es la tasa con la que se leyeron los registros del medidor y la que trae por defecto. Configuraciones para el formato de los bytes trasmitido de 8 bits, sin paridad con dos bit de parada, la constante TxEnablePin, define el número del puerto de propósito general

GPIO04, empleado para controlar la comunicación *halfduplex.* El valor del *TIMEOUT* fue tomado a consideración del desarrollador del código.

```
#define SERIAL_OUTPUT 1
#if SERIAL_OUTPUT
#   define SERIAL_BEGIN(...) Serial1.begin(__VA_ARGS__)
#   define SERIAL_PRINT(...) Serial1.print(__VA_ARGS__)
#   define SERIAL_PRINTLN(...) Serial1.println(__VA_ARGS__)
#else
#   define SERIAL_BEGIN(...)
#   define SERIAL_PRINT(...)
#   define SERIAL_PRINTLN(...)
#endif


#define MB_SERIAL_PORT &Serial
#define MB_BAUDRATE       2400          // b 2400
#define MB_BYTEFORMAT     SERIAL_8N2     // Prty n
#define TxEnablePin        4
#define TIMEOUT           100
```

La clase *modbusmaster* instancia únicamente al objeto *MBSerial* que se encarga de preparar el puerto *serial* de la NodeMCU y gestionar la lista de objetos de la clase *modbusSensor* como se ve en la figura 17.

```
modbusMaster MBserial(MB_SERIAL_PORT, TxEnablePin);  // instance to collect data using Modbus protocol over RS485

//variables to poll, process and send values
modbusSensor volt(&MBserial, ID_1, VOL_ADR, CHANGE_TO_ZERO);
modbusSensor curr(&MBserial, ID_1, CUR_ADR, CHANGE_TO_ZERO);
modbusSensor pwr(&MBserial, ID_1, POW_ADR, CHANGE_TO_ZERO);
modbusSensor enrg(&MBserial, ID_1, PEN_ADR, HOLD_VALUE);
modbusSensor freq(&MBserial, ID_1, FRE_ADR, CHANGE_TO_ZERO);
modbusSensor aPwr(&MBserial, ID_1, APO_ADR, CHANGE_TO_ZERO);
modbusSensor pwrFact(&MBserial, ID_1, PFA_ADR, CHANGE_TO_ONE);
```

En el bucle *setup*, se inicializan las variables que solo se llaman una vez, así como el puerto serie que se habilita cuando la tarjeta se encuentra polarizada, y la sentencia de Blynk para abrir la comunicación con su servidor. Dentro de este bucle, se define el puerto GPIO05 al que se le asigna el nombre de puertoControl,

encargado de cambiar la lectura de la variable medida. La función *Widget LDC* se inicia en este ciclo, y se le asigna la denominación V1, correspondiente a un puerto virtual.

```
unsigned long previousMillis = 0;
unsigned long currentMillis = 0;
boolean       firstData;
int puertoControl = 5;
int control = 1;
long int estado = LOW;
long int estado_anterior = LOW;
WidgetLCD lcd(V1);

void setup() {
  SERIAL_BEGIN(9600);
  MBserial.begin(MB_BAUDRATE, MB_BYTEFORMAT, TIMEOUT, REFRESH_INTERVAL);
  delay(95);
  firstData = false;
  power = 0;
  lastEnergy = 0;  // in case it has been recorded, use it
  energy = lastEnergy;
  pinMode(puertoControl, INPUT);
  Blynk.begin(auth, ssid, pass);
}
```

Después de crear una función en *setup*, que inicializa y establece los valores iniciales, la función de *loop* () hace exactamente lo que su nombre sugiere y hace ciclos consecutivos, lo que permite que su programa cambie y responda. Úselo para controlar activamente la placa Arduino. Acá es donde se lee periódicamente el puerto serial; Así mismo se lee el estado de la variable *puertoControl*, Si se accionó o presionó el botón de control, correspondiente al *widget* nombrado *change_messure*, pase a la siguiente variable a medir, incremente control en 1 o reinicie el contador, si ya alcanzó su valor límite. Si hay algo por leer en el maestro, Según lo que indique la variable control, lea el valor de interés. Voltaje, corriente, potencia activa, potencia aparente, frecuencia, factor de potencia o energía.

```
void loop() {
sei();
estado = digitalRead(puertoControl);
if(estado == HIGH && estado_anterior == LOW){
    if(control < 7)
      control++;
    else
      control = 1;
  }

  if (MBserial.available()) {
    voltage = volt.read(VOL_FAC);
    current = curr.read(CUR_FAC);
    power = pwr.read(POW_FAC);
    aPower = aPwr.read(POW_FAC);
    frequency = freq.read(FRE_FAC);
    energy = enrg.read(ENE_FAC);


     switch(control){
      case 1:  // voltaje
      lcd.clear();
      lcd.print(1, 0, "Voltage (V)");
      lcd.print(2, 1, (float)voltage / VOL_FAC);
      break;
```

Las últimas líneas de código, corresponden a las sentencias de una herramienta de Blynk para visualizar los datos medidos, mediante gráficas de funciones continuas. El widget, sólo admite configurar 4 canales. En este caso se van a leer el voltaje, la corriente, la potencia activa y la energía.

```
Blynk.virtualWrite(V0,(float)voltage / VOL_FAC);
Blynk.virtualWrite(V1,(float)current /CUR_FAC);
Blynk.virtualWrite(V2,(float)power / POW_FAC);
Blynk.virtualWrite(V3,(float)energy /ENE_FAC);
```

En el archivo *.cpp* se define el constructor del objeto *modbusSensor*, que está pensado para contener grupo de valores del dispositivo esclavo. Dentro de cada instancia *modbusSensor* los datos se almacenan como una secuencia de bytes

que refleja el contenido de los registros del esclavo, además del *frame* o estructura de petición.

```
// Constructor
modbusSensor::modbusSensor(modbusMaster * mbm, uint8_t id, uint16_t adr, uint8_t hold) {
  _frame[0] = id;
  _frame[1] = READ_INPUT_REGISTERS;
  _frame[2] = adr >> 8;
  _frame[3] = adr & 0x00FF;
  _frame[4] = 0x00;
  _frame[5] = 0x02;
  uint16_t crc = calculateCRC(_frame, 6);
  _frame[6] = crc & 0x00FF;
  _frame[7] = crc >> 8;
  _status = MB_TIMEOUT;
  _hold = hold;
  _value.f = 0.0;
  (*mbm).connect(this);
}
```

En el archivo ModbusSensor.cpp, se escribe la rutina del constructor para el esclavo, así como algunas funciones para conectar al *MBserial* y conocer el estado de conexión del esclavo con la variable *status*. Hay que recordar que los métodos son funciones y los atributos son variables para una clase.[6]

El control del flujo de datos se realiza con una máquina de estados que le indica al maestro que acción debe realizar entre recibir o enviar los datos. Los estados de envío y recepción de datos se ajustan a lo especificado en el protocolo Modbus RTU.

[6] WIKIBOOKS. Miembros de una clase (métodos y atributos). [en línea]. citado en Enero de 2017. disponible en: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Objetos_y_Clases#Miembros_de_una_clase_.28m.C3.A9todos_y_atributos.29

```
//constructor
modbusMaster::modbusMaster(HardwareSerial * MBSerial, uint8_t TxEnPin) {
  _state = STOP;
  _TxEnablePin = TxEnPin;
  pinMode(_TxEnablePin, OUTPUT);
  _MBSerial = MBSerial;
  _totalSensors = 0;
  for (uint8_t i = 0; i < MAX_SENSORS; i++) {
    _mbSensorsPtr[i] = 0;
  }
}
```

Se muestra el constructor para el maestro, que por norma debe tener el mismo nombre de la clase. la comunicación está dominada por el estado del pin digital asignado para controlar la comunicación *txenablepin*; que sólo se pone en alto al momento de enviar la trama de datos para el medidor.

5. PRUEBAS

En este capítulo se muestran los diferentes ensayos que permitieron ver la funcionalidad tanto del *hardware* como del *software* utilizado, se realizaron pruebas de caracterización de la exactitud de medida del medidor de forma local y remota, mediante. Los instrumentos usados para las pruebas realizadas son un multímetro, un osciloscopio, la terminal serial de arduino; Finalmente, comprobó la funcionalidad de la interfaz.

5.1 ERROR RELATIVO DE MEDICIÓN

En la tabla 8 del ANEXO A, se muestran los porcentajes de error de medición entregados por el fabricante del medidor EASTRON, para las diferentes variables eléctricas, no obstante, se realizó una prueba experimental para comparar los valores en la medición del medidor EASTRON, con un multímetro digital de referencia FLUKE 117, que se comportó como patrón de medida. En la tabla 3, se enuncian algunas de sus características de precisión de este aparato.[7]

Tabla 3. Especificaciones de precisión del fluke117.

Voltaje [V] True RMS	Rango/Resolución: 6,000 V/0,001 V Rango/Resolución: 60,00 V/0,01 V Rango/Resolución: 600,0 V/0,1 V Precisión:1,0 % + 3 (cc, de 45 Hz a 500 Hz) 2,0 % + 3 (de 500 Hz a 1 kHz)
Corriente [A] True RMS	Rango/Resolución: 6,000 A/0,001 A Rango/Resolución: 10,00 A/0,01 A Precisión: 1,5% + 3 Sobrecarga continua de 20 A durante 30 segundos máximo

[7] FLUKE. (s.f.). Multímetro Fluke 117 con detector de tensión sin contacto. [en línea]. Disponible en: http://www.fluke.com/fluke/coes/multimetros-digitales/fluke-117.htm?pid=55996

La prueba consistió en realizar una medición de voltaje y corriente de una bombilla AC incandescente, con el fin de cuantificar el error de medida de nuestro medidor, en la Figura 15 se evidencia el resultado de la medición para las variables que se compararon.

Figura 15. Medición del voltaje y la corriente de la bombilla.

Una vez obtenidos los datos de la medición experimental, se procede a calcular un error absoluto y un error relativo. El error absoluto corresponde a la diferencia entre el valor medido y el valor real. Se calcula mediante la siguiente formula:

$$e_{abs} = f_m - f_r \quad (1)$$

e_{abs}: error absoluto
f_m: valor medido
f_r: valor real

El error relativo es el cociente entre el error absoluto y el valor real. Para este caso, es valor real es el registrado por el patrón de medida, osea el obtenido por el medidor FLUKE; Cabe aclarar que este dispositivo no cuenta con un certificado de calibración.

$$e_{rel} = \frac{f_m - f_r}{f_r} \quad (2)$$

e_{rel}: error absoluto

f_m: valor medido

f_r: valor real

Tabla 4. Error absoluto y Error relativo

variable	Valor real [RMS]	Valor medido [RMS]	Error absoluto	Error relativo (%)
Voltaje [V]	127,8	123,6	4,2 [V]	3,286
Corriente [A]	0,841	0,82	0,021 [A]	2,497

El error absoluto da una medida de la desviación del valor real y el error relativo una desviación porcentual del mismo.

Las pruebas anteriormente realizadas, fueron aplicadas a los valores de la medición remota comparados con el medidor de energía. En la figura 16 se encuentran los valores medidos experimentalmente de las variables eléctricas de voltaje, corriente y potencia.

Figura 16. Resultado de la medición experimental

Tabla 5. Errores calculados por las ecuaciones (1) y (2).

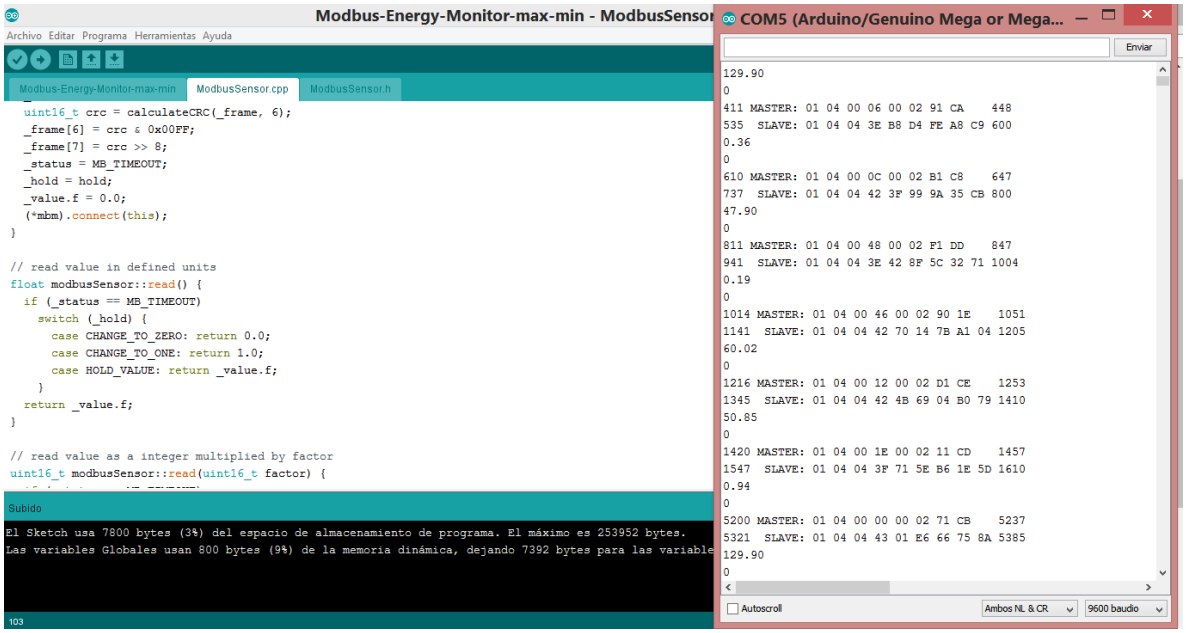
Variable	Valor real	Valor medido	Error absoluto	Error relativo (%)
Voltaje [Vrms]	125,4	125,5	0,01	0,008
Corriente [Arms]	0,82	0,81	0,01	1,219
Potencia [W]	110,6	110,7	0,1	0,09

Las pruebas de medición se realizaron a una tasa de 2400 baudios, esta velocidad parece no ser la indicada si se quieren hacer pruebas de este tipo, ya que nos demora un poco para arrojarnos el resultado de la medida remota.

5.2 GESTIÓN DE DATOS DEL MEDIDOR EASTRON SDM120

El código que se empleó fue desarrollado para gestionar los datos del medidor EASTRON SDM 120C y, además, calcular los valores máximos y mínimos de las variables eléctricas alojadas en los registros de memoria (peninquen, 2015), con una tarjeta Arduino, y el convertidor RS485. Se realizó un montaje que incluía la tarjeta Arduino, el convertidor de protocolo RS485-TTL y el medidor, con el objetivo de ensayar y verificar la funcionalidad de dicho código, que fue desarrollado para visualizar la información de lectura de la medición en el puerto serial del IDE de arduino. El resultado de la gestión de los datos puede verse en la Figura 17.

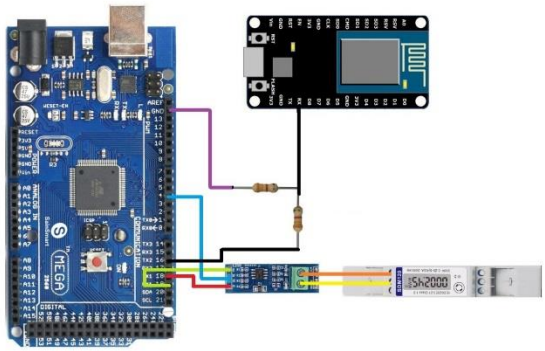
Figura 17. Visualización de la comunicación del Arduino con el medidor SMD120



Una vez revisada la funcionalidad del código, la siguiente labor era lograr compilarlo la tarjeta NodeMCU, tarea que se logró satisfactoriamente, sin embargo, no se podían obtener los datos de la medición en la terminal serial; y aunque, se intentó solucionar el problema de la comunicación por *software*, más adelante se iba a encontrar que la falla se encontraba en el *hardware*.

Una solución que permitía entregar los datos a la web, consistía en usar la tarjeta Arduino para la gestión de los datos, y la tarjeta NodeMCU como un *gateway* o enlazador de red. La conexión realizada se muestra en la Figura 18; consiste en enviar el dato de la medición por el puerto serial de la arduino, al puerto Rx0 de la NodeMCU; se debe tener en cuenta que al usar el puerto Serial de la NodeMCU es posible usar la interfaz del computador debido a que este mismo puerto ya se encuentra ocupado con el código, lo que haría que entre en conflicto el puerto afectando la comunicación. Esta solución, aunque es válida, no es la más rentable ni la más eficiente por usar recursos adicionales.

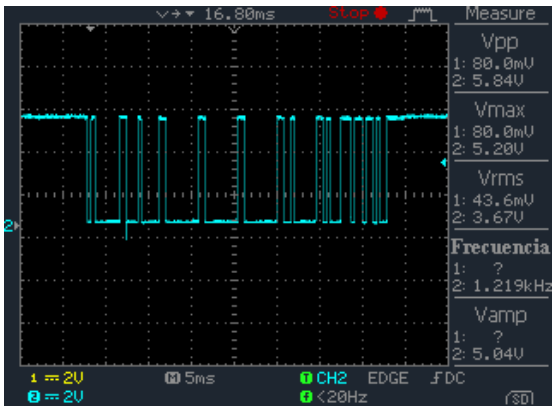
Figura 18. Conexión NodeMCU, Arduino, Medidor



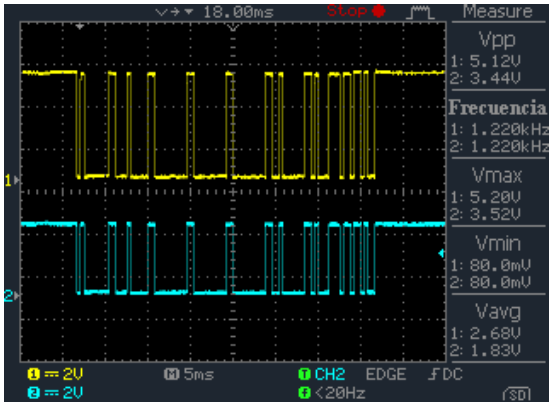
Con la experiencia de haber realizado pruebas de *software* para solucionar el problema de funcionalidad de la tarjeta NodeMCU para hacer la gestión de datos del medidor, se hicieron unas pruebas de *hardware* mediante la visualizaron de las señales digitales generadas con ayuda del osciloscopio; En la **¡Error! No se ncuentra el origen de la referencia.** vemos una señal que representa la trama de datos enviada al medidor para obtener el valor de tensión alojado en el registro de dirección 3001 (ver tabla de direcciones registros 3X), esta señal se encuentra conformada por un paquete de datos que los define el protocolo MODBUS.

La imagen a. de la Figura 19, representa la trama de datos generada por la tarjeta ARDUINO, y la imagen b. de la misma figura, muestra la señal producida por la NodeMCU en azul y después de elevar el nivel de tensión con el convertidor de nivel; como puede verse, las señales tienen la misma frecuencia de operación, el mismo tiempo de visualización, lo que puede garantizar que se están enviando la misma cantidad de datos es ese mismo intervalo de tiempo.

Figura 19. Trama generada transmitida por TX



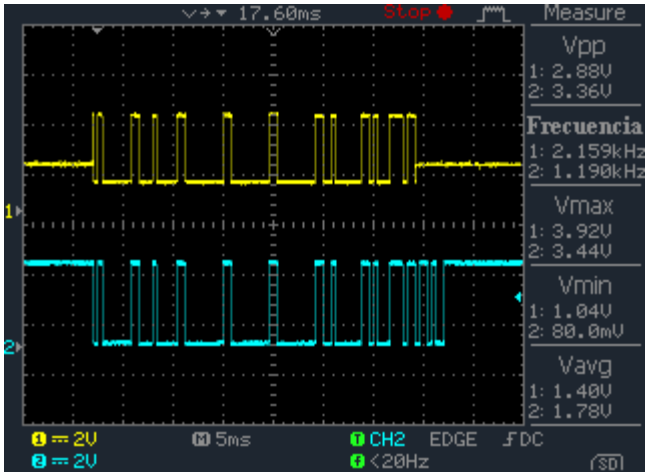
b) Señal Tx NodeMCU



a) Señal Tx ARDUINO

En la Figura 20, la señal azul corresponde a la entrada por serial del convertidor, medida desde el pin de transmisión de la NodeMCU, y representa una señal unipolar con niveles de voltaje entre 0 y 3.3V; mientras que la señal amarilla corresponde a la entrada RS485 del convertidor con valores de voltaje entre +1V y +3.3V, ahora, la señal que controla el flujo de datos, debe ser un pulso de ancho igual o superior al ancho de los datos transmitidos, de acuerdo con la comunicación *halfduplex*, Si esto se cumpliera, señal en amarillo debería tener la misma duración que la señal en azul y se puede ver que esto no se cumple, al no cumplirse ocasiona un truncamiento en la trama y un error en la comunicación entre la NodeMCU y el medidor. Luego de detectar el error, se le dio solución mediante *software,* aumentando el ancho del pulso de la señal de habilitación, obteniendo la respuesta del medidor

Figura 20. Señal truncada de entrada al RS485



6. RESULTADOS

Las integraciones de diferentes dispositivos permitieron el desarrollo de un dispositivo que comprende tanto conmutar como la medición eléctrica de manera remota de una lampara, esta herramienta se denominó CRI, para una red monofásica y corriente máxima de 45A. El sistema fue pensado para controlar una lampara LED pero de acuerdo a nuestra experiencia, consideramos que el sistema tiene la cualidad de adaptarse a cualquier otra carga con unos cambios mínimos; Gracias a esa cualidad dinámica de este sistema remoto inteligente, se pudo implementar un mecanismo de alarma, que envía un mensaje al correo electrónico advirtiéndonos de presencia humana en el lugar.

El objetivo fue de este proyecto era tomar un dispositivo para la medición y volverlo inteligente con un hardware embebido con conectividad wifi, y emplear una de tantas plataformas IoT que existen actualmente, para construir un sistema que le permita al usuario mejorar sus hábitos de consumo de energía que propendan a un mejor URE. En este capítulo se muestran el resultado final de la implementación del CRI. Sobre su software que corresponde al interfaz construido en una aplicación Android para IoT, y su hardware reunido en una PCB, junto con su costo.

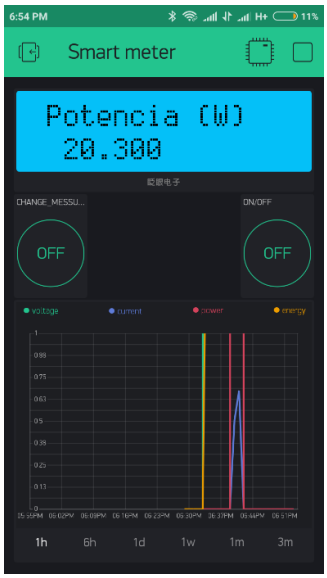
6.1 INTERFAZ

Para construir una interfaz en Blynk, fue necesario descargar la aplicación, crear una cuenta y programar el *hardware* con una llave que nos entrega la aplicación por correo electrónico; cada *widget* tiene una configuración propia, que se realiza rápidamente de forma intuitiva no obstante en puede visitar el ANEXO B para obtener mas información sobre esta configuración.

Fisicamente, en el medidor, se debe accionar un pulsador para cambiar la lectura de la variable eléctrica, visualizada en el display, la interfaz virtual de la Figura 21. se creó bajo esta misma funcionalidad, el de accionar un interruptor virtual (*CHANGE_MESSURE*), que le permitirá al usuario, seleccionar entre las distintas variables eléctricas leídas por el medidor, tensión, corriente, potencia activa, potencia aparente, factor de potencia, frecuencia que serán mostradas en la pantalla LCD virtual; además, la interfaz posee otras funcionalidades, dentro de las cuales está un pulsador tipo conmutador (*ON/OFF*) para controlar la carga remotamente, y un interfaz grafico que construye una señal con los datos leidos en el medidor, y los visualiza a una tasa de muestreo de 1 segundo, este *widget* es configurable, me permite ver el historial de los datos en horas, días semanas y meses.

El manejo del conmutador remoto de la carga se realiza con una señal proveniente de la aplicación Blynk, y que se ve representada en un cambio del nivel de voltaje en un pin digital de la tarjeta NodeMCU, al accionar el botón del pin caracterizado, de esta manera se controla la carga remotamente. No es necesario nombrar el pin físico con el virtual, en este caso el código lo asocia por defecto. El pin virtual se asigna en el interfaz en el *widget* escogido, aunque se puede hacer por medio de la función *pinMode*.

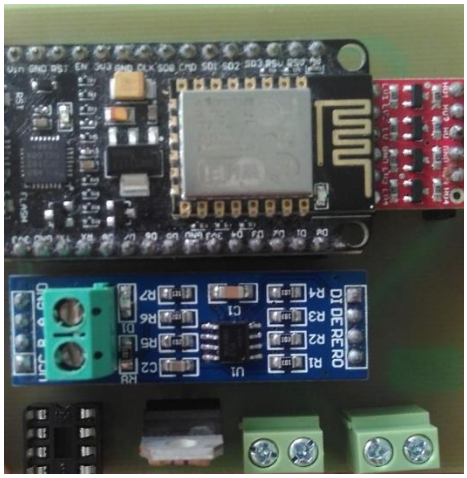
Figura 21. Interfaz de usuario para el control remoto inteligente



6.2 TARJETA FINAL

En la figura 22 se evidencia el hardware para el CRI, que corresponde a la integración de los dispositivos empleados para la gestión de datos del medidor de energía y la conmutación de la carga. Estos elementos se montaron sobre una PCB de una sola capa, esta placa esta compuesta por zocalos tipo hembra, porque todos ellos son de inserción. La conexión externa con el resto del sistema se realiza por medio de las borneras.

Figura 22. Hardware control remoto inteligente



En la siguiente tabla se encuentra toda la información sobre el costo que empleó desarrollar este sistema de medición y control, local y remoto. Los precios mostrados fueron tomados de la tienda de Aliexpress, donde se encuentran todos los elementos para la implementación, y además es una de las plataformas comerciales para comprar tecnología a buenos precios; en estos valores se incluye el costo de envío. La PCB, se hizo en una empresa de Bucaramanga, con una calidad buena, pero existen empresas aún más especializadas en la construcción de estas placas.

El costo del sistema de control inteligente tiene un precio aproximado de 150.000 pesos colombianos, valor obtenido de la conversión de los 47 USD que costó el *hardware,* enunciado en la Tabla 6, mas el costo del permiso para uso de mas herramientas de la aplicación; No deja un precio llamativo si revisamos los dispositivos que se comercializan en diferentes tiendas para el control y monitoreo de cargas, además, estos sistemas están diseñados con un *hardware* más robusto, y ofrecen una solución para el control de diferentes cargas o sistemas enfocados solo en medición de energía. un sistema similar al construido en este proyecto puede tener un costo superior en el mercado al nuestro.

En el análisis de costos que se realizó, no entra el costo del sensor, la lampara utilizada, el servicio de internet en la residencia, el teléfono móvil; Además se invirtió 1USD en la aplicación, que, aunque es gratuita, ofrece un número limitado de widgets para la creación de los proyectos.

Tabla 6. Costos del hardware para el control remoto inteligente

ELEMENTO	PRECIO (USD)
EASTRON SDM120	26,28
NODEMCU	4,78
CONVERSOR 3.3V A TTL	1,20
CONVERSOR RS485-UART	0,38
Q6025L	1,73
MOC3021	2
RESISTENCIA 330 1/8 W	0,02
RESISTENCIA 470 1/8 W	0,02
BORNERAS 8A	0,05
TIRAS DE ZOCALOS	0,25
ZOCALO MOC 3021	0,09
PCB	10
	46,79

7. CONCLUSIONES

- Se implementó un esquema serie, que cuando mide una corriente de 10mA, envía un mensaje al correo reportando presencia en el lugar, un sistema que puede servir para brindar seguridad de manera remota.

- Se logró la comunicación entre el medidor EASTRON mediante el protocolo Modbus con una NodeMCU, mediante un código cargado desde la IDE de arduino; Permitiendo una gestión de los datos de la medición remota a una velocidad de 2400 baudios para monitoreo diario de la energía consumida por la carga; Además de una conmuntación remota que trabaja a una tasa de 9600 baudios, Mediante el manejo de la información en tiempo real, el usuario puede cambiar hábitos de consumo con un sistema de bajo costo.

- Este sistema puede migrar y adaptarse para las diferentes cargas que pueda haber en una residencia. Permitiendo la revisión preventiva de diferentes equipos detectan, con herramientas para análisis gráfico, que permitirán detectar fallas eléctricas en estos. se puede esperar que esta medida pueda reducir el consumo innecesario de equipos en mal estado, pensando en reducir el desperdicio de energía eléctrica por descuido del usuario.

- Se desarrolló un interfaz medición y control fácilmente en una aplicación Se utiliza una interfaz compatible con Android y iOS para la monitorización y control del sistema de iluminación que brinda el soporte para que sea programación de alto nivel y que es compatible con plataformas de desarrollo para IoT, *hardware* embebido también fácil de programar

- El tamaño máximo del *buffer* RX de la NodeMCU es de 128 Bytes; El protocolo *Modbus* puede manejar hasta 256 Bytes de trama, para la gestión de datos de

nuestro medidor no se emplea esa cantidad de bytes. El sistema remoto está Limitado para conectar un máximo 32 dispositivos como esclavos.

- El medidor empleado en el sistema es bidireccional, se puede adaptar para aplicaciones de venta de energía solar fotovoltaica a la red eléctrica convencional, solo bastará con pedir al medidor la lectura de los registros de datos importados, que en este caso corresponde a la energía entregada a la red.

- La NodeMCU cuenta con un solo puerto serial, no se podía usar simultáneamente con la terminal serial0. Solo se ve la trama de datos transmitida por el maestro por medio del puerto de transmisión TX1.El código fue creado para ver la trama de datos que se envía y se recibe al medidor, por el puerto serial0 de la tarjeta se usa para hacer comunicación y el tx1 se emplea para visualizar la trama de datos enviados por el maestro, en el terminal serial de la IDE arduino.

8. RECOMENDACIONES

Desarrollar una plataforma para internet de las cosas, igual o mejor que la usada para desarrollar este proyecto, que ofrezca una interfaz gráfica intuitiva y dinámica, soporte y compatibilidad con las diferentes tarjetas que se ofrezcan en el mercado, con la finalidad de crear soluciones para IoT.

En esta implementación se usó el convertidor, recomendado por el desarrollador del código, y la programación fue en base a este dispositivo, pero tiene la limitación de trabajar solo a 5v, lo que implicó usar un convertidor de nivel para reducir la señal a 3.3V y que fuera adaptable con la NodeMCU. En el mercado se encuentran convertidores que trabajan a 3.3V, como es el sp486, para esta aplicación, usar este convertidor implicaba cambiar parte significativa del código. Recomendamos utilizarlo para una misma implementación, con el fin de reducir costo y espacio.

Construir un medidor de energía, que tenga embebido un módulo WIFI que le permita la conexión a internet, que le permita al usuario interactuar con él, mediante un interfaz como el propuesto en este proyecto.

Existen en el mercado diferentes tarjetas de desarrollo embebidas más avanzas y al mismo costo de la NodeMCU, como es el caso de la ESP32, que pertenece a la misma familia, pero al momento de realizar nuestra implementación no se encontraba este producto. Se recomienda trabajar con este modelo.

BIBLIOGRAFÍA

AI-THINKER TEAM. ESP-12E WiFi Module - Version1.0. [en línea]. Disponible en: http://www.maskau.dk/WP/wp-content/uploads/2017/03/ESP-12E_AI_Thinker.pdf

ALIXPRESS. Automático Interruptor infrarrojo PIR Motion Sensor de Luz LED de Techo 037. [en línea]. Disponible en: https://es.aliexpress.com/store/product/Free-shipping-High-Sensitive-360degree-Motion-Sensor-220-240V-AC-Motion-Sensor-Automatic-Light-Switch-ET037/817842_829256468.html?spm=a219c.search0104.3.17.mFXkpA&ws_ab_test=searchweb0_0,searchweb201602_1_10152_1006.

ALLDATASHEET. (s.f.). Electronic Components Datasheet Search. [en línea]. Disponbile en: http://www.alldatasheet.com/datasheet-pdf/pdf/414525/LITTELFUSE/HQ6025LH5.html

BBVAOPEN4U. El Internet de las Cosas de código abierto: plataformas y aplicaciones para desarrolladores. [en línea]. Citado en 25 de Agosto de 2015. Disponible en: https://bbvaopen4u.com/es/actualidad/el-internet-de-las-cosas-de-codigo-abierto-plataformas-y-aplicaciones-para

BLYNK. How Blynk Works. [en línea]. Disponible en: Obtenido de http://docs.blynk.cc/

CODENSA. Nuevos valores del kilovatio en Colombia. [en línea]. Disponbile en: https://www.codensa.com.co/hogar/valor-del-kilovatio-en-colombia-disminuye

CRESPO, Enrique. IoT conectando Cosas con Arduino. [en línea]. Citado el 1 de abril de 2017. Disponible en: https://www.youtube.com/watch?v=V8WRQ43JDi4

FLUKE. (s.f.). Multímetro Fluke 117 con detector de tensión sin contacto. [en línea]. Disponible en: http://www.fluke.com/fluke/coes/multimetros-digitales/fluke-117.htm?pid=55996

GITHUB. How to install Blynk library. [en línea]. Citado en Octubre de 2016. Disponible en: https://github.com/blynkkk/blynk-library/releases/tag/v0.4.8

GITHUB. Nodemcu. [en línea]. Disponible en: https://github.com/nodemcu

GITHUB. Peninquen - Modbus-Energy-Monitor-Arduino. Disponible en: Modbus: https://github.com/peninquen/Modbus-Energy-Monitor-Arduino

J. EASTRON ELECTRONIC INSTRUMENTS. Single-Phase Multifunction DIN rail Meter [en línea]. Citado el 10 de Octubre de 2016. Disponible en: http://www.eastrongroup.com/data/uploads/Eastron_SDM120CT-

KICKSTARTER. Blynk - build an app for your Arduino project in 5 minutes. [en línea]. Citado en enero de 2015. Disponible en: https://www.kickstarter.com/projects/167134865/blynk-build-an-app-for-your-arduino-project-in-5-m/description

NATIONAL INSTRUMENTS, (s.f.). Información Detallada sobre el Protocolo Modbus. [en línea]. Disponible en: http://www.ni.com/white-paper/52134/es/

OXER, Jonathan, Blemings, Hugh. Practical Arduino: Cool projects for Open Source Hardware. Technology in action. 2010. ISBN-13 (pbk): 978-1-4302-2477-8.

RAMOS SALAVERT, Isidro; LOZANO PEREZ, María Dolores. Entornos de desarrollo Integrados. En Ingenieria del software y base de datos: tendencias

actuales. Ediciones Universidad de Castilla La Mancha. 2000 ISBN 10: 8484270777 ISBN 13: 9788484270775.

REPÚBLICA DE COLOMBIA. Ministerio de Vivienda. Construcción sostenible. [en línea]. Disponible en: http://www.minvivienda.gov.co/cambio-climatico/mitigacion/construccion-sostenible

SCNEIDER ELECTRIC. (s.f.). ¿Cuales son las ventajas y desventajas del relé de estado solido con respecto a un relé electromecánico?. [en línea]. Disponible en: https://www.schneider-electric.com.ar/es/faqs/FA144221/

TOMASI, Wayne. Sistemas de Comunicaciones Electrónica. 4ª. Ed. Mexico: Pearson Educación. 2003.

UPME. Demanda y eficiencia energética (4 de marzo de 2016). Obtenido de http://www1.upme.gov.co/Paginas/Demanda-y-Eficiencia-Energetica.aspx

VANGUARDIA LIBERAL. B., H. C. [en línea]. Citado en 3 de mayo de 2016. Disponbile en: Obtenido de http://www.vanguardia.com/mundo/ola-verde/356976-en-que-gastamos-energia-los-santandereanos

WIKIBOOKS. Miembros de una clase (métodos y atributos). [en línea]. citado en Enero de 2017. disponible en: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Objetos_y_Clases#Miembros_de_una_clase_.28m.C3.A9todos_y_atributos.29

ANEXOS

Anexo A. Caracteristicas del Medidor

Especificaciones del medidor Eastron SMD120C

Nominal Voltage(Un)	230V AC
Operational Voltage	176~276V AC
Insulation Capabilities -AC Voltage Withstand -Impulse Voltage Withstand	 4KV for minutes 6KV-1.2µS waveform
Basic Current (Ib)	5 A
Maximum Rated Current (Imax)	45A
Operational Current Range	0.4% Ib~Imax
Over Current Withstand	20Imax for 0.01s
Operational Frequency Range	50~60Hz
Internal Power Consumption	≤2W/10VA
Pulse Output rate	1000/2000imp/kWh (default) 100/10/1 imp/kWh/kVarh (configurable)
Display LCD	Max. Reading 99999.9kWh
Communication Type	RS485
Communication Protocol	Modbus RTU

Rango de error del medidor.

Voltage	0.5% of range maximum
Current	0.5% of nominal
Frequency	0.2% of mid-frequency
Power factor	1% of Unity
Active power	1% of range maximum
Reactive power	1% of range maximum
Apparent power	1% of range maximum
Reactive energy	1% of range máximum
Active energy	Class 1 IEC62053-21 Class B EN50470-3

Anexo B. Generalidades de Blynk

GENERALIDADES DEL BLYNK

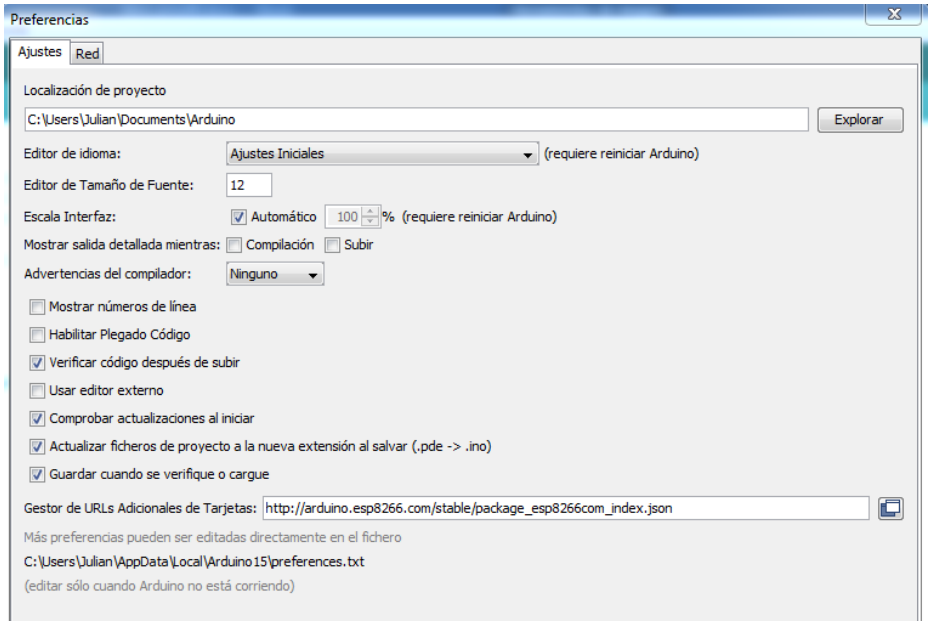
Esta aplicación es creada para la creación de múltiples proyectos del internet de las cosas de manera fácil y rápida. Es una plataforma para iOS y aplicaciones Android para el control de arduino*, raspberry pi* y otras tarjetas con capacidad de conectarse de internet mediante *Wi Fi*, Ethernet o el chip ESP8266. Funciona también a través de USB; Esto significa que se puede usar la aplicación conectando la tarjeta a una computadora portátil o de escritorio. Blynk es compatible con más de 400 tarjetas, e incluye el soporte para todas ellas; Podemos destacar las tarjetas de desarrollo *Arduino, Particle, mbed* de ARM, Energía de Texas Instruments, y diferentes SDK como *MicroPython*, *Node.js, OpenWRT* así como para el IDE de Arduino. La aplicación ofrece soporte para los diferentes protocolos de comunicación que ofrece arduino como bluetooth, GSM, WiFi, ETHERNET.[8]

La aplicación posee soporte para IDE ARDUINO, mediante la instalación de unas librerías, podemos programar nuestro dispositivo para que se conecte con el servidor de Blynk y a todos los recursos que ofrece esta aplicación, para el control y visualización de datos. podemos descargar aquellas librerías (github, 2016). La instalación de librerías de arduino se puede hacer de forma manual, o con ayuda de la IDE; De forma manual, tomamos la carpeta descomprimida y la pegamos en la carpeta de librerías de arduino que por defecto se encuentra en la ruta: C:\Users\USUARIO\Documents\Arduino. Si queremos instalar las librerías de *Blynk* con la IDE, seguimos los siguientes pasos, vamos a programa, incluir librería, añadir librería zip, o con el gestor de librerías. Para la instalación de dichas librerías, recurrimos a los siguientes pasos: Abrimos la interfaz de Arduino y

[8] BLYNK. How Blynk Works. [en línea]. Disponible en: Obtenido de http://docs.blynk.cc/

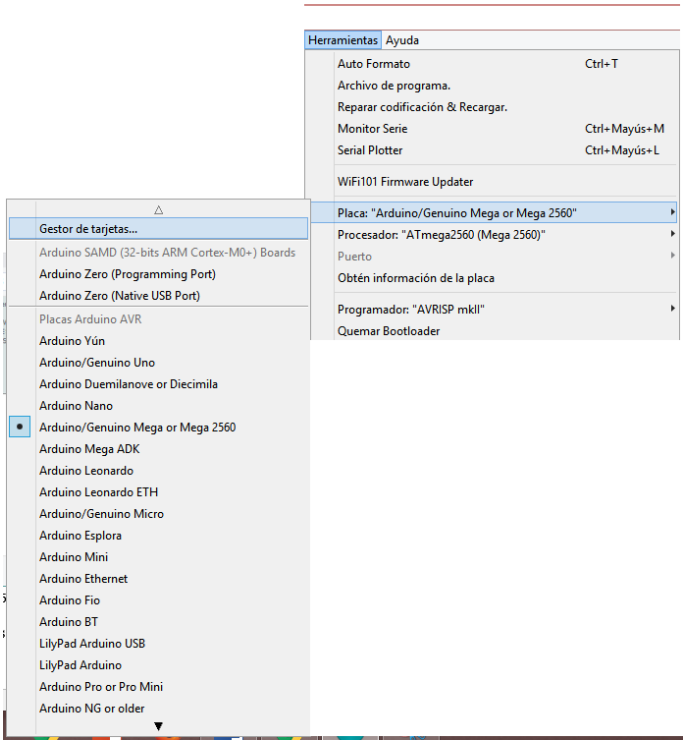
seleccionamos Archivo>Preferencias. En la lista de URLs de placas adicionales, agregamos la dirección mostrada en la siguiente figura:

Instalacion de librerías de ARDUINO



Con esta configuración tendremos acceso a diferentes recursos de código que han sido desarrollados por otras personas y nos facilitarán el cumplimiento de los objetivos propuestos. Para la instalación de paquetes vamos a **Herramientas>Placas>Administrador de Placas** y buscar el paquete "esp8266". Lo instalamos. Con este paso estamos listos para empezar a programar la placa.

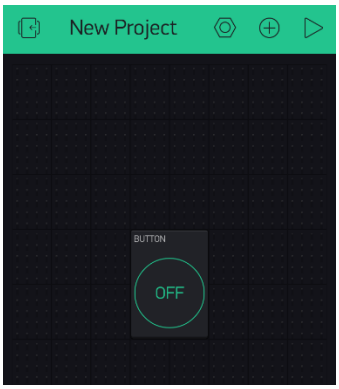
Instrucciones para cargar el paquete de ESP8266.



Una vez descargada la aplicación, se necesita crear una nueva cuenta que va a ser validada por una dirección de correo electrónico; lo que me permite tener acceso a mis proyectos desde diferentes dispositivos.

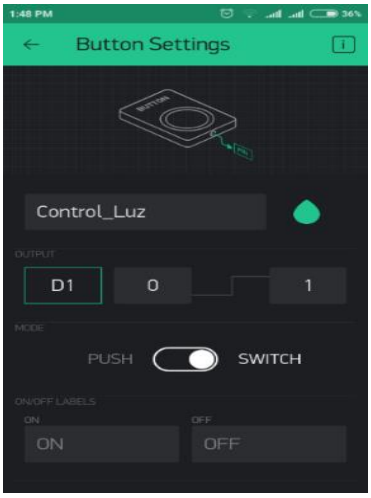
Los *Widgets* son herramientas desarrolladas por Blynk, para la creación de proyectos donde se requiera el control, visualización, tabulación y envío de datos, solo requiere arrastrar, soltar y configurar; cada *Widget* posee un cuadro de ayuda, con información para la configuración de sus parámetros. Vamos a dar el paso a paso de cómo crear un *Switch* virtual; Lo primero es crear un nuevo proyecto, se añade en este caso el *widget Button*, que se encuentra en la barra de herramientas señalizada con el signo (+) nos debe aparecer algo como lo mostrado en la Figura relacionada con la creación del *switch*; Una vez allí pulsamos dos veces el icono para configurar sus parámetros. Nos aparecerá un entorno como el mostrado en la Figura parámetros del *switch* .

Proyecto nuevo. Crear un *switch.*



La aplicación ofrece un interfaz de usuario como el mostrado en la figura, que representa un icono en forma de botón, para usarlo en dos modos de trabajo. como pulsador, donde el cambio se realiza si se mantiene oprimido el icono, o como conmutador que cambia de estado al contacto con el botón. Su configuración es sencilla e intuitiva, se coloca un nombre, se selecciona el modo, y se asocia la salida con un pin real de la tarjeta, en este caso el pin D0.

Parámetros del *switch*.

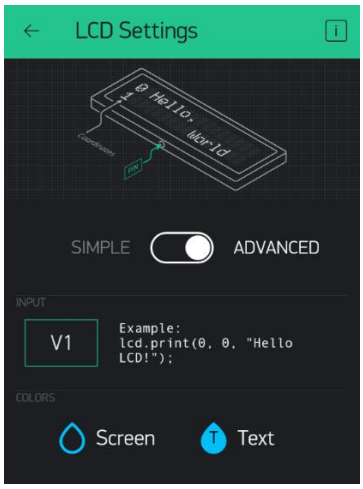


En el primer cuadro de texto, se nombra al *Button*. Existen dos modos de uso para este *widget*, y se configura de acuerdo a la aplicación deseada. Si se pone en

modo “*PUSH*”, vamos a tener un cambio de estado siempre que mantengamos pulsado el icono, y regresará a su estado anterior una vez dejemos de presionarlo. Ahora en el modo “*SWITCH*” solo bastará con pulsar una vez para tener cambio de un estado al otro.

La salida tiene dos opciones, existe el pin virtual y el digital. Para nuestro propósito elegimos un pin digital, nombrados entre D0 y D10, así mismo están marcados en la tarjeta. Aunque no todos son exclusivamente de propósito general, por ejemplo, los pines D4, D7 y D8, son puertos de comunicación UART. Una vez se tengan listos los ajustes en la aplicación, se guarda y se envía automáticamente un mensaje con el *Auth Token* al correo suministrado inicialmente.

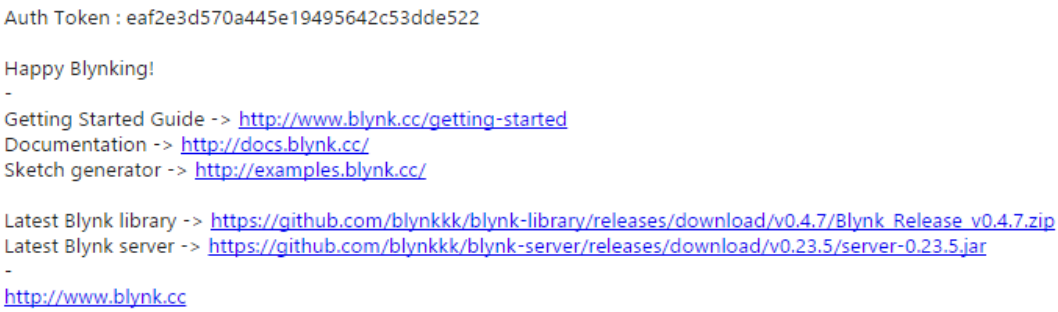
Configurar interfaz LCD



Este icono permite dos modos de trabajo; el modo simple, el icono LCD trabaja como un icono regular con frecuencia de lectura mínima de un segundo, y permite configurar el formato de visualización de los datos. El modo avanzado que fue el escogido; posee algunas ventajas puesto que no pone restricción con respecto a frecuencia de lectura de los datos, y permite trabajar con cualquier formato de visualización de los mismos.

Como podemos ver en la figura, la entrada que se escoge es un pin virtual, en este caso V1, y el color del display es azul.

Recepción del correo con el *Auth Token.*



Auth Token : eaf2e3d570a445e19495642c53dde522

Happy Blynking!
-
Getting Started Guide -> http://www.blynk.cc/getting-started
Documentation -> http://docs.blynk.cc/
Sketch generator -> http://examples.blynk.cc/

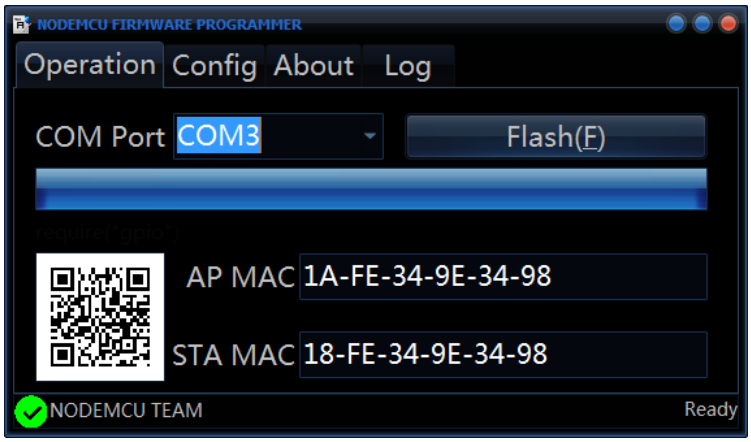
Latest Blynk library -> https://github.com/blynkkk/blynk-library/releases/download/v0.4.7/Blynk_Release_v0.4.7.zip
Latest Blynk server -> https://github.com/blynkkk/blynk-server/releases/download/v0.23.5/server-0.23.5.jar
-
http://www.blynk.cc

Anexo C. Firmware de la nodemcu

FIRMWARE DE LA NODEMCU

La NodeMcu incorpora el integrado CP2102, que permite establecer una conexión USB directamente con el computador y desde allí programar la tarjeta. La instalación del *firmware* es un proceso que se realiza por medio del puerto microUSB de la tarjeta y el puerto USB de la computadora. una vez establecida esta comunicación, comienza la descarga en el computador del driver identificado como “Silicon Labs CP210x USB to UART Bridge”; ejecutamos el NODEMCU *FIRMWARE* PROGRAMMER que es el *software* utilizado para esta tarea. tiene una interfaz como la mostrada en la Figura, se especifica el puerto COM a utilizar y se hace click en flash para cargar el *firmware* a la ESP8266, Esperamos a que se termine el proceso y nos aparezca el icono verde en la parte inferior. Por último, vamos a la pestaña “config” y revisamos que se encuentre el archivo nodemcu-4M.bin alojado en la dirección 0X00000 y la barra se encuentra en color verde, porque cuando el proceso tenga un error (e.g. file not exist), La barra será de color rojo.

Cargar el *firmware* a la NodeMCU



Existe otra manera de instalar el *firmware* al sistema, a partir de un *software*

creado por spressif (compañía que fabrica las ESP8266) para flashear la tarjeta, este *software* se puede encontrar en el enlace:
(http://bbs.espressif.com/viewtopic.php?f=5&t=433)

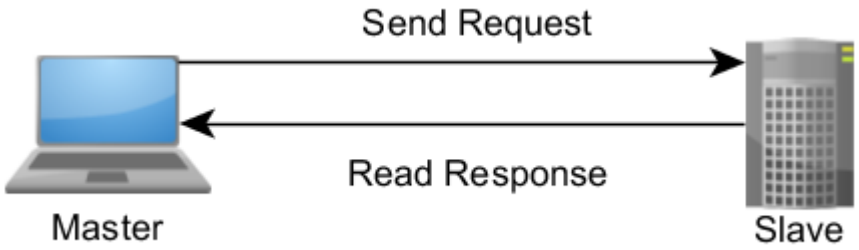
Allí pide usar el modo DIO y la opción de tamaño de flash de 32M y el último *firmware* del flash a 0x00000. Antes de flashear el *firmware*, mantener pulsado el botón FLASH y pulsar el botón RST una vez. se flasheará el *firmware* automáticamente y no necesitará nada más.

Anexo D. Generalides del Protocolo Modbus

GENERALIDES DEL PROTOCOLO MODBUS

Modbus es un protocolo industrial para la comunicación entre dispositivos de automatización, desarrollado para transferir datos por la capa serial, pero que su aplicación se ha extendido a TCP/IP y a UDP; funciona con la lógica solicitud-respuesta y que se implementa mediante la relación maestro-esclavo (ver Figura), donde el maestro es quien inicia la comunicación, realiza una petición y debe esperar que el esclavo responda.

Logica Solicitud-Respuesta

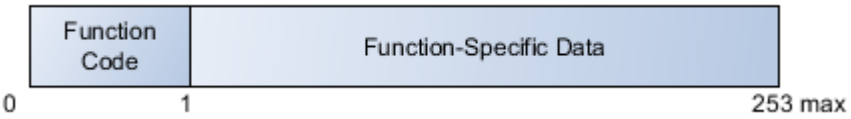


Provisto por National Instruments. ¿Qué es protocolo Modbus?

El protocolo *Modbus* se compone de la unidad de datos de protocolo (PDU) y la capa de red que define la unidad de datos de aplicación (ADU). La PDU consta de un código de función de un byte seguido de hasta 252 bytes de datos de funciones específicas; los códigos de función y sus datos son definidos explícitamente por el estándar. Cada función sigue un patrón. Primero, el esclavo valida entradas como el código de función, dirección de datos y el rango de datos. Después, ejecuta la acción solicitada y envía una respuesta adecuada al código. Si cualquier paso en este proceso falla, se regresa una excepción al solicitante. Por otra parte, Si se acepta el código de función, el dispositivo esclavo comienza a descomponer los datos de acuerdo con la definición de la función.

El transporte de datos para estas solicitudes es lo que se denomina la PDU.

La PDU de Modbus

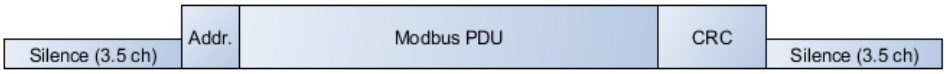


Provisto por National Instruments. Unidad de datos de protocolo [2]

Como se mencionó anteriormente el protocolo *modbus* puede usar diferentes protocolos de red, y existen variantes ADU para cada uno de ellos. Cada ADU es como una PDU más completa, proporciona un mecanismo para determinar el inicio y el final de cada solicitud; para más fiabilidad cada mensaje incluye la información de comprobación de errores. La ADU de la unidad terminal remota (RTU) usa a través de una línea serial.

La ADU de RTU parece ser mucho más simple, como se muestra en la **¡Error! No e encuentra el origen de la referencia.**.

La ADU de RTU



Provisto por National Instruments. Unidad de Datos de Aplicación

Primero, se usa una dirección para definir para que esclavo va la PDU, En la mayoría de las redes, una dirección 0 define la dirección de "*broadcast*". Es decir, un maestro puede enviar un comando de salida a la dirección 0 y todos los esclavos deben procesar la solicitud, pero ningún esclavo debe responder. Además de esta dirección, un CRC es usado para asegurar la integridad de los datos.

Sin embargo, la realidad es que la situación en las implementaciones más modernas está lejos de ser simple. Encapsulando el paquete hay un par de tiempos en silencio, es decir, periodos en los que no hay comunicación en el bus. Para una velocidad de transferencia de 9,600, este tiempo es alrededor de 4 ms. El estándar define una longitud mínima de silencio, independientemente de la velocidad de transferencia, de un poco menos de 2 ms. Primero, esto tiene un inconveniente de rendimiento ya que el dispositivo debe esperar a que el tiempo muerto se cumpla antes de que el paquete pueda ser procesado.

El protocolo MODBUS define el formato de la consulta del maestro y la respuesta del esclavo. La solicitud por parte del maestro contiene la dirección del dispositivo, un código de función que define la acción solicitada, el dato que se va a enviar y un campo de comprobación de errores. La respuesta contiene campos que confirman la acción tomada, los datos a devolver y un campo de comprobación de errores. Si se produjo un error al recibir el mensaje, entonces el mensaje se ignora, si el esclavo no puede realizar la acción solicitada, construirá un mensaje de error y lo enviará como su respuesta. Las funciones de protocolo MODBUS utilizadas por los medidores digitales Eastron, copian valores de registro de 16 bits entre el maestro y los esclavos. Sin embargo, los datos utilizados por el medidor están en formato de punto flotante IEEE 754 de 32 bits. Así, cada parámetro del instrumento se mantiene en dos registros adyacentes.

Consulta en la **¡Error! No se encuentra el origen de la referencia.**se ilustra una olicitud de un único parámetro de coma flotante, es decir, dos registros de protocolo Modbus de 16 bits.

Orden del registro a enviar

Primer Byte							Último Byte
Dirección del esclavo	Código de función	Dirección de memoria (Hi)	Dirección de memoria (Lo)	Cantidad de registros (Hi)	Cantidad de registros (Lo)	Chequeo de error (Lo)	Chequeo de error (Hi)

Dirección del esclavo: un número que identifica al esclavo, las direcciones van de 1 a 247, ósea, se representa con 8 bits, 0 está reservado para la dirección broadcast; Aunque los medidores digitales Eastron no admiten esta dirección.

Código de función: valor de 8 bits que indica al esclavo qué acción debe realizarse. (3, 4, 8 o 16 son válidos para el medidor digital Eastron).

Dirección de memoria (Hi): Los ocho bits más significativos de un número de 16 bits que especifican la dirección de los registros de datos solicitados.

Dirección de memoria (Lo): Los ocho bits menos significativos de un número de 16 bits que especifican la dirección de los registros de datos solicitados. Como los registros se utilizan en parejas y el conteo comienza en cero, entonces este debe ser un número par.

Cantidad de registros (Hi): Los ocho bits más significativos de un número de 16 bits que especifican el número de registros que se solicitan.

Cantidad de registros (Lo): Los ocho bits menos significativos de un número de 16 bits que especifican el número de registros que se solicitan. Como los registros se utilizan en parejas, este debe ser un número par.

Chequeo de error (Lo): Los ocho bits menos significativos de un número de 16 bits que representan el valor de verificación de errores.

Chequeo de error (Hi): Los ocho bits más significativos de un número de 16 bits que representan el error comprobar el valor.

La respuesta a una solicitud de un único parámetro de punto flotante tiene la siguiente estructura:

Estructura de la respuesta del medidor

Primer Byte								Último Byte
Dirección del esclavo	Código de función	Byte de conteo	Primer registro (Hi)	Primer registro (Li)	Segundo registro (Hi)	Segundo registro (Lo)	Chequeo de error (Lo)	Chequeo de error (Hi)

El medidor responde con la dirección de esclavo, el código de función solicitado, los Bytes de conteo y los datos que se encuentran almacenados en dos registros de 16 Bits.

Dirección del esclavo: valor de 8 bits que representa la dirección del esclavo que está respondiendo.

Código de función: valor de 8 bits que, cuando hay una copia del código de función en la consulta, significa que el esclavo reconoció la consulta y ha respondido.

Byte de conteo: valor de 8 bits que indica el número de bytes de datos que contienen la respuesta, en este caso el valor es 4, ya que los datos son en flotante.

Primer registro (Hi): Los ocho bits más significativos de un número de 16 bits que representan el primer registro solicitado en la consulta.

Primer registro (Lo): Los ocho bits menos significativos de un número de 16 bits que representan el primer registro solicitado en la consulta.
Segundo registro (Hi): Los ocho bits más significativos de un número de 16 bits que representan el segundo registro solicitado en la consulta.

Segundo registro (Lo): Los ocho bits menos significativos de un número de 16 bits que representan el segundo registro solicitado en la consulta.

Chequeo de error (Lo): Los ocho bits menos significativos de un número de 16 bits que representan el valor de comprobación de error.

Chequeo de error (Hi): Los ocho bits más significativos superiores de un número de 16 bits que representan el chequeo de error.

Si se detecta un error en el contenido de la consulta, se enviará al maestro una respuesta de error. La respuesta de excepción se identifica por el código de función que es una copia del código de función de consulta. Los datos contenidos en una respuesta de excepción son un código de error de byte único.

Esquema de respuesta para errores.

Primer Byte				Último Byte
Dirección del esclavo	Código de función	Código de error	Chequeo de error (Lo)	Chequeo de error (Hi)

Dirección de esclavo: valor de 8 bits que representa la dirección del esclavo que está respondiendo.

Código de función: valor de 8 bits que es el código de función en la consulta con 80 en hexadecimal, indicando que el esclavo no reconoce la consulta o no puede realizar la acción solicitada.

Código de error: valor de 8 bits que indica la naturaleza de la excepción detectada.
Chequeo de error (Lo): Los ocho bits menos significativos de un número de 16 bits que representan el valor de verificación de errores.

Chequeo de error (Hi): Los ocho bits más significativos de un número de 16 bits que representan el valor de comprobación de errores.

Ejemplo

En el siguiente ejemplo se va a hacer la lectura de la variable voltaje de un medidor con dirección 1.

Ejemplo de solicitud al medidor.

Dirección del esclavo	01
Código de función	04
Dirección de registro (Hi)	00
Dirección de registro (Lo)	00
Cantidad de registros (Hi)	00
Cantidad de registros (Lo)	02
Chequeo de error (Lo)	71
Chequeo de error (Hi)	CB

Los datos deben solicitarse en pares de registros, es decir, la dirección de los registros y la cantidad de registros deben ser números pares para solicitar una variable de punto flotante. De lo contrario, el medidor devolverá un mensaje de error.

En la respuesta devuelve el contenido de la lectura de voltaje como 230.2.

Respuesta del medidor

Dirección del esclavo	01
Código de función	04
Byte de conteo	04
Primer registro (Hi)	43
Primer registro (Lo)	66
Segundo registro (Hi)	33
Segundo registro (Lo)	34
Chequeo de error (Lo)	1B
Chequeo de error (Hi)	38

Anexo E. Código final

CODIGO FINAL

```
/*************************************************************************
  ModbusSensor.h
  create ModbusSensor and ModbusMaster classes to process values from
  a Eastron SMD120 and family.

  version 0.1 ALPHA 14/12/2015

  Author: Jaime García  @peninquen
  License: Apache License Version 2.0.

*************************************************************************/

#ifndef ModbusSensor_h
#define ModbusSensor_h

#include "Arduino.h"
#define MAX_SENSORS 32
// The maximum number of bytes in a modbus packet is 256 bytes.
// The serial buffer limits this to 128 bytes.
#define BUFFER_SIZE 128

// What happens when _status is diferent to MB_VALID_DATA?
#define CHANGE_TO_ZERO 0x00
#define CHANGE_TO_ONE  0x01
#define HOLD_VALUE     0xFF

class modbusMaster;

union pollFrame {
  uint8_t    array[8];
  struct {
    uint8_t  id;
    uint8_t  fc;
    uint16_t address;
    uint16_t data;
    uint16_t crc;
  };
};

union dataFloat {
  float f;
  uint8_t arr[4];
};


class modbusSensor {
  public:
    // Constructor
    modbusSensor(modbusMaster *mbm, uint8_t id, uint16_t adr, uint8_t hold);

    // read value in defined units
    float read();

    // read value as a integer multiplied by factor
    uint16_t read(uint16_t factor);
```

```
    // get status of the value
    uint8_t getStatus();

    // write sensor value
    void write(float value);

    //  change status
    uint8_t putStatus(uint8_t status);

    // get pointer to _poll frame
    uint8_t *getFramePtr();

  private:
    uint8_t     _frame[8];
    dataFloat   _value;
    uint8_t     _status;
    uint8_t     _hold;
};

class modbusMaster {
  public:
    //constructor
    modbusMaster(HardwareSerial *MBSerial, uint8_t TxEnPin);

    // Connect modbusSensor to modbusMaster array of queries
    boolean connect(modbusSensor *mbs);


    // begin comunication using ModBus protocol over RS485
    void begin(uint16_t baudrate, uint8_t byteFormat, uint16_t timeOut, uint16_t pollInterval);

    // process FSM and check if the array of sensors has been requested
    boolean available();

  private:
    void sendFrame();
    uint8_t readBuffer();
    uint8_t  _state;                    // Modbus FSM status (SENDING, RECEIVING, STANDBY, STOP)
    uint8_t  _TxEnablePin;              // pin to enable transmision in MAX485
    uint8_t  _totalSensors;             // constant, max number of sensors to poll
    uint16_t _timeOut;                  // constant, time since lastMillis to fail poll
    uint16_t _pollInterval;             // constant, time between polling same data
    uint16_t _T1_5;                     // inter-character time in microseconds
    uint8_t  _buffer[BUFFER_SIZE];      // buffer to process rececived frame
    HardwareSerial  *_MBSerial;
    modbusSensor    *_mbSensorsPtr[MAX_SENSORS]; // array of modbusSensor's pointers
    modbusSensor    *_mbSensorPtr;
    uint8_t         *_framePtr;
};

#endif
```

```
/*********************************************************************
  ModbusSensor.cpp
  create ModbusSensor and ModbusMaster classes to process values from
  a Eastron SMD120 and energy monitor family.

  version 0.1 ALPHA 14/12/2015

  Author: Jaime García  @peninquen
  License: Apache License Version 2.0.

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

#define MODBUS_SERIAL_OUTPUT  //VERBOSE

#ifdef MODBUS_SERIAL_OUTPUT
#define MODBUS_SERIAL_BEGIN(...) Serial1.begin(__VA_ARGS__)
#define MODBUS_SERIAL_PRINT(...) Serial1.print(__VA_ARGS__)
#define MODBUS_SERIAL_PRINTLN(...) Serial1.println(__VA_ARGS__)
#else
#define MODBUS_SERIAL_BEGIN(...)
#define MODBUS_SERIAL_PRINT(...)
#define MODBUS_SERIAL_PRINTLN(...)
#endif


#include "ModbusSensor.h"

// Finite state machine status
#define STOP      0
#define SENDING   1
#define RECEIVING 2
#define STANDBY   3

#define WAITING_INTERVAL 10

#define READ_INPUT_REGISTERS      0x04 // leer el contenido de los registros 3X del esclavo
#define MB_VALID_DATA     0x00  // ok
#define MB_INVALID_ID     0xE0  // id received don't match
#define MB_TIMEOUT        0xE2  // slave don't respond ¿maybe off?
#define MB_INVALID_CRC    0xE3  // calculated CRC don't match with recived CRC
#define MB_ILLEGAL_FC     0x01  // The function code is not supported by the product
#define MB_ILLEGAL_DATA   0x03  // Attempt to set a floating point variable to an invalid value
#define MB_SLAVE_FAIL     0x05  // An error occurred when the instrument attempted to store an
                                // update to it's configuration

// What happens when _status is diferent to MB_VALID_DATA?
#define CHANGE_TO_ZERO 0x00
#define CHANGE_TO_ONE  0x01
#define HOLD_VALUE     0xFF
```

```
uint16_t calculateCRC(uint8_t *array, uint8_t num) {
  uint16_t temp, temp2, flag;
  temp = 0xFFFF;
  for (uint8_t i = 0; i < num; i++)
  {
    temp = temp ^ array[i];
    for (uint8_t j = 1; j <= 8; j++)
    {
      flag = temp & 0x0001;
      temp >>= 1;
      if (flag)
        temp ^= 0xA001;
    }
  }
  // the returned value is already swapped
  // crcLo byte is first & crcHi byte is last
  return temp;
}

// Constructor
modbusSensor::modbusSensor(modbusMaster * mbm, uint8_t id, uint16_t adr, uint8_t hold) {
  _frame[0] = id;
  _frame[1] = READ_INPUT_REGISTERS;
  _frame[2] = adr >> 8;
  _frame[3] = adr & 0x00FF;
  _frame[4] = 0x00;
  _frame[5] = 0x02;
  uint16_t crc = calculateCRC(_frame, 6);
  _frame[6] = crc & 0x00FF;
  _frame[7] = crc >> 8;
  _status = MB_TIMEOUT;
  _hold = hold;
  _value.f = 0.0;
  (*mbm).connect(this);
}

// read value in defined units
float modbusSensor::read() {
  if (_status == MB_TIMEOUT)
    switch (_hold) {
      case CHANGE_TO_ZERO: return 0.0;
      case CHANGE_TO_ONE: return 1.0;
      case HOLD_VALUE: return _value.f;
    }
  return _value.f;
}

// read value as a integer multiplied by factor
uint16_t modbusSensor::read(uint16_t factor) {
  if (_status == MB_TIMEOUT)
    switch (_hold) {
      case CHANGE_TO_ZERO: return (uint16_t) 0;
      case CHANGE_TO_ONE: return (uint16_t) factor;
      case HOLD_VALUE: return (uint16_t)(_value.f * factor);
```

```
    }
  return (uint16_t)(_value.f * factor);
}

// get status of the value
uint8_t modbusSensor::getStatus() {
  return _status;
}

// write sensor value
void modbusSensor::write(float value) {
  _value.f = value;
}

//  put new status
uint8_t modbusSensor::putStatus(uint8_t status) {
  _status = status;
  return _status;
}

// get pointer to _poll frame
uint8_t *modbusSensor::getFramePtr() {
  return _frame;
}

//--------------------------------------------------------------------------------------------//

//constructor
modbusMaster::modbusMaster(HardwareSerial * MBSerial, uint8_t TxEnPin) {
  _state = STOP;
  _TxEnablePin = TxEnPin;
  pinMode(_TxEnablePin, OUTPUT);
  _MBSerial = MBSerial;
  _totalSensors = 0;
  for (uint8_t i = 0; i < MAX_SENSORS; i++) {
    _mbSensorsPtr[i] = 0;
  }
}

// Connect modbusSensor to modbusMaster array of queries
boolean modbusMaster::connect(modbusSensor * mbs) {
  if (_totalSensors < MAX_SENSORS) {
    _mbSensorsPtr[_totalSensors] = mbs;
    _totalSensors++;
    return true;
  }
  else return false;
}

// begin comunication using ModBus protocol over RS485
void modbusMaster::begin(uint16_t baudrate, uint8_t byteFormat, uint16_t timeOut, uint16_t pollInterval)

{
```

```
  _timeOut = timeOut;
  //_pollInterval = pollInterval - 1; // reduce 1 to compensate proccess delays
  if (baudrate > 19200)
    _T1_5 = 750;
  else
    _T1_5 = 16500000 / baudrate; // 1T * 1.5 = T1.5
  (*_MBSerial).begin(baudrate, SERIAL_8N2);

  _state = SENDING;
  digitalWrite(_TxEnablePin, LOW);
}

// process FSM and check if the array of sensors has been requested and processed
boolean modbusMaster::available() {
  static uint8_t  indexSensor = 0;                 // index of arrry of sensors
  static uint32_t nowMillis = millis();
  static uint32_t lastPollMillis = nowMillis;      // time to check poll interval
  static uint32_t sendMillis = nowMillis;          // time to check timeout interval
  static uint32_t receiveMillis = 0;

  switch (_state) {
    case SENDING:
      if (millis() - receiveMillis < WAITING_INTERVAL)
        return false;
      if (indexSensor < _totalSensors) {
        _mbSensorPtr = _mbSensorsPtr[indexSensor];
        _framePtr = (*_mbSensorPtr).getFramePtr();

        sendFrame();
        sendMillis = millis();
        _state = RECEIVING;
        return false;
      }
      else {
        indexSensor = 0;
        _state = STANDBY;
        return true;
      }
    case RECEIVING:
      if ((*_MBSerial).available()>0) {
        readBuffer();
        MODBUS_SERIAL_PRINTLN((*_mbSensorPtr).getStatus(), HEX);
        receiveMillis = millis();
        indexSensor++;
        _state = SENDING;
      }
      else if (millis() - sendMillis > _timeOut) {
        MODBUS_SERIAL_PRINTLN("fallo");
        (*_mbSensorPtr).putStatus(MB_TIMEOUT);
        indexSensor++;
        _state = SENDING;
      }
      return false;
    case STANDBY:
      nowMillis = millis();
```

```
      if (nowMillis - lastPollMillis > _pollInterval) {
        lastPollMillis = nowMillis;
        _state = SENDING;
      }
      return false;
    case STOP: // do nothing
      return false;
  }
}

uint8_t modbusMaster::readBuffer() {
  uint8_t index = 0;
  boolean ovfFlag = false;
  MODBUS_SERIAL_PRINT(millis());
  MODBUS_SERIAL_PRINT("  SLAVE:");
  while ((*_MBSerial).available()) {
    // The maximum number of bytes is limited to the serial buffer size
    // of BUFFER_SIZE. If more bytes is received than the BUFFER_SIZE the
    // overflow flag will be set and the serial buffer will be read until
    // all the data is cleared from the receive buffer, while the slave is
    // still responding.
    if (ovfFlag)
      (*_MBSerial).read();
    else {
      if (index == BUFFER_SIZE) ovfFlag = true;
      _buffer[index] = (*_MBSerial).read();
      //#ifdef MODBUS_SERIAL_OUTPUT
      if (_buffer[index] < 0x10)
        Serial1.print(F(" 0"));
      else
        Serial1.print(F(" "));
      Serial1.print(_buffer[index], HEX);
      //#endif
      index++;
    }
    // This is not 100% correct but it will suffice.
    // worst case scenario is if more than one character time expires
    // while reading from the buffer then the buffer is most likely empty
    // If there are more bytes after such a delay it is not supposed to
    // be received and thus will force a frame_error.

    delayMicroseconds(_T1_5); // inter character time out
  }
  MODBUS_SERIAL_PRINT(" ");
  MODBUS_SERIAL_PRINTLN(millis());

  // The minimum buffer size from a slave can be an exception response of 5 bytes.
  // If the buffer was partially filled set a frame_error.
  if (index < 5 || ovfFlag)
    return ((*_mbSensorPtr).putStatus(MB_SLAVE_FAIL));

  if (_buffer[0] != _framePtr[0])
    return ((*_mbSensorPtr).putStatus(MB_INVALID_ID));
```

```
  uint16_t crc = calculateCRC(_buffer, index - 2);
  if (_buffer[index-1] != crc >> 8 && _buffer[index-2] != crc & 0x00FF)
    return ((*_mbSensorPtr).putStatus(MB_INVALID_CRC));

  if (_buffer[1] & 0x80 == 0x80) {
    MODBUS_SERIAL_PRINTLN(_buffer[0], HEX);
    MODBUS_SERIAL_PRINTLN(_framePtr[0], HEX);
    return ((*_mbSensorPtr).putStatus(_buffer[2])); // see exception codes in define area
  }

  switch (_buffer[1]) {
    case READ_INPUT_REGISTERS:

      if (_buffer[2] == 4) {
        dataFloat temp;
        temp.arr[3] = _buffer[3];
        temp.arr[2] = _buffer[4];
        temp.arr[1] = _buffer[5];
        temp.arr[0] = _buffer[6];
        MODBUS_SERIAL_PRINTLN(temp.f);
        (*_mbSensorPtr).write(temp.f);
        return ((*_mbSensorPtr).putStatus(MB_VALID_DATA));
      }
      else
        return ((*_mbSensorPtr).putStatus(MB_ILLEGAL_DATA));

    default:
      return ((*_mbSensorPtr).putStatus(MB_ILLEGAL_FC));
  }
}

void modbusMaster::sendFrame() {
  digitalWrite(_TxEnablePin, HIGH);
  MODBUS_SERIAL_PRINT(millis());
  MODBUS_SERIAL_PRINT(F(" MASTER:"));
  for (uint8_t i=0; i < 8; i++) {
    (*_MBSerial).write(_framePtr[i]);
#ifdef MODBUS_SERIAL_OUTPUT
    if (_framePtr[i] < 0x10)
      Serial1.print(F(" 0"));
    else
      Serial1.print(F(" "));
    Serial1.print(_framePtr[i], HEX);
#endif
  }

  (*_MBSerial).flush();
  MODBUS_SERIAL_PRINT("    ");
  MODBUS_SERIAL_PRINTLN(millis());
  delayMicroseconds(_T1_5);
  digitalWrite(_TxEnablePin, LOW);

  // It may be necessary to add a another character delay T1_5 here to
  // avoid truncating the message on slow and long distance connections
```

```
/**********************************************************************
* ModbusEnergyMonitor example
* An example to collect data from a Modbus energy monitor using ModbusSensor class
* to datalogger, include a RTC DS3231 and a SD card
* version 0.1 ALPHA 14/12/2015
*
* Author: Jaime García  @peninquen
* License: Apache License Version 2.0.
*
**********************************************************************/
/*

*/

#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

char auth[] = "7b851565100c449187af92390e5b335e";
char ssid[] = "Redmi";
char pass[] = "31144269";
//char ssid[] = "JESUSGOMEZ";
//char pass[] = "29282930";


#define SERIAL_OUTPUT 1
#if SERIAL_OUTPUT
#   define SERIAL_BEGIN(...) Serial1.begin(__VA_ARGS__)
#   define SERIAL_PRINT(...) Serial1.print(__VA_ARGS__)
#   define SERIAL_PRINTLN(...) Serial1.println(__VA_ARGS__)
#else
#   define SERIAL_BEGIN(...)
#   define SERIAL_PRINT(...)
#   define SERIAL_PRINTLN(...)
#endif

#include "ModbusSensor.h"

#define MB_SERIAL_PORT &Serial
#define MB_BAUDRATE       2400          // b 2400
#define MB_BYTEFORMAT     SERIAL_8N2     // Prty n
#define TxEnablePin        4
#define TIMEOUT           100


#define ID_1  1                       // id 001  modbus id of the energy monitor
#define REFRESH_INTERVAL  5000        // refresh time, 5 SECONDS
#define WRITE_INTERVAL 20000UL        // values send to serial port, 1 minute ( 60 * 1000)
#define KWH_2_WS 36000000

// Direcciones registros de datos solo lectura. Valores tipo float.
// Utilizar funcion 04 lectura, numero de registros 16-bits 2.

#define VOL_ADR 0x0000    // VOLTAJE.
```

```
#define CUR_ADR 0x0006    // CORRIENTE.
#define POW_ADR 0x000C    // POTENCIA ACTIVA.
#define APO_ADR 0x0012    // Potencia Aparente.
#define PFA_ADR 0x001E    // Factor de potencia.
#define FRE_ADR 0x0046    // Frecuencia.
#define PEN_ADR 0x0048    // ENERGIA IMPORTADA KWH
#define REN_ADR 0x004A    // Energia exportada.
#define TEN_ADR 0x0156    // Energia activa Total.
#define TRE_ADR 0x0158    // Energia reactiva Total.

// multiplication factor, store value as an integer
#define VOL_FAC 10
#define CUR_FAC 100
#define POW_FAC 10
#define PFA_FAC 100
#define FRE_FAC 10
#define ENE_FAC 100

// instance to collect data using Modbus protocol over RS485
modbusMaster MBserial(MB_SERIAL_PORT, TxEnablePin);

//variables to poll, process and send values
modbusSensor volt(&MBserial, ID_1, VOL_ADR, CHANGE_TO_ZERO);
modbusSensor curr(&MBserial, ID_1, CUR_ADR, CHANGE_TO_ZERO);
modbusSensor pwr(&MBserial, ID_1, POW_ADR, CHANGE_TO_ZERO);
modbusSensor enrg(&MBserial, ID_1, PEN_ADR, HOLD_VALUE);
modbusSensor freq(&MBserial, ID_1, FRE_ADR, CHANGE_TO_ZERO);
modbusSensor aPwr(&MBserial, ID 1, APO ADR, CHANGE TO ZERO);
```

```
modbusSensor pwrFact(&MBserial, ID_1, PFA_ADR, CHANGE_TO_ONE);

uint16_t voltage; // integer, factor x10
uint16_t current; // integer, factor x100
uint16_t power;       // integer, factor x10
uint16_t lastEnergy, energy, avgPower;    // integer, factor x100
uint16_t frequency;     // integer, factor x100
uint16_t aPower;    // integer, factor x10
uint16_t powerFactor;       // integer, factor x100

unsigned long previousMillis = 0;
unsigned long currentMillis = 0;
boolean       firstData;
int puertoControl = 5;
int control = 1;
long int estado = LOW;
long int estado_anterior = LOW;
WidgetLCD lcd(V1);


void setup()
  SERIAL_BEGIN(9600);
  MBserial.begin(MB_BAUDRATE, MB_BYTEFORMAT, TIMEOUT, REFRESH_INTERVAL);
  delay(95);
  firstData = false;
  power = 0;
  lastEnergy = 0;  // in case it has been recorded, use it
  energy = lastEnergy;
  pinMode(puertoControl, INPUT);


  Blynk.begin(auth, ssid, pass);
}

void loop() {

sei();

 estado = digitalRead(puertoControl); // lee el estado del control

if(estado == HIGH && estado_anterior == LOW){
  // Si se accionó el puerto de control (presión del botón asociado)
    // pase a la siguiente variable a medir (incremente control en 1 o
    // reinícielo si ya alcanzó su valor límite)
    if(control < 6)
      control++;
    else
      control = 1;
  }

  if (MBserial.available()) {       // Si hay algo por leer en el maestro
    // Según lo que indique la variable control, lea el valor de interés.
    voltage = volt.read(VOL_FAC);
     current = curr.read(CUR_FAC);
     power = pwr.read(POW_FAC);
     aPower = aPwr.read(POW_FAC);
     frequency = freq.read(FRE_FAC);
     energy = enrg.read(ENE_FAC);
```

```
    switch(control){
     case 1:  // voltaje
     lcd.clear();
     lcd.print(1, 0, "Voltage (V)");
     lcd.print(2, 1, (float)voltage / VOL_FAC);
     break;
     case 2:  // corriente
     lcd.clear();
     lcd.print(1, 0, "Corriente (A)");
     lcd.print(2, 1, (float)current /CUR_FAC);
     break;
     case 3:  // potencia
     lcd.clear();
     lcd.print(1, 0, "Potencia (W)");
     lcd.print(2, 1, (float)power / POW_FAC);
     break;
     case 4:  // frecuencia
     lcd.clear();
     lcd.print(1, 0, "Frecuencia (Hz)");
     lcd.print(2, 1, (float)frequency / FRE_FAC);
     break;
     case 5:  // factor de potencia
     lcd.clear();
     lcd.print(1, 0, "FactorP.");
     lcd.print(2, 1, (float)powerFactor / PFA_FAC);
     break;
     case 6:  //  energía
     lcd.clear();
     lcd.print(1, 0,"Energía(KWH)");
     lcd.print(2, 1, (float)energy /ENE_FAC);
     break;
    }
  }

 currentMillis = millis();
// Escriba cada WRITE_INTERVAL
  if (currentMillis - previousMillis >= WRITE_INTERVAL) {
// almacene el tiempo de la impresión actual
    previousMillis = currentMillis;

  }

  currentMillis = millis();
  Blynk.virtualWrite(V0,(float)voltage / VOL_FAC);
   Blynk.virtualWrite(V1,(float)current /CUR_FAC);
    Blynk.virtualWrite(V2,(float)power / POW_FAC);
     Blynk.virtualWrite(V3,(float)energy /ENE_FAC);
     Blynk.virtualWrite(V3,(float)energy /ENE_FAC);
  Blynk.run();
}
```