MODBUS RTU.
IMPLEMENTACIÓN DEL PROTOCOLO EN MICROCONTROLADOR.

ROGER TORRES SALAZAR

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2006

MODBUS RTU.
IMPLEMENTACIÓN DEL PROTOCOLO EN MICROCONTROLADOR.

ROGER TORRES SALAZAR

Proyecto de Grado presentado como requisito para optar al título de Ingeniero Electrónico.

Director
MSC JULIO AUGUSTO GELVEZ FIGUEREDO

Codirector
ING. JORGE ELIÉCER DUQUE PARDO
Ingeniero Electricista

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2006

AGRADECIMIENTOS

Expreso mi sincera gratitud a todas las personas que me aportaron tanto en lo técnico como en lo personal para la elaboración de este trabajo. Primero agradezco por los consejos y valiosos aportes al profesor Msc. Julio Augusto Gelvez Figueredo, director del proyecto y al ingeniero Jorge Eliécer Duque Pardo, codirector del mismo. De igual forma agradezco a los ingenieros Pedro Ardila y Marcela Carreño por la implementación del maestro Modbus que permitió validar el procesador de comunicaciones Modbus RTU, y por algunos aportes técnicos que sirvieron para su implementación.

Agradezco a Dios y a mis padres Mariano y Cecilia por brindarme esta invaluable oportunidad, a mis hermanos Mara, Arnold y Aarón por estar siempre presentes y a mi abuelo joche por su amistad.

Por último agradezco a Paola por su inmenso cariño y a mi futuro hijo Santiago Alexander por darme tantas cosas bellas.

TÍTULO: MODBUS RTU. IMPLEMENTACIÓN DEL PROTOCOLO EN UN MICROCONTROLADOR*

AUTOR: ROGER TORRES SALAZAR**

PALABRAS CLAVES: Modbus, bus de campo, red, procesador de comunicaciones, microcontrolador.

DESCRIPCIÓN: El procesador de comunicaciones Modbus RTU es un sistema basado en microcontrolador que ejecuta el protocolo Modbus en el modo de transmisión RTU con capacidad de formar parte de una red cumpliendo los requisitos de este protocolo.

La programación del microcontrolador se hizo con la herramienta software Codewarrior IDE V 3.0 en lenguaje C embebido. También se hizo un programa que permite la configuración por software del procesador.

Este procesador se conecta al bus con los estándares RS-232 ó RS-485 y tiene 1024 entradas y salidas discretas y 64 registros de entrada y salida. De estas entradas y salidas se pueden visualizar 8 salidas discretas, establecer 8 entradas discretas y permite leer un registro externo de una variable análoga.

Se validó el funcionamiento del procesador en una red con un PC como maestro que ejecuta un programa hecho en LabVIEW y un PLC TRILOGI como el otro esclavo de la red.

---

* Trabajo de Grado.

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones, E3T. Director: MSC. Julio Augusto Gélvez Figueredo; Especialista en Telecomunicaciones, Profesor de la E3T. Codirector: Ing. Jorge Eliécer Duque Pardo; Magíster en Ingeniería Electrónica, Profesor de la Universidad Tecnológica de Bolívar.

TITLE: MODBUS RTU. PROTOCOL IMPLEMENTATION IN A MICROCONTROLLER*

AUTHOR: ROGER TORRES SALAZAR**

KEY WORDS: Modbus, Field Bus, Net, Communication Processor, Microcontroller.

DESCRIPTION: The processor of communications Modbus RTU is a system based on microcontroller that executes Modbus protocol in RTU transmission mode with capacity of being part of a net fulfilling the requeriments of this protocol.

The programming of the microcontroller was made with the tool software Codewarrior IDE V3.0 in C embedded. A program was also made that allows software configuration of the processor.

This processor is connected to the bus with the standards RS-232 or RS-485 and it has 1024 discretes outputs and inputs and 64 outputs and input registers. Of these outputs and inputs can be visualized 8 discretes outputs, set 8 inputs and it allows to read and input register from an analog variable.

The operation of the processor was validated in a net with a PC like master that executes a program made in LabVIEW and a TRILOGI PLC like the other slave of this net.

---

* Final Grade Project.

** Faculty of Engineering Physical-mechanics. Electric, Electronic and Telecommunications Engineering Department . Director: Julio Augusto Gélvez Figueredo. Codirector: Jorge Eliécer Duque Pardo.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El estudio de las comunicaciones industriales en las carreras técnicas no resulta sencillo dada la diversidad de protocolos existentes. Esta diversidad que afecta y condiciona la toma de decisiones de una empresa a la hora de seleccionar un bus de campo a utilizar, provoca también dilema en el mundo de la enseñanza técnica a la hora de decidirse por un bus industrial a estudiar. En la última década se ha visto aparecer buses que estaban llamados a convertirse en el estándar de facto, pero las grandes empresas del sector no se han puesto de acuerdo en la elección de un estándar.

Entre los buses de campos más difundidos se pueden mencionar, Profibus, CAN, HART y Modbus entre otros. Se tiene particular interés en Modbus, por ser un protocolo de especificación abierta que permite, como es objetivo de este trabajo, programar un dispositivo que lo ejecute y logre comunicación en una red bajo los estándares de éste.

En este trabajo se describe el protocolo Modbus, y se ha hecho su implementación en el modo de transmisión RTU utilizando el microcontrolador MC68HC908GP32 de la empresa Motorola. De esta forma se ha conseguido un procesador genérico de comunicaciones para actuar en una red digital de datos bajo este estándar, que responde a las funciones soportadas por los controladores Modicon como se muestra en el documento *Modicon Modbus Protocol Reference Guide* **PI-MBUS-300 Rev. J**, Junio 1996.

Teniendo presente la reforma del plan de estudios, el grupo de investigación en automatización CEMOS propone el curso “Redes de Comunicaciones Industriales” como asignatura electiva en razón a que este es un tópico que debe ser conocido por los futuros ingenieros electricistas y electrónicos que pretendan desempeñarse en el campo de la automatización, el control y la instrumentación. Este proyecto

consolida la bases teóricas que se tienen de Modbus y servirá como elemento para la comprensión de las tramas de datos, sus funciones, servicios y configuración, constituyéndose en un elemento que servirá para el desarrollo de las prácticas del nuevo curso propuesto.

En el primer capítulo de este documento se encuentra una breve introducción a los temas relacionados a las comunicaciones digital de datos y a buses de campo. En el capítulo dos se describe el protocolo Modbus para comprender las características de este protocolo y así poder hacer su implementación. El tercer capítulo se dedica a describir la metodología utilizada en la elaboración de este trabajo, las características del microcontrolador utilizadas para la implementación de Modbus en el modo de transmisión RTU y muestra las rutinas para utilizar la memoria Flash como una EEPROM. El capítulo cuatro se encuentra en forma detallada la implementación del protocolo en el microcontrolador y las características del procesador de comunicaciones. Al final de este documento se muestra una aplicación y las conclusiones de este trabajo.

1. REDES DE COMUNICACIÓN DE DATOS

En los últimos años el desarrollo de los sistemas informáticos ha sido vertiginoso, de manera que podemos encontrar computadores en prácticamente todos los ámbitos de la vida cotidiana. Además los computadores han entrado en el ámbito de la industria para el control de plantas, monitorización de procesos productivos, gestión integrada de las diferentes etapas de fabricación , control de máquinas, herramientas, robots, manipuladores, etc. y pese a las altas velocidades, capacidad de proceso, bajo coste y versatilidad de los computadores actuales; por las características mismas del ambiente industrial, ha sido necesaria la incorporación de equipos robustos de altas prestaciones que realicen las tareas de control y monitorización automática como son los PLCs.

En muchas ocasiones estos computadores o equipos de control no realizan operaciones aisladas, sino que necesitan intercambiar datos con otros equipos para desempeñar su función. Las funciones básicas que hacen necesaria la comunicación de datos en el ámbito industrial son:

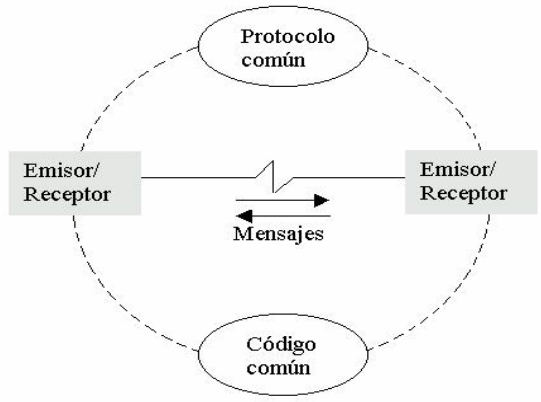
- Coordinar acciones de unidades automatizadas y controlar la transferencia de componentes, a través del intercambio de datos entre las diferentes unidades (autómatas programables o PLCs) que controlan el proceso productivo.
- Monitorizar y modificar estrategias de control desde el puesto de operación, que puede estar situado en la propia planta o en cualquier otro lugar mediante una conexión a través de redes de datos públicas o privadas.

Cuando se hable de comunicación de datos, conviene distinguir entre *datos*, que se define como el conjunto de diferentes estados que puede adoptar una variable e *información*, que es el resultado de procesar e interpretar esos datos.

La comunicación se puede entender como el proceso de intercambio de datos, de cuyo análisis posterior se obtiene la información. En la comunicación de datos intervienen varios elementos (figura 1):

- Equipo emisor/receptor: equipos que intervienen en la comunicación (computadores, PLCs, etc)
- Canal: recurso o medio físico capaz de propagar señales (cable eléctrico, aire, fibra óptica, ...)
- Mensajes: datos que se transfieren entre ambos equipos.

Figura 1 Elementos que intervienen en la comunicación



Fuente: Autor

Como consecuencia de la existencia de un flujo de datos bidireccional surge la necesidad emplear un protocolo, que es un conjunto de reglas que regulan el flujo de datos, y que en general establecen:

- Quién y cómo comienza el dialogo
- Quién puede transmitir en cada momento
- Cómo termina la comunicación.

Además de este protocolo de control de flujo se deben establecer mecanismos de control de detección de errores y recuperación de datos.

La comunicación entre dos equipos (transmisión punto a punto) se puede producir en tres modos diferentes, dependiendo de la dirección del flujo de datos: Simplex Semi-dúplex (*half-duplex*) o Dúplex (*full duplex*)

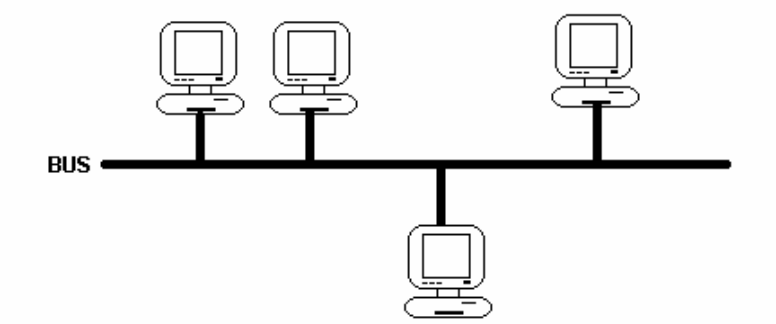
- Simplex, si la comunicación se realiza en un solo sentido, desde un equipo emisor a uno receptor.
- Semi-dúplex (*half-duplex)*, si la comunicación se realizan en ambos sentidos, pero no simultáneamente.
- Dúplex completo (*full duplex*), la comunicación se puede realizar en ambos sentidos simultáneamente.

1.1 Topología de Redes

La configuración de una red, comprende tres campos: físico, eléctrico y lógico. El nivel físico y eléctrico se entiende como la configuración del cableado entre máquinas o dispositivos de control o conmutación. Cuando se habla de la configuración lógica se debe pensar en cómo se tratan los datos dentro de la red, cómo se dirige de un sitio a otro y cómo se recoge en cada estación.

Existen tres topologías básicas para crear una red; Estrella, Bus y Anillo. En la topología de estrella los elementos de la red se encuentran conectados directamente mediante un enlace punto a punto al nodo central de la red, quien se encarga de gestionar las transmisiones de datos por toda la estrella. En la topología de bus, los elementos que constituyen la red se disponen linealmente conectados por medio de un cable; el bus, como se muestra en la figura 2. Las tramas de emitidas por un nodo (terminal o servidor) se propagan por todo el bus en ambas direcciones, alcanzado a todos los demás nodos. Cada nodo de la red se debe encargar de reconocer la información que recorre el bus, para así determinar cual de estas tramas está destinada a él. Es el tipo de instalación más sencillo y un fallo en un nodo no provoca la caída del sistema de la red. Esta es la topología que se utiliza en Modbus RTU y ASCII.

Figura 2 Topología de red en Bus.



Fuente: Autor

La topología en anillo los nodos de la red se disponen en un anillo cerrado conectados a él mediante enlaces punto a punto. Los datos describen una trayectoria circular en una única dirección y el nodo principal es quien gestiona conflictos entre nodos al evitar la colisión de tramas de información. En este tipo de topología, un fallo en un nodo afecta a toda la red. Por último cabe resaltar que las redes de comunicación de datos se pueden clasificar por su extensión en: redes de área local (LAN: *Local Area Network*), redes de área extensa (WAN: Wide Area Network) y redes de áreas metropolitanas (MAN: Metropolitan Area Network).

1.2 El Modelo de Referencia Osi de Iso.

En 1977 la Organización Internacional de Estandarización (ISO: *International Standarization Organization*) constituyó un comité para la creación de una arquitectura de comunicaciones que pudiera ser adoptada por los diferentes fabricantes de computadores y que permitiera interconectar equipos de diferentes naturaleza (computadores personales, estaciones de trabajo, equipos industriales, etc). De este modo surgió el modelo de referencia OSI (*Open Systems Interconnection*: Interconexión de sistemas abiertos), recogidos en la norma ISO7498. El objetivo de este modelo es permitir la comunicación entre aplicaciones

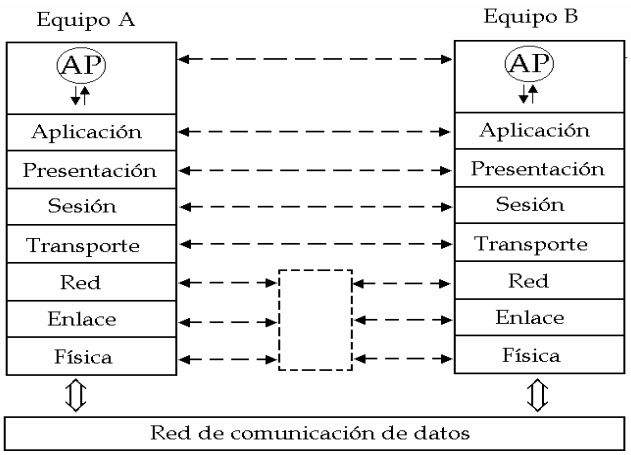
que se ejecuten en diferentes equipos, sin importar la diferencias entre estos componentes físicos.

El modelo de referencia OSI propone una arquitectura de capas (figura 3) que define el comportamiento del subsistema de comunicación de los equipos. Este modelo regula el comportamiento externo de cada capa, dando libertad al fabricante para su implementación interna.

Cada capa del modelo desempeña una función bien definida según el protocolo que se emplee para la comunicación e intercambia datos (virtualmente) con la capa par del sistema remoto.

Gracias a este mecanismo la implementación de cada capa se encuentra totalmente encapsulada (no es visible al exterior). De este modo cada fabricante realiza su implementación interna de las capas sin afectar a su funcionamiento externo, que será compatible con otras aplicaciones.

Figura 3 Modelo de referencia OSI



Fuente: Autor

El objetivo de cada una de las capas presentadas en la figura 3 se pueden encontrar en cualquier libro de redes de computadores.

1.3 Medios de Transmisión (Capa Física)

En la capa física está determinado los medios mecánicos, eléctricos, funcionales y de procedimiento requeridos para la transmisión de datos; esto involucra los tipos y formas de conectores que se deben usar para cada medio de transmisión, ya sea por cable, infrarrojo o microondas.

La transmisión en serie reduce la complejidad y parte del coste del sistema, pero obteniendo a cambio una menor eficacia: es necesario un intervalo de tiempo ocho veces mayor para transmitir ocho bits individuales que para transmitirlos simultáneamente (en paralelo).

1.3.1 Transmisión de Datos en Serie

Las comunicaciones serie se utilizan para enviar datos a través de largas distancias, ya que las comunicaciones en paralelo exigen demasiado cableado para ser operativas. Los datos serie recibidos desde un módem u otros dispositivos son convertidos a paralelo gracias a lo cual pueden ser manejados por el bus del equipo receptor.

Las comunicaciones serie se pueden dividir en síncronas o asíncronas. En una transmisión síncrona los datos son enviados en bloques, el transmisor y el receptor son sincronizados por uno o más caracteres especiales llamados caracteres de sincronismo. En este tipo de transmisión se asocia un pulso de reloj con cada bit transmitido,. requiriéndose de líneas de comunicación (usualmente cables), uno para los bits de datos y otro para los pulsos de reloj. En una transmisión asíncrona, un bit identifica el inicio de comienzo un carácter y 1 o 2 bits identifican su final, no es necesario caracteres de sincronismo. Los bits de datos son enviados al receptor después del bit de inicio. El bit de menos peso del carácter es transmitido primero.

Para la comunicación de datos en serie se han establecido diferentes normas que especifican las características técnicas de la conexión. Existen tres estándares que

son los más usuales en la conexión serial de equipos para conformar una red, estos son RS-232, RS-422 y RS-485[1].

1.4 Buses De Campo[2]

Un bus de campo es un sistema de transmisión de datos que simplifica enormemente la instalación y operación de máquinas y equipamientos industriales utilizados en procesos de producción. El objetivo de estos buses es sustituir las conexiones punto a punto entre los elementos de campo y el equipo de control a través del tradicional bucle de corriente de 4-20mA. Típicamente son redes digitales, bidireccionales y multipunto, montadas sobre un bus serie, que conectan dispositivos de campo como PLCs, transductores, actuadores y sensores. Cada dispositivo de campo incorpora cierta capacidad de proceso, que lo convierte en un dispositivo inteligente, manteniendo siempre un costo bajo. Cada uno de estos elementos será capaz de ejecutar funciones simples de control o diagnóstico, así como de comunicarse bidireccionalmente a través del bus.

El objetivo es reemplazar los sistemas de control centralizados por redes de control distribuido mediante el cual permita mejorar la calidad del producto, reducir los costos y mejorar la eficiencia. Para ello se basa en que los datos que envían y/o reciben los dispositivos de campo es digital, lo que resulta mucho más preciso que si se recurre a métodos analógicos.

La principal ventaja que ofrecen los buses de campo, y la que los hace más atractivos a los usuarios finales, es la reducción de costos**.** El ahorro proviene fundamentalmente de tres fuentes: ahorro en costo de instalación, ahorro en el costo de mantenimiento y ahorros derivados de la mejora del funcionamiento del sistema. Una de las principales características de los buses de campo es su

[1] Una descripción detallada de estos estándares se encuentra en el documento “Modbus. Monitoreo de la Red Empleando LabVIEW”, Sinle Marcela Carreño y Pedro Ardila. Cap. 1. Universidad Industrial de Santander, 2005.
[2] Para más información consultar el documento: “Redes de Comunicaciones Industriales” , Julio A. Gelvez. Cap. 3; Universidad Industrial de Santander, 2002.

significativa reducción en el cableado necesario para el control de una instalación. Cada componente sólo requiere un cable para la conexión de los diversos nodos. Se estima que puede ofrecer una reducción de 5 a 1 en los costos de cableado. En comparación con otros tipos de redes, dispone de herramientas de administración del bus que permiten la reducción del número de horas necesarias para la instalación y puesta en marcha.

Los buses de campo ofrecen mayor flexibilidad al usuario en el diseño del sistema. Algunos algoritmos y procedimientos de control que con sistemas de comunicación tradicionales debían incluirse en los propios algoritmos de control, radican ahora en los propios dispositivos de campo, simplificando el sistema de control y sus posibles ampliaciones.

También hay que tener en cuenta que las prestaciones del sistema mejoran con el uso de la tecnología de los buses de campo debido a la simplificación en la forma de obtener información de la planta desde los distintos sensores. Las mediciones de los distintos elementos de la red están disponibles para todos los demás dispositivos. La simplificación en la obtención de datos permitirá el diseño de sistemas de control más eficientes.

Con la tecnología de los buses de campo, se permite la comunicación bidireccional entre los dispositivos de campo y los sistemas de control, pero también entre los propios dispositivos de campo.

Tres componentes de un bus de campo que configuran su caracterización son:

- el medio físico que utiliza para la transmisión de la información.
- su arquitectura o forma de distribuir la información.
- y los protocolos lógicos que transportan, gestionan y salvaguardan la información en forma de mensajes.

1.5 Protocolos de Comunicación

Un protocolo es un conjunto de reglas y convenciones entre comunicantes. El objetivo del protocolo es establecer una conexión entre DTEs (Equipo Terminal de Datos), identificando el emisor y el receptor asegurando que todos los mensajes se transfieran correctamente controlando toda la transferencia de datos.

Además de los aspectos que definen un protocolo se destacan las siguientes características:

- Los modos de operación, la estructura de los mensajes, los tipos de órdenes y respuestas, constituyen las diferentes piezas constructivas de un protocolo.

- La fase de comunicación puede resumirse en el caso de un servicio de comunicación con confirmación (hay servicios sin confirmación).

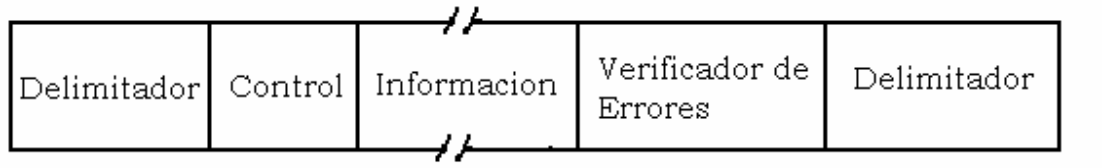
- Pueden definirse 4 funciones básicas o primitivas de un servicio de comunicación:

  I. *RESQUEST* : Un servicio es solicitado por ente usuario.
  II. *INDICATION* : Un ente es notificado de la ocurrencia de un evento.
  III. *RESPONSE* : Un ente responde a un evento.
  IV. *CONFIRM* : Un ente informa sobre un requerimiento anterior.

- Se puede distinguir “niveles” o “Capas” que reciben o solicitan servicios de niveles superiores y/o inferiores.

La estructura general de los mensajes que se intercambian entre dispositivos de una red es como se muestra en la figura 4.

Figura 4. Estructura de un mensaje.



Fuente: Autor

El propósito de cada campo del mensaje se muestra a continuación:

- Campo delimitador: Indica inicio y fin de un mensaje.
- Campo de control: Contiene información como el tipo de mensaje, número de secuencia, destino/origen, código de Request/confirmación.
- Información: Contiene la información a ser transportada. Es de longitud variable.
- Verificación de errores: Implementación de técnicas para detectar errores de transmisión. Hay tres métodos muy utilizados para detectar errores: control de redundancia vertical (VRC), Control de redundancia horizontal (HRC) y Control de redundancia cíclica (CRC). Este último utilizado por Modbus RTU.

Según la estructura de la información para su transmisión, los protocolos utilizan dos estructuras: orientada a caracteres (Ej: Modbus ASCII) y orientada a bit. (Ej: Modbus RTU)

2. EL PROTOCOLO MODBUS

En este capitulo se encuentra una descripción del protocolo Modbus. Esta descripción se hace basándose en el documento *Modicon Modbus Protocol Reference Guide* PI-MBUS-300 REV. J que se puede descargar de la página www.modbus.org, y pretende hacer mayor claridad y/o ampliar algunos conceptos que ayudarán al lector a comprender este protocolo.

2.1 Introducción

Modbus es una especificación desarrollada por Modicon para crear un Nivel de Aplicación estándar para Redes de Comunicación aplicada en ambientes industriales. Su especificación abierta permite la comunicación entre cualquier dispositivo que cumpla con el protocolo.

Modbus es una red digital de comunicaciones enmarcada en el concepto de Bus de Campo de Control y como tal, emplea sólo los niveles 1, 2 y 7 del modelo de referecia OSI (*Open System Interconection*); estos niveles corresponden al Físico, de Enlace y de Aplicación, respectivamente. Su topología es Maestro-Esclavo con una estructura de bus lineal en donde sólo existe un maestro, el cual controla el acceso al medio y monitoriza el funcionamiento de la red, y uno o más dispositivos programables actúan como esclavos, que responden y proceden según lo requerido por el maestro. La comunicación se da en forma serial asíncrona bajo los estándares RS-232 ó RS-485 para enlace semi-dúplex (*half-duplex*) y RS-422 para enlace dúplex (*full-duplex*), utiliza diferentes medios físicos como son los de soporte metálico (cables), radio frecuencia (RF), fibra óptica, o infrarrojo (IR) y cuya velocidad de transmisión está prevista en valores discretos para el rango de 75 a 19.200 baudios.

Así mismo, Modbus posee dos modos de comunicación serie conocidas como ASCII y RTU (o binaria) para el intercambio de mensajes entre los diferentes dispositivos que conforman la red. Estos mensajes son conocidos como tramas (*frames*) y están constituidas por un conjunto de caracteres que tienen una longitud en bits que depende del modo de transmisión que se utilice. Estas tramas contienen los datos necesaria para reconocer el origen y objetivo de cada mensaje puesto en el bus por alguno de los dispositivos y que luego le servirá a un dispositivo receptor para hacer la validación y posterior toma de decisiones.

La longitud de estas tramas es variable y está acotada a un máximo de 256 caracteres. Se pueden establecer comunicaciones en redes estándar Modbus utilizando cualquiera de estos dos modos de transmisión. Los usuarios seleccionan el modo de transmisión deseado, junto con los parámetros de comunicación del puerto serie (velocidad de transmisión, paridad, numero de bits de parada, etc.) durante la configuración de cada controlador. El modo y los parámetros del puerto serie tienen que ser los mismos para todos dispositivos para garantizar en primera instancia el buen funcionamiento de la red.

Modbus maneja básicamente dos tipos de datos: bits individuales y palabras de 16 bits. Los bits individuales corresponden a entradas o salidas discretas con estados ON/OFF y las palabras a registros de entrada o salida cuyos estados indican un valor análogo. En la tabla 1 se muestra estos tipos de datos.

Tabla 1. Datos en un dispositivo de una red Modbus

Sector	Formato	Tipo de acceso	Comentario
Salidas discretas (*Coils*)	bits individuales	lectura-escitura	modificables por un programa de aplicación
Entradas discretas (*Inputs*)	bits individuales	solo lectura	suministrados por un sistema de E/S
Registros de entrada (*Input Registers*)	Palabras de 16 bits	solo lectura	suministrado por un sistema de E/S
Registros de salida (*Holding Registers*)	Palabras de 16 bits	lectura-escritura	modificables por un programa de aplicación

Fuente: Autor

En esta tabla se muestran los cuatro tipos de datos que pueden estar presentes en los controladores que tienen a Modbus como protocolo de comunicación, el formato en el que se encuentran dentro del dispositivo, el tipo de acceso y como pueden ser modificados o suministrados. Estos datos pueden ser modificados sólo si son salidas del controlador programable. Es evidente que las entradas no tienen la posibilidad de ser cambiadas por un software de aplicación ya que éstas hacen referencia a estados externos de los controladores.

En el documento *Modicon Modbus Protocol Reference Guide* PI-MBUS-300 REV. J, se da una referencia a cada uno de los datos manejados por los controladores como se muestra en la tabla 2.

Tabla 2. Referencia de los datos en los controladores Modbus

Sector	**Información**	**Referencia**
Salidas discretas (*Coils*)	Salida digital	0X
Entradas discretas (*Inputs*)	Entrada digital	1X
Registros de entrada (*Input Registers*)	Valores analógicos	3X
Registros de salida (*Holding Registers*)	Entradas analógicas	4X

Fuente: Autor

Por ejemplo, si el maestro consulta a un esclavo el estado de veintidós entradas a partir de la entrada 197 estará haciendo referencia a las entradas **1**0197-**1**0218 del controlador esclavo. Se ha colocado el primer dígito en negrita para resaltar que éste indica la referencia de este tipo de dato.

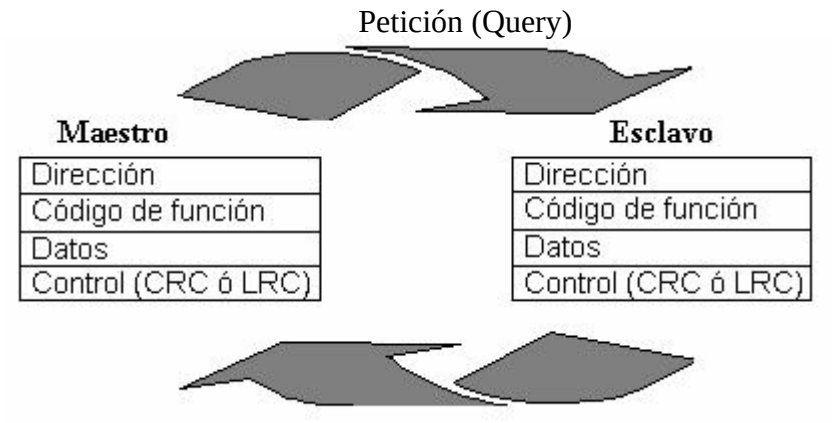
En su definición inicial Modbus era una especificación de tramas, mensajes y funciones utilizada para la comunicación entre los PLCs de la empresa Modicon.

Este protocolo ha sido acogido y actualizado a tal punto que se ha convertido en un estándar de facto para la automatización de la industria gracias a su particular estructura de mensajes, la cual opera con direcciones de memoria y no variables concretas, haciendolo adaptable a diferentes dispositivos; también a su especificación realmente abierta que permite comprender en toda su extensión la forma en que se lleva a cabo la transacción de mensajes y el flujo de información con lo cual ha sido posible la programación de equipos que lo ejecuten y puedan formar parte de una red Modbus. Como protocolo que es, describe el proceso para acceder a información de un dispositivo, cómo debe responder éste, y cómo se notifican las situaciones de error con el fin de garantizar un flujo correcto de datos entre los dispositivos que conforman la red.

2.2 Tramas de Modbus

El modo de transmisión es la estructura de las unidades de información contenidas en un mensaje. El protocolo Modbus utiliza dos modos de transmisión: ASCII (American Satandard Code for Information Interchange) y RTU (Remote Terminal Unit). En una red de dispositivos conectados mediante el protocolo Modbus no se pueden tener dispositivos utilizando diferentes modos de transmisión. Se describirá con detalles estos dos modos de transmisión en un capitulo posterior.

Figura 5. Ciclo de Pregunta/Respuesta



Los intercambios de mensajes cumplen un ciclo de Pregunta/Respuesta (*Query/Response*) en los dos modos de transmisión mencionados; esto se logra mediante tramas como se muestra en la figura 5.

Se observa en esta figura que la estructura de la trama enviada por el maestro al esclavo es similar al enviado por el esclavo al maestro. Estas tramas deben contener por lo menos los siguientes campos: dirección, código de función, datos y chequeo de errores. El contenido y propósito de estos campos dentro de cada trama se explican a continuación:

2.2.1 Campo de Dirección

Modbus es un protocolo multipunto, esto significa que el maestro puede comunicarse a múltiples esclavos utilizando la misma línea de comunicación, lo cual es conocido como topología de bus. Debido a esto, cada esclavo debe tener una identificación única e irrepetible dentro de la red con la que los dispositivos identificarán el destino y el origen de los mensajes que sean puestos en el bus. Duplicar esta identificación puede producir colisiones en el bus o conflictos en la red que conlleve a un flujo de datos no fiables con posteriores consecuencias negativas. La dirección de un dispositivo en la red debe estar, según el documento PI-MBUS-300 rev J., en el rango 1 a 63 con posibles valores que van desde 01_H hasta $3F_H$, esto con el fin de garantizar que los bits que conforman la trama lleguen con los niveles de tensión ó corriente establecidos en los estándares de comunicación serial utilizados.

Se conoce como difusión (*Broadcast*) cuando el maestro se dirige a todos los esclavos de la red al mismo tiempo. En este caso el campo de dirección debe contener como dato un cero (00_H) y todos los esclavos en la red aceptarán la petición enviada y ejecutarán la función indicada, si ésta soporta difusión. Cuando una pregunta se hace por medio de una difusión ninguno de los esclavos debe enviar un mensaje de respuesta.

Así mismo el campo de dirección es utilizado por los esclavos para colocar en él su propia dirección y permitirle al maestro identificar qué esclavo a puesto en el bus un mensaje de respuesta.

2.2.2 Campo de Códigos de Función

Cada función permite transmitir órdenes o datos a un esclavo. Existen dos tipos básicos de órdenes:

- Ordenes de lectura/escritura de datos en los registros o en la memoria del esclavo.
- Ordenes de control (RUN/STOP), carga y descarga de programas, verificación de contadores, etc.).

Tabla 3. Funciones básicas y códigos de operación MODBUS

CÓDIGO	Hex	DESCRIPCIÓN
0	00	Control de estaciones esclavas
1	01	Lectura de n bits de salida
2	02	Lectura de n bits de entrada
3	03	Lectura de n registros de entradas
4	04	Lectura de n registros de salidas
5	05	Escritura de un bit
6	06	Escritura de una palabra o registro
7	07	Lectura rápida de 8 bits
8	08	Diagnóstico
9	09	No utilizada
10	0A	No utilizada
11	0B	Solicitar contador de eventos de comunicaciones
12	0C	Solicitar diario de eventos de comunicaciones
13	0D	No utilizada
14	0E	No utilizada
15	0F	Escritura de n bits (*Coils*)
16	10	Escritura de n palabras (*Holding Registers*)

Fuente: Autor

La tabla 3 muestra la lista de funciones básicas disponibles en el protocolo MODBUS con sus correspondientes códigos de operación.

El campo de código de función de un trama cuando se utiliza el modo transmisión ASCII contiene dos caracteres de este estándar u ocho bits para el caso del modo de transmisión RTU. Los códigos válidos están en el rango decimal de 1 … 255. Cuando el maestro envía un mensaje de petición a un dispositivo esclavo, el campo de código de función le dice al esclavo qué tipo de acción debe ejecutar. Ejemplos de funciones son: leer o forzar los estados ON / OFF de un grupo de salidas discretas, leer o forzar el contenido de un grupo de registros, leer el estado de diagnóstico del esclavo, etc.

Cuando el esclavo responde usa el campo de código de función en el mensaje de respuesta para indicar si es una respuesta normal o si ha ocurrido alguna excepción (respuesta de excepción). Para una respuesta normal, el esclavo debe hacer eco del código de función recibida en la petición. Por otro lado, para una respuesta de excepción el esclavo proporciona un código que es equivalente al código de función original con su bit mas significativo puesto a 1 lógico.

Por ejemplo, un mensaje del maestro al esclavo para leer un grupo de registros de salida (*Holding Register*) tendría el siguiente código de función en el modo de transmisión RTU:

0000 0011 (Hexadecimal 03)

Si el dispositivo esclavo puede ejecutar la acción solicitada, el código de función en el mensaje de respuesta es igual al enviado en la petición. Si ocurre una excepción el contenido del campo de código de función será:

1000 0011 (Hexadecimal 83)

Además de la anterior modificación del código de función para una respuesta de excepción, el esclavo coloca un código en el campo de datos del mensaje de respuesta, el cual le informa al maestro que tipo de error se produjo o la razón de la excepción. Los códigos de excepción se describen en el numeral 6 de este capítulo.

El programa de aplicación del dispositivo maestro tiene la responsabilidad de manejar las respuestas de excepción cuando sean enviadas por los esclavos en los mensajes de respuesta. Los procesos típicos que adelanta un maestro después de recibir un código de excepción son: enviar reiteradas veces el mismo mensaje, enviar mensajes de diagnóstico al esclavo o notificar a los operadores.

2.2.3 Campo de Datos

El campo de datos se construye usando grupos de dos dígitos hexadecimales, en el rango de 00 a FF hexadecimal. Esto se puede hacer a partir de un par de caracteres ASCII, o a partir de un caracter RTU, de acuerdo con el modo de transmisión serie de la red.

El campo de datos de los mensajes enviados por el maestro a los dispositivos esclavos contiene la información adicional que el esclavo debe usar para tomar la acción definida por el código de función. Este campo puede incluir ítem como son direcciones iniciales de entradas o salidas discretas a leer o escribir, direcciones iniciales de registros de entradas o de salidas, el número de datos a leer, etc. Por ejemplo, si el maestro le pide al esclavo leer un grupo de registros de salida (código de función 03), el campo de datos especifica el registro de inicio y cuantos registros se deben leer a partir de éste. Si el maestro escribe un grupo de registros en el esclavo (código de función 10 hex.), en el campo de datos se debe especificar el número del registro desde donde se debe empezar, cuantos registros se van escribir, el número de bytes (contador de bytes) que corresponden a los datos que se transmiten en dicho mensaje, y los datos que deben ser escritos en los registros.

Si no ocurre error, el campo de datos en el mensaje de respuesta contiene los datos solicitados. De lo contrario, este campo contiene un código de excepción que la aplicación del maestro puede usar para determinar la siguiente acción a tomar.

El campo de datos puede ser inexistente (de longitud cero) en ciertos mensajes. Por ejemplo, en una solicitud de un dispositivo maestro a un esclavo para responder con su registro de evento de comunicaciones (código de función 0B hexadecimal), el esclavo no requiere ninguna información adicional. El código de función solo especifica la acción.

2.2.4 Campo de Chequeo de Errores

El campo de chequeo de errores es el último de la trama y permite al maestro y a los esclavos detectar errores de transmisión. Ocasionalmente, debido a ruido eléctrico o a interferencias de otra naturaleza, se puede producir alguna modificación en el mensaje mientras se está transmitiendo. El control de errores asegura que los dispositivos receptores no efectuarán acciones incorrectas debido a una modificación accidental del mensaje.

Para las redes Modbus estándar se usan dos tipos de métodos de chequeo de error y el contenido del campo de chequeo de error depende del método usado.

Cuando se usa el modo ASCII para la comunicación, el campo de chequeo de error contiene dos caracteres ASCII. Los caracteres de chequeo de error son el resultado de un cálculo de (LRC *Redundacy*) Chequeo de Redundancia Longitudinal que se lleva a cabo con los contenidos del mensaje, excluyendo los caracteres dos puntos de inicio y CR/LF de terminación. Los caracteres LRC se añaden al mensaje como el último campo que precede los caracteres CR/LF.

Cuando se usa el modo RTU para la transmisión, el campo de chequeo de error contiene un valor de 16 bits implementado como dos bytes de 8 bits. El valor de chequeo de error es el resultado de un cálculo de Chequeo de Redundancia Cíclica (CRC) aplicado al contenido del mensaje. El campo de CRC se añade al mensaje como el último campo en el mensaje. Cuando esto se hace, se añade primero el byte de orden bajo del campo, seguido por el byte de orden alto. El procedimiento para generar el CRC se encuentra detallado en el numeral 7 de este capítulo.

Los dispositivos esclavos de la red no envían ninguna respuesta cuando detectan un error de CRC o LRC en la trama recibida.

2.3 Modos de Transmisión

Se ha mencionado que Modbus utiliza dos modos de transmisión para el intercambio de mensajes. A continuación se hace una descripción completa de cada uno de estos modos de transmisión, pero antes de continuar cabe hacer una aclaración. Cuando se habla de caracteres en el documento PI-MBUS-300 rev J. que describe el protocolo Modbus, no se hace referencia al caracter conocido en los sistemas de computo programables en los cuales se define como un dato cuya longitud es fija y de ocho bits (un byte). Entonces, cuando se cite que un campo de la trama tiene una longitud de un caracter por ejemplo, se estará haciendo referencia a un caracter definido para el modo de transmisión ASCII ó RTU, cuya longitud en bits son diez y once respectivamente. Recuérdese también que los caracteres ASCII se han definido con una longitud de ocho bits y que se pueden representar con dos dígitos hexadecimal de 0..9, y A..F.

2.3.1 Modo de Transmisión ASCII

Teniendo presente la descripción de cada uno de los campos que debe contener una trama Modbus, en la figura 6 se muestra la trama correspondiente a un mensaje utilizando el modo de transmisión ASCII:

Figura 6. Trama en el modo de transmisión ASCII

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	LCR	FIN
1 Caracter **:**	2 Caracteres	2 Caracteres	n Caracteres	2 Caracteres	2 Caracteres

Fuente: Autor

En modo ASCII los mensajes comienzan con el caracter dos puntos (**:** , ASCII 3A hex.), y terminan con de CR/LF (retorno de carro/ avance de línea), (ASCII 0D y 0A hex. respectivamente) que delimitan en inicio y el final de cada trama. Todos los dispositivos en red monitorizan el bus continuamente para detectar caracteres ASCII válidos, y una vez detectado el caracter dos puntos sabrán que alguno de los dispositivos ha empezado la transmisión de un mensaje y deberán estar presto para analizar el siguiente campo que corresponde a la dirección para determinar el destino del mensaje. Si esta dirección corresponde con la de algún esclavo, éste deberá iniciar la captura de la trama y posteriormente ejecutar la acción solicitada.

2.3.2 Bits Necesarios por Caracter a Transmitir

Como se mencionó anteriormente los caracteres para el modo de transmisión ASCII tienen una longitud de diez bits. Estos diez bits se distribuyen como sigue:

- 1 bit de inicio
- 7 bits de datos, el bit menos significativo se envía primero
- 1 bit para paridad, si se usa bit de paridad
- 1 bit de parada si se usa paridad y dos bits si no se usa paridad

En la figura 7 se muestra en forma gráfica el orden en el que se envía cada caracter de la trama en el modo de transmisión ASCII.

Figura 7. Orden de bits modo de transmisión ASCII

Con chequeo de paridad

Start	1	2	3	4	5	6	7	Par	Stop

Sin chequeo de paridad

Start	1	2	3	4	5	6	7	Stop	Stop

Fuente: Autor

El primer bit indica el inicio de la transmisión de un nuevo caracter de la trama. Es útil en la transmisión serial asíncrona para advertir a los dispositivos receptores que se ha puesto un nuevo caracter en el bus. Se debe hacer la validación de este bit para descartar que se trate de ruido en el canal que pueda alertar a los dispositivos para la recepción de un nuevo caracter inexistente.

Seguido a esto están siete bits para la información contenida en el carácter enviado.

El hecho de que se envíen solo siete bits de información en un caracter en el modo de transmisión ASCII tiene algunas implicaciones. Por ejemplo, en la figura 6 se observa que se deben enviar dos caracteres para indicar la dirección de esclavo. Si un mensaje es enviado al esclavo 06 se deberá enviar en este campo el cero en el primer caracter y el seis en el segundo, que en el código ASCII son el 48 y 54 en decimal respectivamente.

Este último aspecto hace que el aprovechamiento del canal para el flujo de información entre los dispositivos en red sea menor cuando se utiliza este modo de transmisión, comparándolo con el modo de transmisión RTU. Cabe aclarar que la afirmación anterior se hace suponiendo que la velocidad de transferencia de datos es la misma para comparar la eficiencia de los dos modos de operación.

Una ventaja de importancia cuando se utiliza este modo de transmisión es que permite tiempos hasta de un segundo entre caracteres sin que el dispositivo receptor interprete que ha ocurrido un error en la comunicación. Se verá que en el modo RTU el tiempo que transcurre en la recepción de caracteres es de vital importancia para determinar el inicio y fin de una trama libre de errores.

2.3.3 Modo de Transmisión RTU

En la figura 8 se muestra la estructura de un mensaje enviado utilizando este modo de transmisión. En esta figura se observa que los mensajes comienzan con un

intervalo de silencio de al menos 3.5 veces el tiempo necesario para enviar un carácter lo cual es mostrado como T1-T2-T3-T4. Después de este silencio el primer campo transmitido es la dirección del esclavo, cuya longitud es de ocho bits. De igual forma se cuenta con ocho bits para enviar el código de función, múltiplos enteros de ocho bits para los datos si son necesarios y dieciséis para el CRC. La trama terminará con un silencio de al menos 3.5 veces el tiempo necesario para enviar un carácter.

Figura 8. Trama en el modo de transmisión RTU

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CRC	FIN
T1-T2-T3-T4	8 bits	8 bits	n*8 bits	16 bits	T1-T2-T3-T4

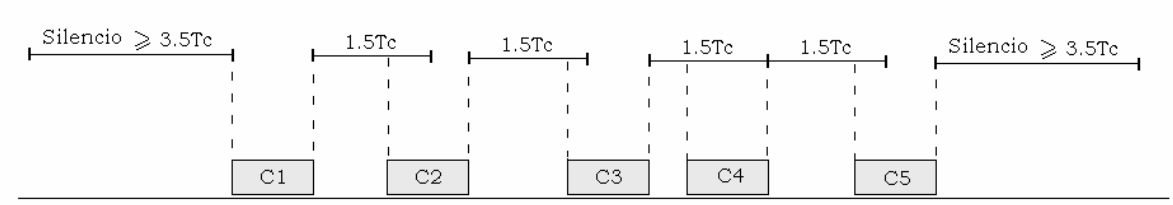
Además del tiempo que limita el inicio y el fin de una trama, en el modo de transmisión RTU se debe tener en cuenta el tiempo que transcurre entre la llegada de caracteres consecutivos. Este tiempo se ha definido para que sea máximo de 1.5 veces el tiempo necesario para enviar un carácter. Si entre el fin de un carácter y el comienzo de otro transcurre un tiempo mayor que 1.5Tc y menor que 3.5Tc se producirá una situación de error en la transmisión y el dispositivo receptor debe ignorar la trama. Cuando se produce este error los esclavos no deberán enviar ningún mensaje de respuesta.

En la figura 9 se muestra en forma gráfica diversas situaciones que se pueden presentar en la recepción o envío de mensaje entre dispositivos. En esta figura 1.5Tc representa 1.5 veces el tiempo necesario para enviar un carácter, 3.5Tc representa 3.5 veces el tiempo necesario para enviar un carácter y los bloques etiquetados como C1, C2, ... son los caracteres que conforman el mensaje puesto en el bus. La figura 9.a representa una situación normal en el intercambio de un mensaje; se observa que el inicio del primer carácter empieza cuando a transcurrido un silencio[3] de por lo menos 3.5Tc y que los restantes caracteres empiezan a llegar antes que ocurra 1.5Tc medido a partir del punto final del carácter anterior. También

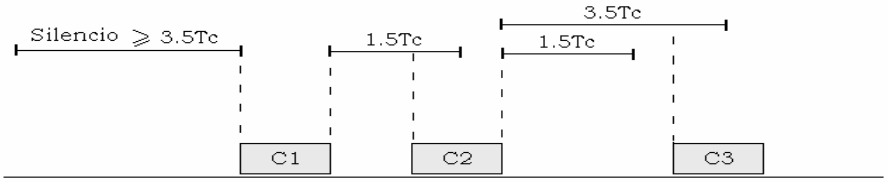
[3] Silencio: tiempo que transcurre sin que se detecte un caracter en el bus.

se observa que desde el final de C5 transcurre un tiempo mayor o igual a 3.5Tc, lo que indica que C5 en es último carácter del mensaje. Cualquier carácter que se reciba después de este último silencio será el primero de otro mensaje puesto en el bus por algún dispositivo de la red.

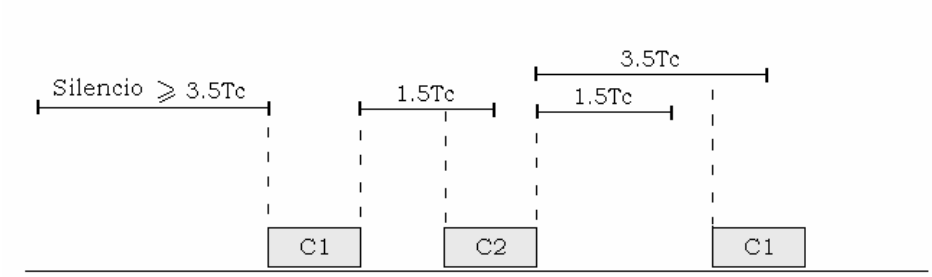
Figura 9. Situaciones posibles en el intercambio de mensajes entre dispositivos conectados en red en el modo de transmisión RTU. a) situación normal. b) situación de error. c) situación de error



a)



b)



c)

La figura 9.b representa una situación de error. Se observa que C1 y C2 se han recibido en tiempo permitidos, pero C3 se ha recibido después de que ha transcurrido 1.5Tc pero antes de que transcurra 3.5Tc. En este caso C3 no se puede entender como un carácter perteneciente al mensaje en recepción que va

después de C2 ni el primer carácter de otro mensaje. Si esto ocurre, el dispositivo receptor debe continuar monitorizando el bus para poder detectar un silencio de por lo menos 3.5Tc y recibir el siguiente carácter como el primero del siguiente mensaje. De igual forma la figura 9.c representa otra situación de error. Se observa que algún dispositivo a intentado enviar el primer carácter de un mensaje (C1) antes de que transcurra un silencio de por lo menos 3.5Tc medidos a partir del fin de C2. Como en el caso anterior C1 no se puede entender como un carácter consecutivo a C2 que pertenece al mensaje en recepción ni como el primer carácter de un nuevo mensaje. Otro posible caso es que C1 del nuevo mensaje llegue antes de que se cumpla 1.5Tc de haber arribado el ultimo carácter del mensaje anterior. Si este es el caso, el dispositivo receptor concatenará ese o esos caracteres al los caracteres del mensaje anterior y tendrá como es de esperarse un mensaje errado. Los dispositivos que reciban este mensaje detectarán el error cuando hagan el cálculo del CRC y hagan la comparación con el supuesto CRC que llega en el mensaje en los campos destinados para el chequeo de errores.

2.3.4 Bits Necesarios por Caracter a Transmitir

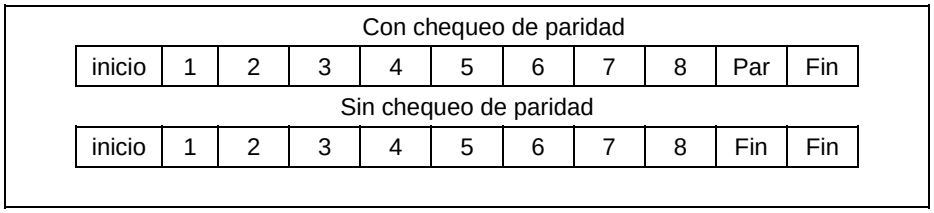
Los caracteres para el modo de transmisión RTU tienen una longitud de once bits. Estos once bits se distribuyen como sigue:

- 1 bit de inicio
- 8 bits de datos, el bit menos significativo se envía primero
- 1 bit para paridad, si se usa bit de paridad
- 1 bit de parada si se usa paridad y dos bits si no se usa paridad

En la figura 10 se muestra en forma gráfica el orden en el que se envía cada caracter de la trama en el modo de transmisión RTU.

Un ejemplo de petición se muestra en la tabla 4; en ésta se observa cómo se construye la consulta en el formato ASCII y RTU. El maestro le pide al esclavo número 06 que envíe tres registros a partir del registro 006B (107d).

Figura 10. Orden de bits modo de transmisión RTU



Es evidente en esta tabla la diferencia en bytes que existe en una misma trama enviada utilizando cada uno de los dos métodos de transmisión utilizadas en Modbus.

Tabla 4. Consulta del maestro con formatos ASCII y RTU.

Nombre del campo Consulta	Ejemplo (Hex)	ASCII Caracteres	RTU Campo de 8 Bit
Cabecera		: (colon)	
Dirección de esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Dirección de inicio (Hi)	00	0 0	0000 0000
Dirección de inicio (Lo)	6B	6 B	0110 1011
No. de Registros (Hi)	00	0 0	0000 0000
No. de Registros (Lo)	03	0 3	0000 0011
Chequeo de errores		LRC (2 caracteres)	CRC (16 Bits)
Fin de la trama		CR LF	
	Total Bytes :	17	8

De igual forma en la tabla 5 se observa un ejemplo de lo que puede ser una respuesta del esclavo 06 a la petición anterior para cada modo de transmisión.

En el campo de dirección se envía la dirección del esclavo que responde, en este caso 06; igualmente en el campo código de función se coloca el número de la

función que el esclavo ha ejecutado. En la respuesta para esta función se debe colocar en el campo contador de bytes, que indica cuantos bytes corresponden al estado de los registros solicitados por el maestro. En este caso, en el campo contador de bytes se encuentra un seis ya que se ha solicitado tres registros y cada registro consta de 16 bits (2 Bytes).

Tabla 5. Respuesta de Esclavo con formatos ASCII y RTU

Nombre del campo Respuesta	Ejemplo (Hex)	ASCII Caracteres	RTU Campo de 8 Bit
Cabecera		: (colon)	Ninguno
Dirección de esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Contador de bytes	06	0 6	0000 0110
Dato Hi	02	0 2	0000 0010
Dato Lo	2B	2 B	0010 1011
Dato Hi	00	0 0	0000 0000
Dato Lo	00	0 0	0000 0000
Dato Hi	00	0 0	0000 0000
Dato Lo	63	6 3	0110 0011
Chequeo de errores		LRC (2 chars)	CRC (16 Bits)
Fin de la trama		CR LF	Ninguno
	Total (Bytes):	23	11

Fuente: Autor

2.4 Descripción de las Funciones Modbus

Cada función permite transmitir órdenes o datos a un esclavo; existiendo dos tipos básicos de órdenes:

- Ordenes de lectura/escritura de datos en los registros o en la memoria del esclavo.
- Ordenes de control (RUN/STOP), carga y descarga de programas, verificación de contadores, etc.)

Antes de hacer la descripción de las funciones predefinidas en el protocolo Modbus cabe hacer las siguientes observaciones.

Todos las direcciones en los mensajes Modbus son referenciadas a cero. La primera unidad de cada tipo de dato es direccionada como la número cero. Por ejemplo, la bobina conocida como ‘bobina 1 ‘ en un controlador programable es direccionada como bobina 0000 en el campo de dirección de un mensaje Modbus.

La bobina 127 decimal es direccionada como bobina 007E hex (126 decimal).

El registro de salida 40001 es direccionado como registro 0000 en el campo de dirección de un mensaje Modbus. El campo código de función ya especifica una operación sobre un ’registro de salida’. Por lo tanto la referencia ‘4XXXX’ está implícita.

2.4.1 Leer Estados de Bobinas (01H).

Esta función permite al maestro indagar por el estado de las salidas discretas de un dispositivo remoto. Cuando se utiliza la palabra bobinas (*Coils*) en la guía PI_MBUS 300 Rev J se hace referencia a las salidas discretas de los controladores que actúan en una red Modbus. Estas salidas tienen un estado binario ON/OFF. Las bobinas tienen referencia 0X. La difusión para esta función no está soportada.
En la figura 11 se muestra un mensaje donde se le pide al esclavo 17 que envíe el estado de las bobinas de la 20 a la 56. En ésta se observa que el maestro indica el número de la bobina a partir de la cual se debe empezar a enviar los estados de las bobinas en los campos *Dirección de inicio. 3 y 4 son la parte alta y baja respectivamente* de este dato; esto da como resultado el número hexagesimal 0013 que hace referencia a la bobina 20. Se debe observar que 13 corresponde al número 19 en decimal.

En los campos *No. de Salidas (5 y 6)* se encuentra la parte alta y baja respectivamente de la cantidad de bobinas de las que se tiene interés en conocer el estado. En este ejemplo el maestro pide el estado de 37 bobinas (00 25 Hex) a partir de la bobina 20.

Figura 11. Pregunta Leer Estado de Bobinas (*Read Coil Status*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	01	00	13	00	25	CRC

1: Dir esclavo; **2**: Función
3,4: Dirección de inicio
5,6: Nº de salidas; **7**: CRC ó LRC

Fuente: Autor

En el mensaje de respuesta el esclavo envía el estado de cada bobina empaquetando en bloques de ocho bits (un byte) el estado de ocho bobinas consecutivas; representando una bobina por cada bit del campo de datos. El estado se indica como: 1 = ON; 0 = OFF.

Figura 12. Respuesta Leer Estado de Bobinas (*Read Coil Status*)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
11	01	05	CD	6B	B2	0E	1B	CRC

1: Dir esclavo; **2**: Función
3: Contador de Bytes
4,5,6,7,8 : Datos; **9**: CRC ó LRC

Fuente: Autor

El LSB del primer byte de datos contiene el estado de la bobina direccionada en la solicitud como *Dirección de inicio*. El estado de las siete bobinas consecutivas siguientes a la de inicio siguen en orden ascendente hasta el MSB de ese byte. Una vez se halla completado este primer byte se inicia el “llenado” del siguiente byte de datos empezando con la siguiente bobina (en este ejemplo la novena del conjunto de bobinas pedido) como el LSB de éste. El mismo procedimiento se sigue hasta conseguir empaquetar el estado de todas las bobinas requeridas por el maestro en la petición. Si la cantidad requeridas de bobinas no es un múltiplo de ocho, los bits

restantes en el byte de datos final se llenarán con ceros hacia el MSB. En la figura 12 se muestra una posible respuesta a la petición de la figura 11.

El campo *Contador de Bytes* especifica la cantidad de bytes de datos que el esclavo envía; en este ejemplo se tiene que el esclavo ha de enviar 5 bytes de datos, en los cuales está contenido el estado de todas las bobinas que han sido solicitadas por el maestro.

El estado de las bobinas 27 ... 20 se muestra como CD hex del byte, o binario 1100 1101. La bobina 27 es el MSB de este byte, y la bobina 20 es el LSB. De izquierda a derecha, el estado de las bobinas 27 ... 20 es ON-ON-OFF-OFF-ON-ON-OFF-ON.

Por convención, los bits dentro de un byte se muestran con el MSB a la izquierda, y el LSB a la derecha. El siguiente byte tiene las bobinas 35 ... 28, de izquierda a derecha.

En el último byte de datos, el estado de las bobinas 56 ... 52 se muestra como el valor 1B hex del byte, o binario 0001 1011. La bobina 56, que es la última bobina pedida, está en la posición del quinto bit, y la bobina 52 es el LSB de este byte. El estado de las bobinas 56...52 es: ON-ON-OFF-ON-ON. Los tres bits restantes como se puede observar se han llenando con ceros.

2.4.2 Leer Estados de las Entradas (02H) .

Esta función permite al maestro indagar por el estado ON/OFF de las entradas discretas de un dispositivo remoto. Las entradas al igual que las salidas discretas se direccionan a partir del cero. La referencia de este tipo de datos es la 1X. La difusión para esta función no está soportada.

En la figura 13 se muestra una petición donde se le pide al esclavo 17 que envíe el estado de las entradas de la 197 a la 218. En esta petición se observa que el maestro indica el número de la entrada a partir de la cual se debe empezar a enviar

los estados de cada entrada en los campos 3 y 4 *Dirección de inicio*. Esto da como resultado el número hexagesimal 00C4 que hace referencia a la entrada 10197. Se debe observar que C4 corresponde al número 196 en decimal.

Figura 13. Pregunta Leer Estado de Entradas (*Read Input Status*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	02	00	C4	00	16	--

1: Dir esclavo; **2**: Función
3,4: Dirección de inicio
5,6: Nº de salidas; **7**: CRC ó LRC

Fuente: Autor

En los campos *No. de salidas* se encuentra la cantidad de entradas de las que se tiene interés en conocer el estado. En este ejemplo el maestro pide el estado de 22 entradas (00 16 Hex) a partir de la entrada 00 C4 Hex.

Figura 14. Respuesta Leer Estado de Entradas (*Read Input Status*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	02	03	AC	DB	35	--

1: Dir esclavo; **2:** Función
3: Contador de Bytes
4,5,6: Datos; **7**: CRC ó LRC

Fuente: Autor

En la figura 14 se muestra lo que puede ser una respuesta a esta petición. La forma en que se ordenan los estados para ser enviados en el mensaje de respuesta es idéntico al utilizado para enviar el estado de las bobinas, en donde el LSB del primer Byte se envía el estado de la primera entrada solicitada y a partir del cual se empieza el llenado de los siguientes bytes.

El estado de las entradas 10204-10197 se representa como el byte de valor AC hex, o binario 1010 1100. La entrada 10204 es el MSB de este byte, y entrada 10197 es el LSB. De izquierda a derecha el estado de las entradas 10204 a 10197 es: ON-OFF-ON-OFF-ON-ON-OFF-OFF.

El estado de las entradas 10218 a 10213 es: ON-ON-OFF-ON-OFF-ON. Nótese que los dos bits más significativos están puestos a cero.

2.4.3 Leer de los Registros de Salida (03 H).

Esta función permite conocer el estado binario de los registro de salida (*Holding Registers*) de un esclavo. Los registros tienen 16 bits de longitud y se empiezan a direccionar a partir del registro cero. Los registros de salida tienen referencia 4X. La difusión para esta función no está soportada.

Figura 15. Pregunta Leer Estado de Registros de Salida (*Read Holding Registers*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	03	00	6B	00	03	--

1: Dir esclavo; **2**: Función
3,4: Dirección de inicio
5,6: Nº de registros; **7:** CRC ó LRC

Fuente: Autor

Como se observa en la figura 15 en el mensaje de petición se utilizan dos bytes para especificar el registro inicial en los campos *Dirección de inicio*. En este ejemplo el registro inicial es el 40108 el cual se da como 00 Hex en la parte alta junto con 6B en la parte baja. Por otra parte en los campos *No de registros* se especifica la cantidad de registros que se deben enviar en la respuesta. En este ejemplo se deben enviar tres registros a partir del 40108, es decir, los 40108 al 40110 del esclavo con dirección 17 (11 Hex).

Un mensaje de respuesta para esta función puede ser la que se muestra en la figura 16. En este ejemplo se observa que el contador de byte contiene un seis, indicando que se enviarán seis bytes de datos en donde están contenido el estado binario de los tres registros solicitados en la petición. Se debe observar en esta figura que cada registro se envía utilizando dos bytes, un byte para la parte alta y un byte para la parte baja.

Figura 16. Respuesta Leer Estado de Registros de Salida (*Read Holding Register*)

1	2	3	4	5	6	7	8	9	10
11	03	06	02	2B	00	00	00	64	--

1: Dir esclavo; **2**: Función; **3:** Contador de Bytes
4,5,6,7,8,9: Datos; **10:** CRC ó LRC

Fuente: Autor

El contenido del registro 40108 se representa con dos bytes de valores 02 2B hex. Los contenidos de los registros 40109-40110 son 00 00 y 00 64 hex. respectivamente.

2.4.4 Leer Registros de Entrada (04H).

Esta función permite leer el estado de los registros de entrada (*Input Registers*) de un esclavo. Los registros de entrada tienen una longitud de 16 bits y se empiezan a direccionar a partir del registro cero. Los registros de entrada tienen referencia 3X. La difusión para esta función no esta permitida.

El mensaje de consulta especifica el registro inicial y cantidad de registros a leer, como se muestra en la figura 17. El registro inicial se indica en los campos *Dirección de inicio* y el número de registros que se deben enviar a partir del registro inicial están indicados en los campos *No de registros.*

Figura 17. Pregunta Leer Estado de Registros Internos (*Read Input Registers*)

1	2	3	4	5	6	7
11	04	00	08	00	01	--

1: Dir esclavo; **2**: Función
3,4: Dirección de inicio
5,6: Nº de registros; **7:** CRC ó LRC

Fuente: Autor

Una respuesta a esta petición puede ser la que se muestra en la figura 18. El contenido binario de cada registro se empaqueta en el mensaje utilizando dos bytes. El primer byte del registro contiene los ocho bits de mayor orden y el segundo contienen los ocho bits de menor orden.

Figura 18. Respuesta Leer Estado de Registros Internos (*Read Input Registers*)

1	**2**	**3**	**4**	**5**	**6**
11	04	02	00	0A	--

1: Dir esclavo; **2**: Función
3: Contador de Bytes
4,5: Datos; **6:** CRC ó LRC

Fuente: Autor

En el campo *Contador de Bytes* se encuentra indicado la cantidad de bytes que corresponden a los campos de datos en la respuesta.

2.4.5 Forzar Bobinas Individuales (05H)

Mediante esta función el maestro puede indicarle a un esclavo que establezca una única bobina (referencia 0X) ya sea ON ó OFF. Cuando el maestro envía una consulta general (difusión), se debe forzar la misma bobina en todos los esclavos conectados en la red. En este caso ningún esclavo debe enviar un mensaje de respuesta.

Figura 19. Pregunta Forzar una única bobina (*Force single Coil*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	05	00	AC	FF	00	--

1: Dir esclavo; **2**: Función
3,4: Dir. Bobina a Forzar
5,6: Dato; **7:** CRC ó LRC

Fuente: Autor

Un mensaje de consulta puede ser el mostrado en la figura 19. En ésta se observa que el maestro utiliza dos campos con nombres *Dir. Bibina a Forzar* para especificar la parte alta y baja respectivamente de la referencia de la bobina a forzar. Las bobinas se direccionan comenzando en cero.

Los campos *Datos* son utilizados por el maestro para indicar al esclavo en que estado se poner a la bobina referenciada. Si en estos campos se coloca FF 00 Hex (parte alta y baja) el esclavo debe colocar la bobina en ON (uno lógico), si se coloca 00 00 Hex lo debe colocar en OFF (cero lógico). Cualquier otra combinación será ilegal y no afectará el estado de la bobina; en cuyo caso el esclavo debe además enviar una respuesta de excepción. Una respuesta normal es un eco de la consulta.

2.4.6 Preestablecer un Único Registro (06H).

Mediante está función el maestro puede poner en un determinado estado binario los bits que conforman un registro de salida (referencia 4X). Cuando esta petición se hace mediante una difusión todos los esclavos establecerán el mismo registro con el estado binario indicado por el maestro en la petición.En la figura 20 se muestra lo que puede ser una petición con este código de función.

Figura 20. Pregunta Forzar un Único Registro (*Force single Register*)

1	**2**	**3**	**4**	**5**	**6**	**7**
11	06	00	01	00	03	--

1: Dir esclavo; **2**: Función
3,4: Dir. Registro a Forzar
5,6: Dato; **7:** CRC ó LRC

Fuente: Autor

En los campos *Dir. Registro a Forzar* se indica la referencia del registro que se desea preestablecer y en los campos *Dato* se especifica el estado binario al que debe ser establecido este registro.

La respuesta normal es un eco de la consulta y se devuelve después que el registro ha sido modificado.

2.4.7 Lectura de Estados de Excepción (07H)

Los controladores Modicon tienen predefinidas unas posiciones de memorias donde se encuentra información almacenada como bits individuales donde cada bit indica el estado ON/OFF de Condiciones de Excepción. Algunas de estos bits son condiciones ya establecidas por el fabricante y otras son dejadas abiertas para que el usuario determine qué tipo de condición indica cada bit. Estas Condiciones de Excepción pueden ser: máquina ON/OFF, “cabezales en reposo”, “Seguridades cumplidas”, “Existen Condiciones de Error”, o como se dijo antes, un estado definido por el usuario.

Tabla 6. Bobinas de excepción en los controladores Modicon

Modelo del Controlador	Bobina	Asignación
M84, 184/384, 584, 984	1 - 8	Definido por el usuario
484	257	Estado de la batería
	258 - 264	Definido por el usuario
884	761	Estado de la batería
	762	Estado de protección de la memoria.
	763	Estado de la RIO
	764-768	Definido por el usuario

Fuente: Autor

Esta función provee un método simple para acceder a esta información, debido a que las referencias de bobinas de Excepción son conocidas (la función no necesita ninguna referencia de bobina). Las asignaciones de bobinas de Excepción predefinidas en los controladores Modicon se muestran en la tabla 6.

Figura 21. Pregunta Lectura Estados de Excepción

1	2	3
11	07	--

1: Dir esclavo; **2**: Función
3: CRC ó LRC

Fuente: Autor

La figura 21 muestra un ejemplo de lectura de Estado de Excepción en el esclavo 17.

La respuesta normal contiene el estado de las ocho bobinas de Condición de Excepción. Las bobinas se empaquetan en un byte de información, con un bit por bobina. El estado de la bobina de referencia más baja está contenido en el bit LSB del byte.

La figura 22 muestra un ejemplo de una respuesta para la consulta anterior:

Figura 22. Respuesta - Lectura Estado de Excepción

1	2	3	4
11	07	6D	--

1: Dir esclavo; **2**: Función
3: Datos; **4:** CRC ó LRC

Fuente: Autor

En este ejemplo, el valor de las bobinas es 6D hex (0110 1101 binario). De izquierda a derecha es: OFF-ON-ON-OFF-ON-ON-OFF-ON. El estado se presenta desde la bobina de dirección más alta a la mas baja.

Si el controlador es un 984, estos bits son el estado de la bobina 8 hasta la 1. Si el controlador es un 484, estos bits son el estado de la bobina 264 hasta la 257. En este ejemplo, la bobina 257 está ON, indica que las baterías del controlador están OK.

2.4.8 Solicitar Contador Eventos de Comunicaciones (0B H)

Devuelve una palabra de estado y un contador de eventos de comunicaciones del esclavo. Solicitando el contador actual antes y después de una serie de mensajes, el maestro puede determinar si los mensajes se han tratado normalmente. La Consulta General no está permitida.

El contador de eventos del controlador se incrementa cada vez que se termina un mensaje correctamente. No se incrementa por respuestas de excepción, consultas generales, o comandos relacionados con el contador de eventos.

El contador de eventos se puede poner a cero por medio de la función de Diagnósticos (código 08), con la subfunción de Opción de Reinicio de Comunicaciones (código 00 01) o Limpiar Contadores y Registros de Diagnóstico (código 00 OA).

Figura 23. Pregunta Solicitud del Contador de Eventos de Comunicaciones

1	**2**	**3**
11	0B	--

1: Dir esclavo; **2**: Función
3: CRC ó LRC

Fuente: Autor

La figura 23 muestra un ejemplo de una solicitud del Contador de Eventos de Comunicaciones en el esclavo 17:

La respuesta normal contiene una palabra de estado de dos byte, y otros dos bytes del contador de eventos. La palabra de estado será todo unos (FF FF hex) si se esta procesando un comando enviado previamente al esclavo (existirá una señal de ocupado). Si esta libre, la palabra de estado será todos ceros.

Figura 24. Respuesta Solicitud del Contador de Eventos de Comunicaciones

1	2	3	4	5	6	7
11	0B	FF	FF	01	08	--

1: Dir esclavo; **2**: Función
3,4: Estados ; **5,6:** Cont. de eventos
7: CRC ó LRC

Fuente: Autor

La figura 24 muestra un ejemplo de respuesta a la solicitud del contador de eventos de comunicaciones.

En este ejemplo, la palabra de estado es FF FF hex, indica que una función del programa esta aún en proceso en el esclavo. El contador de eventos muestra que el controlador ha registrado 264 (01 08 hex) eventos.

2.4.9 Solicitar Diario de Eventos de Comunicaciones (0CH)

Mediante esta función el esclavo envía al maestro una palabra de estado, cantidad de eventos, contador de mensajes, y un campo de bytes de eventos del esclavo. El contador de mensajes contiene la cantidad de mensajes procesados por el esclavo desde su ultimo reinicio, operación de limpieza de los contadores, o arranque. Este contador es idéntico al devuelto por la función de Diagnóstico (código 08), subfunción Devolver Contador de Mensajes de Bus (código 11, OB hex).

Figura 25. Pregunta Solicitud del Diario de Eventos de Comunicaciones

1	2	3
11	0C	--

1: Dir esclavo; **2**: Función
3: CRC ó LRC

Fuente: Autor

El campo de bytes de evento contiene 0-64 bytes, cada byte corresponde al estado de una operación Modbus, envío o recepción, hacia el esclavo. Los eventos los guarda el esclavo en este campo en orden cronológico. El byte 0 es el evento más reciente. Cada nuevo byte elimina el byte más antiguo del campo.

La figura 25 muestra un ejemplo de solicitud del diario de eventos de comunicaciones en el esclavo 17:

La respuesta normal contiene un campo de palabra de estado de dos bytes, un contador de eventos de dos bytes, un contador de mensajes de dos bytes, y un campo que contiene de 0-64 bytes de eventos. Un contador de un byte define la longitud total de la información en estos cuatro campos.

La figura 26 muestra un ejemplo de una respuesta para la consulta realizada en la figura 25:

En este ejemplo, la palabra de estado es 00 00 hex, indica que el esclavo no esta procesando ninguna función de programa. El contador de eventos informa que se han dado 264 (01 08 hex) eventos en el esclavo. El contador de mensajes informa que se han procesado 289 (01 21 hex) mensajes.

Figura 26. Respuesta Solicitud del diario de Eventos de Comunicaciones

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
11	0C	08	00	00	01	08	01	21	20	00	--

1: Dir esclavo; **2**: Función; **3:** Cont. de Bytes; **4,5:** Estado; **6.7:** Cont. de eventos
8,9: Cont. de mensajes; **10:** Evento 0; **11:** Evento 1; **12:** CRC ó LRC

Fuente: Autor

El evento de comunicaciones más reciente se muestra en el byte 0 de Eventos. Su contenido (20 hex) muestra que el esclavo ha entrado últimamente en Modo

Escucha.

El evento anterior se muestra en el byte 1 de Eventos. Su contenido (00 hex) muestra que el esclavo recibió un Reinicio de Comunicaciones.

La disposición de los bytes de evento de respuesta está descrito a continuación.

2.4.10 Contenido de los Bytes de Evento

El byte de evento devuelto por la función de Consulta del Diario de Eventos de Comunicación puede se de cuatro tipos. El tipo esta definido por el bit 7 (el bit de mayor orden) de cada byte. Esto se explica a continuación.

Evento de Recepción de un Esclavo Modbus

Este tipo de byte de evento es almacenado por el esclavo cuando recibe un mensaje de consulta. Lo almacena después de procesar el mensaje. Este evento se define poniendo a “1” el bit 7. Los otros bits podrán estar a “1” si la condición correspondiente es verdadera. La relación de bits se muestra en la tabla 7.

Tabla 7. Relación de bits del byte de eventos.

Bit	Contenido
0	No usado
1	Error de Comunicaciones
2	No usado
3	No usado
4	Sobre-escritura de carácter
5	Actualmente en modo de solo escucha
6	Recibida difusión
7	1

Fuente: Autor

Evento de Envío de un Esclavo Modbus

Este tipo de byte de evento es almacenado por el esclavo cuando finaliza el proceso de un mensaje de consulta. Se almacena si el esclavo a devuelto una respuesta normal o una de excepción, o no ha respondido. Este evento se define poniendo a “0” el bit 7, dejando el bit 6 a “1”. Los otros bits podrán estar a “1” si la condición correspondiente es verdadera. La relación de bits se muestra en la tabla 8.

Tabla 8. Relación de bits del byte de eventos de envío.

Bit	Contenido
0	Enviada excepción de lectura (Códigos de excepción 1-3)
1	Enviada excepción aborto del esclavo (Código de excepción 4)
2	Enviada excepción esclavo ocupado (Código de excepción 5-6)
3	Enviada excepción Slave-Program-NAK (Código de excepción 7)
4	Ha ocurrido un time out en escritura
5	Normalmente solo en modo escucha
6	1
7	0

Fuente: Autor

El Esclavo Entra en Modo de Solo Escucha

Este tipo de byte de evento lo almacena el esclavo cuando entra en “Modo de Solo Escucha”. El evento se define por el contenido 04 hex. La relación de bits es la siguiente:

Bit	7	6	5	4	3	2	1	0
Contenido	0	0	0	0	0	1	0	0

El Esclavo Inicia un Restablecimiento de Comunicaciones

Este tipo de byte de evento lo almacena el esclavo cuando se restablece su puerto de comunicaciones. El esclavo puede restablecerse por la función de Diagnostico

(código 08), mediante la subfunción Opción Restablecer Comunicaciones (código 00 01). La función de diagnóstico se describe en el numeral 5 de este capítulo.

Esta función deja también al esclavo en el modo “Continua ante un Error” o “Para ante un Error”. Si el esclavo queda en modo “Continua ante un error”, el byte de evento se añade al diario de eventos actual. Si el esclavo queda en modo “Para ante un Error”, el byte de evento se añade al diario y el resto de éste es puesto a cero.

El evento se define por un contenido de cero. La relación de bits es la siguiente:

Bit	7	6	5	4	3	2	1	0
Contenido	0	0	0	0	0	0	0	0

2.4.11 Forzar Múltiples Bobinas (0F H)

Fuerza varias bobinas (referencias 0X) consecutivas ya sea ON o OFF. Cuando es una Consulta General, la función fuerza las mismas referencias de bobina en todos los esclavos conectados.

El mensaje de consulta especifica las referencias de boina que se desean forzar. Las bobinas se direccionan comenzando en cero: La bobina 1 se direcciona como 0.

La petición de estado ON / OFF está especificada por el contenido del campo de información de la consulta. Un valor lógico “1” en una posición de bit en el campo solicita que la bobina correspondiente pase a ON. Un valor lógico “0” solicita que la bobina pase a OFF.

En la figura 27 se muestra un ejemplo de solicitud para forzar una serie de diez bobinas, comenzando en la bobina 20 (direccionada como 19,o 13 hex) en el

esclavo 17.

El contenido de la consulta tiene dos bytes: CD 01 hex (1100 1101 0000 0001 binario). Los bits binarios corresponden a las bobinas de la forma siguiente:

Bit	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
Bobina	27	26	25	24	23	22	21	20	-	-	-	-	-	-	29	28

El primer byte transmitido (CD hex) direcciona las bobinas 27-20, con el bit menos significativo se direcciona a la bobina más baja (20) en el conjunto. El byte siguiente (01 hex) direcciona las bobinas 29-28, con el bit menos significativo se direcciona a la bobina más baja (28) en el conjunto. Los bits no utilizados en el ultiman byte de información deberán ser llenados con ceros.

Figura 27. Pregunta Forzar Múltiples Bobinas

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	0F	00	13	00	0A	02	CD	01	--

1: Dir esclavo; **2**: Función; **3,4:** Dir. Inicio ; **5,6:** Nº de Bobinas
7: Cont. de Bytes ; **8,9:** Datos ; **10:** CRC ó LRC

Fuente: Autor

La respuesta normal devuelve la dirección del esclavo, el código de operación, la dirección de inicio y la cantidad de bobinas forzadas.

Figura 28. Respuesta Forzar Varias Bobinas

1	**2**	**3**	**4**	**5**	**6**	**7**
11	0F	00	13	00	0A	--

1: Dir esclavo; **2**: Función
3,4: Dir. de inicio; **5,6:** Nº de Bobinas
7: CRC ó LRC

Fuente: Autor

La figura 28 muestra un ejemplo de respuesta a la consulta que se realizó en la figura 27.

2.4.12 Forzar Múltiples Registros (10 H)

Esta función preajusta valores en una secuencia de registros de de salida (referencias 4x). Cuando hay difusión, la función escribe sobre las mismas referencias de registro en todos los esclavos conectados.

La función sustituirá el estado de protección de memoria de los controladores. Los valores preajustados serán válidos en los registros hasta que la siguiente lógica del controlador resuelva el contenido de los registros. Los valores de los registros permanecerán si no están programados en la lógica del controlador.

La figura 29, presenta un ejemplo de una solicitud para preajustar dos registros empezando en 40002 a 00 0A y 01 02 hex, en el dispositivo esclavo 17.

Figura 29. Pregunta Forzar Múltiples Registros (*Force Múltiple Registers*)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
11	10	00	01	00	02	04	00	0A	01	02	--

1: Dir esclavo; **2**: Función; **3,4:** Dir. de inicio ; **5,6:** Nº de registros
7: Cont. de Bytes ; **8,9,10,11:** Datos; **12:** CRC ó LRC

Fuente: Autor

Figura 30. Respuesta Forzar Múltiples Registros

1	**2**	**3**	**4**	**5**	**6**	**7**
11	10	00	01	00	02	--

1: Dir esclavo; **2**: Función; **3,4:** Dir. de inicio ; **5,6:** Nº de registros ; **7:** CRC ó LRC

Fuente: Autor

La respuesta normal devuelve la dirección del esclavo, el código de operación, la dirección de inicio y la cantidad de registros forzados. La figura 30 muestra lo que puede ser una respuesta a la consulta de la figura 29.

2.5 Función de Diagnóstico (08_H)

La función 08 en el protocolo Modbus tiene una gran importancia en lo que respecta a consultas para comprobar el sistema de comunicación entre maestro y esclavo, o para comprobar diversas condiciones internas de error dentro del esclavo.

Esta función usa en la consulta un código de subfunción de dos bytes para definir el tipo de prueba se va ha realizar. El esclavo devuelve, en una respuesta normal, tanto el código de operación como el de subfunción.

La mayoría de las consultas de diagnóstico utilizan un campo de información de dos byte para enviar información o control de diagnóstico al esclavo. Parte de los diagnósticos motivan que el esclavo devuelva información en el campo de información de una respuesta normal.

En general, la emisión de una función de diagnóstico a un dispositivo esclavo no afecta al programa de usuario que esta trabajando en el esclavo. La lógica de usuario, las entradas discretas y los registros, no se pueden acceder desde los diagnósticos. Ciertas funciones pueden restablecer opcionalmente contadores de error en el esclavo.

Un dispositivo esclavo puede, sin embargo, ser forzado a modo “Solo Escucha” en el cual puede recibir mensajes por el sistema de comunicaciones pero no los responden. Esto puede afectar al resultado de su programa de aplicación si este depende de intercambios de información con algún dispositivo esclavo. Generalmente, el modo se fuerza para retirar un dispositivo esclavo que funciona mal en el sistema de comunicaciones.

Un ejemplo de solicitud para el dispositivo esclavo 17 para Devolver Información de Consulta se puede ver en la figura 31. Esta consulta usa el código de subfunción cero (00 00 hex en campo de dos byte). La información devuelta se envía en campo de información de dos byte (A5 37 hex).

Figura 31. Pregunta Diagnóstico

1	2	3	4	5	6	7
11	08	00	00	A5	37	--

1: Dir esclavo; **2**: Función; **3,4:** Subfunción **5,6:** Datos ; **7:** CRC ó LRC

Fuente: Autor

La respuesta normal es un eco de la consulta.

Los campos de información en las respuestas a otras clases de consulta pueden contener contadores de error u otra información solicitada por el código de subfunción. Estos códigos se encuentran descritos a continuación.

2.5.1 Subfunciones de Diagnósticos

El objetivos de las subfunciones es permitirle a maestro hacer diferentes consultas de diagnóstico a un esclavo. En la tabla 9 se muestran los datos enviados por el maestro en la consulta y los datos enviados por el esclavo en el mensaje de respuesta.

Con la subfunción 0Bhex el maestro le pide a un esclavo que le devuelva el número de mensaje que ha detectado en el bus de comunicaciones desde el último reinicio, última limpieza de contadores, o último arranque.

Tabla 9. Datos de las subfunciones de diagnóstico.

Subfunción	Datos (Consulta)	Datos (Respuesta)
00 0B	00 00	Total mensajes detectados
00 0C	00 00	Total mensajes con error en CRC
00 0D	00 00	Total mensajes de excepción
00 0E	00 00	Total mensajes procesados
00 0F	00 00	Total mensajes no respondidos
00 10	00 00	Total NAK

Fuente: Autor

Con la subfunción 0Chex el maestro le indica al esclavo que devuelva en el mensaje de respuesta el número total de mensajes con errores en el CRC desde su último reinicio, última operación de limpieza de contadores o último arranque.

Con la subfunción 0Dhex el maestro le indica al esclavo que envíe el número de respuestas de excepción que ha enviado desde su último reinicio, última operación de limpieza de contadores o último arranque. Las respuestas de excepción están descritas en la siguiente sección de este capítulo.

Con la subfunción 0Ehex el maestro le indica al esclavo que envíe el número total de mensajes dirigidos al él o de consulta general, que haya procesado desde su ultimo reinicio, ultima operación de limpieza de contadores, o ultimo arranque.

Con la subfunción 0Fhex el maestro le indica al esclavo que envíe el número de mensajes dirigidos a él y no contestó (ni en respuesta normal, ni en respuesta de excepción), desde su ultimo reinicio, ultima operación de limpieza de contadores, o ultimo arranque.

Por último con la subfunción el maestro le indica al esclavo que devuelva la cantidad de mensajes que recibió y para los cuales envió la respuesta de excepción con código 07 “Negativa de reconocimiento” (NAK) desde su último reinicio, última operación de limpieza de contadores o último arranque.

2.6 Respuestas de Excepción

En el numeral 4 de este capítulo se describieron las funciones básicas de Modbus, y para cada una de estas se mostró un ejemplo de consulta y respuesta. En estas respuestas de ejemplo se supuso que no existían errores en la comunicación y que la información suministrada en el mensaje de consulta era suficiente para que el esclavo realizara la función solicitada. Sin embargo durante el ciclo consulta respuesta pueden darse diferentes eventos que obligan al esclavo a enviar un mensaje de excepción.

Los mensajes de excepción no se envían cuando existen errores de comunicación como son errores de paridad en el LRC o CRC. Si existe uno de estos errores el esclavo no enviará algún mensaje.

Un mensaje de excepción se presentará si se dan algunos de los siguientes eventos:

- En el mensaje de consulta se le ha solicitado al esclavo que ejecute una función que no soporta.
- Cuando se le solicita un bit o una palabra que no existe.
- Cuando el número de datos solicitado excede el máximo del esclavo.
- Cuando el esclavo tiene dificultades o detecta errores cuando intenta ejecutar una función.
- Cuando el esclavo está ocupado.
- Cuando el esclavo requiere de un tiempo mayor al *timeout*[4] para ejecutar la función solicitada.
- Cuando se detecta error en la ejecución de una función enviada en un programa.
- Cuando se detectan errores en una memoria extendida.

[4] Tiempo máximo de espera para que el esclavo responda a una consulta.

Cada unos de estos eventos tiene un código con el que el maestro sabrá el origen del mensaje de excepción. En la tabla 10 se muestran los códigos de error definidos en el protocolo Modbus.

Tabla 10. Códigos de excepción

Código	Nombre	Significado
01	Función ilegal	El código de función recibido en la consulta no es soportada por el esclavo.
02	Dirección de dato ilegal	La dirección de datos en la consulta no es una dirección válida para el esclavo.
03	Valor de datos ilegal	Una dirección enviada dentro del campo de datos en la consulta no es una dirección válida para el esclavo.
04	Falló el dispositivo esclavo	Un error desconocido ocurrió mientras el esclavo intentaba ejecutar la función solicitada en la consulta.
05	Reconocimiento	El esclavo ha aceptado la consulta y está procesándola, pero requiere de mayor tiempo para ejecutarla. Esta respuesta es enviada para evitar un error por *timeout* en el maestro.
06	Dispositivo esclavo ocupado	El esclavo está ejecutando un comando de programa que requiere de mucho tiempo.
07	Reconocimiento negativo	El esclavo no puede ejecutar la función recibida el la consulta. Este código es enviado por una consulta de programa infructuosa que usa los códigos de función 13 o 14 decimal. El maestro debe hacer una consulta de diagnóstico o información de error del esclavo.
08	Error de paridad en memoria	El esclavo intenta leer una memoria extendida, pero detecta un error de paridad en esta. El maestro puede enviar la consulta nuevamente, pero algún servicio puede necesario en el esclavo.

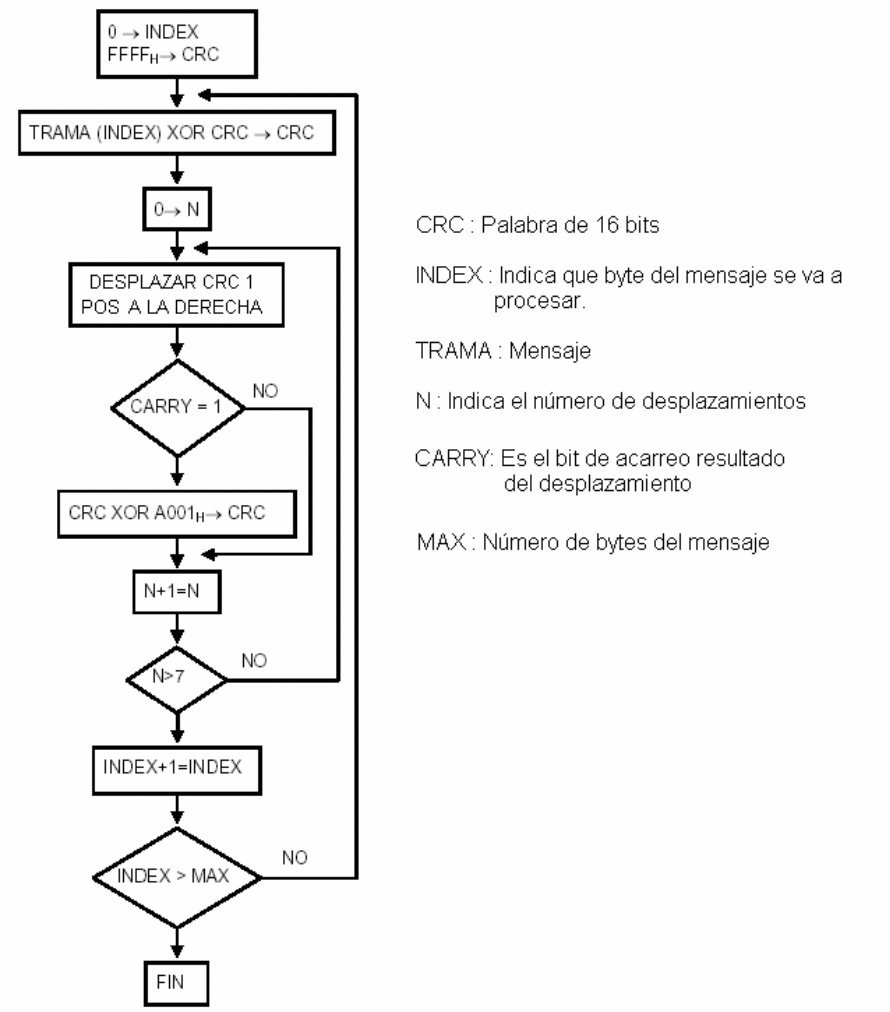
Fuente: Autor

2.7 Generación de CRC

El campo de Chequeo de Redundancia Cíclica (CRC) es de dos bytes, que contiene un valor binario de 16 bits. El valor del CRC es calculado por el dispositivo transmisor, que luego lo adjunta al mensaje. El dispositivo receptor recalcula un CRC con los datos del mensaje (excepto los de este campo) y compara el valor

calculado con el que le ha sido enviado. Si los dos valores no son iguales el equipo receptor interpreta que ha ocurrido un error de comunicación.

Figura 32. Diagrama de flujo para el cálculo del CRC



Fuente: Autor

En la figura 32 se muestra el diagrama de flujo utilizado para calcular el CRC de un mensaje.

A continuación se describen los pasos para calcular el CRC de un mensaje.

Paso 1 Cargar un registro de 16 bits con FFFF hex (todo 1's). Llamarlo registro CRC.

Paso 2 Hacer OR exclusiva del primer byte de ocho bits del mensaje con el byte de orden bajo del registro CRC de 16 bits, poniendo el resultado en el registro CRC.

Paso 3 Desplazar el registro CRC un bit a la derecha (hacia el LSB), llenando con cero el MSB. Examinar el bit de acarreo resultado del desplazamiento a la derecha.

Paso 4 Si el acarreo es 0, repetir el Paso 3 (otro desplazamiento). Si el acarreo es 1, hacer OR exclusiva del registro CRC con el valor polinomial A001 hex.

Paso 5 Repetir los Pasos 3 y 4 hasta que se hayan llevado a cabo ocho desplazamientos. Cuando esto se haya hecho, se habrá procesado un byte completo de ocho bits.

Paso 6 Repetir los Pasos 2 ... 5 para el siguiente byte de ocho bits del mensaje. Continuar haciendo esto hasta que todos los bytes del mensaje hayan sido procesados.

Tabla 11 Ejemplo del cálculo de CRC de un mensaje.

Acción	CRC (Bin)																Acarreo	CRC (Hex)
Cargar FF FF a CRC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		FF FF
FF FF XOR 9D	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0		FF 62
Desplazamiento 1	0	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	FF B1
Desplazamiento 2	0	0	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	3F D8
3F D8 XOR A001	1	0	0	1	1	1	1	1	1	1	0	1	1	0	0	1		9F D9
Desplazamiento 3	0	1	0	0	1	1	1	1	1	1	1	0	1	1	0	0	1	4F EC
4F EC XOR A001	1	1	1	0	1	1	1	1	1	1	1	0	1	1	0	1		EF ED
Desplazamiento 4	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	0	1	77 F6
77 F6 XOR A001	1	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1		D7 F7
Desplazamiento 5	0	1	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	6B FB
6B FB XOR A001	1	1	0	0	1	0	1	1	1	1	1	1	1	0	1	0		CB FA
Desplazamiento 6	0	1	1	0	0	1	0	1	1	1	1	1	1	1	0	1	0	65 FD
Desplazamiento 7	0	0	1	1	0	0	1	0	1	1	1	1	1	1	1	0	1	32 FE
32 FE XOR A001	1	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1		92 FF
Desplazamiento 8	0	1	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	45 7F
Intercambiar Bytes	0	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1		7F 45

Fuente: Autor

Finalmente el CRC se obtiene intercambiando los 8 bits más significativo con los 8 bits menos significativos y viceversa del registro CRC.

En la tabla 11 se muestra un ejemplo donde se calcula el CRC para un mensaje de un solo byte que tiene el valor 9D hex. En esta tabla la última acción (intercambiar bytes) sólo se debe hacer cuando se halla procesado todos los bytes del mensaje. Si en este ejemplo el mensaje tuviese un segundo byte, al comenzar el proceso con éste se deberá hacer OR exclusivo con 45 F7 (Desplazamiento 8) y continuar hasta completar el proceso.

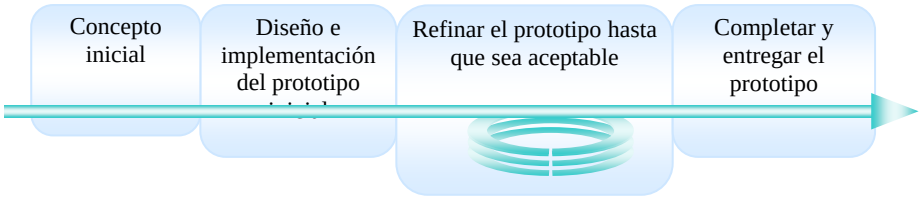
3. IMPLEMENTACIÓN DE MODBUS RTU

En este capítulo se presenta el desarrollo de la implementación de Modbus en el microcontrolador para obtener un procesador genérico de comunicaciones, las características de este último y la utilización de los diferentes módulos que lo componen.

3.1 Metodología de implementación

Para el proceso de implementación de Modbus RTU se utilizó la metodología de prototipado evolutivo, la cual se explica brevemente a continuación, al igual que su uso a lo largo del proyecto.

Figura 33. Prototipado evolutivo



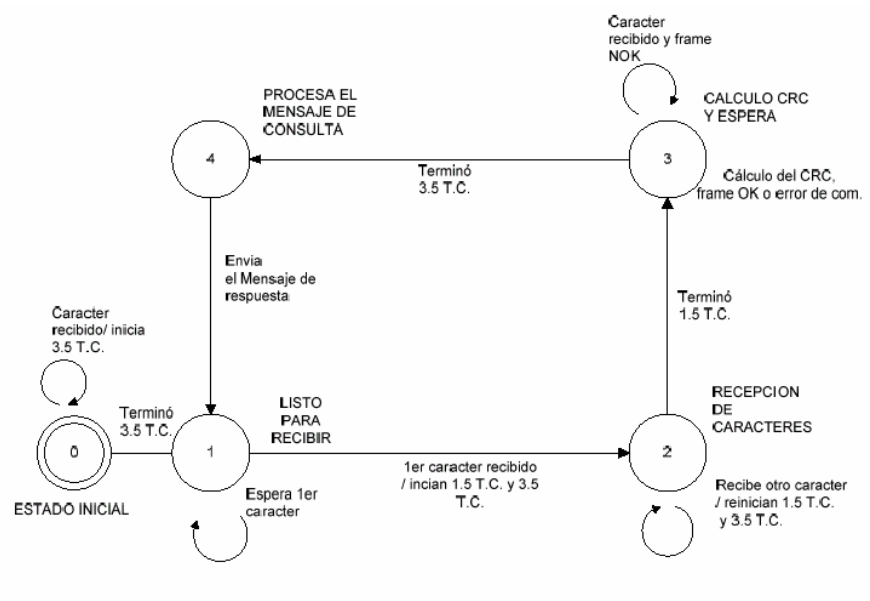
Fuente: Adamix 1.0 Herramienta Software para la Clasificación de Datos con Variables Cualitativas y Cuantitativas. Paola J. Velásquez; Victoria I. Fernandez

El prototipado evolutivo toma sus bases del prototipado simple, pero a diferencia de éste, que es de carácter exploratorio ó para identificar requisitos, el evolutivo posee mayores controles sobre la calidad y desarrolla primero las áreas de mayor riesgo del sistema, de tal forma que el prototipo pueda ser tomado como producto final una vez se llegue a su fin. Es decir, en este modelo se desarrolla el concepto del sistema a medida que avanza el proyecto. El prototipado evolutivo es un enfoque

donde se desarrolla primero las partes seleccionadas del sistema y luego a partir de éstas, se adelanta la ejecución del resto que conformarán la totalidad de dicho sistema. A diferencia de otros tipos de prototipado, en el evolutivo no se descarta el código del prototipo; lo transforma en el código entregado finalmente. El desarrollo de prototipos continúa hasta que se decide que el prototipo es lo suficientemente bueno y se puede entregar como producto final.

En este caso se construirán prototipos exploratorios de las principales subrutinas para la implementación del protocolo Modbus RTU, con el fin de identificar las partes críticas del desarrollo; y guiar dicha implementación en un proceso de reducción de riesgos, mediante los prototipos evolutivos que se vayan dando; y en cada uno de los cuales se irá incorporando funcionalidad, complejidad y optimización de las rutinas, dado que éstas por sus características demandan una cantidad considerable de recursos al microcontrolador en cuanto a procesamiento y memoria.

Figura 34 Diagrama de estados del modo de transmisión RTU.

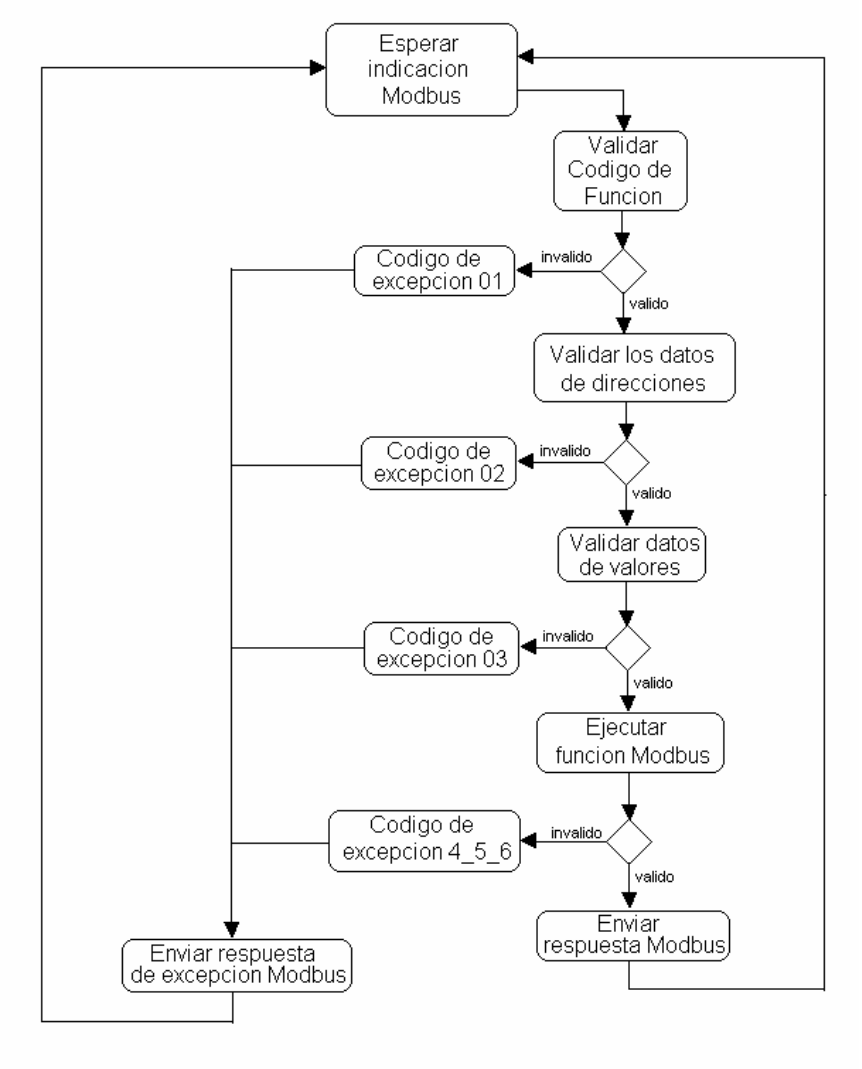


Fuente: Procesador de Comunicaciones Modbus. Implementación con Microcontrolador. Jorge E. Duque

En la figura 35 se muestra el diagrama de estado del modo de transmisión RTU.

Todas las rutinas que hacen posible la ejecución de las funciones Modbus RTU en el microcontrolador han sido desarrolladas teniendo presente el diagrama de estados que se muestra en la figura 35. Esta figura muestra la forma general de cómo se llevan a cabo las transacciones de mensajes Modbus.

Figura 35 Diagrama de estado de transacciones Modbus.



Fuente: MODBUS Application Protocol Specification www.modbus.org

3.1.1 Prototipo inicial.

En este prototipo se desarrollaron las funciones básicas para el manejo de la comunicación serial y de las interrupciones disponibles en el microcontrolador. Primero se configuró el módulo SCI (*Serial Communication Interface*) para que la comunicación se diera a 9600 baudios, paridad impar, 8 bits de datos, y un bit de parada. Se logra que la recepción de datos se dé cuando se detecta las interrupciones generadas por la llegada de los caracteres y no por la lectura permanente del receptor. Se implementan para estos propósitos las funciones SCIRxByte y SCITxByte para la recepción y la transmisión respectivamente, y se utiliza la interrupción 13 para la detección de datos en el receptor. Así mismo se utilizan las interrupciones 4 y 5 del *timer* para conocer si el tiempo de llegada de los caracteres corresponden a los establecidos por Modbus RTU. Ya en esta parte se tiene implementada la rutina “CRC16” para el cálculo del CRC.

En esta primera parte se lograron los siguientes objetivos:

1. Recepción de datos mediante interrupción.
2. Identificación de la dirección de esclavo.
3. Detección de errores de paridad.
4. Control de Redundancia Cíclica (CRC).
5. Control de tiempo en la llegada de los caracteres definidos en Modbus RTU.
6. Recepción de una trama completa libre de errores.

3.1.2 Prototipo intermedio

En este prototipo se incluyen las funciones Modbus que soportará el microcontrolador, las cuales se muestran en la tabla 12.

En esta parte de la implementación se incluyó una librería llamada ***Modbus.h***. La implementación de este archivo y su utilización se describe en el capítulo 4.

Tabla 12 Funciones soportadas por el procesador de comunicaciones.

Nombre	Código (hex)	Descripción
ReadCoilStatus	01	Lee hasta 1024 estados bobinas (salidas discretas)
ReadInputStatus	02	Lee hasta 1024 estados entradas discretas
ReadHoldingRegisters	03	Lee hasta 64 registros de salida
ReadInputRegisters	04	Lee hasta 64 registros de entrada
ForceSingleCoil	05	Fuerza a un estado ON/OFF una única bobina
PresetSingleRegister	06	Fuerza a un valor de 16 bits un único registro de salida
ReadExceptionStatus	07	Lee el contenido de los bits internos de condicion de excepción
DiagnosticSubfun	08	Proporciona al maestro datos para diagnóstico
FetchCommEventCounter	0B	Devuelve una palabra de estado y un contador de eventos
FetchCommEventLog	0C	Devuelve una palabra de estado, un contador de eventos, contador de mensajes.
ForceMultipleCoils	0F	Fuerza varias bobinas consecutivas
PresetMultipleReg	10	Establece varios registros de salida consecutivos

Fuente: Autor

3.1.3 Prototipo Final.

Este prototipo se constituirá como producto final. Además de las rutinas que hacen posible la ejecución de Modbus RTU en el microcontrolador, cuenta con rutinas que permiten mediante software la configuración del modo de operación dentro de la red. La configuración involucra dirección de esclavo, velocidad de transmisión y paridad; Esta configuración es grabada en memoria flash lo que evita tener que configurar cada vez que se enciende el microcontrolador y tiene las ventajas de reducir el número de pines utilizados para dicha configuración y ser muy intuitivo para el usuario.

3.2 Desarrollo del Programa

El programa se ha hecho en lenguaje C por ser el más utilizado para los sistemas que necesitan una programación de bajo nivel y porque los programas hecho en este lenguaje gozan de una gran portabilidad, lo que permite que un código pueda ser embebido con pocas adaptaciones en diferentes sistemas. Adicional a las razones anteriores para utilizar el lenguaje C en la programación del microcontrolador, se ha utilizado pensando en la tendencia mundial de utilizar a éste para la programación de estos equipos. Se ha utilizado Codewarrior IDE V 3.0 como herramienta de desarrollo.

3.3 El Microcontrolador “GP32”

El MC68HC908GP32 conocido familiarmente como GP32, es un microcontrolador de propósito general muy versátil y utilizado en el desarrollo de sistemas embebidos. Tiene cinco puertos que ofrecen en total 33 pines bidireccionales, siendo posible configurar cada uno como entrada o como salida. Las características de interés del GP32 para este trabajo se presentan a continuación:

CPU de 8 bits.
Memoria RAM de 512B.
Memoria flash para usuario de 32kB.
Dos módulos de temporización de 16 bits cada uno.
Doce canales, cada uno con convertidor análogo $\Leftrightarrow$ digital de 8 bits.
Un módulo SCI (*Serial Communication Interface*) para la comunicación serial.

3.3.1 Ubicación de los Datos en Memoria

El GP32 permite a los programadores reservar espacios de memoria para satisfacer las necesidades que se tengan. Por ejemplo en este trabajo se necesitó reservar memoria flash para almacenar las entradas y salidas del procesador de

comunicaciones, el diario de eventos, configuración de esclavo, etc. Así mismo, para poder grabar o borrar datos de la memoria flash se debe cargar en RAM las rutinas de borrado o grabado, ya que estas rutinas no pueden ser ejecutadas desde la misma flash.

En el archivo ***Modbus.prm*** se encuentra los sectores de memoria flash y RAM para la implementación de Modbus RTU.

En tabla 13 se muestra los sectores de memoria flash del microcontrolador utilizada para los datos Modbus, de configuración y propósito particular de este trabajo. Se puede ver en esta tabla el número total de entradas y salidas disponibles para el procesador de comunicaciones. Este número no corresponde al número total de salidas o entradas del microcontrolador cuando se quiera una aplicación particular, sin embargo se utilizó este número elevado para estudiar el comportamiento del microcontrolador si se quisiera utilizar un controlador con esa cantidad de entradas o salidas. Además con esto se puede demostrar que el programa funciona bien con gran cantidad de datos.

Tabla 13. Sectores de memoria flash de datos.

Sector de Memoria (Hex)	Nombre	Descripción	Total
8000 a 807F	Configuración	Datos de Configuración	
8080 a 80FF	Coils	Salidas discretas	1024
8100 a 817F	Inputs	Entradas discretas	1024
8180 a 81FF	Registers	Registros de entrada	64
8200 a 827F	HoldingR	Registros de salida	64
8280 a 82FF	HRAM	Almacenamiento temporal	
8300 a 837F	Diario	Datos del Diario de Eventos	
8380 a 83FF	Mensaje	Mensaje de Respuesta	

Fuente: Autor

En todos los sectores de memoria para las entradas y salidas tienen grabado en forma consecutiva el dato AA en hexadecimal (1010 1010). Este valor ha sido escogido por el autor para comprobar el funcionamiento del procesador de

comunicaciones en cuanto a la lectura de los datos del controlador. Tabla 13. Valores iniciales de las entradas y salidas

3.3.2 El Módulo SCI

El módulo SCI es el encargado de establecer comunicación serial asíncrona con otros equipos, y permite implementar el nivel físico del procesador de comunicaciones. Las características principales de la SCI del GP32 son las siguientes:

- Operación asíncrona *full-duplex*.
- Formato NRZ *non-return-to-zero*
- 32 posibles velocidades de transmisión.
- Longitud programable de 8 o 9bits.
- Habilitación por separado del receptor o transmisor
- Programación de polaridad de transmisión.
- 8 interrupciones:
- 7 registros de monitoreo y control
- Vectores para las rutinas de atención a interrupciones.

Características del SCI

El módulo SCI también llamado UART (*Universal Asynchronous Receiver Transmiter*), proporciona comunicaciones con dispositivos periféricos y otros sistemas en modo *full-duplex* asíncrono. El SCI usa un formato estándar de no retorno a cero (NRZ) e incluye un generador de velocidad de transmisión (*BaudRate*) interno que no requiere del *Timer* del sistema.

El módulo SCI incluye varios registros para controlar y verificar su funcionamiento y se distribuyen de la siguiente manera: un registro de datos, un registro de velocidad de transmisión, tres registros de control y dos registros de estado.

El registro de datos del SCI (SCDR) realmente tiene dos registros que se acceden mediante una dirección. Los dos registros son uno para el dato recibido y otro para el dato a transmitir. Ver figura 36.

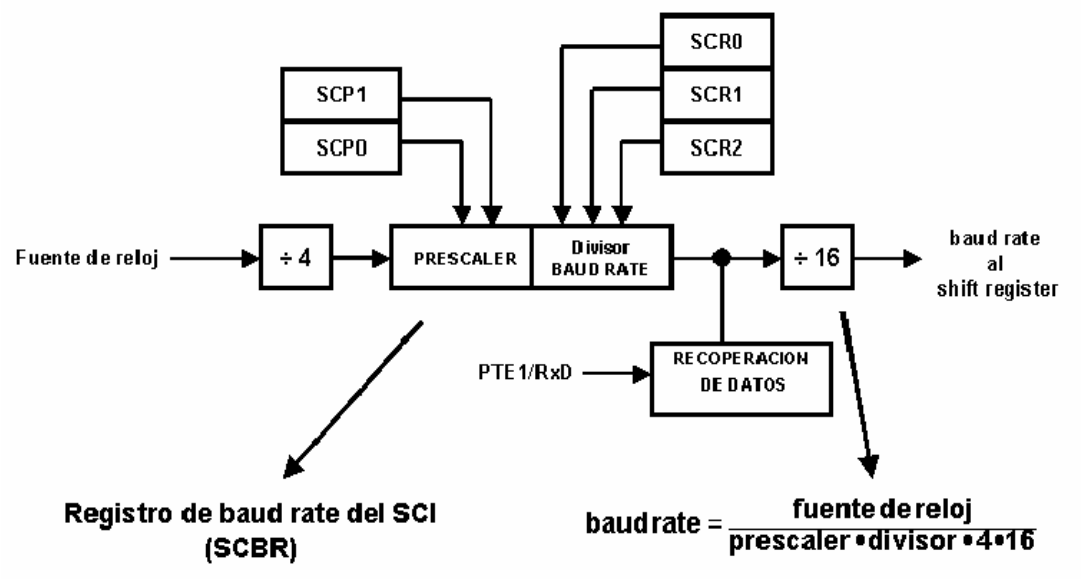
Figura 36. Registro de datos del SCI

SCDR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	R7	R6	R5	R4	R3	R2	R1	R0
Escribir:	T7	T6	T5	T4	T3	T2	T1	T0
Reset:	Un Reset no le afecta							

Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

Se accede a los registro individualmente utilizando comandos de lectura-escritura, leyendo en el registro de datos del SCI se obtiene el dato almacenado en el *buffer* de recepción. Escribiendo en este registro se coloca en el *buffer* los datos a transmitir.

Figura 37 Bloque del generador de *baud rate*



Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

La figura 37 muestra los bloques del generador del *baud rate* que incluye un *prescaler* y un divisor de *baud rate* que se puede programar utilizando el registro que lleva el mismo nombre.

La fuente de reloj del MC68HC908GP32 puede ser la frecuencia del cristal CGMXCLK dividido por cuatro o puede ser la frecuencia del bus interno en dispositivos con PLL.
El divisor por cuatro a la entrada es para bajar la velocidad a propósito. El divisor por 16 que va al registro de desplazamiento (*shift register*) permite al receptor hacer el muestreo de datos a 16 veces la velocidad del *baud rate*. Para calcular el *baud rate* del SCI, hay que dividir la fuente de reloj por el producto del *prescaler* , el divisor de *baud rate*, el divisor por 4 y el divisor por 16.

En el registro SCBR se puede programar el *prescaler* y el divisor de *baud rate*. Este registro se muestra en la figura 38.

Figura 38 El registro SCBR (SCI *Baud Register*)

SCBR	Bit 7	6	5	4	3	2	1	Bit 0
Leer: / Escribir:			SCP1	SCP0	Reser-vado	SCR2	SCR1	SCR0
Reset:	0	0	0	0	0	0	0	0

Fuente: “Technical Data MC68HC908GP32” Rev. 5 www.motorola.com

los bits SCP1 y SCP0 seleccionan uno de los cuatro valores del *prescaler* como se muestra en a continuación:

SCP1 – SCP0	PRESCALER
00	1
01	3
10	4
11	13

Los bits SCR2 a SCR0 seleccionan uno de los ocho divisores posible según se muestra a continuación:

SCR2 a SCR0	DIVISOR DE *BAUD RATE*
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Por ejemplo si se tiene un cristal de 4.9152 MHz, *prescaler* en 2 y el divisor de *baud rate* en 4 se tiene:

$$BaudRate = \frac{4.8152MHz}{2 \times 4 \times 4 \times 16} = 9600$$

o sea 9600 baudios.

Figura 39 Registros de control del SCI

SCC1

	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
Reset:	0	0	0	0	0	0	0	0

SCC2

	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
Reset:	0	0	0	0	0	0	0	0

SCC3

	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
Escribir:								
Reset:	U	U	0	0	0	0	0	0

U: NO afectado

Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

El SCI tiene tres registros de control llamados SCC1, SCC2 y SCC3 como se muestra en la figura 39.

Se puede encontrar una descripción detallada de la función de cada bit de estos registros en "MC68HC908GP32 *Technical Data"* de Motorola, sin embargo se menciona aquí la función de los bits utilizados en estos registros para la configuración del SCI en este trabajo.

En el SCC1

- ✓ **ENSCI** (*Enable SCI*) activa el SCI y el generador de *baud rate.*
- ✓ **M** selecciona la longitud del carácter en 8 ó 9 bits. Un cero en este bit selecciona un carácter de 8 bits, un 1 9 bits.
- ✓ **PEN** (*Parity Enable*) se utiliza para habilitar el chequeo de paridad.
- ✓ **PTY** sirve para seleccionar el tipo de paridad. 0 paridad par, 1 paridad impar.

En el SCC2

- ✓ **SCRIE** sirve para habilitar las interrupciones del receptor.
- ✓ **TE** se utiliza para habilitar el transmisor del SCI.
- ✓ **RE** se utiliza para habilitar el receptor del SCI.

En el SCC3

- ✓ **ORIE** se utiliza para generar una interrupción cuando ocurre un traslape en los caracteres recibidos.
- ✓ **NEIE** se utiliza para generar una interrupción cuando se detecta ruido en el medio físico.
- ✓ **PEIE** es utilizado para generar una petición de interrupción a la CPU cuando se detecta un error de paridad en el carácter recibido.

El registro de estado SCS1 es un registro de solo lectura que contiene ocho indicadores de estados para supervisar el funcionamiento del SCI. Este registro se muestra en la figura 40.

Figura 40 Registro del estado SCS1

SCS1	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	SCTE	TC	SCRF	IDLE	OR	NF	NE	PE
Escribir:								
Reset:	1	1	0	0	0	0	0	0

Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

se ha utilizado los bits OR (*Over Run*), NE (*Noise Error*) y PE (*Parity Error*), para chequear que los datos se reciban sin ninguno de estos errores.

3.3.3 El Temporizador

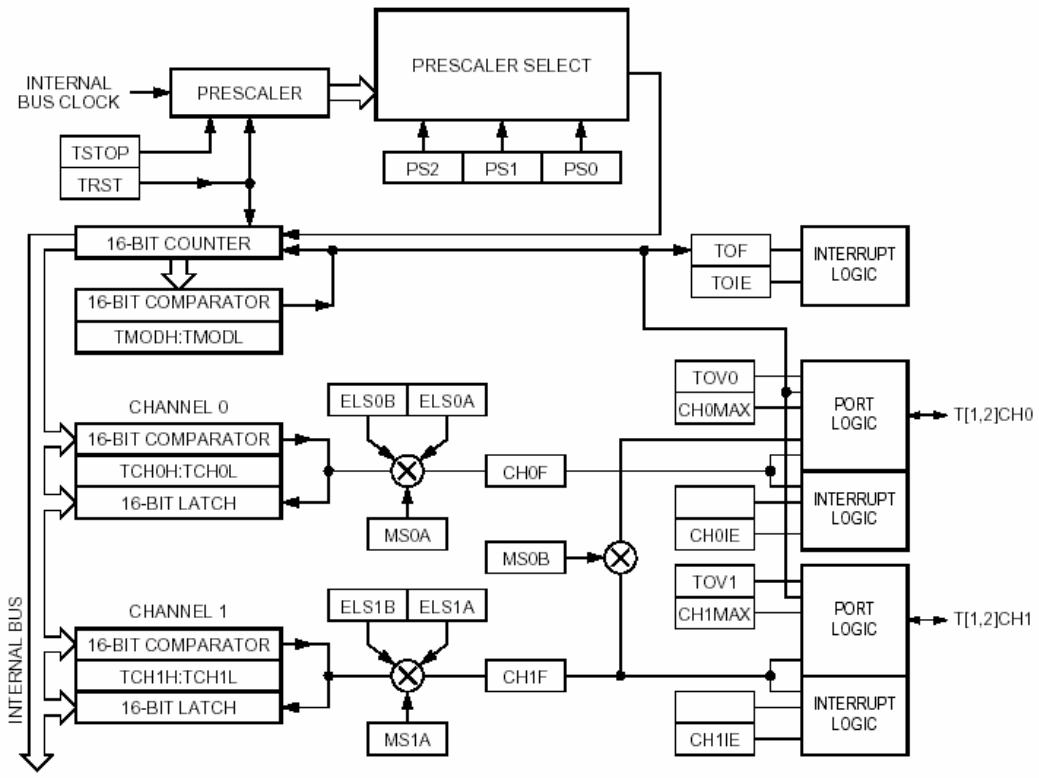
El componente central del TIM es un contador de 16 bits que opera como contador libre o con módulo. Este contador provee la referencia temporal al resto de componentes. Los registros de módulo del contador TMODH:TMODL, controlan el valor del módulo del contador. Por software es posible leer el valor del contador en cualquier momento. En la figura 41 se muestra el diagrama de bloque del TIM.

En esta figura se observa que el TIM tiene dos canales que proporciona una referencia temporal a los capturadores, comparadorres y a las funciones de PWM (Pulse-Width-Modulation). En el caso particular de este microcontrolador, existen dos módulos llamados TIM1 y TIM2;

Cada uno de los TIM incluye:

- Dos canales que pueden actuar como capturadores o comparadores:
- Generación de señal PWM.
- Entrada de reloj del TIM programable con siete posibles frecuencias.
- El contador puede operar como contador libre o con módulo.
- Contador del TIM con bits de stop y reset.

Figura 41 Diagrama de bloques del TIM



Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

Los pines de entrada / salida del TIM se nombran como:

	T[1,2]CH0	T[1,2]CH1
TIM1	PTD4 T1CH0	PTD5 T1CH1
TIM2	PTD6 T2CH0	PTD7 T2CH1

Este módulo fue utilizado para llevar la cuenta temporal entre caracteres entrantes del mensaje. Se configuró el TIM1 como contador con módulo y se utilizaron las interrupción de cada canal para conocer cuando se cumple 1.5 y 3.5 tiempo de carácter sin que se detecte la llegada de un dato; también fue de mucha utilidad la posibilidad que tiene este módulo de parar y restablecer por software los contadores.

Igualmente existen cinco registro que permiten la configuración y utilización de los temporizadores según se necesite. Estos registros son:

- Registro de estado y control (TSC).
- Registros del contador (TCNTH:TCNTL).
- Registros del módulo del contador (TMODH:TMODL).
- Registros de estado y control del canal (TSC0, TSC1).
- Registros del canal (TCH0H:TCH0L, TCH1H:TCH1L).

El canal cero del TIM1 uno fue utilizado para llevar la cuenta de 1.5 tiempo de carácter, el cual genera una interrupción ubicada en la posición cuatro del vector de interrupciones. Esta interrupción se genera si no se detecta ningún carácter cuando este canal halla llegado al final de la cuenta y establece en uno la bandera -_15Tover. De igual forma el canal 1 de este temporizador fue utilizado para llevar la cuenta de 3.5 tiempo de carácter el cual al final de la cuenta genera una interrupción que permite comenzar la ejecución de la función solicitada. Esta interrupción se encuentra ubicada en la posición cinco del vector de interrupciones.

En la tabla 14 se muestra los retardos necesarios de 1.5 y 3.5 tiempo de carácter para las diferentes ratas de baudios.

Tabla 14. Retardos para 1.5 y 3.5 tiempo de carácter

Baudios	retardo 1 [ms]	retardo **1,5** [ms]	retardo **3,5** [ms]
600	18,4	27,6	64,4
1200	9,2	13,8	32,2
4800	2,3	3,45	8,05
9600	1,15	1,73	4,03
14400	0,767	1,15	2,68
19200	0,575	0,863	2,01
38400	0,288	0,431	1,01

Fuente: Autor

En este trabajo se utilizó el temporizador como comparador se salida, que permite al TIM generar pulsos periódicos con polaridad, duración y frecuencia programables.

Cuando el contador alcanza el valor de los registros de un canal comparador, el TIM puede generar una petición de atención a interrupción.

La configuración y uso en detalle del TIM para la implementación del procesador de comunicaciones se presenta a continuación.

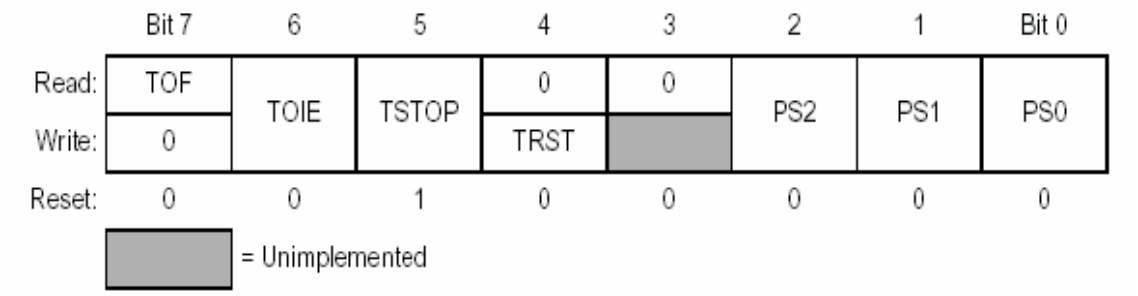
La referencia de tiempo se genera por un contador de 16 bits, un *prescaler* y un comparador. El contador se puede detener (*stop*) o restablecer (*reset*) bajo el control del programa.
El *prescaler* proporciona un valor de reloj derivado del bus interno, del contador de 16 bits y de un divisor programable. Usando los bits PS0, PS1 y PS2 en el registro TSC se puede programar el divisor a uno de los valores mostrados en la tabla 15:

Tabla 15 Configuración del *Prescaler*

PS2	PS1	PS0	Fuente de reloj del TIM
0	0	0	Bus interno ÷ 1
0	0	1	Bus interno ÷ 2
0	1	0	Bus interno ÷ 4
0	1	1	Bus interno ÷ 8
1	0	0	Bus interno ÷ 16
1	0	1	Bus interno ÷ 32
1	1	0	Bus interno ÷ 64
1	1	1	No disponible

Fuente: Autor

Figura 42 Registro de estado y control TSC



Fuente: "Technical Data MC68HC908GP32" Rev. 5 www.motorola.com

En la figura 42 se muestra el registro de estado y control (TSC) del TIM

Este registro tiene las siguientes utilidades:

- ✓ Habilita las interrupción de overflow.
- ✓ bandera de overflow.
- ✓ Para el contador.
- ✓ Resetea el contador.
- ✓ Preescala el reloj del contador.

TOF: Esta bandera de lectura/escritura se pone a uno cuando el contador regresa a 0000 después de alcanzar el valor del módulo programado en los registros del modulo del contador.

1 = El contador del TIM ha alcanzado su valor de modulo.

0 = El contador del TIM no ha alcanzado su valor de módulo.

TOIE: Este bit de lectura/escritura habilita la interrupción de *overflow* del TIM cuando el bit TOF se pone a uno.Reset pone a cero este bit.

1 = Habilita la interrupción de overflow del TIM.

0 = Deshabillita la interrupción de overflow del TIM.

TSTOP: Este bit de lectura/escritura para el contador del TIM. El contador se inicia de nuevo cuando TSTOP sepone a cero.

1 = Parar el contador del TIM.

0 = Contador del TIM activo.

TRST: Poniendo este bit de solo escritura a uno se resetea el contador y el preescalado del TIM. Poner a uno este registro no afecta a ningún otro registro. El contado comienza en 0000.

1 = Contador del TIM a cero.

0 = No tiene efecto.

Estas dos últimas banderas fueron de gran utilidad para verificar que los datos llegarán dentro de los tiempos establecidos por el protocolo.

Los registros de lectura/escritura del módulo del contador del TIM contienen el valor del módulo para el contador del TIM. Cuando el contador alcanza el valor del módulo, la bandera de overflow (TOF) se pone a uno, y el contador del TIM comienza su cuenta en 0000 al siguiente ciclo de reloj. Cada canal del temporizador tiene su propio módulo y es hay donde se almacenan los módulos de la cuenta de 1.5 y 3.5 tiempos de carácter. Se debe tener en cuenta que para las diferentes velocidades de transmisión estos módulos son diferentes.

3.3.4 Interrupciones Utilizadas del Microcontrolador

Se utilizaron tres interrupciones del microcontrolador para el correcto funcionamiento del procesador de comunicaciones. En la figura 43 se muestra todas las posibles fuentes de interrupción del GP32 y se han resaltado las que se utilizaron en este trabajo.

Cada una de estas interrupciones tiene una rutina de atención que se encargan de que la trama recibida sea correcta.

3.4 Ejemplo del Procesador de Comunicaciones

Se puede probar el funcionamiento del procesador de comunicaciones con el programa de distribución gratuita *Windmill ComDebug*[5]. En el campo *Prompt* se envía el mensaje de consulta hacia el procesador y en el campo *Reply* se visualiza el mensaje de respuesta.

[5] Este programa permite comunicar el PC con dispositivos que ejecutan Modbus. Se puede descargar el programa del sitio http://www.windmill.co.uk/serial.html

Figura 43 Fuentes de interrupción del MC68HC908GP32

Source	Flag	Mask(1)	INT Register Flag	Priority(2)	Vector Address
Reset	None	None	None	0	$FFFE–$FFFF
SWI instruction	None	None	None	0	$FFFC–$FFFD
$\overline{\text{IRQ}}$ pin	IRQF	IMASK1	IF1	1	$FFFA–$FFFB
CGM (PLL)	PLLF	PLLIE	IF2	2	$FFF8–$FFF9
TIM1 channel 0	CH0F	CH0IE	IF3	3	$FFF6–$FFF7
TIM1 channel 1	CH1F	CH1IE	IF4	4	$FFF4–$FFF5
TIM1 overflow	TOF	TOIE	IF5	5	$FFF2–$FFF3
TIM2 channel 0	CH0F	CH0IE	IF6	6	$FFF0–$FFF1
TIM2 channel 1	CH1F	CH1IE	IF7	7	$FFEE–$FFEF
TIM2 overflow	TOF	TOIE	IF8	8	$FFEC–$FFED
SPI receiver full	SPRF	SPRIE	IF9	9	$FFEA–$FFEB
SPI overflow	OVRF	ERRIE			
SPI mode fault	MODF	ERRIE			
SPI transmitter empty	SPTE	SPTIE	IF10	10	$FFE8–$FFE9
SCI receiver overrun	OR	ORIE	IF11	11	$FFE6–$FFE7
SCI noise fag	NF	NEIE			
SCI framing error	FE	FEIE			
SCI parity error	PE	PEIE			
SCI receiver full	SCRF	SCRIE	IF12	12	$FFE4–$FFE5
SCI input idle	IDLE	ILIE			
SCI transmitter empty	SCTE	SCTIE	IF13	13	$FFE2–$FFE3
SCI transmission complete	TC	TCIE			
Keyboard pin	KEYF	IMASKK	IF14	14	$FFE0–$FFE1
ADC conversion complete	COCO	AIEN	IF15	15	$FFDE–$FFDF
Timebase	TBIF	TBIE	IF16	16	$FFDC–$FFDD

Fuente: “Technical Data MC68HC908GP32” www.motorola.com

Para comprobar la correcta respuesta del procesador de comunicaciones se ha grabado ciertos datos en el sector de memoria que corresponde a las salidas discretas para que coincidan con la respuesta del ejemplo mostrado en el

documento PI-MB-300 Rev-J. Para este ejemplo el mensaje de consulta es el que se muestra en la figura 44.

Figura 44. Pregunta Leer Estado de Bobinas (*Read Coil Status*)

1	***2***	***3***	***4***	***5***	***6***	***7***
01	01	00	13	00	25	CRC

1: Dir. de esclavo ; **2**: Cód. de función; **3**: Dir. de inicio (H)
4: Dir. de inicio (L) ; **5**: Nº de puntos (H); **6**:Nº de puntos (L)
7: Control de errores

Fuente: Autor

Figura 45. Respuesta Leer Estado de Bobinas (*Read Coil Status*)

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
01	01	05	CD	6B	B2	0E	1B	CRC

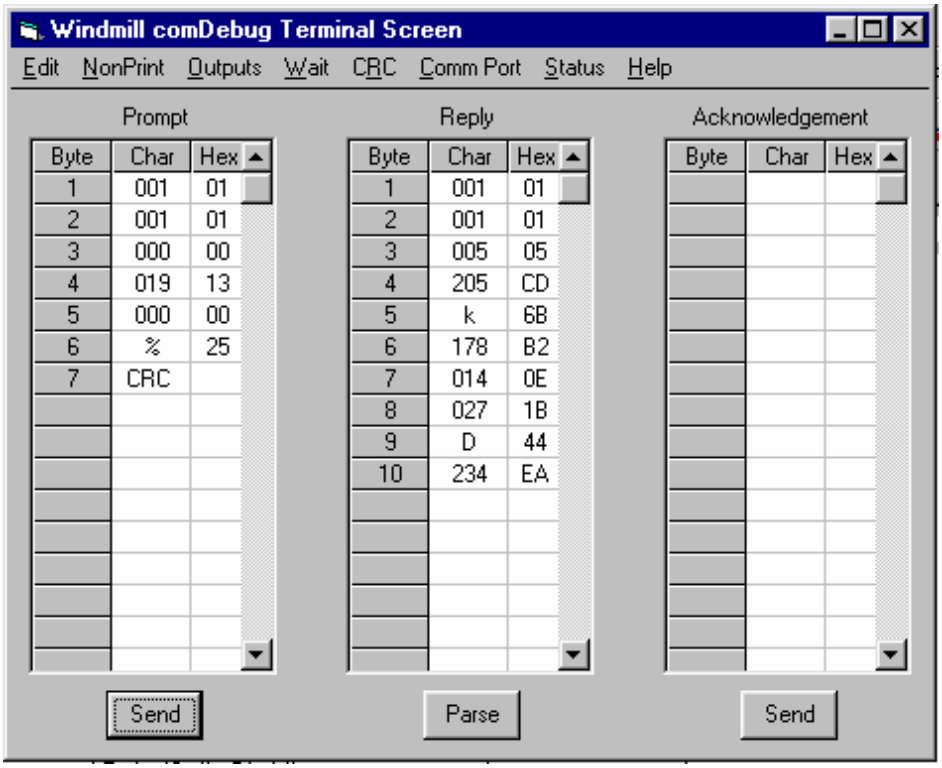
1: Dir. esclavo; **2**: Cód. función; **3**: Contador de bytes;
4,5,6,7 y **8**: Datos; **9**: Control de errores

Fuente: Autor

En la figura 45 se muestra el mensaje de respuesta para el ejemplo de consulta de la figura 44.

En la figura 46 se muestra la pregunta hecha al microcontrolador y el mensaje de respuesta de éste. Se puede corroborar que la respuesta coincide con la del ejemplo.

Figura 46. Ejemplo de Pregunta-Respuesta de la función 01

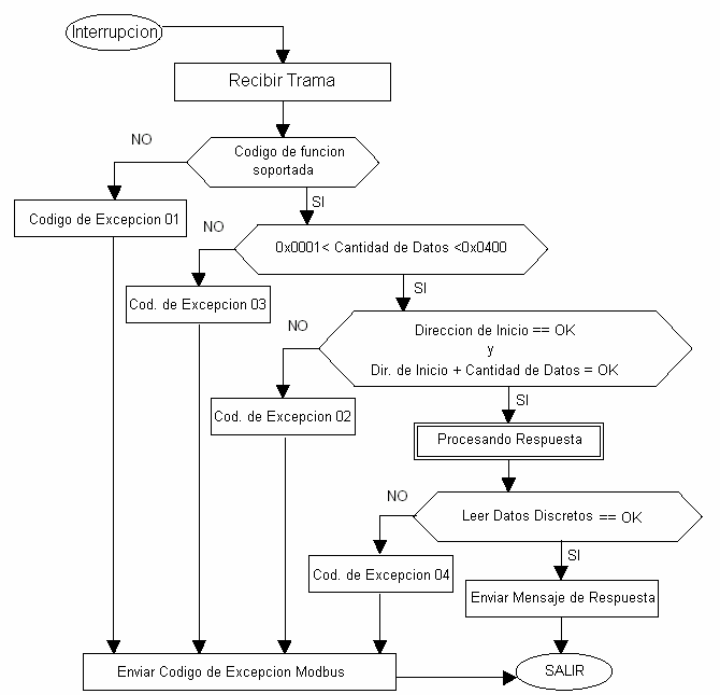


Fuente: Autor

3.5 Implementación de las Funciones

En la figura 47 se muestra el diagrama de flujo de la implementación de las funciones con código 01 y 02, *Read Coils Status* y *Read Input Status* respectivamente. Este diagrama de flujo sirve para la implementación de estas dos funciones ya que se trata de la misma acción (leer n estados discretos consecutivos a partir de una dirección). Si la cantidad de datos discretos está en el rango permitido, se procede verificar que la dirección de inicio mas la cantidad de datos no se sale del rango permitido. Si no ocurre error en la lectura de los datos se procede a enviar el mensaje de respuesta.

Figura 47. Diagrama de flujo de las funciones 01 y 02

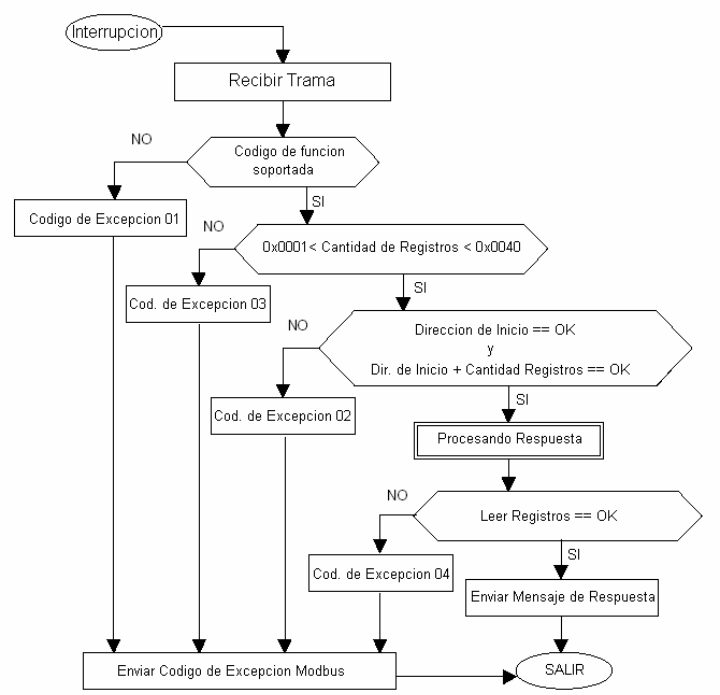


Fuente: Autor

La función 03 (*Read Holding Register*) y la función 04 (*Read Input Register*) se implementaron según el diagrama de flujo mostrado en la figura 48.

se debe comprobar que la dirección del primer registro a leer es un número de registro válido. Si el número del registro solicitado mas la cantidad de registros solicitado es válido se empieza a armar la trama de respuesta, para lo cual se tiene que hacer la lectura de los registros solicitados. Si no ocurre un error en este proceso de lectura se envía la respuesta.

Figura 48. Diagrama de flujo de la funciones 03 y 04



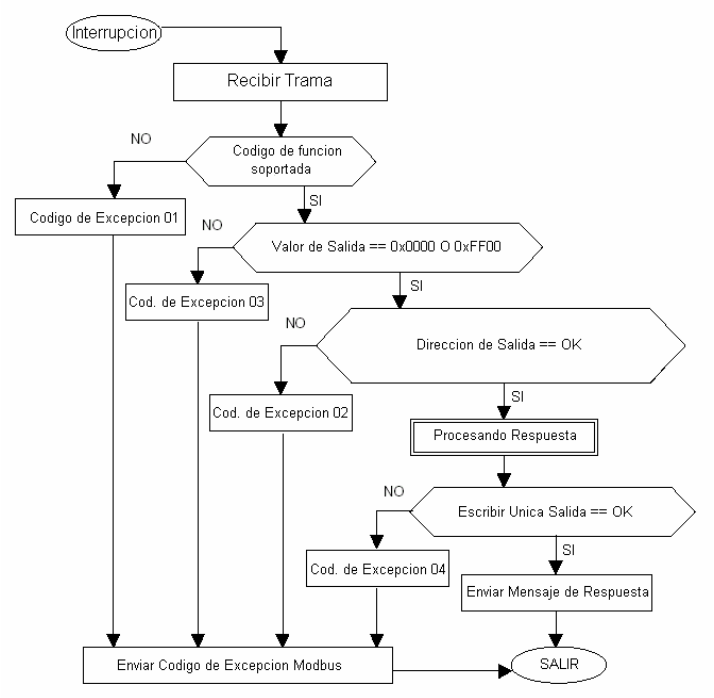
Fuente: Autor

La función 05 (*Force Single Coil*) fue implementada según el diagrama de flujo que muestra en la figura 49.

Se puede observar en esta figura que el esclavo debe verificar que en el campo de datos esté el valor hexadecimal 00 00 o FF 00 para poder forzar una bobina al estado ON o OFF.

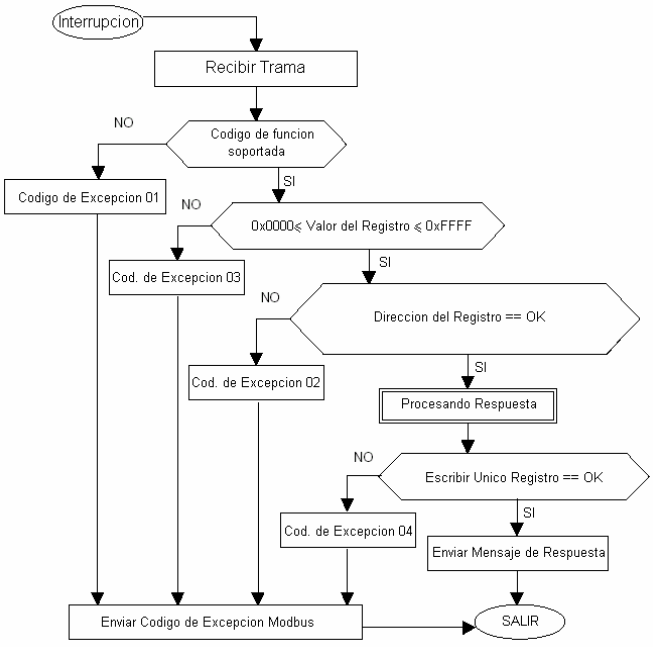
En la figura 50 se muestra el diagrama de flujo para la implementación de la función 06 (*Preset Single Register*). Aunque parezca redundante el esclavo debe verificar que el dato que contendrá el registro está entre 00 00 y FF FF hexadecimal.

Figura 49. Diagrama de flujo de la función 05



Fuente: Autor

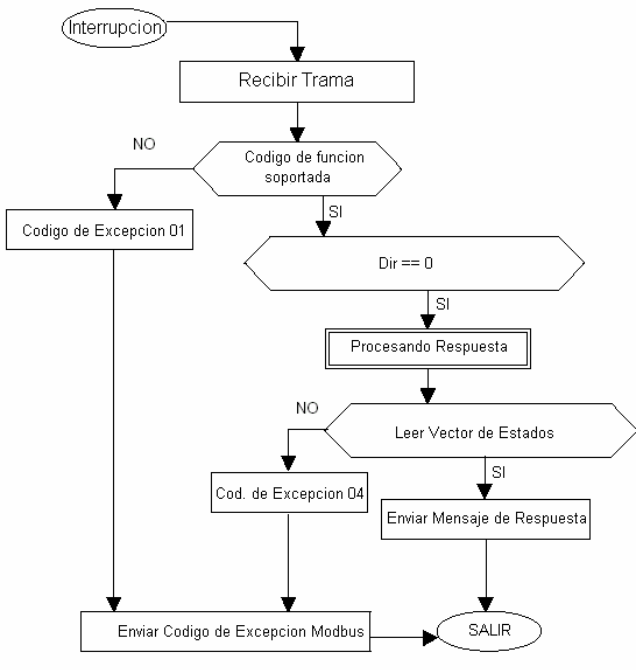
Figura 50. Diagrama de flujo de la función 06



Fuente: Autor

En la figura 51 se muestra el diagrama de flujo para implementar la función 07 (*Read Exception Status*). En esta función el esclavo no necesita que el maestro le suministre la dirección de los datos, ya que cada esclavo tiene una dirección fija y conocida de estos. Por esta razón no habrá respuesta de excepción 02 o 03.

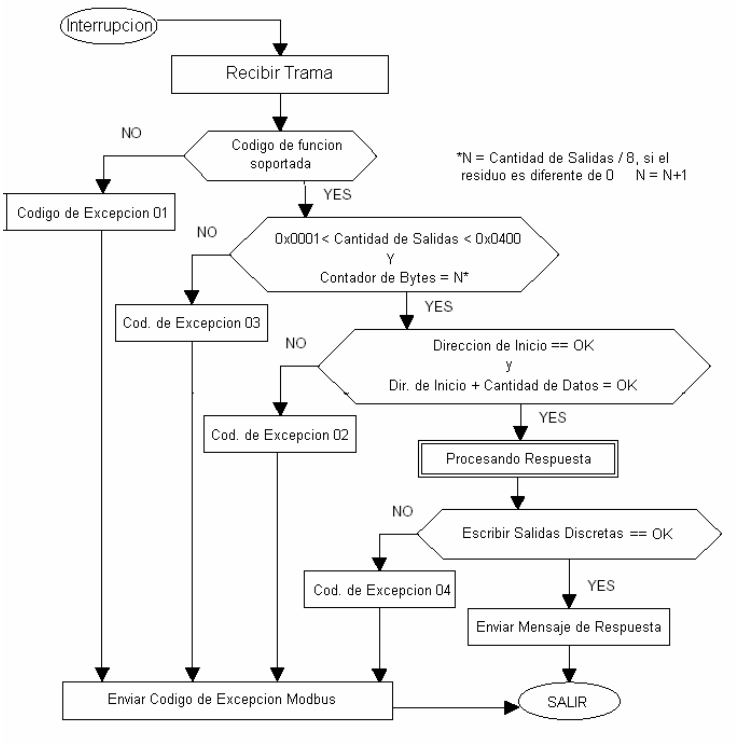
Figura 51. Diagrama de flujo de la función 07



Fuente: Autor

En la figura 52 se muestra el diagrama de flujo para implementar la función 15 (*Force Múltiple Coils*). El esclavo debe verificar que el contador de bytes coincida con la cantidad de bytes que informan el número de salidas a forzar. Por ejemplo, si se van a forzar 256 salidas el contador de bytes debe tener el valor 02 hexadecimal; ya que con un byte (8 bits) se puede representar máximo el 255.

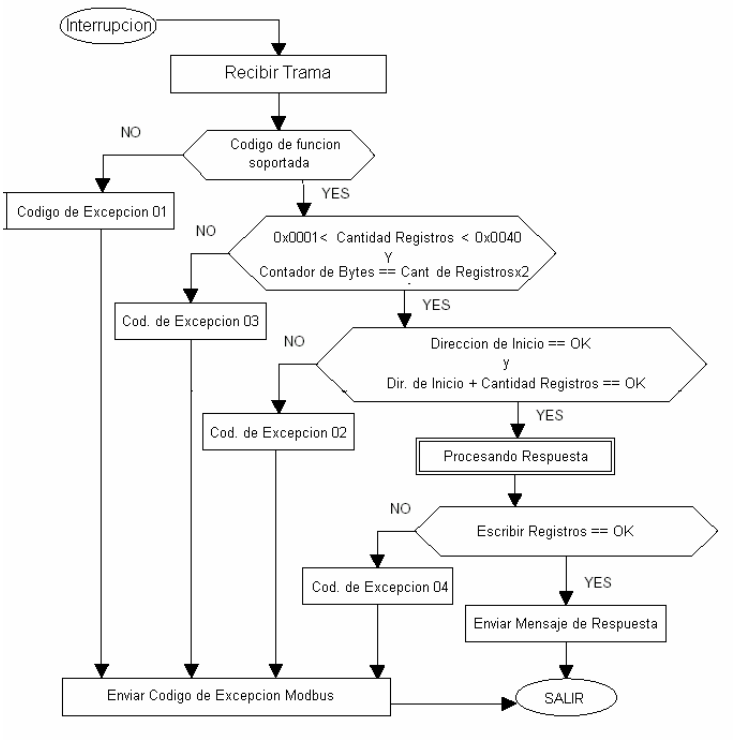
Figura 52. Diagrama de flujo de la función 15



Fuente: Autor

Por último en la figura 53 se muestra el diagrama de flujo que se utilizó para la implementación de la función 16 (*Preset Multiples Registers*). Un código de excepción 03 se puede presentar si el número de registros enviados en la consulta no coincida con los registros del esclavo o el contador de bytes sea diferente a dos veces el número de registros a forzar. Además se tiene que verificar que la dirección de inicio sea correcta y que no exista desbordamiento con la cantidad de registros a forzar.

Figura 53. Diagrama de flujo de la función 16.



Fuente: Autor

3.6 Utilización de la Memoria Flash

La memoria flash tiene dos modos de operación conocidos como modo monitor y modo usuario. El primer modo es utilizado para grabar el programa que luego será ejecutado por el microcontrolador y el segundo para utilizar esta memoria como una EEPROM. Una explicación en detalle de cómo entrar en el modo monitor se encuentra en el documento “MC68HC908GP32 Technical Data” de Motorola.

La memoria flash se ha utilizado para guardar los datos de configuración del procesador de comunicaciones, el estado de las entradas y salidas discretas, el estado de los registros de entrada o salida, guardar el diario de eventos y para construir el mensaje de respuesta. Para todo esto es necesario grabar los datos en

la memoria flash. El proceso de grabado de la flash tiene alguna consideraciones que se describen a continuación.
La memoria FLASH está organizada en páginas de 128 bytes, permitiéndose la grabación mínima de 1 byte, el borrado mínimo de 1 página y el borrado completo (o masivo) en una sola operación. Se pueden grabar hasta 64 bytes (un row) por operación pero el algoritmo presentado aquí lo hace de a un byte por vez para utilizar el mínimo posible de RAM.

Cuando un byte de la memoria está borrado se lee como $FF y sólo se le pueden grabar bits en 0. Para pasar un solo bit de 0 a 1, deben borrarse los 128 bytes de la página.

Mientras se modifica la FLASH ésta no puede ser utilizada. Esto obliga a ejecutar el código de grabación desde RAM y no pueden atenderse las interrupciones. El algoritmo presentado graba 1 byte en 100 microsegundos y borra una página (128 bytes) en 1 milisegundo.

Las siguientes direcciones están implementadas en memoria FLASH:

8000 - FDFF (32256 bytes, páginas 0..251)
FF7E (FLBPR, página 254)
FFDC - FFFF (36 bytes, página 255)

El registro FLBPR (*FLash Block Protect*) permite indicar hasta que página se puede modificar en modo usuario (255 = toda la memoria, 0 = nada), para impedir sobregrabar o borrar por accidente el programa. Este registro sólo puede modificarse desde modo monitor.

Esto obliga a poner los datos que se deseen modificar en la dirección 8000 (y bytes siguientes) y el código a continuación (hasta FDFF, cuidando de no usar una misma página para código y datos).

Algoritmo de grabación:

Para modificar un byte deben seguirse los siguientes pasos:

Inhabilitar las interrupciones
Subir en el registro FLCR (FLash Control Register) el bit PGM ($01)
Leer el registro FLBPR (FLash Block Protect)
Grabar cualquier valor en el byte a modificar (esto selecciona el row)
Delay TNVS = 10 microsegundos (PGM to HVEN setup time)
Subir en el registro FLCR (FLash Control Register) el bit HVEN ($08)
Delay TPGS = 5 microsegundos (Program Hold Time)
Grabar el valor correcto en el byte a modificar
Delay TPROG = 30 microsegundos (Byte Program Time)
Bajar en el registro FLCR (FLash Control Register) el bit PGM ($01)
Delay TNVH = 5 microsegundos (High-Voltage Hold Time)
Bajar en el registro FLCR (FLash Control Register) el bit HVEN ($08)
Delay TRCV = 1 microsegundos (Return to read mode)
Recuperar estado de las interrupciones.

Este proceso lo hace la función llamada FLRam(), que se muestra a continuación. Como se observa esta rutina debe ser escrita en assembler.

```
void FLRam (void) {

#asm
lda #1          ;Subir PGM,
sta 0xFE08      ;
lda 0xFF7E      ;Lee desde Flash Block Protect Register
sta 0x8000      ;
lda #4          ;(TNVS) 10 uS
_d1:dbnza _d1   ;
lda #0x09       ;Subir HVEN
sta 0xFE08      ;en Flash Control Register
lda #2          ;(TPGS) 5 uS
_d2:dbnza _d2   ;
_d1:dbnza _d1   ;
lda #0x09       ;Subir HVEN
sta 0xFE08      ;en Flash Control Register
lda #2          ;(TPGS) 5 uS
_d2:dbnza _d2   ;
```

```
                ;Escribir datos en Flash
lda FLDato      ;Traer dato a grabar
sta 0x8000      ;Escribir en Flash
lda #12         ;(TPROG) 30us
_d3:dbnza _d3
lda #0x08       ;Bajar PGM
sta 0xFE08      ;en Flash Control Register
lda #2          ;(TNVH) 5 uS
_d4:dbnza _d4
clra            ;Bajar HVEN
sta 0xFE08      ;en Flash Control Register
nop             ;1us
rts             ;retorno
#endasm
}
```

Esta rutina debe ser ejecutada desde memoria RAM. Esto se logra utilizando otra rutina llamada EnRAM() que se encarga de trasladar las instrucciones desde memoria Flash hasta un sector de memoria RAM reservada para este propósito. Esta rutina se muestra a continuación:

```
void EnRAM()
{
unsigned char *origen;
unsigned char *destino;
unsigned char i;

destino = &FER[0];          // Destino=Array en Ram
origen  = (char *)FLRam;    // Origen=Función en Flash
FuncionGrabar = destino;
for(i=0; i<51; i++)
*destino++ = *origen++;
}
```

Por último se utilizó la función FLGrabar(int FDir) que tiene tres propósitos: indicar la dirección donde será grabado el dato, deshabilitar las interrupciones y dar la orden de ejecutar la función FLRam(). Las instrucciones de esta función se muestra a continuación:

```
void FLGrabar (int FDir)
{

FER[29]=FDir;
FER[28]=FDir>>8;
DISABLE_INT
(*FuncionGrabar)();     // Rutina de grabacion

ENABLE_INT

}
```

esta función recibe como argumento un dato entero (16 bits) que debe ser la dirección de memoria donde serán grabado los datos.

Para poder rescribir datos en la memoria flash es necesario borrar los datos que se encuentran en el momento de la grabación. Para esto se debe disponer de un proceso de borrado que se describe a continuación:

Algoritmo de borrado:

Para borrar una página deben seguirse los siguientes pasos:

Tapar las interrupciones
Subir en el registro FLCR (FLash Control Register) el bit ERASE ($02)
Leer el registro FLBPR (FLash Block Protect)
Grabar cualquier valor en la página a borrar (esto selecciona la página)
Delay TNVS = 10 microsegundos (ERASE to HVEN setup time)
Delay TERASE = 1 milisegundo (Erase block time)
Bajar en el registro FLCR (FLash Control Register) el bit ERASE ($02)
Delay TNVH = 5 microsegundos (High-Voltage Hold Time)
Bajar en el registro FLCR (FLash Control Register) el bit HVEN ($08)
Delay TRCV = 1 microsegundos (Return to read mode)
Recuperar estado de las interrupciones
Implementación:

Este proceso lo hace la función llamada FLRamB(), que se muestra a continuación:

```
void FLRamB (void)
{

#asm
lda #2          ;Subir ERASE,
sta 0xFE08      ;
lda 0xFF7E      ;Lee desde Flash Block Protect Register
sta 0x8000      ;
lda #4          ;(TNVS) 10 uS
_d1:dbnza _d1   ;
lda #0x0A       ;Subir HVEN
sta 0xFE08      ;en Flash Control Register
nop             ;1us
nop
lda #0x08       ;Bajar ERASE
sta 0xFE08      ;en Flash Control Register
lda #2          ;(TNVH) 5us
_d2:dbnza _d2
clra            ;Bajar HVEN
sta 0xFE08      ;en Flash Control Register
nop             ;1us
nop
rts             ;retorno
#endasm
}
```

Esta rutina debe ser ejecutada desde memoria RAM. Esto se logra utilizando otra rutina llamada EnRAMB() que se encarga de trasladar las instrucciones desde la Flash hasta un sector de memoria RAM reservada para este propósito. Esta rutina se muestra a continuación:

```
void EnRAMB()
{
unsigned char *origen;
unsigned char *destino;
unsigned char i;

destino = &FER[0];          // Destino = Array en Ram
origen  = (char *)FLRamB;   // Origen  = Función en Flash
FuncionBorrar = destino;
for(i=0; i<51; i++)
*destino++ = *origen++;
}
```

Las rutinas presentadas realizan estos dos algoritmos manteniendo al mínimo la cantidad de RAM necesaria. Para ello realizan la mayor cantidad posible de inicializaciones antes de grabar o borrar y no tocan la máscara de interrupciones, por lo que requieren llamarlas con las interrupciones inhabilitadas.

Nota:

Los retardos están calculados para una frecuencia de bus de 4.9152 MHz (lo que equivale a 1 microsegundo = 2 ciclos). Si se decide utilizar otra frecuencia de bus deberá ajustar las constantes asegurándose que se mantengan los tiempos mínimos.

Como ejemplo de cómo grabar cinco datos en un sector de memoria supóngase que se tienen los datos en un vector de cinco posiciones llamado Datos[] y que se van a grabar en forma consecutiva a partir de la dirección $8000. El proceso completo se muestra a continuación.

```
EnRAMB();
FER[9] = 0x80;
FER[10]= 0x00;
DISABLE_INT;
(*FuncionBorrar)();

ENABLE_INT;
EnRAM();
pntr=(unsigned char*)Datos;
for( j=0; j<5; j++)
   {
    FLDato=*pntr++;
    FLGrabar(0x8000 + j );
   }
```

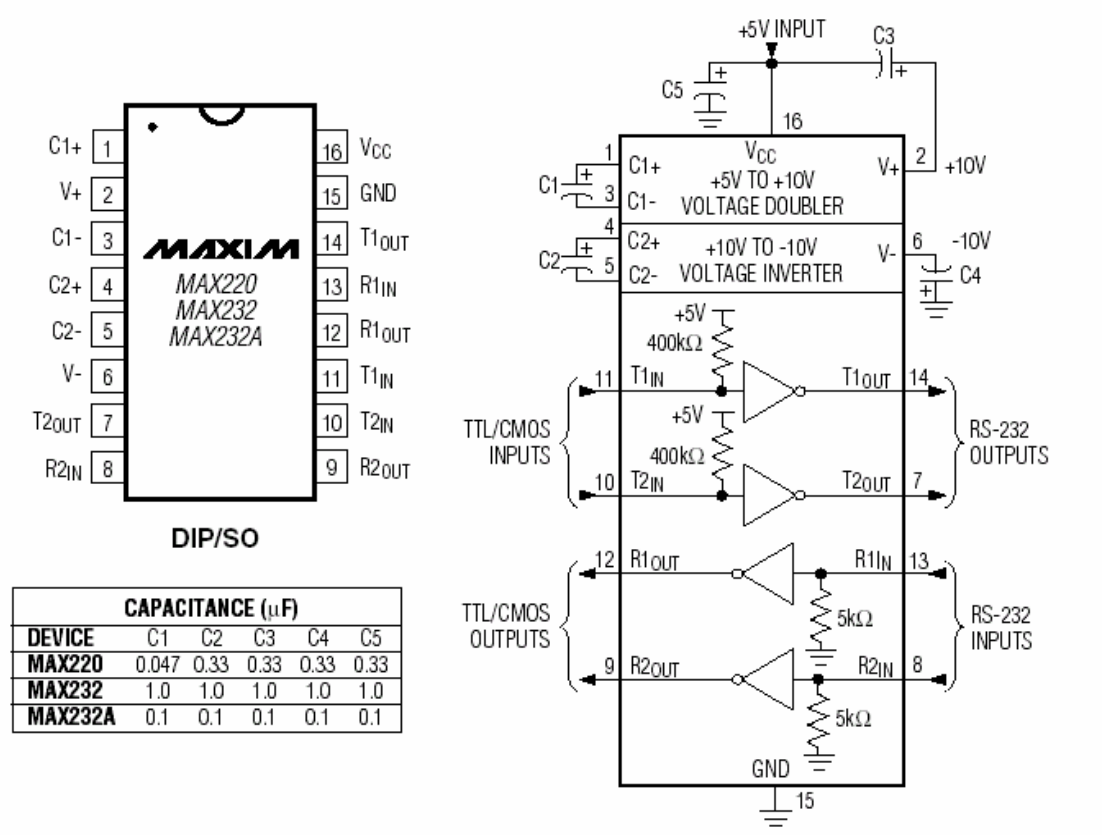
3.7 Conversores de Estándar RS-232 y RS-485

Debido a que el microcontrolador envía y recibe los datos seriales con niveles lógicos TTL/CMOS fue necesario la utilización de circuitos integrados que realizarán la conversión de los niveles TTL al requerido por cada uno de los estándares.

El MAX232 es un circuito integrado que adapta los niveles RS232 y TTL, y permite conectar un PC con un microcontrolador. Sólo es necesario este circuito y 4

condensadores electrolíticos de 1 micro-faradio. El diagrama de pines y el esquema de conexión es el que aparece en la figura 54.

Figura 54. Diagrama de pines y esquema de conexión del MAX232

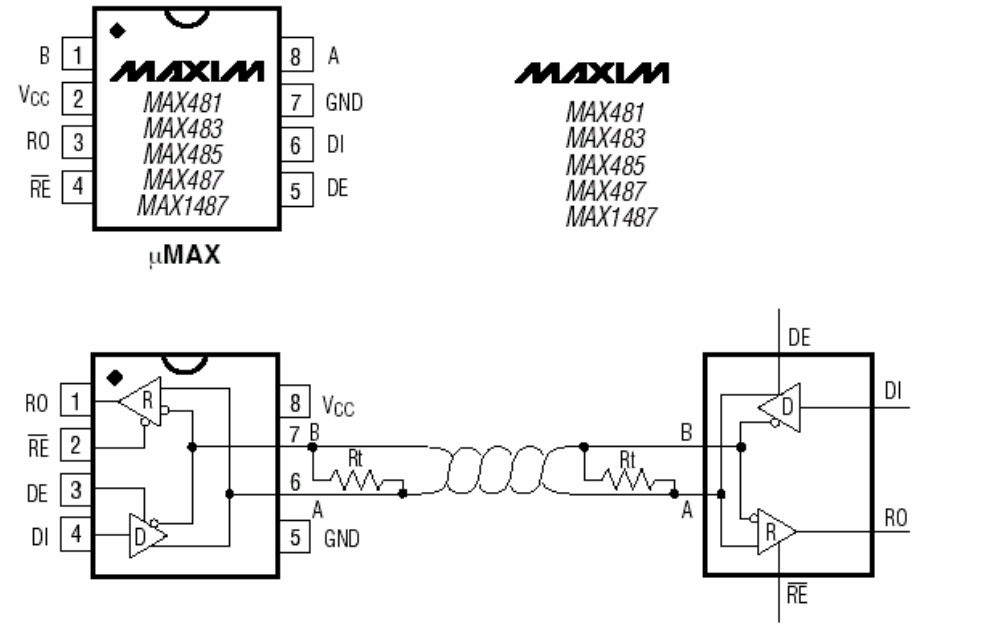


CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Fuente: *MAXIM* www.maxim-ic.com

De igual forma el MAX485 es un circuito integrado que adapta los niveles RS485 y TTL, que permite conectar el microcontrolador a la red que se comunica con este estándar. El diagrama de pines y el circuito de operación típica del MAX485 aparece en la figura 55.

Figura 55. Diagrama de pines y circuito de operación típica del MAX485



Fuente: *MAXIM* www.maxim-ic.com

El microcontrolador recibe los bits por el pin 1 (RO) y los transmite por el pin 4 (DI) del MAX485. Este circuito es *half-duplex* y por esta razón se debe suministrar señales de habilitación en los pines 2 y 3. Para enviar un dato hacia el bus se debe poner en 1 lógico (5V) el pin 3 (DE) y para recibir los datos del bus se debe poner en 0 lógico (0V) el pin 2 ($\overline{RE}$).

Más información de estos circuitos integrados y su operación se encuentra en el documentos “Modbus. Monitoreo de la Red Empleando LabVIEW”, Sinle Marcela Carreño y Pedro Ardila; Cap. 1; Universidad Industrial de Santander, 2005.

4. EL PROCESADOR DE COMUNICACIONES

4.1 Características Generales

El procesador de comunicaciones tiene las siguientes características:

- 1024 Salidas discretas (*Coils*)
- 1024 Entradas discretas (*Inputs*)
- 64 Registros de salidas (*Holding Registers*)
- 64 Registros de entrada (*Input Registers*)
- Velocidad de transmisión de: 600, 1200, 4800, 9600, 14400 y 19200 Baudios
- 1 Puerto RS-232 (DB-9)
- 1 Puerto RS-485 (DB-9)
- Configuración por software.

Para probar físicamente el procesador de comunicaciones basado en el MC68HC908GP32 se construyó un entrenador con las siguientes características:

1 Registro de entrada.
8 Salidas discretas
8 Entradas Discretas.

El registro de entrada se visualiza con un voltímetro análogo ajustado de 0 a 5 voltios DC. Este registro permite la lectura una variable análoga que puede variar de 0 a 5 voltios DC, que corresponde al rango estándar de voltaje obtenidos en la gran mayoría de los medidores; esta lectura corresponde al registro cero del procesador de comunicaciones. Las ocho salidas discretas se visualizan con ocho diodos LED y corresponden a las ocho primeras salidas discretas del procesador (de la 0 a la 7). Las ocho entradas discretas constan de ocho interruptores de codillo con indicador

mediante LED's y corresponden a las ocho primeras entradas discretas del procesador (de la 0 a la 7).
Para obtener como producto final un procesador genérico de comunicaciones que ejecutara el protocolo Modbus RTU se programaron una serie de funciones en C utilizando Codewarrior V3.0. Los archivos que hicieron posible esta implementación son ***Modbus.c***, ***Modbus.h, Vector.c*** y ***Modbus.prm***

En el archivo ***Modbus.c*** se encuentran:

Funciones de comunicación.
Funciones Modbus
Funciones de configuración.

4.2 Funciones de Comunicación

Para la recepción de datos se utilizó la interrupción 13 (SCI *Reciever Full*) que da aviso a la CPU del microcontrolador cada vez que llega un nuevo dato al receptor. Así mismo se utilizaron las interrupciones 4 que da aviso a la CPU cuando se ha cumplido 1.5 tiempo de carácter sin recibir datos y la interrupción 5 que da aviso a la CPU cuando se ha cumplido 3.5 tiempo de carácter sin recibir datos. Para estas interrupciones se programaron rutinas de atención que tiene los siguientes propósitos: Recibir la trama enviada por el maestro,.Validar el tiempo de llegada de los datos, Comprobar que no hallan errores de comunicación.

A continuación se muestran las funciones programadas para la comunicación del procesador:

__interrupt 4 void canal_cero(void)
__interrupt 5 void canal_uno (void)
__interrupt 13 void Rx_receiver(void)
*void SCIRxByte (char *dato)*
void SCITxByte (unsigned char byte)

Las tres primeras son las rutinas de atención a las interrupciones cuyo número aparece en la declaración de cada una de ellas. Las dos últimas son funciones sencillas cuyo único propósito es recibir y transmitir los caracteres por el SCI.

A continuación se muestran las rutinas de atención a las interrupciones 13, 5 y 4.

```
__interrupt 13 void Rx_receiver(void)
{
  if(SCS1_PE) CommErr = 1;
  if(SCS1_OR)
  {
   CharOvrunCount++;
   OverRun = 1;
  }

  SCS1_SCRF = 0;
  if(!_15TOver)
     T1SC_TRST = 1;
  else
     CommErr   = 1;

  SCIRxByte(&DatoRx);
  if(!FrameInit)
  {  T1SC_TSTOP = 0;
     FrameInit  = 1;

     if(DatoRx==Dir||DatoRx==Dif)
     {
        ActiveRx = 1;
        p_Rx = &MenRx[0];
      }
     else Nodir = 1;
   }

   if(ActiveRx && !CommErr)
   {
        *p_Rx++ = DatoRx;
        counter++;
    }

}
__interrupt 5  void canal_uno (void)
{
  T1SC1_CH1F = 0;
  T1SC_TSTOP = 1;
  T1SC_TRST  = 1;
  EvenLog(1);
  Procesar();
}
```

```
__interrupt 4  void canal_cero(void)
{

  T1SC0_CH0F = 0;
  _15TOver   = 1;

  if(!CommErr && !Nodir)
  CRCCheck   = CRC16(&MenRx[0],1,counter-2);

}
```

Para que estas rutinas de atención sean ejecutadas cuando se producen las interrupciones es necesario modificar el archivo ***Vectors.c***, donde se debe hacer la declaración de estas rutinas e indicar en que posición se encuentra dentro del vector de interrupciones. Este archivo se muestra a continuación:

```
#include "Cpu.h"
extern void _EntryPoint(void);
__interrupt 4  void canal_cero (void);
__interrupt 5  void canal_uno  (void);
__interrupt 13 void Rx_receiver(void);
void (* const _vect[])() @0xFFDC = {   // Interrupt table
           Cpu_Interrupt,
           Cpu_Interrupt,
           Cpu_Interrupt,    // 15
           Cpu_Interrupt,
           Rx_receiver,
           Cpu_Interrupt,
           Cpu_Interrupt,
           Cpu_Interrupt,    // 10
           Cpu_Interrupt,
           Cpu_Interrupt,
           Cpu_Interrupt,
           Cpu_Interrupt,
           canal_uno,        // 5
           canal_cero,
           Cpu_Interrupt,
           Cpu_Interrupt,
           Cpu_SWIInterrupt,
           _EntryPoint   };
```

Las funciones SCTxByte() y SCIRxByte() permiten transmitir y recibir un dato por el módulo SCI. El código se muestra a continuación:

```
void SCITxByte (unsigned char byte)
{
  while (!(SCS1 & 0x80));
  SCDR = byte;
}

void SCIRxByte (char *dato)
{
  *dato = SCDR;
}
```

4.3 Funciones Modbus

Este conjunto de funciones tienen como propósito final los siguientes aspectos:

Ejecutar la función solicitada por el maestro.
Enviar excepciones si existen.
Validar el CRC recibido y calcularlo para el envío.
Actualizar el diario de eventos.

El total de las funciones que se encargan de ejecutar las funciones Modbus se listan a continuación:

void Procesar()
void ReadCoilStatus()
void ReadInputStatus()
void ReadHoldingRegisters()
void ReadInputRegisters()
void ForceSingleCoil()
void PresetSingleRegister ()
void ReadExceptionStatus()
void DiagnosticSubfun()
void FetchCommEventCounter()
void FetchCommEventLog()

void ForceMultipleCoils()
void PresetMultipleReg()
void exceptioncode(char)
void EvenLog(unsigned char)
*char CRC16(unsigned char *,char,char)*

Una vez que el canal uno del temporizador halla dado aviso del fin de la trama se empieza a ejecutar la función *Procesar()* que se encarga de llamar a la función que ejecuta la solicitud hecha en la trama recibida y al final de cada proceso llamar la función *EvenLog()* que actualizará el diario de eventos. Existe de igual manera una función llamada *exceptioncode()* que es llamada por las diferentes funciones Modbus cuando se debe enviar un código de excepción al maestro de la red. Las restantes funciones son las que se encargan de la lectura o escritura de las entradas y salidas del procesador de comunicaciones.
Finalmente la función CRC16 calcula el CRC del mensaje recibido o el mensaje a enviar.

4.4 Función de Configuración

Esta función permiten la configuración del procesador con los datos recibidos del programa *ModbusConfig.exe*. En esta función también se reinician las variables del diario de eventos y se dan valores a las entradas y salidas del procesador; esto con el propósito de validar su funcionamiento. En esta función también se configuran los puertos del microcontrolador que serán utilizadas como entradas o salidas reales del procesador de comunicaciones.

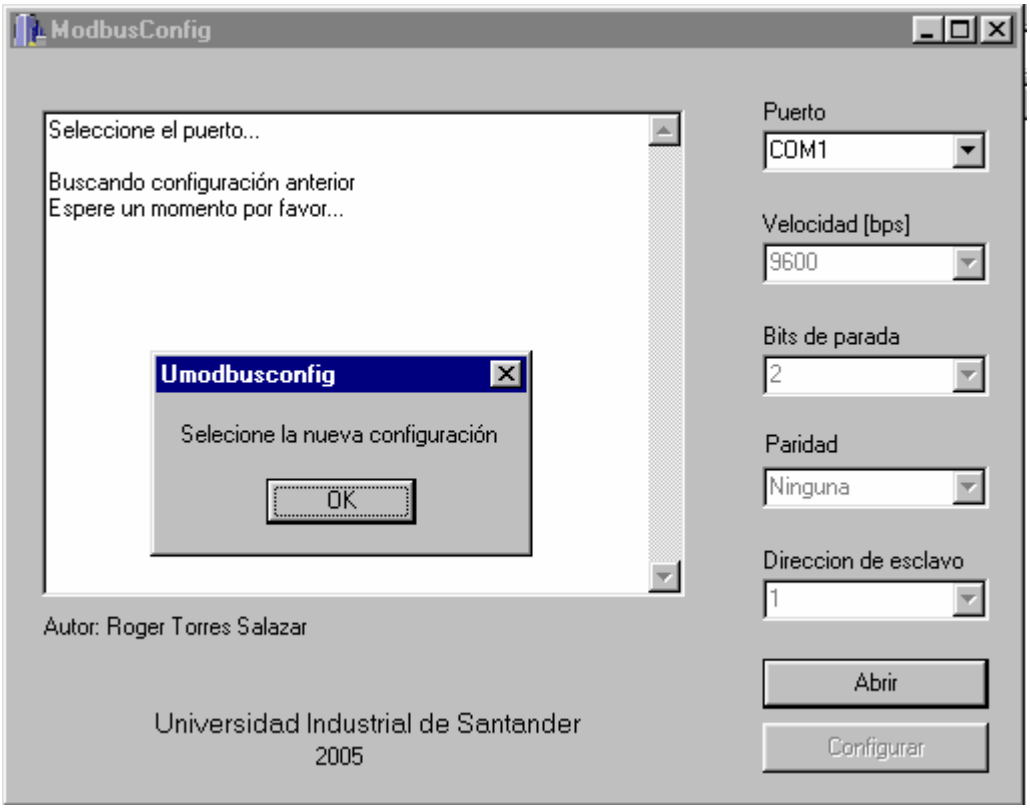
El archivo *ModbusConfig.exe* fue hecho en C++ Builder 5.0 y utiliza funciones de comunicación Win32 API. A continuación se listan las funciones de comunicación utilizadas para crear este archivo:

- *CreateFile,*
- *SetupComm,*

- *GetCommTimeouts*
- *SetCommTimeouts*
- *GetCommState*
- *SetCommState*
- *WriteFile*
- *ReadFile.*

Para más información sobre como utilizar estas funciones consultar el tópico "*Using the communications functions*" en los archivos de ayuda de WinSDK.

Figura 56. Ventana principal *ModbusConfig.exe*



Fuente: Autor

Con los datos recibidos se dan los valores a los registro del SCI y del temporizador para la comunicación dentro de la red. Los parámetros de configuración son: velocidad de transmisión, paridad, número de bits de parada y número de esclavo. Estos datos son grabados en la dirección de memoria 0x8000, de tal forma que el procesador continuará con la última configuración hasta que el usuario decida modificarla. El código de función utilizado para la configuración es 0xFF.

En la figura 56 se muestra la ventana principal de *ModbusConfig.exe*; el cual como se observa permite en forma gráfica seleccionar los parámetros de configuración del procesador de comunicaciones.

Este programa busca la configuración que tenía anteriormente el microcontrolador para la comunicación. Para hacer esto, envía un mensaje con velocidad y paridad diferentes hasta que el microcontrolador haga eco de la trama enviada. Esta detección la debe hacer para poder enviar los datos de la nueva configuración de forma segura. Después de esto, el programa le permite al usuario seleccionar los parámetros de la nueva configuración. Una vez seleccionados estos parámetros en la ventana principal se pueden enviar al microcontrolador haciendo clic en el botón “Configurar”. El programa advertirá si la configuración fue o no exitosa.

4.5 El Archivo de Cabecera Modbus.h

Este archivo se hizo con los siguientes propósitos:

Reservar espacios de memoria RAM y flash para almacenar datos que son necesarios para el programa y la ejecución de Modbus. La memoria Flash se utilizó para almacenar el estado de las bobinas y entradas discretas, estado de los registros mantenidos y los registros de salida, variables involucradas en el diario de eventos, caracteres que conforman el mensaje de respuesta. Parte de la RAM es utilizada para ejecutar las rutinas de grabado y borrado de sectores de la flash ya que estas rutinas no pueden ser ejecutas desde la misma flash.

Definir las banderas que se utilizan para indicar un estado. Estos estados son:

FrameInit	Indica si una trama se está enviando por el bus
ActiveRx	Indica si la recepción de una trama está en adelanto
_15TOver	Indica si a transcurrido 1.5T de caracter
CommErr	Indica si ha ocurrido un error de comunicación
CRCCheck	Indica si es correcto el CRC.
Nodir	Indica si la transmisión está dirigida a otro esclavo
ListenOnly	Indica si el esclavo se a forzado al modo de solo escucha
Ocupado	Indica si el esclavo esta procesando alguna función
ReadExcSend	Códigos de excepción 1 a 3
SlaveAbort	Código de excepción 4
SlaveBusy	Códigos de excepción 5 y 6
SlaveNAK	Código de excepción 7
WTimeOut	Agotado tiempo de escritura

Las primeras ocho banderas son utilizadas para el proceso de recepción de una trama y las siguientes cinco son utilizadas cuando se produce alguna de las excepciones en el intercambio de mensajes. Estas banderas son puestas en uno o cero lógico en diferentes partes del programa y son revisadas donde es necesario para tomar una acción a seguir. La primera bandera FrameInit se encuentra en cero mientras no se esté recibiendo ninguna trama. Si se detecta un carácter de entrada en el modulo SCI se verifica el estado de esta bandera; si su estado es cero si inicia la cuenta de tiempo en el temporizador previamente configurado para indicar mediante interrupciones cuando se cumple 1.5 o 3.5 de carácter que sirven para verificar que no se produzcan errores en la comunicación o cuando se detecta el fin de una trama; una vez se iniciado la cuenta temporal, se pone esta bandera en uno para no repetir este proceso. Este primer dato que ha llegado debe ser la dirección del esclavo al cual va dirigido el mensaje, si este dato corresponde a la dirección con la que se ha configurado al microcontrolador como esclavo o a un cero, que es dirección que se asigna en una difusión, se establece en uno la bandera ActiveRx que indica que se debe recibir y concatenar los siguientes datos

detectados en el bus para luego procesar la trama y ejecutar la acción pedida en esta. Si en este primer campo existe la dirección de otro esclavo de la red, se establece en uno la bandera NoDir para no recibir los datos entrantes. En este último caso se debe llevar la cuenta temporal entre datos entrantes para determinar cuando a terminado una trama y poder detectar correctamente el inicio de otra.

La bandera _15TOver es puesta en uno cuando se ha cumplido 1.5 veces el tiempo de un carácter desde el último recibido sin que se haya detectado la entrada de otro carácter. Cualquier carácter entrante en ese momento genera un error en la comunicación y se debe abortar el proceso de recepción y ejecución de la trama. El estado de esta bandera es revisado cada vez que llega un carácter para verificar que no haya violado el tiempo permitido de llegada entre caracteres. La cuenta temporal de 1.5 tiempo de carácter se realiza con el canal cero del *timer.*

La bandera CommErr se pone en uno cuando se presenta alguno de los siguientes casos: error en la paridad de un carácter recibido, cuando se ha recibido un carater entre 1.5Tc y 3.5Tc; esta bandera se revisa repetidas veces en diferentes parte del programa para evitar por ejemplo que se procesen tramas con errores. Si se detecta un error, se abandona el proceso de recepción y se continua la cuenta temporal para determinar el fin de una trama y el comienzo de otra.

La bandera CRCCheck se utiliza para indicar si el CRC de una trama recibida es correcta; ésta se establece en uno cuando el CRC enviado en la trama coincide con el CRC calculado por el microcontrolador.

La bandera ListenOnly es utilizada para indicar si el modo se solo escucha se le es impuesto al microcontrolador. Se debe recordar que este modo de operación es válido solo cuando el microcontrolador se a configurado como un esclavo de la red, en cuyo caso debe recibir y procesar las tramas pero no enviar mensaje de respuesta.

4.6 El Archivo Modbus.prm

Este archivo contiene las direcciones de las memorias Flash, RAM y ROM. Fue necesario modificar este archivo para reservar espacios de memoria Flash para utilizarla como EEPROM y RAM para ejecutar las rutinas de grabado y borrado de memoria Flash. En la figura 57 se muestra el archivo final **Modbus.prm**.

Figura 57 Archivo *Modbus.prm*

```
NAMES

END

SECTIONS
      ROM    =  READ_ONLY              0x8400 SIZE 0x7A00;
      MYROM  =  READ_ONLY              0x8000 SIZE 0x0400;
      ZPAGE  =  READ_WRITE             0x0040 SIZE 0x0020;
      MYRAM  =  READ_WRITE             0x0060 SIZE 0x0040;
      RAM    =  READ_WRITE             0x00A0 SIZE 0x01A0;

END

PLACEMENT
      DEFAULT_RAM                      INTO  RAM;
      DEFAULT_ROM, ROM_VAR, STRINGS    INTO  ROM;
      _DATA_ZEROPAGE                   INTO  ZPAGE;
      BUFFER                                       INTO  MYRAM;
      REGISTERS                                    INTO  MYROM;

END

INIT _EntryPoint
STACKSIZE 0x0080
```

Fuente: Autor

4.7 Comunicación Serial del Procesador

El estándar a utilizar para la comunicación se debe seleccionar con un interruptor. Esto se hizo teniendo en cuenta que el circuito integrado MAX485 pone en 1 la salida RO si se tiene habilitado el receptor y los pines A y B están en estado de alta impedancia. Este interruptor permite la alimentación de uno de los dos circuitos para que no halla colisión de datos en el receptor del microcontrolador.

El procesador tiene dos conectores DB-9 hembras para la comunicación RS-232 y RS-485; mediante un led se visualiza el estándar que se está utilizando para la comunicación.

CONCLUSIONES Y OBSERVACIONES

Se pudo obtener como producto final un procesador genérico de comunicación que ejecuta el protocolo Modbus en el modo de transmisión RTU utilizando un microcontrolador de propósito general con CPU de 8 bits. Este procesador soporta las funciones generales del protocolo y se conecta a una red con topología en bus según los estándares RS-232 y RS-485 seleccionables por el usuario. El tipo de conexión a la red es *Half-Duplex* y permite velocidades que son múltiplo pares de 150 y 200, alcanzando una velocidad máxima de 76800 baudios utilizando un cristal de 4.9152 MHz.

Esto se pudo lograr gracias a la forma realmente abierta de este protocolo y la documentación que se tiene del mismo, que permite conocer en forma exacta como es el proceso de comunicación entre los diferentes elementos de la red, el formato de las tramas y la ubicación de los diferentes datos que las componen, cómo se detectan posibles errores y cómo tratarlos.

Este trabajo puede tener aplicación en el estudio de redes industriales con Modbus como protocolo de comunicación, y permite implementar estas redes con propósitos académicos. Además se puede pensar en hacer una red industrial real utilizando el programa hecho con este microcontrolador.

La principal ventaja de este procesador de comunicaciones frente a los PLC's es que permite ejecutar todas las funciones Modbus además de funciones de usuario, reservadas para propósitos específicos.

Se podrán desarrollar otros trabajos de grado e investigación que entreguen como producto final un equipo que cumpla con la tarea para los que fueron diseñados y además se pueda conectar a una red Modbus RTU. Esta posibilidad se puede extender también a Modbus ASCII

Se pueden hacer redes Modbus confiables a un costo relativamente bajo, si se compara éste último con el precio que se tiene que pagar para montar una red con PLC's idéntica a la implementada para la validación de este trabajo.

Junto al procesador de comunicaciones se tiene programa *ModbusConfig.exe*[6] para la configuración por software del mismo. Este programa permite de forma sencilla seleccionar la velocidad de comunicación, el tipo de paridad, los bits de parada y el número de esclavo que identifique al procesador de comunicaciones dentro de la red. Con la configuración por software se tiene la ventaja de utilizar el puerto RS232 para hacer llegar los datos de dicha configuración. Otra forma de es utilizar interruptores que permitan establecer en uno o cero diferentes bits que luego se convertirán en información para lograr la configuración. Este último método no fue tenido en cuenta por requerir de bits que luego fueron usados para comprobar el funcionamiento del procesador de comunicaciones en el entrenador.

El funcionamiento de este procesador fue probado en una red constituida por tres elementos con comunicación serial RS485. Estos elementos son: El procesador de comunicaciones Modbus RTU basado en microcontrolador, un autómata programable TRILOGI actuando como esclavos y un PC como maestro. Este procesador tiene entradas y salidas reales y otras que son datos almacenados en la memoria del microcontrolador. Para observar las primeras 8 salidas y establecer el estado de las 8 primeras entradas se construyó un entrenador que permite visualizar con leds el estado de estos datos (*Coils e Inputs de 0 a 7*), y mediante un voltímetro el primer registro de entrada (*Input Register 00 00*) que corresponde a una variable analógica dentro del rango de 0 a 5 VDC.

La comunicación bajo los estándares RS-232 y RS-485 se logró con la utilización de circuitos integrados que proporcionan la conversión de datos binarios con niveles lógicos TTL a los niveles de tensión o corriente de estos estándares, y viceversa. Estos circuitos son el MAX232 y MAX485 de fácil adquisición en el mercado local a un coste relativamente bajo. La forma de utilizar estos circuitos es realmente sencilla

[6] Programa hecho por el autor utilizando la herramienta de programación C++ Builder 5.0

y se reduce a la conexión de pocos elementos discretos y la utilización de una señal de habilitación que permita escuchar ó tener acceso al bus ya que la comunicación es *Half-Duplex*.

El microcontrolador MC68HC908GP32 de motorola sirvió para la implementación del procesador de comunicaciones y se comprobó que éste posee las características y herramientas necesarias para lograr dicho objetivo. Este microcontrolador tiene entre otros un módulo para la comunicación serial (SCI *Serial Communication Interface*), que se utilizó la comunicación con otros dispositivos de una red y dos módulos temporizadores (TIM1 y TIM2 *Timer Interface Module*). Sólo se utilizó uno de estos módulos (TIM1), que sirvió para vigilar que el tiempo de llegada de los caracteres era el permitido por Modbus en este modo de transmisión.

Así mismo, este microcontrolador tiene 5 puertos que ofrecen en total 33 pines configurables por software como entradas o salidas. De estos puertos se utilizó el puerto A para las primeras 8 salidas discretas; del puerto B se utilizó un pin para el primer registro de entrada y los restantes pines junto con dos pines del puerto D para las primeras 8 salidas discretas.

El GP32 tiene un vector que consta de 25 interrupciones, de las cuales se utilizaron la interrupción 13 (*SCI receiver full)*, la interrupción 5 (*TIM1 Channel 1)* y la interrupción 4 (*TIM1 Channel 0*).

Las características del módulo SCI[7] utilizadas para este trabajo fueron:

- 32 velocidades programables
- Longitud de carácter de 8 o 9 bits
- Habilitación separada de transmisor y receptor
- Petición separada de interrupción de transmisor y receptor a la CPU
- Polaridad de salida programable del transmisor

[7] Todas la características del SCI se pueden encontrar en el documento : "MC68HC908GP32 Technical Data" Rev. 5 de motorola www.motorola.com

- Banderas de interrupción
  - Receptor lleno
  - Traslape en el receptor
  - Error por ruido
  - Error de paridad
- Chequeo de paridad por hardware
- Detección de ruido con base 1/16 tiempo de bit
- Configuración de registros bit a bit

El SCI tiene 7 registros de lectura o escritura (o ambas) que permiten seleccionar las características mencionadas anteriormente. El registro SCBR[8] (*SCI Baud Rate Register*) permitió configurar la velocidad de comunicación. De las 32 posibles velocidades que tiene el GP32 para la comunicación serial se dispusieron 18 para la configuración, ya que se utilizó una fuente de reloj externa generada con un cristal de 4.9152 MHz que permite seleccionar 24 velocidades con valores enteros. Para aclarar lo anterior obsérvese la tabla 16.

Tabla 16. Velocidades en baudio del SCI con un cristal de 4.9152 MHz

BR / PS	1	2	4	8	16	32	64	128
1	76800	38400	19200	9600	4800	2400	1200	600
3	25600	12800	6400	3200	1600	800	400	200
4	19200	9600	4800	2400	1200	600	300	150
13	5042,31	2521,15	1260,58	630,29	315,14	157,57	78,79	39,39

Fuente: Autor

En esta tabla se muestran las 32 velocidades que se obtienen para los 4 valores posibles del *prescaler* (PS) y los 8 del *baud rate* (BR). Obsérvese que se repiten 6 velocidades; por ejemplo se puede configurar la velocidad de transmisión a 9600 baudios con el *prescaler* en 4 y el *baud rate* en 2 o *prescaler* en 1 y el *baud rate* en

[8] Ver figura 37

8. Así mismo se observa en esta tabla que con un *prescaler* en 13 la velocidad de transmisión no es un número entero.

Sin embargo no resulta crítico el hecho de seleccionar un cristal con una oscilación diferente como se demuestra en el documento "Procesador de Comunicación Modbus" [9] donde se utilizó un cristal de 8 MHz. En la tabla 14, página 149 del documento referenciado se muestran los errores que resultan al configurar la velocidad de transmisión; estos errores no conducen a error en la comunicación como explica el autor.

La longitud de caracter se seleccionó de 9 bits ya que este el número (de bits) que tienen los datos manipulados por Modbus para el modo de transmisión RTU. El microcontrolador utiliza el noveno bit para enviar un bit de paridad o un segundo bit de parada. Adicional a estos nueve bits el microcontrolador envía un bit de inicio y un bit de parada[10]. En total se tienen los 11 bits requeridos por cada dato a transmitir en este modo de transmisión.

La característica de habilitar separadamente el transmisor y el receptor podría servir para deshabilitar el transmisor cuando se fuerza al procesador de comunicaciones al modo de solo escucha. Sin embargo, cuando este es el caso se utiliza la bandera *ListenOnly* de la estructura de estados definida en el archivo *Modbus.h* para evitar estar configurando reiteradas veces los registros del SCI.

La habilitación de la interrupcion *SCI receiver full* resulta de gran utilidad ya que evita que a la CPU se le obligue, por comandos del programa principal, a estar preguntando constantemente si hay algún dato en el bus de comunicación. Esto permite que la CPU adelante otros procesos indicados en el programa principal[11]. Cuando un dato se ha recibido libre de errores, éste es almacenado en el *buffer* de recepción y la SCI advierte a la CPU del evento mediante esta interrupción, para la

[9] Jorge E. Duque Pardo; Trabajo final de maestría. Universidad Industrial de Santander, 2005.
[10] El programador no tiene que encargarse de estos bits, ya que el microcontrolador los pone para cumplir con los estándares de comunicación serial asíncrona.
[11] Por ejemplo, el control de velocidad de un motor.

cual se ha programado la rutina de atención a interrupción *Rx_receiver(void)*. Esta rutina se encarga de armar la trama con los datos entrantes y vigila que no existan errores de comunicación en el proceso de recepción. Estos errores pueden ser: traslape en el receptor, error por ruido y error de paridad. Aunque estas banderas generan una petición de interrupción a la CPU no se habilitaron; cuando un dato se recibe se verifica el estado de éstas para detectar si existen condiciones de error.

Gracias al chequeo de paridad por hardware que posee este módulo no fue necesario generar código para su verificación. Así mismo, el hardware del SCI discrimina si se ha enviado o no un bit de inicio; esto lo logra muestreando este bit a una velocidad que es 16 veces la rata de baudio[12].

Por otra parte el módulo TIM hizo posible que se verificara el tiempo de llegada de los datos e identificar el fin de una trama. Para esto se colocan valores en los módulos contadores de cada canal que correspondan a los retardos que se muestran en la tabla 13. El canal 0 se utilizó para llevar la cuenta de 1.5Tc y el canal uno para la cuenta de 3.5Tc. El valor que se coloquen en los módulos dependen del *prescaler* que se halla seleccionado y la velocidad de transmisión. Para la implementación de Modbus RTU se debió seleccionar el *prescaler* en 5 para cubrir el rango de retardos necesarios para las velocidades posibles del procesador de comunicaciones.

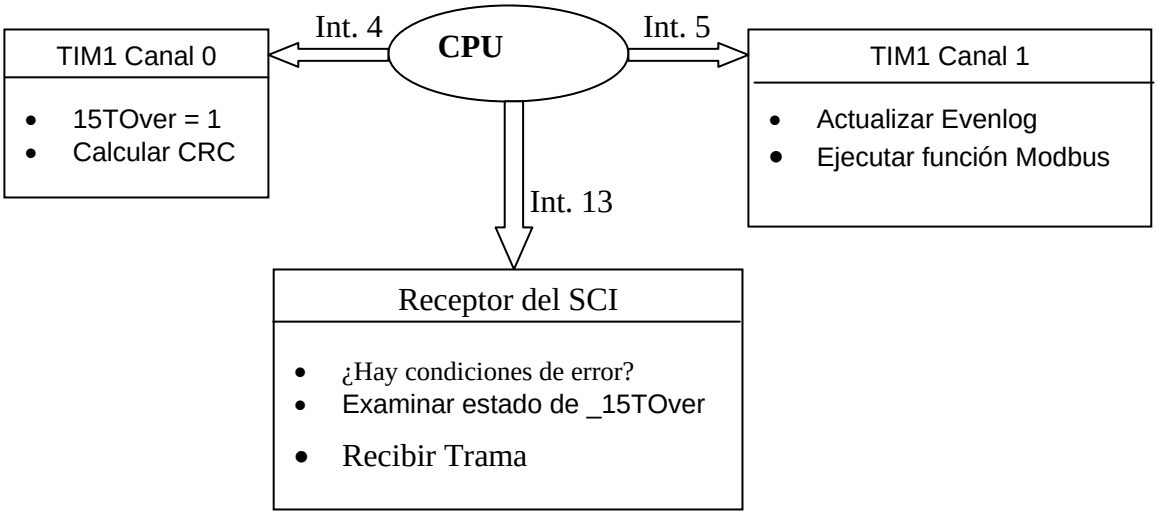
Las bases para la implementación de Modbus RTU en el microcontrolador son en el manejo del módulo de comunicaciones serial y el temporizador. En la figura 58 se muestra un diagrama de estado que simboliza en forma simplificada el proceso de recepción de una trama utilizando las interrupciones de estos módulos.

En la figura 58 se puede observar que el receptor advierte a la CPU que un nuevo dato a llegado mediante la interrupción 13 y la CPU comienza a ejecutar la rutina de atención a esa interrupción. En esta rutina se destacan tres aspectos importantes; primero se debe preguntar si hay condiciones de error, en caso afirmativo se

[12] Para mayor información consultar el documento “MC68HC908GP32 Technical Data” Rev 5, motorola www.motorola.com

establece en 1 la bandera CommError, que se encuentra en la estructura de estados definida en *Modbus.h* y la trama no será procesada. Seguido a esto se debe examinar el estado de la bandera _15Tover. Si se encuentra en 1 se tendrá una condición de error, si está en cero se debe reiniciar el contador del temporizador. Por último si no existe ninguna condición de error debe recibir el dato y armar la trama.

Figura 58. Diagrama de estados del proceso de recepción de una trama.



Fuente: Autor

La interrupción 4 se da cuando el contador del temporizador a alcanzado el valor que contiene el módulo de contador del canal cero. La rutina de atención debe poner en uno el estado de la bandera _15TOver y calcular el CRC de la trama recibida. Finalmente la interrupción 5 se da cuando el contador del temporizador a alcanzado el valor que contiene el módulo de contador del canal 1; este evento indica el fin de la trama. La rutina de atención debe actualizar el diario de eventos (*EvenLog*) y ejecutar la función Modbus indicada en la consulta.

La memoria flash de este microcontrolador se puede utilizar como una EEPROM. Esto permitió grabar en algunos sectores de memoria los datos correspondiente a las salidas y entradas del procesador de comunicaciones. Además permitió tener el

diario de eventos que deben tener los controladores que se comunican utilizando Modbus y grabar los datos de configuración del procesador de comunicaciones. En el capítulo 3.6 se describe en detalle como se utilizó la memoria flash.

La validación del funcionamiento de las rutinas se hizo en primera instancia con el software WindMill ComDebug[13] que permite enviar tramas Modbus desde un PC a un dispositivo conectado al puerto serial. En la aplicación se utilizó el Maestro Modbus[14], que permite monitorizar las tramas que se envían por el bus de comunicaciones.

Las rutinas que se encargan de ejecutar las funciones Modbus fueron utilizadas con pocos cambios en el microcontrolador MC9S12E de Motorola[15], lo que demuestra que las rutinas programadas en lenguaje C gozan de portabilidad que permite utilizarlos en diferentes sistemas.

Finalmente, se ve la posibilidad de implementar un maestro Modbus utilizando un microcontrolador. La forma en que se transmiten y se reciben los datos utilizados en este trabajo servirán para dicha implementación. Sin embargo, para la implementación de un maestro Modbus se deberá considerar los siguientes aspectos:

- Interpretar los datos recibidos en los mensajes. Por ejemplo, cuando el maestro pide el estado de la bobinas 20 a 27 el esclavo envía primero el estado de la bobina 27 y por último el estado de la bobina 20.
- Interpretar y tomar decisiones cuando se presenten estados de excepción o error en los esclavos de la red.
- Conocer el mapa de memoria de los esclavos.
- Utilización de más de un temporizador para el *timeout*.
- Interpretar el diagnóstico enviado por los esclavos.

[13] Software de distribución gratuita. www.windmill.co.uk
[14] Sinle M. Carreño, Pedro Ardila. Monitoreo de la Red Empleado LabVIEW. Proyecto de grado Universidad Industrial de Santander, 2005.
[15] Este microcontrolador se utilizó en el trabajo final de maestría para la implementación del Procesador de Comunicaciones Modbus. Autor: Jorge E. Duque. Universidad Industrial de Santander.

BIBLIOGRAFÍA

- Modicon Modbus Protocol Reference Guide **PI-MBUS-300 Rev. J**, Junio 1996

- Gélvez, Julio A., “Redes de Comunicación Industrial”, Monografía. Universidad Industrial de Santander, 2002.

- Niño, Carlos E. y Becerra, Álvaro B., “Redes Industriales, Modbus y Ethernet Implementados en Autómatas Programables Trilogi, Koyo y Modicón-Telemecanique”. Monografía, Universidad Industrial de Santander, 2002.

- Motorola, “MC68HC908GP32/H Technical Data Rev. 5” Motorola Inc, 2001

- Stuart Robb, “Creatting Efficient C Code for the MC68HC08”, Application Note AN2093. Motorola Semiconductor, 2001.

- Duque, Jorge E., “Procesador de Comunicación Modbus”. Monografía, Universidad Industrial de Santander, 2005.

- Ardila, Pedro y Carreño, Sinle M., ”Modbus. Monitoreo de la red utilizando LabView”. Monografía, Universidad Industrial de Santander, 2005.

- Helbert Schildt “C Guía Para Usuarios Expertos”, Osborne/McGraw Hill, 1991.

- Forouzan , Behrouz A., “Transmisión de Datos y Redes de Comunicación”, Segunda edición; McGraw Hill, 2002