# *Slicetex Ladder Designer Studio*

# *NOTA DE APLICACIÓN*

# AN022
# ModBus TCP – Servidor (Slave)

*Autor: Ing. Boris Estudiez*


[1]

| Modelos Aplicables | AX, CX y DX |
|---|---|

## *1 Descripción General*

*La presente nota de aplicación explica como configurar y utilizar en modo servidor (Slave) el protocolo **ModBus TCP** desde nuestros PLC.*

***ModBus** es un protocolo de comunicaciones creado originalmente por Modicon (ahora Schneider Electric) para su uso en PLC. Simple y robusto, el protocolo ModBus se convirtió en un protocolo estándar de facto con el paso del tiempo. Ampliamente difundido, ahora se utiliza para comunicar miles de dispositivos electrónicos industriales.*

***ModBus TCP** permite el empleo del protocolo original **ModBus** con dispositivos que se comunican mediante una red Ethernet (o cualquier otro medio físico) utilizando el stack TCP/IP (internet).*

*Este documento detalla el uso del protocolo **ModBus TCP** en modo servidor (Slave), que le permitirá conectar otros dispositivos clientes (Master) al PLC. De esta manera podrá recibir y enviar datos a otros PLC, paneles HMI o algún software SCADA que operen como cliente ModBus (Master). Ejemplos completos se encuentran en nuestro sitio Web.*

[1]: Imagen propiedad de www.modbus.org.

# 2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el software StxLadder y el PLC adquirido. Sugerimos leer los siguientes documentos:

1. Manual de Usuario del software StxLadder.
2. Manual de Programación Pawn del PLC (si utiliza lenguaje Pawn)
3. Hoja de datos técnicos del PLC.

Mas documentación puede encontrar en la página del producto: www.slicetex.com.

Para consultas y soporte, ponemos a disposición un foro de discusión en: www.slicetex.com/foro donde puede leer preguntas de otros usuarios y realizar también sus propias preguntas.

Se recomienda leer el estándar del protocolo ModBus, disponible en www.modbus.org :

**Protocolo ModBus:**

http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

**ModBus TCP:**

http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

## 2.1 Ejemplos

En nuestro sitio web, busque la pagina de la nota de aplicación AN022, desde dicha podrá encontrar ejemplos completos para utilizar en el PLC.

# 3 Requerimientos

Para esta nota de aplicación, debe tener instalado en su computadora el entorno de Programación **StxLadder** (Slicetex Ladder) y utilizar un firmware actualizado con soporte **ModBus** en el PLC.

Se recomienda estar familiarizado con los conceptos básicos del protocolo ModBus y su mecanismo.

# 4 Teoría de Funcionamiento

Los PLC de Slicetex Electronics permiten configurarse como un servidor **ModBus TCP**, aceptando conexiones remotas de clientes ModBus TCP. La comunicación se realiza a través de una red Ethernet utilizando una comunicación TCP/IP (internet). Cabe destacar que existen routers o conversores que permiten conectar dispositivos ModBus desde redes Ethernet a redes RS-485.

Una transacción típica **ModBus TCP** se muestra en la Figura 1 a continuación:

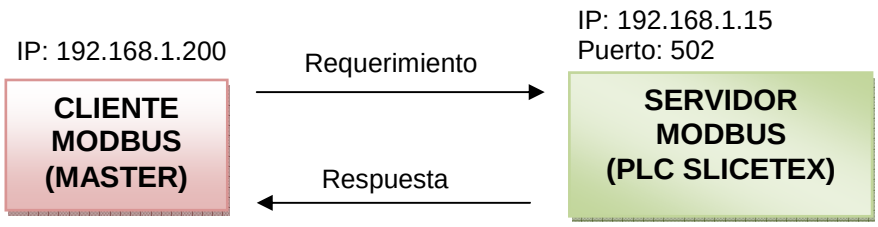



**Fig. 1: Transacción ModBus TCP.**

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción ModBus TCP para procesar un requerimiento de un cliente ModBus. Los pasos son los siguientes:

1. **Configuración**: El PLC debe configurar los parámetros para funcionar como servidor ModBus, por ejemplo puerto TCP de recepción y timeouts.
2. **Conexión**: Si un cliente se conecta, el PLC acepta la conexión (si no está ocupado) y el cliente puede enviar el requerimiento.
3. **Requerimiento**: El cliente envía una función ModBus al servidor para leer o escribir datos.
4. **Respuesta**: El servidor procesa el requerimiento del cliente, y devuelve una respuesta que depende de la función ModBus ejecutada.
5. **Recepción:** El cliente recibe la respuesta.
6. **Fin:** El cliente puede realizar otro requerimiento al servidor o cerrar la conexión para desocupar al servidor.

En la implementación actual, el servidor tiene un límite determinado de conexiones simultáneas de clientes. En caso de que el número de conexiones en paralelo exceda el límite, el servidor devuelve un código de excepción ModBus llamado "**SLAVE BUSY**" (valor 6) y luego desconecta al cliente, según lo especifica el estándar.

El servidor ModBus puede configurarse con un Timeout-Idle, es decir, un valor en segundos que especifica el tiempo que un cliente puede estar conectado al servidor sin realizar ninguna transacción. Si el tiempo se excede, el servidor desconecta al cliente para liberar la conexión a otro posible cliente. Por ello algunos clientes, envían un requerimiento cada X segundos para evitar ser desconectados por inactividad.

## 4.1 Funciones ModBus Soportadas

En la tabla siguiente se listan las funciones ModBus que el PLC soporta como servidor:

**Tabla 1: Funciones ModBus Soportadas**

| Función | Código | Descripción |
|---|---|---|
| Read Coils | 1 | Leer estado de salidas discretas (en general el estado de los reles). |
| Read Discrete Inputs | 2 | Leer estado de entradas discretas. |
| Read Holding Register | 3 | Leer valores de registros “Holding”. |
| Write Single Coil | 5 | Permite modificar el valor de una sola salida discreta. |
| Write Register | 6 | Escribe un valor en un registro. |
| Write Multiple Coils | 15 | Permite modificar el valor de múltiples salidas discretas al mismo tiempo. |
| Write Multiple Registers | 16 | Escribe múltiples registros al mismo tiempo. |

## 4.2 Excepciones ModBus

Cuando el servidor ModBus detecta un error en la transacción, devuelve un error que corresponde a un Código de Excepción. Las posibles excepciones que puede devolver el servidor, se listan a continuación:

**Tabla 2: Excepciones ModBus (Resumen)**

| Excepción | Código Hexa | Descripción |
|---|---|---|
| ILLEGAL FUNCTION | 1 | Función no soportada por el servidor. |
| ILLEGAL DATA ADDRESS | 2 | Dirección invalida. |
| ILLEGAL DATA VALUE | 3 | Valor de datos inválidos.<br>También retornado cuando el paquete recibido está incompleto. |
| SLAVE DEVICE FAILURE | 4 | Error interno en el servidor. |
| SLAVE BUSY | 6 | El servidor no puede aceptar la petición, está ocupado. |

# 5 Particularidades del Servidor ModBus TCP

## 5.1 Mapa de Direcciones

A continuación se muestra el mapa de direcciones del servidor ModBus TCP implementado:

**Tabla 3: Mapa de direcciones por modelo de PLC.**

| PLC (MODELO) | Uso | Rango de Direcciones |
|---|---|---|
| STX8060/8081 | Discrete Outputs (Coils) [1] | 1-8 |
| STX8060/8081 | Discrete Outputs (GP-Coils) [2] | 4001-4128 |
| STX8081 | Discrete Inputs | 10001-10008 |
| STX8060 | Discrete Inputs | 10001-10010 |
| STX8060/8081 | Holding Registers Read / Write [3] | 40001-40032 |

La tabla indica como están asignados los recursos ModBus en los diferentes modelos de PLC que son accesibles por un cliente ModBus.

Por ejemplo, para acceder a los relés del PLC modelo STX8081, hay que enviar un requerimiento con la función ModBus número 1 (**Read Coils**) a un rango valido de direcciones, en este caso las direcciones 1 a 8 reflejan el estado de los 8 reles, en formato 1-bit por salida.

Para leer la entrada discreta número 10 del PLC modelo STX8060, se debe enviar un requerimiento con la función ModBus número 2 (**Read Discrete Inputs**) a la dirección 10010.

Finalmente, los registros “Holding” contienen registros con datos de 16-bits. Dichos datos pueden ser escritos o leídos por el cliente ModBus (a partir de la dirección 42001).

- Nota[1]: Si el PLC funciona con un programa creado en lenguaje Ladder, al escribir sobre un relé desde un cliente, el PLC hará efectivo la escritura en el hardware al final del SCAN CYCLE actual.
- Nota[2]: Se puede leer/escribir un máximo de 32 direcciones al mismo tiempo.
- Nota[3]: Se puede leer/escribir un máximo de 16 direcciones al mismo tiempo.

## 5.2 Parámetros de Conexión

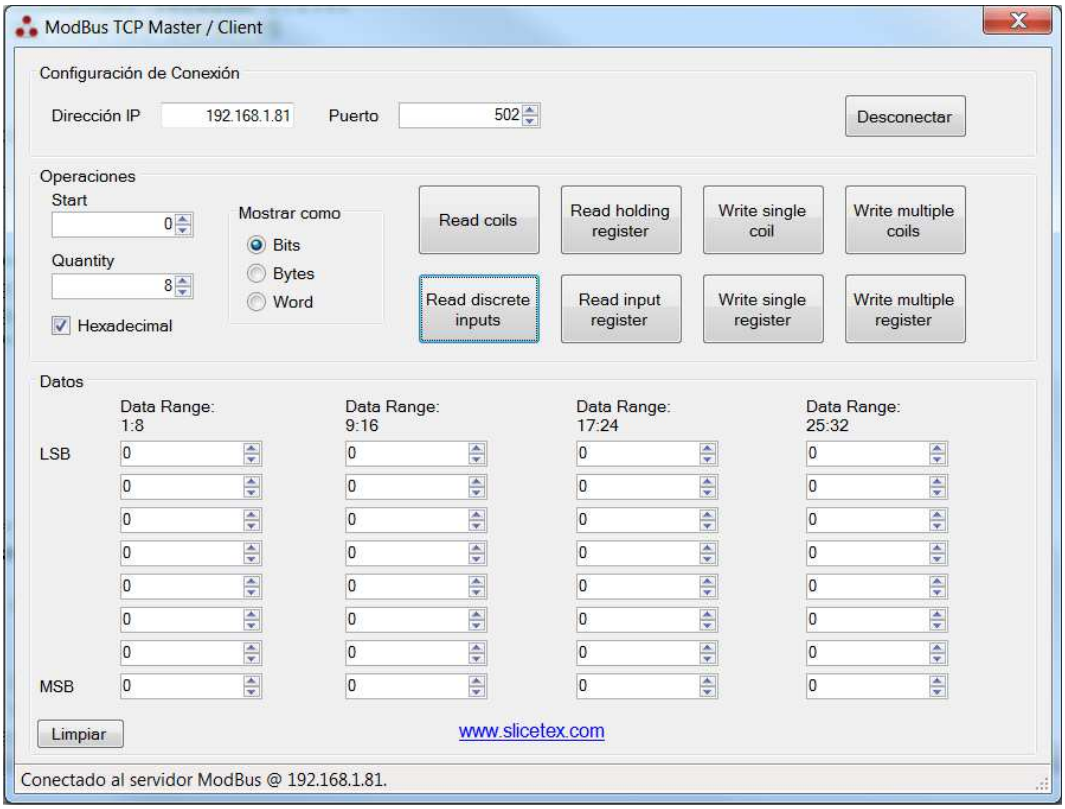
**Tabla 4: Parámetros de Conexión.**

| PLC (MODELO) | Conexiones Simultaneas Máximas (NumberMaxOfTransactions) |
|---|---|
| STX8060/8081 | 2 |

# 6 Software StxLadder – Cliente ModBus TCP de Prueba

Para ayudarlo a desarrollar aplicaciones servidor ModBus TCP con el PLC, hemos incluido un cliente ModBus TCP gratuito para Windows en nuestro entorno de programación StxLadder.

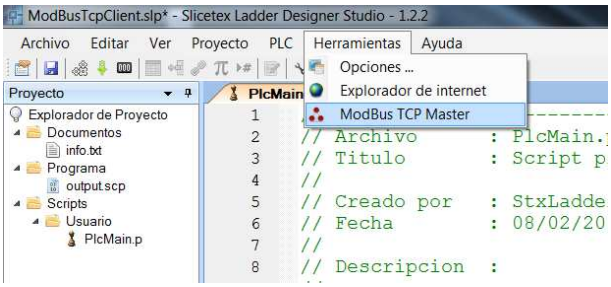
El cliente se muestra en la siguiente figura:




**ModBus TCP Cliente (Master) para Windows**

Cuando desarrolle aplicaciones servidor ModBus TCP, podra probarlas fácilmente con este cliente para Windows. Puede conectarse al PLC para enviar funciones ModBus y escribir o recibir datos.

Para acceder al cliente, cree un proyecto en StxLadder y luego desde el menú “Herramientas / ModBus Tcp Master” accede al programa:



# 7 ModBus Servidor con Lenguaje Ladder

En esta sección explicaremos a modo general como utilizar el protocolo ModBus en modo servidor con el lenguaje Ladder.

Puede bajar ejemplos completos de esta nota aplicación en nuestro sitio Web.

En este documento llamaremos “transacción” al proceso de recibir un requerimiento y enviar una respuesta al cliente ModBus.

Llamaremos “conexión” al periodo de tiempo que la conexión entre el cliente y el servidor esta activa.

En lenguaje Ladder es muy simple configurar el servidor ModBus TCP. Básicamente hay 4 clases de componentes disponibles:

- Funciones para configurar el servidor y la conexión.
- Funciones para cargar registros (que el cliente puede leer).
- Funciones para leer registros (que el cliente ha modificado).
- Funciones para leer y cargar GP-COILS (bits de memoria de uso general).

En la página siguiente repasaremos brevemente como utilizar **ModBus TCP** como servidor en Ladder.

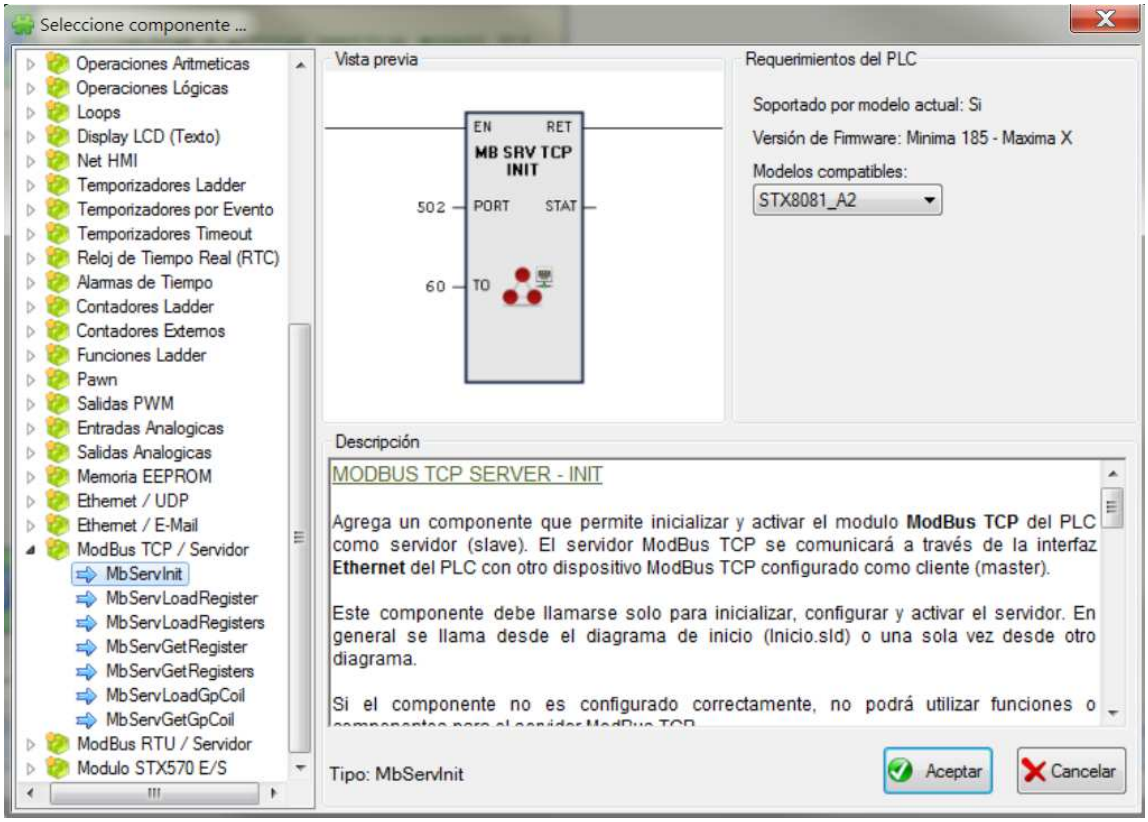
**Importante:**

Recuerde que la documentación detallada de cada componente está disponible en el mismo entorno StxLadder. Para acceder a dicha documentación, solo tiene que insertar el componente, luego seleccionarlo con el botón-derecho del mouse, y posteriormente acceder al ítem “**Ver descripción del componente …**” del menú contextual desplegable del componente.

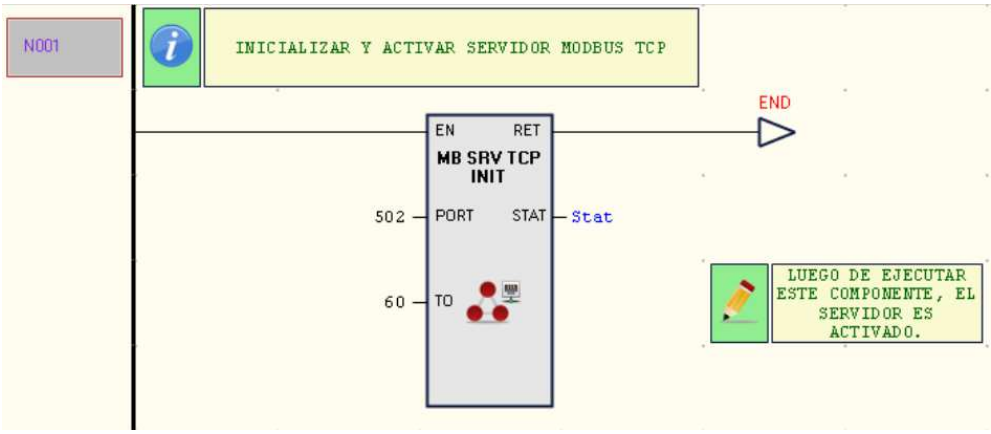
## 7.1 Componentes Ladder

### 7.1.1 Configuración

Los componentes Ladder para el servidor ModBus TCP, se encuentran en el selector de componentes del entorno StxLadder. Dentro del selector, navegue dentro de la sección "**ModBus TCP / Servidor**". Luego puede seleccionar el componente **MbServInit**, como se muestra en la imagen a continuación:




Puede insertar el componente **MbServInit** en el diagrama **Inicio.slp** como se muestra a continuación:



La descripción completa del componente, puede encontrarse en el entorno StxLadder, sin embargo, describiremos a continuación brevemente el mismo:

La entrada **EN** ejecuta el componente si el flujo de corriente es "1". La salida del componente **RET** es "1" si el mismo puedo ser ejecutado con éxito. En caso de error, puede obtener un código de error en la salida **STAT** del componente, que le proporcionará más datos del tipo de error.

La entrada **PORT**, específica el número de puerto TCP que el servidor ModBus TCP escuchará conexiones de clientes remotos. Se utiliza en general 502.

La entrada **TO**, específica que el tiempo en segundos de inactividad que debe pasar antes de desconectar un cliente remoto conectado al servidor. Esto implica que si se especifican 60 segundos, el cliente tiene que realizar al menos una transacción cada menos de 60 segundos, antes que sea desconectado. Esto evita que un cliente "parasito" o con errores, quede indefinidamente conectado al servidor.

El número de clientes que se admiten conectados en simultáneo depende del modelo de PLC utilizado. Ver parámetro **NumberMaxOfTransactions**.

Si el componente es ejecutado sin errores, el servidor ModBus TCP está listo para ser accedido por clientes remotos.

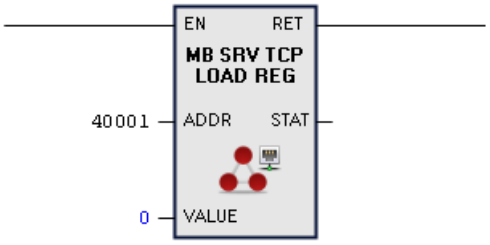
### *7.1.2 Carga de Registros*

Los siguientes componentes permiten cargar los registros "**Holding Registers**" con valores específicos:

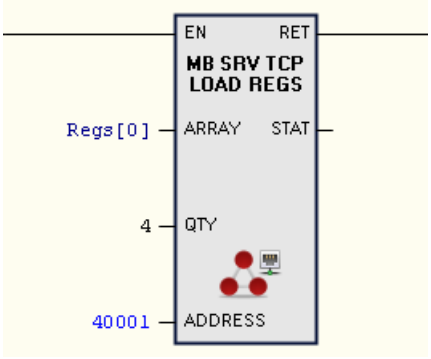
- **MbServLoadRegister**
- **MbServLoadRegisters**

Los registros Holding pueden ser leídos por un cliente que se conecte al servidor.

En el caso del componente **MbServLoadRegister**, la entrada **ADDR** especifica la dirección del registro a escribir en el servidor (en este caso el 40001) con el valor de la entrada **VALUE**. Dicho registro luego puede ser leído por un cliente ModBus TCP.



El componente **MbServLoadRegisters**, permite escribir varios registros al mismo tiempo utilizando un array de datos. En el siguiente ejemplo, la entrada la entrada **ARRAY** se conecta a un array de al menos 4 elementos con valores tipo **Int32**. Luego se especifican la cantidad de elementos a escribir en la entrada **QTY** del componente y en **ADDR** se especifica la dirección del registro inicial a escribir en el servidor (en este caso el 40001).



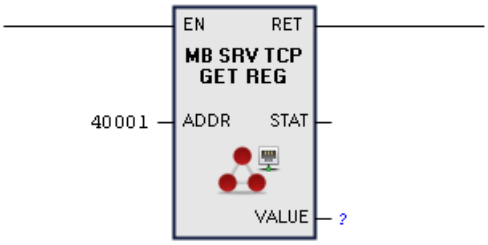

Una vez ejecutado el componente (**EN**=1), la dirección 40001 se escribe con el valor del primer elemento de la variable **Regs**[0] conectada a la entrada **ARRAY**. Luego se escribe la dirección 40002 con el segundo elemento **Regs**[1] y así sucesivamente con los 4 elementos especificados en la entrada **QTY**, hasta llegar al registro 40004.

### *7.1.3 Lectura de Registros*

Los siguientes componentes permiten leer los registros "**Holding Registers**", que es muy útil para recibir los datos escritos en el servidor por un cliente ModBus TCP remoto:

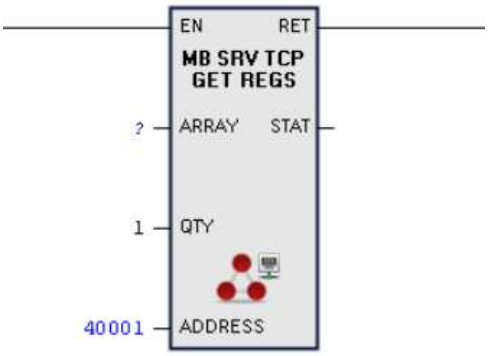
- **MbServGetRegister**
- **MbServGetRegisters**

El componente **MbServGetRegister** devuelve en la salida **VALUE** el valor del registro leído en la dirección especificada por la entrada **ADDR**.




Por ejemplo, si necesitamos leer el registro 40017, especificamos en **ADDR**=40017 y en **VALUE** conectamos una variable del tipo Int32. Luego de ejecutarse el componente, obtendremos el valor del registro.

El componente **MbServGetRegisters** devuelve en **ARRAY** el valor de **QTY** registros leídos a partir de la dirección especificada por la entrada **ADDR**. Esto es muy útil para leer varios registros al mismo tiempo.




Ejemplo, si necesitamos leer 10 registros a partir de la dirección 40001, conectamos un array con al menos 10 elementos del tipo **Int32** en la entrada **ARRAY**, especificamos **QTY**=10 y **ADDRESS**=40001. Al ejecutarse el componente (**EN**=1) el array conectado será actualizado con los valores de los 10 registros en cada uno de sus elementos.

### *7.1.4 Componentes para Manipular GP-COILS*

Las GP-COILS son bits de memoria de propósito general (GP=General Purpose), que pueden ser escritas o leídas por un cliente ModBus.

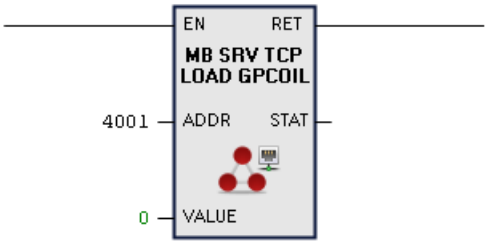
El PLC puede también modificar o leer el valor de las GP-COILS.

Son muy útiles para memoria de uso general en paneles HMI o sistemas SCADA.

A continuación se listan los componentes Ladder para acceder a las GP-COILS:

- **MbServLoadGpCoil**
- **MbServGetGpCoil**

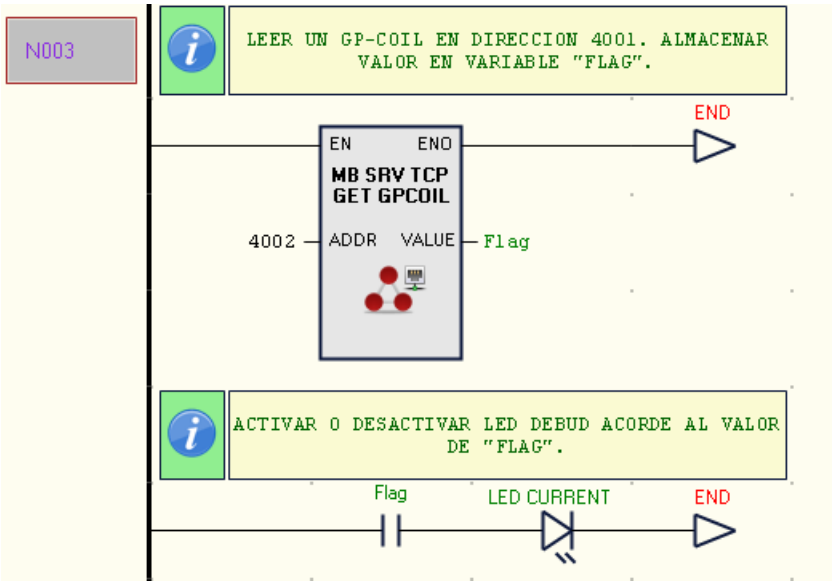
El componente **MbServLoadGpCoil**, permite escribir una GP-COIL en la dirección **ADDR** con el valor del tipo **bool** especificado en la entrada **VALUE**.




Por ejemplo, si escribimos una GP-COIL en la dirección **ADDR**=4001 con el valor **VALUE**=1, un cliente remoto ModBus podrá leer dicho valor y activar una lámpara en un tablero.

El componente **MbServGetGpCoil**, obtiene el valor de una GP-COIL localizada en la dirección **ADDR**, depositando su valor en la variable conectada a la salida **VALUE**.

El siguiente ejemplo, el compoente lee una GP-COIL en la dirección 4002 y deposita su valor en la variable Flag. Luego se comprueba el valor de Flag, si es “1” se activa el led DEBUG del PLC, de lo contrario se desactiva el led DEBUG. Entonces, un cliente remoto podría activar/desactivar el led DEBUG con tan solo escribir un “1” o un “0” en la dirección 4002 del servidor ModBus.




El ejemplo anterior tiene solo fines didácticos, pero considere otras aplicaciones como: Activación o Desactivación de motores, válvulas, etc de forma remota utilizando las GP-COILS.

# 8 ModBus Servidor con Lenguaje Pawn

En esta sección explicaremos a modo general como utilizar el protocolo ModBus en modo servidor con el lenguaje Pawn.

Puede bajar ejemplos completos de esta nota aplicación en nuestro sitio Web.

En este documento llamaremos “transacción” al proceso de recibir un requerimiento y enviar una respuesta al cliente ModBus.

Llamaremos “conexión” al periodo de tiempo que la conexión entre el cliente y el servidor esta activa.

En lenguaje Pawn es muy simple configurar el servidor ModBus TCP. Básicamente hay 5 clases de funciones disponibles:

- Funciones para configurar el servidor y la conexión.
- Funciones para cargar registros (que el cliente puede leer).
- Funciones para leer registros (que el cliente ha modificado).
- Funciones para leer y cargar GP-COILS (bits de memoria de uso general).
- Funciones para activar y desactivar eventos.

En la página siguiente se describen dichas funciones en detalle.

## 8.1 Funciones Nativas en Pawn Disponibles

### 8.1.1 Funciones de Configuración

| **MbServInit(ServerPort, TimeoutIdle):** Inicializa y configura los parámetros para que el PLC pueda funcionar como un servidor ModBus TCP. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| ServerPort | E | Puerto TCP que servidor va a escuchar conexiones, en general se utiliza el 502. |
| TimeoutIdle | E | Tiempo en segundos que deben transcurrir de inactividad para que un cliente sea desconectado. Mínimo 1 y Máximo 125. Recomendado 60 segundos. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, el servidor ya se encuentra inicializado. |
| -2 | S | Error, falla en inicialización, memoria insuficiente. |
| -10 | S | Error, el valor ServerPort es invalido. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

```
//
// Inicializar Servidor ModBus TCP.
// Utilizar puerto 502 y un timeout-idle de 120 segundos.
//
// A partir de este momento, el servidor esta activo.
//

if(MbServInit(502, 120) < 0)
{
      // Mostrar mensaje de error.
      LcdPrintf(0,1, "Error Init")

      // Error, pausar programa en este punto.
      while(true)
      {
            DelayMS(250)
            LedToggle()
      }

}
```

### 8.1.2 Funciones para Cargar Registros

Las siguientes funciones permiten cargar los registros "**Holding Registers**" con valores específicos, dichos registros luego pueden ser leídos por un cliente que se conecte al servidor.

| **MbServLoadRegisters(StartAddr, Qty, Values[ ]):** Carga con valores los registros de lectura Holding. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| StartAddr | E | Dirección del primer registro a cargar. Se utiliza el mismo valor que ve el cliente. |
| Qty | E | Cantidad de registros a cargar. Máximo depende de cantidad de registros disponibles. |
| Values[] | E | Array con los valores a escribir en los registros. De cada elemento del array, solo se toman los 16 bits menos significativos, es decir solo 2 bytes por elemento. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, librería no inicializada. |
| -4 | S | Error, dirección de array inválida. |
| -10 | S | Error, argumento **Qty** invalido. |
| -11 | S | Error, dirección **StartAddr** inválida. |
| -12 | S | Error, dirección **StartAddr** invalidad para el valor **Qty**. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

```
        // Leer canal analogico 1.
        Samples[0] = VinRead(1)

        // Leer canal analogico 2.
        Samples[1] = VinRead(2)

        // Leer canal analogico 3.
        Samples[2] = VinRead(3)

        // Leer canal analogico 4.
        Samples[3] = VinRead(4)

        //
        // Copiar array "Samples" con las cuatro muestras de los
        // canales analógicos en los "Holding Registers"
        // a partir de la dirección 40001. De esta forma
        // seran visibles por un cliente ModBus TCP.
        //

        MbServLoadRegisters(40001, 4, Samples)
```

| **MbServLoadRegister (StartAddr, Value]):** Carga un valor en un registro Holding. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| StartAddr | E | Dirección del registro a cargar. Se utiliza el mismo valor que ve el cliente. |
| Value | E | Valor a escribir en el registro. Solo se toman los 16 bits menos significativos, es decir solo 2 bytes. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, librería no inicializada. |
| -11 | S | Error, dirección **StartAddr** inválida. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

```
//
// Cargar el valor 666 en el registro 40001.
//

MbServLoadRegister(40001, 666)
```

### 8.1.3 Funciones para Leer Registros

Las siguientes funciones permiten leer los registros "**Holding Registers**", dichos registros pueden ser escritos por un cliente que se conecte al servidor.

| **MbServGetRegisters(StartAddr, Qty, Data[ ], Index):** Obtiene los valores de los registros Holding. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| StartAddr | E | Dirección del primer registro a leer. Se utiliza el mismo valor que ve el cliente. |
| Qty | E | Cantidad de registros a leer. Máximo depende de cantidad de registros disponibles. |
| Values[] | S | Array donde se copiaran los valores de los registros leídos. En cada elemento del array se copia un registro de 16-bits. |
| Index | E | Índice donde comiénzala copia en el array. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, librería no inicializada. |
| -4 | S | Error, dirección de array inválida. |
| -10 | S | Error, argumento **Qty** invalido. |
| -11 | S | Error, dirección **StartAddr** inválida. |
| -12 | S | Error, dirección **StartAddr** invalidad para el valor **Qty**. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

```
  new Values[4]

  // Obtener el valor de 4 registros a partir de la dirección
  // 40001 y almacenarlos a partir del elemento 0 en el array Values[].

  MbServGetRegisters(40001, 4, Values, 0)
```

| **MbServGetRegister(StartAddr, &Value):** Obtiene el valor de un registro de lectura Holding. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| StartAddr | E | Dirección del registro a leer. Se utiliza el mismo valor que ve el cliente. |
| Value | S | Valor del registro leído. Solo se toman los 16 bits menos significativos, es decir solo 2 bytes. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, librería no inicializada. |
| -4 | S | Error, dirección de **Value** es invalida. |
| -11 | S | Error, dirección **StartAddr** inválida. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

```
new RegValue = 0

// Leer registro 40001 y almacenar en RegValue.
MbServGetRegister(40001, RegValue)
```

### *8.1.4 Funciones para Manipular GP-COILS*

Las GP-COILS son bits de memoria de propósito general (GP=General Purpose), que pueden ser escritas o leídas por un cliente ModBus.

El PLC puede también modificar o leer el valor de las GP-COILS.

Son muy útiles para memoria de uso general en paneles HMI o sistemas SCADA.

A continuación se listan las funciones Pawn para acceder a las GP-COILS.

| **MbServLoadGpCoil(Address, Value):** Escribe un valor en una GP-COIL. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| Address | E | Dirección de la GP-COIL a escribir. |
| Value | E | Valor a escribir (1 o 0). |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, librería no inicializada. |
| -11 | S | Error, dirección **StartAddr** inválida. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo 1:

A continuación se escriben dos GP-COILS con valores 0 y 1. Luego un cliente ModBus (panel HMI o software SCADA) podrá leer dichos valores en las direcciones correspondientes (4002 y 4032) mediante la función “Read Coils (1)” del protocolo (ver **Tabla 1** en página 4).

```
// Escribir un 0 en GP-COIL en dirección 4002.
MbServLoadGpCoil(4002, 0)

// Escribir un 1 en GP-COIL en dirección 4032.
MbServLoadGpCoil(4032, 1)
```

| **MbServGetGpCoil(Address):** Lee el valor de una GP-COIL. | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| Address | E | Dirección de la GP-COIL a leer. |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | El valor de la GP-COIL es 0. Ver nota[1] también. |
| 1 | S | El valor de la GP-COIL es 1. |
| **Notas** | | **Descripción** |
| 1 | | Si la librería ModBus TCP no fue inicializada o la dirección de la GP-COIL es invalida, el valor retornado es 0 también. |

Ejemplo 1:

A continuación, se lee la GP-COIL en dirección 6002. Si su valor es “1” se activa el led-debug. Si es “0” se desactiva el led-debug. Un cliente/master ModBus TCP puede alterar este valor con la función 5 o 15 del protocolo (ver **Tabla 1** en página 4).

```
// Activar/Desactivar Led Debug de acuerdo a valor del
// GP-COIL en dirección 4002.
if(MbServGetGpCoil(4002))
{
      LedOn()
}
else
{
      LedOff()
}
```

En la práctica, podemos suponer que un panel HMI puede estar alterando este valor para indicar alguna condición al PLC.

### 8.1.5 Funciones para Eventos

Las siguientes funciones le permiten activar o desactivar eventos.

| **MbServSetRxEvent ():** Activa el evento @OnMbServerRx(). | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| - | - | |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, el evento no pudo ser creado. |
| **Notas** | | **Descripción** |
| 1 | | El evento @OnMbServerRx() se activa cuando el cliente escribe algún registro holding a partir de la dirección 40001. |
| 2 | | El uso de eventos, le permite utilizar de forma asíncrona las funciones para realizar requerimientos al servidor ModBus. |

Ejemplo:

El siguiente activa el servidor ModBus y el evento @OnMbServerRx(). Cuando un cliente escribe un registro a partir de la dirección 40001 se genera el evento. Dentro del evento @OnMbServerRx(), leemos el registro 40001, y de acuerdo a su valor activamos o desactivamos un relay.

```
// ********************************************************************************
// Funcion     : PlcMain()
// Descripcion : Punto de entrada principal del PLC.
// ********************************************************************************

PlcMain()
{
   //
   // Inicializar Servidor ModBus TCP.
   // Utilizar puerto 502 y un timeout-idle de 120 segundos.
   //
   // A partir de este momento, el servidor esta activo.
   //

   if(MbServInit(502, 120) < 0)
   {
         // Error, pausar programa en este punto.
         while(true)
         {
               DelayMS(250)
               LedToggle()
         }

   }

   //
```

```
   // Activar evento OnMbServerRx() para
   // que nos avise cuando existan nuevos datos
   // en los Holding Registers (direccion 40001).
   //

   MbServSetRxEvent()

   //
   // Loop Principal.
   //

   for(;;)
   {
         // Conmutar Led Debug.
         DelayMS(1000)
         LedToggle()
   }

   // Retorno.
   return 0
}

// ***************************************************************************
// Funcion     : @OnMbServerRx()
// Descripcion : Evento que es activado cuando algun "Holding
//               Register" es escrito por un cliente ModBus TCP.
// ***************************************************************************

@OnMbServerRx()
{
      new RegValue = 0

      // Leer registro 40001 y almacenar en RegValue.
      MbServGetRegister(40001, RegValue)

      // Procesar el valor del registro y activar
      // o desactivar relays segun corresponda.

      switch(RegValue)
      {
            case 100:
            {
                  // Activar RELAY1.
                  RelayClose(RELAY1)
            }

            case 200:
            {
                  // Activar RELAY2.
                  RelayClose(RELAY2)
            }

            case 300:
```

```
            {
                  // Desactivar RELAY1.
                  RelayOpen(RELAY1)
            }

            case 400:
            {
                  // Desactivar RELAY2.
                  RelayOpen(RELAY2)
            }

            case 500:
            {
                  // Activar ambos reles.
                  RelayClose(RELAY1|RELAY2)
            }

            case 600:
            {
                  // Desactivar ambos reles.
                  RelayOpen(RELAY1|RELAY2)
            }
      }
}
```

| **MbServClrRxEvent ():** Desactiva el evento @OnMbServerRx(). | | |
|---|---|---|
| **Argumentos** | **Tipo** | **Descripción** |
| - | - | |
| **Retorno** | **Tipo** | **Descripción** |
| 0 | S | Operación exitosa. |
| -1 | S | Error, el evento no pudo ser desactivado. |
| **Notas** | | **Descripción** |
| - | | |

Ejemplo:

```
// Desactivar evento "@MbServClrRxEvent”.
if(MbServClrRxEvent() < 0)
{
   // Error.
}
```

# 9 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **IP**: Dirección Internet, conformada por cuatro octetos, por ejemplo 192.168.1.81.
- **MB:** ModBus.
- **Ethernet**: Red de computadoras, que generalmente se utilizan el protocolo de internet TCP/IP o UDP/IP.
- **Transacción**: Proceso de enviar un requerimiento y esperar respuesta de un servidor ModBus.
- **ModBus TCP**: Protocolo ModBus adaptado a la red Internet, en general mediante Ethernet.
- **ModBus RTU**: Protocolo ModBus que utiliza un medio serial (RS232 o RS485) para transferencia de datos.

# 10 Historial de Revisiones

**Tabla 5: Historia de Revisiones del Documento**

| Revisión | Cambios | Descripción | Estado |
|---|---|---|---|
| 03<br>23/APR/2013 | 1 | 1. Se agrega descripción de componentes Ladder, ver pagina 7. | Preliminar |
| 02<br>06/APR/2013 | 1 | 1. Se unifican registros “Holding” a partir de dirección 40001. Ver **Tabla 3** en página 4.<br>2. Se agregan GP-COILS (bits de propósito general) accesibles desde ModBus. Ver **Tabla 3** en pagina 4 y funciones Pawn en pagina 19. | |
| 01<br>04/MAR/2013 | 1 | 1. Versión preliminar liberada. | Preliminar |

# 11 Referencias

Ninguna.

# 12 Información Legal

## 12.1 Aviso de exención de responsabilidad

**General:** La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

**El derecho a realizar cambios:** Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

**Idoneidad para el uso:** Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

**Aplicaciones:** Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

**Los valores límites o máximos:** Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

**Documento**: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

## 13 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Para información técnica, envíe un mail a: devel@slicetex.com

Para información general, envíe un mail a: info@slicetex.com

Para ventas, envíe un mail a: ventas@slicetex.com

Slicetex Electronics
Córdoba, Argentina

## 14 Contenido

## *14.1 Índice de Tablas*