

Automatización Avanzada (37800)
Máster en Automática y Robótica

Práctica 3.
Comunicación con RS-485 y MODBUS



Francisco Andrés Candelas Herías
Grupo de **Innovación Educativa en Automática**



Universitat d'Alacant
Universidad de Alicante

Práctica 3. Comunicación con RS-485 y MODBUS

1. Objetivos

- Conocer el funcionamiento de un bus RS-485 multipunto.
- Conocer el funcionamiento de MODBUS.
- Saber intercambiar datos entre varios PLCs.
- Saber programar comunicaciones serie en un PLC.

2. Comunicaciones serie con el PLC M340 de Schneider

2.1. Conexiones serie

Para la realización de la práctica se utilizarán las maquetas con PLCs Modicom M340 de Schneider, que poseen un módulo de CPU tipo BMX P34 2020, ó BMX P34 1000, versión 1.00, como los mostrados en la Figura 1. Estas CPU incluyen puertos USB y serie. La CPU 2020 incluye además Ethernet.



Figura 1. CPUs de los PLC M340 usados en prácticas

El puerto serie puede funcionar con las señales físicas RS-232 ó RS-485, y tiene una conexión RJ-45 que incluye las líneas básicas para los dos modos. El conector RJ-45 del puerto serie es el situado en la parte inferior y con una marca negra. Aunque el conector RJ-45 es el mismo que el usado en redes de datos Ethernet, el cableado y las señales eléctricas son diferentes.

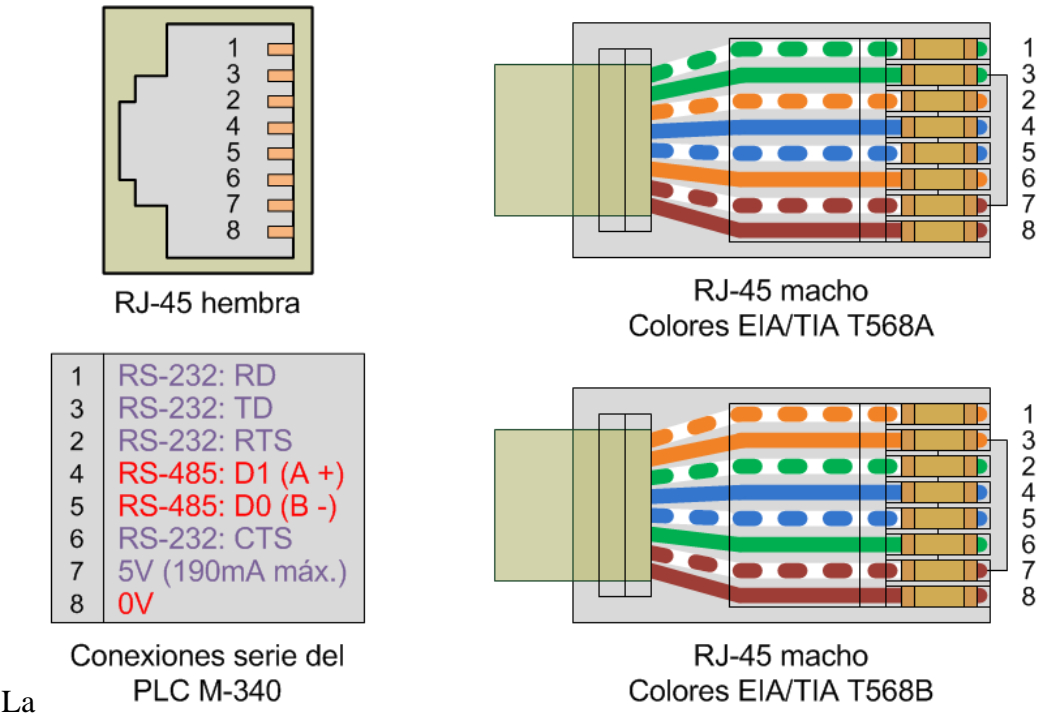


Figura 2 muestra la disposición de las conexiones del puerto serie, así como los dos modos de cableado de colores estándar para un conector RJ-45 de Ethernet. Con el puerto serie en modo RS-232, el PLC actúa como equipo DTE (equipo inteligente) y se puede comunicar con un DCE (módem). Así, TD y RTS son señales de salida del PLC, y RD y CTS son señales de entrada. Si se usa el modo RS-485, se dispone de las líneas D1 y D0 (también denominadas A y B, ó + y -) para el bus, y de la línea de referencias de señales de 0V (GND, o *signal ground*). En la

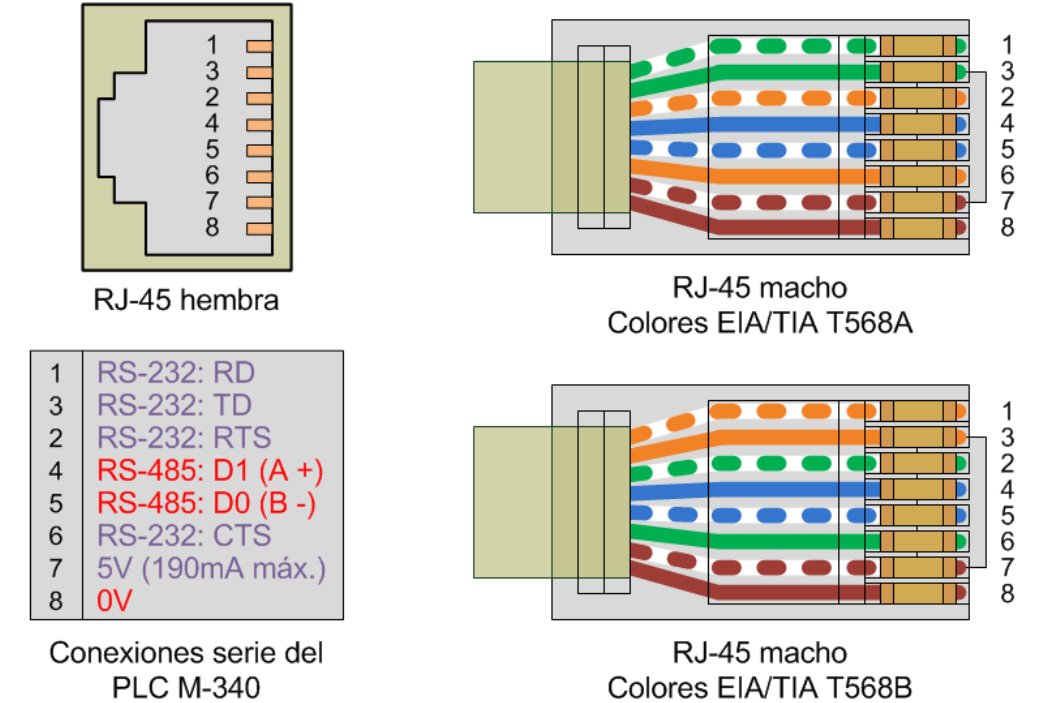


Figura 2 se observa como las líneas D1, D0 y 0V coinciden, respectivamente, con los cables de color estándar azul, azul/blanco y marrón.

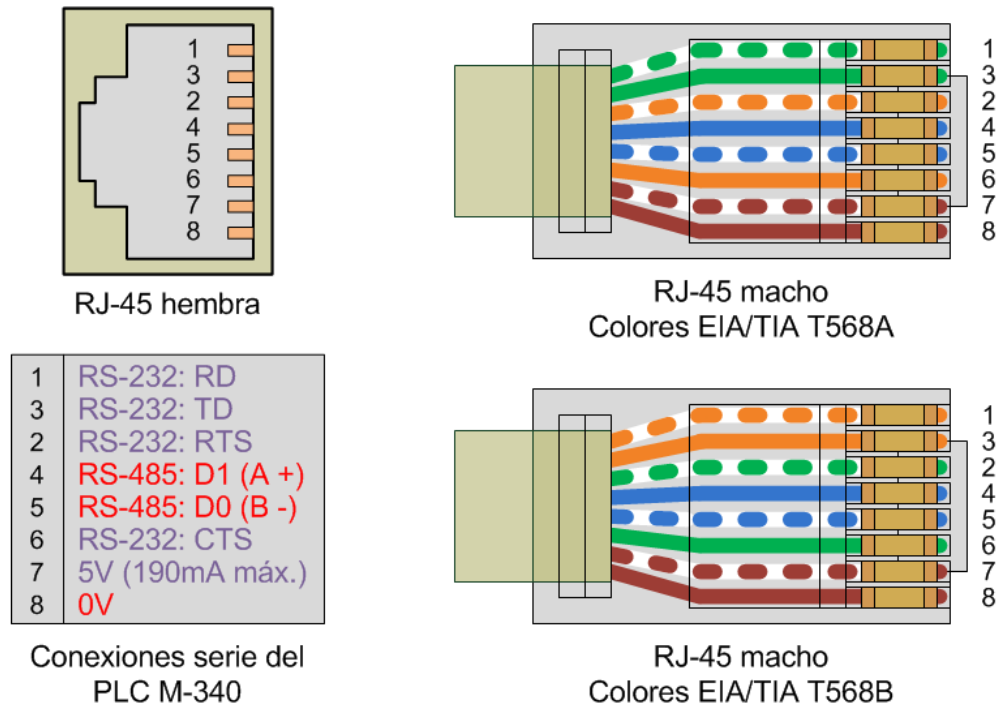


Figura 2. Puerto serie de las CPU para los PLCs M340, y colores del cableado estándar

Además, sobre el puerto serie es posible utilizar el protocolo MODBUS, o un formato de cadenas de caracteres. Con el segundo método, es posible adaptar la comunicación a protocolos no estándar basados en comandos texto, los cuales son bastante comunes en terminales de operador y controladores de máquinas, aunque se requiere una programación más compleja porque es necesario implementar el protocolo dentro del programa del PLC. En cambio, con el protocolo MODBUS, el sistema operativo del PLC ya resuelve muchas operaciones del protocolo, y la programación se simplifica mucho.

2.1.1. Cableado del bus RS-485

El estándar RS-485 define un bus para la transmisión serie multipunto, donde, en un instante, puede haber un equipo transmitiendo y varios recibiendo. La comunicación es semiduplex, de forma un equipo puede enviar y recibir, pero no a la vez. El cableado básico consiste en un par de hilos de cobre trenzados sobre el que se transmite una señal diferencial para enviar los bits de datos, que es bastante inmune a las interferencias y admite largas distancias. Además del par trenzado para datos, pueden usarse líneas de 0V y 5V para alimentar dispositivos del bus. Los bits se transmiten mediante una trama asíncrona.

La Figura 3 describe como se debe realizar la conexión con RS-485 de tres PLC M340. Se usan las líneas de datos D0 y D1 y la de 0V. También se puede utilizar la de chasis (*earth*), que se conecta a la malla de blindaje del cable. En los extremos del cable trenzado del bus RS-485 se requiere unas resistencias terminales R_t de 120Ω , que se pueden conectar en serie con una capacidad de $1nF$. Los PLC M-340 no incluyen internamente las resistencias terminales, por lo que esos componentes hay que colocarlos en el cable. La capacidad hace que la resistencia solo tenga efecto con señales de frecuencias altas, esto es, cuando se transmiten datos. Las resistencias R_p , que se encargan de aplicar la tensión de polarización del bus, las pone el PLC configurado como maestro, por lo que no es necesario cablearlas externamente. Finalmente, se puede mejorar la conexión con unas resistencias R_g entre la línea de 0V y los equipos para limitar las corrientes de retorno al común.

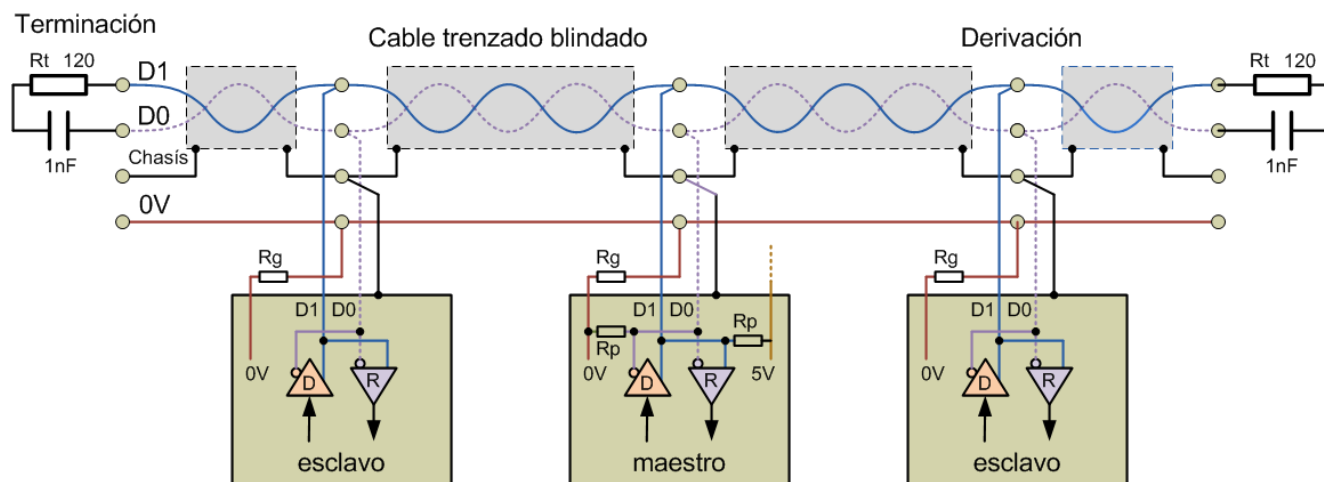
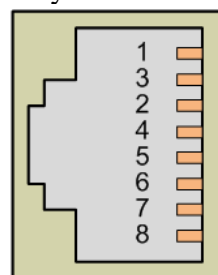
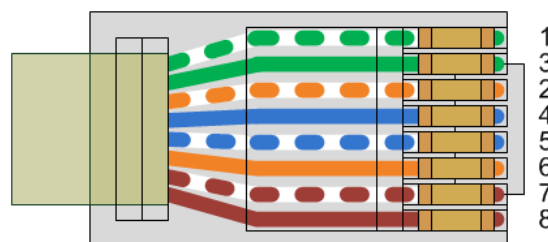


Figura 3. Cableado de los interfaces serie con RS-485

En la realización de esta práctica, para simplificar las conexiones, y dado que las distancias entre equipos serán pequeñas y no habrá un ambiente con ruido electromagnético, se prescindirá de las resistencias R_g , y de la malla del cable. Tampoco se usará la línea de alimentación de 5V, que puede utilizarse para suministrar alimentación a los dispositivos del bus RS-485 sin fuente de alimentación propia. De este modo, en la práctica se montará un cableado como el que se puede ver en el esquema de la Figura 4. Es conveniente comprobar que los conectores RJ-45 utilizados tienen los cables de color azul, azul/blanco y marrón en los contactos correspondientes, como se describe el apartado



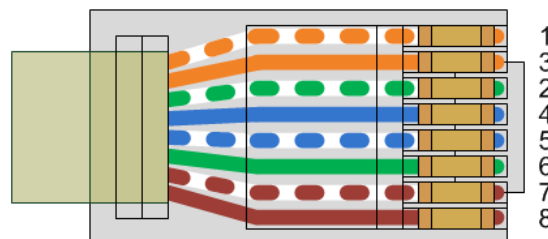
RJ-45 hembra



RJ-45 macho
Colores EIA/TIA T568A

1	RS-232: RD
3	RS-232: TD
2	RS-232: RTS
4	RS-485: D1 (A +)
5	RS-485: D0 (B -)
6	RS-232: CTS
7	5V (190mA máx.)
8	0V

Conexiones serie del
PLC M-340



RJ-45 macho
Colores EIA/TIA T568B

anterior y la

Figura 2. Las distancias de los cables no requieren ser exactamente las indicadas.

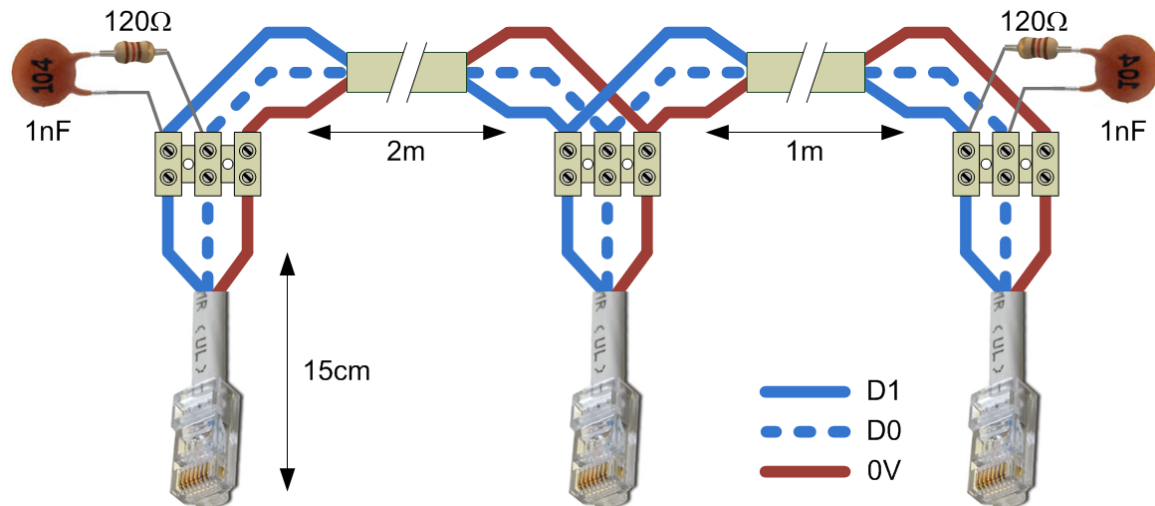


Figura 4. Cableado de los interfaces serie con RS-485

2.2. Configuración de la comunicación serie en un PLC M340 con UnityPro XL

Para configurar y el funcionamiento de un puerto de comunicaciones de un PLC M340 de Schneider se requiere utilizar la aplicación UnityPro XL, que también será necesaria para programar el PLC. En primer lugar, se debe abrir o crear un proyecto con la configuración de bastidor que se utilizará. La

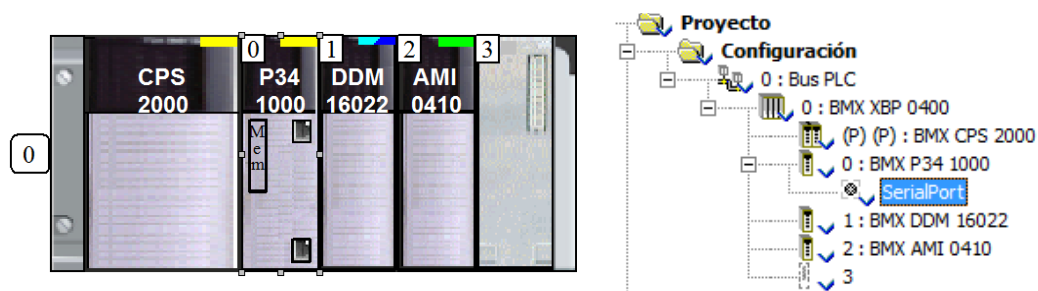


Figura 5 muestra la configuración que se utilizará en esta práctica, y que contiene una fuente de alimentación BMX CPS2000, una CPU BMX P34 2020 ó BMX P34 1000), un módulo de E/S digitales BMX DDM 16022, y un módulo de E/S analógicas BMX AMM 0600 ó BMX AMM 0400, todo ello montado sobre un bastidor BMX XPB 0400.

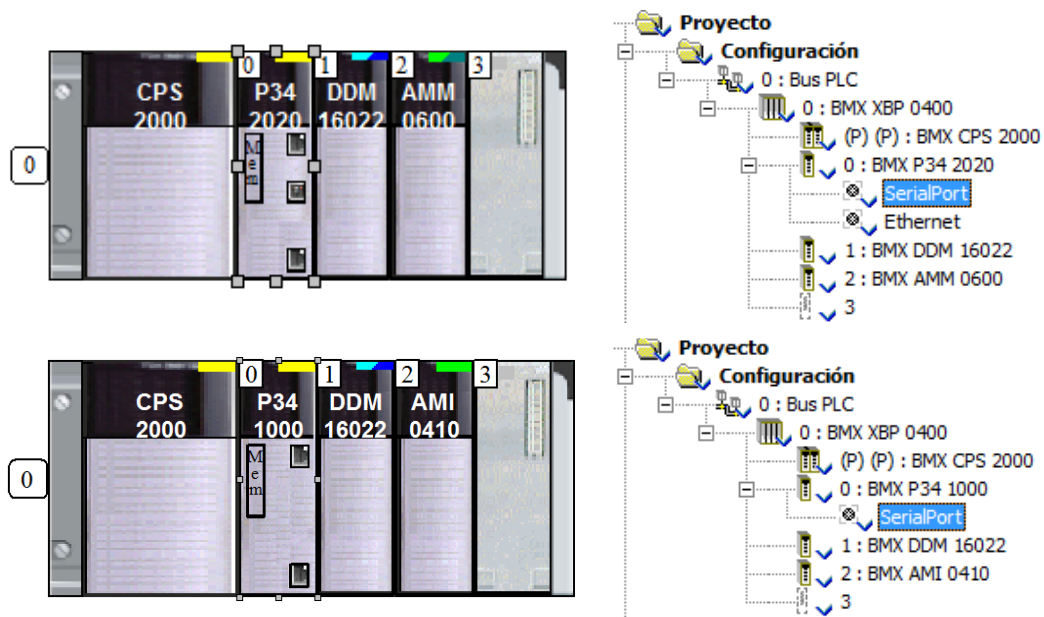


Figura 5. Configuración del PLC

Seguidamente, en el Explorador de Proyectos se debe hacer doble clic con el ratón sobre el puerto serie de la CPU, que está nombrado como *SerialPort*. Esto abrirá la ventana de propiedades y configuración del puerto serie. Dentro de esta ventana, se seleccionará el Canal 0 en la parte izquierda, como muestra la Figura 6, para acceder al formulario de configuración.

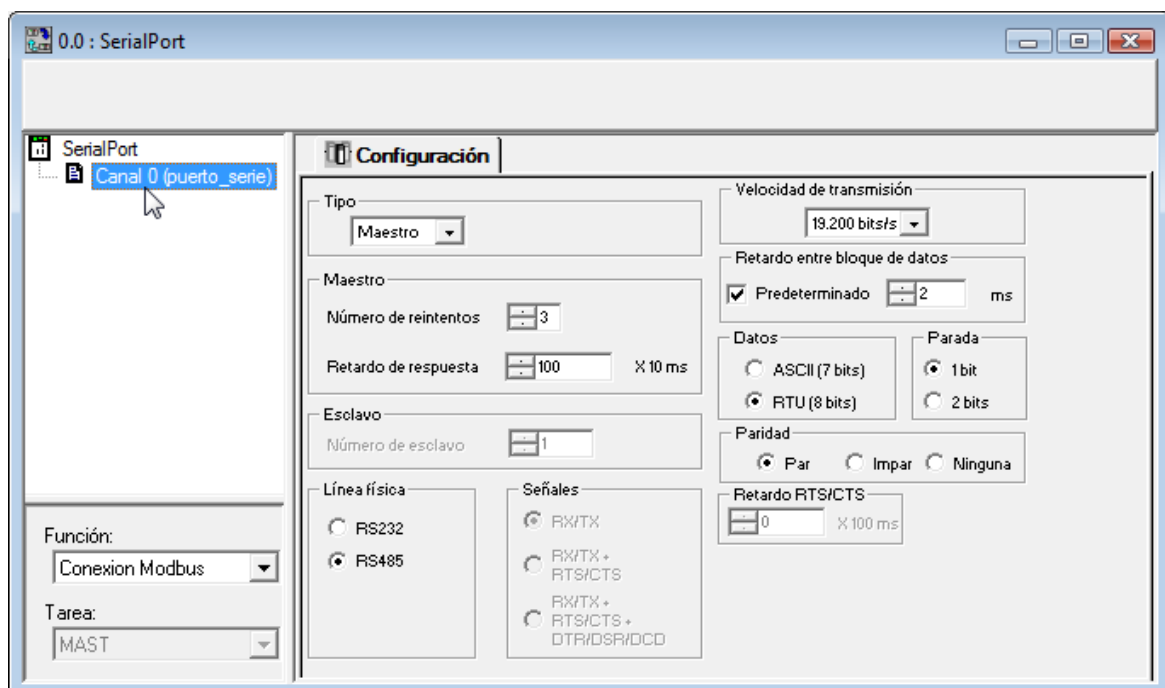


Figura 6. Configuración del puerto serie

Bajo la lista de puertos y canales, después de seleccionar el canal, se puede seleccionar el protocolo a utilizar con el control de Función, que puede ser “Conexión modalidad de caracteres” o “Conexión MODBUS”. En función de la selección, la pestaña de “Configuración” mostrará un formulario con los parámetros configurables del protocolo escogido. A continuación se describen los parámetros que se pueden configurar para MODBUS, que será el protocolo usado en la práctica.

- **Tipo** (*Esclavo, Maestro*). Selecciona el modo de MODBUS con que el PLC accederá al puerto serie. Hay que tener en cuenta que en un momento dado, en una conexión o bus serie solo debe haber un dispositivo maestro.
- **Maestro: Número de reintentos** (0-15). Disponible solo cuando el PLC se configura como maestro. Indica el número de intentos de transmisión a un esclavo antes de darlo como ausente en el bus. El valor 0 indica que no habrá reintentos.
- **Maestro: Retardo de respuesta** (10ms-10s, en pasos de 10ms). Disponible solo cuando el PLC se configura como maestro. Indica el tiempo transcurrido entre la petición inicial enviada por el maestro y un intento repetido de ésta en caso de que el esclavo no responda. Este tiempo debe ser como poco el máximo tiempo entre el envío del último carácter de una petición del maestro y la recepción del primer carácter de respuesta del esclavo. La longitud del cable afectará al tiempo mínimo que conviene escoger, debido al retardo de propagación de la señal.
- **Esclavo: Número de esclavo** (1-247). Disponible solo cuando el PLC se configura como esclavo. Indica la dirección del esclavo para MODBUS. En configuraciones multipunto (con un maestro y más de un esclavo) se usan las direcciones 1 a 247. El valor 248 se utiliza sólo para conexiones punto a punto (un maestro y un esclavo). Cada esclavo debe tener una dirección diferente al resto.
- **Línea Física** (RS-232, RS-485). Especifica qué tipo de líneas físicas del puerto serie se van a utilizar. Aunque el puerto serie de los M340 puede funcionar físicamente como RS-232 ó RS-485, no puede hacerlo simultáneamente, ya que solo dispone de un canal serie que debe ser configurado en uno de los dos modos.
- **Señales** (RX/TX, RX/TX + RTS/CTS). En el caso de utilizar el nivel físico RS-232, este campo se habilita para escoger si se van a usar las líneas de control del interfaz serie además de las líneas de datos.
- **Velocidad de transmisión** (300, 600, 1.200, 2.400, 4.800, 9.600, 19.200 y 38.400 bps). Es la velocidad de transmisión que se usará para el puerto serie, y que debe ser igual a la configurada en los otros dispositivos del bus.
- **Retardo entre bloque de datos** (2ms – 3.413ms). Indica el tiempo mínimo que puede separar dos tramas en la recepción. Este valor debe ajustarse en función de la velocidad. Según el estándar de MODBUS, para velocidades de hasta 19.200bps, el tiempo entre tramas debe ser como mínimo 3,5 veces la duración de un carácter, y para velocidades superiores se recomienda un tiempo fijo de 1,75ms. La duración de un carácter depende de la velocidad (V_t) y el número de bits (N) según N/V_t . Por ejemplo, para 19.200bps, con un bit de parada y un bit de paridad (11 bits en total, sumando el de inicio y 8 de datos) se tiene: $3,5 \cdot 11/19.200=2\text{ms}$.
- **Datos** (ASCII, RTU). Selecciona el formato para las tramas de MODBUS serie.
- **Parada** (1, 2). Permite introducir el número de bits de parada utilizados en la comunicación, que debe ser igual a la configurada en los otros dispositivos del bus.
- **Paridad** (par, impar, ninguna). Permite determinar si se agrega un bit de paridad, así como su tipo. El bit de paridad permite al equipo receptor de una trama de datos determinar si los bits de datos son correctos, o ha habido una alteración en uno de ellos. Esta configuración debe ser igual a la configurada en los otros dispositivos del bus.
- **Retardo RTS/CTS** (0-10s, pasos de 100ms). Cuando es mayor que 0, indica que el PLC tendrá en cuenta las señales RST y CTS para controlar el envío de datos. En este caso, antes de enviar una trama, el PLC activa su salida RTS y espera que, en respuesta, el DCE active la entrada

CTS para dar permiso al envío. Si pasa el tiempo indicado sin que se reciba respuesta en la entrada CTS, el envío se descarta.

2.3. Programación de la comunicación serie en un PLC M340

La programación de comunicaciones con MODBUS en un PLC M340 se puede realizar de forma bastante sencilla con la aplicación UnityPro XL y las funciones que proporciona. Básicamente, se dispone de tres funciones o bloques que se pueden utilizar en el programa del PLC maestro:

- **READ_VAR.** Permite al maestro enviar una petición de lectura de valores a un esclavo, para que este devuelva los valores de una zona de memoria o de entradas. En concreto, se pueden leer objetos de tipo %M, %I, %MW y %IW de los esclavos, esto es, valores de bit (booleanos) o enteros en formato palabra (Word), de memoria o de entradas. Equivale a los códigos de función 16#01¹ (lectura de bits), 16#02 (lectura de bits de entrada), 16#03 (lectura de palabras) y 16#04 (lectura de palabras de entrada) de MODBUS.
- **WRITE_VAR:** Permite al maestro enviar valores a un esclavo para que los escriba en una zona de su memoria. En concreto, se pueden escribir valores en objetos de memoria de tipo %M y %MW de los esclavos, esto es, valores de bit o enteros en formato palabra de la memoria interna. Equivale a los códigos de función 16#0F (escritura de bits) y 16#10 (escritura de palabras) de MODBUS.
- **DATA_EXCH:** Permite especificar el código de función de MODBUS que el esclavo debe ejecutar, enviar una tabla de valores enteros a un esclavo, y esperar la recepción de otra tabla de valores enteros desde el esclavo. De este modo, se pueden programar procedimientos específicos de acceso a los parámetros de un dispositivo esclavo determinado, siguiendo el protocolo MODBUS.

Las tres funciones están disponibles en los lenguajes FBD (diagrama de bloques funcionales), Ladder (diagrama de contactos) y ST (texto estructurado).

En contraste, un equipo o PLC que se configura como esclavo de MODBUS, atenderá automáticamente las peticiones del maestro del bus, y no requiere una programación especial de las comunicaciones. Su programa solamente deberá encargarse de mantener correctamente actualizadas las direcciones de memoria cuyos valores puede solicitar el maestro, así como de atender los valores de las zonas de memoria donde el maestro escribe.

2.3.1. Gestión de direcciones

El primer paso para poder trabajar con las funciones para transmisión de datos en el PLC maestro es generar las direcciones de los esclavos a los que se desea acceder con un formato que entienden las funciones de transmisión. Para ello se dispone de la función ADDM. La Figura 7 muestra ejemplos del uso del bloque FBD y de la sintaxis de ST para esta función.

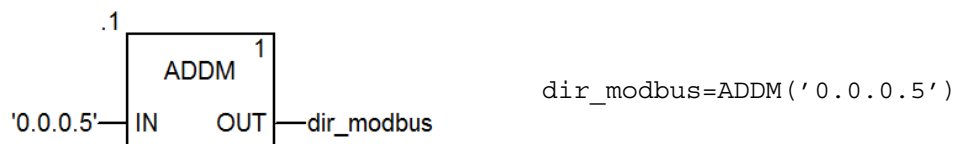


Figura 7. Sintaxis de la función ADDM

¹ 16#01 se refiere al valor hexadecimal 01, que corresponde con el decimal 1. Así por ejemplo, el valor hexadecimal 16#0A se corresponde con el valor decimal 10, y 16#FF se corresponde con el decimal 255. La notaciones equivalentes en los lenguajes de programación C, C++ o C# serían 0x01, 0x0A y 0xFF.

Las entradas y salidas de esta función son:

Parámetros de entrada y salida de la función ADDM			
Parámetro	Sentido	Tipo	Significado
IN	Entrada	STRING	Dirección de dispositivo en una conexión. La sintaxis de la dirección es del tipo ' <i>r.m.c.n</i> ' dónde: <ul style="list-style-type: none"> <i>r</i>: Número de bastidor del procesador. <i>m</i>: Número de ranura del procesador del bastidor. <i>c</i>: Número de canal. <i>n</i>: Número de esclavo al que se envía la solicitud. En comunicaciones serie con la CPU BMX P34 2020, los parámetros <i>r</i> , <i>m</i> y <i>c</i> valen 0.
OUT	Salida	ARRAY [0..7] OF INT	Vector de enteros que representa la dirección de un dispositivo. Este resultado se puede utilizar entrada para otras funciones de comunicación.

2.3.2. Lectura de valores

La función READ_VAR permite solicitar valores a un esclavo de MODBUS identificado por su dirección, previamente formateada con la función ADDM. La Figura 8 muestra ejemplos del uso del bloque FBD y de la sintaxis de ST para esta función.

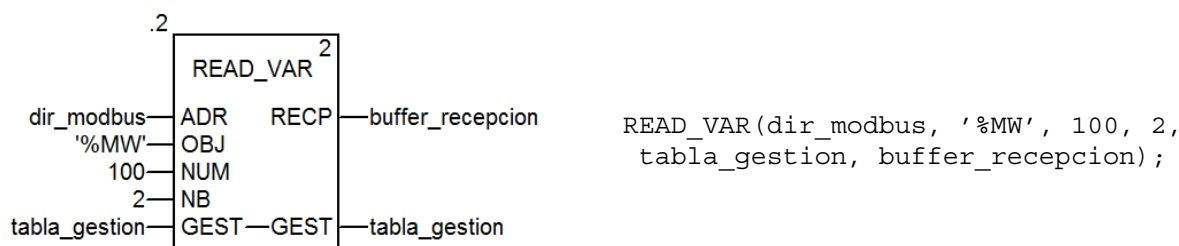


Figura 8. Sintaxis de la función READ_VAR

Parámetros de entrada y salida de la función READ_VAR			
Parámetro	Sentido	Tipo	Significado
ADR	Entrada	ARRAY [0..7] OF INT	Dirección proporcionada por la función ADDM
OBJ	Entrada	STRING	Tipo de valor que se va a leer desde el esclavo. Los tipos disponibles son: <ul style="list-style-type: none"> %M: bit interno. %I: bit de entrada externa. %MW: palabra interna. %IW: palabra de entrada.
NUM	Entrada	DINT	Dirección del primer valor que se va a leer, en la memoria del esclavo.
NB	Entrada	INT	Número de valores consecutivos que se van a leer.
GEST	E/S	ARRAY [0..3] OF INT	Vector que contiene la "tabla de gestión de intercambios" según se describe en el Anexo del apartado 4.1. Cada función de comunicación usada debe tener su propia variable de tabla de gestión.
RECP	Salida	ARRAY [n..m] OF INT	Vector de palabras en donde se depositarán los valores leídos del esclavo.

Como se observa, al usar esta función, los registros que se quieren leer del esclavo se indican directamente mediante su tipo y ubicación en la memoria del PLC esclavo, y no es necesario tener en cuenta la tabla de registros de dispositivo (ver apartado **¡Error! No se encuentra el origen de la referencia.**).

2.3.3. Escritura de valores

La función `WRITE_VAR` permite enviar valores a un esclavo de MODBUS identificado por su dirección, previamente formateada con la función `ADDM`. La Figura 9 muestra ejemplos del uso del bloque FBD y de la sintaxis de ST para esta función.

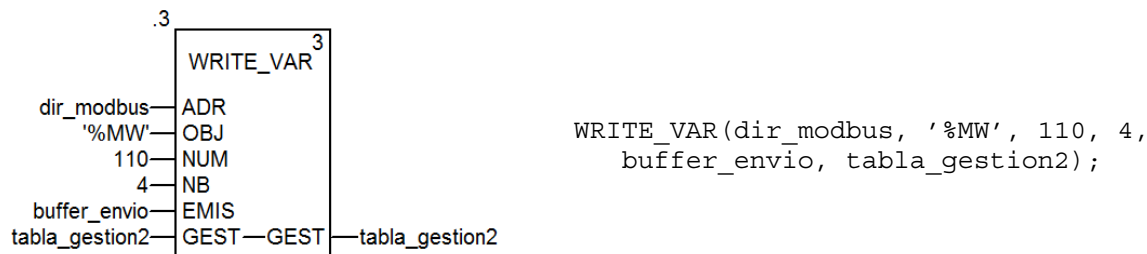


Figura 9. Sintaxis de la función `WRITE_VAR`

Parámetros de entrada y salida de la función <code>WRITE_VAR</code>			
Parámetro	Sentido	Tipo	Significado
ADR	Entrada	ARRAY [0..7] OF INT	Dirección proporcionada por la función <code>ADDM</code>
OBJ	Entrada	STRING	Tipo de valor que se va a modificar en el esclavo. Los tipos disponibles son: %M: bit interno. %I: bit de entrada externa. %MW: palabra interna. %IW: palabra de entrada.
NUM	Entrada	DINT	Dirección del primer valor que se va a modificar en la memoria del esclavo.
NB	Entrada	INT	Número de valores consecutivos que se van a modificar.
EMIS	Entrada	ARRAY [n..m] OF INT	Vector de palabras en donde están los valores que hay que enviar al esclavo.
GEST	E/S	ARRAY [0..3] OF INT	Vector que contiene la "tabla de gestión de intercambios" según se describe en el Anexo del apartado 4.1. Cada función de comunicación usada debe tener su propia variable de tabla de gestión.

Como se observa, al usar esta función, los registros que se quieren modificar en el esclavo se indican directamente mediante su tipo y ubicación en la memoria del PLC esclavo, y no es necesario tener en cuenta la tabla de registros de dispositivo (ver apartado **¡Error! No se encuentra el origen de la referencia.**).

Después de ejecutar una función como `READ_VAR` o `WRITE_VAR`, hay que comprobar cuando acaba su ejecución antes de leer y procesar los valores recibidos. Como ejemplo, se muestra a continuación un programa en lenguaje ST para el PLC M340 que lee los valores de los objetos `%MW10` y `%MW11` del esclavo 2, luego suma 5 a esos valores, y finalmente los escribe en el esclavo 3. Para gestionar las comunicaciones, se ha programado una sencilla máquina de estados.

```
(* Iniciar estado de comunicaciones *)
if (primer_ciclo) then
    estado_maestro := 0;
    error:=false;
end_if;

(* Maquina de estados para controlar la comunicación *)
case estado_maestro of
    (* Esperando señal de inicio en la entrada1*)
    0: if (RE(entrada1)) then
        (* Pide 2 objetos del esclavo 2: MW10 y MW11 *)
```

```

    direccion_s2:= ADDM('0.0.0.2');
    READ_VAR(direccion_s2, '%MW', 10, 2, tabla_ges1, buffer1);
    estado_maestro := 1;
end_if;

(* Esperando a recibir valores *)
1: if ((tabla_ges1[0] & 16#01) = 0) then

    (* Comprueba si hubo errores en comunicación con esclavo 1 *)
    (* Examina el byte de menor peso de la palabra 2 de la tabla
    (* de gestión, que contiene el informe comunicaciones *)
    if ((tabla_ges1[1] & 16#00FF) <> 0) then
        error := true;
    else

        buffer1[0] := buffer1[0] + 5;
        buffer1[1] := buffer1[1] + 5;

        (* Envía resultado a los objetos MW10 y MW11 del esclavo 3 *)
        direccion_s3:= ADDM('0.0.0.3');
        WRITE_VAR(direccion_s3, '%MW', 10, 2, buffer1, tabla_ges2);

        estado_maestro := 2;
    end_if;

    (* Esperando a que acabe envío *)
    2: if (not(tabla_ges1[0].0)) then

        (* Comprueba si hubo errores en comunicación con esclavos *)
        (* Examina el byte de menor peso de la palabra 2 de la tabla
        (* de gestión, que contiene el informe comunicaciones *)
        if ((tabla_ges2[1] & 16#00FF) <> 0) then
            error := true;
        end_if;

        estado_maestro := 0;
    end_if;
end_case;

```

2.3.4. Ejecución de otras funciones de MODBUS

La función **DATA_EXCH** permite enviar y recibir valores enteros a un esclavo de MODBUS identificado por su dirección, previamente formateada con la función **ADDM**. Pero a diferencia de las funciones **READ_VAR** y **WRITE_VAR**, con **DATA_EXCH** se puede especificar el código de la función de MODBUS que el esclavo debe ejecutar. De este modo, se puede acceder a gran variedad de dispositivos que soportan MODBUS. Sin embargo, esta versatilidad hace que el uso de esta función sea más complicado. Por eso, para esta práctica se aconseja el uso de las funciones **READ_VAR** y **WRITE_VAR**. La Figura 10 muestra ejemplos del uso de esta función con los lenguajes FBD y ST.

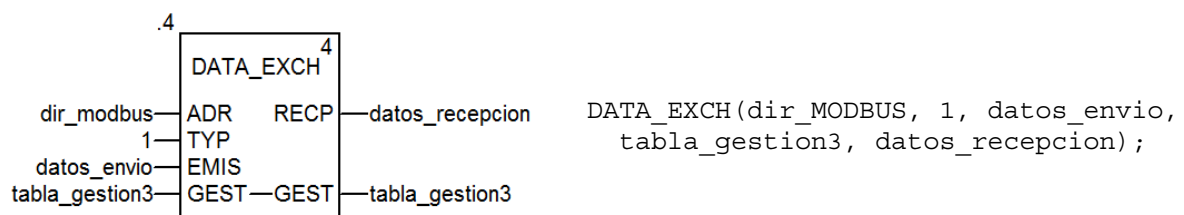


Figura 10. Sintaxis de la función **DATA_EXCH**

Parámetros de entrada y salida de la función DATA_EXCH			
Parámetro	Sentido	Tipo	Significado
ADR	Entrada	ARRAY [0..7] OF INT	Dirección proporcionada por la función ADDM
TYP	Entrada	STRING	Para PLC Modicon M340, el único valor posible es 1 , que significa una transmisión del vector EMIS, y, a continuación, esperar a la recepción del vector RECP.
EMIS	Entrada	ARRAY [n..m] OF INT	Vector de enteros que sirve para indicar la función MODBUS que el esclavo debe ejecutar, los registros del esclavo a los que hay que acceder, y los valores que hay que enviar. Debe tener, como mínimo un elemento, aunque no haya datos que enviar. Antes de llamar a la función, hay que especificar el tamaño del array, en bytes , en la cuarta palabra de la tabla de gestión de intercambios
GEST	E/S	ARRAY [0..3] OF INT	Vector que contiene la “tabla de gestión de intercambios” según se describe en el Anexo del apartado 4.1. Cada función de comunicación usada debe tener su propia variable de tabla de gestión.
RECP	Salida	ARRAY [n..m] OF INT	Vector de enteros en donde se depositarán los valores leídos del esclavo. Debe indicarse y tener un tamaño mínimo de 1, aunque la función se use solo para enviar. Después de la recepción, se actualiza la cuarta palabra de la tabla de gestión con el número de bytes recibidos.

El número máximo de valores que se pueden enviar o recibir es de 256 bytes. La cuarta palabra de la tabla de gestión de intercambios (parámetro GEST) se usa para indicar el número de valores enviados, y para conocer el número de valores recibidos.

El código de función MODBUS que el esclavo debe ejecutar, y la ubicación de los datos que hay que enviarle, se indican a través del vector de palabras en el parámetro de entrada EMIS, que debe organizarse de forma similar a una trama de MODBUS, según de código de función que se ejecute.

A continuación se describe un ejemplo en el que se utiliza DATA_EXCH para realizar una petición MODBUS con el código de función 6 o 16#06 a un esclavo con dirección 7, con el objetivo de asignar a su registro #MW100 el valor que contiene la variable *dato* de tipo INT o WORD. Lo primero que se debe hacer es preparar el vector *datos_envio* con los datos a enviar. Este vector debe incluir el código de función (6), la dirección del registro (100) y el valor a asignar, conforme muestra la Figura 11. En esta figura se observa que palabras en UnityPro emplean un orden LITTLE-ENDIAN, esto es, primero se coloca el byte menos significativo.

Palabra 0		Palabra 1		Palabra 2	
Cod	RegH	RegL	V1H	V1L	NU
6	0	100	valor	valor	0
Byte L	Byte H	Byte L	Byte H	Byte L	Byte H

Figura 11. Formato del vector de datos de entrada para la función DATA_EXCH

Los campos representados en la Figura 11 se describen en la siguiente tabla:

Cod	Código de función MODBUS que el esclavo debe ejecutar.	V1H	Valor a enviar, byte alto.
RegH	Byte de mayor peso del número del registro de memoria del esclavo donde empezar a escribir.	V1L	Valor a enviar, byte bajo.
RegL	Byte de menor peso del número de dirección del registro de memoria del esclavo donde empezar a escribir.	NU	No usado

Después de completar el vector *datos_envio*, se debe comprobar que la función DATA_EXCH no se está ejecutando ya, consultando para ello el bit de actividad de su tabla de gestión de intercambios (ver el Anexo del apartado 4.1), que está almacenada en la variable vector *tabla_gestion3*. Si la función no se está ejecutando, se actualiza la última palabra de la tabla de gestión con el número de bytes útiles que tiene el vector *datos_envio*, y se ejecuta DATA_EXCH. A continuación se muestra el código de programa ST que realiza todas estas operaciones.

```
(* Palabra 0: ByteH = RegH = 0, Byte L = Cod = 6 *)
datos_envio[0] = 6;

(* Palabra 1: ByteH = V1H, Byte 2 = RegL = 100 *)
datos_envio[1] := (valor & 16#FF00) + 100;

(* Palabra 2: ByteH = no usado, ByteL = V1L *)
datos_envio[2] = valor & 16#FF;

(* Si la función no está activa ya *)
IF (tabla_gestion3[0].0 = 0) THEN
  (* Guarda tamaño en bytes del vector con los datos en tabla de gestión *)
  tabla_gestion3[3] := 5;
  DATA_EXCH(ADDM('0.0.0.7'), 1, datos_envio, tabla_gestion3, datos_recepcion);
END_IF;
```

En el caso de que se usase DATA_EXCH para solicitar un valor a un esclavo, y se quisiese recoger ese valor, habría que esperar primero a que acabase la función, lo que se puede determinar consultando cuando el bit de actividad de su tabla de gestión cambia de 0 a 1. En ese momento se podría analizar el valor devuelto. El siguiente código de programa ST realiza estas operaciones.

```
(* Coger el valor devuelto cuando acaba la función *)
IF (not (tabla_gestion3[0].0)) THEN
  (* Conversión BIG/LITTLE ENDIAN *)
  resultado := ROL(datos_recepcion[1], 8);
END_IF;
```

En el vector *datos_recepcion* se encontrarían los valores devueltos por MODBUS desde el esclavo. Los dos primeros bytes, situados en *datos_recepcion[0]* serían la dirección del esclavo y el código de función, y los dos siguientes bytes, situados en *datos_recepcion[1]*, contendrían la palabra solicitada. Aquí hay que tener en cuenta que, mientras que MODBUS utiliza un orden BIG-ENDIAN para los bytes, esto es, con el byte de mayor peso primero, a la izquierda, las palabras en UnityPro emplean un orden LITTLE-ENDIAN, con el byte de menor peso primero. Por ese motivo, en el programa de ejemplo se invierten los bytes *datos_recepcion[1]* de mediante la función ROL.

2.4. Operaciones avanzadas en las comunicaciones MODBUS de un PLC M340

2.4.1. Cancelación de una transmisión en curso

Aunque no sea necesario para esta práctica, a veces puede interesar cancelar un intercambio de datos iniciado con las funciones READ_VAR, WRITE_VAR o DATA_EXCH. Esto se puede realizar de dos formas, que se muestran a continuación con ejemplos en lenguaje ST:

a) Utilización de la función CANCEL

```
IF (tabla_gestion[0].0) THEN
  num_inter:=SHR(tabla_gestion[0], 8);
  CANCEL(num_inter, resultado);
```


END_IF;

En el ejemplo, la variable *tabla_gestion* (vector de enteros) se corresponde con parámetro GEST (tabla de gestión de intercambios) de la función que se desea cancelar. Así, *tabla_gestion[0].0* corresponde al bit de actividad de la función a cancelar y se establece en 1 cuando la función de comunicación está activa. Si el bit se establece en 1, el programa anterior lleva a cabo las instrucciones siguientes:

- Desplaza los bits *tabla_gestion[0]* un byte (8 bits) hacia la derecha y carga el byte correspondiente al número de intercambio de comunicación en la variable entera *num_inter*.
- Cancela el intercambio cuyo número de intercambio está contenido en la variable *num_inter* con la función CANCEL. En la variable entera *resultado* se almacena un informe del resultado.

b) Utilización del bit de cancelación de la función de comunicación

```
IF (tabla_gestion[0].0) THEN  
  SET(tabla_gestion[0].1);  
  READ_VAR(ADDM('0.0.0.6'), '%MW', 100, 10, tabla_gestion, buffer_rec);  
END_IF;
```

En el ejemplo, la variable *tabla_gestion* (vector de enteros) se corresponde con parámetro GEST (tabla de gestión de intercambios) de una función READ_VAR que se desea cancelar. Así, *tabla_gestion[0].0* corresponde al bit de actividad la función READ_VAR, y se establece en 1 cuando la función de comunicación está activa. Si este bit se ha establecido en 1, el programa anterior establece el bit *tabla_gestion[0].1*, el bit de cancelación de la función, en 1. Esto detiene la comunicación de la función READ_VAR del programa de ejemplo. Aunque en este ejemplo de programación se cancela una función READ_VAR, es igualmente aplicable a las funciones WRITE_VAR y DATA_EXCH.

Este segundo método también es válido para cancelar una comunicación desde una tabla de animación. Para ello, basta con establecer el bit de cancelación de la función en 1 y, a continuación, iniciar de nuevo la función de comunicación.

2.4.2. Acceso a características del protocolo MODBUS

Si se desea acceder a toda la información de estado de un canal E/S del PLC, es necesario definir una variable IODDT (objeto de tipos derivado de datos de entrada/salida) del tipo adecuado al canal deseado. A través de los campos de esa variable se puede acceder al estado del canal, para detectar posibles errores, o para cambiar alguna configuración durante la ejecución del programa.

Por otra parte, cuando se desea un mayor control por programa de un canal E/S de un PLC, es necesario recurrir a un acceso directo e inmediato a las variables del módulo. Esto es lo que se denomina un intercambio de datos explícito a petición del programa de usuario que ejecuta el PLC, en contraste con el intercambio implícito determinado por el ciclo de lectura de entradas, ejecución de programa y escritura de salidas. Para realizar un intercambio explícito en un M340, se dispone de dos funciones: READ_STS para lectura de palabras de estado y WRITE_CMD para escritura de palabras de comando. Las instrucciones READ_STS y WRITE_CMD se ejecutan al mismo tiempo que la tarea que las llama y siempre correctamente, y su resultado queda disponible automáticamente después de su ejecución.

Para el protocolo MODBUS sobre el puerto serie, el PLC M340 admite un tipo de datos IODDT denominado T_COM_MB_BMX. La forma más sencilla de definir una variable de este tipo para una CPU BMX P24 2020 es crear directamente en la tabla de datos derivados esa variable indicando su nombre, el tipo T_COM_MB_BMX, y la dirección %CH0.0.0, que se corresponde con el canal 0 del

puerto serie. La Figura 12 muestra la lista de campos que contiene la variable *PuertoSerie*, creada según se ha descrito.
































Nombre	Tipo	Valor	Direcc...	Comentario
 PuertoSerie0	T_COM_MB_BMX		%CH0.0.0	
 CH_ERROR	BOOL		%I0.0.0.ERR	Error de canal
 INPUT_SIGNALS	INT		%IW0.0.0.0	Señales de entrada
 DCD	BOOL		%IW0.0.0.0.0	Detección de portadora de datos
 CTS	BOOL		%IW0.0.0.0.2	Listo para transmitir
 DSR	BOOL		%IW0.0.0.0.3	Conjunto de datos preparado
 LISTEN_ONLY	BOOL		%IW0.0.0.0.8	Modalidad de sólo escucha (sólo esclavo de Modbus)
 EXCH_STS	INT		%MW0.0.0.0	Estado de intercambio
 STS_IN_PROGR	BOOL		%MW0.0.0.0.0	Lectura de parámetros de estado en curso
 CMD_IN_PROGR	BOOL		%MW0.0.0.0.1	Escritura de parámetros de comando en curso
 ADJ_IN_PROGR	BOOL		%MW0.0.0.0.2	Intercambio de parámetros de ajuste en curso
 EXCH_RPT	INT		%MW0.0.0.1	Informe de canal
 STS_ERR	BOOL		%MW0.0.0.1.0	Error al leer el estado del canal
 CMD_ERR	BOOL		%MW0.0.0.1.1	Error al enviar un comando por el canal
 ADJ_ERR	BOOL		%MW0.0.0.1.2	Error al ajustar el canal
 CH_FLT	INT		%MW0.0.0.2	Fallos de canal
 NO_DEVICE	BOOL		%MW0.0.0.2.0	No hay ningún dispositivo funcionando en el canal
 ONE_DEVICE_FLT	BOOL		%MW0.0.0.2.1	Un dispositivo del canal tiene fallos
 BLK	BOOL		%MW0.0.0.2.2	Fallo externo: Bloque de terminales
 TO_ERR	BOOL		%MW0.0.0.2.3	Error de timeout (verificar cableado)
 INTERNAL_FLT	BOOL		%MW0.0.0.2.4	Fallo interno: Canal inoperativo
 CONF_FLT	BOOL		%MW0.0.0.2.5	Fallo de configuración de hardware o software
 COM_FLT	BOOL		%MW0.0.0.2.6	Fallo de comunicación de bus
 APPLI_FLT	BOOL		%MW0.0.0.2.7	Fallo de aplicación
 PROTOCOL	INT		%MW0.0.0.3	6 para maestro Modbus, 7 para esclavo Modbus, 3 para modalidad de caracteres
 CONTROL	INT		%MW0.0.0.24	Protocolo/señales de control
 DTR_ON	BOOL		%MW0.0.0.2...	Terminal de datos preparada Con
 DTR_OFF	BOOL		%MW0.0.0.2...	Terminal de datos preparada Des
 TO_MODBUS_MASTER	BOOL		%MW0.0.0.2...	Conmutar a maestro Modbus
 TO_MODBUS_SLAVE	BOOL		%MW0.0.0.2...	Conmutar a esclavo Modbus
 TO_CHAR_MODE	BOOL		%MW0.0.0.2...	Conmutar a modalidad de caracteres

Figura 12. Campos de una variable tipo T_COM_MB_BMX

La columna “comentario” indica la utilidad de cada campo. En particular los siguientes campos son interesantes para la realización de la práctica:

- **CH_ERROR.** Este campo booleano de lectura se pone a TRUE cuando el protocolo MODBUS detecta algún error. Como es un valor %I que se actualiza de forma implícita, se puede consultar directamente desde el programa, con una sintaxis como *PuertoSerie.CH_ERROR*. Para conocer más detalles sobre el error hay que consultar otros campos de la variable.
- **TO_MODBUS_MASTER.** Al poner a TRUE este campo, se fuerza el cambio a modalidad de maestro de MODBUS. Pero para acceder a campo de tipo %MW hay que ejecutar una operación explícita. Así, para asignar un nuevo valor, es necesario actualizar primero el campo con *PuertoSerie.TO_MODBUS_MASTER:=true* y después hacer una operación de escritura explícita: *WRITE_CMD(PuertoSerie)*. De forma similar, para leer su valor, primero hay que ejecutar *READ_STS(PuertoSerie)*.
- **TO_MODBUS_SLAVE.** Este campo funciona como el anterior, pero sirve para forzar el cambio a modalidad de esclavo de MODBUS.

Existe una gran variedad de tipos IODDT y de posiciones de memoria que permiten acceder a propiedades concretas de los canales de E/S, para comprobar su estado y cambiar su configuración en tiempo de ejecución. Si el alumno desea más información sobre estos parámetros, se pueden consultar los manuales del PLC o la ayuda de UnityPro XL.

3. Experimento a realizar

Como aplicación práctica, se pondrá en marcha un sistema distribuido en el que se utilizarán dos PLCs M-340 conectados mediante un bus RS-485, para intercambiar información entre ellos. Los alumnos formarán equipos, de forma que cada equipo programará una pareja de PLCs.

Cada equipo realizará dos programas, uno para un PLC maestro y otro para un PLC esclavo. Las funciones que deben realizar los dos PLCs están esquematizadas en la Figura 13, y se describen en los siguientes apartados.

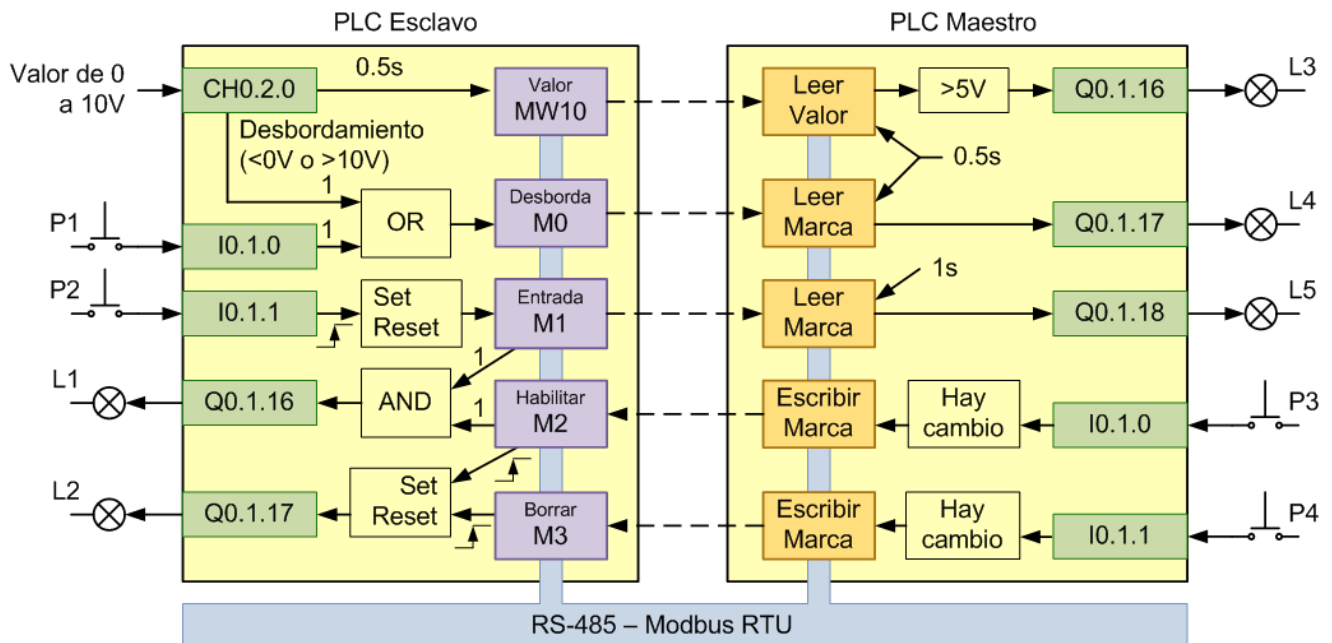


Figura 13. Sistema a programar

3.1. Funcionamiento del esclavo.

Un PLC que funcione como esclavo debe realizar estas tareas:

1. Cada 0,5 segundos se debe obtener el valor de la entrada analógica 0 (%CH0.2.0 o %IW0.2.0) y almacenarlo en la variable entera “valor” que estará alojada a la palabra de memoria %MW10. Inicialmente esta variable debe tener el valor 0.
2. La variable booleana “Desborda”, alojada en la marca de memoria %M0, debe tener el valor 1 (true) cuando hay desbordamiento en la entrada analógica 0, o cuando se activa el pulsador P1 conectado a la entrada digital %I0.1.0.
3. Una pulsación del pulsador P1, conectado a la entrada digital %I0.1.1, debe poner a 1 (true) la variable booleana “Entrada”, asociada a la dirección %M1. La siguiente pulsación de P1 debe poner a 0 (false) la variable “Entrada”. Así sucesivamente, de forma que cada cambio de 0 (false) a 1 (true) en P1 conmuta el valor de la variable “Entrada”.
4. Cuando las variables “Entrada” y “Habilitar” estén simultáneamente a 1 (true), el piloto L1 conectado a la salida digital %Q0.1.16 se encenderá. Este piloto L1 se activa con un 1 (true). La variable booleana “Habilitar” estará alojada en la marca de memoria %M2.

5. Cuando la variable booleana “Habilitar” cambie de 0 (false) a 1 (true), se activará el piloto L2 conectado a la salida digital Q0.1.17. Cuando la variable booleana “Borrar”, que estará alojada en la marca de memoria %M3, cambie de 0 (false) a 1 (true), se desactivará el piloto L2. Es decir, L2 se activa con la variable “Habilitar” y se desactiva con la variable “Borrar”.

Los objetos o variables %MW10, %M0, %M1, %M2 y %M3 podrán ser accedidos para lectura o escritura desde un PC maestro.

La entrada analógica 0 se configurará para leer valores de tensión de 0 a 10V, que serán escalados a un valor entero de 0 a 10.000. Además, se debe configurar dicha entrada para generar un desbordamiento cuando se obtenga un valor inferior a 0V o superior 11V.

El estado de desbordamiento se puede consultar asociando una variable de tipo T_ANA_IN_BMX al canal 0 del módulo analógico (%CH0.2.0), y accediendo a la marca de estado RANGE_FLT del canal. Por ejemplo, si se define la variable “analog0” de tipo T_ANA_IN_BMX, se puede acceder a la marca de desborde con “analog0.RANGE_FLT”, de la misma forma que “analog0.VALUE” permite acceder al valor entero leído de la entrada.

3.2. Funcionamiento del PLC maestro

El PLC maestro monitorizará y controlará un PLC esclavo de la siguiente forma:

6. Cada 0,5 segundos se debe leer del esclavo los objetos %MW10 y %M0. El piloto L3 conectado a la salida digital %Q0.1.16 debe encenderse cuando el valor entero leído de %MW10 representa un valor mayor de 5V. El valor booleano leído de %M0 indica si la salida digital %Q0.1.17, a la que se conecta el piloto L4, debe estar a 0 (false) o a 1 (true).
7. Cada 1 segundo se debe leer del esclavo el objeto %M1. El valor booleano ese objeto indica si la salida digital %Q0.1.18, a la que se conecta el piloto L5, debe estar a 0 (false) o a 1 (true).
8. Cuando se detecte un cambio de estado en el pulsador P3 conectado a la entrada digital %I0.1.0, se debe enviar el valor de dicha entrada a la marca %M2 del esclavo.
9. Cuando se detecte un cambio de estado en el pulsador P4 conectado a la entrada digital %I0.1.1, se debe enviar el valor de dicha entrada a la marca %M2 del esclavo.

Es importante que las funciones de leer o escribir en un esclavo no se ejecuten constantemente, sino solo en las situaciones que se han indicado anteriormente. Además hay que tener en cuenta que una función de comunicación debe acabar su ejecución antes de utilizar los resultados recibidos del esclavo, y antes de ejecutar una nueva función de comunicación.

3.3. Parámetros de la comunicación

Los parámetros para la comunicación serie para todos los PLC deben ser:

- Modo RS-485, MODBUS-RTU 8 bits.
- Velocidad de transmisión de 19.200bps.
- 1 bit de parada, paridad par.
- Retardo entre tramas de 2ms (ver apartado 2.2).
- Para el maestro: 3 reintentos y retardo de respuesta de 1s.
- Para el esclavo: dirección en el rango 1 a 20.

3.4. Cuestiones optativas

En vez de escribir dos programas separados, uno para el PLC maestro y otro para el PLC esclavo, se puede desarrollar un único programa que conmute entre las funciones de maestro y esclavo. Dicha conmutación se puede activar cuando se pulse un botón. Para conmutar de función, se requiere usar las características descritas en el apartado 2.4.2.

Se puede utilizar el convertidor de RS-485 a RS-232 o a USB para monitorizar desde un PC las comunicaciones del sistema desarrollado, mediante un programa de terminal. Para interpretar más fácilmente los datos transmitidos por el bus, se puede configurar los interfaces serie de los PLC para usar el modo MODBUS-ASCII en vez de MODBUS-RTU.

Colocar más de un PLC esclavo en un bus, y probar a controlar los diferentes esclavos desde el PLC maestro del bus.

4. Anexos

4.1. Introducción a MODBUS

MODBUS es un protocolo estándar que puede gestionar una comunicación tipo cliente-servidor entre distintos equipos conectados físicamente con un bus serie. Este protocolo fue ideado para los PLCs Modicon (marca que ahora pertenece a Schneider Electric) en 1979, y con el tiempo se ha convertido en un protocolo muy empleado en las comunicaciones industriales. Las principales razones de ello son la sencillez del protocolo, versatilidad, y que sus especificaciones, gestionadas por la MODBUS Organization, son de acceso libre y gratuito.

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

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

4.1.1. Tramas de MODBUS

Para intercambiar las peticiones y respuestas, los dispositivos de una red MODBUS organizan los datos en tramas. Dado que MODBUS es un protocolo de nivel de aplicación, se requiere utilizarlo sobre una pila de protocolos que resuelva los temas específicos del tipo de red empleada. En función de la arquitectura de protocolos usada, se distinguen tres tipos de MODBUS: **RTU**, **ASCII** y **MODBUSTCP**.

MODBUS RTU y ASCII están pensados para ser utilizadas directamente sobre un medio físico serie asíncrono, como por ejemplo EIA/TIA RS-232, EIA/TIA RS-485, o EIA RS-422. En contraste, MODBUSTCP está desarrollado para funcionar sobre redes que utilizan la arquitectura TCP/IP, por lo que permite usar MODBUS sobre redes como Ethernet o WiFi.

No se debe confundir MODBUSTCP con MODBUS over TCP o MODBUS over UDP. Estas dos son otras opciones que, mediante un convertidor TCP/IP Ethernet, Wifi o de otro tipo, permiten transportar directamente una trama de MODBUS RTU o ASCII sobre redes TCP/IP. Hay que entender bien estos conceptos ya que las diferentes opciones tienen funcionamientos diferentes.

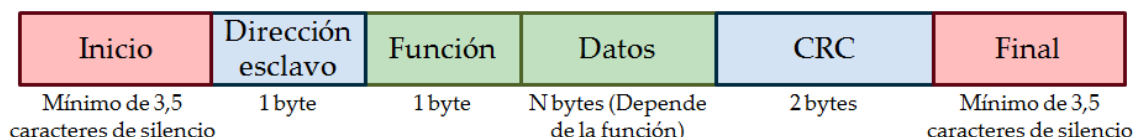
La Figura 14 muestra el formato de trama de las tres opciones de MODBUS, que se describen con más detalle a continuación. Los campos **Función** y **Datos** representan la trama de nivel de aplicación de MODBUS, y dependen de las distintas opciones de peticiones y respuestas que describirán en el apartado 0. El tamaño del campo de datos siempre depende de la función utilizada. Para la colocación de los bytes en los distintos campos hay que tener en cuenta que MODBUS siempre utiliza codificación BIG-ENDIAN, según la cual el byte más significativo se envía primero (a la izquierda).

La **dirección** es un valor que debe identificar unívocamente a un dispositivo esclavo de la red. Este valor de identificación debe corresponderse con un número entre 1 y 247 en configuraciones **multipunto**, como son los buses RS-422 y RS-485 que tienen un maestro y un esclavo o más. El valor especial de dirección **248** se utiliza sólo cuando MODBUS se emplea sobre una conexión **punto a**

punto, por ejemplo con un maestro y solo un esclavo en una conexión RS-232. Por último, el valor 0 es la dirección de difusión o *broadcast*, y una petición enviada a esta dirección es atendida por todos los esclavos. Este tipo de peticiones no producen una respuesta de los esclavos ya que no se podría controlar el acceso al medio de estos y habría colisiones. Por eso mismo, tampoco se recibirá ninguna respuesta si se ejecuta una petición de lectura con dirección de *broadcast*.

Nótese que, si se quiere conmutar la función del equipo maestro a esclavo en determinados momentos de la comunicación, hay que reservar también un identificador único para él.

- **Trama RTU:**



- **Trama ASCII:**

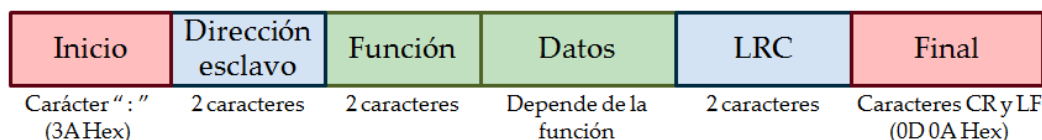


Figura 14. Formato de las tramas de MODBUS serie

4.1.2. MODBUS RTU

MODBUS RTU (*Remote Terminal Unit*) se caracteriza por que los bytes se envían en su codificación binaria plana, sin ningún tipo de conversión. Está inicialmente pensado para comunicaciones en bus serie. Como ventaja principal tiene el buen aprovechamiento del canal de comunicación, mejorando la velocidad de la transmisión de los datos. El inconveniente es que requiere una gestión de tiempos entre bytes recibidos para saber cuando empiezan y terminan las tramas.

Con la trama MODBUS RTU, la delimitación de la misma se realiza por intervalos de tiempo de caracteres de silencio, como muestra la Figura 14. Un carácter de silencio tiene la duración de un byte de datos enviado por el medio, pero no transporta datos, y su duración (T) depende de la velocidad (Vt) y del número bits que se usen para su codificación (N) según $T = N/V_t$. Según el estándar de MODBUS, para velocidades de hasta 19.200bps, el tiempo entre tramas debe ser como mínimo 3,5 veces la duración de un carácter, y para velocidades superiores se recomienda un tiempo fijo de 1,75ms. Por ejemplo, para una configuración del puerto serie de 19.200bps, con un bit de parada y un bit de paridad (11 bits en total, sumando el de inicio y 8 de datos) se tiene: $3,5 \cdot 11 / 19.200 = 2\text{ms}$.

La trama MODBUS RTU incorpora un código Cyclical Redundancy Check (CRC) de 16 bits (ver Figura 14) para poder detectar errores, que debe ser calculado por el emisor a partir de todos los bytes de la trama enviados antes del CRC, exceptuando los delimitadores. Para ello se usa un algoritmo específico, bien definido en la especificación de MODBUS serie. El receptor debe volver a calcular el código de igual forma que el emisor, y comprobar que el valor obtenido del cálculo es igual al valor presente en la trama para poder validar los datos.

4.1.3. MODBUS ASCII

Los datos se codifican como caracteres ASCII entre el "0" (16#30) y el "9" (16#39) y entre "A" (16#41) y "F" (16#46). Por ejemplo, si se requiere enviar el byte de valor 16#FF, se tiene que enviar la cadena "FF", por lo que realmente se enviarían dos bytes: 16#46 y 16#46. Además se utilizan 3 caracteres especiales. El carácter ":" (16#3A) se emplea para marcar el comienzo de la trama y el par

de caracteres no imprimibles "CRLF" (16#0D, retorno de carro, y 16#0A, salto de línea) se emplean como delimitador del fin de la trama.

Este formato tiene dos grandes ventajas. Primero, ofrece una facilidad de detección del principio y del fin de trama gracias a los campos de inicio y fin (caracteres ":" y "CRLF"), con independencia de los tiempos de la transmisión del canal de comunicación. Segundo, permite trabajar con equipos de procesamiento lento sin tener que bajar la velocidad de comunicación siempre que tengan buffers de almacenamiento de los datos recibidos. Los inconvenientes son que requiere un mayor ancho banda que MODBUS RTU para el envío de la misma petición o respuesta, o visto de otra manera, para el mismo ancho de banda, el envío de una trama con ASCII es más lento que con RTU.

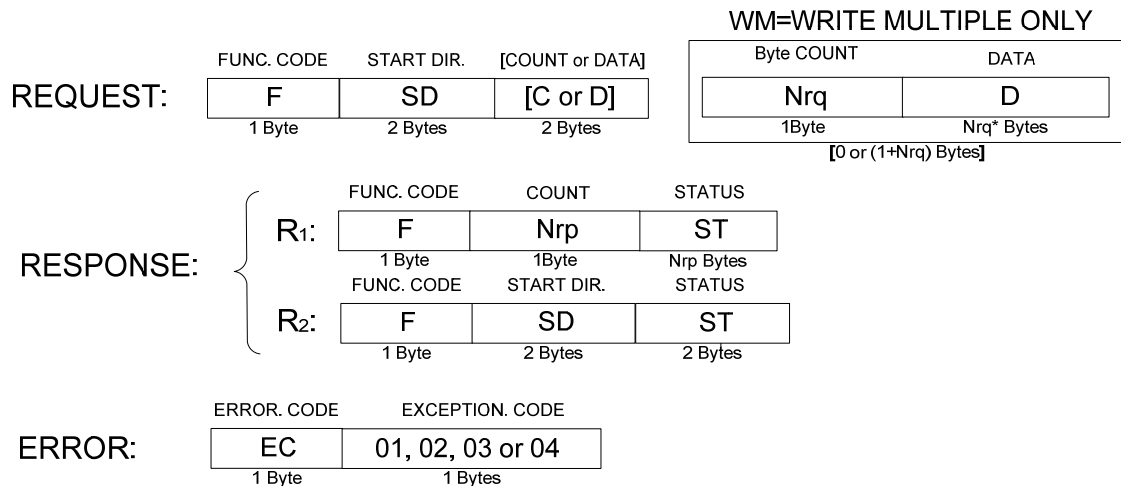
Funciones y registros

La siguiente tabla muestra las funciones más utilizadas en las peticiones y respuestas de MODBUS, con sus códigos.

Códigos de función más comunes de MODBUS			
Código decimal	Código hexadecimal	Función	Tipo de datos
1	16#01	Leer estado de marcas y salidas digitales (bobinas)	Bit
2	16#02	Leer estado de entradas digitales	Bit
3	16#03	Leer registros	Entero 16 bits
4	16#04	Leer entradas analógicas	Entero 16 bits
5	16#05	Forzar valor de una salida digital (bobina)	Bit
6	16#06	Establecer valor de un registro	Entero 16 bits
15	16#0F	Forzar múltiples marcas o salidas digitales (bobinas)	Bit
16	16#10	Establecer múltiples registros	Entero 16 bits

El formato de los campos de función y de datos de las tramas de MODBUS (ver apartado 4.1.1) depende de la función utilizada. La Figura 15 resume el patrón de esos campos para las tramas de petición (request), respuesta (response) y error. Esta última trama es un caso especial de respuesta enviada por un esclavo cuando tiene problemas para atender una petición. A continuación se describen los símbolos utilizados en el esquema de la Figura 15.

- **FUNC. CODE (F):** Código de la función MODBUS a ejecutar en el esclavo.
- **START DIR. (D):** Dirección de inicio del primer objeto de datos afectado por la función.
- **COUNT (C):** Cantidad de objetos a leer o escribir.
- **DATA (D):** Datos a escribir en los registros u objetos del dispositivo.
- **SATUS (ST):** Valor actual de los objetos del dispositivo. Permite verificar que una escritura ha sido realizada correctamente, o, en caso de lectura permite obtener los datos leídos.
- **ERROR. CODE (EC):** Código de error de MODBUS. Se corresponde normalmente al valor 16#80 más el valor de la función que originó el error.
- **EXCEPCION. CODE:** Código de excepción (indica que ha causado el error).
- **ACR:** Es un acrónimo que se compone de tres letras, la primera es si se trata de un comando de lectura (**R**) o escritura (**W**) en el dispositivo. La segunda se refiere al tipo de objeto de datos sobre el que actúa dicho comando **C**: *Coil* o bobina (1 bit), **D**: *Discrete* o entrada digital (1 bit), **R**: *Register* o registro (16 bits), **I**: *Input* o entrada analógica (16 bits). Y la última letra se corresponde con una **M** (*Multiple*) si la función actúa sobre múltiples objetos y **S** (*Single*) si actúa sobre uno.



F	SD	C	WM	Response	EC	ACR
0X01	0x0000 .. 0xFFFF	1 .. 2000(0x07D0)	NO	R1 - Nrp=(C/8)	0X80	RCM
0X02	0x0000 .. 0xFFFF	1 .. 2000(0x07D0)	NO	R1 - Nrp=(C/8)	0X82	RDM
0X03	0x0000 .. 0xFFFF	1 .. 125(0x007D)	NO	R1 - Nrp=2*C	0X83	RRM
0X04	0x0000 .. 0xFFFF	1 .. 125(0x007D)	NO	R1 - Nrp=2*C	0X84	RIM
0X05	0x0000 .. 0xFFFF	0x0000 or 0xFF00	NO	R2	0X85	WCS
0X06	0x0000 .. 0xFFFF	0x0000 .. 0xFFFF	NO	R2	0X86	WRS
0X0F	0x0000 .. 0xFFFF	0x0000 .. 0x07B0	Nrq=(C/8)	R2	0x8F	WCM
0X10	0x0000 .. 0xFFFF	0x0000 .. 0x07B0	Nrq=2*C	R2	0x90	WMM

Figura 15. Formato de las peticiones y respuestas de MODBUS

Respecto a las direcciones de registros, cabe destacar que el maestro tiene que indicar que salidas, entradas, registros o conjunto de estos van a ser afectados por la función enviada. Para ello, el maestro debe enviar al esclavo la dirección del primer objeto de datos (bobina, registro o entrada). Además existen funciones que actúan sobre múltiples objetos al mismo tiempo por lo que en este caso también se requiere el número de objetos implicados.

El problema consiste en saber, para un dispositivo concreto, con qué dirección se corresponde un determinado parámetro. Para ello, MODBUS define una tabla de correspondencias entre las direcciones representables por MODBUS, las direcciones físicas del dispositivo y las direcciones del formato IEC61131. Como siempre, es responsabilidad del fabricante seguir esta tabla de correspondencias o no, por lo que es recomendable tener a mano las especificaciones del fabricante sobre el mapa de memoria del dispositivo y donde se están almacenando cada uno de los datos, ya que de otro modo habría que realizar ingeniería inversa para obtener esta información.

Tabla de correspondencias de registros de MODBUS					
Función	Dirección MODBUS	Dirección en dispositivo	Dirección IEC61131	Tipo de registro	Tipo de acceso
16#01, 16#05, 16#0F	0 a 9.999	1 a 10.000	%M0, %M1...	Salidas o registros de aplicación digitales (bits)	Lectura y escritura
16#02	0 a 9.999	10.001 a 20.000	%I0, %I1...	Entradas digitales (bits)	Lectura
16#04	0 a 9.999	30.001 a 40.000	%IW0, %IW1...	Entradas analógicas (entero)	Lectura
16#03, 16#06, 16#10	0 a 9.999	40.001 a 50.000	%MW0, %MW1...	Registro general de la aplicación (entero)	Lectura y escritura

Nótese que cada una de las direcciones del mapa de memoria de un dispositivo, se corresponde con un par (función, dirección) en MODBUS. Esto quiere decir que, para leer la dirección 10.001, habría que usar el par (0x02, 0x0001), puesto que con la misma dirección, pero con distinta función, se está accediendo a una dirección de dispositivo diferente.

Como ejemplo del formato de una petición de MODBUS, la Figura 16 muestra un ejemplo del formato que tendría una trama RTU con una petición de lectura del estado de varias entradas (función 16#02) al esclavo número 21 (16#15). La petición hace referencia a la entrada con dirección de dispositivo 10.931 y a las 4 siguientes, es decir, se aplica a 5 entradas. Para indicar la dirección 10.931, al tratarse de una entrada, hay que indicar el valor $10.931 - 10.001 = 930$ (16#03A2) en la trama. Según la notación IEC61131 se está accediendo a las entradas %I930 a %I934.

Petición:



Respuesta:

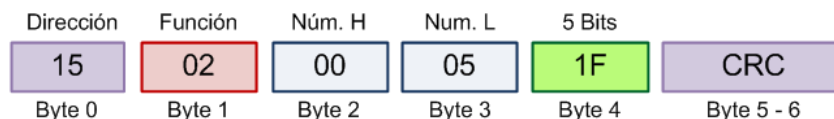


Figura 16. Ejemplo de tramas de petición y respuesta de MODBUS-RTU

La Figura 16 también muestra la respuesta que enviaría el esclavo, y que contiene los valores de los bits de las 5 entradas, que se empaquetan en un mismo byte. En el ejemplo se considera que todas las entradas están activadas, por lo que se devuelve el byte 16#1F (0001 1111).

4.2. Tabla de gestión de intercambios de los PLCs M340

La **tabla de gestión de intercambios** es una variable vector compuesta de cuatro palabras enteras (ARRAY [0..3] OF INT), que permite controlar la ejecución y conocer el estado de la transmisión iniciada con funciones de comunicaciones como READ_VAR. Cada función de comunicación en el programa debe tener su propia variable de tabla de gestión, ya que varias funciones de comunicación se pueden ejecutar a la vez (hasta 16 en una CPU BMX P34 2020), y cada función debe mantener su propio estado. La tabla de gestión se compone de las siguientes palabras:

- Palabra 1 (índice 0): Palabra administrada por el sistema que se compone de dos bytes:
 - Byte de menor peso: Contiene el **Bit de actividad** (bit 0), que está a 1 mientras se ejecuta la función, y el **Bit de cancelación** (bit 1), que se puede poner a 1 para forzar la cancelación anticipada de la transmisión, según se describe en el apartado 2.4.1.
 - Byte de mayor peso: Representa el **Número de intercambio**, que sirve para identificar la transmisión.
- Palabra 2 (índice 1): Palabra administrada por el sistema que se compone de dos bytes:
 - Byte de menor peso: Contiene el **Informe de comunicación**, que especifica el resultado de la operación de transmisión y tiene significado cuando el bit de actividad pasa de 0 a 1. El valor 0 indica una transmisión correcta. El resto de valores indican posibles fallos en la transmisión, conforme se describe en el Anexo del apartado 4.

- Byte de mayor peso: Contiene el **Informe de operación**, cuyo valor detalla los motivos de posibles fallos en la transmisión, según se describe en el Anexo en el apartado 4.
- Palabra 3 (índice 2): Es una palabra gestionada por el usuario, que define el tiempo máximo (**Timeout**) que el maestro esperará la respuesta desde el esclavo, con una base de tiempo de 100 ms. El valor 0 indica una espera infinita. Debe indicar un valor mayor el resultado de multiplicar los valores configurados como Retardo de respuesta y Numero de reintentos (ver apartado 2.2).
- Palabra 4 (índice 3): Palabra gestionada por el usuario, que sirve para indicar el número de bytes enviados o recibidos con algunas funciones, como DATA_EXCH.

El **informe de comunicación** es un valor devuelto por las funciones de comunicaciones de un PLC M340, y es significativo cuando el valor del bit de actividad de la función cambia de 1 a 0. Los valores entre 1 y 254 (16#FE) se refieren a errores detectados por el procesador que ejecutó la función. La siguiente tabla indica los diferentes valores que pueden obtenerse:

Valor	Significado del informe de comunicación
16#00	Intercambio correcto
16#01	Detención del intercambio al producirse un exceso en el tiempo de espera (timeout)
16#02	Detención del intercambio a petición del usuario (CANCEL)
16#03	Formato de dirección incorrecto
16#04	Dirección de destino incorrecta
16#05	Formato incorrecto de parámetro de gestión
16#06	Parámetros específicos incorrectos
16#07	Problema en el envío al destino
16#08	Reservado
16#09	Tamaño del búfer de recepción insuficiente
16#0A	Tamaño del búfer de envío insuficiente
16#0B	Sin recursos de sistema del procesador
16#0C	Número de intercambio incorrecto
16#0D	Ningún telegrama recibido
16#0E	Longitud incorrecta
16#0F	Servicio de telegramas sin configurar
16#10	Módulo de red ausente
16#11	Petición ausente
16#12	Servidor de la aplicación ya activo
16#13	Número de transacción UNI-TE V2 incorrecto
16#FF	Mensaje rechazado, o intercambio correcto tras utilizar WRITE_VAR en una petición de broadcast MODBUS

El valor de **informe de operación** es específico de cada función y especifica el resultado de la operación en la aplicación remota. Sólo es significativo si el informe de comunicación tiene los valores 0 (16#00) o 255 (16#FF).

Valor	Significado del informe de operación cuando el informe de comunicación es 0
16#00	Resultado positivo
16#01	Petición no procesada
16#02	Respuesta incorrecta
16#FB	Si existe una respuesta a solicitudes menores
16#FD	Error operativo
16#FE	Si la respuesta es positiva para ciertas solicitudes

Valor	Significado del informe de operación cuando el informe de comunicación es 255
16#01	No hay recursos respecto al procesador
16#02	No hay recursos de línea
16#03	Sin dispositivo o bien dispositivo sin recursos (para dispositivos extraíbles)
16#04	Error de línea
16#05	Error de longitud
16#06	Canal de comunicación defectuoso
16#07	Error de direccionamiento
16#08	Error de aplicación
16#0B	Sin recursos de sistema
16#0C	Función de comunicación inactiva
16#0D	Destino ausente
16#0F	Problema de acceso entre estaciones o canal sin configurar
16#11	Formato de dirección no gestionado
16#12	Sin recursos de destino
16#14	Conexión no operativa (ejemplo: Ethernet TCP/IP)
16#15	Sin recurso en el canal local
16#16	Acceso no autorizado (ejemplo: Ethernet TCP/IP)
16#17	Configuración de red incoherente (ejemplo: Ethernet TCP/IP)
16#18	Conexión no disponible temporalmente
16#21	Servidor de la aplicación detenido
16#30	Error de transmisión

4.3. Información de depuración de comunicación serie en UnityPro

Con UnityPro se pueden monitorizar la información básica de una comunicación serie cuando la aplicación está conectada a una CPU M340 en ejecución. Esta información está disponible en la pantalla de depuración para MODBUS serie, a la cual se accede a través de la configuración del puerto serie (ver sección 2.2). En esa pantalla, se dispone de dos áreas que muestran el modo de trabajo de la CPU, maestro o esclavo, y un área de contadores como la mostrada en la Figura 17.

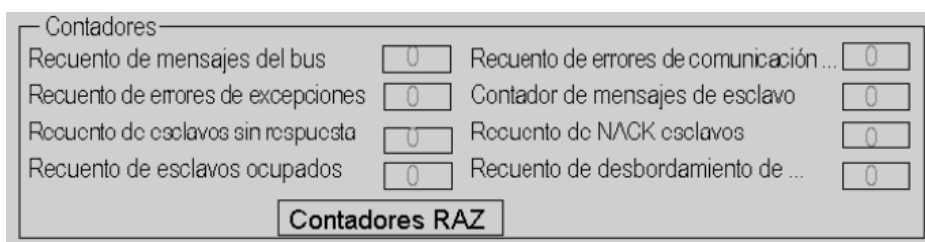


Figura 17. Ejemplo de tramas de petición y respuesta de MODBUS-RTU

Los contadores de depuración de la comunicación MODBUS serie son los siguientes:

- Contador de mensajes del bus: este contador indica el número de mensajes que el procesador ha detectado en la conexión serie. No incluye mensajes con un resultado negativo de CRC.
- Contador de errores de comunicación del bus: este contador indica el número de resultados negativos de CRC que ha contado el procesador. Si se detecta un error de caracteres (error de paridad, desborde), o el mensaje es inferior a tres bytes de longitud, el sistema que recibe los datos no puede calcular el CRC. En estos casos, el contador se incrementa en consecuencia.
- Contador de errores de excepción de esclavo: este contador indica el número de errores de excepción de MODBUS detectados por el procesador.

- Contador de mensajes de esclavo: este contador indica el número de mensajes recibidos y procesados por la conexión MODBUS.
- Contador de esclavos sin respuesta: este contador indica el número de mensajes enviados de los que no ha recibido respuesta (ni una respuesta normal ni una respuesta de excepción). Incluye además el número de mensajes recibidos en modalidad Broadcast.
- Contador de confirmaciones de esclavo negativas: este contador indica el número de mensajes enviados al sistema remoto para los que ha devuelto una confirmación negativa.
- Contador de esclavos ocupados: este contador indica el número de mensajes enviados al sistema remoto para los que ha devuelto un mensaje de excepción de esclavo ocupado.
- Contador de desbordamiento de caracteres del bus: este contador indica el número de mensajes enviados al procesador que no pueden adquirirse debido al desbordamiento de caracteres del bus. El desbordamiento ha sido provocado por uno de estos motivos: Datos de tipo carácter que se transmiten por el puerto serie más rápidamente de lo que pueden almacenarse, o pérdida de datos debida a una anomalía del hardware.

Para todos los contadores, la cuenta comienza en el reinicio, la operación de borrado de contadores o el arranque del procesador más reciente.

4.4. Tabla de códigos ASCII

Para la modalidad ASCII de MODBUS, conviene tener a mano la siguiente tabla con los códigos ASCII del 0 al 127, que son usados en las tramas del protocolo.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	H	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

5. Referencias

- Modicon M340 con Unity Pro. Conexión serie. Manual del usuario. Archivo disponible en el Campus Virtual (CRI_M340-Serie.PDF).
- MODBUS Organization. En su web se puede encontrar las especificaciones del protocolo MODBUS, así como herramientas software. <http://www.MODBUS.com/>
- Manuales de prácticas de la asignatura Automatización Avanzada del Máster en Automática y Robótica.
- Ayuda de la aplicación UnityPro XL.