

Comunicación por protocolo Modbus Ascii mediante arduino

Communication via Modbus Ascii protocol through arduino

DOI: 10.46932/sfjdv3n4-077

Received in: April 14th, 2022

Accepted in: June 30th, 2022

Mariano Garduño Aparicio

Doctor en Ciencias en Ingeniería Electrónica

Institución: Universidad Autónoma de Querétaro (UAQ)

Dirección: Cerro de las Campanas, s/n, Querétaro Qro. México, CP 76010

Correo electrónico: magaap@yahoo.com.mx

José Luis González Córdova

Doctor en Mecatrónica

Institución: Universidad Autónoma de Querétaro (UAQ)

Dirección: Cerro de las Campanas, s/n, Querétaro Qro. México, CP 76010

Correo electrónico: maggaap@hotmail.com

Edgar Rivas Araiza

Doctor en Ciencias en Instrumentación

Institución: Universidad Autónoma de Querétaro (UAQ)

Dirección: Cerro de las Campanas, s/n, Querétaro Qro. México, CP 76010

José Luis Avendaño Juárez

Maestría en Ciencias en Instrumentación y Control

Institución: Universidad Autónoma de Querétaro (UAQ)

Dirección: Cerro de las Campanas, s/n, Querétaro Qro. México, CP 76010

RESUMEN

En el presente trabajo se muestra la forma de comunicación de tarjetas Arduino con dispositivos industriales como PLCs o servodrives que utilicen el protocolo de comunicación MODBUS ASCII, se explica la constitución de las tramas utilizadas en servodrives, además de los algoritmos propuestos para dicha comunicación, usando un microcontrolador Atmel como dispositivo maestro y que emplea sus puertos seriales bajo diferentes parámetros de modo asíncrono y diferentes velocidades de comunicación.

Palabras clave: Modbus Ascii, arduino, protocolos de comunicación, servodrives.

ABSTRACT

This paper shows how to communicate Arduino cards with industrial devices such as PLCs or servodrives using the MODBUS ASCII communication protocol, explains the constitution of the frames used in servodrives, in addition to the algorithms proposed for such communication, using an Atmel microcontroller as the master device and using its serial ports under different parameters of asynchronous mode and different communication speeds.

Keywords: Ascii modbus, arduino, communication protocols, servodrives.

1 INTRODUCCIÓN

En la actualidad el uso de las herramientas para control industrial se ha vuelto muy importante y necesario, la mayoría de industrias hacen uso de los últimos controladores y tarjetas embebidas para el funcionamiento de sus sistemas de producción, dicha utilización de tarjetas embebidas ha ido en aumento, debido a su facilidad de programación, manejo, capacidades de utilización, aplicaciones y su costo relativamente reducido [1-4].

Una tarjeta embebida como Arduino considera muchos aspectos que le hacen fácilmente útil para el desarrollo de aplicaciones de control, instrumentación, comunicación y aplicación industrial, aspectos como son: contar con un dispositivo programador compatible, compilador y lenguaje de programación de alto nivel, software para desarrollo de fácil manejo, hardware de plataforma abierta, diferentes tipos de tarjetas según las necesidades a cubrir, alto rendimiento, fácil adquisición, bajo costo, entre otras [5-10].

Las tarjetas Arduino utilizan un microcontrolador Atmel, con diferentes prestaciones de Hardware como lo son puertos de entrada/salida digitales, ADCs, DACs, comparadores analógicos, PWM, timers, memoria flash, sram, eeprom, puertos seriales UART, SPI, USB, etc. Dependiendo del tipo de tarjeta son sus características internas de procesamiento e interface, el lenguaje con el cual pueden ser programadas puede ser lenguaje C, C++, AVR, lenguaje arduino (processing and wiring) y existen casos especiales de arduinos con LabView, Linux (OpenWrt), python entre otros, con lo que se convierte en uno de los microcontroladores más versátiles y aplicables del mercado [7, 9, 11, 12].

Un servodrive es un sistema electrónico que permite el control de velocidad, torque, inercia, arranque de motores eléctricos, ya sean de corriente directa (CD) o de corriente alterna (CA), un servo sistema trabaja mediante retroalimentación negativa permitiendo que la salida sea controlada según un valor deseado (setpoint), por ejemplo para el caso de controles de velocidad con un valor de setpoint se determina la velocidad de trabajo de un motor, ya sea un valor constante o variable, incluso modificar el tiempo en el que se requiere que el motor llegue a la velocidad requerida (rampa de aceleración), a su vez para el tiempo de frenado conocido como rampa de desaceleración, para todas estas acciones el servodrive requiere de parámetros de inercia, torque, etc. con los que se calculan los demás parámetros internos del lazo de control [13-15]. En los últimos años ha aumentado mayormente el uso de servodrives para motores de CA, esto a causa de que permiten un control más fino de la velocidad/torque y menor gasto de energía, a comparación de los motores de CD, el funcionamiento de un motor de CA se basa en la variación de frecuencia de las señales trifásicas que se aplican al motor de CA, y en otros casos también se varia la amplitud de los voltajes trifásicos de alimentación [16, 17].

Modbus es un protocolo de comunicaciones que está situado en el nivel 7 del modelo OSI (Modelo de Interconexión de sistemas abiertos) y es el protocolo que ha gozado de mayor uso y en muchos casos como un estándar de fábrica de múltiples dispositivos industriales, ha conseguido esto por tres

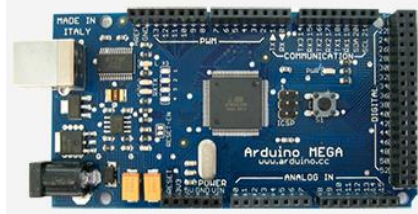
características importantes: Su uso y conceptos son públicos, su implementación puede ser sencilla de realizar y maneja bloques de datos sin suponer restricciones [18]. El protocolo Modbus tiene dos variantes el modo ASCII y el modo RTU, en el modo RTU se utiliza un control de redundancia cíclica (CRC) al final de la trama, mientras que el modo ASCII emplea un control de redundancia longitudinal (LRC), ambos con finalidad de evitar errores en los valores que se envían o se reciben, asegurando el entendimiento exacto entre los dispositivos [4].

2 DESCRIPCIÓN DEL MÉTODO

2.1 ARDUINO MEGA

Una de las tarjetas arduino más utilizadas es el llamado arduino Mega, su versatilidad ofrecida se debe en gran medida a su gran cantidad de puertos de entrada y salida (54 pines digitales I/O), 16 entradas analógicas, 4 puertos seriales UART en hardware, etc. Por lo que si se utiliza un puerto serie para comunicación con protocolo modbus a dispositivos industriales le quedan otros 3 puertos para comunicarse igual de forma serial con una computadora, con una tarjeta bluetooth o con un módulo Wi-Fi pudiéndose realizar múltiples aplicaciones interactivas, inalámbricas o mediante internet. En la figura 1 se muestra el Arduino Mega, recordando que se trata de una plataforma de hardware y software libre, por lo que puede ser aplicado a múltiples proyectos con relativa facilidad de uso y manejo [7].

Figura 1. Tarjeta Embebida Arduino Mega



2.2 PROTOCOLO MODBUS ASCII

En un red Modbus cada dispositivo posee una única dirección en la red, desde la cual se puede comunicar enviando o recibiendo información, además de que la comunicación se puede realizar en forma asíncrona, en la cual los datos se transmiten en serie (uno después del otro) y cada dispositivo lo reconoce según su ciclo de reloj interno, Los datos están encapsulados en paquetes entre un bit de inicio (start) y un bit de fin (stop), como se muestra en la figura 2, para este ejemplo se envían 8 bits en cada paquete o lo que equivale a un carácter [4].

Figura 2. Forma de paquete de datos en la comunicación serial asíncrona.



Una trama Modbus se constituye de varios caracteres como por ejemplo el envío de datos a un servodrive para motor de CA puede ser de la siguiente forma :050610230001C1\r\n, en la cual los datos que se envían están entre los caracteres “:” y “\r\n” el carácter “:” marca el inicio de los datos, mientras “\r\n” son el fin de la información equivalen a un salto y retorno de línea; en este ejemplo mostrado los caracteres “05” son el valor de la dirección a donde se envía la información que equivale al dispositivo número 5 de la red Modbus, los caracteres “06” son el número de instrucción a realizar en el dispositivo, en este caso el 6 indica escritura, donde “1023” es la dirección donde escribirá el valor de “0001” y finalmente el valor de “C6” es el código de verificación de redundancia longitudinal (LRC) que se obtiene del cálculo de las operaciones binarias siguientes $LRC = (FF - (05 + 06 + 10 + 23 + 00 + 01)) + 1 = C1$, notándose que el valor 1023 se divide en dos para que la suma sea en grupos de 8 bits.

Con el ejemplo anterior se puede también observar que el cálculo del valor LRC se puede realizar en forma simple por el microcontrolador y después realizar el envío convirtiendo estos valores numéricos a una cadena de caracteres ASCII.

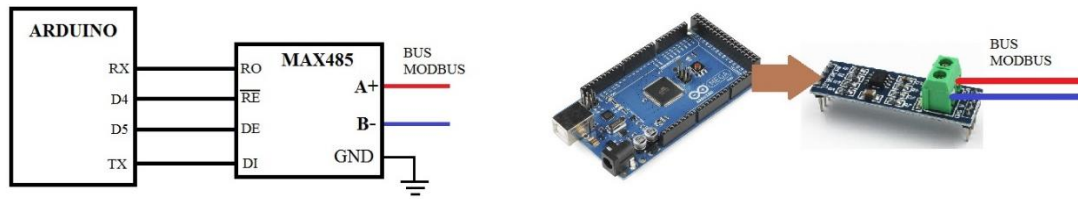
La otra operación del protocolo modbus más utilizada es la instrucción de lectura, la cual llevaría un “03” como en el siguiente ejemplo :040300010001F7, donde “04” sería para leer el dispositivo número 4 de la red modbus, seguida por la instrucción de lectura (03), después la dirección a leer que es la “0001”, seguido por la cantidad de “0001” bytes y finalmente el valor de redundancia longitudinal que en este caso es un “F7”.

2.3 INTERFACE RS485

Para conectar los dispositivos en una red Modbus es necesario que las conexiones y valores de voltaje en la transmisión de datos estén estandarizados con la norma ANSI/TIA/EIA-485, que es mejor conocida como interface RS485, con la que se pueden realizar enlaces multipunto a través de largas distancias y con una reducción al ruido notable, los drivers RS485 utilizan los tres estados lógicos, permitiendo que los transmisores individuales (dispositivos) se desactiven de forma natural y con ello solamente utilizar dos cables para la transmisión de información, empleando topologías de bus lineal para conectar los dispositivos de la red [18].

Un microcontrolador con puerto serial UART puede ser conectado a una red Modbus utilizando el driver MAX485, conectando las salidas seriales del microcontrolador RX con RO y TX con DI del driver MAX485, las entradas $\overline{RE}$ y DE son las habilitaciones de lectura o de escritura hacia la red, estos pines pueden ser también controlados por el microcontrolador, para decidir en qué momento se está escribiendo o leyendo, esto se muestra en la figura 3 y adicionalmente se menciona que ya existen módulos en el mercado para la realización de esta conexión.

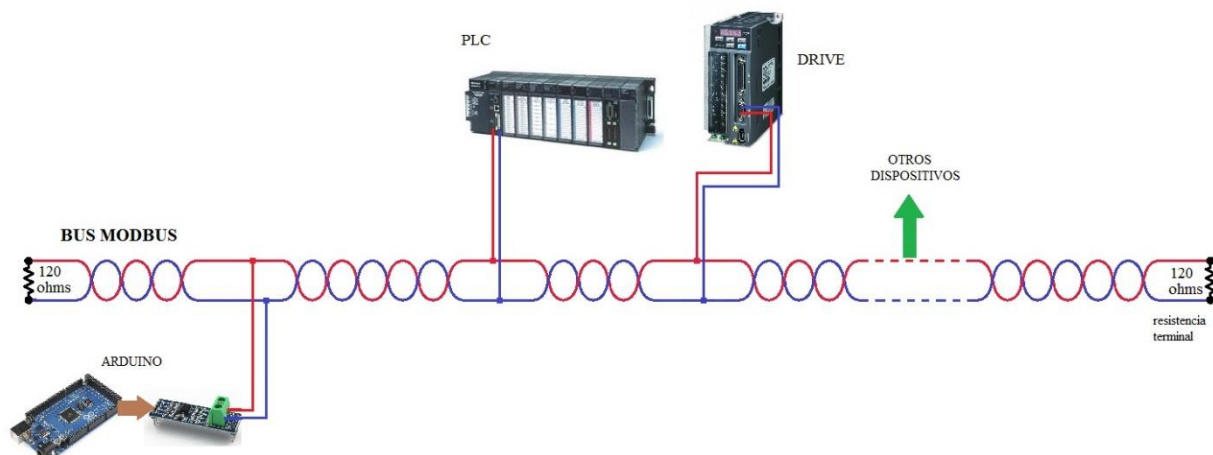
Figura 3.- Uso de MAX485 para conectar el arduino con una red Modbus.



2.4 CONEXIONES DE UNA RED MODBUS

La capa física del protocolo Modbus está constituido por las conexiones entre los dispositivos de la red, los cuales como ya se mencionó deben de estar conectados en una topología de bus lineal, así como se puede observar en la figura 4, en este caso un dispositivo maestro como lo sería el microcontrolador arduino puede, según sea el caso asumir el papel de maestro y controlar la comunicación con los demás dispositivos de la red, enviando y recibiendo la información para determinadas tareas de control. En el cableado que es mediante par trenzado se deben considerar mucho las resistencias terminales normalmente sugeridas de 120 ohms, para un buen rechazo al ruido y la eficiencia en la comunicación [18, 19].

Figura 4.- Uso de MAX485 para conectar el arduino con una red Modbus.



2.5 LIBRERÍA DESARROLLADA PARA PROTOCOLO MODBUS EN ARDUINO

Se desarrolló entonces una librería para Arduino con la finalidad de realizar las operaciones numéricas binarias para el código LRC (código de redundancia longitudinal), convertir esos valores numéricos en caracteres y a su vez construir todo el string, validar el rango de valores de entrada del programa, para evitar un problema en la trama final y que se cause un problema en el dispositivo final, este rango de valores puede variar según el tipo de dispositivo terminal.

La librería consta entonces de dos archivos, uno es el llamado TramasMB.h que contiene todas la cabeceras y variables de la librería, mostrado en la Tabla 1 y otro el archivo llamado TramasMB.cpp que contiene todos los procesos matemáticos, las conversiones y la construcción del string final que es la trama

ya completa, este último archivo es mostrado en la Tabla 3 en el anexo. Mientras en la Tabla 2 es un sketch de ejemplo de utilización de la librería.

TABLA 1.- Librería TramasMB.h para que el arduino pueda comunicarse en Modbus.					
1	/* TramasMB.h - librería para	Comentario	20	String_elem4=""; //bytes	Elementos de
2	construir la trama que se envía a	inicial	21	String_elem5=""; //código de	entrada
3	servodrives o PLCs*/		22	comprobación	
4	#ifndef TramasMB_h	Definiciones	23	String_fin="\r\n";	
5	#define TramasMB_h	principales	24		
6	#include "Arduino.h"	Incluir librería	25	int_numins=0;	Variables internas
7	class TramasMB {	Arduino	26	int_code=0;	
8	public:	Creación de	27	int_direc1=0;	
9	TramasMB();	clase	28	int_direc2=0;	
10	String servoR(int numservo,		29	int_bytes1=0;	
11	int direc, int bytess);		30	int_bytes2=0;	
12	String servoW(int numservo,	Función servoR	31	int_suma=0;	
13	int direc, int bytess);	(operación de	32	};	
14	private:	escritura)	33	#endif	
15	String_tempo="hh";	Función servoW			Fin de asignaciones
16	char_inicio=':';	(operación de			
17	String _elem1="";	lectura)			
18	//numServo	Variables			
19	String_elem2=""; //numIns (3	internas			
	para lectura = Read)	String inicial			
	String_elem3=""; //adress	Elementos de			
		entrada			

Un ejemplo de utilización de esta librería es como se muestra en el sketch de la Tabla 2, en el cual se configuran a dos puertos seriales uno para ver en la computadora que el código se está ejecutando (línea 10), mientras el otro serial se configura para el manejo del protocolo modbus ASCII (línea 11), notándose que este serial se configura para ocho bits, sin paridad pero con 2 bits de stop, esta configuración puede cambiar según el dispositivo que se vaya a controlar con el uso de la red modbus. En las líneas 12 a 15 se configuran dos pines digitales para controlar las entradas $\overline{RE}$ y DE del integrado MAX485 y que pueda configurarse para el envío de valores.

En la línea 22 del sketch mostrado en la Tabla 2, se hace uso de la función servoR como Tramas.servoR(2,2071,1), que es en realidad la lectura de la dirección 2071 (hexadecimal) en el dispositivo 2 de la red Modbus y se lee un byte de información, esto puede ser fácilmente utilizado cuantas veces se requiera y con los valores que se necesiten, solo se tiene que utilizar la forma Tramas.servoR (servo, address, bytes) para invocar a dicha función.

TABLA 2.- Código Del programa para que el arduino pueda comunicarse en Modbus.					
1	#include <TramasMB.h>	Llamado a la	20	void loop()	Bucle infinito
2		librería	21	{	
3	TramasMB Tramas;		22	trama=Tramas.servoR(2,2071,1);	Uso de la
4	String trama="";	Creación de	23		función de la
5		objeto	24	Serial1.print(trama);	librería
6	int RE=4;	String a utilizar	25		Envío de la
7	int DE=5;		26		trama por el
8		Uso de pin 4	27	delay(2000);	serial utilizado
9	void setup() {	para RE	28	}	para modbus.
10	Serial.begin(9600);	Uso de pin 5	29		Delay de 2
11	Serial1.begin(9600,	para DE			segundos.
12	SERIAL_8N2);	(MAX485)			Se repite
13	pinMode(RE,OUTPUT);				continuamente
14	pinMode(DE,OUTPUT);	Serial de la PC.			el loop.
15	digitalWrite(RE,LOW);	Serial utilizado			
16	digitalWrite(DE,HIGH);	para Modbus			Los valores a
17	Serial1.println("probando	Uso de pines de			enviar hacia la
18	envio");	salida			función tienen la
19	Serial.println("escriba una	Inicialización de			forma:
	letra: ");	pines de salida			Tramas.servoR
	delay(200);	Inicio de envio			(servo, address,
	}				bytes)

También con la librería se pueden obtener tramas para realizar operaciones de escritura en los dispositivos de la red Modbus, el cambio que se tiene que realizar es el utilizar ahora la función servoW, la cual se podría llamar como: Tramas.servoR(2,11,1) por decir un ejemplo; en el cual se escribe un valor uno en la dirección 11 del dispositivo número dos de la red Modbus.

3 COMENTARIOS FINALES

La librería propuesta ha sido probada para múltiples valores y utilizada para la comunicación con diferentes dispositivos, las conexiones son muy importantes y permiten utilizar tarjetas embebidas como el caso de arduino para controlar, manipular y analizar dispositivos industriales como lo son PLCs, drives de motores, sensores inteligentes, etc.

4 RESUMEN DE RESULTADOS

El control de dispositivos industriales fue posible de realizar con un microcontrolador arduino Mega, por medio del cual se puede establecer comunicación basada en el protocolo Modbus ASCII, con el cual se pueden hacer fácilmente las operaciones más utilizadas en este tipo de red, que es la lectura y escritura de valores en cualquier dispositivo de la red, siempre que este configurado como esclavo y se tenga acceso a sus registros de trabajo.

5 CONCLUSIONES

El microcontrolador ATMEL en las tarjetas embebidas como el arduino puede ser utilizado para comunicarse con dispositivos industriales que utilicen protocolo Modbus, por medio del ajuste correcto de los parámetros de comunicación serial, la conversión de sus señales RX y TX por medio del integrado MAX485 y conectándose a una red modbus como dispositivo maestro, con la finalidad de analizar lo que suceda en esa red Modbus o controlar a los dispositivos industriales que se conecten en ella, por lo que se pueden conseguir mayores y mejores aplicaciones para esta tarjeta embebida.

RECOMENDACIONES

En el caso de las comunicaciones industriales es necesario considerar múltiples factores que pueden alterar la comunicación entre dispositivos, en primer lugar las características de comunicación, como son: velocidad de transferencia de información, bit de paridad, bits de parada, bits de envío, sincronización o no sincronización, reducción de ruido en las conexiones, longitud entre los dispositivos, ruidos externos, resistencias terminales, etc. Es necesario en todos estos casos conocer y distinguir las bases primordiales para que se realice la comunicación entre dispositivos, sus configuraciones de fábrica, establecimiento de dispositivos maestro y esclavos, modo de comunicación y conexiones requeridas para cada tipo de transferencia de información.

REFERENCIAS

- [1] Pérez, F.E.V. and R.P. Areny, *Microcontroladores: fundamentos y aplicaciones con PIC*. 2007: Marcombo.
- [2] Li, Q. and C. Yao, *Real-Time Concepts for Embedded Systems*. 2003: Taylor & Francis.
- [3] Heath, S., *Embedded Systems Design*. 2002: Elsevier Science.
- [4] Axelsson, B. and G. Easton, *Industrial Networks: A New View of Reality*. 2016: Taylor & Francis.
- [5] Reyes, F., J. Cid, and E. Vargas, *Mecatrónica Control y Automatización*. 2013, México: Alfaomega Grupo Editor.
- [6] Ozer, J. and H. Blemings, *Practical Arduino: Cool Projects for Open Source Hardware*. 2009: Apress.
- [7] Artero, Ó.T., *ARDUINO Curso práctico de formación*, ed. Alfaomega. Vol. 1. 2013: Alfaomega.
- [8] Perea, F., *Arduino Essentials*. 2015: Packt Publishing, Limited.
- [9] Purdum, J., *Beginning C for Arduino: Learn C Programming for the Arduino and Compatible Microcontrollers*. 2012: Apress.
- [10] Leaver, C., *Introduction to Atmel AVR Microcontroller Development: Using Free Software with Worked Examples*. 2010: Sylvania Books.
- [11] Schwartz, M. and O. Manickum, *Programming Arduino with LabVIEW*. 2015: Packt Publishing.
- [12] Barrett, S.F. and D.J. Pack, *Atmel Avr Microcontroller Primer: Programming and Interfacing, Second Edition*. 2012: Morgan & Claypool.
- [13] Younkin, G.W., *Industrial Servo Control Systems: Fundamentals And Applications, Revised And Expanded*. 2002: CRC Press.
- [14] Mohan, N., *Electric Drives: An Integrative Approach*. 2003: MNPERE.
- [15] Ogata, K., *Ingeniería de control moderna*. tercera edición. 2003: Pearson Educación.
- [16] Gilberto Enríquez Harper, G.E. and G. Enriquez, *Experimentos con máquinas eléctricas: máquinas rotatorias y transformadores*, ed. primera. 2005: Editorial Limusa S.A. De C.V.
- [17] Richardson, D.V. and A.J. Caisse, *Máquinas Electricas Rotativas y Transformadores*. Cuarta Edición. 1997: Prentice Hall.
- [18] Tanenbaum, A.S., *Redes de computadoras*. 2003: Editorial Alhambra S. A. (SP).
- [19] O, J.É.B., *Prácticas de redes de datos e industriales*. 2009: Universidad de La Salle, Facultad de Ingeniería.

ANEXO

TABLA 3.- Código librería TramasMB.cpp para que el arduino pueda comunicarse en Modbus.				
<pre>/*TramasMB.cpp -Programa para comunicación modbus*/ #include "arduino.h" #include "TramasMB.h" TramasMB::TramasMB() { } String TramasMB::servoR(int numservo, int direc, int bytess) { _numins=3; if (bytess>=0&&bytess<256) { if (direc>=0&&direc<256) {_suma=numservo+_numins+d irec+bytess; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} } else if (direc>=256&&direc<5000) {_direc1=direc/256; _direc2=direc%256; _suma=numservo+_numins+_d irec1+_direc2+bytess; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} if(_suma>768) {_code=256-(_suma-768);} } else { Serial.print("error en la direccion"); } } else if (bytess>=256&&bytess<5000) {_bytes1=bytess/256; _bytes2=bytess%256; _direc1=direc/256; _direc2=direc%256; _suma=numservo+_numins+_d irec1+_direc2+_bytes1+_bytes 2; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} if(_suma>768) {_code=256-(_suma-768);} }</pre>	Comentario inicial Librerías empleadas Asignación de clases y objetos Función servoR para operación de lectura en otro dispositivo Instrucción 3 para lectura Acondicionamiento de bytes Operaciones numéricas Para el previo cálculo del LRC Validación de dirección de lectura Validación de número de bytes para la lectura	<pre>String TramasMB::servoW(int numservo, int direc, int bytess) { //código _numins=6; // instrucción para escritura (6) if (bytess>=0&&bytess<256) { if (direc>=0&&direc<256) {_suma=numservo+_numins+di rec+bytess; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} } else if (direc>=256&&direc<5000) {_direc1=direc/256; _direc2=direc%256; _suma=numservo+_numins+_di rec1+_direc2+bytess; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} if(_suma>768) {_code=256-(_suma-768);} } else { Serial.print("error en la direccion"); } } else if (bytess>=256&&bytess<5000) {_bytes1=bytess/256; _bytes2=bytess%256; _direc1=direc/256; _direc2=direc%256; _suma=numservo+_numins+_di rec1+_direc2+_bytes1+_bytes2; _code=256-(_suma); if(_suma>256) {_code=256-(_suma-256);} if(_suma>512) {_code=256-(_suma-512);} if(_suma>768) {_code=256-(_suma-768);} if(_suma>1024) {_code=256-(_suma-1024);} } else {Serial.print("error en bytes/data R/W");} }</pre>	Función servoW para operación de escritura en otro dispositivo Numero de instrucción igual a 6. Operaciones numéricas Para el previo cálculo del LRC Validación de dirección de escritura Validación de número de bytes para la escritura Conversión de los elementos a String para la formación de la trama	

<pre> if(_suma>1024) { _code=256-(_suma-1024);} } else { Serial.print("error en bytes/data R/W");} _elem1 = String(numservo); if (_elem1.length()==1) { _elem1 = '0'+String(numservo);} else { _elem1 = "00";} _elem2 = '0'+String(_numins); _elem3 = String(direc,HEX); _elem3.toUpperCase(); if (_elem3.length()==1) { _elem3 = "000"+_elem3;} else if (_elem3.length()==2) { _elem3 = "00"+_elem3;} else if (_elem3.length()==3) { _elem3 = '0'+_elem3;} else if (_elem3.length()==4) { _elem3 = _elem3;} else { _elem3 = "0000";} _elem4 = String(bytess, HEX); _elem4.toUpperCase(); if (_elem4.length()==1) { _elem4 = "000"+_elem4;} else if (_elem4.length()==2) { _elem4 = "00"+_elem4;} else if (_elem4.length()==3) { _elem4 = '0'+_elem4;} else if (_elem4.length()==4) { _elem4 = _elem4;} else { _elem4 = "0000";} _elem5=String(_code, HEX); _elem5.toUpperCase(); if (_elem5.length()==1) { _elem5 = '0'+_elem5;} else if (_elem5.length()==2) { _elem5 = _elem5;} else { _elem5 = "00";} _tempo = _inicio +_elem1 +_elem2 +_elem3 +_elem4 +_elem5+_fin; if (_tempo.length()!=17) { _tempo="error L";} return _tempo; } </pre>	Conversión de los elementos a String para la formación de la trama Primero elemento 1 que es el número de dispositivo. Después el elemento 2 que es el número de instrucción. Después el elemento 3 que es la dirección a donde se va a leer o donde inicia la lectura de datos. El elemento 4 que es la cantidad de Bytes que se van a leer desde el dispositivo. El elemento 5 que es el código LRC Construcción final de la trama. El string construido se regresa dentro de la función	<pre> _elem1 = String(numservo); if (_elem1.length()==1) { _elem1 = '0'+String(numservo);} else { _elem1 = "00";} _elem2 = '0'+String(_numins); _elem3 = String(direc,HEX); _elem3.toUpperCase(); if (_elem3.length()==1) { _elem3 = "000"+_elem3;} else if (_elem3.length()==2) { _elem3 = "00"+_elem3;} else if (_elem3.length()==3) { _elem3 = '0'+_elem3;} else if (_elem3.length()==4) { _elem3 = _elem3;} else { _elem3 = "0000";} _elem4 = String(bytess, HEX); _elem4.toUpperCase(); if (_elem4.length()==1) { _elem4 = "000"+_elem4;} else if (_elem4.length()==2) { _elem4 = "00"+_elem4;} else if (_elem4.length()==3) { _elem4 = '0'+_elem4;} else if (_elem4.length()==4) { _elem4 = _elem4;} else { _elem4 = "0000";} _elem5=String(_code, HEX); _elem5.toUpperCase(); if (_elem5.length()==1) { _elem5 = '0'+_elem5;} else if (_elem5.length()==2) { _elem5 = _elem5;} else { _elem5 = "00";} _tempo = _inicio +_elem1 +_elem2 +_elem3 +_elem4 +_elem5+_fin; if (_tempo.length()!=17) { _tempo="error L";} return _tempo; } </pre>	Primero elemento 1 que es el número de dispositivo. Después el elemento 2 que es el número de instrucción. Después el elemento 3 que es la dirección a donde se va a escribir. El elemento 4 que son los datos a escribir en forma de Bytes y que se modificarán en el dispositivo. El elemento 5 que es el código LRC Construcción final de la trama. El string construido se regresa dentro de la función
--	--	--	--