ISPC INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

ARQUITECTURA Y CONECTIVIDAD

Tecnicatura Superior en Telecomunicaciones. **Año 2025**
Profesor Ing. Jorge Morales.
Grupo 2 - Alumna: Emma Gutiérrez

Eje 3- Actividad Nro. 5

Protocolos Industriales

PROTOCOLO MODBUS

Descripción detallada:

MODBUS es un protocolo de comunicación de capa de aplicación que se utiliza principalmente en sistemas de automatización industrial. Fue desarrollado originalmente por Modicon para facilitar la comunicación entre controladores lógicos programables (PLC) y dispositivos periféricos, como sensores y actuadores. Existen varias variantes de **MODBUS: MODBUS RTU (**para comunicación serial), **MODBUS ASCII, y MODBUS TCP** (que usa **TCP/IP** para la comunicación sobre Ethernet). **MODBUS** es ampliamente utilizado en aplicaciones donde se requieren tiempos de respuesta rápidos y comunicaciones de bajo costo.

Modbus es un protocolo de comunicación utilizado en la automatización industrial para permitir la comunicación entre dispositivos como los controladores lógicos programables (PLC). Opera en una configuración maestro-esclavo, donde el maestro consulta y controla múltiples dispositivos esclavos. El puerto 502 se utiliza habitualmente para Modbus TCP/IP, lo que lo convierte en un objetivo clave para las pruebas de penetración y la detección de vulnerabilidades de seguridad.

Características clave:

Simplicidad: La estructura de los mensajes es muy sencilla, lo que facilita su implementación en diversos dispositivos.
Maestro-esclavo: El maestro solicita datos a los esclavos y estos responden con la información solicitada.
Versatilidad: Compatible con diversas formas de comunicación, como serial (RS-232, RS-485) y Ethernet (MODBUS TCP).
Uso en SCADA: Es utilizado por sistemas SCADA para la supervisión y control de dispositivos en tiempo real.

Flujo de trabajo

Buscar dispositivos: utilizar Nmap para encontrar dispositivos que escuchen en el **puerto 502.**
Tomar huellas digitales del sistema: utilizar herramientas como el módulo **Modbusdetect de Metasploi**t para conocer la versión y las capacidades de Modbus.
Comprobar características de seguridad: pruebe si el sistema requiere autenticación, ya que muchas implementaciones de Modbus no lo hacen, lo que permite que cualquiera pueda emitir comandos.
Códigos de función de prueba: explorar los códigos de función **Modbus** (como 0x01 para bobinas de lectura) para ver si pueden leer o escribir datos confidenciales.
Buscar exploits: verificar si hay desbordamientos de búfer enviando solicitudes de gran tamaño y pruebe ataques de intermediario, dado que el tráfico Modbus a menudo no está cifrado.

Falta de cifrado

Un punto interesante es que la comunicación Modbus normalmente no está cifrada, lo que significa que un atacante puede interceptar y modificar mensajes fácilmente, aumentando el riesgo en entornos industriales.

Buscando dispositivos

Comenzar usando Nmap para buscar dispositivos Modbus en el puerto 502. Ejecute este comando:
nmap -p 502 --script modbus-discover <RANGO_IP>p 502 --script modbus-discover <RANGO_IP>
Esto ayudará a identificar dispositivos y recopilar información inicial.

Reconocimiento inicial

Utilizar Metasploit para un reconocimiento más profundo. Primero, detecte el servicio Modbus:
msf > usar auxiliar/escáner/scada/modbusdetect
msf auxiliar(modbusdetect) > establecer RHOSTS <DIRECCIÓN IP>establecer RHOSTS <DIRECCIÓN IP>
msf auxiliar(modbusdetect) > ejecutar
Luego, se enumera los ID de las unidades:
msf > usar auxiliar/escáner/scada/modbus_findunitid
msf auxiliar(modbus_findunitid) > establecer RHOSTS <DIRECCIÓN IP>establecer RHOSTS <DIRECCIÓN_IP>
msf auxiliar(modbus_findunitid) > ejecutar

Interactuando con dispositivos

Instala y usa Smod para una interacción detallada. Clónalo y ejecútalo:
clon git https://github.com/enddo/smod
cd smod
python smod.py

Conectarse al dispositivo:

conectar -ip <DIRECCIÓN IP> -puerto 502< DIRECCIÓN IP > -puerto 502

Enumerar códigos de función y leer/escribir registros, por ejemplo:

enum_ func lectura_registro_de_retención - addr 0 - cuenta 1

escritura_registro_de_retención - addr 0 - valor 100

Explotación de vulnerabilidades

Comprobar si hay desbordamientos de búfer enviando solicitudes de gran tamaño en Smod. Para ataques de intermediario, capture el tráfico con Wireshark:

wireshark -i <INTERFAZ>i <INTERFAZ>

Luego, usar Scapy para modificar y reenviar paquetes:

```
de scapy. todo  importa *
modbus_packet = Ether() / IP() / TCP() / Raw(carga= '<datos_modificados>' )
sendp(modbus_packet)
```

Modbus: una puerta de entrada a la comunicación industrial

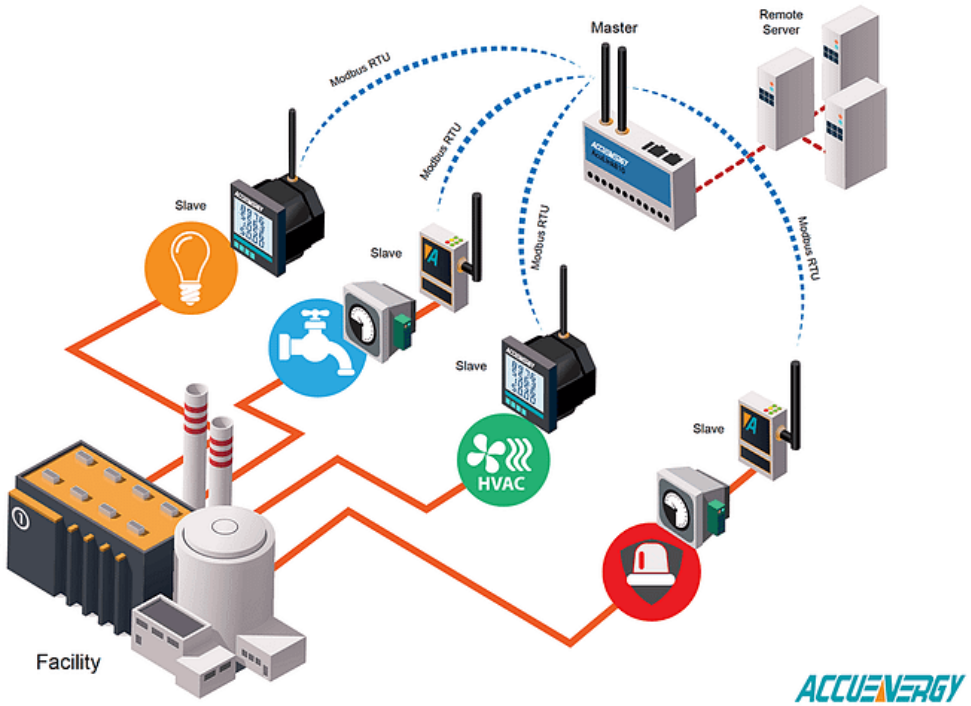
cakpolat

Seguir

5 minutos de lectura

18 de febrero de 2025

El protocolo de comunicación actual proviene del mundo de la comunicación industrial, un recurso indispensable durante décadas: **Modbus** . Este protocolo se implementa ampliamente en fábricas, centrales eléctricas e incluso en hogares inteligentes. En esta guía, presentamos los fundamentos de Modbus, exploramos algunas aplicaciones reales y, a continuación, presentamos una demostración con un ejemplo maestro-esclavo.



https://www.accuenergy.com/support/reference-directory/wireless-modbus/

Cubriremos los temas a continuación:

1. **Introducción al protocolo Modbus:** qué es, por qué es importante y los conceptos fundamentales.
2. **Casos de uso:** dónde destaca Modbus y por qué se inventó.
3. **Una implementación cliente-servidor Modbus:** examen de la estructura del código y las funciones clave, mostrando la aplicación práctica de los principios Modbus.

1. Comprensión de Modbus: el lenguaje de la automatización

Modbus es un protocolo de comunicación serie publicado originalmente por Modicon (ahora Schneider Electric) en 1979. Debido a las dificultades de los PLC entre los diferentes proveedores, Modbus se convirtió en una forma estandarizada para que estos dispositivos intercambiaran información, independientemente del fabricante. Su atractivo reside en su simplicidad y naturaleza abierta, lo que propició su amplio uso en diversas industrias.

Caracteres clave de Modbus:

- **Arquitectura Maestro-Esclavo (Cliente-Servidor):** Modbus funciona según el principio de solicitud-respuesta. Un dispositivo **maestro** (o **cliente**) envía una solicitud a un dispositivo **esclavo** (o **servidor**), y este responde con los datos solicitados o ejecuta el comando. Solo el maestro puede iniciar la comunicación.
- **Comunicación en serie:** Originalmente diseñado para líneas seriales como RS-232 y RS-485, Modbus ha evolucionado para funcionar en redes TCP/IP (Modbus TCP).
- **Organización de datos:** Modbus define cuatro tablas de datos principales:

1. Bobinas (salidas discretas): datos de lectura/escritura de un solo bit, que se utilizan normalmente para controlar los estados de ***encendido/apagado*** *de los dispositivos* (p. ej., encender/apagar un motor).

2. Entradas discretas: datos de solo lectura de un solo bit, que se utilizan para *representar el* ***estado*** *de los sensores* (p. ej., un interruptor de límite está abierto/cerrado).

3. Registros de retención: datos de lectura/escritura de 16 bits, que se utilizan para *almacenar parámetros de configuración o valores numéricos* (p. ej., un punto de ajuste de temperatura).

4. Registros de entrada: datos de solo lectura de 16 bits, que se utilizan para representar valores de entrada analógicos (p. ej., una lectura de temperatura).

5. Códigos de función: los comandos específicos se definen utilizando códigos de función, que indican al dispositivo esclavo qué acción realizar (p. ej., leer registros de retención, escribir una sola bobina).

2. Modbus en acción: dónde y por qué se utiliza

Modbus se utiliza en casi todas partes del mundo. Su simplicidad, baja sobrecarga y facilidad de implementación lo han consolidado en diversas aplicaciones, entre ellas:

- **Automatización industrial:** control de procesos de fabricación, supervisión del estado de los equipos y recopilación de datos de sensores.
- **Automatización de edificios:** gestión de sistemas HVAC, controles de iluminación y sistemas de seguridad.
- **Gestión energética:** monitorización del consumo energético, control de fuentes de energía renovables (solar, eólica) e integración con redes inteligentes.
- **Transporte:** Control de señales de tráfico, monitoreo de sistemas ferroviarios y gestión de operaciones de flotas.

- **Y mucho más:** en cualquier lugar donde necesite una comunicación simple y confiable entre dispositivos, Modbus es un fuerte competidor.

3. Demostración práctica: Modbus maestro-esclavo en Python

Para demostrar el funcionamiento de Modbus, se implementó el proyecto modbussim . Puede clonarlo y probarlo en su entorno de desarrollo.

El repositorio del proyecto proporciona una implementación cliente-servidor Modbus escrita en Python, junto con una **API Flask** para interactuar con el servidor Modbus a través de solicitudes HTTP.

- main.py:Este es el punto de entrada de la aplicación. Inicia el servidor Modbus y la API de Flask en subprocesos separados, lo que permite su ejecución simultánea. También incluye un bucle principal que realiza llamadas periódicas update_registers()para simular cambios en los datos del sensor.
- server.py: Contiene la implementación del servidor Modbus.
- flask_api.py: Contiene la implementación de la API de Flask.
- client.py: Contiene la implementación del cliente Modbus.

Funciones cruciales y fragmentos de código:

- server.py: start_modbus_server(): Esta función se encarga de configurar e iniciar el servidor Modbus TCP. Utiliza la pymodbusbiblioteca. Observe cómo toma el contextobjeto (que define las tablas de datos Modbus) y lo vincula a un puerto TCP:

```
desde pymodbus.server.async_io importar StartTcpServer
# …
StartTcpServer(contexto=contexto, dirección=(“localhost”, MODBUS_PORT),
identidad=identificación)
```

Esto refleja directamente el concepto Modbus TCP de servir datos a través de una red.

- server.py: update_registers()Esta función simula los datos del sensor y actualiza los bloques de datos Modbus. Muestra cómo se asignan los datos reales (en este caso, temperaturas simuladas) a los registros Modbus.

```
def update_registers():def  update_registers ():
 # …
   temperaturas_actuales, promedios_históricos, estado_de_enfriamiento, estados_de_alarma =
simular_datos_de_temperatura(
  temperaturas_actuales, promedios_históricos
)
 # Actualizar los bloques de datos
 holding_registers.setValues( 0 , temperaturas_actuales) # Datos de temperatura
 coils.setValues( 0 , estado_de_enfriamiento) # Estado de enfriamiento como booleanos
```

La holding_registers.setValues()función es una implementación directa de la escritura de datos en una tabla de datos Modbus específica.

- **client.py: run_modbus_client () Esta función implementa el cliente Modbus. Se conecta al servidor y lee periódicamente datos de varios registros:**

```
de pymodbus.client importar ModbusTcpClient
cliente = ModbusTcpClient(MODBUS_SERVER_IP, puerto=MODBUS_SERVER_PORT)
respuesta = cliente.read_holding_registers(dirección= 0 , conteo= 9 , esclavo= 1 ) #Leer datos
temporales
si respuesta.isError():
logging.info( f"Error al leer los registros de retención: {respuesta} " )
 de lo contrario :
logging.info( f"Temperaturas actuales: {respuesta.registros} " )
```

La client.read_holding_registers() función demuestra la funcionalidad principal del cliente Modbus de solicitar datos utilizando un código de función específico (en este caso, leer registros de retención).

- **flask_api.py: modbus_register ()** Esta función de la API de Flask permite interactuar con el servidor Modbus mediante solicitudes HTTP. Permite obtener y colocar valores en registros Modbus con diferentes tipos de datos.

```
# API REST para interactuar con el servidor Modbus
@app.route( '/modbus/<string:register>/< int :address>', methods=['GET', 'PUT'] )
def  modbus_register ( register, address ):
  try :
    register = register.lower() # Estandarizar caso
     store = get_register(register)
    if request.method == 'PUT' :
      data = request.get_json() # Usar get_json para un análisis más seguro
       decimal_value = data[ "value" ]
      data_type = str (data[ "data_type" ]).upper() # Asegurar el tipo de cadena
       hex_values = convert_to_registers(decimal_value, data_type)
     # Actualizar el/los registro(s) Modbus
      set_register_value(register, address, hex_values)
    log.info( f"Valores actualizados en la dirección {address} : {current_values} " )
```

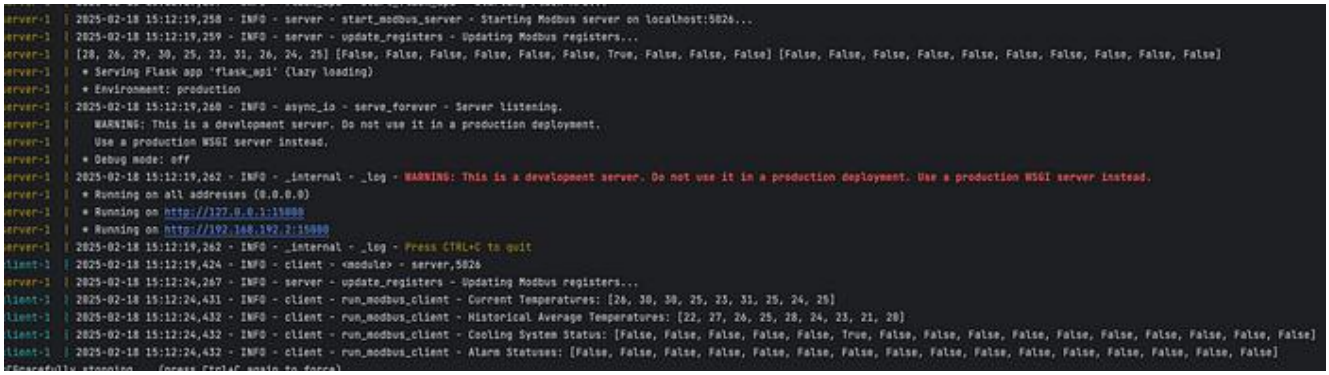
Cómo se conectan la teoría y la implementación:

El código refleja directamente los principios básicos de Modbus:

- **Modelo Cliente-Servidor:** El **client.py** solicita datos activamente, mientras que el server.pysirve pasivamente.
- **Organización de datos**: Los **ModbusSequentialDataBlock**objetos representan las bobinas, las entradas discretas, los registros de retención y los registros de entrada.
- **Códigos de función:** aunque no los hemos definido explícitamente como variables, las client**.read_holding_registers (), client.read_coils ()** funciones, etc., son abstracciones de los códigos de función Modbus.
- **Espacio de direcciones:** el **addressparámetro** en las funciones de lectura y escritura especifica la dirección del registro dentro de la tabla de datos Modbus.

Para ejecutar esta demostración:

1. Clonar el **repositorio:** git clone git@github.com:cemakpolat/iot-simulated-devices.git
2. **cd modbussim**, y luego tienes dos opciones: instalación manual o docker compose
3. Aplique uno de los enfoques definidos en el repositorio de GitHub y los resultados de la consola serán los siguientes:



Para interactuar con la API de Flask y manipular los valores de registro, puede probar los ejemplos a continuación:

- **GET dirección de registro de tenencia 0**

rizo http://localhost:15000/modbus/holding_registers/0

PUT dirección del registro de tenencia 0

curl -X PUT -H "Tipo de contenido: aplicación/json" -d"Tipo de contenido: aplicación/json" -d
'{ "valor" : 1234 , "tipo_de_datos" : "UINT16" }'
http: //localhost:15000/modbus/holding_registers/0

Al examinar el código y ejecutar la demostración, puede obtener una comprensión más profunda de cómo funciona Modbus y cómo se implementan sus principios en el software.

Resumen

Modbus es un pilar de la comunicación industrial, conocido por su simplicidad y robustez. Desde fábricas hasta hogares inteligentes, permite que los dispositivos intercambien datos de forma fiable. Al explorar el repositorio de GitHub, has adquirido una comprensión práctica de cómo los principios de Modbus se traducen en código real. Puedes ampliar tu aprendizaje modificando el código, añadiendo nuevas funciones y experimentando con diferentes configuraciones de Modbus. ¡Que disfrutes programando!

Código fuente : https://github.com/cemakpolat/iot-simulated-devices/modbussim

Referencias:

1. https://www.modbus.org/
2. https://theautomization.com/modbus-ascii-vs-modbus-rtu-vs-modbus-tcpip/

herramientas Modbus

Para pruebas, simulación y programación.

HOGAR PRODUCTOS ORDEN DESCARGAR MODBUS CONTACTO

Descripción del protocolo Modbus

El protocolo MODBUS© es una estructura de mensajería ampliamente utilizada para establecer la comunicación maestro-esclavo entre dispositivos inteligentes. Un mensaje MODBUS enviado de un maestro a un esclavo contiene la dirección del esclavo, el comando (p. ej., "leer registro" o "escribir registro"), los datos y una suma de comprobación (LRC o CRC).

Dado que el protocolo Modbus es simplemente una estructura de mensajería, es independiente de la capa física subyacente. Tradicionalmente, se implementa mediante RS232, RS422 o RS485.

La solicitud.

El código de función en la solicitud indica al dispositivo esclavo destinatario qué tipo de acción realizar. Los bytes de datos contienen cualquier información adicional que el esclavo necesitará para realizar la función. Por ejemplo, el código de función 03 solicitará al esclavo que lea los registros de retención y responda con su contenido. El campo de datos debe contener la información que indica al esclavo en qué registro comenzar y cuántos registros leer. El campo de comprobación de errores proporciona un método para que el esclavo valide la integridad del contenido del mensaje.

La respuesta.

Si el esclavo da una respuesta normal, el código de función en la respuesta es un eco del código de función en la solicitud. Los bytes de datos contienen los datos recopilados por el esclavo, como los valores de registro o el estado. Si se produce un error, el código de función se modifica para indicar que se trata de una respuesta de error, y los bytes de datos contienen un código que describe el error. El campo de comprobación de errores permite al maestro confirmar la validez del contenido del mensaje.

Los controladores se pueden configurar para comunicarse en redes Modbus estándar utilizando cualquiera de dos modos de transmisión: ASCII o RTU.

Modo ASCII.
Cuando los controladores se configuran para comunicarse en una red Modbus mediante el modo ASCII (Código Estándar Americano para el Intercambio de Información), cada byte de ocho bits de un mensaje se envía como dos caracteres ASCII. La principal ventaja de este modo es que permite intervalos de hasta un segundo entre caracteres sin causar errores.
Sistema de codificación
Caracteres hexadecimales ASCII imprimibles 0 ... 9, A ... F
Bits por byte
1 bit de inicio
7 bits de datos, el bit menos significativo se envía primero
1 bit para paridad par/impar - ningún bit si no hay paridad
1 bit de parada si se utiliza paridad - 2 bits si no hay paridad
Comprobación de errores
Comprobación de redundancia longitudinal (LRC)
Modo RTU:
Cuando los controladores se configuran para comunicarse en una red Modbus mediante el modo RTU (Unidad Terminal Remota), cada byte de ocho bits de un mensaje contiene dos caracteres hexadecimales de cuatro bits. La principal ventaja de este modo es que su mayor densidad de caracteres permite un mejor rendimiento de datos que el ASCII para la misma velocidad en baudios. Cada mensaje debe transmitirse en un flujo continuo.
Sistema de codificación
Binario de ocho bits, hexadecimal 0 ... 9, A ... F
Dos caracteres hexadecimales contenidos en cada campo de ocho bits del mensaje
Bits por byte
1 bit de inicio
8 bits de datos, el bit menos significativo se envía primero
1 bit para paridad par/impar - ningún bit si no hay paridad
1 bit de parada si se utiliza paridad - 2 bits si no hay paridad
Campo de comprobación de errores
Comprobación de redundancia cíclica (CRC)
En modo ASCII, los mensajes comienzan con dos puntos (:) (ASCII 3A hexadecimal) y terminan con un par de retorno de carro y avance de línea (CRLF) (ASCII 0D y 0A hexadecimal).
Los caracteres permitidos para todos los demás campos son hexadecimales 0 ... 9, A ... F. Los dispositivos en red monitorean el bus de red continuamente para detectar el carácter de dos puntos. Al recibir uno, cada dispositivo decodifica el siguiente campo (el campo de dirección) para determinar si corresponde al dispositivo al que se dirige.
Pueden transcurrir intervalos de hasta un segundo entre caracteres dentro del mensaje. Si el intervalo es mayor, el dispositivo receptor asume que se ha producido un error. A continuación se muestra una trama de mensaje típica.

Comenzar	DIRECCIÓN	Función	Datos	LRC	Fin
:	2 caracteres	2 caracteres	N caracteres	2 caracteres	CR LF

Entramado RTU
En el modo RTU, los mensajes comienzan con un intervalo de silencio de al menos 3,5 veces el carácter. Esto se implementa más fácilmente como un múltiplo de veces el carácter a la velocidad en baudios que se utiliza en la red (mostrado como T1-T2-T3-T4 en la figura siguiente). El primer campo que se transmite es la dirección del dispositivo.
Los caracteres permitidos transmitidos para todos los campos son hexadecimales 0 ... 9, A ... F. Los dispositivos en red monitorean el bus de red continuamente, incluso durante los intervalos de silencio. Cuando se recibe el primer campo (el campo de dirección), cada dispositivo lo decodifica para averiguar si es el dispositivo direccionado.
Después del último carácter transmitido, un intervalo similar de al menos 3,5 veces el carácter marca el final del mensaje. Un nuevo mensaje puede comenzar después de este intervalo.
La trama completa del mensaje debe transmitirse como un flujo continuo. Si ocurre un intervalo de silencio de más de 1,5 veces el carácter antes de que se complete la trama, el dispositivo receptor vacía el mensaje incompleto y asume que el siguiente byte será el campo de dirección de un nuevo mensaje.

De igual forma, si un nuevo mensaje comienza más de 3,5 caracteres después de un mensaje anterior, el dispositivo receptor lo considerará una continuación del mensaje anterior. Esto generará un error, ya que el valor del campo CRC final no será válido para los mensajes combinados. A continuación se muestra un marco de mensaje típico.

Comenzar	DIRECCIÓN	Función	Datos	CRC	Fin
3,5 Tiempo de char	8 bits	8 bits	N * 8 bits	16 bits	3,5 Tiempo de char

Campo de dirección:
El campo de dirección de una trama de mensaje contiene dos caracteres (ASCII) u ocho bits (RTU). A cada dispositivo esclavo se le asignan direcciones en el rango de 1 a 247.

Campo de función
El campo Código de Función indica al esclavo direccionado la función que debe realizar.
Las siguientes funciones son compatibles con Modbus Poll.
01 (0x01) Leer bobinas
02 (0x02) Leer entradas discretas
03 (0x03) Leer registros de retención
04 (0x04) Leer registros de entrada
05 (0x05) Escribir bobina única
06 (0x06) Escribir registro único
08 (0x08) Diagnóstico (solo línea serie)
11 (0x0B) Obtener contador de eventos de comunicación (solo línea serie)
15 (0x0F) Escribir varias bobinas
16 (0x10) Escribir varios registros
17 (0x11) ID de servidor de informes (solo línea serie)
22 (0x16) Enmascarar registro de escritura
23 (0x17) Leer/escribir varios registros
43 / 14 (0x2B / 0x0E) Leer identificación del dispositivo
El campo de datos contiene los datos solicitados o enviados.

Contenido del campo de comprobación de errores.
Se utilizan dos tipos de métodos de comprobación de errores para las redes Modbus estándar. El contenido del campo de comprobación de errores depende del método utilizado.

ASCII:
Cuando se utiliza el modo ASCII para la estructura de caracteres, el campo de comprobación de errores contiene dos caracteres ASCII. Estos caracteres son el resultado de un cálculo de Comprobación de Redundancia Longitudinal (LRC) realizado sobre el contenido del mensaje, excluyendo los dos puntos iniciales y los caracteres CRLF finales.
Los caracteres LRC se añaden al mensaje como el último campo que precede a los caracteres CRLF.
Código de ejemplo LRC.

RTU:
Cuando se utiliza el modo RTU para la estructuración de caracteres, el campo de comprobación de errores contiene un valor de 16 bits implementado como dos bytes de ocho bits. El valor de comprobación de errores es el resultado de un cálculo de comprobación de redundancia cíclica (CRC) realizado sobre el contenido del mensaje.
El campo CRC se añade al mensaje como último campo. En este caso, se añade primero el byte de orden inferior del campo, seguido del byte de orden superior. El byte de orden superior de CRC es el último byte que se envía en el mensaje.

Código de ejemplo de CRC.

Función 01 (01hex) Leer bobinas

Lee el estado de encendido/apagado de bobinas discretas en el esclavo.

Solicitud

El mensaje de solicitud especifica la bobina inicial y la cantidad de bobinas a leer.

Ejemplo de una solicitud para leer 13 direcciones de bobinas 10...22 (bobinas 11 a 23) desde la dirección del dispositivo esclavo 4:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	01	0 1
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	0A	0 A
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0D	0 días
Comprobación de errores Lo	DD	LRC (E 4)
Comprobación de errores Hola	98	
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

El mensaje de respuesta de estado de la bobina se empaqueta como una bobina por bit del campo de datos. El estado se indica como: 1 es el valor ON y 0 es el valor OFF. El LSB del primer byte de datos contiene la bobina a la que se dirige la solicitud. Las demás bobinas se dirigen hacia el extremo de orden superior de este byte y, de orden inferior a orden superior, en los bytes subsiguientes. Si la cantidad de bobinas devuelta no es múltiplo de ocho, los bits restantes del último byte de datos se rellenarán con ceros (hacia el extremo de orden superior del byte). El campo de recuento de bytes especifica la cantidad de bytes de datos completos.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	01	0 1
Recuento de bytes	02	0 2
Datos (Bobinas 18...11)	0A	0 A
Datos (Bobinas 23...19)	11	1 1
Comprobación de errores Lo	B3	LRC (DE)
Comprobación de errores Hola	50	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	7	15

Función 02(02hex) Leer entradas discretas

Lee el estado ON/OFF de las entradas discretas en el esclavo.

Solicitud

El mensaje de solicitud especifica la entrada inicial y la cantidad de entradas a leer.
Ejemplo de una solicitud para leer 13 entradas dirección 10...22 (entradas 10011 a 10023) desde la dirección del dispositivo esclavo 4:

Nombre del campo	**RTU (hexadecimal)**	**Caracteres ASCII**
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	02	0 2
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	0A	0 A
Cantidad de insumos Hola	00	0 0
Cantidad de insumos Lo	0D	0 días
Comprobación de errores Lo	99	LRC (E 3)
Comprobación de errores Hola	98	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

El mensaje de respuesta de estado de entrada se empaqueta como una entrada por bit del campo de datos. El estado se indica como: 1 es el valor ON y 0 es el valor OFF. El LSB del primer byte de datos contiene la entrada a la que se dirige la solicitud. Las demás entradas se dirigen hacia el extremo de orden superior de este byte y, de orden inferior a orden superior, en los bytes subsiguientes. Si la cantidad de entrada devuelta no es múltiplo de ocho, los bits restantes del último byte de datos se rellenarán con ceros (hacia el extremo de orden superior del byte). El campo de recuento de bytes especifica la cantidad de bytes de datos completos.

Ejemplo de respuesta a la solicitud:

Nombre del campo	**RTU (hexadecimal)**	**Caracteres ASCII**
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	04	0 4
Función	02	0 2
Recuento de bytes	02	0 2
Datos (Entradas 18...11)	0A	0 A
Datos (Entradas 23...19)	11	1 1
Comprobación de errores Lo	B3	LRC (DD)
Comprobación de errores Hola	14	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	7	15

Función 03 (03hex) Leer registros de retención

Leer el contenido binario de los registros de retención en el esclavo.

Solicitud

El mensaje de solicitud especifica el registro inicial y la cantidad de registros que se leerán.

Ejemplo de una solicitud para leer 0...1 (registros 40001 a 40002) desde el dispositivo esclavo 1:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	03	0 3
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	00	0 0
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	C4	LRC (FA)
Comprobación de errores Hola	0B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:

Los datos de registro en el mensaje de respuesta se empaquetan en dos bytes por registro, con el contenido binario justificado a la derecha dentro de cada byte. Para cada registro, el primer byte contiene los bits de orden superior y el segundo, los de orden inferior.

Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	03	0 3
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	06	0 6
Hola datos	00	0 0
Datos Lo	05	0 5
Comprobación de errores Lo	DA	LRC (ED)
Comprobación de errores Hola	31	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	9	19

Función 04 (04hex) Leer registros de entrada
Leer el contenido binario de los registros de entrada en el esclavo.
Solicitud
El mensaje de solicitud especifica el registro inicial y la cantidad de registros que se leerán.
Ejemplo de una solicitud para leer 0...1 (registros 30001 a 30002) desde el dispositivo esclavo 1:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	04	0 4
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	00	0 0
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	71	LRC (F 9)
Comprobación de errores Hola	CB	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:
Los datos de registro en el mensaje de respuesta se empaquetan en dos bytes por registro, con el contenido binario justificado a la derecha dentro de cada byte. Para cada registro, el primer byte contiene los bits de orden superior y el segundo, los de orden inferior.
Ejemplo de respuesta a la solicitud:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	01	0 1
Función	04	0 4
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	06	0 6
Hola datos	00	0 0
Datos Lo	05	0 5
Comprobación de errores Lo	Base de datos	LRC (CE)
Comprobación de errores Hola	86	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	9	19

Función 05 (05hex) Escribir bobina simple
Escribe una sola bobina en ON o OFF.
Solicitud:
El mensaje de solicitud especifica la referencia de la bobina que se escribirá. Las bobinas se direccionan desde cero; la bobina 1 se direcciona como 0.

El estado de encendido/apagado solicitado se especifica mediante una constante en el campo de datos de solicitud. Un valor de FF 00 hexadecimal indica que la bobina está encendida. Un valor de 00 00 indica que está apagada. Todos los demás valores son ilegales y no afectan a la bobina.

A continuación se muestra un ejemplo de una solicitud para escribir la bobina 173 ON en el dispositivo esclavo 17:

Nombre del campo	**RTU (hexadecimal)**	**Caracteres ASCII**
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	05	0 5
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	C.A.	C.A.
Escribir datos Hola	FF	0 0
Escribir datos Lo	00	FF
Comprobación de errores Lo	4E	LRC (3 pisos)
Comprobación de errores Hola	8B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta:
La respuesta normal es un eco de la solicitud, que se devuelve después de escribir el estado de la bobina.
Ejemplo de respuesta a la solicitud:

Nombre del campo	**RTU (hexadecimal)**	**Caracteres ASCII**
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	05	0 5
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	C.A.	C.A.
Escribir datos Hola	FF	0 0
Escribir datos Lo	00	FF
Comprobación de errores Lo	4E	LRC (3 pisos)
Comprobación de errores Hola	8B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 06 (06hex) Escribir un solo registro

Escribe un valor en un único registro de retención.

Solicitud:

El mensaje de solicitud especifica la referencia del registro que se escribirá. Los registros se direccionan desde cero; el registro 1 se direcciona como 0.

El valor de escritura solicitado se especifica en el campo de datos de la solicitud. A continuación, se muestra un ejemplo de una solicitud de escritura del registro 40002 al 00 03 hexadecimal en el dispositivo esclavo 17.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	06	0 6
Registrar dirección Hola	00	0 0
Registrar Dirección Lo	01	0 1
Escribir datos Hola	00	0 0
Escribir datos Lo	03	0 3
Comprobación de errores Lo	9A	LRC (E 5)
Comprobación de errores Hola	9B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Respuesta

La respuesta normal es un eco de la solicitud, devuelta después de que se hayan escrito los contenidos del registro.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	06	0 6
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	01	0 1
Escribir datos Hola	00	0 0
Escribir datos Lo	03	0 3
Comprobación de errores Lo	9A	LRC (E 5)
Comprobación de errores Hola	9B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 15 (0Fhex) Escribir múltiples bobinas Escribe cada bobina en una secuencia de bobinas en ON o OFF.

Solicitud: El mensaje de solicitud especifica las referencias de bobina que se escribirán. Las bobinas se direccionan desde cero; la bobina 1 se direcciona como 0. Los estados de encendido/apagado solicitados se especifican mediante el contenido del campo de datos de solicitud. Un 1 lógico en una posición de bit del campo solicita que las bobinas correspondientes estén encendidas. Un 0 lógico solicita que estén apagadas. A continuación se muestra un ejemplo de una solicitud para escribir una serie de diez bobinas comenzando en la bobina 20 (direccionada como 19, o 13 hexadecimal) en el dispositivo esclavo 17. El contenido de los datos de la solicitud consta de dos bytes: CD 01 hexadecimal (1100 1101 0000 0001 binario). Los bits binarios corresponden a las bobinas de la siguiente manera:

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 hexadecimal) se dirige a las bobinas 27... 20, y el bit menos significativo se dirige a la bobina más baja (20) de este conjunto. El siguiente byte transmitido (01 hexadecimal) direcciona las bobinas 29 y 28, y el bit menos significativo direcciona la bobina más baja (28) de este conjunto. Los bits no utilizados del último byte de datos deben rellenarse con ceros.

Nombre del campo	**RTU (hexadecimal)**	**Caracteres ASCII**
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	0F	0 F
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	13	1 3
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0A	0 A
Recuento de bytes	02	0 2
Escribir datos Hola	CD	CD
Escribir datos Lo	01	0 1
Comprobación de errores Lo	BF	LRC (F 3)
Comprobación de errores Hola	0B	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	11	23

Respuesta: La respuesta normal devuelve la dirección del esclavo, el código de función, la dirección inicial y el número de bobinas escritas. A continuación, se muestra un ejemplo de respuesta a la solicitud mostrada arriba.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	0F	0 F
Dirección de la bobina Hola	00	0 0
Dirección de bobina Lo	13	1 3
Cantidad de bobinas Hola	00	0 0
Cantidad de bobinas Lo	0A	0 A
Comprobación de errores Lo	26	LRC (C 3)
Comprobación de errores Hola	99	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Función 16 (10hex) Escribir múltiples registros
Escribe valores en una secuencia de registros de retención

Solicitud: El mensaje de solicitud especifica las referencias de registro que se escribirán. Los registros se direccionan desde cero; el registro 1 se direcciona como 0.

Los valores de escritura solicitados se especifican en el campo de datos de la solicitud. Los datos se empaquetan en dos bytes por registro. A continuación se muestra un ejemplo de una solicitud para escribir dos registros que comienzan en 40002 hasta 00 0A y 01 02 hexadecimal, en el dispositivo esclavo 17:

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	10	1 0
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	01	0 1
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Recuento de bytes	04	0 4
Hola datos	00	0 0
Datos Lo	0A	0 A
Hola datos	01	0 1
Datos Lo	02	0 2
Comprobación de errores Lo	C6	LRC (CB)
Comprobación de errores Hola	F0	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	13	23

Respuesta: La respuesta normal devuelve la dirección del esclavo, el código de función, la dirección inicial y la cantidad de registros escritos. A continuación, se muestra un ejemplo de respuesta a la solicitud mostrada anteriormente.

Nombre del campo	RTU (hexadecimal)	Caracteres ASCII
Encabezamiento	Ninguno	: (Dos puntos)
Dirección de esclavo	11	1 1
Función	10	1 0
Dirección de inicio Hola	00	0 0
Dirección de inicio Lo	01	0 1
Cantidad de registros Hola	00	0 0
Cantidad de registros Lo	02	0 2
Comprobación de errores Lo	12	LRC (DC)
Comprobación de errores Hola	98	Ninguno
Tráiler	Ninguno	CR LF
Bytes totales	8	17

Código de ejemplo LRC Esta función es un ejemplo de cómo calcular un BYTE LRC usando el lenguaje C.

```
BYTE LRC (BYTE *nData, WORD wLength)
{
BYTE nLRC = 0 ; // LRC char initialized
for (int i = 0; i < wLength; i++)
nLRC += *nData++;
return (BYTE)(-nLRC);
} // End: LRC
```

Código de ejemplo CRC

Esta función es un ejemplo de cómo calcular una palabra CRC usando el lenguaje C.

Explicador

```
WORD CRC16 (const BYTE *nData, WORD wLength)
{
static const WORD wCRCTable[] = {
   0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
   0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
   0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
   0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
   0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
   0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
   0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
   0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
   0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
   0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
   0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
   0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
   0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
   0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
   0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
   0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
   0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
   0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
   0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
   0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
   0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
   0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
   0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
   0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
   0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
   0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
   0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
   0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
   0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
   0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
   0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
   0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040 };
BYTE nTemp;
WORD wCRCWord = 0xFFFF;
   while (wLength--)
   {
      nTemp = *nData++ ^ wCRCWord;
      wCRCWord >>= 8;
      wCRCWord  ^= wCRCTable[nTemp];
   }
   return wCRCWord;
} // Fin: CRC16
```

Comparación de MODBUS con otros protocolos de distribución de datos

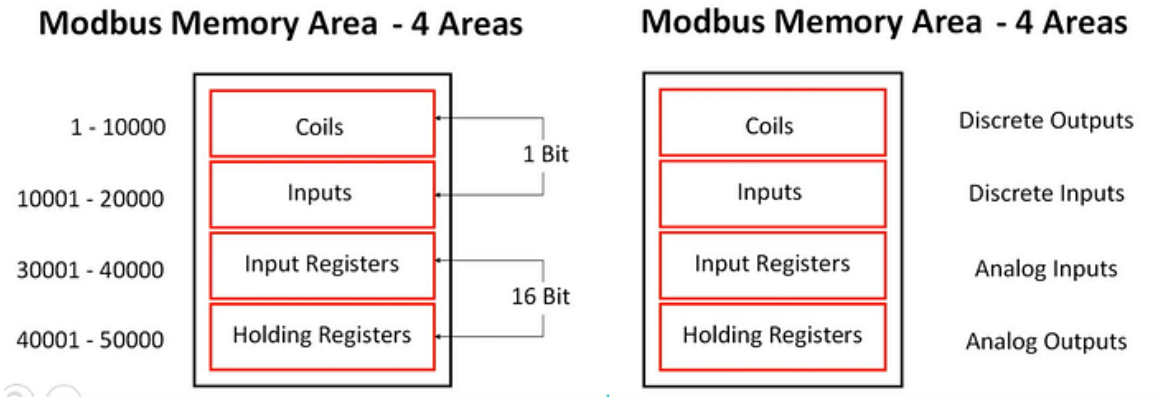
Característica	MODBUS	PROFIBUS	PROFINET	CANopen
Año de Creación	1979	1989	2003	1995
Capa Física	RS-232 / RS-485 / TCP/IP	RS-485	Ethernet Industrial	Bus CAN
Topología	Línea / Bus / Estrella	Línea Bus	Estrella, Línea, Árbol	Línea Bus
Velocidad de Comunicación	Hasta 10 Mbps (TCP)	Hasta 12 Mbps	100 Mbps - 1 Gbps	1 Mbps
Tipo de Comunicación	Maestro-Esclavo	Maestro-Esclavo	Tiempo Real, Cíclico	Maestro-Esclavo
Facilidad de Implementación	Muy Alta	Media	Media-Alta	Alta
Determinismo	Bajo (TCP), Medio (RTU)	Alto	Muy Alto (Tiempo Real)	Medio-Alto
Robustez	Media	Alta	Muy Alta	Alta
Coste de Implementación	Bajo	Medio-Alto	Alto	Bajo-Medio
Aplicaciones Típicas	Automatización simple, SCADA	Automatización industrial	Automatización avanzada, Robots	Vehículos, maquinaria médica
Compatibilidad	Muy alta	Alta	Alta	Alta

Un breve recorrido por Modbus con diagramas
Tablas y diagramas de flujo de Modbus

Yu-Cheng (Morton) Kuo
3 minutos de lectura ·
18 de noviembre de 2023

Modbus es un protocolo de comunicación comúnmente utilizado en sistemas de automatización y control industrial. Aquí, utilizamos algunos diagramas para explicarle rápidamente sus conceptos principales.
Descripción
(1) Bloques del modelo de datos/áreas de memoria
(2) Diagramas de flujo de Modbus maestro y esclavo
(3) Desglose de Modbus en el contexto del modelo de cinco capas TCP/IP
(4) Enmarcado y desenmarcado en Modbus
(5) Códigos de función Modbus
(6) Trama de datos de comandos y respuestas Modbus
(7) CRC
(8) Referencias

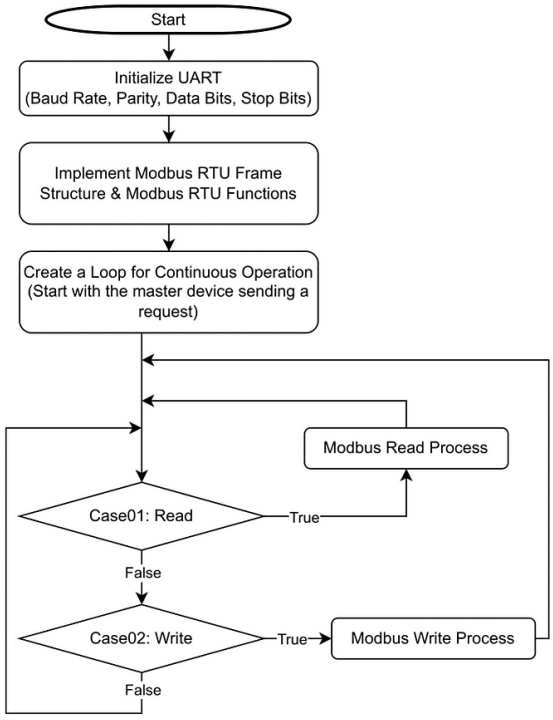
(1) Bloques del modelo de datos/Áreas de memoria

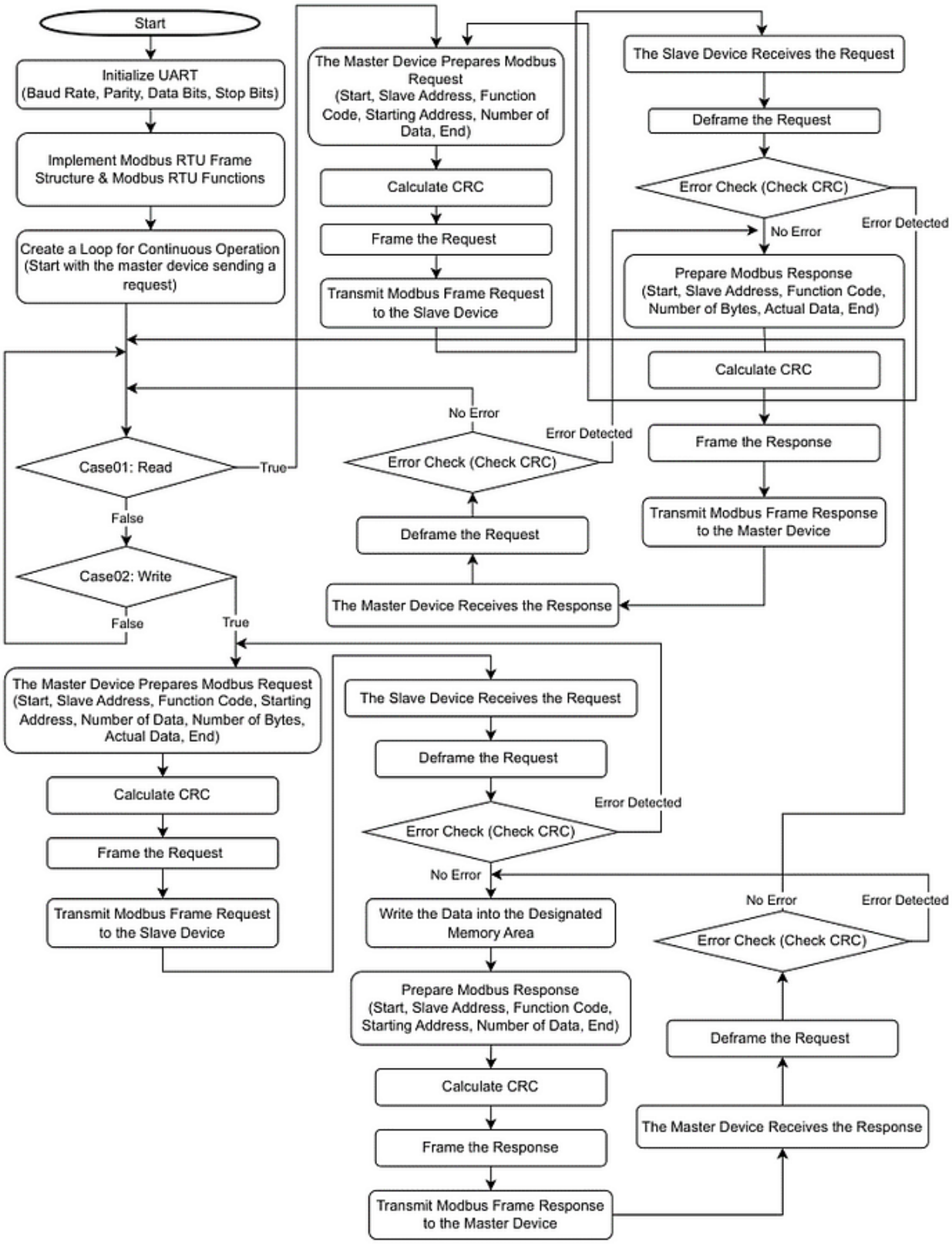


Memory Block	Data Type	Master Access	Slave Access
Coils	Boolean	Read/Write	Read/Write
Discrete Inputs	Boolean	Read-only	Read/Write
Holding Registers	Unsigned Word	Read/Write	Read/Write
Input Registers	Unsigned Word	Read-only	Read/Write

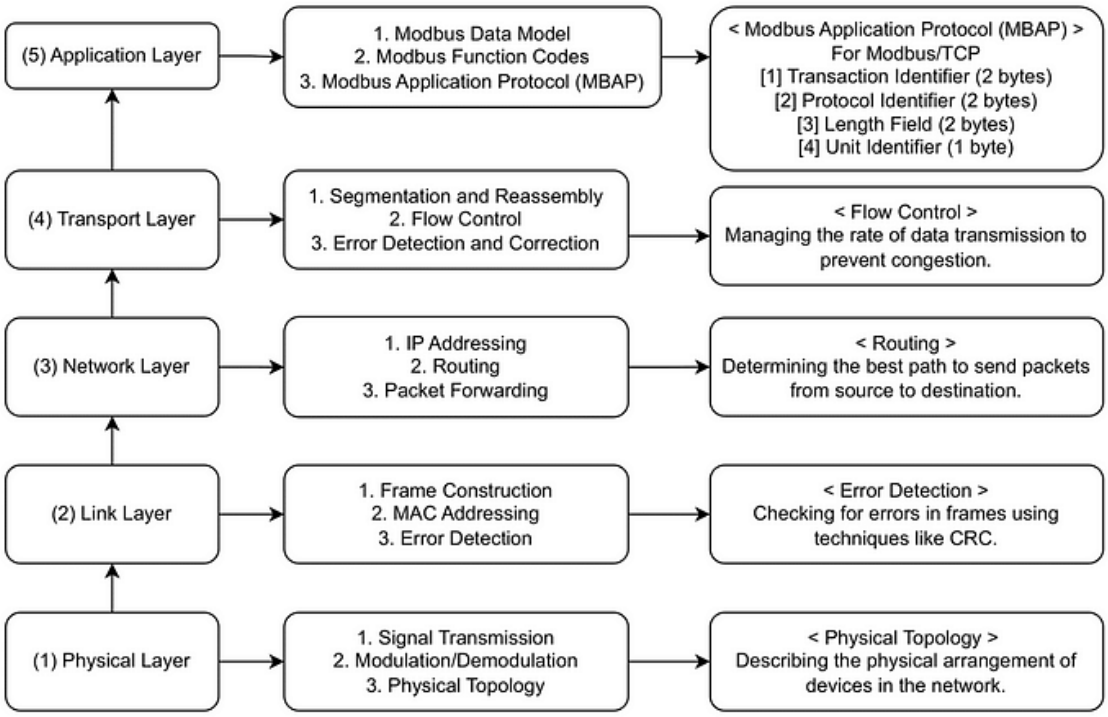
	Coils	Inputs	Input Registers	Holding Registers
Function	Discrete output	Discrete input	Analog input	Analog output
Address	1 ~ 10000	10001 ~ 20000	30001 ~ 40000	40001 ~ 50000
Size	1 bit	1 bit	16 bits	16 bits
R / W	R / W	R	R	R / W

(2) Diagramas de flujo del Modbus Maestro y Esclavo

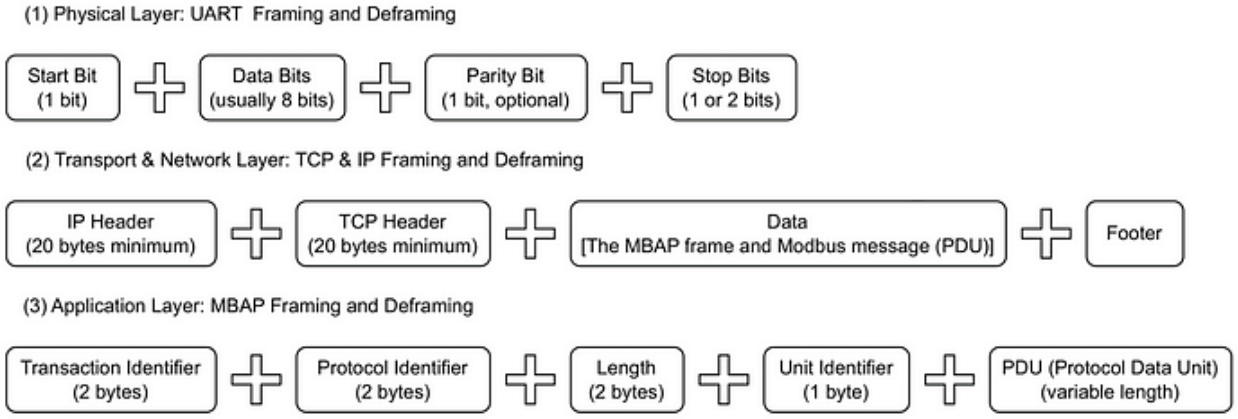



Start
Initialize UART (Baud Rate, Parity, Data Bits, Stop Bits)
Implement Modbus RTU Frame Structure & Modbus RTU Functions
Create a Loop for Continuous Operation (Start with the master device sending a request)
Case01: Read
True
False
Case02: Write
False
True
The Master Device Prepares Modbus Request (Start, Slave Address, Function Code, Starting Address, Number of Data, End)
Calculate CRC
Frame the Request
Transmit Modbus Frame Request to the Slave Device
The Slave Device Receives the Request
Deframe the Request
Error Check (Check CRC)
Error Detected
No Error
Prepare Modbus Response (Start, Slave Address, Function Code, Number of Bytes, Actual Data, End)
Calculate CRC
Frame the Response
Transmit Modbus Frame Response to the Master Device
The Master Device Receives the Response
Deframe the Request
Error Check (Check CRC)
Error Detected
No Error
The Master Device Prepares Modbus Request (Start, Slave Address, Function Code, Starting Address, Number of Data, Number of Bytes, Actual Data, End)
Calculate CRC
Frame the Request
Transmit Modbus Frame Request to the Slave Device
The Slave Device Receives the Request
Deframe the Request
Error Check (Check CRC)
Error Detected
No Error
Write the Data into the Designated Memory Area
Prepare Modbus Response (Start, Slave Address, Function Code, Starting Address, Number of Data, End)
Calculate CRC
Frame the Response
Transmit Modbus Frame Response to the Master Device
Error Check (Check CRC)
No Error
Error Detected
Deframe the Request
The Master Device Receives the Response

(3) Falla de Modbus en el contexto del modelo de cinco capas TCP/IP



(4) Enmarcado y desenmarcado en Modbus



(5) Códigos de función Modbus

Function Codes Description

Modbus Addressing Model

Decimal	Hexadecimal	Description
01	0x01	Read Coil Status
02	0x02	Read Input Status
03	0x03	Read Holding Registers
04	0x04	Read Internal Registers
05	0x05	Force Single Coil
06	0x06	Preset Single Register
15	0x0F	Force Multiple Coils
16	0x10	Preset Multiple Registers
22	0x16	Masked Write Register

Code	1/16-bit	Description	I/O Range
01	1-bit	Read coils	00001 – 10000
02	1-bit	Read contacts	10001 – 20000
05	1-bit	Write a single coil	00001 – 10000
15	1-bit	Write mulitple coils	00001 – 10000
03	16-bit	Read holding registers	40001 – 50000
04	16-bit	Read input registers	30001 – 40000
06	16-bit	Write single register	40001 – 50000
16	16-bit	Write mulitple registers	40001 – 50000
22	16-bit	Mask write register	40001 – 50000
23	16-bit	Read/write mulitple registers	40001 – 50000
24	16-bit	Read FIFO queue	40001 – 50000

Modbus Function Codes

Code	Function
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
07	Read Exception Status
08	Diagnostics
09	Program 484
10	Poll 484
11	Fetch Comm Event Counter
12	Fetch Comm Event Log
13	Program Controller
14	Poll Controller
15	Force Multiple Coils
16	Preset Multiple Registers
17	Report Slave ID
18	Program 884/M84
19	Reset Comm. Link
20	Read General Reference
21	Write General Reference
22	Mask Write 4x Registers
23	Read/Write 4x Registers
24	Read FIFO Queue

Function Code (Decimal)	Function Code (Hexadecimal)	1/16-bit	Description	R/W Single or Multiple Values	Memory Block
01	0x01	1-bit	Read Coil Status	Multiple	Coils
02	0x02	1-bit	Read (Discrete) Input Status	Multiple	Discrete Input
03	0x03	16-bit	Read Holding Registers	Multiple	Holding Registers
04	0x04	16-bit	Read Input Registers	Multiple	Input Registers
05	0x05	1-bit	Write Single Coil	Single	Coils
06	0x06	16-bit	Write Single Register	Single	Holding Registers
15	0x0F	1-bit	Write Multiple Coils	Multiple	Coils
16	0x10	16-bit	Write Multiple Registers	Multiple	Holding Registers

(6) Marco de datos de comandos y respuestas Modbus

Ahora, profundicemos en los detalles relacionados con los marcos de datos de los comandos y respuestas de la función Modbus:

Para una solicitud de lectura de múltiples registros (lectura de registros de retención) [0x03] de **Modbus RTU:**

Inicio (siempre 1 bit)
Dirección de esclavo (1 byte)
Código de función 0x10 (1 byte)
Dirección de inicio (2 bytes)
Cantidad de registros (2 bytes)
Comprobación CRC (2 bytes)
Fin (1 o 2 bits)

| Inicio (siempre 1 bit) | Dirección de esclavo (1 byte) | Código de función [0x03] (1 byte) | Dirección de inicio (alta) (1 byte) | Dirección de inicio (baja) (1 byte) | Cantidad (alta) (1 byte) | Cantidad (baja) (1 byte) | CRC bajo (1 byte) | CRC alto (1 byte) | Fin (1 o 2 bits) |

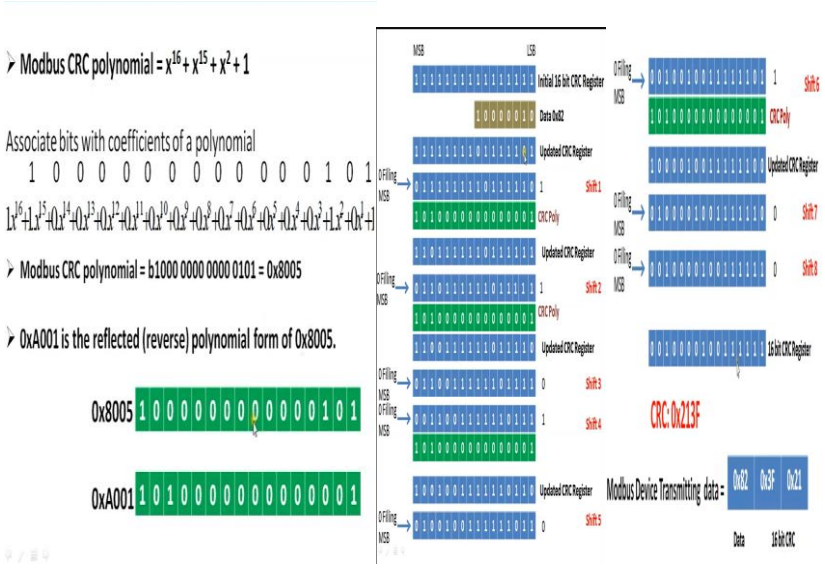
Query message (Function code 03)

Start	Slave address	Function code	Data				Error check	End
			Starting address		Number of registers			
	XX	03	Hi	Lo	Hi	Lo	XX	
Start								*Stop*

Figura: Diagrama delicado de formatos de comandos de función Modbus

Los formatos de comandos de funciones Modbus ilustraron de forma excelente todas las tramas de datos de los comandos (del dispositivo maestro) y las respuestas (de los dispositivos esclavos) de todas las funciones Modbus.

(7) Convención sobre los Derechos del Niño



(8) Referencias

- Un mapa de memoria típico de un dispositivo Modbus
- Desmitificando los códigos de función Modbus
- Códigos de función Modbus
- Formatos de comandos de función Modbus

Modbus
Comunicación en serie
Automatización industrial
Sistema de control

Publicado en Nerd For Tech
12.2K seguidores ·Última publicación : 15 de abril de 2025
NFT es una empresa de medios educativos. Nuestra misión es acercar el valioso conocimiento y la experiencia de expertos de todo el mundo a los principiantes. Para saber más sobre nosotros, visite https://www.nerdfortech.org/ .
Siguiente

Escrito por Yu-Cheng (Morton) Kuo
238 seguidores ·123 Siguiendo
Blog de CS/ML con C++/C/Python. Ingeniero de software en C++. Correo electrónico: morton.kuo.28@gmail.com

BIBLIOGRAFIA

- https://akpolatcem.medium.com/demystifying-modbus-a-gateway-to-industrial-communication-2c57e051b209
- herramientas Modbus Para pruebas, simulación y programación. https://www.modbustools.com/modbus.html
- Un breve recorrido por Modbus con diagramas Tablas y diagramas de flujo de Modbus https://medium.com/nerd-for-tech/a-brief-walkthrough-of-modbus-with-diagrams-a0bd4133f370