Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Implementación de la comunicación con la serie de actuadores Exlar TLMXX utilizando Modbus RTU

Autor: Yevang Nhiavue

Tutor: Ing. Richar Sosa López

Santa Clara

2014

"Año 56 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

DASC

Digitally signed by DASC
DN: cn=DASC, c=<n, o=UCLV, ou=FIE
Date: 2014.06.11 14:39:56 -05'00'

TRABAJO DE DIPLOMA

Implementación de la comunicación con la serie de actuadores Exlar TLMXX utilizando Modbus RTU

Autor: Yevang Nhiavue

yevang@uclv.edu.cu

Tutor: Ing. Richar Sosa López

Dpto. de Automática, Facultad de Ing. Eléctrica, UCLV

rslopez@uclv.edu.cu

Santa Clara

2014

"Año 56 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

Nunca deja de aprender porque la vida nunca deja de enseñar.

El éxito es el premio del esfuerzo personal; sigue siempre adelante te espera un mejor futuro. El éxito es el fruto del trabajo y la grandeza personal para poder llegar a obtenerlo. El éxito se obtiene solo con pensamiento firme y seguro de saber lo que se quiere llegar a ser.

(H. Miranda)

DEDICATORIA

A mis Padres.

A mis Hermanos.

A mi TíoXongLao y Tía Leepaia Bouapao.

A todas personas en mi Familia.

AGRADECIMIENTOS

A mis Padres, por la confianza, el apoyo y su ayuda en toda la vida.

A mis hermanos y mis cuñadas, por su apoyo.

A mi familia por su confianza y el apoyo en todo.

A mi Tío XongLao y Tía Leepaia Bouapao, por su ayuda.

A mi Tutor Richar Sosa López, por la ayuda en todo el tiempo, que sin su ayuda este trabajo no hubiese sido posible.

A mis amigos en el aula, por la ayuda en todos momentos que necesité durante el tiempo de la carrera.

A todos mis amigos Cubanos y extranjeros por compartir, por ser amistad.

A mis amigos Laosianos por ser amistad, por la ayuda, por compartir en todos momentos buenos como difíciles durante el tiempo que estudiamos en Cuba.

A mis Profesores en la Facultad de Ingeniería Eléctrica, especialmente a los profesores en el Departamento de Automática, por sus conocimientos que me han brindado.

A la Universidad Central¨ Marta Abreu¨ de Las Villas.

Gracias a todos.

TAREAS DE INVESTIGACIÓN

- El análisis de la situación actual del uso del MODBUS como protocolo de comunicaciones en aplicaciones industriales y sus variantes más reconocidas.
- La asimilación del principio estructural del protocolo MODBUS para la comunicación en buses de campo.
- El estudio de los campos de funciones específicos del fabricante Exlar para el actuador TLMXX.
- La implementación de un método de generación de cadena del protocolo con cálculo de CRC.
- La implementación de un método de lectura e interpretación de cadena del protocolo.
- El análisis de los aspectos y procedimientos a realizar en las comunicaciones resultantes.

Firma del Autor	Firma del Tutor

RESUMEN

A partir de la necesidad de mejorar el la solución para el mando de los actuadores de la serie TLMXX del fabricante Exlar, esta investigación arroja una clase de programación que permite el encapsulamiento de los métodos necesarios para la comunicación con este dispositivo utilizando el estándar Modbus RTU. La misma es capaz de mejorar la señal de mando y aportar la posibilidad de configuración on-line, actualmente inexistente esta última sin el software profesional del fabricante.

Se realiza una interfaz que prueba la efectividad de la clase aportada, así como el modelado de la misma utilizando los preceptos de la ingeniería de software. Es desarrollada utilizando el framework Qt 5.2 ostentando las ventajas de ser multiplataforma y estar bajo las licencias que brinda el software libre por lo que puede utilizarse en posteriores aplicaciones comerciales por nuestro país.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La automatización industrial es un conjunto de técnicas que involucran la aplicación e integración de sistemas mecánicos, electrónicos y computacionales para operar y controlar diferentes tipos de sistemas industriales de forma autónoma. El desarrollo de aplicaciones industriales incluye procesos de control y automatización manejados e implantados mayormente por ingenieros eléctricos y electrónicos, y en menor escala por ingenieros de sistemas. Estos ingenieros emplean herramientas versátiles seleccionadas dependiendo de las características de los sistemas de control, de la red industrial y de algunos otros elementos. Para el óptimo manejo de dichas herramientas es indispensable contar con fundamentos teórico – prácticos que permitan (en este caso al ingeniero de sistemas) ir acorde con las exigencias y oportunidades del medio; es por ello que se hace necesario que las bases adquiridas en la universidad permitan acortar la brecha existente entre el conocimiento académico y las necesidades del sector industrial(García, 2003).

Como aparte de la automatización industrial, las comunicaciones industriales son aquellas que permiten el flujo de información del controlador a los diferentes dispositivos a lo largo del proceso de producción: detectores, actuadores, sensores entre otros. Dada la gran variedad de sistemas de comunicación entre equipos industriales, de los cuales la mayoría son cerrados, se ha optado por el desarrollo de un entorno que permita tanto la implementación de protocolos de especificaciones conocidas en un sistema de comunicación completo, desde el medio físico hasta el nivel más alto de red(2011, Domingo and Caro, 2003).

Las redes industriales nacieron con el fin de unir todos los dispositivos que coexisten dentro de una empresa, dedicados al control de máquinas o partes de un proceso cerrado. Debido a la variedad de equipos y necesidades que se presentan dentro de la industria (que difieren muy a menudo entre una empresa y otra), se ha desarrollado una gran variedad de protocolos que permitan la comunicación de PCs (Computadores Personales) o PLCs (Controladores Lógicos Programables o Autómatas Programables) con los diferentes instrumentos de campo. Uno de estos protocolos es *MODBUS*, desarrollado por Modicon hacia el año de 1979.

En el curso 2011-2012 se realizó una colaboración por parte del CIDNAV (Centro de Investigación y Desarrollo Naval) y el GARP (Grupo de Automatización Robótica y Percepción) mediante la cual se generó la problemática de la sustitución del sistema de actuación hidráulico Vetus en el HR-CAUV por alguna variante eléctrica lineal. Después de realizados los estudios correspondientes de fiabilidad y factibilidad de los posibles productos a importar se decidió que la mejor solución de las disponible era el Actuador Lineal serie TLMXX del fabricante TRITEX.

Después de importado, específicamente el modelo TLM30 fue el utilizado para validar los resultados del trabajo de diploma del curso 2012-2013: "Actuador lineal TRITEX TLM30 en aplicaciones de control" del ahora ingeniero Alejandro Vázquez Hernández(Vazquez, 2013). En el mismo se obtiene como resultados principales la evaluación y estudio de la posible utilización de este actuador en las diferentes aplicaciones de control en que viene trabajando el GARP, así como la sistematización del conocimiento adquirido mediante la documentación de los procedimientos principales y la elaboración de un manual de usuario para el trabajo con el mismo. Como continuación de este trabajo proponemos, basado en las recomendaciones del mismo, mejorar la comunicación y configuración para lograr la robustez de los mismos mediante el protocolo Modbus.

Se escogió el protocolo Modbus, para el desarrollo de esta investigación, porque a pesar de no estar estandarizado internacionalmente, ha tenido una gran acogida desde su creación; logrando una gran difusión dentro de las redes industriales. Además es un protocolo de mediana complejidad, por lo que constituye una opción factible para el desarrollo de esta investigación.

Una red industrial Modbus está compuesta por dos dispositivos principales, un dispositivo maestro (PC) y un dispositivo esclavo

Los dispositivos esclavos, en este caso el actuador de la serie Tritex Exlar TLMXX, el cual permite adquirir y almacenar datos provenientes del maestro mediante la interfaz RS485.

Todo dispositivo maestro necesita de un programa de aplicación capaz de transmitir y recibir datos cuyas tramas están estructuradas siguiendo las especificaciones del protocolo Modbus.

Para esta tesis, se implementa una clase de software que contemple la configuración y la comunicación con el actuador que permita obtener un canal de transmisión y recepción de datos que sea robusto en contra de los ruidos y otras perturbaciones del entorno industrial.

Situación del problema

En el panorama actual, las comunicaciones del actuador con su medio de cómputo, distan de ser robustas, debido a que las señales de mando se realizan mediante voltajes proporcionales comprendidos entre 0 y 10 V. Como el entorno industrial es extremadamente ruidoso y la robótica de servicios demanda altos niveles de fiabilidad e independencia, consideramos que esta situación deba de ser mejorada con las comunicaciones digitales como es la transmisión de datos por el estándar RS485 mediante el protocolo Modbus RTU.

Por otra parte no existen oportunidades de configurar el actuador on-line y cualquier posible error de configuración, es necesario resolverlo mediante el software profesional del dispositivo, que debe de funcionar en otro medio de cómputo que soporte Microsoft Windows solamente.

Así mismo, no se cuenta con una guía que documente el conocimiento alrededor de los procedimientos y operaciones para la comunicación del actuador mediante el estándar Modbus RTU. Basado en estas problemáticas es posible definir el siguiente problema científico:

Es necesario mejorar la comunicación y configuración existente para el actuador TLMXX y documentar los procedimientos para que sea posible la explotación del mismo por los futuros usuarios.

Interrogantes Científicas

- ¿Cuál es la situación actual del uso del Modbus como protocolo de comunicaciones en aplicaciones industriales y sus variantes más reconocidas?
- ¿Cuál es el principio estructural del protocolo Modbus para la comunicación en buses de campo?
- ¿Cuáles son los campos de funciones específicos del fabricante Exlar para el actuador TLMXX?

- ¿Cómo implementar un método de generación de cadena del protocolo con cálculo de CRC?
- ¿Cómo implementar un método de lectura e interpretación de cadena del protocolo?
- ¿Cuáles son los aspectos y procedimientos a realizar en las comunicaciones resultantes?

Como **Posible Impacto,** con la puesta en práctica de las soluciones arrojadas por esta investigación es posible mejorar la comunicación y configuración on-line del actuador, lo que permite un considerable aumento en la fiabilidad del sistema que se traduce en un notable ahorro por concepto de malfuncionamientos innecesarios y de falta de operatividad en determinadas circunstancias de ambiente ruidoso, las cuales pudieran ocasionar daños irreversibles también al sistema controlado o al actuador en sí.

A partir de la finalización de la misma se cuenta con una clase que encapsula un conjunto de funciones y atributos que permiten trabajar con el mando y configuración del equipo ahorrando su encargo a una empresa de software, así como los métodos obtenidos y debidamente documentados son de gran uso en paradas y mantenimientos del sistema.

Como **resultados** de esta investigación se pretende incorporar nuevos conocimientos a la temática del uso de los actuadores lineales inteligentes, que contribuyan a la asimilación de este tipo de tecnología en el GARP. La misma está enfocada en el problema de mejorar apreciablemente las comunicaciones de la señal de mando y configuración on-line hacia el dispositivo TLMXX de la Exlar, utilizando un bus de campo, así como generar y documentar los procedimientos y herramientas necesarios para alcanzar estas metas y que simplifique el posterior trabajo de los futuros investigadores que asuman tareas relacionadas con estas tecnologías.

La viabilidad del presente trabajo nace como complemento de la implementación de los actuadores lineales prismáticos en las aplicaciones del GARP. A partir de un conjunto de experiencias y previamente abordada la decisión de su importación y utilización se demostró que era necesario la mejora de la comunicación del mando y la configuración de estos equipos. Por lo que resulta plenamente viable debido a que el único gasto

considerable que se hace es el tiempo y la dedicación de los recursos humanos envueltos en esta tarea, ya que la importación de los medios y componentes necesarios fueron realizados en investigaciones anteriores y se aprovechan en la presente para dar completamiento a las mismas.

En consecuencia con dichas problemáticas anteriormente mencionadas, los objetivos de esta investigación son los siguientes:

Objetivo General:

Programar una clase que implemente la comunicación con el actuador TLMXX mediante el protocolo Modbus RTU que permita mejorar la señal de mando existente y añadir las opciones de configuración on-line del dispositivo.

Objetivos Específicos:

- Revisar las fuentes bibliográficas especializadas con el tema objeto de investigación.
- Asimilar la información provista por el fabricante para implementar tal comunicación.
- Implementar una clase en lenguaje C++ que encapsule los métodos necesarios que permitan la comunicación del medio de cómputo con el dispositivo Tritex mediante el protocolo Modbus RTU.
- Documentar el modelado de la clase antes mencionada para su posterior utilización por los programadores futuros.

Estructura y contenido de la tesis:

La tesis, posterior a la introducción, incluye tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos. A continuación se muestra un resumen del contenido de cada capítulo:

CAPITULOI*:* Se dedicara al análisis de las metodologías de trabajo para los buses de campo industriales en el mundo, así como las diferentes variantes de aplicación de las mismas en las diferentes fuentes bibliográficas consultadas.

CAPITULO II: Se utiliza para elaborar el basamento teórico de la investigación partiendo de la solución propuesta, todas las consideraciones y herramientas a utilizar, así como los procedimientos que dieron origen a los resultados

CAPITULO III: Se dedica a expresar los resultados obtenidos a partir de la solución de los objetivos planteados particularmente. Se validan los métodos propuestos desde pruebas experimentales, así como la documentación y modelado de la solución planteada.

CAPÍTULO 1. MARCO TEÓRICO

A nivel internacional, el área de las redes industriales ha sido bastante explotada, sobre todo por los Ingenieros de Sistemas, un ejemplo de esto han sido los proyectos relacionados con las redes industriales utilizando el protocolo Modbus. Existen importantes universidades del mundo que se han dedicado a la investigación de esta temática el cual podemos citar: En 2005, en la Facultad de Ingeniería de Sistemas, Corporación Universitaria Rafael Núñez, Cartagena de Indias(Correa and Manjarrez, 2005). En 2009, en el Departamento de Ingeniería Tecnología e Industrial, Universidad de Texas A$M(Papasideris and Landry, 2009). En 2009, en el departamento de Eléctrica y Electrónica de la Universidad de Las Fuerzas Armadas ESPE Extensión Latacunga(Gallo and Herrera, 2009). En 2010, en el Departamento de MVP Samaj's KABGT, Colegio de Ingeniería, Nashik, Maharashtra(Dhumane, 2010).

En este capítulo se realiza un análisis introductorio al tema de redes industriales, así como la situación actual del uso del Modbus como protocolo de comunicaciones en aplicaciones industriales y sus variantes más reconocidas.

1.1 Fundamentos de redes industriales

Las redes industriales nacieron con el fin de unir todos los dispositivos que coexisten dentro de una empresa, dedicados al control de máquinas o partes de un proceso cerrado. Toda esa gran variedad de equipos que se presentan en la industria dentro de los que están los PC (Computadores Personales), PLC (Controladores Lógicos Programables o Autómatas Programables), instrumentos de campo, entre otros, son articulados en una red industrial, aumentando de paso el rendimiento de cada uno de ellos(Mak and Radford, April 1996).

Desde el punto de vista institucional, al tener una red de comunicación industrial representa una ayuda muy importante que permite estudiar como coexisten en las empresas, diferentes equipos y dispositivos dedicados al control de una máquina o una parte cerrada de un proceso, como los PCs, PLCs, instrumentos de campo, sensores, actuadores, etc. A nivel industrial estas redes proporcionan múltiples ventajas como son:

- Visualización y supervisión de todo el proceso productivo.
- Toma de datos del proceso más rápida o instantánea.
- Mejora del rendimiento general de todo el proceso.

- Posibilidad de intercambio de datos entre sectores del proceso y entre departamentos.
- Programación a distancia, sin necesidad de estar a pie de fábrica.

Las ventajas son evidentes, a cambio de un cierto costo que es amortizado a largo plazo con las mejoras en los procesos.

Las necesidades primordiales del usuario común de una red industrial es que su implementación es sencilla y requiere tiempos de desarrollo reducidos, aumenta las prestaciones del sistema, además reduce el cableado para la transmisión y recepción de datos en el control, programación y diagnosis sobre la misma red.

1.1.1 Estructura Jerárquica de las Comunicaciones Industriales

En una red industrial coexisten equipos y dispositivos de todo tipo, los cuales suelen agruparse jerárquicamente para establecer conexiones lo más adecuadas a cada área. De esta forma se definen cuatro niveles dentro de una red industrial(T.Amy, 2008):

- **Nivel de Gestión**: Es el nivel más elevado y se encarga de integrar los niveles siguientes en una estructura de fábrica, e incluso de múltiples factorías. Las máquinas aquí conectadas suelen ser estaciones de trabajo que hacen de puente entre el proceso productivo y el área de gestión, en el cual se supervisan las ventas, stocks, etc. Se emplea una red de tipo LAN (Local Area Network) o WAN (Wide Area Network).
- **Nivel de Control**: Se encarga de enlazar y dirigir las distintas zonas de trabajo. A este nivel se sitúan los autómatas de gama alta y los ordenadores dedicados a diseño, control de calidad, programación, etc. Se suele emplear una red de tipo LAN.
- **Nivel de Campo y de Proceso**: Se encarga de la integración de pequeños automatismos (autómatas compactos, multiplexores de E/S, controladores PID, etc.) dentro de subredes o "islas". En el nivel más alto de estas redes se suelen encontrar uno o varios autómatas modulares, actuando como maestros de la red o maestros flotantes. En este nivel se emplean los buses de campo.
- **Nivel de Entrada/Salida**: Es el nivel más próximo al proceso. Aquí están los instrumentos de campo (sensores, actuadores, etc.), encargados de manejar el proceso productivo y tomar las medidas necesarias para la correcta automatización y supervisión.

1.2 Bus de campo

El bus de campo(CEKIT, 1999) constituye el nivel más simple y próximo al proceso dentro de la estructura de comunicaciones industriales. Está basada en procesadores simples y utiliza un protocolo mínimo para gestionar el enlace entre ellos. Los buses de campo más recientes permiten la comunicación con buses jerárquicamente superiores y más potentes.

En un bus de campo se engloban las siguientes partes:

- **Estándares *de* Comunicación**: cubren los niveles físico, de enlace y de comunicación establecidos en el modelo OSI (Open Systems Interconnection).
- **Conexiones Físicas**: en general, las especificaciones de un determinado bus admiten más de un tipo de conexión física. Las más comunes son semi-dúplex (comunicación en banda base tipo RS485), RS422 y conexiones en bucle de corriente.
- **Protocolo de Acceso al Medio (MAC) y de Enlace (LLC)**: consiste en la definición de una serie de funciones y servicios de la red mediante códigos de operación estándar.
- **Nivel de Aplicación**: es el dirigido al usuario, apoyándose en las funciones estándar antes mencionadas para crear programas de gestión y presentación. La aplicación suele ser propia de cada fabricante, permitiendo a lo sumo la programación en un lenguaje estándar.

1.2.1 Buses de Campo más Importantes

Hay diversos buses según fabricantes y agrupaciones de fabricantes, siendo los más extendidos los siguientes:

- **MODBUSMODICON**: Marca registrada de GOULD INC., define un protocolo de comunicación de topología maestro-esclavo. Su principal inconveniente es que no está reconocido por ninguna norma internacional.
- **BITBUS**: Marca registrada por Intel. De bajo coste y altas prestaciones. Intel cedió a dominio público el estándar, por lo que se considera un estándar abierto. Está reconocido por la normativa IEEE 1118. Se trata de un bus síncrono, cuyo protocolo se gestiona completamente mediante el microcontrolador 8044.

- **PROFIBUS**: Impulsado por los principales fabricantes alemanes. El protocolo es un subjuego de MINIMAP. Está impulsado por ser un estándar abierto y bajo norma DIN 19.245.
- ***S*-BUS**: No es un bus de campo propiamente dicho, sino un sistema multiplexor/demultiplexor que permite la conexión de E/S remotas a través de dos pares trenzados.
- **FIB *(*Factory Instrumentation Bus*)***: Impulsado por fabricantes y organismos oficiales franceses.
- **MIL-STD-1553B**: Adoptado por algunos fabricantes en USA.

1.3 Protocolo de Comunicación Modbus

El Protocolo Modbus fue desarrollado por MODICON hacia el año 1979, para interconectar inicialmente sus controladores programables, pero con el pasar del tiempo ha tenido una gran acogida y se ha convertido en el "lenguaje" común utilizado por muchos controladores y otros dispositivos de monitoreo y control desarrollados por otros fabricantes**(CIA, 2010, OMEGA, 2005, MODBUS.Org, 2002, Modbus-IDA, 2006)**.

Este protocolo define una estructura de mensaje que los dispositivos interconectados reconocen y usan, con independencia del tipo de redes sobre la cual se comuniquen. Describe el proceso que usa un dispositivo, para pedir acceso a otro dispositivo, cómo responde a las peticiones desde otros dispositivos y cómo se detectan y notifican los errores. Establece un formato común para la disposición y contenido de los campos de mensaje.

Durante la comunicación sobre una red Modbus, el protocolo determina cómo cada dispositivo reconoce su dirección, reconoce un mensaje enviado a él, determinará el tipo de acción a tomar y extrae cualquier dato u otra información contenida en el mensaje. Si se requiere una repuesta, el dispositivo construye el mensaje respuesta y lo envía utilizando el Protocolo Modbus.

Los dispositivos interconectados que utilizan el Protocolo Modbus para comunicarse, normalmente tienen como medio físico una interfaz RS485, para lograr una conexión multipunto o multinodo, aunque en algunas ocasiones, se pueden encontrar dispositivos

estableciendo comunicación bajo el Protocolo Modbus; usando como medio físico una interfaz RS232. Las características de cada una de estas normas (RS232 y RS485).

Los dispositivos se comunican usando una técnica **maestro – esclavo**, en la cual sólo un dispositivo (el maestro) puede iniciar transacciones (llamadas 'peticiones'-'queries'). Los otros dispositivos (los esclavos) responden suministrando al maestro el dato solicitado, o realizando la acción solicitada en la petición. Entre los dispositivos maestros típicos se incluyen los procesadores centrales, los paneles de programación, PC (Computadores Personales) y PLC (Controladores Lógicos Programables). Algunos de los esclavos típicos son los PLC (Controladores Lógicos Programables), controladores, actuadores, analizadores y tarjetas de adquisición de datos.

El maestro puede comunicarse con esclavos individualmente o puede generar un mensaje en modo difusión a todos los esclavos. Los esclavos devuelven un mensaje (llamado 'respuesta') a las peticiones que les son enviadas individualmente. No se devuelven respuestas a peticiones en modo difusión enviadas desde el maestro.

El protocolo Modbus establece el formato para la petición del maestro, colocando en ella la dirección del dispositivo esclavo (0 en caso de 'difusión'), un código de función que define la acción solicitada, cualquier dato que haya de enviarse y un campo de comprobación de error. El mensaje de respuesta del esclavo está también definido por el protocolo Modbus. Contiene campos confirmando la acción tomada, cualquier dato que haya de devolverse y un campo de comprobación de error. Si el mensaje recibido por el esclavo es defectuoso o el esclavo es incapaz de realizar la acción solicitada, construye un mensaje de error y lo envía como respuesta.

Ciclo Petición – Respuesta

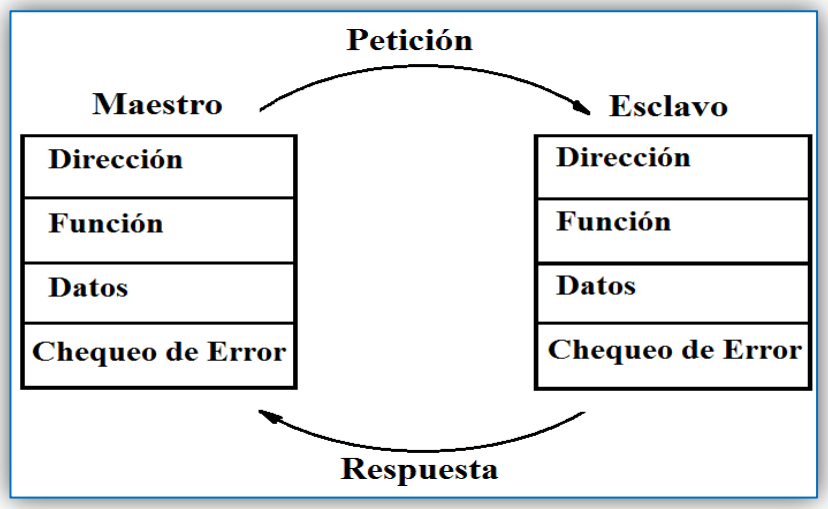

Figura: 1 Ciclo petición-respuesta del maestro-esclavo

- ***La Petición*:** El código de función en la petición indica al dispositivo esclavo, al cual es mensaje es enviado, el tipo de acción a realizar. Los bytes de datos contienen cualquier información adicional que el esclavo necesita para llevar a cabo la función. Por ejemplo el código de función 03 pide al esclavo que lea registros de lectura y escritura (Holding Registers) y responda enviando los valores contenidos en estos registros del esclavo. El campo de datos debe contener la información que indique al esclavo en qué registro debe iniciar la lectura y cuántos ha de leer. El campo de comprobación de error proporciona un método para que el esclavo valide la integridad del contenido del mensaje recibido.
- ***La Respuesta***: Si el esclavo elabora una respuesta normal, el código de función contenido en la respuesta es una réplica del código de función enviado en la petición. Los bytes de datos contienen los datos solicitados por el maestro en la petición, y que han sido recolectados por el esclavo, tales como valores de registros o estados de registros. Si ocurre un error, el código de función contenido en la respuesta es diferente al código de función enviado en la petición, para indicar que la respuesta es una respuesta de error y los bytes de datos contienen un código que describe el error. El campo de comprobación de error permite al maestro confirmar que los contenidos del mensaje son válidos.

1.4 Descripción General del Actuador EXLAR TLM20

La unidad de actuación empleada para la realización de esta tesis, la constituye el actuador lineal eléctrico *TRITEX* de modelo TLM20 patentado por *EXLAR Co*(Vazquez, 2013). Los actuadores de la serie *TRITEX*, emplean un mecanismo de tornillo de rodillo planetario, diseñado para convertir el torque rotacional generado por el motor de corriente directa, en un movimiento lineal prismático bidireccional; como alternativa en aplicaciones que anteriormente requerían de cilindros neumáticos o hidráulicos.

La tecnología del mecanismo de tornillo de rodillo planetario (*roller screw*), se basa en un arreglo planetario alrededor de un husillo que es conectado al vástago del actuador. Representa la solución de *EXLAR Co*. a la anterior técnica de transmisión mediante rodillo de pelota (*roller ball)*, incrementando considerablemente el área de contacto; lo cual resulta en un aumento de hasta 15 veces la capacidad para el manejo de cargas, además de mejorar la fuerza aplicada y la tiesura, en una moción altamente robusta y de larga vida.

Este dispositivo de actuación combina, un motor *DC* sin escobillas (*brushless motor*), un amplificador para el servo, y el control de posición a lazo cerrado; dentro de un compacto encapsulado de calidad industrial. Realimenta la posición a través de, encoders incrementales y absolutos, o por LVDT; además de proporcionar la medida de fuerza de tiempo real exacta, mientras el movimiento se genera.

Integra para la optimización del control, un conjunto de *entradas-salidas*, digitales como analógicas; además de la capacidad para comunicación serie con protocolo Modbus RTU mediante la norma *RS-485*.

A continuación se presentan las principales características asociadas a este dispositivo(Tritex, 2009):

Tabla1: Características técnicas del *TRITEX TLM20*.

Características de actuación	*Valor*	*Unidad de Medida*
Señal de entrada	(4-20) o (0-10)	mA o V
Impedancia de entrada	500	Ohm

Longitud de carrera	0 - 12	pulgadas
Fuerza Máxima	500	Lbf
Velocidad Máxima	33	cm / s
Resolución	<0.025	% de rango
Características de instalación	*Valor*	*Unidad de Medida*
Alimentación	24-48 (10)	V(A)
Precisión de posicionamiento	<1	% de rango
Norma de Sellado	IP54/65	
Temperatura de operación	0 - 55	° C
Conectores	M23	

Este actuador en particular, fue seleccionado por parte del *GARP*, para la sustitución de la actuación lineal hidráulica, en la manipulación de las superficies de control de rumbo y profundidad del vehículo autónomo subacuático (*HRC-AUV)*; desarrollado en colaboración con el *CIDNAV*. La selección corresponde a que presenta el mayor índice de fuerza aplicada, a pesar de limitar la velocidad para la actuación.

1.4.1 Instalación del Actuador

Una parte importante dentro de la instalación del dispositivo, en el entorno de cualquier aplicación; lo constituye la configuración del suministro de poder para la actuación. En este caso se recomienda diferenciar los suministros asociados al TLM20, dedicando una fuente para el consumo de poder, así como otra para el respaldo eléctrico de la activación de las entradas/salidas digitales; manteniéndose de esta forma, el aislamiento eléctrico entre las unidades de potencia y control dentro del actuador.

En vista a conseguir la máxima capacidad de potencia para la actuación, se considera necesario emplear para el suministro al bus de poder, una fuente no-regulada con características de 48 VDC y una entrega mayor a 10 Amp. De esta forma no se limita el consumo para la operación; además de que se mejora el tiempo de respuesta para la actuación, dado que el desplazamiento se realiza al máximo de la velocidad lineal disponible para el dispositivo.

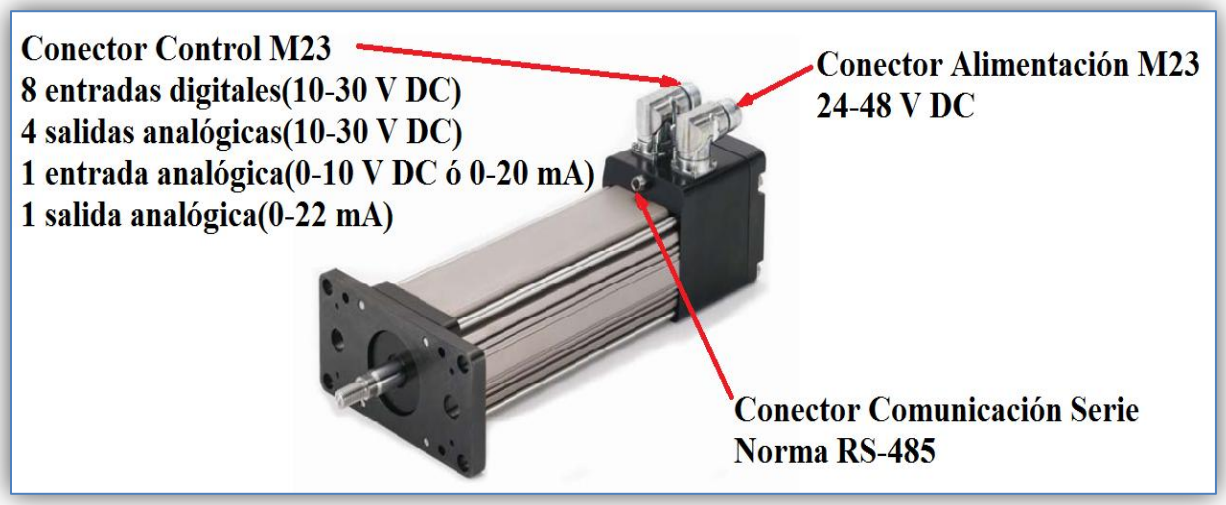

Figura 2: Ilustración de la instalación del *TLM20* según la capacidad de sus conectores.

(a) Conector M23 de 8 pines.

(b) Conector M23 de 19 pines.

Figura 3: Ilustración de conectores en el TLM20, referente a los buses de poder y control respectivamente.

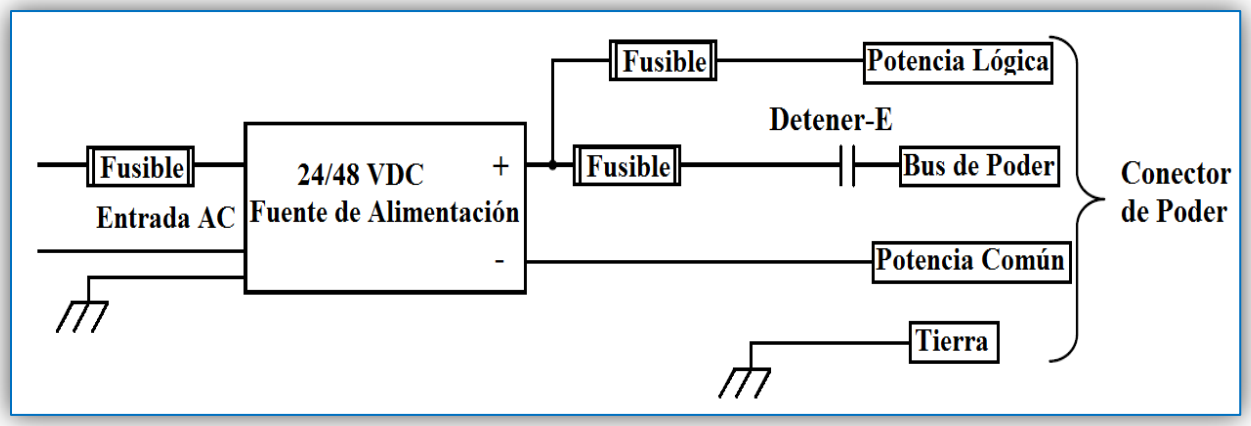

Figura 4: Ilustración de la conexión para el suministro al bus de poder.

1.4.2 Configuración del Actuador

Para la configuración en general del actuador Exlar *TLM20*, se pudiera emplear el software *EXPERT*; el cual constituye la herramienta predeterminada por el fabricante para este propósito.

En esta tesis, la configuración del actuador se basa en el estándar Modbus RTU apoyado en la interfaz serie RS-485, para establecer la comunicación con el dispositivo. Posteriormente, se establecen las unidades de medida para las distintas variables de la actuación, así como el desplazamiento lineal por revolución del tornillo (*lead*), de acuerdo al modelo de actuador. Seguidamente, deben configurarse los modos de trabajo en correspondencia con los requerimientos y diseño de la aplicación a controlar. En función de esto, se habilitan las entradas/salidas; las cuales se calibran para el caso de las analógicas, mientras que por su parte para las digitales se establece la asignación de funciones. Se deben habilitar las distintas faltas, ante condiciones de peligro o precaución para el dispositivo; así como parametrizar las diferentes variables de actuación consideradas dentro de la aplicación.

Para hacer válida la configuración realizada, se debe cargar en la memoria no-volátil del actuador; donde es guardad hasta una nueva escritura. Se permite también, salvar la configuración actual del dispositivo en el *PC*.

1.5 Arquitectura de Hardware

Una red de comunicación Modbus, generalmente, consta de dos partes principales: un dispositivo maestro, normalmente implementado en un PC (Computador Personal), el cual utiliza un programa de aplicación para llevar a cabo sus funciones, y un dispositivo esclavo, el cual normalmente es un dispositivo de campo, tales como actuadores y sensores.

En el caso de esta tesis, el dispositivo esclavo es el actuador de la serie Exlar TLMXX, el cual cuenta con la interfaz RS-485, la misma permite realizar las operaciones de señales de mando sobre el dispositivo esclavo y proporcionarle las informaciones obtenidas al dispositivo maestro a través del adaptador RS-422/485 USB ULinx del modelo USOPTL4.

1.5.1 Convertidor de Comunicación ULinx USOPTL4

Soporta a 2 hilos RS-485 o un sistema de comunicaciones RS-422/485 a 4 hilos, este dispositivo es ideal para cualquier aplicación que requiera de largo alcance o las capacidades multipunto. El modelo USOPTL4 utiliza bloques de terminales conectados en el lado RS-422/485 y tiene un par de LEDs que indican los datos que se transmiten o reciben. Modelo USOPTL4 incluye circuitos especiales que añade 2.000 voltios de aislamiento de protección contra los lazos de tierra y voltaje pico, y se alimenta desde el puerto USB lo que no requiere fuentes de alimentación(ELECTRONICS, 2006).

Características:

- Aislamiento óptico 2000 V RMS
- Protección contra sobretensiones 15KV ESD
- Añade un puerto COM de su PC
- Protege contra sobretensiones, picos y lazos de tierra
- LED para la transmisión y recepción de líneas
- USB 1.0, 1.1 y 2.0 compatibles (12 Mbps)
- Configuración automática en Windows 98, ME, 2000, XP, Vista
- No necesita fuente de alimentación (alimentado desde el bus USB)
- Incluye cable USB de 1 metro

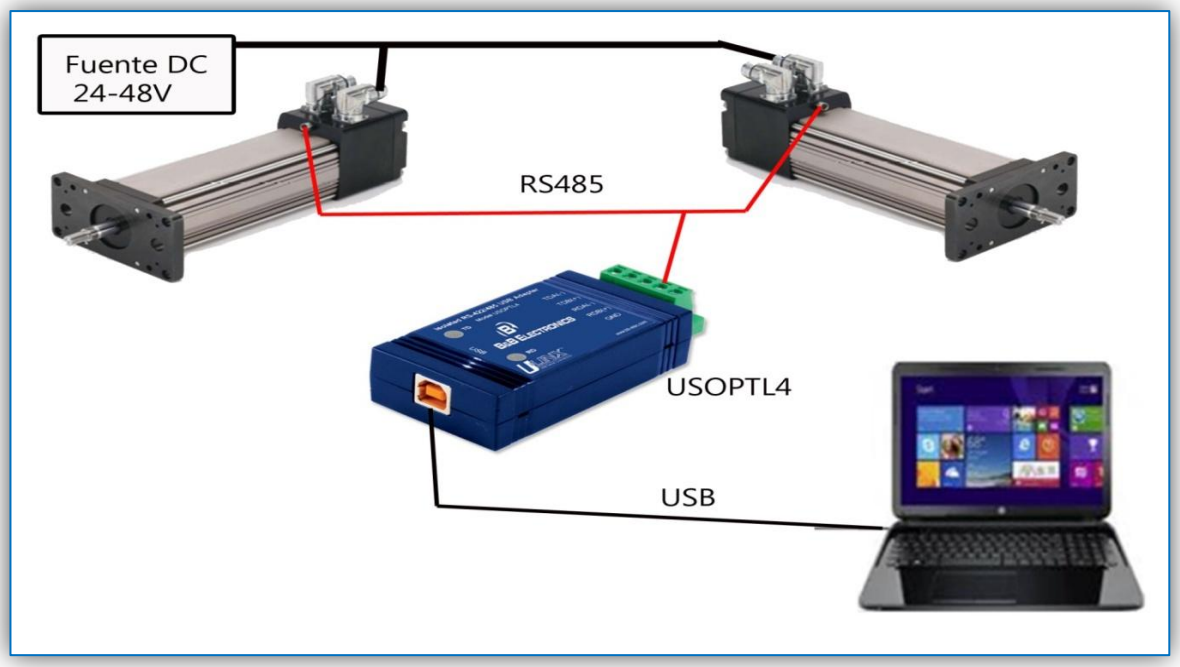

Figura 5: Red de comunicación Modbus RTU

1.6 Consideraciones finales del capítulo

En este capítulo se reportan los fundamentos de redes industriales en sistemas de comunicaciones, los buses de campos y sus variantes más importantes, de estas se concluye que el Modbus representa la solución a seguir debido a que es el bus implementado en la serie de actuadores TLMXX de la Exlar, este en su variante RTU.

Se realiza una clase de software para implementar la comunicación en el lenguaje C++ para garantizar su escalabilidad casi todos los dispositivos empotrados utilizados en el GARP.

Esta solución es capaz de mejorar la calidad de la configuración y mando de señales con el dispositivo y a su vez lo simplifica utilizando el estándar RS-485 donde todos los actuadores que sean necesarios en la futura aplicación podrán ser conectados a la red de 2 hilos de transmisión.

CAPÍTULO 2. MATERIALES Y MÉTODOS

En este capítulo se abordan en detalle los elementos y herramientas necesarias para resolver el problema de nuestra investigación. A partir de un conocimiento esencial sobre los diferentes tipos y formas que presenta el protocolo Modbus, así como los métodos de programación a abordar para encapsular los mismos en funciones de una clase que permita la realización de los objetivos propuestos.

2.1 Descripción del Protocolo MODBUS

Este epígrafe describe el método para la implementación de Tritex basando en el protocolo de comunicación Modbus RTU para la transferencia de datos entre un anfitrión serie y una unidad Exlar.

Modbus es un protocolo de tipo Petición/Respuesta, por lo que en una transacción de datos se puede identificar al dispositivo que realiza una petición como el maestro, y al que devuelve la respuesta como el esclavo de la comunicación. En una red Modbus se dispone de un equipo maestro que puede acceder a varios equipos esclavos. Cada esclavo de la red se identifica con una dirección única de dispositivo.

Un maestro puede hacer dos tipos de peticiones a un esclavo: para enviar datos a un esclavo y esperar su respuesta o confirmación; o para pedir datos a un esclavo y esperar su respuesta. Las peticiones de lectura y escritura que envía un maestro llevan asociado un código de función que el esclavo debe ejecutar. Según ese código, el esclavo interpreta los datos recibidos del maestro y decide qué datos debe devolver. Los códigos de función dependen de los dispositivos y de las tareas.

2.2 Modo de Transmisión Serie del Protocolo Modbus

El protocolo Modbus puede ser implementado en una variedad de plataformas de redes de hardware. Con este Trabajo se trata sólo con la implementación de Tritex sobre una línea serie mediante el Modbus RTU(Remote Terminal Unit) del modo de transmisión ya que la implementación del modo ASCII (que codifica cada byte como dos caracteres ASCII de siete bits) no es compatible con este dispositivo(Modbus, 2009).

2.2.1 Modo de transmisión Modbus RTU

Cuando los actuadores son configurados para comunicar en una red Modbus usando el modo RTU, cada byte en un mensaje contiene un conjunto dos dígitos hexadecimales de 4 bits. Cada mensaje debe ser transmitido en un flujo continuo.

El formato para cada byte en modo RTU es:

Sistema de codificación	Binario 8-bits, hexadecimal 0-9, A-F.
	Dos dígitos hexadecimales contenidos en cada campo de 8 bits del mensaje.
Bits por byte	1 bit de arranque
	8 bits de datos, el menos significativo se envía primero
	1 bit para paridad Par o Impar; ningún bit para No paridad.
	1 bit de paro si se usa paridad; 2 bits si no se usa paridad.
Campo de Comprobación de Error	Comprobación de Redundancia Cíclica (CRC).

2.2.2 Trama del mensaje Modbus RTU

Un mensaje Modbus es situado por el dispositivo que transmite, en una trama que tiene un comienzo y un final conocidos. Esto permite a los dispositivos receptores comenzar en el inicio del mensaje, leer la parte de la dirección y determinar qué dispositivo es solicitado (o todos los dispositivos si es una difusión 'dirección = 0') y conocer cuándo se ha completado el mensaje. Los mensajes parciales pueden ser detectados y establecer errores como resultado.

En modo RTU, los mensajes comienzan con un intervalo silencioso de al menos 3.5 veces el tiempo de un carácter. Esto es fácilmente implementado como un múltiplo de tiempos de

carácter a la velocidad de transmisión configurada en la red (ver Figura6). El primer campo transmitido es entonces la dirección del dispositivo destinatario.

Los caracteres a transmitir permitidos para todos los campos son 0-9, A-F hexadecimal. Los dispositivos conectados en la red revisan el bus de red continuamente incluso durante los intervalos 'silenciosos'. Cuando el primer campo (el campo de dirección) es recibido, cada dispositivo lo decodifica para enterarse si es el dispositivo requerido.

Siguiendo al último carácter transmitido, un intervalo de al menos 3.5 veces el tiempo de un carácter, señala el final del mensaje. Un nuevo mensaje puede comenzar después de transcurrido este intervalo.

La trama completa del mensaje debe ser transmitida como un flujo continuo, si un intervalo silencioso de más de 1.5 veces el tiempo de un carácter tiene lugar antes de completar la trama, el dispositivo receptor desecha el mensaje incompleto y asume que el próximo byte es el campo de dirección de un nuevo mensaje.

De forma similar, si un nuevo mensaje comienza antes de que transcurran 3.5 veces el tiempo de un carácter después de un mensaje previo, el dispositivo receptor lo considera una continuación del mensaje previo. Esto da lugar a un error, ya que el valor en el campo final CRC no es válido para el mensaje combinado(Modbus, 2009).

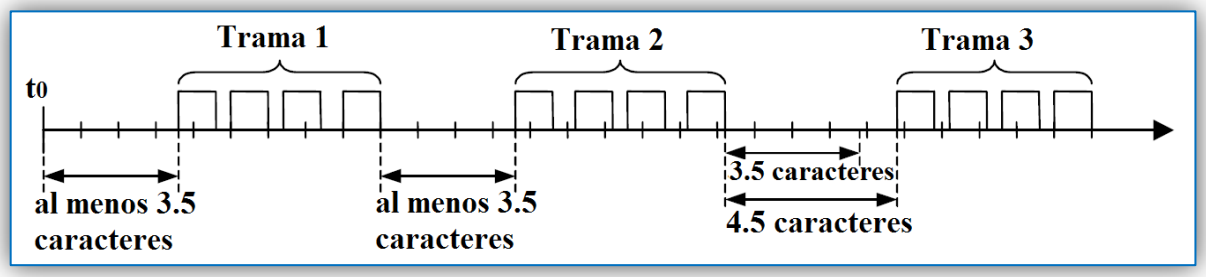

Figura6: Trama del mensaje RTU

Arranque	Identificador del Dispositivo	Código de Función	Datos	CRC	Final
3.5 caracteres	1 Byte	1 Byte	N Bytes	2 Bytes	3.5 caracteres

Figura7: Trama del mensaje Modbus RTU típica

2.2.2.1 Cómo es manipulado el campo de identificador del dispositivo

El campo de identificador del dispositivo de un mensaje contiene ocho bits. La dirección del esclavo válida está en el rango de 0– 247 en sistema decimal. Los dispositivos esclavos individuales tienen direcciones asignadas en el rango 1–247. Un maestro se comunica con un esclavo situando la dirección del mismo en el campo de dirección del mensaje. Cuando el esclavo envía su respuesta, sitúa su propia dirección en el campo de dirección de la respuesta para dar a conocer al maestro qué esclavo está respondiendo.

La dirección 0 es utilizada como dirección de difusión, la cual todos los dispositivos esclavos reconocen. Cuando el protocolo Modbus es usado en redes de nivel más alto, las difusiones pueden no estar permitidas o pueden ser reemplazadas por otros métodos.

2.2.2.2 Cómo es manipulado el campo de función

El campo de código de función de una trama de mensaje contiene ocho bits. Los códigos válidos están en el rango de 1–255. De los cuales, solo algunos códigos pueden ser utilizados.

Cuando un mensaje es enviado desde un maestro a un dispositivo esclavo, el campo de código de función indica al esclavo qué tipo de acción ha de ejecutar. Por ejemplo: Lectura de los estados ON/OFF de un grupo de bobinas o entradas discretas; lectura de los datos contenidos en un grupo de registros; lectura del estado de diagnóstico de un esclavo; escribir en determinadas bobinas o registros; o permitir cargar, salvar o verificar el programa dentro del esclavo(MODBUS.Org, 2002).

Cuando el esclavo responde al maestro, utiliza el campo de código de función para indicar, ya sea, una respuesta normal (libre de error) o que algún tipo de error ha tenido lugar (denominado respuesta de excepción). Para una respuesta normal, el esclavo simplemente replica el código de función original. Para una respuesta de excepción, el esclavo devuelve un código que es equivalente al código de función original con su bit más significativo (MSB) puesto en 1.

Además de la modificación del código de función para una respuesta de excepción, el esclavo sitúa un único código en el campo de datos del mensaje respuesta. Esto indica al maestro qué tipo de error ha tenido lugar, o la razón para la excepción.

El programa de aplicación del maestro tiene la responsabilidad de manejar las respuestas de excepción. Los procedimientos típicos son: enviar subsiguientes reintentos de mensaje, intentar mensajes de diagnóstico al esclavo y notificar operadores.

2.2.2.3 Contenido del Campo de Datos

El campo de datos se construye utilizando conjuntos de 2 dígitos hexadecimales, en el rango de 00 a FF hexadecimal.

El campo de datos de los mensajes enviados desde un maestro a un esclavo, contiene información adicional que el esclavo debe usar para tomar la acción definida por el código de función. Esto puede incluir partes como direcciones discretas y de registros, la cantidad de partes que han de ser manipuladas y la cantidad de bytes de datos contenidos en el campo.

Si no ocurre error, el campo de datos de una respuesta desde un esclavo al maestro, contiene los datos solicitados. Si ocurre un error, éste campo contiene un código de excepción que la aplicación del maestro puede utilizar para determinar la próxima acción a tomar. El campo de datos puede ser inexistente (de longitud cero) en ciertos tipos de mensajes.(MODBUS.Org, 2002).

2.2.2.4 Contenido del Campo Comprobación de Error

En modo RTU, los mensajes incluyen un campo de comprobación de error que está basado en un método Comprobación de Redundancia Cíclica (CRC). El Campo de Comprobación de Error contiene un valor de 16 bits implementado como dos bytes de 8 bits. El valor de comprobación de error es el resultado de un cálculo de Comprobación de Redundancia Cíclica (CRC), realizado sobre el contenido del mensaje. El campo CRC es añadido al mensaje como último campo del mensaje. La forma de hacerlo es, añadir primero el byte de orden bajo del campo, seguido del byte de orden alto. El byte de orden alto del CRC es el último byte a enviar en el mensaje(Modbus, 2009).

2.3 Método de Comprobación de Error CRC

En este trabajo se explica únicamente el método de comprobación de error CRC (Comprobación de Redundancia Cíclica), puesto que tanto el dispositivo esclavo a

comunicar utilizando Modbus RTU solo soporta para el mismo. Las comprobaciones de carácter y de trama del mensaje son generadas en el dispositivo maestro y aplicadas a los contenidos del mensaje antes de la transmisión. El dispositivo esclavo comprueba cada carácter y la trama del mensaje completo durante la recepción.

El maestro es configurado por el usuario para aguardar durante un tiempo de espera predeterminado antes de abortar la transacción. Este intervalo es establecido para ser lo suficientemente largo para que cualquier esclavo responda normalmente. Si el esclavo detecta un error de transmisión, el mensaje no es tenido en cuenta. El esclavo no construya una respuesta para el maestro. Así el tiempo de espera expira y permite al programa del maestro tratar el error. Observe que un mensaje enviado a un dispositivo esclavo inexistente también causará un error de tiempo excedido *time out*.

El campo CRC controla el contenido del mensaje completo. Se aplica con independencia de cualquier método de control de paridad utilizado para los caracteres individuales del mensaje. El valor CRC es calculado por el dispositivo emisor, que añade el CRC al mensaje. El dispositivo receptor calcula el CRC durante la recepción del mensaje y compara el valor calculado con el valor recibido en el campo CRC. Si los dos valores no son iguales, resulta un error.

Un dispositivo esclavo no puede responder a una solicitud, incluso si parece abordarse adecuadamente y formateado de otro modo, cuando se ha detectado un error de CRC, ya que no puede contar con el estado de cualquiera de los datos en la transacción. En lugar de ello, el maestro eventualmente debe agotar el tiempo de espera para una respuesta y puede volver a enviar la solicitud si se desea.

Para calcular el valor CRC se precarga un registro de 16 bits, con cada uno de los bits puestos en 1. Luego comienza un proceso que toma los sucesivos bytes del mensaje y los opera con el contenido del registro y actualiza éste con el resultado obtenido. Sólo los 8 bits de dato de cada carácter son utilizados para generar el CRC.

Los bits de arranque y paro y el bit de paridad, no se tienen en cuenta para el CRC. Durante la generación del CRC, se efectúa una operación booleana OR exclusivo (XOR) a cada carácter de 8 bits con el contenido del registro. Entonces al resultado se le aplica un desplazamiento de bit en la dirección de bit menos significativo (LSB), rellenando la

posición del bit más significativo (MSB) con un cero. El LSB es extraído y examinado. Si el LSB extraído fuese un 1, se realiza un XOR entre el registro y un valor fijo preestablecido3. Si el LSB fuese un 0, no se efectúa un el XOR.

Este proceso es repetido hasta haber cumplido 8 desplazamientos. Después del último desplazamiento (el octavo), el próximo byte es operado XOR con el valor actual del registro y el proceso se repite con ocho desplazamientos más, como se ha descrito más arriba y así con todos los bytes del mensaje. El contenido final del registro, después de que todos los bytes del mensaje han sido procesados, es el valor del CRC. Cuando el CRC es añadido al mensaje, primero se añade el byte de orden bajo seguido del byte de orden alto(OMEGA, 2005).

Procedimientos para la Generación de CRC:

1) Cargar un registro de 16 bits que denominaremos registro CRC, con FFFF (todos 1).
2) XOR del primer byte - 8 bits - del mensaje con el byte de orden bajo del registro CRC de 16 bits, colocando el resultado en el registro CRC.
3) Desplazar el registro CRC un bit a la derecha (hacia el LSB "bit menos significativo") rellenando con un cero el MSB (bit más significativo). Extraer y examinar el LSB.
4) (Si el LSB era 0): Repetir paso 3 (otro desplazamiento). (Si el LSB era 1): Hacer XOR entre el registro CRC y el valor polinómico A001hex (1010 0000 0000 0001).
5) Repetir los pasos 3 y 4 hasta que se hayan efectuado 8 desplazamientos. Una vez hecho esto, se haya procesado un byte completo – 8 bits.
6) Repetir los pasos 2 al 5 para el próximo byte – 8 bits – del mensaje. Continuar haciendo esto hasta que todos los bytes hayan sido procesados.
7) El contenido final del registro CRC es el valor CRC.
8) Cuando el CRC es situado en el mensaje, sus bytes de orden alto y bajo han de ser permutados(OMEGA, 2005).

2.3.1 Colocación de la CRC en el mensaje

Cuando el CRC de 16 bits (dos bytes de 8 bits) se transmite en el mensaje, el byte de orden inferior se transmite en primer lugar, seguido por el byte de orden superior. Por ejemplo, si el valor de CRC es 1241 hexadecimal (0001 0010 0100 0001):

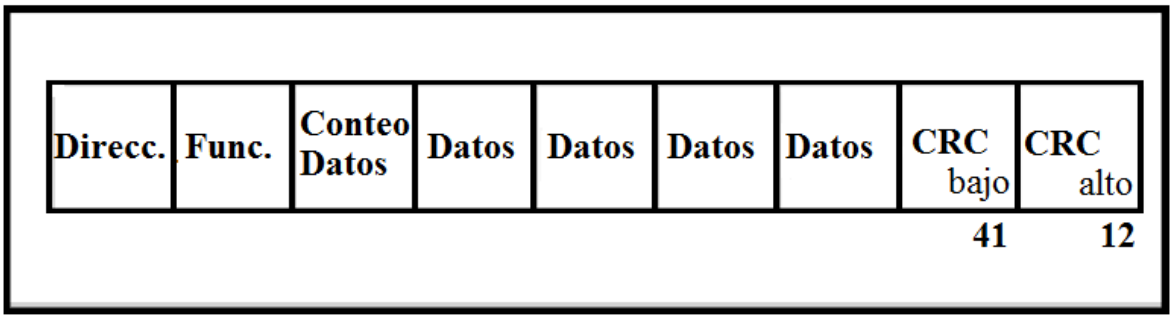

Figura 8: Secuencia de Byte CRC

2.3.2 Tiempo de espera del maestro

El maestro debe implementar un tiempo máximo (tiempo de espera) que está dispuesto a esperar una respuesta, lo que le permite recuperarse cuando una respuesta no sea remitida por la unidad. Tenga en cuenta que un tiempo de espera también permite que se recupere de una petición a un dispositivo esclavo no existente(Modbus, 2009).

2.3.3 Razón de Transmisión

La implementación Tritex establece 19.2K baudios (bits / seg) como la tasa de señal en serie por defecto especificada por la especificación de Modbus. Otros tipos estándar disponibles incluyen 4800, 9600, y 38,4 K baudios. El tiempo requerido para transmitir un carácter en serie depende de la velocidad de transmisión y puede ser calculado como(Modbus, 2009):

Tiempo de caracteres (seg / carácter) = (11 bits / carácter) / (bits de transmisión / seg)

Utilizando la expresión anterior llegamos a la conclusión de que para nuestra frecuencia 19200b/seg, el tiempo de carácter estimado es de 0.5729 mseg/carácter. Esta fórmula constituye un aproximado teórico porque no considera los retardos del sistema ni los espacios entre trama.

2.4 Códigos de Función MODBUS Implementados para Tritex

2.4.1 Códigos de Función Pública

El Protocolo Modbus define los códigos de función públicamente documentado varios estándares cuyo formato y el uso deben ser conformes a la norma existente. Los siguientes códigos de función pública están soportados en la implementación Tritex(Modbus, 2009).

- 03 (0x03) - Leer registros.
- 04 (0x04) - Leer registros de entrada.
- 06 (0x06) - Escribir Registro Único.
- 16 (0x10) - Escribir registros múltiples.
- 17 (0x11) - Informe Esclavo ID.

2.4.2 Códigos de funciones personalizadas

Los códigos de funciones personalizadas (definidas por el usuario) dentro de la especificación Modbus deben estar en el rango de 65 a 72 o de 100 a 110 decimal. La implementación Tritex utiliza los siguientes códigos de función personalizados para hacer frente a los datos de 32 bits.

- 103 (0x67) – Leer registros de lectura y escritura de 32 bits.
- 104 (0x68) - Leer registros de entrada de 32 bits.
- 106 (0x6A) - Escribir registros de lectura y escritura de 32 bits.

Los códigos de función de 32 bits permiten al usuario una alternativa a la lectura o escritura múltiple 16 - Registros de bits y esperan o devuelven el valor de 32 bits en Modbus estándar del Big Endian (palabra alta: palabra baja) en formato. Aunque cada registro de 32 bits puede ser leído o escrito mediante el estándar de lectura / escritura de múltiples códigos de función de registro, los códigos de 32 bits personalizados garantizan que el valor completo registro de 32 bits se lee o escribe en una sola operación no interrumpible (atómica), garantizando la integridad de datos. Sin esta garantía, los procesos asíncronos en la unidad podrían actuar en un valor parcialmente escrito o sobrescribir un valor que se leyó solo parcialmente.

2.4.3 Códigos de Función no Admitidos

Códigos de función pública Modbus estándar que se ocupan de la lectura y la escritura de bits individuales o múltiples (entradas y bobinas discretas) no son compatibles. Todos los datos se trata en cualquiera de 16 bits (palabra) o 32 bits (palabra doble) registran cantidades.

2.4.4 Descripciones de los Códigos de función

Esta sección se describe el formato tanto de la petición como de la respuesta por parte de la operación Modbus para todos los códigos de función soportados. También se enumeran los códigos de excepción que pueden ser devueltas en una respuesta de error a la solicitud. Todas las descripciones muestran sólo el formato de la trama, el ID de Modbus y CRC. Se da una respuesta de error especificando un código de excepción Función Ilegal (01) para cualquier (con el formato correcto) solicitud recibida por la unidad que contiene un código de función que no sea uno de los enumerados en esta sección(Modbus, 2009).

2.4.4.1 Código de función 03 (0x03) Leer Registros de 16 bits

Este código de función lee un bloque contiguo de registros de 16 bits. La solicitud de la trama especifica la dirección del registro de inicio y el número de registros para leer. La respuesta devuelve los valores de registro para llevar como dos bytes por registro - el primer byte contiene los bits de orden superior y el segundo byte contiene los bits de orden inferior.

Petición		
Código de función	1 byte	0x03
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros	2 bytes	1 a 60
Respuesta		
Código de función	1 byte	0x03
Byte a contar	1 byte	2xN*

Valor(es) del registro	2xN*bytes	Valor(es)
***N=número de registros		
Respuesta de Excepción		
Código de función	1 byte	0x83
Código de excepción	1 byte	01,02,03,04

2.4.4.2 Código de función 04 (0x04) Leer Registros de Entrada de 16 bits

Este código de función lee un bloque contiguo de registros de entrada de 16 bits. La solicitud de la trama, especifica la dirección del registro de inicio y el número de registros para leer. La respuesta devuelve los valores de registro para llevar como dos bytes por registro - el primer byte contiene los bits de orden superior y el segundo byte contiene los bits de orden inferior.

Petición		
Código de función	1 byte	0x04
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros	2 bytes	1 a 60
Respuesta		
Código de función	1 byte	0x04
Byte a contar	1 byte	2xN*
Valor del registro	2xN*bytes	Valor(es)
*N=número de registros		
Respuesta de Excepción		

Código de función	1 byte	0x84
Código de excepción	1 byte	01,02,03,04

2.4.4.3 Código de función 06 (0x06) Escribir Registro Único

Este código de función escribe un único registro de retención de 16 bits. La solicitud de la trama especifica la dirección del registro a ser escrito y el valor de 16 bits para escribir en el registro de envasado en dos bytes con el primer byte contiene los bits de orden superior y el segundo byte contiene los bits de orden inferior. La respuesta es un eco de la petición.

Petición		
Código de función	1 byte	0x06
Dirección del registro	2 bytes	0x0000 a 0xFFFF
Valor del registro	2 bytes	0x0000 a 0xFFFF
Respuesta		
Código de función	1 byte	0x03
Dirección del registro	2 bytes	0x0000 a 0xFFFF
Valor del registro	2 bytes	0x0000 a 0xFFFF
Respuesta de Excepción		
Código de función	1 byte	0x86
Código de excepción	1 byte	01,02,03,04

2.4.4.4 Código de función 16 (0x10) Escribir Registros Múltiples

Este código de función escribe un bloque contiguo de registros de 16 bits. La solicitud de la trama especifica la dirección del registro de salida, el número de registros a escribir, el número de bytes de valores de datos en la solicitud, y la lista de valores de datos para

escribir embalado como dos bytes por registro con el primer byte contiene los bits de alto orden y el segundo byte contiene los bits de bajo orden. La respuesta se hace eco de la dirección de partida y el número de registros escritos (pero no devuelve el número de bytes y los datos grabados).

Petición		
Código de función	1 byte	0x10
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros	2 bytes	1 a 60
Byte a contar	1 byte	2xN*
Valor(es) del registro	2xN* bytes	Valor(es)
***N=**número de registros		
Respuesta		
Código de función	1 byte	0x10
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros	2 bytes	1 a 60
Respuesta de Excepción		
Código de función	1 byte	0x90
Código de excepción	1 byte	01,02,03,04

2.4.4.5 Código de función 17 (0x11) Reportar ID del Esclavo

Este código de función se puede utilizar para la unidad. La solicitud de la trama tiene un valor nulo (longitud cero) campo de datos. La respuesta devuelve el tipo de unidad y el nombre de 16 caracteres ASCII.

Petición		
Código de función	1 byte	0x11
Respuesta		
Código de función	1 byte	0x11
Byte a contar	1 byte	0x12
Tipo de Unidad	1 byte	*
Nombre de la Unidad	16 bytes	0x20 a 0x7F
Indicador de estado de ejecución	1 byte	0xFF (ON)
*0=EM20, 1=EM30		
Respuesta de Excepción		
Código de función	1 byte	0x91
Código de excepción	1 byte	01,04

2.4.4.6 Código de la función 103(0x67) Leer registros de lectura y escritura de 32 bits

Este código de función personalizada lee un bloque contiguo de registros de salida de 32 bits. La solicitud de la trama especifica la dirección de registro de partida y el número de registros de 32 bits para leer. La respuesta devuelve cada uno valores de registro envasados en cuatro bytes con el primer byte que contiene el más significativo (de orden alto) de ocho bits, el segundo byte que contiene los siguientes ocho bits más significativos (es decir, el byte bajo de la palabra alta), el tercer byte contiene los siguiente ocho bits más significativos (es decir, el byte alto de la palabra baja), y el cuarto byte contiene los menos ocho bits significativos (es decir, el byte bajo de la palabra baja). Cada valor del registro de 32 bits leída en una operación atómica.

Petición		
Código de función	1 byte	0x67
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros de 32bits	2 bytes	1 a 30
Respuesta		
Código de función	1 byte	0x03
Byte a contar	1 byte	4xN*
Valor(es) del registro	4xN*bytes	Valor(es)
*N=número de registros de 32bits		
Respuesta de Excepción		
Código de función	1 byte	0xE7
Código de excepción	1 byte	01,02,03,04

2.4.4.7 Código de la función 104 (0x68) Leer registros de entrada de 32 bits

Este código de función personalizada lee un bloque contiguo de registros de entrada de 32 bits. La solicitud de la trama especifica la dirección de registro de partida y el número de registros de 32 bits para leer. La respuesta devuelve cada uno valores de registro envasados en cuatro bytes con el primer byte que contiene el más significativo (de orden alto) de ocho bits, el segundo byte que contiene los siguientes ocho bits más significativos (es decir, el byte bajo de la palabra alta), el tercer byte contiene los siguiente ocho bits más significativos (es decir, el byte alto de la palabra baja), y el cuarto byte contiene los menos ocho bits significativos (es decir, el byte bajo de la palabra baja). Cada valor del registro de 32 bits leída en una operación atómica.

Petición		
Código de función	1 byte	0x68
Dirección de arranque	2 bytes	0x0000 a 0xFFFF
Número de registros de 32bits	2 bytes	1 a 30
Respuesta		
Código de función	1 byte	0x03
Byte a contar	1 byte	4xN*
Valor(es) del registro	4xN*bytes	Valor(es)
*N=número de registros de 32bits		
Respuesta de Excepción		
Código de función	1 byte	0xE8
Código de excepción	1 byte	01,02,03,04

2.4.4.8 Código 106 Función (0x6A) Escribir registros de lectura y escritura de 32 bits

Este código de función personalizada, escribe un único registro de retención de 32 bits. La solicitud de la trama especifica la dirección del registro que se escribe. Los registros Modbus en la trama se abordan a partir de cero. La respuesta es un eco de la petición. Este código de función escribe un único registro de retención de 32 bits. La solicitud de la trama especifica la dirección del registro a ser escrito y el valor de 32 bits a escribir en el registro lleno en cuatro bytes con el primer byte contiene los más significativos (de orden superior) de ocho bits, el segundo byte contiene el siguiente más importante ocho bits (es decir, el byte bajo de la palabra alta), el tercer byte contiene los siguientes ocho bits más significativos (es decir, el byte alto de la palabra baja), y el cuarto byte contiene los menos

ocho bits significativos (es decir, el byte bajo de la palabra baja). La respuesta es un eco de la petición.

Tenga en cuenta que el código de función 16 (0x10) para escribir varios registros de 16 bits se puede utilizar (con un recuento de registro de dos) para leer un valor del registro de la celebración de 32-bit. La unidad, sin embargo, almacena números de 32 bits internamente en formato Little Endian con la palabra baja primero lo que el código de función 16 debe especificar la palabra baja (menos significativos de 16 bits) en primer lugar, seguido de la palabra alta (más significativos de 16 bits) para escribir el valor del registro de 32 bits. El uso de código de función (personalizado) 106 (0x6A) en vez ofrece las siguientes ventajas:

- Registrar el orden de bytes es el Modbus normal de mayor a menor
- La escritura de 32 bits se garantiza que sea atómica (es decir, una sola operación no interrumpible).
- La eficiencia de transmisión - el total de la transacción de petición / respuesta requiere catorce bytes en lugar de los quince requerido por el código de función 16.

Petición		
Código de función	1 byte	0x6A
Dirección del registro	2 bytes	0x00000000 a 0xFFFFFFFF
Valor del registro	4 bytes	0x00000000 a 0xFFFFFFFF
Respuesta		
Código de función	1 byte	0x6A
Dirección del registro	2 bytes	0x00000000 a 0xFFFFFFFF
Valor del registro	4 bytes	0x00000000 a 0xFFFFFFFF
Respuesta de Excepción		
Código de función	1 byte	0xEA

Código de excepción	1 byte	01,02,03,04

2.4.5 Respuesta de Excepción

El mensaje de una respuesta de excepción (error) del esclavo está formado por el código de función original con el bit alto establecido y un campo de datos de un solo byte (el código de excepción) que indica el tipo de error que se ha producido. La siguiente tabla indica los posibles códigos de excepción utilizados en la ejecución Tritex.

Código	Nombre	Significado
01	Función ilegal	El código de función recibido en la petición no es una acción permitida para el esclavo. Este error puede producirse cuando el código de función sólo se aplica a los dispositivos más nuevos y no se implementó en el dispositivo actual, cuando el código de función es desconocida, o cuando el esclavo está en un estado no válido para procesar el código de función especificada.
02	Dirección de datos ilegal	La dirección de datos especificado en la solicitud no es una dirección permitida para el esclavo. Más específicamente, la combinación de número de referencia y la longitud de transferencia no es válida.
03	Valor de datos ilegal	Un valor especificado en la solicitud no es un valor permisible para el esclavo, que indica un fallo en la estructura del resto de la solicitud. Este error no significa que un elemento de datos enviado para su almacenamiento en un registro tiene un valor fuera del rango esperado para el registro.
04	Fallo del dispositivo esclavo	Se ha producido un error irrecuperable mientras el esclavo intentaba realizar la solicitud.

2.5 Software de implementación de la comunicación

Para la implementación del software de mando y configuración se utiliza el Framework de **Qt**, el cual permite, dadas su característica multiplataforma migrar de sistema operativo sin mucha dificultad en la estructura del código.

Las clases fundamentales utilizadas fueron **QtserialPort y QtserialPortInfo,** las cuales controlan el acceso flujo y configuración de los puertos disponibles.

Con respecto a la clase diseñada, es preciso señalar que posee una clase **ModbusExlar** que contiene los métodos para la generación y temporización de cadena, selección del id del dispositivo y la confección de la trama, así como para el manejo de las excepciones y errores de la comunicación.

Estas clases implementan objetos jerárquicamente y mediante el mecanismo de signals y slots, se comunican para mandar datos mediante el puerto serie cada 100 ms.

Se creó una aplicación capaz de probar este conjunto de funciones y poder realizar el mando y la configuración del equipo aunque el objetivo de este trabajo está en entregar un módulo de funciones que permitan ser utilizadas por los programadores del GARP en los lugares que se requiera la utilización del dispositivo TLMXX de Exlar.

2.6 Consideraciones finales del capítulo

Para la implementación de una clase que cumpla con los requerimientos del estándar Modbus RTU es necesario contar con métodos de generación e interpretación de la trama establecida para este tipo de comunicaciones.

Es importante considerar el manejo de errores de la comunicación para poder salir de situaciones adversas que interrumpen la respuesta como los errores de CRC o los tiempos agotados de espera máxima o *timeouts.*

Se seleccionó el framework de Qt 5.2 para desarrollar el software por su elevada versatilidad y capacidad multiplataforma, además de que se puede realizar un código bajo los preceptos y regulaciones del software libre mediante el cual es posible comercializarlo posteriormente.

CAPÍTULO 3. RESULTADOS

Para el desarrollo de software es necesario su correcto modelado de acuerdo a los principios de la Ingeniería de Software y el Lenguaje Unificado de Modelado (*UML por sus siglas en inglés*). Es por esto que en el presente capítulo evaluaremos el desempeño e implementación de la clase proporcionada para poder obtener como resultados, además del software en sí, su documentación para los próximos programadores en utilizarla, así como la medición de algunas características de su desempeño.

3.1 Modelado del software

No es posible concebir un proyecto de software en la actualidad que no posea una fase de modelado seria y discusión con los usuarios finales del mismo. De estas sesiones es posible obtener un producto mejorado y acorde con las necesidades reales del cliente y refinar antes de la fase de desarrollo posibles errores de concepto del mismo.

El UML ha ido ganado en gran medida con el transcurso del tiempo una importancia clave para todo proyecto serio de software y sobre todo ha demostrado ser una fase necesaria en los procesos de diseño debido a que a partir de su utilización se ahorra tiempo y dinero, así como es posible realizar contratos de producción de software concretos con los clientes finales de la aplicación.

Para el diseño de nuestra clase y la aplicación respectiva que permite su implementación final, hemos obtenido en la fase de diseño los diagramas de Clase y de Casos de Uso para documentar de una mejor manera el funcionamiento del software desarrollado.

El diagrama de casos de uso representado en la figura9 ilustra las relaciones entre los actores y usos de la aplicación. Como se puede observar el sistema Modbus_Exlar _RTU es capaz de conformar la trama en el formato deseado y a su vez puede interpretar una cadena recibida y darle tratamiento a errores o incongruencias con el estándar en la misma. Para que el usuario interactúe con la clase diseñada es necesario hacerlo a través de dos interfaces gráficas que permiten mediante estrechas relaciones de trabajo la operatividad de la aplicación.

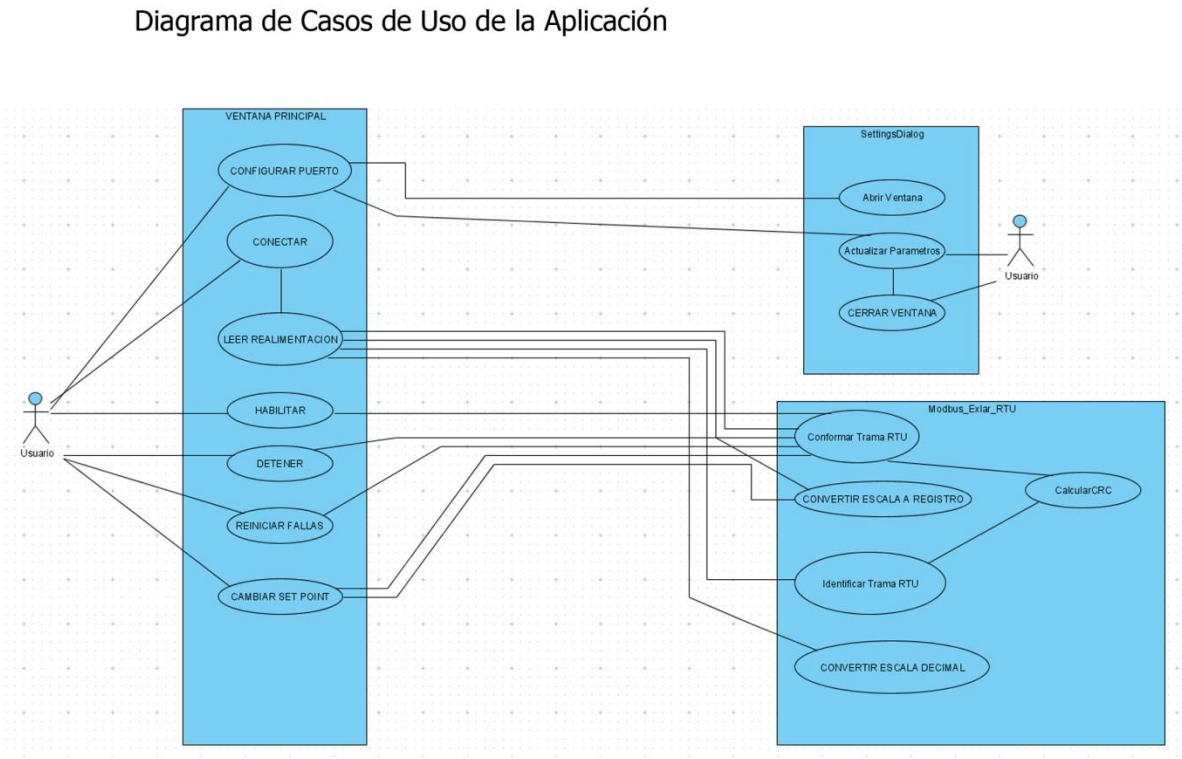

Figura9: Trama del mensaje Modbus RTU típica

Se evidencia que el funcionamiento lógico de este software indica que de esta dos interfaces la SettingsDialog representa un formulario de configuración del puerto serie, capaz de actualizar los parámetros del mismo en la comunicación implementada en la ventana principal. La misma cuenta con un conjunto de acciones como, conectar el puerto, habilitar el actuador, detener el movimiento, reiniciar las fallas, cambiar el set point del movimiento deseado y leer la realimentación de las variables del movimiento dedicado, el estatus del actuador y los registros de banderas de fallas en caso de haberlas.

Habiendo descrito el funcionamiento básico del sistema, es posible modelar su interacción entre clases como se muestra en la figura 10.

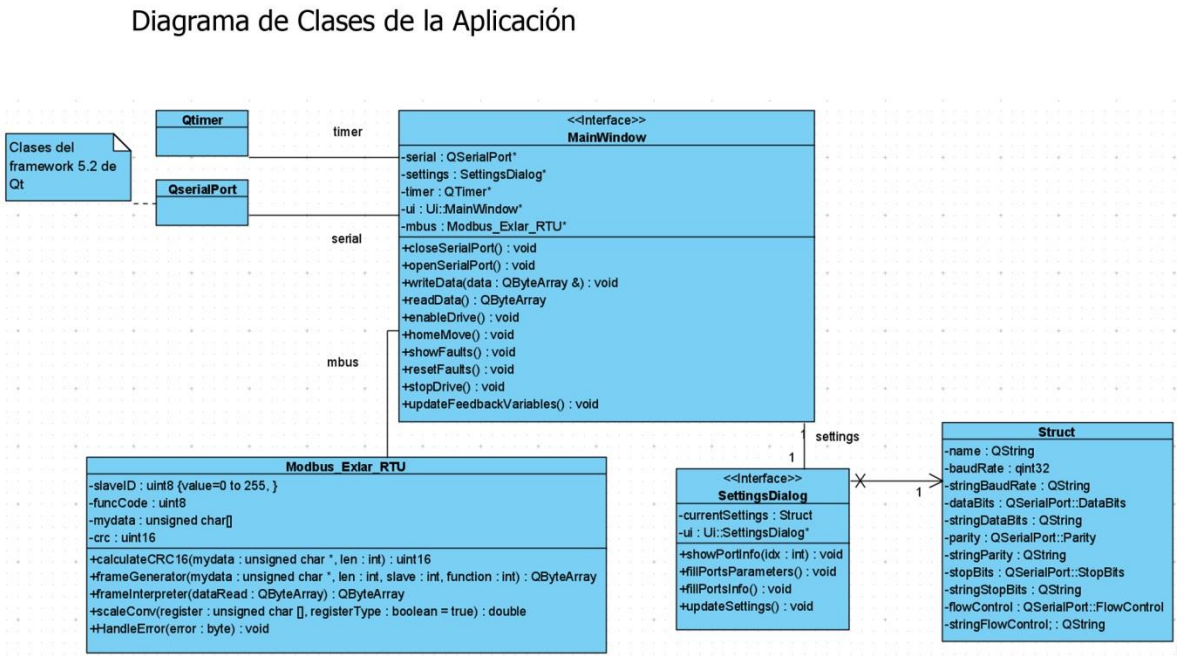

Figura10: Diagrama de Clases de la Aplicación de Prueba.

Nótese como la clase Modbus_Exlar_RTU posee un conjunto de atributos y operaciones (*métodos*) orientados a la generación e interpretación de las tramas del estándar Modbus RTU. Esta puede generar objetos, como es el caso de **mbus,** capaces de contener en su estado los campos fundamentales de dicho estándar y a su vez en su comportamiento conformar e interpretar las tramas de mensajes para la comunicación utilizando este bus de campo. Vale destacar que la función scaleConv es necesaria para la conversión de la escala brindada por el fabricante del dispositivo de los registros de 32 y 16 bits a números flotantes para la representación física de los parámetros del mismo, el algoritmo para dicha conversión es proporcionado en el Anexo I.

Esta clase interactúa con una interfaz que controla un conjunto reducido de funciones que son las que el cliente final necesita de este actuador. Estas funciones están basadas en la tabla de direcciones del Anexo I y mediante el objeto creado mbus, es posible acceder al dispositivo pasando el campo de datos afín a cada operación correspondiente en dicha tabla.

Todo el control de entrada salida por los puertos y temporización es manejado en esta clase mediante la implementación de métodos como:

- closeSerialPort.
- openSerialPort.
- readData.
- writeData.

3.2 Interfaz gráfica

La Figura 11 muestra la ventana principal de la interfaz gráfica. En ella se agruparon un conjunto de funciones básicas requeridas para la utilización de este actuador. Los tres controles del tipo dial en la izquierda, permiten al usuario ajustar el set point de uno de los modos de movimiento del actuador que es el Dedicated Move(Exlar, 2009). Conectados por la señal sliderReleased() indican que cuando se sueltan dichos controles se actualiza uno de los tres registros destinados al Dedicated Move en la posición proporcional al valor en que se liberan. Desde el momento que se conecta el puerto serie se comienza por la encuesta constante de los tres parámetros principales de realimentación, que luego son mostrados en el cuadro de controles Estatus. También se encuesta continuamente el registro de bandera destinado a las fallas del actuador, el cual es posible reiniciar en caso de que aparezca alguna mediante el botón Reiniciar Fallas.

El comando Habilitar mediante el botón del mismo nombre, constituye una acción necesaria para empezar a mover el dispositivo y que el modo de trabajo del actuador sea el por defecto, Activo(Exlar, 2009). También los comandos de Detener el movimiento y Parqueo son útiles (*STOP y HOME respectivamente*), debido a que mediante el primero podemos abortar cualquier movimiento y el segundo representa la posición de inicio de dicho dispositivo.

En el momento en que es presionado el botón Configurar Puerto, se abre la interfaz referente a la configuración del puerto serie mostrada en la Figura 12. La misma contiene un conjunto de configuraciones de los parámetros del puerto serie en general. Vale destacar que la configuración por defecto de la serie TLMXX de Exlar tiene un baudrate de 19200b/seg, una paridad Par, un bit de stop, 8 bits de datos y sin control de flujo.

Como se mostró en diagramas anteriores uno de los atributos de la clase implementada por esta ventana es una estructura que hace referencia a la clase de Qt: QSerialPortInfo que permite actualizar los parámetros del objeto serial de la clase QtSerialPort en el formulario principal.

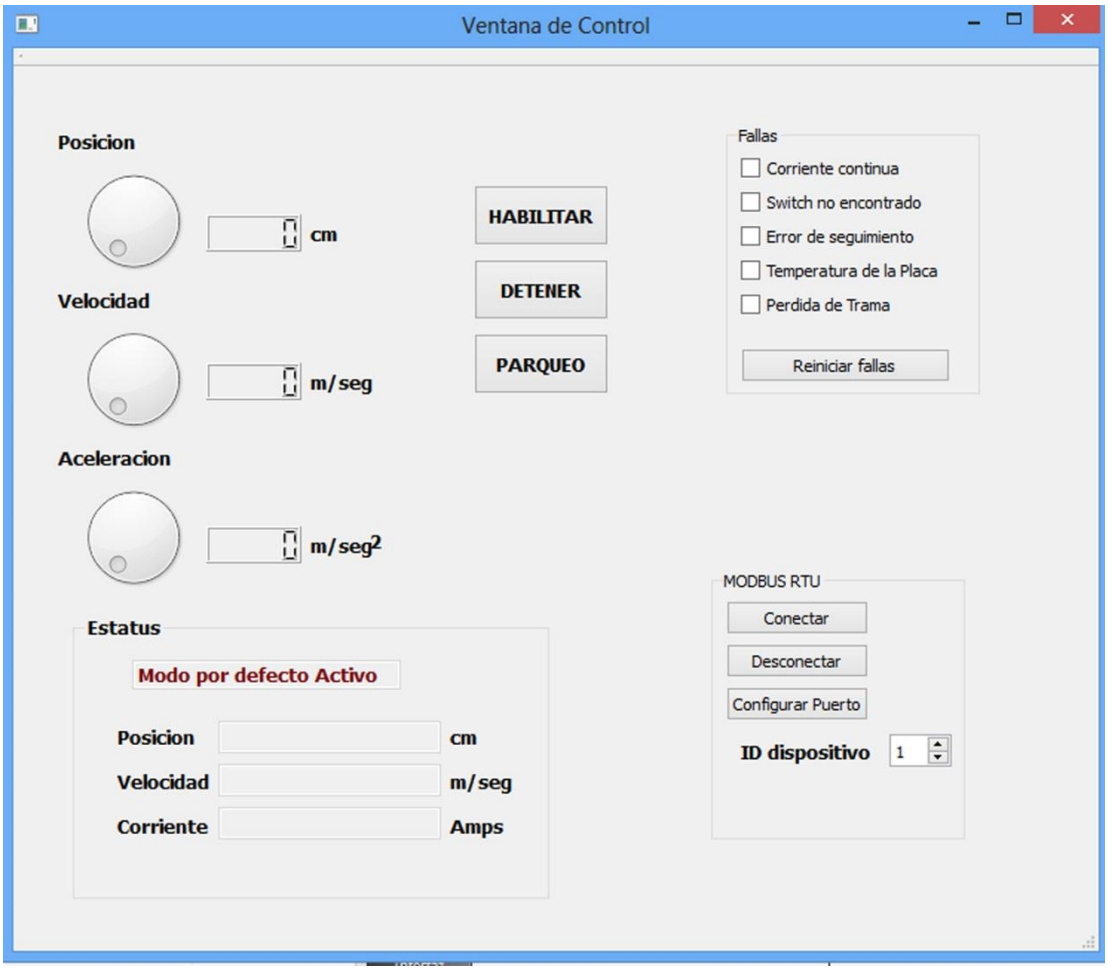

Figura11: Diagrama de Clases de la Aplicación de Prueba.

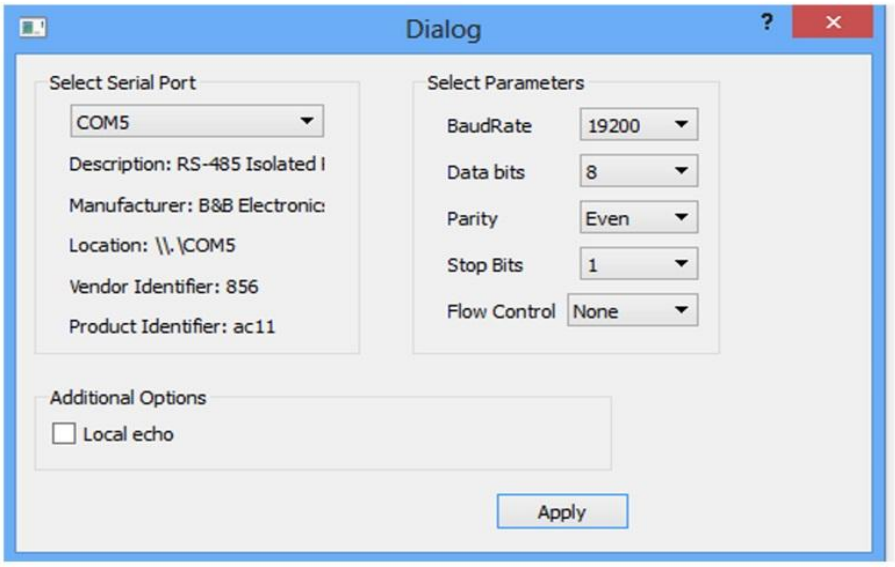

Figura12: Diagrama de Clases de la Aplicación de Prueba.

3.3 Pruebas realizadas al sistema

3.4 Pruebas realizadas a la aplicación.

La aplicación fue probada en dos experimentos por un conjunto de 36 y 24 horas ininterrumpidas y fue modificada para medir los siguientes parámetros.

- Cantidad de *timeouts* de la comunicación.
- Cantidad de errores de CRC.

Para esta prueba los resultados fueron los siguientes.

Tiempo Experimento	Cantidad de *timeouts*	Cantidad de errores CRC
Experimento 1: 36 h	18	7
Experimento 2: 24 h	10	5

Esto demuestra que para un envío de mensajes cada 100 ms la aplicación es robusta y es posible utilizar la clase Modbus_Exlar_RTU para la comunicación con el actuador de manera fiable sin perder datos debido a que para el gran volumen de información enviado

las pérdidas son aceptables. Este experimento se realizó en condiciones de laboratorio y no de en un ambiente industrial, por lo que pudieran incrementarse estos errores pero sería el objeto de un estudio posterior el análisis de la red de comunicaciones bajo estas condiciones, pero para los objetivos de esta investigación queda demostrada que el sistema de software es estable.

Por otra parte se realizó una comparación entre el canal analógico y el movimiento dedicado digital, la cual brindó interesantes resultados.

Se analizaron parámetros como la desviación estándar de las señales de mando digital mediante la red Modbus y la señal analógica proporcional durante un tiempo de 30 minutos. El software fue modificado para salvar la posición realimentada y el registro asociado a la posición dedicada como señal de mando en un archivo de texto para su posterior análisis en el software MatLab 7.8. A su vez se midió la señal de mando analógica efectuada por una tarjeta de adquisición de datos HUMMUSOFT MF624 utilizando también el Matlab, en este caso el paquete Real Time Window Target.

Tiempo Experimento	Desviación estándar mando analógico	Desviación estándar mando red Modbus
Experimento : 30 min	0.1 cm	0

Por otra parte el resultado de la realimentación de la posición en ambos casos quedó especificado de la manera siguiente.

Tiempo Experimento	Desviación estándar realimentación analógica	Desviación estándar realimentación red Modbus
Experimento : 30 min	0.169 cm	0.0563 cm

Esto significa que el sistema actual mejora la precisión de la salida de la variable en el actuador debido a que es posible eliminar el ruido en la señal de mando, por otra parte es posible evitar posibles inducciones en dicha señal que causen graves daños en el dispositivo

y permiten que el mismo no trabaje de manera continua para lograr alcanzar una señal indeseablemente cambiante en el tiempo que puede lograr aumentos en la temperatura y corriente del dispositivo.

3.5 Análisis económico

Con la presente investigación se propone una aplicación que permite al usuario ahorrar un tiempo precioso en la configuración de los actuadores lineales eléctricos. El mismo está orientado a la explotación de la serie de actuadores *TRITEX TLMXX* de *EXLAR Co*, y permite que el recurso humano en cuestión se capacite acerca del software profesional de una manera más eficiente.

Dentro de esta implementación es posible encontrar una clase Modbus RTU que permite realizar el mando y la configuración de dicho actuador, el cual posibilita a los usuarios la utilización de esta clase en futuras aplicaciones. Por lo cual con esta investigación, se ahorra paradas y mantenimientos del sistema, debido a que el actuador permite realizar todas las operaciones de manera on-line.

Esta investigación, se ahorra al país toda hora-hombre implicada en el estudio de la comunicación en Modbus de dichos actuadores mediante la síntesis de este conocimiento brindada en este informe. Por otra parte, la implementación de la configuración y mando de señales en el actuador TLMXX, facilita su futura inclusión en próximas aplicaciones de robótica en Cuba.

Con un costo aproximado de 4000.00 CUC por unidad, el TLM20 constituye un dispositivo único en la línea de actuación, al estar incluidos el pistón con el hardware de control y comunicación. Pero vale destacar que en el momento de la realización de esta investigación ya se encontraban importados todos los medios en ella utilizados como resultado de la necesidad de proyectos anteriores. Por lo que para evaluar los gastos realizados por la presente solo podemos destacar el tiempo de los recursos humanos para asimilar esta rama del conocimiento y desarrollar la aplicación en sí.

Con la implementación de una red Modbus se ahorra la importación de los cables de entradas análogo-digitales del dispositivo, debido a que mediante este bus de campo es

posible realizar todas las operaciones sin necesidad de adquirir los dichos cables que le cuestan al cliente alrededor de 300 CUC por unidad.

Queda concluido que el desarrollo y utilización de esta investigación, ahorra cuantiosos recursos a todos los técnicos e ingenieros que necesiten el uso de estas tecnologías.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones:

Como conclusiones finales de esta investigación podemos citar las siguientes:

1 La implementación de la clase Modbus _Exlar_RTU permite realizar el mando y la configuración de manera on-line de un dispositivo de la serie Exlar TLXX.

2 La aplicación desarrollada demuestra la mejora sobre la señal de mando y por ende sobre la presición del dispositivo en comparación con el mando analógico existente.

3 La presente investigacion, aparejada de todos los diagramas, resultados y codigos presentados permiten el uso en dispositivos empotrados utilizando el Modbus para la utilización de los actuadores de la serie Exlar TLMXX.

Recomendaciones:

Las siguientes recomendaciones que se realizan permiten ampliar las posibilidades de uso de estas tecnologías en futuras aplicaciones.

1 Realizar una interfaz que permita la aplicación de todas las características del mapa de direcciones del actuador Exlar TLMXX. Esto permite independizarse del software profesional y obtener datos importantes que el mismo no permite almacenar.

2 Añadir a la aplicación brindada la funcionalidad de graficar datos de manera on-line.

3 Utilizar la clase Modbus_Exlar_RTU en aplicaciones de control real de sistemas mediante el uso de dispositivos empotrados.

REFERENCIAS BIBLIOGRÁFICAS

2011. Protocolos de comunicaciones industriales. *Asociación de la Industria Eléctrica y Electrónica, Vicuña ,Chile*.

CEKIT 1999. Bus de Campo. *Electrónica & Computadores*.

CIA 2010. CAN Protocol.

CORREA, J. V. & MANJARREZ, N. C. 2005. *Diseño e Implementación de una Red Industrial bajo protocolo MODBUS que permita la comunicación digital con instrumentos de Campo.* Monografía del Trabajo de Grado Corporación Universitaria Rafael Núñez.

DHUMANE, S. A. 2010. Client-Server Data Communication between Computer and Intelligent Circuit Breaker using Modbus/RTU Protocol *Proceedings of IEEE Power Engineering Society General Meeting.*

DOMINGO, J. & CARO, J. 2003. Comunicaciones en el entorno industrial. *Pereira ,Colombia*.

ELECTRONICS, B. B. 2006. Isolated RS-422/485 USB ADAPTER Model USOPTL4.

EXLAR, T. 2009. Tritex Modbus ID Table.

GALLO, L. & HERRERA, D. 2009. Diseño e Implementación de una red industrial utilizando protocolo Modbus y comunicación inalámbrica con tecnología Allen Bradley para monitoreo y control local y remoto de las estaciones de nivel, flujo y presión en el laboratorio de redes industriales y control de procesos de la ESPE Extensión Latacunga. *Departamento de Eléctrica y Electrónica de la Universidad de Las Fuerzas Armadas ESPE Extensión Latacunga*.

GARCÍA, E. 2003. Automatización de procesos industriales.

MAK, S. & RADFORD, D. April 1996. Communication system requirements for implementation of distribution automation,. *IEEE Trans. Power Delivery*.

MODBUS-IDA 2006. MODBUS APPLICATION PROTOCOL SPECIFICATION. 1.1b.

MODBUS 2009. Tritex Modbus Protocol Specification.

MODBUS.ORG 2002. Modicon Modbus Protocol Reference Guide. *PI-MBUS-300 REV.J*.

OMEGA, E. 2005. Modbus RTU Protocol Communication Guide.

PAPASIDERIS, K. & LANDRY, C. 2009. Environment Temperature Control Using Modbus and RS485 Communication Standards.

T.AMY, L. 2008. *Automation Systems for Control and Data Acquisition,* Editorial Instruments Society of America.

TRITEX, E. 2009. Linear & Rotary Actuator Installation, Operation and Service Manual. *Exlar Corporation*.

VAZQUEZ, A. 2013. *Actuador lineal TRITEX TLM30 en aplicaciones de control.* Trabajo de diploma, UCLV.

ANEXOS

Anexo I Tabla ID del Actuador TLM20

MODBUS Variables Table

Scale X .Y indicates a binary fraction where...
X specifies the number of integer bits
Y specifies the number of fractional bits
value = integer part + (fractional part / 2^Y)

Scale Example
Scale = 11.5 AMPS X = 11, Y = 5
Value = 80
80 decimal = 0x50 = 0000 0000 0101 0000 B
integer part = 0000 0000 010 B = 2 decimal
fractional part = 10000 B = 16
value = 2 + 16/2^5 = 2+16/32 = 2.5 AMPS

Access
RO - READ ONLY (MODBUS INPUT REGISTER)
RW - READ/WRITE (MODBUS HOLDING REGISTER)
RPW - READ/PROTECTED-WRITE (MODBUS HOLDING REGISTER)

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
2	Firmware Version	UINT16	RO	FALSE		0.01 Version				
3	(reserved)	UINT16	RO	FALSE						
4	Disables	UINT16	RO	FALSE		DISABLING_SOURCES_ENUM				
5	Faults	UINT16	RO	FALSE		FAULTS_ENUM				
6	Hard Faults	UINT16	RO	FALSE		FAULTS_ENUM				
7	Soft Faults (warnings)	UINT16	RO	FALSE		FAULTS_ENUM				
8	Bus Voltage (Filtered)	UINT32	RO	FALSE	11.21	VOLTS				
9	Bus Voltage (Filtered) (H)	UINT16	RO	FALSE	11.5	VOLTS				
10	Board Temperature	UINT32	RO	FALSE	11.21	DEG C				
11	Board Temperature (H)	UINT16	RO	FALSE	11.5	DEG C				
12	Power	UINT32	RO	FALSE	11.21	AMPS^2				
13	Power (H)	UINT16	RO	FALSE	11.5	AMPS^2				
14	Position Tracking Error	INT16	RO	FALSE	1.15		-32768	32767		
15	Bus Voltage	UINT16	RO	FALSE	11.5	VOLTS				
100	Active Input Functions	UINT32	RO	FALSE		INPUT_FUNCTIONS_ENUM				
102	Active Output Functions	UINT32	RO	FALSE		OUTPUT_FUNCTIONS_ENUM				
104	Active Inputs	UINT16	RO	FALSE		INPUTS_ENUM				
105	Active Outputs	UINT16	RO	FALSE		OUTPUTS_ENUM				
106	H/W Inputs	UINT16	RO	FALSE		INPUTS_ENUM				
107	H/W Outputs	UINT16	RO	FALSE		OUTPUTS_ENUM				
200	Analog Input (raw)	UINT16	RO	FALSE	1.15	ADC	0	0x7FFF		
201	Analog Output (raw)	UINT16	RO	FALSE	1.15	PWM ON-TIME				
202	Analog Input	INT16	RO	FALSE	1.15	USER CAL				
203	Analog Input (calibrated)	INT16	RO	FALSE	6.10	VOLTS or mAmps			1.12	
204	Analog Output	INT32	RO	FALSE	1.31		0	0x7FFFFFFF		
206	Analog Position Target	INT32	RO	FALSE	16.16	REVS				
208	Analog Velocity Target	INT16	RO	FALSE	8.8	RPS				
209	Analog Current Target	INT16	RO	FALSE	9.7	AMPS				

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
300	Command Mode	UINT16	RO	FALSE		COMMAND_MODE_ENUM				
301	I Feedback (abs)	UINT16	RO	FALSE	9.7	AMPS				
302	I Continuous	UINT32	RO	FALSE	9.23	AMPS				
303	I Continuous (H)	UINT16	RO	FALSE	9.16	AMPS				
304	P command	INT32	RO	FALSE	16.16	REVS				
306	P feedback	INT32	RO	FALSE	16.16	REVS				
308	P error	INT32	RO	FALSE	16.16	REVS				
310	V command	INT32	RO	FALSE	8.24	RPS				
311	V command (H)	INT16	RO	FALSE	8.8	RPS				
312	V feedback	INT32	RO	FALSE	8.24	RPS				
313	V feedback (H)	INT16	RO	FALSE	8.8	RPS				
314	V error	UINT32	RO	FALSE	8.24	RPS				
315	V error (H)	INT16	RO	FALSE	8.8	RPS				
316	V display	INT32	RO	FALSE	8.24	RPS				
317	V display (H)	INT16	RO	FALSE	8.8	RPS				
318	I display	INT32	RO	FALSE	9.23	AMPS				
319	I display (H)	INT16	RO	FALSE	9.7	AMPS				
320	Pos Limit Minus	INT32	RO	FALSE	16.16	REVS			1.12	
322	Pos Limit Plus	INT32	RO	FALSE	16.16	REVS			1.12	
400	Scope Flags	UINT16	RO	FALSE		BITMAP				
401	Scope Timestamp	UINT16	RO	FALSE		uSec				
402	Scope Variable Data 1	INT32	RO	FALSE						
404	Scope Variable Data 2	INT32	RO	FALSE						
406	Scope Variable Data 3	INT32	RO	FALSE						
408	Scope Variable Data 4	INT32	RO	FALSE						
500	Hall Sine Min	INT16	RO	FALSE	1.15					
501	Hall Sine Max	INT16	RO	FALSE	1.15					
502	Hall Cosine Min	INT16	RO	FALSE	1.15					
503	Hall Cosine Max	INT16	RO	FALSE	1.15					
504	Psine In	INT16	RO	FALSE	1.15					
505	Pcosine In	INT16	RO	FALSE	1.15					
506	Psine Out	INT16	RO	FALSE	1.15					
507	Pcosine Out	INT16	RO	FALSE	1.15					
508	Paccel	INT32	RO	FALSE	1.31	REV/PWM_UPDATE^2				
510	Pvelocity	INT32	RO	FALSE	1.31	REV/PWM_UPDATE				
512	PrevAngle	UINT32	RO	FALSE	0.32	REV				
514	PeAngle	UINT16	RO	FALSE	0.16	ELEC ANGLE				
515	Electric Angle	UINT16	RO	FALSE	0.16	ELEC ANGLE				
516	E sine	INT16	RO	FALSE	1.15					
517	E cosine	INT16	RO	FALSE	1.15					
518	IR (raw)	INT16	RO	FALSE	1.15	ADC				
519	IS (raw)	INT16	RO	FALSE	1.15	ADC				
520	IT (raw)	INT16	RO	FALSE	1.15	ADC				
521	IR	INT16	RO	FALSE	9.7	AMPS				
522	IS	INT16	RO	FALSE	9.7	AMPS				
523	IT	INT16	RO	FALSE	9.7	AMPS				
524	IR (scaled)	INT16	RO	FALSE	9.7	AMPS				
525	IS (scaled)	INT16	RO	FALSE	9.7	AMPS				
526	IT (scaled)	INT16	RO	FALSE	9.7	AMPS				

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
527	(reserved)	UINT16	RO	FALSE						
528	ID command	INT16	RO	FALSE	9.7	AMPS				
529	IQ command	INT16	RO	FALSE	9.7	AMPS				
530	ID feedback	INT16	RO	FALSE	9.7	AMPS				
531	IQ feedback	INT16	RO	FALSE	9.7	AMPS				
532	ID error	INT16	RO	FALSE	9.7	AMPS				
533	IQ error	INT16	RO	FALSE	9.7	AMPS				
534	ID rotating	INT16	RO	FALSE	9.7	AMPS				
535	IQ rotating	INT16	RO	FALSE	9.7	AMPS				
536	ID voltage	INT16	RO	FALSE	11.5	VOLTS				
537	IQ voltage	INT16	RO	FALSE	11.5	VOLTS				
538	ID rotating voltage	INT16	RO	FALSE	11.5	VOLTS				
539	IQ rotating voltage	INT16	RO	FALSE	11.5	VOLTS				
540	R voltage	INT16	RO	FALSE	11.5	VOLTS				
541	S voltage	INT16	RO	FALSE	11.5	VOLTS				
542	T voltage	INT16	RO	FALSE	11.5	VOLTS				
543	Harmonic voltage	INT16	RO	FALSE	11.5	VOLTS				
544	R PWM	UINT16	RO	FALSE	11.5	VOLTS				
545	S PWM	UINT16	RO	FALSE	11.5	VOLTS				
546	T PWM	UINT16	RO	FALSE	11.5	VOLTS				
547	(reserved)	UINT16	RO	FALSE	11.5	VOLTS				
548	ID Controller	INT32	RO	FALSE	11.21	VOLTS				
550	IQ Controller	INT32	RO	FALSE	11.21	VOLTS				
552	P Controller	INT32	RO	FALSE	9.23	AMPS				
4000	Security Key	UINT16	RW	FALSE						
4001	Commands	UINT16	RW	FALSE		EXEC_CMD_ENUM				
4100	(reserved)	UINT16	RW	FALSE						
4101	(reserved)	UINT16	RW	FALSE						
4102	Scope Var 1 MODBUS ID	UINT16	RW	FALSE			0	65535		
4103	Scope Var 1 Flags	UINT16	RW	FALSE		VMONITOR_ENUM				
4104	Scope Var 2 MODBUS ID	UINT16	RW	FALSE			0	65535		
4105	Scope Var 2 Flags	UINT16	RW	FALSE		VMONITOR_ENUM				
4106	Scope Var 3 MODBUS ID	UINT16	RW	FALSE			0	65535		
4107	Scope Var 3 Flags	UINT16	RW	FALSE		VMONITOR_ENUM				
4108	Scope Var 4 MODBUS ID	UINT16	RW	FALSE			0	65535		
4109	Scope Var 4 Flags	UINT16	RW	FALSE		VMONITOR_ENUM				
4200	Serial Errors	UINT16	RW	FALSE		SERIAL_ERROR_ENUM				
4201	Serial Rx Unexpected	UINT16	RW	FALSE						
4300	Host Input Functions	UINT32	RW	FALSE		INPUT_FUNCTIONS_ENUM				
4302	Host Disables	UINT16	RW	FALSE		BITMAP (Host Defined)	0	65535		
4303	Host Command Mode	UINT16	RW	FALSE		COMMAND_MODE_ENUM				
4304	Host Position Command	INT32	RW	FALSE	16.16	REVS				
4306	Host Velocity Command	INT16	RW	FALSE	8.8	RPS				
4307	Host Accel	UINT16	RW	FALSE	12.4	RPS/S				
4308	Host Current	INT16	RW	FALSE	9.7	AMPS				
4309	Host Voltage	INT16	RW	FALSE	11.5	VOLTS				
4310	Host Input Inhibits	UINT16	RW	FALSE		INPUTS_ENUM				
4311	Host Output Inhibits	UINT16	RW	FALSE		OUTPUTS_ENUM				

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
4312	Host Outputs	UINT16	RW	FALSE		OUTPUTS_ENUM				
4313	Factory Test Word	UINT16	RW	FALSE						
5000	Drive Name	STRING	RW	TRUE		ASCII				
5100	Options	UINT16	RW	TRUE		OPTION_FLAGS_ENUM				
5101	Power-up Delay	UINT16	RW	TRUE		MS			1.12	
5102	Hard Fault Enables	UINT16	RW	TRUE		FAULTS_ENUM				
5103	Soft Fault Enables	UINT16	RW	TRUE		FAULTS_ENUM				
5104	Default Command Mode	UINT16	RW	TRUE		COMMAND_MODE_ENUM				
5105	Alternate Command Mode	UINT16	RW	TRUE		COMMAND_MODE_ENUM				
5106	Fault Stop Enables	UINT16	RW	TRUE		FAULTS_ENUM			1.12	
5107	Fault Move Enables	UINT16	RW	TRUE		FAULTS_ENUM			1.12	
5200	User I Peak	UINT16	RW	TRUE	9.7	AMPS				
5201	I Limit Time	UINT16	RW	TRUE		MS				
5202	Position Error Limit	UINT32	RW	TRUE	16.16	REVS				
5204	In Position Window	UINT32	RW	TRUE	16.16	REVS				
5206	Position Error Time	UINT16	RW	TRUE		MS				
5207	In Position Time	UINT16	RW	TRUE		MS				
5208	Pos Limit Minus Revs	INT32	RW	TRUE	16.16	REVS			1.12	
5210	Pos Limit Plus Revs	INT32	RW	TRUE	16.16	REVS			1.12	
5212	Pos Limit Minus Percent	UINT16	RW	TRUE	1.15	PERCENT	0	32767	1.12	
5213	Pos Limit Plus Percent	UINT16	RW	TRUE	1.15	PERCENT	0	32767	1.12	
5214	Pos Limit Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767	1.12	
5215	Pos Limit I Foldback	UINT16	RW	TRUE	9.7	AMPS	0	32767	1.12	
5216	Pos Limit I Peak	UINT16	RW	TRUE	9.7	AMPS	0	32767	1.12	
5217	Pos Limit I Peak Time	UINT16	RW	TRUE		MS			1.12	
5300	Serial Flags	UINT16	RW	TRUE		SERIAL_FLAGS_ENUM				
5301	Serial Axis ID	UINT16	RW	TRUE			1	247		
5302	Serial Baud	UINT16	RW	TRUE		BITS/S				
5303	Serial Rx Timeout	UINT16	RW	TRUE		MS	0			
5304	Serial Frame Delay	UINT16	RW	TRUE		MS	0			
5305	(reserved)	UINT16	RW	TRUE						
5400	Ploop KP	UINT16	RW	TRUE	1.15		0	32767		
5401	Ploop KD	UINT16	RW	TRUE	1.15		0	32767		
5402	Ploop KI	UINT16	RW	TRUE	1.15		0	32767		
5403	Ploop KVF	UINT16	RW	TRUE	1.15		0	32767	1.12	
6000	Stop Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6001	Home Flags	UINT16	RW	TRUE		HOME_FLAGS_ENUM				
6002	Home Position	INT32	RW	TRUE	16.16	REVS				
6004	Home Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6005	Home Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6006	Home I Limit	UINT16	RW	TRUE	9.7	AMPS	0	32767		
6007	Jog Flags	UINT16	RW	TRUE		JOG_FLAGS_ENUM				
6008	Jog Slow Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6009	Jog Fast Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6010	Jog Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6011	(reserved)	UINT16	RW	TRUE						
6012	(reserved)	UINT16	RW	TRUE						
6013	(reserved)	UINT16	RW	TRUE						
6014	Dedicated Move Position	INT32	RW	TRUE	16.16	REVS				
6016	Dedicated Move Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6017	Dedicated Move Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6018	Move 1 Flags	UINT16	RW	TRUE		MOVE_FLAGS_ENUM				
6019	Move 2 Flags	UINT16	RW	TRUE		MOVE_FLAGS_ENUM				
6020	Move 3 Flags	UINT16	RW	TRUE		MOVE_FLAGS_ENUM				
6021	Move 4 Flags	UINT16	RW	TRUE		MOVE_FLAGS_ENUM				

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
6022	Move 1 Position/Distance	INT32	RW	TRUE	16.16	REVS				
6024	Move 2 Position/Distance	INT32	RW	TRUE	16.16	REVS				
6026	Move 3 Position/Distance	INT32	RW	TRUE	16.16	REVS				
6028	Move 4 Position/Distance	INT32	RW	TRUE	16.16	REVS				
6030	Move 1 Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6031	Move 2 Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6032	Move 3 Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6033	Move 4 Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
6034	Move 1 Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6035	Move 2 Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6036	Move 3 Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6037	Move 4 Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
6038	Move 1 Feed Velocity	INT16	RW	TRUE	8.8	RPS	-32768	32767		
6039	Move 2 Feed Velocity	INT16	RW	TRUE	8.8	RPS	-32768	32767		
6040	Move 3 Feed Velocity	INT16	RW	TRUE	8.8	RPS	-32768	32767		
6041	Move 4 Feed Velocity	INT16	RW	TRUE	8.8	RPS	-32768	32767		
6042	Move 1 Feed I Limit	UINT16	RW	TRUE	9.7	AMPS	0	32767		
6043	Move 2 Feed I Limit	UINT16	RW	TRUE	9.7	AMPS	0	32767		
6044	Move 3 Feed I Limit	UINT16	RW	TRUE	9.7	AMPS	0	32767		
6045	Move 4 Feed I Limit	UINT16	RW	TRUE	9.7	AMPS	0	32767		
7000	H/W Input Polarities	UINT16	RW	TRUE		INPUTS_ENUM				
7001	H/W Output Polarities	UINT16	RW	TRUE		OUTPUTS_ENUM				
7002	Input 1 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7004	Input 2 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7006	Input 3 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7008	Input 4 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7010	Input 5 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7012	Input 6 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7014	Input 7 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7016	Input 8 Function Assignment	UINT32	RW	TRUE		INPUT_FUNCTIONS_ENUM				
7018	Output 1 Function Assignmer	UINT32	RW	TRUE		OUTPUT_FUNCTIONS_ENUM				
7020	Output 2 Function Assignmer	UINT32	RW	TRUE		OUTPUT_FUNCTIONS_ENUM				
7022	Output 3 Function Assignmer	UINT32	RW	TRUE		OUTPUT_FUNCTIONS_ENUM				
7024	Output 4 Function Assignmer	UINT32	RW	TRUE		OUTPUT_FUNCTIONS_ENUM				
7100	AIN Filter Coef	UINT16	RW	TRUE	1.15		0	32767		
7101	AOUT Filter Coef	UINT16	RW	TRUE	1.15		0	32767		
7102	AIN Cal Low	INT16	RW	TRUE	1.15	ADC	0	32767		
7103	AIN Cal High	INT16	RW	TRUE	1.15	ADC	0	32767		
7104	AOUT Cal Low	INT16	RW	TRUE	1.15	DAC	-32768	32767		
7105	AOUT Cal High	UINT16	RW	TRUE	1.15	DAC	0	32767		
7106	AIN Position Minimum	INT32	RW	TRUE	16.16	REVS				
7108	AIN Position Maximum	INT32	RW	TRUE	16.16	REVS				
7110	AIN Velocity Minimum	INT16	RW	TRUE	8.8	RPS	-32768	32767		
7111	AIN Velocity Maximum	INT16	RW	TRUE	8.8	RPS	-32768	32767		
7112	AIN Current Minimum	INT16	RW	TRUE	9.7	AMPS	-32768	32767		
7113	AIN Current Maximum	INT16	RW	TRUE	9.7	AMPS	-32768	32767		
7114	AOUT MODBUS ID	UINT16	RW	TRUE			0	65535		
7115	AOUT Flags	UINT16	RW	TRUE		VMONITOR_ENUM				

ID	Variable Name	Type	Access	NVM	Scale	Units	Min	Max	Fw Min	Fw Max
7116	AOUT Variable Minimum	INT32	RW	TRUE						
7118	AOUT Variable Maximum	INT32	RW	TRUE						
7120	Analog Velocity	UINT16	RW	TRUE	8.8	RPS	0	32767		
7121	Analog Position Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767		
7122	Analog Velocity Accel	UINT16	RW	TRUE	12.4	RPS/S	0	32767	1.14	
9000	Part Number	STRING	RPW	TRUE						
9016	Serial Number	STRING	RPW	TRUE						
9100	Model	STRING	RPW	TRUE						
9116	Electrical Cycles Per Rev	UINT16	RPW	TRUE		CYCLES	1	5		
9117	R	UINT16	RPW	TRUE	8.8	OHMS L-L				
9118	L	UINT16	RPW	TRUE	8.8	mH L-L				
9119	KE	UINT16	RPW	TRUE		Vrms/KRPM				1.10
9120	KT	UINT16	RPW	TRUE	6.10	Nm/AMP				
9121	J	UINT16	RPW	TRUE	0.18	kg-m^2				
9200	Low Voltage Trip	UINT16	RPW	TRUE	11.5	VOLTS				
9201	High Voltage Trip	UINT16	RPW	TRUE	11.5	VOLTS				
9202	Board Temperature Trip	UINT16	RPW	TRUE	11.5	DEG C				
9203	I trip	UINT16	RPW	TRUE	9.7	AMPS				
9204	I peak	UINT16	RPW	TRUE	9.7	AMPS				
9205	I continuous	UINT16	RPW	TRUE	9.7	AMPS				
9206	Power Filter Coef	UINT16	RPW	TRUE	1.15		0	32767		
9207	Ain 20ma Trip Level	UINT16	RPW	TRUE	6.10	mAmps			1.12	
9300	Vbus Offset	INT16	RPW	TRUE	11.5	VOLTS				
9301	Vbus Scale	UINT16	RPW	TRUE	11.5	VOLTS/ADC				
9302	Board Temp Offset	INT16	RPW	TRUE	11.5	DEG C				
9303	Board Temp Scale	UINT16	RPW	TRUE	11.5	DEG C/ADC				
9304	Psine Zero	INT16	RPW	TRUE	1.15					
9305	Psine Scale	INT16	RPW	TRUE	3.13					
9306	Pcosine Zero	INT16	RPW	TRUE	1.15					
9307	Pcosine Scale	INT16	RPW	TRUE	3.13					
9308	Rphase Offset	INT16	RPW	TRUE		ADC BITS				
9309	Rphase Scale	INT16	RPW	TRUE	10.6	AMPS FULL SCALE				
9310	Sphase Offset	INT16	RPW	TRUE		ADC BITS				
9311	Sphase Scale	INT16	RPW	TRUE	10.6	AMPS FULL SCALE				
9312	Tphase Offset	INT16	RPW	TRUE		ADC BITS				
9313	Tphase Scale	INT16	RPW	TRUE	10.6	AMPS FULL SCALE				
9314	AOUT Zero Calibration	INT16	RPW	TRUE	1.15		-32768	32767		
9315	Electric Angle Offset	UINT16	RPW	TRUE	0.16	2PI RADIANS	0	65535		
9317	Vloop Bandwidth	UINT16	RPW	TRUE	0		0	32767	1.12	
9318	Ain Vlow	INT16	RPW	TRUE	6.10	VOLTS			1.12	
9319	Ain Vhigh	INT16	RPW	TRUE	6.10	VOLTS			1.12	
9320	Ain Vlow Raw	INT16	RPW	TRUE	1.15	ADC			1.12	
9321	Ain Vhigh Raw	INT16	RPW	TRUE	1.15	ADC			1.12	
9322	Ain Ilow	INT16	RPW	TRUE	6.10	mAmps			1.12	
9323	Ain Ihigh	INT16	RPW	TRUE	6.10	mAmps			1.12	
9324	Ain IlowRaw	INT16	RPW	TRUE	1.15	ADC			1.12	
9325	Ain IhighRaw	INT16	RPW	TRUE	1.15	ADC			1.12	
9400	Position tracking gain	UINT16	RPW	TRUE	1.15		0	32767		
9401	Position tracking damping	UINT16	RPW	TRUE	1.15		0	32767		
9402	Iloop KP	UINT16	RPW	TRUE	1.15		0	32767		
9403	Iloop KI	UINT16	RPW	TRUE	1.15		0	32767		