Desarrollo De Una Red MODBUS RTU Con Raspberry.

Jose Fernando Mejía Jaramillo

Ingeniería Mecatrónica

Director del trabajo de grado:

Juan Gonzalo Ardila Marín

INSTITUTO TECNOLÓGICO METROPOLITANO

Fecha

RESUMEN

DIMETRON S.A.S es una empresa dedicada a la implementación de la tecnología a los procesos industriales, con servicios de automatización industrial, eficiencia energética, montajes eléctricos y proyectos especializados en minería. Están a la vanguardia de los grandes retos del siglo XXI y la nueva revolución industrial.

Dentro de DIMETRON S.A.S se determina crear una nueva área de Innovación y Desarrollo, para ofertar nuevos servicios y para esto se empezó elaborando un algoritmo que comunicara todos los sensores, actuadores y controladores en la etapa de final de un proceso de clasificación industrial, además accediera igualmente a las variables energéticas de toda la planta; de igual manera clasificara y ordenara esta información para posteriormente ser enviada a una base de datos no relacional. En la elaboración se resolvieron algunos interrogantes importantes, como cuál sería el dispositivo a implementar y qué protocolo de comunicación implementar.

RECONOCIMIENTOS

Tengo mucho que agradecer a muchas personas y no quisiera olvidar a ninguna por esa razón primero que todo agradezco a todas las personas que de una u otra manera me ayudaron a crecer y aprender, compañeros, amigos, profesores y sobre todo a mi familia a todos ellos gracias, pero a la persona que quiero darle todas mis gratitudes es a mi querido abuelo ya que gracias a él y su ejemplo me encuentro hoy alcanzando esta meta muchas gracias, también quiero agradecer a mi asesor Juan Gonzalo Ardila Marín por la ayuda y por sus lecciones.

TABLA DE CONTENIDO

TABLA DE IMAGENES

INDICE DE TABLAS

1. INTRODUCCIÓN

DIMETRON S.A.S es una empresa dedicada a la implementación de la tecnología a los procesos industriales, con servicios de automatización industrial, eficiencia energética, montajes eléctricos y proyectos especializados en minería. Están a la vanguardia de los grandes retos del siglo XXI y la nueva revolución industrial.

Dentro de DIMETRON S.A.S se determina crear una nueva área de Innovación y Desarrollo, para ampliar su oferta de servicios e implementar su nueva idea de negocio y así entrar a competir en el área de toma y análisis de datos en el sector energético.

Se trabajó en el desarrollo de un algoritmo que permitiera comunicar todos los sensores, actuadores y controladores en la etapa de final de un proceso de clasificación industrial, además acceder igualmente a las variables energéticas de toda la planta; asimismo debía clasificar y ordenar esta información para ser enviada a una base de datos no relacional. El desarrollo trajo algunos interrogantes importantes tales como qué dispositivo usar, qué protocolo de comunicación implementar ya que se requería de un alto poder de procesamiento, también la facultad de enviar y recibir información de Internet y la posibilidad de conectar una gran cantidad de equipos, los cuales fueron resueltos con la implementación de una Raspberry pi con un protocolo de comunicación MODBUS.

Objetivo general

- Desarrollar un algoritmo que permita comunicar e implementar un sistema de adquisición y monitoreo de las variables físicas implícitas en un proceso industrial.

Objetivos específicos

- Diseñar los módulos
- Cuantificar todas las variables físicas implícitas en el proceso.
- Clasificar, ordenar las variables y enviar la información a una base de datos.
- Crear una base de datos interna como soporte.

- Implementar el sistema de comunicación en un proceso de clasificación.
- Poner a punto la red de comunicación.

2. MARCO TEÓRICO

Dada la creciente demanda de industria en obtener cada vez más información proveniente las redes de comunicación en sistemas de automatización dispuestos en campo, se hace necesario establecer un número de reglas y estándares con el fin de garantizar una eficiente utilización de las arquitecturas y flujos de información, este ha sido uno de los trabajos liderados por algunas organizaciones tales como ISO e ISA entre otras. A continuación no solo detallaremos un poco sobre dichas organizaciones sino también en otra clase de aportes en cuanto a arquitecturas de comunicación refiere, todo esto con el fin de estudiar sus contribuciones a lo que hoy se conoce como comunicación inalámbrica industrial. (G., 2011)

Modelo OSI

El modelo OSI propone una estructura muy amplia la cual es adoptada por cada red en diferentes formas (ciertos niveles son adoptados y otros no). El estándar final fue publicado en 1984, como ISO 7498, donde se destacan una arquitectura de 7 niveles o capas. Los cuales suponen una comunicación entre niveles y entre locutores. El Sistema de niveles o capas OSI busca especificar el sistema de transmisión, el método de acceso a la red y todo lo referente a cómo realizar un intercambio de información eficiente entre dos o más interlocutores. Para el caso del sistema a implementar con protocolo Modbus RTU donde solo se usan 3 de los 7 niveles propuestos, como se muestra en la figura. (G., 2011)

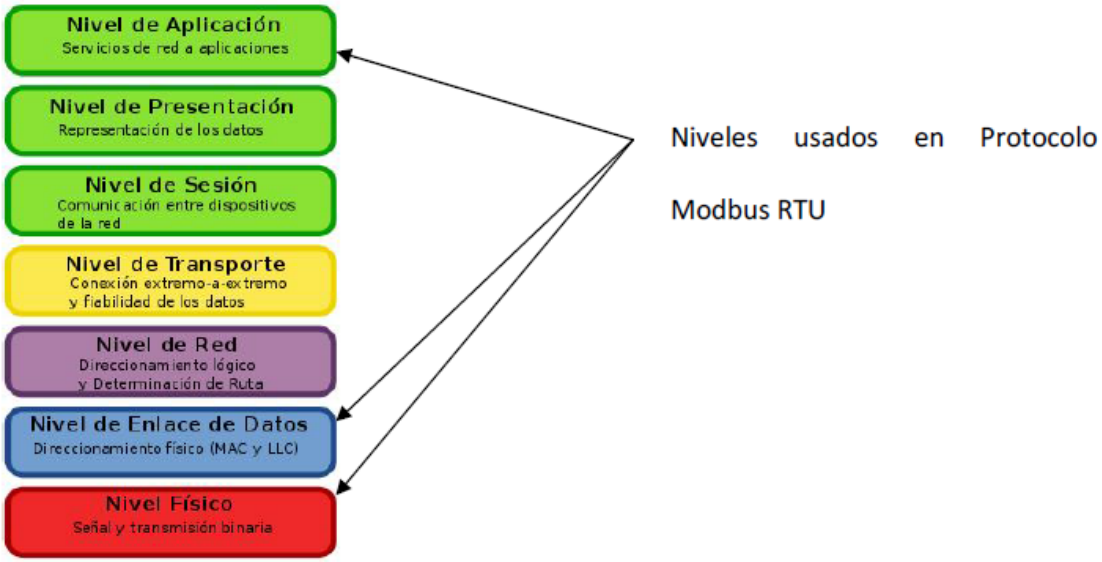

IMAGEN 1 NIVELES OSI

Fuente: (G., 2011)

Estándar ISA /SP50.

A pesar de que el modelo OSI es muy completo, casi todos los buses de campo ofrecen grandes problemas para poder satisfacer los niveles intermedios, del 3 al 6, dado que los fabricantes no previeron la conexión con otros buses. Para completar el paquete de protocolo propuesto por el modelo OSI, la sociedad para instrumentación, sistemas y automatización, ISA, propone una serie de complementos o mejoras bajo la denominación ISA-SP50, donde se trata de desarrollar las normas necesarias para definir las características que deben cumplir las señales (analógicas y digitales) usadas en medidas de proceso y control, y transmitir las información entre subsistemas o elementos separados de sistemas.

ANSI/ISA-50.1-1982, como complemento al modelo OSI, este estándar propone 2 capas adicionales: Capa de usuario: Bloques de Funciones y gestiones de base de datos a disposición del usuario como tal para facilitar el control y adquisición de datos. Capa de Supervisión: funciones de monitoreo, parametrización y configuración de dispositivos entre otras, son las herramientas para la gestión de redes y sistemas. Adicionalmente propone adiciones y/o mejoras a la composición de las capas 1, 2 y 7. La capa 1 de ISA 50.1, se adapta al nivel 1 del modelo OSI, tal como es definido en ISO 7498, con la excepción de que los limitadores de trama también hacen parte de esta capa, especificándolo en ANSI/ISA-S50.02.En caso de la capa 2, esta es mejorada complementándola con características de servicios de enlace de datos conveniente para comunicaciones críticas, el estándar que lo define es el ANSI/ISA-S50.03, y por último la capa 7 de Aplicación, complementada por el estándar ANSI/ISA-S50.02, parte 5 y parte 6, define una interface común para la interconexión de componentes de sistemas de medida y control.

Modbus RTU

Protocolo desarrollado por Modicon en 1979, diseñado para la comunicación con dispositivos inteligentes, como controladores, y unidades de adquisición de datos. Desde entonces se ha posicionado como uno de los protocolos más usados en la industria, la modalidad RTU inicialmente es ideal para la monitorización remota vía radio de elementos de campo, de allí su nombre (RTU, Remote Terminal Unit). Dicho protocolo cumple con los modelos anteriores ISO e ISA en solo 3 de las 7 capas posibles: capa 1: físico, capa 2: enlace y capa 7: aplicación. Modbus RTU ha tenido amplia utilización en el medio Industrial, soportado por una amplia gama de fabricantes y entidades que lo promueven y estudian. La organización Modbus – IDA constituida en el 2002 expone un amplio marco de aplicaciones para esta clase de protocolos facilitando su uso e implementación. (G., 2011)

Modbus usa una topología Maestro-Esclavo, donde la comunicación es manejada solo por el maestro mediante el envío y recepción de mensajes a solicitud del aplicativo. Bajo el funcionamiento de solicitud-respuesta se ofrecen servicios especificados en los códigos de función (método en el cual se solicitan acciones a cliente desde un servidor). Los códigos de función MODBUS son elementos de la solicitud, claramente visibles en la trama de solicitud y respuesta. (G., 2011)

Existen dos variantes del protocolo Modbus: ASCII y RTU; siendo este último especialmente escogido para las aplicaciones industriales. La modalidad RTU, nace de la aplicación de esta en comunicaciones con estaciones remotas, de allí deriva su nombre. Este modo es eficiente en cuanto ocupa por menor tiempo el medio en trasmisión y recepción. (G., 2011)

La modalidad ASCII básicamente usa caracteres ASCII para la codificación de sus caracteres junto caracteres espaciales para el final de trama permitiendo al dispositivo conocer con certeza el final de esta, con esto se evita hacer vigilancia a las tramas con el uso de temporizadores. Esta modalidad es menos eficiente que la RTU. Para esta implementación se ha escogido la modalidad RTU, la cual ha sido ampliamente adoptada en el medio industrial. En la actualidad existen varias formas de implementación de este protocolo (G., 2011):

- Sobre TCP / IP en Ethernet, en el cual básicamente los mensajes Modbus son encapsulados en los datos útiles de una comunicación TPC / IP. Este caso no será tratado en este documento.

- Comunicación asíncrona serie, transmisión a través de varios tipos de interfaces y medios, bajo las normativas: EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fibra, radio, etc.

Modbus RTU cuenta con las siguientes ventajas:

- Protocolo con gran cantidad de información abierta al público en diferentes fuentes.

- Se encuentran una alta variedad de fabricantes que respetan las reglas en la forma como se programa y se manejan las variables.

Las ventajas mencionadas que pertenecen a Modbus lo convierte, en una gran solución para ser implementada en entornos industriales.

3. METODOLOGÍA

Para logra la meta se hizo un estudio de:

- Módulos Comerciales compatibles con Raspberry.
- Instalación de las bibliotecas que soportan los módulos MODBUS.
- Codificación del maestro y los esclavos.

En este capítulo se dará una descripción teórica de las tecnologías y componentes implementadas en el sistema.

Diseño de la red y tecnologías empleadas.

Para implementar la red de comunicación MODBUS RTU se usó una placa Raspberry como maestro, un PLC y un analizador de redes como esclavos. Iniciando por la capa física se emplearon pares trenzados de cable para conexiones ya que son los más económicos, para la comunicación física se usó el estándar RS-485.

En los esclavos se implementan diferentes sensores, estos al tomar sus respectivas medidas guardan esta información en sus registros MODBUS. Información que será leída constantemente por el maestro logrando de esta manera centralizar la monitorización en el dispositivo maestro tal como hacen los sistemas SCADA en entornos industriales. Es ya tarea del maestro tomar, clasificar y procesar esta información leída de los sensores. En este caso estos datos clasificados y procesados son y enviados a una base de datos en Internet y otra base de datos es guardada en el procesador como soporte para los momentos en que no esté disponible la conexión a internet, brindando así acceso remoto y local a toda la información del estado de todo el proceso y de esta forma entregando información valiosa para la toma de decisiones.

Transmisión física

La transmisión física establece la forma en que los bits se convierten en señales apropiadas para ser enviadas por el medio físico también distinguido como canal. Dependiendo de cuál sea el medio, el canal tendrá ciertos dominios que hacen que haya señales que sean más o menos fuertes a la atenuación del canal.

RS-485

RS-485 o EIA-485 se hizo estándar en 1983 por la *TIA/EIA*. Es un estándar de comunicaciones en bus para la capa física del modelo *OSI* y define las características eléctricas de los transmisores y receptores. La transmisión es serial y asíncrona, lo cual quiere decir que los bits se van transmitiendo uno detrás de otro y sin una señal que sincronice transmisor y receptor. El medio físico es un par trenzado (A, B) que admite hasta 256 estaciones. La comunicación es semidúplex y se pueden cubrir hasta 1200 metros con un mismo bus sin pérdida de información gracias a la transmisión diferencial que cancela gran cantidad de ruido. Las velocidades de transmisión oscilan entre los 300 y los 19200 bit/s.

La transmisión diferencial se logra transmitiendo en cada cable una señal de igual magnitud desfasada 180 grados, de esta forma se logra que las interferencias afecten por igual ambos cables e invirtiendo una de las señales se consigue que el ruido se anule al sumarlas como se puede apreciar en la siguiente figura:

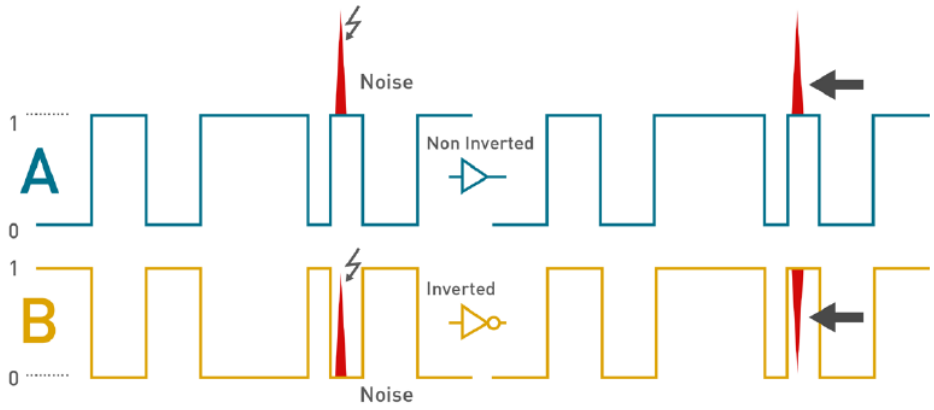

IMAGEN 2-TRASMISION DIFERENCIAL

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Las especificaciones estándar se resumen en la siguiente tabla:

TABLA 1-ESPECIFICACIONES DEL ESTANDAR RS-485

RS-485	
Estándar	TIA/EIA-485-A
Medio físico	Par trenzado
Topología de red	Punto a punto, punto a multipunto, multi-drop
Modo de comunicación	Semiduplex, dúplex
Máximo de dispositivos	Originalmente 32, actualmente 256 e incluso más usando repetidores
Modo de operación	Diferencial
Niveles de tensión	-7V / +12V
"1" Lógico	Tensión positiva (B-A > +200mV)
"0" Lógico	Tensión negativa (B-A < -200mV)

Fuente: (Barastegui, bibliote de ingenieria sevilla, 2017)

El diagrama de topología MODBUS lo podemos ver en la siguiente imagen:

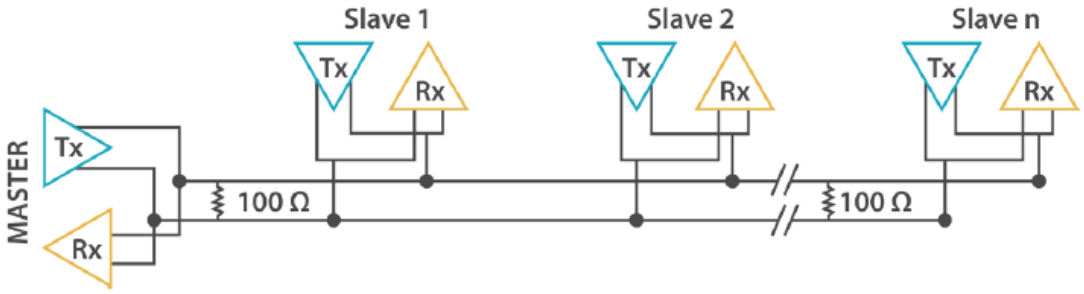

IMAGEN 3-DIAGRAMA DE RED RS-485

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Protocolo MODBUS

MODBUS es un protocolo de solicitud-respuesta implementado usando una relación maestro-esclavo. En una relación maestro-esclavo, la comunicación siempre se produce en pares, un dispositivo debe iniciar una solicitud y luego esperar una respuesta y el dispositivo de inicio (el maestro) es responsable de iniciar cada interacción. Por lo general, el maestro es una interfaz humano-máquina (HMI) o sistema SCADA y el esclavo es un sensor, controlador lógico programable (PLC) o controlador de automatización programable (PAC). El contenido de estas solicitudes y respuestas, y las capas de la red a través de las cuales se envían estos mensajes, son definidos por las diferentes capas del protocolo.

En este apartado se hará una descripción del protocolo MODBUS sobre línea serie basada en las Especificaciones que se pueden descargar de www.modbus.org.

El protocolo MODBUS es una especificación que define un protocolo de nivel de aplicación que, aunque no cumple todo el modelo OSI, se podría enmarcar en los niveles 2 y 7 para MODBUS serie y MODBUS TCP/IP respectivamente. MODBUS no especifica cuál ha de ser la capa física, aunque la más usada es el soporte metálico y su velocidad de transmisión va desde los 75 a los 19200 baudios recomendados por el estándar, aunque en condiciones adecuadas se pueden configurar velocidades mayores.

La topología del protocolo es maestro-esclavo, pudiendo existir uno o varios esclavos que responderán las peticiones del maestro, ya que sólo éste puede iniciar la comunicación. Teniendo en cuenta lo anterior se puede decir que los esclavos son por analogía servidores y el maestro actúa como cliente.

MODBUS serie tiene dos modos de funcionamiento que se detallarán posteriormente; ASCII o RTU dependiendo del formato en que se codifique la información dentro de la trama ya sea en ASCII o en binario. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Modelo de datos

Según su tamaño podemos clasificar en dos los tipos de datos que maneja el protocolo: bits individuales y registros de 2 Bytes. Los bits individuales para entradas y salidas digitales y los registros para variables que requieran mayor tamaño. En la siguiente tabla se muestran los tipos de datos disponibles:

TABLA 2-MODELO DE DATOS MODBUS.

Tipo de objeto	Acceso	Tamaño
Discrete input	Solo leer	1 bit
Coil	Leer/escribir	1 bit
Input register	Solo leer	16 bits
Holding register	Leer/escribir	16 bits

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

- Discrete input: Puede ser generado por un sistema de salida/entrada.
- Coil: Puede ser modificado.
- Input register: Puede generarse `por una entrada o una salida.
- Holding: Puede ser modificado.

Modelo de direccionamiento

Las direcciones de memoria en una red MODBUS están arregladas por los tipos de datos:

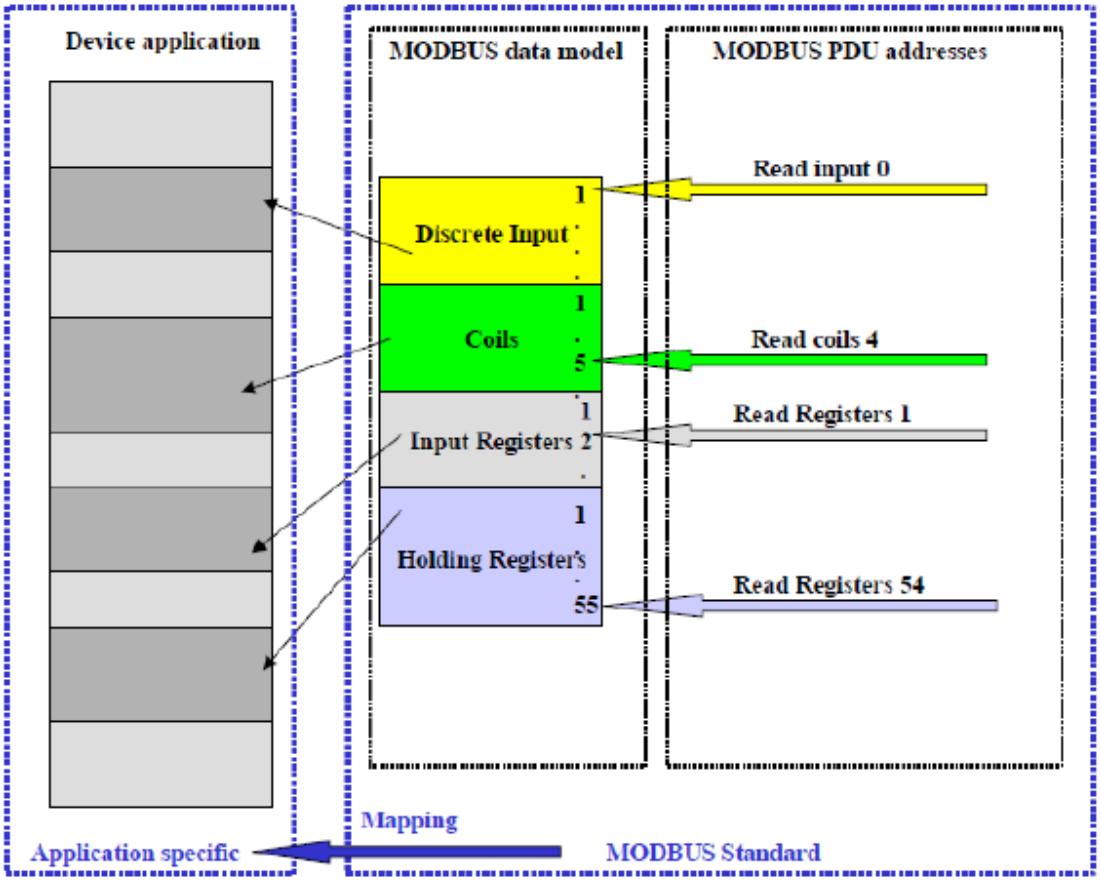

IMAGEN 4-MODELO DE DIRECCIONAMIENTO MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Como se ve en la figura poseen cuatro bloques que corresponden al tipo de datos. Todas las direcciones están referenciadas a cero de manera que si queremos leer el elemento N de un determinado bloque, éste será direccionado como N-1.

El esquema de las direcciones MODBUS con las direcciones reales del dispositivo es independiente del estándar MODBUS, dependiendo totalmente del fabricante. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Formato de la trama

El protocolo de aplicación MODBUS define una unidad de datos de protocolo (PDU) independientemente de las capas inferiores: pero en MODBUS serie hay que añadir algunos campos adicionales para construir una PDU adecuada para establecer la comunicación en redes o buses. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

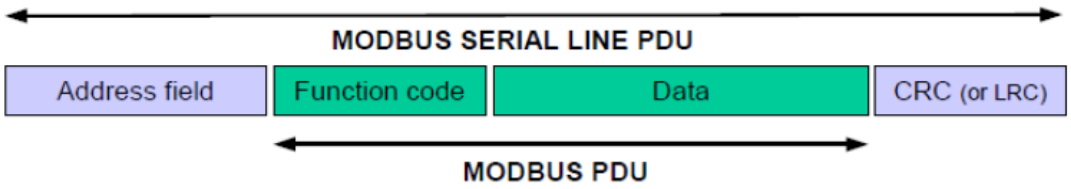

IMAGEN 5-TRAMA MODBUS SERIE.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

- Address Field: Sirve para indicar la dirección del esclavo al que va dirigida la trama. El rango válido va desde 0 a 247, siendo el 0 la dirección de Broadcast y quedan reservadas las direcciones 248 a 255. Cuando el esclavo recibe una trama dirigida a él; construye la respuesta y pone su propia dirección en este campo, para que el maestro sepa de qué esclavo viene la respuesta.
- Función Code: Indica el código de la operación que el maestro solicita al esclavo; por ejemplo, leer un determinado registro.
- Data: Lleva la información que se necesite para realizar determinada función; por ejemplo, escribir un valor en el registro indicado.
- CRC o LRC: Chequeo de redundancia cíclica o chequeo de redundancia longitudinal: sirve para asegurarse de que la información llega sin errores.

Diagrama de estado maestro

Este diagrama ilustra el comportamiento del maestro:

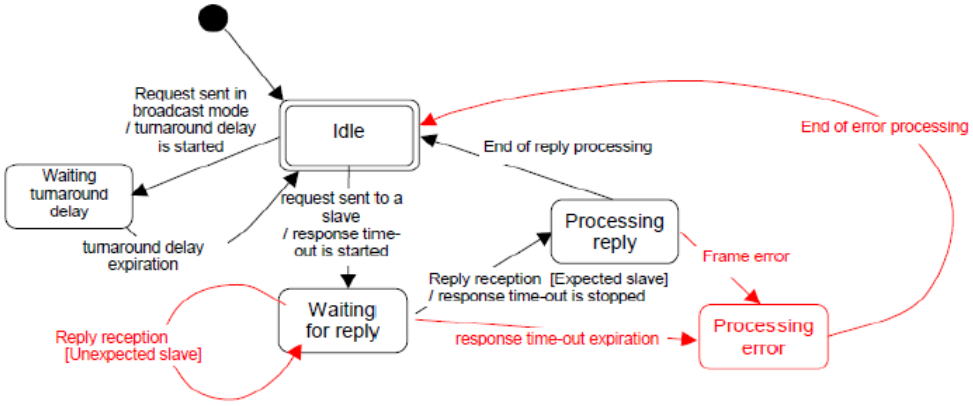

IMAGEN 6-DIAGRAMA DE ESTADOS DEL MAESTRO.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Existen 5 estados: “Idle” es el estado inicial de reposo, sólo se pueden enviar solicitudes desde este estado y no se sale de él a no ser que se envíe una solicitud en modo broadcast (dirigida a todos los esclavos) o unicast (dirigida a un único esclavo). Cuando se manda una solicitud en modo broadcast se activa una espera “turnaround” y permanece en estado de “Espera turnaround” para que los esclavos tengan tiempo de atender la solicitud antes de volver al estado de reposo desde el cuál se podría lanzar una solicitud nueva. Si se manda una solicitud en modo unicast se activa un temporizador y se pasa al estado “Esperando respuesta”. De este estado se sale si: expira el temporizador; pasando al estado “Procesar error”, o si se recibe una respuesta; deteniendo el temporizador y pasando al estado “Procesar respuesta”. Por último tras procesar la respuesta vuelve al estado de reposo en caso de que la respuesta fuese correcta (no se detectaron errores de paridad ni de CRC/LRC) o pasa al estado “Procesar error” tras el cual vuelve de nuevo al estado de reposo desde el cuál se realizaría un reintento. El número de reintentos depende de la configuración del

maestro. Los errores de trama consisten en: Comprobación de paridad de cada carácter y comprobación de redundancia de la trama completa. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Diagrama temporal de la comunicación maestro / esclavo

En este diagrama se pueden ver tres posibilidades de intercambio de información maestro / esclavo:

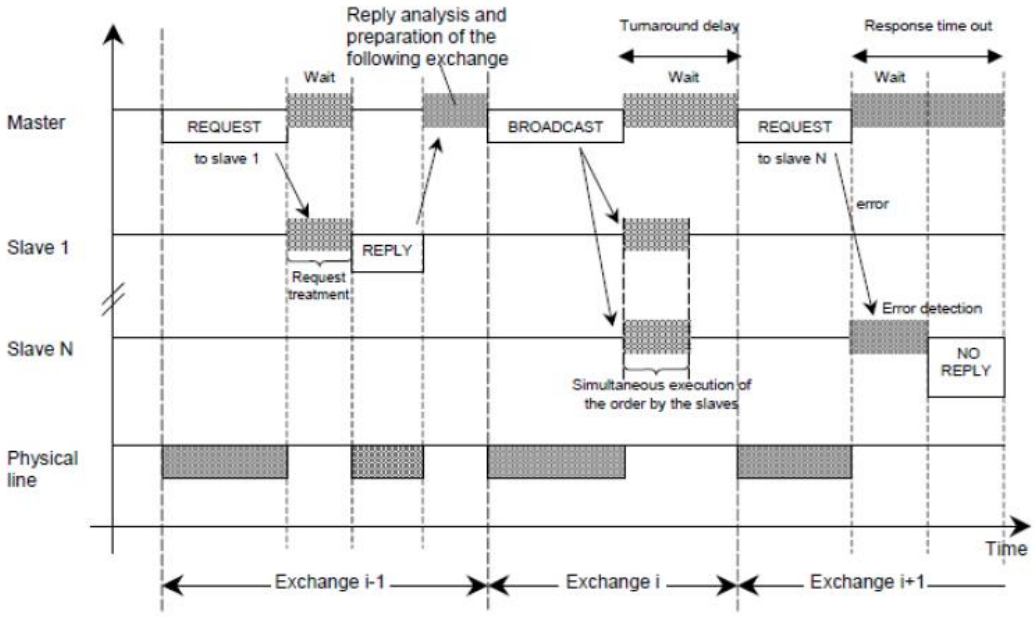

IMAGEN 7-DIAGRAMA TEMPORAL DE LA COMUNICACIÓN MAESTRO / ESCLAVO.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

El primer intercambio (el primero empezando por la izquierda) es una solicitud en modo unicast que el esclavo 1 completa sin errores y envía la respuesta al maestro. Se pueden ver en gris los tiempos de espera del maestro y de procesamiento de la solicitud del esclavo. El segundo intercambio es una solicitud en modo broadcast. Se puede ver como los esclavos una vez la reciben la ejecutan en paralelo y como el maestro permanece a la espera un "Turnaround delay" antes de volver a enviar otra solicitud. El tercer y último intercambio es

una solicitud al esclavo N; el esclavo detecta un error en la trama y la ignora, el maestro espera un tiempo hasta que expira el temporizador a partir de

entonces, aunque no se ve reflejado en el diagrama, podría enviar un reintento. Los tiempos de Request y Broadcast dependen de las características de la trama (longitud, throughput) Los tiempos de espera y de tratamiento de la solicitud dependen de la capacidad del esclavo para el procesamiento de la solicitud. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Modo de transmisión RTU

En el modo de transmisión RTU para transmitir un Byte de información se necesitan agregar unos bits de inicio y stop para que el receptor sepa cuando empieza y acaba la información. Como se puede ver en la figura existen dos modos dependiendo de si se comprueba la paridad o no. El modo definido por defecto por el estándar MODBUS es el de paridad par, quedando los modos de paridad impar o no paridad como opcionales. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

El bit de paridad es un bit que se pone a “1” de manera que el número total de bits a “1” en el Byte que se está enviando coincida con el modo de paridad elegido.

Un ejemplo: se transmite “10010001” y el modo configurado es paridad par; como hay tres bits a “1”, para que el número de bits total sea par el bit de paridad se tendrá que poner a “1”. En caso contrario se pondría a “0”.

Los bits se transmiten en el siguiente orden: de izquierda a derecha; desde el bit menos significativo (LSB) hasta el más significativo (MSB).

En el modo de no paridad como se puede observar se sustituye dicho bit por otro bit de stop. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

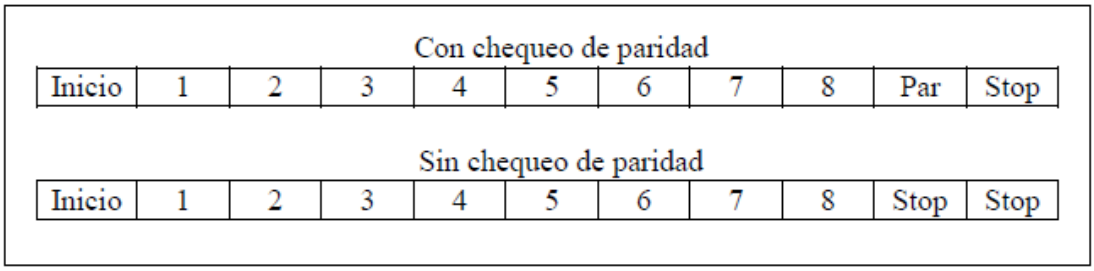

IMAGEN 8-SECUENCIA DE BITS EN MÓDULO RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Con este precedente se da claridad acerca de cómo se envían los Bytes de información de cada campo y es preciso enfatizar que para transmitir un Byte de información en modo *RTU* se necesitan enviar 11 bits y la trama se puede observar en la siguiente figura:

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

IMAGEN 9-TRAMA MODBUS RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

La siguiente figura muestra cómo se identifica el inicio y final de una trama; gracias a unos intervalos de silencio de al menos 3.5 veces el tiempo que se tarda en enviar un carácter que llamaremos a partir de ahora t3.5:

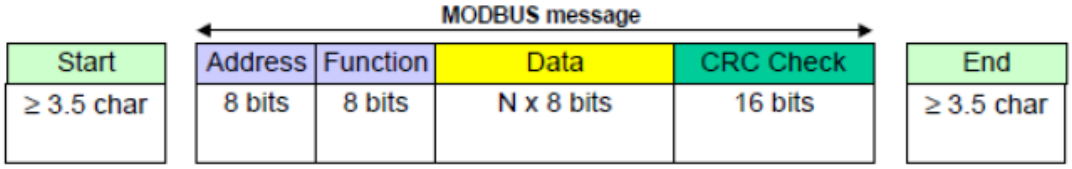

IMAGEN 10-SECUENCIA DE TRAMA MODBUS RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Es preciso aclarar que entre dos tramas consecutivas no se suma el t3.5 de fin de una trama con el t3.5 de inicio de la siguiente. Esto queda reflejado en la siguiente figura:

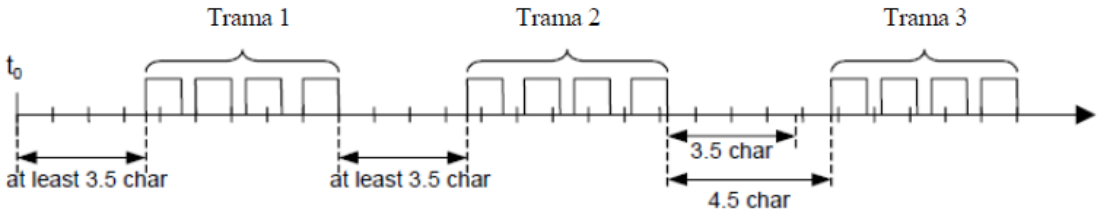

IMAGEN 11-SEPARACIÓN DE TRAMAS MODBUS RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Tal como representa la siguiente figura dentro de cada trama los Bytes se tienen que transmitir de manera continua. Si entre dos Bytes se produce un silencio de más de 1.5 veces el tiempo de carácter se considera errónea la trama y se descarta.

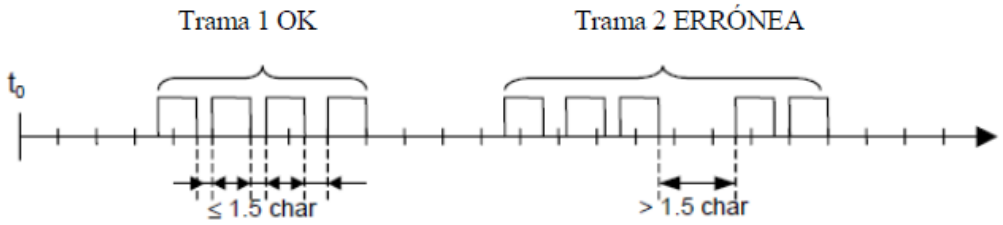

IMAGEN 12-SEPARACIÓN DE BYTES EN TRAMAS MODBUS RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Modo de transmisión ASCII

Como se puede ver en la figura existen dos modos dependiendo de si se comprueba la paridad o no. El modo definido por defecto por el estándar MODBUS es el de paridad par, quedando los modos de paridad impar o no paridad como opcionales.

Los bits se transmiten en el siguiente orden: de izquierda a derecha; desde el bit menos significativo (LSB) hasta el más significativo (MSB).

En el modo de no paridad como se puede observar se sustituye dicho bit por otro bit de stop. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

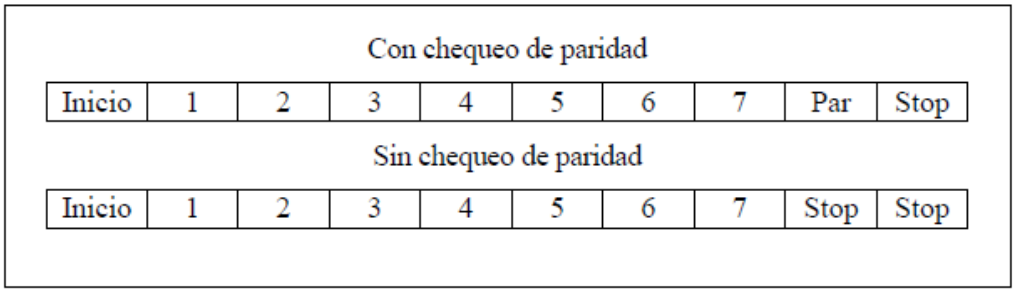

IMAGEN 13-SEPARACIÓN DE BYTES EN TRAMAS MODBUS RTU.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

La gran desventaja del modo de transmisión ASCII estriba en que un byte de información se codifica en dos caracteres ASCII, ocupando cada carácter ASCII 7 bits, más los bits extra que se necesitan de alineación y paridad de forma que un carácter ASCII ocuparía en total 10 bits. Si esto se compara con el modo RTU podría interpretarse erróneamente que el modo ASCII fuese más eficiente que el modo RTU que ocupa 11 bits por byte, nada más lejos de la realidad puesto que ASCII necesita enviar dos caracteres para transmitir un byte de Información, lo que hace en total 20 bits, sin embargo, en modo RTU el byte se envía con 11 bits.

El hecho de que se use el modo ASCII responde a cuestiones relacionadas con los medios físicos y la capacidad de los dispositivos usados, de manera que no sean compatibles con las especificaciones de los temporizadores definidos para MODBUS RTU.

A diferencia del modo RTU, en modo ASCII se permiten intervalos de hasta un segundo entre caracteres. A no ser que el usuario haya configurado un temporizador mayor, se considerará erróneo un intervalo de más de un segundo entre caracteres. En algunas aplicaciones de red de área extensa se usan intervalos de entre 4 o 5 segundos.

En modo ASCII las tramas se delimitan con caracteres en lugar de silencios: el carácter de inicio es “dos puntos” (:); en ASCII (0x3A) y los de final de trama el “retorno de carro” (CR) y “salto de línea” (LF); en ASCII (0x0D y 0x0A).

Los caracteres permitidos en todos los campos de la trama son hexadecimales 0-9, A-F codificados en ASCII. Los dispositivos monitorizan el bus continuamente hasta que encuentran el carácter “:”. Entonces decodifican los caracteres siguientes hasta que detectan el fin de trama. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

A continuación se muestra una trama MODBUS ASCII:

Start	Address	Function	Data	LRC	End
1 char :	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR,LF

IMAGEN 14-TRAMAS MODBUS ASCII.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Esta tabla expone la misma trama enviada en cada modo y la diferencia en bits entre cada uno de ellos:

TABLA 3-COMPARATIVA ENTRE ASCII Y RTU.

Campo	Hexadecimal	ASCII	Caracteres ASCII (10 bits)	RTU	Bytes RTU (11 bits)
Cabecera		:	1		
Dirección del esclavo	07	0 7	2	0000 0111	1
Código de función	03	0 3	2	0000 0011	1
Dirección de inicio (Alto)	00	0 0	2	0000 0000	1
Dirección de inicio (Bajo)	0A	0 A	2	0000 1010	1
Nº de registros (Alto)	00	0 0	2	0000 0000	1
Nº de registros (Bajo)	02	0 2	2	0000 0010	1

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

TABLA 4-COMPARATIVA ENTRE ASCII Y RTU

Chequeo de errores		LRC	2	CRC	2
Fin de trama		CR LF	2		
Total de Bits			170		88

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Comprobación de errores

Chequeo de paridad: Dentro de la trama, se aplica a cada byte que se transmite de manera que el transmisor calcula para cada byte el bit de paridad y lo adjunta al mensaje. El receptor cuando lo recibe calculará la paridad del mensaje y comprobará que coincida con el bit de paridad que el transmisor adjuntó al mensaje. En caso contrario ignora el mensaje. El esclavo no responderá al maestro y el temporizador del maestro expirará.
Este tipo de verificación sólo es eficaz en caso de que el número de bits erróneos sea impar ya que si fuese par no cambiaría la paridad del mensaje.
Si el modo seleccionado es el de no paridad, no se comprueba, lógicamente, y se sustituye dicho bit por otro bit de stop. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Chequeo de CRC / LRC: Se aplica a la trama completa. Es un algoritmo que aplicado por el transmisor a la trama (sin incluir delimitadores) da como resultado un campo de dos bytes que se incluye en la trama antes de enviarla, de esta manera el receptor aplica el mismo algoritmo y comprueba si el resultado coincide con el campo de CRC / LRC que el transmisor incluyó en la trama. En caso contrario se ignora el mensaje. El esclavo no respondería al maestro y es el maestro el que tomará la decisión oportuna cuando expire su temporizador. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Códigos de función

Ya que el campo de código de función mide un byte y el bit de mayor peso se usa para que el esclavo indique si ha habido error; devolviendo el mismo código que se le solicitó con el bit de mayor peso a “1” y un byte en el campo de datos indicando el código de error que ha tenido lugar, quedan disponibles 127 códigos de función. Cada código de función se

corresponde con una operación diferente. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Existen tres categorías de códigos de función:

- Públicos: Estos códigos de función están bien definidos, aprobados, testeados y documentados por la comunidad MODBUS.

- Definidos por el usuario: Son implementaciones específicas que un usuario crea para su aplicación y que no está soportada por el estándar.

- Reservados: Algunas compañías tienen funciones reservadas para sus productos que no son accesibles públicamente.

Según el tipo de operación que se desea realizar sobre el esclavo se distinguen dos tipos:

- Lectura / escritura: Sirven para consultar o escribir datos en la memoria del esclavo.

- Control: Permiten realizar algunas acciones sobre el esclavo. Por ejemplo, ponerlo en modo de sólo escucha.

La siguiente tabla muestra las funciones más usadas:

TABLA 5-FUNCIONES MODBUS MÁS USADAS.

Command	Function Code
01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Register
07	Read Exception Status
08	Diagnostics

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

A continuación se van a describir las funciones a través de ejemplos que hagan más clara la explicación. Las funciones de lectura se pueden agrupar dos a dos ya que son completamente equivalentes, la única diferencia estriba en el tipo de dato leído; uno se puede también escribir y otro no, pero el tamaño es el mismo. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Read Coils (01) – Read Discrete Inputs (02)

Estas dos funciones sirven para que el maestro pueda consultar el estado de una cantidad, que se pasa como dato en la trama, determinada de coils o entradas contiguas en el esclavo. Pongamos como ejemplo la siguiente trama codificada en hexadecimal para ahorrar espacio, donde el maestro consulta el estado de los 10 primeros coils del esclavo con dirección 16:

Dirección esclavo	Código de función	Dirección de inicio	Cantidad de bits	CRC
10	01	00 00	00 0A	----

IMAGEN 15-TRAMA MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Como se puede apreciar la dirección del esclavo se corresponde con el número 16 en decimal, la dirección de inicio indica la dirección a partir de la cual se quiere consultar la información, está referenciada a cero, y la cantidad de bits que se van a leer se indica en el campo siguiente también en hexadecimal, correspondiendo al valor deseado 10 en decimal. El CRC no se ha calculado por simplificar, pudiéndose consultar el algoritmo para calcularlo en la especificación del estándar. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

A continuación se muestra la respuesta del esclavo a la consulta anterior:

Dirección esclavo	Código de función	Bytes de respuesta	Estado de los bits	CRC
10	01	02	FF 03	----

IMAGEN 16-TRAMA MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

En la respuesta se envía el código del esclavo para que el maestro sepa de quién es la respuesta y el mismo código de función. Si hubiese ocurrido algún error el código de función sería 0x81, esto es, 0x01 + 0x80 ya que 0x80=10000000b, que como se dijo, poner el MSB a "1" es la forma de indicar que ha habido un error.

En el campo bytes de respuesta se indica la cantidad de bytes que se han leído; si el número de bits solicitados no es múltiplo de 8 sobrarán bits que se rellenarán con ceros hacia el MSB. Esto se entiende mejor a través del ejemplo: en nuestro caso se pidieron 10 bits, vamos a suponer que el estado de esas diez entradas es "1", los dos bytes tomarían los valores hexadecimales 0xFF03, que en binario serían:

11111111 00000011

Se han marcado en negrita los bits que se corresponden con el estado de las entradas, quedando los ceros de relleno para completar el segundo byte. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Real Holding Registers (03) – Read Input Registers (04)

Estas dos funciones son totalmente análogas a las anteriores pero en lugar de leer entradas de un bit, leen registros de dos bytes. Un ejemplo:

Dirección esclavo	Código de función	Dirección de inicio	N° de registros	CRC
10	03	00 00	00 0A	----

IMAGEN 17-TRAMA MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

En este caso la dirección de inicio indicada es la misma que en el caso anterior, y al ser diferente el tipo de dato que queremos leer podría parecer erróneo el hecho de que hagamos referencia a la misma dirección de memoria para intentar leer datos diferentes. Esto no es así porque el código de función lleva implícita la información que el dispositivo necesita para conocer a qué tipo de dato nos estamos refiriendo y como se puede ver en la Figura 3; para cada tipo de dato hay un rango de direcciones aunque todas empiezan en cero.

La respuesta del esclavo es evidente por analogía con la función explicada anteriormente. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Write Single Coil (05)

Esta función se usa para modificar el estado de una salida tipo coil. Se pasa la dirección que se quiere modificar, seguida del estado: 0x0000 para “0” y 0xFF00h para “1”. Por ejemplo:

Dirección esclavo	Código de función	Dirección del coil	Valor a escribir	CRC
10	05	00 00	FF 00	----

IMAGEN 18-TRAMA MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Como ya se ha explicado en el apartado anterior se usa la misma dirección para el coil (el primero), no queriendo decir esto que ocupe la misma dirección de memoria que los datos leídos con las funciones anteriormente descritas, ya que estamos haciendo referencia al bloque de memoria de los coils.

La respuesta del esclavo sería un eco de la solicitud, esto es, idéntica si se ha escrito sin errores para confirmar esclavo, dirección y valor escrito. En caso de error se suma 0x80 al código de función, como ya se explicó anteriormente, resultando el código de la respuesta 0x85. (Barastegui, Diseño y desarrollo de una red MODBUS RTU, 2017)

Write Single Register (06)

Esta función es análoga a la anterior pero en lugar de escribir un bit escribe un registro, es decir, 2 bytes. Un posible ejemplo:

Dirección esclavo	Código de función	Dirección del registro	Valor a escribir	CRC
10	06	00 00	AA 05	----

IMAGEN 19-TRAMA MODBUS.

Fuente: http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

La respuesta del esclavo en caso de que todo haya ido bien sería un eco de la solicitud igual que en el caso anterior.

Raspberry

Raspberry PI es una placa computadora (SBC) de bajo coste, se podría decir que es un ordenador de tamaño reducido, del orden de una tarjeta de crédito, desarrollado en el Reino Unido por la Fundación Raspberry PI (Universidad de Cambridge) en 2011, con el

objetivo de estimular la enseñanza de la informática en las escuelas, aunque no empezó su comercialización hasta el año 2012. (Valencia, 2013). El concepto es el de un ordenador desnudo de todos los accesorios que se pueden eliminar sin que afecte al funcionamiento básico. Está formada por una placa que soporta varios componentes necesarios en un ordenador común y es capaz de comportarse como tal. A la Raspberry Pi la han definido como una maravilla en miniatura, que guarda en su interior un importante poder de cómputo en un tamaño muy reducido. Es capaz de realizar cosas extraordinarias. (Valencia, 2013)

Un Chipset Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos Turbo para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) Video Core IV un módulo de 512 MB de memoria RAM (aunque originalmente al ser lanzado eran 256 MB). Un conector de RJ45 conectado a un integrado lan9512 -jzx de SMSC que nos proporciona conectividad a 10/100 Mbps 2 buses USB 2.0, una Salida analógica de audio estéreo por Jack de 3.5 mm. Salida digital de video + audio HDMI, salida analógica de video RCA, Pines de entrada y salida de propósito general, conector de alimentación micro USB y lector de tarjetas SD. (Valencia, 2013)

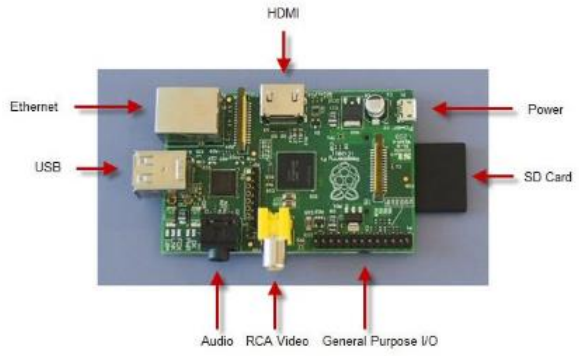

IMAGEN 20-TARJETA RASPBERRY.

Fuente: https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Componentes del sistema: Raspberry pi b2

Se implementó una Raspberry pi b2 una versión actualizada y con mejores prestaciones para este proyecto las características de esta Rasberry son:

- Procesador BCM2836 ARMv7 multinucleo (4 nucleos) de 32 bits.
- Velocidad de reloj: 900 Mhz.
- 1 GB en RAM.
- Conector GPIO de 40 pines (conector extendido)
- Salida de audio analógico / video analógico de 4 pines
- 4 Puertos USB
- Salida de video HDMI
- Interfaz para cámara y pantalla táctil (venta por separado)
- Capacidad de alimentar periféricos de alto consumo a través de USB.
- Controlador Ethernet 100 Mbps.
- Funciona a 5V – 1.5A (recomendado).
- Compatible a nivel software con el primer Raspberry Pi (requiere actualizar sistema operativo y software).
- Compatible a nivel hardware (layout) con el Raspberry Pi 1

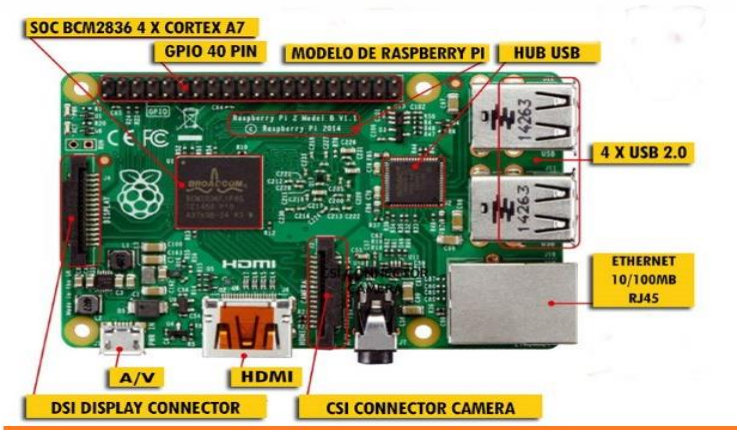

IMAGEN 21-TARJETA RASPBERRY PI B2.

Fuente: https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Conversor USB a RS-485

Se implementó este Conversor por que no necesita de fuente externa y con conexión directa a un puerto USB, compatible con USB 2.0 y 1.1 y también porque tanto el PLC como el analizador de redes cuentan con puertos RS-485 y la Raspberry con puertos USB facilitando así el emparejamiento de estos componentes, este conversor contiene las siguientes especificaciones:

- No necesita energía externa, accionado por el puerto USB
- Totalmente compatible con los estándares USB 2.0 y USB 1.1
- Sistema Operativo: Windows XP, Vista, Windows 7, Linux, MacOS, y unidad WinCE5.0
- Soporta rata de baudios rango: 75 - 115200 bps, hasta 6Mbps
- Plug, Play y hot-swap (USB-side);
- Comunicación a distancia hasta de 1.2Km, con baja interferencia.
- Rango de temperatura de trabajo: -40 ° C ~ + 85 ° C

IMAGEN 22-CONVERSOR RS-485.

Fuente: https://www.didacticaselectronicas.com/index.php/comunicaciones/conversores/usb-rs232-rs485/conversor-usb-a-rs-485-detail

PM-PA/PM-PAC POWER ANALYZER

Se implementó este dispositivo porque es ideal para medir y monitorear todos los parámetros eléctricos de un sistema. Por medio de sus cuatro filas de visualización, todos los parámetros son leídos fácilmente desde la pantalla principal. El dispositivo es aplicable para sistemas mono, bifásicos y trifásicos. Puede medir y mostrar 123 parámetros eléctricos y cuenta con RS 485 para la comunicación a través del protocolo Modbus.

IMAGEN 23-PM-PA/PM-PAC POWER ANALYZER.

Fuente: propia

PLC XINJE:

Se implementó este PLC porque posee Capacidad de programa: 8000 Pasos tiene 18 entradas

Con tipo de entrada: Contacto libre de voltaje o NPN, voltaje Señal de entrada: 24VDC +/- 10 %

Y un número de 14 salidas también, tiene salidas tipo Relé 3A 250V AC / 30VDC, carga resistiva y 80VA de carga Inductiva también cuenta con máximos Puntos de Entrada y Salida:

228 (7 Módulos de Expansión),bobinas Internas (M): 8,512,tiempo de Scan: 0 - 99 ms, reloj tiempo real, puertos de comunicación: COM 1: RS232 / COM2: RS-485,alimentación: 90- 260 VAC,soporte de control lógico básico y operación de datos, admite conteo de alta velocidad, interrupción externa, bloque de función de lenguaje C y comunicación de protocolo libre y comunicación MODBUS.

IMAGEN 24-PLC XINJE.

Fuente: propia

Instalación de las bibliotecas que soportan los módulos MODBUS

En la Raspberry se instaló un sistema operativo Rasbian el cual no es más que una derivación de LINUX para Raspberry esta viene con el lenguaje de programación Python instalado y con sus librerías estándar, pero para este desarrollo necesitaremos instalar una librería externa llamada PyModBus para facilitar la comunicación por medio del protocolo MODBUS y otra llamada PyMongo para el manejo y envió de los datos una base de datos no relacional llamada MongoDB.

Para esto solo se necesita abrir la terminal del sistema y escribir, para MoDbus lo siguiente:

- pip install -U pymodbus

Para MongoDB lo siguiente:

- sudo apt-get install python3-pymongo

En este último especificamos que versión de Python vamos a usar.

Codificación del maestro y los esclavos

Debido a que la Raspberry es la encargada de procesar y enviar los datos, esta fue usada como maestro, y tanto el PLC como el analizador de redes fueron usados como esclavos.
El PLC fue el encargado de tomar toda la información de un proceso de clasificación a este se le fue otorgada la dirección de esclavo 2 para atreves de esta poder acceder a sus registros. Se trabajó sobre un proceso de selección y clasificación de piezas metálicas en el cual se crearon las siguientes variables:

- contador_de_Piezas: Esta era la encargada del conteo de cada pieza que entraba en el proceso sea metálica o no y se le asignó el registro 49281.
- Piezas Desechadas: Esta se encargaba de contar el número de piezas que no cumplían los estándares en este caso ser metálica y se le asignó el registro 49282.
- piezas_Metelicas: Este tomaba medida de todas las piezas aceptas como metálicas y se le asignó el registro 49283.
- Interrupciones: Variable que contaba el número de veces que el proceso sufría una interrupción se le asignó el registro 49285.
- tiempo_de_motores: Variable en donde se tenía una medida de tiempo de funcionamiento de los motores se le asignó el registro 50369.
- velocidad_de_banda: Esta variable era la encargada de medir la velocidad a la que trabajaba la banda trasportadora del proceso se le asignó el registro 50378.
- Tiempo_de_matenimiento: En esta variable se acumulaba la cantidad de tiempo en ejecución de trabajo para tener un estándar en relación a cada cantidad de tiempo se debe hacer un mantenimiento y se le asignó el registro 50428.

En el analizador se tomaron las variables físicas de voltaje necesarias para tener conocimiento del consumo energético del proceso, pero en este caso el analizador de redes cuenta con unas especificaciones de fábrica las cuales permitían asignar un número de esclavo al cual se le asignó el número de esclavo 2, pero el número de registro no fue posible asignarse, porque él cuenta con unos registros ya asignados para cada variable al cual solo debíamos llegar a leer este valor y las variables que tomamos para este proceso fueron las siguientes:

- Freq: Encargada de entregar a que frecuencia se encuentra el sistema.
- VLN1: Encargada de guardar la magnitud del voltaje en la línea de tensión 1.
- VLN2: Encargada de guardar la magnitud del voltaje en la línea de tensión 2.
- VLN3: Encargada de guardar la magnitud del voltaje en la línea de tensión 3.
- I1: Variable donde se almacenaba la magnitud de la corriente en la línea de tensión 1.
- I2: Variable donde se almacenaba la magnitud de la corriente en la línea de tensión 2.
- I3: Variable donde se almacenaba la magnitud de la corriente en la línea de tensión 3.
- Corriente total: Variable encargada de guardar la suma de todas las corrientes I1+I2+I3.
- energia_activa: Variable encargada de entregar el total de la energía activa.
- factor_de_potencia: Variable donde se guardaba el factor de potencia.

Implementación

Para la implementación se creó un algoritmo en el lenguaje de programación Python ejecutado en la Raspberry pi B2, este se estableció de forma funcional para simplificar y evitar líneas de código repetidas.

Se empezó creando una función que tomara los datos del PLC y retornara su valor llamada Read_Registers_plc, esta función llama otra función de la biblioteca pymodbus llamada ModbusClient en la cual se define el método, el puerto, los baudios y el tiempo de espera, y también se usa el método connect para conectar el sistema, luego se usa el método read_holding_registers el cual es el encargado de leer los registros por lo tanto en primer lugar se le debe ingresar la dirección del registro, en segundo lugar se le indica si el dato está fraccionado en dos bits o solo en uno y por último se le indica a cual esclavo debe ingresar posteriormente se guarda en la variable en la posición 1 que en Python siempre es la numero 0 y finalmente todas las variables son guardadas en una lista llamada lecturas para ser retornadas cuando se haga uso de la función Read_Registers_plc.

En este no sé tuvo mayores complicaciones porque al poder decidir qué tipo de dato entregar desde el plc, se facilitó el proceso enviando el valor como entero directamente como lo podemos ver a continuación:

```
def Read_Registers_plc():
    modbus = ModbusClient(method='rtu', port ='/dev/ttyUSB0', baudrate=9600, timeout=1)
    modbus.connect()

    leer = modbus.read_holding_registers(49281,1,unit=2)
    contador_de_Piezas = leer.registers[0]

    leer = modbus.read_holding_registers(49282,1,unit=2)
    piezas_Desechadas = leer.registers[0]

    leer = modbus.read_holding_registers(49283,1,unit=2)
    piezas_Metelicas = leer.registers[0]

    leer = modbus.read_holding_registers(49285,1,unit=2)
    interrupciones = leer.registers[0]

    leer = modbus.read_holding_registers(50369,1,unit=2)
    tiempo_de_motores = leer.registers[0]

    leer = modbus.read_holding_registers(50378,1,unit=2)
    velocidad_de_banda = leer.registers[0]

    leer = modbus.read_holding_registers(50428,1,unit=2)
    Tiempo_de_matenimiento = leer.registers[0]


    lecturas = (contador_de_Piezas,piezas_Desechadas,piezas_Metelicas,
                interrupciones,tiempo_de_motores ,velocidad_de_banda ,
                Tiempo_de_matenimiento)

    return lecturas
```

IMAGEN 25-FUNCIÓN PARA LEER REGISTROS DEL PLC.

Fuente: propia

Para la lectura de los registros del analizador se tuvo que implementar una función auxiliar que tomara los valores encontrados y los pasara de binarios a flotantes ya que los analizadores no entregaban estos datos de manera flotante si no de manera binaria y en dos particiones, para solucionar este problema se desarrolló una función llamada convertirFloat a la cual se le entregaba como parámetro un numero binario y esta lo guardaba en una variable llamada f que convertía en entero partido en 2, luego se creó la variable dato la cual hacia uso de la biblioteca struct y su método unpack al cual en primer lugar se le indica que tipo de dato debe retornar , en segundo lugar se usa el método pack que en su primer parámetro se le indica que integre y en el segundo se le pasa el valor a

Integrar el cual es f, para el final ser guardado en la posición 0 y por último se retorna la variable dato, a continuación se podrá ver la forma en que este problema fue resuelto:

```
def convertirFloat(binario):
    f=int(binario,2)
    dato = struct.unpack('f', struct.pack('I',f))[0]
    return dato
```

IMAGEN 26-FUNCIÓN PARA CONVERTIR DE BINARIO A FLOTANTE.

Fuente: propia

Luego de resolver el problema de los números binarios se desarrolló la función Read_registers_analyzer que fue la encargada de leer los registros del analizador y retornar sus valores esta función llama otra función de la biblioteca pymodbus llamada ModbusClient la cual fue explicada anteriormente, y también se usa el método connect para conectar el sistema, luego se usa el método read_holding_registers el cual también ya fue explicó su funcionamiento para luego ser asignados a las variables LB en la posición 0 y MB en la posición 1 luego estas variables son sumadas en la variable binario que usa la función format donde se le pasa por primer parámetro la variable MB que es el bit más representativo y como segundo parámetro se le indica que es binario se suma con otro

format donde se hace de igual manera solo cambiando el primer parámetro por LB que es el bit menos representativo y luego en una variable se usa la función convertirFloat pasándole como parámetro la variable binario y por último se guarda en una lista todas las variables requeridas a este dispositivo para retornar sus valores como podemos ver a continuación:

```
def Read_registers_analyzer():
modbus = ModbusClient(method='rtu', port ='/dev/ttyUSB0', baudrate=9600, timeout=1)
modbus.connect()

leer=modbus.read_holding_registers(0x30,2,unit=3)
LB = leer.registers[0]
MB = leer.registers[1]
binario = format(MB, '016b')+format(LB, '016b')
freq = convertirFloat(binario)
```

IMAGEN 27-FUNCIÓN PARA LEER REGISTROS DEL ANALIZADOR DE REDES.

Fuente: propia

```
lecturas = (freq,VLN1,VLN2,VLN3,I1,I2,I3,
            Corriente_total,energia_activa,
            factor_de_potencia)

return lecturas
```

IMAGEN 28-FUNCIÓN PARA LEER REGISTROS DEL ANALIZADOR DE REDES.

Fuente: propia

Y por último se crea una función para que se encargue de gestionar el envió de los datos a la base de datos no relacional Mongodb, a esta se le entrega como parámetro la información procesada previamente en una estructura de diccionario, esta función en primera instancia abre la base de datos usando la función pymongo.MongoClient de la librería pymongo, a pymongo.MongoClient se le pasa como parámetro el cliente de la base de datos luego se crea la variable db y se igual al cliente y por último se crea el post y el id como se puede apreciar en la siguiente imagen:

```python
def enviarMongo(data):
    client = pymongo.MongoClient("mongodb+srv://datainsitu:d4t41ns1tu*@monitoreo-cl5aq.mongodb.net/energia?retryWrites=true")
    db = client.energia
    posts = db.imsa
    post_id = posts.insert_one(post).inserted_id
```

IMAGEN 29-FUNCIÓN PARA ÉL ENVIÓ DE DATOS A INTERNET.

Fuente: propia

Después de crear las funciones necesarias para las lecturas de los registros, se procede a crear un algoritmo que se ejecute en el mismo instante que se le dé inicio al proceso de selección piezas metálicas y que inmediatamente comience a tomar estos registros, los guarde en una lista y luego los convierta en un diccionario para ser enviados a una base de datos en internet y otra base de datos interna, que no es más que un archivo .csv. A continuación, se presenta un diagrama de flujo que explica el funcionamiento del código encargado de leer, procesar, clasificar y enviar la información a una base de datos no relacional en red y de igual manera a una base de datos local:

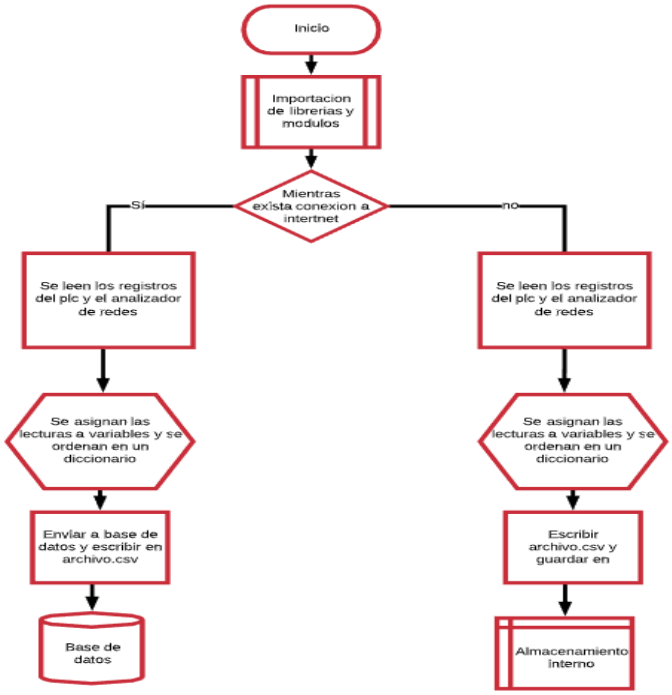

IMAGEN 30-DIAGRAMA DE FLUJO DEL PROGRAMA IMPLEMENTADO.

Fuente: propia

Implementación y Puesta a punto

En este punto se procedió a la implementación del desarrollo en la planta de selección de piezas metálicas y se realizaron unas pruebas, donde se tomaron lecturas de los registros del PLC y del analizador de redes con la planta en funcionamiento, donde encontramos que el desarrollo estaba concluido de manera exitosa por esta razón no se tuvo que hacer ajustes ni modificaciones, a continuación se muestra la planta e información tomada de esta planta:

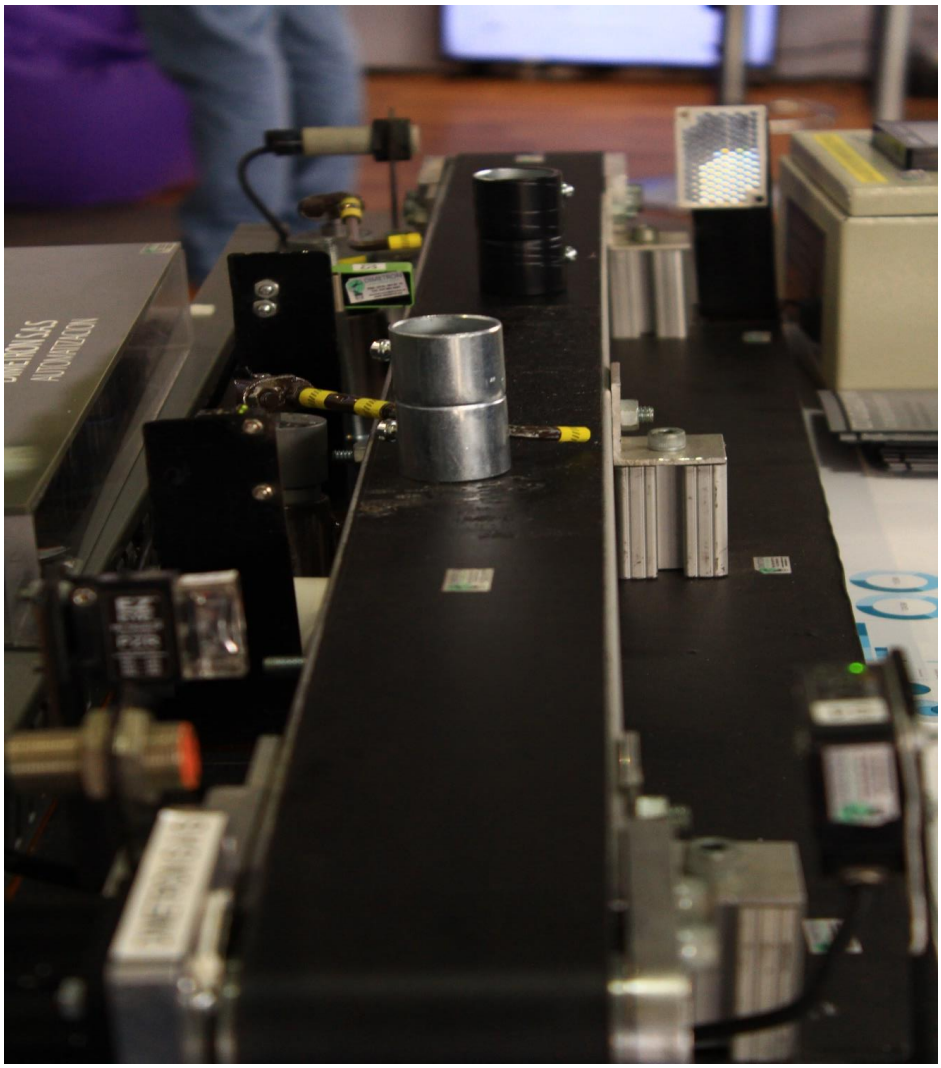

IMAGEN 31-PLANTA DE SELECCIÓN DE PIEZAS.

Fuente: propia

{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Frecuencia': 60.0146484375, 'Corrient
{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Frecuencia': 60.0146484375, 'Corrient
{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Frecuencia': 60.0263671875, 'Corrient
{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Frecuencia': 60.0380859375, 'Corrient
{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Frecuencia': 60.0263671875, 'Corrient
{'Voltaje En Linea 3': 0.0, 'Energia Actica': 1.5274153261140506e-43, 'Factor De Potencia': 0.0, 'Tiempo De M
{'Velocidad de Banda': 55, 'Energia Actica': 1.5274153261140506e-43, 'Voltaje En Linea 2': 121.75390625, 'C
{'Corriente Total': 0.0, 'Contador De Piezas': 151, 'Interrupciiones': 34, 'Voltaje En Linea 2': 122.08203125, 'F
{'Corriente Total': 0.0, 'Voltaje En Linea 3': 0.0, 'Tiempo De Mantenimiento': 9000, 'Contador De Piezas': 151
{'Voltaje En Linea 2': 122.310546875, 'Voltaje En Linea 3': 0.0, 'Interrupciiones': 34, 'Piezas Desechadas': 71,
{'Corriente En Linea 3': 0.0, 'Interrupciiones': 34, 'Corriente En Linea 1': 0.0, 'Frecuencia': 59.9921875, 'Tiem
{'Interrupciiones': 34, 'Corriente Total': 0.0, 'Tiempo De Mantenimiento': 9000, 'Voltaje En Linea 3': 0.0, 'Vol

IMAGEN 32-DATOS TOMADOS DE LA PLANTA.

Fuente: propia

4. RESULTADOS Y DISCUSIÓN

Ya con el desarrollo terminado e implementado se presentó el proyecto en expoingeniería del año 2018 donde con los datos tomados se podía tener conocimiento del consumo energético en tiempo real y el acumulado dando así la posibilidad de tener la trazabilidad del consumo en voltaje, corriente y potencia además también se podía tener acceso a la información del proceso como cantidad de piezas e interrupciones entre otros ya mencionados anteriormente, a continuación se puede ver el proyecto en expoingeniería:

IMAGEN 33-PROYECTO EN EXPOINGENIERÍA.

Fuente: propia

IMAGEN 34-PROYECTO EN EXPOINGENIERÍA.

Fuente: propia

Con la información tomada se presentaron las siguientes graficas:

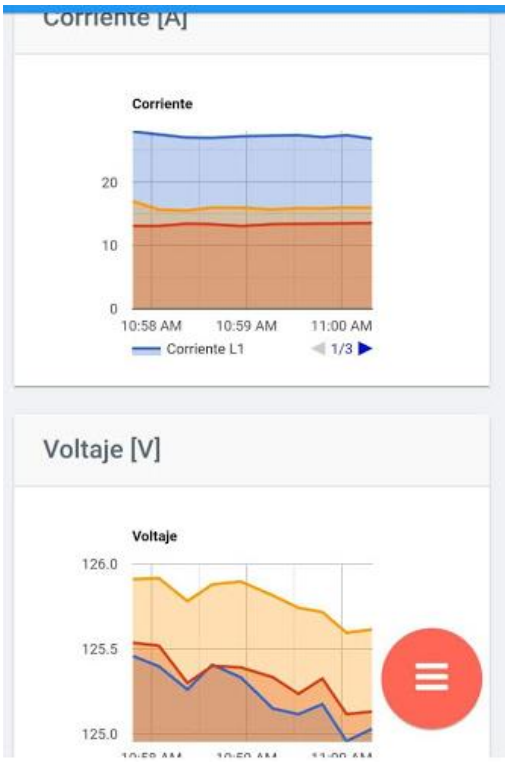

IMAGEN 35-DIAGRAMAS DE VOLTAJE Y CORRIENTE VS TIEMPO.

L1, L2 Y L3 RESPRESENTAN LA LÍNEA DE FASE EN UN SISTEMA TRIFÁSICO, EN EL EJE X TIEMPO Y EN EL EJE Y VOLTAJE Y CORRIENTE.

Fuente: propia

En la anterior imagen se puede observar 6 graficas distribuidas en dos partes, en la superior se puede observar las gráficas del consumo en corriente de todo el pabellón azul donde me encontraba tomando las medidas de las variables energéticas de todas las líneas de fase, fase 1 circuito de iluminación que se le nombro L1 representada con la línea de color azul, fase 2 circuito de alimentación que se le nombro L2 representada con la línea de color café y la fase 3 circuito de alimentación nombrada L3 representada con la línea de color rojo, este consumo se graficó en Amperios vs tiempo con una preciosidad más o menos de 10 minutos por medida de igual manera se le otorgó el mismo código de colores para las medidas de voltaje, algo que se puede observar en estas graficas es que las distribuciones de cargas en los circuitos no están equilibradas a pesar de tener dos circuitos independientes para alimentación de equipos se pudo notar que el circuito de iluminación tiene más consumo en corriente pero menos en voltaje, esto deja ver que esta sobre cargada y necesita un rediseño y distribuir mejor las cargas ya que los equipos de iluminación no son equipos que demanden una alta carga como la maquinaria de procesos industriales, también se puede observar que pasa todo lo contrario con los circuitos de alimentación están bien balanceados y no dejaron ver ninguna anomalía a pesar de tener cientos de equipos conectados en la exhibición de ingeniería.

Con los datos de voltaje y corriente se crearon también las gráficas de consumo total de energía un gráfico de torta y consumo de energía por día en todo el pabellón azul un gráfico de barras:

IMAGEN 36-DIAGRAMAS DE CONSUMO ENERGÉTICO POR DÍAS.

Fuente: propia

También se creó un informe del proceso en tiempo real donde podías ver todas las variables implicadas en el proceso tales como voltaje, corriente, consumo, piezas aceptadas, numero de mantenimientos, interrupciones y número de piezas en el proceso de la siguiente forma:

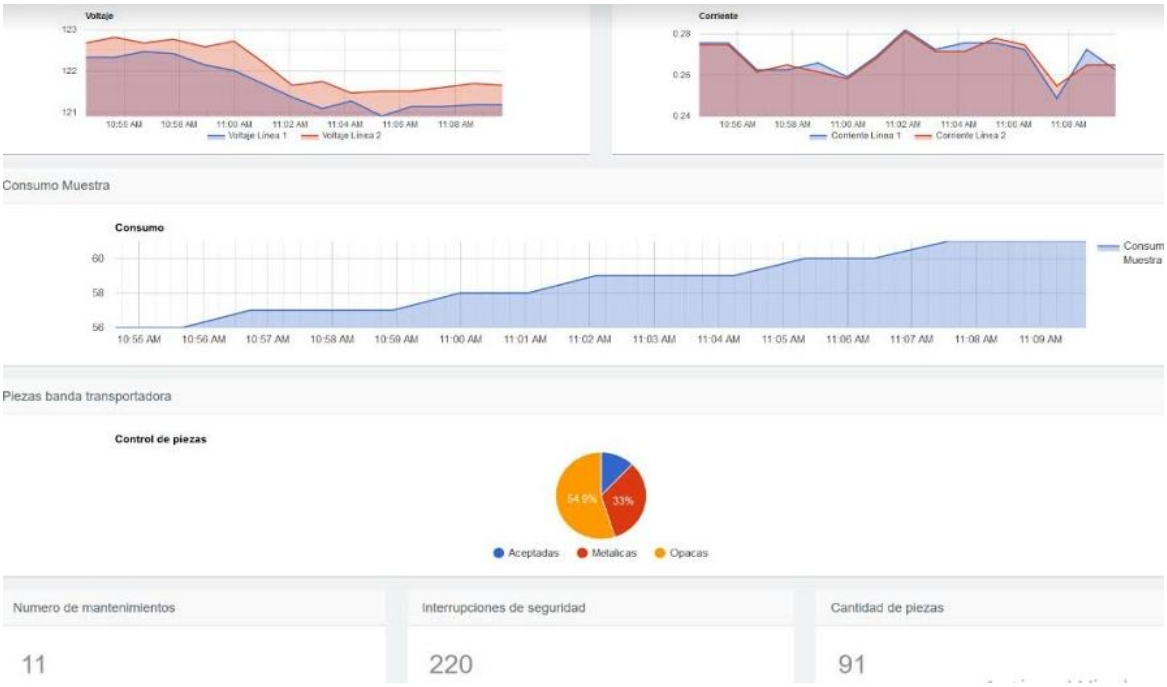

IMAGEN 37-INFORME.

DIAGRAMAS DE VOLTAJE Y CORRIENTE VS TIEMPO DE L1, L2 DONDE L SIMBOLIZA LA LÍNEA DE FASE, EN EL EJE X TIEMPO Y EN EL EJE Y VOLTAJE Y CORRIENTE, CON UN DIAGRAMA DE TORTA DONDE SE MUESTRA EN PORCENTAJES LAS PIEZAS ACEPTADAS, METÁLICAS Y OPACAS Y CON UN PANEL DONDE SE PUEDE VISUALIZAR EL NÚMERO DE MANTENIMIENTOS, INTERRUPCIONES DE SEGURIDAD Y CANTIDAD DE PIEZAS.

Fuente: propia

Esta parte es el plus de este trabajo ya que como muestra la imagen se puede ver toda información acerca del proceso, se tiene acceso a información de consumo energético por pieza permitiendo también saber el costo energético de cada pieza y de esta manera, no solo saber el costo de material y mano de obra si no también el costo energético además se puede saber el estado de la producción permitiendo a un gerente o encargado tomar decisiones con base a toda la información tomada del proceso.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

En este proyecto se creó e implemento una red MODBUS RTU maestro-esclavos. Para esto se buscaron los dispositivos comerciales más adecuados para su integración en el proyecto; se eligió la tarjeta de desarrollo Raspberry con un conversor USB a RS-485 para la facilitación en la conexión con un PLC y una analizador de redes, tras desarrollar todos los módulos e instalar todas las librerías a nivel software, conectarlos a nivel hardware, se ha logrado la comunicación entre el maestro y los esclavos por lo que podemos dar por exitosos los objetivos de este trabajo.

- La definición del maestro es de suma importancia en el presente trabajo, ya que este es el encargado de centralizar toda la información.
- Direccionar y dejar claro el número correspondiente a cada esclavo es clave para tener comunicación entre el maestro y sus esclavos
- Se hicieron pruebas en el sistema de comunicación dejando ver la importancia de que tiene la ficha técnica de los componentes ya que brindaron información clara y concisa en el caso del PLC, que dirección de memoria se podía usar para el protocolo de comunicación ModBus y en el caso del analizador en que registros de memoria se encontraban las variables solicitadas.
- Para un futuro se propone usar esta información para tomar decisiones en cuanto producción ya que con esta se sabe la capacidad exacta de producción y cuando se debe hacer un paro para mantenimiento dando así la posibilidad de proyectar y planificar de una manera ideal todo tipo de pedidos, además también se propone usar la información de consumo para monitoreo y diagnóstico de problemas en la planta ya que al saber el

consumo en corriente en correcto funcionamiento se puede comparar con el funcionamiento diario y al encontrar una alza en la corriente se evidencia que hay un sobre esfuerzo que no es más que un problema encontrado.

REFERENCIAS

(Tinoco, 2016)

(s.f.). Obtenido de https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Barastegui, J. J. (23 de 9 de 2017). *bibliote de ingenieria sevilla*. Obtenido de http://bibing.us.es/proyectos/abreproy/91400/fichero/Memoria+TFG+JJRB.pdf

Barastegui, J. J. (2017). *Diseño y desarrollo de una red MODBUS RTU.* Sevilla: Escuela Técnica Superior de Ingeniería.

G., R. D. (2011). *Implementación de una red Modbus RTU.* Cartagena: UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR.

I + D electronicas. (23 de 10 de 2018). *I + D electronicas*. Obtenido de Fuente: https://www.didacticaselectronicas.com/index.php/comunicaciones/conversores/usb-rs232-rs485/conversor-usb-a-rs-485-detail

Leonardo Barrera D., C. C. (2016). *Sistema domótico básico utilizando la tarjeta Raspberry pi.* Bogota: Universidad Distrital Francisco José de Caldas.

Ramírez, L. I. (2018). *Análisis, diseño e implementación de una plataforma triada integrada por Raspberry Pi, Arduino y dispositivos móviles con conectividad en red local, para la enseñanza de las ciencias naturales: estudio de caso en Física.* Medellin: Univercidad Nacional De Colombia .

Tinoco, M. H. (2016). *Desarrollo e implementación de una red de datos basada en Modbus y Ethernet para autómatas industriales.* Sevilla: Universidad de Sevilla.

universidad politecnica de sevilla. (18 de 12 de 2013). *museo de informatica*. Obtenido de https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Valencia, U. P. (28 de 12 de 2013). *Historia* . Obtenido de Historia de la informatica: https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

FIRMA ESTUDIANTES _____

_____

_____

FIRMA ASESOR _____

FECHA ENTREGA: 22/11/2019 entrega definitiva del informe final aprobado

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____