ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA

IMPLEMENTACIÓN DE UN NODO DE COMUNICACIÓN MODBUS-TCP/IP MEDIANTE EL USO DE UN MODULO DE CONTROL EMBEBIDO DE LA MARCA RABBIT SEMICONDUCTOR

CARLOS EDUARDO BOHORQUEZ GUTIERREZ

SANGOLQUÍ – ECUADOR

2008

CERTIFICACIÓN

Certificamos que el presente proyecto de grado fue realizado por el Sr. Carlos Eduardo Bohórquez Gutiérrez bajo nuestra dirección.

_____

Ing. Rodolfo Gordillo
DIRECTOR

_____

Ing. Jaime Andrango
CODIRECTOR

AGRADECIMIENTO

En primer lugar agradezco a Dios por ser el pilar más fuerte en el cual me apoye durante los momentos más duros y complicados de este proceso.
Agradezco a mi padre, por toda la paciencia, amor y apoyo que tuvo en la ayuda de este proyecto.
A mi madre por ser el lado sensible y sentimental que me apoyo en la realización de esta tesis.
A mi abuela y hermano, por su apoyo.
A mi novia por ser mi compañera en los momentos difíciles y por toda la paciencia para apoyarme y ayudarme con este proyecto.
A mi familia, por estar siempre preocupados por mi bienestar.
A mis compañeros de trabajo por darme todas las facilidades para que yo complete este requerimiento de vida.
A mi director y codirector por la guía técnica y personal y la ayuda incondicional ofrecida para desarrollar este proyecto.

DEDICATORIA

Este trabajo está dedicado a mis padres, quienes han dado todo su esfuerzo y cariño para darme una educación profesional, espiritual y moral de alto nivel. Este es el primero éxito de muchos que les entrego para demostrarle la gratitud inmensa que tengo por todo el amor y cariño. A mis familiares y seres queridos, que de alguna manera formaron parte de este proyecto y estuvieron presentes hasta el último momento del mismo.

PRÓLOGO

Dentro de una empresa de manufactura se pueden distinguir dos áreas. El área de administración y gestión, la cual se encarga de las finanzas, la administración de los recursos humanos y de la contabilidad en general de la empresa. Y el área de manufactura, que se encarga del control y de la planificación de los procesos de manufactura, de los requerimientos de materiales y diseño de productos.

Gracias a la integración de estas dos áreas, se ha podido agrupar todas las fuentes de información de la empresa y como resultado se ha facilitado la toma de decisiones tanto a nivel administrativo como de manufactura.

Las redes industriales son una herramienta tecnológica que ha existido desde el final de los años 70’s y ha permitido la agrupación de los dispositivos industriales dentro una misma red de comunicación. Gracias a dicha agrupación, la industria ha podido mejorar los procesos y reducir el tiempo de producción garantizando un mayor grado de confiabilidad.

En una red industrial estarán presentes equipos y dispositivos de todo tipo. A dichos dispositivos se los agrupa de una manera jerárquica dependiendo de sus posibilidades y fortalezas tecnológicas, con el objeto de establecer conexiones que aprovechen de mejor manera dichos recursos. Es así como se forman los niveles jerárquicos dentro de las redes industriales.

El nivel de gestión, nivel superior dentro de la jerarquía, se encarga de integrar los niveles inferiores dentro de la estructura de la empresa. Los equipos ubicados dentro de este nivel son computadores que sirven de intermediarios entre los equipos industriales y el área administrativa y de gestión. Dentro de este nivel se utiliza una red de datos tipo LAN.

El nivel de control, nivel ubicado bajo el nivel de gestión, se encarga de enlazar y dirigir las distintas zonas de trabajo. Los equipos dentro de este nivel son aquellos autómatas de una gama alta y computadores dedicados al diseño y programación. En este

nivel se utilizan redes tipo LAN o versiones mejoradas de los protocolos de campo como Modbus TCP, Profibus Ethernet, Profibus FMS.

El nivel de Campo, inferior al nivel de control, se encarga de la integración de los equipos controladores y módulos de entradas y salidas. Dentro de este nivel se establecen sub-redes, dependiendo del protocolo de comunicación que usen dichos dispositivos. Dentro de este nivel se utilizan redes en base a buses de campo tales como, Profibus PA, Modbus, Modbus Plus.

El Nivel de E/S, es el nivel inferior dentro de la jerarquía y a la vez es el nivel más próximo al proceso. Los equipos que encontramos aquí son los sensores y actuadores, cuya labor es la de manejar el proceso productivo y tomar las medidas necesarias para la correcta automatización y supervisión. Protocolos de comunicación como Profibus DP, AS-I y DeviceNet son algunos de los que permiten la comunicación de estos equipos a este nivel.

Actualmente los equipos industriales están migrando a un protocolo de comunicación basado en el estándar abierto Ethernet, el cual presenta grandes ventajas de integración y permite el desarrollo de un mayor número de aplicaciones dentro del nivel de gestión.

Pero en el mercado existen muchos equipos que no tiene las posibilidades de comunicarse a través del protocolo Ethernet. Por este motivo dichos equipos no pueden integrarse en las redes industriales actuales. El presente proyecto tiene como objetivo el implementar una opción económica y flexible para que equipos antiguos pueden formar parte de redes industriales de última tecnología. Adicionalmente, se desarrollará la opción de monitoreo a través de aplicaciones HTTP con el objeto de proporcionar una herramienta económica y amigable para realizar monitores de variables a nivel industrial.

INDICE

CAPITULO 1

INTRODUCCIÓN

1.1 Redes Industriales.

1.1.1 Introducción

Actualmente las empresas ecuatorianas buscan volverse competitivas dentro de un entorno comercial internacional. La competitividad se la elabora en base a una eficiente interrelación de los elementos ubicados en diferentes niveles dentro del esquema funcional de la empresa. Las ventajas competitivas de una empresa no aseguran el obtener mejores beneficios, pero posibilitan su logro.

Una definición ambigua de competitividad la vincula estrictamente con la disminución de costos en los productos. Hoy en día, este término abarca más características además del costo, tales como, el diseño del producto, velocidad de entrega, la infraestructura para dar servicios, control de calidad e impacto en el mercado.

La implementación de una red de datos industrial aporta con muchas ventajas al proceso de competitividad de las empresas. Dentro de las ventajas que nos presentan las redes industriales tenemos mayor rapidez en la adquisición de datos, flexibilidad en las operaciones industriales y ahorro en proceso de mantenimiento.

Debido a la existencia de diversas áreas dentro de la industria, las redes se dividen según el entorno en el que son aplicadas. De esta manera se identifican las redes de datos convencionales, las cuales en su mayoría basan su conectividad en la suite de protocolos TCP/IP, y las redes de datos formadas por los protocolos que constituyen los buses de campo.

En ese capítulo se hará una breve introducción a la diversas variedades de buses de campo existentes, identificándolos según sus características y su correspondencia con los requerimientos en la industria. De igual manera se hablará sobre la arquitectura de los buses de campo en función de los protocolos por los cuales están constituidos y de su distribución en los niveles industriales. Finalmente se hará una reseña general de la integración de las redes de buses seriales a sistemas dentro de una red Ethernet.

1.1.2 Buses de Campo

Se conoce como buses de campo a los sistemas de comunicación que permiten la interconexión de dispositivos de adquisición de datos, sensores, actuadores y controladores entre sí, dentro de ambientes industriales. Los buses de campo aparecen gracias a la creación de dispositivos industriales inteligentes, definiendo como inteligencia a la capacidad de estos para gestionar el flujo de información de entrada y salida, además de poseer características de autocontrol y autodiagnóstico.

Dichos buses de campo tiene vital importancia en los sistemas de automatización y control, ya que son las vías por donde circula toda la información involucrada con el lazo de control.

Las ventajas con las que aporta el uso de los buses de campo en la industria son:

- Ahorro económico que presentan al usuario. Esto se debe a la disminución en el uso de cables de conexión y costos de instalación en general. A pesar de que la inversión inicial requiera una mayor cantidad de dinero, el ahorro se evidencia a largo plazo con todas las comodidades y prestaciones de estos equipos.

- Ahorro en los procesos de mantenimiento y mejoras del funcionamiento de todo el sistema, gracias a las características inteligentes y de autodiagnóstico que presentan los dispositivos incluidos en las redes de buses de campo.

- Facilidad en la ampliación de instalaciones y capacidades de los sistemas de comunicaciones.

- Facilidad en el monitoreo de los procesos e interpretación de las variables, lo que permite una rápida detección de errores dentro de los elementos de la red.

1.1.3 Tipos de Buses de Campo.

A pesar de que en la mayoría de escritos, se consideran como buses de campo solo a aquellos sistemas de comunicaciones que interconectan a los elementos primarios de medición y los elemento finales de control, entre sí y con sus controladores respectivos en los ambientes industriales, actualmente existen una amplia gama de variaciones y mejoras de dichos buses de campo, los cuales soportan procesos mas complejos y funciones mas elaboradas, permitiendo un mejor nivel de automatización dentro de la industria.

Es por este motivo que los buses de campo se los puede clasificar en tres subgrupos, dependiendo de la complejidad de sus funciones y de la capacidad de información que manipulan. Se subdividen a los buses de campo en [1]:

1. SENSORBUS
2. DEVICEBUS
3. FIELDBUS

En el *Figura 1.1* se puede ver la relación existente entre estas tres subdivisiones, respecto a la capacidad de información que transmiten y al tipo de funciones que desempeñan.

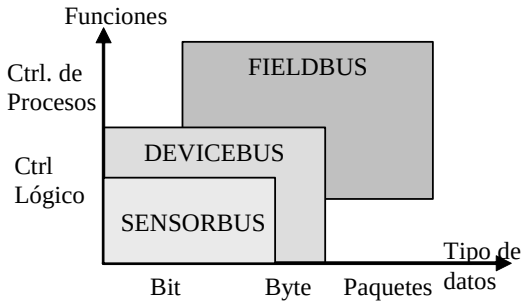


Figura 1.1. Subdivisión de los Buses de Campos

Sensorbus.-Diseñados para integrar dispositivos simples como fines de carrera, celdas fotoeléctricas, actuadores simples. Si tomamos como referencia el modelo OSI, este subgrupo estaría formado solamente por las dos primeras capas de dicho modelo de referencia, la capa física y la capa de enlace.

Devicebus.- Son buses de alta velocidad pero de funcionalidad media. Establecen una estructura de datos un poco más compleja en donde se encapsulan datos de configuración, calibración o programación del dispositivo. Con referencia al modelo OSI ocuparían la capa física, capa de enlace y de aplicación. Esta ultima con aplicaciones de capacidades limitadas.

Fieldbus.- Buses de altas prestaciones, la capa de aplicaciones presenta una gran variedad de servicios a la capa de usuario. Bloques de mensajes más complejos y de un gran tamaño.

En el *Figura 1.2* se describen ejemplos de buses de campo, según la clasificación antes señalada, en relación con las aplicaciones industriales de entrada y salida de información.

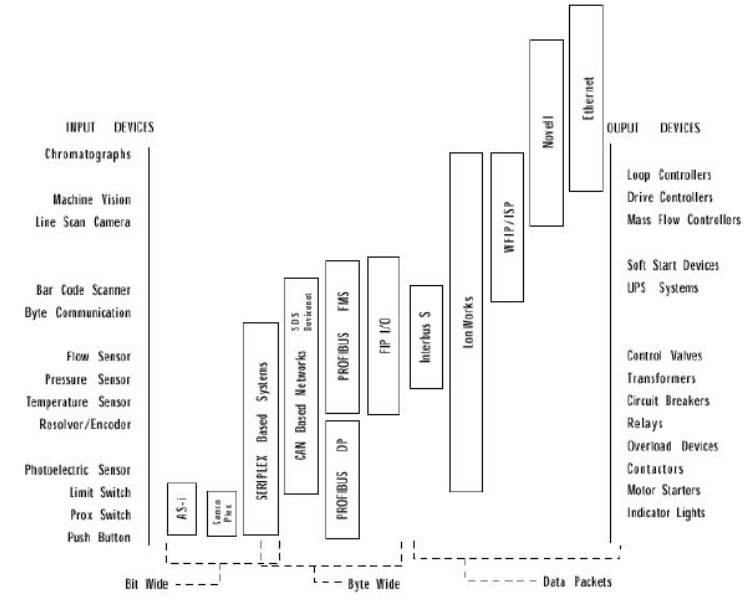


Figura 1.2. Ejemplos de Aplicaciones de los Buses de Campo.

1.1.4 Ethernet Industrial

El estándar IEEE 802.3 o comúnmente conocido como Ethernet es un protocolo que ocupa las capas 1 y 2 del modelo de referencia OSI. Pero también el termino Ethernet se lo ha popularizado para la denotación de las redes de datos entre computadores a lo ancho del mundo.

Gracias al desarrollo de la suite de protocolos TCP/IP es posible la intercomunicación de computadores sin importar el fabricante ni el tipo de aplicación que ejecuten. En resumidas cuentas el uso del estándar IEEE 802.3 y los protocolos TCP/IP dan el soporte necesario para que lo que hoy se conoce como Internet tenga tanto éxito a nivel mundial.

El éxito de Ethernet dentro del mundo de networking se debe a que incluso hasta usuarios inexpertos pueden construir simples redes y conectar computadores entre sí.

Gracias a la aceptación de estas dos herramientas tecnológicas dentro de las redes industriales, se visualiza un futuro tecnológico lleno de desarrollo de aplicaciones y adaptaciones de dicho protocolo para el mejoramiento de los actuales servicios que prestan los Buses de Campo.

1.1.5 Comparación del estándar Ethernet vs. Buses de Campo

Antes de que el estándar Ethernet entre en cuadros comparativos con los demás Buses de Campo y empiece a ocupar lugar en los ambientes industriales debe definir su condición frente a asuntos muy importantes, que son:

1. Se debe establecer una capa de Aplicación común. La necesidad de definir una capa de aplicación común nace del hecho que el paquete de datos puede ser de diferentes formatos, puede ser una cadena de valores de E/S, un documento de texto, o una serie de parámetros para un variador de frecuencias. Este punto es polémico ya q Ethernet ha tenido tanto éxito debido a su transparencia frente a las aplicaciones a nivel de usuario. Pero dentro del mundo industrial esta libertad y

transparencia no puede ser tan evidente debido a las necesidades de confiabilidad y seguridad requeridas por los entornos industriales [2].

2. Los conectores RJ45, los más comunes en redes Ethernet, no cumplen con los requerimientos que deben tener los conectores que van a trabajar en ambientes industriales [2].

3. Es común la utilización de buses de campo con la capacidad de alimentación de poder sobre las mismas líneas de comunicación con el objeto de minimizar cableado y ahorrar recursos. Este una gran incapacidad que presenta Ethernet, pero en la actualidad se encuentra en desarrollo su solución [2].

4. En ambientes industriales muchas aplicaciones requieren un esquema determinístico. El estándar Ethernet en su uso típico no es determinístico o repetible. En otras palabras, las tasas de rendimiento no son garantizadas. Sin embargo existen métodos para establecer a los sistemas Ethernet en una arquitectura deterministica [2].

Desde un punto de vista general, los buses de campo, tales como DeviceNet, Profibus, Interbus entre otros, poseen una estructura ordenada y predefinida de la ubicación de la información (bits, bytes). Mientras que Ethernet y TCP/IP, a pesar de tener una trama conocida en la cual se encapsula la información, envían cualquier tipo de información fragmentando la trama para su más rápido envío. Dejando el trabajo de ordenamiento e interpretación de la información a la aplicación de usuario final.

Gracias a que Ethernet y TCP/IP pueden encapsular cualquier tipo de información en su trama, es posible que los paquetes de información, provenientes de los buses de campo, puedan ser enviados sobre una red Ethernet, dejando a la capa de aplicación el trabajo de utilizar esta información para fines de control y monitoreo a nivel industrial.

1.1.6 Protocolos de la Capa de Aplicación TCP/IP para ambientes industriales

La siguiente etapa para las organizaciones que regulan la operación e implementación de los buses de campo es el desarrollar capas de aplicación Ethernet TCP/IP. Actualmente los cuatro contendientes más grandes son [2]:

- *Modbus/TCP (Modbus protocol on TCP/IP),*
- *EtherNet/IP (the ControlNet/DeviceNet objects on TCP/IP)*
- *Profibus on Ethernet.*
- *Foundation Fieldbus High Speed Ethernet (HSE)*

A pesar de que existen un infinito número de potenciales protocolos de la capa de aplicación y una gran variedad de estándares propietarios para dicha capa, el emplear las arquitecturas de buses ya existentes aporta con significantes ventajas, tales como:

- Perfiles para muchos dispositivos ya han sido definidos, y pueden ser transferidos a Ethernet con poco esfuerzo.

- En sistemas los cuales usan, por ejemplo, Profibus como red del nivel de E/S, y Profibus sobre Ethernet en el nivel de supervisión, la relación entre las dos redes es relativamente transparente.

- Muchos de los desarrolladores y usuarios están familiarizados con estos protocolos, con eso se gana rapidez en los procesos de desarrollo de los productos.

A continuación una breve descripción de cada uno de los protocolos antes mencionados:

Modbus/TCP
Actualmente como el estándar por defecto para Ethernet en ambientes industriales. Este protocolo toma ventaja de la simplicidad y la disponibilidad de Modbus, pero también se ve afectado por sus limitaciones tal como la limitada capacidad para transmitir grupos complejos de parámetros de información entre dispositivos [2].

Ethernet/IP
A pesar de que recién ahora esta siendo lanzada al mercado, sus especificaciones poseen fuertes posibilidades. Información sobre sus especificaciones y ejemplos de código van a estar disponibles a través de la ODVA (por las siglas en ingles de Open DeviceNet Vendor Organization), ControlNet Internacional y Industrial Ethernet Association [2].

Profibus on Ethernet
Esta especificación aun no ha sido lanzada al mercado. Sin embargo, con la gran aceptación de Profibus a nivel mundial, se visiona un gran éxito comercial [2].

Foundation Fieldbus High Speed Ethernet (HSE)
Lo que hace HSE es poner el protocolo H1 de fieldbus en TCP/IP y adicional a esto suma servicios como OPC, XML y SOA (por las siglas en ingles de Simple Object Access) [2].

Al igual que actualmente en una red LAN de oficinas pueden estar corriendo diferentes tipos de aplicaciones como archivos HTML, documentos de Word u otras aplicaciones. Estos cuatro protocolos pueden coexistir en la misma red bajo el mismo medio físico y operándose al mismo tiempo [2].

1.1.7 Conectores TCP/IP industriales y alimentación sobre Ethernet.

Actualmente muchas de las empresas a nivel mundial que entregan servicios de integración industrial sobre TCP/IP han mejorado las características de los medios físicos, tales como conectores y cableado en general para que cumplan con normas industriales y puedan coexistir con otros equipos de alto rendimiento para la industria.

Otro de los retos para los impulsadores del estándar Ethernet como bus de campo es el desarrollar la capacidad para que el medio pueda llevar poder de alimentación a los dispositivos que se encuentran en contacto con el proceso. Actualmente ya existe la variación del estándar IEEE 802.3 que se refiere a la alimentación de energía sobre Ethernet. Este estándar se conoce como IEEE 802.3af [3].

El estándar IEEE 802.3af proporciona 48VDC sobre dos de los 4 pares disponibles en un cable Cat3 o Cat5, con un valor de corriente máxima de 400 mA para una potencia de carga máxima de 15.4 W. Solo 12.95 W están disponibles después de contar con las perdidas. La *tabla 1* muestra valores de potencia configurables en equipos Ethernet [2].

Clase	Uso	Niveles de Poder Máximos
0	Defecto	0.44 a 12.95 W
1	Opcional	0.44 a 3.84 W
2	Opcional	3.84 a 6.49 W
3	Opcional	6.49 a 12.95 W

Tabla 1.1. Valores de Poder en el Estándar Ethernet.

A pesar de que la potencia no es muy alta, no mayor de 13 W, actualmente la mayoría de los sensores industriales modernos tiene un consumo de entre 5 a 10 W. Esto nos hace pensar en la posibilidad de alimentar todo un sistema SCADA a través de puertos de los switches Ethernet [3].

Un estándar futuro, comúnmente referido como PoE+ (Power on Ethernet Plus) está siendo desarrollado por el equipo de investigación de la IEEE 802.3at. Este estándar va a presentar una extendida capacidad de alimentación de poder usando los cuatro pares de un cable Cat 5 y con una capacidad de 56W [2].

1.1.8 Ventajas y Desventajas de las Redes Industrial Ethernet

Ventajas

1. Ethernet es el estándar de networking mas aceptado a nivel mundial. Ethernet puede manipular grandes cantidades de información a altas velocidades y servir a las necesidades de amplias instalaciones.

2. Gracias a la libertar de desarrollo de aplicaciones para TCP/IP, los procesos de control y monitoreo pueden ser mejorados y poseer una mejor adaptabilidad con los niveles administrativos de una red industrial.

3. El uso de Ethernet por si mismo sugiere el final de la guerra de protocolos de red, permitiendo así una estandarización que facilitaría procesos de mantenimiento y flexibilidad en su distribución.

4. Gracias al uso de Ethernet se puede transportar paquetes de los buses de campo sobre Internet, logrando con esto la comunicación global y la integración una realidad.

5. Mejor aprovechamiento de tecnologías inalámbricas pertenecientes al mundo de redes sobre TCP/IP, tales como Wi-Fi y Wi-Max, logrando así mejores condiciones para el desarrollo de aplicaciones remotas.

Desventajas

1. A pesar de que ya existen estudios sobre como transmitir potencia a través de Ethernet, la capacidad aún no es la suficiente para que Ethernet sustente por si solo todo un sistemas SCADA completo.

2. En contraste con la mayoría de buses de campo, Ethernet es más susceptible a la interferencia Electromagnética y a la interferencia por Radio Frecuencia.

3. El uso de Ethernet no puede alcanzar el nivel de sensores debido a que no se justifica su costo para la operación de funciones básicas tales como encendido y apagado de valores discretos.

1.2 Arquitectura de las Redes Industriales.

1.2.1 Introducción

Las estructuras tradicionales dentro de los procesos industriales o de manufactura han sido una restricción para una integración eficiente de los dispositivos involucrados en los

procesos. Como requerimiento esencial para una integración eficiente se necesita una comunicación confiable entre la variedad de dispositivos que se encuentran ubicados en los diferentes niveles funcionales de la empresa, ya sea funciones de operación o de planificación.

Las industrias se valen de la tecnología en desarrollo para el mejoramiento de sus sistemas de comunicación y por ende de la integración total. Tomando como referencia un sistema de manufactura integrada por computadora (CIM), ya que este es el mayor nivel de integración debido al uso de redes industriales, se puede distinguir diferentes arquitecturas que permiten su integración final.

Se dice que un sistema CIM es la máxima expresión de integración debido a que involucra la participación de diferentes disciplinas, las cuales manejan diferentes tipos de información. Entre estas disciplinas podemos identificar algunas tales como el Diseño Asistido por Computadora (CAD), la Manufactura Asistida por Computador (CAM), Planeación Asistida por Computadora (CAP), Control de Calidad Asistido por Computadora (CAQC) y el Control y Planeación de Producción (PP&C).

1.2.2 Arquitectura de las redes industriales en función de los niveles industriales.

La arquitectura para la distribución de las redes industriales, también se vincula a la distribución departamental que posee la industria sobre la cual se va a trabajar. El reconocimiento de dicha distribución nos ayuda a definir los parámetros de selección de las características que deben presentar los buses de campo frente a las necesidades dentro de cada nivel funcional. Estos parámetros se refieren a valores de velocidad de transmisión, capacidades de información y lógica de funcionamiento en general.

Dentro del mundo industrial, las empresas se encuentran constituidas por muchos departamentos, cada uno encargado de aportar con algo para el correcto funcionamiento de dicha empresa. Esta constitución se la puede esquematizar en base a una estructura piramidal *Figura 1.3*. La cual distribuye de manejar jerárquica tres niveles bien definidos en las empresas industriales [4].

Figura1.3 Esquema piramidal de los Niveles Industriales.

Nivel Estratégico o de Gestión Empresarial.- Este es nivel más alto dentro de la jerarquía industrial, y es aquí en donde se manejan toda la gestión y administración global de la empresa. En este nivel se define una filosofía laboral, la cual incluye directrices de producción, estrategias de mercadeo y políticas empresariales.

Nivel Táctico.- Dentro de este nivel se genera todas las actividades que luego van a hacer realizadas por el nivel operativo. Diseños de ingeniería, planificación de mantenimientos y operaciones, control de calidad y de inventarios.

Nivel Operacional.- En este nivel se realizan las actividades y procesos, planificados por el nivel táctico, mediante los cuales la empresa logra su objetivo, como por ejemplo, la elaboración de algún producto determinado, o la generación de un servicio.

Para una mejor identificación del tipo de red que se utiliza para la vinculación de cada uno de los niveles, al nivel táctico y al nivel operacional se divide en dos niveles, cada uno, con el objeto de visualizar de mejor manera la red industrial que se desea utilizar. En el *Figura 1.4* se esquematiza la distribución jerárquica y la ubicación de las redes de comunicación entre cada uno de los niveles.

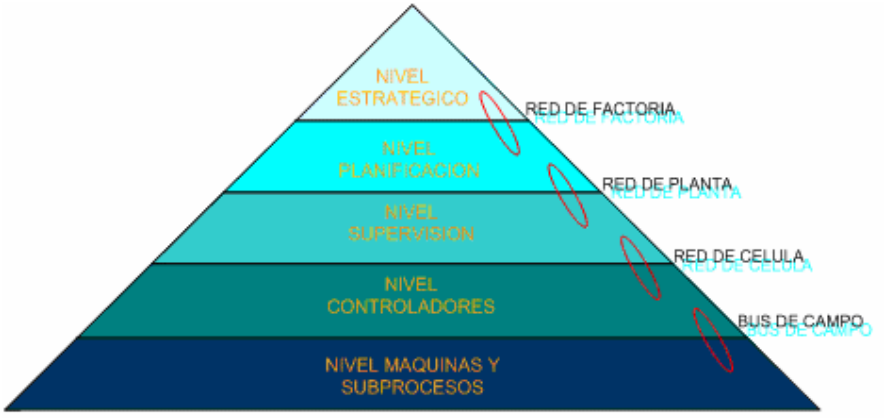


Figura 1.4 Relación entre los Niveles Industriales

Red de Factoría.- Dentro de este nivel se interconectan oficinas administrativas, de contabilidad, ventas y marketing con el nivel de planificación. En su mayoría están basadas en tecnología Ethernet, y tiene acceso al Internet a través de un firewall, el cual provee de hermeticidad hacia el mundo exterior. La velocidad de transferencia de información se encuentra en el orden de los 100Mbps o 1Gbps [5].

Red de Planta.- Dentro de este nivel se interconectan módulos y células con los departamentos de diseño y planificación. Aplicaciones SCADA son utilizadas para la administración y control de la información. La información que se maneja va desde mensajes cortos de comandos de ejecución hasta mensajería para terminales de operarios. Las soluciones clásicas son Ethernet y Token Ring. La velocidad de transferencia de información se encuentra en el orden de los 20Mbps. Las características que deben cumplir son [5]:

- Manejo de mensajes de cualquier tamaño.
- Detección y corrección de errores.
- Cobertura en áreas extensas.
- Manejo de prioridad en mensajería.
- Disponibilidad de ancho de banda.

Red de Célula.- Dentro de esta red se intercomunican elementos controladores de procesos que operan de forma secuencia tales como PLCs u otros tipos de controladores que se

encuentran en el mercado y posean características de comunicaciones. Varias son las opciones de solución que se pueden dar a este nivel, inclusive se cuenta con opciones Ethernet. Las características que deben presentas esta red son [5]:

- Gestión de Mensajes cortos eficientes.
- Capacidad de manejar trafico de eventos discretos.
- Detección y corrección de errores.
- Posibilidad de transmitir mensajes prioritarios.
- Recuperación rápida a eventos anormales en la red.
- Alta disponibilidad.

Bus de Campo.- A pesar de que en las capas superiores también se puede hacer uso de buses de campo con mayores prestaciones. Dentro de este nivel identificamos a los buses de datos utilizados para la interconexión de dispositivos vinculados directamente con le proceso o con la maquinaría involucrada, como variadores de velocidad, módulos de entradas y salidas, sensores inteligentes, transductores y actuadores. El manejo de información se encuentra a nivel de bits, con una constante utilización del medio.

De una manera general, los requerimientos necesarios para una correcta integración de los niveles estratégico, táctico y operacional son los siguientes [4]:

a) Una infraestructura sólida de comunicaciones que comprenda:
   - Redes orientadas al control local
   - Redes orientadas al control supervisor
   - Redes orientadas al soporte de la planificación, ingeniería, gerencia y administración.
   - Interconexión con Redes Externas a la Empresa.

b) El diseño y construcción de un sistema que permita la conectividad e interoperabilidad de todos los sistemas de información y control.

c) Facilidad de acceso a subsistemas.

d) Flexibilidad, calidad y confiabilidad en el flujo de información entre los niveles Operacional, Táctico y Estratégico.

1.2.3 Arquitectura de las redes industriales en función de la lógica de control.

Desde el punto de vista de la distribución dada en la lógica de control, se distinguen dos partes principales:

1. *Parte Operativa.*- En esta parte de la lógica de control tenemos el hardware y el software necesario para elaborar una interfaz amigable que permita al operador recibir la información necesaria para realizar las tareas de planta y tener el control del proceso [5].

2. *Parte Control.*- En esta parte encontramos los dispositivos que realizan las operaciones de control deseadas desde la parte operativa. Dentro de esta parte tenemos equipos tales como PLCs, DCS, PC industriales, actuadores y sensores [5].

Entre todos estos dispositivos, tanto los de la parte operativa como los de la parte de control, existe comunicación. La comunicación puede ser vertical u horizontal. La comunicación vertical se da cuando es entre las dos partes y la comunicación horizontal cuando es entre los dispositivos de la misma parte [5].

Existen dos arquitecturas de control industrial que son las más comunes, a pesar de esto, muchas de las soluciones reales están formadas por la combinación adecuada de estas dos arquitecturas, las cuales son:

1. *Control Centralizado.*- Consta de una computadora central la cual dirige el flujo de la información hacia los demás controladores dentro del proceso. El usuario trabaja sobre la interfaz de la computadora central para llegar a los demás procesos. A pesar de presenta un manejo fácil del flujo de información, su

debilidad radica en que la computadora central debe presentar altos niveles de confiabilidad [5]. El *Figura 1.5* es un diagrama de bloques ejemplificando una distribución de Control Centralizado.

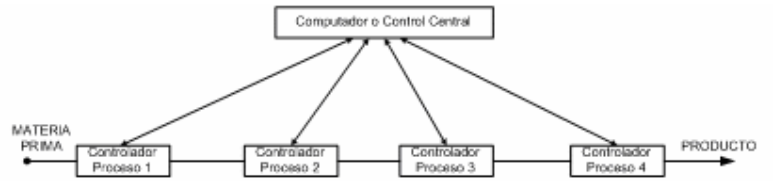


Figura 1.5. Esquema de Control Centralizado.

2. *Control Distribuido.*- En parte similar al centralizado con la diferencia que los controladores tiene comunicación entre ellos. En caso de falla o de sobre carga de trabajo, pueden balancear tareas entre ellos con el objeto de agilitar las operaciones. El mayor requerimiento necesario es que los controladores tengan asignación dinámica de las tareas con lo cual exigen gran capacidad de acceso a la comunicación. El *Figura 1.6* ejemplifica un esquema de la distribución del Control Distribuido.

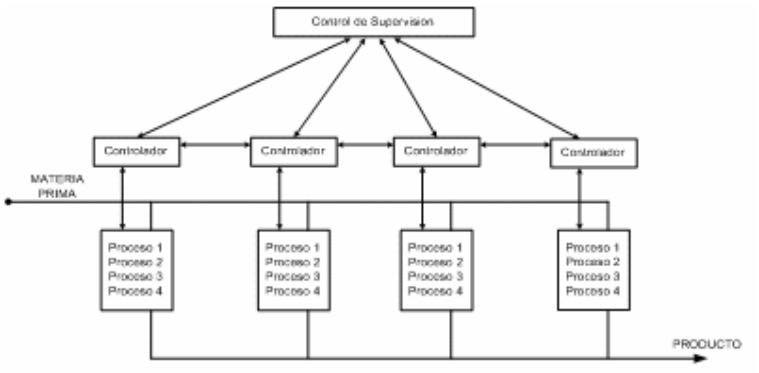


Figura 1.6. Esquema de Control Distribuido.

Tanto la estructura centralizada y distribuida pueden estar presentes no solo en lazos de control, si que pueden formar parte de subsistemas de control o subsistemas de instrumentación inteligente. Es importante la identificación del tipo de control que se desea utilizar para en función de sus requerimientos establecer la solución de comunicación industrial mas acertada.

1.2.4 Arquitectura de las redes industriales en función del Modelo de Referencia OSI

El desarrollo de los buses de campo se lo ha hecho fundamentándose en el modelo de referencia OSI, el cual consta de 7 capaz muy bien conocidas. Dependiendo de la complejidad de las funciones que soporta el bus de campo, este puede hacer uso de ciertas capas, las más indispensables para su implementación, dejando excluidas las que no sean necesarias. La mayoría de los buses de campo tienen especificadas las siguientes capas:

- Capa 1, Capa Física.
- Capa 2, Capa de Enlace.
- Capa 7, Capa de Aplicación.

Adicional a estas tres capas, existe una capa superior a la aplicación, denominada la capa de usuario, es aquí en donde los integradores realizan su trabajo y desarrollan sus propias aplicaciones dependiendo de las necesidades presentes en la industria y de las prestaciones de cada uno de los equipos y de los protocolos de aplicación que se estén utilizando.

A continuación se detallará de manera general, para la diversidad de buses de campo existentes, las tres capas antes mencionadas:

Capa Física (Capa 1).- Dentro de esta capa se especifican las características mecánicas tales como tipo de conectores y medios para la comunicación, las características eléctricas, tales como niveles de voltaje y polaridades. Adicional a esto se definen los tipos de codificación, velocidades de transmisión y distancias máximas de conexión de los buses de campo. Esta capa tiene como tarea, la de enviar los bits a lo largo del medio de comunicaciones.

Los estándares mas usados en este nivel son:

Los estándares RS-232/RS-485/RS-422.- Los estándares RS-232/RS-422/RS-485 son los elementos claves para la transferencia digital de información entre las

unidades remotas y los módems, los cuales conviertes esta señal digital en una analógica para su transmisión sobre grandes distancias.

Estos estándares definen las características eléctricas y mecánicas para que las interfaces permitan la interconexión y comunicación efectiva entre equipos de diferentes marcas. Tanto el estándar RS-232 y los estándares RS-422 y RS-485 cumplen las mismas funciones con la diferencia que los dos últimos permiten transmisiones sobre distancias de 1200m. Para mayor detalle de las características se puede referir a los documentos oficiales de estos estándares **EIA-RS Serial Communications All Models User's Guide** de la empresa General Electric**.**

IEEE 802.3.- El estándar IEEE 802.3 no solo define la capa física, también específica las características de la capa de enlace. A continuación solo se hace referencia a su descripción para la capa física. El estándar IEEE 802.3 enlista los tipos de cables que pueden ser utilizados como medios y las velocidades de transmisión posibles con dichos medios.

Los medios que soporta este estándar son:

- Cable Coaxial.
- Cable Par Trenzado.
- Cable de Fibra Óptica.

La *tabla 1.2* describe los medios y las velocidades de transmisión más comunes dentro de este estándar:

	Medio	Velocidad	Distancia(m)
10BASE2	Cable coaxial	10Mbps	200
10BASE5	Cable coaxial	10Mbps	500
10BASE-T	par trenzado	10Mbps	100
100BASE-TX	par trenzado	100Mbps	100
100BASE-FX	Fibra óptica	100Mbps	224-412
1000BASE-SX	Fibra óptica	1000Mbps	220-550
1000BASE-LX	Fibra óptica	1000Mbps	550-5000

Tabla 1.2. Estándares Ethernet

Cada uno de los estándares mencionados en la *tabla 1.2* posee sus propias características de codificación y de modulación, para realizar la transmisión sobre el medio. Las variaciones de velocidad del estándar Ethernet inicialmente concebido se encuentran detalladas en los estándares IEEE 802.3u y IEEE 802.3z. Para mayores detalles en las especificaciones referirse al los documentos de la IEEE (Institute of Electrical and Electronics Engineers).

IEEE 802.4.- Estándar definido como *Paso de Testigo en Bus*, las características principales de este estándar pertenecen a la capa de enlace de datos, pero existen ciertas especificaciones que pertenecen a la capa física, las cuales son las siguientes:

- Tipo de transmisión Broadband (Banda Ancha):
  - Cable Coaxial de 75ohms o par trenzado Cat 4.
  - Velocidad de transmisión de 10Mbps.
  - Tipo de modulación PSK.

- Tipo de transmisión Carrierband (Banda Portadora):
  - Cable Coaxial de 75ohms o par trenzado Cat 4.
  - Velocidad de transmisión de 5Mbps y 10Mbps.

- Tipo de modulación FSK.

Para mayores detalles en las especificaciones referirse al los documentos de la IEEE (Institute of Electrical and Electronics Engineers).

IEEE 802.5.- Estándar definido como *Paso de Testigo en Anillo*, las características principales de este estándar pertenecen a la capa de enlace de datos, pero existen ciertas especificaciones que pertenecen a la capa física, las cuales son las siguientes:

- Velocidad de transmisión entre 4Mbps y 16Mbps
- Medio utilizado es cable par trenzado apantallado y no apantallado.
- La longitud máxima del cable entre el centro del Token-Ring y el punto de conexión para el nodo de la red puede ser de hasta 350m (recomendado 110m).
- Cuando el cable es no apantallado la distancia máxima se reduce a 110m (recomendado 50m).

IEC 1158-2.- Similar al estándar RS-485, el estándar IEC 1158-2 utiliza un cable de par trenzado apantallado como medio físico para las comunicaciones. Una de las características que más resalta es que presenta seguridad intrínseca, esto quiere decir que por su constitución y diseño, puede formar parte de dispositivos dentro de zonas peligrosas. Mayores detalles se pueden encontrar en los documentos oficiales de la IEC (International Electrotechnical Commission).

Capa de Enlace de Datos (Capa 2 (DLL)).- Dentro de esta capa se define el formato de la trama y el modo en que dicha trama accede al medio físico de transmisión. Adicional a estas dos características, esta capa está encargada de realizar los procesos de detección y corrección de errores que ocurran en la capa inferior (capa física). Esta capa se la puede dividir en dos subcapas, la subcapa de Control de Acceso al Medio (MAC) y la subcapa de Control de Enlace Lógico (LLC). La subcapa MAC es la encargada de vincular la trama con los medios físicos, mientras que la subcapa LLC se encarga de la comunicación con las capas superiores, en el caso de que estas existan.

No todos las redes industriales presenta esta subdivisión en la capa de enlace, la mayoría definen un solo estándar para toda la capa de enlace. Los tipos de acceso a los medio más comunes son:

CSMA/CD.- También conocido como acceso al medio por contienda, para la transmisión los nodos deben escuchar el canal, y esperar a que esté disponible para poner los datos en el medio de comunicaciones. Existe la posibilidad de que dos o más nodos envían información a la vez, lo cual produciría una colisión. En el caso que esto suceda los nodos inician un temporizador aleatorio, luego de la terminación del mismo, el primero que acabe está disponible para reenviar la información.

Este tipo de acceso también se lo conoce como no determinístico, ya que no se puede estimar cuando se va a realizar el envío de la información.

Paso de testigo.- Este estándar consiste en la circulación constante de un testigo (token), el cual es tomado por la estación que desea realizar la transmisión, de esta manera nadie puede acceder al medio hasta que el testigo (token) no sea dejado de ser utilizado. Tanto el estándar IEEE 802.4 e IEEE 802.5, define su funcionamiento por un paso de testito, la diferencia es que el primero lo realiza en una topología en bus y el segundo en una topología en anillo.

Maestro/Esclavo.- Se define un dispositivo maestro el cual solicita información a otros dispositivos denominados esclavos. Los mismos que responden a los maestros con la información solicitada por estos. Esta información puede ser valores de sus entradas/salidas, registros de memoria u otro tipo de información dependiendo de la complejidad de los servicios. Solo el maestro puede iniciar la comunicación.

Datos Cíclicos.- Se definen dispositivos maestros y dispositivos esclavos. La transmisión maestro/esclavo se la realiza a intervalos de tiempo regulares. La programación de los tiempos de los ciclos de comunicación los realiza el usuario.

Este tipo de acceso al medio tiene gran utilidad en redes en las que los cambios de los valores se dan de una manera lenta.

Productor/Consumidor.- Se establece la comunicación luego de definir un dispositivo productor, quien es el que envía el mensaje, y un dispositivo consumidor quien es quien responde al mensaje. En este tipo de acceso al medio, los mensajes son reconocidos por su contenido, mas no por las direcciones destino que lleven.

Cambio de Estado.- El cambio es enviado por parte del dispositivo remoto, únicamente cuando los valores de sus entradas/salidas, o de sus registros de memoria cambian.

SDLC.- Fue el primer protocolo de la capa de enlace, desarrollado por IBM en los años 1970s, viene de las siglas en ingles de Synchronous Data Link Control, luego de su creación muchos otros protocolos fueron desarrollados en base a la estructura de su trama, tales como HDLC (High-Level Data Link Control), LAP (Link Access Procedure) y LAPB (Link Access Procedure Balanced).Este protocolo permite la comunicación en cuatro formas diferentes:

1. Punto a Punto.- Comunicación entre dos nodos, uno primario y otro secundario. Los términos primario y secundario son sinónimos a los términos maestro y esclavo respectivamente.

2. Multipunto.- Comunicación entre un nodo primario y varios nodos secundarios.

3. Lazo.- Una topología en forma de lazo, en donde el primario es conectado al primero y al último secundario.

4. Hub go-ahead.- Involucran un canal interno y un canal externo. El primario se comunica con los secundarios por el canal externo, mientras que los secundarios con el primario a través del canal interno.

HDLC.- Su nombre proviene de las siglas en ingles de *High-Level Data Link Control*. Es un protocolo de propósito general, basado en bits, utilizado tanto para la comunicación punto-punto como la comunicación multipunto. Es el resultado de variaciones del protocolo SDLC.

Dentro de las similitudes con el protocolo SDLC tenemos que los dos manejan la misma estructura de la trama. Como diferencias tenemos que HDLC solo soporta comunicación punto a punto y multipunto. Otra diferencia es que HDLC puede trabajar en tres modos de operación, a diferencia de SDLC que solo posee uno. Los tres modos de operación de HDLC son:

1. NRM (Normal Response Mode), usado cuando solo la estación primaria (master) es la única que puede empezar una transmisión. Este es el único modo que soporta SDLC.

2. ARM (Asynchronous Response Mode), permite que un secundario (slave) comience la comunicación con un primario (master), sin la necesidad del permiso de este último.

3. AMB (Asynchronuos Balanced Mode), en el uso de este modo todas las estaciones tiene la posibilidad de trabajar como maestros o esclavos y empezar la transmisión.

Las variaciones de este protocolo han resultado en otros como el PPP, SLIP y el bien conocido IEEE 802.2 (LLC).

IEEE 802.2.- Este estándar tiene gran popularidad en las redes LAN y está interrelacionado con otros protocolos tales como IEEE 802.3, IEEE 802.4, y IEEE 802.5. Los servicios que presta son:

Servicio de entrega no confiable de paquetes (tipo 1).- Esto quiere decir que el protocolo no confirma que la información haya llegado, este trabajo lo realizan las capas superiores.

Servicio con acuse de paquetes (tipo 2).- No establece una conexión entre los nodos involucrados, pero soporta acuse de recepción de la información, como una manera de control. Este servicio en particular es muy utilizado en ambientes de automatización industrial.

Servicio de entrega confiable "orientado a conexión" (tipo 3).- Este servicio establece una conexión lógica entre los nodos involucrados, luego de esto empieza el envío de información.

Finalmente existen protocolos propietarios que realizan las tareas al nivel de las capas de enlace, alguno de ellos son:

FDL.- Por las siglas en ingles de *Fieldbus Data Link*, y al igual que los demás protocolos de esta capa se encarga de la comunicación con los protocolos de transmisión y suma a la trama valores de seguridad y detección de errores. Es un protocolo utilizado por las redes Profibus. Este protocolo de la capa de enlace maneja dos tipos de acceso al medio. El primero es del tipo paso de testigo, utilizado para la comunicación entre nodos primarios (masters). El segundo tipo de acceso al medio es del tipo master/slave. Este protocolo establece una lista de estaciones activas (LSA) para saber de cuales estaciones esperar respuesta y de cuales no.

Modbus Link Layer.- Los equipos Modbus utilizan un acceso al medio del tipo Master/Slave, esto quiere decir que solamente la estación Master puede iniciar la

comunicación. Dentro de esta capa de enlace se distinguen dos subcapas. La primera es la que especifica el acceso el medio, la cual en este caso es del tipo Masters/Slave. La segunda viene dada por los modos transmisión, esta puede ser de dos tipos RTU y ASCII. Estos modos definen el tamaño de la trama y su manipulación.

CAN Link Layer.- Los equipos que utilizan las redes CAN utilizan un acceso al medio del tipo Productor/Consumidor, es decir, no basan el envío de la información en direcciones de dispositivos, sino los identificadores de los mensajes. Un nodo (productor) envía un mensaje de multidifusión al medio, todos los nodos conectados reciben el mensaje, lo filtran y deciden procesarlo o no (nodos consumidores).

Capa de Aplicación (Capa 7).- La capa de aplicación es la encargada de la generación de instrucciones y de la correcta interpretación de la información para que sea útilmente usada por el usuario final. A nivel industrial esta capa se la maneja, en su mayoría, con la administración de mensajes, es a este nivel en donde los protocolos hacen la diferencia, ya que dependiendo de su complejidad pueden contener mayor capacidad de mensajes o instrucciones posibles de ser generadas.

MMS (Manufacturing Message Specification).- Se define como un servicio de mensajería para entornos industriales, fue diseñado con el propósito de brindar soporte a comunicaciones entre dispositivos inteligentes y aplicaciones industriales, en tiempo real. Este tipo de servicio no se complica por la existencia de equipos desarrollados por varios fabricantes, ya que basa su funcionamiento en el desarrollo de un modelo virtual. Este modelo virtual contiene solamente el comportamiento esencial del dispositivo y de la aplicación, dejando a un lado las especificaciones de capas inferiores. Estos modelos virtuales son construidos en base a herramientas de modelamiento como objetos modelos [8].

Los servicios de mensajería proporcionados por MMS son lo suficientemente genéricos para ser apropiados para la amplia variedad de dispositivos, aplicaciones

e industrias. Por ejemplo, el servicio de lectura de MMS, permite a una aplicación leer el valor de una variable desde otra aplicación o dispositivo.

Los beneficios del uso de MMS se evidencian cuando se desea implementar un sistema de comunicaciones común para una gran variedad de dispositivos de comunicaciones. Estos beneficios se centran en tres aspectos claves que son la interoperabilidad que presta para la comunicación entre diferentes tipos de aplicaciones dentro de una red, la independencia de los dispositivos reales ubicados en la red, gracias al uso de modelos virtuales los cuales dan una idea general para todos los dispositivos y la facilidad con que sus funciones pueden acceder a los datos de información de las diferentes aplicaciones [8].

FTAM File Transfer Access Managment.- Es un protocolo de la capa de aplicación que permite la transferencia y acceso a archivos. Su comportamiento es similar al protocolo FTP(File Transfer Protocol) de la pila TCP/IP, para la transferencia de archivos y similar a NFS(Network File System) la administración de archivos. Fue diseñado con el propósito de brindar a la capa de usuario la posibilidad de acceder a archivos de diversos sistemas que usan implementaciones compatibles con FTAM. Es un protocolo orientado a conexión el cual guarda en un servidor información referente sobre el usuario y la sesión establecida, hasta que esta termine. De igual manera a las anteriores aplicaciones, usa una comunicación cliente/servidor. FTAM usa el concepto de un almacén de de archivos virtual, el cual le da una vista común de todos los archivos. FTAM presenta las siguientes clases de servicios:

- *Clase de Servicio de Transferencia*.- permite el intercambio de archivos o partes de archivos ente dispositivos FTAM.

- Clase de Servicio de Acceso.- Permite la inicialización de un sistema para desempeñar algunas operaciones.

- Clase de Servicio de Administración.- Permite que el usuario tenga control sobre el almacén virtual de archivos, para crear o borrar archivos o para leer y modificar atributos.

- *Clase de Servicio de Transferencia y Administración.*- La combinación de las capacidades de transferencia con las de administración permiten implementar la navegación en directorios y la ejecución funciones simples.

FMS (Fielbus Message Specification).- Es un protocolo a nivel de la capa de aplicación, constituido por un conjunto de subfunciones del protocolo MMS. De igual manera se definen objetos los cuales se asocian son recursos y procedimiento de los elementos reales. El objeto mas importante es el Dispositivo Virtual de Campo (Virtual Field Device-VFD).Es un modelo que representa de manera abstracta el comportamiento de los dispositivos reales, dicho modelo virtual puede ser usado para cualquier dispositivo y los servicios son similares a los MMS, tales como invocación de programas, lectura de variables, dominio de memoria, etc. Este tipo de protocolo de aplicación es altamente usado en redes FIELDBUS y PROFIBUS.

Existen protocolos diseñados para permitir la interconexión de sistemas computacionales en el nivel de redes de factoría (según los niveles industriales), dichos protocolos también presentan una estructura basa en el modelo de referencia OSI. A continuación se hará referencia a estos protocolos:

Protocolo MAP.- Es el protocolo de automatización de manufactura (Manufacturing Automation Protocol), ampliamente usado en el control de procesos industriales. Su estructura cubre las 7 capas del modelo OSI y se describe de la siguiente manera [7]:

Capa Física.- Uso del estándar IEEE 802.4, paso de testigo en bus.
Capa de Enlace.- Uso del estándar 802.2, conocido como LLC.
Capa de Red.- Uso del protocolo ISO 8473, similar al bien conocido IP.
Capa de Transporte.- Basado en el protocolo ISO 8073.
Capa de Sesión.- Basado en el protocolo ISO 8327.

Capa de Presentación.- Basado en el protocolo ISO 8823.

Capa de Aplicación.- Basado en los protocolos MMS, FTAM y Servicio de Directorios.

Dentro de la lógica de funcionamiento del protocolo MAP no están considerada la comunicación con los dispositivos conectados a las redes que ocupan los niveles más bajos dentro de los niveles industriales. MAP asume que dicha información ya está presente en dispositivos más inteligentes como PCLs y PCs dedicadas. Debido a que se supone que todos los nodos pertenecientes a una red MAP tienen implementados todas las capas, existe demora en las comunicaciones y un nivel de complejidad extra en los procesos. Las características más destacables de este protocolo son [7]:

- Todas las señales se transmiten moduladas en frecuencia.
- Ancho de Banda de 6Mhz.
- Número de nodos 10000 y la distancia entre ellos 10km.

Protocolo Mini-MAP.- Este protocolo es una versión mas pequeña del protocolo MAP, es más pequeña porque a diferencia del MAP, solo tiene especificadas las capas 1,2 y 7 del modelo de referencia OSI, con esto se gana velocidad en su funcionamiento y se reduce complejidad en la comunicación. La descripción de sus capas es la siguiente [7]:

Capa Física.- Usa el estándar IEEE 802.4, pero basado en transmisión por banda portadora.

Capa de Enlace.- Hace uso del protocolo IEEE 802.2 tipo 3, conocido como "orientado a conexión", de esta manera la estación secundaria responde a la petición de una estación primaria de una manera inmediata.

Capa de Aplicación.- Hace uso de los mismo servicios MMS que el protocolo MAP original.

La interoperabilidad de estas dos redes es posible, siempre y cuando se haga uso de un gateway que se encargue de llenar las capas vacías en la red Mini-MAP. Sus características más destacables son:

- Topología del tipo bus.
- El número máximo de nodos es de 64.
- Transmisión del tipo síncrono.

Protocolo CNMA.- Su nombre viene de las siglas en ingles de Communications Network for Manufacturing Applications. Fue diseñado para especificar, implementar y validar un sistema de comunicaciones abierto. CNMA desarrolla una arquitectura de comunicaciones que sirva como herramienta para el desarrollo rápido de pasarelas para redes no basadas en el modelo OSI. Es un protocolo formado de 7 capas, similar al protocolo MAP, pero con ciertas variaciones en la capa física y en la capa de aplicación. En la capa física difiere del protocolo MAP ya que soporta tanto la transmisión por paso de testigo en bus (IEEE 804.2) y la transmisión por contienda (IEEE 802.3). A nivel de la capa de aplicación, adicional a los servicios ya presentados por MAP, que son: FTAM, Servicio de Directorios, Administración de Red, y MMS. CNMA suma a sus servicios de aplicaciones el protocolo de bases de datos remotas [7].

Actualmente existen en el mercado varias redes de campo que basan su funcionamiento en las funciones que desempeñan los servicios de la capa de aplicación desarrollados por los propietarios de algunos productos. La mayoría de estoy establecen una lista de funciones posibles para la manipulación de la información, permitiendo así un intercambio seguro y eficiente para la capa siguiente, conocida como la capa de USUARIO.

Capa de Usuario.- Es una capa, que a pesar de no estar contenida en el modelo de referencia OSI, siempre esta presente en la integración de redes industriales, ya que es aquí en donde los integradores hacen uso de los servicios de la capa de aplicación para la manipulación de información y desarrollo de aplicaciones que cumplan con los requerimientos individuales de cada industria. El *Figura 1.7* da una referencia de la pila de protocolos del modelo OSI, con una pila de protocolos que describe la arquitectura de los buses de campo [7].

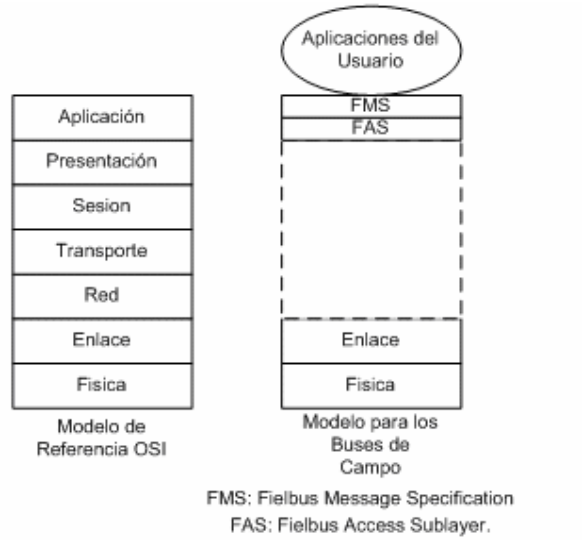


Figura 1.7. Comparación entre Modelo de Referencia Comparación entre y Modelo de Bus de Campo.

Con ciertas excepciones como MAP y CNMA, los demás buses de campo cumplen con el esquema mostrado por la *Figura 1.7.*

1.3 Arquitecturas de Integración Serial-Ethernet.

Muchos de los productos industriales fueron diseñados sin la visión de un crecimiento hacia la tecnología Ethernet, pero la mayoría incluye interfaces seriales. Los beneficios de la migración a interfaces Ethernet industriales en productos antiguos y nuevos son ahora poderosos y fascinantes. La proliferación de la tecnología Ethernet en sistemas de automatización esta en crecimiento, forzando así a los vendedores de productos de control industrial adopten esta nueva tendencia [9]. Los fabricantes actuales tienen una agresiva estrategia para introducir en el mercado los nuevo equipos con posibilidades de integración en redes Ethernet, y han desarrollado muchas aplicaciones que son muy útiles en entornos industriales como monitoreo WEB, reportes via e-mails, accesibilidad global.

Para realizar una integración serial-Ethernet efectiva es importante identificar tres componentes básicos que se encuentran presentes en todo protocolo industrial [9]:

1. Modelo de Comunicación.- El modelo de comunicación define la manera en que dos nodos de la red intercambian información. Esta información puede ser de dos tipos:

Mensajería Explícita: Son mensajes del tipo solicitud/respuesta, usados para fijar parámetros de configuración.

Mensajería Implícita: Son mensajes que transportan información sobre el proceso E/S hacia y desde el dispositivo.

2. Estructura de la Información.- La estructura de la información de un protocolo proporciona un esquema para la organización, almacenamiento y recuperación de la información dentro del dispositivo.

Un ejemplo específico es el modelo de objetos dentro de CIP (por las siglas en ingles de Common Industrial Protocol), el cual se identifica como la capa de aplicación de los protocolos DeviceNet, ControlNet y EtherNet/IP. La información es organizada en una jerarquía de Objeto/Instancia/Atributo. En el nivel más alto están los Objetos, los cuales agrupan tipos similares de datos tales como información de identificación de dispositivos, parámetros de conexión, información específica para la aplicación [9].

Un objeto puede contener uno o más Instancias, por ejemplo, un dispositivo con 8 salidas digitales podría contener 8 instancias del Objeto de Salidas Digitales. En el siguiente nivel están los atributos de las Instantes, los cuales son los datos específicos de parámetros de configuración, datos de aplicación, y demás [9].

Un ejemplo de una diferente estructura de datos está contenido dentro del protocolo de aplicación Modbus el cual es usado por Modbus RTU/ASCII sobre línea serial y Modbus/TCP sobre Ethernet. Los datos de Modbus están organizados dentro de arreglos de registros de 16-bits, 1-bit de entradas, y 1-bit de bobinas (salidas).

Cada registro, entrada y salida, es asignada a una dirección usada para localizar el dato que va a ser leído y/o escrito [9].

La organización de la información dentro de CIP y Modbus es algo diferente, pero cada uno tiene sus propias ventajas y desventajas.

3. Servicios.- Agrupa los comandos y funciones que el dispositivo soporta. Algunos de los servicios comunes son Lectura de Datos, Escritura de Datos, Reset, Abrir Conexión y Cerrar Conexión. A pesar de sonar simple conceptualmente, cada protocolo implementa muchos de estos en una manera diferente y esto se debe a sus estructuras de datos y modelos de comunicación que les subsiguen dentro de una jerarquía vertical.

Por ejemplo, para leer un dato dentro de CIP, se hace uso del mensaje *Get_Attribute_Single,* para el cual se debe especificar el ID del objeto, el ID de la instancia y el ID del atributo, todo esto para localizar el dato deseado [9].

El servicio *Get_Attribute_Single* puede leer cualquier atributo dentro de cualquier objeto, y algunos dispositivos pueden adicionalmente implementar el servicio *Get_Attribute_All* para leer todos los atributos de un objeto. CIP no define un servicio para leer solo un grupo de atributos. Dentro de Modbus, un tipo de servicio para leer información desde un registro especifica la dirección de un registro base donde empieza la lectura y el número de registros que van a ser leídos. Muy similares son los servicios definidos para el manejo de entradas y salidas [9].

Todas las características antes mencionadas son importantes al momento de desarrollar la migración de un protocolo serial a la tecnología TCP/IP. La conversión de los datos seriales, desde un dispositivo existente, en paquetes Ethernet puede ser trivial dependiendo de la compatibilidad y similitud de los protocolos requeridos en cada uno de los puertos.

El hecho de desarrollar un gateway transparente para los datos desde el puerto serial al puerto Ethernet, implica la necesidad de que los dispositivos del lado de la red Ethernet puedan interpretar y entender el formato del dato que les es entregado. Y en muchas

ocasiones este no es el caso. Si el puerto serial del dispositivo soporta el protocolo Modbus y se requiere una conexión a una red Ethernet Modbus/TCP, afortunadamente la capa de aplicación de Modbus es muy manejable desde su formato serial hacia su formato Ethernet.

Una arquitectura clásica que permitiría la integración de datos provenientes de un protocolo serial a una red Ethernet serial mediante el uso de un computador personal el cual reciba la información por un puerto serial. Dicha información es manipulada dentro del computador mediante el uso de un programa de desarrollo de HMIs, como por ejemplo Intouch o Labview, y luego esta información sería compartida en un ambiente Ethernet gracias a las posibilidades de conectividad que presentan actualmente los computadores personales.

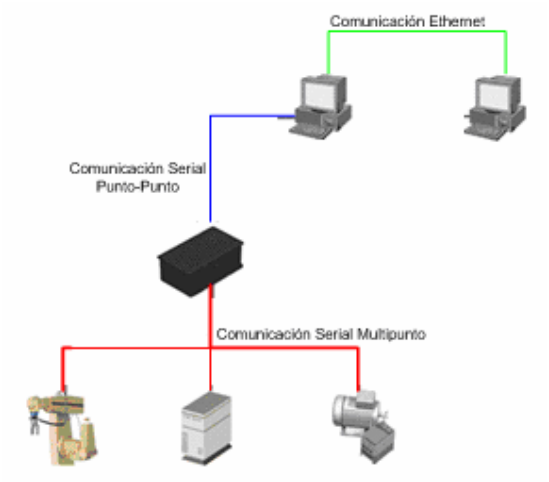


Figura 1.8. Diagrama Serial-Ethernet en base a PCs

En el *Figura 1.8* se muestra que para poder migrar datos de una red serial sería necesario una PC que reciba los datos en forma serial, y luego mediante el uso de un programa de desarrollo de HMIs, compartir estos datos en una red TCP/IP, pero con computadoras que posean el mismo software de aplicación. Esto indudablemente es un gasto mayor debido a la inversión en la adquisición del software de desarrollo y en equipos y consumo de energía.

Actualmente los equipo nuevos vienen con una interfaz de comunicaciones TCP/IP, permitiendo así una interconexión más económica.

Para los equipos antiguos que no poseen una interfaz de comunicación TCP/IP incorporada, las empresas están desarrollando pasarelas industriales. Algunos de estas pasarelas son equipos conversores de protocolos de capaz superiores y otros solo son acopladores de capaz inferiores. Las pasarelas conversores de protocolos de capa superior son dispositivos que permiten, además de la transmisión confiable de una tecnología a otra, el desarrollo de aplicaciones en base a protocolos que soportan funciones y mensajería. Las pasarelas que solo son acopladores de capaz inferiores, convierten niveles de voltaje y sincronización de los bits para que la aplicación final se encargue de interpretar los datos.

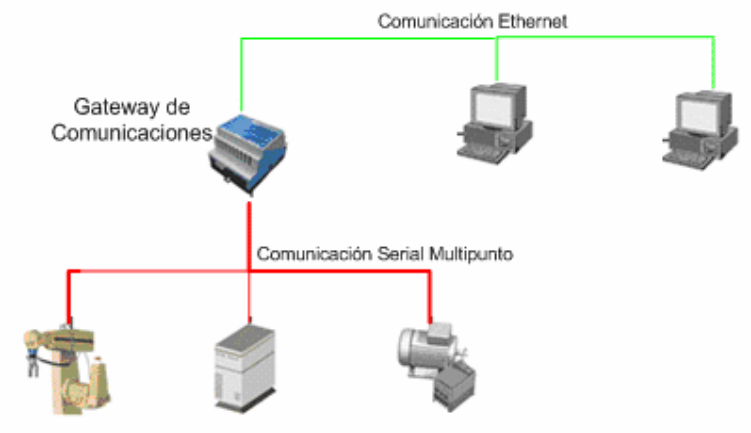


Grafico 1.9. Diagrama Serial-Ethernet en base a pasarelas propietarios.

Básicamente, todos los detalles del protocolo de red están ahora embebidos en nuevos componentes electrónicos y pueden ser utilizados como puentes para la información de un producto especifico dentro de una red industrial.

Esta es una buena opción que ahorra tiempo y facilita la conectividad entre redes seriales y Ethernet. Pero su costo puede ser algo no muy favorable para las pequeñas industrias, y debido a que son desarrollados en empresas de producción en masa, no se le puede hacer algunas modificaciones que permitan su integración con redes seriales no conocidas en el mercado, pero que en ocasiones se encuentran presentes en la industria.

Otra alternativa es el desarrollo de estos pasarelas de comunicaciones mediante el uso de microcontroladores o dispositivos embebidos que contengan toda la pila de

comunicaciones TCP/IP. Esta solución es una alternativa más económica y permite la adaptación de cualquier tipo de bus de campo, a la tecnología Ethernet. Adicional a estoy da libertan al desarrollador de crear servicios que realmente cubran las necesidades de las empresas nacionales, que difieren en muchas maneras unas de las otras.

Grafico 1.10. Diagrama Serial-Ethernet en base a controladores embebidos.

CAPITULO 2

MARCO TEORICO

2.1 Protocolo Modbus.

2.1.1 Introducción al Protocolo Modbus.

El protocolo MODBUS fue desarrollado por la empresa MODICON en 1979, con el objeto de permitir conectividad entre sus dispositivos inteligentes, tales como los Controladores Lógicos Programables (PLC). Es un protocolo que con referencia al modelo OSI ocupa la capa de aplicación del mismo, está basado en una estructura de mensajería que permite el intercambio de información en base a registros.

El protocolo MODBUS describe el proceso que usa un controlador para pedir acceso a otro dispositivo, cómo responderá a las peticiones desde otros dispositivos y cómo se detectarán y notificarán los errores. Establece un formato común para la disposición y contenido de los campos de mensaje [1].

Los controladores Modicon poseen una interfaz serie compatible RS-232C para la comunicación Modbus punto-punto. Otros dispositivos poseen una interfaz RS-485, que permite la comunicación punto-multipunto. Todas las especificaciones mecánicas, eléctricas y lógicas de funcionamiento para esta interfaz se encuentran descritas en las norma EIA RS-232C y EIA RS-485, respectivamente. Los controladores pueden ser conectados en red directamente o vía módems. Los controladores usando una técnica de acceso al medio del tipo maestro/esclavo, esto quiere decir que, 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 [1].

A pesar de los años, MODBUS es un protocolo muy usado a nivel industrial, debido a que es un protocolo abierto y que posee una estructura de mensajería no complicada para su entendimiento. Su implementación ya no solo se encuentra en líneas seriales convencionales como la RS-232 o RS-485, su crecimiento en la industria ha permitido su desarrollo en redes Ethernet y actualmente la comunidad de Internet tiene acceso a este protocolo a través del puerto reservado 502.

2.1.2 Capa Física del Protocolo Modbus.

Para la implementación de la capa física del protocolo MODBUS se tiene dos opciones. La primera es bajo el estándar RS485 y la segunda bajo el estándar RS232. Bajo el estándar RS485 se tiene dos opciones de implementación, la interfaz a 2 cables (2W) o la interfaz a 4 cables (4W). Mayores detalles de dicha opciones serán detalladas en este documentos. Como ya se señaló anteriormente la interfaz serial RS232 es utilizada para establecer comunicación punto-punto, mientras que la interfaz RS485 se la usa para comunicaciones punto-punto o punto-multipunto. Para mayor detalle en las características de estos dos estándares se pueden consultar los escritos oficiales de la norma TIA/EIA-232 TIA/EIA-485, respectivamente.

Características de la Comunicación vía RS232

La comunicación vía el estándar RS232 se la realiza entre dos equipos cuando la distancia no sea mayor a 20m (recomendado distancias no mayores a 15m). El cable que se debe usar para esta conexión puede ser un cable categoría 5. Según las normas de implementación de este protocolo, la capacitancia máxima, con respecto a tierra, del cableado debe ser no mayor a 2500pf. De esta característica se deduce el uso de un cable de 100pf/m para una distancia máxima de 25m [2].

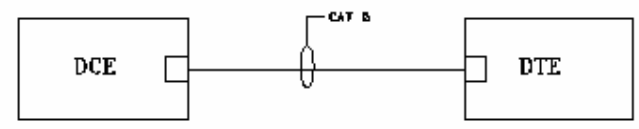


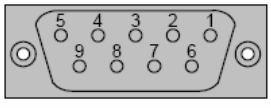
Figura 2.1. Esquema Típico Comunicación RS-232

Para la correcta implementación se debe especificar si el equipo que se va a conectar se lo debe considerar como DCE o como DTE. Una manera simple de distinguir el tipo de dispositivo es a través del tipo de conector que posee. Siendo un conector DB9 macho para los equipos DTE y un conector DB9 hembra para los equipos DCE. Para la conexión de un equipo DTE y un equipo DCE se utiliza un cable con configuración de sus pines en modo directo, mientras q para la conexión de dos equipos DTE o dos equipos DCE entre si se hace uso de un cable NULL MODEM, en donde se cruzan los pines de transmisión y recepción de los equipos respectivamente. Para mayor detalles para la asignación de los pines para cada uno de los conectores antes enunciado se puede hacer referencia al documento **EIA-RS Serial Communications All Models User's Guide** de la empresa General Electric**.** Según el documento **MODBUS Over Serial Line Specification and Implementation Guide V1.01**, proporcionado por MODBUS ORG. Solos los pines de transmisión, recepción y de tierra deben ser cableados en una implementación física de este protocolo bajo el estándar RS232.

Además del conector DB9, ampliamente relacionado con el estándar RS232, se puede hacer unos de un conector RJ45. La *Tabla 2.1* describe respectiva asignación de pines para su correcto funcionamiento.

DCE Underlined pins can be output			Circuit			DTE Underlined pins can be output		
Pin on RJ45	Pin on D9-shell	Level of requirement	Name	Description	RS232 Source	Level of requirement	Pin on RJ45	Pin on D9-shell
1	2	required	TXD	Transmitted Data	DTE	required	2	3
2	3	required	RXD	Received Data	DCE	required	1	2
3	7	optional	CTS	Clear to Send	DCE	optional	6	8
6	8	optional	RTS	Request to Send	DTE	optional	3	7
8	5	required	Common	Signal Common	--	required	8	5

Tabla 2.1. Asignación de pines en comunicación RS-232



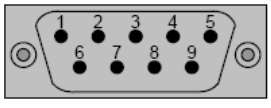


Figura 2.2 Distribución de pines en conector DB9

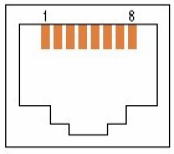


Figura 2.3 Distribución de pines en conector RJ45

El *Figura 2.2* y el *Figura 2.3* muestran como es la secuencia de numeración de los pines, para tener en consideración el momento de implementar esta opción de conectividad. Las velocidades de transmisión se rigen enteramente al estándar TIA/EIA-232.

Características comunicación vía RS485

El estándar RS485 es un estándar de comunicación serial digital sobre líneas balanceadas, esto quiere decir que los valores lógicos 0 y 1 son producidos por cambios de polaridad entre las líneas de transmisión, mas no como variaciones de voltaje dentro de un rango definido, como sucede en el estándar RS232. Este estándar permite la comunicación punto a punto, así como la comunicación punto multipunto.

El bus serial de comunicaciones es un cable principal conocido como la troncal, de este cable principal se toma derivaciones que llegan a cada uno de los equipos conectados a esta red multipuntos, a estos equipos se los conoce como nodos. Existen tres formas en las que un nodo puede integrarse al cable principal de la red.

La primera opción es mediante el uso de unos dispositivos conocidos como Active Taps, estos dispositivos poseen un transductor de comunicaciones, son usados con dispositivos que no tienen dicho transductor, brindándoles la posibilidad de conectividad sobre la red. La segunda opción es usando Passive Taps, estos dispositivos permiten conectividad con nodos que si posee transductores de comunicación integrados en si mismos. Finalmente los nodos pueden ser conectados directamente a bus principal (troncal), este tipo de conexión se conoce como conexión en cadena (daisy-chain).

El estándar RS485 puede ser implementado sobre un par balanceado (D0 D1) o puede ser implementado sobre dos pares balanceados (TXD0 TXD1 RXD0 RXD1), en ambos casos es necesario sumar un cable extra (común) que se debe conectar a la tierra del sistema. Para mayores detalles sobre configuraciones a un par y a dos pares referirse al documento **MODBUS Over Serial Line Specification and Implementation Guide.**

En una red RS485 se pueden conectar hasta 32 dispositivos sin la necesidad de un repetidor. Adicional a esto es importante considerar la distancia que se va a recorrer y el medio físico que se va a utilizar. Por ejemplo, para una taza de transmisión máxima de 9600 baudios y un cable categoría 26AWG, la longitud máxima es de 1000m. Adicional a esto las derivaciones deben ser cortas, nunca deben ser mayores a 20m [2].

A diferencia de una red RS232, en una red RS485 se necesitan colocar terminadores de línea a cada extremo del bus troncal para evitar perdidas por desacoples de impedancias en las líneas de transmisión. Para esto se hace uso de resistencias de 150ohm a 0.5W [2].

2.1.3 Capa de Enlace del Protocolo Modbus.

Para un mejor entendimiento de la Capa de Enlace del Protocolo Modbus se dividirá su explicación en dos partes. La primera que explique como se accede al medio y la segunda explicara los modos de transmisión que este protocolo posee.

1. **Acceso al Medio.**

   Una red Modbus esta constituida por un dispositivo MAESTRO y hasta 247 dispositivos ESCLAVOS. Solamente el dispositivo MAESTRO es el que puede empezar las transmisiones. Para esto, estas pueden ser de dos tipos: monodifusión y multidifusión. Los primeros son mensajes direccionados a un solo dispositivo, mientras que los mensajes multidifusión son enviados a todos los dispositivos de la red. Los dispositivos ESCLAVOS solo pueden transmitir únicamente como respuesta a peticiones solicitadas por el dispositivo MAESTRO.

2. **Modos de Transmisión.**

   El protocolo Modbus en su capa de enlace tiene dos tipos de modos de transmisión posibles. Estos son el Modo RTU y el Modo ASCII. Estos modos de transmisión definen como esta estructurada la trama de información que se va a transmitir serialmente. Es decir define los campos de bits y su contenido. Es importante señalar que el tipo de transmisión debe ser el mismo para todos los dispositivos dentro conectados al mismo bus troncal.

a) Modo RTU.

El en modo de transmisión RTU los mensajes provenientes de la capa superior, la capa de aplicación, son manejados teniendo como unidad el byte (8-bits), y estos a su ves son manejados como dos caracteres hexadecimales de 4-bits de longitud. Es así como se establece una secuencia de envío de la información para cada byte según el *Figura 2.4*

Figura 2.4. Distribución de bits en un byte de comunicación

Los mensajes deben ser enviados en un flujo de caracteres continuos. Y la secuencia de envío para un mensaje está descrita por el *Figura 2.5*.

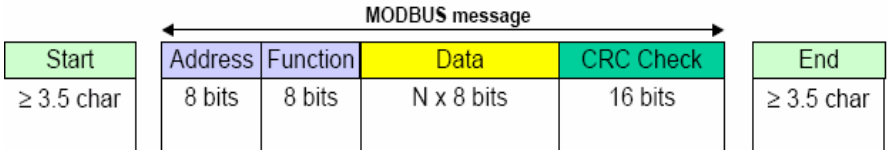


Figura 2.5. Secuencia de envió de datos en comunicación MODBUS

A continuación se explicará brevemente cada uno de los campos identificados en el *Gráfico 2.5*

- **Start (Inicio).-** Es un intervalo de silencio que permite a los dispositivos distinguir el comienzo de la transmisión de un nuevo mensaje. Su duración es de un valor mayor o igual al tiempo de transmisión de un carácter.

- **Address (Dirección).-** Este campo está formado por un byte, y representa en su mayoría la dirección que tiene un dispositivo dentro de la red. Los Dispositivos maestros o esclavos pueden tomar valores de dirección de 1 a 247 (valores decimales). La dirección 0 se la utiliza para transmisión de multidifusión y las direcciones de 248 y 255 se encuentran reservadas.

- **Function (Código de Función).-** En este campo se representa con 1 byte el tipo de función que el mensaje lleva. Es decir, describe la acción que debe realizar el esclavo o el maestro cuando reciben el mensaje. Más adelante se detallara de mejor manera cada una de las funciones posibles dentro de este protocolo (capa de aplicación).

- **Data (Datos de Información).-** Este campo tiene una longitud variable de bytes. Dentro de este campo se lleva información que junto con el código de la función describen de una manera completa y exacta un mensaje del protocolo MODBUS.

- **CRC Check (Cheque de Redundancia Ciclica)** Este campo tiene una longitud de dos bytes. El valor que toma este campo es calculado por el dispositivo que envía el mensaje, mediante el uso del método de chequeo por redundancia cíclica. El dispositivo que recibe el mensaje esta en cargado de recalcular este valor para comprobar la calidad del mensaje. A continuación se detallan los pasos a seguir para el cálculo CRC[2]:

    1. Cargar un registro de 16-bits con el valor hexadecimal 0xFFFF, es decir, todos los bits en 1 lógico. Este registro se llamará el registro CRC.

    2. Realizar una operación de OR exclusivo entre el byte de menor orden del registro CRC y el primer byte del mensaje en

cuestión. El resultado almacenar en el byte de menor orden del registro CRC.

3. Desplazar un bit el registro CRC hacia la derecha, llenando con cero el MSB. A continuación se analizará el valor del LSB.

4. Si LSB fue 0, se debe repetir el paso 3, es decir, desplazar el registro CRC una vez mas hacia la derecha. Si el valor del LSB fue 1, se procede a ejecutar la operación de OR exclusiva entre el registro CRC y un valor polinomial, el cual es 0xA001.

5. Repetir los pasos 3 y 4 hasta que se hayan realizado 8 desplazamientos. Luego de esto se habrá realizado el chequeo del primer byte del mensaje.

6. Repetir los pasos del 2 al 5con el siguiente byte del mensaje. Realizar esto hasta acabar con todos los bytes contenidos en el mensaje.

7. El contenido final del registro CRC es el valor que va a ocupar el espacio CRC en el mensaje en cuestión.

8. Finalmente, antes de colocar el valor del registro dentro del campo CRC el byte menos significativo debe intercambiar posiciones con el byte más significativo.

Para mayores detalles sobre este método de chequeo por redundancia cíclica consultar el documento **MODBUS Over Serial Line Specification and Implementation Guide** que se lo puede adquirir por la organización MODBUS**.**

b) Modo ASCII.

En este modo de transmisión la codificación utilizada es la codificación ASCII (American Standard Code for Information Interchange). Cada byte del mensaje MODBUS es enviado como dos caracteres ASCII. Por este motivo este modo de transmisión es menos efectivo que el modo RTU. No se dará mayor información sobre este modo de transmisión debido a que no es caso de desarrollo para el presente proyecto. Toda la información referente a este tema se puede encontrar de una manera detallada en el documento **MODBUS Over Serial Line Specification and Implementation Guide.**

2.1.4 Capa de Aplicación del Protocolo Modbus.

La capa de Aplicación del Protocolo Modbus esta basada en un conjunto de funciones las cuales pueden ser implementadas por los equipo y permiten el intercambio de información y la ejecución de lazos de control, en los equipos destinados a este propósito. Independientemente de las capas inferiores, la capa de aplicación define una estructura de mensaje denominada ADU, por sus siglas en ingles de **Application Data Unit**. Esta estructura esta formada por dos partes. La primera es el código de la función y la segunda es información adicional para el desempeño de dicha función.

Como se señaló anteriormente el protocolo Modbus es un protocolo de comunicaciones Cliente/Servidor. Esto quiere decir que el cliente es quien construye el mensaje y empieza la transmisión, y el servidor es quien contiene la información y responde a los requerimientos del cliente. La longitud del campo FUNCION es de 1 byte, esto quiere decir, que puede tomar valores desde 0 a 255 (en notación decimal). En el caso práctico ninguna función tiene el código de 0. Adicional a esto los valores del 128 al 255 están reservados para respuestas de excepción. Estas respuestas de excepción son generadas cuando una función no es aplicable a un determinado dispositivo. En este caso, el dispositivo en cuestión responde con el mismo código de función solicitado por el cliente, modificando el bit más significativo a un valor de 1 lógico.

Antes de identificar los tipos de mensajes que pueden ser enviados en el protocolo Modbus es importante entender en que forma se distribuye la información dentro de los dispositivos que basan su comunicación en Modbus.

Dentro de la estructura de datos de Modbus se identifican 4 tipos de datos, los cuales se enumeran a continuación:

1. *Entradas Discretas (Discrete Inputs).*- Es un dato de solo lectura, de longitud de un bit. Este valor puede ser proporcionado por un sistema de entradas y salidas.

2. *Bobinas (Coils).*- Es un dato de lectura y escritura, de longitud de un bit. Su valor puede ser cambiado por un programa de aplicación.

3. *Registro de Entrada (Input Registers).*- Es un dato de solo lectura, de longitud de 16 bits. Este valor puede ser alterado por un sistema de entradas y salidas.

4. *Registros de Memoria (Holding Registers).*- Es un dato de lectura y de escritura, de longitud de 16 bits. Este valor puede ser manpulado por un programa de aplicación.

Como esté distribuida la información dentro del dispositivo que lo contenga es responsabilidad absoluta de los diseñadores del dispositivo. Teniendo en cuenta que el direccionamiento debe ser claro, transparente y seguir los siguientes puntos, concebidos dentro de los manuales del estándar Modbus[3]:

1. En una Unidad de Datos de Protocolo (Protocol Data Unit) Modbus cada dato es diseccionado desde 0 hasta 65535[3].

2. Dentro de cualquiera de los bloques de datos de los 4 tipos antes enunciados en este documento, los ítems contenidos dentro están enumerados desde el 1 hasta n [3].

Es importante señalar, por motivos de diseño, que la información es manipulada en base a una filosofía “big-Endian”. Esto quiere decir que cuando un dato dentro de un mensaje tiene una longitud mayor a un byte, el momento de enviar el dato, se procede primero con la parte más significativa del dato (longitud máxima de un byte), para luego seguir con la parte menos significativa [3].

Descripción de los Códigos de Funciones:

Existen tres clases de funciones dentro de Modbus:

1. Funciones Públicas: Son funciones validadas por la comunidad MODBUS-IDA.org. Son únicas y su descripción se encuentra bien documentada y es de libre acceso.

2. Funciones definidas por el Usuario. Son códigos de funciones libres para desarrollo por parte de los diseñadores. Mediante el uso de estos códigos de funciones se pueden implementar tareas que no están estandarizadas, pero que pueden ser necesarias para ciertas aplicaciones.

3. Funciones Reservadas: Estos códigos de funciones son utilizados por empresas para la implementación de ciertos servicios en particular. No son del dominio público y no se encuentran en todos los dispositivos con comunicación Modbus.

Sin importa el tipo de función que se esté ejecutando, en su mayoría se definen tres tipos de PDUs (Protocol Data Unit)[3]:

1. PDU de petición.
2. PDU de respuesta.
3. PDU de respuesta de excepción

Básicamente su forma es la misma, pero dependiendo de la función, el campo de datos varía. Al final se anexa un cuadro que muestra todas los PDU de petición y respuesta de todas las funciones MODBUS, y las longitudes de cada uno de sus campos.

2.2 Suite de Protocolos TCP/IP.

2.2.1 Introducción.

Debido al gran contenido involucrado con la suite de protocolos TCP/IP, se requeriría de un documento muy extenso para estudiar con precisión cada detalle concerniente a este fascinante tema. Lo que se pretende dentro de esta sección es dar una idea general del protocolo, lo cual sirva para visualizar y entender de mejor manera la compatibilidad y la correspondencia al momento de realizar la migración del protocolo Modbus al protocolo TCP/IP.

En el modelo de referencia OSI se identifican 7 capas, dentro de las cuales se ubican un sin número de protocolos, cada uno de ellos con fortalezas y debilidades, lo que les hace apropiados para diferentes aplicaciones. Este modelo tiene por objeto el de servir de referencia para el desarrollo de aplicaciones en redes digitales. De la misma manera muchos de los dispositivos involucrados en dichas redes digitales se valen de este modelo para su implementación lógica del lazo de comunicaciones.

El modelo de referencia TCP/IP fue creado por el Departamento de Defensa de EE.UU (DoD), con el objeto de solucionar problemas de diseño de redes digitales [4]. Este modelo comprime las 7 capas del modelo OSI en 4 capas. A pesar de parecer mas simple, de ninguna manera el modelo TCP/IP tiene menos funcionalidad que el modelo OSI, la diferencia radica en la agrupación de los servicios y protocolos bajo otra filosofía conceptual. Debido a su simplicidad, el modelo TCP/IP es usado más ampliamente en el mercado.

En el *Figura 2.6* podemos ver de manera gráfica la relación entre estos dos modelos de referencia.

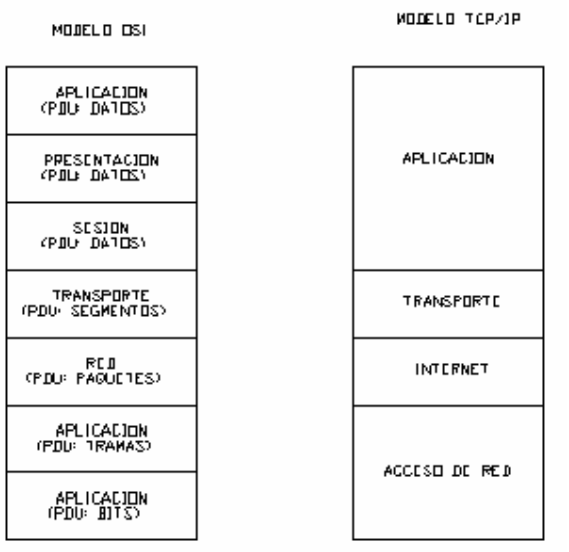


Figura 2.6. Comparación entre modelo de referencia OSI y modelo TCP/IP

Gracias a la suite de protocolos TCP/IP es posible la comunicación de dispositivos de diferentes proveedores y la interoperabilidad entre diferentes tipos de aplicaciones. En pocas palabras, gracias a TCP/IP la red más grande del mundo, el Internet, puede funcionar con tanto éxito y demanda como funciona hasta el momento.

2.2.2 Capa de Acceso de Red.

La capa de Acceso de Red cumple con las funciones desempeñadas por la capa 1 y la capa 2 del modelo OSI, la capa física y la capa de enlace de datos, respectivamente. A pesar de que existen varias tecnologías relacionadas a estos niveles funcionales, Ethernet es, en la actualidad, el conjunto de tecnologías dominante en las redes LAN (Redes de Área Local) y con sus nuevas actualizaciones como GigaEthernet también tiene amplio uso en las redes WAN (Redes de Área Extendida).

Se dice que Ethernet es un conjunto de tecnologías, por que desde sus primeras implementaciones, ha sufrido variaciones las cuales han permitido alcanzar velocidades de transmisión más rápidas y sobre un medio diferente al cobre, la fibra óptica.

Dentro de las funciones de la capa de Acceso de Red podemos enunciar a las siguientes:

1. Definir los procedimientos necesarios para que la información pueda ser transmita sobre el medio física. Dentro de esto incluimos niveles de voltaje, temporización de bits, lo cual esta asociado a las velocidades de transmisión, y conexiones físicas y conectores.

2. Encapsular los paquetes provenientes de la capa superior en tramas, para luego ser enviadas a manera de bits sobre el medio de transmisión.

3. En este nivel se suma a la información las direcciones físicas de origen y destino necesarias para que el paquete pueda viajar a través de la red.

4. Esta encargada de manejar la lógica de acceso al medio, la cual esta regida por el estándar CSMA/CD, termino que proviene de las en ingles de CARRIER SENSE MULTIPLE ACCESS WITH COLISION DETECTION, lo que en español significa Acceso Múltiple con Escucha de portadora y Detección de Colisión. Según este estándar el acceso al medio se lo puede describir de la siguiente manera. La estación que desea transmitir, escucha el medio para saber si esta o no ocupado, si el medio esta disponible, la estación en cuestión envía su información. El momento que se produzca una colisión, las estaciones involucradas detectan la colisión por una variación de voltaje en la línea. Luego de esto unos temporizadores con valores aleatorios son disparados. Una vez que dichos temporizadores terminan su cuenta, la estación vuelve a escuchar el medio, si esta disponible vuelve a enviar el mensaje que tuvo el problema de colisión, caso contrario espera hasta que el medio este libre. Por este motivo se conoce a este tipo de acceso al medio como un acceso al medio aleatorio, ya no se puede determinar de manera exacta a que momento la estación va a transmitir su información.

El formato básico de la trama sigue siendo el mismo para todas las formas de Ethernet. Cuando es necesario expandir Ethernet para agregar un nuevo medio o capacidad, el IEEE publica un nuevo suplemento del estándar 802.3. Por dicho motivo se dice que Ethernet es muy escalable, debido a que para incrementar la velocidad y ancho de banda no exige un cambio dramático en la lógica del protocolo, sino, un mejor aprovechamiento del medio y

una temporización más rápida. En el *Figura 2.7* podemos ver la distribución de campos dentro de la trama Ethernet.

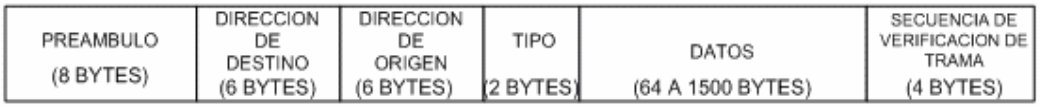


Figura 2.7. Distribución de bytes en la trama Ethernet.

2.2.3 Capa de Internet.

El trabajo de esta capa es el de seleccionar la mejor ruta para el envío de la información y conmutar los paquetes de información. Existen muchos protocolos que realizan dicha labor en esta capa, pero el protocolo que ha sido icono para esta tarea y es usado en Internet es el Protocolo de Internet (IP).

> *Protocolo IP (Internet Protocol):* Proporciona un enrutamiento de paquetes no orientado a conexión de máximo esfuerzo. Eso quiere decir, que a pesar de buscar la mejor ruta para la transmisión, no se garantiza que los paquetes lleguen a su destino. Los paquetes pueden perderse, y debido a que este protocolo no tiene funciones de control de errores ni de flujo, se lo considera un protocolo poco confiable. Las tareas de confiabilidad son proporcionadas por las capas superiores. Para mayores detalles sobre este protocolo se puede recurrir a sus especificaciones actuales enunciadas en los RFCs 791, 950, 919 y 922, actualizado en el RFC 1349. El *Figura 2.8* nos muestra la distribución de los bits dentro de un paquete IP.

Figura 2.8. Distribución de bits en un paquete IP

Además de los servicios del protocolo IP encontramos otros protocolos que desempeñan sus funciones en esta capa simultáneamente con el protocolo IP. Entre ellos tenemos:

Protocolo de Mensajes de Control en Internet (ICMP): Suministra capacidades de control y envío de mensajes [4]. Para mayores detalles se puede consultar en el documento RFC 792, actualizado en el RFC 950.

Protocolo de Resolución de Direcciones (ARP): Determina la dirección de la capa de enlace de datos, la dirección MAC, para las direcciones IP conocidas [4].

Protocolo de resolución inversa de direcciones (RARP): Determina las direcciones IP cuando se conoce la dirección MAC [4].

2.2.4 Capa de Transporte.

La capa de transporte es la tercera capa del modelo de referencia TCP/IP y se encarga de transportar y regular el flujo de información desde un nodo hacia el otro. Internamente esta capa está constituida por dos protocolos: el protocolo TCP (Transmisión Control Protocol) y el protocolo UDP (User Datagram Protocol). Básicamente la principal diferencia entre estos dos protocolos es que el TCP es un protocolo orientado a conexión, mientras que el protocolo UDP no lo es.

Protocolo TCP: Es un protocolo orientado a conexión. Esto quiere decir que la información transmitida a través de TCP se la realiza de una manera confiable. Para ello, se establece una conexión virtual previa con el nodo destino antes de empezar a transmitir los datos. Esta conexión previa se la conoce como INTERCAMBIO DE SEÑALES DE TRES VIAS. Con esto el nodo destino se pone listo para que reciba la información transmitida por el nodo origen. Adicional ha esto, TCP lleva un control de flujo de la información mediante el uso de ventanas deslizantes, las cuales establecen la cantidad de paquetes que se pueden enviar antes de recibir un acuse de recibo. Finalmente decimos que este protocolo realiza una transmisión confiable ya que se vale del uso de mensajes de acuso de recibo para generar retransmisiones de mensajes que no llegan a su destino. En el *Figura 2.9* se distinguen los diferentes campos que conforman un segmento TCP [4]. Se describe en el RFC 793 – TCP ("Transmission Control Protocol").

PUERTO ORIGEN (16 bits)	PUERTO DESTINO (16 bits)	NUMERO DE SECUENCIA (32 bits)	NUMERO ACUSE DE RECIBO (32 bits)	LONGITUD ENCABEZADO (4 bits)	RESERVADO (6 bits)	BITS DE CODIGO (6 bits)	VENTANA (16 bits)	CHECKSUM (16 bits)	URGENTE (16 bits)	OPCIONES (0 ó 32 bits)	DATOS (VARIA)

Figura 2.9. Distribución de bits en un segmento TCP

Protocolo UDP: El Protocolo de datagrama de usuario (UDP: User Datagram Protocol) es el protocolo de transporte no orientado a conexión de la pila de protocolo TCP/IP. El UDP es un protocolo simple que intercambia datagramas sin acuse de recibo ni garantía de entrega. El procesamiento de errores y la retransmisión deben ser manejados por protocolos de capa superior. El UDP no usa ventanas ni acuses de recibo de modo que la confiabilidad, de ser necesario, se suministra a través de protocolos de la capa de aplicación. El UDP está diseñado para aplicaciones que no necesitan ensamblar secuencias de segmentos. En el *Figura 2.10* se distinguen los diferentes campos que conforman un segmento UDP [4] El RFC 768 - "User Datagram Protocol" describe UDP.

PUERTO ORIGEN (16 bits)	PUERTO DESTINO (16 bits)	LONGITUD ENCABEZADO (16 bits)	CHECKSUM (16 bits)	DATOS (VARIA)

Figura 2.10. Distribución de bits en un segmento UDP.

Es importante definir lo que son los puertos, ya que estos están presentes tanto en el segmento del protocolo TCP así como en el segmento del protocolo UDP. Los números de puertos o “sockets” son unas puertas virtuales a través de las cuales se envía la información. Los números de puerto se utilizan para mantener un registro de las distintas conversaciones que atraviesan la red al mismo tiempo.

La institución llamada Agencia de Asignación de Números de Internet (IANA: Internet Assigned Numbers Authority) es la encargada de asignar los número de puertos a las aplicaciones que lo requieran. Existen tres grandes grupos en los que se agrupan los números de puertos, estos grupos son los siguientes:

Puertos bien conocidos (well-know ports): Se denominan así por que son asignados a aplicaciones comúnmente usadas y de dominio general, como por ejemplo HTTP, FTP, etc. A estos puertos se les asignan valores inferiores a 1024.

Puertos de asignación dinámica: Estos puertos tienen valores superiores a 1023 y son asignados dinámicamente en el host de origen el momento que este desea establecer comunicación con alguna aplicación.

Puertos Reservados: Son puertos registrados por propietarios de aplicaciones no muy comunes o que no están al alcance del dominio público. Generalmente tiene valores superiores a 1024.

2.2.5 Capa de Aplicación.

En la capa de aplicación del modelo TCP/IP ejecutan sus funciones protocolos de alto nivel que interactúan con el usuario final. Adicional a esto, en este nivel se realizan las funciones de representación, codificación y control de dialogo. Muchas son las aplicaciones relacionadas con esta capa. A continuación se describen las más comunes agrupadas según sus funciones [4]:

Transferencia de Archivos:

1. **Protocolo de transferencia de archivos (FTP)**: Esta aplicación es un servicio confiable para la transferencia de archivos entre sistemas que utilicen este protocolo. Es utilizado para copiar y mover archivos en estructuras cliente/servidor. Su función la realiza a través de dos puertos TCP, el puerto 20 utilizado para la transmisión de datos y el puerto 21 usado para el control de la transmisión. Los archivos transmitidos pueden estar en modo ASCII o modo binario Se describe en el RFC 959 – FTP ("File Transfer Protocol").

2. **Protocolo trivial de transferencia de archivos (TFTP):** Esta aplicación es un servicio no confiable para la transferencia de archivos. Este protocolo hace uso del puerto UDP 69 para realizar sus transferencias. Es útil en algunas LAN porque opera más rápidamente que FTP y, en un entorno estable, funciona de forma confiable [4]. TFTP está diseñado para ser pequeño y fácil de implementar. Por lo tanto, carece de la mayoría de las características de FTP. TFTP puede leer o escribir archivos hacia o desde un servidor remoto pero no puede enlistar directorios y actualmente no proporciona autenticación de usuarios [4] describe en el RFC 1350 - El protocolo TFTP (Revisión 2).

3. **Sistema de archivos de red (NFS):** Es una aplicación que permite establecer un sistema de ficheros dentro de una red. Con el objeto de compartir archivos remotos, brindando cierto nivel de seguridad y flexibilidad a la información. Los puertos que puede usar no son fijos pero se vale del protocolo UDP para las tareas de la capa de transporte La especificación actual del NFS de Sun se puede encontrar en el RFC 1094 - especificación de NFS.

4. **El Protocolo de Transferencia de Hipertexto (HTTP: Hypertext Transfer Protocol):** Para un mejor entendimiento de este servicio es necesario dar ciertos conceptos de elementos que intervienen en el funcionamiento de este servicio.

   World Wide Web (www): También conocida como la WEB, es una enorme red conectada a todos los sitios del Internet. Su existo se debe a la flexibilidad y facilidad para acceder a la información que se encuentra en todo el mundo.

   Navegador WEB: Es una aplicación cliente/servidor que presenta datos en formatos multimediales en las páginas Web que usan texto, gráficos, sonido y vídeo.

HTML (HTML: Hypertext Markup Language): Es un lenguaje de programación basado en marcas, las cuales le dan las pautas al navegador WEB para que presente la pagina WEB ante el usuario. Luego de estas definiciones podemos decir que el servicio que presenta el protocolo HTTP es el de transferencia de los archivos de hipertexto o páginas web a los Navegadores WEB que solicitan estos archivos. Esta función es transparente al usuario ya que este solo define la dirección que desea visitar y el navegador WEB hace uso del protocolo HTTP para solicitar y publicar la página web requerida. El protocolo HTTP hace uso del puerto 80 del protocolo TCP para realizar el transporte de la información.

Correo Electrónico:

1. **Protocolo Simple de Transferencia de Correo (SMTP):** Este protocolo administra la transmisión de correo electrónico a través de las redes informáticas. Los mensajes que este protocolo transportan debe estar en formato ASCII [4]. Este protocolo hace uso del puerto 25 del protocolo TCP para realizar el transporte se sus paquetes.

2. **Protocolo de Correo de Oficina (POP3):** Este protocolo es utilizado para la descarga de correo electrónico desde un servidor remoto. Básicamente lo que este protocolo hace es permitir descargar el correo electrónico desde el servidor de correo, para que el usuario pueda leer tranquilamente su correo sin tener la necesidad de estar conectado al servidor. El puerto 110 del protocolo TCP es utilizado para cumplir los propósitos de este protocolo.

Conexión Remota:

1. **Emulación de Terminal (Telnet):** Telnet tiene la capacidad de acceder de forma remota a otro computador. Permite que el usuario se conecte a un host de Internet y ejecute comandos. Resumiendo, a través de Internet el cliente de Telnet o host local, envía sus pulsaciones de teclado hacia el servidor Telnet o host remoto, el cual luego de procesar las operaciones según los comando ingresados por el host local, responde hacia el monitor del host local [4]. Para su funcionamiento Telnet hace uso del puerto 23 del protocolo TCP Se describe en el RFC 854 - Especificaciones del protocolo TELNET y RFC 855 - "TELNET Option Specifications".

Administración de Red:

1. **Protocolo simple de administración de red (SNMP):** Es un protocolo que provee una manera de monitorear y controlar mediante la transmisión de información de administración entre dispositivos dentro de una red. Entre los aspectos que pueden ser monitoreados tenemos el rendimiento de la red, detección y soluciones a los problemas de red, planificación de crecimientos en la red, seguridades y desempeño. El puerto 161 del protocolo UDP es utilizado por este protocolo para el transporte de sus paquetes [4].

Gestión de Nombre:

1. **Sistema de denominación de dominio (DNS):** Las direcciones IP (dirección de capa de Internet) tiene un esquema jerárquico. Esto quiere decir que las direcciones pueden ser agrupadas en clases para facilitar su direccionamiento. Todas las direcciones WEB que se visitan a través de INTERNET poseen una dirección IP, de lo contrario no pudieran ser accedidas. Debido a que maneras todos los número de una dirección IP para referirse a una dirección es mas complicado y difícil de recordar, existen nombres que son mas fáciles de usar, a estos nombres se los conoce como dominios. Por este motivo se desarrollo un sistema llamado SISTEMA DE DENOMINACION DE DOMINIOS (DNS), el cual se

encarga de trasladar el dominio de una dirección a su correspondiente dirección IP para que pueda producirse la comunicación [4] Se describe en los RFCs 1034 y 1035.

2.3 MODBUS TCP/IP

El protocolo MODBUS sobre TCP/IP es conocido actualmente como MODBUS TCP/IP. Un sistema de comunicaciones sobre MODBUS TCP/IP puede incluir diferentes tipos de dispositivos:

- Un dispositivo cliente MODBUS TCP/IP y un dispositivo servidor MODBUSTCP/IP conectados a una red TCP/IP [5].

- Dispositivos de interconexión similares a puentes, ruteadores o pasarelas de interconexión entre redes TCP/IP y subred basada en líneas serial que permiten la conexión de Clientes y Servidores MODBUS [5].

En el *Figura 2.11* se muestra una arquitectura típica en la cual se identifican los dispositivos involucrados en la comunicación [5].

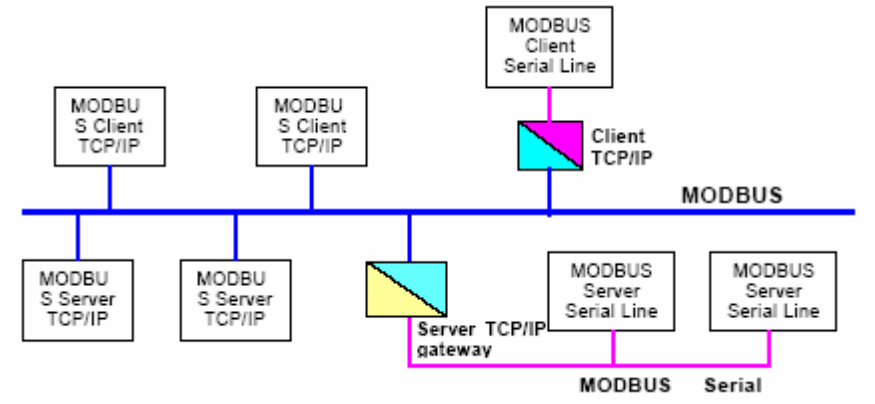


Figura 2.11. Arquitectura de comunicación MODBUS [5].

Como se mencionó anteriormente en este capitulo, el protocolo MODBUS define una unidad de datos para su comunicación conocida como PDU por sus siglas en ingles Protocol Data Unit, Dicho PDU es independiente de los protocolos de capas inferiores al de Aplicación Modbus, lo que lo hace fácil de ser integrado dentro de otro tipo de redes que usen otro tipo de capas inferiores, como es el caso de la suite de protocolos TCP/IP.

Para que el PDU del protocolo MODBUS pueda ser transmitido a través de otros tipos de buses o redes se deben introducir cierta información adicional, que en conjunto con el PDU MODBUS formarán lo que se conoce como ADU, por sus siglas en ingles de Application Data Unit [5].

El *Figura 2.12* muestra un esquema de la distribución del ADU de MODBUS TCP/IP.

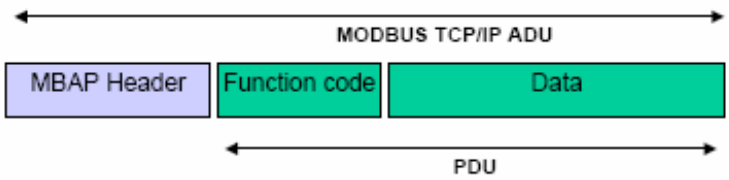


Figura 2.12. ADU MODBUS TCP/IP. [5].

Los campos FUNCTION CODE y DATA ya fueron explicados con anterioridad en este capítulo, a continuación se enfocará la explicación sobre el campo denominado encabezado MBAP, que viene de las siglas en ingles de MODBUS Application Protocol Header.

> *Encabezado MBAP:* Este es un encabezado utilizado en las redes TCP/IP para identificar el ADU MODBUS. En el *Figura 2.13* nos muestra la distribución de bytes en los campos que conforman dicho encabezado.

IDENTIFICADOR DE TRANSACCIÓN (2 BYTES)	IDENTIFICADOR DE PROTOCOLO (2 BYTES)	LONGITUD TOTAL (2 BYTES)	IDENTIFICADOR DE UNIDAD (1 BYTES)

Figura 2.13. Distribución de bytes en encabezado MODBUS TCP/IP

Identificador de Transacción: Esta sección del MBAP sirve para que los dispositivos involucrados en la comunicación mantengan control en la correspondencia entre solicitudes y respuestas [5].

Identificador de Protocolo: Es utilizado para casos en donde exista multiplexación dentro de un sistema. El identificador para el protocolo MODBUS es el valor 0 [5].

Longitud: Este campo es un contador de bytes que lleva la cuenta de los campos siguientes a este. Se incluyen en la cuenta el Identificador de Unidad y los campos de datos [5].

Identificador de Unidad: Este campos es utilizado para propósitos de enrutamiento dentro de sistemas que lo requieran. Es típicamente usado para comunicar a un esclavo MODBUS o un esclavo MODBUS+ a través de un gateway entre una red Ethernet TCP/IP y una red Serial convencional [5].

Gracias a la normalización de este protocolo se ha asignado al puerto registrado TCP 502 la transmisión de los ADU’s del protocolo MODBUS/TCP [5].

CAPITULO 3

DISEÑO DE HARDWARE

3.1 Análisis de posibles soluciones.

Como se mencionó en el capitulo 1, existen varias posibilidades de integración de redes seriales convencionales a las nuevas tecnologías en redes TCP/IP. Una de ellas es mediante el uso de pasarelas de comunicación. Es importante distinguir que en el mercado existe una gran diversidad de pasarelas de todos los tipos. Por un lado tenemos Las pasarelas de capa física, o simplemente llamados conversores. Dichos conversores realizan la función de adaptar un tipo de conexión física a otro tipo de conexión física diferente. Estos conversores son transparentes al tipo de protocolo de comunicación que se este utilizando, sobreentendiendo que debe ser el mismo protocolo manejado por ambos tipos de conexión física involucrados en la comunicación.

Por otro lado tenemos Las pasarelas de protocolos, estos dispositivos desempeñan una función más allá de la de simple conversión de la conexión física. Estas pasarelas de protocolos están encargadas de adaptar la trama de un mensaje de un protocolo para que pueda ser manipulada y utilizada por otro protocolo de distinta configuración.

Realizando una consulta del mercado de Las pasarelas de protocolos se pueden distinguir muchas opciones que satisfarán las necesidades industriales. Es así que se identificó como una de las empresas más influyentes en el mercado a la marca PROSOFT. Pero se debe recalcar que en el mercado existen algunas opciones posibles de otras marcas para la selección de pasarelas de comunicaciones.

3.2 Justificación de la alternativa seleccionada.

El presente proyecto se encuentra orientado a plantear la opción de un producto y un servicio que permita la integración de equipos con protocolos de comunicaciones serial en redes Ethernet. Dentro de la oferta de dicho producto y servicio se busca la integración rápida y confiable de protocolos de comunicaciones seriales, a un costo competitivo respecto a otras posibles soluciones y sobre todo que sea de fácil acceso a las Pequeñas y Medianas Empresas (PYMES) para esto se determino que el uso de un módulo de control embebido que contenga toda la pila del protocolo TCP/IP es la mejor opción para cumplir con dicho propósito por las siguientes características:

a) Un módulo de control embebido posee todas las herramientas necesarias para que el desarrollador pueda crea una aplicación personalizada a la necesidad y requerimientos de las PYMES.

b) El costo de un módulo de control embebido es mucho más reducido que adquirir un software de desarrollo de HMIs que sirva de pasarela para los protocolos seriales-TCP/IP.

c) El consumo de energía de un módulo de control embebido es mucho más económico que cualquiera de las opciones presentadas inicialmente. Adicional a esto, sus características de funcionamiento autónomo (stand-alone) permite que toda la conversión del protocolo sea transparente al usuario final, y de los equipos que estén conectados al nodo de comunicaciones.

d) Las posibilidades de publicaciones WEB permite un ahorro económico en software de desarrollo, permitiendo un monitoreo básico de procesos industriales con variables lentas a través de aplicaciones HTTP.

La familia de módulos de control embebido RCM4000 de la marca Rabbit Semiconductors presenta una amplia gama de posibilidades para el desarrollo de dicho nodo de comunicaciones. Entre ellas tenemos:

a) Variedad de puertos seriales con diferentes opciones de configuración.
b) Puerto Ethernet RJ45 10BASET integrado en el módulo.
c) Toda la pila de protocolos TCP/IP embebido en su núcleo.
d) Su programación se la realiza en lenguaje C, lo cual facilita enormemente su configuración y reduce tiempos de desarrollo y cambios en aplicaciones deseadas.
e) Conjunto de entradas y salidas de propósito general que permiten una amplia gama de posibilidades de desarrollo dependiendo de las aplicaciones requeridas por las industrias.

3.3 Descripción del dispositivo seleccionado.

Como se señaló en el punto anterior, para la implementación del nodo de comunicaciones MODBUS-TCP/IP se ha seleccionado un módulo de control embebido perteneciente a la serie RCM4000 de la marca RABBIT SEMICONDUCTORS. Dentro de esta familia de módulos de control embebido se trabajó sobre el RCM4010 y el KIT de desarrollo de la misma marca. Dicho KIT de desarrollo facilita el uso de los diferentes puertos de comunicación que posee integrados el RCM4010. Adicional a esto, este KIT de desarrollo de facilita la alimentación del módulo ya que provee pines de fácil acceso para la alimentación de voltaje. Las descripciones y funcionalidades del KIT de desarrollo serán analizadas en las etapas siguientes de este documento.

3.3.1 RabbitCore RCM4000

La serie RCM4000 de los módulos RabbitCore es una de las siguientes generaciones de módulos que toma ventaja de las nuevas características del procesador Rabbit 4000. El cual tiene una historia de desarrollo desde el antiguo microprocesador Z80, se podría decir que el microprocesador que ahora se utiliza en este proyecto, el Rabbit 4000, es el predecesor mas reciente del Z80. La serie RCM4000 también presenta un Puerto Ethernet 10Base-T.

En la *Tabla 3.1* están enunciadas las especificaciones del módulo RCM4010 el cual se utilizo para el desarrollo del presente proyecto.

CARACTERISTICAS	RCM4010
Microprocesador	Rabbit 4000 @ 58.98 MHz
Ethernet	Conector 10Base-T
Flash Memory	512K (16-bit)
SRAM	512K (16-bit)
NANA Flash	N/A
E/S Propósito General	25 E/S digitales, configurables hasta 4 alternativas de funciones
Entradas análogas	N/A
Entradas adicionales	2 modos de arranque, entrada de reset
Salidas adicionales	Estado, Salida de Reset
Modulador de ancho de pulsos	2 canales de PWM sincronizado con contador de 10 bits. 2 canales de fase variable o PWM sincronizado con contador de 16 bits
Puertos seriales	5 puertos compartidos de alta velocidad, compatibles con CMOS:
Tasa de transferencia serial	Máxima tasa de transmisión asincrónica = CLK/8
Batería de respaldo	SI (para soportar RTC y la SRAM)
Reloj en tiempo real	SI
Temporizadores	10 temporizadores de 8 bits, 1 temporizador de 10 bits con dos registros de igualdad, 1 temporizador de 16 bits con 4 salidas y 8 registros del set/reset.
Watchdog	SI
Captura de entrada	2 canales de captura de entra pueden ser usados para temporizar señales de entrada desde varios puertos.
Decodificador de cuadratura	2 canales para decodificador en cuadratura aceptan señales de entrada desde módulos codificadores incrementales externos.
Consumo de energía	3.0 – 3.6 VDC, 90mA
Temperatura de operación	0 ºC a +70 ºC
Humedad	5-95%, no condensado
Conectores	1 conector 2x25 para E/S, 1 conector 2x5 para programación

Tabla 3.1. Descripción RabbitCore RCM4000

3.3.2 Microprocesador Rabbit 4000

El microprocesador Rabbit 4000 es un microprocesador de alto rendimiento con baja interferencia electromagnética (EMI), y fue diseñado especialmente para el control embebido, desarrollo de comunicaciones y conectividad con Ethernet. Este microprocesador presenta un set de instrucciones que ofrecen al desarrollador una gran eficiencia y rapidez de ejecución de los códigos en lenguaje C. Las instrucciones incluyen numerosos opcodes de un byte de longitud que ejecutan en dos ciclos de reloj funciones tales como: carga y almacenamiento de 16 bits y 32 bits, operaciones lógicas y aritméticas de 16 bits y de 32 bits, multiplicación de 16 × 16 (ejecutada en 12 ciclos de reloj), saltos largos y retornos para acceder una memoria de 16Mb, y un byte usado como prefijo para direccional el acceso de memoria de entre una instrucciones E/S internas y externas [1].

El microprocesador 4000 no requiere de drivers para memorias externa o interfaces lógicas. Su bus de Direccionamiento de 24 bits, bus de datos de 8 o 16 bits, tres líneas de selección de chips, 2 líneas de habilitación de salida y dos líneas de habilitación de escritura, pueden ser interconectadas directamente con hasta 2 dispositivos de memoria. Hasta 1 MB de memoria puede ser accedida directamente desde el software de programación Dynamic C. Y hasta 16MB pueden ser manipulados mediante el uso de software adicional [1].

Otra característica que presenta este microprocesador es el poseer un puerto que puede ser configurado como maestro o esclavo dentro de una red de microprocesadores en las cuales las tareas se las puede distribuir entre todos los dispositivos interconectados.

Un arranque remoto en frío habilita el arranque y la programación a través de un Puerto serial o del Puerto esclavo. En el *Figura 3.1* tomado del Manual de Usuario del Microprocesador Rabbit 4000 se puede ver un esquema de la distribución de los diferentes dispositivos y su interconexión.

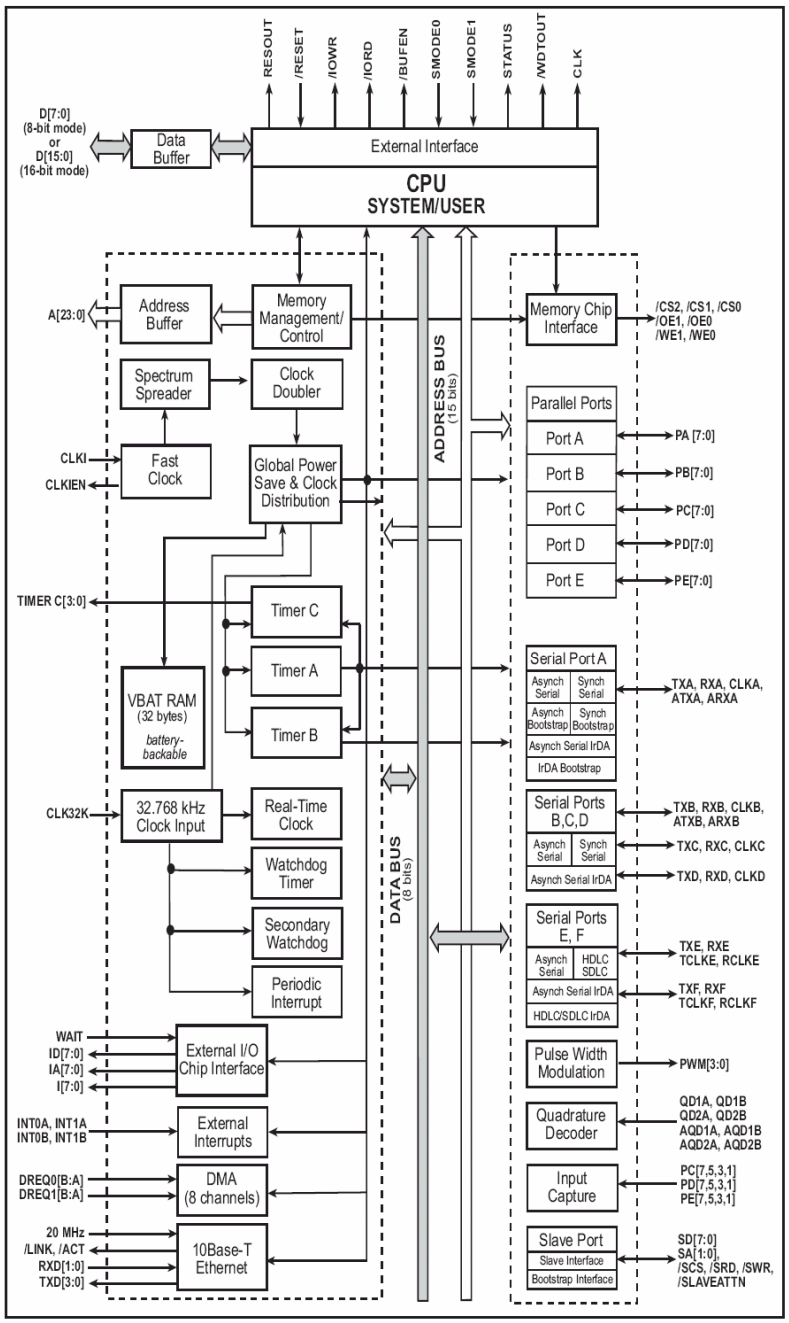


Figura 3.1. Diagrama de Bloques RCM4000

A continuación se resumirá sobre los dispositivos que están directamente involucrados con la realización del presente proyecto. Para mayores detalles y conocimiento de la arquitectura detrás de este microprocesador se puede consultar los documentos enunciados en este capitulo como referencias bibliográficas.

a) Administración de la Memoria.- Como se mencionó anteriormente el microprocesador Rabbit 4000 soporta dispositivos externos de memoria de 8 o 16 bits. Posee 3 líneas seleccionadoras de chips y dos habilitadores de lectura y escritura que permiten que hasta 6 dispositivos de memoria puedan ser conectados a la vez. El espacio físico de memoria del Rabbit 4000 contiene 4 bancos consecutivos, cada uno de los cuales puede ser direccionado hacia un par individual de selección y habilitación de chip. Los bancos pueden ser configurados para poseer iguales tamaños con valores desde 128KB hasta 4MB, con lo cual proporciona un rango de memoria física total de 512KB a 16MB. En el *Figura 3.2* siguiente se muestra un esquema de la distribución de memoria.

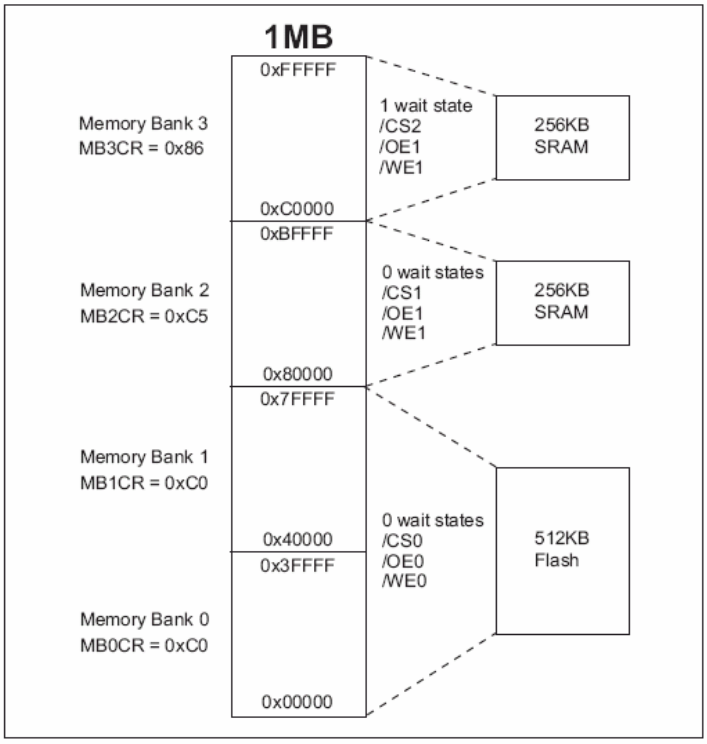


Figura 3.2. Distribución de Memoria en modulo RCM4000

b) Puertos Seriales.- El módulo de control embebido posee 5 puertos seriales denominados puertos A, B, C, D y F. Estos puertos pueden operar en modo asíncrono hasta velocidades iguales al valor del reloj dividido para 8. Un puerto asíncrono puede manejar siete u ocho bits de datos [1]. El Puerto serial A además de ser usado para programación también puede ser usado como un puerto asíncrono una vez la aplicación haya sido realizada y el módulo este en modo RUN. Los puertos C y D pueden ser usados en modo serial dirigidos por un reloj. En este modo una línea de reloj sincroniza los relojes de los

datos de entrada y de salida. El puerto serial F puede ser configurado como SDLC/HDLC. El protocolo IrDA también puede ser implementado en este puerto serial. La *Tabla 3.2* tomada del Manual de Usuario del RCM4000 describe los pines asociados a los puertos seriales mencionados.

Puerto Serial	**Pines Seriales**	**Pines Puertos Paralelos**
Puerto A	TXA	PC6, PC7, PD6
	RXA	PC7, PD7, PE7
	SCLKA	PB1
Puerto B	TXB	PC4, PC5, PD4
	RXB	PC5, PD5, PE5
	SCLKB	PB0
Puerto C	TXC	PC2, PC3
	RXC	PC3, PD3, PE3
	SCLKC	PD2, PE2, PE7, PC7
Puerto D	TXD	PC0, PC1
	RXD	PC1, PD1, PE1
	SCLKD	PD0, PE0, PE3, PC3
Puerto F	TXF	PD6, PE6, PC6
	RXF	PD3, PE3, PC3
	RCLKF	PD1, PE1, PC1
	TCLKF	PD0, PE0, PC0

Tabla 3.2. Asignación de puertos en modulo RCM4000

Un detalle de las funciones y posibles configuraciones de cada uno de los pines se lo puede encontrar en el manual de usuario. Debido a que el proyecto se limita al uso de los puertos seriales, su respectiva asignación se la puede tomar de la *Tabla 3.2*.

c) Puerto de Programación.- El RCM4000 es programado a través de un conector de 10 pines denominado como J1. El puerto de programación hace uso del puerto Serial A del Rabbit para realizar la comunicación. El software de programación, Dynamic C usa este puerto para descargar los programas y para hacer las depuraciones de los mismos. Mayores detalles del software de programación serán indicados en el capítulo siguiente.

El puerto serial A es usado también para las siguientes operaciones:

• Arranque en Frio del Rabbit 4000 en el RabbitCore luego de un reset.

• Descargas remotas de programas y depuraciones de los mismos sobre una conexión Ethernet usando el dispositivo RabbitLink EG2110.

Este puerto tiene los siguientes usos alternativos:

• Como un puerto serial síncrono.

• Como un puerto serial asíncrono, con una línea de reloj disponible para ser usada como una I/O de propósito general.

Adicionalmente, ciertos pines de status están disponibles en el Puerto serial A, estos pines son:

- *startup-mode (SMODE0, SMODE1).*- describen lo que hace el microprocesador luego de un reset.
- *status.*- Determinan la presencia de un Rabbit conectado.
- *reset.*- Es una entrada externa para resetear el microprocesador.

d) Puerto Ethernet.- El RCM4010 posee un conector RJ-45 para la interfaz física del módulo con redes Ethernet. Junto a dicho puerto existen dos leds los cuales indican el status de conexión del puerto y la transferencia de información a través de él.

3.4 Diseño de Hardware.

El modulo de control embebido posee un conector IDC 2x5 utilizado para la programación del mismo y un conector IDC 2x25 utilizado para acceder a los pines de entradas y salidas del microcontrolador Rabbit 4000, así como para proporcionar de alimentación de voltaje al mismo. En los siguientes gráficos se muestran las medidas del RCM4010.

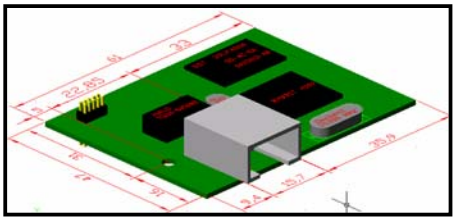


Figura 3.3.a. Vista perspectiva izquierda RCM4010

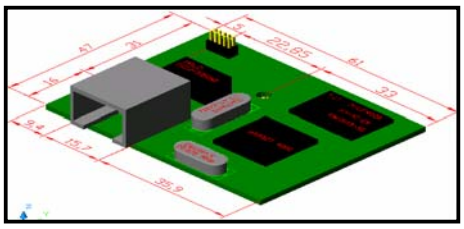


Figura 3.3.b Vista perspectiva derecha RCM4010

Las medidas mostradas en los anteriores gráficos se encuentran dadas en *mm*. Es importante tener en cuenta dichas mediciones para las consideraciones físicas de diseño, tales como ubicación del equipo dentro de un entorno industrial, así como diseño de encapsulados de asilamiento y protección para dicho elemento electrónico.

3.4.1 Interfaz con los Pines del Microprocesador.

El RCM4010 posee un conector tipo SMT 2x25 pines, con un espaciamiento entre pines de 1.27mm. Estos 50 pines son usados para la interfaz con las entradas/salidas del microprocesador, así como para la alimentación del módulo. En el *Figura 3.4* se ubica el conector SMT 2x25. En el *Figura 3.5* se muestra la distribución de los pines del conector SMT 2x25 al cual se le hace referencia mediante el tag J3.

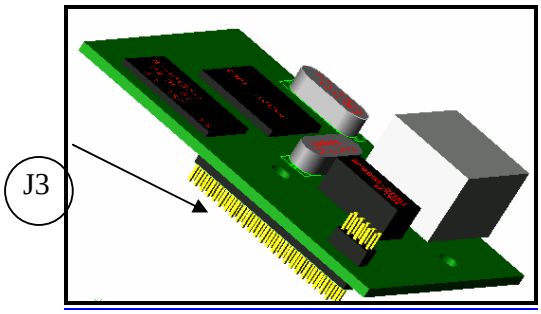


Figura 3.4. Vista 3D RCM 4010-localización conector J3

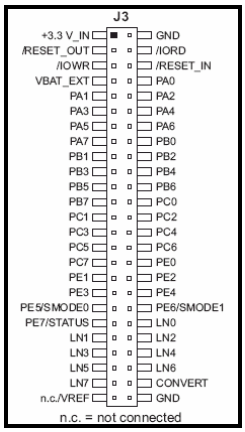


Figura 3.5. Pines del conector J3

A través de este conector se tiene acceso a los pines de los puertos que tiene el microprocesador. Durante el proceso de desarrollo del proyecto toda la interfaz con este puerto se la realizo a través de la placa de prototipo (101-1097) que esta incluida dentro del kit de desarrollo. Dicha placa de prototipo posee un socket en donde se conecta el header J3 del RabbitCore. Este socket tiene la denominación de J2.

En el caso de que no se utilice la placa prototipo es necesario hacer un cable de conexión con terminales SMT 2x25 para la interconexión con otros dispositivos o implementar un conector SMT 2x25 hembra en una placa de circuito integrado, similar a la placa para prototipos que viene dentro del KIT de desarrollo.

3.4.2 Interfaz de Programación.

Para la programación del RabbitCore se hizo uso del cable Rabbit Program Cable (101-0542) que viene incluido dentro del KIT de desarrollo. Dicho cable posee un conector RS-232 a un extremo y al otro extremo un conector 2x5 pines y con un espaciamiento de pin de 1.27mm denominado J1, para la interfaz con el puerto de programación del RabbitCore mostrado en el *Figura 3.6.* El conector RS-232 va en la PC de programación, mientras el conector 2x5 va conectado al conector J1.

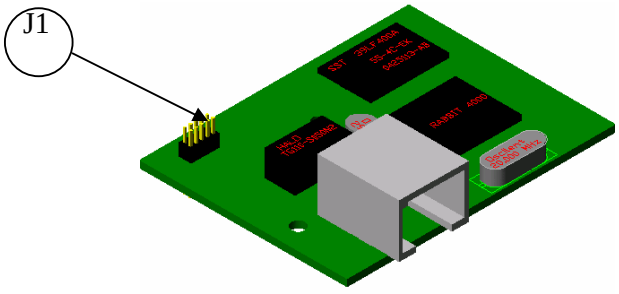


Figura 3.6. Vista 3D – localización conector J1

3.4.3. Alimentación del RCM4010

La alimentación del modulo se la realiza a través de la placa de prototipos por un conector de tres pines destinado para esta función. El nivel de voltaje que recibe este conector es de 12VDC. Internamente esta placa posee una fuente reguladora de tipo *switching* de 5VDC. Adicional a esto, existe una salida de voltaje de 3.3VDC para la alimentación del módulo. La alimentación del módulo se la hace a través del conector J2 en donde se encuentran los terminales positivo y negativo del módulo. El *Figura 3.7* muestra la ubicación de los terminales de alimentación para el módulo en el conector J2.

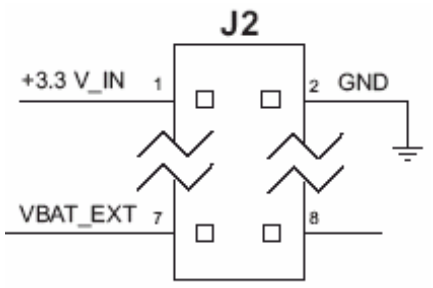


Figura 3.7. Pines de alimentación en conector J2

En el caso que no se posea la placa prototipo, a continuación en el *Figura 3.8* se muestra el diagrama de la fuente de energía que se debe implementar para el funcionamiento del módulo.

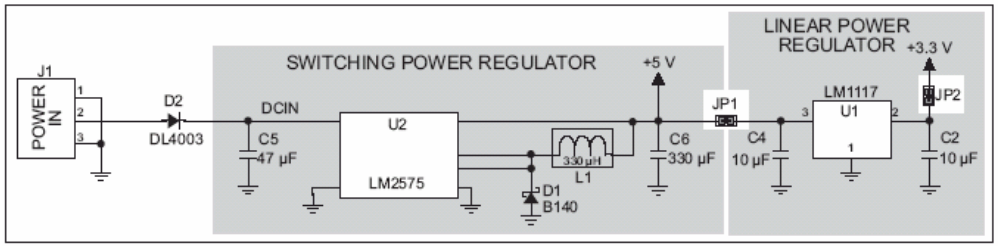


Figura 3.8. Fuente de alimentación para modulo RCM4010

Descripción del circuito.- Al inicio del circuito se encuentra un diodo el cual protege a todo el circuito de una la polarización de alimentación incorrecta. El circuito integrado LM2575 es un regulador de voltaje reductor que entrega hasta 1A. Este circuito integrado proporciona todas las funciones activas para un regulador *switching* reductor y es

capaz de manejar una carga de un amperio con excelente regulación línea. En este caso la salida que obtenemos de este circuito integrado es de 5V. Siguiendo con el diagrama encontramos el circuito integrado LM1117 el cuales es un regulador lineal de baja salida de voltaje, el cual nos entrega un voltaje de 3.3V para la alimentación del módulo. Dicho valor de salida es regulable, dependiendo de los elementos conectados a este. Es decir, varia de acuerdo a los valores de capacitancia de los capacitares C4 y C2. Para mayor información del mismo se puede consultar la hoja técnica de este elemento.

Adicional a la alimentación antes mencionada, el RabbitCore presenta dos terminales destinados a la alimentación del módulo por una batería de backup. Dicha alimentación garantiza el funcionamiento del reloj en tiempo real del módulo para que las aplicaciones que hacen uso de esta función no sufran cambios el momento de que la alimentación principal se pierda. Así como para garantizar la integridad del programa y de la información almacenada en la SRAM del módulo.

3.4.4 Puerto RS-232

Para la interfaz del controlador embebido con el dispositivo MODBUS se utilizará el puerto serial C del RabbitCore. La interfaz física con este puerto serial se lo hará a través del puerto RS-232 que se encuentra en la tarjeta de prototipos. El puerto RS-232 se encuentra en el conector J4 en la placa prototipo. Este conector puede ser configurado para ser usado como una interfaz RS-232 de 3 cables (sin control de flujo) o como una interfaz RS-232 de 5 cables (con control de flujo). Para que el RabbitCore maneje el estándar RS-232 es necesario hardware adicional para la conversión de los niveles de voltaje TTL a niveles de voltaje y polarización establecidos por el estándar. Este hardware adicional mencionado ya se encuentra instalado en la placa prototipo. Con lo que solo es necesario un cable de comunicación RS-232 para la transmisión de la información al dispositivo MODBUS.

En el caso de que se realice la implementación del gateway de comunicaciones sin la tarjeta prototipo es necesario la implementación del hardware adicional para la conversión de los niveles de voltaje. Para esto se hace uso del circuito integrado MAX232 el cual se encarga de hacer la conversión del voltaje tanto en nivel como en polaridad para tener

compatibilidad con el estándar RS-232. Este circuito integrado MAX232 necesita de una fuente de alimentación de 5V y elementos electrónicos adicionales para su funcionamiento. El detalle del circuito a hacer implementado se lo encuentra en la hoja técnica de dicho elemento.

El *Figura 3.9* muestra la ubicación de los principales conectores de la placa prototipo del RabbitCore.

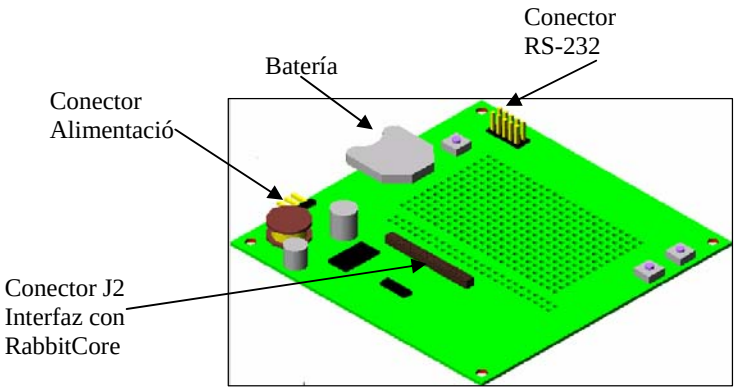


Figura 3.9. Distribución General Placa de Desarrollo

El *Figura 3.10* y el *Figura 3.11* son modelos en 3D de un prototipo de lo que sería el gateway de comunicaciones.

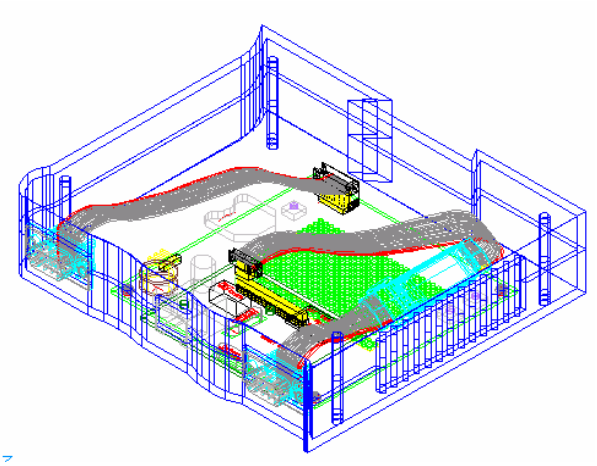


Figura 3.10. Modelo 3D gateway de comunicaciones

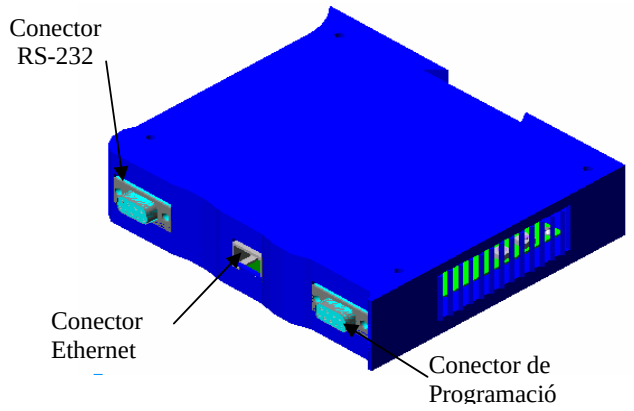


Figura 3.11. Sólido del modelo 3D del gateway de

CAPITULO 4

DESARROLLO DE SOFTWARE

4.1 Introducción

La función del controlador embebido como gateway de comunicaciones es la de encapsular la trama de un protocolo para que pueda ser entendida por otro protocolo. En este caso en particular, se habla de que la solicitud realizada por un dispositivo maestro Modbus TCP llegará al controlador embebido por el puerto Ethernet. Aquí el controlador recibirá la trama y realizará las rutinas necesarias para que la trama sea compatible con el protocolo RTU y pueda ser enviada por el puerto serial y entendida por el dispositivo esclavo Modbus RTU.

Una vez que el dispositivo Modbus haya recibido la petición, armará el mensaje de respuesta y lo transmitirá hasta el controlador embebido a través del puerto serial. Una vez aquí el controlador volverá a encapsular el mensaje recibido para que sea compatible con el protocolo MODBUS TCP y de esta manera el dispositivo maestro reciba la información solicitada. De esta manera se termina un ciclo de comunicación Maestro/Esclavo, en donde el maestro se comunica a través de Modbus TCP y el esclavo a través del protocolo Modbus RTU.

Dentro de este capítulo se detallará el proceso que realiza el controlador para encapsular y desencapsular las tramas de los mensajes, así como el programa en lenguaje C desarrollado para que el controlador realice dicho procedimiento.

4.2 Conceptualización del Programa.

Antes de la realización de dicho programa es importante realizar una conceptualización de lo que se necesita realizar. Básicamente lo que el controlador va a realizar es la manipulación de la trama de los protocolos involucrados en la comunicación. Para definir los procesos que se deben realizar con las tramas de comunicación es importante encontrar sus diferencias y semejanzas. En la *figura 4.1* se muestra el PDU (PROTOCOL DATA UNIT) de MODBUS así como las tramas de MODBUS RTU. En la *figura 4.2* se muestran el PDU y la trama MODBUS TCP.

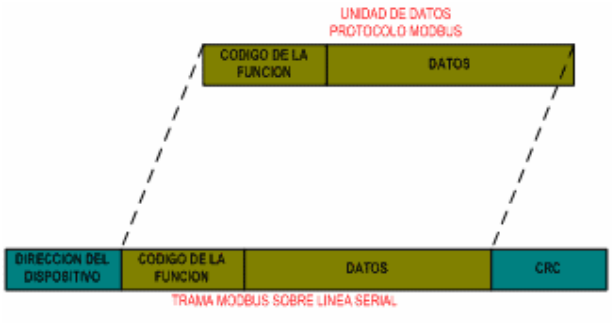


Figura 4.1. Encapsulamiento trama MODBUS RTU

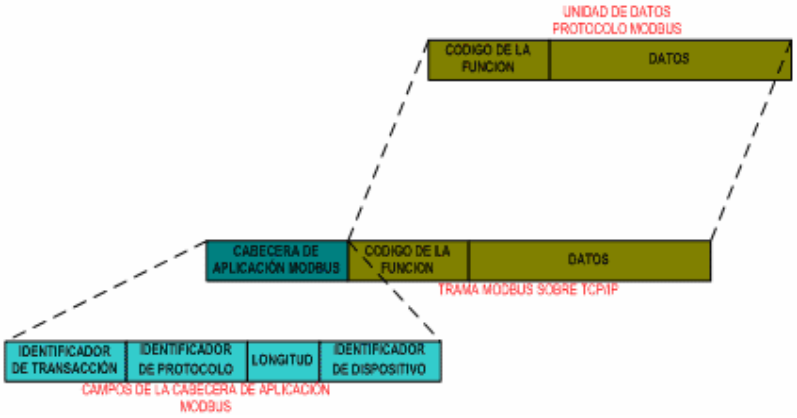


Figura 4.2. Encapsulamiento trama MODBUS TCP/IP

De los dos gráficos anteriores se pueden sacar las siguientes observaciones:

1. Una característica clave que facilita la integración de protocolos seriales a Ethernet es que su unidad de datos base se mantenga igual sin importar el medio físico en el que se propague. De los gráficos

antes mostrados podemos ver que la unidad de datos del protocolo Modbus es la misma tanto para serial RTU como para TCP/IP.

2. En la trama serial existe un campo de control de errores que se transmite al final de la trama. A diferencia de la trama TCP/IP en la cual la verificación de errores es realizadas por protocolos de capas inferiores (capa dos del modelo de referencia TCP/IP).

3. La trama Modbus sobre TCP/IP tiene una cabecera compuesta por cuatro campos. De estos cuatro campos, uno de ellos es común para las dos tramas. Este campo común es el número identificador del dispositivo.

Estas observaciones sirven para identificar que porción de la trama es reutilizable, que parte de la trama debe ser desechable y que es necesario adicionar a la trama para que pueda ser entendida por protocolo bajo el cual se la transmita.

4.3 Software de Programación: Dynamic C

Para que el controlador embebido pueda realizar las rutinas necesarias para que encapsule y extraiga la trama del protocolo, es necesario que se desarrolle un programa que realice todos estos procedimientos. Para la programación del controlador embebido es necesaria la utilización del software Dynamic C, el cual es proporcionado al adquirir el controlador.

El momento de adquirir el controlador, el software que viene en conjunto con el paquete incluye ciertas librerías y drivers para manejar todos los recursos del hardware. De igual manera, son incluidos ejemplos de utilizaciones de las funciones más comunes y útiles para que el desarrollador optimice su tiempo y saque el mayor provecho al controlador embebido.

4.3.1 Referencias del Software Dynamic C

El software Dynamic C es un sistema integrado para el desarrollo de software embebido. Puede ser ejecutado en una computadora personal IBM-compatible y esta diseñado para el uso con controladores de la marca Z-World y otros controladores basados en microprocesadores de la marca Rabbit. Dynamic C integra las siguientes funciones de desarrollo:

1) Edición de la programación.
2) Compilación de los programas desarrollados.
3) Enlace con el hardware.
4) Descarga directa de los programas a las memorias del controlador.
5) Depuración de los programas ejecutados.

Otra característica importante que posee este software es que soporta programación en lenguaje ensamblador simultánea con el lenguaje C. Esto quiere decir, que no es necesario dejar el código desarrollado en lenguaje C, para desarrollar código en lenguaje ensamblador, los dos códigos pueden ir mezclados. Lo único que se necesita hacer es color delimitaciones entre el código en ensamblador y el código en lenguaje C. Para mayores detalles del lenguaje de programación y de la interfaz de programación se puede consultar con el Manual de Usuario este producto.

4.4 Desarrollo del Programa.

Previo a la realización del programa dentro del controlador embebido es necesario entender como el mensaje viaja a través de la red hasta llegar al módulo. El *Figura 4.3* describe brevemente el proceso en que el PDU MODBUS forma parte de una trama TCP/IP.

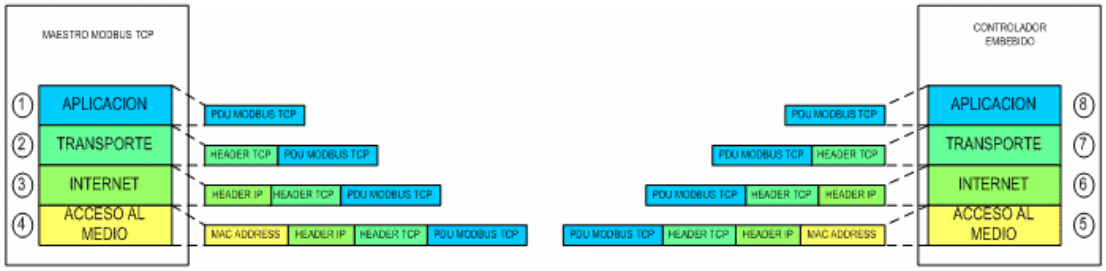


Figura 4.3. Encapsulamiento del PDU MODBUS en trama TCP/IP

En el paso 1 el dispositivo maestro MODBUS construye una requisición para una unidad esclava MODBUS. Al ser un dispositivo con comunicación MODBUS TCP armará la trama utilizando una cabecera ya antes indicada en este capitulo mas el código de la función y mas la información adicional necesario según las especificaciones MODBUS. En el paso 2 se le suma a los datos que viene de la capa superior (aplicación) información de la capa de transporte necesaria para la comunicación. Los más importante y relevante de esta acción es que a este nivel se le agregan los puertos de origen y de destino. De esta manera, el puerto de inicio es cualquier valor numérico mayor a 1024 asignado al azar por la pila TCP/IP del dispositivo maestro. En tanto que el puerto final debe tomar el valor 502 ya que este es el valor reservado para MODBUS TCP.

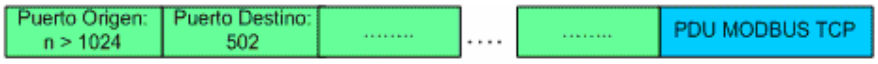


Gráfico 4.4. Encapsulamiento del PDU MODBUS TCP en la capa de Transporte

En el paso 3 a los datos recibidos desde las capas superiores se les suma información de la capa de Internet necesaria para la transmisión de la información. Lo más importante a este nivel es que se le suman a la trama las direcciones IP de origen y de destino. Los valores de estas direcciones dependerán de la configuración que tenga la red en la que se está trabajando. La conectividad dependerá de que direcciones tome y que equipos se encuentran en la red. Finalmente en el paso 4 además de sumarle toda la información necesaria a la trama para que pueda viajar por el medio le suman las direcciones MAC o direcciones físicas de origen y de destino.

Estos primeros 4 pasos resumen como una requisición MODBUS es empaquetada para que viaje a través de la red Ethernet. Luego de cumplir con todo el empaquetamiento la información esta lista para atravesar la red Ethernet hasta su destino. En este caso, el paquete está diseccionado hacia el controlador embebido (dirección IP destino) en donde se trabajará la trama para su posterior envío a través de la red MODBUS serial.

Los pasos 5, 6 y 7 son implementados gracias a que el controlador embebido tiene integradas todas las librerías necesarias para la manipulación de la trama. Es así que una

ves que la trama llega al puerto Ethernet del controlador embebido lo primero en realizarse es verificar si la dirección MAC coincide con la del modulo embebido. Luego de esto el paquete pasa a la siguiente capa en donde se verifica que la dirección IP destino sea la que tiene asignada el módulo. Hasta este punto la comunicación es transparente para el desarrollador del programa dentro del módulo. Una vez aquí, mediante comandos el desarrollador decide bajo que puerto se va a realizar la comunicación de llegada, luego de esto el desarrollador tiene libre acceso a la parte de la trama conocida como datos, que en este caso es la requisición MODBUS TCP ensamblada en el dispositivo maestro.

Para la realización del programa es necesario plantearse un diagrama de flujos que describa de manera general lo que el controlador embebido debe realizar para manipular la trama del protocolo Modbus. A continuación el *Figura 4.5* diagrama de flujos describe el procesado a ser programado en el controlador.

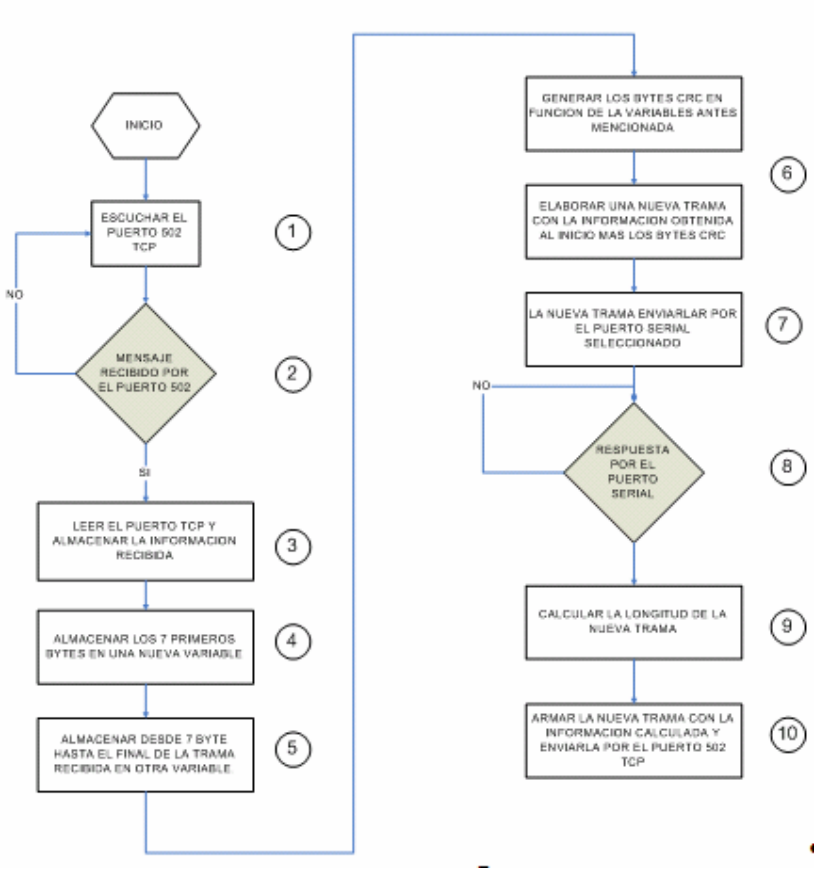


Figura 4.5. Diagrama de Flujo de la lógica del programa

Para que el controlador pueda realizar todos estos procesos haremos usos de las funciones de muchas de las librerías contenidas dentro del software Dynamic C. A continuación se detallará como se lleva acabo cada uno de los pasos enunciados en el anterior diagrama de flujos.

1. Escuchar el puerto 502 TCP.

Como se mencionó en el capítulo 2, el puerto destinado a hacer usado para la comunicación MODBUS TCP es el puerto TCP 502. Esto quiere decir que cualquier equipo conectado a la interfaz Ethernet del controlador que maneje el protocolo MODBUS TCP va a enviar sus tramas desde cualquier puerto mayor a 1024 asignado al azar hacia el puerto 502. Caso contrario, sino no se escucha bajo este puerto TCP, la información nunca llegaría a ser procesada y transmitida hacia la red MODBUS RTU.

Para realizar la escucha de un puerto TCP se hace uso de la función *tcp_listen*. A continuación se detalla información sobre esta función.

	tcp_listen(parámetro 1,parámetro 2, parámetro 3, parámetro 4,parámetro 5, parámetro 6);
parámetro 1	Este parámetro es una variable de tipo *tcp_socket*. Este tipo de variable es propia de la librería que maneja el protocolo TCP y representa el socket por el cual se va a realizar la comunicación. Debe ser previamente definida para poder ser usada en esta función.
parámetro 2	Este parámetro es una variable de tipo *int*. Esta variable es el número del puerto por el cual se va a recibir el mensaje. En este caso tiene el valor de 502.
parámetro 3	Este parámetro es una variable del tipo *long int*. Esta variable contiene la dirección IP del dispositivo del cual se espera llegue el mensaje. Si se coloca el valor de 0 en esta variable, se recibirán los mensajes de cualquier dirección dentro de la red. Esto es lo optimo porque en el caso práctico se pueden tener más de un dispositivo maestro conectado a la red Ethernet y en cualquier momento estos equipos maestros van a solicitar información a los esclavos en la red serial.
parámetro 4	Este parámetro es una variable del tipo *int*. Representa el número del puerto del cual se acepta la conexión para recibir la información
parámetro 5	Este parámetro permite colocar una función la cual es llamada luego de que la información ha sido recibida. Si se coloca NULL en este parámetro, la información recibida se locota en el buffer del socket.
parámetro 6	Este parámetro esta presente por compatibilidad y posible futuro uso. Por el momento este parámetro toma el valor de 0.

Antes de que se empiece a escuchar cualquier puerto de la comunicación TCP o UDP es necesario hacer uso de la función *tcp_tick.* Esta función realiza continuamente pequeñas tareas de la pila de TCP/IP, tareas tales como chequear la presencia de nuevos paquetes,

procesar los paquetes que llegan y retransmitir los paquetes perdidos. La sintaxis de esta función es la siguiente:

	tcp_tick(parámetro 1);
parámetro 1	Este parámetro es una variable del tipo *tcp_socket*

Esta función regresa un valor distinto a cero cuando ve que el socket está haciendo alguna actividad. Y regresa cero cuando el socket esta completamente cerrado y listo para ser usado nuevamente.

2. Mensaje recibido por el puerto 502

Una vez que se realizo la escucha de un puerto TCP y se determina que se quiere establecer comunicación desde otro dispositivo, previo a leer el mensaje, es necesario usar un par de funciones que ayudan a verificar la estabilidad de la conexión y verificar que sigue activa. Estas funciones son *sock_established* and *sock_bytesready*. A continuación se explica la sintaxis de estas dos funciones:

	sock_established(parámetro 1);
parámetro 1	Este parámetro es una variable del tipo *tcp_socket*.

Esta función retorna el valor si el socket esta activo, caso contrario retorna cero.

	sock_bytesready(parámetro 1);
parámetro 1	Este parámetro es una variable del tipo *tcp_socket*.

.

Esta función retorna el número de bytes que están esperando para ser leídos, si no hay bytes esperando retorna el valor de 0.

3. Leer el puerto y almacenar la información recibida

Una vez que se haya establecido la conexión en el socket, es necesario hacer la lectura de la información y el almacenamiento respectivo de la trama para que pueda ser manipulada dentro del controlador embebido y se pueda formar la nueva trama compatible con

MODBUS RTU. Para esto es usada la función *sock_read*. La sintaxis de esta función es la siguiente:

	sock_read(parámetro1, parámetro2, parámetro3);
parámetro 1	Este parámetro es una variable del tipo *tcp_socket*, y representa el socket bajo el cual se esta realizando la comunicación.
parámetro 2	Este parámetro es un arreglo del tipo *char*, es una variable que almacenará la información que se recibe.
parámetro 3	Este parámetro es una variable del tipo *int*. Este valor numérico indica el numero de bytes que van a ser almacenados dentro del buffer.

Al hacer uso de esta función los datos que llegan por el puerto Ethernet son almacenados dentro de un arreglo de variables del tipo *char*. Una variables de tipo *char* representa un byte de información, ya que el PDU MODBUS esta basado en grupos de bytes, al usar este tipo de variable es más fácil la manipulación de la información.

Luego que se ha hecho la lectura es necesario que se realice una copia de esta información para tener herramientas suficientes para poder manipularla y lograr el propósito. Para el almacenamiento de estas variables se hace uso de la función *memcpy*. La sintaxis de esta función es la siguiente:

	memcpy(parámetro1, parametro2, parametro3)
parámetro 1	Este parámetro es un puntero al arreglo de variables tipo *char* que representa el buffer del cual se va a copiar la información.
parámetro 2	Este parámetro es un puntero al arreglo de variables tipo *char* que representa el buffer al cual se va a copiar la información.
parámetro 3	Este parámetro es una variable del tipo *int* la cual indica el número de bytes que va a ser copiados desde el buffer origen hasta el buffer destino.

4. Almacenar los primero 7 bytes de los datos recibidos.

Como muestra el *Figura 4.6*, los primeros 7 bytes recibidos forman parte del encabezado que se le suma al PDU MODBUS para que pueda ser entendido por los dispositivos MODBUS TCP en una red Ethernet. Al almacenar estos bytes en otra variable estamos eliminando de la trama principal esta información que no es necesaria para que pueda ser

transmitida por la red serial. Para esto se hace uso de la función *memcpy()* mas los valores necesarios para grabar solo los 7 primeros bytes en otra variables.

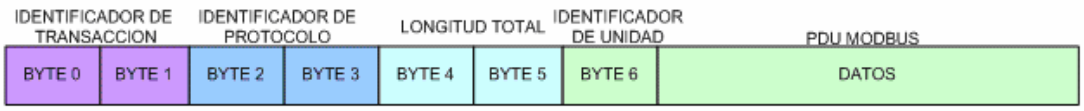


Figura 4.6. Distribución de bytes en paquete MODBUS TCP

5. Almacenar los bytes restantes luego de extraer los 7 primeros bytes

Una vez extraídos los 7 primeros bytes, el resto de la información es pasada a otra variable para proceder con el siguiente paso el cual es el cálculo de los bytes que forman el campo de chequeo de errores por redundancia cíclica. Para determinar cual es la longitud de la información que llega es necesario crear una función que permita obtener esta información de la trama recibida ya que en la cabecera que tiene la trama MODBUS TCP existe un campo llamado longitud. Este campo longitud ocupa dos bytes de información esto quiere decir que la información se encuentra en numeración binaria. Para que el valor binario que se encuentran en estos dos bytes de información pueda ser manipulado como un valor entero (*int*) se implemento una función que hace la simple conversión de estos valores binarios a un número entero. Dicha función se llama *DataLen.* La sintaxis de esta función es la siguiente:

	DataLen(parámetro1)
parámetro 1	Este parámetro es un arreglo de datos del tipo *char*

Al usar esta función se envía el arreglo de variables de tipo *char* que almacena la información que se recibió. El cuerpo de la función es el siguiente:

```
int DataLen (char *bufferlen)
{
        int value;
        value = (int)bufferlen[4]*256+(int)bufferlen[5];
        return (value);
}
```

Esta función retorna el valor entero de la longitud del PDU MODBUS recibido por el puerto Ethernet.

6. Calculo de los bytes de Chequeo por Redundancia Cíclica.

La redundancia cíclica es un algoritmo usado para verificar la integridad de la información a nivel de bits. Es decir, es un procedimiento mediante el cual la unidad que recibe la información verifica que este correcto el mensaje que recibe a nivel de bits. En el caso en el que el valor del campo de Chequeo por Redundancia Cíclica no coincida con el valor obtenido por el algoritmo de cálculo en el dispositivo receptor, la trama será descartada como incorrecta y la información no será procesada. La explicación del algoritmo verificador fue explicada en el capitulo 2. Dicha explicación fue tomada de los documentos de implementación MODBUS emitidos por MODBUS.ORG. A continuación se muestra el código que implementa este procedimiento de verificación. Dicho código también fue tomado de los documentos de implementación MODBUS. Ya que MODBUS es un protocolo abierto esta información es de fácil acceso para las personas interesadas en el desarrollo de soluciones compatibles con el estándar.

```
unsigned char uchCRCHi, uchCRCLo ;
unsigned uIndex ;
 uchCRCHi = 0xFF;
 uchCRCLo = 0xFF;
 usDataLen =    usDataLen - 2;
 while (usDataLen--)
{
        uIndex = uchCRCLo ^ *puchMsg++ ;
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
        uchCRCHi = auchCRCLo[uIndex] ;
}
 puchMsg[usDataLen +1] = uchCRCLo;
puchMsg[usDataLen +2] = uchCRCHi;
```

La variables *auchCRCHi* y *auchCRCLo* son arreglos de variables *char* que contienen valores predeterminados que sirve para la ejecución de la rutina de Chequeo de Redundancia cíclica. La tabla de datos de los arreglos *auchCRCHi* y *auchCRCLo* se adjunta como anexo al final de este documento.

Para la implementación de esta subrutina dentro del programa se creo una función la cual ejecuta los comandos antes mostrados. La función se llama *CRC16*. La sintaxis de esta función es la siguiente:

	CRC16(parámetro 1, parámetro 2)
parámetro 1	Este parámetro es un arreglo de variables del tipo *char*
parámetro 2	Este parámetro es una variable del tipo *int*

Dentro del programa se uso esta función utilizando como parámetros un arreglo de tipo *char* que contiene todo el PDU MODBUS y el valor calculado por la función *DataLen* mas 2 unidades. Esta acción de sumar dos al valor de la longitud de la trama es debido a que luego de usar esta función se arma un nuevo arreglo de tipo *char* que contiene todo el PDU MODBUS más dos bytes que son los valores calculados con el algoritmo de redundancia cíclica.

7. Envío de la nueva trama por el puerto serial hacia la red MODBUS RTU.

Antes de enviar la nueva trama por el puerto serial, se almacena en otra variable para realizar el envío de la misma. El envió de la trama por el puerto serial se lo realiza con la función *serZwrite()*. Esta función es el resultado de la generalización de las funciones de un puerto determinado, en este caso del puerto D a través de la función *serDwrite()*. Es decir, se asocia las funciones de operación de uno de los puertos seriales con una función constante, de esta manera el momento que se desee cambiar de puerto serial no hace falta buscar en todo el programa en donde se hizo uso de la función, sino únicamente se reemplaza la asignación de la función a las constantes y de esta manera todas la funciones quedan referidas al nuevo puerto que se desea usar. A continuación se muestra la parte del código en la que se hace este procedimiento:

```
#define serZopen     serDopen
#define serZread     serDread
#define serZgetc     serDgetc
#define serZrdUsed   serDrdUsed
#define serZwrite    serDwrite
#define serZclose    serDclose
#define serZwrFlush  serDwrFlush
#define serZrdFlush  serDrdFlush
#define serZwrFree   serDwrFree
#define serZrdFree   serDrdFree
#define serZparity   serDparity
```

De esta manera habilitamos el uso de las funciones del puerto D a través de las funciones del puerto Z, en realidad no existe ningún puerto Z pero se define esto como constantes de

esta manera si se cambia de puerto solo se deberá hacer el cambio en esta parte del programa. La sintaxis de la función *serZwrite()* (*serDwrite()*) es la siguiente:

	serZwrite(parámetro1, parámetro2);
parámetro 1	Es un arreglo de variables de tipo *char*. Representan los datos a ser transmitidos.
parámetro 2	Es una variables del tipo *int*. Representa el número de bytes del parámetro 1 a ser transmitidos.

Previo al uso de esta función de transmisión serial es necesario configurar ciertos parámetros del puerto serial que son indispensables para la transmisión y recepción de datos. Parámetros tales como velocidad de transmisión, paridad, etc.

Primero antes de usar la función de envío a través del puerto serial es necesario abrir el puerto serial. Para esto se hace uso de la función *serZopen (serDopen)*. La sintaxis de esta función es la siguiente:

	serZopen(parámetro 1)
parámetro 1	Este parámetro es una variable de tipo *long int*. Este valor representa la velocidad de transmisión a la que va a trabajar el puerto serial. En el caso que sea la velocidad correcta de la red esta función retorna el valor de uno, caso contrario el valor de cero.

Para especificar la paridad con la que va a trabajar el puerto serial se hace uso de la siguiente función *serZparity (serDparity)*, la sintaxis de esta función es la siguiente:

	serZparity(parámetro 1)
parámetro 1	Este parámetro es una variable de tipo *int* el cual especifica bajo que tipo de paridad el puerto va a funcionar. Los valores que puede tomar esta variable son los siguientes:
	PARAM_NOPARITY 0x00 PARAM_EPARITY 0x01 PARAM_OPARITY 0x02 PARAM_2STOP 0x03 PARAM_MPARITY 0x04 (Mark - Rabbit 4000 only) PARAM_SPARITY 0x05 (Space - Rabbit 4000 only)

8. Respuesta por el puerto serial

Una vez que se haya realizado el envío de datos a través del puerto serial hacia la red MODBUS RTU, se espera que el nodo al cual fue enviada la petición de información envíe su respuesta. Para verificar que el buffer de entrada del puerto serial tenga información se hace uso de la función *serZrdUsed (serDrdUsed)*. Esta función no tiene ningún variables

como parámetro. Al ser invocada retorna el número de bytes que están en el buffer de llegada del puerto serial y están en espera de ser leídos.

9. Calcular la longitud de la nueva trama

Primeramente se debe leer la información que llega por el puerto serial. Para esto se hace uso de la funcion *serZread()*. La cual es el resultado de la generalización de la función *serDread()* del puerto D. La sintaxis de esta función es la siguiente:

	serZread(parámetro1, parametro2, parametro3)
parámetro 1	Este parámetro es un arreglo de variables de tipo char que representa el buffer de recepción del puerto serial.
parámetro 2	Este parámetro es una variable de tipo *int* que representa el número de bytes a ser leídos del puerto serial.
parámetro 3	Este parámetro es una variable de tipo *int* que representa un retardo de tiempo máximo de espera entre un byte de llegada y el próximo.

Una vez que se ha leído la información del puerto serial a través de la función Para determinar la longitud de la trama que va a ser enviada por el puerto TCP hacia la red MODBUS TCP primero se debe conocer la longitud de la trama MODBUS RTU recibió por el puerto serial. Dependiendo del PDU de respuesta MODBUS RTU esta información puede ser obtenida de uno de los campos de esta trama. Es así que para extraer el valor de longitud contenido dentro de la trama MODBUS RTU se hace uso de la función *DataLen1()*. La sintaxis de esta función es la siguiente:

	DataLen1(parámetro 1, parámetro 2)
parámetro 1	Este parámetro es un arreglo de variables de tipo char que representa el buffer de llegada del puerto serial sobre el cual se esta realizando la comunicación MODBUS RTU.
parámetro 2	Este parámetro es un arreglo de variables de tipo char que representa el buffer de salida del puerto TCP por el cual se realiza la comunicación con la red MODBUS TCP.

Esta función lo que hace es tomar el valor del campo que contiene la longitud del PDU de respuesta y luego de sumarle 3 unidades lo asigna a los campos que corresponden a la longitud de la nueva trama MODBUS TCP. Es necesario sumar 3 unidades ya que el valor de longitud de la trama no incluye el byte que contiene la dirección del dispositivo, ni el byte que contiene el código de la función, ni el byte que contiene la longitud.

Adicional a esto es necesario sumar el resto de bytes que forman la cabecera TCP/IP para que el valor de la longitud este completo y sea entendido por el dispositivo MODBUS TCP. Para esto se hace uso de la función *DataLen2()*. La sintaxis de esta función es la siguiente:

	DataLen2(parámetro 1)
parámetro 1	Este parámetro es un arreglo de variables del tipo *char* que representa el buffer de llegada del puerto serial. Esta función suma al valor de la longitud los bytes que forman parte de la cabecera TCP para la trama MODBUS TCP.

1. **Armar la nueva trama y enviarla por el puerto TCP**

La nueva trama se ha ido formando desde que se empezó a calcular la longitud de la misma bajo el uso de la función *DataLen1()*. Para forma la nueva trama se hizo uso de la función *memcpy()*. El *Figura 4.7* representa como es formada la nueva trama MODBUS TCP que va a ser enviada por puerto TCP.

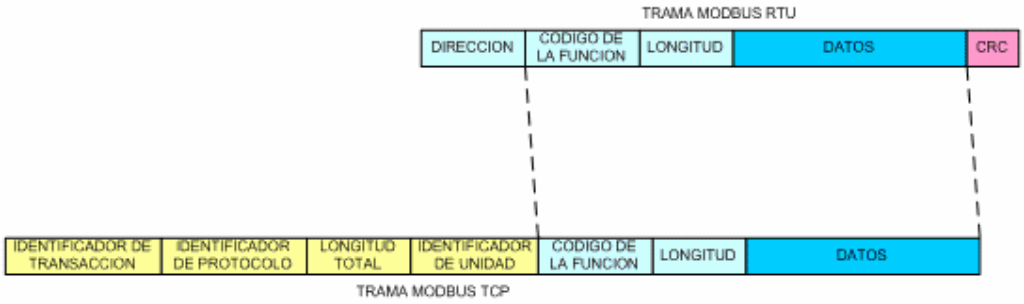


Figura 4.7. Ubicación de la trama MODBUS RTU en la trama MODBUS TCP

Una vez armada la nueva trama, el envío por el puerto TCP se lo hace mediante el uso de la función *sock_write()*. La sintaxis de esta función es la siguiente:

	sock_write(parametro1, parametro2, parametro3)
parámetro 1	Este parámetro es una variable del tipo *socket_tcp*, que representa el socket bajo el cual se empezó inicialmente la comunicación.
parámetro 2	Este parámetro es una arreglo de variables de tipo *char* que representa el buffer en donde se encuentra almacena la trama que va a ser enviada por el puerto TCP.
parámetro 3	Este parámetro es una variable de tipo *int* que representa el numero de bytes que van a ser enviado por el puerto TCP.

4.4 Software de aplicación.

Luego de haber implementado la lógica de programación explicada en la sección anterior hemos logrado que un dispositivo con capacidades de comunicación serial pueda comunicarse con dispositivos y paquetes de programas de monitoreo cuya único puerto de comunicación disponible sea una interfaz Ethernet. Para aquellas aplicaciones en las cuales se desea integrara un dispositivo MODBUS RTU en una red MODBUS TCP, se ha diseñado al gateway de comunicaciones con capacidad de servidor WEB. De esta manera, es posible tener una interfaz de monitoreo a traves de paginas WEB a las cuales se tiene acceso mediante el uso de navegadores de paginas web (web browsers), programas que son adquiridos de manera gratuita para las computadoras.

Estas páginas web tienen la flexibilidad de ser configuradas o diseñadas de acuerdo a los requerimientos del cliente y aplicaciones en las cuales se lo vaya a implementar. Es así como a través de páginas WEB se puede visualizar valores que pueden ser de vital importancia dentro de un proceso industrial. Como ventajas del monitoreo WEB se pueden considerar las siguientes:

1. Dependiendo de la calidad y de la infraestructura de la red, es posible tener acceso a través de INTERNET a variables de vital importancia para la toma de decisiones dentro de la empresa.

2. Las variables registradas en la página WEB pueden ser fácilmente vinculadas a otros programas de bases de datos o de reportes tales como Microsoft Access o

Excel. De esta manera el manejo de la información es más fácil y se requiere menor tiempo y recursos.

3. El diseño de la interfaz grafica (pagina web) puede ser adaptada fácilmente a los requerimientos del cliente y sobre todo pueden ser de características muy amigables para interactuar con el usuario final.

4. Los navegadores Web son de los programas mas utilizados por las personas en cualquier área de trabajo. Por tal motivo se reduce cualquier tiempo de capacitación que pudiera requerir un software propietario diseñado para el monitoreo a través de Ethernet.

Debido a las capacidades de procesamiento y de memoria que posee el controlador embebido existen ciertas limitaciones para el diseño de las paginas Web para el monitoreo de variables. Las principales desventajas o limitaciones son las siguientes:

1. La capacidad de memoria no permite que las páginas WEB tenga muchas animaciones o contengan gráficos de alta calidad. Lo cual reduciría las posibilidades de diseño estético de las interfaces.

2. Debido a que la acción de cargar las paginas WEB implica unos periodos mínimos de retardo, dicha aplicación no es aplicable con variables de cambio rápido o que estén involucradas dentro de lazos de control en donde el tiempo de respuesta la principal característica para un optimo desempeño del controlador como tal.

4.4.1 Uso de las herramientas del Controlador Embebido para la implementación del servidor WEB.

Gracias que el controlador embebido posee extensas librerías para el manejo de los protocolos de aplicación del modelo de referencia TCP/IP, la incorporación de las características de servidor WEB a la aplicación no requieren mayor programación que el uso adecuado de las herramientas que posee el controlador.

En primera instancia es necesario tener claro que se va a ser uso de un protocolo de la capa de aplicación. Dicho protocolo, como se mencionó en el capitulo 2, es el protocolo HTTP (Hyper Text Transfer Protocol). Este protocolo hace uso del puerto TCP 80 para realizar su comunicación.

4.4.1.1 Inclusión de las librerías HTTP

La librería que contiene todas las funciones para la implementación del protocolo HTTP es *http.lib*. El comando para incluir la librería es el siguiente *#use "http.lib"* el cual se lo ubica al comienzo del programa junto con los demás parámetros de configuración iniciales y librerías necesarias.

4.4.1.2 Inclusión de la página WEB a la memoria del Controlador Embebido

Como se comento en el inicio de esta sección, el diseño de la página WEB dependerá de los requerimientos del cliente y de la aplicación. Independiente de esto, el procedimiento para incluir la página WEB dentro del controlador es el mismo.

En primer lugar tenemos que guardar en la memoria de nuestro controlador el archivo que contiene la página WEB y todos los archivos necesarios que contengan imágenes y otros elementos para esto se hizo uso de la directiva de compilación *#ximport* . La sintaxis de esta directiva es la siguiente.

	***#ximport** "filename" symbol*
Filename	Es la dirección en donde se encuentra el archivo que se desea guardar [1]|.
Symbol	Es una macro generada por el compilador para relacionar el archivo dentro del programa [1].

Lo que hace esta función hace es almacenar la longitud del archivo y su contenido en forma binaria en la siguiente localidad de memoria disponible dentro de la memoria flash.

Por ejemplo tenemos lo siguiente:

1. *#ximport "samples/tcpip/http/pages/static.html" index_html*

2. #ximport "samples/tcpip/http/pages/rabbit1.gif" rabbit1_gif

3. #ximport "samples/tcpip/http/ssi.c" ssi_c

En el caso numero 1 tenemos el ejemplo de como se almacena un archivo de pagina WEB con extensión *html*. En el caso número 2 tenemos el ejemplo del almacenamiento de una imagen de formato *gif*. Finalmente en el caso número 3 tenemos el ejemplo de almacenamiento de un programa en lenguaje C.

Luego es necesario definir una tabla de recursos en donde se asocie los archivos con macros utilizadas en el programa. A continuación se muestra un ejemplo de la tabla de recurso.

```
SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/index.html", index_html),
        SSPEC_RESOURCE_XMEMFILE("/rabbit1.gif", rabbit1_gif)
SSPEC_RESOURCETABLE_END
```

Cuando se manejan archivos de diferentes extensiones es necesario realizar una tabla MIME (Multipurpose Internet Mail Extensions). MIME no es mas que especificaciones orientadas a especificar el intercambio de diferentes tipos de archivos con diferentes extensiones y que esto sea transparente al usuario final. Es decir, la transferencia de un archivo de video o de texto es igual y el usuario no percibe diferencias en los procedimientos aun que estos si existan.
La siguiente porción de código ejemplifica la estructura de una tabla MIME.

```
SSPEC_MIMETABLE_START
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".pdf", "application/pdf"),
        SSPEC_MIME(".gif", "image/gif")
SSPEC_MIMETABLE_END
```

Luego de estas especificaciones, el controlador embebido esta listo para almacenar cualquier tipo de archivo que se vaya a utilizar en su servidor WEB.

4.4.1.3 Desarrollo de Páginas WEB Dinámicas

El propósito de implementar un servidor Web en el controlador embebido es el de tener la capacidad de monitorear variables útiles para el desarrollo de reportes, cuadros estadísticos o supervisar procesos de producción. Nada de esto sería posible si no se hiciera uso de páginas Web dinámicas.

Se entiende por paginas Web dinámicas a aquellas cuya información presentada en pantalla varia de acuerdo a parámetros programados o registros configurables. Dentro del controlador embebido existen dos tipos de páginas Web dinámicas. Paginas Web con formas HTML y paginas sin formas HTML.

Las páginas Web sin formas HTML se basan en dos tipos de recursos para hacer las páginas dinámicas.

1. *Características SSI (Server Side Incluyes).-* Las características SSI son un conjunto comandos que permiten la interrelación entre el navegador que ejecuta la pagina Web y variables almacenadas en el Servidor Web que contiene la misma pagina.

2. *Características CGI (Common Gateway Interface).-* Estos son programas que están almacenados dentro del Servidor Web que contiene la pagina y que se ejecutan el momento que se los llame desde la página Web que esta siendo ejecutada en el navegador. De esta manera se interactúa con la información que se presenta en las páginas Web.

Las páginas Web con formas HTML se basan en funciones CGI para manipular los datos que son ingresados en los formularios formados en la programación HTML de la página Web. Esta es una manera en la cual se registra información en línea y se la envía al servidor para que se ejecutan comandos, acciones o se realicen reportes.

Dependiendo de los requerimientos del cliente y/o del proceso, cualquiera de estas alternativas puede ser implementada dentro del controlador embebido para que cumpla el alcance del proyecto.

4.4.1.4 Autenticación para la página Web.

Debido a que la información que se va a presentar en la página Web puede ser confidencial y solamente debe ser manipulada y visualizada por personal autorizado, es necesario implementar algún tipo de seguridades que garanticen el acceso a la información. Es por este motivo que se implementa técnicas de Autenticación en el servidor Web contenido dentro del controlador embebido.

Existe una Autenticación Básica usada por defecto, esta es la autenticación HTTP/1.0. Este esquema no es un método seguro de autenticación de usuarios a través de una insegura red, como por ejemplo el Internet. Sin embargo, adicionales esquemas de autenticación y mecanismos de encriptación son empleados para incrementar la seguridad. Desde la versión de Dynamic C 8.01 un nuevo sistema de autenticación conocido como HTTP Digest Authentication esta incluido dentro de sus capacidades. Este sistema en vez de enviar la clave en texto claro, como lo hace la autenticación básica, utiliza un método de encriptación conocido como MD5 (Message-Digest algorithm 5).

Para hacer uso de esta opción de encriptación se define la macro USE_HTTP_DIGEST_AUTHENTICATION con el valor de 1.

CAPÍTULO 5

IMPLEMENTACION

5.1 Integración.

Luego de haber desarrollado en el capítulo anterior la lógica de programación que se debe implementar dentro del controlador. Es necesario la implementación de la misma dentro del módulo, así como la integración del controlador como gateway dentro de una aplicación.

5.1.1 Integración del Hardware.

En el capitulo 3 se presentó un modelo gráfico de un prototipo del controlador embebido como un dispositivo electrónico con conectores externos listos para conexión en una carcasa plástica lista para ser montada en una Riel DIN. Para el desarrollo de este proyecto no se hizo mayor estudio ni esfuerzo sobre este modelo prototipo. Las conexiones se deben realizar directamente a los conectores de la tarjeta de prototipos, a la cual esta conectada el controlador embebido como ya se detallo en el capítulo 3.

a) Conexión al Puerto de Programación

La programación del controlador se hace a través de un puerto serial RS-232. También puede hacerse uso de conversores USB-SERIAL para programas desde computadores que no posean el puerto serial RS-232. Adicional a esto es necesario configurar ciertos parámetros de comunicación en el software Dynamic C. Estas configuraciones adicionales serán detalladas mas adelante. El *Figura 5.1* muestra la conexión desde el computador en el cual fue desarrollado el programa y el módulo.

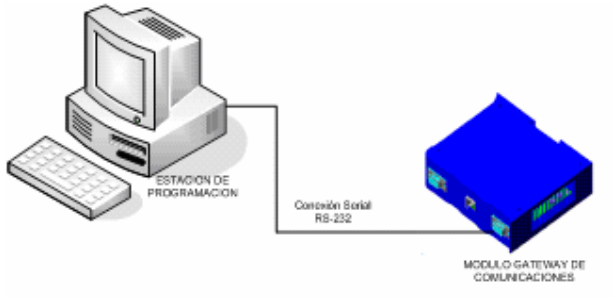


Figura 5.1. Conexión básica para programar el gateway de comunicaciones

b) Conexión a la red Ethernet

El único requerimiento físico para la conexión del controlador a la red Ethernet es la selección apropiada del cable Ethernet. Para la selección apropiada del cable se deben considerar dos aspectos. El primero es la velocidad a la que se va a trabajar y el segundo a que tipo de dispositivo se va a conector el controlador.

Con respecto a la velocidad de transmisión, se conoce que el controlador posee una interfaz Ethernet de 10Mbps, basados en este valor determinamos que el tipo de cable a ser usado es un par trenzado Cat 5 o Cat 5e. Cualquier categoría superior sería un sobre dimensionamiento de capacidad de transmisión ya que las características de la tarjeta Ethernet del controlador no da mayor capacidad. Al contrario, si se utilizara categoría inferior a la Cat 4, se estaría limitando el ancho de banda al medio de conexión, en este caso el cable Ethernet.

El conocer el tipo de equipo al cual se va a conectar el controlador es importante ya que define que tipo de cable se vaya a usar en términos de conexionado. Este puede ser un cable cruzado o un cable directo. El *Figura 5.2* la distribución en el conexionado tanto del cable directo como la del cable cruzado.

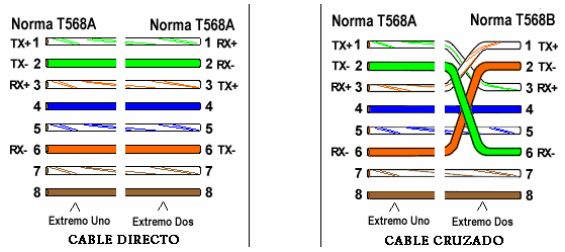


Gráfico 5.2. Conexionado de cable directo y cable cruzado.

El caso del cable cruzado se utilizará cuando el controlador se conecte directo a un computador con una aplicación MODBUS TCP Maestro o cuando se conecte a un Controlador Lógico Programable (PLC) con interfaz MODBUS TCP. Este cable también se lo utiliza para conectar el módulo a un router Ethernet. El *Figura* 5.3 esquematiza esta conexión.

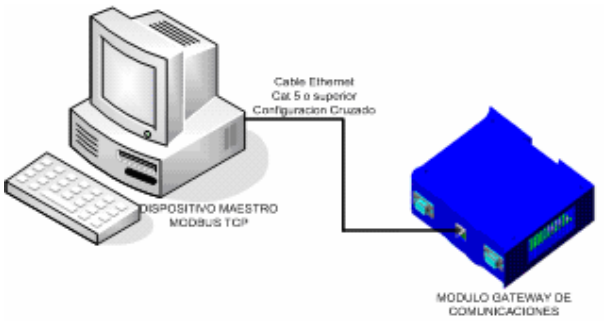


Figura 5.3. Conexión al puerto Ethernet al gateway de comunicaciones

El caso del cable directo es utilizado cuando el controlador se conecte a través de un equipo de networking como pasarelas, hubs, switches, etc. El caso del router es un caso particular por que a pesar de ser un equipo de networking, la conexión a este se la hace a través de un cable directo. El *Figura* 5.4 esquematiza esta conexión.

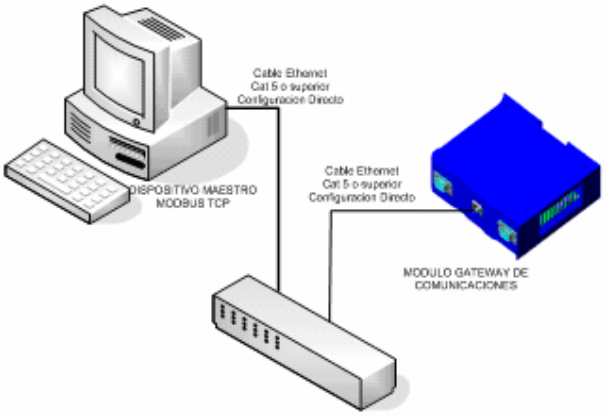


Figura 5.4. Integración del gateway de comunicaciones a una red Ethernet.

c) Conexión a la red Serial

Para la conexión hacia la red serial se hará uso del puerto C, el cual tiene disponibilidad de conectividad a través de un cable con conector RS-232. Por ser un conector RS232, la conexión MODBUS seria punto a punto con otro equipo MODBUS serial. El *Figura 5.5* esquematiza la conexión.

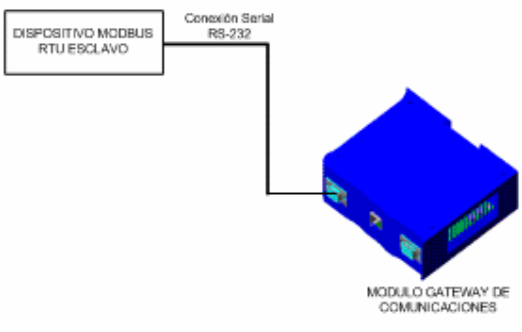


Figura 5.5. Conexión serial punto- punto al gateway de comunicaciones

Mediante el uso de conversores RS232 a RS485 se puede tener una red serial de varios dispositivos esclavos. De los cuales el dispositivo maestro conectado en la red Ethernet podrá leer la información necesaria. El *Figura 5.6* esquematiza la conexión.

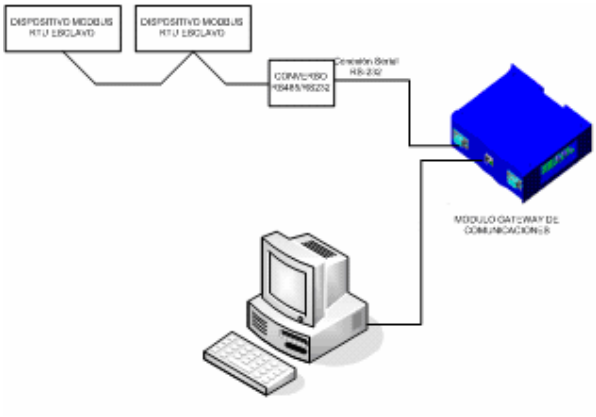


Figura 5.6. Conexión serial multipunto al gateway de comunicaciones

5.1.2 Integración del Software.

Luego de haber explicado la lógica que debe ejecutar el controlador para realizar las tareas de gateway de comunicaciones entre la red MODBUS RTU y la red MODBUS TCP, es necesario implementar dicha lógica con el uso de las funciones y librerías disponibles en Dynamic C.

a) Conexión a la Red Ethernet.

En primer lugar se le debe dar al controlar una configuración IP. Esta configuración IP incluye:

- Dirección IP
- Mascara de Red
- Dirección de Gateway

Toda la configuración IP esta resumida en la asignación de un valor determinado a la macro TCPCONFIG. El valor asignado a esta macro determinara la manera en la que el controlador recibirá su dirección IP. Si la macro toma el valor de uno, la configuración IP será la que esta guardada en la librería *tcp_ip* y se le conoce como estática. Si la macro toma el valor de 3, la configuración IP se asignara dinámicamente en base al protocolo DHCP.

Es preferible usar una configuración estática debido a que en la configuración de comunicaciones del dispositivo MODBUS TCP se debe determinar la dirección IP a la cual se va a conectar. En este caso, la dirección del controlador embebido.

#define TCPCONFIG 1

Luego de incluir el comando indicado en la línea superior, es necesario modificar de dirección IP que se encuentran dentro de la librería para que la dirección este dentro de la red a la cual el modulo embebido va a formar parte. A continuación se muestran los pasos para realizar esta modificación.

Primero, luego de instalar el software de programación Dynamic C, se crea una carpeta dentro del disco principal (típicamente disco C) con nombre *DCRABBIT_10.05*. Dentro de esta carpeta se encuentran todos los archivos de librerías, los archivos de ejemplos y demás archivos necesarios para desarrollo de proyectos con el módulo.

Dentro de esta carpeta, se debe ingresar a la carpeta *lib* (contiene todas las librerías). Una vez aquí, se debe entrar en la carpeta *tcpip*. Dentro de esta carpeta se debe abrir el archivo *tcp_config.lib*. La *Figura 5.7* muestra la ubicación del archivo *tcp_config.lib*.

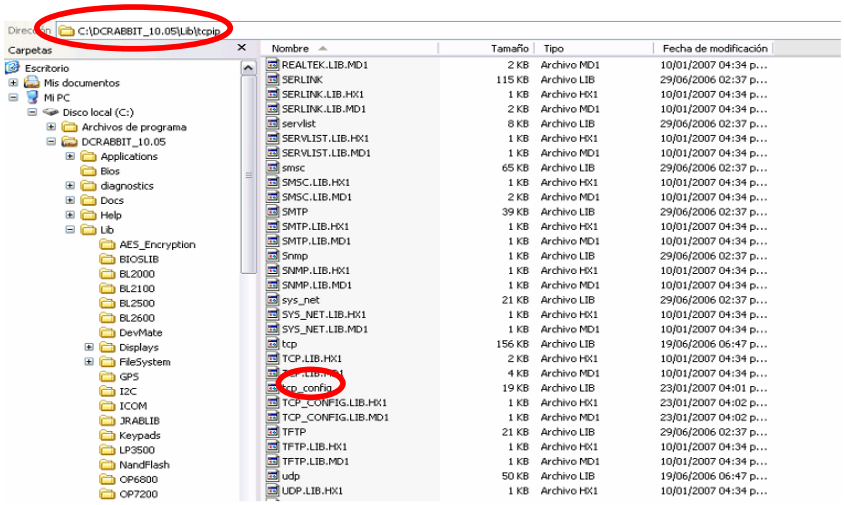


Figura 5.7. Ubicación archivo de configuración TCP/IP

Dentro del programa es necesario editar los siguientes parámetros:

_PRIMARY_STATIC_IP.- Representa la dirección IP que va a tener el modulo.
_PRIMARY_NETMASK.- Representa la masca de red para la dirección IP
MY_NAMESERVER.- Representa la dirección IP del servidor de la red.
MY_GATEWAY.- Representa la dirección IP gateway de la red Ethernet.

La *Figura 5.8* muestra la ubicación de los parámetros de configuración IP a ser cambiados para que el modulo forme parte de la red Ethernet.

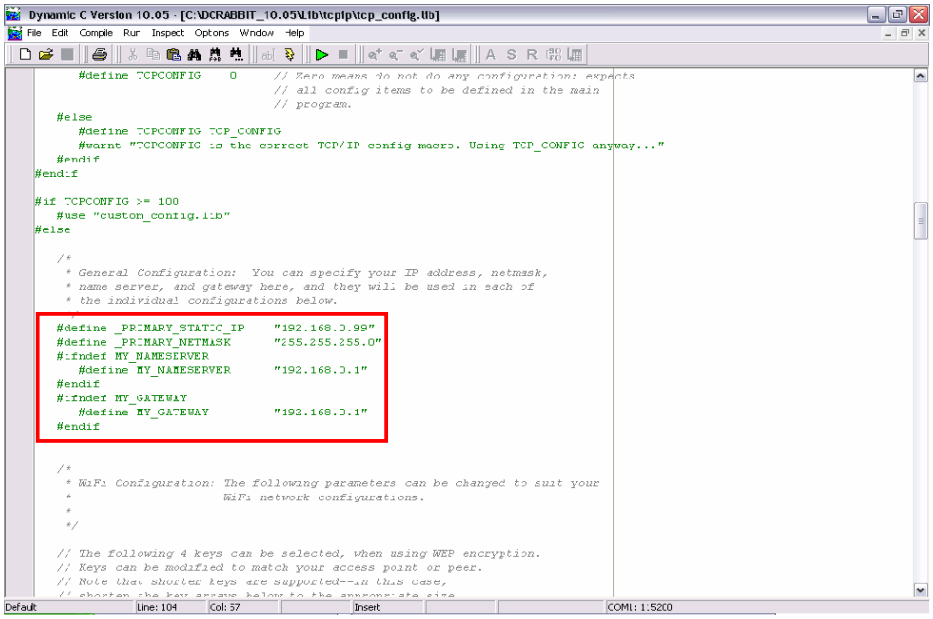


Figura 5.8. Parámetros de configuración IP

La asignación de direcciones IP se la realizo en base a una clase C usando la red 192.168.0.0. De esta manera se pueden usar cualquier dirección desde 192.168.0.1 hasta 192.168.0 255. Las pruebas de configuración se las realizo según el esquema mostrado en la *Figura 5.9*

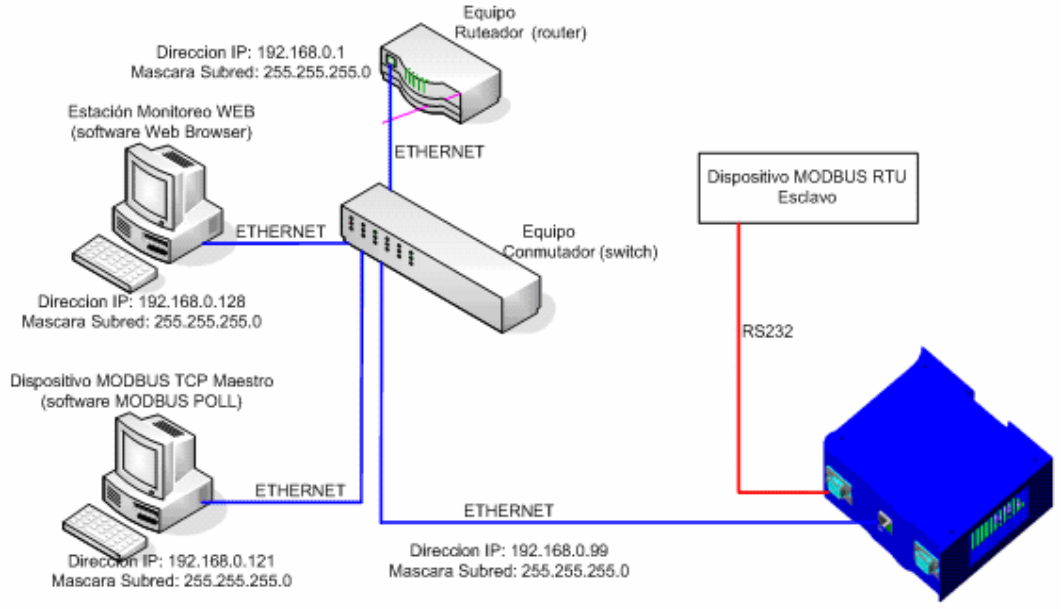


Figura 5.9. Ejemplo de conexión es una red Ethernet

Luego de realizar la configuración de los parámetros IP, el módulo embebido esta listo para formar parte de una red Ethernet.

b) Conexión al puerto de programación

Antes de descargar el programa a la memoria del controlador embebido es necesario realizar ciertas configuraciones en el software de programación Dynamic C. Primero dentro del Dynamic C se debe entrar en la pestaña de *Options* de la barra del menú principal del programa, ubicado en la parte superior de la pantalla. La *Figura 5.10* muestra la ubicación de la pestaña de *Options*.

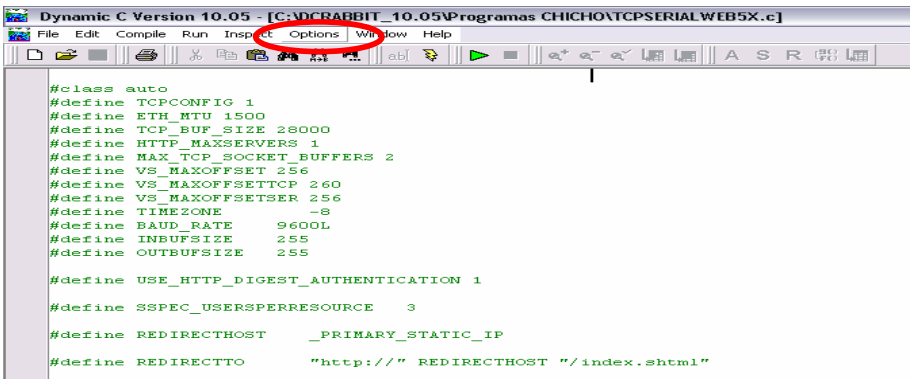


Figura 5.10. Opciones en la barra de Herramientas

La *Figura 5.11* muestra la ventaja que aparece al hacer clic en la pestaña *Options* y tomar la opción *Project Options*

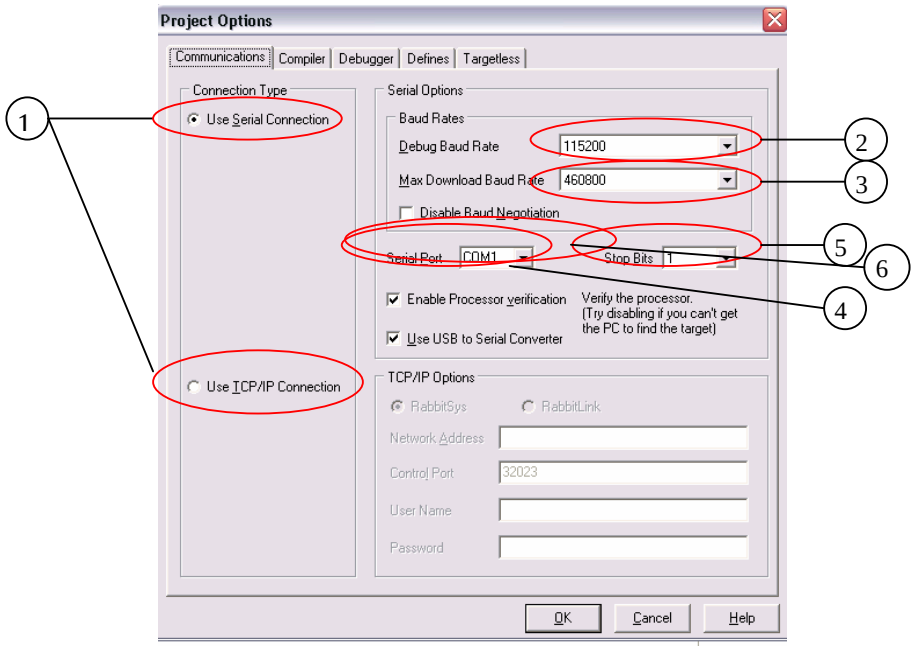


Figura 5.11. Opciones de programación

La primera opción configurable es la de las comunicaciones:

1. Se debe escoger que tipo de comunicación se va a realizar con el módulo embebido. Esta puede ser de dos opciones. La primera es una conexión serial y la segunda opción es una conexión TCP/IP. Ya que nuestro puerto TCP/IP va a ser utilizado dentro de la aplicación. Se escoge como opción la comunicación serial.

2. y 3. Son las velocidades seriales a las cuales se va a realizar la depuración del programa y se va a descargar el programa a la memoria del módulo embebido.

4. Es la opción para seleccionar que puerto serial va a usar el computador, donde se encuentra el programa Dynamic C, para comunicarse con el módulo embebido. Es necesario verificar que el puerto que se haya escogido no este siendo usando por ninguna

otra aplicación. Este puede ser un punto muy común de error ya que si el puerto esta siendo ocupado por otra aplicación el compilador nunca podrá comunicarse con el módulo.

5. Esta la opción del número de bit de parada que van a ser utilizados en la comunicación serial.

6. Esta opción debe ser seleccionada en el caso que el computador en donde se este trabajando no posean un puerto serial nativo y sea necesario el uso de un conversor USB serial, muy comunes en el mercado.

Luego de realizar las configuraciones de comunicaciones es necesario escoger el tipo de modulo al cual se va a conectar. Esto se realiza en la pestaña *Targetless.* La *Figura 5.12* muestra la pantalla de esta selección.

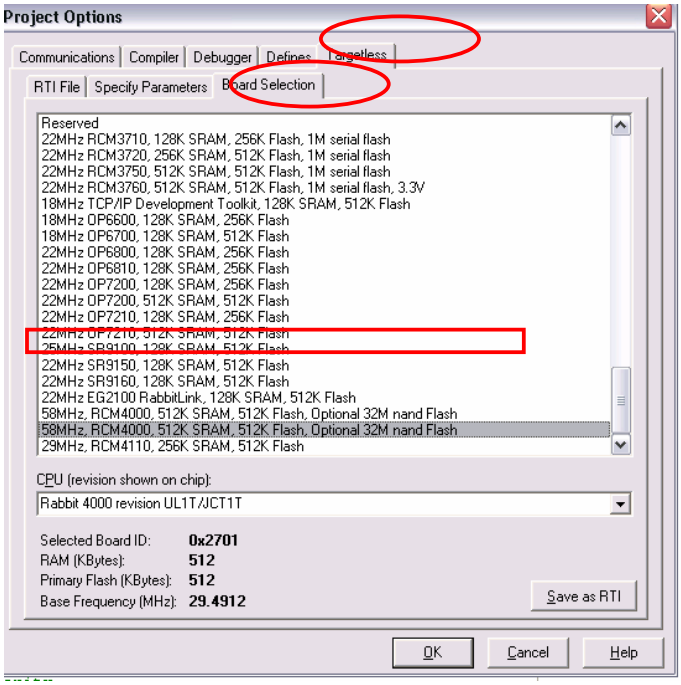


Grafico 5.12. Selección del modulo a programas

Para este proyecto se trabajo con el módulo Rabbit RCM4000 de 512K SRAM, 512K Flash y velocidad de reloj de 58MHz. Su selección se muestra en el grafico 5.12. Luego de realizar todo este proceso de configuración, se ha configurado el software de programación

Dynamic C para que pueda comunicarse con el modulo embebido y pueda descargarse el programa desarrollado.

c) Conexión a la Red Serial.

Luego de realizar la conexión física de acuerdo a lo explicado en el inicio de este capitulo. Existen ciertos parámetros de comunicación que deben ser configurados para establecer una comunicación exitosa a través del puerto serial con el dispositivo MODBUS RTU esclavo. Estos parámetros son los siguientes:

1. Velocidad de Transmisión: Para definir la velocidad de transmisión bajo la cual el puerto serial va a trabajar es necesario usar una función de la librería *RS232.lib. serDopen(vs_info.baud).* Esta función tiene como parámetro un valor *long int* que representa la velocidad de transmisión del enlace serial. Es indispensable que la velocidad de transmisión sea la misma tanto para el modulo embebido así como para el dispositivo MODBUS RTU esclavo conectado a este puerto serial. Caso contrario, la comunicación no se puede realizar.

2. Chequeo de Error: La trasmisión serial tiene un método de verificación de la integridad de los bits que recibe. Este método es conocido como paridad. La transmisión puede ser configurada para que el chequeo de error se lo realice mediante paridad par o impar. Esta configuración se la realiza mediante el uso de la función *serDparity(PARAM_EPARITY).* Esta función tiene como parámetro un valor *int* que representa el tipo de paridad con la que se va a trabajar de acuerdo a lo siguiente:

PARAM_NOPARITY	*0x00*
PARAM_EPARITY	*0x01*
PARAM_OPARITY	*0x02*

En el programa final a través de una definición de una variable se le asigno el valor de 1 al texto *PARAM_EPARITY* .

5.2 Puesta en Punto.

Previo a la realización de las pruebas que verifiquen el correcto funcionamiento del módulo de control embebido como un gateway de comunicaciones MODBUS RTU a MODBUS TCP, es necesario definir ciertos aspectos sobre el desarrollo del mismo.

La *figura* 5.13 muestra el esquema bajo el cual se realizarán las pruebas:

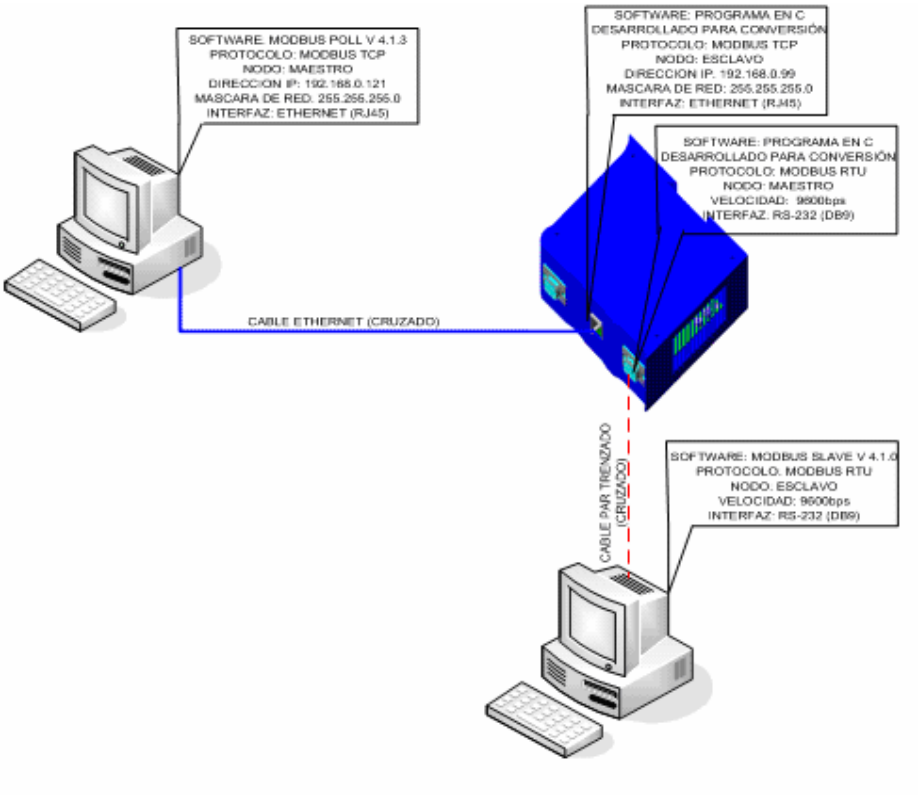


Figura5.13. Conexión y configuración para pruebas

CAPITULO 6

PRUEBAS Y RESULTADOS

6.1 Introducción

Para la realización de la etapa de pruebas del dispositivo en su etapa final se hizo uso de simuladores los cuales fueron usados en la etapa de desarrollo para entender de mejor manera la trama de comunicaciones tanto del protocolo MODBUS RTU como del protocolo MODBUS TCP. Dichos simuladores fueron obtenidos de una página WEB que contiene herramientas de desarrollo para personas interesadas en el protocolo MODBUS (www.modbustool.com), por lo cual su adquisición fue gratuita. Estos programas fueron utilizados para simular el funcionamiento de un dispositivo MODBUS TCP master y un dispositivo MODBUS serial. En este capitulo se detallará como fueron realizadas las pruebas de simulación. Adicionalmente, se mostrarán las pantallas que verifiquen sus resultados. Finalmente se entregara una lista de acciones que verifiquen la integridad de la conectividad para cada uno de los enlaces.

Para finalizar la etapa de pruebas se usara el modulo Controlador Lógico Programable Compact 602 de la marca MODICON para verificar que el dispositivo es de propósito general y puede ser usado con cualquier equipo con comunicación MODBUS.

6.2 Descripción de las Herramientas de Pruebas.

6.2.1 Programa MODBUS SLAVE

El programa Modbus Slave Versión 4.1.0 desarrollado por la empresa Witte Software es un programa creado para ser ejecutado para la plataforma Windows de 32 bits. Sirve como elemento de simulación de un dispositivo esclavo tanto para

redes seriales así como para redes Ethernet. Dentro de sus prestaciones tenemos las siguientes características:

a) Interfaz de usuario amigable que permite la rápida utilización del programa.

b) Soporta las siguientes funciones MODBUS:

- 01: Read coil status
- 02: Read input status
- 03: Read holding register
- 04: Read input registers
- 05: Force single coil
- 06: Preset single register
- 15: Force multiple coils
- 16:Preset multiple registers
- 22: Mask write register
- 23: Read/Write registers

c) Permite el monitoreo del tráfico de comunicaciones a través del puerto serial.

d) Fácil comunicación a través de OLE con programas tales como Visual Basic, Excel, etc.

e) Comunicación configurable para MODBUS RTU, MODBUS ASCII y MODBUS TCP

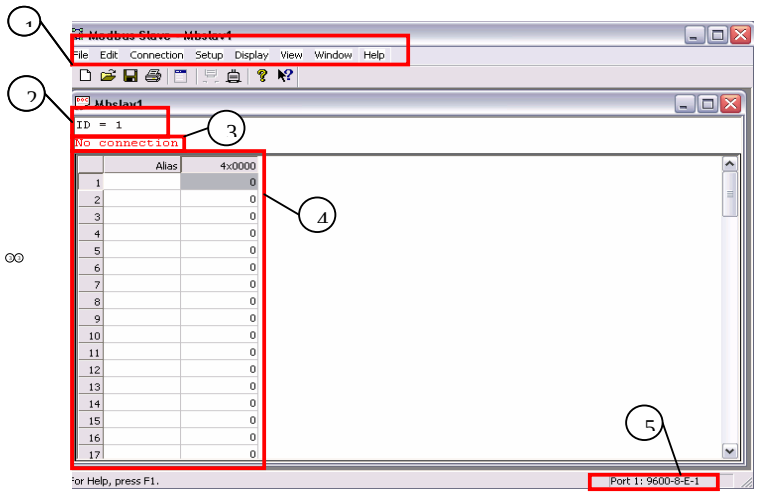


Figura 6. 1. Pantalla principal software MODBUS SLAVE.

La *Fgiura 6.1* muestra la pantalla principal del programa MODBUS SLAVE. A continuación explicaremos cada una de las partes importantes dentro de esta interfaz.

1. Menú de Herramientas

a) File.- Permite las acciones comunes en todo programa. Crear un nuevo archivo, guardar cambios de un archivo, guardar el archivo, imprimir, etc.

b) Edit.- Permite las acciones de copiar, cortar dentro del programa.

c) Connection.- Permite dar el comando de para que el simulador se conecte a la red serial y pueda comunicarse.

d) Setup.- Permite la configuración los parámetros de un dispositivo modbus slave en el simulador. Tomando la opción *Slave Definition* aparece la ventana mostrada en la *Figura* 6.2.

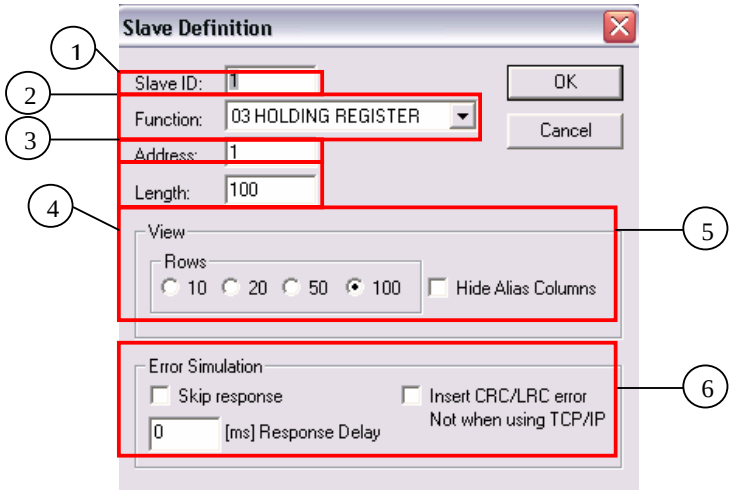


Figura 6.2. Parámetros de configuración dispositivo esclavo.

1. *SlaveID:* Entrada para el valor de la dirección que va a tener el simulador como dispositivo esclavo dentro de la red MODBUS.

2. Function: Configura para que el simulador entienda la función que se ha configurado y de respuesta a un mensaje de petición de esta función.

3. Address: Es la entrada para la dirección inicio desde la cual se va a realizar la función Modbus especificada en el ítem anterior. Como se estudio en el capitulo 2, las funciones MODBUS realizan sus operaciones con la información en base a direcciones de registros. En su mayoría las funciones trabajan con unas direcciones de inicio y una longitud que le indica hasta que registro se debe trabajar partiendo desde la dirección inicio.

4. Length: Es un valor numérico que indica el número de registro con los que va a trabajar la función MODBUS. Por ejemplo, si se usa la función 03 (lectura de registros de memoria) el valor de length establecerá hasta cuantos registros tiene disponible el dispositivo esclavo para ser leído por cualquier dispositivo maestro.

5. View: Es una herramienta que te define el número de registros o de variables que se desean visualizar en el simulador. Ayuda a manejar de mejor manera las variables que se muestran en el simulador.

6. Error Simulation: Herramienta que permite simular errores en la transmisión tales como retarnos en la transmisión o errores en el campo de verificación de redundancia cíclica.

e) Display.- Permite definir el formato numérico utilizado para la visualización de la información de los registros y variables discretas. Entre los diferentes tipos de formatos tenemos: enteros, enteros con signo, base hexadecimal, base binaria, flotante, etc. Adicional a esto tiene una opción de comunicación que permite la visualización de los paquetes que se envían y se reciben a lo largo de la comunicación. Dichos paquetes separados por bytes y visualizados en formato hexadecimal.

f) view.- Permite mostrar y ocultar las barra de herramientas y otras opciones de edición.

g) window.- Permite ordenar las diferentes pantallas generadas por el uso de varios dispositivos esclavos en el mismo simulador.

h) help.- Despliega los contenidos de ayuda desarrollados por los creadores del programa de simulación.

2. Numero ID.- Muestra el numero ID bajo el cual el dispositivo esclavo esta conectado a la red. Es decir su dirección dentro de la red Modbus.

3. Mensaje de Estado.- Es un mensaje que indica si el dispositivo esta conectado a la red o no.

4. Matriz de Variables.- Es una matriz de datos en donde se muestran los diferentes registros de datos y variables discretas, dependiendo del tipo de función que se este simulando.

5. Mensaje Configuración del Puerto.- Este mensaje muestra brevemente los parámetros bajo los cuales está configurado el puerto serial. La *Figura 6.3* muestra que significa cada uno de los campos en el mensaje de configuración del puerto.

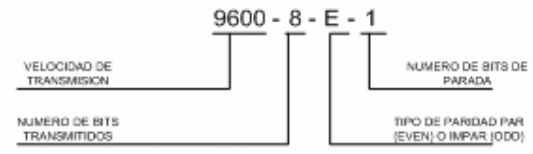


Figura 6.3. Mensaje de configuración del puerto en el software MODBUS SLAVE.

6.2.2 Programa MODBUS POLL

El programa Modbus Poll Version 4.1.3 desarrollado por la empresa Witte Software es un programa creado para ser ejecutado para la plataforma Windows de 32 bits. Sirve como elemento de simulación de un dispositivo master tanto para redes seriales así como para redes Ethernet. Mediante el uso de esta herramienta de software se pueden monitorear varios dispositivos esclavos Modbus y/o áreas de información al mismo tiempo. Para lograr esto en cada ventana se debe especificar el ID del dispositivo esclavo, que tipo de función se está usando, dirección de registro, tamaño y frecuencia del envío de mensajes. Adicionalmente, este software muestra las excepciones de error que pueden generarse en la comunicación con otros dispositivos. Dentro de sus prestaciones tenemos las siguientes características:

a) Interfaz de usuario amigable que permite la rápida utilización del programa.

b) Soporta las siguientes funciones MODBUS:

-
- 01: Read coil status
- 02: Read input status
- 03: Read holding register

- 04: Read input registers
- 05: Force single coil
- 06: Preset single register
- 15: Force multiple coils
- 16: Preset multiple registers
- 22: Mask write register
- 23: Read/Write registers

c) Permite el monitoreo del tráfico de comunicaciones a través del puerto serial.

d) Fácil comunicación a través de OLE con programas tales como Visual Basic, Excel, etc.

e) Comunicación configurable para MODBUS RTU, MODBUS ASCII y MODBUS TCP/IP.

f) Fácil control de convertidores RS-485 con la conmutación de la seña RTS.

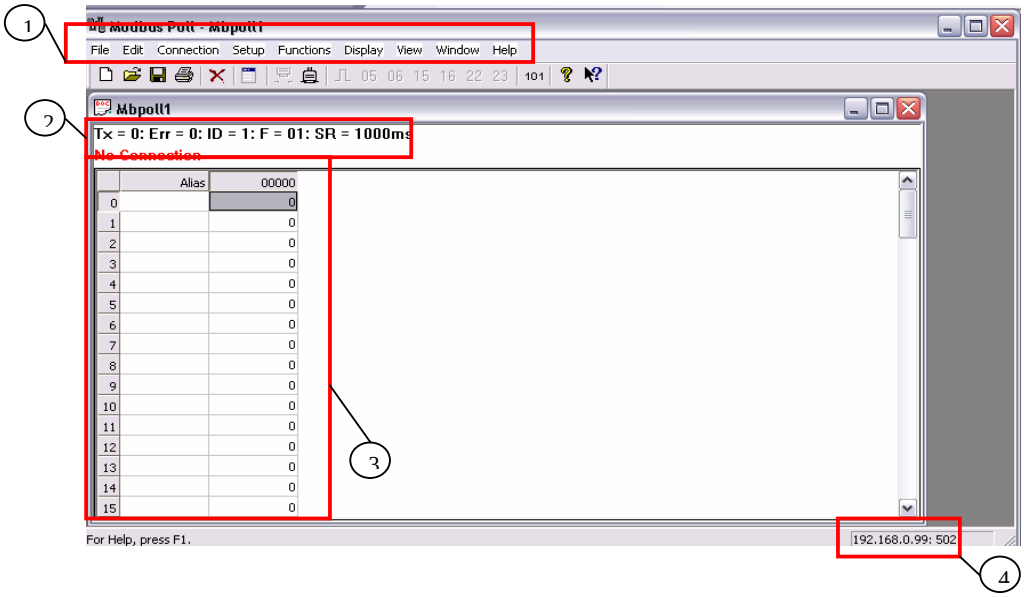


Figura 6.4. Pantalla principal del software MODBUS POLL

La *Figura 6.4* muestra la pantalla principal del programa MODBUS SLAVE. A continuación explicaremos cada una de las partes importantes dentro de esta interfaz.

1. Menú de Herramientas

a) File.- Permite las acciones comunes en todo programa. Crear un nuevo archivo, guardar cambios de un archivo, guardar el archivo, opciones de impresion, etc.

b) Edit.- Permite las acciones de copiar, cortar dentro del programa.

c) Connection.- Despliega las opciones de configuación necesarias para la conexión ya sea por puerto serial o por conexión TCP/IP. A continuación la *Figura 6.5* muestra todas las opciones configurables dentro de esta opción.

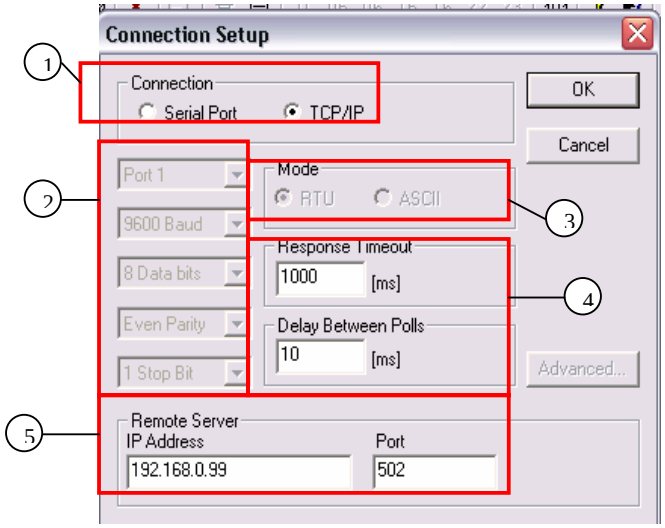


Figura 6.5. Pantalla de configuración MODBUS POLL

1. Conexión.- Esta opción permite establecer que tipo de conexión se va a realizar con los dispositivos Modbus. Teniendo como opciones el tradicional puerto serial y el novedoso puerto TCP/IP.

2. Configuración Conexión Serial.- Estos son los parámetros de configuración para que la comunicación serial con otros dispositivos Modbus puedan comunicarse con el panel central. De entre las opciones tenemos: el puerto del computador que va a hacer usado, velocidad de

transmisión es quien regula la chimenea. El siguiente campo define el tamaño de la trama de información, luego viene el tipo de paridad utilizado para la verificación de errores en los mensajes, finalmente el campo de los bits de parada.

3. Modo de Transmisión Modbus: Define el modo de transmisión bajo el cual se va a realizar la comunicación. Este puede ser RTU o ASCII.

4. Parámetros de Tiempo: Dentro de este campo de configuración se especifican dos valores importantes. El primero parámetro es el tiempo de espera que se da para recibir respuesta a una petición enviada antes de que se considere reenviar la petición por no haber sido respondida. El segundo parámetro es define el periodo de tiempo que debe transcurrir entre el envío de una petición y la siguiente.

5. Servidor Remoto: Estos parámetros son configurables para una conexión con un dispositivo esclavo bajo una red MODBUS TCP. Dentro de este campo tenemos dos parámetros configurables. El primero parámetro es la dirección IP del dispositivo esclavo al que se le van a enviar los mensajes. El segundo parámetro es el puerto al cual se van a enviar los mensajes. Dicho puerto debe estar previamente habilitado en los dispositivos esclavos que se encuentren conectados a la red. Por definición de protocolo se ha destinado al puerto 502 como el puerto de comunicación de MODBUS TCP.

d) Setup.- Permite la configuración de los parámetros de las funciones MODBUS. Adicionalmente permite la configuración de parámetros de restablecimiento de contadores y de administración de archivos de configuración. La *Figura 6.6* muestra las opciones configurables dentro de la pestaña *POLL DEFINITION*.

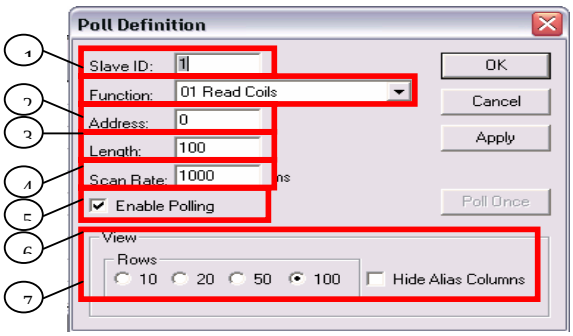


Figura 6.6. Pantalla de Configuración de dispositivo MODBUS maestro.

1. Slave ID: Representa la dirección del dispositivo MODBUS esclavo al cual se quiere enviar las peticiones de información.

2. Function: Representa una de las funciones del protocolo MODBUS. En este software de simulación se encuentran soportadas las siguientes funciones:

01: Lectura de Bobinas
02: Lectura de Entradas Discretas
03: Lectura de Registros
04: Lectura de Registros de Entrada

3. Address: Es un valor que sirve de referencia para especificar un origen de acuerdo a los requerimientos de las funciones MODBUS antes mencionadas.

4. Length: Es un valor que sirve de referencia para definir la cantidad de registro que se van a procesar de acuerdo a la funciones MODBUS utilizada.

5. Scan Time: Define el periodo de tiempo bajo el cual se ejecutan las peticiones.

6. Enabling Polling: Habilita la emisión de peticiones periódicas.

7. View: Permite configurar la ventana principal de visualización para definir el rango de valores que se van a mostrar.

e) Display.- Permite definir el formato numérico utilizado para la visualización de la información de los registros y variables discretas. Entre los diferentes tipos de formatos tenemos: enteros, enteros con signo, base hexadecimal, base binaria, flotante, etc. Adicional a esto tiene una opción de comunicación que permite la visualización de los paquetes que se envían y se reciben a lo largo de la comunicación. Dichos paquetes separados por bytes y visualizados en formato hexadecimal.

f) view.- Permite mostrar y ocultar las barra de herramientas y otras opciones de edición.

g) window.- Permite ordenar las diferentes pantallas generadas por el uso de varios dispositivos esclavos en el mismo simulador.

h) help.- Despliega los contenidos de ayuda desarrollados por los creadores del programa de simulación.

6.3 Procedimiento para Pruebas

Para la realización de las pruebas se utilizaron los softwares de simulación antes mencionados. El escenario para la realización de las pruebas responde al diagrama mostrado en la *Figura 6.7.*

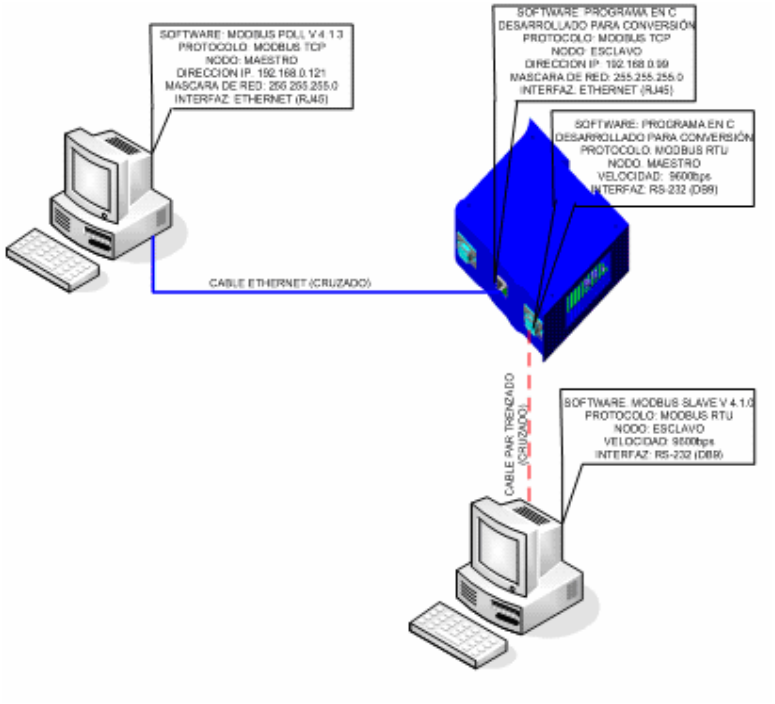


Figura 6.7. Esquema de conexionado para pruebas

De esta manera se verifica simultáneamente la correcta conectividad del Protocolo MODBUS sobre línea serial (MODBUS RTU) y el Protocolo MODBUS sobre la red Ethernet (MODBUS TCP).

Para comprobar que la tarjeta embebida esta realizando la conversión de protocolo correctamente se simulan el envío y recepción de mensajes MODBUS.

Durante el proceso de pruebas se simularon las funciones 01, 02, 03, 04 del conjunto de funciones del protocolo MODBUS. Cada una de las funciones se las prueba durante el periodo de tiempo que permite los software, ya q son versiones trial solo funcionan un periodo de tiempo de 10 minutos.

A continuación las siguientes imágenes muestran los resultados obtenidos en los procesos de pruebas

a) Función 01 (Lectura de Bobinas)

Se configuraron los programas de simulación de acuerdo al *gráfico 6.7* y se programa el envío de tramas de petición de la función 01 durante 10 minutos. En este lapso de tiempo se variaron ciertos valores de entradas en el programa *MODBUS SLAVE* para simular la transición de estado en las bobinas. En la *Figura 6.8* se visualizan las pantallas de ambos simuladores y se verifica que los valores que presenta el nodo maestro (programa MODBUS POLL) son los mismos valores que presenta el nodo esclavo (programa MODBUS SLAVE).

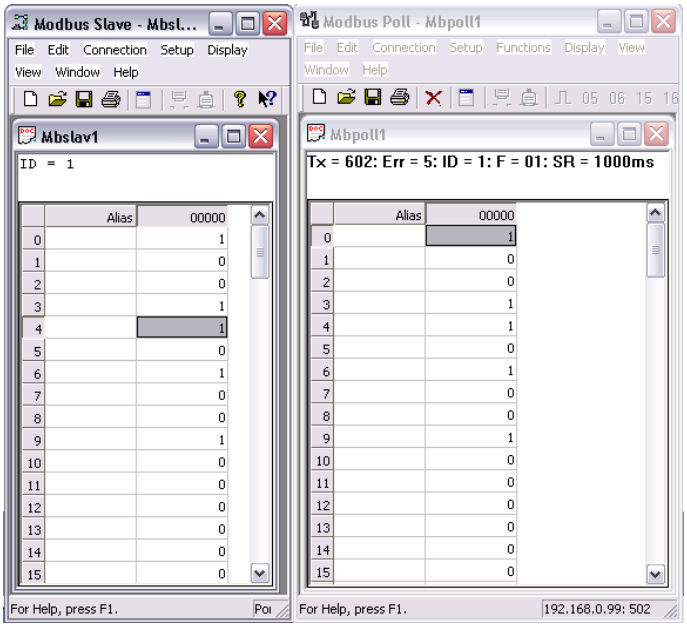


Figura 6.8. Pantallas de simuladores operando en función 1.

En la *Figura 6.8* se pueden obtener los siguientes comentarios: El número de tramas enviadas en el lapso de 10 minutos fueron 602. De las cuales 5 tramas se recibieron con algún error o distorsión en sus bits de información. Este valor se puede porcentualizar diciendo que menos del 1% del total de tramas de pruebas fueron no exitosas. Este margen de error obtenido es producto de la falta de sincronización inicial de arrancar los dos simuladores a la vez y la falta de sincronización al momento de expirar los tiempos de la versión trial.

b) Función 02

Se configuraron los programas de simulación de acuerdo a la *Figura 6.7* y se programa el envío de tramas de petición de la función 02 durante 10 minutos. En

este lapso de tiempo se variaron ciertos valores de entradas en el programa *MODBUS SLAVE* para simular la transición de estado en las entradas discretas. En la *Figura 6.9* se visualizan las pantallas de ambos simuladores y se verifica que los valores que presenta el nodo maestro (programa MODBUS POLL) son los mismos valores que presenta el nodo esclavo (programa MODBUS SLAVE).

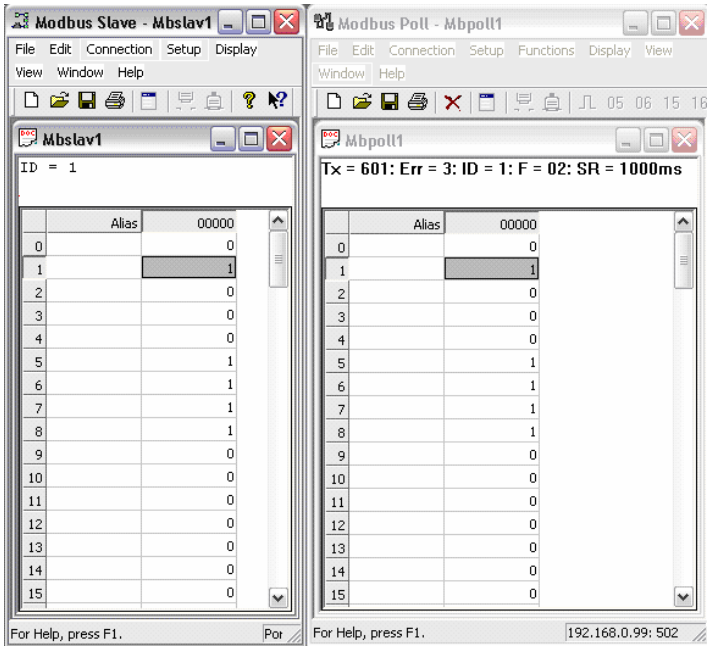


Figura 6.9. Pantallas de simuladores en función 2

En la *Figura 6.9* se pueden obtener los siguientes comentarios: El número de tramas enviadas en el lapso de 10 minutos fueron 601. De las cuales 3 tramas se recibieron con algún error o distorsión en sus bits de información. Este valor se puede porcentualizar diciendo que menos del 0.5% del total de tramas de pruebas fueron no exitosas. Este margen de error obtenido es producto de la falta de sincronización inicial de arrancar los dos simuladores a la vez y la falta de sincronización al momento de expirar los tiempos de la versión trial.

c) Función 03

Se configuraron los programas de simulación de acuerdo a la *Figura 6.7* y se programa el envío de tramas de petición de la función 03 durante 10 minutos. En este lapso de tiempo se variaron ciertos valores de entradas en el programa

MODBUS SLAVE para simular la transición de estado de los registro de memoria. En la *Figura 6.10* se visualizan las pantallas de ambos simuladores y se verifica que los valores que presenta el nodo maestro (programa MODBUS POLL) son los mismos valores que presenta el nodo esclavo (programa MODBUS SLAVE).

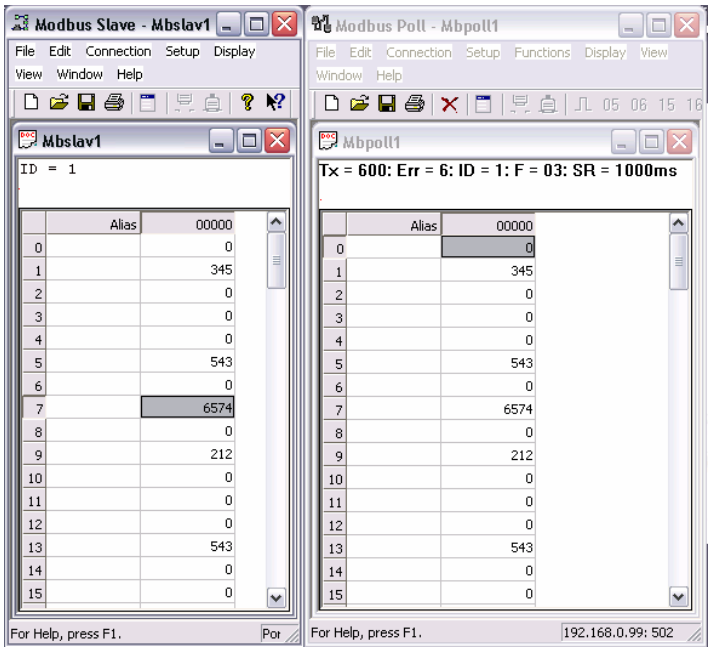


Figura 6.10. Pantalla de simuladores en función 3

En la *Figura 6.10* se pueden obtener los siguientes comentarios: El número de tramas enviadas en el lapso de 10 minutos fueron 600. De las cuales 6 tramas se recibieron con algún error o distorsión en sus bits de información. Este valor se puede porcentualizar diciendo que el 1% del total de tramas de pruebas fueron no exitosas. Este margen de error obtenido es producto de la falta de sincronización inicial de arrancar los dos simuladores a la vez y la falta de sincronización al momento de expirar los tiempos de la versión trial.

d) Función 04

Se configuraron los programas de simulación de acuerdo a la *Figura 6.7* y se programa el envío de tramas de petición de la función 04 durante 10 minutos. En este lapso de tiempo se variaron ciertos valores de entradas en el programa *MODBUS SLAVE* para simular la transición de estado de los registro de entrada. En

la *Figura 6.11* se visualizan las pantallas de ambos simuladores y se verifica que los valores que presenta el nodo maestro (programa MODBUS POLL) son los mismos valores que presenta el nodo esclavo (programa MODBUS SLAVE).

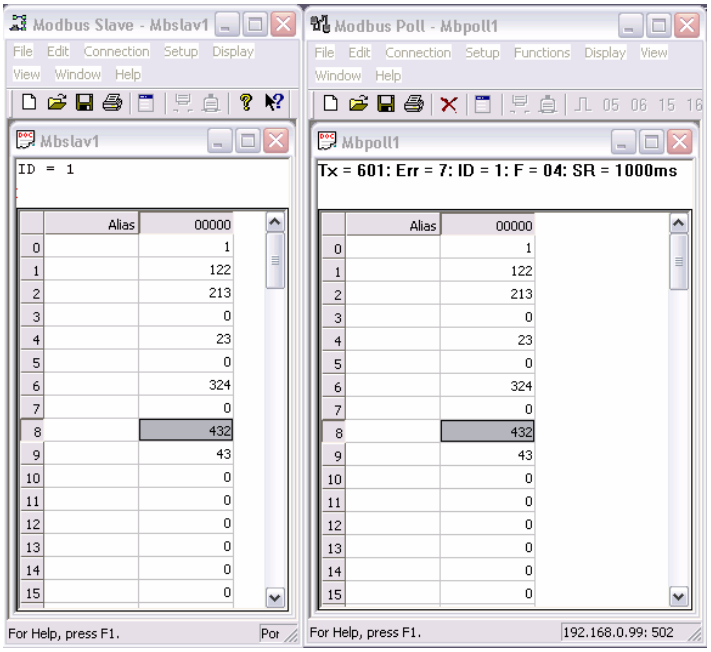


Figura 6.11. Pantalla de simuladores en función 4

En la *Figura 6.11* se pueden obtener los siguientes comentarios: El número de tramas enviadas en el lapso de 10 minutos fueron 601. De las cuales 7 tramas se recibieron con algún error o distorsión en sus bits de información. Este valor se puede porcentualizar diciendo que el 1% del total de tramas de pruebas fueron no exitosas. Este margen de error obtenido es producto de la falta de sincronización inicial de arrancar los dos simuladores a la vez y la falta de sincronización al momento de expirar los tiempos de la versión trial.

Finalmente para verificar que los valores de proceso que viajan dentro de las tramas de MODBUS son visualizables a través de una aplicación http se desarrollo una página WEB de prueba donde se presenta los 10 primeros valores de cada una de las funciones antes probadas (funciones: 01, 02, 03, 04). Las *Figuras* 6.12a/b muestran lo visualizado por la página WEB de prueba.

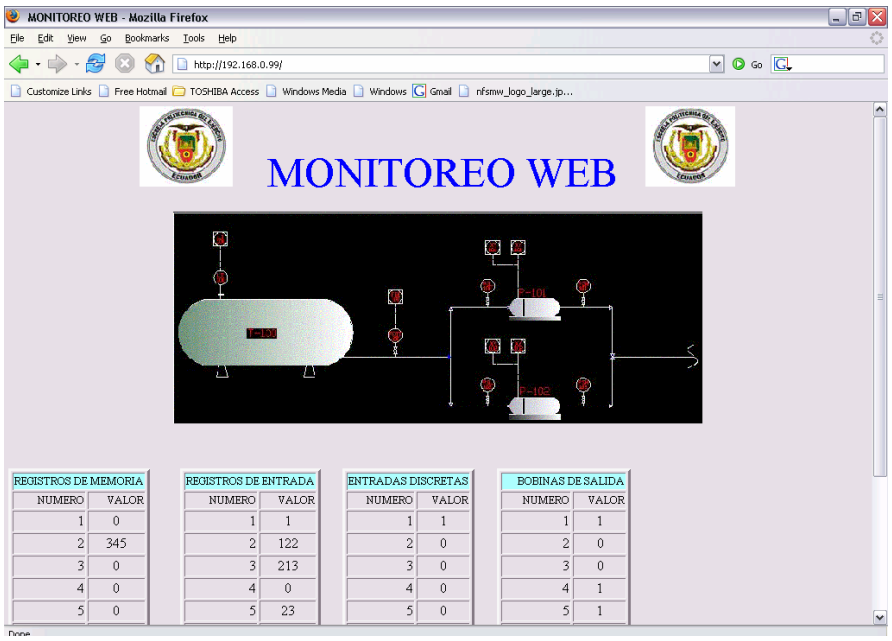


Figura 6.12a. Pantalla de pagina de monitoreo WEB (1).

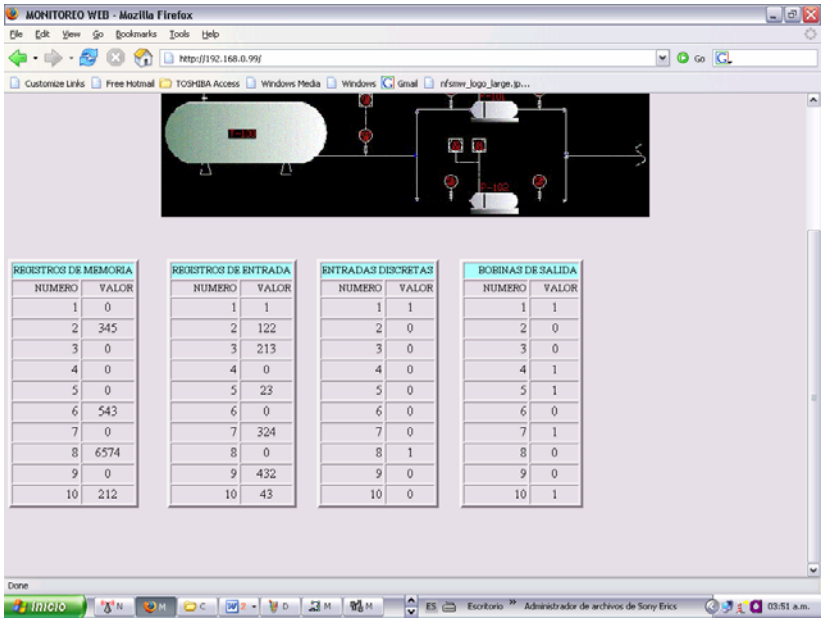


Figura 6.12b Pantalla de pagina de monitoreo WEB (2)

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones

1. La migración de protocolos seriales de buses de campo a redes Ethernet es posible mediante el uso de módulos de control embebido. Para lograr dicha migración es necesario el poseer total conocimiento de la composición de las tramas y características de transmisión del protocolo en cuestión.

2. La utilización de módulos de control embebido configurados como pasarelas de comunicación permiten la integración de dispositivos seriales en arquitecturas de control en donde estén presentes dispositivos de control más sofisticados y con comunicación Ethernet como opción integrada.

3. Gracias al hecho de la migración de redes seriales convencionales a redes Ethernet, es posible sacar provecho a tecnologías más desarrolladas como la tecnología inalámbricas (WIFI) o agrupar mayores dispositivos de control y anchos de banda mucho mayores como los que brindan las tecnologías 100BaseFX y 1000BaseFX.

4. Es posible realizar el monitoreo de variables de proceso a través de redes Ethernet sin la necesidad de gastar grandes cantidades de dinero en paquetes de programas informáticos complejos de visualización capaces de desarrollar una interfaz grafica, estos pueden ser reemplazados por aplicaciones HTTP, que a pesar de tener sus limitaciones visuales, pueden convertirse en una herramienta muy útil para la visualización de ciertos procesos industriales.

5. La lectura de la página WEB durante la comunicación de la red Ethernet y la red serial produce un retardo en la lógica del programa. Dicho retardo se debe a las simultaneas sesiones de puertos TCP que maneja el procesador del modulo de control embebido lo cual le exige mayores ciclos de maquinas para atender a todas las conexiones TCP establecidas.

6. La utilización de páginas WEB como herramientas de visualización de variables de proceso permite la fácil transmisión de información entre paquetes de programas compatibles y de fácil utilización tales como hojas de cálculo o procesadores de texto.

7. La utilización de módulos de control embebido con la pila del protocolo TCP/IP permite la utilización de aplicaciones TCP/IP tales como autenticación, encriptación correo electrónico, etc. La inclusión de código de autenticación en el modulo embebido permitió brindar niveles de seguridad requeridos en aplicaciones industriales.

7.2 Recomendaciones

1. Para un desarrollo exitoso de la migración de protocolos seriales a redes Ethernet es importante identificar cada una de las capas involucradas en el protocolo de acuerdo al modelo de referencia OSI. De esta manera es más fácil identificar la compatibilidad en los protocolos y desarrollar una migración exitosa.

2. El desarrollo de la aplicación HTTP (pagina WEB) no debe contener demasiado archivos de gran tamaño debido a que la memoria del módulo de control embebido no posee una gran capacidad de memoria para este propósito.

3. Debido al retardo de tiempo que puede haber al momento de actualizar la página WEB de visualización, no es recomendable involucrar variables de control dentro de la interfaz WEB ya que pueden ser parte de lazos críticos de control los cuales no toleran retardos de tiempo

4. Durante el proceso de direccionamiento IP se recomienda asignar al módulo de control embebido una dirección fija, ya que para la visualización y configuración de la red Ethernet es necesario conocer esta dirección.

REFERENCIAS BIBLIOGRAFICAS

CAPITULO 1

[1] ROBERTO VIGNONI FACULTAD DE INGENIERÍA UNLP, *Comunicaciones Industriales*, http://www.ing.unlp.edu.ar/electrotecnia/procesos/redesind.pdf, Octubre 2006

[2] THE INDUSTRIAL ETHERNET BOOK, *Technical Article: The Battle For Motion Control*, http://ethernet.industrial-networking.com/articles/articledisplay.asp?id=72Grid Connect , Octubre 2006

[3] *Power over Ethernet*, http://en.wikipedia.org/wiki/Power_over_Ethernet, Noviembre 2006.

[4]*CAPITULO V Comunicaciones Industriales*.
http://www.unap.cl/public/Redes%20Industriales.pdf, Noviembre 2006

[5] JUAN PABLO FERRARI, *Monografía sobre Sistemas de Control Distribuido*, Año 2005

[6] UNIVERSIDAD DE OVIEDO E INGENIERIA DE SISTEMAS Y AUTOMATICA, *Redes Locales*, www.isa.uniovi.es/docencia/redes/apuntes/tema4.pdf

[7] A.I KOKKINAKI AND M.J MORSE, *Industrial Communications Architectures And Their Application in a Garment Computer Integrated Manufacturing Cell*,

[8] SISCO, *Overview and Introduction to the Manufacturing Message Specification (MMS)*, http://www.sisconet.com/downloads/mmsovrlg.pdf, Noviembre 2006

[9] JOHN KORSAKAS Y GRID CONNECT, *Adding Industrial Ethernet with minimal software changes*, http://www.industrial-embedded.com/PDFs/GridConnect.Oct05.pdf, Octubre 2006

CAPITULO 2

[1] MODICON, INC., INDUSTRIAL AUTOMATION SYSTEMS**,** *Modicon Modbus Protocol Reference Guide PI–MBUS–300 Rev. J*, 1996

[2] MODBUSIDA.ORG, *MODBUS over Serial Line Specification and Implementation Guide*, 2006

[3] MODBUS-IDA, *MODBUS Application Protocol Specification v1.1a, 2004*

[4] ACADEMIA CISCO, *Curriculum CCNA 1*, http://www.cisco.com/web/LA/netacad/index.html, Noviembre 2006

[5] MODBUS-IDA.ORG, *MODBUS_Messaging_Implementation_Guide_V1_0a, 2004*

CAPITULO 3

[1] RABBIT SEMICONDUCTOR, *RabbitCore RCM4000 C-Programmable Analog Core Module with Ethernet User's Manual*, http://www.impulse-corp.co.uk/assets/imgs/prods/1293RCM4000_UM.pdf, Noviembre 2006.

CAPITULO 4

[1] Z-WORLD, INC***, Dynamic C TCP/IP*** *User's Manual Volume 2, Z-World,* http://www.compel.ru/images/catalog/222/tcpV1.pdf Diciembre 2006.

ANEXOS

ANEXO A

PROGRAMA LENGUAJE C PARA EL MODULO DE CONTROL

```
ESCUELA POLITECNCIA DEL EJÉRCITO
NOMBRE: CARLOS EDUARDO BOHORQUEZ GUTIERREZ
TEMA: IMPLEMENTACION DE UN NODO DE COMUNICACIONES MODBUS - TCP/IP A TRAVES DEL
USO DE UN MODULO DE CONTROL EMBEBIDO DE LA MARCA RABBIT SEMICONDUCTOR
PROGRAMA PARA EL DESARROLLO DE UN NODO DE COMUNICACIONES MODBUS-TCP/IP
17-04-2008
SANGOLQUI-ECUADOR
*/
#class auto
#define TCPCONFIG 1
#define ETH_MTU 1500
#define TCP_BUF_SIZE 28000
#define HTTP_MAXSERVERS 1
#define MAX_TCP_SOCKET_BUFFERS 2
#define VS_MAXOFFSET 256
#define VS_MAXOFFSETTCP 260
#define VS_MAXOFFSETSER 256
#define TIMEZONE        -8
#define BAUD_RATE    9600L
#define INBUFSIZE    255
#define OUTBUFSIZE   255

#define USE_HTTP_DIGEST_AUTHENTICATION        1

#define SSPEC_USERSPERRESOURCE   3

#define REDIRECTHOST              _PRIMARY_STATIC_IP

#define REDIRECTTO                "http://" REDIRECTHOST "/index.shtml"

#define serZopen     serDopen
#define serZread     serDread
#define serZgetc     serDgetc
#define serZrdUsed   serDrdUsed
#define serZwrite    serDwrite
#define serZclose    serDclose
#define serZwrFlush  serDwrFlush
#define serZrdFlush  serDrdFlush
#define serZwrFree   serDwrFree
#define serZrdFree   serDrdFree
#define serZparity   serDparity


#define DINBUFSIZE   INBUFSIZE
#define DOUTBUFSIZE  OUTBUFSIZE

#memmap xmem
```

```
#use "dcrtcp.lib"
#use "http.lib"

#ximport "samples/tcpip/http/pages/espe.gif"                    espe_gif
#ximport "samples/tcpip/http/pages/pid1.gif"                    pid1_gif
#ximport "samples/tcpip/http/pages/PWEB4.shtml"    PWEB4_html

SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".shtml", "text/html", shtml_handler),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
        SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END

int reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10;
int bob1,bob2,bob3,bob4,bob5,bob6,bob7,bob8,bob9,bob10;
int dic1,dic2,dic3,dic4,dic5,dic6,dic7,dic8,dic9,dic10;
int rent1,rent2,rent3,rent4,rent5,rent6,rent7,rent8,rent9,rent10;
int valreg,fun,var;
int reg[512],bobina[512],discreta[512],entrada[512];
int aux;
char  bufferweb[VS_MAXOFFSET];

SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/", PWEB4_html),
  SSPEC_RESOURCE_XMEMFILE("/index.shtml", PWEB4_html),
  SSPEC_RESOURCE_XMEMFILE("/espe.gif", espe_gif),
        SSPEC_RESOURCE_XMEMFILE("/pid1.gif", pid1_gif),
  SSPEC_RESOURCE_ROOTVAR("reg1", &reg1, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("reg2", &reg2, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("reg3", &reg3, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("reg4", &reg4, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("reg5", &reg5, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("reg6", &reg6, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("reg7", &reg7, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("reg8", &reg8, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("reg9", &reg9, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("reg10", &reg10, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("bob1", &bob1, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("bob2", &bob2, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("bob3", &bob3, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("bob4", &bob4, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("bob5", &bob5, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("bob6", &bob6, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("bob7", &bob7, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("bob8", &bob8, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("bob9", &bob9, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("bob10", &bob10, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("dic1", &dic1, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("dic2", &dic2, INT8, "%d"),
```

```
  SSPEC_RESOURCE_ROOTVAR("dic3", &dic3, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("dic4", &dic4, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("dic5", &dic5, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("dic6", &dic6, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("dic7", &dic7, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("dic8", &dic8, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("dic9", &dic9, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("dic10", &dic10, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("rent1", &rent1, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("rent2", &rent2, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("rent3", &rent3, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("rent4", &rent4, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("rent5", &rent5, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("rent6", &rent6, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("rent7", &rent7, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("rent8", &rent8, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("rent9", &rent9, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("rent10", &rent10, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("fun", &fun, INT8, "%d"),
  SSPEC_RESOURCE_ROOTVAR("valreg", &valreg, INT8, "%d"),
        SSPEC_RESOURCE_ROOTVAR("var", &var, INT8, "%d")
SSPEC_RESOURCETABLE_END

typedef struct
{
  int state;
  tcp_Socket socket;
  char  buffer[VS_MAXOFFSET];
  char  bufferTCP[VS_MAXOFFSETTCP];
        char  bufferTCPo[VS_MAXOFFSETTCP];
  char  bufferSER[VS_MAXOFFSETSER];
  char  bufferSERo[VS_MAXOFFSETSER];

} VsState;

int ch, bytes_written,cont;
int bytes_to_write, bytes_left;

//////////////////////////////////////////////////////

typedef struct
{
  int   port;           // port to listen on when in VS_MODEPASSIVE
  long  baud;             // serial port baud rates
  int   binary;         // TCP_MODE_BINARY or TCP_MODE_ASCII
} VsInfo;
//////////////////////////////////////////////////////

VsInfo vs_info;
VsState vs_state;
```

```
const VsInfo factory_defaults =
{
  502,                // port to listen on when in VS_MODEPASSIVE
  BAUD_RATE,          // serial port baud rates
  TCP_MODE_BINARY,    // TCP_MODE_BINARY or TCP_MODE_ASCII

};
//////////////////////////////////////////////////////

/* Table of CRC values for high–order byte */
const unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0,0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,
0x81, 0x40, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
0x41, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80,
0x41, 0x00, 0xC1, 0x81,0x400x40
} ;

/* Table of CRC values for low–order byte */
const char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5,
0xC4,0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B,
0xC9, 0x09,0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE,
0xDF, 0x1F, 0xDD,0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6,
0xD2, 0x12, 0x13, 0xD3,0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7,0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,
0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8,
0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C,
0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21,
0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,0x6E, 0xAE, 0xAA, 0x6A,
0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,0x7B, 0x7A, 0xBA,
0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,0x77, 0xB7,
0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,0x51,
0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C,
```

```
0x5C,0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58,
0x98, 0x88,0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D };
//////////////////////////////////////////////////////
void vs_init(VsState* vs_state)
{
  vs_state->state=VS_INIT;
  memcpy(&vs_info,&factory_defaults,sizeof(vs_info));
}
//////////////////////////////////////////////////////
int DataLen (char *bufferlen)
{
        int value;
                value = (int)bufferlen[4]*256+(int)bufferlen[5];
        return (value);
}
//////////////////////////////////////////////////////
void DataLen1 (char *bufferin, char*bufferout)
{
        int value, aux, cont;
        cont=0;

        value=(int)bufferin[1]+3;
        aux=value;

        if ( value <= 255)
        {
                bufferout[5]=(char)value;
        }
        else
        {
                while(value>255)
          {
                aux=aux-256;
             cont=cont+1;
          }
                bufferout[4]=(char)cont;
           bufferout[5]=(char)aux;
        }
}
//////////////////////////////////////////////////////
int DataLen2 (char *bufferin)
{
        int value;
                value=bufferin[1]+9;
        return(value);
}
//////////////////////////////////////////////////////
DELAY(int a)
{
        int cont,cont1;
```

```
        cont=0;
        while(cont < a)
                {
                  cont++;
                }
}
//////////////////////////////////////////////////////
/*Función para verificar los bytes CRC*/
void CRC16 ( char *puchMsg, int usDataLen )
{
        unsigned char uchCRCHi, uchCRCLo ;
        unsigned uIndex ;
  uchCRCHi = 0xFF;
  uchCRCLo = 0xFF;
  usDataLen =    usDataLen - 2;
        while (usDataLen--)
        {
                uIndex = uchCRCLo ^ *puchMsg++ ;
                uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
                uchCRCHi = auchCRCLo[uIndex] ;
        }
  puchMsg[usDataLen +1] = uchCRCLo;
        puchMsg[usDataLen +2] = uchCRCHi;
}
//////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
cofunc int webview(VsState* state)
{

int cn,fc,cn1,cn2;
char auxreg,auxreg1;
http_handler();
fc=(int)state->bufferTCPo[7];
fun=fc;
valreg=0;

 if(fc == 1)
 {
                valreg = (int)state->bufferTCPo[8];
     cn=0;
     cn1=0;
     cn2=0;
    while(cn<13)
    {
      while(cn1<8)
       {
          auxreg=state->bufferTCPo[9+cn]>>cn1;
          auxreg1=auxreg&(char)1;
          bobina[cn1+cn2]=auxreg1;
```

```
          cn1++;
         yield;
        }
        yield;
      cn2=cn2+8;
      cn1=0;
      cn=cn+1;
    }
//APLICACION A LA WEB
                    bob1=bobina[0];
          bob2=bobina[1];
          bob3=bobina[2];
          bob4=bobina[30];
          bob5=bobina[39];
          bob6=bobina[50];
          bob7=bobina[60];
          bob8=bobina[70];
          bob9=bobina[15];
          bob10=bobina[9];
     return 1;     //}
 }
 if(fc == 2)
 {
                    var=(int)state->bufferTCPo[8]+1;
     valreg = (int)state->bufferTCPo[8];
     cn=0;
     cn1=0;
     cn2=0;
     while(cn<13)
     {
       while(cn1<8)
       {
        auxreg=state->bufferTCPo[9+cn]>>cn1;
        auxreg1=auxreg&(char)1;
          discreta[cn1+cn2]=auxreg1;
          cn1++;
        yield;
       }
        yield;
          cn2=cn2+8;
          cn1=0;
          cn=cn+1;
                    }
//APLICACION A LA WEB
          dic1=discreta[0];
            dic2=discreta[1];
            dic3=discreta[2];
            dic4=discreta[3];
            dic5=discreta[4];
            dic6=discreta[5];
```

```
            dic7=discreta[6];
            dic8=discreta[7];
            dic9=discreta[8];
            dic10=discreta[15];
     return 1;
  }

if(fc == 3)
{
                    var=(int)state->bufferTCPo[8]+1;
                    valreg = (int)state->bufferTCPo[8];
     cn=0;
     cn1=0;
    while(cn < 100)
    {
                     reg[cn]=(int)state->bufferTCPo[9+cn1]*256+(int)state-
>bufferTCPo[9+cn1+1];
      cn1=cn1+2;
      cn=cn+1;
    yield;
   }
//APLICACION A LA WEB
          reg1=reg[0];
            reg2=reg[1];
            reg3=reg[2];
            reg4=reg[3];
            reg5=reg[4];
            reg6=reg[5];
            reg7=reg[6];
            reg8=reg[7];
            reg9=reg[8];
            reg10=reg[9];
     return 1;
}
if (fc == 4)
{
          var=(int)state->bufferTCPo[8]+1;

                    valreg = (int)state->bufferTCPo[8];
     cn=0;
     cn1=0;
     while(cn < 100)
     {
           entrada[cn]=(int)state->bufferTCPo[9+cn1]*256+(int)state-
>bufferTCPo[9+cn1+1];
       cn1=cn1+2;
       cn=cn+1;
       yield;
     }
//APLICACION A LA WEB
```

```
          rent1=entrada[0];
            rent2=entrada[1];
            rent3=entrada[2];
            rent4=entrada[3];
            rent5=entrada[4];
            rent6=entrada[5];
            rent7=entrada[6];
            rent8=entrada[7];
            rent9=entrada[8];
            rent10=entrada[9];
     return 1;
 }
 return 1;
}

//////////////////////////////////////////////////////////////

cofunc int vs_handler(VsState* state)
{
  auto tcp_Socket* socket;
  auto int cont1, leng;
  bytes_left=0;
  bytes_to_write=0;
  bytes_written=0;
  cont=0;
  cont1=0;
  leng=0;
  socket=&state->socket;

  serZopen(vs_info.baud);
  tcp_listen(socket,vs_info.port,0,0,NULL,0);

  while((-1 == sock_bytesready(socket)) && (0 == sock_established(socket)))

          yield;

   while(sock_established(socket)) {
          sock_mode(socket,vs_info.binary);
          bytes_to_write=sock_bytesready(socket);
          if(bytes_to_write>serZwrFree())
                    bytes_to_write=serZwrFree();

                if(bytes_to_write>(int)sizeof(state->buffer))
                bytes_to_write=sizeof(state->buffer);

            if(bytes_to_write>0)
           {
                    sock_read(socket,state->buffer,bytes_to_write);
             memcpy(state->bufferTCP,state->buffer,bytes_to_write);
                                        memcpy(state->bufferTCPo,state->buffer,8);
```

```
                memcpy(state->bufferSER,&state->bufferTCP[6],DataLen(state-
>bufferTCP));
                CRC16(state->bufferSER,DataLen(state->bufferTCP)+2);
                memcpy(state->buffer,&state->bufferSER,DataLen(state->bufferTCP)+2);
          serZwrite(state->buffer,DataLen(state->bufferTCP)+2);
         }


        bytes_to_write = serZrdUsed();
            if(bytes_to_write>sock_tbleft(socket))
                                bytes_to_write=sock_tbleft(socket);

                                if(bytes_to_write>(int)sizeof(state->buffer))
                                        bytes_to_write=sizeof(state->bufferSERo);

                                if(bytes_to_write > 0)
        {
                 serZread(state->bufferSERo,bytes_to_write,0);
            DataLen1(state->bufferSERo,state->bufferTCPo);
            memcpy(&state->bufferTCPo[8],&state->bufferSERo[1],DataLen2(state-
>bufferSERo)-8);
            DELAY(10000);
                                        DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            sock_write(socket,state->bufferTCPo,DataLen2(state->bufferSERo));
                 DELAY(10000);
                                        DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
            DELAY(10000);
        }

                yield;
  }

  sock_close(socket);
        return 1;
}

//////////////////////////////////////////////////////////////
void main()
{
  int user1, user1_enabled,page1,ch;
  sock_init();
```

```
  http_init();
  http_setauthentication(HTTP_DIGEST_AUTH);
  user1_enabled = 1;

        user1 = sauth_adduser("monitor", "tor", SERVER_HTTP);

        page1 = sspec_addxmemfile("/", PWEB4_html, SERVER_HTTP);
        sspec_adduser(page1, user1);
        sspec_setrealm(page1, "Admin");


  tcp_reserveport(80);
        vs_init(&vs_state);
  serZparity(PARAM_EPARITY);
  serZopen(vs_info.baud);

  while (1) {
                    costate {
          tcp_tick(NULL);
      }
             costate {
        http_handler();
      }
      costate {
                wfd vs_handler(&vs_state);
      }
      costate{
                wfd webview(&vs_state);
      }
   }
}
```

ANEXO B

TABLA DE FUNCIONES MODBUS

Función	Request PDU					Response PDU					Error	
Lectura de Bobinas	Codigo de la Función (0x01)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF)	Cantidad de Bobinas (2 bytes) (1 a 2000)			Codigo de la Función (0x01)	Cuenta de Bytes (1byte) (N)	Estado de las Bobinas (n bytes) (n = N o N+1)			Codigo de la Funcion (0x81)	Código de Excepción (1byte) (01, 02, 03 o 04)
Lectura de Entradas Discretas	Codigo de la Función (0x02)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF)	Cantidad de Entradas (2 bytes) (1 a 2000)			Codigo de la Función (0x02)	Cuenta de Bytes (1byte) (N)	Estado de las Entradas (N x 1byte)			Codigo de Error (0x82)	
Lectura de Registros de Memoria	Codigo de la Función (0x03)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF)	Cantidad de Entradas (2 bytes) (1 a 125)			Codigo de la Función (0x03)	Cuenta de Bytes (1byte) (2xN)	Valor de Registros (N x 2byte)			Codigo de la Funcion (0x83)	Código de Excepción (1byte) (01, 02, 03 o 04)
Lectura de un Registro en Entrada	Codigo de la Función (0x04)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF)	Cantidad de Registro (2 bytes) (0x0001 a 0x007D)			Codigo de la Función (0x04)	Cuenta de Bytes (1byte) (2xN)	Valor de Registros (N x 2byte)			Codigo de la Funcion (0x84)	Código de Excepción (1byte) (01, 02, 03 o 04)
Escritura de una sola Bobina	Codigo de la Función (0x05)	Dirección de Salida (2 bytes) (0x0000 a 0xFFFF)	Valor de Salida (2 bytes) (0x0000 a 0xFF00)			Codigo de la Función (0x05)	Dirección de Salida (2 bytes) (0x0000 a 0xFFFF)	Valor de Salida (2 bytes) (0x0000 a 0xFF00)			Codigo de la Funcion (0x85)	Código de Excepción (1byte) (01, 02, 03 o 04)
Escritura de un registro	Codigo de la Función (0x06)	Dirección del Registro de Salida (2 bytes) (0x0000 a 0xFFFF)	Valor de Registros (2 bytes) (0x0000 a 0xFFFF)			Codigo de la Función (0x06)	Dirección del Registro de Salida (2 bytes) (0x0000 a 0xFFFF)	Valor de Registros (2 bytes) (0x0000 a 0xFFFF)			Codigo de la Funcion (0x86)	Código de Excepción (1byte) (01, 02, 03 o 04)
Read Exception Status.	Codigo de la Función (0x07)					Codigo de la Función (0x07)	Dato de Salida (1 bytes) (0x00 a 0xFF)				Codigo de la Funcion (0x87)	Código de Excepción (1byte) (01, 02, 03 o 04)
Diagnostic	Codigo de la Función (0x08)	Sub - Función (2 bytes) (00-18, 20)	Datos (N x 2 bytes)			Codigo de la Función (0x08)	Sub - Función (2 bytes) (00-18, 20)	Datos (N x 2 bytes)			Codigo de la Funcion (0x88)	Código de Excepción (1byte) (01o 03 o 04)
Get COM event counter	Codigo de la Función (0x0B)					Codigo de la Función (0x0B)	Estado (2 bytes) (0x0000 a 0xFFFF)	Cuenta de Eventos (2 bytes) (0x0000 a 0xFFFF)			Codigo de la Funcion (0x8B)	Código de Excepción (1byte) (01 o 04)
Get COM Event Log	Codigo de la Función (0x0C)					Codigo de la Función (0x0C)	Cuenta de bytes (2 bytes)(N)	Estado (2 bytes) (0x0000 a 0xFFFF)	Cuenta de Eventos (2 bytes) (0x0000 a 0xFFFF)	Cuenta de Mensajes (2 bytes)(0x0000 a 0xFFFF)	Eventos ((N-6) x 1 byte)	Codigo de la Funcion (0x8C)	Código de Excepción (1byte) (01 o 04)
Escritura de Multiples Bobinas	Codigo de la Función (0x0F)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF	Cantidad de Salidas (2 bytes) (0x0001 a 0x07B0)	Cuenta de bytes (1 bytes)(N)	Valor de Salida (N x 1 bytes)	Codigo de la Función (0x0F)	Dirección de Inicio (2 byte) (0x0000 a 0xFFFF)		Cantidad de Salidas (2 byte) (0x0000 a 0x07B0)		Codigo de la Funcion (0x8F)	Código de Excepción (1byte) (01 or 02 or 03 or 04)
Escritura de Multiples Registros	Codigo de la Función (0x0F)	Dirección de Inicio (2 bytes) (0x0000 a 0xFFFF)	Cantidad de Registros (2 bytes) (0x0001 a 0x0078)	Cuenta de bytes (1 bytes)(2 x N)	Valores de los Registros (N x 2 bytes) (valor)	Codigo de la Función (0x10)	Dirección de Inicio (2 byte) (0x0000 a 0xFFFF)		Cantidad de Salidas (2 byte) (1 a 123(0x7B)		Codigo de la Funcion (0x90)	Código de Excepción (1byte) (01 or 02 or 03 or 04)
Report Slave ID	Codigo de la Función (0x11)					Codigo de la Función (0x11)	Cuenta de Bytes (1 byte)	ID del Esclavo (especificado por el dispositivo)		Indicador de Estado (1 byte)	Datos Adicionales	Codigo de la Funcion (0x91)	Código de Excepción (1byte) (01 or 04)
Lectura y Escritura de Multiples Registros	Codigo de la Función (0x17)	Direccion de Inicio de Lectura (2 bytes) (0x0000 a 0xFFFF	Cantidad para Leer (2 bytes) (0x0000 0x0076	Direccion de Inicio de Escritura (2 bytes) (0x0000 a 0xFFFF)	Cantidad para Escribir (2 bytes) (0x0000 0x0076	Contardor de bytes Escritos (1bytes) (2 x N)	Valores de los Registros a ser Escritos (N x 2bytes)	Codigo de la Función (0x17)	Cuenta de Bytes (1 byte(2 x N)		Valor de los Registros Leidos (0x17)	Codigo de la Funcion (0x97)	Código de Excepción (1byte) (01 or 02 or 03 or 04)
Escritura enmascara de registros	Codigo de la Función (0x16)	Direccion de Referencia (2 bytes) (0x0000 a 0xFFFF)	Mascara AND (2 bytes) (0x0000 a 0xFFFF)		Mascara OR (2 bytes) (0x0000 a 0xFFFF)	Codigo de la Función (0x16)	Direccion de Referencia (2 byte)	Mascara AND (2 byte) (0x0000 a 0xFFFF)		Mascara OR (2 byte) (0x0000 a 0xFFFF)		Codigo de la Funcion (0x96)	Código de Excepción (1byte) (01 or 02 or 03 or 04)
Read FIFO queue	Codigo de la Función (0x18)		Direccion del Puntero FIFO (2 byte)(0x0000 a 0xFFFF)			Codigo de la Función (0x18)	Cuenta de Bytes (2 byte)	Cuenta FIFO (2 byte) (<=31)		Valor de Registro FIFO (N x 2 byte)		Codigo de la Funcion (0x98)	Código de Excepción (1byte) (01 or 02 or 03 or 04)
Read File Record	Codigo de la Función (0x14)	Cuenta de Bytes (1 byte)(0x07 a 0xF5)	Sub - Req. X, Tipo de Referencia (1 byte)(0x06)	Sub - Req. X, Numero de Archivo (2 bytes) (0x0000 a 0xFFFF)	Sub - Req. X, Numero de Grabacion (2 byte) (0x0000 a 0x270F)	Sub - Req. X,Longitud del Registro (2 byte) (N)	Sub - Req. X+1	Codigo de la Función (0x14)	Resp. Longitud de Información (1 byte) (0x07 a 0xF5)	Sub - Req. X, Longitud del Archivo de Respuesta (1 byte) (0x07 a 0xF5)	Sub - Req. X, Tipo de Referencia (1 byte) (6)	Sub - Req. X,Datos Grabados (N x 2 byte)	Sub - Req. X+1	Codigo de la Funcion (0x94)	Código de Excepción (1byte) (01 or 02 or 03 or 04 or 08)
Write File Record	Codigo de la Función (0x15)	Longitud de la Información de Petición (1 byte)(0x07 a 0xF5)	Sub - Req. X, Tipo de Referencia (1 byte)(0x06)	Sub - Req. X, Numero de Archivo (2 bytes) (0x0000 a 0xFFFF)	Sub - Req. X, Numero de Grabación (2 bytes) (0x0000 a 0x270F)	Sub - Req. X, Longitud de Grabación (2 bytes)(N)	Sub - Req. X, Información Grabada (N x 2 bytes)	Sub - Req. X+1	Codigo de la Función (0x15)	Resp. Longitud de Información (1 byte)	Sub-Req. X, Tipo de Referencia (0x06)	Sub-Req. X, Numero de Archivo (2 bytes)(0x0000 a 0xFFFF)	Sub-Req. X, Numero de Grabación (2 bytes)(0x0000 a 0xFFFF)	Sub-Req. X, Longitud de Grabación (N x 2 bytes)	Sub - Req. X,Datos Grabados (N x 2 byte)	Sub - Req. X+1	Codigo de la Funcion (0x95)	Código de Excepción (1byte) (01 or 02 or 03 or 04 or 08)
Read device Identification	Codigo de la Función (0x2B)	Tipo de MEI (1 byte)(0x0E)		Información Especifica del Tipo de MEI (n byte)		Codigo de la Función (0x2B)	Tipo de MEI (1 byte)(0x0E)		Información Especifica del Tipo de MEI (n byte)			Codigo de la Funcion (0xAB: Fc 0x2B 0x80)	Tipo de MEI (1byte) (0x0E)	Código de Excepción (1byte) (01 or 02 or 03 or 04)

ANEXO C

MATRICES DE VARIABLES PARA LA VERIFICACION DE REDUNDANCIA CICLICA (CRC)

Valores Matriz ***auchCRCHi***

auchCRCHi

0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0, 0x80, 0x41, 0x01,0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40

Valores Matriz ***auchCRCLo***

auchCRCLo

0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,0x40

ANEXO E

DOCUMENTO DE VERIFICACION DE PRUEBAS

Documento de Verificación de Pruebas

Informacion General

1) **Nodo Maestro:**
   Unidad: *Computador de Escritorio con programa de simulación MODBUS POLL*
   Interfaz Fisica: *Conector RJ45*
   Protocolo Capa 1 *Ethernet*
   Protocolo Capa 2 *Ethernet*
   Protocolo Capa 7 *MODBUS TCP*

2) **Nodo Esclavo:**
   Unidad: *Computador de Escritorio con programa de simulación MODBUS SLAVE*
   Interfaz Fisica: *Conector DB9*
   Protocolo Capa 1 *RS-232*
   Protocolo Capa 2 *Maestro-Esclavo / codificacion RTU*
   Protocolo Capa 7 *MODBUS RTU*

3) **Gateway de Comunicaciones:**
   Unidad: *Controlador Embebido Rabbit RCM4000*
   Interfaz Física: *Conector RJ45 / Conector DB9*
   Protocolo Capa 1 Ethernet / RS-232
   Protocolo Capa 2 *Ethernet / Maestro-Esclavo / codificacion RTU*
   Protocolo Capa 7 *MODBUS TCP / MODBUS RTU*

Información de Configuración

1) **Nodo Maestro:**
   Dirección IP *192.168.0.121*
   Mascara de Red *255.255.255.0*
   Dirección de Gateway *192.168.0.1*
   Velocidad Transmisión *De acuerdo a medio 10/100Mbps*

2) **Nodo Esclavo:**
   Direccion *1*
   Velocidad Transmisión *9600 bps*
   Paridad: *Par*

Verificaciones Visuales

	SI	NO	OBSERVACIONES
Cable de Red Conectado	*X*		
Cable Serial Conectado	*X*		
Cable de Poder Conectado	*X*		
Led de Comunicación Ethernet Encendido	*X*		*Led de color ambar siempre encendido. Led de color verde intermitente*
Ping a la interfaz Ethernet del Gateway	*X*		*Se ejecuto el comando ping 192.168.0.99. Resultado exitoso*

Pruebas de Comunicación Protocolo

Función	# de solicitudes enviadas	tiempo de transmision	# de solicitudes error	% de error	Observaciones
01	*602*	*10 min*	*5*	*0,83056478*	*Errores producidos al iniciar simuladores y al expirar tiempo de simulacion*
02	*601*	*10 min*	*3*	*0,49916805*	*Errores producidos al iniciar simuladores y al expirar tiempo de simulacion*
03	*600*	*10 min*	*6*	*1*	*Errores producidos al iniciar simuladores y al expirar tiempo de simulacion*
04	*601*	*10 min*	*7*	*1,16472546*	*Errores producidos al iniciar simuladores y al expirar tiempo de simulacion*

Visualizacion de Resultados

Funcion: 01

# de variables	Valor en unid. Esclavo	Valor en unid. Maestra	Valor en pagina WEB	Observaciones
0	*1*	*1*	*1*	Transicion de valores rapida
1	*0*	*0*	*0*	Transicion de valores rapida
2	*0*	*0*	*0*	Transicion de valores rapida
3	*1*	*1*	*1*	Transicion de valores rapida
4	*1*	*1*	*1*	Transicion de valores rapida
5	*0*	*0*	*0*	Transicion de valores rapida
6	*1*	*1*	*1*	Transicion de valores rapida
7	*0*	*0*	*0*	Transicion de valores rapida
8	*0*	*0*	*0*	Transicion de valores rapida
9	*1*	*1*	*1*	Transicion de valores rapida
10	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
11	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
12	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
13	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
14	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros

Funcion: 02

# de variables	Valor en unid. Esclavo	Valor en unid. Maestra	Valor en pagina WEB	Observaciones
0	*0*	*0*	*0*	Transicion de valores rapida
1	*1*	*1*	*1*	Transicion de valores rapida
2	*0*	*0*	*0*	Transicion de valores rapida
3	*0*	*0*	*0*	Transicion de valores rapida
4	*0*	*0*	*0*	Transicion de valores rapida
5	*1*	*1*	*1*	Transicion de valores rapida
6	*1*	*1*	*1*	Transicion de valores rapida
7	*1*	*1*	*1*	Transicion de valores rapida
8	*1*	*1*	*1*	Transicion de valores rapida
9	*0*	*0*	*0*	Transicion de valores rapida
10	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
11	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
12	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
13	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
14	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros

Funcion: 03

# de variables	Valor en unid. Esclavo	Valor en unid. Maestra	Valor en pagina WEB	Observaciones
0	*0*	*0*	*0*	Transicion de valores rapida
1	*345*	*345*	*345*	Transicion de valores rapida
2	*0*	*0*	*0*	Transicion de valores rapida
3	*0*	*0*	*0*	Transicion de valores rapida
4	*0*	*0*	*0*	Transicion de valores rapida
5	*543*	*543*	*543*	Transicion de valores rapida
6	*0*	*0*	*0*	Transicion de valores rapida
7	*6574*	*6574*	*6574*	Transicion de valores rapida
8	*0*	*0*	*0*	Transicion de valores rapida
9	*212*	*212*	*212*	Transicion de valores rapida
10	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
11	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
12	*543*	*543*	*n/a*	En pagina WEB solo se visualiza 10 primeros
13	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
14	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros

Funcion: 04

# de variables	Valor en unid. Esclavo	Valor en unid. Maestra	Valor en pagina WEB	Observaciones
0	*1*	*1*	*1*	Transicion de valores rapida
1	*122*	*122*	*122*	Transicion de valores rapida
2	*213*	*213*	*213*	Transicion de valores rapida
3	*0*	*0*	*0*	Transicion de valores rapida
4	*23*	*23*	*23*	Transicion de valores rapida
5	*0*	*0*	*0*	Transicion de valores rapida
6	*324*	*324*	*324*	Transicion de valores rapida
7	*0*	*0*	*0*	Transicion de valores rapida
8	*432*	*432*	*432*	Transicion de valores rapida
9	*43*	*43*	*43*	Transicion de valores rapida
10	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
11	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
12	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
13	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros
14	*0*	*0*	*n/a*	En pagina WEB solo se visualiza 10 primeros

ÍNDICE DE FIGURAS

CAPITULO 1

CAPITULO 2

CAPITULO 3

CAPITULO 4

CAPITULO 5

CAPITULO 6

ÍNDICE DE TABLAS

CAPITULO 1

CAPITULO 2

CAPITULO 3

Fecha de entrega del Proyecto de Grado

El presente Proyecto de Grado "IMPLEMENTACIÓN DE UN NODO DE COMUNICACIÓN MODBUS-TCP/IP MEDIANTE EL USO DE UN MODULO DE CONTROL EMBEBIDO DE LA MARCA RABBIT SEMICONDUCTOR" fue entregado al Departamento de Eléctrica y Electrónica y reposa en la Escuela Politécnica del Ejército desde…….

Sangolquí, Ecuador

_____

Carlos Eduardo Bohórquez

Autor

_____

Ing. Víctor Proaño

Coordinador de Carrera