ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SERVIDOR DE DATOS INDUSTRIAL PARA EL PROTOCOLO MODBUS RTU PARA LECTURA Y ESCRITURA DE DATOS BOOLENOS, ENTEROS Y FLOTANTES, A DISPOSITIVOS ESCLAVOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

BRYAN ESTEBAN CARGUA MEDINA

DIRECTORA: DRA.-ING. SILVANA DEL PILAR GAMBOA BENÍTEZ

CODIRECTOR: MBA. ANA VERÓNICA RODAS BENALCÁZAR

Quito, Septiembre 2022

AVAL

Certificamos que el presente trabajo fue desarrollado por Bryan Esteban Cargua Medina, bajo nuestra supervisión.

Firmado electrónicamente por:
SILVANA DEL PILAR GAMBOA BENITEZ

NOMBRE DIRECTOR

Dra.-Ing. Silvana del Pilar Gamboa Benítez

Firmado electrónicamente por:
ANA VERONICA RODAS BENALCAZAR

NOMBRE CODIRECTOR

MBA. Ana Verónica Rodas Benalcázar

DECLARACIÓN DE AUTORÍA

Yo, Bryan Esteban Cargua Medina, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Bryan Esteban Cargua Medina

DEDICATORIA

Dedico este trabajo con mucho cariño a mis padres Guillermo Cargua y Elvia Medina que por el apoyo que me han brindado durante todo este tiempo, a mis hermanos Juan, Cristian, Jefferson y Yadira por estar siempre a mi lado en todo momento.

AGRADECIMIENTO

Mi profundo agradecimiento a la Escuela Politécnica Nacional, a mis profesores por todas las enseñanzas impartidas, en especial a la Dra.-Ing. Silvana Gamboa y la MBA. Ana Rodas por ser mi guía durante todo el proceso de desarrollo de la tesis, quien con su ayuda permitieron el desarrollo de este trabajo.

ÍNDICE DE CONTENIDO

RESUMEN

A nivel industrial, existen una variedad de softwares para implementar sistemas de control supervisorios que trabajan con diversos protocolos de comunicación industrial, uno de los más importantes es el protocolo MODBUS con sus variantes ASCII, RTU y TCP. Existen servidores comerciales que trabajan con este protocolo, pero su costo de adquisición resulta ser elevado lo cual limita su utilización en PYMES (Pequeñas y Medianas Empresas).

El trabajo planteado permite el desarrollo de un servidor de datos para MODBUS RTU, que contará con los 8 principales códigos de función establecidos por el estándar para la lectura y escritura de datos booleanos, enteros y flotantes. Para su desarrollo se usará software libre tanto para la creación del servidor como para la gestión de una base de datos, usando JAVA con el IDE Eclipse y MySQL. El uso de software libre permitirá reducir los costos de este tipo de servidores, haciendo que estos sean más accesibles para las PYMES con la ventaja que estos servidores pueden ser mejorados ya que también se puede acceder a su código de programación.

El programa tiene dos modos de funcionamiento, el primero trabaja en modo maestro MODBUS RTU, que permite la comunicación con un dispositivo esclavo MODBUS RTU para probar los 8 códigos de función, el segundo modo de funcionamiento es el modo servidor de datos, el cual realiza una comunicación continua con el dispositivo esclavo y una base de datos, la que a su vez se conecta a un HMI generado en el software Wonderware Intouch.

PALABRAS CLAVE: MODBUS RTU, Comunicación, servidor, Base de datos, dispositivos esclavos, tramas, JAVA, MySQL.

ABSTRACT

At industrial level, there are a variety of software to implement supervisory control systems that work with various industrial communication protocols, one of the most important is the MODBUS protocol with its variants ASCII, RTU and TCP. There are commercial servers that work with this protocol, but their acquisition cost is high, which limits their use in SMEs.

The proposed work allows the development of a data server for MODBUS RTU, which will have the 8 main function codes established by the standard for reading and writing Boolean, integer, and floating data. For its development, free software will be used for both the creation of the server and for the management of a database, using JAVA with the Eclipse IDE and MySQL. The use of free software will allow to reduce the costs of this type of servers, making them more accessible to SMEs with the advantage that these servers can be improved since their programming code can also be accessed.

The program has two modes of operation, the first works in MODBUS RTU master mode, which allows communication with a MODBUS RTU slave device to test the 8 function codes, the second mode of operation is the data server mode, which performs a continuous communication with the slave device and a database, which in turn connects to an HMI generated in the Wonderware Intouch software.

KEYWORDS: MODBUS RTU, Communication, Server, Database, Slave Devices, Frames, JAVA, MySQL.

1. INTRODUCCIÓN

Los protocolos de comunicación, dentro de los procesos industriales, son ampliamente usados a nivel mundial debido a que posibilitan el intercambio de información entre los actuadores conocidos como dispositivos de campo y los dispositivos de control por lo que permiten el trabajo en redes permitiendo el aumento de la eficiencia de los procesos.

Dentro de las comunicaciones industriales más usadas a nivel mundial se tiene el protocolo MODBUS, Profibus, Profinet y DeviceNet [38]. El Ecuador no es una excepción en el uso de redes de comunicación industriales y actualmente existe un crecimiento en su uso, pero un problema que se puede encontrar para la implementación de estos tipos de comunicaciones en industrias ecuatorianas, sobre todo en pequeñas y medianas industrias es el costo de adquisición del software necesario para estos tipos de comunicaciones, debido a que la mayoría de estos softwares son propietarios y llegan a tener un costo muy elevado, como se podrá observar más adelante en la tabla 1.3.1, lo cual dificulta mucho la implementación de sistemas de control automatizados en dichas industrias [39].

El protocolo MODBUS, y especialmente la variación RTU del protocolo MODBUS es la más antigua de las formas de comunicación industrial, por lo que en la actualidad varios proveedores de software industrial están dejando de proveer servidores de datos para este protocolo, justificado en que éste está siendo desplazado por su versión sobre TCP/IP que es MODBUS TCP. Sin embargo, en la industria nacional, MODBUS RTU sigue siendo utilizado, encontrándose maquinaria que aun trabaja bajo este protocolo, e implementadas sobre un interfaz serial usando interfaces RS-232 y RS-485.
El presente proyecto desarrolla un servidor de datos basado en el protocolo MODBUS RTU utilizando el software libre JAVA usando el IDE Eclipse, donde se implementan los 8 códigos de función principales para la lectura de variables booleanas, enteras y flotantes, para que sea capaz de comunicarse con cualquier dispositivo que trabaje bajo este protocolo, a la vez que el servidor también interactúa con una base de datos en MySQL, misma que tiene la capacidad de conectarse con un software para generación de HMIs como lo es Intouch. Por último, se realiza una prueba de funcionamiento del servidor de datos y se le compara su desempeño con un servidor de datos comercial llamado DASMBSerial.

1.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Desarrollar un servidor de datos industrial para el protocolo MODBUS RTU para lectura y escritura de datos booleanos, enteros y flotantes a dispositivos esclavos.

Los objetivos específicos del Proyecto Técnico son:

- Estudiar la normativa del protocolo MODBUS RTU y determinar la estructura de las tramas de los códigos de función para lectura y escritura de datos booleanos, enteros y flotantes.
- Seleccionar el software de programación de código abierto para el desarrollo del servidor de datos y el sistema de registro de los datos adquiridos en base a un análisis previo
- Desarrollar la programación necesaria para implementar el servidor de datos MODBUS RTU, así como la interface para configuración del driver, utilizando software de código abierto.
- Desarrollar la programación necesaria para implementar la aplicación para el registro de los datos adquiridos por el servidor de datos.
- Validar el desempeño del driver desarrollado con esclavos Modbus y HMIs industriales comerciales y compararlo con el funcionamiento de un software comercial con funciones similares.

1.2 ALCANCE

En el siguiente Proyecto Técnico, se estudiará la normativa del protocolo MODBUS RTU y se determinará la estructura de las tramas de los códigos de función para lectura y escritura de datos booleanos, enteros y flotantes.

Se estudiará los métodos necesarios para establecer conexión con esclavos MODBUS sobre una interfaz RS-485.

Se determinará la estructura de las tramas de los códigos de función Modbus para lectura y escritura de datos booleanos, enteros y flotantes y los parámetros que deberán ser ingresados por el usuario como parte de la configuración.

Se seleccionará el software de programación de código abierto para el desarrollo del servidor de datos (driver de comunicación) y el sistema de registro de los datos adquiridos.

Se desarrollará el servidor de datos en el software de código abierto previamente establecido.

Se determinará la estructura de las tablas de datos a generar en la base de datos.

Se implementará una base de datos que servirá como sistema de registro de los datos accedidos por el servidor desarrollado desde los dispositivos MODBUS RTU.

Se implementará un interfaz gráfico de usuario para la configuración del servidor de datos, en donde se ingresen los parámetros de configuración del servidor propuesto.

Se implementará la comunicación entre el sistema de registro y el servidor de datos, para que posteriormente se pueda realizar lectura y escritura de datos.

Se realizará las pruebas de funcionamiento del servidor de datos con el software simulador de Modbus RTU en modo esclavo llamado Modbus Slave.

Se realizará las pruebas de funcionamiento del servidor de datos con dispositivos industriales que puedan trabajar con Modbus RTU en modo esclavo como lo es el PLC Micrologix ML1100 disponible en el laboratorio de Redes Industriales.

Se realizará las pruebas de funcionamiento integrando el servidor de datos y el sistema de registro desarrollado con esclavos Modbus RTU conectados a una interface RS-485, así como al Software HMI Wonderware Intouch, cuya licencia se dispone en la EPN, que adquirirá datos desde el sistema de registro y se comprobará el funcionamiento del servidor desarrollado como parte integral de un sistema de control industrial.

Se establecerá comparaciones entre el funcionamiento del software desarrollado con software comercial con similares funciones tal como DASMBSerial de Wonderware con el objetivo de validar el funcionamiento del servidor de datos desarrollado y su sistema de registros, y se comprobará que cumplen con las características de conexión y comunicación.

1.3 MARCO TEÓRICO

Esta sección será dedicada para explicar conceptos teóricos necesarios para comprender el presente proyecto, se iniciará con una breve explicación de lo que son los sistemas SCADA y como están constituidos, siguiendo con conceptos sobre el protocolo MODBUS, el software de JAVA y sus entornos de desarrollo integrados, conceptos de MySQL y sobre el software Wonderware.

1.3.1 SERVIDORES DE DATOS INDUSTRIALES COMERCIALES

Los servidores de datos son una parte fundamental dentro de un SCADA, dado que realizan la comunicación en tiempo real con dispositivos de campo, para intercambiar datos

de un proceso industrial y poder interactuar con dicho proceso, todo esto a nivel de software, en la actualidad existen una gran variedad de servidores de datos bajo diversos protocolos y marcas de equipos industriales, por ejemplo, se tiene servidores de datos para el protocolo MODBUS y sus variantes ASCII, RTU y TCP, servidores de datos para dispositivos Siemens, Allen Bradley, etc.

A continuación, se detallan las características de 3 servidores de datos comerciales que trabajan con el protocolo MODBUS RTU.

Tabla 1.3.1. Ejemplos de servidores de datos para MODBUS RTU comerciales.

Servidor	**Características**	**Licencia**
Modbus Suite de KEPServerEx	Comunicación con dispositivos MODBUS ASCII, RTU, TCP o PLUS mediante comunicación ethernet, serial RS-232 y R-485. Trabaja bajo el sistema operativo de Windows. Integración con sistemas SCADA, bases de datos y plataformas IoT [1].	El costo de la licencia es de $ 1497 También puede pagar el costo de la licencia como suscriptor con un valor de $500 por año [2]. Posee una versión demo con la restricción que funciona solo durante 2 horas. [3].
MasterOPC Universal Modbus Server de MasterOPC	El servidor soporta comunicación Modbus ASCII, RTU, TCP y Modbus sobre TCP mediante convertidores Ethernet-COM. Integración con equipos IoT. Integración con bases de datos mediante ODBC [4]. Trabaja con el sistema operativo de Windows.	El costo de la clave para el uso del software es de $239, mientras que el costo de una llave USB que permite su uso en diferentes computadores es de $272, sin IVA. Tiene un modo de prueba con tags ilimitados y funciona durante 1 hora. Su compra se la puede realizar en línea ya que no poseen sede en Ecuador. Dispone también de una versión gratuita que soporta hasta 32 tags, que pueden servir para pequeños procesos [4].
MBSerial DA Server de Wonderware	Permite la integración con equipos de Schneider Electric como los PLC TSX momentum, y los Modicon. También puede comunicarse con cualquier PLC que maneje el protocolo MODBUS RTU sea de 4, 5 o 6 dígitos. Conexión con software OPC, DDE y Suitelink. Trabaja con el sistema operativo de Windows [5].	El costo de la licencia de los servidores y de Intouch y para 500 tags es de $ 3541, para 3000 tags el valor es $5000 [6]. Intouch cuenta con una versión demo que dura 2 horas [40].

El presente proyecto presenta el desarrollo de un servidor de datos industrial bajo el protocolo MODBUS RTU utilizando software libre para su diseño y desarrollo, además de almacenar valores de un proceso en bases de datos en el software también libre de MySQL, la base de datos podrá comunicarse con el software HMI de Intouch para su monitoreo.

Más adelante se explicarán las diferencias que existe entre el protocolo MODBUS RTU y el protocolo MODBUS TCP.

1.3.2 SISTEMAS SCADA

Los sistemas de control supervisorio y adquisición de datos, más conocido como SCADA, son sistemas que constan de software, hardware y un medio de comunicación que permiten controlar un proceso de manera local o remota, a la vez que recopila información de sus operaciones en tiempo real. En la figura 1.3.1 se observa la estructura de un sistema SCADA.

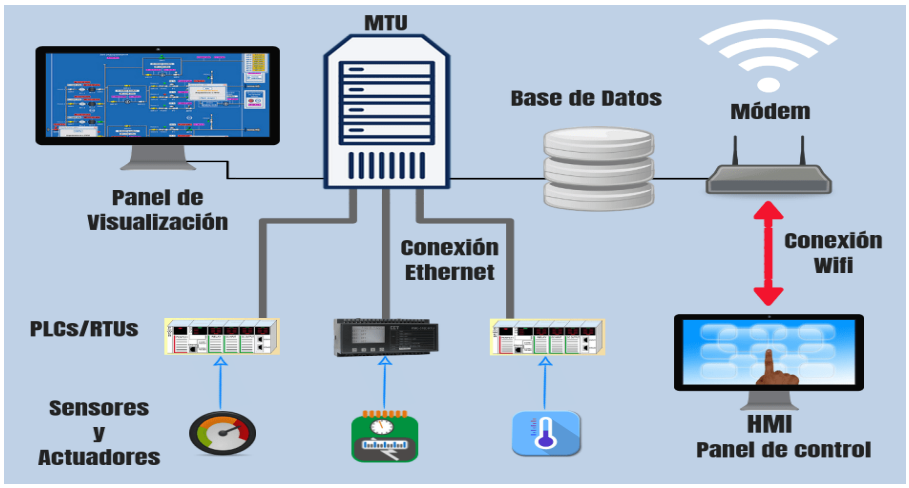

Figura 1.3.1. Estructura de un sistema SCADA [41].

Los sistemas SCADA están conformados por:

1.3.2.1 HMI

Es la interfaz humano-máquina, la cual se encarga de mostrar todos los datos del proceso para que un operador pueda monitorear y controlar un proceso industrial [42].

1.3.2.2 TERMINAL MAESTRA (MTU)

Es la encargada de recolectar y centralizar la información del proceso industrial, también es la encargada de enviar instrucciones a las terminales remotas mediante protocolos de comunicación, las MTU pueden ser computadoras o PLCs y estas deben tener la capacidad

de almacenar información, por lo general la información recopilada por la MTU es almacenada en una base de datos, las MTU también pueden ser servidores de datos [40].

1.3.2.3 TERMINAL REMOTA (RTU)

Puede estar conformado por PLCs o microcontroladores que obtienen las señales del proceso, es decir, son los dispositivos que interactúan con los sensores y actuadores, a su vez que intercambia esta información con los MTU [42].

1.3.2.4 DISPOSITIVOS DE CAMPO

Formado por sensores y actuadores, donde los sensores se encargan de recopilar las variables físicas o químicas del proceso, en cambio los actuadores son los que se encargan de accionar dispositivos mecánicos [42].

1.3.2.5 RED DE COMUNICACIÓN

La red de comunicación es la encargada de interconectar las RTUs, MTUs y el HMI mediante diferentes tipos de redes, por ejemplo, redes ethernet, redes seriales, wi-fi o fibra óptica, etc [42].

El presente proyecto utiliza una red de comunicación de tipo serial bajo el protocolo MODBBUS RTU para comunicarse entre el servidor de datos y los RTUs.

1.3.3 PROTOCOLO MODBUS.

El nacimiento del protocolo MODBUS se da en el año de 1973 ante la necesidad de comunicar equipos industriales, fue desarrollado por la compañía MODICON, la cual años atrás también desarrolló el primer controlador lógico programable (PLC) con el mismo nombre [22].

El protocolo fue estandarizado y declarado de libre acceso en el año de 1979 ante la dificultad que presentan los softwares propietarios por el costo de licencias de uso, al ser de libre acceso y de fácil implementación, empezó a ganar popularidad a nivel industrial, sigue siendo ampliamente utilizado en la actualidad.

MODBUS es un protocolo de mensajería con arquitectura maestro-esclavo conocida también como cliente-servidor, ubicada en la capa de aplicación o capa 7 del modelo OSI, esto permite que pueda implementarse en diferentes capas físicas, específicamente puede ser implementado en una red serial (MODBUS RTU/ASCII) o en una red ethernet (MODBUS TCP) [23].

El protocolo es diseñado específicamente para la intercomunicación entre equipos inteligentes, estos pueden ser: PCs, HMIs, SCADAS, PLCs, dispositivos de campo y dispositivos de entradas y salidas, en donde se tiene un único maestro y múltiples esclavos.

Debido a su arquitectura cliente-servidor, la comunicación siempre será en pares, es decir el maestro se comunica con un dispositivo esclavo a la vez, cabe destacar que el maestro

es que realiza las solicitudes a los esclavos y procesa sus respuestas, la comunicación únicamente se puede inicializar desde el maestro, los dispositivos esclavos nunca transmiten datos sin previa solicitud del maestro ni se comunican entre ellos [22].

En la figura 1.3.2 se puede observar la manera en la que se da la comunicación.

Figura 1.3.2. Arquitectura maestro-esclavo.

1.3.4 TRAMA DE DATOS EN EL PROTOCOLO MODBUS

El protocolo MODBUS fue desarrollado inicialmente para ser utilizado en una red de tipo serial, pero esta se ha ido adaptando con el paso del tiempo permitiendo a esta dividirse en múltiples capas, pudiéndose en la actualidad implementarse en redes Ethernet, esto ha generado que existan cuatro variantes de ésta, que son:

- MODBUS ASCII
- MODBUS RTU
- MODBUS TCP
- MODBUS PLUS

Estas variaciones del protocolo provocaron adaptaciones que se diferencian del protocolo original, que definía la unidad de datos de protocolo (PDU) que es fija en todas las versiones de MODBUS, y la capa de red, que es la parte en donde se diferencian las distintas versiones, estas en conjunto definen la unidad de datos de aplicación (ADU) [24].

En la figura 1.3.3 se observa la estructura general de la trama MODBUS.

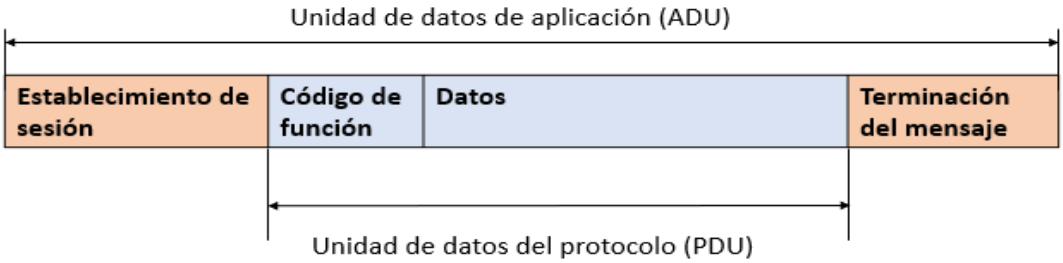

Figura 1.3.3. Estructura general de la trama modbus.

1.3.5 PDU de MODBUS

El PDU es la parte principal de la trama Modbus ya que aquí se ingresa el código de función y sus datos específicos, su tamaño no es fijo y su longitud dependerá de la solicitud que se desea realizar, pero esta no deberá superar la longitud máxima de 253 bytes [24].

Para los códigos de función de lectura, la estructura del PDU de la solicitud estará conformada por el código de función, la dirección de inicio de la lectura y la cantidad de datos a leer, mientras que el PDU de la respuesta consta del código de función, el contador de bytes de respuesta y los datos de respuesta, tal como se puede observar en la gráfica 1.3.4.

Figura 1.3.4. PDU para códigos de función de lectura.

Para el caso de códigos de función de escritura de datos, la estructura del PDU varía entre los códigos de escritura única y escritura múltiple, para los códigos de función de escritura simple, el PDU de la solicitud está formado por el código de función, la dirección de inicio y el valor a escribir, mientras que el PDU de la respuesta es el mismo que el de la solicitud tal como se observa en la figura 1.3.5.

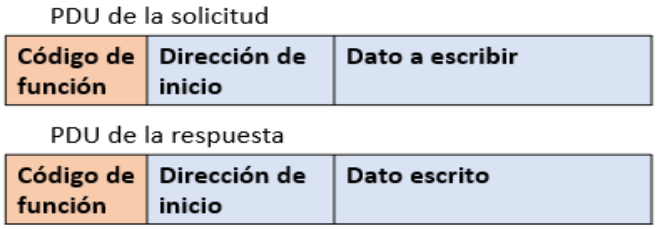

Figura 1.3.5. PDU para códigos de función de escritura única.

Para los códigos de función de escritura múltiple, el PDU de la solicitud tendrá el código de función, la dirección de inicio, la cantidad de datos a escribir, cantidad de bytes a escribir y los datos a escribir, mientras que el PDU de la respuesta tendrá el mismo código de función de la solicitud, la dirección de inicio de la escritura de datos y la cantidad de datos escritos como se puede observar en la figura 1.3.6.

PDU de la solicitud

Código de función	Dirección de inicio	Cantidad de datos a escribir	Contador de bytes a escribir	Datos a escribir

PDU de la respuesta

Código de función	Dirección de inicio	Cantidad de datos escritos

Figura 1.3.6. PDU para códigos de función de escritura única.

El PDU en el caso que se produzcan códigos de excepciones, estará formado por el código de función de excepción, seguido de los códigos de errores generados durante la solicitud tal como se ve en la figura 1.3.7.

PDU de la respuesta

Código de función de excepción	Códigos de errores

Figura 1.3.7. PDU para códigos de función de excepción.

1.3.6 ADU de MODBUS

La ADU corresponde a la totalidad de la trama MODBUS, esta variará de acuerdo a la versión del protocolo MODBUS que se esté usando, a continuación, se mostrará el ADU para cada versión de MODBUS.

1.3.6.1 Modbus ASCII

Es la ADU más compleja de implementar, su trama es más extensa que las otras debido a que cada dato es enviado como un carácter ASCII, la trama inicia con un ":", seguido por la dirección del dispositivo, luego se tiene el PDU que se detalló anteriormente, pero en formato ASCII, seguido por la comprobación de redundancia longitudinal (LRC) que verifica que la trama no haya sufrido alteraciones, por último, la trama termina con un ingreso de carro (CR) y línea de alimentación (LF) [24]. En la figura 1.3.8 se observa la estructura.

:	Dirección (ASCII)	MODBUS PDU (ASCII)	LRC (ASCII)	CR	LF

Figura 1.3.8. ADU MODBUS ASCII.

1.3.6.2 Modbus TCP

La ADU para MODBUS TCP está formado por el encabezado de protocolo de aplicación Modbus (MBAP) y el PDU de MODBUS, en la figura1.3.9 se puede observar la su estructura.

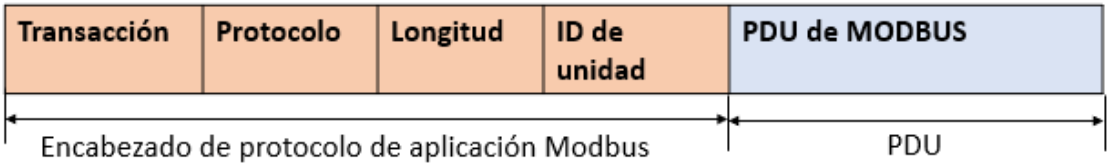

Figura 1.3.9. ADU MODBUS TCP.

El encabezado es pensado para adaptarse a redes ethernet, ya que esta puede soportar múltiples solicitudes al mismo tiempo, posee cuatro campos que son:

- Transacción
- Protocolo
- Longitud
- ID de la unidad

El primer campo identifica el número de transacción o solicitud que se está realizando, el protocolo que es un identificador cuyo valor comúnmente es igual a 0 para MODBUS TCP, pero puede tomar diferentes valores en caso de ser necesario una extensión, la longitud es un contador de bytes que indica la cantidad de bytes del PDU de Modbus, por último, tenemos el campo de identificador de unidad, que indica a cual dispositivo esclavo se le envía la solicitud. Para el protocolo MODBUS TCP no se tiene el campo de chequeo de errores dado que TCP ya tiene integrado un campo de verificación de errores [24].

1.3.6.3 Modbus RTU

La ADU para MODBUS RTU es la más simple de las tres, ya que su estructura consta de un encabezado que contiene la dirección del dispositivo esclavo, seguido del PDU de Modbus y termina con un chequeo de errores, que utiliza el método de comprobación de redundancia cíclica (CRC). Cabe destacar que para Modbus RTU es necesario tener un tiempo de silencio entre tramas de datos donde no exista ninguna comunicación en el medio físico (bus serial), el tiempo de silencio debe ser equivalente a 3.5 caracteres de silencio, donde un carácter de silencio es igual al tiempo que se demoraría en enviar un byte por el bus serial, el cual puede variar dependiendo de la velocidad de transmisión de datos, el tiempo mínimo definido por el estándar es de 2ms. La estructura de la ADU se muestra en la figura 1.3.10 [24].

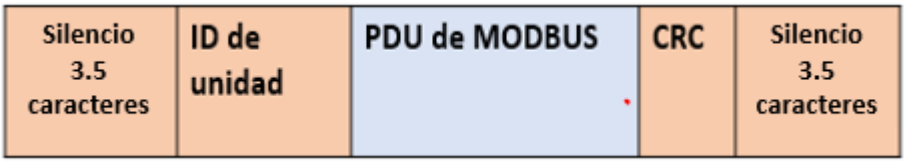

Figura 1.3.10. ADU MODBUS RTU.

Comprobación de Redundancia Cíclica (CRC)

Permite verificar el contenido de un flujo de datos, detectando afectaciones o modificaciones dentro del flujo de datos que pueda generarse durante la transmisión, ampliamente usado por la facilidad de implementar y la fiabilidad para detectar errores.

Tanto el dispositivo que recibe el mensaje como el que envía realizan este procedimiento, si el CRC no es igual en ambos lados de la comunicación el mensaje es descartado.

Para calcular el CRC se debe seguir el siguiente procedimiento.

1. Darle valor 0xFFFF al registro CRC.
2. Tomar el primer byte del flujo de datos a enviar, y realizarle la operación XOR con el byte menos significativo del registro CRC, almacenar el valor en el registro CRC.
3. Desplazar un bit a la derecha al registro CRC, rellenar el bit desplazado con 0.
4. Verificar el valor del desplazamiento realizado anteriormente, si este valor es 0 se almacena el CRC obtenido en el paso 3, caso contrario aplicar la operación XOR entre el CRC y el valor 0xA001, y almacenar el nuevo valor en CRC.
5. Repetir los pasos 3 y 4 con todos los bits de un byte.
6. Repetir desde el paso 2 al 5 hasta completar todos los bytes del flujo de datos.

El valor resultante en el CRC es la comprobación de redundancia cíclica, y este valor es el que se añade en la parte final de la ADU cuando se utiliza MODBUS RTU [28].

1.3.7 MAPA DE MEMORIA

El mapa de memoria en el protocolo MODBUS está dividido en 4 secciones o bloques de memoria, asociadas a 4 tipos de datos, que son:

- Coils o salidas discretas
- Contacts o entradas discretas
- Holding register o registros de retención
- Input register o registros de entradas

El estándar define subíndices de memoria para cada bloque con el fin de identificar el tipo de registro que maneja cada bloque, teniendo la distribución que se observa en la tabla 1.3.2.

Tabla 1.3.2. Bloques de memoria MODBUS.

Bloques de memoria	Subíndice de memoria	Tamaño	Acción
Coil	0	1 bit	Lectura/Escritura
Contact	1	1 bit	Lectura

Input register	3	2 bytes (16 bits)	Lectura
Holding register	4	2 bytes (16 bits)	Lectura/Escritura

Para cada bloque el estándar define que la capacidad máxima de registros es de 2^{16} elementos, es decir 65536 registros [24], en donde no todos los dispositivos cuentan con la cantidad de registros máximos, para lo cual cada dispositivo delimita la cantidad de registros que posee cada bloque de memoria mediante la cantidad de dígitos de la memoria, en la tabla 1.3.3 se puede observar y la cantidad de registros máximos para cada uno.

Tabla 1.3.3. Cantidad de registros en función al digito.

Cantidad de dígitos de memoria	Cantidad máxima de registros
4	999
5	9999
6	65535

Cada registro ocupa una dirección de memoria, en donde no siempre es lo mismo dirección y registro, es decir, existen dispositivos que representan la dirección usando base 0, por ejemplo, para el registro de entrada 1 su dirección será 0, cuando se usa base 0, se referencia la dirección definida por el protocolo, en cambio existen dispositivos que referencian la dirección en base al número de registro usando base 1, en la tabla 1.3.4, se ejemplifica estas dos maneras de representar una dirección.

Tabla 1.3.4. Direccionamiento de registros en MODBUS.

Dirección	Número de registro	Representación Base 1	Representación Base 0
0	1	30001	30000
1	2	30002	30001
2	3	30003	30002

1.3.8 ENDIANESS

Endianness u orden de bytes, establece la manera en la que se envía un registro, existen 2 maneras de enviar bytes que son:

- **Big endian**: arma el registro o dato iniciando del byte más significativo al menos significativo.
- **Little endian**: arma el registro o dato iniciando del byte menos significativo al más significativo.

El estándar define que un registro o Word (2 bytes) debe ser enviado en formato big endian, pero existen datos de múltiples registros, el más común de tipo flotante (4 bytes), pero se le puede sumar datos de tipo doble y String. Estos no tienen definido un orden de bytes dentro del protocolo, dicho de otra manera, no están estandarizados, por tal motivo estos datos son divididos en múltiples registros, para dar concordancia similar al usado en el estándar de registros Word, muchos dispositivos invierten el orden de los registros en el caso de múltiples bytes, como se puede ver en el ejemplo de la figura 1.3.11.

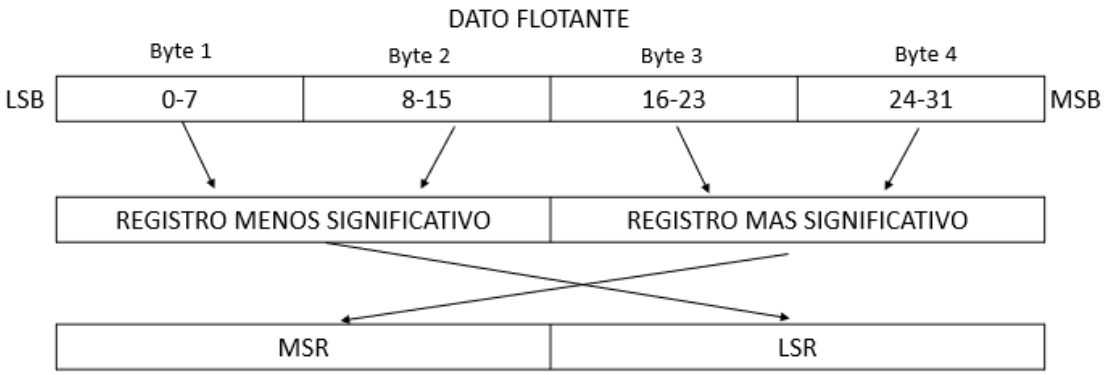

Figura 1.3.11. Endianness para datos de múltiples registros.

Muchos dispositivos realizan la inversión de registros para datos de múltiples registros, pero al no ser estandarizado la interpretación para este tipo de datos de múltiples registros depende del dispositivo maestro, por lo cual el parámetro Endianness es necesario a tener en cuenta para codificar y decodificar este tipo de datos correctamente.

1.3.9 CÓDIGOS DE FUNCIÓN

El código de función es una de las partes más importantes dentro de la trama Modbus, dado a que indica la acción que se realizará al dispositivo esclavo, los códigos de función pueden dividirse en 3 partes que son:

- **Códigos de función públicos:** son códigos de función definidos y validados por el estándar MODBUS, son códigos de uso público, es decir de libre uso y no pueden ser modificables.
- **Códigos de función definidos por el usuario:** son códigos de función definidos por el usuario, no estandarizados por MODBUS, es decir que cada usuario que realice su código de función le dará su propio uso y este puede variar conforme lo haya diseñado, con la oportunidad de solicitar su estandarización a la organización MODBUS.org.
- **Códigos de función reservados:** códigos de función desarrollados por empresas específicas y de uso en sus productos, estos códigos de función son de uso privado.

En la figura 1.3.12 se puede observar un esquema con la distribución de los códigos de función [25].

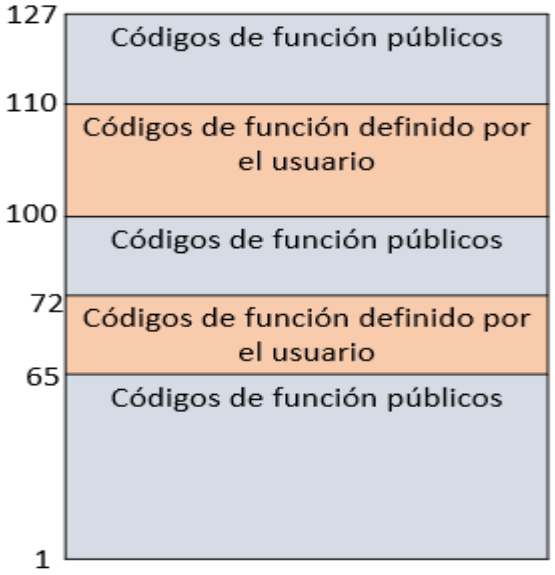

Figura 1.3.12. Distribución de los códigos de función MODBUS.

Dentro de los códigos de función públicos, los más importantes se muestran en la tabla 1.3.5.

Tabla 1.3.5. Códigos de función públicos en MODBUS.

Código de función	Descripción	Tipo de función
01	Leer registros coils	Acceso al dato
02	Leer registros contacts	
03	Leer registros de retención (holding)	
04	Leer registros de entrada	
05	Escribir un coil	
06	Escribir un registro holding	
15	Escribir múltiples coils	
16	Escribir múltiples registros holding	
22	Máscara de registro de escritura	
23	Leer o escribir múltiples registros	
24	Leer FIFO en cola	
20	Leer registro de archivo	Acceso a los registros de archivo
21	Escribir registro de archivo	
07	Leer estados de excepción	Diagnóstico
08*	diagnóstico	
11	Obtener contador de eventos COM	
12	Obtener registros de eventos COM	
17	Reportar ID del servidor	
43*	Leer identificación del dispositivo	
43*	Interfaz encapsulado de transporte	Otros

43*	Referencia general CANopen	

Los códigos de función marcados con “*”, requieren de un subcódigo de función adicional. Ahora bien, los códigos de función públicos más usados son los códigos de función de lectura y escritura, mismos que se usarán en este proyecto, los cuales se detallarán más a fondo a continuación.

- **FC=01. Leer Coils**

Permite leer el estado (ON/OFF) de los registros coils, su valor máximo de lectura establecido por el protocolo será de 2000 coils, aunque puede variar de acuerdo con el dispositivo, para armar el PDU de la solicitud se debe especificar el código de función, dirección de inicio y la cantidad de coils a leer, mientras que el PDU de la respuesta consta del mismo código de función, seguido de un contador de bytes de respuesta y el estado de los coils leídos.

Tabla 1.3.6. PDU solicitud y respuesta FC=01.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección de inicio	2 bytes	Contador de bytes	1byte
Cantidad de coils a leer	2 bytes	Estado de los coils	N bytes

Como se observa en la tabla 1.3.6, el PDU de la solicitud tendrá un tamaño de 5 bytes, mientras que el PDU de la respuesta no tendrá un tamaño fijo, esto depende de la cantidad de coils a leer, el dispositivo esclavo, empaqueta el valor de 8 coils por cada byte, donde el valor del cada bit representa el estado de un coil, si el bit es “1L” el coil está encendido, si es “0L” el coil está apagado.

- **FC=02. Leer Contacts**

Permite leer el estado de los registros contacts, el PDU de la solicitud y la respuesta tienen la misma estructura que la usada para código de función 01, se puede leer máximo 2000 contacts por cada trama, cada byte de respuesta contiene el valor de 8 contacts. Si el bit es “1L” el contact está encendido, si es “0L" el contact está apagado.

Tabla 1.3.7. PDU solicitud y respuesta FC=02.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección de inicio	2 bytes	Contador de bytes	1byte
Cantidad de coils a leer	2 bytes	Estado de los coils	N bytes

- **FC=03. Leer registros holdings.**

Permite la lectura del contenido de un conjunto de registros holding de un dispositivo esclavo, se utilizan 2 bytes por cada registro holding que se quiera leer, teniendo un límite de lectura de 125 registros, el orden de envío de los registros utiliza el Endianness big endian, es decir, se envía primero el byte más significativo. Para leer datos de tipo flotante el límite se reduce a la mitad, además se debe considerar el Endianness de los registros que se desea leer, es decir, el orden en el que se recibirán los registro que almacenan el dato flotante. En la tabla 1.3.8 se observa la estructura del PDU de la solicitud y respuesta para este código de función.

Tabla 1.3.8. PDU solicitud y respuesta FC=03.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección de inicio	2 bytes	Contador de bytes	1byte
Cantidad de Registros holding a leer	2 bytes	Valor de los Registros holding	N bytes (2 bytes por cada registro leído)

- **FC=04. Leer registros de entrada.**

Su función es leer los valores de los registros de entrada, la estructura del PDU de solicitud y respuesta es idéntico al utilizado para el código de función 03, el límite de registros de entrada que se pueden leer por mensaje de solicitud es de 125, en el caso de lectura de datos de tipo flotante el límite de lectura es de la mitad y además se debe considerar el Endianness del dispositivo esclavo, el PDU de la trama de solicitud y respuesta se puede observar en la tabla 1.3.9.

Tabla 1.3.9. PDU solicitud y respuesta FC=04.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección de inicio	2 bytes	Contador de bytes	1byte
Cantidad de registros de entrada a leer	2 bytes	Valor de los registros de entrada	N bytes (2 bytes por cada registro leído)

- **FC=05. Escribir un coil**

Permite la escritura de un registro coil, es decir permite dar el valor de “1L” o “0L” a un registro coil, en la tabla 1.3.10 se puede observar la estructura del PDU de la solicitud y de la respuesta.

Tabla 1.3.10. PDU solicitud y respuesta FC=05.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección del registro coil	2 bytes	Dirección del registro coil	2byte
Valor a escribir en el coil	2 bytes	Valor escrito en el coil	2 bytes

Para este código de función, el PDU de la respuesta es el mismo de la solicitud, es decir en caso de una escritura de un coil, si el dato se escribió correctamente en el dispositivo esclavo, este nos retornará el mismo PDU, el valor a escribir en un coil solo puede aceptar 2 tipos de valores, para “1L” el valor en hexadecimal debe ser 0xFF00, para “0L” el valor en hexadecimal que se debe escribir es 0x0000, cualquier otro valor resultará en un mensaje de excepción.

- **FC=06. Escribir un registro holding.**

Es usado para escribir un registro holding en el dispositivo esclavo, posee la misma estructura de PDU que el usado para el código de función 05, puede tomar un valor en representación hexadecimal desde 0x0000 a 0xFFFF, si es un entero con signo su valor será de -32767 a 32767, si es un entero sin signo el valor 0 a 65535, en la tabla 1.3.11 se puede observar la estructura del PDU de la trama de solicitud y respuesta.

Tabla 1.3.11. PDU solicitud y respuesta FC=06.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección del registro holding	2 bytes	Dirección del registro holding	2byte
Valor a escribir en el registro holding	2 bytes	Valor escrito en el registro holding	2 bytes

- **FC=15. Escribir múltiples coils.**

Permite modificar el valor de un conjunto de registros coils, el PDU de la solicitud constará del código de función, dirección del coil desde el cual se iniciará la escritura, un contador de bytes que indica la cantidad de bytes utilizados para escribir los registros coils, seguido del valor de los registros coils, donde cada byte contiene el valor de 8 coils, cada coil es representado por un bit, y el valor del bit será el valor del registro coil a escribir, para el PDU de la respuesta se tiene el código de función, la dirección del registro desde donde

se inicia la escritura de coils y la cantidad de coils escritos, esto se puede observar de mejor manera en la tabla 1.3.12.

Tabla 1.3.12. PDU solicitud y respuesta FC=15.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección del registro de inicio de la escritura	2 bytes	Dirección del registro de inicio de la escritura	2 bytes
Cantidad de coils a escribir	2 bytes	Cantidad de coils escritos	2 bytes
Cantidad de bytes	1 byte		
Valores a escribir	N bytes		

La cantidad máxima de registros coils que se puede escribir por mensaje es de 1968 (246 bytes), también es necesario considerar que no siempre se ocupan todos los bits que tiene en un byte, por ejemplo, deseo escribir 20 coils, para lo cual se necesitan 3 bytes, que en total dan 24 bits, si se presta atención, quedarán 4 bits no utilizados, los bits no utilizados deben tener el valor de 0.

- **FC=16. Escribir múltiples registros holding.**

Usado para escribir múltiples registros holding en dispositivos esclavos. El PDU de la solicitud está formado por el código de función, la dirección del registro desde el cual se iniciará la escritura, la cantidad de registros a escribir, la cantidad de bytes que se usarán para la escritura de los registros y el valor de los registros, considerando que el máximo de registros que se puede escribir por cada mensaje es de 123. El PDU de la respuesta contendrá el código de función, la dirección del registro inicial desde donde empieza la escritura y la cantidad de registros escritos tal como se observa en la tabla 1.3.13.

Tabla 1.3.13. PDU solicitud y respuesta FC=16.

Solicitud		**Respuesta**	
Código de función	1 byte	Código de función	1 byte
Dirección del registro de inicio de la escritura	2 bytes	Dirección del registro de inicio de la escritura	2 bytes
Cantidad de registros holding a escribir	2 bytes	Cantidad de registros holding escritos	2 bytes
Cantidad de bytes	1 byte		
Valores a escribir	N bytes		

Es importante destacar que este código de función es útil para escribir datos que ocupen múltiples registros, por ejemplo, un dato de tipo flotante el cual está formado de 4 bytes,

ocupando 2 registros de memoria, para el envío de este tipo de dato se debe considerar siempre el Endianness con el cual se envía, es decir, si se envía en formato big endian (el registro más significativo es enviado primero) o little endian (el registro menos significativo es enviado primero). Es importante ya que el orden de envío afecta el valor del dato flotante, a continuación, se explicará cómo está estructurado un dato de tipo flotante, donde se comprenderá la importancia del orden en el que es enviado un dato de tipo flotante.

Estándar IEEE 754 para representación de números con coma flotante

Establece la representación de un número real en formato binario para que estas puedan ser procesadas por dispositivos electrónicos (PCs, calculadoras, PLCs, etc.), definiendo dos formas de representación, precisión simple de 32 bits y precisión doble de 64 bits [26], en ambas se tiene la siguiente estructura y que se muestra en la Figura 1.3.13.:

- **Signo:** establece el signo del número, su tamaño es de 1 bit, donde 0 representa un número positivo y 1 un número negativo.
- **Mantisa:** contiene todos los dígitos del número real, su tamaño varía dependiendo su precisión, si es de precisión simple su tamaño es de 23 bits y si es de precisión doble su tamaño es de 52 bits.
- **Exponente:** indica el lugar de ubicación de la coma con relación a la mantisa, su tamaño será de 8 bits para precisión simple, y de 11 bits para precisión doble.

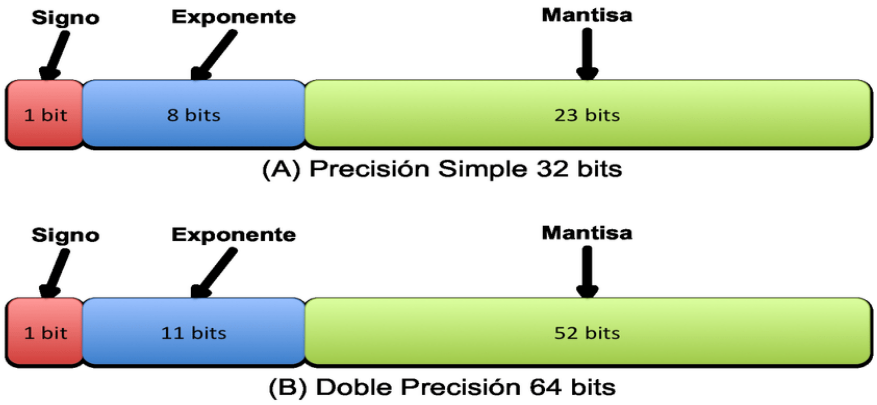

Figura 1.3.13. Estándar para representación de números con coma flotante [27].

Procedimiento para pasar un número flotante al estándar IEEE 754

A continuación, un ejemplo de cómo pasar el numero 80.125 flotante al estándar IEEE 754 de precisión simple.

1. Primero separar la parte entera y la parte fraccionaria, también definir el signo

Entero=80.

Fracción=0,125.

Signo= 0 (positivo).

2. Pasar cada parte del número a su representación en binario, quedando de la siguiente forma:

   80 = 1010000

   0,125 = 001

3. Unir los 2 números en binarios como si fueran un número real.

   80 ,125=1010000,001

4. Desplazar la coma a la izquierda hasta alcanzar el último “1”, contar el número de desplazamientos realizados y extraer la mantisa que es todo lo que se encuentra a la derecha de la coma y completar los bits restantes con 0, hasta tener 23 bits.

   1,010000001.

   Desplazamientos = 6.

   Mantisa=01000000100000000000000.

5. Para obtener el exponente sumar los desplazamientos más 127 y el resultado representar en binario.

   Exponente=127+6=133=10000101.

6. Por último, se procede a armar el número en el estándar IEEE 754, primero el signo luego el exponente y al último la mantisa.

   0 10000101 01000000100000000000000.

   Para mayor entendimiento pasar a su forma hexadecimal.

   42 A0 40 00.

Procedimiento para pasar de estándar IEEE 754 a número flotante

Se tomará el mismo número obtenido anteriormente.

0 10000101 01000000100000000000000

1. Separar el signo, el exponente y la mantisa.

Signo = 0

Exponente = 10000101

Mantisa = 01000000100000000000000

2. Pasar el exponente de binario a decimal, y al valor obtenido se le resta 127 para obtener los desplazamientos hacia la derecha que se debe dar en la mantisa para colocar la coma.

10000101=133

Desplazamiento = 133-127 = 6

3. Realizar los desplazamientos en la mantisa y colocar la coma (,).

010000,00100000000000000

4. Agregar “1” al inicio a la izquierda de la mantisa.

1010000,00100000000000000

5. Separar la parte entera de la fraccionaria.

Entero = 1010000

Fracción = 00100000000000000

6. Pasar de binario a decimal las 2 partes y unir, obteniendo el número flotante

1010000 = 80

001 = 125

1010000,001=80,125 [26].

1.3.10 CÓDIGOS DE EXCEPCIÓN

Los mensajes de excepción que se generan durante la comunicación mediante el protocolo Modbus se dan porque el dispositivo esclavo no puede procesar la solicitud realizada por parte del dispositivo maestro, el mensaje de excepción depende del tipo de error que exista durante la comunicación, estos se muestran en la tabla 1.3.14.

Tabla 1.3.14. Códigos de excepción [8].

Código de excepción	Error	Descripción
01	Función ilegal	El código de función recibido es erróneo. El esclavo no puede realizar la solicitud.
02	Dirección ilegal	La dirección de datos recibida es incorrecta.
03	Valor ilegal	El valor del dato no es un valor válido para el esclavo.
04	Fallo del esclavo	El esclavo no ejecuta la acción debido a un error irrecuperable.
05	Acuse de recibo	El esclavo recibe la solicitud, pero se demora mucho tiempo en procesarla.
06	Esclavo ocupado	El esclavo está ejecutando otro comando y no está disponible hasta terminarlo.
07	Acuse negativo	El esclavo no puede realizar la solicitud dada por el dispositivo maestro.
08	Error de paridad en memoria	El esclavo encuentra error de paridad al leer la memoria.
10	Ruta a la puerta de enlace no disponible	La puerta de enlace está sobrecargada o mal configurada.
11	El dispositivo de puerta de enlace no responde	El esclavo no se encuentra en la red

La estructura del PDU del mensaje de excepción está formada por el código de función de excepción y los códigos de excepción que se generen, tal como se puede observar en la figura 1.3.14.

Código de función de excepción	Códigos de excepción

Figura 1.3.14. PDU mensaje de excepción.

1.3.11 ALGORITMO DE LECTURA DE LA TRAMA DE SOLICITUD POR PARTE DE LOS DISPOSITIVOS ESCLAVOS

El dispositivo esclavo al momento que recibe una solicitud sea de lectura o escritura, realiza un proceso de análisis de la solicitud realizada, para lo cual analiza las partes del mensaje como lo son el código de función, la cantidad de registros que se desea leer o escribir, la dirección de inicio y si la solicitud fue realizada correctamente, esto se puede observar en las gráficas 1.3.15 (a) que muestra el algoritmo para la lectura de registros, y la gráfica 3.1.15 (b) que muestra el algoritmo para las lecturas de registros, este proceso lo realizan internamente los dispositivos esclavos.

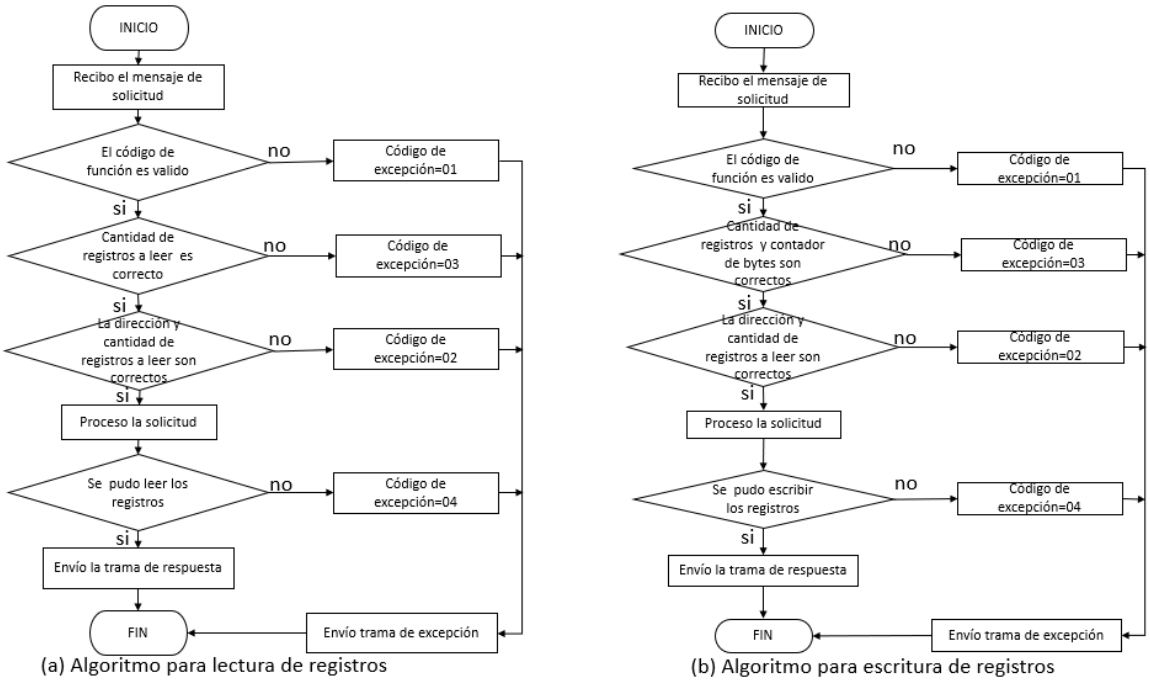

Figura 1.3.15. Algoritmos para lectura y escritura de registros por parte de un dispositivo esclavo MODBUS RTU.

1.3.12 JAVA

Java es un lenguaje de programación de libre acceso utilizado ampliamente en la actualidad, su programación está orientada a objetos y cuenta con una máquina virtual propia, permitiéndole ser ejecutada en cualquier tipo de equipo [29].

Es importante hablar de JAVA dado que es el lenguaje de programación que se utilizará para realizar el presente proyecto, se revisará la historia de JAVA, las características de la programación y sus entornos de desarrollo.

1.3.12.1 HISTORIA DE JAVA

Sus inicios se remontan a los años de 1991, cuando el equipo “Green team” de Sun Microsystems desarrolla un lenguaje de programación llamado “OAK” con características similares al lenguaje C++ que pueda ser utilizado en electrodomésticos [30].

En el año de 1994, cuando el internet se estaba popularizando, el equipo empezó a trabajar en un navegador web que terminó llamándose HotJava y en el año de 1995 Sun Microsystem publica la primera versión de JAVA la cual estaba incorporado en el navegador Netscape.

Adquiere el nombre de JAVA en el año de 1996 a la par que lanza la versión 1.0.2 del lenguaje, su licencia es liberada en el año 2006 haciéndola de libre acceso, Sun Microsystem es adquirida en el año 2010 por Oracle [31].

1.3.12.2 CARACTERÍSTICAS DEL LENGUAJE DE PROGRAMACIÓN JAVA

La principal característica de JAVA es que esta puede correr en cualquier sistema operativo de manera independiente sin necesidad de modificar su código, ya que dispone de su propia máquina virtual conocida como Java Virtual Machine (JVM) [31].

Su programación es orientada a objetos, es decir, permite estructurar el desarrollo de un programa en clases y objetos asociados a cada clase con el fin de poder ser reutilizados sin necesidad que el programador vuelva a reescribir código, facilitando el desarrollo de aplicaciones complejas dividiéndoles en tareas más simples, en este punto es necesario profundizar en el concepto de clase y objeto:

- **Clase:** es la definición de un conjunto de objetos con características y comportamiento similar, las clases se generan en el desarrollo del programa.
- **Objeto:** es una entidad que adopta atributos y métodos particulares, estos se crean mientras se ejecuta el programa [32].

Las características principales de la programación orientada a objetos son:

- **Encapsulación:** unión de atributos y métodos que pertenecen a un mismo objeto.

- **Polimorfismo:** es la capacidad de invocar un mismo método desde diferentes objetos [33].
- **Herencia:** forma en la que un objeto hereda las propiedades de otro sin necesidad de volverlas a definir.

1.3.12.3 ENTORNOS DE DESARROLLO DE JAVA

Son plataformas que contienen un conjunto de herramientas integradas que permiten el desarrollo de la programación de aplicaciones. Java posee 2 entornos de programación que son:

- **NetBeans**

Es un entorno de desarrollo que permite realizar aplicaciones de manera modular, soporta el desarrollo de programas en PHP, C y C++, tiene soporte multiplataforma que le permite trabajar bajo cualquier sistema operativo, posee todas las herramientas incorporadas, tiene incluido el soporte para base de datos SQL y MySQL.

- **Eclipse**

Es un entorno similar a NetBeans, también soporta PHP, C y C++, es multiplataforma, no posee herramientas integradas, por lo tanto, el programador tendrá la capacidad de instalar las herramientas necesarias para el desarrollo de programas, esto permite que el entorno de desarrollo se ejecute de manera más rápida, tiene compatibilidad con controladores para conectarse con bases de datos MySQL [34].

1.3.13 MYSQL

Es un gestor de bases de datos relacional, de libre acceso, fue desarrollado en el año de 1995 por la compañía sueca MySQL AB, la cual es adquirida en el año de 2009 por Sun Microsystem, misma empresa a la cual pertenecía JAVA que un año más tarde fue adquirida por Oracle.

Utiliza el lenguaje de consulta estructurada SQL (Structure-Query Language), el cual permite la administración de la base de datos MySQL [36].

Las características del MySQL son:

- **Portabilidad:** es un gestor multiplataforma y tiene compatibilidad con diferentes lenguajes de programación con C, C++, Java, PHP, etc. puede ser usado en entornos cliente/servidor dado a que el servidor está como un programa separado.
- **Soporte de diferentes tipos de columnas:** sus columnas soportan datos de tipo enteros, bytes, flotantes, dobles, char, varchar, time, date, etc. y puede tener registros de longitud fija y variable.

- **Soporte de funciones y sentencias SQL:** soporta los comandos SQL y no tienen riesgo de colisión con datos de las columnas que pudieran tener el mismo nombre del comando. También puede mezclar tablas de distintas bases de datos en una misma consulta.
- **Seguridad:** utiliza un sistema de privilegios y contraseñas, donde las contraseñas son cifradas al conectar con el servidor y solo el usuario puede ingresar al servidor.
- **Capacidad:** soporta bases de datos grandes, donde se puede tener 64 índices por tabla, lo que da una capacidad de aproximadamente 1024 columnas por tabla, soporta 60000 tablas o 50 millones de registros.
- **Conectividad:** los clientes se pueden conectar mediante sockets TCP/IP en diferentes plataformas. Posee conectores ODBC para programas clientes y conectores JBDC para clientes Java [18].

1.3.14 WONDERWARE

Wonderware es una compañía que ofrece software industrial basado en el sistema operativo de Microsoft, permitiendo la integración de los diferentes niveles de una empresa para el mejoramiento de la parte operativa, sus productos emblemas son el software HMI llamado Intouch y las plataformas del sistema Wonderware que abarcan los servidores de datos [35].

1.3.14.1 INTEGRACIÓN DE SERVIDORES DE OPERACIÓN INTOUCH

Es un software que permite el desarrollo de HMI, es el software HMI más conocido en el mundo y es usado por cerca de un tercio de las industrias a nivel mundial, usado para el desarrollo de aplicaciones visualización y control de procesos industriales, posee varias librerías gráficas y funciones que facilitan el diseño del HMI, Intouch está dividida en dos partes que son:

- **WindowViewer:** ejecuta la aplicación desarrollada en WindowMaker en tiempo real para su monitoreo.

- **WindowMaker:** permite el desarrollo de aplicaciones de entorno gráfico del interfaz humano-máquina de un proceso.

El programa ofrece varias herramientas para el desarrollo de un HMI, como librerías gráficas que permiten representar variables de un proceso industrial, desarrollo de

múltiples pantallas, desarrollo de la programación mediante scripts, tagnames para variables del proceso, etc.

Intouch tiene la capacidad de conectarse con servidores de datos desarrollados por Wonderware, así como la conexión con bases de datos externos a Wonderware mediante conectores ODBC [40].

2. METODOLOGÍA

En el presente capítulo se realizará el análisis estructural de las tramas de datos que se deben generar para los diferentes códigos de función que se maneja en la normativa MODBUS RTU, en donde se desglosará cada uno de los bytes que debe tener la trama de solicitud y la trama de respuesta, para determinar cuáles son los parámetros que se deben ingresar como usuario y la manera como se debe analizar la respuesta del dispositivo esclavo MODBUS RTU.

Como siguiente paso, se realizará el diseño del servidor de datos para MODBUS RTU, el cual será programado en el Software libre llamado JAVA, mediante el entorno de desarrollo integrado (IDE) ECLIPSE, el cual permitirá la comunicación tanto con dispositivos esclavos MODBUS RTU mediante la interfaz RS-485 de tipo serial y la comunicación con una base de datos generada en el Software de código abierto MYSQL.

Para terminar, se detallará la manera en la cual debe conectarse tanto al dispositivo esclavo MODBUS RTU, la conexión con la base de datos en MYSQL y la conexión con el HMI en el software Intouch.

2.1 DISEÑO DEL SERVIDOR DE DATOS MODBUS RTU A DESARROLLAR.

Antes del desarrollo del servidor de datos, es necesario establecer los requerimientos a tener en cuenta para su diseño, las cuales serán explicadas en la sección 2.1.1, en la sección 2.1.2 se detallará como se alcanzarán los requerimientos.

2.1.1 REQUERIMIENTOS.

A continuación, se enlistan los requerimientos a cumplir para desarrollar un servidor de datos que trabaje con software libre, bajo el protocolo MODBUS RTU mediante una interfaz serial RS-485 y que pueda satisfacer necesidades industriales.

- Se debe utilizar programas que trabajen mediante software libre, es decir que no sea necesario pagar licencias para su funcionamiento.
- Tener la capacidad de comunicarse con los dispositivos industriales mediante una interfaz RS-232/RS-485.

- Emplear el protocolo de comunicación industrial MODBUS RTU para la lectura y escritura de las variables de tipo booleanas, enteras y flotantes de los dispositivos industriales.
- Crear una base de datos para almacenar la información enviada por los dispositivos industriales.
- Recolectar la información de dispositivos industriales. y ésta sea almacenada en un histórico.
- Las variables de un proceso en tiempo real, para lo cual se requiere una tabla que almacene la información de forma continua.
- Para probar su funcionamiento, el servidor debe tener la capacidad de conectarse con una interfaz de control supervisorio o HMI, mediante la base de datos.

2.1.2 DISEÑO.

Partiendo de los requerimientos planteados en la sección anterior, se procedió al diseño del servidor de datos, para lo cual se eligió el software libre JAVA, bajo el entorno de desarrollo ECLIPSE, para el manejo y gestión de la base de datos, se seleccionó el software de código abierto MySQL, ambos softwares no requieren pago para su uso, lo que permite reducir el costo del servidor desarrollado.

Ahora bien, el servidor de datos debe manejar el protocolo MODBUS RTU para la lectura y escritura de los datos booleanos, entero y flotantes, por lo cual se desarrollaron los 8 códigos de función principales establecidos por el protocolo. El programa desarrollado se divide en dos modos de funcionamiento que son el modo maestro y el modo servidor, el modo maestro permite probar cada código de función de forma individual ya sea con el software simulador como con los dispositivos industriales (PLCs), éste no tiene conexión con la base de datos, es decir sólo establece comunicación entre el programa y el dispositivo esclavo mediante una interfaz RS-232/RS-485 como se indica en la figura 2.1.1.

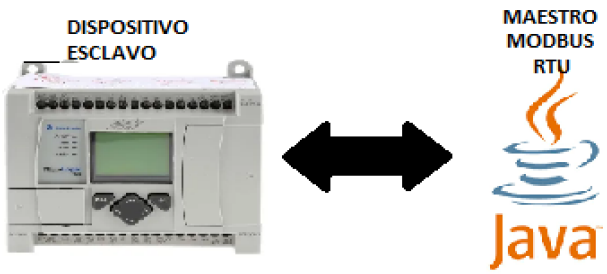

Figura 2.1.1. Modo de funcionamiento maestro.

El segundo modo de funcionamiento es el modo servidor, el cual es más completo y cumple todos los requerimientos detallados anteriormente, es decir establece una comunicación con el dispositivo esclavo y a su vez almacena la información recopilada en la base de

datos, todo esto de manera continua, este modo de funcionamiento permite monitorear un proceso completo, en la figura 2.1.2 se puede observar la arquitectura de este modo de funcionamiento.

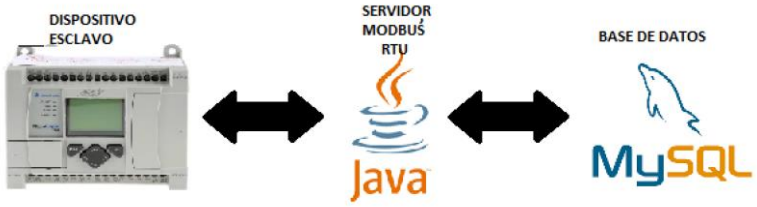

Figura 2.1.2. Modo de funcionamiento servidor.

La ventaja de tener una base de datos es que permite conectar el servidor con un sistema SCADA para el monitoreo de forma gráfica, para lo cual se utilizará el software INTOUCH de la plataforma WONDERWARE. En la figura 2.1.3 se puede observar la arquitectura completa al unirse a la plataforma INTOUCH.

Figura 2.1.3. Modo de funcionamiento servidor integrado a Intouch.

2.1.3 VENTANAS A IMPLEMENTAR.

Se implementarán 3 ventanas para el funcionamiento del proyecto propuesto, cada ventana es creada en clases tipo JFrame, que es la manera en cómo JAVA permite crear interfaces gráficas de usuario (GUIs), la primera ventana consta de la portada del proyecto y cuenta con 2 botones que direccionarán a los 2 modos de funcionamiento que se tiene, la segunda ventana la cual contiene al modo de funcionamiento como maestro MODBUS RTU, ésta permite comunicarnos directamente con dispositivos esclavos MODBUS y también con el simulador de MODBUS RTU, permitiendo probar los 8 códigos de función generados, mientras que la última ventana es la que trabaja como en modo servidor de datos MODBUS RTU, el cual permitirá realizar la comunicación con un dispositivo esclavo para llevar la información de las diferentes variables que maneje el dispositivo esclavo y éstas almacenarlas en una base de datos, que a su vez podrá ser monitoreada por medio del software Intouch.

La figura 2.1.4 muestra el diagrama de las clases utilizadas en el proyecto, la figura 2.1.5 muestra el diagrama de clases para el modo de funcionamiento como maestro MODBUS

RTU y la figura 2.1.6 muestra el diagrama de clases para el modo de funcionamiento como servidor de datos MODBUS RTU.

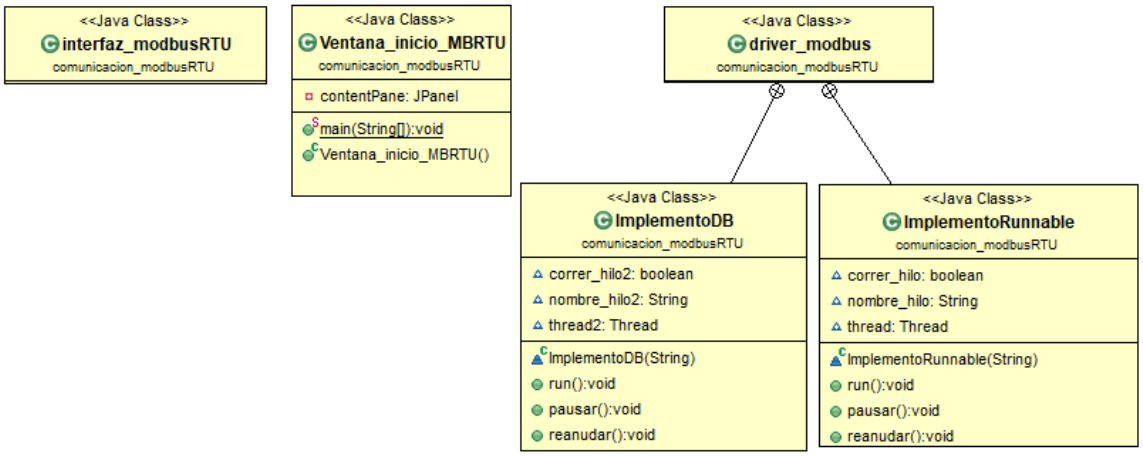

Figura 2.1.4. Diagrama de clases de las 3 ventanas.

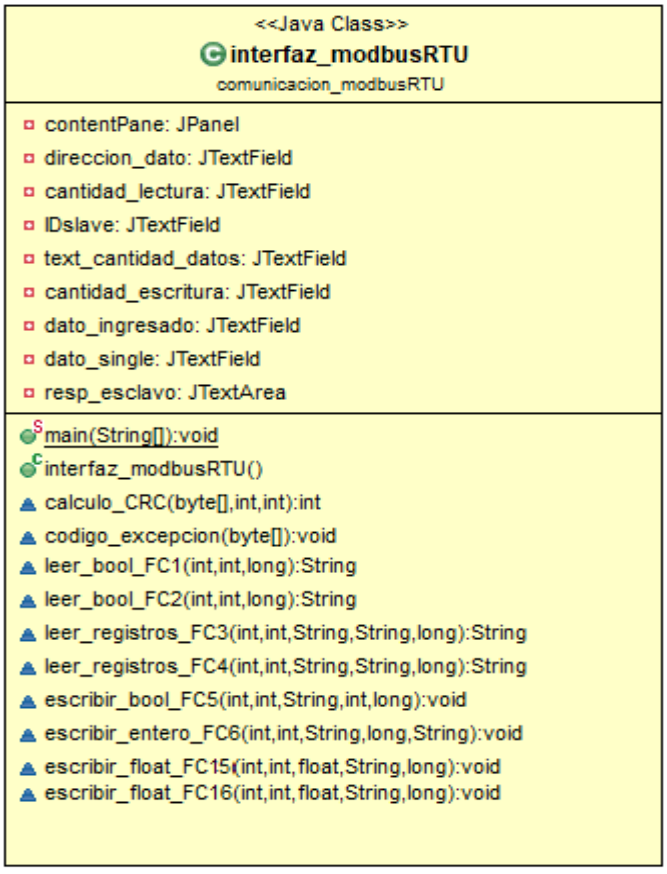

Figura 2.1.5. Diagrama de clases modo maestro MODBUS RTU.

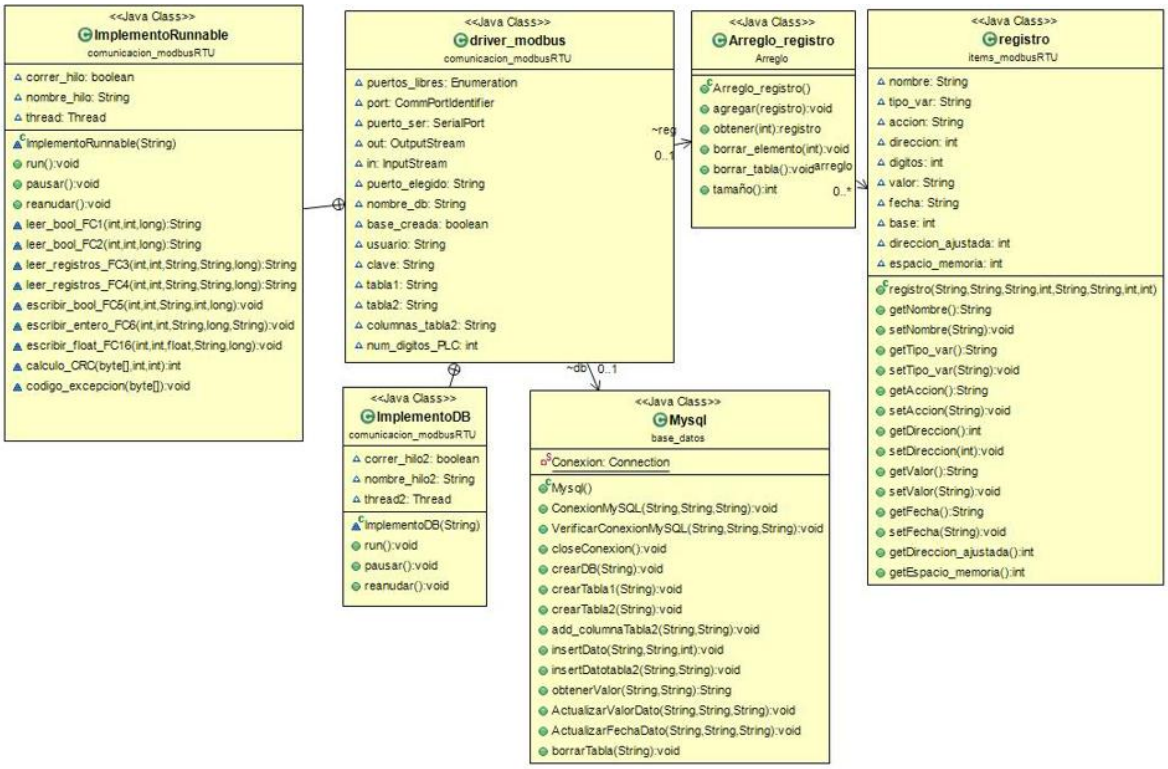

Figura 2.1.6. Diagrama de clases modo servidor de datos MODBUS RTU.

2.2 ESTRUCTURA DE LOS CÓDIGOS DE FUNCIÓN BAJO LA NORMATIVA MODBUS RTU.

El protocolo de mensajería MODBUS RTU es utilizado para comunicación maestro-esclavo por medio de una interfaz serial entre dispositivos inteligentes, mediante una estructura Request-Reponse, en donde el dispositivo maestro es el que envía la solicitud, mediante una trama de datos, al dispositivo esclavo esperando la respuesta del mismo, el cual enviará una trama de similares características en el caso de realizarse correctamente la comunicación.

La estructura general de las tramas MODBUS RTU es de la siguiente manera [24]:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DATO	CRC	Fin
3.5 Char	1 byte	1 byte	N bytes	2 bytes	3.5 char

Figura 2.2.1 Estructura de la trama MODBUS RTU.

En donde:

- **Inicio:** tiempo de espera antes de enviar el mensaje, el tiempo mínimo recomendado es el que se demoraría en enviar 3.5 bytes, el cual variará de acuerdo a la velocidad de transmisión en el que esté configurado el dispositivo.

- **Dirección del dispositivo:** su tamaño es de 1 byte, en el cual se encuentra la dirección del dispositivo esclavo, el cual puede ir desde 1 a 255.
- **Código de función**: su tamaño es de 1 byte, es el que describe la acción que debe realizar el dispositivo esclavo, para los cuales se tiene valores ya establecidos que se explicarán más adelante.
- **Dato:** su tamaño no es fijo, ya que dependerá del tipo del código de función, es decir esta parte constará de n bytes, el cual puede constar de la dirección del registro inicial y cantidad de registros a leer o escribir.
- **CRC:** definido como Comprobación de Redundancia Cíclica, es el campo de comprobación de errores, su tamaño es de 2 bytes, este campo permite verificar el contenido de la trama, este valor es verificado tanto por el maestro y el esclavo, para comprobar que el mensaje ha sido recibido correctamente y no ha sufrido alteraciones en el transcurso del envío del mensaje [7].
- **Fin:** tiempo de espera después de enviar el mensaje, el tiempo mínimo recomendado es el que se demoraría en enviar 3.5 bytes, el cual variará de acuerdo a la velocidad de transmisión en el que esté configurado el dispositivo.

Para los 8 códigos de función más importantes dentro del protocolo MODBUS, se generaron clases que permitan estructurar la trama, para lo cual los principales parámetros ingresados a las mismas son:

- Dirección del dispositivo
- Dirección del registro inicial
- Cantidad de datos a leer
- Cantidad de datos a escribir
- Tipo de endian
- Cantidad de dígitos del dispositivo, para determinar el tamaño de la memoria del dispositivo esclavo

Dependiendo del tipo función se ingresarán más o menos parámetros.

Se debe recordar que dentro del protocolo MODBUS RTU, cada uno de estos campos se trabaja por medio de bytes, es decir los todos los datos que se ingresen, ya sean booleanos, enteros o flotantes, es necesario transformarlos al formato byte.

Para el envío y recepción de las tramas, dentro de JAVA se usarán las librerías *InputStream* y *OutputStream*, para enviar la trama se usará el método *write (byte [] trama)* [9] que se tiene en la librería OutputStream, este método escribe directamente en el puerto serial y realiza el envío al esclavo, ahora para la lectura de la respuesta del dispositivo esclavo se

utilizará el método definido en InputStream *read (byte [] trama)* [10] el cual permite leer el buffer en el puerto serial.

2.2.1 LEER COILS (FC=01).

El presente código de función permite la lectura del estado (alto/bajo) de los coils en el dispositivo esclavo, los datos que ha de proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo
- Dirección del registro de inicio desde donde empezará la lectura
- Cantidad de coils que se quiere leer

Para lo cual se tendrá 2 tramas, una de solicitud y otra de respuesta que se detallarán a continuación.

2.2.1.1 Solicitud

La estructura de la trama de datos para el código de función 01, se muestra en la figura 2.2.1.

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	COILS A LEER	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.1. Estructura de la trama de solicitud para FC=01.

Mediante un ejemplo se presentará la estructura de la trama de solicitud realizada por el maestro.

Suponer que el dispositivo maestro requiere leer el estado de 13 coils del dispositivo esclavo cuyo ID es 4 y se quiere que la lectura inicie desde la dirección 11.

Quedando la trama de la solicitud como se observa en la tabla 2.2.1.1.:

Tabla 2.2.1.1. Trama de la solicitud del maestro para FC=01

Campo	Bytes	Valor (hex)
ID del dispositivo	1 byte	04
Código de función	1 byte	01
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	0A
Cantidad de coils (Hi)	1 byte	00
Cantidad de coils (Lo)	1 byte	0D
CRC (Lo)	1 byte	DD
CRC (Hi)	1 byte	98
TOTAL	8	-

En la tabla 2.2.1.1. se puede notar que los registros de 2 bytes como lo son la dirección de inicio, cantidad de coils y CRC, tienen un orden de envío de bytes, esto se debe a que el tamaño de un registro dentro del PLC es de 16 bits de longitud, es decir puede tomar valores de 0 a 65536 en teoría, pero como se dijo en el capítulo 1, este valor dependerá del número de dígitos del dispositivo.

En el caso de estos registros siempre se enviará primero la parte alta y luego la parte baja, a excepción del CRC ya que el protocolo mismo dicta que en el caso del CRC primero se envía la parte baja y luego la parte alta.

2.2.1.2 Respuesta

La trama de la respuesta del dispositivo esclavo tendrá la estructura que se observa en la figura 2.2.2:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	CONTADOR DE BYTES	ESTADO COILS LEIDOS	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	N bytes	2 bytes	3.5 char

Figura 2.2.2. Estructura de la trama de respuesta para FC=01.

La respuesta de la solicitud realizada será de la siguiente manera:

Tabla 2.2.1.2. Trama de la respuesta del esclavo para FC=01

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	04
Código de función	1 byte	01
Contador de bytes	1 byte	02
Dato	1 byte	0A
Dato	1 byte	11
CRC (Lo)	1 byte	B3
CRC (Hi)	1 byte	50
TOTAL	7	-

Como se observa en la tabla 2.2.1.2, el esclavo responde con la misma ID de dispositivo y el mismo código de función, a continuación se tiene el contador de bytes, el cual indica la cantidad de bytes que envió de respuesta el dispositivo, este valor dependerá de la cantidad de registros coils que haya solicitado el dispositivo maestro, dado que la respuesta del estado de los registros coils que son simplificados a 1bit por coil, en este caso al solicitar la lectura de 13 coils, y sabiendo que en cada byte se tiene 8 bits, la respuesta es que se enviarán 2 bytes con el valor de los estados de dichos coils, cabe recalcar que el primer bit menos significativo del primer byte del dato de respuesta es el coil de la dirección inicial solicitada los siguientes bits son los valores de las consiguientes direcciones, donde “1” significa ON y “0” OFF, en este caso existen 3 bits sobrantes en la

respuesta, estos bits toman el valor de 0 automáticamente, y por último están los bytes del chequeo de error o CRC.

2.2.2 LEER CONTACTS (FC=02).

El presente código de función permite la lectura del estado (alto/bajo) de las entradas discretas, conocidas también como contacts, en el dispositivo esclavo, los datos a proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo
- Dirección del registro de inicio desde donde empezará la lectura
- Cantidad de contacts que se quiere leer

Para lo cual se tendrá 2 tramas, una de solicitud y otra de respuesta que se detallarán a continuación.

2.2.2.1 Solicitud

La trama de la solicitud es similar a la utilizada para la FC=01, la única diferencia es el código de función, el cual permitirá acceder a los registros de memoria de los contacts en el dispositivo esclavo, quedando la estructura como se observa en la figura 2.2.3:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	CONTACTS A LEER	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.3. Estructura de la trama de solicitud para FC=02.

Como ejemplo, el dispositivo maestro requiere leer el estado de 13 contacts del dispositivo esclavo, de 5 dígitos, cuyo ID es 4, y se quiere que la lectura inicie desde la dirección 10011. Quedando la trama de la solicitud de la siguiente manera.:

Tabla 2.2.2.1. Trama de la solicitud del maestro para FC=01.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	04
Código de función	1 byte	02
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	0A
Cantidad de coils (Hi)	1 byte	00
Cantidad de coils (Lo)	1 byte	0D
CRC (Lo)	1 byte	99
CRC (Hi)	1 byte	98
TOTAL	8	-

En la tabla 2.2.2.1. se puede notar que los registros de 2 bytes como lo son la dirección de inicio, cantidad de coils y CRC, se sigue un orden de envío de bytes, esto se debe a que el

tamaño de un registro dentro del PLC es de 16 bits de longitud, es decir puede tomar valores de 0 a 65536 en teoría, pero como se dijo en el capítulo 1, este valor dependerá del número de dígitos del dispositivo.

En el caso de estos registros siempre se enviará primero la parte alta y luego la parte baja, a excepción del CRC ya que el protocolo mismo dicta que en el caso del CRC primero se envía la parte baja y luego la parte alta.

2.2.2.2 Respuesta

La trama de la respuesta del dispositivo esclavo tendrá la siguiente estructura como se observa en la figura 2.2.4:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	CONTADOR DE BYTES	ESTADO CONTACTS LEIDOS	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	N bytes	2 bytes	3.5 char

Figura 2.2.4. Estructura de la trama de respuesta para FC=02.

La respuesta de la solicitud realizada será de la siguiente manera:

Tabla 2.2.2.2. Trama de la respuesta del esclavo para FC=01.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	04
Código de función	1 byte	02
Contador de bytes	1 byte	02
Dato	1 byte	0A
Dato	1 byte	11
CRC (Lo)	1 byte	B3
CRC (Hi)	1 byte	14
TOTAL	7	-

Como se observa en la tabla 2.2.2.2, el esclavo responde con la misma ID de dispositivo y el mismo código de función que fueron enviados por el maestro, a continuación se tiene el contador de bytes, el cual indica la cantidad de bytes que envió de respuesta el dispositivo, este valor dependerá de la cantidad de registros contacts que haya solicitado el dispositivo maestro, dado que la respuesta del estado de los registros contacts son simplificados a 1 bit por contacts, en este caso al solicitar la lectura de 13 contacts, y sabiendo que en cada byte se tiene 8 bits, la respuesta tendrá 2 bytes con el valor de los estados de dichos contacts, cabe recalcar el primer bit menos significativo del primer byte del dato de respuesta corresponde al contact de la dirección inicial solicitada, los siguientes bits son los valores de las consiguientes direcciones, donde “1” significa ON y “0” OFF, en este

caso existen 3 bits sobrantes en la respuesta, estos bits toman el valor de 0 automáticamente, y por último el chequeo de error o CRC.

El diagrama de flujo para la lectura de registros contacts se observa en la figura 2.2.5.

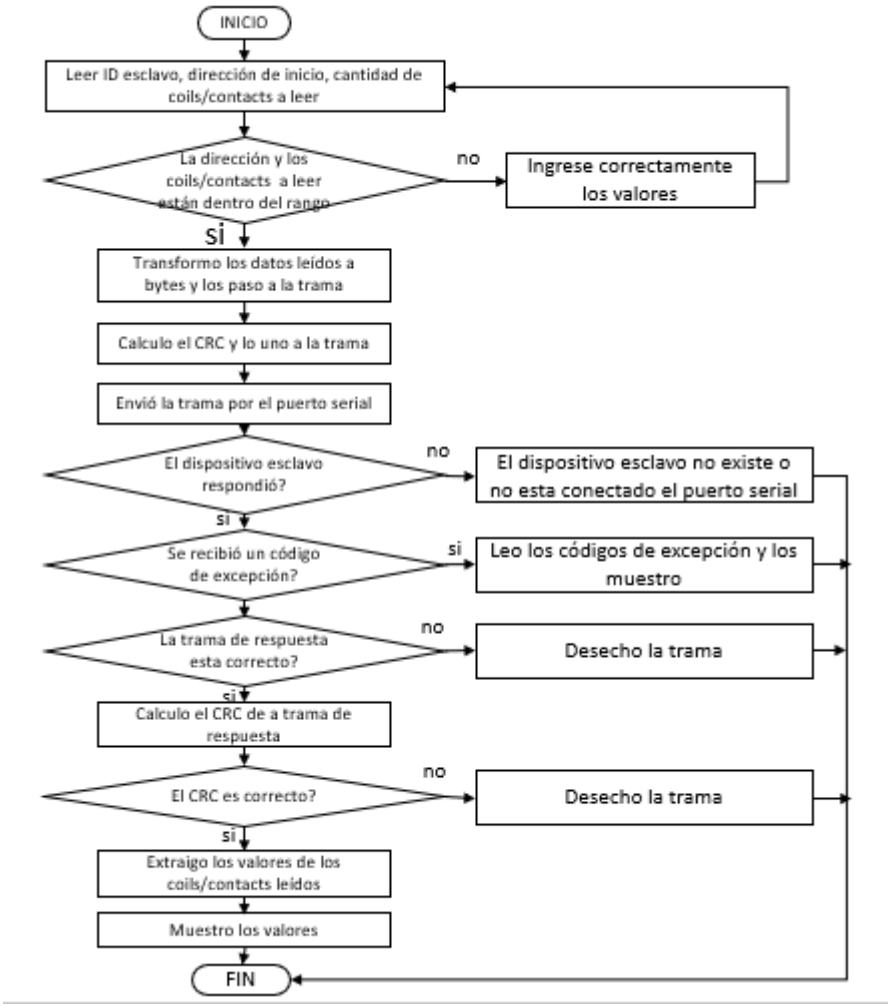

Figura 2.2.5. Algoritmo para generación de tramas de solicitud y lectura de tramas de respuesta para lectura de coils y contacts (FC=01 y FC=02).

2.2.3 LEER HOLDING REGISTER (FC=03).

El presente código de función permite la lectura de los registros de retención, conocidos también como holding, en el dispositivo esclavo, los datos que ha de proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo
- Dirección del registro de inicio desde donde empezará la lectura
- Cantidad de registros que se quiere leer

- Tipo de dato que ha leer (entero o flotante)

En caso de tener un dato de tipo flotante también es necesario definir el endian del dispositivo.

A continuación, se detallará la trama de solicitud y la de respuesta:

2.2.3.1 Solicitud

La trama de la solicitud permitirá acceder a los registros de memoria de los holdings en el dispositivo esclavo, quedando la estructura como se observa en la figura 2.2.6.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	REGISTROS HOLDING A LEER	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.6. Estructura de la trama de solicitud para FC=03.

Como ejemplo suponer que el dispositivo maestro requiere leer 2 registros de tipo entero del dispositivo esclavo, de 5 dígitos, cuyo ID es 1, y se quiere que la lectura inicie desde la dirección 40001. Quedando la trama de la solicitud de la siguiente manera:

Tabla 2.2.3.1. Trama de la solicitud del maestro para FC=03

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	01
Código de función	1 byte	03
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	00
Cantidad de registros holding (Hi)	1 byte	00
Cantidad de registros holding (Lo)	1 byte	02
CRC (Lo)	1 byte	C4
CRC (Hi)	1 byte	0B
TOTAL	8	-

En la tabla 2.2.3.1. se notar que los registros de 2 bytes como lo son la dirección de inicio, cantidad de coils y CRC tienen un orden de envío de bytes, esto se debe a que el tamaño de un registro dentro del PLC es de 16 bits de longitud, es decir puede tomar valores de 0 a 65536 en teoría, pero como se dijo en el capítulo 1, este valor dependerá del número de dígitos del dispositivo.

En el caso de estos registros siempre se enviará primero la parte alta y luego la parte baja, a excepción del CRC ya que el protocolo mismo dicta que en el caso del CRC primero se envía la parte baja y luego la parte alta.

En esta parte es necesario considerar la cantidad de registros que ocupan cada una de las variables, en el caso de las variables enteras, éstas ocupan un registro de memoria mientras que las variables de tipo flotantes ocupan 2 registros. Por ejemplo, si una variable

fuera de tipo entero ocuparía el registro 40001, en cambio si la variable fuera de tipo flotante esta ocuparía los registros 40001 y 40002.

2.2.3.2 Respuesta

La trama de la respuesta del dispositivo esclavo tendrá la estructura que se observa en la figura 2.2.7.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	CONTADOR DE BYTES	VALOR DE LOS REGISTROS HOLDING	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	N bytes	2 bytes	3.5 char

Figura 2.2.7. Estructura de la trama de respuesta para FC=03.

La respuesta de la solicitud realizada será la siguiente:

Tabla 2.2.3.2. Trama de la respuesta del esclavo para FC=03

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	01
Código de función	1 byte	03
Contador de bytes	1 byte	04
Registro1 (Hi)	1 byte	00
Registro1 (Lo)	1 byte	06
Registro2 (Hi)	1 byte	00
Registro2 (Lo)	1 byte	05
CRC (Lo)	1 byte	DA
CRC (Hi)	1 byte	31
TOTAL	9	-

Como se puede observar en la tabla 2.2.3.2, el esclavo responde con la misma ID de dispositivo y el mismo código de función que fueron enviados por el maestro, a continuación se tiene el contador de bytes, el cual indica la cantidad de bytes que envió de respuesta el dispositivo, en este caso como el esclavo solicitó leer 2 registros enteros, el esclavo enviará 4 bytes de respuesta, en donde el contador de bytes toma el valor de 4, el primer registro enviado corresponde al indicado como dirección inicial, siempre enviando primero la parte más significativa del registro y luego la menos significativa, para continuar con el siguiente registro hasta completar la cantidad de registros solicitados, por último se tiene los 2 bytes del chequeo de error o CRC.

En este punto es necesario considerar el tipo de dato que se está leyendo, en este ejemplo se tuvo un dato entero, el cual no genera ningún inconveniente al momento de rearmar el número, ya que siempre irá primero la parte más significativa del registro y luego la menos significativa, en el caso mostrado en la tabla se tiene que el registro 40001 tiene un valor hexadecimal 0x0006, que al pasarlo a entero da el número 6.

Pero en el caso de lectura de números flotantes se debe considerar necesariamente el tipo de endian para el rearmado del número flotante, tomar como ejemplo el número flotante 12.5, su valor hexadecimal es 0x41480000, si estuviera este valor en formato big endian, el registro más significativo seria 0x4148 y el menos significativo sería 0x0000, si al rearmar este dato no se lo rearma como big endian y se lo arma por el contrario como Little endian su valor hexadecimal seria 0x00004148, que al pasarlo a flotante da como resultado $2.49x10^{-41}$, aquí se observa cómo se altera completamente el dato al no leerlo correctamente. Como parte importante siempre se debe considerar el endian para este tipo de dato.

2.2.4 LEER INPUT REGISTERS (FC=04).

El presente código de función es similar al código de función FC=03, permite la lectura de los registros de entrada, conocidos también como input register, en el dispositivo esclavo los datos a proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo.
- Dirección del registro de inicio desde donde empezará la lectura.
- Cantidad de registros que se quiere leer.
- Tipo de dato que ha leer (entero o flotante).

En caso de tener un dato de tipo flotante, es necesario definir el endian del dispositivo.

A continuación, se detallará la trama de solicitud y la de respuesta:

2.2.4.1 Solicitud.

La trama de la solicitud es idéntica a las anteriores, la diferencia es el código de función, el cual permitirá acceder a los registros de memoria de los holdings en el dispositivo esclavo, quedando la estructura como se observa en la figura 2.2.8.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	REGISTROS DE ENTRADA A LEER	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.8. Estructura de la trama de solicitud para FC=04 [24].

Por ejemplo, el dispositivo maestro requiere leer 2 registros de entrada, de tipo entero del dispositivo esclavo, de 5 dígitos, cuyo ID es 1, y se quiere que la lectura inicie desde la dirección 30001. Quedando la trama de la solicitud de la siguiente manera:

Tabla 2.2.3.1. Trama de la solicitud del maestro para FC=03.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	01
Código de función	1 byte	04
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	00
Cantidad de registros de entrada (Hi)	1 byte	00
Cantidad de registros de entrada (Lo)	1 byte	02
CRC (Lo)	1 byte	71
CRC (Hi)	1 byte	CB
TOTAL	8	-

En la tabla 2.2.4.1. se puede notar que la estructura es idéntica a la que se utilizó en el código de función de lectura de holdings.

Para este código de función hay que tener las mismas consideraciones que se tuvo para la lectura de los holdings register, lo único que cambia es el código de función.

2.2.4.2 Respuesta.

La trama de la respuesta del dispositivo esclavo tendrá la estructura como se observa en la figura 2.2.9:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	CONTADOR DE BYTES	REGISTROS DE ENTRADA LEIDOS	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	N bytes	2 bytes	3.5 char

Figura 2.2.9. Estructura de la trama de respuesta para FC=04.

La respuesta de la solicitud realizada será:

Tabla 2.2.4.2. Trama de la respuesta del esclavo para FC=03.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	01
Código de función	1 byte	04
Contador de bytes	1 byte	04
Registro1 (Hi)	1 byte	00
Registro1 (Lo)	1 byte	06
Registro2 (Hi)	1 byte	00
Registro2 (Lo)	1 byte	05
CRC (Lo)	1 byte	DB
CRC (Hi)	1 byte	86
TOTAL	9	-

Como se observa en la tabla 2.2.4.2, el esclavo responde con la misma ID de dispositivo y el mismo código de función que fueron enviados por el maestro, a continuación se tiene el contador de bytes, el cual indica la cantidad de bytes que envió de respuesta el dispositivo, en este caso como el esclavo solicitó leer 2 registros enteros, el esclavo enviará 4 bytes de respuesta, en donde el contador de bytes toma el valor de 4, el primer registro enviado corresponde al indicado como dirección inicial, siempre enviando primero la parte más significativa del registro y luego la menos significativa, para continuar con el siguiente registro hasta completar la cantidad de registros solicitados, por último se tiene los 2 bytes del chequeo de error o CRC.

En la figura 2.2.10 se observa el diagrama de flujo utilizado para los códigos de función 03 y 04

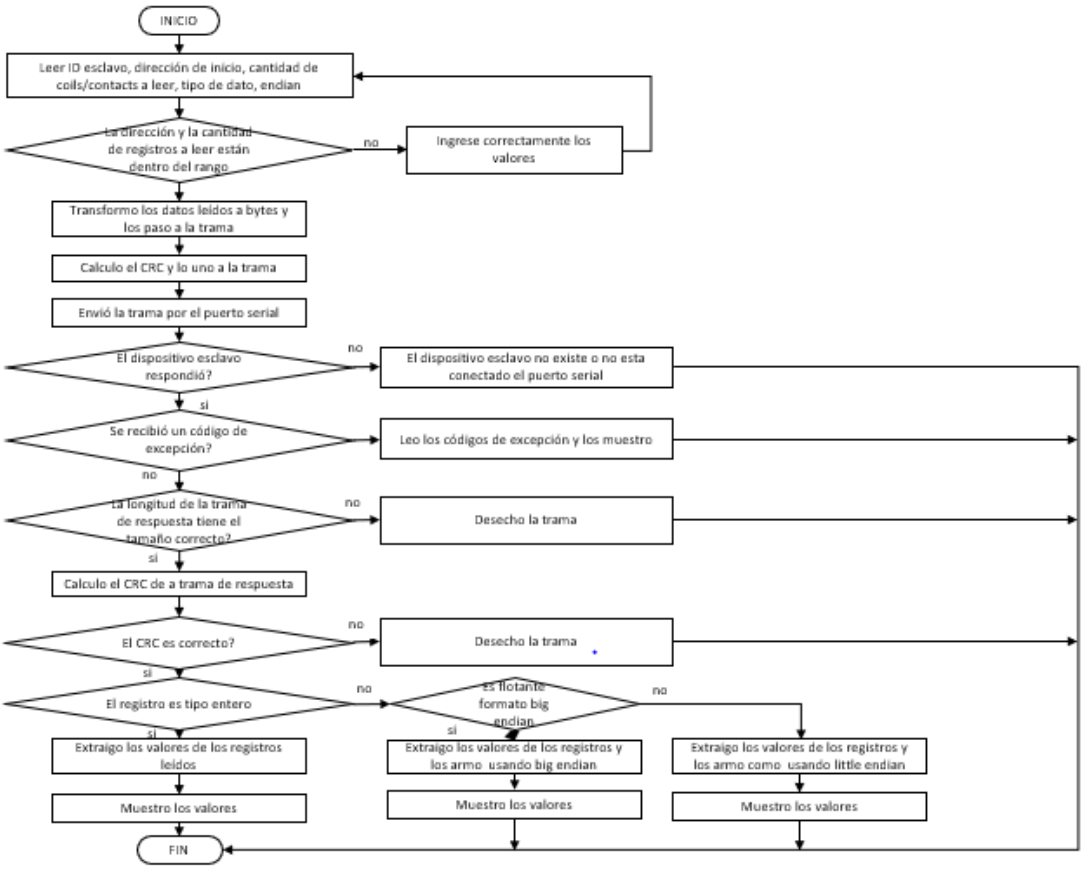

Figura 2.2.10. Algoritmo para generación de tramas de solicitud y lectura de tramas de respuesta para lectura de registros de entrada y registros holding (FC=03 y FC=04).

2.2.5 ESCRIBIR COIL (FC=05).

El siguiente código de función permite escribir ON u OFF en un coil, donde los datos a proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo.

- Dirección del registro de donde se realizará la escritura.
- Valor del coils a escribir (ON/OFF).

La trama de solicitud y de respuesta para el presente código de función se muestran a continuación:

2.2.5.1 Solicitud

La trama de la solicitud tiene la estructura que se observa en la figura 2.2.11.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	VALOR A ESCRIBIR EN EL COIL	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.11. Estructura de la trama de solicitud para FC=05.

Por ejemplo, suponer que el dispositivo maestro, requiere establecer en valor del coil 173 a “1L” del dispositivo esclavo, de 5 dígitos, cuyo ID es 17. Quedando la trama de la solicitud de la siguiente manera:

Tabla 2.2.5.1. Trama de la solicitud del maestro para FC=05.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	05
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	CA
Dato a escribir (Hi)	1 byte	FF
Dato a escribir (Lo)	1 byte	00
CRC (Lo)	1 byte	7E
CRC (Hi)	1 byte	8B
TOTAL	8	-

En la tabla 2.2.5.1. se observa una estructura idéntica a las anteriores, la diferencia radica en el dato a escribir, para escribir en un coil un “1L” (ON) el valor del dato en hexadecimal debe ser 0xFF00 y para escribir un “0L” (OFF) el valor del dato en hexadecimal deber ser 0x0000, éstos son los únicos valores posibles que permiten escribir un coil, cualquier otro valor provocará un mensaje de excepción y no realizará ninguna acción sobre el coil.

2.2.5.2 Respuesta

La trama de la respuesta para este código de función es la misma a la enviada en la solicitud por parte del maestro, la cual devuelve después de haber escrito el coil, quedando como se observa en la figura 2.2.12.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	VALOR ESCRITO EN EL COIL	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.12. Estructura de la trama de respuesta para FC=05.

Tabla 2.2.5.2. Trama de la respuesta del esclavo para FC=05.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	05
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	CA
Dato a escribir (Hi)	1 byte	FF
Dato a escribir (Lo)	1 byte	00
CRC (Lo)	1 byte	7E
CRC (Hi)	1 byte	8B
TOTAL	8	-

En la tabla 2.2.5.2. se puede observar que se obtiene la misma trama enviada como respuesta, lo cual hace que la verificación de la trama sea fácil, que en el caso lectura.

En la figura 2.2.13 se observa el diagrama de flujo utilizado la generación de la trama de solicitud y lectura de la respuesta para la escritura de un coil.

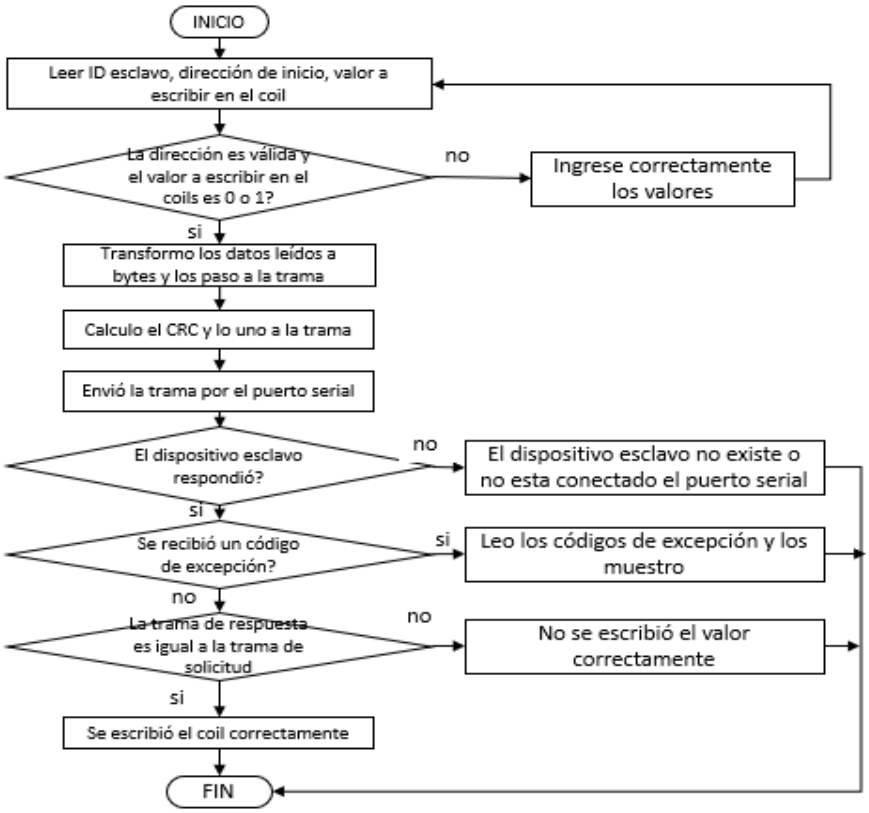

Figura 2.2.13. Algoritmo para generación de la trama de solicitud y lectura de la trama de respuesta para la escritura de un coil (FC=05).

2.2.6 ESCRIBIR SINGLE REGISTER (FC=06).

El siguiente código de función permite escribir un registro, en este caso un entero en un registro holding, donde los datos que ha proveer para la trama son los siguientes:

- Dirección del dispositivo esclavo.
- Dirección del registro de donde se realizará la escritura.
- Dato a escribir en el registro holding.

La trama de solicitud y de respuesta para el presente código de función se muestran a continuación:

2.2.6.1 Solicitud

La trama de la solicitud para la lectura de un registro holding se observa en la figura 2.2.14.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	VALOR A ESCRIBIR EN EL REGITRO HOLDING	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.14. Estructura de la trama de solicitud para FC=05.

A modo de ejemplo, suponer que el dispositivo maestro, requiere establecer el valor del registro holding 40002 a “3” del dispositivo esclavo, de 5 dígitos, cuyo ID es 17. Quedando la trama de la solicitud de la siguiente manera:

Tabla 2.2.6.1. Trama de la solicitud del maestro para FC=05.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	06
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	01
Dato a escribir (Hi)	1 byte	00
Dato a escribir (Lo)	1 byte	03
CRC (Lo)	1 byte	9A
CRC (Hi)	1 byte	9B
TOTAL	8	-

En la tabla 2.2.6.1. se observa una estructura idéntica a las anteriores, la diferencia radica en el dato a escribir, en este caso, en un holding register se puede escribir valores en hexadecimal de 0x0000 a 0xFFFF, el valor dependerá netamente del tipo de entero, si es con signo o sin signo.

2.2.6.2 Respuesta

La respuesta del dispositivo esclavo será la misma trama enviada por el maestro, luego de haber escrito el valor del registro holding como se observa en la figura 2.2.15..

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	VALOR ESCRITO EN EL REGITRO HOLDING	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.15. Estructura de la trama de respuesta para FC=06.

Tabla 2.2.6.2. Trama de respuesta del esclavo para FC=06.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	06
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	01
Dato a escribir (Hi)	1 byte	00
Dato a escribir (Lo)	1 byte	03
CRC (Lo)	1 byte	9A
CRC (Hi)	1 byte	9B
TOTAL	8	-

Como se ve en la tabla 2.2.6.2, se recibió la misma trama enviada por el maestro, lo cual facilita verificar si la escritura fue realizada correctamente, si no se obtiene la misma trama enviada se considera que existió alguna falla en el envío de la solicitud.

En la figura 2.2.16 se observa el diagrama de flujo de la solicitud y respuesta para la escritura de un registro holding.

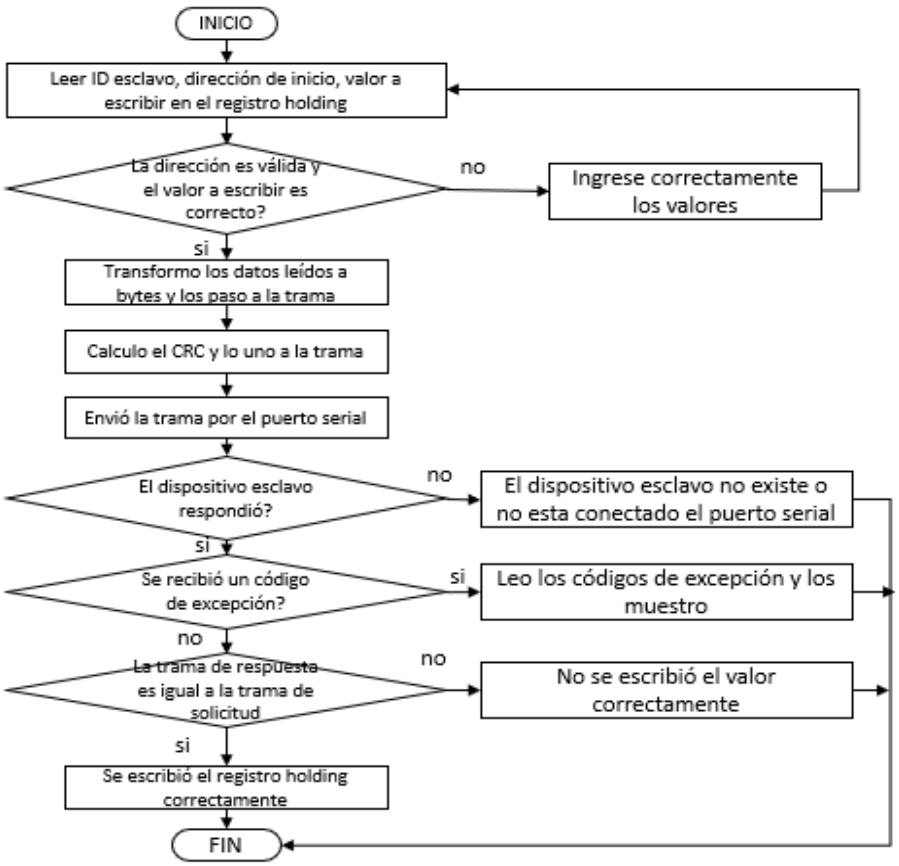

Figura 2.2.16. Algoritmo para generación de la trama de solicitud y lectura de la trama de respuesta para la escritura de un registro holding (FC=06).

2.2.7 ESCRIBIR MULTIPLES COILS (FC=15).

El código de función FC=15, permite escribir (ON/OFF) en múltiples coils, de acuerdo a lo solicitado por el maestro, para lo cual es necesario considerar los siguientes parámetros a ingresar en la trama de datos:

- ID del dispositivo.
- Dirección de inicio de la escritura de coils.
- Cantidad de bytes a escribir.
- Valores de los coils a escribir.

Es importante considerar que para escribir múltiples coils, cada bit representará un coil, y el bit menos significativo será el de la dirección de inicio de la escritura, para escribir (ON)

en un coil, el bit deberá tener el valor de “1”, para escribir (OFF) el bit deberá tener el valor de “0”, tener en cuenta que se puede escribir 8 coils por cada byte, y si se tiene bits sobrantes esos bits deberán ser llenados con el valor de “0”.

A continuación, se detallará la estructura de la trama de escritura de múltiples coils por parte del maestro y la trama de respuesta del esclavo.

2.2.7.1 Solicitud.

En el caso de escritura de múltiples coils, la estructura de la trama de solicitud del maestro tendrá una variación en comparación con los anteriores códigos de función estudiados, quedando como se observa en la figura 2.2.17:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	CANTIDAD DE COILS A ESCRIBIR	CONTADOR DE BYTES	VALORES DE LOS COILS A ESCRIBIR	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	1 byte	N bytes	2 bytes	3.5 char

Figura 2.2.17. Estructura de la trama para FC=15.

A modo de ejemplo, se desea escribir 10 coils, iniciando desde la dirección 20, en el dispositivo esclavo cuya ID es 17. El dato a escribir en binario es el siguiente: en el primer byte “11001101” y en el segundo byte”00000001”.

Prestar atención que sólo 10 bits tienen valor, el resto de los bits se los llena de “0”, y considerar que el bit menos significativo es el que se escribirá en la dirección de inicio, el resto de bits son las direcciones siguientes, tener en cuenta que siempre se inicia desde el bit menos significativo.

La trama quedará de la siguiente manera:

Tabla 2.2.7.1. Trama de solicitud del maestro para FC=15.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	0F
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	13
Cantidad de coils (Hi)	1 byte	00
Cantidad de coils (Lo)	1 byte	0A
Contador de bytes	1 byte	02
Dato a escribir (Hi)	1 byte	CD
Dato a escribir (Lo)	1 byte	01
CRC (Lo)	1 byte	BF
CRC (Hi)	1 byte	0B
TOTAL	11	-

Como se observa en la tabla 2.2.7.1, en el caso de escritura de coils se incrementa el número de bytes en la trama, la cantidad de bytes dependerá de la cantidad de coils a escribir, para la escritura se poseen más campos que se deben llenar como lo es la cantidad de coils, contador de bytes y la cantidad de datos a escribir.

2.2.7.2 Respuesta.

La trama de la respuesta retorna la dirección del esclavo, el código de función, la dirección de inicio, la cantidad de coils escritos y el CRC, quedando la trama como se observa en la figura 2.2.18.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	CANTIDAD DE COILS A ESCRITOS	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.17. Estructura de la trama de respuesta para FC=15.

La respuesta de la solicitud realizada será:

Tabla 2.2.7.2. Trama de la respuesta del esclavo para FC=15.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	11
Código de función	1 byte	0F
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	13
Coils escritas (Hi)	1 byte	00
Coils escritas (Lo)	1 byte	0A
CRC (Lo)	1 byte	26
CRC (Hi)	1 byte	99
TOTAL	8	-

Como se observa en la tabla 2.2.7.2, el esclavo responde con la misma ID de dispositivo y el mismo código de función que fueron enviados por el maestro, a continuación, se tiene la dirección de inicio de la escritura y la cantidad de coils escritos, por último, se tiene los 2 bytes del chequeo de error o CRC.

En la figura 2.2.19 se observa el diagrama de flujo utilizado para la realización de la trama de solicitud y respuesta para la escritura de múltiples coils.

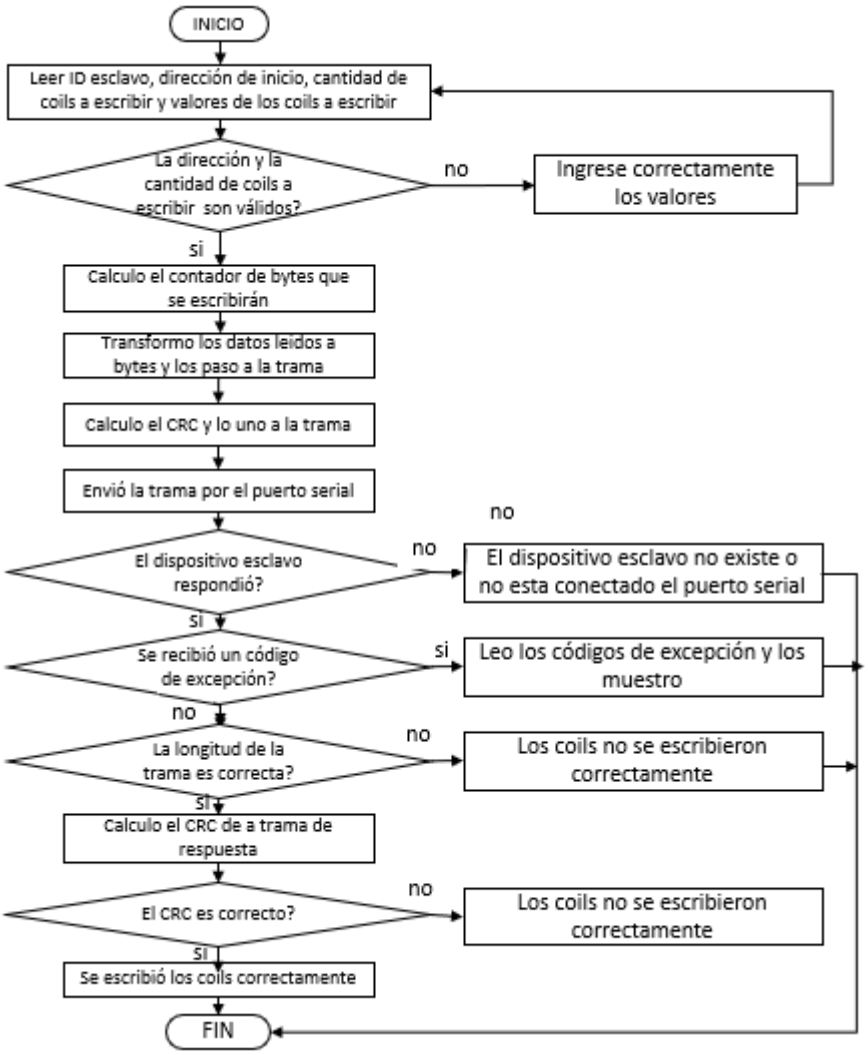

Figura 2.2.19. Algoritmo para generación de la trama de solicitud y lectura de la trama de respuesta para la escritura de múltiples coils (FC=15).

2.2.8 ESCRIBIR MÚLTIPLES REGISTERS (FC=16).

Esta función permite escribir múltiples registros holding, de estructura similar a la usada en la FC=15, los parámetros necesarios a ingresar para este código de función son los siguientes:

- ID del dispositivo
- Dirección de inicio de escritura
- Cantidad de registros a escribir
- Valor de los datos a escribir

En este punto es necesario considerar que los datos que ha escribir pueden ser de tipo entero y flotante, en el caso de datos de tipo entero, por cada dato de tipo entero se escribirá un registro, en el caso de tener dato de tipo flotante se ocuparán dos registros por

cada dato flotante, además de tener la consideración del tipo de endian con el que trabaja el dispositivo esclavo.

A continuación, se detallará las tramas de solicitud de parte del maestro y la trama de respuesta del esclavo:

2.2.8.1 Solicitud

En el caso de escritura de múltiples registros, la estructura de la trama de solicitud del maestro es similar a la usada para escribir múltiples coils, quedando como se observa en la figura 2.2.20.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	CANTIDAD DE REGISTROS HOLDIG A ESCRIBIR	CONTADOR DE BYTES	VALORES DE LOS REGISTROS HOLDING A ESCRIBIR	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	1 byte	N bytes	2 bytes	3.5 char

Figura 2.2.20. Estructura de la trama de solicitud para FC=16.

A modo de ejemplo, se desea escribir 2 datos flotantes, iniciando desde la dirección 40001, en el dispositivo esclavo cuya ID es 3. Los datos a escribir son “12.5” y “45.75”.

Como es un dato de tipo flotante, se asume que el dispositivo trabaja con little endian, quedando la trama de la siguiente manera:

Tabla 2.2.8.1. Trama de solicitud del maestro para FC=16.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	03
Código de función	1 byte	10
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	00
Cantidad de registros (Hi)	1 byte	00
Cantidad de registros (Lo)	1 byte	04
Contador de bytes	1 byte	08
Float 1(LSB) (Hi)	1 byte	00
Float 1 (LSB)(Lo)	1 byte	00
Float 1(MSB) (Hi)	1 byte	41
Float 1 (MSB)(Lo)	1 byte	48
Float 2(LSB) (Hi)	1 byte	00
Float 2 (LSB)(Lo)	1 byte	00
Float 2(MSB) (Hi)	1 byte	42
Float 2 (MSB)(Lo)	1 byte	37
CRC (Lo)	1 byte	AA
CRC (Hi)	1 byte	12
TOTAL	17 bytes	-

Se puede observar en la tabla 2.2.8.1 que en el caso de escritura de registros se incrementa el número de bytes en la trama, la cantidad de bytes dependerá de la cantidad de datos a escribir y el tipo de dato (entero o flotante), aquí también se observa la manera en la que se envía el dato de tipo flotante, al ser little endian, primero se envía el registro menos significativo (LSB) y luego el registro más significativo (MSB).

2.2.8.2 Respuesta

La trama de la respuesta retorna la dirección del esclavo, el código de función, la dirección de inicio, la cantidad de registros escritos y el CRC, quedando la trama como se observa en la figura 2.2.21.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION	DIRECCION DE INICIO	CANTIDAD DE REGISTROS HOLDIG ESCRITOS	CRC	Fin
3.5 Char	1 byte	1 byte	2 bytes	2 bytes	2 bytes	3.5 char

Figura 2.2.21. Estructura de la trama de respuesta para FC=16.

La respuesta de la solicitud realizada será:

Tabla 2.2.8.2. Trama de la respuesta del esclavo para FC=16.

Campo	Bytes	Valor (HEX)
ID del dispositivo	1 byte	03
Código de función	1 byte	10
Dirección de inicio (Hi)	1 byte	00
Dirección de inicio (Lo)	1 byte	00
Registros escritos (Hi)	1 byte	00
Registros escritos (Lo)	1 byte	04
CRC (Lo)	1 byte	C0
CRC (Hi)	1 byte	28
TOTAL	8	-

En la tabla 2.2.8.2 se puede notar que el esclavo responde con la misma ID de dispositivo y el mismo código de función que fueron enviados por el maestro, a continuación, se tiene la dirección de inicio de la escritura y la cantidad de coils escritos, por último, se tiene los 2 bytes del chequeo de error o CRC.

En la figura 2.2.22 se observa el diagrama de flujo utilizado para la realización de la trama de solicitud y respuesta para la escritura de múltiples registros.

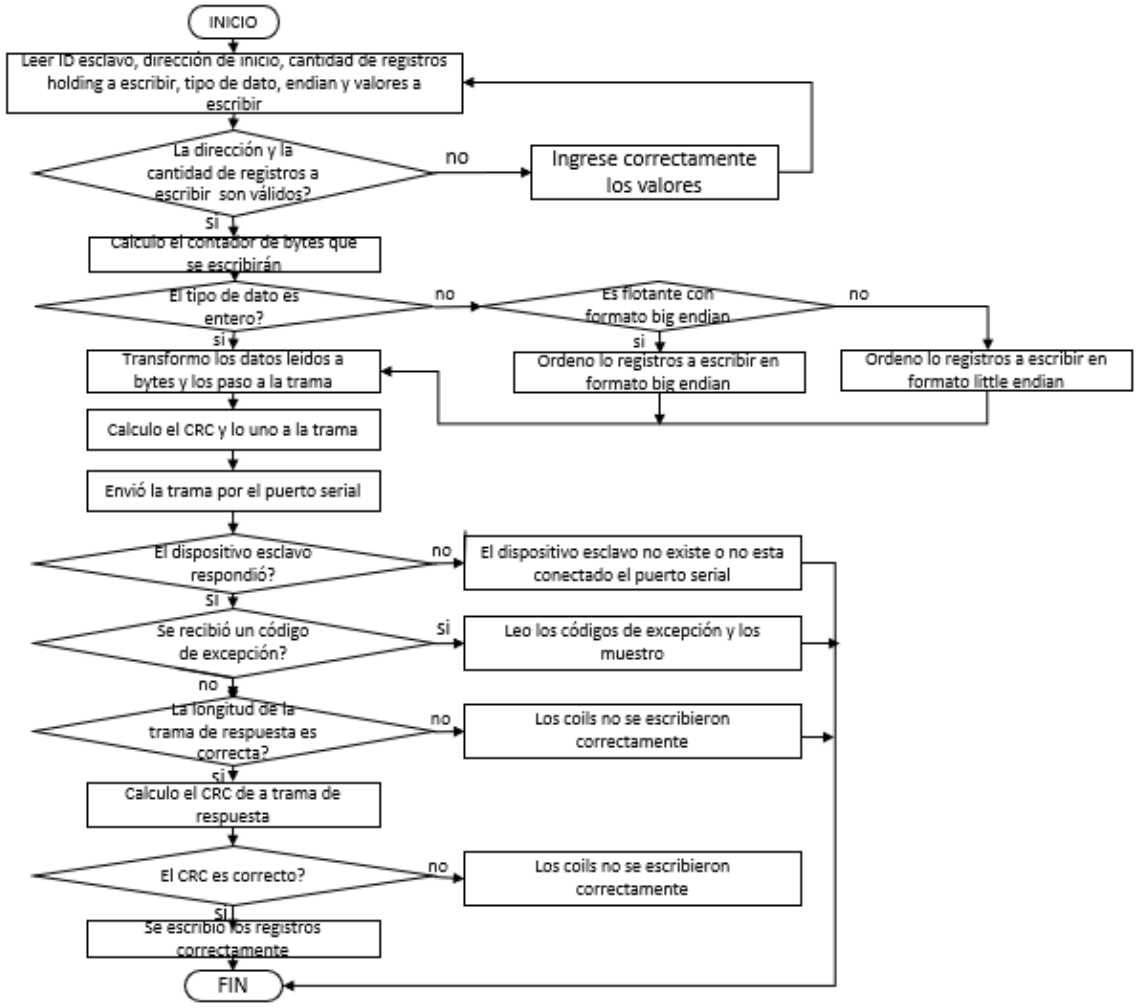

Figura 2.2.22. Algoritmo para generación de la trama de solicitud y lectura de la trama de respuesta para la escritura de múltiples registros holding (FC=16).

2.2.9 CÓDIGOS DE EXCEPCIÓN EN LA NORMATIVA MODBUS RTU

Los códigos de excepción dentro de protocolo MODBUS RTU se pueden dar por errores en el armado de la trama de datos por parte del dispositivo maestro o por error de procesamiento de los datos en el dispositivo esclavo, o ya sea por distorsión de la trama en la transmisión del mismo.

La estructura de la trama de un mensaje de excepción será de la siguiente manera como se observa en la figura 2.2.23.:

INICIO	ID DISPOSITIVO	CODIGO DE FUNCION DE EXCEPCION	CODIGOS DE EXCEPCION	CRC	Fin
3.5 Char	1 byte	1 byte	N bytes	2 bytes	3.5 char

Figura 2.2.23. Estructura de la trama del código de excepción.

La estructura de la trama es muy similar a las anteriores, pero se destacan las principales diferencias que son:

- El código de función de excepción es igual al código de función enviado más 128, es decir, el bit más significativo del código de función se pone en “1”.
- Los códigos de excepción dependerán del error de comunicación que se haya generado entre el dispositivo maestro y esclavo [8].

2.3 MANEJO DEL PUERTO SERIAL EN JAVA.

Para implementar la comunicación serial en el software de JAVA, como primer paso es necesario instalar el archivo de Java “comm.jar”, el cual proveerá de las clases necesarias para poder realizar la conexión del puerto serial.

Para realizar la comunicación serial es necesario considerar los parámetros a configurar, los cuales son:

- Puerto Serial (COM)
- Velocidad de transmisión de datos
- Los bits de datos
- Los bits de parada
- Paridad

Se inicia con la búsqueda de los puertos disponibles en la computadora, mediante la clase *CommPortIdentifier*, a continuación, se elige el puerto en el que está conectado el dispositivo esclavo. Para la apertura del puerto y la configuración de los parámetros del puerto serial se utilizará la clase *SerialPort* [12] , se debe destacar que la configuración del puerto serial en la computadora debe tener la misma configuración que el dispositivo esclavo, caso contrario se presentarán errores en el momento del envío y recepción de datos, por último para el envío de la trama de solicitud se usan los métodos de la clase *OutputStream*, y para la recepción de la trama de solicitud se usan los métodos de la clase *InputStream*, en estas clases obligatoriamente se utiliza el constructor TRY y CATCH para atrapar errores que puedan darse al momento de realizar la conexión o en el envío y recepción de datos.

En la tabla 2.3.1, se presentan los métodos usados para la conexión, desconexión, envío y recepción de datos del puerto serial.

Tabla 2.3.1. Métodos usados para la conexión del puerto Serial [11].

Clase	Método	Descripción
CommPortIdentifier	getPortIdentifiers ()	Identifica cuantos puertos seriales se tiene disponibles
	nextElement ()	Recorre cada puerto serial disponible
SerialPort	open (String nombre, int tiempo)	Abre el Puerto serial seleccionado
	setSerialPortParams (int baudrate, int databits, int stopbits, int paridad)	Configura los parámetros del puerto serial.
	close ()	Cierra el puerto serial.
	setDTR (boolean dtr)	Habilita o limpia la terminal de datos lista (DTR).
OuputStream	getOutputStream ()	Flujo de salida de JAVA al puerto serial.
	write (byte [] b)	Escribe el dato a enviar al flujo de salida.
InputStream	getInputStream ()	Flujo de entrada de datos del puerto serial a JAVA.
	read (byte [] b)	Lee el dato enviado por el puerto serial y lo almacena en b.

El algoritmo utilizado para la realización de la conexión del puerto serial se muestra en la figura 2.3.1.

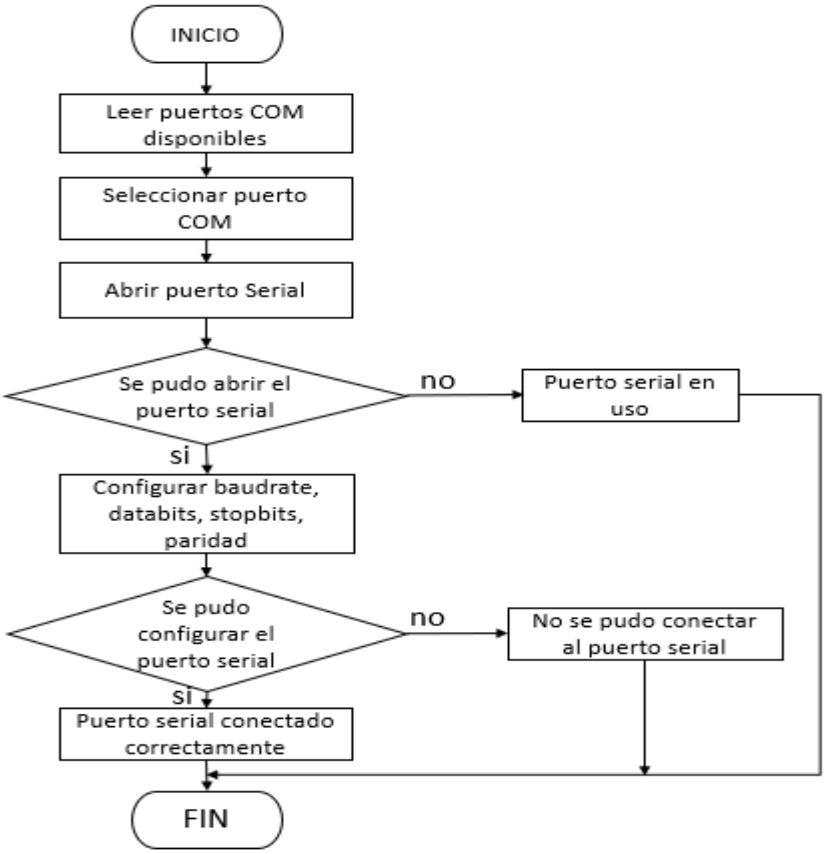

Figura 2.3.1. Diagrama de flujo de la conexión del puerto serial.

2.4 MANEJO DE LA BASE DE DATOS EN MYSQL DESDE JAVA.

Para el manejo de la base de datos, se ha seleccionado MySQL, dado que este tipo de gestor de base de datos permite la interacción con diferentes lenguajes de programación, ya que utiliza un lenguaje de consulta estructurado (SQL), en este caso al gestionar la base de datos de MySQL desde JAVA es necesario instalar el conector JDBC (Java Data Base Connectivity).

Este conector permite la realización de diferentes tareas como:

- Conectar la aplicación con una base de datos.
- Crear, eliminar, modificar y realizar consultas de tablas de datos que se encuentren en la base de datos.

Existen varias versiones de este conector, para el presente proyecto se ha elegido la versión 5.1.47, el cual se adapta a las características de la versión de JAVA utilizada en el proyecto, el JDK usado es la versión 14 y el JRE es la versión 8 [13].

La compatibilidad se puede observar en la gráfica 2.4.1.

Versión del Conector	Versión JBDC	Versión MyQSL	JRE	JDK
5.1	3.0, 4.0, 4.1 y 4.2	8.0	JRE 5 o superior	JDK 5.0, 8.0 o superior

Figura 2.4.1. Compatibilidad del driver JDBC.

2.4.1 PARÁMETROS PARA ESTABLECER LA CONEXIÓN CON LA BASE DE DATOS.

Para establecer la conexión con la base de datos es necesario considerar los siguientes parámetros:

- Dirección del controlador JBDC: en java la dirección siempre será *com.mysql.jdbc.Driver.*
- URL: es el enlace que permite la conexión de la base de datos en MySQL desde JAVA, el cual tiene la siguiente estructura:

  jdbc: mysql: // "nombre de host": "puerto" /" base de datos".
- Host: lugar donde está instalado el servidor de MySQL, por defecto es localhost, esto en el caso que se ejecute en la misma máquina donde corre JAVA, el host también puede ser una dirección IP de la máquina donde encuentre MySQL.
- Puerto: es el puerto TCP/IP donde el servidor recibe los requerimientos, por defecto el puerto es 3306, el cual puede cambiar de acuerdo a la configuración.
- Base de datos: base a la cual se desea acceder [14].
- Usuario: usuario que tiene acceso al servidor de MySQL, el usuario por defecto es root, pero es recomendable utilizar un usuario creado, ya que root tiene muchas limitaciones.
- Clave: contraseña del usuario que permite el acceso al servidor de MySQL [15].
- Los métodos usados para la conexión se muestran en la tabla 2.4.1.

Tabla 2.4.1. Comandos usados para la conexión [16].

Método	Descripción
Class.forName(String controlador)	Realiza la búsqueda de la clase solicitada y carga la clase.
DriverManager.getConnection(String URL, String usuario, String clave);	Establece la conexión con la base de datos, donde DriverManager administra los drivers JBDC.

2.4.2 ENVÍO DE COMANDOS A MYSQL DESDE JAVA

Teniendo la conexión con la base de datos, el siguiente paso es enviar los comandos desde java a MySQL para realizar distintas acciones como lectura, escritura actualización o

eliminación de datos en la base de datos [17], para lo cual se debe seguir el procedimiento detallado a continuación:

- Crear el método para ejecución de consultas a la base de datos mediante *createStatement ()*;
- Ejecutar la consulta a la base de datos.

A continuación, un ejemplo de la realización de consultas a la base de datos:

```
try {
String Query = "UPDATE " + table_name + " SET VALOR ='"+Valor+"' WHERE NOMBRE ='"+nombre_var+"'";

Statement st = Conexion.createStatement();

st.executeUpdate(Query);

} catch (SQLException ex) {
}
```

Como se puede ver en el ejemplo anterior, primero se crea el statement y luego se ejecuta el comando, todo esto dentro de un bloque try catch para capturar alguna falla en la solicitud.

2.4.3 INSERTAR Y ACTUALIZAR DATOS EN MYSQL DESDE JAVA.

Para ejecutar comandos que permitan modificar la base de datos, se utiliza el método *executeUpdate ()*, los comandos utilizados en el presente trabajo se muestran en la siguiente tabla:

Tabla 2.4.2. Comandos para insertar y actualizar datos en MySQL [18].

Sentencias	Definición
CREATE DATABASE	Permite la creación de una nueva base de datos Ej: *CREATE DATABASE nombreDB*
CREATE TABLE	Crea una nueva tabla en la base de datos conectada. Ej: *CREATE TABLE nombre_tabla (ID Columna1 NOT NULL AUTO_INCREMENT, Columna2 VARCHAR (45))*
ALTER TABLE	Permite modificar la estructura de la tabla, en este caso permite agregar o quitar columnas de una tabla específica. Ej: *ALTER TABLE nombre_tabla ADD COLUMN nombre_columna VARCHAR (25)"*
DROP TABLE	Elimina la tabla. Ej: *DROP TABLE table_name*
INSERT	Permite el ingreso de nuevos registros (filas) en una tabla específica. Ej: *INSERT INTO table_name VALUES ("dato1"," dato2"," dato3")*
UPDATE	Actualiza valores de la columna en un registro específico.

	Ej: *UPDATE table_name SET VALOR =Valor WHERE NOMBRE =nombre_var*

2.4.4 CONSULTAR O LEER DATOS EN MYSQL DESDE JAVA

Para recibir valores desde una tabla de MySQL, el método a utilizar es el *executeQuery ()*, el cual tendrá como retorno una variable de tipo Resultset el cual almacena el valor de la solicitud realizada, ya sea ésta el valor de toda una tabla o el de una fila en específico, para lo cual su estructura quedará de la siguiente manera:

```
String Query = "SELECT * FROM table_name WHERE NOMBRE=Start";

Statement st = Conexion.createStatement();

java.sql. ResultSet resultSet;

resultSet = st. executeQuery (Query);
if (resultSet.first()) {

Valor= resultSet.getString("VALOR");

}
```

Como se puede observar, se utiliza la sentencia *SELECT* para solicitar el valor de la fila donde la variable se llame start de la table_name. El valor es almacenado en *resultSet*, en donde se usa los siguientes métodos para extraer el dato o datos requeridos [19].

Tabla 2.4.3. Métodos usados en resultSet.

Método	Descripción
next ()	Recorre a la siguiente fila y retorna true si lo consigue, retorna false cuando ya no existen más filas.
Last ()	Retorna un true cuando se alcanza la última fila, caso contrario retorna un false.
First ()	Retorna un true cuando se da con la primera fila del resultSet, caso contrario retorna un false.
getString (String column_name)	Obtiene el valor de la columna solicitada.

2.5 DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO (GUI)

Dentro de los requerimientos establecidos al inicio del capítulo, se estableció el diseño de 3 ventanas, la primera ventana será de presentación y de selección del modo de funcionamiento del programa, el primer modo de funcionamiento es el modo maestro MODBUS RTU, el cual permite realizar la comunicación con el dispositivo esclavo mediante los ocho códigos de función principales que se maneja en el protocolo MODBUS RTU, en este modo sólo se puede realizar solicitudes al dispositivo esclavo de uno en uno. El

segundo modo de funcionamiento es el modo servidor de datos MODBUS RTU el cual también realiza la comunicación con el dispositivo esclavo, pero además almacena la información obtenida en una base de datos, en este modo de funcionamiento se podrá crear una tabla de datos que permita leer múltiples registros, ya sean estos booleanos, enteros o flotantes, de manera continua. La base de datos generada permitirá la interconexión con el software Intouch, que sirve para monitorear las variables de manera gráfica.

2.5.1 VENTANA DE PRESENTACIÓN

La ventana de presentación se muestra en la figura 2.5.1.1, en donde se puede observar el encabezado con la información de la universidad, facultad y departamento, en la parte superior, en la parte media se encuentra el nombre del autor y los directores del proyecto, mientras que en la parte inferior se dispone dos botones que permiten la selección del modo de funcionamiento, que como se explicó anteriormente son 2, el modo maestro y el modo servidor.

Figura 2.5.1. Ventana de presentación.

2.5.2 VENTANA MODO MAESTRO MODBUS RTU.

Esta ventana permite establecer comunicación entre un dispositivo esclavo MODBUS RTU y el programa desarrollado en Java, usando este mismo protocolo mediante ocho códigos de función que sirven para la lectura y escritura de variables booleanas, enteras y flotantes. Su diseño se puede apreciar en la figura 2.5.2.

Figura 2.5.2. Ventana Maestro Modbus RTU.

Para detallar mejor las partes que componen esta ventana, se dividirán en 3 secciones las cuales son:

2.5.2.1 Conexión del puerto serial.

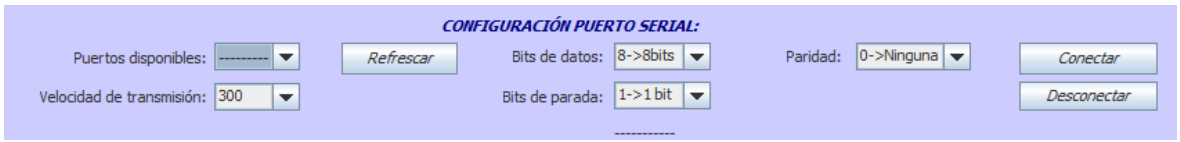

Figura 2.5.3. Parámetros para la configuración del puerto serial.

En la figura 2.5.3 se observan los parámetros necesarios a configurar para realizar la conexión y desconexión del puerto serial, éstos se detallan a continuación:

- **Puertos disponibles:** permite elegir entre los puertos seriales disponibles en el computador.
- **Refrescar:** botón que actualiza el listado de puertos disponibles.
- **Velocidad de transmisión:** permite elegir la velocidad de transmisión de datos.
- **Bits de datos:** permite elegir la longitud de los bits de datos, éste puede ser 7 u 8, para Modbus RTU lo más común es usar 8 bits.
- **Bits de parada:** permite elegir los bits de parada, éstos pueden ser 1 o 2 bits de parada.
- **Paridad:** permite elegir la paridad, ésta puede ser par, impar o ninguna.

- **Conexión:** botón que realiza la conexión del puerto serial con los parámetros establecidos en los anteriores ítems.
- **Desconexión:** desconecta el puerto serial.

2.5.2.2 Parámetros de configuración.

Figura 2.5.4. Parámetros a configurar para la generación de la trama de solicitud.

Como se puede observar en la figura 2.5.4, esta sección permite configurar los parámetros para generar la trama de solicitud al dispositivo esclavo, a continuación, se detalla cada una de sus partes:

- **ID esclavo:** aquí se ingresa la identificación del dispositivo, éste puede ir de 1 a 255, según el dispositivo.
- **Código de función:** permite elegir entre los ocho códigos de función, los códigos 01.02.03 y 04, son códigos de función para lectura de coils, contacts, holding e input register, mientras que los códigos 05.06,15 y 16 son códigos de función para la escritura de coils y holding register.
- **Tipo de dato:** permite seleccionar el tipo de dato, éste puede ser booleano, entero con signo o Word, entero sin signo o uint y flotante.
- **Dirección Base:** permite elegir la base del dispositivo, es decir si el registro inicial inicia en 0 o 1, si el dispositivo trabaja con dirección de protocolo su registro inicial es 0, si trabaja con dirección de PLC su registro inicial es 1.
- **Endianness:** permite elegir la manera en cómo se envían los datos flotantes, si es big endian la parte más significativa del dato se envía primero y luego la parte menos significativa, si es little endian la parte menos significativa del dato se envía primero y luego la parte más significativa.

- **Dígitos:** permite seleccionar el tamaño del espacio de memoria de los registros, si es 4 la cantidad máxima de registros será 999, si es 5 la cantidad máxima de registros 9999 y si es 6 la cantidad máxima de registros será de 65535, esto depende exclusivamente de las capacidades del dispositivo esclavo. Los dígitos incluyen internamente el subíndice de memoria 0,1,3 y 4 de acuerdo con el código de función seleccionado.
- **Dirección de inicio:** aquí se ingresa la dirección desde la cual se desea iniciar la lectura o escritura de registros. Hay que considerar que se debe ingresar la dirección sin el subíndice de memoria ya que ésta se genera internamente al seleccionar el código de función. Por ejemplo, si nuestro registro en el PLC es el 3001, en este campo ingresaremos sólo 001.
- **Cantidad a leer:** permite ingresar la cantidad de datos a leer, esto en caso de elegir los códigos de función de lectura y del tipo de dato a leer.
- **Valor:** permite ingresar el valor a escribir en un registro, esto en el caso de escribir un solo dato, es decir, cuando sean seleccionados los códigos de función 05 y 06, éste depende también del tipo de dato, puede ser booleano, entero o flotante.
- **Cantidad a escribir:** aquí se ingresa la cantidad de datos a escribir en el caso de seleccionar los códigos de función 15 y 16.
- **Ingreso de datos:** permite el ingreso de datos, para el caso que se requiera escritura múltiple. Cabe indicar que los datos deben ser ingresados uno a uno en el caso de enteros y flotantes, en el caso de booleanos se debe ingresar byte por byte.
- **Número de datos:** permite visualizar la cantidad de datos ingresados, en el caso que se requiera escritura múltiple.
- **Datos ingresados:** muestra los datos ingresados, esto en caso de haber elegido escribir múltiples datos.
- **Agregar dato:** botón que agrega un nuevo dato, funciona únicamente cuando se realiza escritura de múltiples datos.
- **Borrar datos:** botón que borra todos los datos ingresados, funciona únicamente cuando se realiza escritura múltiple.
- **Enviar:** botón que envía la trama de solicitud al dispositivo esclavo.

2.5.2.3 Visualización de la respuesta del dispositivo esclavo

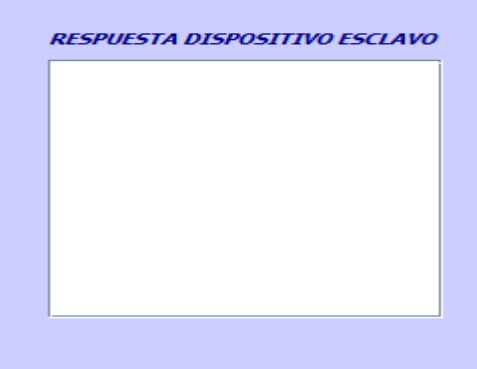

Figura 2.5.5. Visualización de la respuesta del dispositivo esclavo.

En la figura 2.5.5. se puede observar el área en donde se mostrarán los valores de los registros leídos del dispositivo esclavo, en caso de escritura se mostrará el mensaje escritura correcta cuando se escriba en un registro correctamente.

2.5.3 VENTANA MODO SERVIDOR DE DATOS

La ventana de modo servidor de datos, realiza una doble comunicación, es decir, se comunica con el dispositivo esclavo, y a su vez también se comunica con una base de datos generada en MySQL, con la ventaja que la base de datos puede interactuar también con sistemas de control supervisorios y HMIs, por ejemplo, el desarrollo de un HMI en el software Intouch que se comunique con la base de datos generada. En la figura 2.5.6. se puede observar un esquema de la interconexión del programa creado con la base de datos y el dispositivo esclavo.

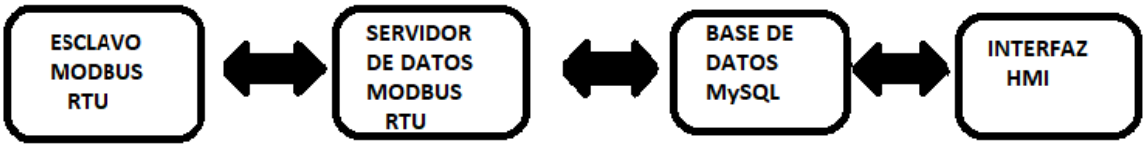

Figura 2.5.6. Modo servidor de datos MODBUS RTU.

Para este modo de funcionamiento, se usan 7 códigos de función, por lo cual se descarta el uso del código de función 15, el cual se encargaba de escribir múltiples registros coils, se descartó este código de función ya que el servidor de datos trabajará con datos individuales, es decir, cada variable tendrá un sólo tipo de dato, como se muestra en la tabla 2.5.1.

Tabla 2.5.1 Códigos de función usados en el modo servidor de datos

Código de función	Acción
01	Leer una variable booleana de los registros coils
02	Leer una variable booleana de los registros contacts
03	Leer una variable entera o flotante de los registros holding

04	Leer una variable entera o flotante de los registros de entrada
05	Escribir una variable booleana en los registros coil
06	Escribir una variable entera en los registros holding
16	Escribir una variable flotante en los registros holding

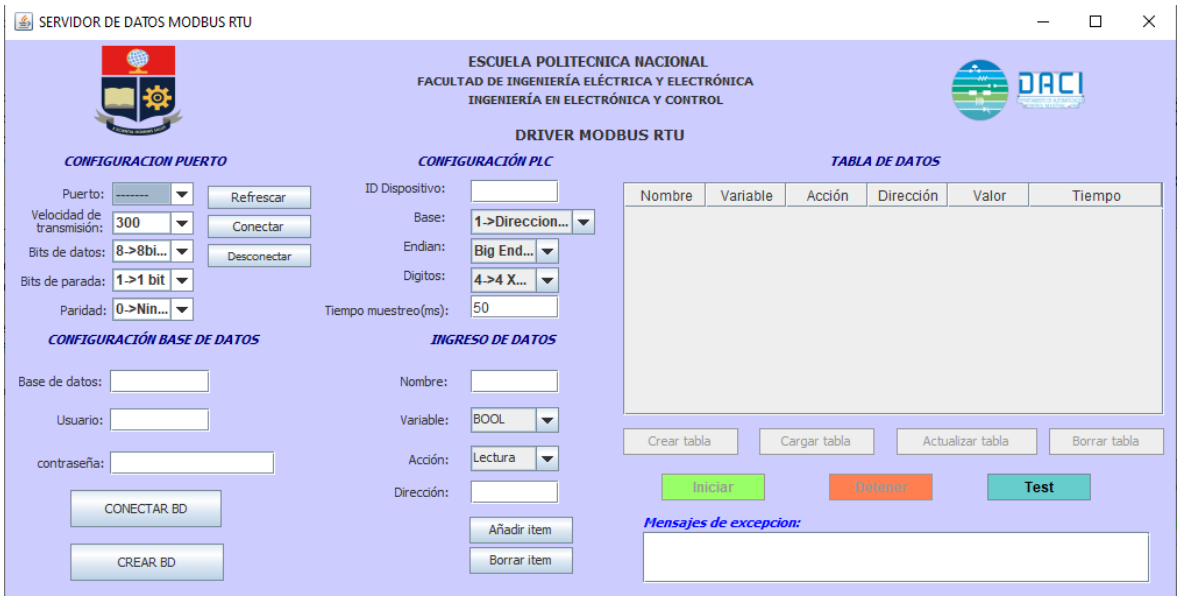

Figura 2.5.7. Visualización de la respuesta del dispositivo esclavo.

En la figura 2.5.7. se puede observar toda la ventana del modo servidor de datos, pero para poder explicar cada parte de esta ventana, se la dividirá en 6 partes que se detallan a continuación.

2.5.3.1 CONFIGURACIÓN DEL PUERTO SERIAL.

Figura 2.5.8. Configuración del puerto serial.

En la figura 2.5.8 se observan los parámetros necesarios a configurar para realizar la conexión y desconexión del puerto serial, éstos se detallan a continuación:

- **Puertos disponibles:** permite elegir entre los puertos seriales disponibles en el computador.
- **Refrescar:** botón que actualiza el listado de puertos disponibles.
- **Velocidad de transmisión:** permite elegir la velocidad de transmisión de datos.

- **Bits de datos:** permite elegir la longitud de los bits de datos, éste puede ser 7 u 8, para Modbus RTU lo más común es usar 8 bits.
- **Bits de parada:** permite elegir los bits de parada, éstos pueden ser 1 o 2 bits de parada.
- **Paridad:** permite elegir la paridad, ésta puede ser par, impar o ninguna.
- **Conexión:** botón que realiza la conexión del puerto serial con los parámetros establecidos en los anteriores ítems.
- **Desconexión:** desconecta el puerto serial.

2.5.3.2 CONFIGURACIÓN DE LA BASE DE DATOS.

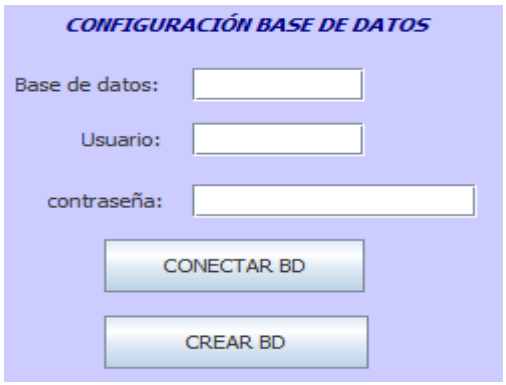

Figura 2.5.9. Configuración de la base de datos.

En la figura 2.5.9 se observan los parámetros necesarios para establecer comunicación con la base de datos, enseguida se detalla cada uno de estos parámetros:

- **Base de datos:** aquí se ingresa el nombre de la base de datos a la cual se quiere acceder o que se quiere crear.
- **Usuario:** aquí se ingresa el nombre del usuario que tiene acceso a la base de datos de MySQL.
- **Contraseña:** en este campo se ingresa la contraseña del usuario con el cual se quiere acceder a la base de datos de MySQL.
- **Conectar bd:** se conecta con la base de datos ingresada, usando el usuario y la clave ingresada, en caso de no existir la base de datos enviará un mensaje de error.
- **Crear bd:** crea una nueva base de datos con los parámetros ingresados, en caso de ya existir esa base de datos enviará un mensaje de error.

2.5.3.3 CONFIGURACIÓN DEL PLC.

Figura 2.5.10. Configuración del PLC.

En la figura 2.5.10, se observa las principales características que posee el dispositivo esclavo con el cual se desea comunicar, cada parámetro se detallará a continuación.

- **ID del dispositivo:** aquí se ingresa la identificación del dispositivo, éste puede ir de 1 a 255, según el dispositivo.
- **Base:** permite elegir la base del dispositivo, es decir si el registro inicial inicia en 0 o 1, si el dispositivo trabaja con dirección de protocolo su registro inicial es 0, si trabaja con dirección de PLC su registro inicial es 1
- **Endian:** permite elegir la manera en cómo se envían los datos flotantes, si es big endian la parte más significativa del dato se envía primero y luego la parte menos significativa, si es little endian la parte menos significativa del dato se envía primero y luego la parte más significativa.
- **Dígitos:** permite seleccionar el tamaño del espacio de memoria de los registros, si es 4 la cantidad máxima de registros será 999, si es 5 la cantidad máxima de registros 9999 y si es 6 la cantidad máxima de registros será de 65535, esto depende exclusivamente de las capacidades del dispositivo esclavo.
- **Tiempo de muestreo:** tiempo en el que se realiza la lectura de la trama de respuesta dada por el dispositivo esclavo.

2.5.3.4 INGRESO DE DATOS.

Figura 2.5.11. Ingreso de datos.

En la figura 2.5.11. se puede observar los parámetros a ingresar para crear una variable, a continuación, se explica cada campo.

- **Nombre:** nombre de la variable a ingresar.
- **Variable:** permite seleccionar el tipo de variable, ésta puede ser BOOL, INT, UINT o FLOAT.
- **Acción:** permite seleccionar la acción a realizar, ésta puede ser lectura o escritura de la variable.
- **Dirección:** dirección donde se encuentra una variable, para este caso se debe ingresar el subíndice de memoria seguido con la dirección del registro, por ejemplo, la variable a crear se encuentra en los registros contacts en la dirección 2, la dirección será 1002, en caso de que sea de 4 dígitos, donde se puede notar que el subíndice es 1 y la dirección es 2.
- **Añadir ítem:** añade una nueva variable a la tabla que se desea crear.
- **Borrar ítem:** borra la última variable ingresada en la tabla.

2.5.3.5 TABLA DE DATOS.

Figura 2.5.12. Tabla de datos.

En la figura 2.5.12 se puede observar la plantilla de la tabla de datos, en la cual se mostrará las variables ingresadas, en el caso de iniciar la comunicación se mostrarán los valores de las variables continuamente y el tiempo en el que se actualizó dicha variable.

En este campo existen varios botones y a continuación se detalla su funcionalidad:

- **Crear tabla:** permite crear tablas de datos en MySQL con los datos ingresados en el servidor.
- **Cargar tabla:** permite traer los valores de una tabla de datos de MySQL al servidor.
- **Actualizar tabla:** actualiza el valor de las tablas de la base de datos.
- **Borrar tabla:** permite borrar las tablas de la base de datos.
- **Iniciar:** inicia la comunicación del servidor de datos.
- **Detener:** detiene la comunicación del servidor de datos.
- **Test:** el botón test permite escribir el valor de una variable en la base de datos, se puede usar también para verificar si se está comunicando correctamente con la base de datos y si el valor se escribe correctamente en el dispositivo esclavo.

A continuación, en la figura 2.5.13 se muestra el diagrama de flujo para lectura y escritura continua del servidor con el PLC y en la figura 2.5.14 se muestra el diagrama de flujo lectura y escritura continua entre la base de datos y el servidor de datos

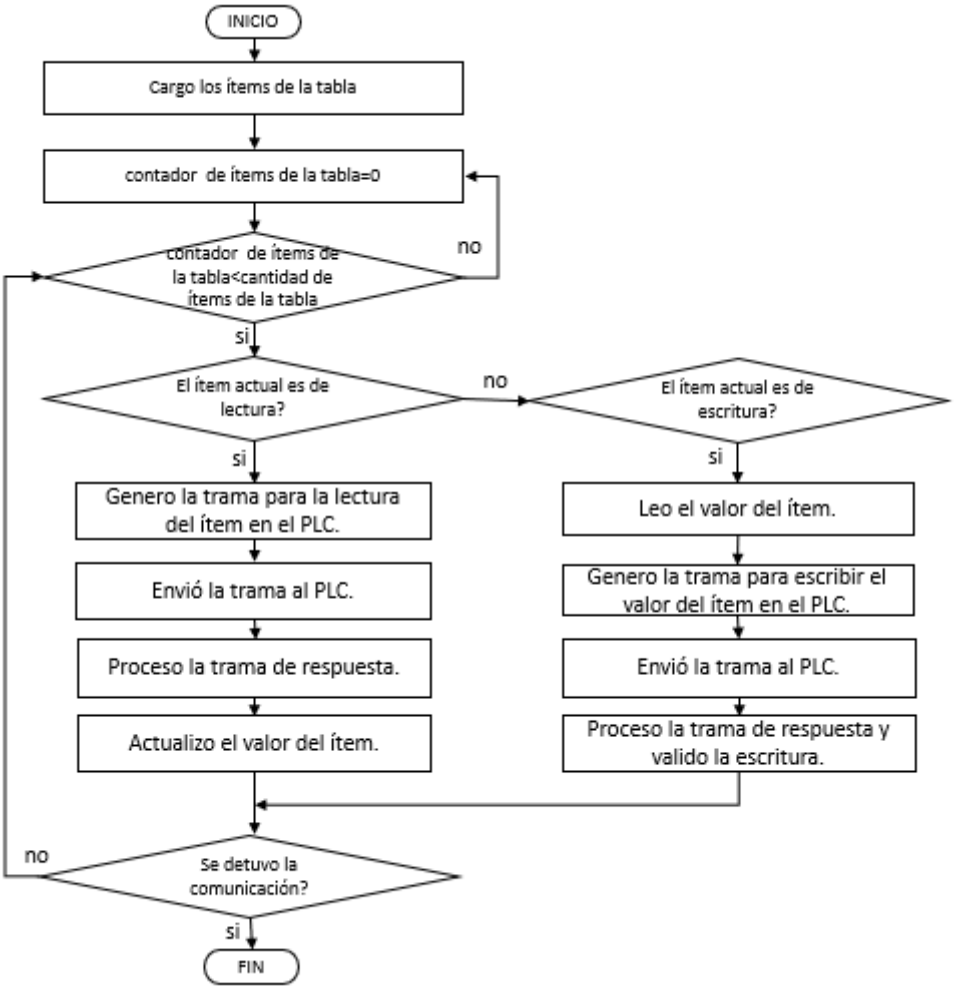

Figura 2.5.13. Diagrama de flujo para lectura y escritura continua del servidor con el PLC.

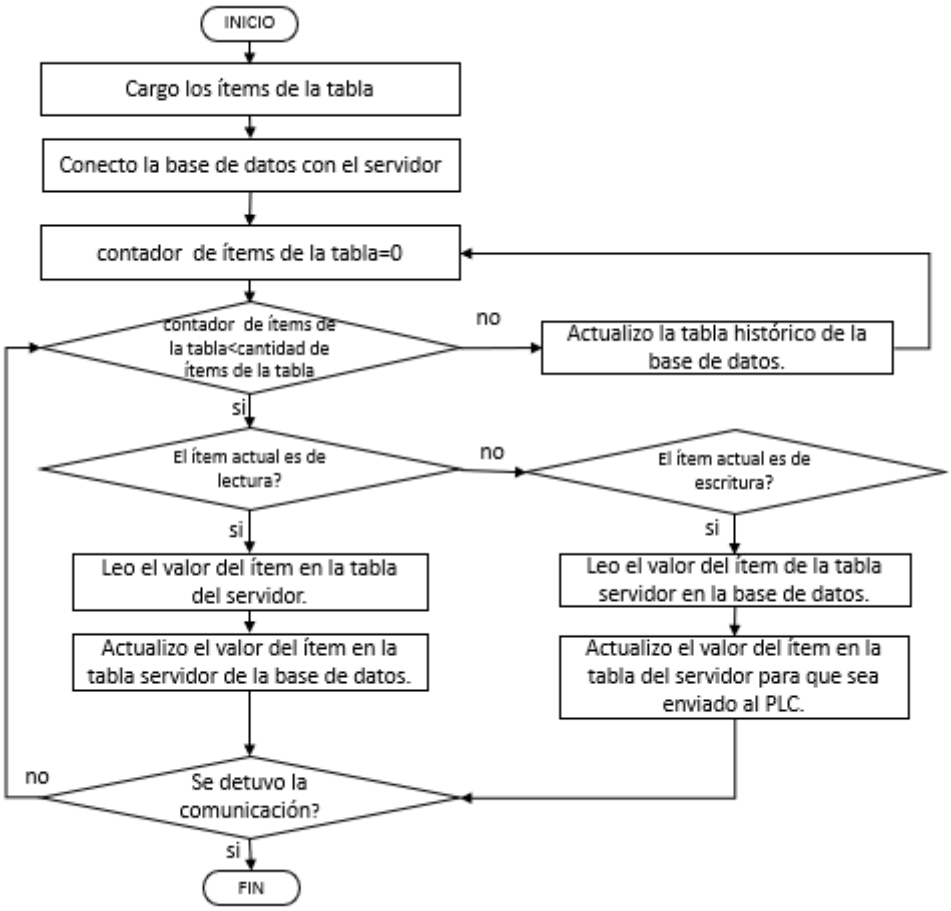

Figura 2.5.14. Diagrama de flujo para lectura y escritura continua del servidor con la base de datos.

2.5.3.6 MENSAJES DE EXCEPCIÓN.

Figura 2.5.15. Mensajes de excepción.

En la parte de mensaje de excepción se mostrarán el código de excepción generado y su significado en el caso de darse durante la comunicación.

2.6 CONEXIÓN ENTRE MYSQL E INTOUCH

Parte del presente trabajo es la validación del servidor de datos creado, para lo cual es necesario probar la interconexión con sistemas de control supervisorio, tal como lo es el Software Wonderware Intouch, el cual permite monitorear procesos a través de la comunicación con los servidores de datos provistos por el mismo fabricante. Para poder realizar la prueba y comparación del servidor creado con el servidor comercial es necesario

poder establecer comunicación con Intouch por medio de una base de datos generada en MySQL, con el cual se tiene compatibilidad.

Para poder realizar la comunicación entre MySQL e Intouch se necesita instalar el conector ODBC (Open DataBase Conectivity), que permite a Intouch utilizar peticiones SQL tal como se lo hacía con JAVA [20], una vez instalado este conector, es necesario configurarlo para que pueda comunicarse con la base de datos, los parámetros a configurar son:

- Host o dirección IP donde se encuentra MySQL
- Puerto de conexión
- Usuario con el que se accede a MySQL
- Contraseña
- Nombre de la base de datos [20]

Intouch provee la información de la configuración de las sentencias SQL necesarias para modificar o adquirir valores de las diferentes tablas en MySQL, en donde las utilizadas para el presente trabajo se muestran en la siguiente tabla:

Tabla 2.5.1. Sentencias SQL en Intouch.

Función	Descripción
SQLConnect (ConnectionId, ConnectString);	Conecta Intouch con una base de datos específica, en este caso con el conector ODBC. Ej: SQLConnect (ConnectionId," Provider=MSDASQL; DSN=MySQL");
SQLDisconnect (ConnectionId);	Desconecta Intouch con la base de datos de MySQL.
SQLUpdate (ConnectionId, TableName, BindList, WhereExpr);	Actualiza el valor de una variable dentro de una tabla específica de la base de datos, para lo cual se ingresa el nombre de la conexión, nombre de la tabla, BindList que asocia el tagname y el registro a escribir y la condición.
SQLSelect (ConnectionId, TableName, BindList, WhereExpr, OrderByExpr);	Retorna el valor de un registro de una tabla, para esto se ingresa el nombre de la conexión, nombre de la tabla, el BlindList que asocia el registro y el tagname y por último la condición. Ej: SQLSelect (ConnectionId,"servidor","LECTURA_INT","Dirección=30001",""");
SQLLast (ConnectionId);	Retorna el ultimo valor leído luego de haber ejecutado SQLSelect ().
SQLEnd (ConnectionId);	Libera memoria usada para guardar el contenido del resultado de una tabla, se ejecuta también luego de la sentencia SQLSelect ().

Para relacionar los valores de las tablas de MySQL con las variables o tagnames creados en Intouch se debe crear "Bind List", dado que es quien se encarga de asociar los tagnames con los valores de una columna específica dentro de una tabla.

3. RESULTADOS Y DISCUSIÓN.

En el presente capítulo se mostrarán los resultados obtenidos en las pruebas de funcionamiento del servidor de datos creado.

El capítulo se dividirá en tres partes, la primera es la comprobación de las tramas de solicitud y respuesta de los principales códigos de función implementados, que se realizaron tanto en el software simulador de MODBUS RTU como con el PLC Micrologix ML1100, además de la verificación del envío de códigos de excepción.

En la segunda parte del capítulo se realizará la comprobación del funcionamiento del servidor de datos realizado, para lo cual realizará un ejemplo de un proceso que dispone el uso de los principales códigos de función, que interactúan con los 4 espacios de memoria establecidas por el protocolo Modbus, el cual a su vez se interconectará con un HMI desarrollado en el software Intouch por medio de una base de datos en MySQL.

Como última parte se realizará el mismo ejemplo, pero esta vez usando el software DASMBSERIAL de Wonderware, luego del cual se detallarán las semejanzas y diferencias entre este servidor de datos y el servidor desarrollado.

3.1 PRUEBA DE FUNCIONAMIENTO MAESTRO MODBUS RTU.

Este modo de funcionamiento permite comprobar el funcionamiento de los 8 códigos de función utilizados en este trabajo, se iniciará las pruebas utilizando el software simulador MODBUS Slave, para después continuar las pruebas ya con un dispositivo físico, el PLC Micrologix ML1100 para los mismos códigos de función.

3.1.1 PRUEBA DE FUNCIONAMIENTO CON EL SIMULADOR MODBUS RTU.

Para la conexión con el software MODBUS Slave, se utilizan puertos seriales virtuales para comunicar el programa con el simulador, para este caso es necesario instalar el driver Virtual Serial Port Driver, el cual permite crear puertos seriales virtuales en pares con un comportamiento idéntico a los puertos seriales reales [21].

Para iniciar la comunicación en el simulador se debe configurar los siguientes parámetros:

- Puerto Serial
- Velocidad de transmisión
- Bits de datos
- Bits de parada
- Paridad

En la figura 3.1.1. se muestra la ventana de configuración del puerto serial.

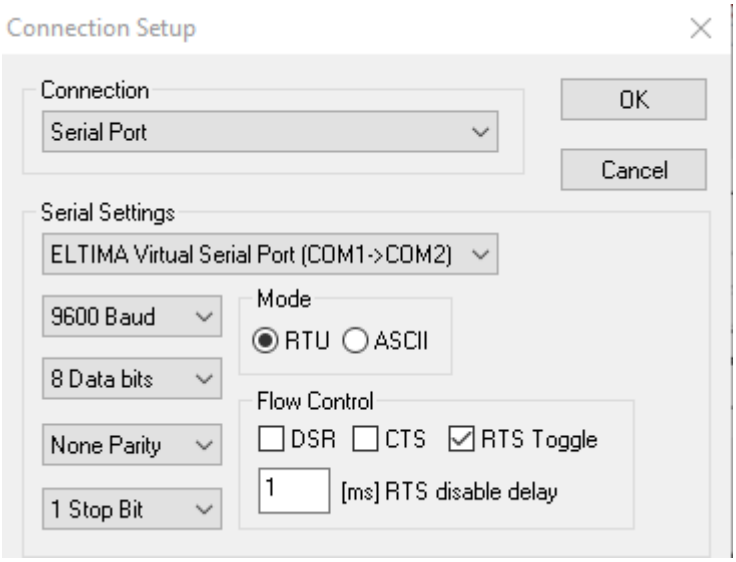

Figura 3.1.1. Configuración del puerto serial en MODBUS Slave

Al ser puertos virtuales en pares, el primer puerto se le asigna al simulador, en ese caso "COM1", mientras el maestro debe estar asociado al puerto virtual "COM2". Cabe mencionar que la configuración del puerto serial debe ser el mismo en el simulador y en el programa realizado, caso contrario la trama no se envía correctamente y el dispositivo esclavo enviará un mensaje de excepción o simplemente no responderá a la solicitud.

Ahora bien, es necesario detallar las características principales del simulador. Las cuales se destacan a continuación:

- Maneja dirección de protocolo (base 0) y dirección de PLC (base 1)
- 5 dígitos de tamaño
- Para el caso de variables flotantes, puede manejar Big endian (Float Inverso) y Little endian (Float)

Una vez conectado el puerto serial en el simulador, se procede a probar el programa.

El primer código de función a probar es FC=01, el cual permite leer coils, para probar, se leerán 5 coils, del dispositivo con ID=1, de base 1, desde la dirección 1.

Figura 3.1.2. Prueba con el simulador para lectura de coils.

En la parte izquierda de la ventana del programa se tienen los parámetros para generar la trama de solicitud para la lectura de los 5 coils, en la parte derecha en cambio se tiene el valor de los 5 coils solicitados.

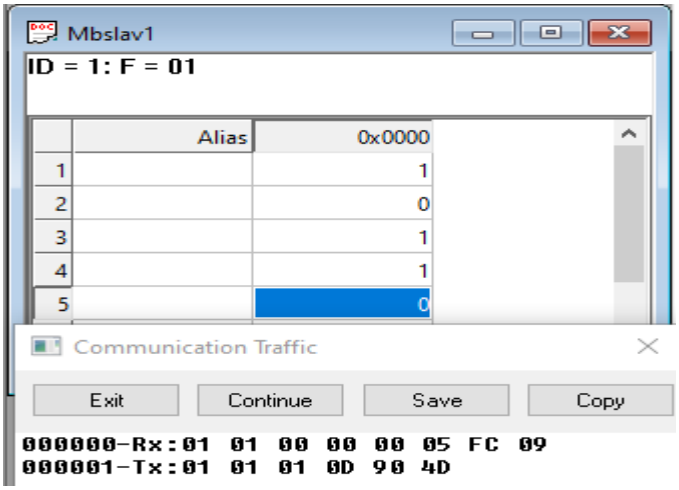

Figura 3.1.3. Valores de los coils en el simulador y muestra de la trama de solicitud y respuesta

En la figura 3.1.3 se tiene los valores de los 5 coils leídos, mientras que en la parte baja se muestra las tramas de solicitud y respuesta generadas durante el proceso.

Para el código de función FC=02, que corresponde a la lectura de contacts, se leerán el valor de 7 contacts, iniciando desde la dirección 2, del dispositivo con ID=1, con base 0.

Figura 3.1.4. Prueba de funcionamiento para lectura de contacts con el simulador.

En la parte izquierda se tienen los parámetros ingresados para solicitar leer 7 contacts, iniciando de la dirección 2. En la parte derecha la respuesta dada por el simulador.

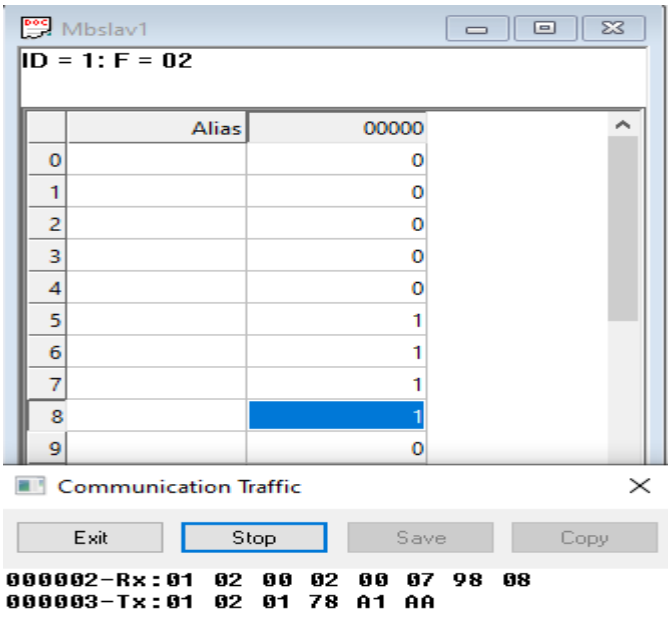

Figura 3.1.5. Valores de los contacts en el simulador y muestra de la trama de solicitud y respuesta.

En la figura 3.1.5. se observan los valores de los contacts, en la parte inferior la trama de solicitud y respuesta generada durante este proceso.

Para la función FC=03, la cual se encarga de leer registros holding, e procede a leer 3 registros holding empezando desde la dirección 40001, usando dirección de protocolo, el tipo de dato que a leer será de tipo Word.

Figura 3.1.6. Prueba de funcionamiento de lectura de registros holding con el simulador.

En la parte izquierda se observan los parámetros seleccionados para crear la trama de solicitud por parte del maestro, en la parte derecha se observar la respuesta del esclavo.

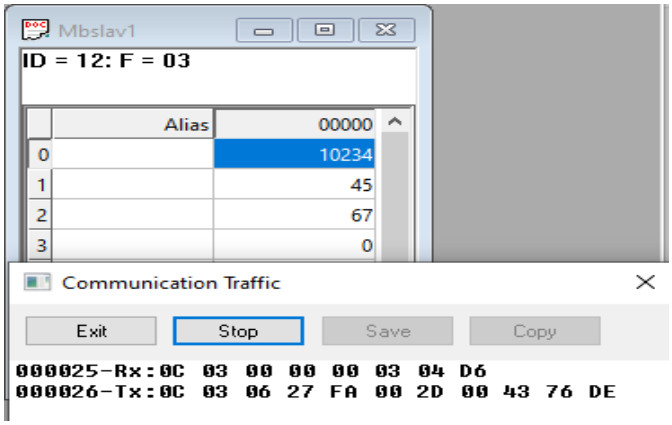

Figura 3.1.7. Valores de los registros holding en el simulador y muestra de la trama de solicitud y respuesta.

Continuando con los códigos de función, ahora se probará el código de función FC=04, el cual corresponde a la lectura de los registros de entrada o input register, en este caso se leerá 5 registros de tipo entero desde la dirección 2, del dispositivo con ID igual a 12, con dirección de protocolo.

Figura 3.1.8. Prueba de funcionamiento de lectura de registros de entrada con el simulador.

En la figura 3.1.8 se puede ver en la parte izquierda la configuración de parámetros para la lectura de 5 enteros, mientras en la parte derecha se observa los valores leídos del simulador, para la validación de la información, en la figura 3.1.9 se muestran los valores de los registros del simulador, la trama de solicitud y respuesta generada en la comunicación entre el programa creado y el simulador.

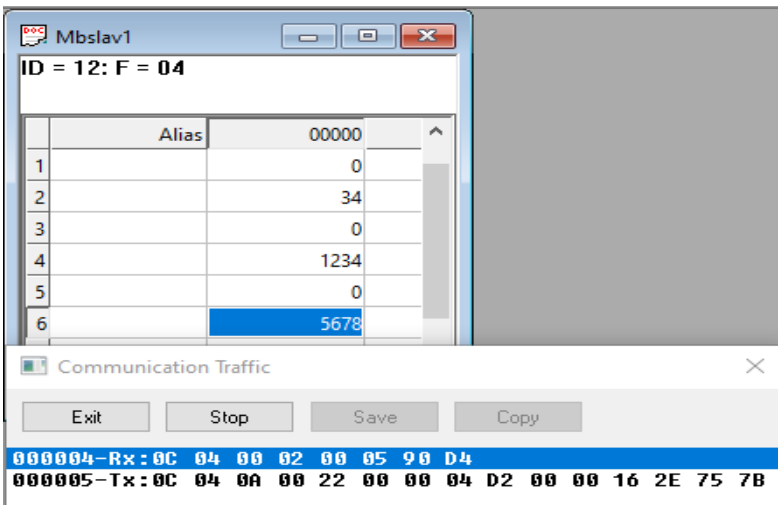

Figura 3.1.9. Valores de los registros input en el simulador y muestra de la trama de solicitud y respuesta.

Ahora, se comenzará con los códigos de función para la escritura, el primero que se probará es el código de función FC=05, el cual permite la escritura de un coil, en este caso

se escribirá el valor de “1L” en la dirección 4, del dispositivo con ID igual a 12, usando dirección de PLC.

Figura 3.1.10. Prueba de funcionamiento de escritura de un coil con el simulador.

En la parte izquierda de la figura 3.1.10 se observan los parámetros ingresados para escribir el valor de “1L” en la dirección 4 del dispositivo 12, en la parte derecha se observa un mensaje de verificación que indica que la escritura se realizó correctamente.

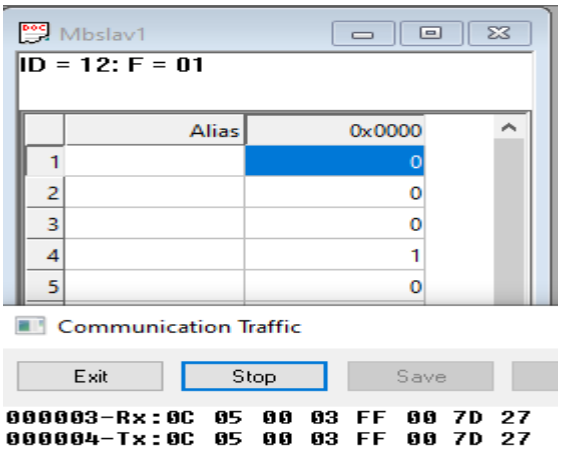

Figura 3.1.11. Valores de los registros holding en el simulador y muestra de la trama de solicitud y respuesta.

En la figura 3.1.11 se observa el valor escrito en el simulador y la trama de solicitud y respuesta ejecutada para este código de función, en donde se puede notar que la trama de solicitud y respuesta son idénticas, lo cual ya se lo había detallado en el capítulo anterior.

El siguiente código de función para escritura es el FC=06, el cual corresponde a la escritura de un registro holding, para el cual se escribirá en el registro 40020 usando dirección de PLC un dato de tipo entero con el valor “450”, en el dispositivo cuya ID es 12.

Figura 3.1.12. Prueba de funcionamiento de escritura de un registro holding con el simulador.

En la figura 3.1.12, se observa en la parte izquierda los parámetros establecidos para escribir el valor “450” en el dispositivo 12, a continuación, en la figura 3.1.13 se muestra el valor escrito en el simulador y las tramas de solicitud y respuesta generadas.

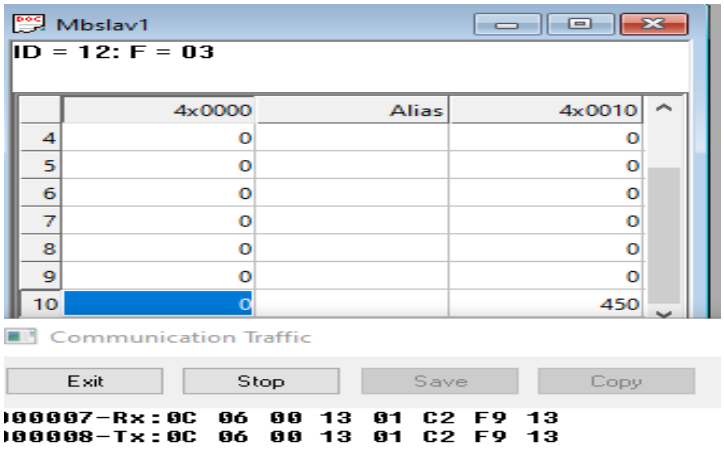

Figura 3.1.13. Valores de los registros holding en el simulador y muestra de la trama de solicitud y respuesta.

El siguiente código de función FC=15, se encarga de escribir múltiples coils, para probar este código de función, se desea escribir en el dispositivo con ID=10, usando base 0, los

valores de “1L” iniciando del registro 0 al registro 7, para lo cual se ingresa el valor 255 que en binario es 11111111, en la figura 3.1.14 se puede observar la configuración para el envío de la solicitud en la parte izquierda, en la parte derecha se observa el mensaje de envío correcto lo que significa que se escribieron correctamente los valores en el simulador.

Figura 3.1.14. Prueba de funcionamiento de escritura de múltiples coils con el simulador.

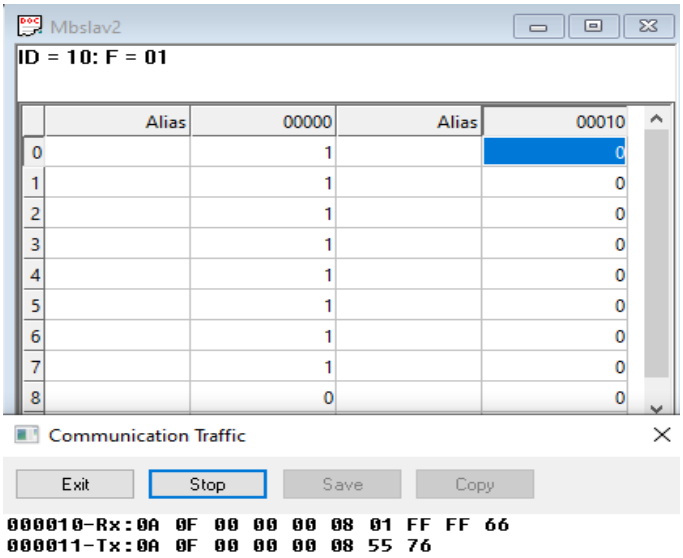

Figura 3.1.15. Valores de los registros coils escritos en el simulador y muestra de la trama de solicitud y respuesta.

En la figura 3.1.15 se observa que se escribió con el valor de “1L” los ochos primeros registros coils, lo cual era lo solicitado desde el programa creado.

El último código de función implementado es el FC=16, el cual escribe múltiples registros holding, los valores a escribir pueden ser enteros o flotantes, para el caso de simulación se probará con la escritura de 2 valores flotantes, en el dispositivo con ID=5, con little endian, dirección de PLC, iniciando desde el registro 5, en la figura 3.1.16 se observa la configuración realizada para escribir los valores dos valores flotantes que serán -13.5 y 3200.75 respectivamente, en la parte derecha se observará escritura correcta lo que significa que los datos fueron escritos correctamente en el simulador.

Figura 3.1.14. Prueba de funcionamiento de escritura de múltiples registros holding con el simulador

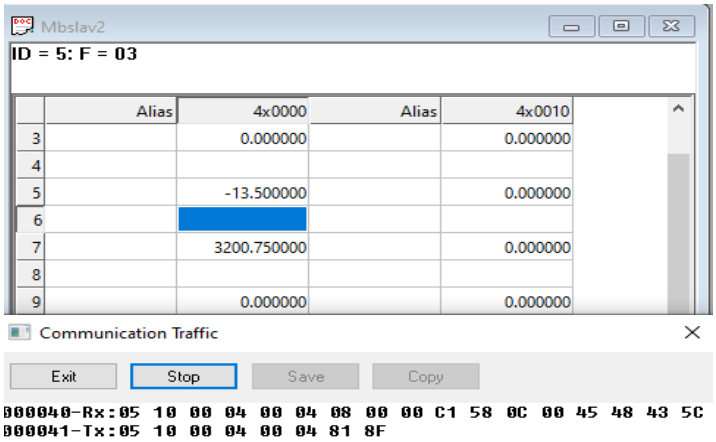

Figura 3.1.17. Valores de los registros holding en el simulador y muestra de la trama de solicitud y respuesta.

En la figura 3.1.17 se observan los dos valores flotantes escritos, que inicia en el registro 5 y las tramas de solicitud y respuestas que se generaron durante la comunicación.

3.1.2 PRUEBA DE FUNCIONAMIENTO MAESTRO MODBUS RTU CON EL PLC MICROLOGIX ML1100.

Una vez comprobado el funcionamiento con el software simulador, se procede a verificar si el programa se comunica correctamente con el dispositivo esclavo, en este caso con el PLC Micrologix ML1100 de la marca Allen Bradley, para lo cual se realizarán las pruebas para los 8 códigos de función, para lo cual se ha configurado al dispositivo con los siguientes parámetros:

- ID del dispositivo: 1
- Puerto: COM5
- Velocidad de transmisión: 9600
- Paridad: ninguna
- Bits de datos: 8
- Bits de parada:1

En la figura 3.1.18 se observa la configuración del dispositivo para que pueda trabajar como esclavo modbus RTU. Donde es necesario crear 4 nuevos espacios de memoria que almacenarán los diferentes registros que maneja el protocolo MODBUS.

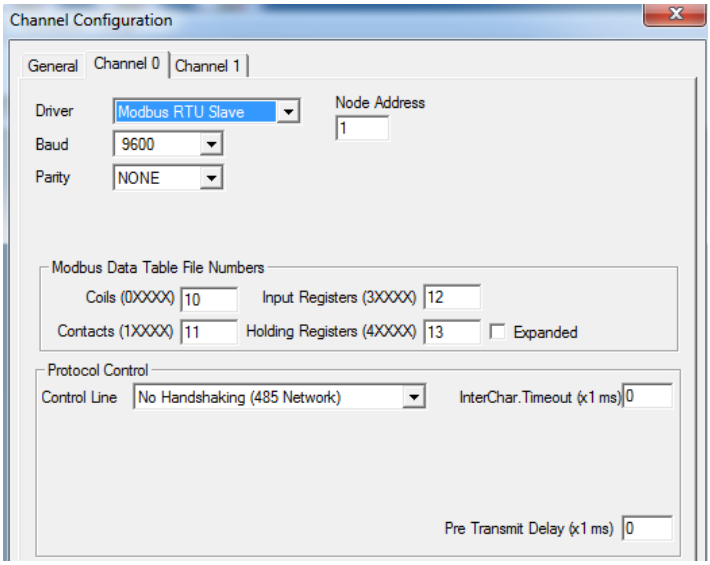

Figura 3.1.18. Configuración Micrologix ML1100 como esclavo Modbus.

Se inicia con el código de función FC=01, para lo cual se leerán los 4 primeros registros coils como se observa en la figura 3.1.19 mientras que en la figura 3.1.20 se observan los valores que tenían dichos registros, así se comprueba que el programa se comunica correctamente con el dispositivo esclavo y además funciona correctamente el código de función para la lectura de coils.

Figura 3.1.19. Prueba de funcionamiento de lectura de coils con el dispositivo ML1100.

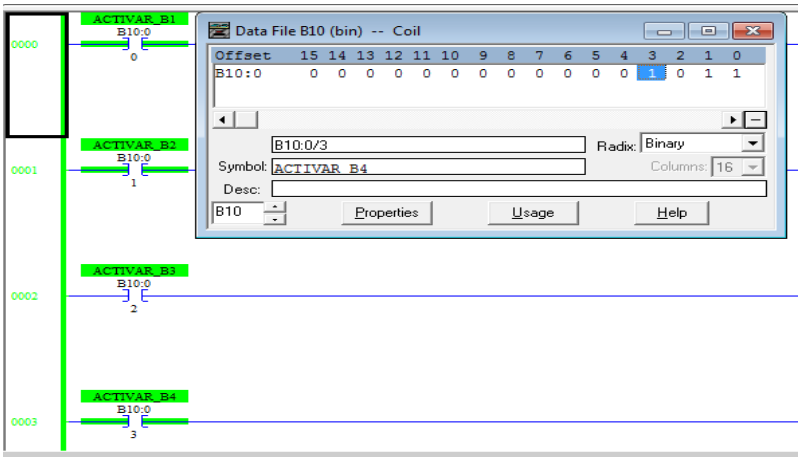

Figura 3.1.20. Valores de los registros coils en el dispositivo ML1100

Para el código de función FC=02, el cual realiza la lectura de contacts, se tomará la misma configuración del PLC sólo se modificará la dirección de protocolo para mayor facilidad de entendimiento de los registros, entonces en este caso se leerán los registros 2 y 3 únicamente, la configuración para este caso se puede observar en la figura 3.1.21 y la verificación se observa en la figura 3.1.22.

Figura 3.1.21. Prueba de funcionamiento Para lectura de contacts con el dispositivo ML1100.

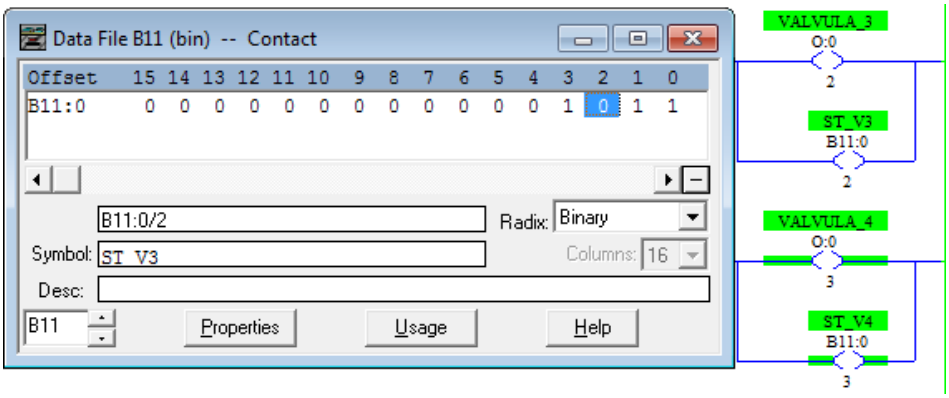

Figura 3.1.22. Valores de los registros contacts en el dispositivo ML1100.

Para el código de función FC=03, se sabe que, en el caso de variables enteras, el PLC ML1100 trabaja con enteros con signo, (valores positivos y negativos) en este caso se verificará que se lea correctamente este tipo de enteros, leyendo 3 registros iniciando desde el registro 6 hasta el 8 como se observa en la figura 3.1.23 y 3.1.24.

Figura 3.1.23. Prueba de funcionamiento de lectura de registros holding con el dispositivo ML1100.

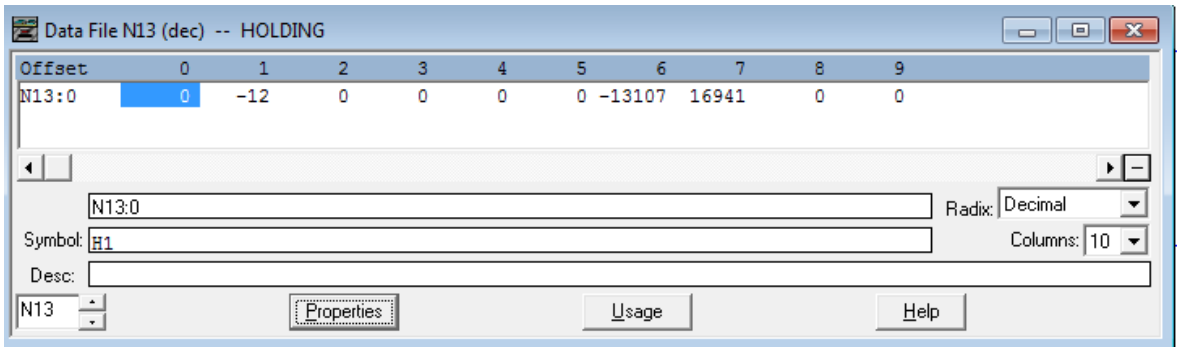

Figura 3.1.24. Valores de los registros holding en el dispositivo ML1100.

Ahora bien, se utilizará el código de función FC=04 para leer el valor de una entrada analógica, en este caso la entrada analógica es un entero que se encuentra en el registro 0, en la figura 3.1.25 y 3.1.26 se observa el valor de esta variable analógica.

Figura 3.1.25. Prueba de funcionamiento de lectura de registros de entrada con el dispositivo ML1100.

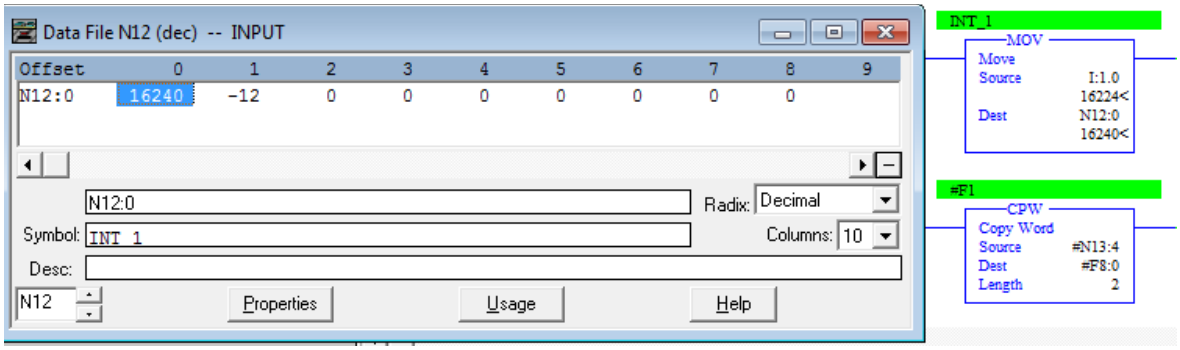

Figura 3.1.26. Valores de los registros de entrada en el dispositivo ML1100.

Ahora bien, se iniciará con las funciones de escritura, la primera es FC=05, que permite escribir un coil, para lo cual se pondrá el valor del registro 0 de los coils a “0L”, en la figura 3.1.27 y 3.1.28 se muestra la configuración realizada en el programa y el valor de la variable en el dispositivo esclavo.

Figura 3.1.27. Prueba de funcionamiento de escritura de un coil con el dispositivo ML1100.

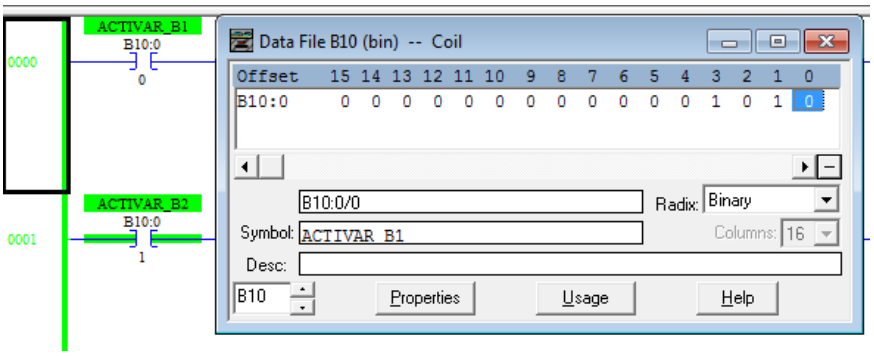

Figura 3.1.28. Valor del coil 0 escrito en el dispositivo ML1100.

El código de función FC=06, permite la escritura de un registro holding, para probar su funcionamiento se escribirá el valor máximo de un entero con signo que es 32767 en el registro 1, como se observa en la figura 3.1.29 y 3.1.30 se realizó correctamente la escritura.

Figura 3.1.29. Prueba de funcionamiento de escritura de un registro holding con el dispositivo ML1100.

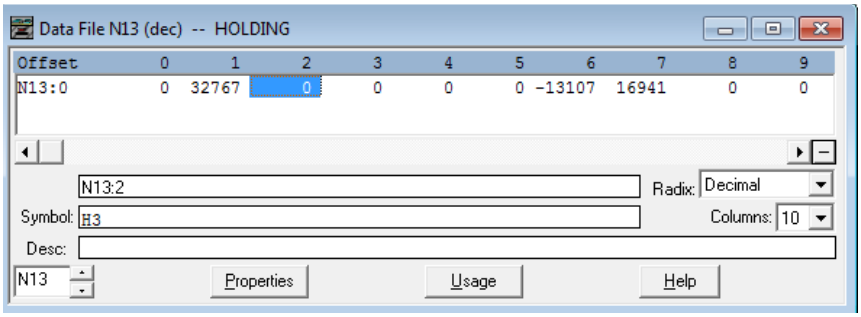

Figura 3.1.30. Valor escrito en el registro holding 1 del dispositivo ML1100

El código de función FC=15, permite la escritura de múltiples coils, para probar su funcionamiento se escribirá el valor de “0L” en los 4 primeros registros, como se podrá observar en la figura 3.1.31 y 3.1.32.

Figura 3.1.31. Prueba de funcionamiento de escritura de múltiples coils con el dispositivo ML1100.

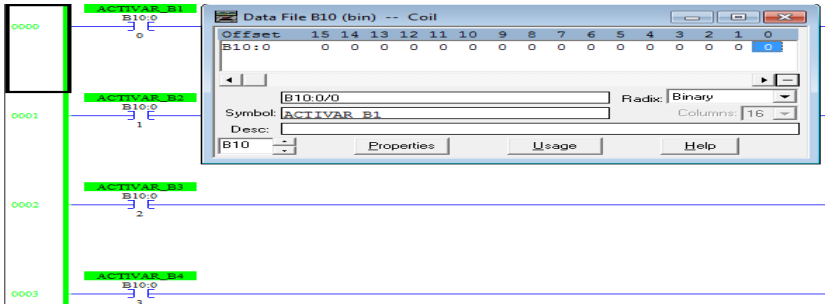

Figura 3.1.32. Valores escritos en los registros coils del dispositivo ML1100

El código de función FC=16, que permite escribir múltiples holdings, para lo cual el mejor ejemplo es la escritura de una variable de tipo flotante, ya que ésta requiere escribir 2 registros, para lo cual se escribirá el valor 99.75 en el registro 4 y 5 de los holdings.

Figura 3.1.33. Prueba de funcionamiento de escritura de múltiples registros holding con el dispositivo ML1100.

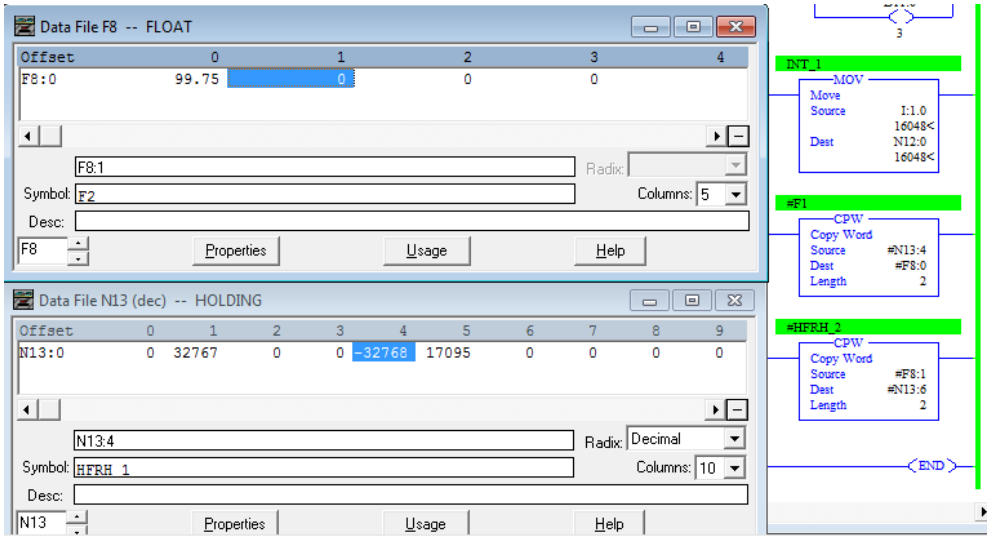

Figura 3.1.34. Valores escritos en los registros holding 4 y 5 del dispositivo ML1100.

Como se observa en la figura 3.1.33 se realiza el envío del dato de tipo flotante cuyo valor es 99.75, los cuales en la figura 3.1.34 se observa que se almacenan en 2 registros, los cuales son el 4 y el 5 de los registros holding, los cuales se observan en su valor como entero, pero al reconstruirlo como flotante da el valor enviado inicialmente. Con lo cual se comprueba el correcto funcionamiento del programa.

3.2 PRUEBA DE FUNCIONAMIENTO DEL SERVIDOR DE DATOS MODBUS RTU

En el modo de funcionamiento como servidor, la comunicación tanto con el dispositivo esclavo como con la base de datos es de manera continua, se utilizarán 7 de los 8 códigos de función desarrollados, se accederá a cada código de función de acuerdo al subíndice de memoria (dirección) y la acción que se desea realizar, es decir, leer o escribir un registro, esto se puede observar en la tabla 3.2.1.

Tabla 3.2.1. Código de función en base a su dirección y acción.

Dirección	Tipo de registro	Acción	Código de función
0 XXXX	Coil	Lectura	01
		Escritura	05
1 XXXX	Contact	Lectura	02
3 XXXX	Registro Input	Lectura	04
4 XXXX	Registro Holding	Lectura (Entero o Float)	03
		Escritura (Entero)	06
		Escritura (Float)	16

Para la comprobación del funcionamiento en modo servidor se realizarán 2 tipos pruebas, la primera usando el software simulador MODBUS RTU esta se podrá observar en la sección 3.2.1, la segunda prueba es utilizando el PLC Micrologix ML1100 la cual se detalla en la sección 3.2.2.

Se realizará la simulación del llenado y vaciado de un tanque, el cual tiene un valor máximo y un mínimo seteado por el usuario, posee una bomba que llena el tanque y una válvula que permite vaciar el tanque, éstas son accionadas por el usuario, posee 2 luces que indican cuando el tanque llega al nivel máximo o cuando llega al nivel mínimo, se tiene la lectura de una variable analógica de tipo entera y su valor escalado de tipo flotante que representa de 0 a 100% el llenado del tanque. El proceso dispone de todos los códigos de función descritos.

En la tabla 3.2.2 se pueden observar las variables a utilizar en el proceso.

Tabla 3.2.1. Mapa MODBUS del proceso.

VARIABLE	TIPO	ACCIÓN	DIRECCIÓN
start_b1	Bool	escritura	1
stop_b1	Bool	escritura	2
start_b2	Bool	escritura	3
stop_b2	Bool	escritura	4
estado_b1	Bool	lectura	10001
estado_b2	Bool	lectura	10002

nivel_raw	Int	lectura	30001
nivel_scl	Float	lectura	30002
nvl_max	Float	escritura	40001
nvl_min	Float	escritura	40003
st_max	Bool	lectura	10003
st_min	Bool	lectura	10004

Ambas pruebas simulan el proceso descrito anteriormente, en cada prueba se observa el desempeño del servidor conectándolo a un HMI desarrollado en Intouch para el monitoreo.

3.2.1 PRUEBA DE FUNCIONAMIENTO CON EL SIMULADOR MODBUS RTU

El primer paso a realizar en el servidor de datos es la configuración de los parámetros de conexión con el puerto serial del simulador y su configuración.

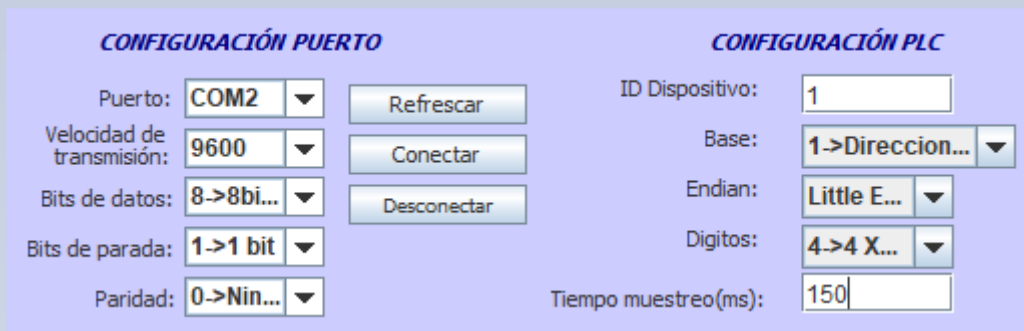

Figura 3.2.1. Configuración de los parámetros del simulador MODBUS RTU.

En segundo paso es la creación de la base de datos que almacenará la tabla histórica y la tabla servidor, para lo cual se debe ingresar el nombre que se le dará a la base de datos, el usuario y contraseña con la que se accede a MySQL, se recomienda evitar el uso del usuario root, debido a que es un usuario local y que más adelante no permitirá conectar la base de datos a Intouch.

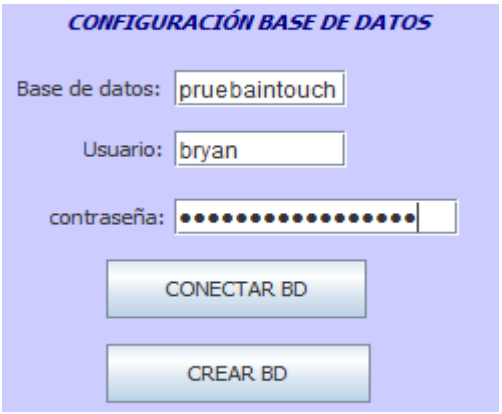

Figura 3.2.2. Creación de la base de datos en MySQL.

Se procede con el ingreso de las variables a monitorear con el servidor, para cada ítem a ingresar se debe ingresar el nombre, el tipo de variable, la acción que se desea realizar con esa variable y la dirección, aquí además se debe considerar el número de dígitos y la base, seleccionados anteriormente, en caso de agregar mal un parámetro, el ítem no se agregará a la tabla. También es importante recalcar que no se debe repetir el nombre de los ítems ingresados en la tabla ya que esto provoca un error al momento de crear las tablas en MySQL.

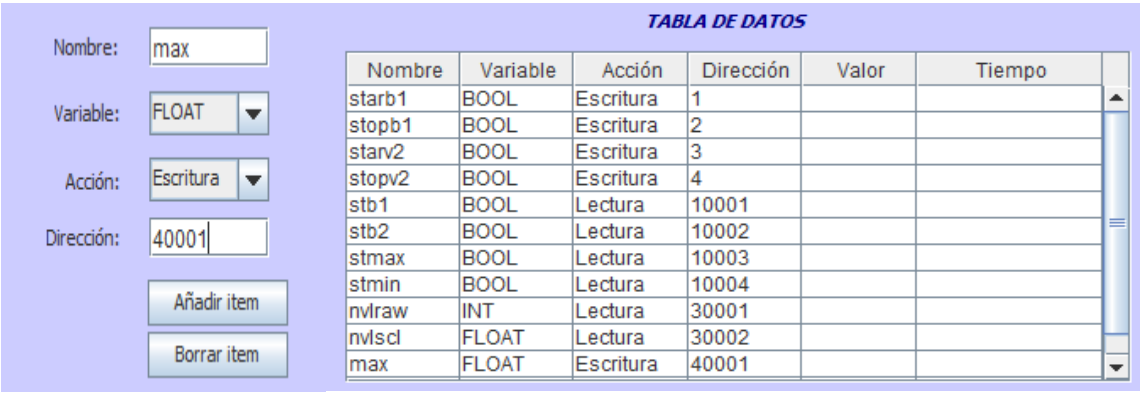

Figura 3.2.3. Ingreso de las variables del proceso.

Ahora se procede a la creación de las tablas servidor e históricos en la base de datos creada anteriormente.

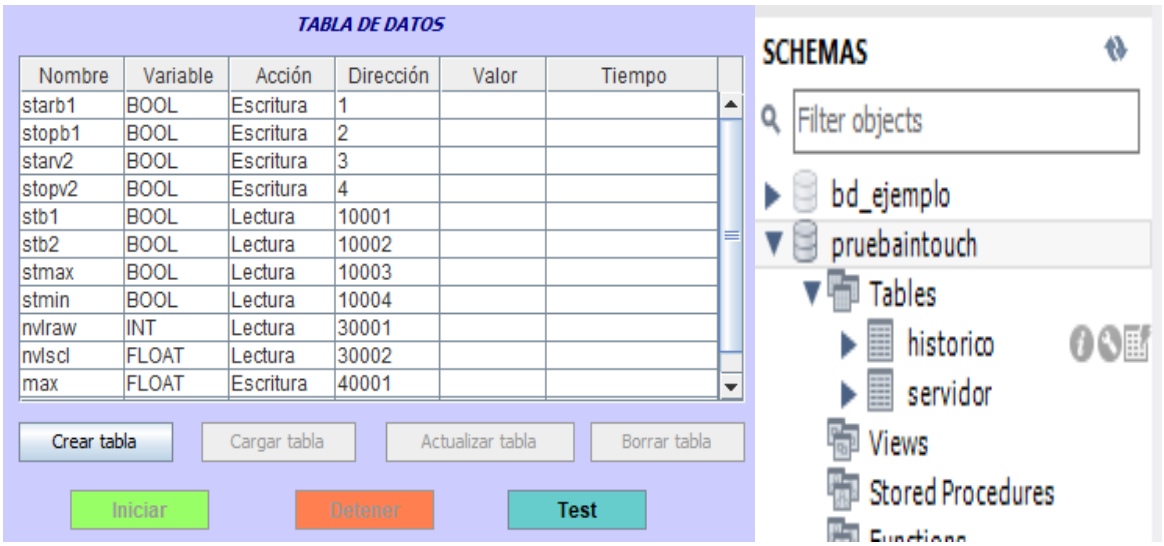

Figura 3.2.4. Creación de las tablas del proceso en MySQL.

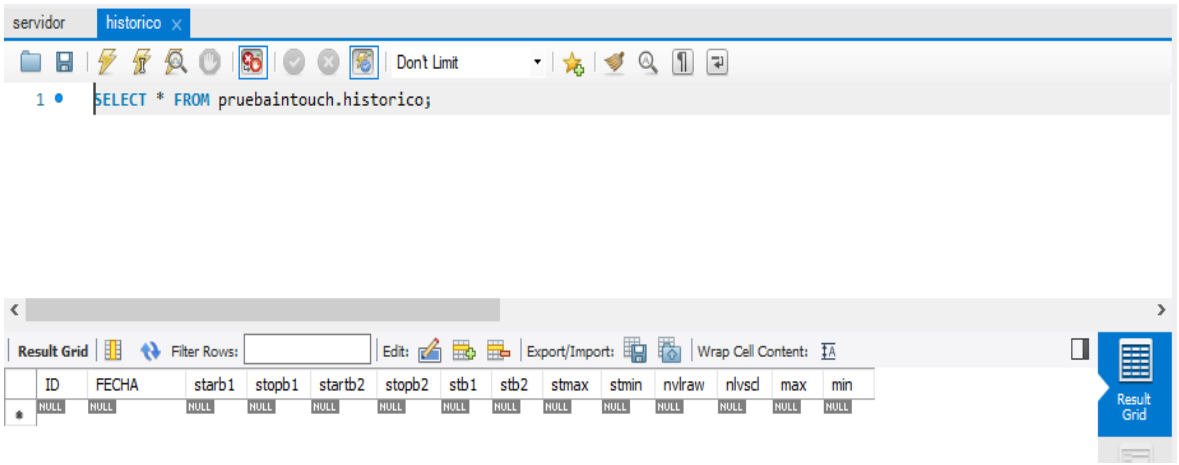

Figura 3.2.5. Tabla "histórico" creada en MySQL.

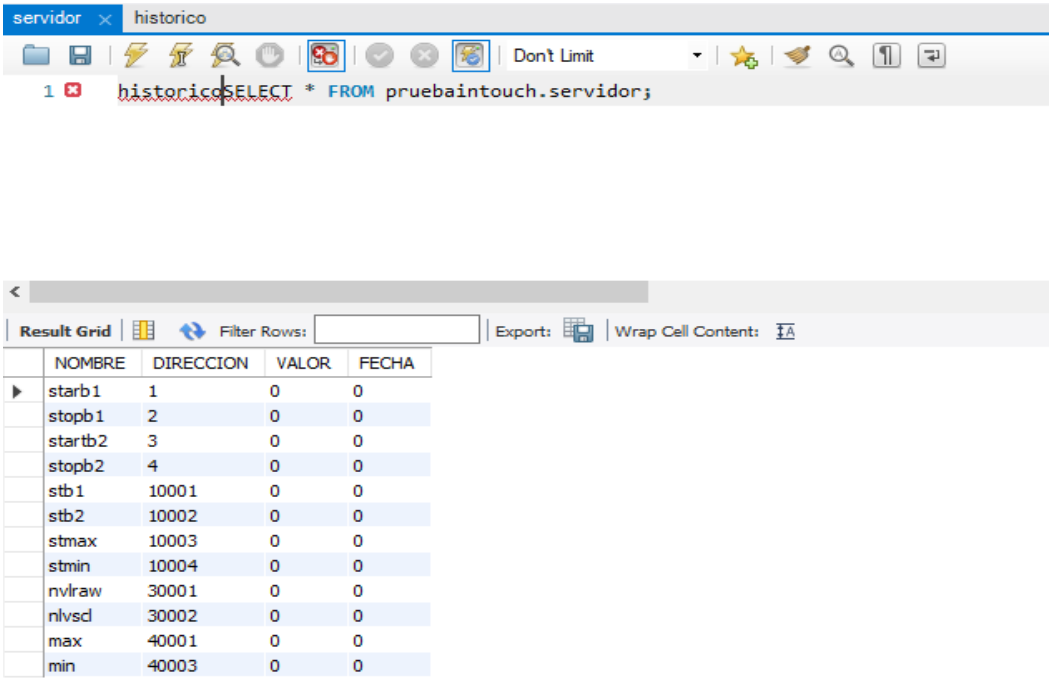

Figura 3.2.6. Tabla servidor creada en MySQL.

Dar clic en iniciar para poner en funcionamiento al servidor, en la pantalla de java que se encuentra en la figura 3.2.7 se observa los valores en tiempo real de las variables que tiene el simulador.

Figura 3.2.7. Funcionamiento del servidor de datos creado.

En la figura 3.2.8 y 3.2.9 se puede observar cómo se almacenan los valores del servidor de java en la base de datos, tanto en la tabla servidor como en la tabla histórico.

	NOMBRE	DIRECCION	VALOR	FECHA
▶	starb1	1	1	11:36...
	stopb1	2	0	11:36...
	startb2	3	1	11:36...
	stopb2	4	0	11:36...
	stb1	10001	1	11:36...
	stb2	10002	1	11:36...
	stmax	10003	0	11:36...
	stmin	10004	0	11:36...
	nvlraw	30001	12000	11:36...
	nlvscl	30002	45,00	11:36...
	max	40001	80.0	11:36...
	min	40003	10.0	11:36...

Figura 3.2.8. Almacenamiento de valores en la tabla servidor.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	ID	FECHA	starb1	stopb1	startb2	stopb2	stb1	stb2	stmax	stmin	nvlraw	nlvscl	max	min
	80	11:35 03 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	81	11:35 09 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	82	11:35 16 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	83	11:35 21 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	84	11:35 24 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	85	11:35 28 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	86	11:35 36 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	87	11:35 40 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	88	11:35 44 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	89	11:35 51 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	90	11:35 56 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	91	11:36 04 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0
	92	11:36 08 0...	1	0	0	0	1	1	0	0	12000	45,00	80.0	10.0

Figura 3.2.9. Almacenamiento de valores en la tabla histórico.

El servidor de datos desarrollado cuenta con el botón “Test”, la cual es útil para variables de tipo escritura en el caso que sea necesario dar valores a una variable, para lo cual se puede ingresar el nombre de la variable que se quiere escribir y el valor que se le va a dar a dicha variable, como se observa en la figura 3.2.10.

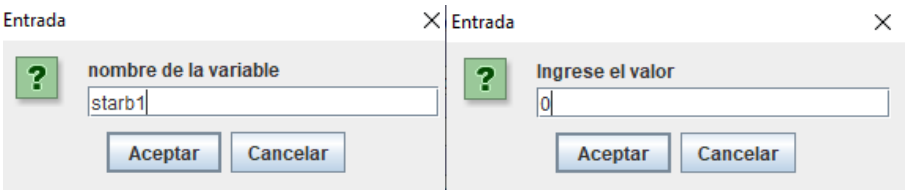

Figura 3.2.10. Escribir una variable usando el botón Test.

Como parte de lo planteado en el presente proyecto es la comunicación del servidor con Intouch, para lo cual se desarrolló un HMI en dicho software, en la figura 3.2.11 se puede observar el HMI con los valores de las variables almacenadas en la base de datos por el servidor creado.

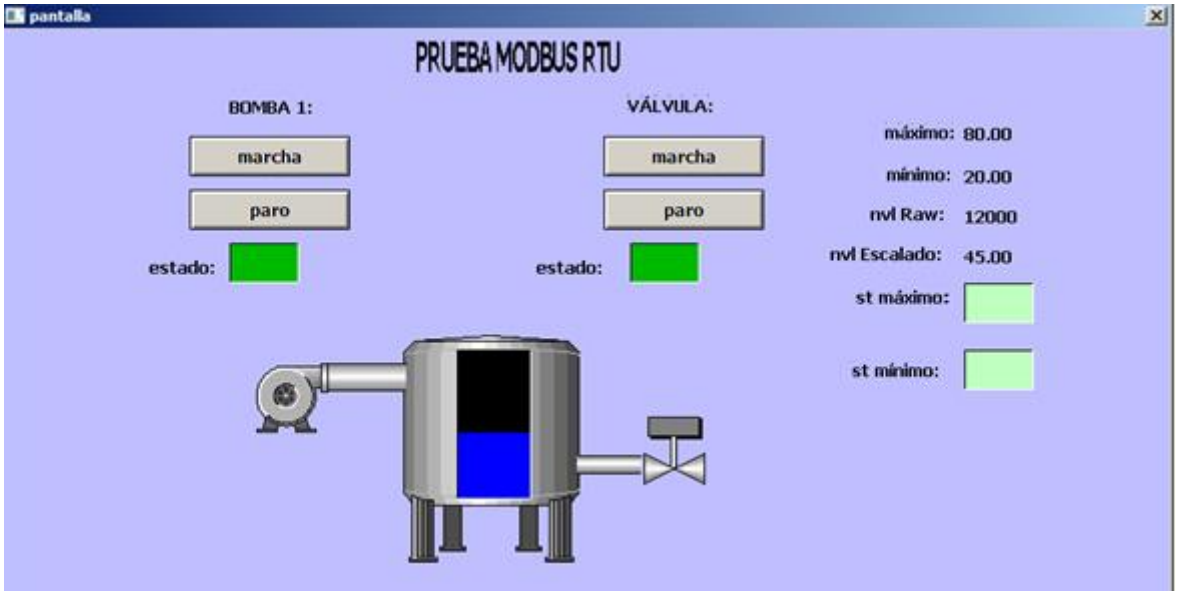

Figura 3.2.11. Monitoreo del proceso en Intouch.

Como se puede observar en las figuras 3.2.12 y 3.2.13, los valores visualizados en el HMI coinciden con los valores almacenados en las tablas de la base de datos creada en MySQL, la figura 3.2.12 es la tabla histórico encargada de almacenar la información del proceso cada que completa la lectura de todas las variables del proceso, en cambio, la figura 3.2.13 muestra la tabla servidor la que permite visualizar los valores del proceso en tiempo real, esta tabla es la que se relaciona con el HMI para poder monitorear los variables de manera gráfica.

ID	FECHA	starb1	stopb1	starb2	stopb2	stb1	stb2	stmax	stmin	nlvraw	nvlscl	max	min
89	12:20 11 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
90	12:20 19 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
91	12:20 25 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	20.00
92	12:20 30 1...	1	0	0	0	1	0	0	0	11500	45,00	80.00	20.00
93	12:20 38 1...	1	0	0	0	1	0	0	0	12000	45,00	80.00	20.00
94	12:20 47 1...	1	0	0	0	1	1	0	0	12000	45,00	80.00	20.00
95	12:20 53 1...	1	0	0	0	1	1	0	0	12000	45,00	80.00	20.00
96	12:20 58 1...	1	0	1	0	1	1	0	0	12000	45,00	80.00	20.00
97	12:21 06 1...	1	0	1	0	1	1	0	0	12000	45,00	80.00	20.00
98	12:21 15 1...	1	0	1	0	1	1	0	0	12000	45,00	80.00	20.00
99	12:21 19 1...	1	0	1	0	1	1	0	0	12000	45,00	80.00	20.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.2.12. Valores almacenados en la tabla histórico.

NOMBRE	DIRECCION	VALOR	FECHA
starb1	1	1	12:22 16 11-12-21
stopb1	2	0	12:22 16 11-12-21
starb2	3	1	12:22 17 11-12-21
stopb2	4	0	12:22 17 11-12-21
stb1	10001	1	12:22 22 11-12-21
stb2	10002	1	12:22 22 11-12-21
stmax	10003	0	12:22 22 11-12-21
stmin	10004	0	12:22 20 11-12-21
nlvraw	30001	12000	12:22 20 11-12-21
nvlscl	30002	45,00	12:22 20 11-12-21
max	40001	80.00	12:22 20 11-12-21
min	40003	20.00	12:22 21 11-12-21

Figura 3.2.13. Valores almacenados en la tabla servidor.

Ahora bien, los valores almacenados en la base de datos deben coincidir con los valores en el simulador como se observa en la gráfica 3.2.14, lo que indica que el servidor desarrollado, realiza la comunicación tanto con la base de datos como con el simulador de manera exitosa.

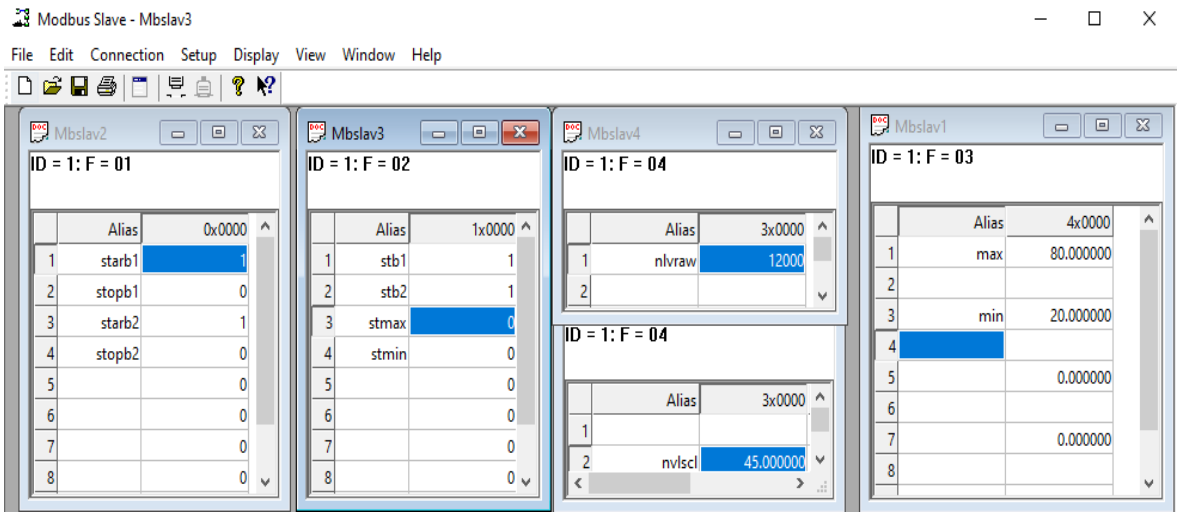

Figura 3.2.14. Valores de las variables en el simulador de MODBUS RTU.

Por último, se tiene la gráfica 3.2.15 que muestra los valores en tiempo real en el servidor de datos diseñado.

Figura 3.2.15. Ventana servidor de datos MODBUS RTU funcional.

Para comprobar que el HMI está interactuando con la base de datos, se modificó el valor de diferentes variables como se puede notar en la figura 3.2.16.

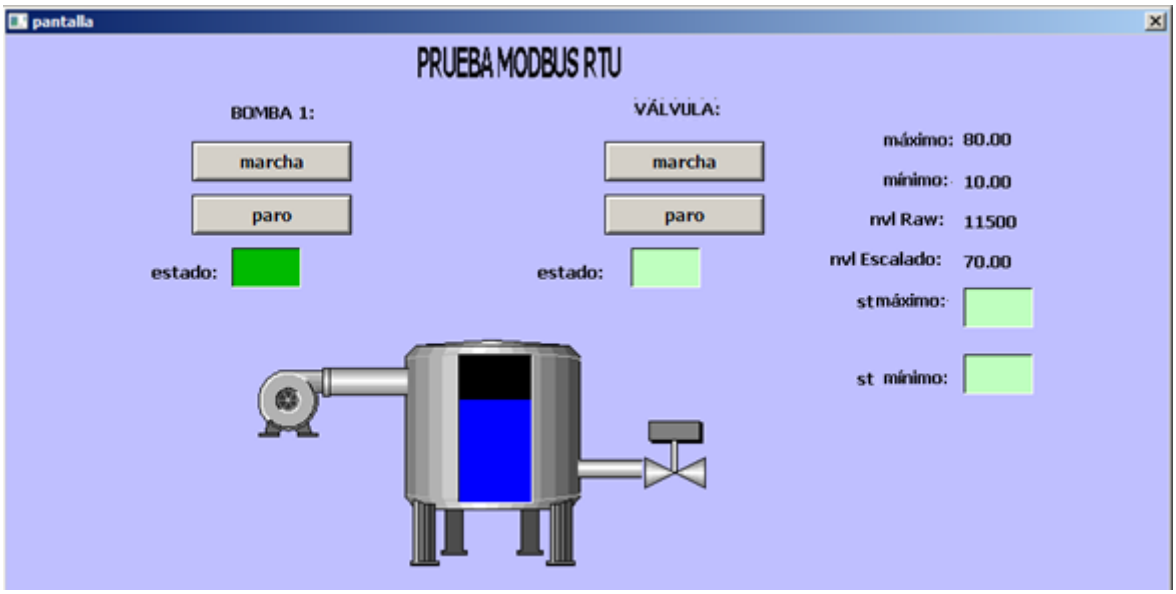

Figura 3.2.16. HMI del proceso con valores actualizados.

Las gráficas 3.2.17 y 3.2.18 muestran que efectivamente los valores modificados en el HMI fueron actualizados en las tablas histórico y servidor respectivamente, en las gráficas 3.2.19 y 3.2.20 se puede observar que éstos también fueron modificados en el software simulador de MODBUS RTU y en el servidor. Con lo que se comprueba que el HMI interactúa con el servidor de datos de manera correcta.

NOMBRE	DIRECCION	VALOR	FECHA
starb1	1	1	12:17 04 11-12-21
stopb1	2	0	12:17 05 11-12-21
starb2	3	0	12:17 00 11-12-21
stopb2	4	0	12:17 00 11-12-21
stb1	10001	1	12:17 02 11-12-21
stb2	10002	0	12:17 03 11-12-21
stmax	10003	0	12:17 03 11-12-21
stmin	10004	0	12:17 03 11-12-21
nlvraw	30001	11500	12:17 03 11-12-21
nvlscl	30002	70,00	12:17 03 11-12-21
max	40001	80.00	12:17 03 11-12-21
min	40003	10.00	12:17 04 11-12-21

Figura 3.2.17. Valores actualizados en la tabla servidor.

ID	FECHA	starb1	stopb1	starb2	stopb2	stb1	stb2	stmax	stmin	nlvraw	nvlscl	max	min
54	12:16 21 1...	1	0	0	0	1	0	0	0	11500	45,00	80.00	10.00
55	12:16 29 1...	1	0	0	0	1	0	0	0	11500	45,00	80.00	10.00
56	12:16 34 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
57	12:16 40 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
58	12:16 45 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
59	12:16 50 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
60	12:16 58 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
61	12:17 05 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
62	12:17 13 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
63	12:17 20 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
64	12:17 26 1...	1	0	0	0	1	0	0	0	11500	70,00	80.00	10.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.2.18. Valores agregados a la tabla histórico.

Figura 3.2.19. Ventana servidor de datos MODBUS RTU con valores actualizados.

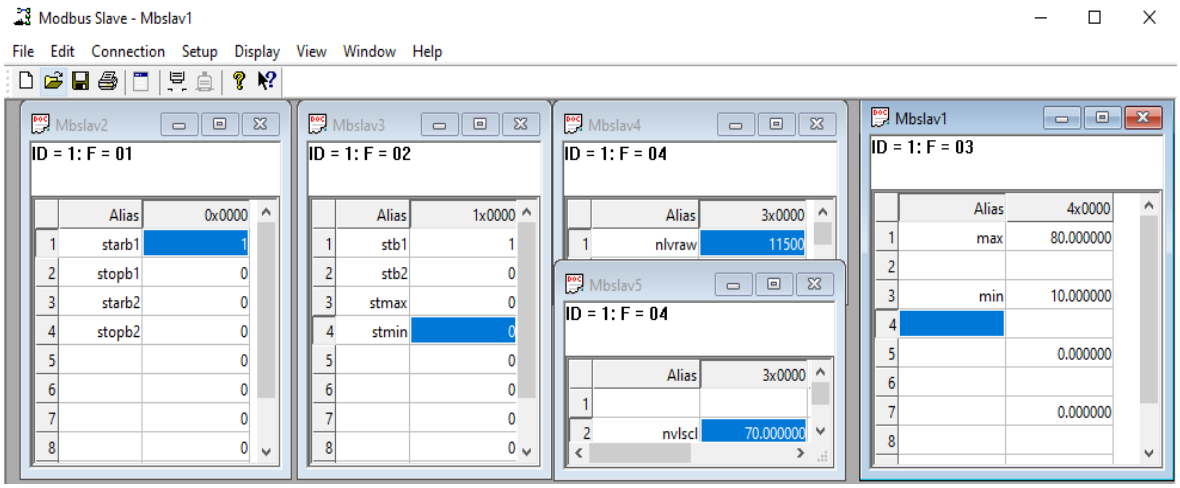

Figura 3.2.20. Valores actualizados en el simulador MODBUS RTU.

3.2.2 PRUEBA DE FUNCIONAMIENTO CON EL PLC MICROLOGIX ML1100

Una vez comprobado el funcionamiento del servidor de datos con el simulador, se procede a realizar la prueba con el PLC Micrologix ML1100, para lo cual se mantiene la base de datos y las tablas creadas, sólo se modificará la configuración del puerto serial que debe ser los mismos parámetros de comunicación del PLC, como se observa en la figura 3.2.21:

Figura 3.2.21. Configuración de los parámetros del PLC Micrologix ML1100.

En la figura 3.2.22 se observan los valores de las variables del PLC leídos por el servidor de datos, antes de realizar la comunicación con el HMI, se utilizará el botón “Test” para modificar las variables de tipo escritura y verificar que el PLC las actualice dentro de su programa.

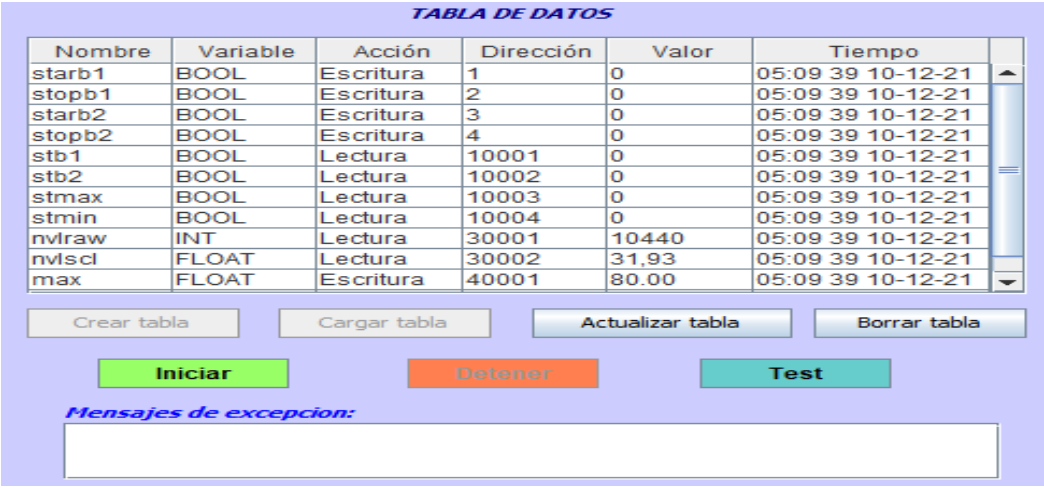

Figura 3.2.22. Valores de las variables del PLC Micrologix ML1100 leídos por el servidor de datos.

Se modificará la variable starb1, que enciende la bomba 1, su valor inicial es “0L”, su valor pasará a “1L”, lo que prenderá la bomba, como resultado deberá modificar la variable starb1 y a su vez la variable stb1 también deberá pasar a “1L” puesto que esto indicaría que la bomba fue encendida, en la figura 3.2.23 se puede observar cómo se modifica una variable usando el botón “Test.”

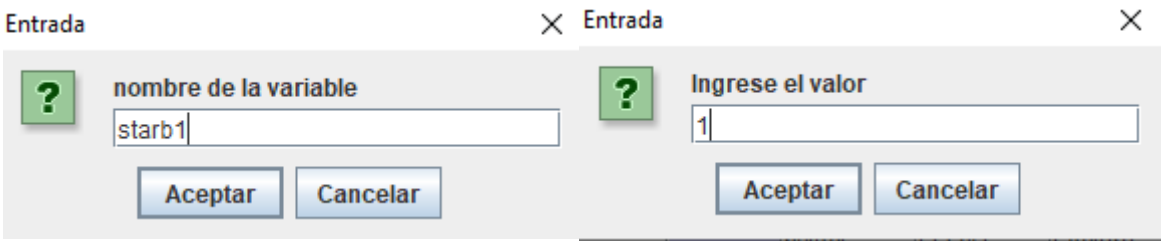

Figura 3.2.23. Actualización de la variable starb1 por medio del botón test.

Tal como se observa en la figura 3.2.24 se actualizó el valor de starb1 y el valor de stb1, lo cual demuestra que el botón “Test” funciona correctamente con dispositivos reales.

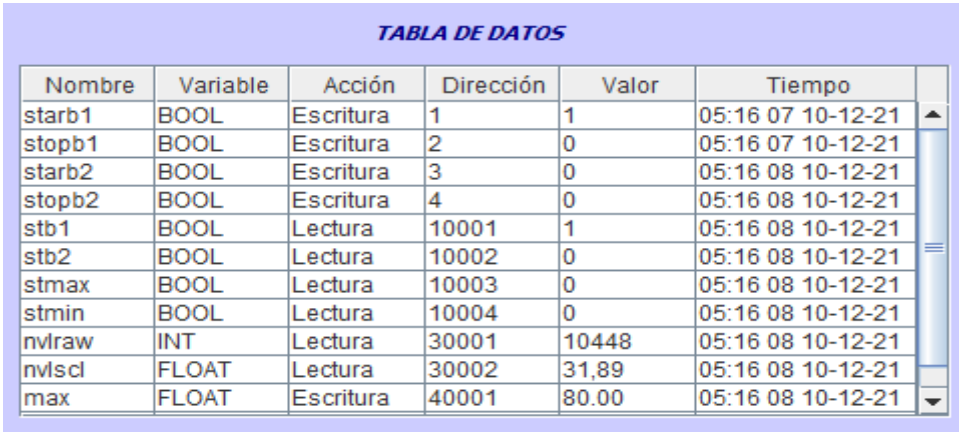

Figura 3.2.24. Valores actualizados en el servidor de datos.

Ahora bien, se procede a realizar las pruebas entre el servidor de datos creado y el HMI de Intouch, la gráfica 3.2.27 muestra el estado de las variables MODBUS del PLC, donde se tiene que la bomba 1 está encendida, el valor máximo de llenado del tanque es 80.00, el valor mínimo de llenado es 10, el valor de la variable analógica es 19168 y el valor escalado es 58.00%, estos valores deben coincidir con los valores en la base de datos, cabe destacar que en el caso de variables analógicas siempre variarán un poco en el valor porque existe un pequeño tiempo de retardo al leer las variables de la base de datos y pasarlos al HMI.

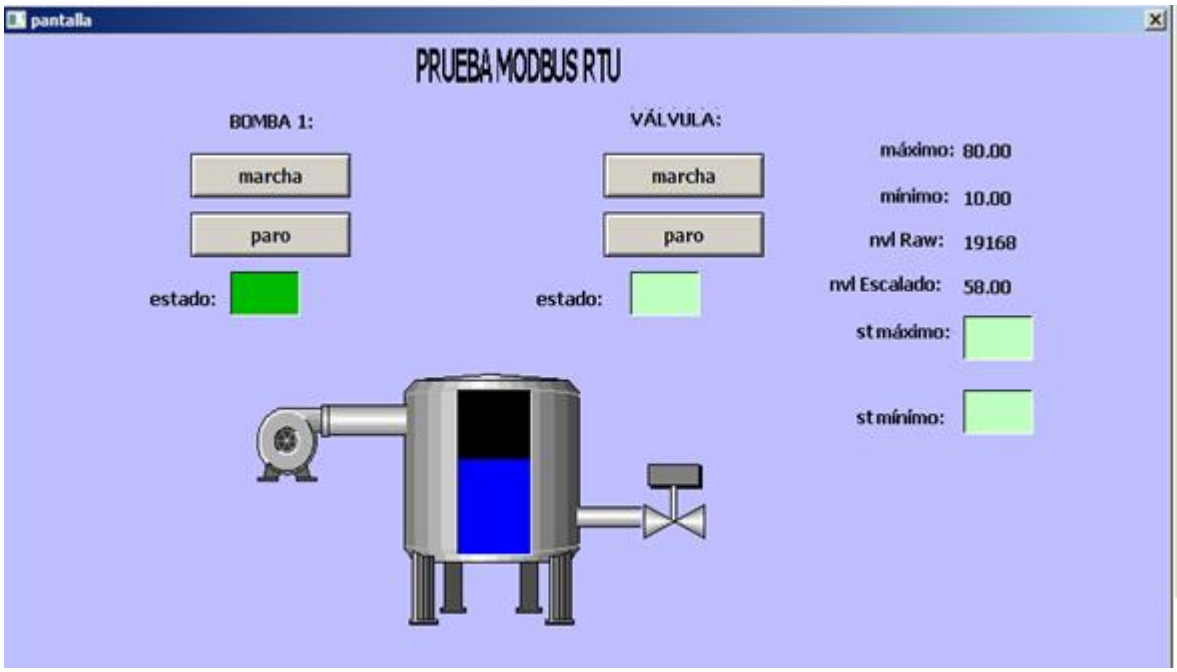

Figura 3.2.27. Monitoreo de las variables del PLC Micrologix ML1100.

En las gráficas 3.2.28 y 3.2.29 correspondientes a las tablas generadas en la base de datos, se puede observar que los valores almacenados con la base de datos se corresponden con los valores visualizados en el HMI.

ID	FECHA	starb1	stopb1	starb2	stopb2	stb1	stb2	stmax	stmin	nvlraw	nvlscl	max	min
74	05:18 22 1...	0	0	0	0	1	0	0	0	10448	31,89	80.00	0.00
75	05:18 30 1...	0	0	0	0	1	0	0	0	13712	41,87	80.00	10.00
76	05:18 36 1...	0	0	0	0	1	0	0	0	16776	51,20	80.00	10.00
77	05:18 44 1...	0	0	0	0	1	0	0	0	21432	65,48	80.00	10.00
78	05:18 52 1...	0	0	0	0	1	0	0	0	19184	58,52	80.00	10.00
79	05:18 59 1...	0	0	0	0	1	0	0	0	19160	58,52	80.00	10.00
80	05:19 07 1...	0	0	0	0	1	0	0	0	19152	58,42	80.00	10.00
81	05:19 15 1...	0	0	0	0	1	0	0	0	19144	58,42	80.00	10.00
82	05:19 21 1...	0	0	0	0	1	0	0	0	19168	58,47	80.00	10.00
83	05:19 28 1...	0	0	0	0	1	0	0	0	19152	58,45	80.00	10.00
84	05:19 34 1...	0	0	0	0	1	0	0	0	19152	58,45	80.00	10.00
85	05:19 41 1...	0	0	0	0	1	0	0	0	19144	58.42	80.00	10.00

Figura 3.2.28. Valores almacenados en la tabla histórico.

NOMBRE	DIRECCION	VALOR	FECHA
starb1	1	0	05:20 21 10-12-21
stopb1	2	0	05:20 21 10-12-21
starb2	3	0	05:20 22 10-12-21
stopb2	4	0	05:20 22 10-12-21
stb1	10001	1	05:20 25 10-12-21
stb2	10002	0	05:20 25 10-12-21
stmax	10003	0	05:20 22 10-12-21
stmin	10004	0	05:20 22 10-12-21
nvlraw	30001	19160	05:20 22 10-12-21
nvlscl	30002	58,45	05:20 24 10-12-21
max	40001	80.00	05:20 24 10-12-21
min	40003	10.00	05:20 24 10-12-21

Figura 3.2.29. Valores almacenados en la tabla servidor.

Los valores vistos en las gráficas anteriores deben corresponderse con los valores del servidor de datos, lo cual si sucede como se puede observar en la figura 3.2.30, esto indica que la comunicación entre el PLC, el servidor, la base de datos y el HMI se está realizando correctamente.

Figura 3.2.30. Monitoreo de las variables del PLC Micrologix ML1100 desde el servidor de datos creado.

Para corroborar el funcionamiento, se procedió a modificar los valores desde el HMI creado en Intouch, y verificar que éstos se modificarán tanto en la base de datos, y en el servidor que a su vez modifica al PLC, cambiando el mínimo en 30.00%, se apaga la bomba 1 y se prende la bomba 2, tal como se observa en la gráfica 3.2.31.

En la gráfica 3.2.32 se observa los valores que desde el HMI de Intouch se modificaron en la tabla servidor, mientras en la figura 3.2.33 se observa que los valores cambiaron en el

servidor de datos, por ende, también en el PLC, comprobando una vez más que el servidor de datos está comunicándose correctamente.

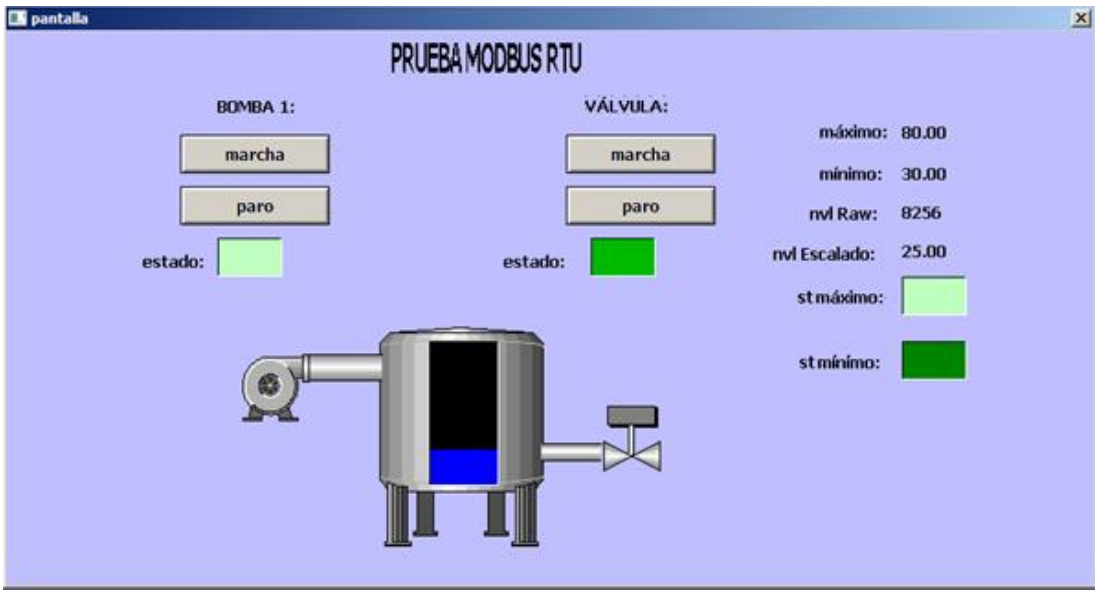

Figura 3.2.31. Modificación de las variables del proceso desde Intouch.

NOMBRE	DIRECCION	VALOR	FECHA
starb1	1	0	05:23 19 10-12-21
stopb1	2	1	05:23 19 10-12-21
starb2	3	1	05:23 21 10-12-21
stopb2	4	0	05:23 21 10-12-21
stb1	10001	0	05:23 21 10-12-21
stb2	10002	1	05:23 22 10-12-21
stmax	10003	0	05:23 22 10-12-21
stmin	10004	1	05:23 17 10-12-21
nvlraw	30001	8264	05:23 18 10-12-21
nvlscl	30002	25,22	05:23 18 10-12-21
max	40001	80.00	05:23 15 10-12-21
min	40003	30.00	05:23 17 10-12-21

Figura 3.2.32. Actualización de valores en la tabla servidor.

Figura 3.2.33. Valores actualizados en el servidor de datos.

3.3 COMPARACIÓN DEL SERVIDOR DE DATOS DESARROLLADO CON EL SOFTWARE DASMBSERIAL DE WONDERWARE.

En esta sección se realizarán las comparaciones entre modo servidor de datos MODBUS RTU desarrollado y el servidor de datos comercial disponible en la plataforma Wonderware conocida como DASMBSerial [37], el cuadro comparativo se muestra en la tabla 3.3.1.

Tabla 3.3.1. Comparación entre MB Serial DA Server y el servidor de datos MODBUS RTU desarrollado

Comparativa	MB Serial DA Server	Servidor de datos MODBUS RTU desarrollado
Sistema operativo	Corre sólo en el sistema operativo Windows	Puede funcionar en cualquier sistema operativo
Configuración de dispositivos	Puede configurar varios dispositivos a través de varios puertos seriales.	Permite configurar un dispositivo por medio de un puerto serial
Parámetros de configuración de dispositivos	Puerto COM, paridad, bits de parada, bits de datos, velocidad de transmisión, tiempo de actualización de datos y tiempo de espera de respuesta	Puerto COM, paridad, bits de parada, bits de datos, velocidad de transmisión y tiempo de muestreo.
Compatibilidad de dispositivos	Controladores de la familia MODICON de manera predeterminada, y compatibilidad con dispositivos MODBUS RTU de 4, 5 y 6 dígitos.	Compatibilidad con cualquier dispositivo que maneje el protocolo MODBUS RTU.
Compatibilidad con programas externos	Programas de Windows que trabajen como cliente DDE, SuiteLink y OPC	Compatibilidad con programas que puedan integrarse a una base de datos
Compatibilidad con InTouch	Trabaja con Intouch de manera predeterminada	Se puede comunicar con Intouch mediante un conector ODBC
Licencia	Licencia de paga	Libre acceso
Permite ejecutar y detener el servidor	Si	Si
Tipos de datos que soportan	Bool, enteros, flotantes y de tipo String	Bool, enteros, flotantes.
Ingreso y eliminación de registros	Mediante la pestaña device ítem	Permite ingresar y borrar ítems en la pantalla principal mediante el botón agregar ítem y borrar ítem

Configuración de grupos de dispositivos	Permite configuración de grupos de dispositivos mediante diferentes redes de comunicación serial	Permite configurar un dispositivo.
Configuración del tamaño de los registros de memoria	Si permite ingresar la cantidad de registros coils, contact, registros input y holding	No permite configurar la cantidad de registros.
Configuración del orden de bits	De mayor a menor o de menor a mayor	Mediante el Endianness, big endian o little endian.
Conexión con base de datos	Si	Si
Configuración de la base de datos	No	Si

Cabe destacar que el software comercial dispone de mejores prestaciones, pero el objetivo del proyecto era desarrollar un servidor de datos para el protocolo MODBUS RTU para lectura y escritura de datos booleanos, enteros y flotantes, mismas que puedan conectarse con dispositivos esclavos mediante comunicación serial RS 232/RS-485, a su vez que almacene estos datos recopilados en una base de datos de MySQL, todo esto usando software libre, para su posterior integración con interfaces HMI que puedan interactuar con bases de datos.

Una vez comprobado el correcto funcionamiento del servidor de datos y haber sido comparado con un servidor comercial, es necesario realizar un análisis de costos del software desarrollado, que se muestra en la tabla 3.3.2.

Tabla 3.3.2. Costos del servidor de datos MODBUS RTU desarrollado.

Detalle	Software	Costo	Observaciones
Servidor de datos MODBUS RTU	JAVA con IDE Eclipse	$ 0	Licencia libre
Base de datos	MySQL 8.0	$ 0	Licencia libre
HMI	Aplicación que soporte conexión con MySQL	-	El valor dependerá del software que se utilice para generar HMIs
Desarrollo del programa		$2400	Horas de programación ($5/Hora)
Instalación		$60	Instalación y configuración de los programas para correr el software desarrollado

Como se puede observar en la tabla 3.3.2, el mayor costo se da en el desarrollo de la programación del servidor, el costo que le sigue es el de instalación del software en el equipo del usuario, cabe destacar que en el caso de comercializar el programa realizado, el costo del software sería mucho menor en comparación con el costo de diferentes

servidores comerciales vistos en el primer capítulo, el valor del software desarrollado es accesible. Además al tener el código fuente del servidor desarrollado, éste puede ser mejorado continuamente dependiendo de nuevas necesidades que requieran los potenciales clientes.

4. CONCLUSIONES Y RECOMENDACIONES

En base a las pruebas de funcionamiento realizadas al servidor de datos desarrollado, se han llegado a las siguientes conclusiones y recomendaciones.

4.1. CONCLUSIONES

Se desarrolló un servidor de datos industrial para la comunicación industrial con dispositivos que trabajen bajo el protocolo MODBUS RTU, mismo que se encarga de traer la información de los dispositivos esclavos y almacenarlos en una base de datos para un posterior monitoreo en un HMI como es el caso de Wonderware Intouch, utilizando para este proyecto software libre.

Se estudió la normativa MODBUS RTU para el desarrollo de los 8 principales códigos de función que permiten la lectura y escritura de los 4 tipos de registros modbus existentes en dispositivos industriales que trabajan bajo este protocolo, teniendo dos modos de funcionamiento, el primero que se encarga de realizar comunicación directa con un dispositivo esclavo y permite probar los 8 códigos de función desarrollados, el segundo modo de funcionamiento se encarga de realizar una comunicación continua con dispositivos esclavos para extraer la información de un proceso y ésta sea almacenada en una base de datos, cabe destacar que para este segundo modo de funcionamiento no fue necesario utilizar el código de función 15 el cual se encargaba de escribir múltiples coils.

El desarrollo del servidor de datos en JAVA permite una independencia de la plataforma de trabajo, dado que JAVA está desarrollado para trabajar libremente en cualquier sistema operativo sin necesidad de modificar el código, lo mismo pasa al usar MySQL, ésta también trabaja de manera independiente al sistema operativo, teniendo la ventaja que existen cada vez más programas que pueden interactuar con bases de datos, permitiendo que ésta pueda ser integrada en un sistema de control supervisorio o SCADA.

El desarrollo de este proyecto fue pensado para su implementación en PYMES, dado que en la actualidad existen servidores de datos comerciales con costos elevados, este proyecto es una opción más accesible que permitirá a dichas empresas poder empezar mejoras de procesos mediante la automatización.

Las interfaces desarrolladas en el presente proyecto están divididas en áreas acorde a su funcionalidad, lo cual permite que las ventanas sean más intuitivas al momento de usarlo

y probar su funcionamiento, en el caso del modo maestro MOBDUS RTU los resultados de las solicitudes se muestran en la misma ventana, mientras que para el modo de funcionamiento como servidor, el monitoreo de variables se puede observar en la misma ventana y también en la base de datos en tiempo real, teniendo de manera adicional un botón que permite escribir variables del proceso sin necesidad de tener conectado un HMI.

Mediante la realización de diferentes pruebas tanto en el software simulador de MODBUS RTU, así como en el PLC Micrologix ML1100, se pudo comprobar el correcto funcionamiento tanto del modo maestro como del modo servidor, convirtiéndole en una herramienta que puede ser integrada en PYMES que tengan pequeños procesos industriales y que busquen automatización a un costo relativamente bajo.

Este proyecto puede ser mejorado dependiendo de las nuevas necesidades que se puedan presentar, dado que se tiene el código fuente del servidor desarrollado, ésta es una gran ventaja, ya que puede ser adaptada a las necesidades propias de cada PYME.

4.2. RECOMENDACIONES

Dada la ventaja que presenta el desarrollo de programas en software libre, se recomienda el uso de este tipo de programas para el desarrollo de servidores de datos para otros protocolos industriales tales como DeviceNet, Controlnet, Profibus y Profinet, protocolos altamente usados en la actualidad por una gran cantidad de industrias.

Se recomienda el desarrollo de un software que permita generar pantallas HMI mediante el uso de software libre, la cual pueda integrarse con servidores ya desarrollados como el realizado en este trabajo.

Se recomienda implementar un plan piloto para el uso de los servidores de datos industriales desarrollados en la institución dentro del laboratorio de redes industriales con la finalidad de generar mayor interés en los nuevos estudiantes en el desarrollo de servidores industriales con el uso de software libre.

Se recomienda iniciar un plan piloto que permita a PYMES poder acceder a este tipo de servidores con el fin de dar a conocer su funcionalidad, ventajas y accesibilidad económica frente a servidores comerciales.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] KEPware, “OPC Drivers Bundles: modbus suite”. [En línea]. Disponible en: https://www.kepserverexopc.com/modbus-suite-drivers-bundles/# [Accedido:15-dic-2021]

[2] KEPware, “Modbus suite”. [En línea]. Disponible en: https://www.kepware.com/en-us/products/kepserverex/suites/modbus-suite/ [Accedido:15-dic-2021]

[3] KEPware, “KEPServerEX demo download”. [En línea]. Disponible en: https://n9.cl/xalyo [Accedido: 15-Dic-2021]

[4] MasterOPC, “modbus OPC server”, 2021. [En línea]. Disponible en: https://opc-server.com/catalog/modbusopc/modbus_opc/ [Accedido: 15-Dic-2021]

[5] “ModbusSerial DAServer User’ guide”, Wonderware, ver 2.5, 2006 [En línea]. Disponible en: https://cdn.logic-control.com/media/DASMBSerial.pdf. [Accedido:15-Dic-2021]

[6] “Cotización licencia InTouch” [En línea]. Disponible en: https://n9.cl/es/s/4rb2z [Accedido:15-Dic-2021]

[7] Guía protocolo modbus, ed. 1.3, ASTRALPOOL, Alicante, España. [En línea] Disponible en: https://gestor-doc-s3.s3.eu-west-1.amazonaws.com/documents/category/man_60363_modbus_es-131499.pdf. [Accedido:1-Dic-2021].

[8] Schneider Electric, “Códigos de excepción MODBUS”, 2020. [en línea]. Disponible en: https://product-help.schneider-electric.com/ED/ES_Power/PP-HJL_Modbus_Guide/EDMS/0611IB1303/0611IB13xx/NSX_MB_Modbus_Protocol/NSX_MB_Modbus_Protocol-5.htm [Accedido:1-Dic-2021].

[9] Oracle Corporation,” JAVA Platform Standard Documentation”, ed. 7, [en línea]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html [Accedido:1-Dic-2021].

[10] Oracle Corporation,” JAVA Platform Standard Documentation”, ed. 7, [En línea]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html [Accedido:1-Dic-2021].

[11] Oracle Corporation," JAVA Platform Standard Documentation", [en línea]. Disponible en:https://docs.oracle.com/cd/E17802_01/products/products/javacomm/reference/api/javax/comm/CommPortIdentifier.html [Accedido:1-Dic-2021].

[12] Oracle Corporation," JAVA Platform Standard Documentation", [en línea]. Disponible en: https://docs.oracle.com/cd/E17802_01/products/products/javacomm/reference/api/javax/comm/SerialPort.html#setSerialPortParams(int,%20int,%20int,%20int) [Accedido:1-Dic-2021].

[13] Oracle Corporation, "Connector/J 5.1 Developer Guide", 2021 https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-versions.html

[14] "Conectar Java a una base de datos MySQL", [en línea]. Disponible en: https://www.it-swarm-es.com/es/java/conectar-java-una-base-de-datos-mysql/970045845/ [Accedido:1-Dic-2021].

[15] "Guía de laboratorio #1 Programación Orientada a Objetos JDBC", Universidad Don Bosco, San Salvador, El Salvador, [En línea]. Disponible en: https://www.udb.edu.sv/udb_files/recursos_guias/informatica-tecnologico/programacion-orientada-a-objetos-(fet)/2020/i/guia-4.pdf [Accedido:1-Dic-2021].

[16] "Ejemplos java y C/Linux", 2007 [En línea]. Disponible en: http://www.chuidiang.org/java/mysql/mysql-java-basico.php [Accedido:1-Dic-2021].

[17]"Stament Interface", Javatpoint. [En línea]. Disponible en: https://www.javatpoint.com/Statement-interface [Accedido:1-Dic-2021].

[18] "MySQL 5.0 Reference Manual", Oracle corporation, Redwood City, CA, USA, 2011. [en línea]. Disponible en: https://downloads.mysql.com/docs/refman-5.0-es.pdf [Accedido:1-Dic-2021].

[19] "La interfaz resultSet", [en línea]. Disponible en: http://eolo.cps.unizar.es/java/JDBC/resultset.html [Accedido:1-Dic-2021].

[20] TechTarget, “Open Database Connectivity (ODBC)”, 2021, [En línea]. Disponible en: https://www.computerweekly.com/es/definicion/Open-Database-Connectivity-ODBC [Accedido:1-Dic-2021].

[21]Electronic Team, “Virtual Serial Port Driver”, 2021, [En línea]. Disponible en: https://www.eltima.com/es/products/vspdxp/ [Accedido:1-Dic-2021].

[22] “MODBUS”, [En línea]. Disponible en: https://aquiro.weebly.com/modbus.html [Accedido:1-Dic-2021].

[23] Opiron electronics, “¿Que es MODBUS?”,2017. [En línea]. Disponible en: https://www.opiron.com/que-es-modbus/ [Accedido:1-Dic-2021].

[24] National Instruments, “Información detallada sobre el protocolo Modbus”, 2021. [En línea] https://www.ni.com/es-cr/innovations/white-papers/14/the-modbus-protocol-in-depth.html [Accedido:1-Dic-2021].

[25] Modbus application protocol specification, ver. 1.1, modbus organization, 2012. [en línea]. Disponible en: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[26] Geeksforgeeks, “IEEE Standard 754 floating point numbers”, 2020. [En línea]. Disponible en: https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/ [Accedido:1-Dic-2021].

[27] Ordoñez, S. “Esquemas iterativos en Paralelo con OpenMP”. [En línea]. Disponible en: https://www.researchgate.net/figure/Figura-23-Modelo-para-representacion-de-numeros-en-punto-flotante-IEEE-754_fig1_317225784 [Accedido:1-Dic-2021].

[28] Programador clic, “Comprobación de CRC en el protocolo MODBUS”. [En línea]. Disponible en: https://programmerclick.com/article/80991954676/ [Accedido:1-Dic-2021].

[29]Gonzales, C, “¿Cuál es la historia de Java?”, 2017. [En línea]. Disponible en: https://www.emagister.com.co/blog/cual-es-la-historia-de-java/ [Accedido:1-Dic-2021].

[30] Walton, Alex, “Breve historia de Java: características y aplicaciones”, 2020, [en línea]. Disponible en: https://javadesdecero.es/fundamentos/breve-historia-caracteristicas-y-aplicaciones/ [Accedido:1-Dic-2021]

[31] Manual Web, “Historia del lenguaje Java”, 2021. [en línea]. Disponible en: https://www.manualweb.net/java/historia-java/ [Accedido:1-Dic-2021]

[32] GameDevTraum, “Diferencia entre clase y objeto en POO”, 2021. [En línea]. Disponible en: https://gamedevtraum.com/es/programacion-informatica/programacion-orientada-a-objetos/diferencia-entre-clase-y-objeto/ [Accedido:1-Dic-2021]

[33] desarrolloweb.com, “Polimorfismo en Programación Orientada a Objetos”, 2021. [En línea]. Disponible en: https://desarrolloweb.com/articulos/polimorfismo-programacion-orientada-objetos-concepto.html [Accedido:1-Dic-2021]

[34] Vergara, A. “Netbeans vs Eclipse, ¿Cuál elegir?”, 2015. [En línea]. Disponible en: https://www.facilcloud.com/noticias/netbeans-o-eclipse-cual-elegir/ [Accedido:1-Dic-2021]

[35] Wonderware Iberia, “Como se convirtió Wonderware en una de las compañías de software con más rápido crecimiento”, 2021. [En línea]. Disponible en: https://www.wonderware.es/nosotros/wonderware/historia/ [Accedido:1-Dic-2021]

[36] Tic Portal, “MySQL”, 2019. [En línea]. Disponible en: https://www.ticportal.es/glosario-tic/mysql [Accedido:1-Dic-2021]

[37] Wonderware, “Wonderware ModbusSerial DAServer 2.0 Readme”, 2005. [En línea]. Disponible en: http://www.maha-net.co.kr/m-file/dasmbserial20ReadMe.html [Accedido:1-Dic-2021]

[38] L. Martínez. “Comunicaciones industriales”, 1st. Ed, Ciudad de México, Alfaomega, 2009.

[39] D. Aguirre. “Desarrollo de una herramienta computacional que contenga comunicación Modbus RTU y Modbus TCP para la implementación de sistemas de control

supervisorio y adquisición de datos a bajo costo", M.S tesis, Departamento de Automatización y Control Industrial, EPN, Quito, Ecuador, 2018.

[40] V. Maldonado, "Desarrollo de un servidor de datos industrial con protocolo Modbus TCP para los 8 códigos de función básicos", tesis pregrado, Departamento de Automatización y Control Industrial, EPN, Quito, Ecuador, 2021.

[41] Aula 21, "Qué es un sistema SCADA, para qué sirve y cómo funciona", 2005. [En línea]. Disponible en: https://www.cursosaula21.com/que-es-un-sistema-scada/. [Accedido:2-Dic-2021]

[42] Electronic Board, "' ¿Qué es SCADA?", [2022]. [En línea]. Disponible en: https://www.electronicboard.es/que-es-scada/ [Accedido:3-Dic-2021]

6. ANEXOS

ANEXO 1: MANUAL DE INSTALACIÓN.

ANEXO 2: MANUAL DE USUARIO.

ANEXO 1: MANUAL DE INSTALACIÓN

CONTENIDO

A.1. INTRODUCCIÓN

El presente manual es una guía de instalación de programas necesarios para la ejecución del servidor de datos MODBUS RTU, el cual requiere conectarse con una base de datos en MySQL, también requiere la instalación de librerías para la poder establecer una comunicación serial y la instalación del driver ODBC que permita a la base de datos poder interactuar con programas externos por ejemplo Intouch.

En las siguientes páginas se detalla el procedimiento de instalación de cada uno de los programas necesarios para el presente proyecto.

A.1. REQUISITOS PREVIOS

Para la instalación de Eclipse, MySQL Workbench y el conector ODBC debemos tener instalado los siguientes programas:

- Microsoft Visual C++ redistributable 2019 x64.
- Microsoft Visual C++ redistributable 2015 x64.

A.2. INSTALACION DE JAVA JRE 8

1. Descargar Java JRE 8 Update 191.
2. Dar clic en Instalar.

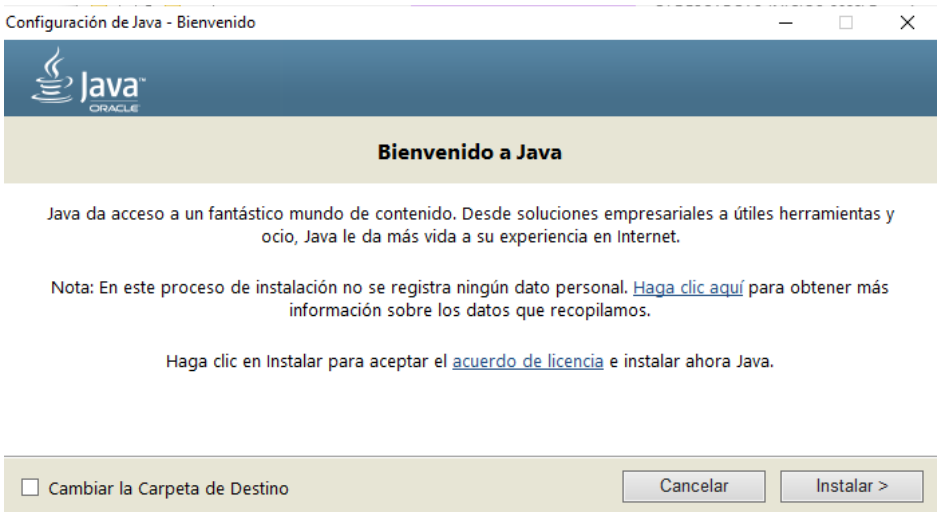

Figura A.2.1. Ventana de instalación de Java 8.

3. Dar clic en aceptar y esperar a que termine la instalación.

Figura A.2.2. Ventana del estado de instalación de Java 8.

Ahora se debe agregar la versión de JAVA a las variables de entorno, para lo cual se debe seguir el siguiente procedimiento:

1. Vamos al buscador y digitar “Variables de entorno” y dar clic en “Editar las variables de entorno del sistema”.

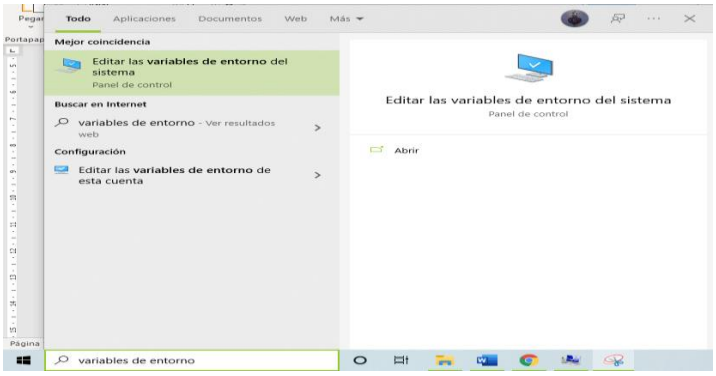

Figura A.2.3. Búsqueda de variables de entorno.

2. Ir a variables de sistema y dar clic en nuevo.

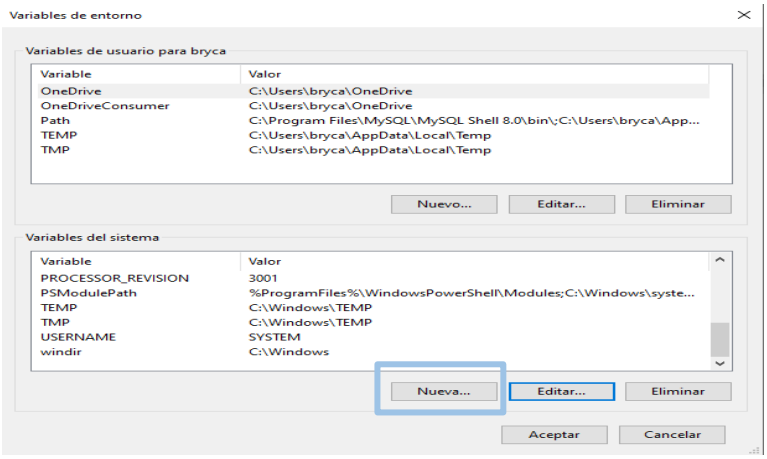

Figura A.2.4. Ventana de las variables de entorno del sistema.

3. Escribir en Nombre de variable de la variable “JAVAHOME”, y en valor de la variable seleccionar la carpeta donde fue instalado JAVA.

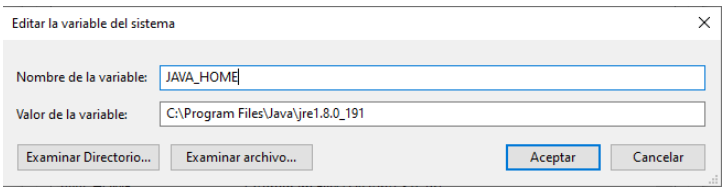

Figura A.2.5. Ventana de edición de variables de entorno.

4. Dentro de variables de sistema vamos a “Path” y dar clic en editar

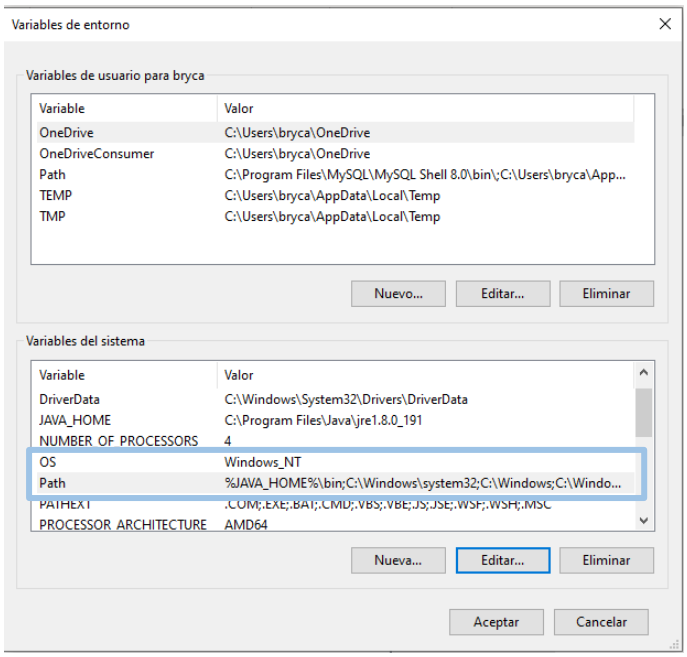

Figura A.2.6. Path de las variables de entorno.

5. Escribir dentro de Path “%JAVA_HOME%\bin” y dar clic en “Aceptar”.

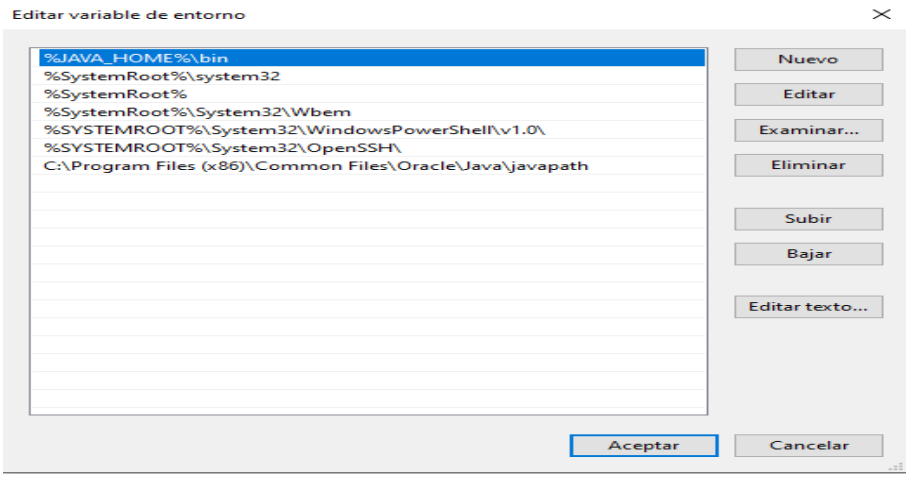

Figura A.2.7. Ventana de path de las variables del sistema.

A.3. AGREGAR LIBRERIAS SERIALES EN JAVA

En la carpeta librerías seriales tenemos 3 archivos que son:

-rxtxserial.dll

-rxtxParallel.dll

-RXTXcomm.jar

Se debe ir a la dirección de la carpeta donde se instaló java, vamos a la carpeta donde está instalado el JRE (Java Runtime Enviroment), dentro de esta carpeta tenemos la carpeta “bin”, aquí es donde pegamos los archivos “.dll”, como se observa en la figura.

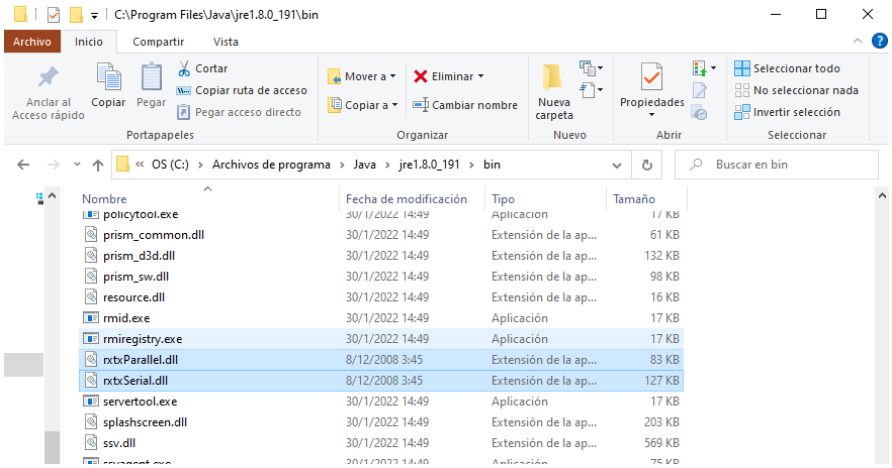

Figura A.3.1. Inclusión de librerías seriales.

El siguiente paso es ir a la carpeta donde está instalado el JRE, dentro de esta vamos a la carpeta “lib” y encontramos la carpeta “ext”, en esta carpeta pegamos el archivo RXTXcomm.jar

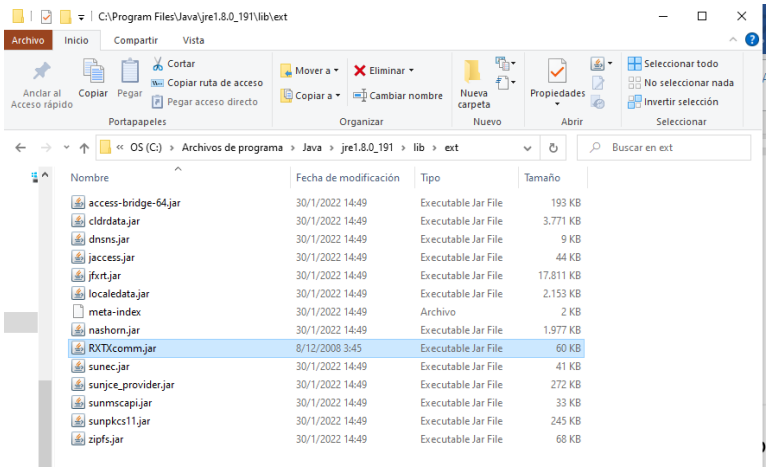

Figura A.3.2. Inclusión de librerías seriales.

A.4. CONFIGURAR EL COMPILADOR EN ECLIPSE

El programa desarrollado trabaja con la versión del JRE 8 Update 191, por lo que es necesario cambiar el compilador en eclipse para que el programa pueda correr sin problemas. A continuación, se mostrará el procedimiento para cambiar el compilador en eclipse.

1. Ir a la pestaña Windows, dar clic en la opción “preferences”.

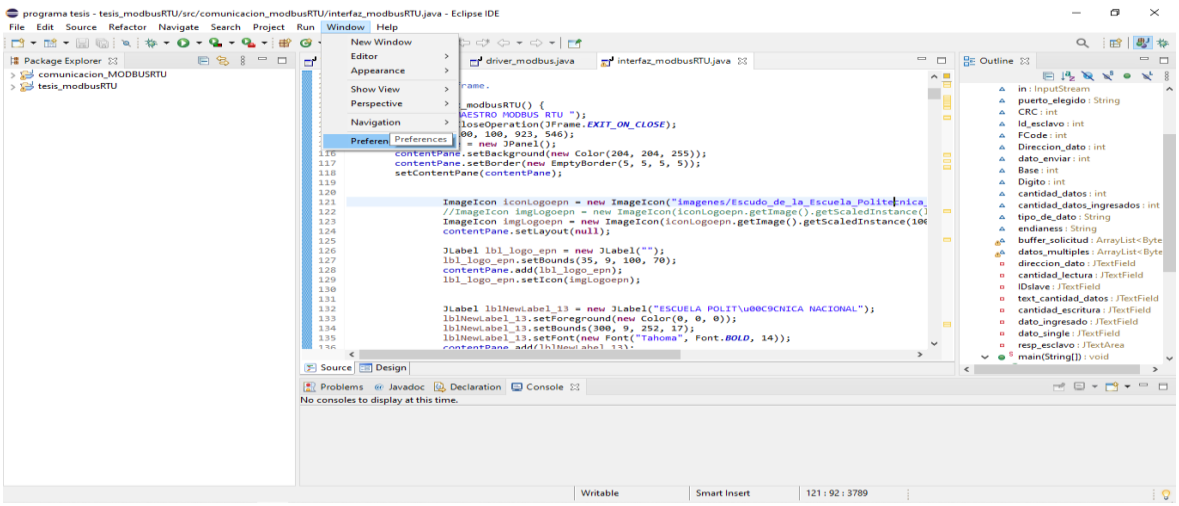

Figura A.4.1. Pantalla principal de Eclipse.

1. Dentro de la ventana, ir a Java, luego dar clic en compiler, en el apartado “Compiler compliance level” seleccionar la opción 1.8, por último, dar clic en aplicar y cerrar.

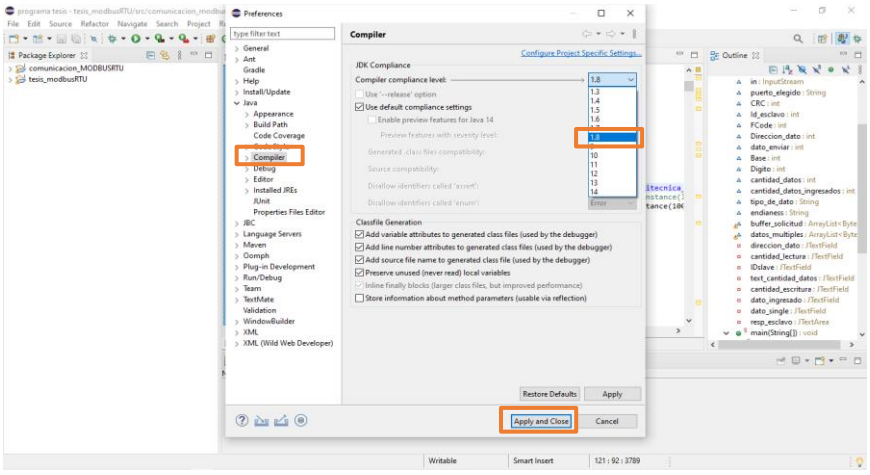

Figura A.4.1. Menú para cambiar el compilador de Eclipse.

A.5. INSTALACIÓN DE MYSQL WORKBENCH

El programa MySQL Workbench es parte importante para la implementación del servidor MODBUS RTU, ya que este programa será el que gestionará la base de datos del servidor MODBUS RTU desarrollado. A continuación, se detalla el procedimiento para su correcta instalación:

1. Ir a la página oficial de MySQL mediante el siguiente enlace: https://dev.mysql.com/downloads/windows/installer/8.0.html

2. Nos aparecerá la siguiente ventana, en la cual debemos ir a la pestaña “Archivos”.

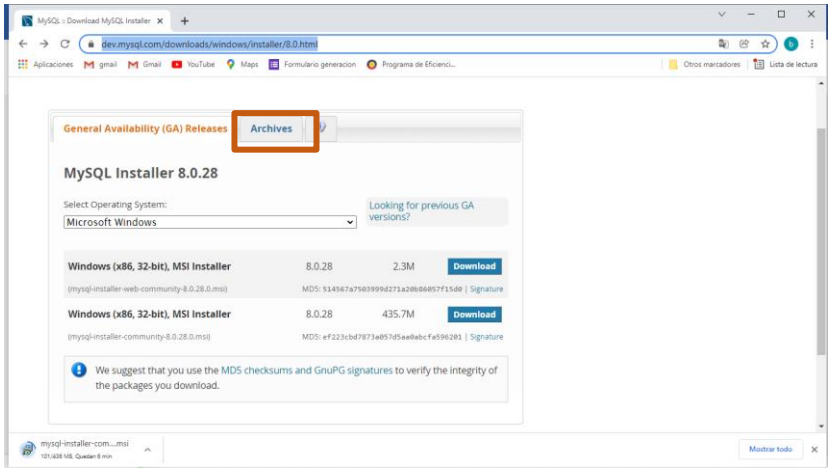

Figura A.5.1. Página de descarga de MySQL.

3. Seleccionamos la versión de MySQL Workbench que se desea instalar, en este caso, se instalará la versión 8.0.25

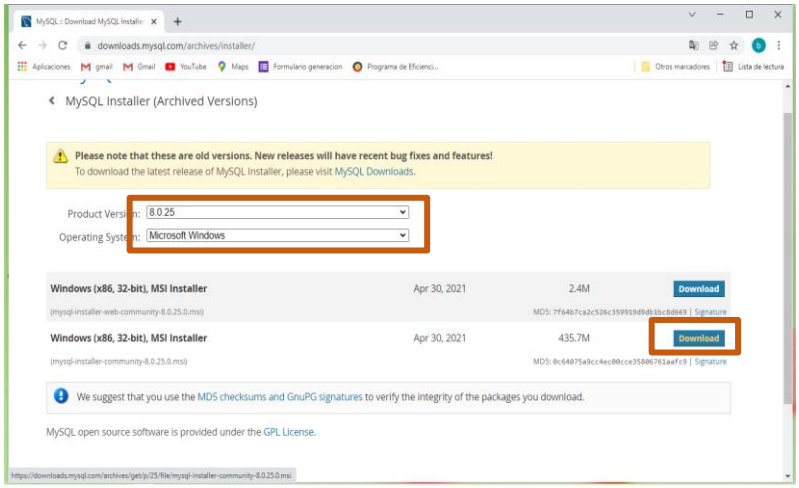

Figura A.5.2. Selección de la versión de descarga de MySQL.

4. Dar clic en la opción “no gracias, inicie la descarga”.

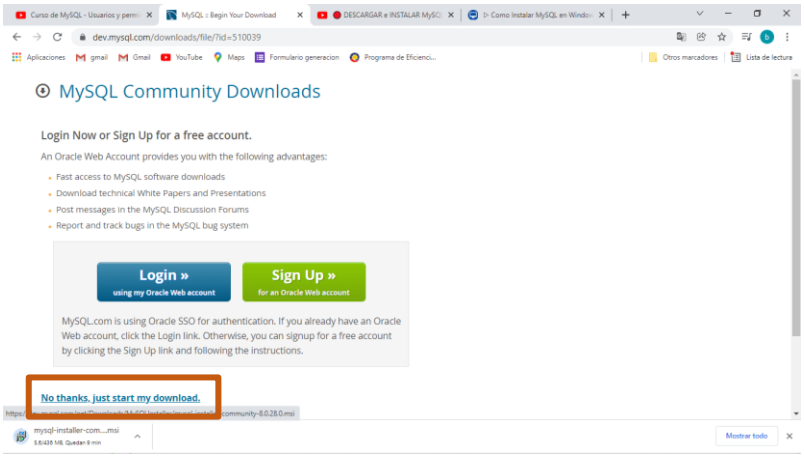

Figura A.5.3. Selección de la opción que comienza la descarga de MySQL.

5. Una vez descargado, ejecutar el archivo, esto desplegará una ventana que nos permite elegir entre el tipo de instalación, para lo cual se debe seleccionar la opción “custom” ya que no deseamos instalar todos los paquetes de MySQL.

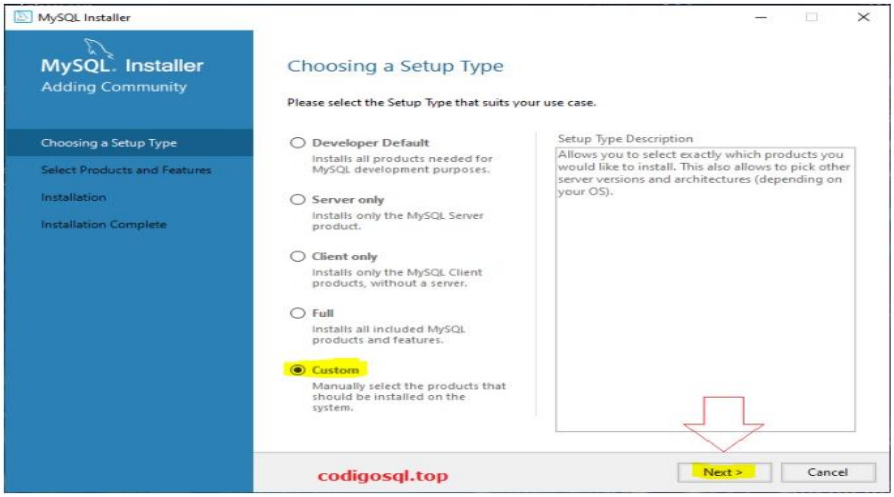

Figura A.5.4. Pantalla de instalación de MySQL.

6. Seleccionar todos los elementos a instalar, para este caso se instalará MySQL Server, MySQL Workbench, MySQL Shell.

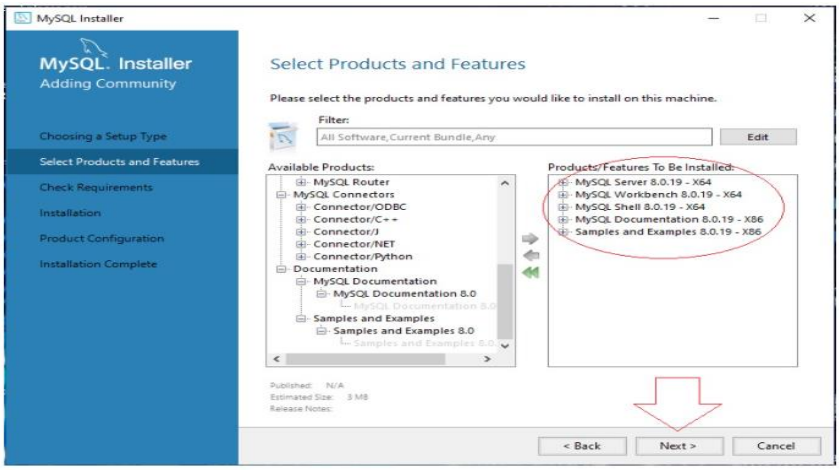

Figura A.5.5. Selección de los productos de MySQL a instalar.

7. En caso de tener requerimientos del sistema, le damos clic en ejecutar para instalar estos requerimientos.

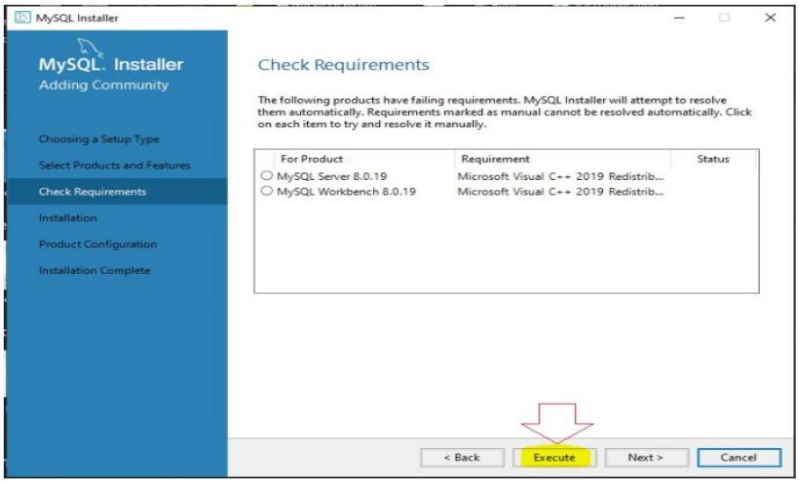

Figura A.5.6. Visualización de requerimientos para la instalación de MySQL.

8. Después de instalarse los complementos, le damos “next” y nos aparecerá la pantalla con la lista de herramientas seleccionadas anteriormente para instalar.

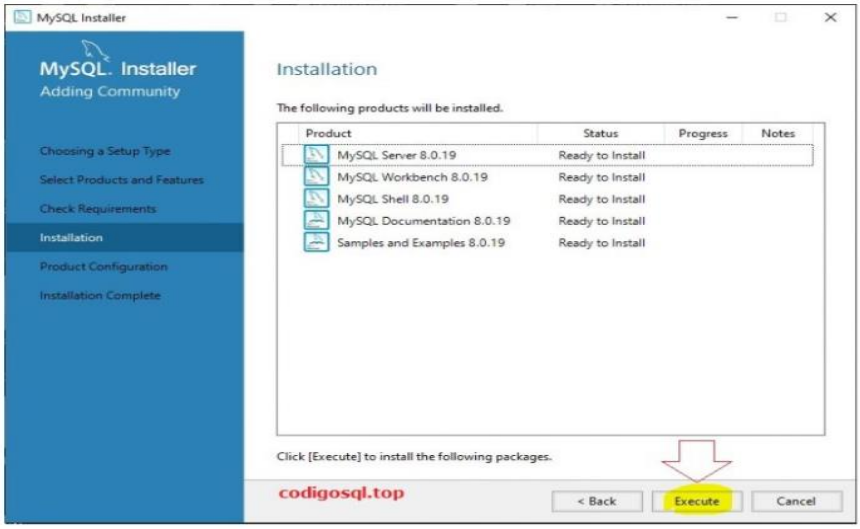

Figura A.5.7. Programas a Instalar de MySQL.

9. Le damos clic en ejecutar y esperamos que termine la instalación, una vez instalado le damos “next”. Luego seleccionamos “Standalone MySQL Server” y le damos “next”

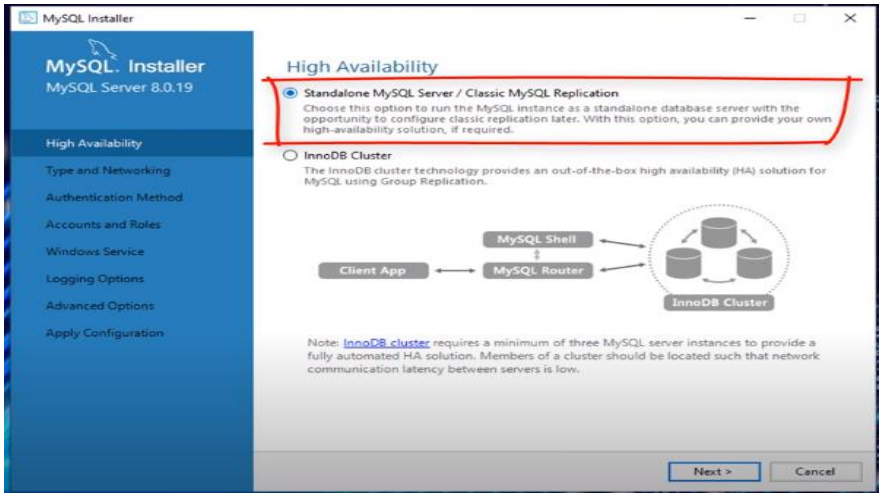

Figura A.5.8. Asistente de instalación de MySQL.

10. Seleccionamos “Development Computer”, mantenemos el puerto en 3306 y le damos “next”.

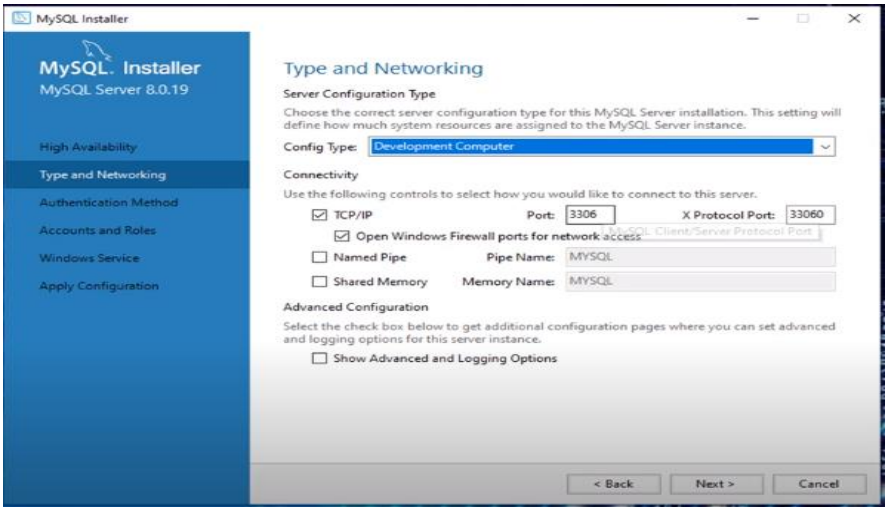

Figura A.5.9. Configuración del server de MySQL.

11. Elegimos el método de autenticación por medio de contraseña y le damos “next”.

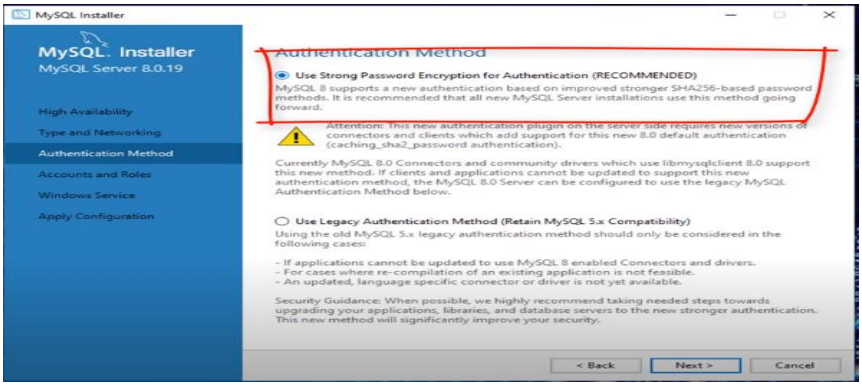

Figura A.5.10. Método de autenticación de la contraseña de MySQL.

12. Ingresamos la contraseña que deseamos que tenga nuestro usuario local llamado “root” y le damos “next”.

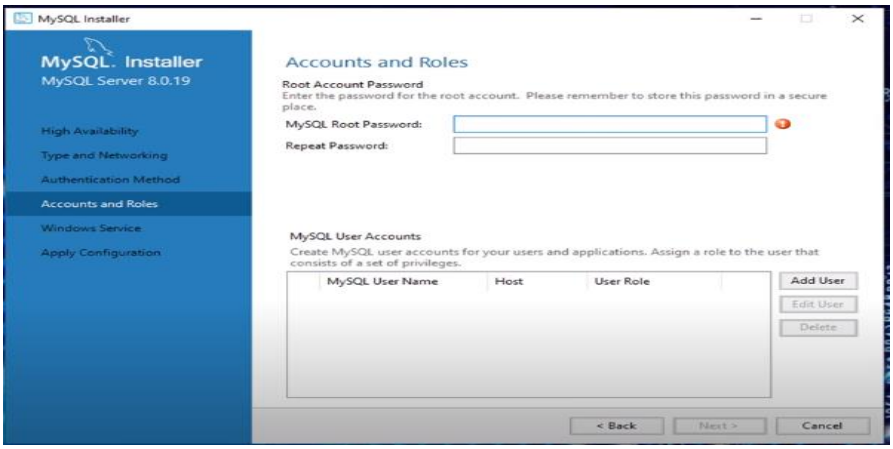

Figura A.5.11. Ingreso de la contraseña para el usuario root de MySQL.

13. Dar nombre a los servicios de Windows, posteriormente le damos clic en “next”

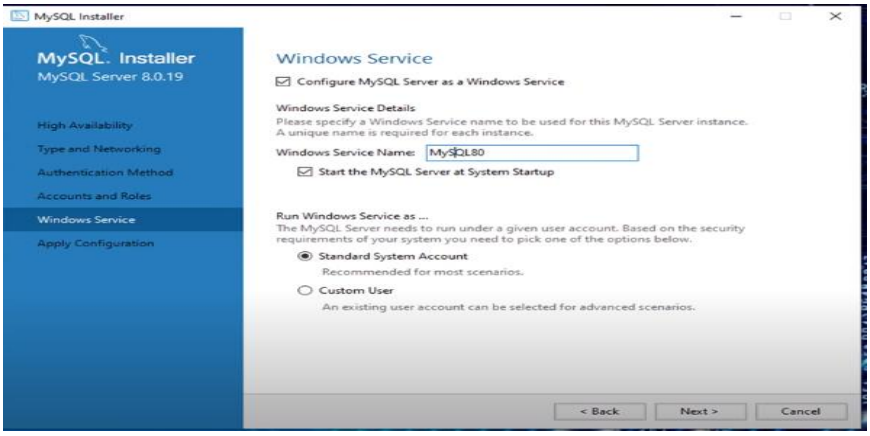

Figura A.5.12. Configuración de los servicios de Windows en MySQL.

14. Nos aparecerá la siguiente pantalla, dar clic en ejecutar y esperar a que termine de realizarse la configuración.

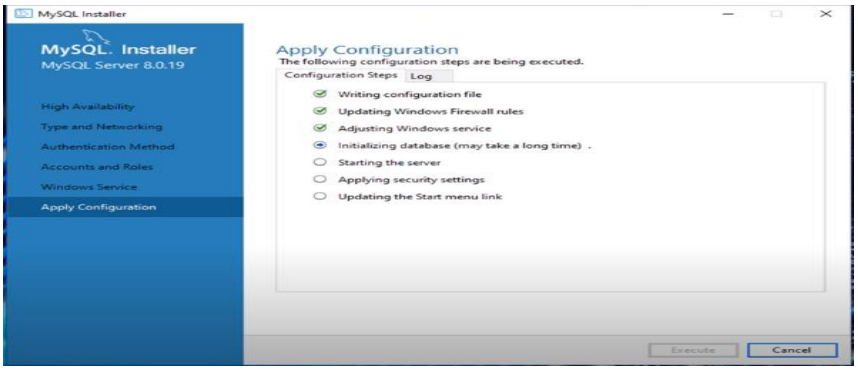

Figura A.5.13. Verificación de la aplicación de configuraciones de MySQL.

15. A continuación, dar clic en “next”, hasta que aparezca la siguiente pantalla, en esta pantalla nos pedirá la contraseña creada para root, ingresar la contraseña para comprobar que la conexión se realizó correctamente.

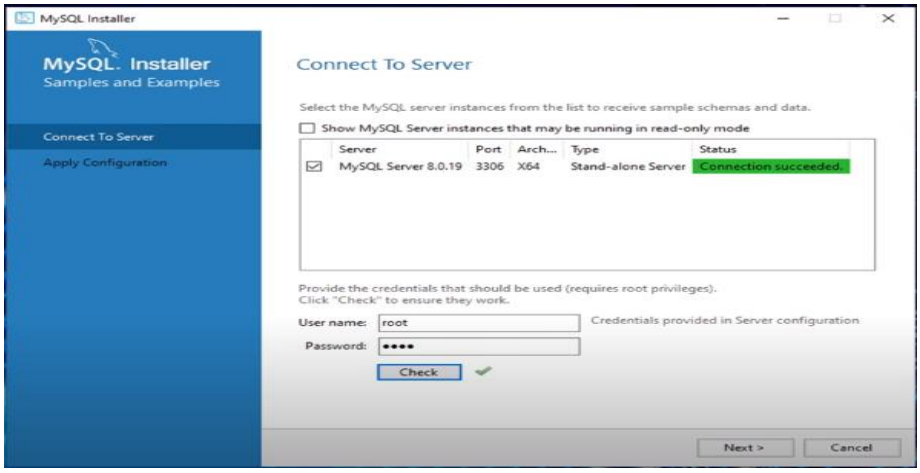

Figura A.5.14. Verificación del usuario y contraseña de MySQL.

16. Dar “next” hasta terminar la instalación.

A.6. INSTALACIÓN DEL CONECTOR ODBC 8.0

El conector ODBC 8.0, permite a programas externos poder acceder a una base de datos en MySQL, puntualmente para poder conectar la base de datos con el software de visualización de HMIs llamado Intouch, a continuación, se muestra el procedimiento para su instalación.

1. Dirigirse a la página oficial de descargas de MySQL, o copiar el siguiente enlace: https://dev.mysql.com/downloads/connector/odbc/

2. Dentro de esta ventana dirigirnos a la opción “Ir a la página de descarga”

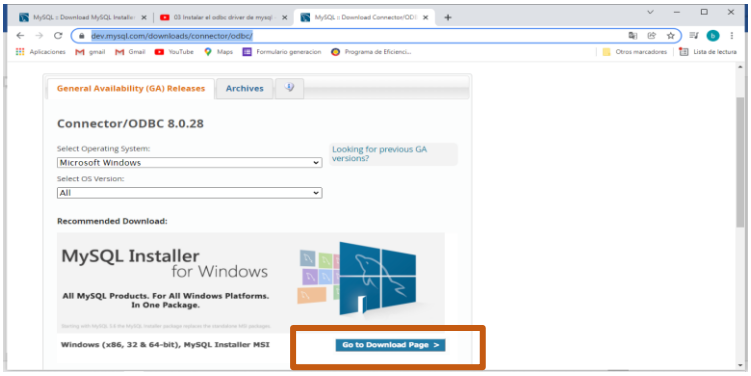

Figura A.6.1. Página principal de descarga del conector ODBC.

3. Ir a la opción “Archivos”

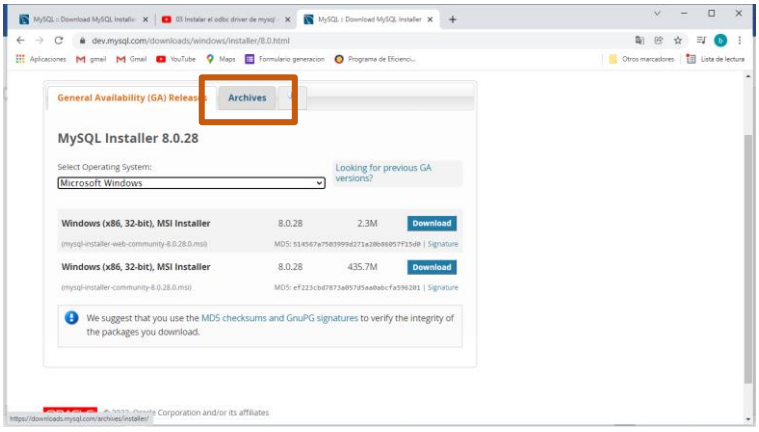

Figura A.6.2. Selección de la versión del conector ODBC.

4. Seleccionar la versión 8.0.16 para el sistema operativo de Windows, luego descargar el instalador.

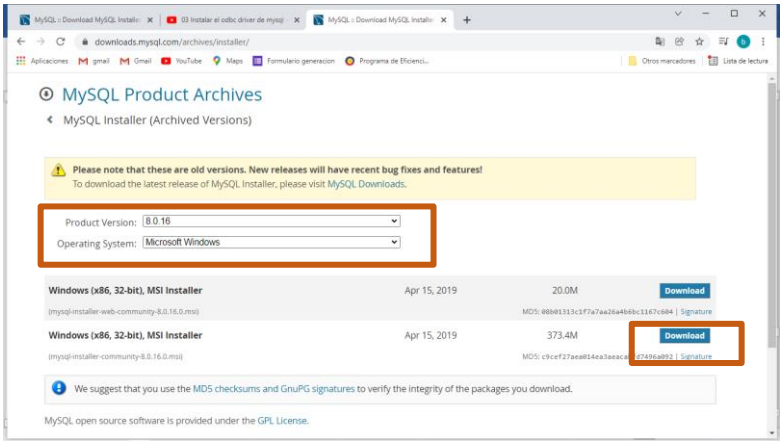

Figura A.6.3. Descarga del conector ODBC.

5. Ejecutamos el archivo descargado y dar clic en “next”.

Figura A.6.4. Ventana de instalación del conector ODBC.

6. Aceptar los términos de la licencia, y dar clic en “next”.

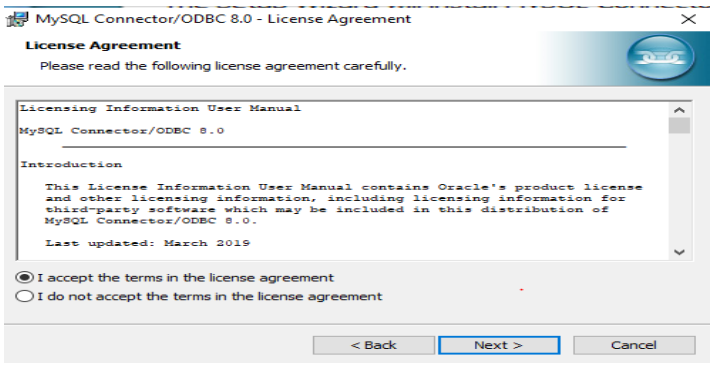

Figura A.6.5. Aceptación de los términos de la licencia del conector ODBC.

7. Seleccionar la opción de instalación completa

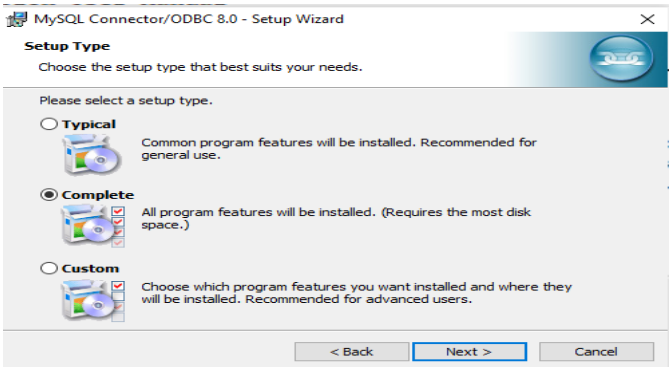

Figura A.6.6. Selección del tipo de instalación del conector ODBC.

8. Dar clic en Instalar.

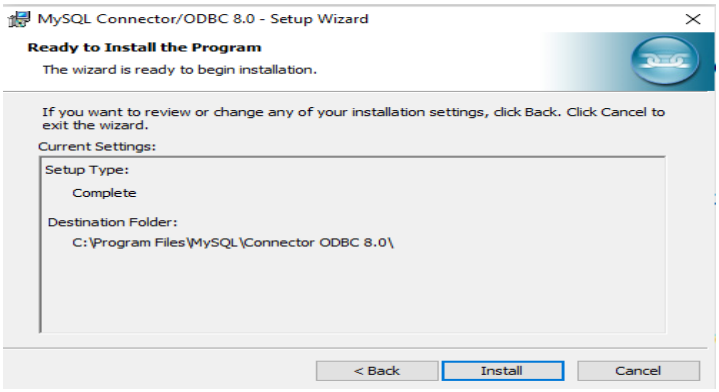

Figura A.6.7. Instalación del conector ODBC.

ANEXO 2: MANUAL DE USUARIO

CONTENIDO

A2.1 INTRODUCCIÓN

El siguiente manual contiene información para el manejo del servidor de datos, el cual maneja el protocolo MODBUS RTU mediante la comunicación con dispositivos esclavos por medio de una red serial R2-232/RS-485, tiene la capacidad de conectarse con dispositivos que posean direcciones de 4, 5 y 6 dígitos, manejen base 0 o 1.

El programa dispone de dos modos de funcionamiento, que son el modo maestro MODBUS RTU el cual permite comunicarnos con un dispositivo esclavo y poder leer o escribir variables booleanas, enteras y flotantes mediante los 8 códigos básicos de función del protocolo por medio del envío individual de tramas de solicitud. El otro modo de funcionamiento es el modo servidor RTU que trabaja de modo continuo, el cual puede ser integrado en sistemas SCADA ya que posee comunicación con bases de datos en MySQL, esto permite que programas de HMI externos puedan interactuar con esta base de datos, como es el caso de Intouch.

A continuación, se detallará los requisitos previos para su uso, las configuraciones necesarias para el uso de cada uno de estos modos de funcionamiento.

A2.2 REQUISITOS PREVIOS

Para el funcionamiento de la ventana en modo servidor MODBUS RTU es necesario tener instalado los siguientes softwares:

- MySQL Workbench 8.0
- Conector ODBC 8.0

MySQL Workbench es el programa que administra la base de datos que donde se almacenarán las variables generadas en el modo servidor.

El conector ODBC (Open DataBase Connectivity) es un driver que permite la comunicación entre la base de datos y el software para generación de HMIs llamado Intouch.

Cabe destacar que no es necesario que estos dos programas estén instalados en la misma máquina, es decir, MySQL Workbench debe estar instalada en la máquina que se encargue de la administración de bases de datos, mientras que el conector ODBC debe estar instalada en la máquina donde se visualizará el HMI.

Se debe considerar también que las máquinas que administren la base de datos y el HMI deben estar conectados en la misma red, antes de ejecutar el programa se recomienda revisar que las máquinas que encuentren dentro de la misma red.

A2.3 CONFIGURACIONES NECESARIAS PARA LA EJECUCIÓN DEL PROGRAMA EN MODO SERVIDOR MODBUS RTU.

Para tener acceso a una base de datos en MySQL, es necesario contar con usuarios diferentes al usuario local (local host), dado que el usuario local puede causar que se produzcan varios errores durante la conexión entre la base de datos y la máquina que opera el HMI en Intouch, él o los usuarios creados para la administración de bases de datos en MySQL podrán ser usados tanto para la conexión de la base de datos con nuestro servidor MODBUS RTU y con la maquina donde se ejecutará el HMI.

Este mismo usuario es el que se configurará dentro de los parámetros del conector ODBC, el cual permitirá a la máquina donde se ejecutará el HMI poder conectarse con la base de datos.

A continuación, se mostrará cómo se debe crear usuarios en MySQL y como configurar los parámetros del conector ODBC.

A.3.1. Creación de usuarios en MySQL Workbench.

El usuario en MySQL es el que permite acceder a las bases de datos y otorga permisos para la interacción entre la base de datos y programas externos como es el caso del servidor de datos desarrollado en Java y el conector ODBC. A continuación, se muestra el procedimiento para la creación de usuarios en MySQL Workbench:

1. Abrir MySQL Workbench e ingresar con el usuario local root.

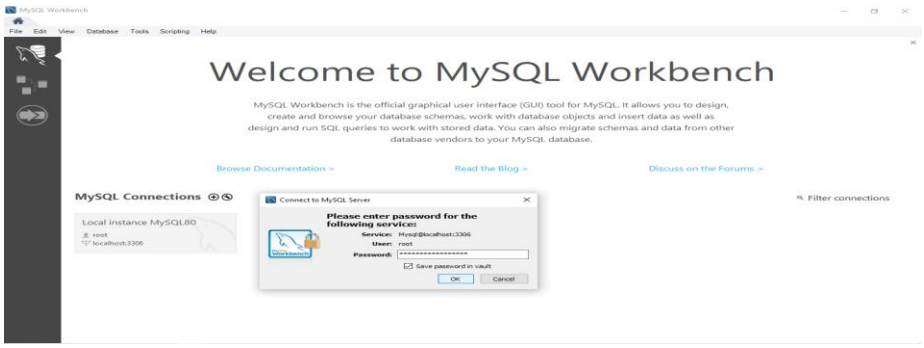

Figura A2.3.1. Ventana de inicio de MySQL Workbench.

2. Ir a la opción “User and privilegies”, luego dar clic a “Add account”.

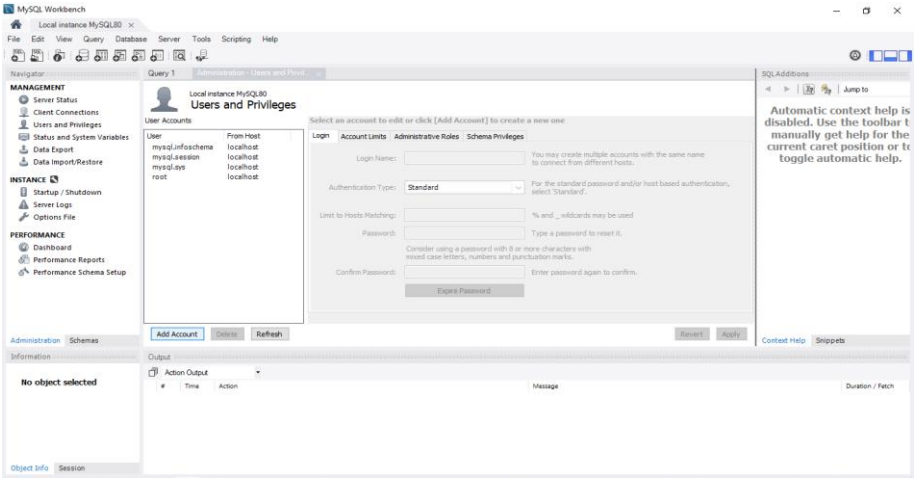

Figura A2.3.2. Añadir de usuarios.

3. En la pestaña “Login”, ingresar el nombre que se desea dar al usuario y la contraseña para el mismo. Luego damos clic en “Apply”.

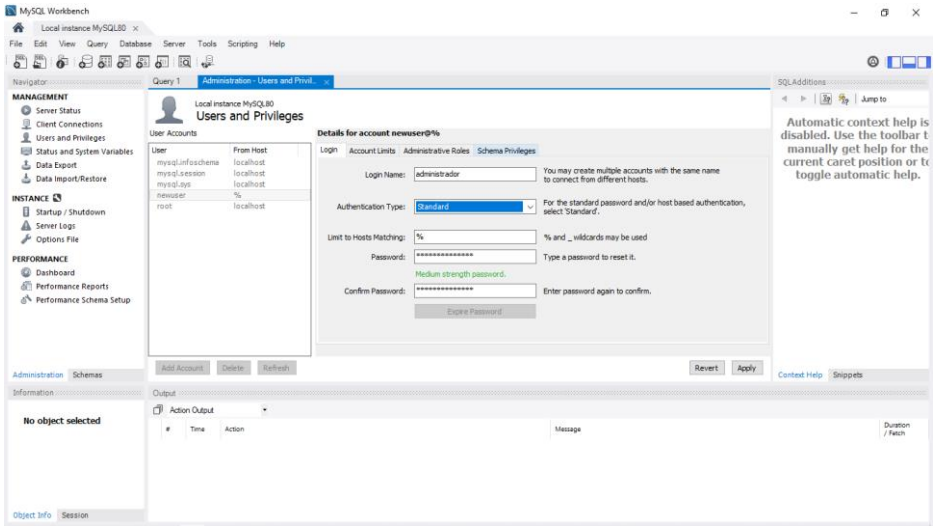

Figura A2.3.3. Nombre del usuario a crear.

4. Ir a la pestaña Schema Privileges.

5. Para agregar los privilegios al nuevo usuario, dar clic en “New Entry”.

6. Se desplegará una ventana, seleccionar “All Schemas” y dar clic en OK.

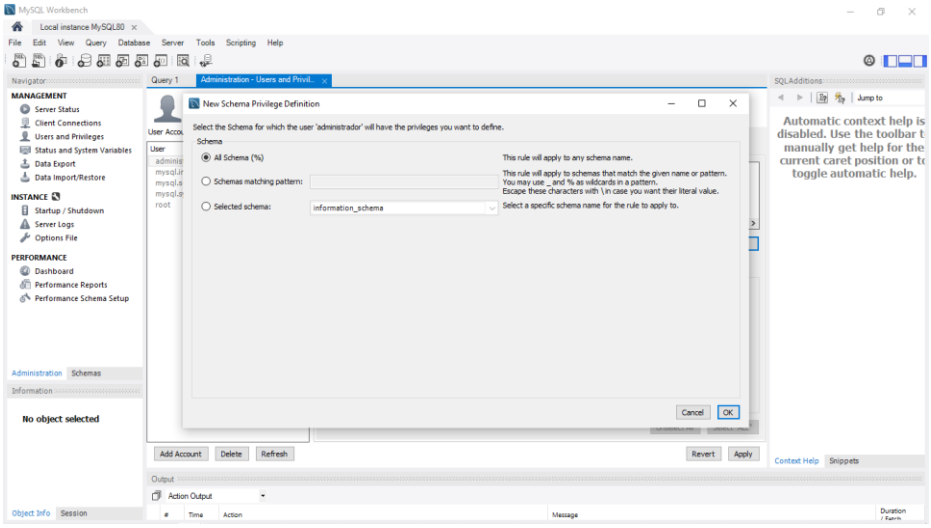

Figura A2.3.4. Otorgar privilegios.

7. Dar clic en “Select All”, para poder dar todos los privilegios al usuario creado, por último, dar clic en “Apply” y el usuario se habrá creado correctamente.

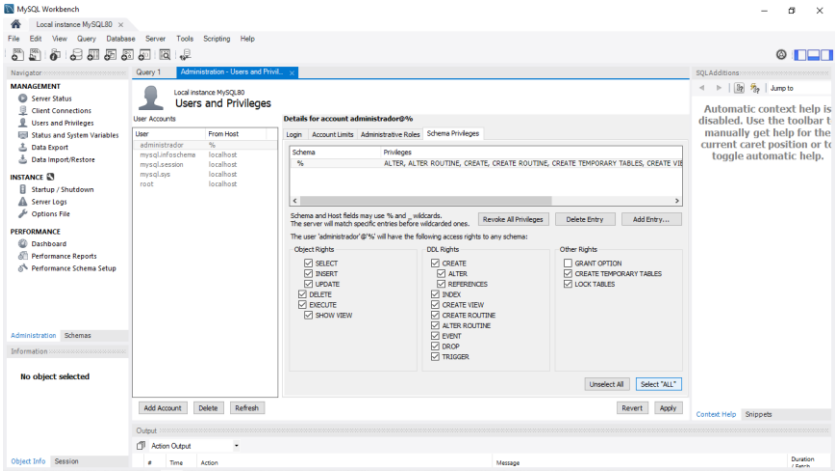

Figura A2.3.5. Selección de privilegios a asignar al usuario.

Para agregar una nueva conexión a la pantalla de inicio de MySQL Workbench debemos realizar el siguiente procedimiento.

1. Abrir MySQL Workbench, en la parte inferior en MySQL Connections dar clic en el símbolo “+”.

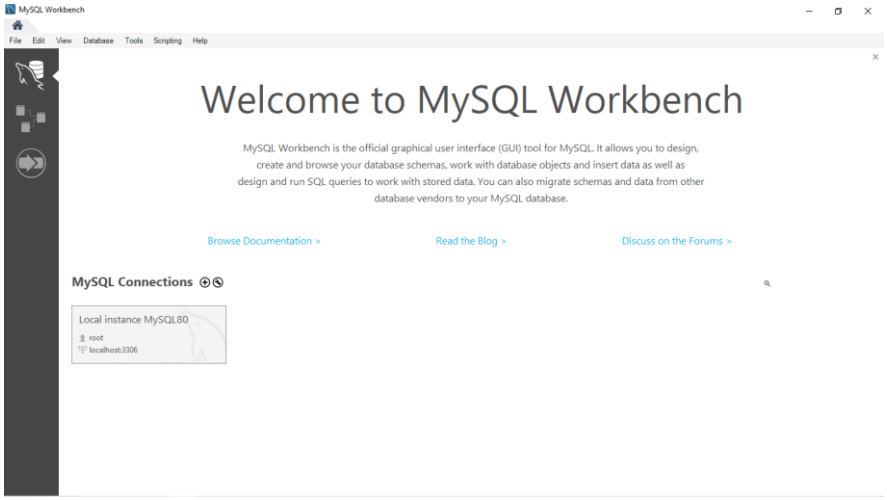

Figura A2.3.6. Ventana de inicio de MySQL Workbench.

2. se nos desplegara una ventana, aquí debemos ingresar el nombre que se desea dar a la conexión, el usuario creado y la dirección Ip de la computadora donde está corriendo nuestra base de datos.

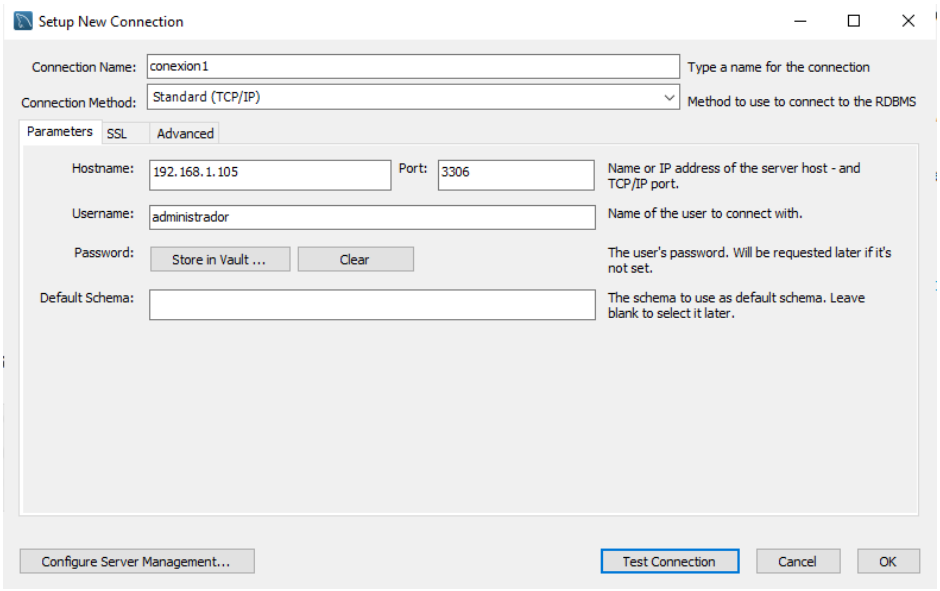

Figura A2.3.7. Ventana de creación de conexiones en MySQL Workbench.

3. dar clic en “Test Connection” para verificar que se haya realizado la conexión con MySQL.

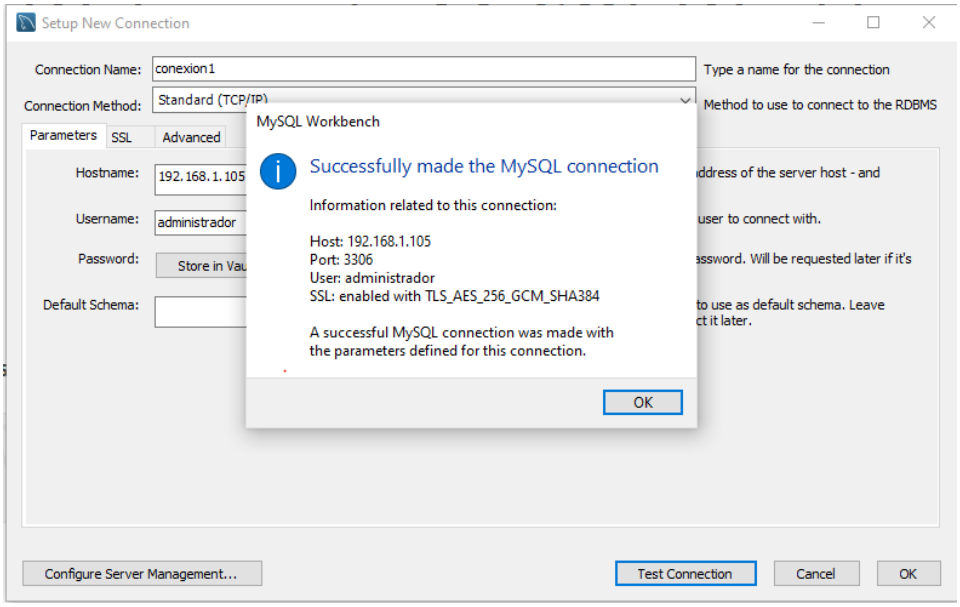

Figura A2.3.8. Verificación de correcta conexión del usuario.

4. Dar clic en Ok para crear la conexión, como se puede observar en la figura, se ha creado la conexión correctamente

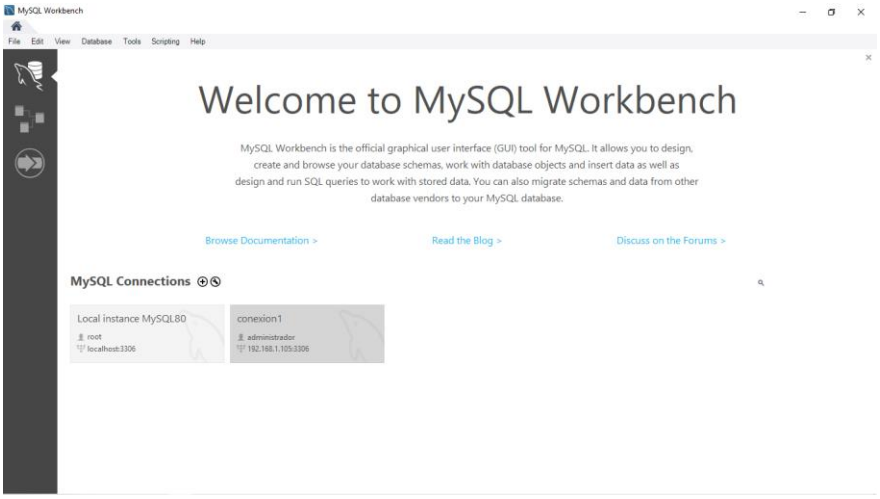

Figura A2.3.9. Ventana de inicio en MySQL Workbench.

A2.3.2. Configuración del conector ODBC.

Para la configuración del conector ODBC previamente se debe tener instalado el Driver MySQL ODBC 8.0.16. el procedimiento de su configuración es de la siguiente manera:

1. En la barra de búsqueda, buscar “ODBC”, nos aparecerá orígenes de datos ODBC, le damos clic en ejecutar como administrador.

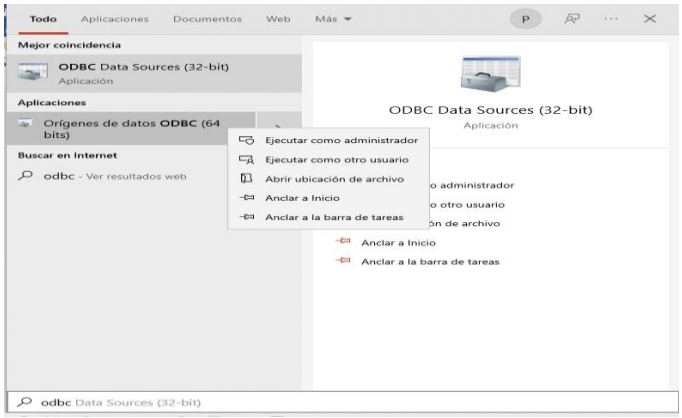

Figura A2.3.10. Programa ODBC.

2. Se desplegará una ventana, dentro de esta ventana, ir a la pestaña DSN de usuario y dar clic en “Agregar”.

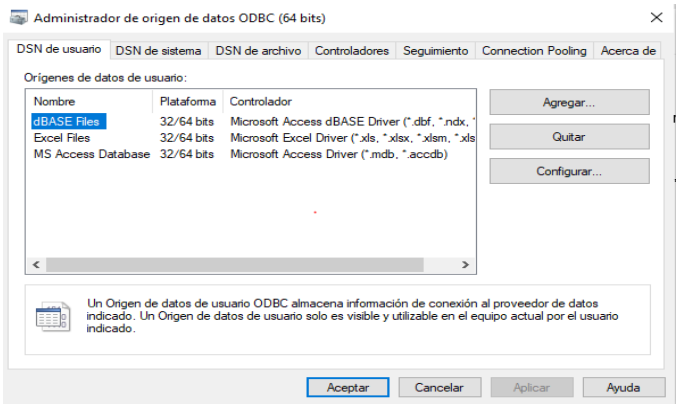

Figura A.3.11. Creación de nuevo DSN en ODBC.

3. Se desplegará una ventana con diversos orígenes de datos, del cual se seleccionará “MySQL ODBC 8.0 Unicode Driver”. Por último, dar clic en finalizar

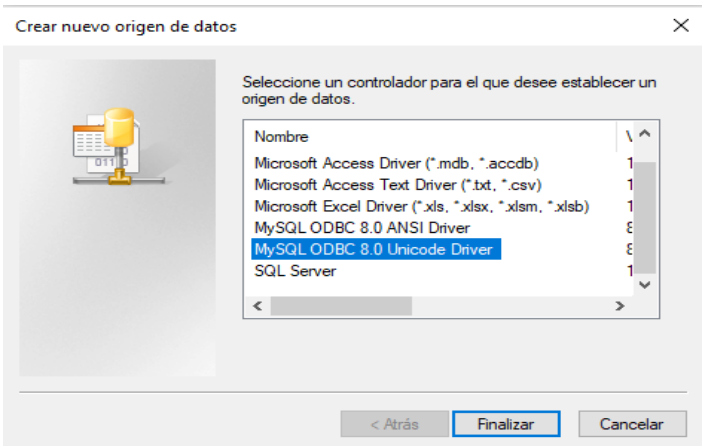

Figura A2.3.12. Selección del DNS a crear.

4. Llenar la ventana que se desplegará con el nombre que se desea dar a la conexión, una pequeña descripción, la dirección IP de la conexión de la base de datos, el

usuario y la contraseña. Para comprobar que existe comunicación con la base de datos de MySQL se le puede dar clic en Test, si la conexión es correcta nos desplegará el siguiente mensaje.

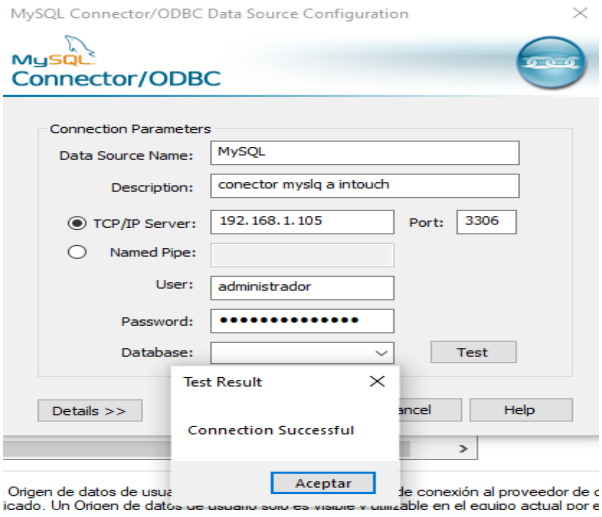

Figura A2.3.13. Configuración del conector DNS creado.

5. Seleccionar la base de datos con la cual se quiere conectar, ya que un usuario puede manejar varias bases de datos. Por último, dar clic en “Ok” para guardar la configuración del controlador ODBC.

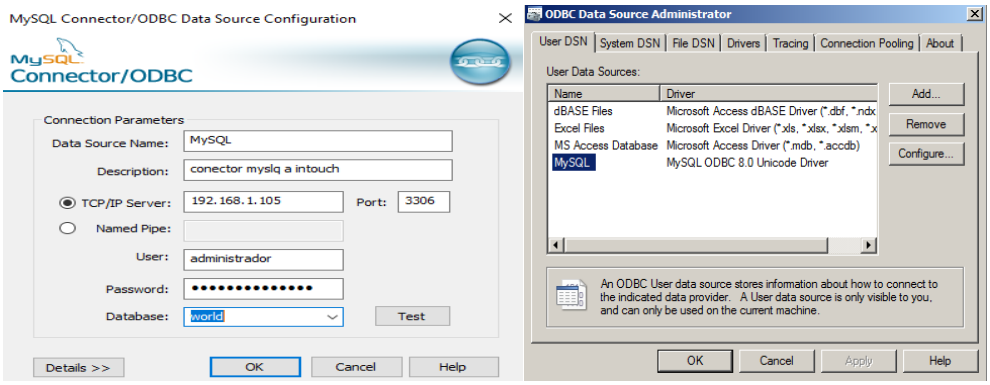

Figura A2.3.14. Configuración del conector ODBC.

A2.4 VENTANAS DEL PROGRAMA DESARROLLADO.

A2.4.1. Ventana de inicio

La ventana de inicio contiene la portada del trabajo realizado, y cuenta con 2 botones que cumplen la función de direccionarnos a los dos modos de funcionamiento que son:

- **Maestro:** abre la ventana del modo maestro MODBUS RTU.
- **Servidor:** abre la ventana del modo servidor MODBUS RTU.

Figura A2.4.1. Ventana de inicio.

A2.4.2. Ventana modo maestro MODBUS RTU.

Esta ventana permite establecer comunicación entre un dispositivo esclavo MODBUS RTU y ventana maestro MODBUS RTU, usando este mismo protocolo mediante ocho códigos de función que sirven para la lectura y escritura de variables booleanas, enteras y flotantes. Su ventana se puede apreciar en la figura.

Figura A2.4.2. Ventana Maestro Modbus RTU.

Existen 2 campos de configuración y uno de visualización, los cuales se explicarán a continuación:

- **Conexión del puerto serial.**

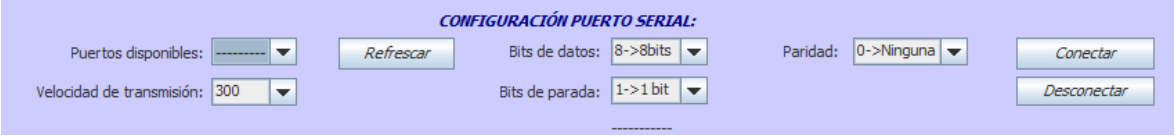

Figura A2.4.3. Parámetros para la configuración del puerto serial.

En la figura se observan los parámetros necesarios a configurar para realizar la conexión y desconexión del puerto serial, éstos se detallan a continuación:

- **Puertos disponibles:** permite elegir entre los puertos seriales disponibles en el computador, se debe seleccionar el puerto en el que se encuentra conectado nuestro dispositivo esclavo.
- **Refrescar:** botón que actualiza el listado de puertos disponibles.
- **Velocidad de transmisión:** permite elegir la velocidad de transmisión de datos.
- **Bits de datos:** permite elegir la longitud de los bits de datos, éste puede ser 7 u 8, para Modbus RTU lo más común es usar 8 bits.
- **Bits de parada:** permite elegir los bits de parada, éstos pueden ser 1 o 2 bits de parada.
- **Paridad:** permite elegir la paridad, ésta puede ser par, impar o ninguna.
- **Conexión:** botón que realiza la conexión del puerto serial con los parámetros establecidos en los anteriores ítems.
- **Desconexión:** desconecta el puerto serial.

Estos parámetros deben configurarse de la misma manera que está configurado nuestro dispositivo esclavo, por ejemplo, nuestro dispositivo esclavo está configurado a 9600 baud, sin paridad, 8 bits de datos y un bit de parada, esta misma configuración se debe ingresar en la configuración del puerto serial, caso contrario no se podrá establecer comunicación con el dispositivo.

- ## Parámetros de configuración.

Figura A2.4.4. Parámetros a configurar para la generación de la trama de solicitud.

Como se puede observar en la figura, esta sección permite configurar los parámetros para generar la trama de solicitud al dispositivo esclavo, a continuación, se detalla cada una de sus partes:

- **ID esclavo:** aquí ingresamos la identificación del dispositivo, éste puede ir de 1 a 255, según el dispositivo.
- **Código de función:** permite elegir entre los ocho códigos de función, los códigos 01, 02, 03 y 04, son códigos de función para lectura de coils, contacts, holding y registros de entrada, mientras que los códigos 05, 06,15 y 16 son códigos de función para la escritura de coils y holding register.
- **Tipo de dato:** permite seleccionar el tipo de dato, éste puede ser booleano, entero con signo o int, entero sin signo o uint y flotante.
- **Dirección Base:** permite elegir la base del dispositivo, es decir si el registro inicial comienza en 0 o 1, si el dispositivo trabaja con dirección de protocolo su registro inicial es 0, si trabaja con dirección de PLC su registro inicial es 1.
- **Endianness:** permite elegir la manera en cómo se envían los datos flotantes, si es big endian la parte más significativa del dato se envía primero y luego la parte menos significativa, si es little endian la parte menos significativa del dato se envía primero y luego la parte más significativa.
- **Dígitos:** permite seleccionar el tamaño del espacio de memoria de los registros, si es 4 la cantidad máxima de registros será 999, si es 5 la cantidad máxima de registros 9999 y si es 6 la cantidad máxima de registros será de 65535, esto depende exclusivamente de las capacidades del dispositivo esclavo. Los dígitos incluyen internamente el subíndice de memoria 0, 1, 3 y 4 de acuerdo con el código de función seleccionado.
- **Dirección de inicio:** aquí se ingresa la dirección desde la cual se desea iniciar la lectura o escritura de registros. Considerar que se debe ingresar la dirección sin el subíndice de memoria ya que ésta se genera internamente al seleccionar el código de función. Por ejemplo, si nuestro registro en el PLC es el 3001, en este campo ingresaremos sólo 001.
- **Cantidad a leer:** permite ingresar la cantidad de datos a leer en el caso de haber seleccionado códigos de función para lectura.
- **Valor:** permite ingresar el valor a escribir en un registro cuando se ha seleccionado los códigos de función 05 y 06, éste depende también del tipo de dato, puede ser booleano, entero con signo o sin signo.
- **Cantidad a escribir:** aquí se ingresa la cantidad de datos a escribir en el caso de seleccionar los códigos de función 15 y 16 que son para escritura múltiple.
- **Ingreso de datos:** permite el ingreso de datos, en el caso de haber seleccionado los códigos de función 15 y 16. Cabe indicar que los datos deben ser ingresados

uno a uno en el caso de enteros y flotantes, en el caso de booleanos se debe ingresar byte por byte.

- **Número de datos:** permite visualizar la cantidad de datos ingresados, en el caso de haber seleccionado escritura múltiple.
- **Datos ingresados:** muestra los datos ingresados, esto en caso de haber seleccionado escritura múltiple.
- **Agregar dato:** botón que agrega un nuevo dato, funciona únicamente cuando se realiza escritura de múltiples datos.
- **Borrar datos:** botón que borra todos los datos ingresados, funciona únicamente cuando se realiza escritura múltiple.
- **Enviar:** botón que envía la trama de solicitud al dispositivo esclavo.
- **Visualización de la respuesta del dispositivo esclavo**

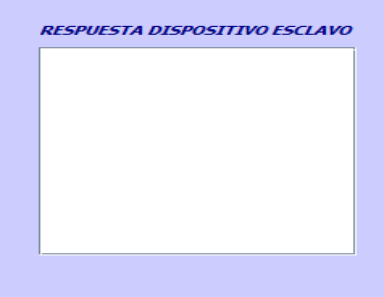

Figura A2.4.5. Visualización de la respuesta del dispositivo esclavo.

En esta área se mostrarán los valores de los registros leídos del dispositivo esclavo, en caso de escritura se mostrará el mensaje escritura correcta cuando se escriba uno o varios registros correctamente.

A2.4.3 Ventana modo servidor MODBUS RTU.

En la figura se puede observar toda la ventana del modo servidor de datos, pero para poder explicar cada parte de esta ventana, se la dividirá en 6 partes que se detallan a continuación.

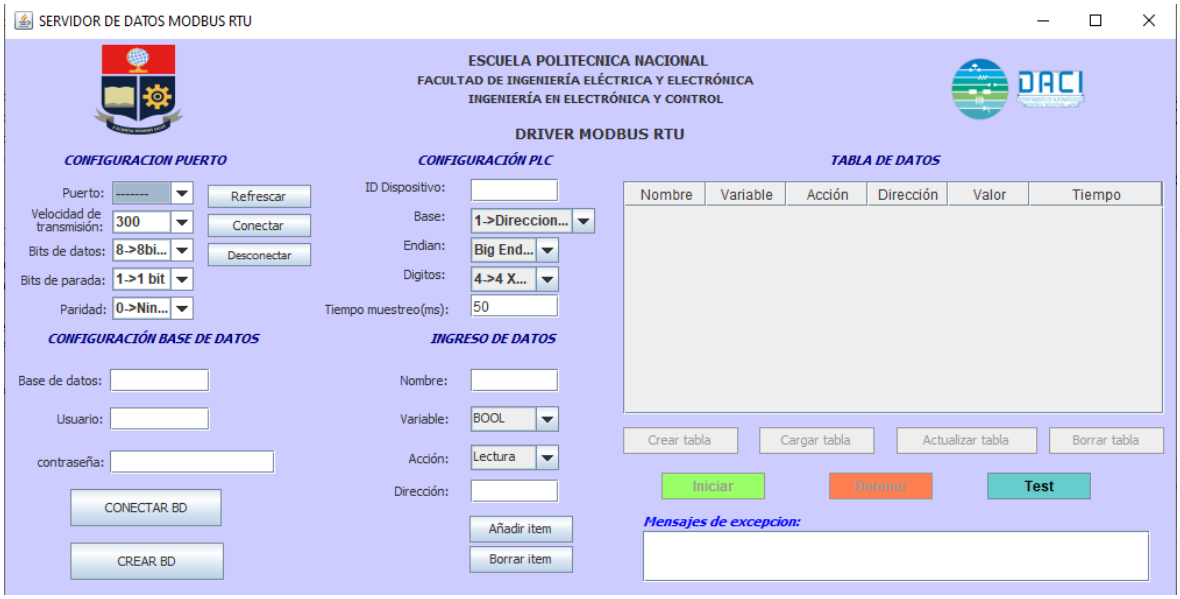

Figura A2.4.7. Visualización de la respuesta del dispositivo esclavo.

- ## CONFIGURACIÓN DEL PUERTO SERIAL.

Figura A.4.8. Configuración del puerto serial.

En la figura se observan los parámetros necesarios a configurar para realizar la conexión y desconexión del puerto serial, éstos se detallan a continuación:

- **Puertos disponibles:** permite elegir entre los puertos seriales disponibles en el computador, se debe elegir el puerto en el que se encuentra conectado nuestro dispositivo esclavo.
- **Refrescar:** botón que actualiza el listado de puertos disponibles.
- **Velocidad de transmisión:** permite elegir la velocidad de transmisión de datos.
- **Bits de datos:** permite elegir la longitud de los bits de datos, éste puede ser 7 u 8, para Modbus RTU lo más común es usar 8 bits.
- **Bits de parada:** permite elegir los bits de parada, éstos pueden ser 1 o 2 bits de parada.
- **Paridad:** permite elegir la paridad, ésta puede ser par, impar o ninguna.
- **Conexión:** botón que realiza la conexión del puerto serial con los parámetros establecidos en los anteriores ítems.

- **Desconexión:** desconecta el puerto serial.

Considerar que los parámetros de comunicación configurados en el dispositivo esclavo deben ser los mismos que los configurados en el servidor de datos, caso contrario no se realizará correctamente la comunicación.

- ## CONFIGURACIÓN DE LA BASE DE DATOS.

Figura A2.4.9. Configuración de la base de datos.

En la figura se observan los parámetros necesarios para establecer comunicación con la base de datos, enseguida se detalla cada uno de estos parámetros:

- **Base de datos:** aquí se ingresa el nombre de la base de datos a la cual se quiere acceder o que se quiere crear.
- **Usuario:** aquí se ingresa el nombre del usuario que tiene acceso a la base de datos de MySQL.
- **Contraseña:** en este campo se ingresa la contraseña del usuario con el cual se quiere acceder a la base de datos de MySQL.
- **Conectar bd:** se conecta con la base de datos ingresada, usando el usuario y la clave ingresada, en caso de no existir la base de datos no enviará mensaje de error.
- **Crear bd:** crea una nueva base de datos con los parámetros ingresados, en caso de ya existir esa base de datos enviará un mensaje de error.

El modo servidor permite crear bases de datos directamente desde el programa sin necesidad de abrir MySQL Workbench, esta opción es útil cuando se requiera realizar un servidor de datos desde 0, en el caso de ya disponer de una base de datos creada, podemos acceder a ésta mediante la opción conectar bd.

- ## CONFIGURACIÓN DEL PLC.

Figura A2.4.10. Configuración del PLC.

En la figura se observa las principales características que posee el dispositivo esclavo con el cual se desea comunicar, cada parámetro se detallará a continuación.

- **ID del dispositivo:** aquí se ingresa la identificación del dispositivo, éste puede ir de 1 a 255, según el dispositivo.
- **Base:** permite elegir la base del dispositivo, es decir si el registro inicial comienza en 0 o 1, si el dispositivo trabaja con dirección de protocolo su registro inicial es 0, si trabaja con Dirección de PLC su registro inicial es 1
- **Endian:** permite elegir la manera en cómo se envían los datos flotantes, si es big endian la parte más significativa de dato se envía primero y luego la parte menos significativa, si es little endian la parte menos significativa del dato se envía primero y luego la parte más significativa.
- **Dígitos:** permite seleccionar el tamaño del espacio de memoria de los registros, si es 4 la cantidad máxima de registros será 999, si es 5 la cantidad máxima de registros 9999 y si es 6 la cantidad máxima de registros será de 65535, esto depende exclusivamente de las capacidades del dispositivo esclavo, los dígitos incluyen los subíndices de memoria 0, 1, 3 y 4.
- **Tiempo de muestreo:** tiempo en el que se realiza la lectura de la trama de respuesta dada por el dispositivo esclavo.
- **INGRESO DE DATOS.**

Figura A2.4.11. Ingreso de datos.

En la figura 2.5.11. se puede observar los parámetros a ingresar para crear una variable, a continuación, se explica cada campo.

- **Nombre:** nombre de la variable a ingresar.
- **Variable:** permite seleccionar el tipo de variable, esta puede ser BOOL, INT, UINT o FLOAT.
- **Acción:** permite seleccionar la acción a realizar, ésta puede ser lectura o escritura de la variable.
- **Dirección:** dirección donde se encuentra una variable, para este caso se debe ingresar el subíndice de memoria seguido con la dirección del registro, por ejemplo, la variable a crear se encuentra en los registros contacts en la dirección 2, la dirección será 1002, donde se puede notar que el subíndice es 1 y la dirección es 2, en caso de que sea de 4 dígitos.
- **Añadir ítem:** añade una nueva variable a la tabla que se desea crear.
- **Borrar ítem:** borra la última variable ingresada en la tabla.
- **TABLA DE DATOS.**

Figura A2.4.12. Tabla de datos.

En la figura se puede observar la plantilla de la tabla de datos, en la cual se mostrará las variables ingresadas, en el caso de iniciar la comunicación se mostrarán los valores de las variables continuamente y el tiempo en el que se actualizó dicha variable.

En este campo existen varios botones y a continuación se detalla su funcionalidad:

- **Crear tabla:** permite crear tablas de datos en MySQL con los datos ingresados en el servidor. Usar esta opción en el caso de haber creado una base de datos nueva.
- **Cargar tabla:** permite traer los valores de una tabla de datos de MySQL creada anteriormente, al servidor. Esta opción se la debe usar en el caso de disponer de tablas creadas anteriormente en MySQL y que se quiera volver a utilizar.

- **Actualizar tabla:** actualiza el valor de las tablas de la base de datos, esta opción se debe utilizar en el caso de agregar o quitar variables en el servidor de datos.
- **Borrar tabla:** permite borrar las tablas de la base de datos y dentro del servidor.
- **Iniciar:** inicia la comunicación continua del servidor de datos con la base de datos y con el dispositivo esclavo.
- **Detener:** detiene la comunicación del servidor de datos.
- **Test:** el botón test permite escribir el valor de una variable en la base de datos, se debe usar para verificar si se está comunicando correctamente con la base de datos y si el valor se escribe correctamente en el dispositivo esclavo o en el caso que no se disponga HMI y se necesite escribir desde el servidor de datos.

A2.5 CONFIGURACIÓN DE LAS VARIABLES EN INTOUCH PARA LA COMUNICACIÓN CON LA BASE DE DATOS.

Parte importante del proyecto, es la conexión de la base de datos del servidor MODBUS RTU e Intouch, para lo cual es necesario realizar la siguiente configuración para poder enlazar las variables del servidor con las variables de Intouch.

El procedimiento es el siguiente:

1. Abrir el programa creado en Intouch.
2. En la parte izquierda de la ventana, nos dirigimos a “Tools”, desplegar el listado “SQL Access Manager” y dar clic en “Bind List”.

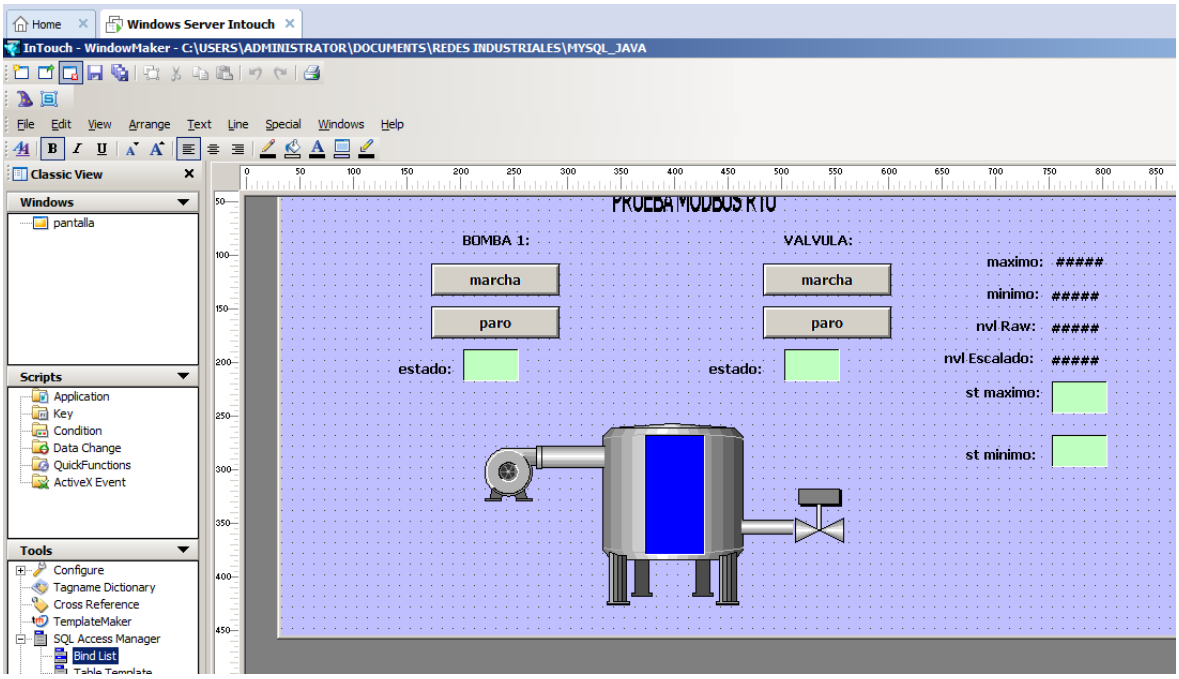

Figura A2.5.1. Pantalla de Intouch.

3. Se desplegará una pequeña ventana, para crear una variable, dar clic en “New”.

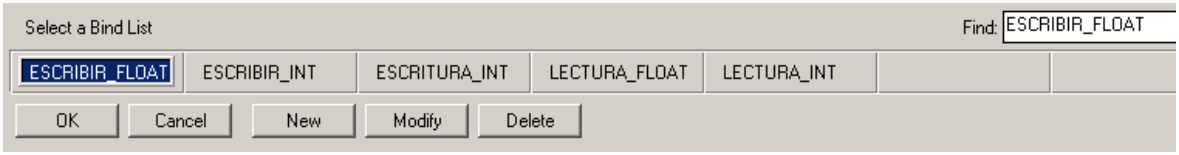

Figura A2.5.2. Menú de Bind List.

4. Se desplegará una ventana, dentro de esta ventana, dar un nombre al bind list que se desea crear, un nombre al tagname que se asociará con la variable de la base de datos en MySQL, y el nombre de la columna donde se encuentra la variable que se desea acceder, por último dar clic en “Move down” para agregar los datos al bind list. En la figura se muestra dos ejemplos de creación de bind list.

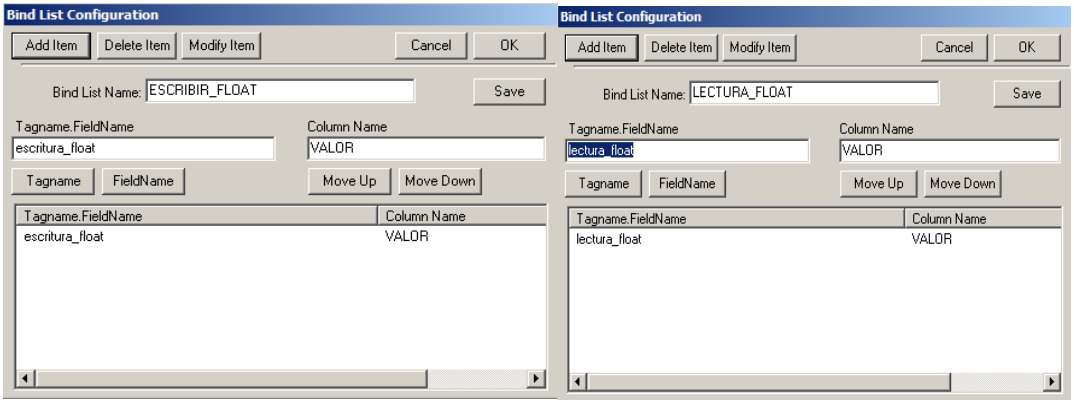

Figura A2.5.3. Configuración del Bind List.

5. Por último dar clic en “Add Ítem” para terminar la creación del bind list.

La cantidad de bind list a crear dependerá de las necesidades que tenga el proceso a supervisar.

Al tener ya creados los bind list, Se debe realizar el script que asocie estas variables con las creadas en Intouch, que realice la conexión y desconexión con la base de datos del servidor. Para lo cual se explicará el funcionamiento de las principales funciones a usar.

- **Consideraciones para leer y escribir en la base de datos del servidor desde Intouch**

El servidor de datos posee 2 tablas, una llamada servidor y otra llamada “histórico”, para interactuar con Intouch, se debe utilizar la tabla “servidor”. La cual tiene el siguiente formato:

NOMBRE	DIRECCION	VALOR	FECHA
starb1	1	0	05:20 21 10-12-21
stopb1	2	0	05:20 21 10-12-21
starb2	3	0	05:20 22 10-12-21
stopb2	4	0	05:20 22 10-12-21
stb1	10001	1	05:20 25 10-12-21
stb2	10002	0	05:20 25 10-12-21
stmax	10003	0	05:20 22 10-12-21
stmin	10004	0	05:20 22 10-12-21
nvlraw	30001	19160	05:20 22 10-12-21
nvlscl	30002	58,45	05:20 24 10-12-21
max	40001	80.00	05:20 24 10-12-21
min	40003	10.00	05:20 24 10-12-21

Figura A2.5.4. Tabla servidor.

La columna “VALOR” tiene formato String, esto para que se pueda soportar las variables Bool, Int, Uint y Float en la misma columna, para lo cual en Intouch se debe realizar conversión de formatos.

- **Comandos para transformar de String a Int y viceversa.**

Para pasar una variable de String a Int se utiliza el siguiente comando:

Entero= StringToIntg (“nombre de la variable”);

Ejemplo:

nivel_raw=StringToIntg(lectura_int);

Para pasar una variable de Int a String se utiliza el siguiente comando:

String=StringFromIntg (“nombre de la variable”, “base”*);*

Donde la base puede ser binaria o 2, octal o 8, decimal o 10

Ejemplo:

escritura_int=StringFromIntg(stop_b1,10);

esta conversión se utiliza tanto para variables enteras como para booleanos, en el caso de booleanos aceptará valores de 1 y 0.

- **Comandos para transformar de String a Float y viceversa.**

Para pasar una variable de String a Float se utiliza el siguiente comando:

Flotante= StringToReal ("nombre de la variable");

Ejemplo:

nvl_scl=StringToReal(lectura_float);

Para pasar una variable de Float a String se utiliza el siguiente comando:

String= *StringFromReal* ("nombre de la variable", "precisión", "tipo de dato");

Donde la precisión será el número de decimales que se desea tener, el tipo de dato que puede ser flotante o "f" o exponencial o "e".

Ejemplo:

escritura_float=StringFromReal (nvl_min, 2, "f");

- **Comandos para conectar y desconectar una base de datos de MySQL a Intouch**

Para conectar Intouch con la base de datos del servidor, usar el siguiente comando:

SQLConnect (ConnectionId," Provider=MSDASQL; DSN=MySQL");

Considerar que el DSN es el conector ODBC creado anteriormente. Como se puede ver en la gráfica

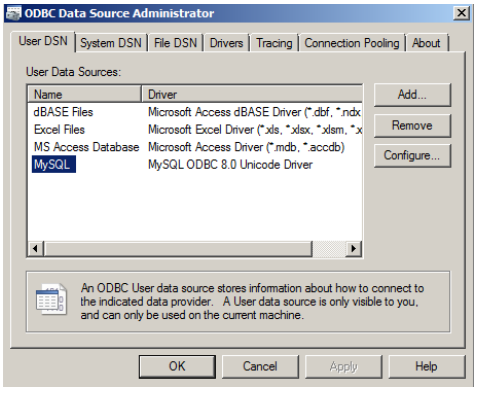

Figura A2.5.5. Conector ODBC creado.

El para desconectar la base de datos se debe usar el siguiente comando:

SQLDisconnect (ConnectionId);

- **Comandos para escribir valores desde Intouch a MySQL**

El comando para actualizar valores en la tabla de datos desde Intouch es el siguiente

SQLUpdate (ConnectionId, Nombre de la Tabla, Bind List, Condición);

Donde el nombre de la tabla, es la tabla creada en MySQL donde se encuentran las variables del servidor, el bind list donde se asocia el tagname con las variables en MySQL, por último, la condición que permitirá direccionar a una variable en especifica dentro de la base de datos que queremos actualizar.

Para acceder a una variable en específico, podemos realizarlo mediante el nombre de la variable o la dirección, se ha decidido usar la dirección como condición para direccionar a la variable elegida.

A continuación, se muestra un ejemplo de cómo implementar el comando.

SQLUpdate (ConnectionId, "servidor", "ESCRITURA_INT", "Dirección=30001", "");

- **Comandos para leer elementos de una base de datos desde Intouch a MySQL**

Los comandos para leer el valor de una variable de la base de datos del servidor a Intouch es el siguiente:

SQLSelect (ConnectionId, Nombre de la tabla, Bind List, condición);

SQLLast (ConnectionId);

SQLEnd (ConnectionId);

Donde el nombre de la tabla, es la tabla creada en MySQL donde se encuentran las variables del servidor, el bind list donde se asocia el tagname con las variables en MySQL, por último, la condición que permitirá direccionarnos a una variable en especifica dentro de la base de datos que queremos actualizar.

Para acceder a una variable en específico, podemos realizarlo mediante el nombre de la variable o la dirección, se ha decidido usar la dirección como condición para seleccionar a la variable elegida.

A continuación, se muestra un ejemplo de cómo implementar el comando.

SQLSelect (ConnectionId, "servidor", "LECTURA_INT", "Dirección=30001", "");

En la siguiente imagen se muestra un ejemplo de implementación de un script que permita la interacción de Intouch con la base de datos de MySQL, tanto para lectura como escritura de datos.

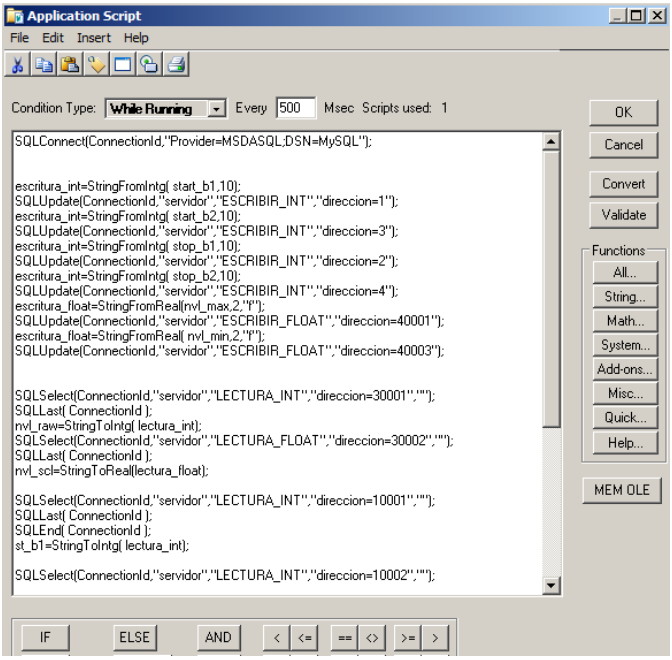

Figura A2.5.6. Ejemplo de script en Intouch con conexión a la base de datos.