UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

TRABAJO DE TITULACIÓN PREVIO A LA

OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO

PROYECTO TÉCNICO:

DISEÑO DE UN PROTOTIPO ANALIZADOR DE ENERGÍA PARA MONITOREO DE PARÁMETROS TENSIÓN, CORRIENTE Y TEMPERATURA CON REGISTRO DE EVENTOS EN SISTEMAS DE DISTRIBUCIÓN MONOFÁSICOS EN 240VAC DE HASTA 2KVA DE POTENCIA, MEDIANTE LA PLATAFORMA ARDUINO Y LA APLICACIÓN SCADA MONITORIZA, DIRIGIDO A LOS CLIENTES DE AMPER ECUADOR S.A.

AUTORES:

Kleber José Iglesias Vargas

Raúl Héctor Varas Borja

TUTOR:

Ing. Luis Antonio Neira Clemente MSc.

GUAYAQUIL – ECUADOR

2020

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

Nosotros, **KLEBER JOSE IGLESIAS VARGAS y RAUL HECTOR VARAS BORJA** autorizamos a la **Universidad Politécnica Salesiana** la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, se declara que los conceptos y análisis desarrollados y conclusiones del presente trabajo son de exclusiva responsabilidad del autor.

---------- ----------

Kleber José Iglesias Vargas
Cédula: 1207469592

Raúl Héctor Varas Borja
Cédula: 1714499066

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR

Nosotros, KLEBER JOSE IGLESIAS VARGAS, con documento de identificación N.º 1207469592 y RAUL HECTOR VARAS BORJA con documento de identificación N.º 1714499066, manifestamos nuestra voluntad y cedemos a la UNIVERSIDAD POLITÉCNICA SALESIANA la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de grado titulado: "DISEÑO DE UN PROTOTIPO ANALIZADOR DE ENERGIA PARA MONITOREO DE PARÁMETROS: TENSIÓN, CORRIENTE Y TEMPERATURA CON REGISTRO DE EVENTOS EN SISTEMAS DE DISTRIBUCIÓN MONOFÁSICOS EN 240VAC DE HASTA 2KVA DE POTENCIA, MEDIANTE LA PLATAFORMA ARDUINO Y LA APLICACIÓN SCADA MONITORIZA, DIRIGIDO A LOS CLIENTES DE AMPER ECUADOR S.A., en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos antes cedidos.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscrito este documento en el momento que hacemos entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

---------- ----------

Kleber José Iglesias Vargas
Cédula: 1207469592

Raúl Héctor Varas Borja
Cédula: 1714499066

CERTIFICADO DE DIRECCIÓN DE TRABAJO DE TITULACIÓN

Yo, **ING. LUIS NEIRA CLEMENTE.** Director del proyecto de titulación denominado: "**DISEÑO DE UN PROTOTIPO ANALIZADOR DE ENERGIA PARA MONITOREO DE PARÁMETROS: TENSIÓN, CORRIENTE Y TEMPERATURA CON REGISTRO DE EVENTOS EN SISTEMAS DE DISTRIBUCIÓN MONOFÁSICOS EN 240VAC DE HASTA 2KVA DE POTENCIA, MEDIANTE LA PLATAFORMA ARDUINO Y LA APLICACIÓN SCADA MONITORIZA, DIRIGIDO A LOS CLIENTES DE AMPER ECUADOR S.A.**" realizado por los estudiantes **KLEBER JOSE IGLESIAS VARGAS**, con documento de identificación N° 1207469592 y **RAUL HECTOR VARAS**, con documento de identificación N° 1207469592, certifico que ha sido orientado y revisado durante su desarrollo, por cuanto se aprueba la presentación del mismo ante las autoridades pertinentes.

Guayaquil, Abril del 2020

Ing. Luis Antonio Neira Clemente MSc

Docente

DEDICATORIA

Este proyecto de titulación está dedicado a mi madre Elicia Noemí Vargas Verdezoto, quien me ha incentivado y apoyado sin condición alguna durante toda esta etapa universitaria y a mi hermano Bayron Teofilo Iglesias Vargas quien es para mí un ejemplo a seguir.

Kleber José Iglesias Vargas

DEDICATORIA

Este proyecto de titulación está dedicado a una persona a la cual quiero y valoro mucho, la misma que dio inicio y oportunidad a una de mis grandes metas que es ser Ingeniero, mi querida Madrina Zoila Marchan Barragán.

Raúl Héctor Varas Borja

AGRADECIMIENTO

Agradezco a todas las personas que estuvieron presentes en esta etapa de mi vida, ya que gracias a ellas he logrado uno de mis más grandes objetivos.

Kleber José Iglesias Vargas

AGRADECIMIENTO

Agradezco a Dios por hacer posible esta gran meta, así como a mi pequeña familia, que es el motor de mi existencia, mi madre Jaqueline Borja, mi tía Mary Borja, mis hermanas Sol e Indi; las cuales de una u otra forma han aportado con su grano de arena para hacer esto posible, también agradezco a la señorita María Gracia que siempre ha estado junto a mi apoyándome para llegar a la meta.

Raúl Héctor Varas Borja

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO TÉCNICO	TEMA DE PROYECTO TÉCNICO
2020	Kleber José Iglesias Vargas Raúl Héctor Varas Borja	ING. Luis Antonio Neira Clemente, MSc.	"DISEÑO DE UN PROTOTIPO ANALIZADOR DE ENERGIA PARA MONITOREO DE TENSIÓN, CORRIENTE Y TEMPERATURA EN SISTEMAS DE DISTRIBUCIÓN MONOFÁSICOS A 240VAC DE HASTA 2KVA DE POTENCIA MEDIANTE LA PLATAFORMA ARDUINO Y LA APLICACIÓN SCADA MONITORIZA, DIRIGIDO A LOS CLIENTES DE AMPER ECUADOR S.A".

El presente proyecto técnico "DISEÑO DE UN PROTOTIPO ANALIZADOR DE ENERGIA PARA MONITOREO DE TENSIÓN, CORRIENTE Y TEMPERATURA EN SISTEMAS DE DISTRIBUCIÓN MONOFÁSICOS A 240VAC DE HASTA 2KVA DE POTENCIA MEDIANTE LA PLATAFORMA ARDUINO Y LA APLICACIÓN SCADA MONITORIZA, DIRIGIDO A CLIENTES DE AMPER ECUADOR S.A" tiene como objetivo diseñar un sistema que sea de fácil acople y

adaptable a cualquier tablero eléctrico monofásico con una capacidad máxima de hasta 2KVA y un voltaje máximo de 240VAC, el cual admita leer los parámetros básicos de voltajes, corrientes y temperatura en tiempo real; y así puedan ser monitoreados desde una computadora por medio de una red LAN.

Para el censado de los parámetros, se usa los disyuntores de entrada (alimentación del UPS) o disyuntores de salida (salida del UPS); los sensores de temperatura se encuentran adheridos en cada una de las acometidas de salida y otro dentro del tablero para medir la temperatura ambiente del interior.

Este sistema visualiza parámetros como: potencia, energía, voltaje, corriente y temperatura; todos estos parámetros se almacenan en una memoria SD, la misma que guarda un historial de máximo dos días; tiempo en el cual el cliente puede hacer uso de estos datos. Además, en caso de que se presente una falla en el sistema, se emite una señal de emergencia tanto en el SCADA como en el display de monitoreo.

El proyecto está orientado a los clientes de Amper Ecuador S.A para que implementen dicho sistema en sus tableros eléctricos no mayores a 2KVA monofásicos a 220AC.

Palabras claves: Energía, Arduino, monofásico, SCADA

ABSTRACT

YEAR	STUDENTS	PRJ.DIRECTOR	SUBJECT
2020	Kleber José Iglesias Vargas Raúl Héctor Varas Borja	ING. Luis Antonio Neira Clemente, MSc.	"DESIGN OF AN ENERGY ANALYZER PROTOTYPE FOR MONITORING OF VOLTAGE, CURRENT AND TEMPERATURE IN SINGLE-PHASE DISTRIBUTION SYSTEMS AT 240VAC OF UP TO 2KVA OF POWER THROUGH THE ARDUINO PLATFORM AND THE SCADA MONITORING APPLICATION, DIRECTED TO CLIENTS OF AMPER ECUADOR S.A ".

This technical project "DESIGN OF AN ENERGY ANALYZER PROTOTYPE FOR MONITORING OF VOLTAGE, CURRENT AND TEMPERATURE IN SINGLE-PHASE DISTRIBUTION SYSTEMS AT 240VAC OF UP TO 2KVA OF POWER THROUGH THE ARDUINO PLATFORM AND THE SCADA MONITORING APPLICATION, DIRECTED TO CLIENTS OF AMPER ECUADOR S.A ", its objective is to design a system that is easily attached and adaptable to any single-phase electrical panel with a maximum capacity of up to 2KVA and a maximum voltage of 240VAC, which allows reading the basic parameters of

voltages, currents and temperature in real time; and so they can be monitored from a computer through a LAN network.

For the census of the parameters, the input circuit breakers (UPS power) or output circuit breakers (UPS output) are used; temperature sensors are attached to each of the output connections and another one inside the panel to measure the ambient temperature inside.

This system displays parameters such as: power, energy, voltage, current and temperature; all these parameters are stored in an SD memory, the same one that keeps a history of maximum two days; time in which the client can make use of this data. Furthermore, in the event of a fault in the system, an emergency signal is emitted both on the SCADA and on the monitoring display.

The project is aimed at the clients of Amper Ecuador S.A so that they can implement this system on their electrical panels no larger than 2KVA single-phase at 220AC.

Keywords: Energy, Arduino, monophasic, SCADA.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El presente proyecto tiene como objetivo principal diseñar un sistema que sea de fácil manejo y se adapte a cualquier tablero eléctrico monofásico de capacidad máxima de hasta 2KVA y voltaje máximo de 240VAC; el cual permite leer parámetros de voltajes, corrientes y temperaturas en tiempo real, los mismos que son monitoreados desde una computadora.

Este trabajo ha sido estructurado en cinco capítulos que se detallan a continuación:

El primer capítulo contiene antecedentes, explicación del problema, importancia, alcance, delimitaciones y objetivos tanto generales como específicos.

El marco teórico, en el cual se fundamenta la investigación se encuentra en el segundo capítulo; en el mismo se detallan conceptos básicos de Arduino, sensores, módulos analógicos, termopares y software utilizados.

En el tercer capítulo se detallan las técnicas y métodos de investigación, así como cada uno de los diseños que se realizaron para llevar a cabo el proyecto.

El cuarto capítulo es una descripción de la implementación del sistema y como ésta permite el cumplimiento de los objetivos.

Finalmente, se realiza un análisis de los resultados que se obtuvieron con la ejecución del proyecto, así como también se detalla las respectivas conclusiones y recomendaciones.

1. EL PROBLEMA

1.1. Antecedentes

Amper Ecuador S.A es una empresa dedicada a dar solución de calidad de energía, la problemática se creó cuando algunos de los clientes solicitaron monitorear el sistema y de ser posible llevar un registro de eventos en los tableros de distribución eléctrica; lo que generó necesidad de colocar dispositivos que visualicen en tiempo real ciertos parámetros de energía en los tableros de Bypass que alimentan las cargas críticas, con el debido conocimiento que instalar un analizador de energía en tableros eléctricos es factible pero la solución tiende a ser muy costosa.

1.2. Importancia y Alcances

Actualmente en las industrias del país se usan sistemas de medición de energía con el fin de registrar parámetros eléctricos, los mismos que ayudan a determinar consumos y prevenir fallas que se puedan presentar.

Este proyecto permite realizar un aporte tecnológico, innovador y económico aplicado directamente a la ingeniería, ya que se puede visualizar en tiempo real parámetros que son de vital importancia en la industria; en el mismo se utilizaron equipos de licencia libre y sin restricciones de uso. Esta solución fue desarrollada con programación en lenguaje C y conocimientos básicos de electrónica adquiridos en el trascurso de la carrera.

1.3. Delimitación

1.3.1. Temporal

La implementación de este proyecto se realizó entre noviembre 2018 y marzo 2020.

1.3.2. Espacial

Este proyecto está implementado en Amper Ecuador de acuerdo con las especificaciones de gerencia general, esta empresa está ubicada en Urdenor 1 mz133 solar#4.

1.3.3. Académica

En el proyecto se plasmaron los conocimientos aprendidos en el proceso de formación como Ingeniero electrónico, tales como: programación basada en lenguaje C con sistemas Arduino, SCADA de software libre y equipamiento fácil de usar.

Además, para el diseño y construcción del prototipo se usaron los softwares AutoCAD, Proteus, Arduino, Nextion Editor y Acimut Monitoriza.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar un sistema analizador de parámetros de energía eléctrica y temperatura utilizando un SCADA de licencia libre para tableros eléctricos de los clientes de la empresa Amper Ecuador.

1.4.2. Objetivos específicos

Diseñar la estructura mecánica del proyecto en software AutoCAD 2018.

Diseñar las diferentes tarjetas electrónicas que constituyen el proyecto mediante software Proteus.

Plasmar los diferentes circuitos electrónicos y eléctricos en una sola tarjeta principal.

Utilizar el software SCADA Monitoriza de licencia libre para que el usuario no tenga restricciones en cuanto al uso de este.

Configurar el sistema SCADA y pantalla TFT en función de los parámetros adquiridos.

Establecer la comunicación entre todos los componentes del proyecto.

Configurar alarmas visuales y audibles mediante mecanismos lumínicos y sonoros integrados en el mismo.

Registrar en una base de datos y almacenar en memoria SD configurada dentro del prototipo, la cual respalde todos los parámetros registrados en un periodo de tiempo máximo de dos días.

Generar un informe digital en formato PDF.

2. FUNDAMENTOS TEÓRICOS

2.1. Arduino MEGA

Arduino Mega es una tarjeta de desarrollo basada en el microcontrolador ATmega2560. Cuenta con 54 entradas/salidas digitales (15 usadas como salidas PWM), 16 entradas analógicas, conexión USB, jack para alimentación DC, conector ICSP, 4 UARTs, un cristal de 16Mhz y un botón de reseteo. La placa Mega 2560 es compatible con la mayoría de los shields para Arduino UNO. (ARDUINO, ARDUINO MEGA, 2020).

En la ***Figura 1*** se observa la tarjeta Arduino Mega.

Figura 1. Arduino Mega (ARDUINO, ARDUINO MEGA, 2020)

2.2. Sensor de voltaje AC

El zmpt101b es un sensor mayormente utilizado en aplicaciones de medición de diferencia de potencial alterno, el rango de medición aproximado de este sensor tiene un límite de 250 voltios. Este sensor tiene un led indicador que tiene su propio circuito electrónico, de la misma forma, el módulo ZMPT101B tiene su sistema de acondicionamiento que consta de los siguientes elementos. (MECHATRONICS, 2018)

En la Tabla **1** se observan los componentes que se utilizaron para el sensor de voltaje AC ***y*** en la ***Figura 2*** se visualiza el sensor de Voltaje AC.

Tabla 1
Componentes electrónicos del sensor de voltaje AC

Cantidad	Componente	Detalles
1	Transformador	ZMPT101B
1	Operacional	LM358
1	Diodo Led	Rojo
1	Resistencia 100 Ω	Smd
1	Resistencia 1.001KΩ	Smd
6	Resistencia 10KΩ	Smd
1	Resistencia 10Ω	Variable
2	Resistencias 100KΩ	Smd
1	Resistencia 820KΩ	Smd
2	Capacitores 1uf	Smd
3	Capacitores 1nf	Smd

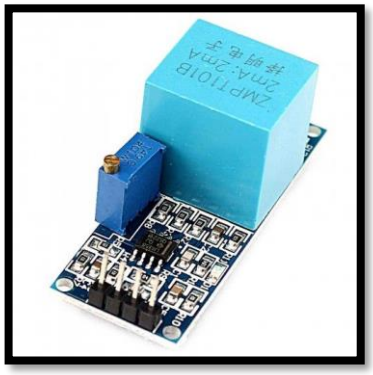


Figura 2. Sensor de voltaje AC (Electropeak, 2019)

2.3. Sensor de corriente AC

El SCT-013 es un sensor de corriente alterna tipo no invasivo. Este sensor nos permite medir intensidad de corriente. Consta de un núcleo ferromagnético tal cual una pinza amperimétrica comercial, la misma que nos permite abrazar el cable sin la necesidad de cortarlo.

El error en este sensor puede llegar al 2%, teniendo como condición principal que el núcleo ferromagnético se selle de manera correcta.

El sensor SCT-013-000 tiene una medición máxima de corriente igual a 100 amperios, con una relación de salida 100A:50mA.

Básicamente es un transformador con un núcleo partido que busca generar una intensidad en el secundario que sea proporcional a la intensidad que atraviesa el primario, pero para que ello funcione, el número de espiras del primario debe ser menor que el número de espiras del secundario. (Llamas, 2017).

En la ***Figura 3*** se observa el sensor de corriente AC.

Figura 3. Sensor de Corriente AC SCT-013 (YHDC)

2.4. Arduino Ide

Arduino IDE es una plataforma de código abierto basado en software y hardware de fácil uso. Pueden leer entradas y convertirlo en una salida: activar un motor, encender una luz, publicar algo on-line. Puede programar en el microcontrolador de su placa una serie de instrucciones o acciones a ejecutar a

base de ciertas variables con el fin de resolver un problema determinado. Para hacerlo, utiliza el lenguaje de programación C y el software Arduino (IDE), basado en el procesamiento. (ARDUINO, 2020)

2.5. Lector Micro SD Arduino

El dispositivo o módulo SD permite emplear como medio de almacenamiento una tarjeta externa SD, la misma que puede ser incorporada en proyectos de electrónica y Arduino.

Las SD y micro SD en la actualidad son un estándar, desplazando así a otros medios de almacenamiento de datos por su gran ventaja de capacidad y pequeño tamaño, por este motivo son parte fundamental de una gran cantidad de dispositivos, siendo en la actualidad componentes frecuentes en, tablets, ordenadores, Smartphone, entre otros.

El mundo Arduino es muy amplio, dentro de este podemos encontrar lectores de bajo coste tanto para tarjetas SD como micro SD.

Generalmente las lecturas de la tarjeta SD suelen ser a través del bus SPI. Aunque disponen de otras interfaces, como el bus I2C o UART, normalmente es preferible trabajar con SPI por su alta tasa de transferencia. (LLamas, 2016).

En la ***Figura 4*** se observa la tarjeta lectora micro SD.

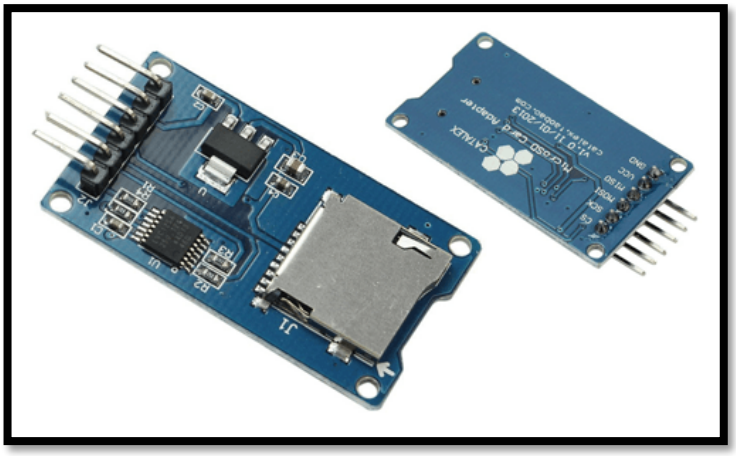


Figura 4. Tarjeta Lector Micro SD (LLamas, 2016)

2.6. Módulo de reloj DS3231

El módulo DS3231 es un reloj de alta precisión en tiempo real, cuenta con un oscilador de cristal con compensación de temperatura (TCXO). Esta integración del oscilador/cristal en el propio circuito integrado, en conjunto con la compensación de temperatura, asegura una precisión a largo plazo.

El RTC conserva un registro del tiempo en unidades de segundos, minutos, horas, día de la semana, fecha, mes y año, al final de cada mes la fecha es ajustada automáticamente para meses con menos de 31 días, incluyendo los años bisiesto.

El DS3231 es capaz de generar señales de reloj cuadradas de frecuencia configurable y además cuenta con 2 alarmas programables que pueden generar interrupciones en el microcontrolador principal en tiempos específicos.

El módulo a través del bus I2C se comunica con el microcontrolador el cual posee 2 pines los cuales pueden ser compartidos por varios dispositivos como,

expansores de IO, memorias EEPROM, controladores PWM, etc. (Factory, geekfactory, 2017).

En la ***Figura 5*** se visualiza la tarjeta reloj ds3231.

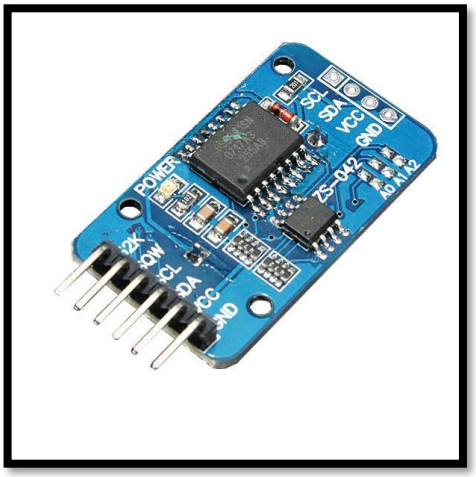


Figura 5. Tarjeta Reloj DS3231 (Factory, geekfactory, 2017)

2.7. Modulo para termopar tipo K

Existen muchos sensores de temperatura en el mercado, sin embargo, las soluciones basadas en silicio, como el LM35 por citar un ejemplo que sea familiar, están normalmente limitados a un rango de temperatura por debajo de los 150 grados centígrados. Lo que los deja fuera de consideración cuando debemos monitorear algún proceso con temperaturas superiores. Afortunadamente los termopares vienen a salvar el día, permitiéndonos mediciones en un rango más amplio, usualmente de varias centenas de grados centígrados y sin un costo excesivo.

Hacer uso de un convertidor Analógico/Digital como el MAX6675 especializado para termopares tipo K. A través de una interfaz SPI unidireccional

con este módulo será posible conectar fácilmente un termopar a cualquier microcontrolador. Dentro del pequeño circuito que posee este módulo se encuentra la electrónica necesaria para compensar, amplificar, y convertir a digital el voltaje generado por el termopar, lo que hace muy sencilla la tarea de conectar un termopar a un microcontrolador. El único problema es que este circuito solo se consigue en encapsulado SOIC, por lo que no es posible su uso mediante un protoboard. Sin embargo, en este módulo encontramos el MAX6675 con toda la electrónica necesaria y las terminales apropiadas para su fácil su uso. (Factory, geekfactory, 2017).

En la ***Figura 6*** se observa la Tarjeta acondicionador de Termopar tipo K.

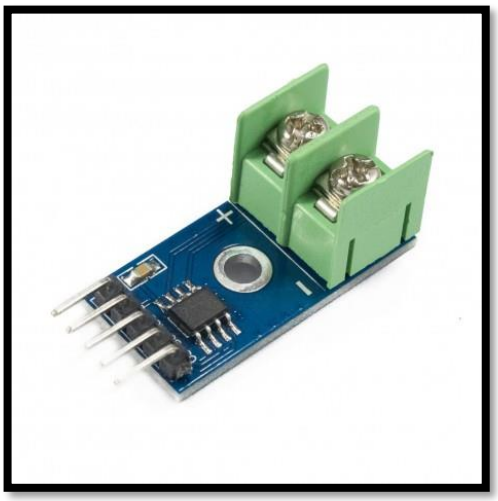


Figura 6. Tarjeta acondicionador de Termopar tipo K (Naylampmechatronics, 2018)

2.8. Termopar tipo K

La unión de dos metales distintos los cuales producen una caída de tensión que va en función de la diferencia de temperatura entre uno de los extremos denominado “punto caliente” o unión caliente o de medida y el otro denominado

"punto frío" o unión fría o de referencia, esa es la característica principal por lo que está formado un termopar.

Este tipo de sensores son ampliamente utilizados en aplicaciones de instrumentación industrial debido principalmente a su bajo costo y su amplio rango de temperaturas. La principal desventaja del termopar es su exactitud, ya que rara vez se consiguen errores menores que 1 grado centígrado. (Factory, geekfactory, 2017).

En la ***Figura 7*** se observa el Termopar tipo K.

Figura 7. Termopar tipo K (Factory, geekfactory, 2017)

2.9. LCD TFT NEXTION 7 PULGADAS

Las soluciones HMI que ofrece Nextion permiten una interfaz de control y visualización entre humanos y máquinas aplicado principalmente al internet de las cosas(IoT) o a su vez a los consumidores de la electrónica.

Nextion se compone de hardware y software, cuando hablamos de hardware nos referimos a una serie de tableros TFT, mientras que el software hace referencia a la programación de dichos tableros.

La comunicación de las TFT es a través de un puerto serie, evitando así el exceso de cableado. El software nos permite el desarrollo de un sin número de aplicaciones, pero con resultados insatisfactorios.

Las funciones arrastrar y soltar ayuda que se pierda tiempo en códigos de programación más extensos, por lo tanto, con el editor WYSIWYG, es muy fácil diseñar una GUI.

Para adaptar proyectos existentes a la familia NEXTION es necesario proporcionar un UART.

El módulo de visualización de pantalla HFT TFT de Nextion 7.0 " con panel táctil resistivo integrado de 4 cables para Arduino DIY NX8048T070 800x480 es un potente HMI de 7.0 ", que es miembro de la familia Nextion. (MACTRONICA, 2019).

En la ***Figura 8*** se visualiza la LCD Touchpad Nextion 7”

Figura 8. LCD TOUCHPAD NEXTION 7” (MACTRONICA, 2019)

2.10. Circuito integrado LM358

Está formado por circuitos independientes que al encontrar dentro del encapsulado que compensan la frecuencia del amplificador operacional y cada uno opera como suplemento de poder que operan a diferentes rangos de voltaje, el drenaje es posible también bajo las operaciones de fuerza independientemente de la magnitud del suministro de voltaje. (ECURED, 2012).

En la ***Figura 9*** se puede observar el integrado LM358.

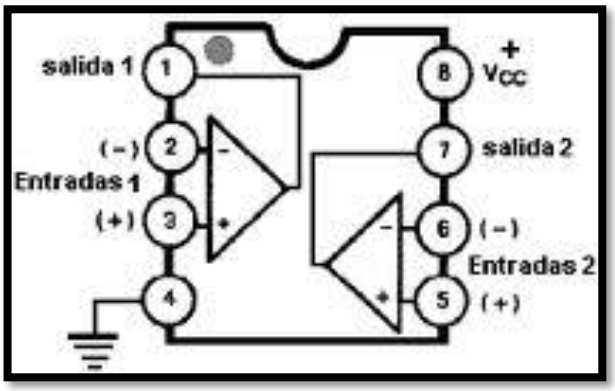


Figura 9. Integrado LM358 (ECURED, 2012)

2.11. Software Proteus Designa Suite

Proteus es uno del software más utilizados porque permite el diseño, simulación y ejecución de proyectos electrónicos en todas sus etapas: construcción de la placa de circuito impreso, depuración de errores, documentación y construcción.

En la ***Figura 10*** se observa un diseño realizado en Proteus.

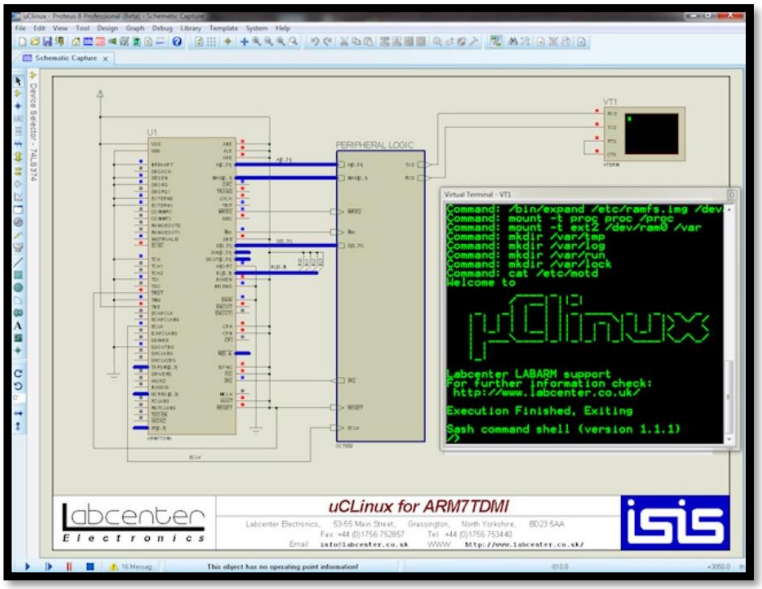


Figura 10. Proteus Design Suite (Electronics, 2020)

2.12. Software Acimut Integración de Sistemas

Según Acimut (2010) Monitoriza es la aplicación SCADA de software libre que está revolucionando la industria, permite controlar cualquier proceso, de forma inmediata, desde su propio puesto de trabajo. Con menos programación ahora se logra conseguir un producto libre que cumpla con los ideales con respecto a la automatización de sistemas y su control.

En la ***Figura 11*** se observa el esquema que utiliza Acimut Monitoriza.

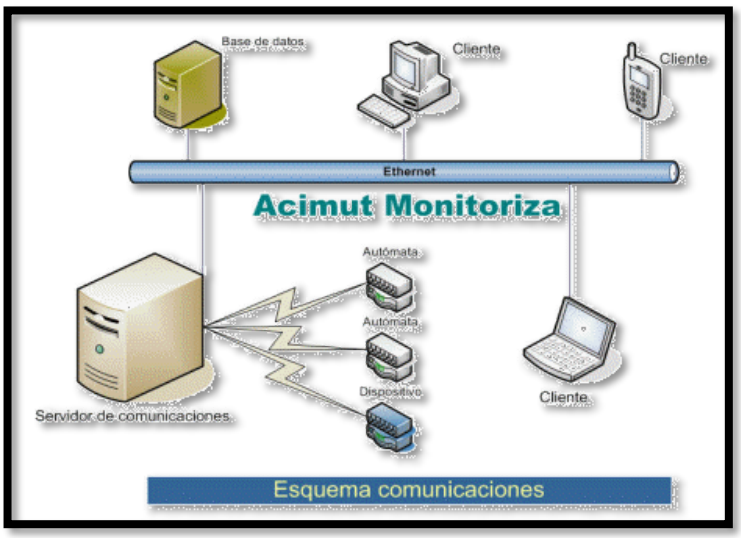


Figura 11. Acimut Monitoriza (Acimut, Acimut, 2010)

2.13. Software AutoCAD 2018

De acuerdo a lo indicado por ALL3dp (2020), AutoCAD es un programa o software de diseño asistido por computadora (CAD) en 2D y modelado 3D; es considerado por muchos como el padre de los programas CAD. Ésta es una herramienta muy variable que cuenta con una amplia gama de características y funciones. Este programa CAD se usa en diversos ámbitos profesionales y de ingeniería entre los que se incluyen: ingeniería, arquitectura, diseño gráfico y administración.

2.14. Software Nextion Editor

Según Nextion Editor (2019) es un software de desarrollo GUI gratuito de interfaz hombre máquina (HMI) para Nextion Basic Series, Enhanced Series y Intelligent Series.

En la ***Figura 12*** se observa una pantalla en Nextion Editor.

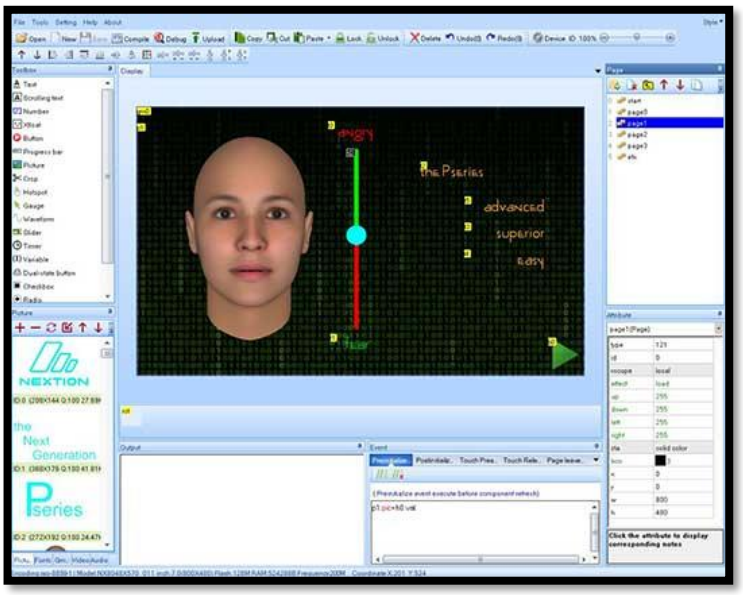


Figura 12. Nextion Editor (Acimut, Acimut, 2010)

3. MARCO METODOLÓGICO

3.1. Metodología de la Investigación

Este proyecto busca satisfacer las necesidades que tienen los clientes de Amper Ecuador S.A. con respecto a la visualización y registro en tiempo real de ciertos parámetros de energía en los tableros de Bypass que alimentan las cargas críticas.

El método que se utilizó para elaborar el proyecto es el inductivo, debido a que se basó en la observación, análisis de instrumentos y programación de software para encontrar la solución más adecuada y que cumpla con las expectativas de los clientes.

3.2. Técnica de Investigación

La técnica que se utilizó para la elaboración del proyecto fue la entrevista, puesto que se necesitó conocer cuáles eran los requerimientos de los clientes con respecto a los parámetros de energía que requerían visualizar en los tableros.

3.3. Procedimiento del proyecto

3.3.1. Diseño de la estructura del prototipo

La estructura donde se encuentra todos los componentes electrónicos fue diseñada mediante el software AutoCAD 2018, éste prototipo está diseñado de tal forma que su manipulación no es difícil, su apariencia es agradable a la vista; además se puede compactar todo el circuito electrónico dentro de la misma.

En la ***Figura 13*** se observa el diseño de la carcasa de prototipo.

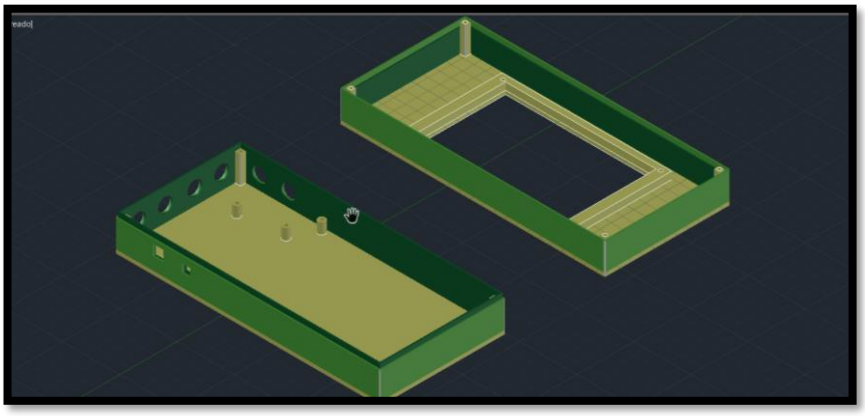


Figura 13. Diseño de carcasa de prototipo

.

3.3.2. Diseño de acondicionador de señal de corriente.

El sensor de corriente que se usó tiene una señal de salida proporcional a la corriente que circula por el cable, por lo que se requiere acondicionar la señal debido a que el procesador solo mide señales de tensión.

La señal de salida del sensor es de -50mA a +50mA, lo apropiado en estos casos es acondicionar ésta señal a un rango de 0-5V, para esto se usó un circuito operacional LM358 que con la electrónica necesaria brinda una señal de voltaje con un rango de +/-1V, el operacional trabaja con la polaridad positiva por lo tanto se tiene una señal de salida de 0-1V, la misma que es procesada por el Arduino.

En la ***Figura 14*** se observa el esquema del acondicionador de señal utilizando LM358 y en la **Tabla 2** se visualiza los componentes que se utilizaron para el circuito acondicionador de señal de corriente.

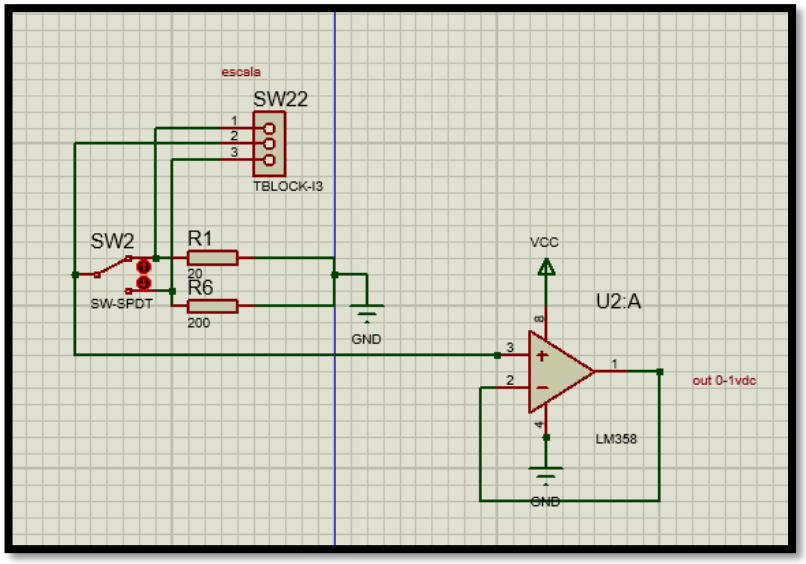


Figura 14. Diseño del acondicionador de señal con operacional LM358

Tabla 2

Lista de Componentes del circuito acondicionador de señal de corriente

Ítems	Cantidad	Descripción	Especificación
1	1	Resistencia	220Ω
2	1	Resistencia	20Ω
3	1	Switch	Dos posiciones
4	1	LM358	Integrado Operacional

3.3.3. Diseño de tarjeta principal del prototipo.

El prototipo está diseñado con varios módulos, los que están listos para adaptarse a procesadores tales como Arduino, con la finalidad de ser reemplazados fácilmente en caso de falla alguna.

Estos módulos están energizados con una fuente común, ya que se realizó el diseño de una tarjeta principal que incorpore todos los sistemas y lleve la señal de estos hacia el controlador principal.

Como se puede ver en la ***Figura 15*** los módulos tienen pines para sus conexiones; debido a esto se usó conectores tipo DIP para alimentarlos y llevar datos hasta la PCB y luego a los pines de entrada y salida del Arduino.

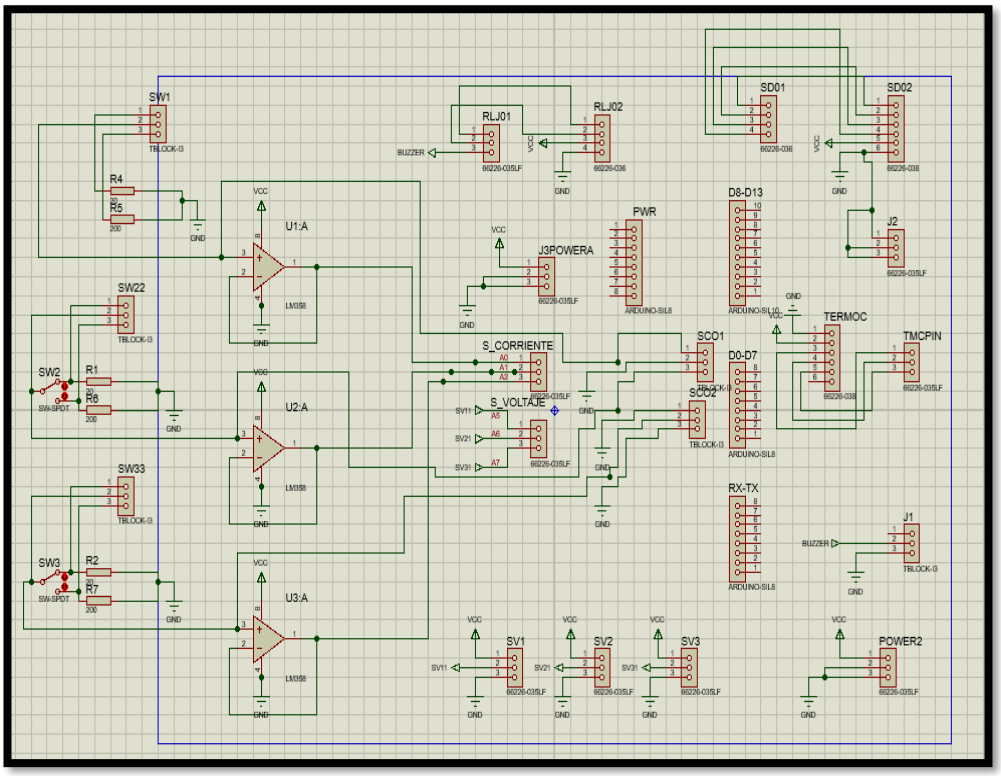


Figura 15. Esquemático de la tarjeta principal.

En la ***Figura 16*** se observa el circuito PCB de la tarjeta principal y en la **Tabla 3** se visualiza la lista de componentes que forman parte de la placa principal de prototipo.

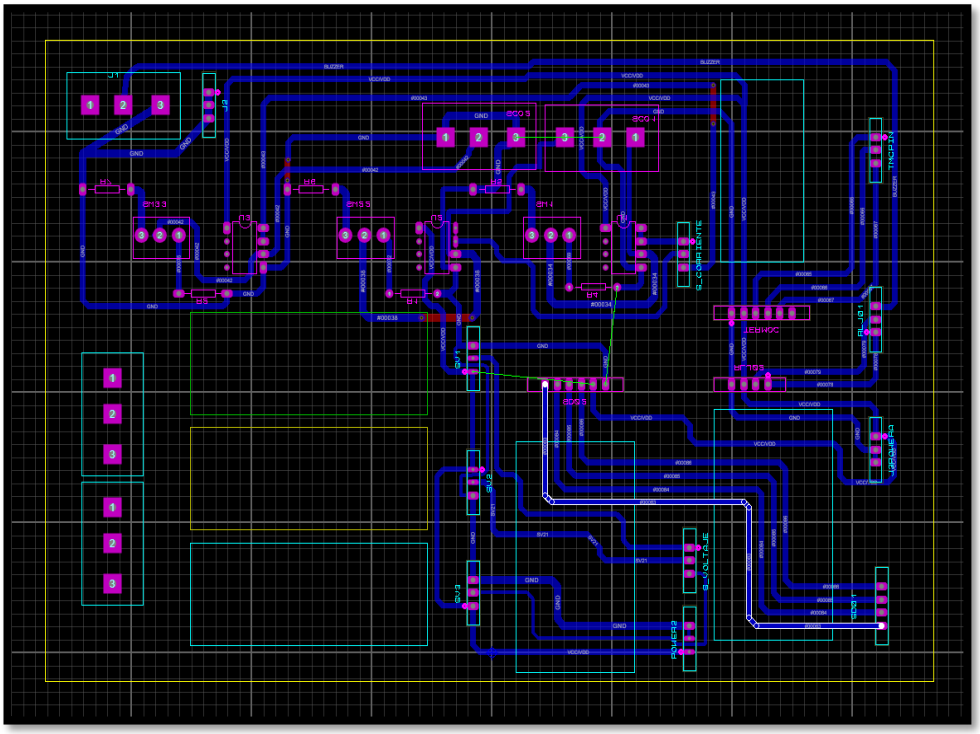


Figura 16. PCB de tarjeta principal de prototipo

Tabla 3
Lista de módulos que conforman la placa principal de prototipo.

Ítems	Cantidad	Descripción	Especificación
1	3	Módulo de sensor de voltaje.	ZMPT101B
2	1	Módulo Lector de SD.	Micro SD
3	1	Módulo RTC.	DS3231
4	1	Módulo acondicionador de Termopar.	Max6675

3.3.4. Diseño de pantallas HMI

Para el diseño de la interfaz Hombre-Máquina se usó un software dedicado a la programación de pantallas Nextion, este software es gratuito y fácil de descargar de la web.

Para cargar el diseño a la pantalla TFT, se debe tener conectada la misma a los pines 1 y 0 del puerto serial, se da click en Upload y el código se carga automáticamente; los datos mostrados en la TFT son procesados mediante código por el controlador Arduino y enviados a través de los pines RX y TX del puerto serial al HMI. La comunicación entre el Arduino mega y el HMI es a través del puerto serial del Arduino por medio de los pines RX y TX.

En la ***Figura 17*** se visualiza la pantalla de diseño del HMI y en la ***Figura 18*** se muestra la pantalla de los parámetros eléctricos.

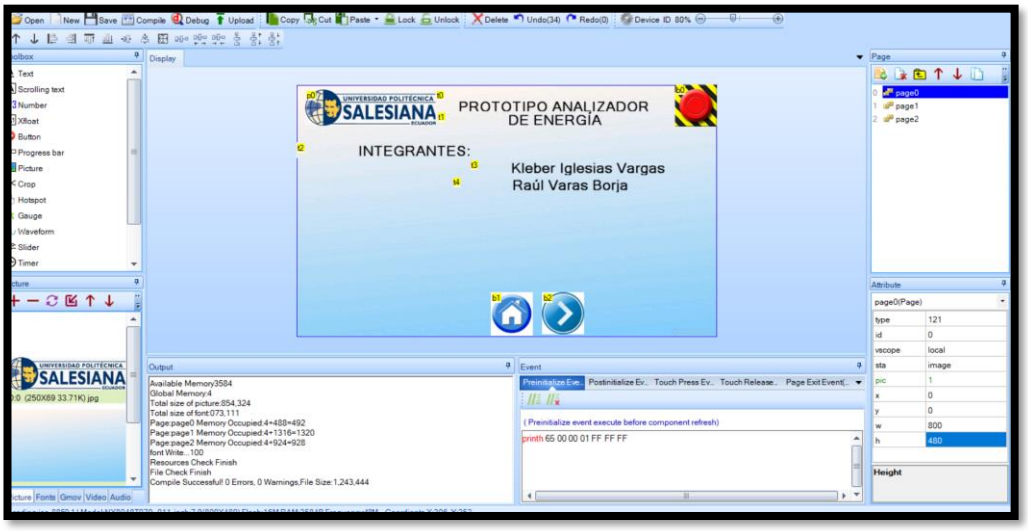


Figura 17. Diseño de HMI en software Nextion Editor.

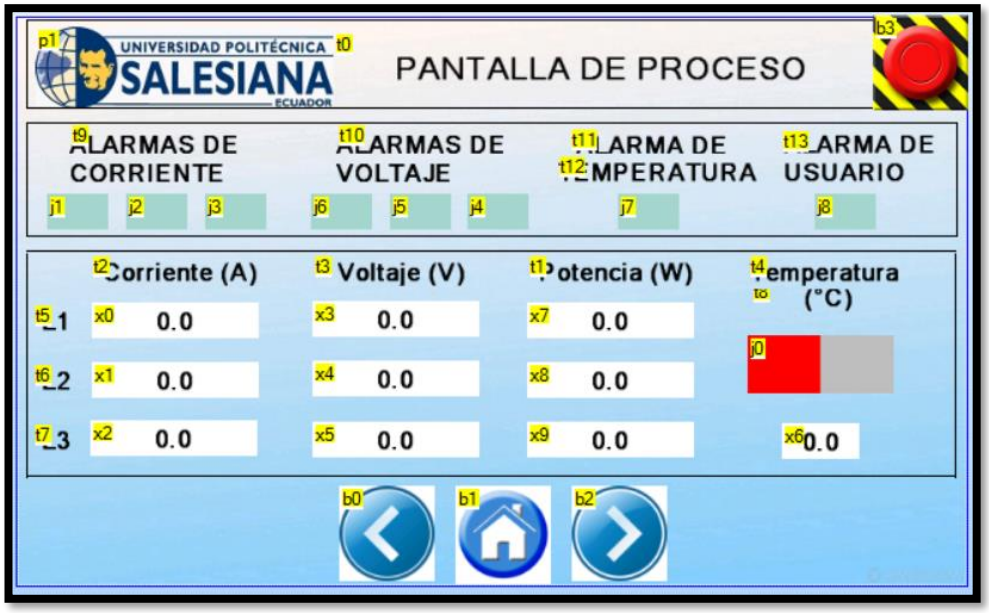


Figura 18. Diseño de pantalla HMI- Parámetros eléctricos.

3.3.5. Diseño de SCADA

Para el SCADA del proyecto se usó el software de licencia libre Acimut Monitoriza, éste sistema se comunica con el Arduino a través del puerto serial y usa el protocolo Modbus RTU configurado a una velocidad de 19200 baudios; los datos recibidos por el SCADA son previamente procesados por el Arduino mega mediante código.

En el sistema SCADA se muestran los datos en tiempo real, así como alarmas predefinidas de corriente, voltaje y temperatura; además se visualizan históricos de datos, los mismos que son guardados en una base de datos programada en SQL para su posterior impresión en PDF en caso de ser requeridos físicamente dichos registros.

En la ***Figura 19*** se visualiza la pantalla de inicio del Sistema SCADA.

Figura 19. Pantalla de Inicio del Sistema SCADA.

3.3.6. Dispositivos para la toma de datos en campo.

La mayoría de los dispositivos medidores de parámetros eléctricos cuentan con accesorios que permiten al usuario la toma de datos de forma segura y fácil. Para éste proyecto se usó lagartos tipo pinza calibre 26, color rojo y negro para medir entre dos fases o entre fase y neutro; es importante mencionar que el dispositivo es capaz de medir hasta tres fases, lo mismo que depende del uso del cliente.

Como se puede observar en la ***Figura 20*** a las pinzas se les adaptó conectores RCA Hembra para que en caso de desgaste puedan ser reemplazados.

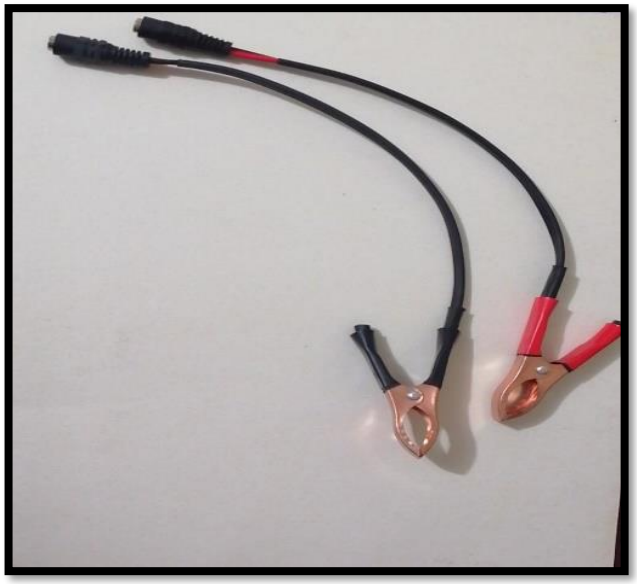


Figura 20. Pinzas tipo lagarto con conector RCA Hembra.

El prototipo cuenta con tres módulos para medir voltaje, por lo tanto, permite realizar mediciones a sistemas trifásicos de hasta 240vac entre líneas. En la ***Figura 21*** se muestra la sonda de voltaje con su respectivo conector JA-143 hembra.

Figura 21. Sonda de Voltaje con conector de 2.15 metros de longitud.

Para la sonda de voltaje se utilizaron los siguientes elementos:

- 1 – Conector JA-143 hembra.
- 1 – Conector JA-143 macho ya integrado en el dispositivo.
- 1 – Pinza tipo lagarto calibre 26 capucha roja.
- 1 – Pinza tipo lagarto calibre 26 capucha negra.
- 2 - Conector RCA hembra.
- 2 - Conector RCA macho.
- 2.15m – cable concéntrico 3x18mm AWG.

En las mediciones de corriente se usó directamente el sensor no invasivo SCT-013 (Ver ***Figura 22);*** el cual abraza al cable que es de calibre máximo 6AWG.

Para conectar el transformador de corriente al prototipo se usó los siguientes elementos:

- 1 – Conector JA-143 hembra.
- 1 – Conector JA-143 macho ya integrado en el dispositivo.

Figura 22. Pinza para medir corriente (90 cm de longitud).

4. RESULTADOS

4.1. Impresión placa para tarjeta electrónica.

Luego de realizar el diseño en AutoCAD 2018, se procedió a la impresión de la placa en baquelita de cobre tal como se visualiza en las ***Figuras 23 y 24.***

Figura 23. Proceso de Impresion Baquelita para placa.

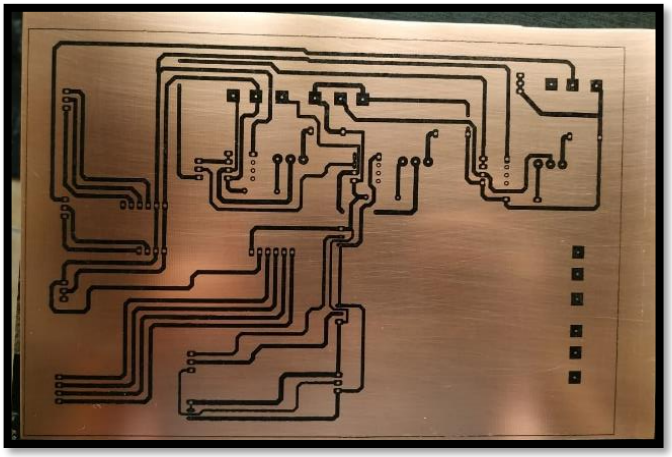


Figura 24. Placa Terminada.

4.2. Montaje de Componentes Electrónicos en placa.

Después del proceso de impresión de la placa en la baquelita de cobre, se procede a colocar los diferentes componentes electrónicos que intervienen en el prototipo.

Como se puede ver ***Figura 25*** los componentes electrónicos quedaron correctamente distribuidos en la respectiva placa.

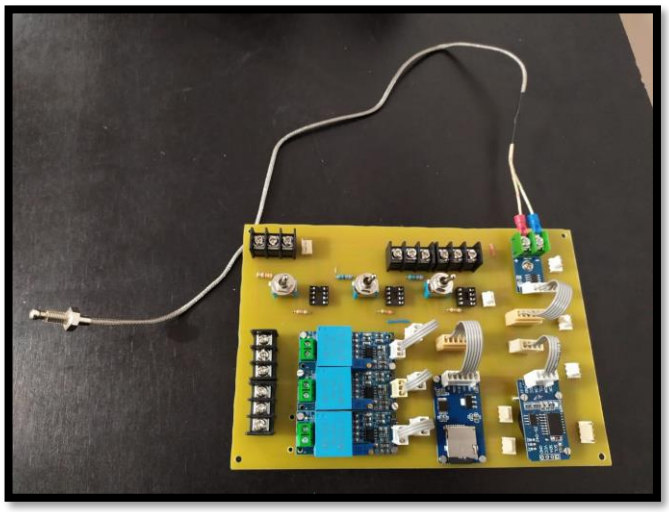


Figura 25. Montaje de componentes electrónicos en placa.

4.3. Enlace entre Controlador y HMI

Se puede apreciar en la ***Figura 26*** la conexión física entre el Controlador Arduino y el HMI, la conexión entre ambos equipos es serial.

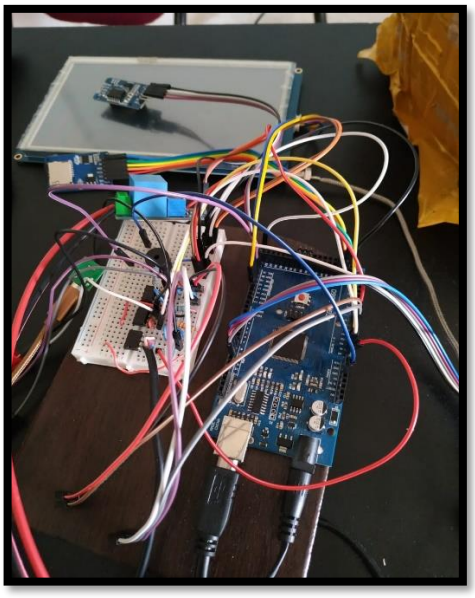


Figura 26. Conexión Arduino y HMI.

4.4. Enlace Controlador y SCADA

El Enlace de controlador y SCADA utiliza el protocolo Modbus, como muestra la ***Figura 27***, el software Acimut gestiona los registros.

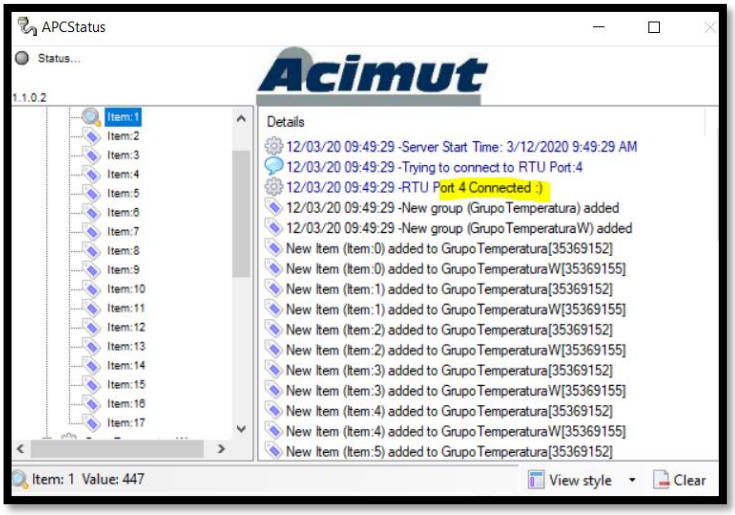


Figura 27. Gestionador de Registros y Conexiones.

En la ***Figura 28*** se muestra el enlace del servidor SCADA MONITORIZA con el controlador, a través del mismo se pueden visualizar las variables configuradas en los registros Modbus.

Figura 28. Conexión servidor SCADA.

4.5. Mediciones físicas de nivel de tensión y corriente.

Para garantizar la correcta operación del prototipo es necesario tomar mediciones iniciales, que permitan saber que la adquisición de datos es confiable.

Como se aprecia en la ***Figura 29*** las mediciones de tensión fueron realizadas con un instrumento de medición marca Truper modelo MUT-202.

Figura 29. Voltaje medido con pinza amperimétrica Truper.

En la ***Figura 30*** se muestra la medición fisica de corriente, utilizando el mismo equipo que se usó para las mediciones de tensión.

Figura 30. Corriente medida con Pinza amperimétrica Truper.

4.6. Operación del Prototipo.

Después del proceso de diseño y ensamblaje de componentes electrónicos, así como la programación del controlador (ver **ANEXO 1**), HMI y SCADA; se visualizaron satisfactoriamente los parámetros de voltaje, corriente, potencia y temperatura.

En la ***Figura 31,*** se observa la operación del prototipo; la tarjeta electrónica enlazada con el HMI y el sistema SCADA.

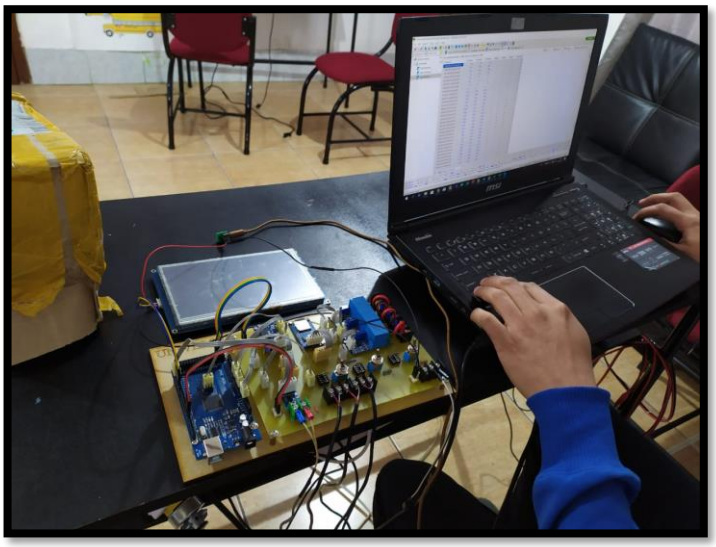


Figura 31. Operación de Prototipo.

Como se aprecia en la ***Figura 32,*** el HMI muestra en su pantalla las variables deseadas por el cliente, con los valores de voltaje, corriente y temperatura.

Para obtener la potencia por línea, se realiza el cálculo por medio de la fórmula matemática $P = V * I$

Figura 32. Puesta en Marcha de HMI

La ***Figura 33*** muestra la pantalla de configuraciones avanzadas de las señales de corriente, tensión y temperatura; lo que permite al operador cambiar los valores en caso de daño de algún sensor, ésta pantalla es muy importante debido a que no es necesario ingresar a la programación para el cambio de valores.

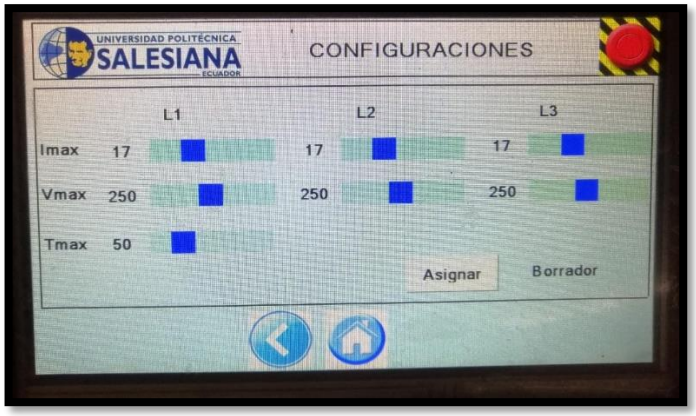


Figura 33. Pantalla de configuración avanzada de señales

En la ***Figura 34*** se puede visualizar las variables en el sistema SCADA de todos los registros Modbus previamente configurados.

Figura 34. Parámetros de SCADA.

En la ***Figura 35,*** se aprecia la pantalla de históricos del sistema SCADA; la cual permite acceder a la base de datos en formato de lista.

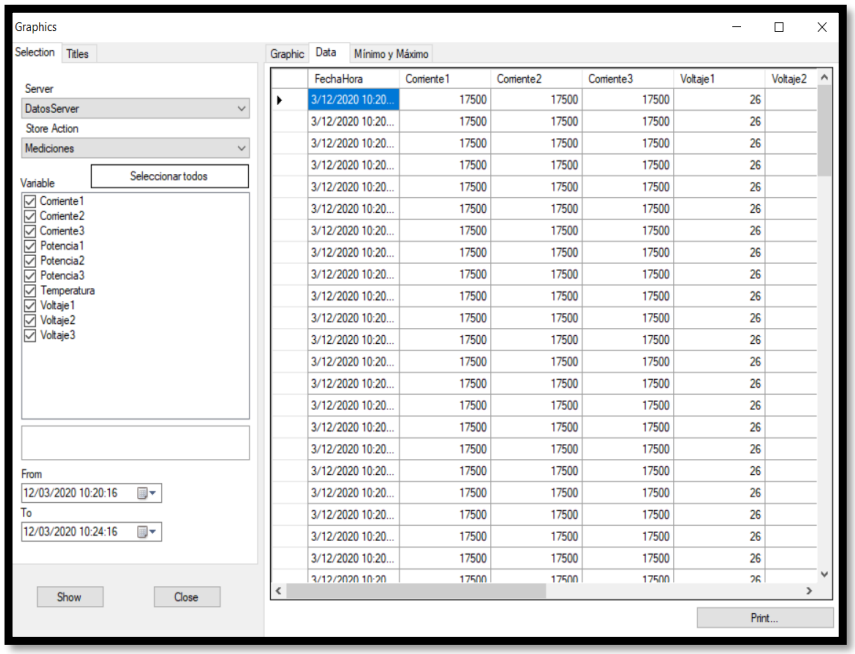


Figura 35. Pantalla de Históricos en formato de lista.

La ***Figura 36,*** muestra la pantalla de históricos del sistema SCADA; la cual permite acceder a la base de datos formato de registros en el tiempo (Tendencia), para esto es necesario seleccionar un rango de tiempo.

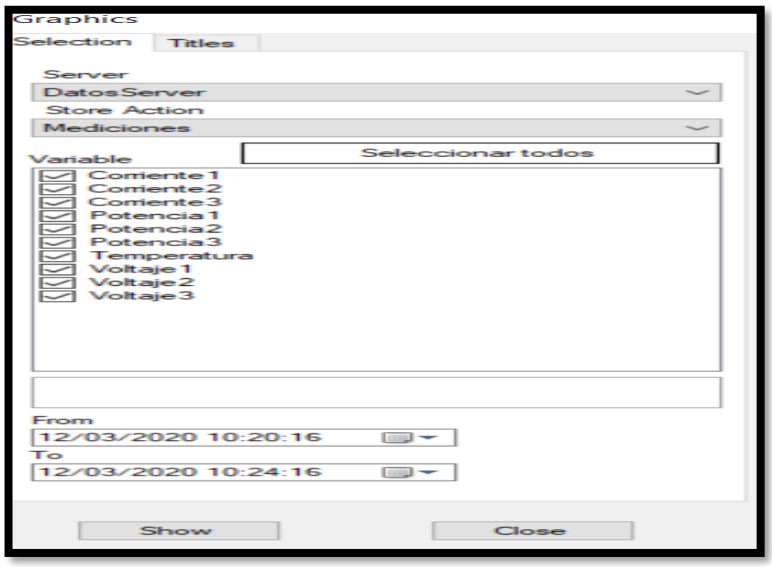


Figura 36. Pantalla históricos en formato Tendencia.

Como se puede observar en la ***Figura 37,*** la pantalla de alarmas del sistema SCADA permite acceder a las alarmas que se generan en el sistema, en caso de alguna falla que se presente.

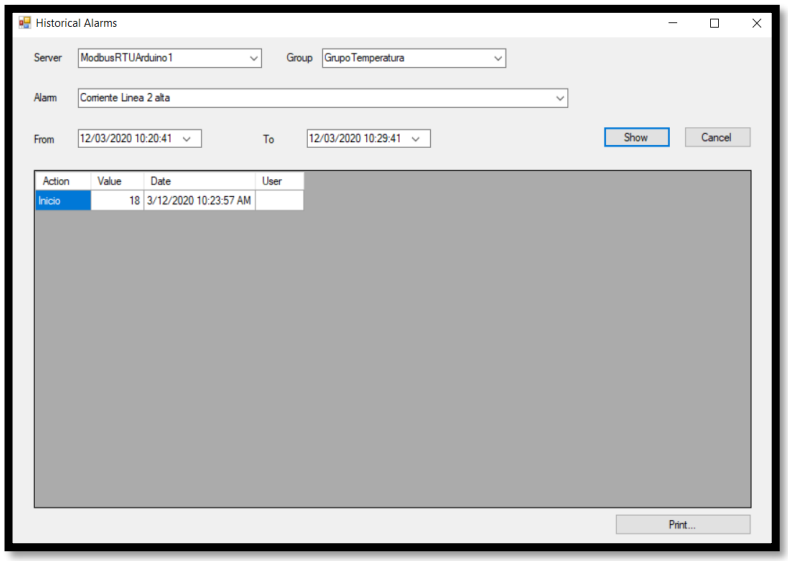


Figura 37. Pantalla de Alarmas.

La ***Figura 38,*** muestra la opción de imprimir el reporte en formato PDF, la interfaz es la misma que se usa en Windows, facilitando al operador la interacción con el software.

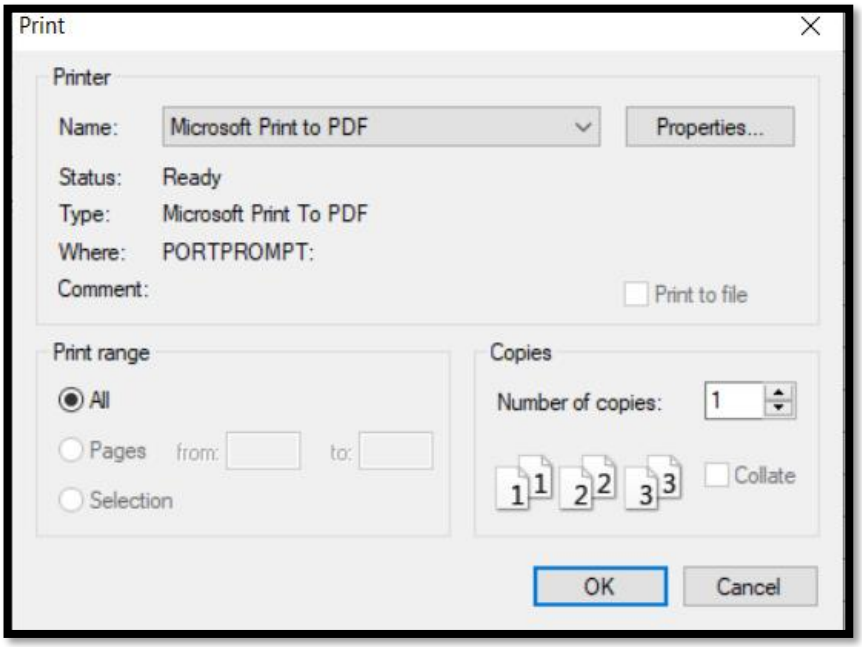
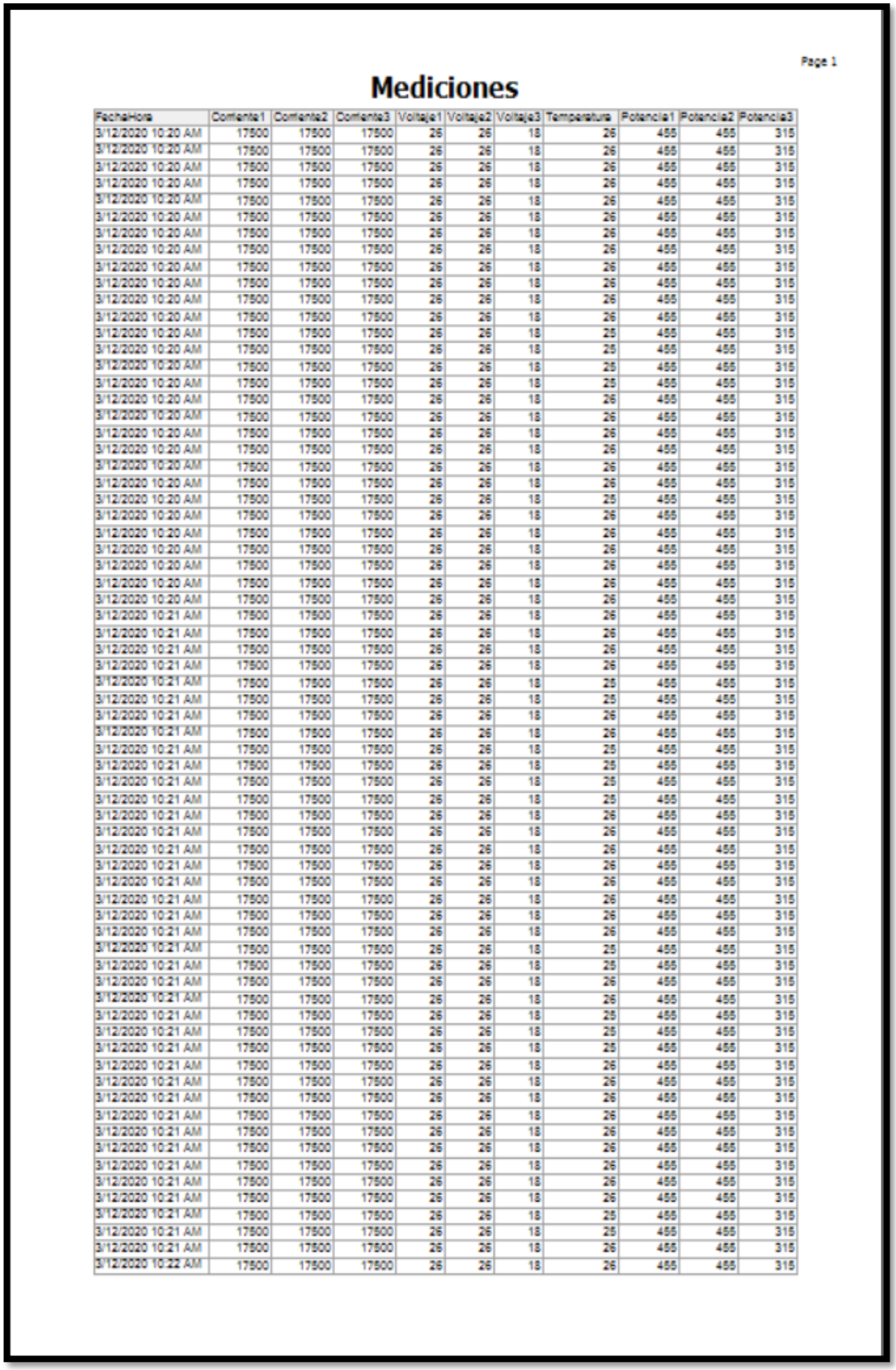


Figura 38.- Selección de Impresión de Informe.

En la ***Figura 39*** se observa el resultado de la impresión de un informe del sistema SCADA, el cual permite tener la información necesaria para el cliente



Page 1

Mediciones

FechaHora	Corriente1	Corriente2	Corriente3	Voltaje1	Voltaje2	Voltaje3	Temperatura	Potencia1	Potencia2	Potencia3
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:20 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	25	455	455	315
3/12/2020 10:21 AM	17500	17500	17500	26	26	18	26	455	455	315
3/12/2020 10:22 AM	17500	17500	17500	26	26	18	26	455	455	315

Figura 39. Reporte de parámetros registrados desde fecha solicitada en ventana de históricos.

5. ANALISIS DE RESULTADOS

Después de realizar pruebas al sistema se obtuvieron varios resultados, los mismos que han sido comparados con equipos certificados para el posterior análisis.

5.1. Medición de corriente

En la **Tabla 4** se detallan los resultados de la medición de corriente con el sistema implementado en comparación con las mediciones que se realizaron con el Fluke. Las pruebas se realizaron con tres cargas diferentes.

Con las mediciones tanto del Fluke como del sistema implementado se procedió a calcular el error; los resultados del porcentaje de error varían entre 1.1% y 3.7%, los mismos que se encuentran dentro de los parámetros de error aceptable.

Tabla 4
Resultados de medición de corriente entre dispositivo Fluke y Prototipo

Medido con Fluke			**Medido con Prototipo**			**% Error**		
L1	L2	L3	L1	L2	L3	L1	L2	L3
0	0	0	0	0	0	0	0	0
5,9	5,9	5,9	5,7	5,8	6	3,38983051	1,69491525	1,69491525
10,8	10,8	10,8	10,4	11,2	11,1	3,7037037	3,7037037	2,77777778
16,7	16,7	16,7	16,4	16,9	17	1,79640719	1,19760479	1,79640719

5.2. Medición de voltaje

De la misma forma que con la corriente, se realizó un comparativo de los valores obtenidos entre las mediciones de voltaje del fluke con respecto al prototipo pudiéndose observar en la **Tabla 5** que los porcentajes de error que se encuentran entre voltajes de 60 a 250 voltios alternos no supera el 1 %, mientras que para voltajes menores a 60 voltios su porcentaje de error es mucho mayor.

Es importante mencionar que el prototipo está enfocado a cargas monofásicas que operan entre los 100 a 240 vac.

Tabla 5

Resultados de medición de voltaje entre dispositivo Fluke y Prototipo

Medición con Fluke	Medición con Prototipo	% de Error
48,3	50	3,52
55,3	57	3,07
63,2	64	1,27
69,3	70	1,01
119,7	119	0,58
130,4	130	0,31
200,3	200	0,15
221,4	221	0,18
230	231	0,43
241	240	0,41
249	250	0,40
253	253	0,00

De acuerdo a las dos comparaciones realizadas tanto de voltaje como de corriente se puede observar que los equipos instalados son confiables ya que el porcentaje de error se encuentra dentro de los límites establecidos.

6. CONCLUSIONES

Se puede determinar que el prototipo a pesar de tener un costo muy por debajo de los equipos existentes e incluso más económicos que muchos productos chinos que existen en el mercado, no deja de ser un producto muy versátil y confiable.

Éste prototipo fue diseñado para pequeñas aplicaciones de cargas no mayores de 2KVA, pero igual de críticas e importantes como las grandes cargas; se ajusta perfectamente en el exterior o interior de tableros eléctricos de alimentación, distribución o de Bypass que energicen algún UPS u otro tipo de carga como por ejemplo un Plotter. En cualquiera de estos casos permite al cliente tener información del sistema y de esta manera realizar correcciones de forma oportuna en caso que se presente alguna falla.

De acuerdo a las pruebas de funcionamiento, el sistema es capaz de medir datos como: Tensión, Corriente, Potencia y temperatura cada segundo y almacenarlos en una base de datos desde una PC, mediante un SCADA ACIMUT; así como también almacena los datos en una tarjeta de memoria interna. Esto brinda al usuario dos alternativas de visualización para posterior análisis, la una que es el informe en PDF que se puede imprimir desde el SCADA ACIMUT y la otra opción es descargar los datos almacenados directamente desde la memoria SD y abrirlos en un archivo Excel y de esta manera hacer un análisis exhaustivo.

El sistema cuenta con una pantalla TFT TOUCH de 7”, para una mejor visualización de los parámetros eléctricos ya mencionados anteriormente. Por medio de esta pantalla se calibra rangos de tolerancia, es decir, permite configurar umbrales de Tensión, Corriente y temperatura; además si los parámetros sobrepasan los límites permitidos inmediatamente se activa una alarma sonora indicando que los niveles de Tensión, corriente y temperatura están fuera del rango. Las alarmas se almacenan en la base de datos del SCADA, lo que permite al usuario u operario de la carga actuar inmediatamente ante el evento y así tomar las correcciones respectivas.

El producto tiene tres sensores de corriente de núcleo abierto lo que significa que son sensores no invasivos, por lo que su instalación hacia las acometidas que alimentan la carga es práctica y sencilla; es decir no requiere de desconexión de la carga para realizar la adquisición o censado de datos. Además, cuenta con tres pares de pinzas tipo lagarto, las cuales permiten medir tres puntos diferentes de Tensión L-N o L-L, aunque no sea parte del alcance del proyecto puede medir la Tensión y la Corriente de un sistema trifásico.

El error obtenido en las mediciones de corriente (entre 1.1 y 3.7%) entre el sistema y el Fluke está dentro de los parámetros aceptables, a pesar de que para realizar la lectura de corriente se tuvo que diseñar un circuito acondicionador y aparte un pequeño código para obtener la corriente RMS final.

Para la lectura de la señal analógica de voltaje se usó el sensor ZMPT101B, en el código se utilizó algoritmos basados en funciones polinomiales para calibrar el sensor;

estos algoritmos permitieron tomar medidas precisas, lo que se puede comprobar con el porcentaje de error de los voltajes de 50 a 250Vac que no supera el 1%.

7. RECOMENDACIONES

Diseñar estructuras robustas con una protección IP alta, para proteger al sistema de los ambientes rústicos.

Usar equipos que tengan una capacidad de procesamiento mucho mayor tales como raspberry.

Extender los estudios realizados en esta investigación para poder tener comunicación vía Ethernet y así unirnos al IoT.

Colocar sensores de mayor precisión para tener un porcentaje de error más bajo en la lectura de datos.

Ampliar los rangos de lectura de parámetros, para tener menos limitaciones a la hora de realizar mediciones en industrias.

Extender la capacidad de almacenamiento de datos en la microSD para poder realizar estudios más a fondo sobre consumos eléctricos.

8. REFERENCIAS BIBLIOGRÁFICAS

Acimut. (2010). *Acimut*. Obtenido de https://www.acimut.com/monitoriza/monitorizaintroduccion.htm

ALL3dp. (Enero de 2020). *ALL3dp*. Obtenido de https://all3dp.com/es/1/programa-autocad-gratis-alternativas-autocad/

ARDUINO. (2020). Obtenido de https://www.arduino.cc/en/Guide/Introduction

ARDUINO. (2020). *ARDUINO MEGA*. Obtenido de https://arduino.cl/arduino-mega-2560/

ECURED. (2012). *Ecured*. Obtenido de https://www.ecured.cu/Circuito_integrado_LM358

Electronics, L. (2020). *RobotShop*. Obtenido de https://www.robotshop.com/us/es/software-proteus-vsm-para-arm-cortex-m3.html

Electropeak. (2019). *Electropeak electronics*. Obtenido de https://electropeak.com/ac-voltage-transformer-module-sensor-zmpt101b

Factory, G. (2017). *geekfactory*. Obtenido de https://www.geekfactory.mx/tienda/modulos-para-desarrollo/ds3231-modulo-reloj-en-tiempo-real/

Factory, G. (2017). *geekfactory*. Obtenido de https://www.geekfactory.mx/tienda/modulos-para-desarrollo/max6675-modulo-interfaz-termopar/

LLamas, L. (16 de Octubre de 2016). *luisllamas*. Obtenido de https://www.luisllamas.es/tarjeta-micro-sd-arduino/

Llamas, L. (24 de enero de 2017). *luisllamas*. Obtenido de https://www.luisllamas.es/arduino-sensor-corriente-sct-013/

MACTRONICA. (2019). *Mactronica*. Obtenido de https://www.mactronica.com.co/lcd-tft-touch-pulgadas-nextion-114179243xJM

MECHATRONICS, N. (2018). *NAYLAMP MECHATRONICS*. Obtenido de https://naylampmechatronics.com/sensores-corriente-voltaje/393-transformador-de-voltaje-ac-zmpt101b.html

Naylampmechatronics. (2018). *naylampmechatroncis*. Obtenido de https://naylampmechatronics.com/sensores-temperatura-y-humedad/331-transmisor-para-termocupla-max6675.html

NextionEditor. (2019). *NextionTech*. Obtenido de https://nextion.tech/nextion-editor/

Proteus, H. (2015). *HuborProteus*. Obtenido de http://www.hubor-proteus.com/proteus-pcb/proteus-pcb/2-proteus.html

YHDC. (s.f.). *MCIELECTRONICS.* Obtenido de https://www.mcielectronics.cl/website_MCI/static/documents/Datasheet_SCT013.pdf

9. ANEXOS

Anexo 1: Código de sistema.

```
#include "RTClib.h" //libreria para modulo reloj
#include <SD.h> //libreria para modulo SD

#include "Nextion.h" //libreria para pantalla Nextion
#include "max6675.h" //Libreria para Termopar
#define PIN_CS 53 //definimos pin para datos de SD
#define PIN_CS_CONVENCIONAL 10 //definimos pin para datos de SD
#define SIRENA 19 //definimos pin para buzzer

//inicializacion de variables para Sensor volt1
int voltagePin = A5;
float x = 0;
float V = 0;
//inicializacion de variables para sensor volt2
int voltagePin2 = A6;
float Vb = 0;
float V2 = 0;
//inicializacion de variables para sensor volt3
int voltagePin3 = A7;
float x3 = 0;
float V3 = 0;
//inicializacion de variables para sensores de corriente
int Sensor = 0; //A0
int Sensor1 = 0; //A1
int Sensor2 = 0; //A2

//Definimos pines max6675 termopar
int ktcSO = 8;
int ktcCS = 9;
int ktcCLK = 10;
MAX6675 ktc(ktcCLK, ktcCS, ktcSO);

File myFile;  //TARJETA SD
RTC_DS3231 rtc; //LLamado a segmento


char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
String extension = ".csv";
String file_name = "data";
char date_str[16];
char time_str[16];

void configure_mb_slave(long baud, char parity, char txenpin);
//Asignacion a nextion hmi
NexNumber i1 = NexNumber(1, 15, "x0");
NexNumber i2 = NexNumber(1, 10, "x1");
NexNumber i3 = NexNumber(1, 11, "x2");
NexNumber v1 = NexNumber(1, 12, "x3");
NexNumber v2 = NexNumber(1, 17, "x4");
```

```
NexNumber v3 = NexNumber(1, 18, "x5");
NexNumber p1 = NexNumber(1, 20, "x7");
NexNumber p2 = NexNumber(1, 21, "x8");
NexNumber p3 = NexNumber(1, 22, "x9");
NexNumber t1 = NexNumber(1, 14, "x6");

NexProgressBar tbar = NexProgressBar(1, 13, "j0");
NexProgressBar ii1 = NexProgressBar(1, 25, "j1");
NexProgressBar ii2 = NexProgressBar(1, 26, "j2");
NexProgressBar ii3 = NexProgressBar(1, 27, "j3");
NexProgressBar iv1 = NexProgressBar(1, 30, "j6");
NexProgressBar iv2 = NexProgressBar(1, 29, "j5");
NexProgressBar iv3 = NexProgressBar(1, 28, "j4");
NexProgressBar itemp = NexProgressBar(1, 31, "j7");
NexProgressBar iuser = NexProgressBar(1, 49, "j8");

NexNumber ai1 = NexNumber(2, 7, "n0");
NexNumber ai2 = NexNumber(2, 9, "n1");
NexNumber ai3 = NexNumber(2, 11, "n2");
NexNumber av1 = NexNumber(2, 12, "n3");
NexNumber av2 = NexNumber(2, 14, "n4");
NexNumber av3 = NexNumber(2, 16, "n5");
NexNumber at1 = NexNumber(2, 24, "n6");

NexSlider ah0 = NexSlider(2, 6, "h0");
NexSlider ah1 = NexSlider(2, 8, "h1");
NexSlider ah2 = NexSlider(2, 10, "h2");
NexSlider ah3 = NexSlider(2, 13, "h3");
NexSlider ah4 = NexSlider(2, 15, "h4");
NexSlider ah5 = NexSlider(2, 17, "h5");
NexSlider ah6 = NexSlider(2, 25, "h6");

NexPage page0 = NexPage(0, 0, "page0");
NexPage page1 = NexPage(1, 0, "page1");
NexPage page2 = NexPage(2, 0, "page2");

NexButton bem1 = NexButton(0, 5, "b0");
NexButton bem2 = NexButton(1, 23, "b3");
NexButton bem3 = NexButton(2, 3, "b2");

NexButton b3 = NexButton(2, 34, "b3");
NexText t7 = NexText(2, 35, "t7");

NexTouch *nex_listen_list[] = {
  &b3,
  &bem1,
  &bem2,
  &bem3,
  &page0,
  &page1,
  &page2,
  NULL
};
```

```
//Adignacion de valores maximos por default en HMI
uint32_t i1max = 17;
uint32_t i2max = 17;
uint32_t i3max = 17;
uint32_t v1max = 250;
uint32_t v2max = 250;
uint32_t v3max = 250;
uint32_t tempmax = 50;

bool fi1 = false;
bool fi2 = false;
bool fi3 = false;
bool fv1 = false;
bool fv2 = false;
bool fv3 = false;
bool ft1 = false;

bool fbuzzer = false;
double tbuzzer = 0;
double t_sound = 1000;

int CurrentPage = 0;

void setConfig(void *ptr){
  ai1.getValue(&i1max);
  delay(10);
  ai2.getValue(&i2max);
  delay(10);
  ai3.getValue(&i3max);
  delay(10);
  av1.getValue(&v1max);
  delay(10);
  av2.getValue(&v2max);
  delay(10);
  av3.getValue(&v3max);
  delay(10);
  at1.getValue(&tempmax);
  delay(10);
  t7.setText("Asignado");
  initIndicators();
}

void emergencyUserCall(){
  page1.show();
  iuser.setValue(100);
  fbuzzer = true;
  tbuzzer = millis();
}

void callEmergencyPage0(void *ptr){
  emergencyUserCall();
}
```

```
void callEmergencyPage1(void *ptr){
  emergencyUserCall();
}
void callEmergencyPage2(void *ptr){
  emergencyUserCall();
}

void initIndicators(){
  fi1 = false;
  fi2 = false;
  fi3 = false;
  fv1 = false;
  fv2 = false;
  fv3 = false;
  ft1 = false;
}

// Page change event:
void page0PushCallback(void *ptr)  // If page 0 is loaded on the display, the following is
going to execute:
{
  CurrentPage = 0;  // Set variable as 0 so from now on arduino knows page 0 is loaded on
the display
  initIndicators();
}  // End of press event


// Page change event:
void page1PushCallback(void *ptr)  // If page 1 is loaded on the display, the following is
going to execute:
{
  CurrentPage = 1;  // Set variable as 1 so from now on arduino knows page 1 is loaded on
the display
}  // End of press event


// Page change event:
void page2PushCallback(void *ptr)  // If page 2 is loaded on the display, the following is
going to execute:
{
  CurrentPage = 2;  // Set variable as 2 so from now on arduino knows page 2 is loaded on
the display
  ah0.setValue(i1max);
  ah1.setValue(i2max);
  ah2.setValue(i3max);
  ah3.setValue(v1max);
  ah4.setValue(v2max);
  ah5.setValue(v3max);
  ah6.setValue(tempmax);
}  // End of press event


int update_mb_slave(unsigned char slave, int *regs,
```

```
unsigned int regs_size);

/* Modbus RTU common parameters, the Master MUST use the same parameters */
enum {
      COMM_BPS = 19200, /* baud rate */
      MB_SLAVE = 1, /* modbus slave id */
      PARITY = 'n' /* even parity */
};

//COMIENZA NUESTRO PROGRAMA
/* slave registers */
enum {
      MB_CO1, //Registro de lectura de corriente linea 1
      MB_CO2, //Registro de lectura de corriente linea 2
      MB_CO3, //Registro de lectura de corriente linea 3
      MB_VL1, //Registro de lectura de voltaje linea 1
      MB_VL2, //Registro de lectura de voltaje linea 2
      MB_VL3, //Registro de lectura de voltaje linea 3
      MB_POT1,//Registro de lectura de Potencia en linea 1
      MB_POT2,//Registro de lectura de Potencia en linea 2
      MB_POT3,//Registro de lectura de Potencia en linea 3
      MB_TEMP, //Registro de lectura de temperatura
      hora, //Registro de lectura de hora
      MB_ALM2, //Registro de lectura de alarma para
      MB_ALM3, //Registro de lectura de alarma para
      MB_ALM4, //Registro de lectura de alarma para
      MB_ALM5, //Registro de lectura de alarma para
      MB_ALM6, //Registro de lectura de alarma para
      MB_ALM7, //Registro de lectura de alarma para
      MB_ALM8, //Registro de lectura de alarma para
      MB_REGS   /* TOTAL DE REGISTROS DEL ARDUINO, NO BORRAR */
};

int regs[MB_REGS];//NO BORRAR EL ARREGLO

int sensorValue;

void setup()
{
  delay(3000); // wait for console opening
 // analogReference(INTERNAL1V1); //solo Arduino Mega

  if (! rtc.begin()) {
    while (1);
  }
  if (rtc.lostPower()) {
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    }
  configure_mb_slave(COMM_BPS, PARITY, 0);
  pinMode(13, OUTPUT);
  pinMode(SIRENA, OUTPUT);

  nexInit();
```

```
  b3.attachPush(setConfig, &b3);
  bem1.attachPush(callEmergencyPage0, &bem1);
  bem2.attachPush(callEmergencyPage1, &bem2);
  bem3.attachPush(callEmergencyPage2, &bem3);

  page0.attachPush(page0PushCallback);  // Page press event
  page1.attachPush(page1PushCallback);  // Page press event
  page2.attachPush(page2PushCallback);  // Page press event
  SD_ready();
}

void loop(){

  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

float Va = 0;
float Vsquare = 0;
float Vsquareb = 0;
float x2 = 0;
float Vc = 0;
float Vsquarec = 0;

                    //Calculo de voltaje usando teoria del tercer polinomio
for (int i=0;i<1000;i++){
  x = (analogRead(voltagePin)- 512);
  V = (0.00000412*x*x*x - 0.000857*x*x + 2.675*x - 3.198);
  Va = V*V;
  Vsquare = Vsquare + Va;
  delay(0.1);
}

for (int i=0;i<1000;i++){
  x2 = (analogRead(voltagePin2)-512);
  V2 = (0.00000412*x2*x2*x2 - 0.000857*x2*x2 + 2.675*x2 - 3.198);
  Vb = V2*V2;
  Vsquareb = Vsquareb + Vb;
  delay(0.1);
}


for (int i=0;i<1000;i++){
  x3 = (analogRead(voltagePin3)-512);
  V3 = (0.00000412*x3*x3*x3 - 0.000857*x3*x3 + 2.675*x3 - 3.198);
  Vc = V3*V3;
  Vsquarec = Vsquarec + Vc;
  delay(0.1);
}

float Vrms1 = sqrt(Vsquare/1000);

if(Vrms1 >= 70 & Vrms1 < 230)
{Vrms1=Vrms1 + 2;}
```

```
if(Vrms1 >= 0 & Vrms1 < 14)
{Vrms1= 1;}
if(Vrms1 >= 14 & Vrms1 < 20)
{Vrms1= Vrms1 - 3;}
if(Vrms1 >= 20 & Vrms1 < 30)
{Vrms1= Vrms1 - 1.5;}

float Vrms2 = sqrt(Vsquareb/1000);
if(Vrms2 >= 70 & Vrms2 < 230)
{Vrms2=Vrms2 + 2;}
if(Vrms2 >= 0 & Vrms2 < 14)
{Vrms2= 1;}
if(Vrms2 >= 14 & Vrms2 < 20)
{Vrms2= Vrms1 - 3;}
if(Vrms2 >= 20 & Vrms2 < 30)
{Vrms2= Vrms2 - 1.5;}

float Vrms3 = sqrt(Vsquarec/1000);
if(Vrms3 >= 70 & Vrms3 < 230)
{Vrms3=Vrms3 + 2;}
if(Vrms3 >= 0 & Vrms3 < 14)
{Vrms3= 1;}
if(Vrms3 >= 14 & Vrms3 < 20)
{Vrms3= Vrms3 - 3;}
if(Vrms3 >= 20 & Vrms3 < 30)
{Vrms3= Vrms3 - 1.5;}

//Cambio de variable para uso en scada
float Irms1=get_corriente(); //Corriente eficaz (A)
float Irms2=get_corriente2(); //Corriente eficaz (A)
float Irms3=get_corriente3(); //Corriente eficaz (A)


//Cambio de variable para uso en scada
float supplyVoltage=Vrms1;
float supplyVoltage2=Vrms2;
float supplyVoltage3=Vrms3;
float potencia1=Irms1*Vrms1;
float potencia2=Irms2*Vrms2;
float potencia3=Irms3*Vrms3;

//Asignacion de variales a registros de escada
  regs[MB_CO1] = Irms1*1000;
  regs[MB_CO2] = Irms2*1000;
  regs[MB_CO3] = Irms3*1000;
  regs[MB_VL1] = supplyVoltage;
  regs[MB_VL2] = supplyVoltage2;
  regs[MB_VL3] = supplyVoltage3;
  regs[MB_TEMP] = ktc.readCelsius()- 2;
  regs[MB_POT1] = potencia1;
  regs[MB_POT2] = potencia2;
  regs[MB_POT3] = potencia3;
```

```
  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

  i1.setValue(regs[MB_CO1]*0.01);
  i2.setValue(regs[MB_CO2]*0.01);
  i3.setValue(regs[MB_CO3]*0.01);
  v1.setValue(regs[MB_VL1]*10);
  v2.setValue(regs[MB_VL2]*10);
  v3.setValue(regs[MB_VL3]*10);

  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

  p1.setValue(potencia1*10);
  p2.setValue(potencia2*10);
  p3.setValue(potencia3*10);
  t1.setValue(regs[MB_TEMP]*10);
  tbar.setValue(regs[MB_TEMP]);

  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

 if(Irms1 > i1max or Irms2 > i2max or Irms3 > i3max or supplyVoltage > v1max or
supplyVoltage2 > v2max or supplyVoltage3 > v3max or regs[MB_TEMP] > tempmax){
  if(!fi1 and !fi2 and !fi2 and !fv1 and !fv2 and !fv3 and !ft1){
    page1.show();
    fbuzzer = true;
    tbuzzer = millis();
  }
 }

 if(Irms1 > i1max){
  if(!fi1){
    fi1 = true;
    ii1.setValue(100);
  }
 }else{
  if(fi1){
    fi1 = false;
    ii1.setValue(0);
  }
 }
 if(Irms2 > i2max){
  if(!fi2){
    fi2 = true;
    ii2.setValue(100);
  }
 }else{
  if(fi2){
    fi2 = false;
    ii2.setValue(0);
```

```
  }
}
if(Irms3 > i3max){
  if(!fi3){
    fi3 = true;
    ii3.setValue(100);
  }
}else{
  if(fi3){
    fi3 = false;
    ii3.setValue(0);
  }
}

update_mb_slave(MB_SLAVE, regs, MB_REGS);
nexLoop(nex_listen_list);

if(supplyVoltage > v1max){
  if(!fv1){
    fv1 = true;
    iv1.setValue(100);
  }
}else{
  if(fv1){
    fv1 = false;
    iv1.setValue(0);
  }
}
if(supplyVoltage2 > v2max){
  if(!fv2){
    fv2 = true;
    iv2.setValue(100);
  }
}else{
  if(fv2){
    fv2 = false;
    iv2.setValue(0);
  }
}
if(supplyVoltage3 > v3max){
  if(!fv3){
    fv3 = true;
    iv3.setValue(100);
  }
}else{
  if(fv3){
    fv3 = false;
    iv3.setValue(0);
  }
}
if(regs[MB_TEMP] > tempmax){
  if(!ft1){
    ft1 = true;
```

```
      itemp.setValue(100);
    }
  }else{
    if(ft1){
      ft1 = false;
      itemp.setValue(0);
    }
  }
  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

  if(fbuzzer){
    alarmbuzzer();
  }

  DateTime now = rtc.now();
  sprintf(date_str, "%02d/%02d/%02d", now.day(), now.month(), now.year());
  sprintf(time_str, "%02d:%02d:%02d",  now.hour(), now.minute(), now.second());

  update_mb_slave(MB_SLAVE, regs, MB_REGS);
  nexLoop(nex_listen_list);

  save_data();

}

//Calculo de corriente en linea 1

float get_corriente()
{
  float voltajeSensor;
  float corriente=0;
  float Sumatoria=0;
  long tiempo=millis();
  int N=0;
  while(millis()-tiempo<500)//Duración 0.5 segundos(Aprox. 30 ciclos de 60Hz)
  {
    voltajeSensor = analogRead(A0) * (5.58/ 1023.0);////voltaje del sensor
    corriente=voltajeSensor*60.0; //corriente=VoltajeSensor*(30A/1V)
    Sumatoria=Sumatoria+sq(corriente);//Sumatoria de Cuadrados
    N=N+1;
    delay(1);
  }
  Sumatoria=Sumatoria*2;//Para compensar los cuadrados de los semiciclos negativos.
  corriente=sqrt((Sumatoria)/N); //ecuación del RMS
if(corriente >=1 & corriente <2.5)
{corriente =corriente-0.3;}
if(corriente >=4 & corriente <4.5)
{corriente =corriente - 0.5;}
if(corriente >=4.5 & corriente <5.1)
{corriente =corriente + 1;}
if(corriente >=18)
{corriente =corriente - 1;}
```

```
return(corriente);}

//Calculo de corriente en linea 2
float get_corriente2()
{
  float voltajeSensor2;
  float corriente2=0;
  float SumatoriaU=0;
  long tiempo=millis();
  int NU=0;
  while(millis()-tiempo<500)//Duración 0.5 segundos(Aprox. 30 ciclos de 60Hz)
  {
    voltajeSensor2 = analogRead(A1) * (5.58 / 1023.0);////voltaje del sensor
    corriente2=voltajeSensor2*60.0; //corriente=VoltajeSensor*(30A/1V)
    SumatoriaU=SumatoriaU+sq(corriente2);//Sumatoria de Cuadrados
    NU=NU+1;
    delay(1);
  }
  SumatoriaU=SumatoriaU*2;//Para compensar los cuadrados de los semiciclos negativos.
  corriente2=sqrt((SumatoriaU)/NU); //ecuación del RMS

if(corriente2 >=1 & corriente2 <2.5)
{corriente2 =corriente2-0.3;}
if(corriente2 >=4 & corriente2 <4.5)
{corriente2 =corriente2 - 0.5;}
if(corriente2 >=4.5 & corriente2 <5.1)
{corriente2 =corriente2 + 1;}
if(corriente2 >=18)
{corriente2 =corriente2- 1;}
return(corriente2);
}

//Calculo de corriente en linea 3
float get_corriente3()
{
  float voltajeSensor3;
  float corriente3=0;
  float Sumatoria3=0;
  long tiempo=millis();
  int NV=0;
  while(millis()-tiempo<500)//Duración 0.5 segundos(Aprox. 30 ciclos de 60Hz)
  {
    voltajeSensor3 = analogRead(A2) * (5.58/ 1023.0);////voltaje del sensor
    corriente3=voltajeSensor3*60.0; //corriente=VoltajeSensor*(30A/1V)
    Sumatoria3=Sumatoria3+sq(corriente3);//Sumatoria de Cuadrados
    NV=NV+1;
    delay(1);
  }
  Sumatoria3=Sumatoria3*2;//Para compensar los cuadrados de los semiciclos negativos.
  corriente3=sqrt((Sumatoria3)/NV); //ecuación del RMS
if(corriente3 >=1 & corriente3 <2.5)
{corriente3 =corriente3-0.3;}
if(corriente3 >=4 & corriente3 <4.5)
```

```
{corriente3 =corriente3 - 0.5;}
if(corriente3 >=4.5 & corriente3 <5.1)
{corriente3 =corriente3 + 1;}
if(corriente3 >=18)
{corriente3 =corriente3 - 1;}
return(corriente3);
}



void alarmbuzzer(){
  if((millis() - tbuzzer) < t_sound){
    digitalWrite(SIRENA, HIGH);
  }else{
    digitalWrite(SIRENA, LOW);
    fbuzzer = false;
  }
}

bool SD_ready(){
  bool ret = false;
  pinMode(PIN_CS_CONVENCIONAL, OUTPUT);
  if (!SD.begin(PIN_CS)) {
    ret = false;
  }
  pinMode(PIN_CS_CONVENCIONAL, OUTPUT);

  if (!SD.begin(PIN_CS)) {
    ret = false;
  }
    if(!SD.exists(file_name+extension))
    {
      myFile = SD.open(file_name+extension, FILE_WRITE);
      if (myFile) {
        myFile.println("Fecha,Hora,Corriente 1,Corriente 2,Corriente 3,Voltaje 1,Voltaje
2,Voltaje 3,Temperatura");
        myFile.close();
        ret = true;
      } else {
        ret = false;
      }
    }else{
      ret = true;
    }
  return ret;
}

void save_data(){
  pinMode(PIN_CS_CONVENCIONAL, OUTPUT);
  if (!SD.begin(PIN_CS)) {
  }
    if(!SD.exists(file_name+extension))
    {
```

```
    myFile = SD.open(file_name+extension, FILE_WRITE);
    if (myFile) {
      myFile.println("Fecha,Hora,Corriente 1,Corriente 2,Corriente 3,Voltaje 1,Voltaje
2,Voltaje 3,Temperatura");
      myFile.close();
    } else {
      return false;
    }
   }
  myFile = SD.open(file_name+extension, FILE_WRITE);
   if (myFile) {
    myFile.print(date_str);
    myFile.print(",");
    myFile.print(time_str);
    myFile.print(",");
    myFile.print(regs[MB_CO1]);
    myFile.print(",");
    myFile.print(regs[MB_CO2]);
    myFile.print(",");
    myFile.print(regs[MB_CO3]);
    myFile.print(",");
    myFile.print(regs[MB_VL1]);
    myFile.print(",");
    myFile.print(regs[MB_VL2]);
    myFile.print(",");
    myFile.print(regs[MB_VL3]);
    myFile.print(",");
    myFile.println(regs[MB_TEMP]);
    myFile.close();
   }
}

//FINALIZA NUESTRO PROGRAMA

// ******* CONFIGURACION DE COMUNICACION CON MONITORIZA NO
TOCAR**************
/**************************************************************************
**
 * BEGIN MODBUS RTU SLAVE FUNCTIONS

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

/* global variables */
unsigned int Txenpin = 0;       /* Enable transmission pin, used on RS485 networks */


/* enum of supported modbus function codes. If you implement a new one, put its function
code here ! */
enum {
      FC_READ_REGS  = 0x03,   //Read contiguous block of holding register
      FC_WRITE_REG  = 0x06,   //Write single holding register
      FC_WRITE_REGS = 0x10,    //Write block of contiguous registers
```

```
        FC_READ_DEV_ID = 0x2B  //Read device information
};

/* supported functions. If you implement a new one, put its function code into this array! */
const unsigned char fsupported[] = { FC_READ_REGS, FC_WRITE_REG,
FC_WRITE_REGS, FC_READ_DEV_ID };

/* constants */
enum {
        MAX_READ_REGS = 0x7D,
        MAX_WRITE_REGS = 0x7B,
        MAX_MESSAGE_LENGTH = 256
};


enum {
        RESPONSE_SIZE = 6,
        EXCEPTION_SIZE = 3,
        CHECKSUM_SIZE = 2
};

/* exceptions code */
enum {
        NO_REPLY = -1,
        EXC_FUNC_CODE = 1,
        EXC_ADDR_RANGE = 2,
        EXC_REGS_QUANT = 3,
        EXC_EXECUTE = 4
};

/* positions inside the query/response array */
enum {
        SLAVE = 0,
        FUNC,
        START_H,
        START_L,
        REGS_H,
        REGS_L,
        BYTE_CNT
};


/*
CRC

 INPUTS:
  buf   ->  Array containing message to be sent to controller.
  start ->  Start of loop in crc counter, usually 0.
  cnt   ->  Amount of bytes in message being sent to controller/
 OUTPUTS:
  temp  ->  Returns crc byte for message.
 COMMENTS:
  This routine calculates the crc high and low byte of a message.
```

Note that this crc is only used for Modbus, not Modbus+ etc.

```
**********************************************************************
**/

unsigned int crc(unsigned char *buf, unsigned char start,
unsigned char cnt)
{
	unsigned char i, j;
	unsigned temp, temp2, flag;

	temp = 0xFFFF;

	for (i = start; i < cnt; i++) {
		temp = temp ^ buf[i];

		for (j = 1; j <= 8; j++) {
			flag = temp & 0x0001;
			temp = temp >> 1;
			if (flag)
				temp = temp ^ 0xA001;
		}
	}

	/* Reverse byte order. */
	temp2 = temp >> 8;
	temp = (temp << 8) | temp2;
	temp &= 0xFFFF;

	return (temp);
}




/**********************************************************************
 *
 *  The following functions construct the required query into
 *  a modbus query packet.
 *
 **********************************************************************/

/*
 * Start of the packet of a read_holding_register response
 */
void build_read_packet(unsigned char slave, unsigned char function,
unsigned char count, unsigned char *packet)
{
	packet[SLAVE] = slave;
	packet[FUNC] = function;
	packet[2] = count * 2;
}
```

```
/*
 * Start of the packet of a preset_multiple_register response
 */
void build_write_packet(unsigned char slave, unsigned char function,
unsigned int start_addr,
unsigned char count,
unsigned char *packet)
{
	packet[SLAVE] = slave;
	packet[FUNC] = function;
	packet[START_H] = start_addr >> 8;
	packet[START_L] = start_addr & 0x00ff;
	packet[REGS_H] = 0x00;
	packet[REGS_L] = count;
}

/*
 * Start of the packet of a write_single_register response
 */
void build_write_single_packet(unsigned char slave, unsigned char function,
	unsigned int write_addr, unsigned int reg_val, unsigned char* packet)
{
	packet[SLAVE] = slave;
	packet[FUNC] = function;
	packet[START_H] = write_addr >> 8;
	packet[START_L] = write_addr & 0x00ff;
	packet[REGS_H] = reg_val >> 8;
	packet[REGS_L] = reg_val & 0x00ff;
}


/*
 * Start of the packet of an exception response
 */
void build_error_packet(unsigned char slave, unsigned char function,
unsigned char exception, unsigned char *packet)
{
	packet[SLAVE] = slave;
	packet[FUNC] = function + 0x80;
	packet[2] = exception;
}


/*****************************************************************************
 *
 * modbus_query( packet, length)
 *
 * Function to add a checksum to the end of a packet.
 * Please note that the packet array must be at least 2 fields longer than
 * string_length.

*****************************************************************************/
```

```
void modbus_reply(unsigned char *packet, unsigned char string_length)
{
	int temp_crc;

	temp_crc = crc(packet, 0, string_length);
	packet[string_length] = temp_crc >> 8;
	string_length++;
	packet[string_length] = temp_crc & 0x00FF;
}



/*****************************************************************************
 *
 * send_reply( query_string, query_length )
 *
 * Function to send a reply to a modbus master.
 * Returns: total number of characters sent
 *****************************************************************************/

int send_reply(unsigned char *query, unsigned char string_length)
{
	unsigned char i;

	if (Txenpin > 1) { // set MAX485 to speak mode
		UCSR0A=UCSR0A |(1 << TXC0);
		digitalWrite( Txenpin, HIGH);
		delay(1);
	}

	modbus_reply(query, string_length);
	string_length += 2;

	for (i = 0; i < string_length; i++) {
		Serial.write(query[i]);
	}

	if (Txenpin > 1) {// set MAX485 to listen mode
		while (!(UCSR0A & (1 << TXC0)));
		digitalWrite( Txenpin, LOW);
	}

	return i;	/* it does not mean that the write was succesful, though */
}

/*****************************************************************************
 *
 *  receive_request( array_for_data )
 *
 * Function to monitor for a request from the modbus master.
 *
 * Returns: Total number of characters received if OK
 * 0 if there is no request
```

```
 * A negative error code on failure
 ******************************************************************************/

int receive_request(unsigned char *received_string)
{
      int bytes_received = 0;

      /* FIXME: does Serial.available wait 1.5T or 3.5T before exiting the loop? */
      while (Serial.available()) {
            received_string[bytes_received] = Serial.read();
            bytes_received++;
            if (bytes_received >= MAX_MESSAGE_LENGTH)
                  return NO_REPLY;  /* port error */
      }

      return (bytes_received);
}


/******************************************************************************
 *
 *  modbus_request(slave_id, request_data_array)
 *
 * Function to the correct request is returned and that the checksum
 * is correct.
 *
 * Returns: string_length if OK
 *    0 if failed
 *    Less than 0 for exception errors
 *
 *  Note: All functions used for sending or receiving data via
 *        modbus return these return values.
 *
 ******************************************************************************/

int modbus_request(unsigned char slave, unsigned char *data)
{
      int response_length;
      unsigned int crc_calc = 0;
      unsigned int crc_received = 0;
      unsigned char recv_crc_hi;
      unsigned char recv_crc_lo;

      response_length = receive_request(data);

      if (response_length > 0) {
            crc_calc = crc(data, 0, response_length - 2);
            recv_crc_hi = (unsigned) data[response_length - 2];
            recv_crc_lo = (unsigned) data[response_length - 1];
            crc_received = data[response_length - 2];
            crc_received = (unsigned) crc_received << 8;
            crc_received =
                  crc_received | (unsigned) data[response_length - 1];
```

```
            /********** check CRC of response **********/
            if (crc_calc != crc_received) {
                return NO_REPLY;
            }

            /* check for slave id */
            if (slave != data[SLAVE]) {
                return NO_REPLY;
            }
        }
        return (response_length);
}

/*****************************************************************************
 *
 *  validate_request(request_data_array, request_length, available_regs)
 *
 * Function to check that the request can be processed by the slave.
 *
 * Returns: 0 if OK
 *    A negative exception code on error
 *
 *****************************************************************************/

int validate_request(unsigned char *data, unsigned char length,
unsigned int regs_size)
{
        int i, fcnt = 0;
        unsigned int regs_num = 0;
        unsigned int start_addr = 0;
        unsigned char max_regs_num;

        /* check function code */
        for (i = 0; i < sizeof(fsupported); i++) {
            if (fsupported[i] == data[FUNC]) {
                fcnt = 1;
                break;
            }
        }
        if (0 == fcnt)
            return EXC_FUNC_CODE;

        if (FC_WRITE_REG == data[FUNC]) {
            /* For function write single reg, this is the target reg.*/
            regs_num = ((int) data[START_H] << 8) + (int) data[START_L];
            if (regs_num >= regs_size)
                return EXC_ADDR_RANGE;
            return 0;
        }

        if (FC_READ_DEV_ID == data[FUNC]) {
            /* For function device info.*/
```

```
            if (data[3] != 1) /* Only Basic device identification */
              return EXC_REGS_QUANT;
            return 0;
        }

        /* For functions read/write regs, this is the range. */
        regs_num = ((int) data[REGS_H] << 8) + (int) data[REGS_L];

        /* check quantity of registers */
        if (FC_READ_REGS == data[FUNC])
            max_regs_num = MAX_READ_REGS;
        else if (FC_WRITE_REGS == data[FUNC])
            max_regs_num = MAX_WRITE_REGS;

        if ((regs_num < 1) || (regs_num > max_regs_num))
            return EXC_REGS_QUANT;

        /* check registers range, start address is 0 */
        start_addr = ((int) data[START_H] << 8) + (int) data[START_L];
        if ((start_addr + regs_num) > regs_size)
            return EXC_ADDR_RANGE;

        return 0;     /* OK, no exception */
}



/**************************************************************************
 *
 *  write_regs(first_register, data_array, registers_array)
 *
 *  writes into the slave's holding registers the data in query,
 * starting at start_addr.
 *
 * Returns:   the number of registers written
 **************************************************************************/

int write_regs(unsigned int start_addr, unsigned char *query, int *regs)
{
        int temp;
        unsigned int i;

        for (i = 0; i < query[REGS_L]; i++) {
            /* shift reg hi_byte to temp */
            temp = (int) query[(BYTE_CNT + 1) + i * 2] << 8;
            /* OR with lo_byte          */
            temp = temp | (int) query[(BYTE_CNT + 2) + i * 2];

            regs[start_addr + i] = temp;
        }
        return i;
}
```

```
/**************************************************************************
 *
 *  preset_multiple_registers(slave_id, first_register, number_of_registers,
 * data_array, registers_array)
 *
 *  Write the data from an array into the holding registers of the slave.
 *
 **************************************************************************/

int preset_multiple_registers(unsigned char slave,
unsigned int start_addr,
unsigned char count,
unsigned char *query,
int *regs)
{
	unsigned char function = FC_WRITE_REGS; /* Preset Multiple Registers */
	int status = 0;
	unsigned char packet[RESPONSE_SIZE + CHECKSUM_SIZE];

	build_write_packet(slave, function, start_addr, count, packet);

	if (write_regs(start_addr, query, regs)) {
		status = send_reply(packet, RESPONSE_SIZE);
	}

	return (status);
}


/**************************************************************************
 *
 * write_single_register(slave_id, write_addr, data_array, registers_array)
 *
 * Write a single int val into a single holding register of the slave.
 *
 **************************************************************************/

int write_single_register(unsigned char slave,
	unsigned int write_addr, unsigned char *query, int *regs)
{
	unsigned char function = FC_WRITE_REG; /* Function: Write Single Register */
	int status = 0;
	unsigned int reg_val;
	unsigned char packet[RESPONSE_SIZE + CHECKSUM_SIZE];

	reg_val = query[REGS_H] << 8 | query[REGS_L];
	build_write_single_packet(slave, function, write_addr, reg_val, packet);
	regs[write_addr] = (int) reg_val;
/*
	written.start_addr=write_addr;
	written.num_regs=1;
*/
	status = send_reply(packet, RESPONSE_SIZE);
```

```
	return (status);
}


/*****************************************************************************
 *
 *  read_holding_registers(slave_id, first_register, number_of_registers,
 * registers_array)
 *
 * reads the slave's holdings registers and sends them to the Modbus master
 *
 *****************************************************************************/

int read_holding_registers(unsigned char slave, unsigned int start_addr,

unsigned char reg_count, int *regs)
{
	unsigned char function = 0x03;  /* Function 03: Read Holding Registers */
	int packet_size = 3;
	int status;
	unsigned int i;
	unsigned char packet[MAX_MESSAGE_LENGTH];

	build_read_packet(slave, function, reg_count, packet);

	for (i = start_addr; i < (start_addr + (unsigned int) reg_count);
	 i++) {
		packet[packet_size] = regs[i] >> 8;
		packet_size++;
		packet[packet_size] = regs[i] & 0x00FF;
		packet_size++;
	}

	status = send_reply(packet, packet_size);

	return (status);
}

/*****************************************************************************
 *
 *  read_device_id(slave_id, query , length)
 *
 * reads the Device identification and sends them to the Modbus master
 *
 *****************************************************************************/

int read_device_id(unsigned char slave, unsigned char *query, unsigned char lenght)
{
	unsigned char function = 0x2B;  /* Function 2B: Read Deice identification */
	unsigned char MEIType = 0x0E;   /* MEI Type */
	unsigned char ReadDeviceIDCode = 0x01; /* Read device id code */
	unsigned char ConformityLevel = 0x01; /* Conformity level */
```

```
        unsigned char MoreFollows = 0x0; /* More Follows */
        unsigned char NextObjectID = 0x0; /* Next Object id */
        unsigned char NumObjects = 0x03; /* Number of objects */
        int packet_size = 3;
        int status;
        unsigned int i;
        unsigned char packet[MAX_MESSAGE_LENGTH];

        packet[SLAVE] = slave;
        packet[FUNC] = function;
        packet[2] = MEIType;
        packet[3] = ReadDeviceIDCode;
        packet[4] = ConformityLevel;
        packet[5] = MoreFollows;
        packet[6] = NextObjectID;

        unsigned int StartObject = 7;
        packet[StartObject] = NumObjects;
/* Object ID 0 */
        String company ="ARDUINO";
        int length=company.length();
        packet[StartObject+1] = 0;  /* Object id = 0 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = company.charAt(i);
        }
        StartObject = StartObject + 2 + length;
/* Object ID 1 */
        String product ="MEGA2560";
        length=product.length();
        packet[StartObject+1] = 1;  /* Object id = 1 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = product.charAt(i);
        }
        StartObject = StartObject + 2 + length;
/* Object ID 2 */
        String ver ="V1";
        length=ver.length();
        packet[StartObject+1] = 2;  /* Object id = 2 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = ver.charAt(i);
        }
        StartObject = StartObject + 2 + length + 1;

        status = send_reply(packet, StartObject);

        return (status);
}


void configure_mb_slave(long baud, char parity, char txenpin)
```

```
{
	Serial.begin(baud);

	switch (parity) {
	case 'e': // 8E1
		UCSR0C |= ((1<<UPM01) | (1<<UCSZ01) | (1<<UCSZ00));
		//	UCSR0C &= ~((1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
		break;
	case 'o': // 8O1
		UCSR0C |= ((1<<UPM01) | (1<<UPM00) | (1<<UCSZ01) | (1<<UCSZ00));
		//	UCSR0C &= ~((1<<UCSZ02) | (1<<USBS0));
		break;
	case 'n': // 8N1
		UCSR0C |= ((1<<UCSZ01) | (1<<UCSZ00));
		//	UCSR0C &= ~((1<<UPM01) | (1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
		break;
	default:
		break;
	}

	if (txenpin > 1) { // pin 0 & pin 1 are reserved for RX/TX
		Txenpin = txenpin; /* set global variable */
		pinMode(Txenpin, OUTPUT);
		digitalWrite(Txenpin, LOW);
	}

	return;
}

/*
 * update_mb_slave(slave_id, holding_regs_array, number_of_regs)
 *
 * checks if there is any valid request from the modbus master. If there is,
 * performs the action requested
 */

unsigned long Nowdt = 0;
unsigned int lastBytesReceived;
const unsigned long T35 = 5;

int update_mb_slave(unsigned char slave, int *regs,
unsigned int regs_size)
{
	unsigned char query[MAX_MESSAGE_LENGTH];
	unsigned char errpacket[EXCEPTION_SIZE + CHECKSUM_SIZE];
	unsigned int start_addr;
	int exception;
	int length = Serial.available();
	unsigned long now = millis();
	unsigned char MEIType = 0x0E;   /* MEI Type for device identification */

	if (length == 0) {
		lastBytesReceived = 0;
```

```
        return 0;
}

if (lastBytesReceived != length) {
        lastBytesReceived = length;
        Nowdt = now + T35;
        return 0;
}
if (now < Nowdt)
        return 0;

lastBytesReceived = 0;

length = modbus_request(slave, query);
if (length < 1)
        return length;


exception = validate_request(query, length, regs_size);
if (exception) {
                build_error_packet(slave, query[FUNC], exception,
                errpacket);
                send_reply(errpacket, EXCEPTION_SIZE);
                return (exception);
}


start_addr = ((int) query[START_H] << 8) +
             (int) query[START_L];
switch (query[FUNC]) {
        case FC_READ_REGS:
                return read_holding_registers(slave,
                start_addr,
                query[REGS_L],
                regs);
        break;
        case FC_WRITE_REGS:
                return preset_multiple_registers(slave,
                start_addr,
                query[REGS_L],
                query,
                regs);
        break;
        case FC_WRITE_REG:
                write_single_register(slave,
                start_addr,
                query,
                regs);
        break;
        case FC_READ_DEV_ID:
                if (query[START_H]==MEIType) {
                   return read_device_id(slave, query, length);
                }
```

```
                break;
            }
        }
```