UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Emulador de comunicaciones serie

Autor/es

FELIPE JORDÁN SAAVEDRA

Director/es

JUAN MARTÍN MIRURI SÁENZ

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

EMULADOR DE COMUNICACIONES SERIE

DIRECTOR: Juan Martin Miruri Sáenz

AUTOR: Felipe Jordán Saavedra

Curso: 2016-2017

Resumen

El principal objetivo de este proyecto es desarrollar un software para emular la comunicación de un controlador lógico programable (PLC) mediante el software de programación Borland C++ Builder, utilizándolo en un entorno virtual debido a la difícil adquisición de dicho programa. Dicho emulador evitara la necesidad de conexión con el dispositivo real, respondiendo del mismo modo que este ante las peticiones que aplicaciones externas hagan a través del puerto de comunicaciones serie, todo esto es baso en el protocolo de comunicaciones Host Link de OMRON y específicamente en autómata CQM1/CPM1/CPM1A/SRM1.

Este TFG solo se basa en los comandos Host link que están documentados que facilita OMRON, ya que este fabricante tiene aparte de estos otros comandos que utiliza internamente en sus aplicaciones para el desarrollo de automatización, simulador, supervisión, etc.,.. Intente obtener estos comando pero me resulto muy complicado debido a que los dispositivos envían y reciben mucha cantidad de información en estos comandos que para mí me fue imposible decodificar.

Pero no obstante el emulador responde muy bien a las tramas recibidas de los comandos documentados que le envían aplicaciones externas, analizando el comando, ejecutando y respondiendo.

Abstract

The main objective of this project is to develop software to emulate the communication of a programmable logic controller (PLC) using Borland C ++ Builder programming software, using it in a virtual environment due to the difficult acquisition of said program. This emulator will avoid the need to connect with the real device, responding in the same way as it is before the requests that external applications make through the serial communications port, all this is based on the protocol of communications Host Link of OMRON and specifically in automaton CQM1 / CPM1 / CPM1A / SRM1.

This TFG is only based on the Host link commands that are documented that OMRON facilitates, since this manufacturer has besides these other commands that it uses internally in its applications for the development of automation, simulator, supervision, etc., .. Try to obtain These command but I found it very complicated because the devices send and receive a lot of information in these commands that for me it was impossible to decode.

But nevertheless the emulator responds very well to the received frames of the documented commands that sends him external applications, analyzing the command, executing and responding.

Contenido

Antecedentes

El propósito del proyecto es construir un Emulador o plataforma virtual que permita imitar la experiencia de estar interactuando con un PLC real aunque sea todo virtualmente, mediante una de las comunicaciones establecida por el fabricante “OMRON”, que se basa en comandos de comunicaciones HOST LINK, este protocolo industrial es utilizado por esta marca comercial de autómatas y normalmente se transfiere bajo el medio físico soportado por el puerto serie (RS-232). Este proyecto e decidido realizarlo debido a la curiosidad que me causa todo lo relacionado con la automatización Industrial y más específicamente el tema de comunicaciones, ya que adquirí en las dos asignaturas de Informática Industrial unos conocimientos bastante buenos y me dieron visión o ideas sobre de que podría realizar dicho TFG.

El PC, SCADA, pantallas táctiles HDMI o cualquier dispositivo de supervisión, control y adquisición de datos industrial de OMRON, se encargará de establecer una comunicación tipo *maestro-esclavo* con el autómata y proporcionará una interfaz para controlar diversas funcionalidades del dispositivo, tales como activar y desactivar el autómata, y escritura y lectura de áreas de memoria del mismo.

Objetivos y alcance

La conclusión del proyecto que en este documento se propone, implica la aplicación práctica de conceptos y conocimientos adquiridos durante la carrera en diversas de las asignaturas del Grado en Ingeniería Electrónica, especialmente en las asignaturas de informática industrial que son en las que he adquirido más conocimientos teórico prácticos en relación a dicho TFG. En este trabajo se documenta el desarrollo de un sistema que me permite simular la comunicación de un dispositivo físico real con aplicaciones externas que hacen peticiones de funciones específicas de dicho dispositivo. Este dispositivo (Emulador), software es programado o diseñado todo utilizando el entorno de programación C++ Builder, porque he decidido hacerlo así?, por la sencilla razón de que en las dos asignaturas de informática industrial adquirí unos muy buenos conocimientos sobre las comunicaciones industriales, como poder generarme mis propias aplicaciones, poder supervisar, controlar y adquirir datos, de un PLC sin necesidad de ninguna aplicación externa por el fabricante, y que son relativamente costosas y vienen muy limitadas, de esta manera soy yo y únicamente yo el que podría limitar mi aplicación de control del PLC.

El dispositivo físico real, a simular su comunicación es la gama de autómatas programables de la marca OMRON de la series CQM1/CPM1/CPM1A/SRM1,

Objetivos específicos

El objetivo específico es la obtención de un entorno o programa que estando en ejecución pueda simular el PLC bajo las órdenes de los comandos Hostlink, para ello me he tenido que ir especificando los objetivos:

- Implementar un interface para interactuar con él y poder hacer las depuraciones y pruebas correspondientes

- Como reutilizar programas de las asignaturas cursadas para poder controlar el puerto serie

- Desarrollar o pensar cómo iba a gestionar, ejecutar y responder a los comandos recibidos

- Como depurar los posibles errores que puede tener cada uno de los comandos

- Como iba a mantener estados de memoria en mi emulador para asemejarlo al PLC real

- Desarrollar aplicaciones externas de visualización y control del emulador

INTRODUCCION.

A finales de la década de 1960, la industria buscó una nueva tecnología que sustituyera los sistemas de control basados en circuitos eléctricos con relés. Debida a esta necesidad, la empresa estadounidense Bedford Associates (Bedford, MA) creó el que llamó Modular Digital Controller o MODICON, siendo el MODICON 084 el primer PLC comercial. Desde ese primer MODICON y junto a la aparición y desarrollo del
Microprocesador se ha ido mejorando esta tecnología hasta llegar al PLC que conocemos hoy en día. Actualmente estos controladores son indispensables en la industrial debido a su robustez, su gran número de entradas y salidas y su funcionamiento en tiempo real. Pero su característica principal, a la que se debe su masiva utilización, es la reprogramación, permitiendo el cambio de las secuencias automatizadas por otras con el mismo equipo, potenciando la mejora continua de la industria.
Los PLC's son utilizados para numerosas aplicaciones industriales, sirven tanto para procesos continuos como para los discretos. Con un gran éxito en cualquier sistema sin tener en cuenta la complejidad del mismo.

El presente documento "EMULADOR DE COMUNICACIONES SERIE", ha sido realizado por el alumno D. Felipe Jordán Saavedra, con el objetivo de superar el trabajo de fin de grado y obtener el título de Grado en ingeniería Electrónica Industrial y Automática. Para la realización del mismo se han seguido las indicaciones de D. Juan Martin Miruri Sáenz, que han actuado como director del proyecto.

Las áreas científico técnicas en las que se enmarca este trabajo son, principalmente, el Área de Ingeniería de Sistemas y Automática, y el de Tecnología Electrónica.

1.2 Definición de Autómata (PLC)

Un autómata programable (PLC) es un equipo electrónico diseñado para ser utilizado en un entorno industrial y destinado al control de procesos industriales con un hardware independiente del proceso a controlar. Dicho hardware se adapta al proceso mediante un programa (software), que contiene las instrucciones a realizar. Esta secuencia de operaciones se define sobre una serie de entradas-salidas cableadas directamente al autómata y con las que interactúa con el proceso.

El PLC esta basado en un microprocesador o microcontrolador, que tiene generalmente una configuración modular. Este dispositivo utiliza instrucciones almacenadas en una memoria programable para implementar la lógica, secuenciación, temporización, conteo y funciones aritméticas mediante módulos analógicos o digitales. Su diseño permite controlar en tiempo real y en ambiente industrial máquinas y procesos que presentan una evolución secuencial.
A medida de que ha ido aumentando el mercado de procesos industriales y control, los PLC's se han mostrado como la base sobre la cual se fundamentan estos sistemas, sustituyendo a la tradicional lógica cableada.

Las principal ventaja de la lógica programada frente a la cableada es su robustez y capacidad de interconectividad con los procesos, potenciándolo para comunicación entre si y con sistemas de computación (CIM).

Presentan importantes ventajas innovadoras, tanto en la posibilidad de realizar tareas complejas, las cuales no permitían los sistemas anteriores; por ejemplo: el control integral de procesos y maquinaria en tiempo real; así como la capacidad de comunicación entre equipos, la supervisión y almacenamiento en base de datos.

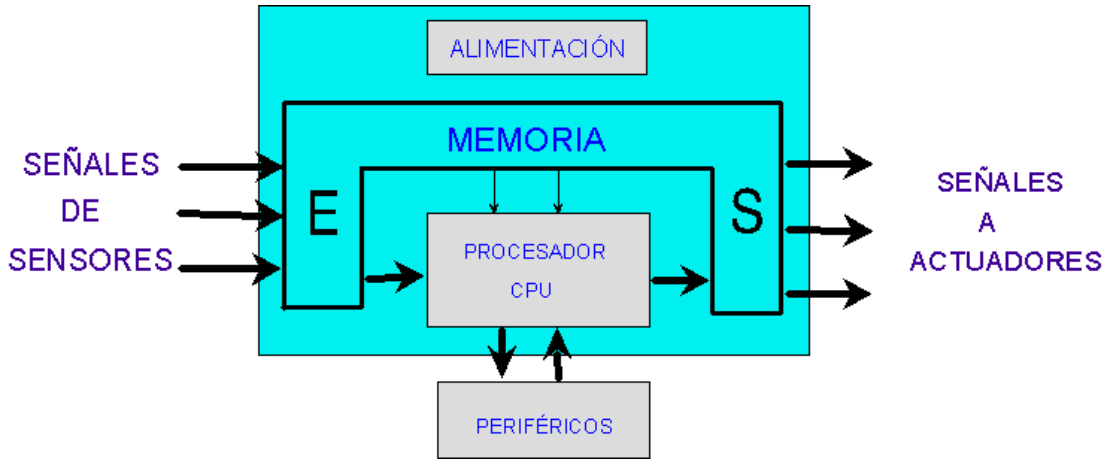
Para la supervisión de los distintos estados del proceso y proporcionar nuevas órdenes o parámetros, las comunicaciones entre PLC's, PC's, pantallas HMI (Interface Hombre Maquina), bases de datos y telefonía móvil, se realizan mediante redes de datos y buses de campo.

Otras ventajas de los PLC's son:

- Programar un PLC es más fácil que cablear un panel de control de relés

- El PLC Puede ser reprogramado
- El PLC Ocupan menos espacio
- Mayor fiabilidad y mantenimiento fácil
- PLC puede realizar una mayor variedad de funciones de control.

1.2.1 Estructura de un Autómata



▶ Unidad central de procesos

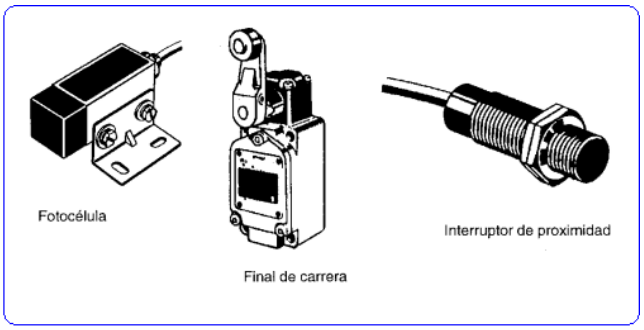
▶ Memoria de programación (RAM, EPROM, EEPROM)

▶ Sistema de control de E/S y perifericos

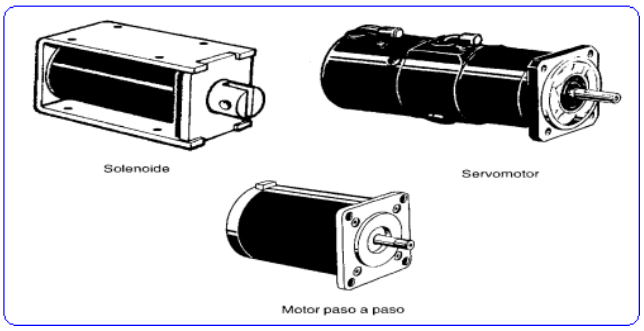
▶ Dispositivo de entradas / salidas.

1.2.2 Dispositivos de Entrada / Salida

- El PLC recibe señales de entrada tales como, Fotocélulas, Pulsadores, Teclados, etc.



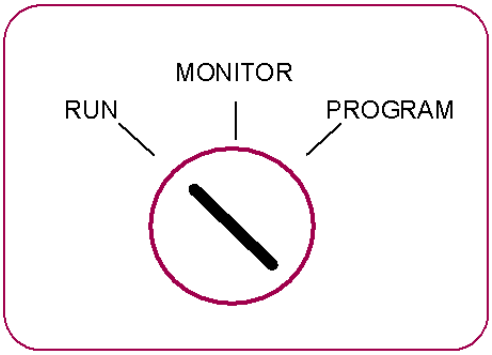
- El PLC activa mediante sus salidas, Válvulas, Solenoides, Contactores, Indicadores luminosos, etc.



1.2.3 Estados de Funcionamiento

- Program. El PLC está en reposo y puede recibir o enviar programa a un periférico (consola, PC, etc.)

- Monitor o run. El PLC ejecuta el programa que tiene en memoria, permitiendo en modo monitor el cambio de valores en los registros del mismo



1.2.4 Modo de funcionamiento

El PLC trabaja con programación interpretada y mediante programación cíclica.

- Ciclo SCAN

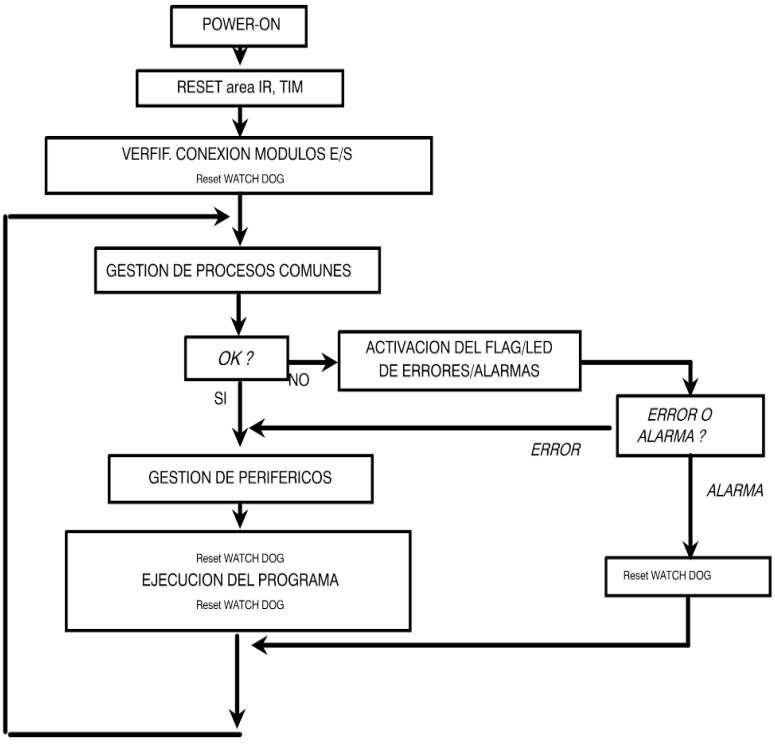
Se llama así al conjunto de tareas que el autómata lleva a cabo cuando está controlando un proceso.

- Tareas comunes: (Supervisión general)
- Aceptación de entradas y actuación sobre salidas
- Ejecución de las instrucciones
- Servicio a periféricos

- Tiempo de respuesta

Tiempo necesario para llevar a cabo las distintas operaciones de control. En particular, el tiempo de respuesta de un sistema (Activación de una señal de salida en relación a una entrada) viene determinado principalmente por:

- Tiempo de SCAN de la CPU
- Tiempo de ON/OFF de los módulos de E/S



1.2.5 Ciclo de Trabajo

-programación Watch Dog
-Verificar memoria de usuario
-Verificar BUS E/S

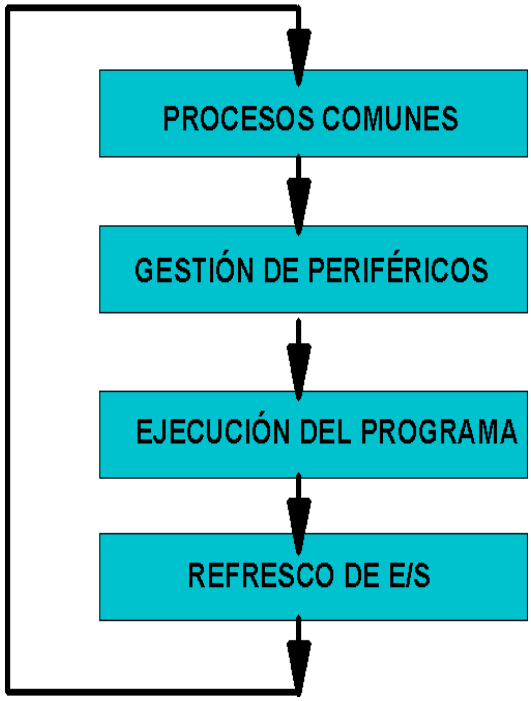
-Gestión de transmisión con:

Consola de programación Interface de comunicaciones

-Scan secuencial de las instrucciones del programa

-Lectura del estado de los módulos de E/S

- Transferencia de estados a las salidas



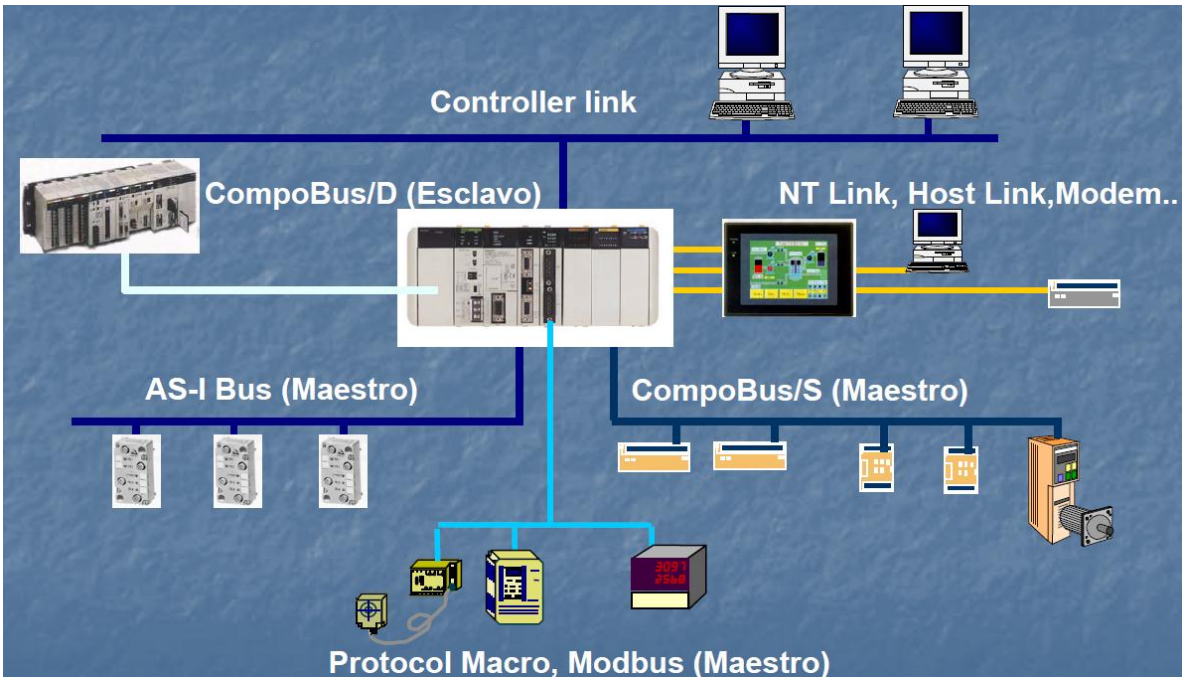
Calculo del ciclo SCAN

	Gestión de procesos comunes	T1 = 1.26 ms
	Gestión de periféricos	T2 = (T1+T3+T4)*0.05 ms Si T2<1ms, T2 = 1ms Si T2>1ms, T2 va redondeado por defecto al 0.5ms
	Ejecución de instrucciones	T3 = Suma de los tiempos de ejecución de las diversas instrucciones del programa
	Actualización de E/S	T4 = 0.29 + (0.07*N) ms Dónde: N = número de GATE ARRAY-1

Ciclo SCAN y tiempo de Respuesta µs

Instrucción o Proceso	CPM1A	SRM1	CPM2	CQM1H
Supervisión	0.6 ms	0.18 ms	0.3 ms	0.8 ms
Ejecución del Programa	1.43 ms	0.8 ms	0.6 ms	1.25 ms
Refresco de E/S	0.06 ms	0.02/0.05 ms	0.3 ms	0.04 ms
Servicio de Host Link	--	0	0.55 ms	0
Servicio de Periféricos	0.26 ms	0.7 ms	0.55 ms	0.34 ms
Servicio de Comboard	--	--	--	0.66 ms
Tiempo Total del ciclo de scan	2.35 ms	1.75 ms	1.75 ms	3.27 ms
Instrucciones básicas LD	1.72	0.97	0.64	0.375
MOV (21)	16.3	9.1	7.8	17.7
ADD (30)	29.5	15.9	14.7	37.5
Otras : PID	--	420.0	0.39 ms	1.59 ms

1.2.6 Comunicaciones



1.2.7 Familia de PLC,s OMRON

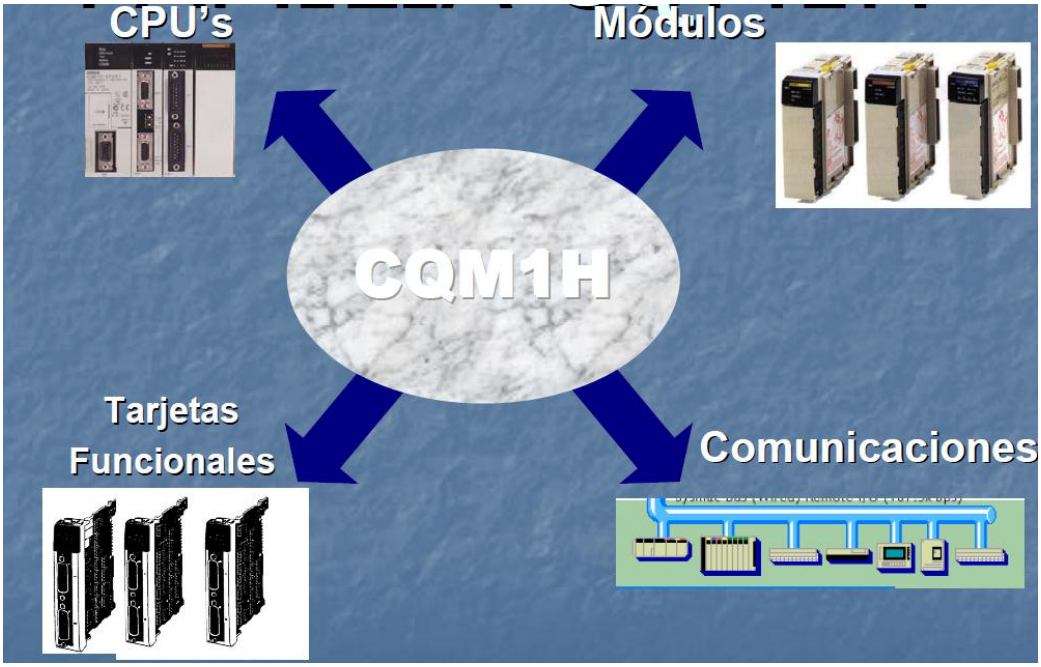
1.3 Definición de Emulador

Un emulador es un software que permite ejecutar programas en una plataforma (sea una arquitectura hardware o sistema operativo) diferente de aquella para la cual fueron escritos originalmente. A diferencia de un simulador, que solo trata de reproducir el comportamiento del programa, un emulador trata de modelar de forma precisa el dispositivo de manera que este funcione como si estuviese siendo usado en el dispositivo original. Cualquier ambiente funcional puede ser emulado dentro de otro. En la práctica, esto puede resultar realmente difícil, particularmente cuando el comportamiento exacto del sistema emulado no está documentado y debe ser deducido mediante ingeniería inversa. También puede ser que el emulador no actué tan rápidamente como el dispositivo original.

MEMORIA

2 AUTÓMATA FAMILIA CQM1/CPM1/SRM1 (OMRON)

Como ya se ha mencionado con anterioridad el proyecto se basa en el protocolo de comunicación Host link y tomando como referencia el autómata CQM1, por eso se va a hacer mayor hincapié en él y se especificara más en profundidad sobre aquel autómata, ya descatalogado y viejo pero que muchas empresas hoy en día lo utilizan debido a que es un autómata programable muy adaptable a cualquier tipo de maquina o aplicación media, flexibilidad, rapidez y sencillez con sus rangos principales, sobre todo se hablara sobre la configuración de comunicación de dicho protocolo Host link y de la memoria de dicho autómata para el correcto entendimiento del emulador de comunicaciones.



2.1 Memoria del PLC

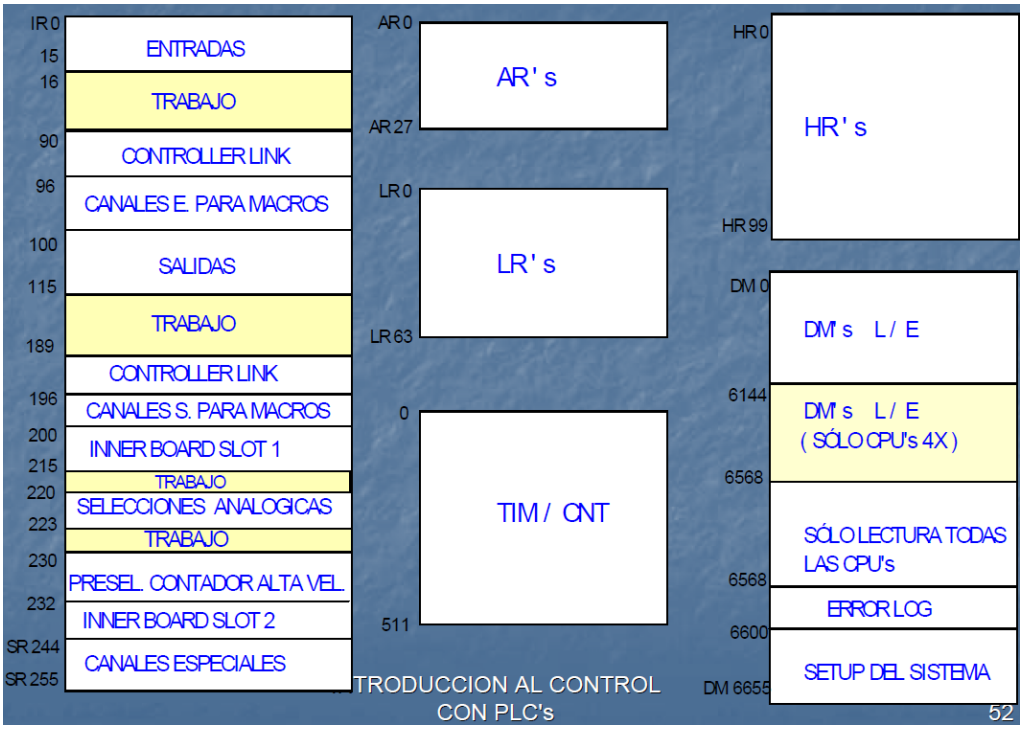
La memoria del plc se encuentra dividida en varias áreas, cada una de ellas con un contenido y características distintas:

- Área de programa:

En esta área es donde se encuentra almacenado el programa del PLC (que se puede programar en lenguaje Ladder o nemónico).

- Área de datos:

Esta área es usada para almacenar valores o para obtener información sobre el estado del PLC. Está dividida según funciones en IR, SR, AR, HR, LR, DM, TR, T/C.



Direccionamiento

Formato de las direcciones:

XXX YY

XXX: Numero del canal (Registro)

YY: Numero de bit (Relé), (entre 0 y 15)

2.1.1 Área de E/S y Área Interna (IR)

➢ Esta área de memoria comprende:

- Los canales asociados a los terminales externos (Entradas y salidas)

- Los relés internos (no correspondidos con el terminal externo), gestionados como relés de E/S.

➢ Accesibles como bits o canales

➢ Los relés E/S no usados pueden usarse como IR

➢ No retiene estado frente a falta de alimentación o cambio de modo de operación.

Área de datos		Canales	Función
Área de IR	Área entrada	IR 000 a IR 015	Unidades de entrada
	Área salida	IR 100 a IR 115	Unidades de salida
	Áreas de trabajo	IR 016 a IR 089	Área de trabajo sin función específica
		IR 116 a IR 189	
		IR 216 a IR 219	
		IR 224 a IR 229	
Áreas de estado de Controller Link		IR 090 a IR 095	Área de estado 1
		IR 190 a IR 195	Área de estado 2
Área de operando de MACRO	Área entrada	IR 096 a IR 099	Área de utilización por la instrucción MACRO
	Área salida	IR 196 a IR 199	
Área de hueco de tarjeta interna 1		IR 200 a IR 215	Bit asignados a hueco 1 de un CQM1H-CPU51/61
Área de selecciones analógicas		IR 220 a IR 223	Asignado a módulo CQM1H-AVB41
PV de contador alta velocidad 0		IR 230 a IR 231	Valor presente del contador de alta velocidad 0 (contador de la CPU)
Área de hueco de tarjeta interna 2		IR 232 a IR 243	Bit asignados a hueco 2

2.1.2 Área especial (SR)

➢ Son relés de señalización de funciones particulares como:

- Servicio (siempre ON, OFF)
- Diagnostico (Señalización o anomalías)
- Temporizadores (Relojes a varias frecuencias)
- Calculo (<,>,=)
- Comunicaciones

Descripción	CQM1H	
	Canal	Bit
Bit de reset del puerto de periféricos	SR 252	08
Bit de reset del puerto RS-232C		09
Indicador siempre en ON	SR 253	13
Indicador siempre en OFF		14
Indicador de primer ciclo		15
Reloj de 1 minuto	SR 254	00
Reloj de 0.02 segundos		01
Reloj de 0.1 segundos	SR 255	00
Reloj de 0.2 segundos		01
Reloj de 1 segundo		02
Indicador (ER) de error de ejecución de instrucción		03
Indicador de acarreo (CY)		04
Indicador de Mayor que (GR)		05
Indicador de Igual que (EQ)		06
Indicador de Menor que (LE)		07

2.1.3 Área auxiliar (AR)

➢ Contiene bits de control e información de recursos del PLC como: puerto RS232, puerto de periféricos, casetes de memoria,…

➢ Se divide en dos bloques:

- Señalización

1. Errores de configuración

2. Datos del sistema

- Memorización y gestión de datos

➢ Es un área de retención

Descripción	CQM1H	
	Canal	Bit
Código de error en Puerto RS-232C	AR 08	00 a 03
Indicador de error en Puerto RS-232C		04
Indicador de transmisión habilitada por Puerto RS-232C		05
Indicador de recepción completada por Puerto RS-232C		06
Indicador de overflow en recepción de Puerto RS-232C		07
Indicador de error en Puerto Periféricos		12
Contador de recepción del Puerto RS-232C	AR 09	
Contador de recepción del Puerto Periféricos	AR 10	
Segundos: 00 a 59 (BCD)	AR 18	00 a 07
Minutos: 00 a 59 (BCD)		08 a 15
Horas: 00 a 23 (BCD)	AR 19	00 a 07
Día del mes: 01 a 31 (BCD)		08 a 15
Mes: 01 a 12 (BCD)	AR 20	00 a 07
Año: 00 a 99 (BCD)		08 a 15
Día de la semana	AR 21	00 a 07
Contador de Power-Off	AR 23	
Tiempo de ciclo máximo	AR 26	
Tiempo de ciclo actual	AR 27	

2.1.4 Área de enlace (LR)

- Se utilizan para el intercambio de datos entre dos PLC's unidos en forma PC Link (1:1)
- Dedicados al intercambio de información entre PLC's
- Si no se utilizan como LR pueden usarse como IR

Todas estas áreas (IR, SR, AR, LR) tienen como características comunes:

- Accesibles en forma de BIT o canal.
- Los relés de E/S no utilizados como E/S físicas o desempeñando la función específica, pueden utilizarse como relés internos.
- No conservan su estado en caso de fallo de alimentación o cambio de modo de PLC (PROGRAM - RUN)

2.1.5 Área de retención (HR)

- Mantienen su estado ante fallo de alimentación o cambio de modo del PLC.
- Son gestionados igual que los IR, y direccionables como bit o como canal.

2.1.6 Memoria de datos (DM)

- Se trata de memoria de 16 bits (palabra)
- Utilizable para gestión de valores numéricos
- Mantiene su estado ante cambios en modos de trabajo o fallos de tensión
- Direccionable como canal
- Este área suele contener los parámetros de configuración del PLC (SETUP)

Área DM 0000 a DM 3071

A esta zona se accede solo en unidades de palabra o canal. Mantiene su estado ante cortes de alimentación o cambio de estado. Es un área de utilización libre para programación.

Área DM 6144 a DM 6568

Memoria de almacenamiento de parámetros de controller link, tablas de rutas, setup de tarjeta de comunicaciones,…

Área DM 6569 a DM6599

La CPU almacena de manera automática el código de error, asi como la hora y fecha en registros. Hasta un total de 10 errores (fatal y no-fatal)

Área DM 6600 a DM 6655

En esta área se registra la configuración del PC setup del CQM1, modo arranque, configuración de los puertos, selección de entradas de interrupción,… la única configuración que no se graba en esta zona es la referente a configuración de la unidad de comunicaciones serie

2.1.7 Temporizadores y contadores (TIM y CNT)

- Es el área de memoria que simula el funcionamiento de estos dispositivos.
- Son usados por el PLC para programar retardos y contajes
- Elementos característicos :

- **SV.** Valor de preselección
- **PV**. Valor actual
- **BIT.** Valor de estado

2.2 Interfaces hombre – maquina y sistemas de adquisicion de datos.

Son sistemas que permiten al operador de una planta interactuar con el proceso. Una interfaz hombre-máquina (HMI) es una abstracción de la forma en que el usuario interactúa con una máquina y puede ser desde una botonera en un cuadro eléctrico hasta una pantalla táctil comunicada con uno o varios autómatas programables a través de una red de comunicación industrial.

Un sistema de supervisión y adquisición de datos (SCADA) es una aplicación informática que se comunica con los elementos de campo de una planta para obtener y procesar los datos del proceso y mostrarlos al operador de una forma intuitiva y comprensible. Además también permite al operador operar sobre las máquinas por lo que podríamos decir que un HMI es un subconjunto de un los sistemas SCADA.

Un SCADA consta de las siguientes partes:

- Una base de datos que permite guardar los parámetros de configuración de las variables que tomaran los valores obtenidos del proceso y a las comunicaciones con los dispositivos e historiar dichas variables.

- Un software de gestión de alarmas que permita al operador localizar de forma rápida las averías que se produzcan en el proceso.

- Un software que implemente un driver de comunicaciones para enviar y recibir información hacia el proceso.

Con respecto a las comunicaciones, debido a que cada fabricante de dispositivos de automatización industrial emplea un protocolo diferente, para cada sistema SCADA existen multitud de drivers que permiten intercambiar información con autómatas y dispositivos de múltiples compañías.

2.2.1 Algunos ejemplos de interface hombre – máquina y SCADA

En la actualidad existen en el mercado incontables aplicaciones SCADA, tanto soluciones comerciales como de software libre, mientras que en el caso de os HMI al implicar un hardware más o menos complejo, el número de productos es menos y se restringe a unos pocos fabricantes de dispositivos de automatización industrial como Omron o Schneider Electric, Siemens, etc.

Ejemplo de HMI

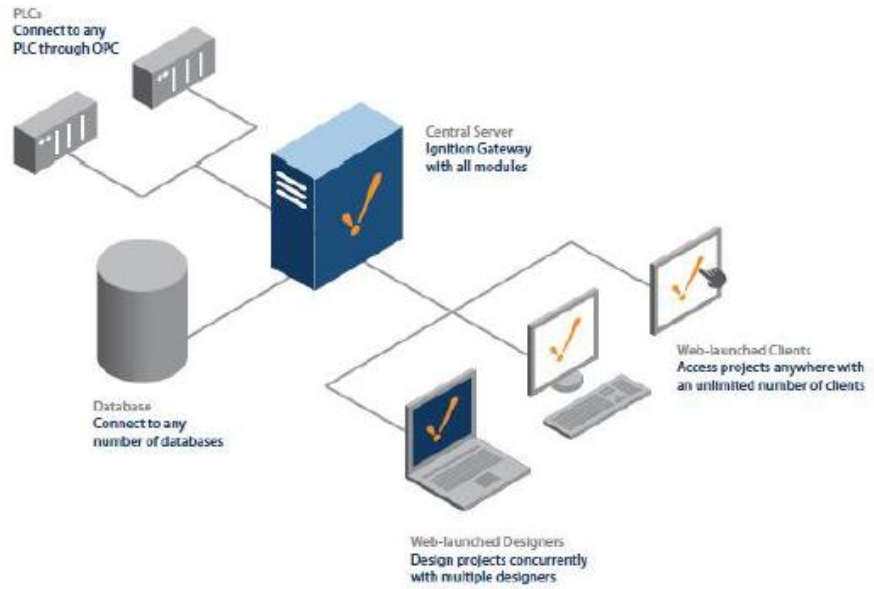
Hoy en día existen infinidad de HMIs en el mercado. A diferencia de los SCADA en los que mediante un driver es posible comunicarse con dispositivos de múltiples fabricantes, las interfaces hombre-máquina suelen estar preparadas para obtener datos de dispositivos de su mismo fabricante.

Lo dispositivos más utilizados en la actualidad son las pantallas táctiles. Estas pantallas están preparadas para ser montadas directamente en los cuadros eléctricos de control y se configuran y programan mediante un software que proporciona el fabricante. Aunque como se ha dicho en el anterior apartado, una HMI puede ser una botonera convencional de un cuadro eléctrico, la versatilidad de las pantallas táctiles programables hace que sean una opción más barata a largo plazo, puesto que si el sistema de automatización cambia estas pueden adaptarse de forma sencilla.

Ejemplo de SCADA

En los últimos años se ha producido un gran avance cualitativo en el mundo de los SCADA motivado por la aparición de algunas aplicaciones que han trasladado conceptos más propios de la programación de dispositivos móviles, aplicaciones web, o aplicaciones distribuidas al campo de la automatización industrial.

Un buen ejemplo es Ignition de la compañía norte americana Inductive Automation. Este software está programado en Java puro, por lo que puede utilizarse con cualquier sistema operativo que soporte este lenguaje de programación. Su arquitectura permite ejecutar la aplicación en un servidor y lanzarla vía web en múltiples clientes desde los cuales además puede editarse. Ignition cuenta con drivers de comunicaciones para prácticamente todos los protocolos de comunicación industrial.



Otras características muy interesantes de este software son: su sencilla conexión con bases de datos, la posibilidad de crear arquitecturas redundantes y la existencia de un SDK que permite añadirle funcionalidades programadas en Java por el usuario.

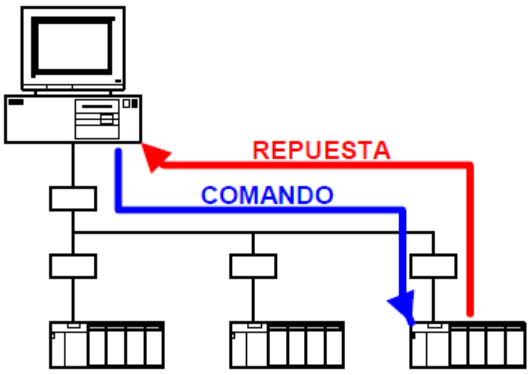
2.3 Comandos Host Link

Las comunicaciones Host link se ejecutan por medio de un intercambio de comandos y respuestas entre el ordenador y el PLC. Con el CQM1, se pueden utilizar dos métodos de comunicaciones. Uno es el método normal, en el que los comandos son generados por el ordenador y enviados al PLC. El otro método posibilita que el PLC genere comandos para el ordenador en este trabajo no compete hablar sobre estos.

Para que entre el ordenador y el PLC haya una comunicación efectiva se han de establecer unas normas de comunicación, un protocolo de comunicaciones, que para el caso particular de los PLC's de Omron se denomina protocolo Host Link. Las características más relevantes son:

- En una sola Transmisión el bloque de datos transferidos se denomina "Trama". Una trama está compuesta de un máximo de 131 caracteres de datos.
- El ordenador tiene prioridad de transmisión, y por lo tanto, la transferencia de datos entre el ordenador y el autómata comienza cuando el PC envía un comando a la CPU del PLC. El PLC envía luego automáticamente una respuesta
- La trama de datos enviada por el PLC al ordenador se denomina "trama de comando", mientras que la enviada por el PLC al PC se denomina "trama respuesta".
- Cada trama comienza con un numero de unidad y una cabecera y finaliza con un código de secuencia de chequeo de trama (FCS) y una terminación (* y CR). La terminación de la trama de comando permite señalizar al PLC que la trama enviada ha concluido y por tanto el PLC debe de verificar dicha trama y enviar respuesta.

A continuación se muestra la comunicación establecida mediante un PC y 3 PLC, para interactuar mediante el protocolo host link.



2.3.1 Formato de comando y respuesta

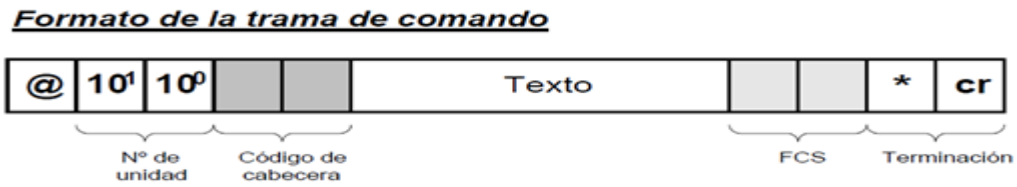
Esta sección explico los formatos para comandos y respuestas que se intercambian en comunicaciones host link.

Comandos del ordenador

Cuando se genera un comando desde el ordenador, los formatos del comando y de la respuesta son los siguientes.

Formato de comando

El formato comando del ordenador es el siguiente.



@ → Se debe colocar un símbolo "@" al principio.

Nº de unidad → Identifica el PLC que está comunicando con el ordenador. Especificar el número de nodo seleccionado en la configuración del PLC (DM 6648, DM 6653).

Código de cabecera → Selecciona el código de comando de 2 caracteres.

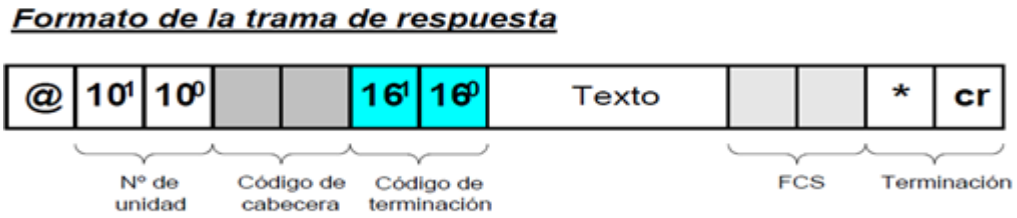
Texto → Selecciona los parámetros de comando.

FCS → Selecciona un código de 2 caracteres de secuencia de chequeo de trama.

Terminación → Seleccionar dos caracteres, "*" y retorno de carro (CHR$(13)) para indicar el fin del comando.

Formato de respuesta.

La respuesta del PLC se devuelve en el formato mostrado a continuación. Preparar un programa de tal forma que los datos de respuesta se puedan interpretar y procesar.



@, Nº de unidad, Código de cabecera → se devuelven los idénticos a los del comando.

Código de terminación → Devuelve el estado de finalización del comando(es decir, si se ha producido o no un error).

Texto → Se devuelve texto solo cuando hay tal tipo de datos para leer.

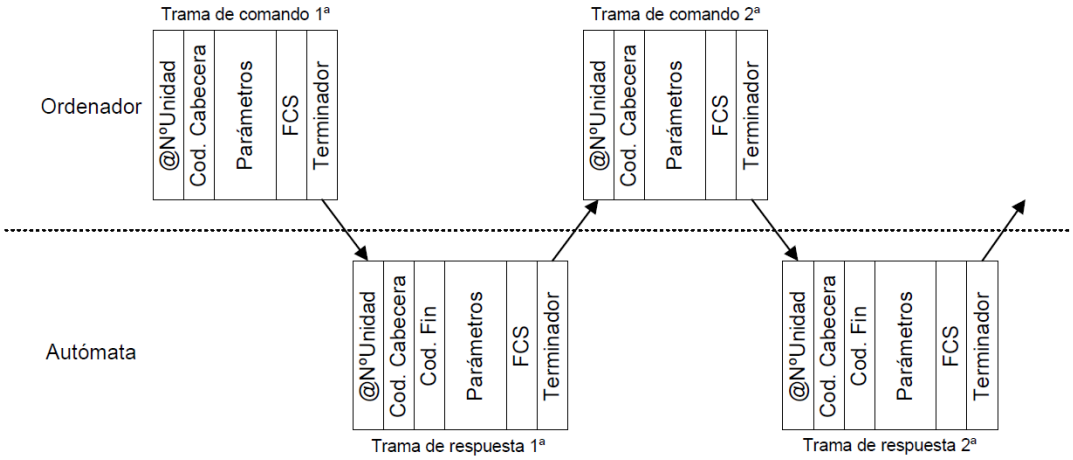
FCS → Selecciona un código de 2 caracteres de secuencia de chequeo de trama.

Terminación → Seleccionar dos caracteres, "*" y retorno de carro "CR" para indicar el fin del comando.

Transmisión y recepción de trama.

El orden o protocolo de intercambio de comandos y respuestas entre el ordenador y el PLC es el de la figura siguiente. La máxima cantidad de información que se puede transmitir en una sola trama es 131 caracteres contando la terminación. Por lo tanto un comando o respuesta

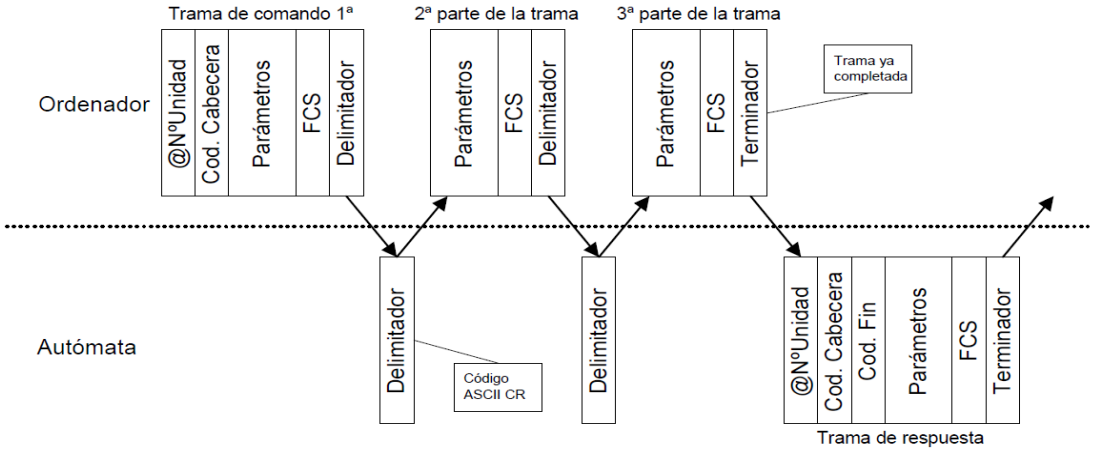
de 132 caracteres o más, se ha de dividir en más de una trama antes de la transmisión. Cuando se divide una transmisión, las tramas primera e intermedias se marcan con un delimitador “CR” en lugar de una terminación.



Esta será la forma de proceder más habitual, ya que salvo algún comando particular, el tamaño de la trama de comando suele ser inferior a los 131 caracteres. En la mayoría de los casos aunque el comando exceda de este número de bytes se puede buscar la forma de enviar varios comandos consecutivos que sean equivalentes.

Dividir comandos (del ordenador al PLC)

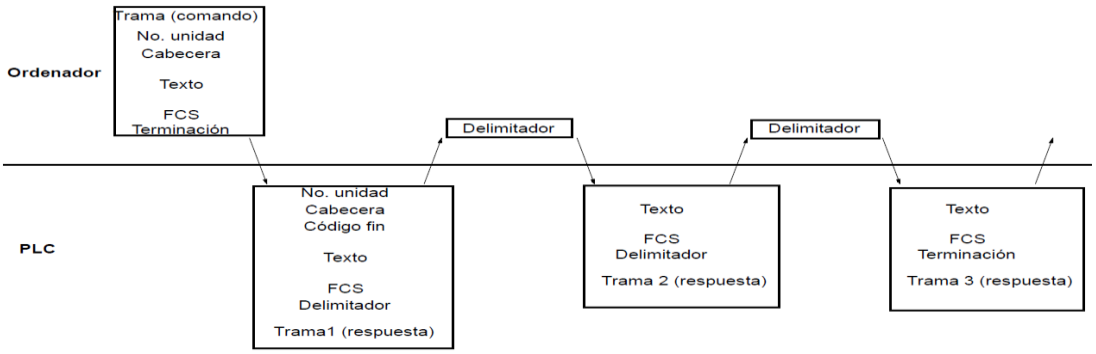
Cuando no sea posible enviar solo los 131 caracteres y haya que enviar más, estas tramas se llaman trasmisiones largas y se deberá actuar de la como indica la figura siguiente.



Según se transmite cada trama, el nodo que las recibe espera a que se transmita el delimitador. Después de transmitir el delimitador, se enviara la siguiente trama. Este procedimiento se repite hasta que se haya transmitido el comando o respuesta completa.

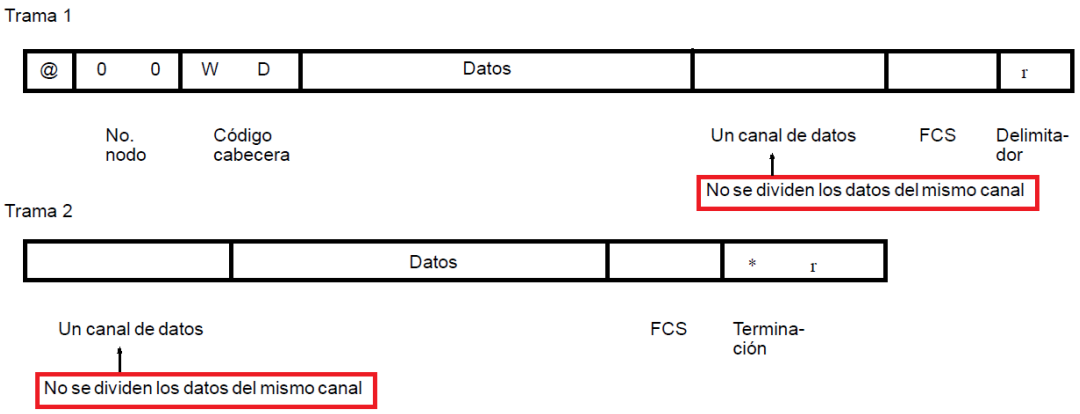
Dividir respuesta (del PLC al ordenador)

Según recibe el ordenador cada trama, se transmite un delimitador al PLC. Después de haber transmitido el delimitador, el PLC transmitirá la siguiente trama. Esto se repetirá hasta que se haya transmitido la respuesta completamente, como se puede ver en la siguiente figura.



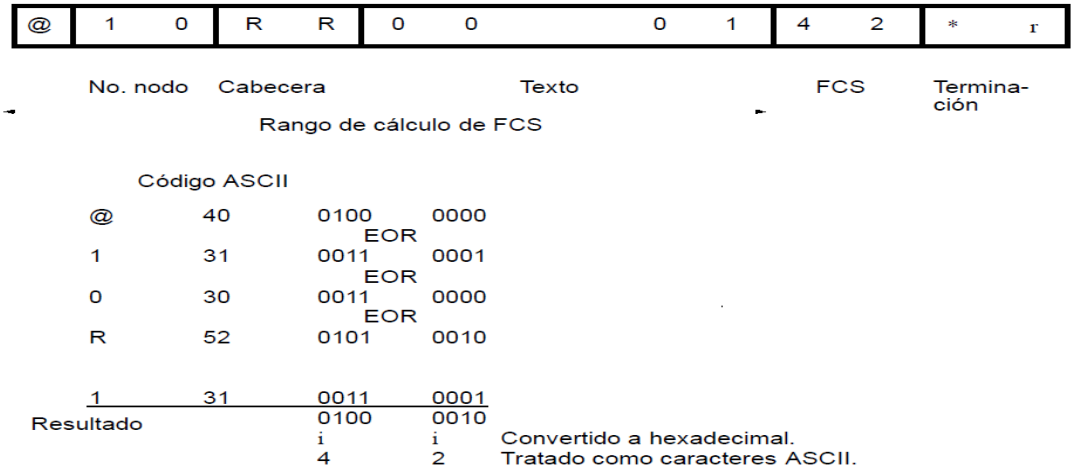
Precauciones para las transmisiones largar

Cuando se dividen comandos tales como WR, WL, WC o WD que ejecutan operaciones de escritura, prestar atención para no dividir en tramas separadas un dato que se debe escribir en un solo canal. Como se muestra en la siguiente figura, verificar que se divide las tramas de tal forma que coincidan con las divisiones entre canales.



FCS (Secuencia de control de trama)

Cuando se transmite una trama, se coloca un FCS justo antes del delimitador o Terminación para chequear si se ha generado algún error de datos. El FCS es un dato de 8 bits convertido a dos caracteres ASCII. El dato de 8 bits es el resultado de una OR exclusiva de los datos, des del principio dela trama hasta el final del texto dela trama (es decir, inmediatamente antes del FCS).Calculando el FCS cada vez que se recibe una trama con el FCS incluido en ella, posibilita detectar errores de datos generados en la trama. Como se puede ver en la siguiente figura.



2.3.2 Repertorio de comandos Host Link

Aquí explicare todos los comandos que se pueden generar desde el ordenador al PLC.

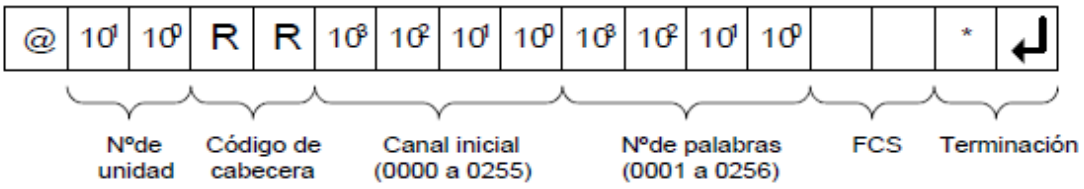
La siguiente tabla ilustra los comandos que se utilizan para las comunicaciones host link con los PLC de OMRON CQM1/CPM1/CPM1A/SRM1.

Código de cabecera	Modo del PLC			Nombre
	RUN	MON	PRG	
RR	Válido	Válido	Válido	LECTURA DE AREA IR/SR
RL	Válido	Válido	Válido	LECTURA DE AREA LR
RH	Válido	Válido	Válido	LECTURA DE AREA HR
RC	Válido	Válido	Válido	LECTURA DE PV
RG	Válido	Válido	Válido	LECTURA DE ESTADO DE TC
RD	Válido	Válido	Válido	LECTURA DE AREA DE DM
RJ	Válido	Válido	Válido	LECTURA DE AREA DE AR
WR	No Válido	Válido	Válido	ESCRITURA DE AREA IR/SR
WL	No Válido	Válido	Válido	ESCRITURA DE AREA LR
WH	No Válido	Válido	Válido	ESCRITURA DE AREA HR
WC	No Válido	Válido	Válido	ESCRITURA DE PV
WG	No Válido	Válido	Válido	ESCRITURA DE ESTADO DE TC
WD	No Válido	Válido	Válido	ESCRITURA DE AREA DM
WJ	No Válido	Válido	Válido	ESCRITURA DE AREA AR
R#	Válido	Válido	Válido	LECTURA SV 1
R$	Válido	Válido	Válido	LECTURA SV 2
R%	Válido	Válido	Válido	LECTURA SV 3 (Sólo PLCs CQM1)
W#	No Válido	Válido	Válido	CAMBIAR SV 1
W$	No Válido	Válido	Válido	CAMBIAR SV 2
W%	No Válido	Válido	Válido	CAMBIAR SV 3 (Sólo PLCs CQM1)
MS	Válido	Válido	Válido	LECTURA DE ESTADO
SC	Válido	Válido	Válido	ESCRITURA DE ESTADO
MF	Válido	Válido	Válido	LECTURA DE ERROR
KS	No Válido	Válido	Válido	FORZADO A ON
KR	No Válido	Válido	Válido	FORZADO A OFF
FK	No Válido	Válido	Válido	FORZADOS MULTIPLES A ON/OFF
KC	No Válido	Válido	Válido	CANCELACION DE FORZADOS
MM	Válido	Válido	Válido	LECTURA DE MODELO DE PLC
TS	Válido	Válido	Válido	PRUEBA DE COMUNICACIONES
RP	Válido	Válido	Válido	LECTURA DE PROGRAMA
WP	No Válido	No Válido	Válido	ESCRITURA DE PROGRAMA
QQ	Válido	Válido	Válido	COMANDO COMPUESTO
XZ	Válido	Válido	Válido	ABORTAR (sólo comando)
**	Válido	Válido	Válido	INICIALIZAR (sólo comando)
IC	---	---	---	Comando indefinido (sólo respuesta)

Lectura de área IR/SR -- RR

Lee los contenidos del número especificado de canales en Nº de palabras de IR y SR, comenzando por el canal especificado en canal inicila.

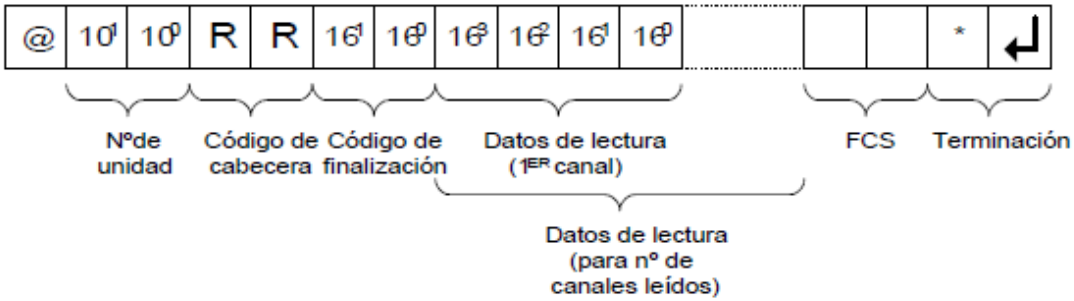
Formato de comando



Canal inicial: 0000 a 0255 en PLCs CQM1, 0000 a 0019 y 0200 a 0255 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras: 0001 a 0256 en PLCs CQM1, 0001 a 0076 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



No se pueden especificar los canales 0020 a 0199 en los PLCs CPM1/CPM1A/SRM1. Si se intenta leer cualquiera de estos canales, se devolverá una respuesta de 0000.

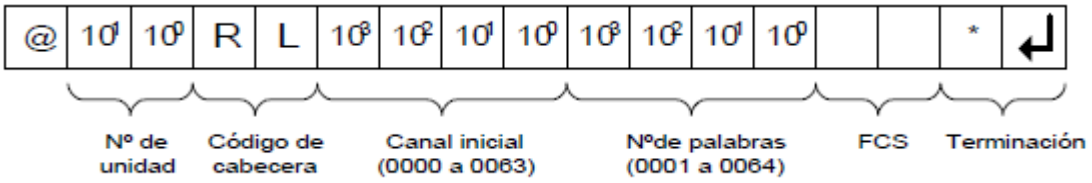
Cuando se lean más de 30 canales de datos, se dividirá la respuesta.

Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Lectura de área LR -- RL

Lee los contenidos del número especificado de canales en Nº de palabras de LR, comenzando por el canal especificado en canal inicial.

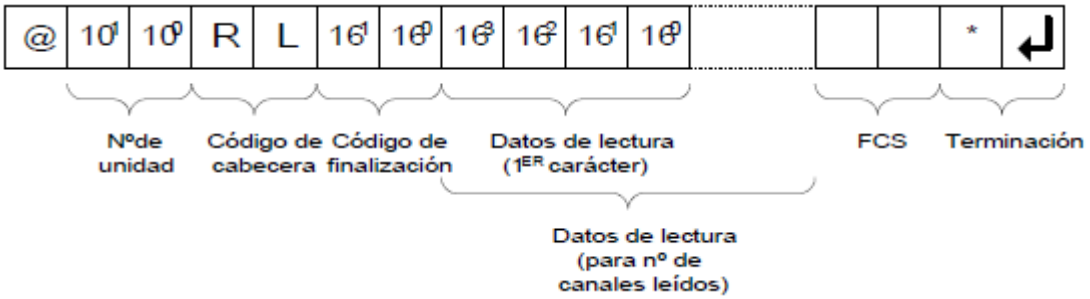
Formato de comando



Canal inicial: 0000 a 0063 en PLCs CQM1 y 0000 a 0015 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras: 0001 a 0064 en PLCs CQM1, 0001 a 0016 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



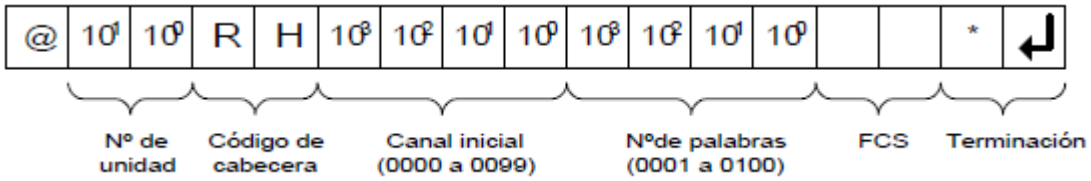
Cuando se lean más de 30 canales de datos, se dividirá la respuesta.

Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Lectura de área HR -- RH

Lee los contenidos del número especificado de canales en Nº de palabras de HR, comenzando por el canal especificado en canal inicial.

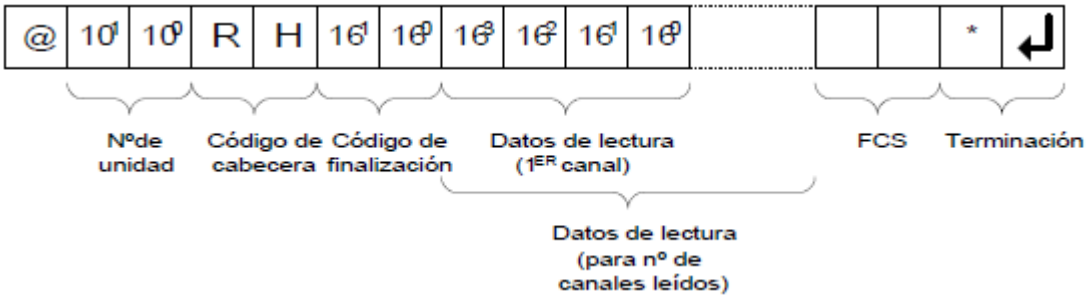
Formato de comando



Canal inicial: 0000 a 0099 en PLCs CQM1, 0000 a 0019 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras: 0001 a 0100 en PLCs CQM1, 0001 a 0020 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



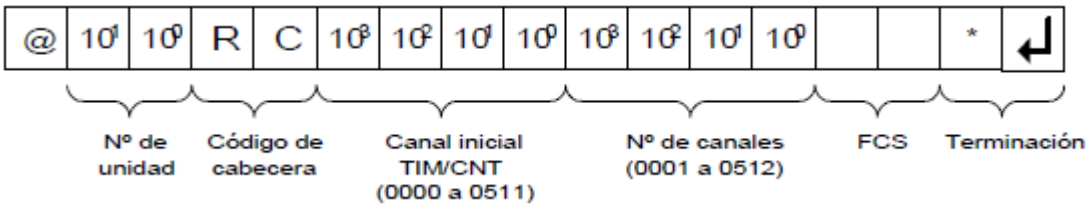
Cuando se lean más de 30 canales de datos, se dividirá la respuesta.

Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Lectura de área PV -- RC

Lee los contenidos del número especificado de canales en Nº de palabras de PV(valor presente) de temporizador/contador, comenzando por el temporizador/contador especificado en canal inicial.

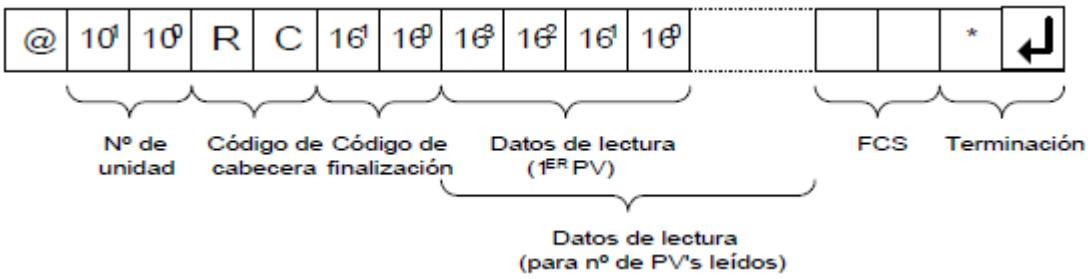
Formato de comando



Canal inicial: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras (Nº T/Cs): 0001 a 0512 en PLCs CQM1, 0001 a 0128 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



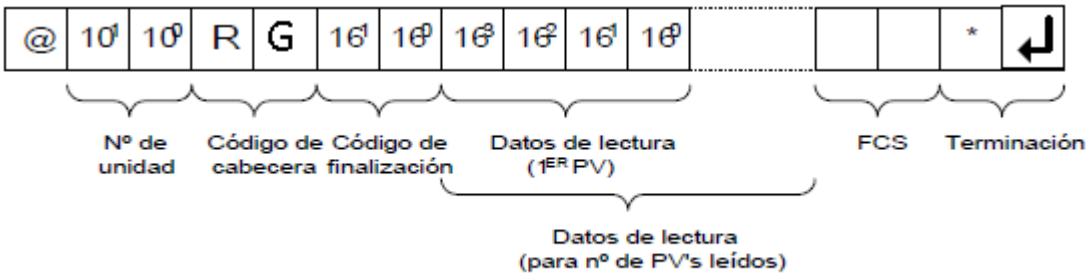
Cuando se lean más de 30 canales de datos, se dividirá la respuesta.

Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Lectura de Estado de TC -- RG

Lee el estado de los indicadores de finalización del número de temporizadores/contadores especificados, comenzando por el temporizador/contador especificado.

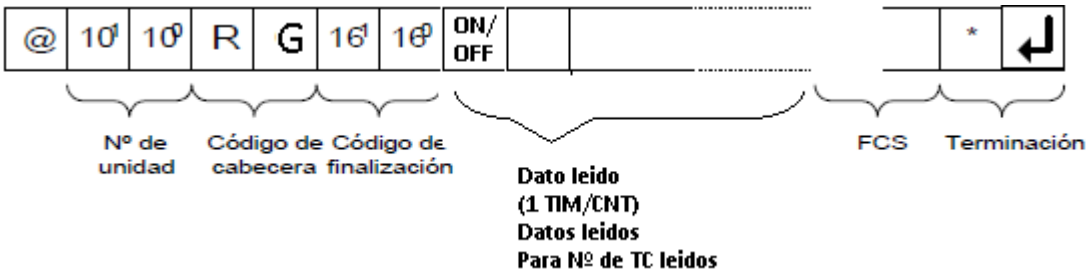
Formato de comando



Canal inicial: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras (Nº T/Cs): 0001 a 0512 en PLCs CQM1, 0001 a 0128 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



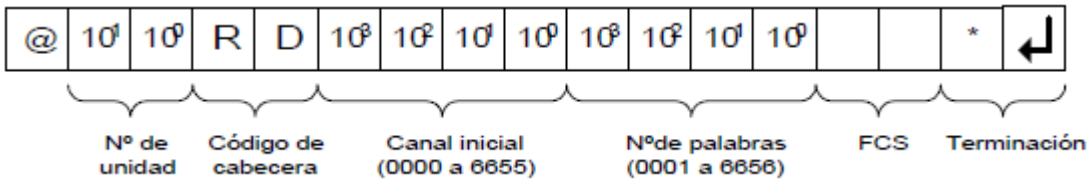
Cuando se lean más de 119 TIM/CNT, se dividirá la respuesta.

Datos leídos (Respuesta)→ Se devuelve como respuesta el estado del número de indicadores de finalización especificados por el comando. “1” significa que el indicador está en ON. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Lectura de área DM -- RD

Lee los contenidos del número especificado de canales en Nº de palabras de DM, comenzando por el canal especificado en canal inicial.

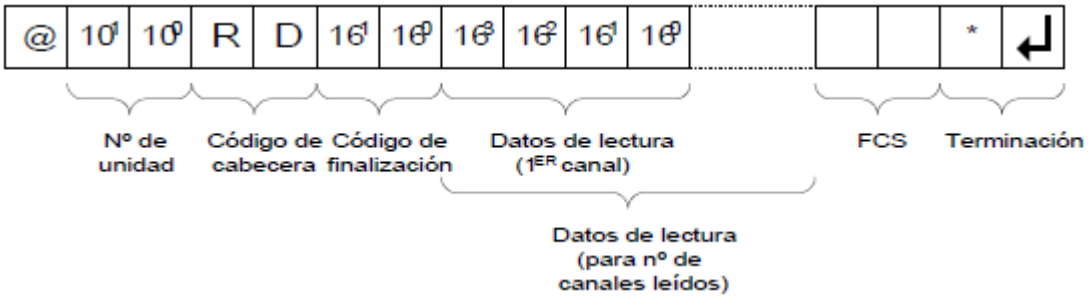
Formato de comando



Canal inicial: 0000 a 6655 en PLCs CQM1, 0000 a 1023 y 6144 a 6655 en PLCs CPM1/CPM1A, 0000 a 2047 y 6144 a 6655 en PLCs SRM1.
Nº de palabras: 0001 a 6656 en PLCs CQM1, 0001 a 1536 en PLCs CPM1/CPM1A, 0001 a 2560 en PLCs SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.



Cuando se lean más de 30 canales de datos, se dividirá la respuesta.

No se pueden especificar los canales 1024 a 6143 en PLCs CPM1/CPM1A y 2048 a 6143 en PLCs SRM1. Si se intenta leer cualquiera de estos canales, se devolverá una respuesta de 0000.

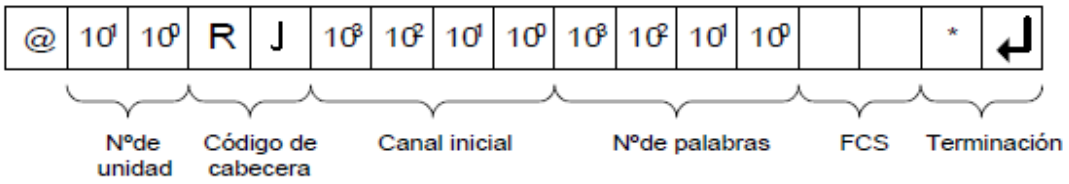
Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Tener cuidado a la configuración del área DM, debido a que difiere según los modelos de PLC.

Lectura de área AR -- RJ

Lee los contenidos del número especificado de canales en Nº de palabras de AR, comenzando por el canal especificado en canal inicial.

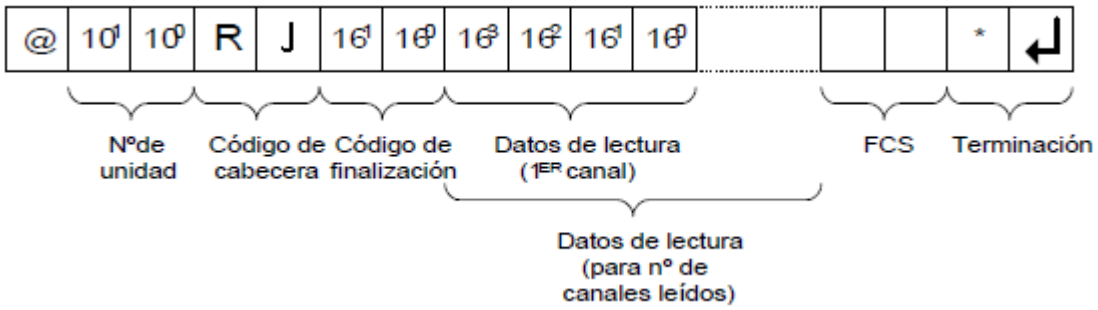
Formato de comando



Canal inicial: 0000 a 0027 en PLCs CQM1 y 0000 a 0015 en PLCs CPM1/CPM1A/SRM1.
Nº de palabras: 0001 a 0028 en PLCs CQM1, 0001 a 0016 en PLCs CPM1/CPM1A/SRM1.

Formato respuesta

Un codigo de finalizacion “00” finalizacion sin errores.

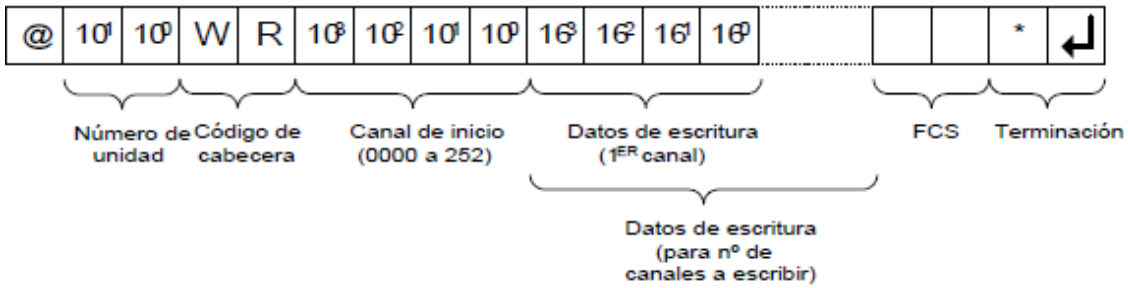


Datos leídos (Respuesta)→ Los contenidos del número decanales especificados por el comando se devuelven en hexadecimal como respuesta. Los canales se devuelven en orden, comenzando por el canal inicial especificado.

Escribir área IR/SR-- WR

Escribe datos en las áreas de IR y SR, comenzando por el canal especificado.
La escritura se realiza canal por canal.

Formato de comando

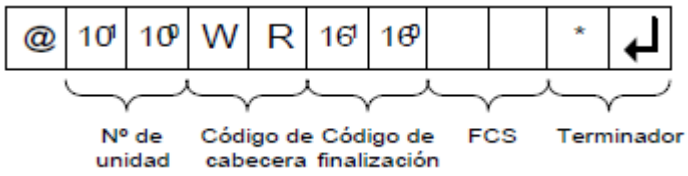


Canal inicial : 0000 a 0252 en PLCs CQM1, 0000 a 0019 y 0200 a 0252 en PLCs CPM1/CPM1A/SRM1.

Hay que dividir el comando cuando se escriban más de 29 canales de datos.

Formato respuesta

Un codigo “00” indica finalizacion normal.



No se pueden especificar los canales 0020 a 0199 en PLCs CPM1/CPM1A/SRM1.Si se intenta escribir en uno de estos canales, no se ejecutará la operación de escritura y dará finalización normal.

Escribir datos (Comando)

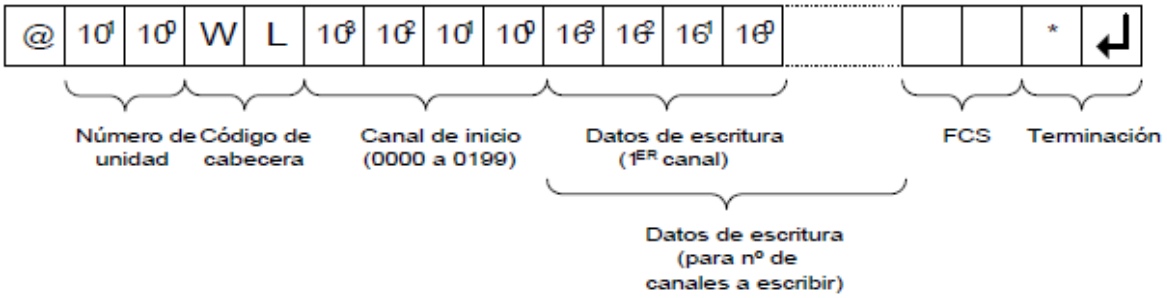
Especificar en orden los contenidos del número de canales a escribir en el área IR o SR en hexadecimal, comenzando con el canal inicial especificado.

Si se especifican datos de escritura que exceden el rango permisible, se generará un error y no se ejecutará la operación de escritura. Si, por ejemplo, se especifican dos canales de escritura designando como canal inicial el 252, se debería escribir en los canales 252 y 253, pero dado que este último está fuera del rango, se generará un error y no se ejecutará el comando.

Escribir área LR—WL

Escribe datos en las áreas de LR, comenzando por el canal especificado. La escritura se realiza canal por canal.

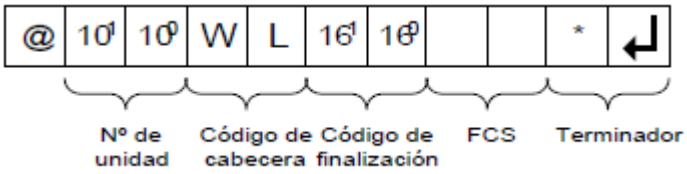
Formato de comando



Canal inicial: 0000 a 0063 en PLCs CQM1, 0000 a 0015 en PLCs PM1/CPM1A/SRM1

Formato respuesta

Un código de fin 00 indica finalización normal.



Escribir datos (Comando)

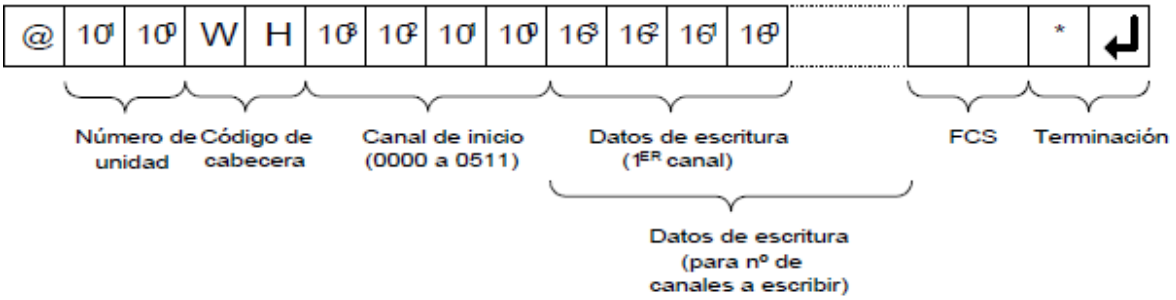
Especificar en orden los contenidos del número de canales a escribir en el área LR en hexadecimal, comenzando con el canal inicial especificado.

Si se especifican datos de escritura que exceden el rango permisible, se generará un error y no se ejecutará la operación de escritura. Si, por ejemplo, se especifican cinco canales de escritura designando como canal inicial el 60, se debería escribir en los canales 60 a 64, pero dado que este último está fuera del rango, se generará un error y no se ejecutará el comando.

Escribir área HR—WH

Escribe datos en las áreas de HR, comenzando por el canal especificado. La escritura se realiza canal por canal.

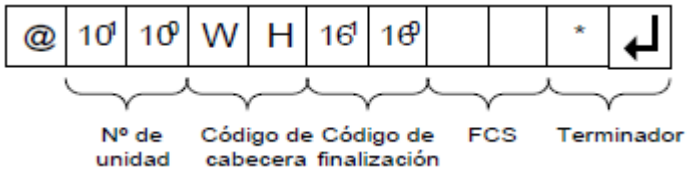
Formato de comando



Canal inicial: 0000 a 0099 en PLCs CQM1, 0000 a 0019 en PLCs CPM1/CPM1A/SRM1

Formato respuesta

Un código de fin 00 indica finalización normal.



Escribir datos (Comando)

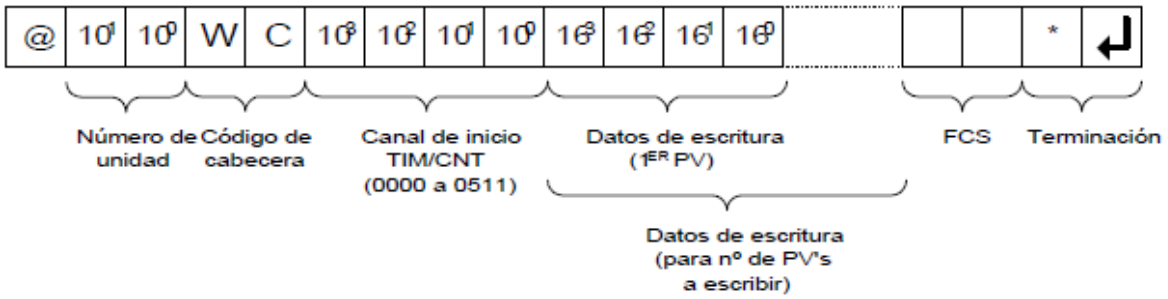
Especificar en orden los contenidos del número de canales a escribir en el área HR en hexadecimal, comenzando con el canal inicial especificado.

Si se especifican datos de escritura que exceden el rango permisible, se generará un error y no se ejecutará la operación de escritura. Si, por ejemplo, se especifican tres canales de escritura designando como canal inicial el 98, se debería escribir en los canales 98 a 100, pero dado que este último está fuera del rango, se generará un error y no se ejecutará el comando.

Escribir área PV—WC

Escribe los PVs (valor presente) de temporizadores/contadores comenzando por el temporizador/contador especificado.

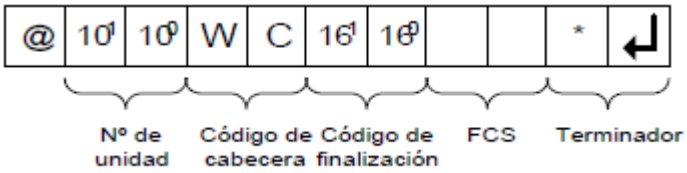
Formato de comando



T/C inicial: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1
Dividir el comando cuando se escriban más de 29 canales de datos.

Formato respuesta

Un código de fin 00 indica finalización normal.



Escribir datos (Comando)

Especificar los números BCD para valores presentes de temporizadores/contadores que se han de escribir, comenzando por el temporizador/contador inicial.

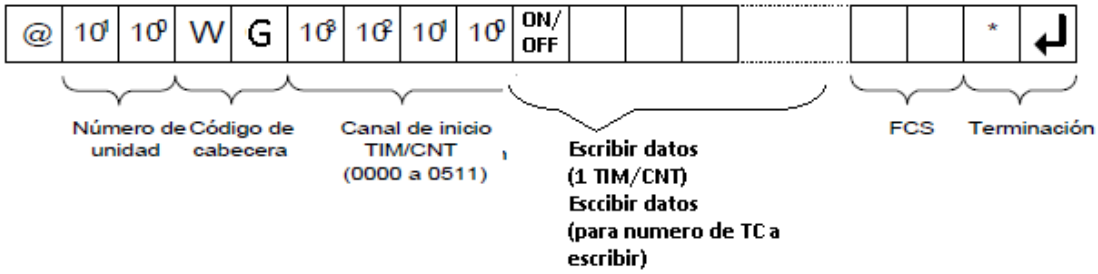
Cuando se utiliza este comando para escribir datos en el área de PV, el indicador de finalización para los temporizadores/contadores que se han escrito se pondrán en OFF.

Si los datos especificados para escritura exceden el rango permisible, se generará un error y no se ejecutará el comando. Si, por ejemplo, se especifica 510 como canal inicial y se han de escribir tres datos, el 512 será el último canal para escribir el dato y el comando no se ejecutará dado que TC512 está fuera del rango.

Escribir estado de TC -- WG

Escribe el estado de los indicadores de finalización para temporizadores y contadores en el área de TC, comenzando por el número de temporizador/contador especificado. La escritura se realiza canal por canal.

Formato de comando.

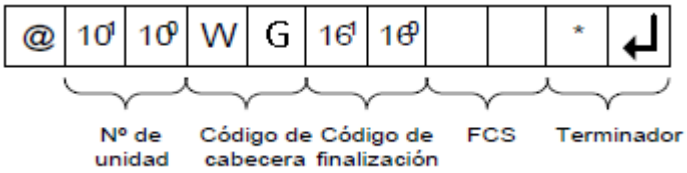


T/C inicial: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1

Dividir el comando cuando se escriba el estado de más de 118 temporizadores/contadores.

Formato respuesta.

Un código de fin 00 indica finalización normal.



Escribir datos (Comando)

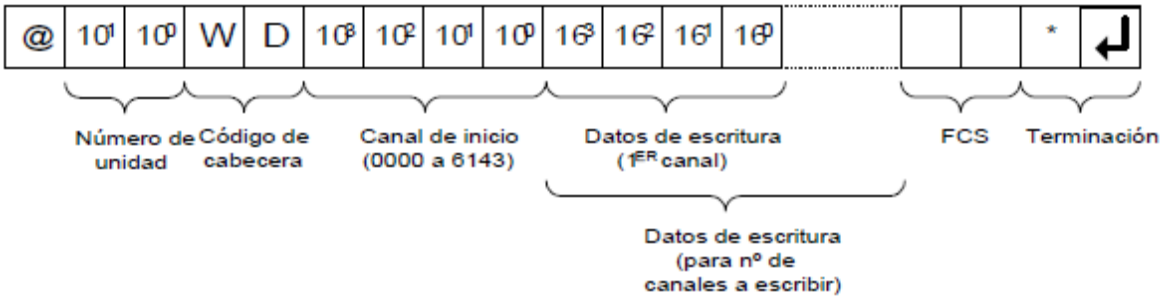
Especifica el estado ON u OFF de los indicadores de finalización, para el número de temporizadores/contadores a escribir, desde el canal inicial. Cuando el indicador de finalización está en ON, indica que el tiempo o contaje se ha alcanzado.

Si los datos especificados para escritura exceden el rango permisible, se generará un error y la operación no se ejecutará. Si, por ejemplo, se especifica 510 como canal inicial y se han de escribir tres datos, el 512 será el último canal para escribir el dato y el comando no se ejecutará dado que TC512 está fuera del rango.

Escribir área DM – WD

Escribe datos en el área de DM comenzando por el canal especificado. La escritura se realiza canal por canal.

Formato de comando

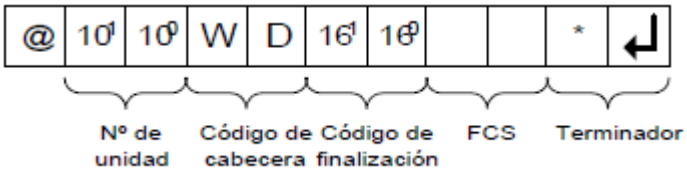


Canal inicial: 0000 a 6143 en PLCs CQM1, 0000 a 1023 y 6144 a 6655 en PLCs CPM1/CPM1A y 0000 a 2047 y 6144 a 6655 en PLCs SRM1.

Dividir el comando cuando se escriban de 29 canales de datos.

Formato respuesta

Un código de fin 00 indica finalización normal.



No se pueden especificar los canales 1024a 6143en los PLCs CPM1/CPM1A y 2048 a 6143 en PLCs SRM1. Si se intenta escribir en cualquiera de estos canales, no se ejecutará la operación de escritura y finalizará normal.

Escribir datos (Comando)

Especificar en orden los contenidos del número de canales a escribir en el área de DM en hexadecimal, comenzando con el canal inicial especificado.

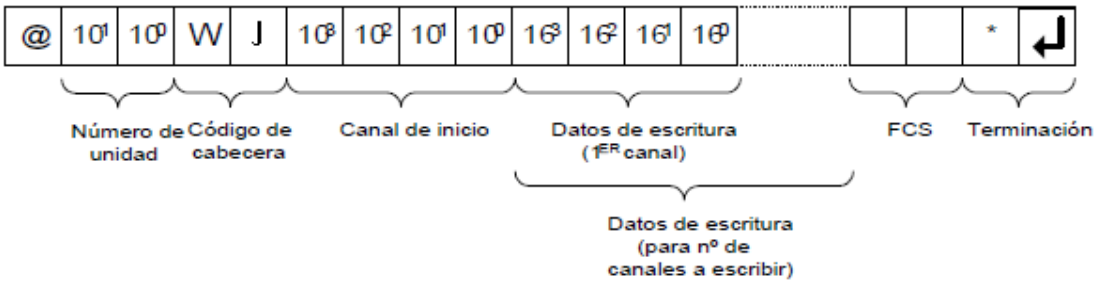
Si los datos especificados para escritura exceden el rango permisible, se generará un error y la operación no se ejecutará. Si, por ejemplo, se especifica 6142 como canal inicial y se han de escribir tres canales, el 6144 será el último canal para escribir el dato y el comando no se ejecutará dado que DM6144 está fuera del rango.

Prestar atención a la configuración del área de DM, dado que difiere según los modelos.

Escribir área AR – WJ

Escribe datos en el área de AR comenzando por el canal especificado. La escritura se realiza canal por canal.

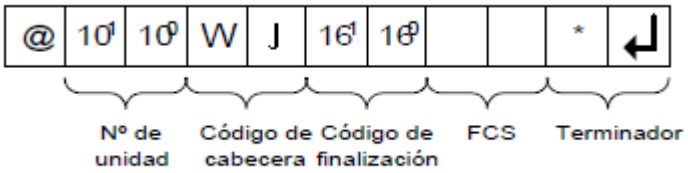
Formato de comando.



Canal inicial: 0000 a 0027 en PLCs CQM1, 0000 a 0015 en PLCs CPM1/CPM1A/SRM1

Formato respuesta.

Un código de fin 00 indica finalización normal.



Escribir datos (Comando)

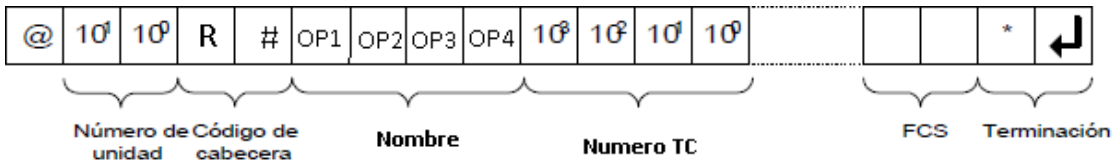
Especificar en orden los contenidos del número de canales a escribir en el área de DM en hexadecimal, comenzando con el canal inicial especificado.

Si los datos especificados para escritura exceden el rango permisible, se generará un error y la operación no se ejecutará. Si, por ejemplo, se especifica 26 como canal inicial y se han de escribir tres canales, el 28 será el último canal para escribir el dato y el comando no se ejecutará dado que AR28 está fuera del rango.

Lectura de SV--R#

Busca la primera ocurrencia de una instrucción TIM, TIMH(15), CNT y CNTR(12) con el número de TC especificado en el programa de usuario y lee el SV, asumiendo seleccionado como una constante. El SV leído es un número decimal de 4 dígitos (BCD). Se busca desde el principio del programa y tardará aproximadamente 10 segundos en producir una respuesta.

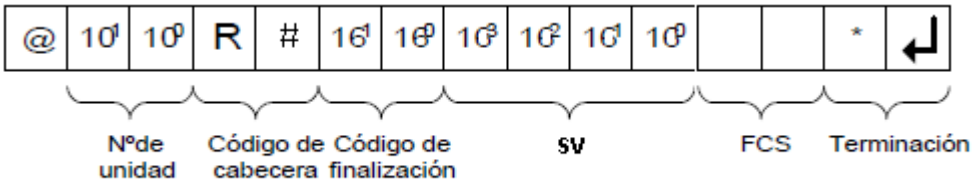
Formato de comando.



Numero de TC: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1

Formato respuesta.

Un código de fin 00 indica finalización normal.



Nombre, Número de TC (Comando)

Especificar la instrucción para leer su SV en ”Nombre” en cuatro caracteres. En”Número de TC” especificar el número de temporizador/contador utilizado para la instrucción.

Nombre de instrucción				Clasificación
OP1	OP2	OP3	OP4	
T	I	M	(Espacio)	TEMPORIZADOR
T	I	M	H	TEMPORIZADOR DE ALTA VELOCIDAD
C	N	T	(Espacio)	CONTADOR
C	N	T	R	CONTADOR REVERSIBLE

SV (Respuesta) Devuelve la constante SV.

La instrucción especificada en ”Nombre” debe darse en cuatro caracteres. Si quedan huecos vacíos, rellenar con espacios hasta completar 4 caracteres. Si la instrucción aparece más de una vez en el programa, sólo se leerá la primera.

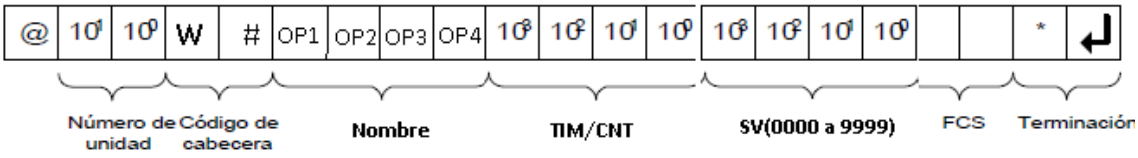
Utilizar este comando sólo cuando sea definitivo que se ha seleccionado una constante para SV.

El código de respuesta de fin indicará un error (16) si el SV no se introdujo como una constante.

Cambiar SV--W#

Busca la instrucción TIM, TIMH(15), CNT o CNTR(12) especificada en el programa de usuario y cambia el SV por la nueva constante de SV especificada en el segundo dato de la instrucción. La búsqueda empieza desde el principio del programa y tardará aproximadamente 10 segundos para producir una respuesta.

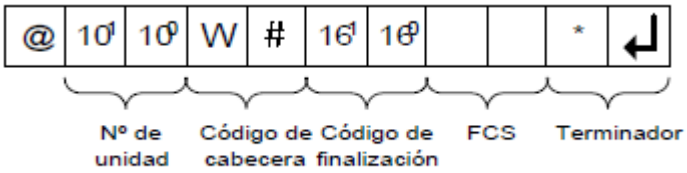
Formato de comando.



Número de TC: 0000 a 0511 en PLCs CQM1, 0000 a 0127 en PLCs CPM1/CPM1A/SRM1

Formato respuesta

Un código de fin 00 indica finalización normal.



Nombre, Número de TC (Comando)

Especificar la instrucción para leer su SV en ”Nombre” en cuatro caracteres. En ”Número de TC” especificar el número de temporizador/contador utilizado para la instrucción.

Nombre de instrucción				Clasificación
OP1	OP2	OP3	OP4	
T	I	M	(Espacio)	TEMPORIZADOR
T	I	M	H	TEMPORIZADOR DE ALTA VELOCIDAD
C	N	T	(Espacio)	CONTADOR
C	N	T	R	CONTADOR REVERSIBLE

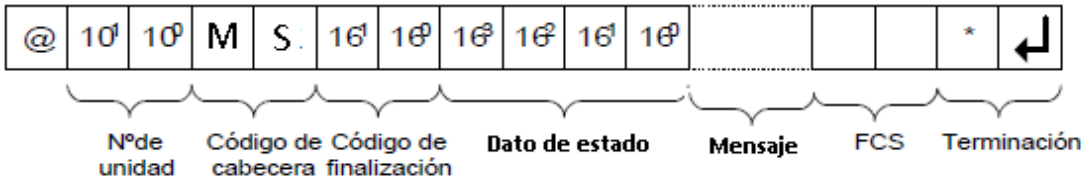
Leer estado del PLC--MS#

Lee las condiciones de funcionamiento del PLC.

Formato de comando.

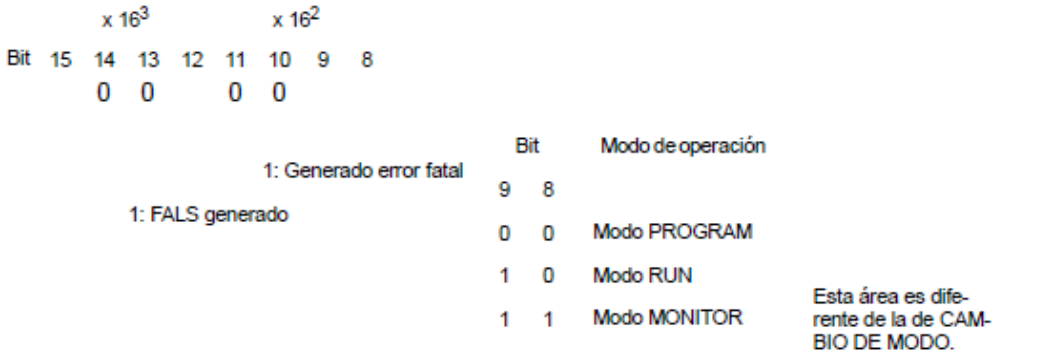
Formato respuesta.

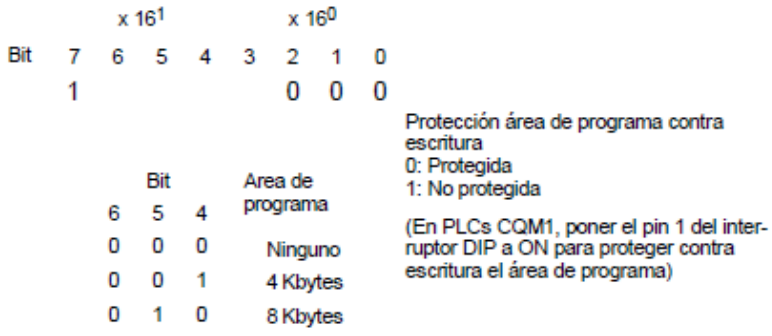
Un código de fin 00 indica finalización normal.



Dato de estado, Mensaje (Respuesta)

"Dato de estado" consta de cuatro dígitos (dos bytes) hexadecimales. El byte de la izquierda indica el modo de operación de la CPU y el byte de la derecha indica el tamaño del área de programa.



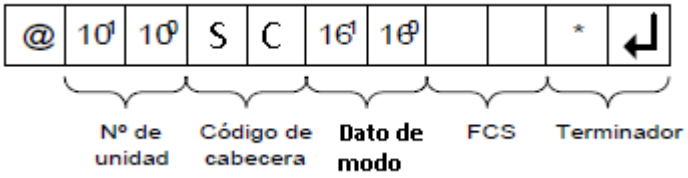


En los PLCs CQM1, el parámetro "Mensaje" es un número FAL/FALS que existe cuando el comando se ejecuta. Cuando no hay mensaje, este parámetro se omite. En los PLCs CPM1/CPM1A/SRM1, el parámetro "Mensaje" es un mensaje de 16 caracteres que existe cuando se ejecuta el comando. Cuando no hay mensaje, este parámetro se omite.

Cambio de modo del PLC--SC#

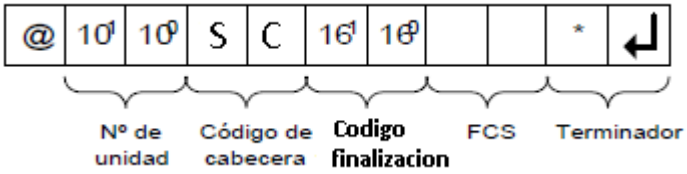
Cambia el modo de operación del PLC.

Formato de comando.



Formato respuesta.

Un código de fin 00 indica finalización normal.



Dato de modo (Comando)

"Dato de modo" consta de dos dígitos (un byte) hexadecimales. Con los dos bits de mayor peso se especifica el modo de operación del PLC. Seleccionar el resto de bits a "0."

	x 16^1					x 16^0		
Bit	7	6	5	4	3	2	1	0
	0	0	0	0	0	0		

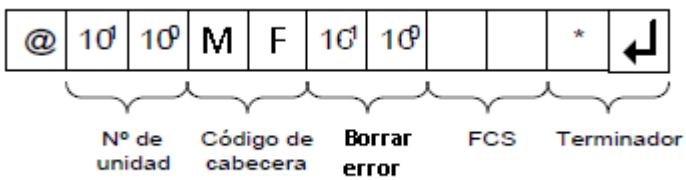
Bit		Modo de operación
1	0	
0	0	Modo PROGRAM
1	0	Modo MONITOR
1	1	Modo RUN

Esta área es diferentes de la de LECTURA DE MODO.

Leer error—MF

Lee y borra errores en el PLC. También chequea si se han borrado los errores anteriores.

Formato de comando.

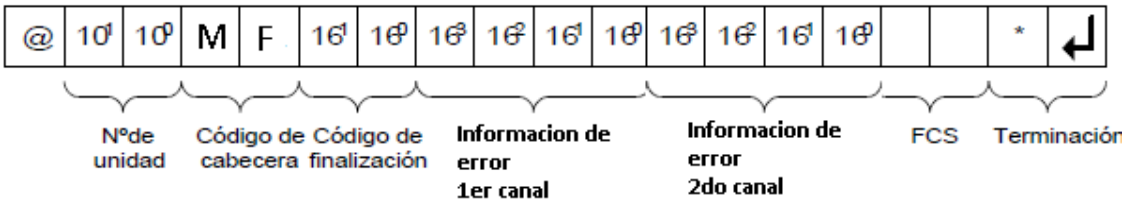


Borrar error

Especificar 01 para borrar errores y 00 para no borrar errores (BCD). Los errores fatales se pueden borrar sólo cuando el PLC está en modo PROGRAM.

Formato respuesta.

Un código de fin 00 indica finalización normal.



Información de error (Respuesta)

La información de error se da en dos canales.

CQM1/CPM1/CPM1A

1er canal

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x 16^3				x 16^2					x 16^1				x 16^0		
			0	0	0		0			0		0	0	0	0	0

ON: Error de batería (Código de error F7, sólo CQM1)
ON: Error de sistema (FAL)
ON: Error de memoria (Código de error F1)
ON: Error de bus de E/S (Código de error C0)
ON: Error de ausencia instrucción end (FALS)
ON: Error de sistema (FAL)

2do canal

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x 16^3				x 16^2					x 16^1				x 16^0		
	0	0			0	0	0	0								

FAL, FALS No. (BCD 00 a FF)
ON: Excedido tiempo scan (Cód. error F8)
ON: Overflow unidad E/S (Cód. error E1)

SRM1

1er canal

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x 16^3				x 16^2					x 16^1				x 16^0		
			0	0	0	0	0			0		0	0	0	0	0

ON: Error de batería (Código de error F7)
ON: Error de sistema (FAL)
ON: Error de memoria (Código de error F1)

ON: Error de ausencia de instrucción end (FALS)
ON: Error de sistema (FAL)

2do canal

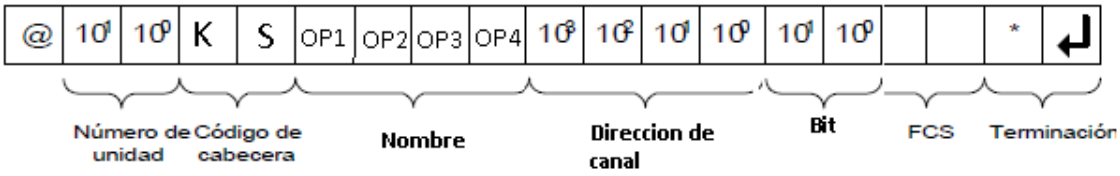
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x 16^3				x 16^2					x 16^1				x 16^0		
	0	0	0		0	0	0	0								

FAL, FALS No. (00 a 99)
ON: Excedido tiempo de scan (Código de error F8)

Forzado a ON – KS

Fuerza a set un bit del área de IR, SR, LR, HR, AR o TC. Una vez que se ha forzado a set o a reset un bit, ese estado se retendrá hasta que se transmita CANCELACION DE FORZADOS (KC) o se transmita el siguiente comando FORZAR SET/RESET.

Formato de comando.



Nombre, dirección de canal, Bit

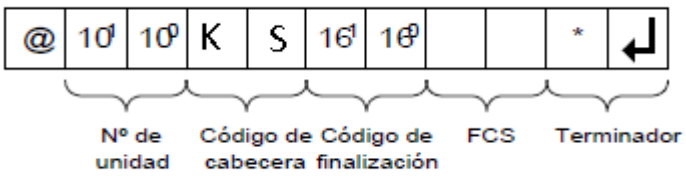
En “Nombre”, especificar el área (es decir, IR, SR, LR, HR, AR o TC) que se ha de forzar a set. Especificar el nombre en cuatro caracteres. En “Dirección de canal”, especificar la dirección del canal y en “Bit” el número del bit que se ha de forzar a set.

Nombre				Clasificación	Rango de selección de direcciones de canal		Bit
OP1	OP2	OP3	OP4		PLCs CQM1	PLCs CPM1/CPM1A/ SRM1	
C	I	O	(Espacio)	IR or SR	0000 a 0252	0000 a 0019 0200 a 0252	00 a 15 (decimal)
L	R	(Espacio)	(Espacio)	LR	0000 a 0063	0000 a 0015	
H	R	(Espacio)	(Espacio)	HR	0000 a 0099	0000 a 0019	
A	R	(Espacio)	(Espacio)	AR	0000 a 0027	0000 a 0015	
T	I	M	(Espacio)	Indicador de finalización (temporiza-dor)	0000 a 0511	0000 a 0127	Siempre 00
T	I	M	H	Indicador de finalización (contador de alta velocidad)			
C	N	T	(Espacio)	Indicador de finalización (contador)			
C	N	T	R	Indicador de finalización (contador reversible)			

El área especificada en “Nombre” se debe dar en cuatro caracteres. Rellenar los huecos vacíos con espacios hasta completar los cuatro caracteres.

Formato respuesta.

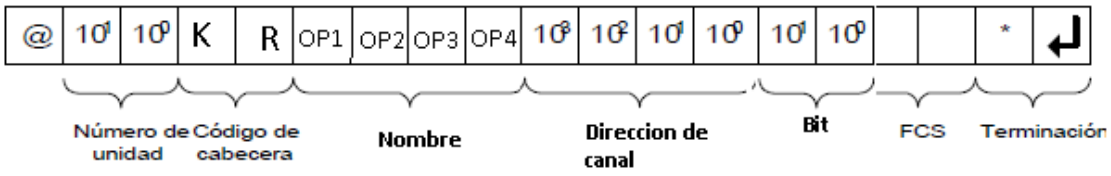
Un código de fin 00 indica finalización normal.



Forzado a OFF – KR

Fuerza a reset un bit del área de IR, SR, LR, HR, AR o TC. Una vez que se ha forzado a set o a reset un bit, ese estado se retendrá hasta que se transmita CANCELACION DE FORZADOS (KC) o se transmita el siguiente comando FORZAR A ON/OFF.

Formato de comando.



Nombre, dirección de canal, Bit.

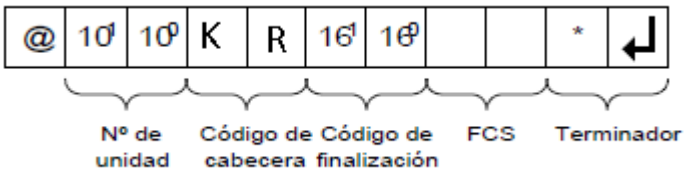
En “Nombre,” especificar el área (IR, SR, LR, HR, AR o TC) que se ha de forzar a off. Escribir el nombre en cuatro caracteres. En “Dirección de canal”, especificar la dirección del canal y en “Bit” el número del bit que se ha de forzar a off.

Nombre				Clasificación	Rango de selección de direcciones de canal		Bit
OP1	OP2	OP3	OP4		PLCs CQM1	PLCs CPM1/CPM1A/ SRM1	
C	I	O	(Espacio)	IR or SR	0000 a 0252	0000 a 0019 0200 a 0252	00 a 15 (decimal)
L	R	(Espacio)	(Espacio)	LR	0000 a 0063	0000 a 0015	
H	R	(Espacio)	(Espacio)	HR	0000 a 0099	0000 a 0019	
A	R	(Espacio)	(Espacio)	AR	0000 a 0027	0000 a 0015	
T	I	M	(Espacio)	Indicador de finalización (temporizador)	0000 a 0511	0000 a 0127	Siempre 00
T	I	M	H	Indicador de finalización (contador de alta velocidad)			
C	N	T	(Espacio)	Indicador de finalización (contador)			
C	N	T	R	Indicador de finalización (contador reversible)			

El área especificada en”Nombre” debe darse en cuatro caracteres. Rellenar con espacios hasta completar los cuatro caracteres.

Formato respuesta.

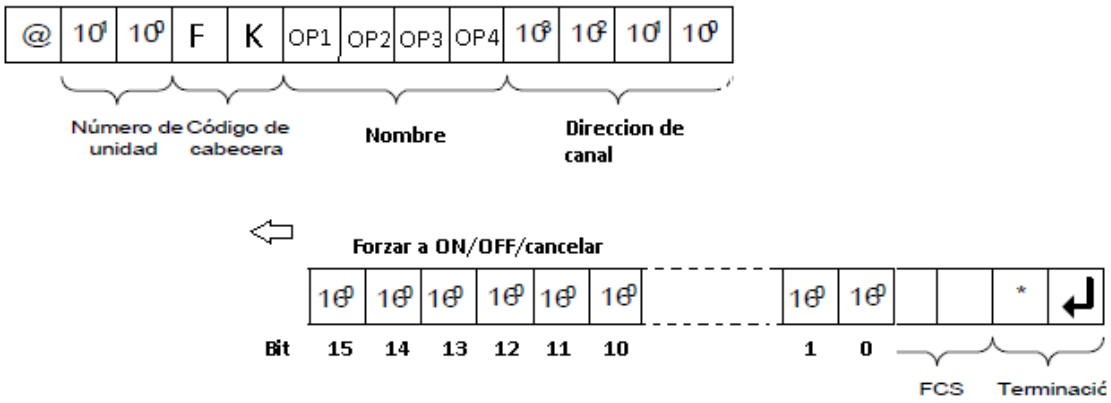
Un código de fin 00 indica finalización normal.



Forzados multiples a ON/OFF --FK.

Fuerza a ON, fuerza a OFF o cancela el estado de los bits en un canal en el área IR, SR, LR, HR, AR o TC.

Formato de comando.



Nombre, dirección de canal

En "Nombre", especificar el área (es decir, IR, SR, LR, HR, AR o TC) que se ha de forzar a ON/OFF. Especificar el nombre en cuatro caracteres. En "Dirección de canal", especificar la dirección del canal y en "Bit" el número del bit que se ha de forzar a ON/OFF.

Nombre				Clasificación	Rango de selección de dirección de canal	
OP1	OP2	OP3	OP4		PLCs CQM1	PLCs CPM1/CPM1A/SRM1
C	I	O	(S)	IR o SR	0000 a 0252	0000 a 0019 0200 a 0252
L	R	(S)	(S)	LR	0000 a 0063	0000 a 0015
H	R	(S)	(S)	HR	0000 a 0099	0000 a 0019
A	R	(S)	(S)	AR	0000 a 0027	0000 a 0015
T	I	M	(S)	Indicador de terminación (temporizador)	0000 a 0511	0000 a 0127
T	I	M	H	Indicador de terminación (temporizador de alta velocidad)		
C	N	T	(S)	Indicador de terminación (contador)		
C	N	T	R	Indicador de terminación (contador reversible)		

(S): Espacio

Dato de forzado set/reset/cancelar

Si se especifica un indicador de finalización de temporizador o de contador, sólo es efectivo el bit 15 y se ignoran el resto de bits. Con temporizadores y contadores sólo es posible forzar a ON o forzar a OFF.

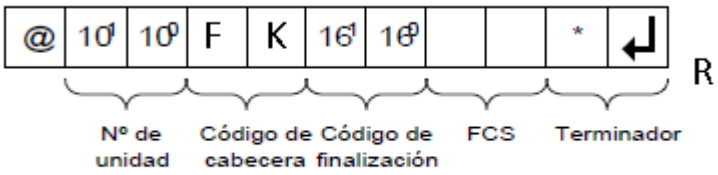
Si se especifica una dirección de canal, el contenido del canal especifica el proceso deseado para cada bit en el canal especificado, como se indica en la siguiente tabla.

Selección hexadecimal	Proceso
0000	No acción (no cambia el estado del bit)
0002	Reset
0003	Set
0004	Forzar-reset
0005	Forzar-set
0008	Cancelar estados set/reset forzados

Los bits no forzados pueden cambiar su estado en la siguiente ejecución del programa, pero aquéllos forzados mantendrán el estado forzado hasta que se cancele.

El concepto especificado en "Nombre" se debe dar en cuatro caracteres. Rellenar con espacios los huecos vacíos hasta completar cuatro caracteres.

Formato respuesta.
Un código de fin 00 indica finalización normal.

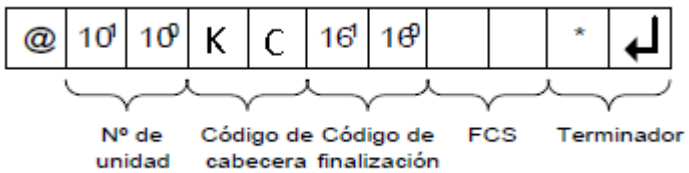


Cancelar forzados --KC.

Cancela todos los bits forzados a ON o a OFF (incluyendo los seleccionados por FORZAR A ON, FORZAR A OFF y FORZADOMULTIPLE). Si se han forzado a set varios bits, el estado forzado se cancelará para todos ellos. Utilizando KC no es posible cancelar bits uno por uno.

Formato de comando.

Formato respuesta.
Un código de fin 00 indica finalización normal.



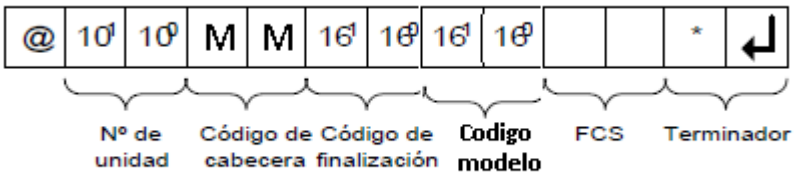
Leer Modelo de PLC --MM.

Lee el modelo de PLC.

Formato de comando.

Formato respuesta.

Un código de fin 00 indica finalización normal.



Código de modelo

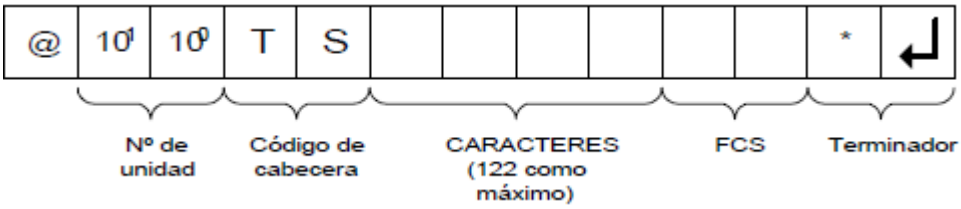
"Código de modelo" indica mediante dos dígitos hexadecimales el modelo de PLC.

Código de modelo	Modelo
01	C250
02	C500
03	C120
0E	C2000
10	C1000H
11	C2000H/CQM1/CPM1/CPM1A/SRM1
12	C20H/C28H/C40H/C200H/C200HS
20	CV500
21	CV1000
22	CV2000
40	CVM1-CPU01-E
41	CVM1-CPU11-E
42	CVM1-CPU21-E

Prueba de comunicación --TS.

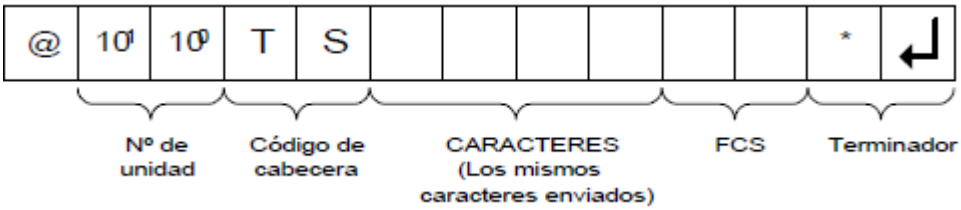
Devuelve, sin cambios, un bloque de datos transmitido desde el ordenador.

Formato de comando.



Formato respuesta.

Un código de fin 00 indica finalización normal.



Caracteres (Comando, Respuesta)

Para el comando, esta selección especifica cualquier carácter distinto de retorno de carro (CHR$(13)). Para la respuesta, se devolverán los mismos caracteres especificados por el comando, si el test es positivo.

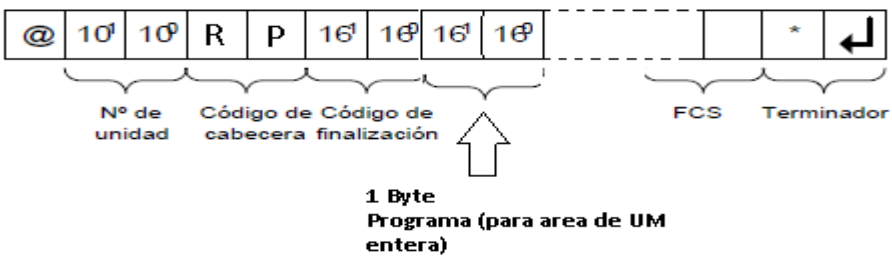
Lectura de programa --RP.

Lee los contenidos del área de programa de usuario del PLC en lenguaje máquina (código objeto). Los contenidos se leen como un bloque, desde el principio hasta el final.

Formato de comando.

Formato respuesta.

Un código de fin 00 indica finalización normal.



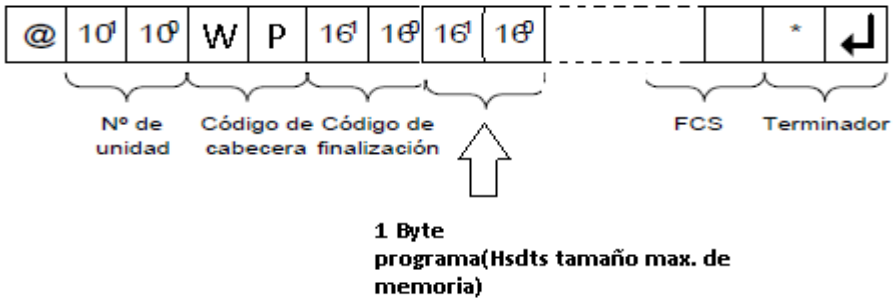
Programa (Respuesta)
Se lee el programa desde el área completa de programa.

Para parar la ejecución de esta operación, ejecutar el comando ABORTAR (XZ).

Escritura de programa --RP.

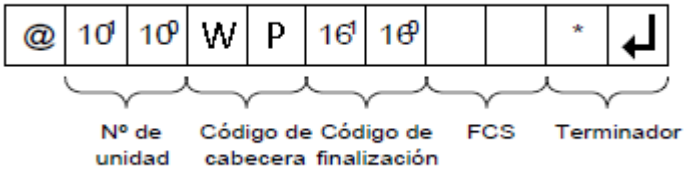
Escribe en el área de programa de usuario del PLC el programa en lenguaje máquina (código objeto) transmitido desde el ordenador. Los contenidos se escriben como un bloque, desde el principio.

Formato de comando.



Programa
Programa hasta capacidad máxima de memoria.

Formato respuesta.
Un código de fin 00 indica finalización normal.



Comando múltiple – QQ.

Registra en el PLC todos los bits, canales y temporizadores/contadores que se han de leer y lee el estado de todos ellos como un grupo.

Formato de comando.

@ x 10^1 x 10^0 Q Q M R OP1 OP2 OP3 OP4 x 10^3 x 10^2 x 10^1 x 10^0 OP1 OP2 ,

No. nodo | Cabecera | Sub-Cabecera | Lectura área | Lectura direcciones de canales | Formato datos | Separador datos

Información de una lectura

Información de lectura total (128 máx.)

, OP1 OP2 OP3 OP4 x 10^3 x 10^2 x 10^1 x 10^0 OP1 OP2 * r

Separador datos | Lectura área | Lectura de direcciones de canal | Formato datos | FCS | Termina-ción

Información de una lectura

Información de lectura total (128 máx.)

Leer área

Especifica en cuatro caracteres el área a leer. Los códigos que se pueden especificar se listan en la siguiente tabla.

Leer dirección de canal, formato de datos

Dependiendo del área y del tipo de datos a leer, la información a leer se muestra en la siguiente tabla. Los “datos a leer” se especifican en cuatro dígitos BCD y el formato de los datos se especifica en dos dígitos BCD.

Clasificación área	Leer datos	Leer área	Leer canal		Formato de datos
			CQM1	CPM1/ CPM1A/ SRM1	
IR o SR	Bit	C I O (S)	0000 a 0255	0000 a 0019 0200 a 0255	00 a 15 (decimal)
	Canal				“CH”
LR	Bit	L R (S) (S)	0000 a 0063	0000 a 0015	00 a 15 (decimal)
	Canal				“CH”
HR	Bit	H R (S) (S)	0000 a 0099	0000 a 0019	00 a 15 (decimal)
	Canal				“CH”
AR	Bit	A R (S) (S)	0000 a 0027	0000 a 0015	00 a 15 (decimal)
	Bit				“CH”
Temporizador	Indicador terminación	T I M (S)	0000 a 0511	0000 a 0127	2 caracteres distintos de “CH”
	PV				“CH”
Temporizador de alta velocidad	Indicador terminación	T I M H	0000 a 0511	0000 a 0127	2 caracteres distintos de “CH”
	PV				“CH”
Contador	Indicador terminación	C N T (S)	0000 a 0511	0000 a 0127	2 caracteres distintos de “CH”
	PV				“CH”
Contador reversible	Indicador terminación	C N T R	0000 a 0511	0000 a 0127	2 caracteres distintos de “CH”
	PV				“CH”
DM	Canal	D M (S) (S)	0000 a 6655	0000 a 1023 6144 a 6655	2 caracteres cualquiera

Para los PLcs SRM1, el rango de DM es de 0000 a 2047.

(S): Espacio

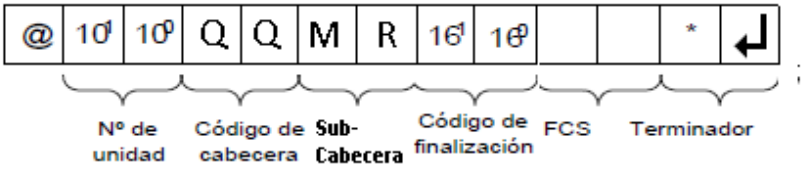
Separador de datos (Comando)

La información se especifica por conceptos separados por códigos separadores (,). El número máximo de conceptos que se pueden especificar es 128.

(Cuando se especifica elPV de un temporizador/contador, se devuelve también el estado del indicador de terminación y por lo tanto se debe contar como dos conceptos).

Formato respuesta.

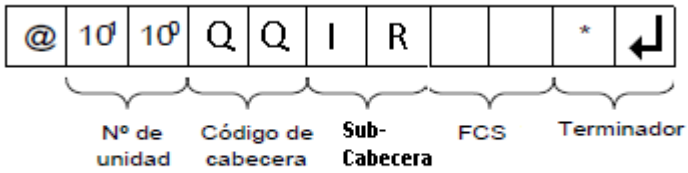
Un código de fin 00 indica finalización normal.



Lectura de bloque

El estado de bit, canal y temporizador/contador se lee como un grupo, de acuerdo con la información leída registrada con QQ.

Formato de comando.



Formato respuesta.

Un código de fin 00 indica finalización normal.

@ x 10^1 x 10^0 Q Q I R x 16^1 x 16^0 ON/OFF x 10^3 x 10^2 x 10^1 x 10^0 ,

No. nodo | Cabecera | Sub-Cabecera | Fin | Temporizador/contador Si se especifica PV, también se devuelve el estado del indicador de terminación. | Separador de datos

, ON/OFF , , x 16^3 x 16^2 x 16^1 x 16^0 * r

Bit data ON/OFF | Word data IR, SR, LR, HR, AR, DM | FCS | Terminación

Dato leído.

Los datos leídos se devuelven de acuerdo con el formato de datos y en el orden en que fueron registrados utilizando QQ. Si se especificó “Indicador de terminación”, se devuelven los datos de bit (ON u OFF). Si se especificó “Canal”, se devuelven datos de canal. Si se especifica para temporizadores/contadores “PV”, el PV se devuelve seguido del indicador de terminación.

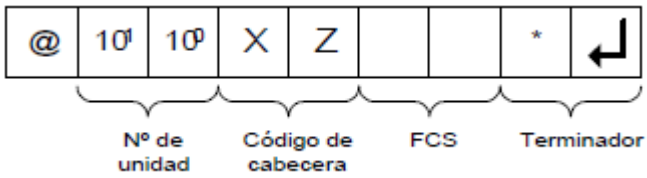
Separador de datos

El código separador (,) se devuelve entre secciones leídas

Abortar – XZ.

Aborta la operación Host Link actualmente en proceso y luego habilita la recepción del siguiente comando. El comando ABORTAR no recibe respuesta.

Formato de comando.



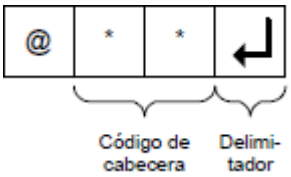
Formato respuesta.

No se gerea respuesta.

Inicializar -- **

Inicializa el procedimiento de control de transmisión para todos los PLCs conectados al ordenador. El comando INICIALIZAR no utiliza números de nodo o FCS y no recibe respuesta.

Formato de comando.



Formato respuesta.

No se gerea respuesta.

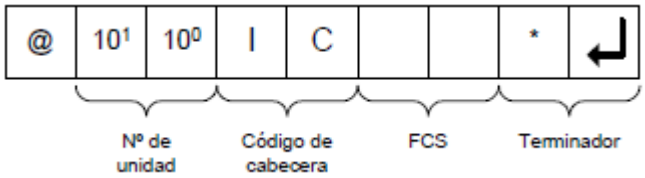
Comando indefinido – IC

Este comando se devuelve si la cabecera de un comando no se puede decodificar. Chequear la cabecera.

Formato de comando.

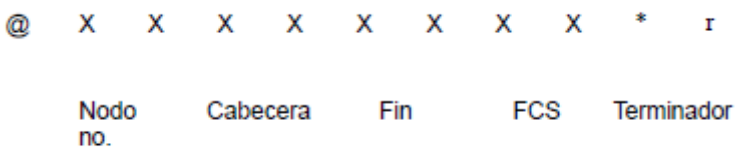
No existe ya que es una respuesta del PLC

Formato respuesta.



2.3.3 Errores de Host Link

Estos códigos de error se reciben como código de respuesta (código fin) cuando no se puede procesar un comando recibido por el PLC enviado por un ordenador. El formato del código de error es el siguiente.



El código de cabecera variara de acuerdo con el comando y puede contener un subcódigo (para comandos compuestos).

Cód. fin	Contenidos	Causa probable	Corrección
00	Finalización normal	---	---
01	No ejecutable en modo RUN	El comando enviado no se puede ejecutar cuando el PLC está en modo RUN.	Comprobar la relación entre el comando y el modo del PLC.
02	No ejecutable en modo MONITOR	El comando enviado no se puede ejecutar cuando el PLC está en modo MONITOR.	
04	Dirección demasiado alta (PLCs CPM1/CPM1A/SRM)	Se ha excedido la dirección más alta del área de programa de usuario.	Comprobar el programa.
0B	No ejecutable en modo PROGRAM	El comando enviado no se puede ejecutar cuando el PLC está en modo PROGRAM.	Este código no está siendo utilizado actualmente.
13	Error de FCS	El FCS es falso. El cálculo del FCS es erróneo o hay influencias adversas de ruido.	Comprobar el método de cálculo de FCS. Si había influencia de ruido, transferir de nuevo el comando.
14	Error de formato	Formato de comando erróneo.	Comprobar el formato y transferir de nuevo el comando.
15	Error de datos de número de entrada	Las áreas para leer y escribir son erróneas.	Corregir las áreas y transferir de nuevo el comando.
16	Comando no soportado	El comando especificado no existe en la dirección especificada. (Leer SV, etc.)	Comprobar dirección e instrucción.
18	Error de longitud de trama	Se ha excedido la longitud máxima de trama.	Dividir el comando en varias tramas.
19	No ejecutable	Parámetros a leer no registrados para comando compuesto (QQ).	Ejecutar QQ para registrar parámetros a leer antes intentar leerlos.
23	Memoria de usuario protegida contra escritura	PLCs CQM1: Pin 1 del interruptor DIP del CQM1 en ON. PLCs CPM1/CPM1A/SRM1: La memoria está protegida en la configuración del PLC	PLCs CQM1: Poner el pin 1 a OFF para ejecutar. PLCs CPM1/CPM1A/SRM1: Cambiar la selección en la configuración del PLC (DM 6602).
A3	Abortado debido a error de FCS en transmitir dato	El error se generó mientras se estaba ejecutando un comando que ocupa más de una trama. **Nota:** Los datos hasta ese punto se graban en la área apropiada de la CPU.	Comprobar los datos de comando e intentar transferir de nuevo.
A4	Abortado debido a error de formato en transmitir dato		
A5	Abortado debido a error de dato de número de entrada en transmitir dato		
A8	Abortado debido a error de longitud de trama en transmitir dato		
Otro	---	Se ha recibido ruido.	Transferir de nuevo el comando.

Cortes de alimentación

Si se produce un corte de alimentacion, se pueden recibir las siguientes respuesta del PLC. Si cualquiera de estas respuestas se recibe durante o despues de un corte de alimentacion, repetir el comando.

Respuesta de comando indefinido

@00IC4A*\r

No Respuesta

Si no se recibe respuesta, abortar el ultimo comando y volver a enviarlo.

3 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Breve explicación sobre POO de lo utilizado en este proyecto para una mayor información consultar con manuales de programación específicos de POO.

La **programación orientada a objetos** (**POO**, u **OOP** según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Muchos de los objetos pre-diseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

La **POO** permite realizar grandes programas mediante la unión de elementos mas simples, que pueden ser diseñados y comprobados de manera independiente del programa que a a usarlo. Muchos de estos elementos podrán ser reutilizados en otros programas.

A estas piezas, módulos o componentes que interactúan entre si cuando se ejecuta un programa, se les denomina **objetos**. Estos objetos contienen tanto datos como las funciones que actúan sobre estos datos.

Durante la ejecución del programa, los objetos interactúan pasándose mensajes y respuestas. Es fundamentalmente darse cuenta de que un objeto no necesita conocer el funcionamiento interno de los demás objetos para poder interactuar con ellos, si no que le es suficiente con saber la forma en que debe enviarle sus mensajes y como va a recibir la respuesta.

La definición genérica de esos objetos se realiza mediante las **clases.** Así, una clase contiene una completa y detallada descripción de la información y las funciones que contendrán cada objeto de esa clase. Las clases de C++ se pueden ver como una generalización de las estructuras.

En C++ las clases son verdaderos tipos de datos definidos por el usuario y pueden ser utilizados de igual manera que los tipos de datos propios de C++, tales como **int** o **float,** Los objetos son a las clases como las variables a los tipos de variables. Un objeto tiene su propio conjunto de datos o variables miembro, aunque no de funciones, que aunque se aplican a un objeto concreto son propias de la clase a la que pertenece el objeto.

Ejemplo ilustrativo de una clase vehículo:

Datos (variables)	Métodos (funciones)
Color Puertas	Arrancar Parar
Cilindrada	Girar
Potencia	Acelerar
Tipo de combustible	Encender/apagar luces
Luces	…
…	

Al instanciar o crear un objeto de la clase vehículo en el programa principal ese objeto instanciado tendrá todas las características de la clase vehículo.

En la POO, el centro del lenguaje son los objetos, que contienen datos y funciones concretas que permiten manipularlos y trabajar sobre ellos.

Para proteger a las variables de modificaciones no deseadas esta la encapsulación, ocultamiento o abstracción de datos. Los miembros de una clase se pueden dividir en públicos y privados. Los públicos son aquellos a los que se pueden acceder libremente desde fuera de la clase. Los privados, por lo contrario, solamente pueden ser accedidos por los métodos de la propia clase.

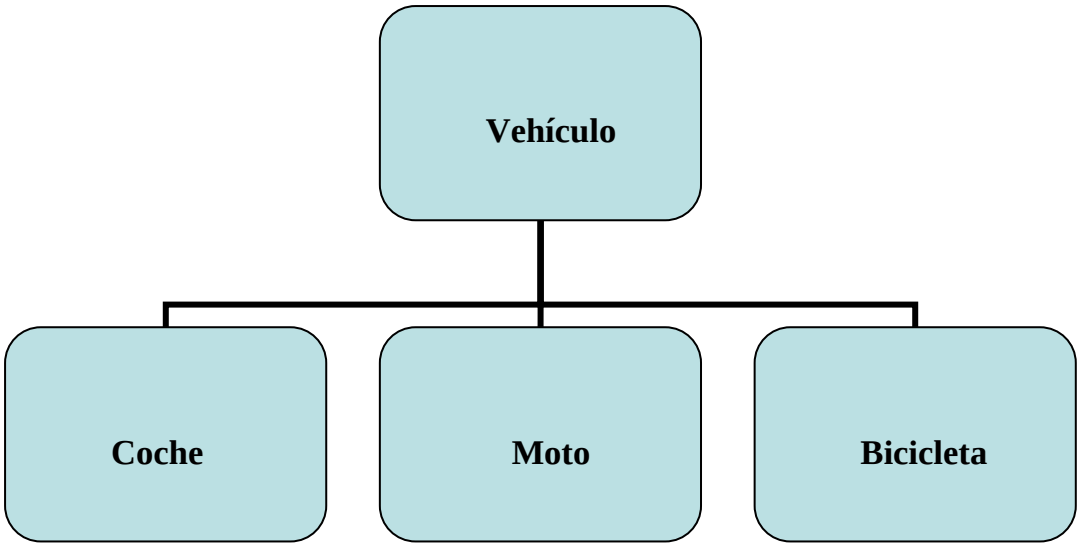
Las clases ofrecen un conjunto de funciones públicas a través de las cuales se pueden actuar sobre los datos, que serán privados. Estas funciones o métodos públicos constituyen la interface de la clase. De esta manera se garantiza que se hace buen uso de los objetos, manteniendo la coherencia de la información. Esto sería imposible si se accediera libre e independientemente a cada variable miembro.

Herencia

Es una característica de la programación orientada a objetos y más concretamente del C++, permite definir una clase modificando una o más clases ya existentes. Estas modificaciones consisten habitualmente en añadir nuevos miembros (variables o funciones) a la clase que se está definiendo, aunque también se puede redefinir variables o funciones miembro ya existentes.

La clase de la que se parte en este proceso recibe el nombre de **clase base**, y la nueva clase que se obtiene se denomina clase derivada. Esta a su vez puede ser clase base en un nuevo proceso de derivación, iniciando de esta manera una jerarquía de clases. De ordinario las clases bases suelen ser más genéricas que las clases derivadas.

En algunos casos una clase no tiene otra utilidad que la de ser clase base para otras clases que se deriven de ella. A este tipo de clases base, de las que no se declara ningún objeto, se les denomina **clase base abstracta** (ABC, Abstract Base Class) y su función es la de agrupar miembros comunes de otras clases que se deriven de ellas. Por ejemplo:



Pero todos los objetos que se declaren o instancien pertenecerán a alguna clase de las últimas de la jerarquía. Pero también puede ser que la clase base se declaren objetos, ya no siendo una **clase base abstracta.**

Este mecanismo de herencia presenta múltiples ventajas evidentes a primera vista, como la posibilidad de reutilizar código sin tener que escribirlo de nuevo. Esto es posible porque todas las clases derivadas pueden utilizar el código de la clase base sin tener que volver a definirlo en cada una de ellas.

Uno de los principales problemas que aparece en la herencia es el control de acceso a los datos. ¿ puede una función de una clase derivada acceder a los datos privados de su clase base? En principio una clase no puede acceder a los datos privados de otra, pero podría ser muy conveniente que una clase derivada accediera a todos los datos de su clase base. Para hacer posible esto, existe el tipo de dato protected. Este tipo de dato es privado para todas aquellas clases que no son derivadas, pero público para una clase derivada de la clase en la que se ha definido la variable como protected.

En la siguiente tabla se ilustra cómo se derivan los datos:

Tipo de dato clase Base	Clase derivada de una clase base publica	Clase derivada de una clase base privada	Otras clases sin relación de herencia con clase base
Privada	No accesible Directamente	No accesible Directamente	No accesible Directamente
Protegido	Protegida	Privado	No accesible Directamente
publico	publico	privado	Accesible mediante

4 FUNCIONES API DE WINDOWS

La **interfaz de programación de aplicaciones de Windows**, **Windows API** (*Windows application programming interface*), es un conjunto de **funciones** residentes en **bibliotecas** (generalmente **dinámicas**, también llamadas DLL por sus siglas en inglés, término usado para referirse a éstas en Windows) que permiten que una aplicación corra bajo un determinado sistema operativo.

Debido a su estrecha relación con el desarrollo de **software**, los programas en sus especificaciones generalmente explicitan la versión de la API del sistema operativo, mediante diversas nomenclaturas tales como la versión específica del sistema operativo (para **Windows 98**, por ejemplo), o explicitando la versión del conjunto de bibliotecas (*Plataforma Win32*, etc.).

Las funciones API se dividen en varias categorías:

- Depuración y manejo de errores
- E/S de dispositivos
- Varias DLL, procesos e hilos
- Comunicación entre procesos
- Manejo de la memoria
- Monitorización del desempeño
- Manejo de energía
- Almacenamiento
- Información del sistema
- GDI (interfaz para dispositivos gráficos) de Windows (tales como impresoras)
- Interfaz de usuario de Windows

4.1.1 Funciones API para comunicaciones serie

La comunicación por el puerto serie puede hacerse de las siguientes maneras, utilizando programación a bajo nivel (directamente sobre la UART), a nivel medio (mediante la BIOS) y mediante lenguaje de alto nivel (Builder C++) y las funciones que la API de Windows (API 32) nos proporciona, ya que como se ha explicado con anterioridad las API son funciones, esta es la manera con la cual realizo la comunicación entre mi PLC virtual y el dispositivo de monitorización. Así generando un emulador de comunicaciones serie.

Para ello deberemos ver como se estructura las aplicaciones en Win32, que funciones disponemos para el acceso al puerto y que posibilidades tenemos.

Es importante destacar que los sistemas operativos MS Windows 9X y NT son sistemas operativos multitarea. Es decir que simultáneamente se están ejecutando en el ordenador diversos programas; no en la CPU que solo puede ejecutar uno a la vez. Esto significa que todos los programas comparten la misma CPU y por tanto, mientras se está ejecutando un proceso o tarea no se está ejecutando realmente otro.

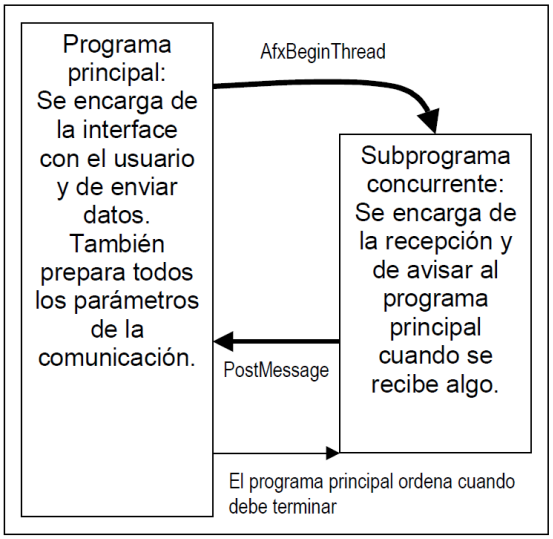
Por ello mi programa no debe ejecutarse "inútilmente" para malgastar tiempo de CPU: no utilizo algoritmos de POOLING o consulta para realizar acciones, por ejemplo comprobar si hemos recibido algún dato por el puerto serie. Sin embargo directamente desde Win32 tampoco podemos utilizar el método de interrupciones puesto que las gobierna el sistema operativo y son "utilizables directamente" como ocurría en MS-DOS o en los microcontroladores.

Solución: Las funciones del sistema operativo son bloqueantes de forma que "desalojan" la CPU mientras no se pueden realizar (porque nadie ha enviado nada). Es decir, si ejecutamos la rutina de leer del puerto serie (o de donde sea), podemos hacer que el programa se pare hasta que esta lectura del dato se realice:

```
…..programa…..

Leer puerto serie

.. .ya tenemos el dato leído, y si no había ninguno para leer se

… habrá esperado hasta tenerlo….
```

Problema: si el programa se queda bloqueado esperando recibir algo, no puede realizar ninguna acción más: es decir no puede enviar tampoco.

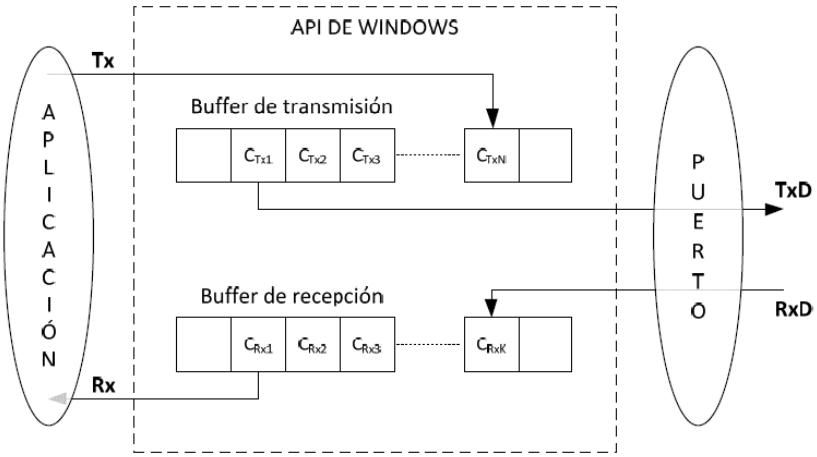
Y aquí es donde la cosa es complicada: la solución pasa por arrancar un segundo *thread* (o hilo, o hebra): Es una rutina, programa, etc. Que se ejecuta al mismo tiempo que el programa principal. Con ello, podemos arrancar un thread que se encargue de la recepción de los datos del puerto serie, mientras que el programa principal se encarga de controlar el interfaz con el usuario y el envió de datos por el puerto. NO UTILIZO ESTE METODO POR SU COMPLEJIDAD.



Para simplificar el programa realizaremos envíos y recepción que bloquean al programa principal. Esto no supondrá ningún problema para el cas del envió, ya que los bits se envían aunque nadie esté recibiéndolos, pero sí que será un poco más complicado para la recepción, ya que cuando tengamos que recibir X bit, hasta que no se reciben, no sale de la función.

Buffer de transmisión y recepción:

Con esta API lo que se realiza es un buffer virtual donde se almacenan los bits recibidos por el puerto serie hasta que la aplicación requiera de ellos o directamente envía los datos cuando la aplicación pone los bits en este buffer y son llevados directamente al puerto y transmitidos instantáneamente.



Llamadas de Win32 relacionadas con la comunicación serie:

CreateFile()	**Abre el puerto serie. Igual que con los ficheros**
SetCommState()	Configura las comunicaciones
GetCommState()	Lee la configuración actual de las comunicaciones
ReadFile()	Lee del puerto serie.
WriteFile()	Escribe en el puerto serie.
CloseHandle()	Cierra el puerto serie para que lo puedan usar otros programas
SetCommMask()	Establece que tipo de información despierta a WaitCommEvent
WaitCommEvent()	Espera a que ocurra un evento: recepción, envio, errores,..
SetupComm()	Establece el tamñano de los buffers.
PurgeComm()	Limpia los buffer y posibles errores ocurridos.
SetCommTimeouts()	Establece los tiempos máximos de espera para las operaciones.
ClearCommError()	Limpia errores producidos.

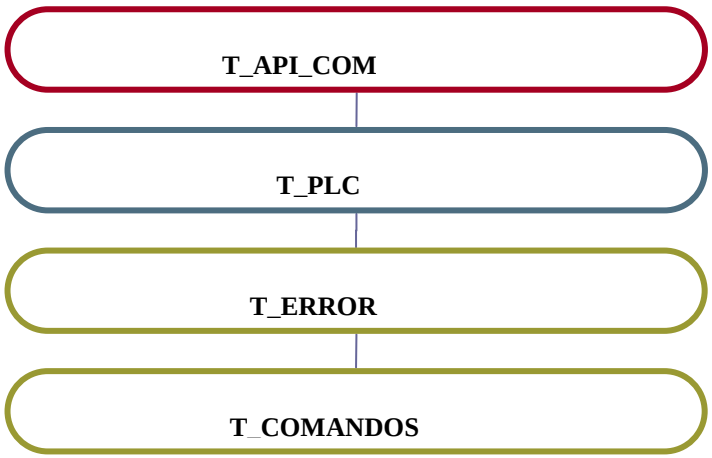
5 DISEÑO E IMPLEMENTACIÓN DEL EMULADOR

Mi primer problema para empezar a realizar el emulador de comunicaciones fue que no sabía cómo podía depurar cada comando con los errores Host link, para así asegurar que el comando estaba bien recibido y era correcto, ósea simulando el PLC.

Viendo detalladamente todos los errores Host link vi que con cada comando no tenía sentido revisar todos dichos errores ya que algunos son específicos. Entonces la idea mía fue la de crearme una clase (T_Error) y que en esta clase tuviera métodos o funciones que depuraran cada uno de los errores Host link, estos métodos lo que realizan es la comprobación del comando, si cumple con las especificaciones del comando y devolviendo un valor booleano si todo es correcto o no es correcto.

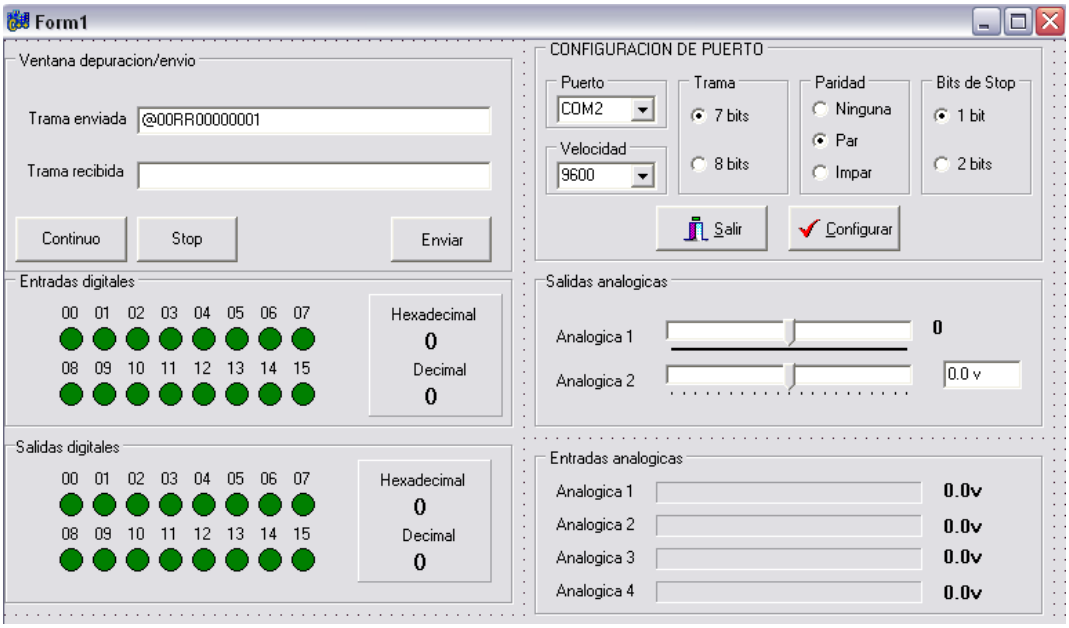
Ya depurado los errores sabiendo si el comando está bien o mal pase a la siguiente fase de cómo gestionar o como realizar lo que el comando mandaría a ejecutar y entonces para ello lo que pensé fue en crearme la clase (T_Comandos) con todos los comandos como métodos de dicha clase, cuando se llama un método de estos en este caso un comando, se le pase el comando y el internamente lo que hace es revisar si el comando está bien y lo hace uno a uno de los posibles errores que puede tener dicho comando por medio de los métodos de la clase (T_error) si esta cadena de caracteres esta bien el método devuelve un booleano donde dirá si esta todo correcto o no y se retornara a la función (**compro**) principal del programa principal que será la encargada de ejecutar o realizar la parte correspondiente de dicho comando ya sea leer, escribir, ver estados, ver errores, prueba, etc. Ya ejecutado el comando, la función devuelve la cadena de respuesta ya sea correcta o falsa a devolver al dispositivo de supervisión y así supervisando el correcto funcionamiento del emulador.

El software o aplicación del emulador de comunicaciones serie está organizado con la siguiente distribución de las clases que lo componen y de su correspondiente herencia (POO) ya explicada en aparatados anteriores.



La realización del proyecto fue tomando como base una práctica que realice en la asignatura de informática industrial que tuve en el curso 3 de carrera dicha práctica se basaba en el control y monitorización del autómata utilizando el medio físico de comunicación el puerto serie del PC. El protocolo a nivel de enlace es el definido por Omron para sus autómatas programables, denominado Host Link.

El interface es el siguiente:



Como se observa tiene 6 módulos que realizaban las siguientes tareas:

1) Envió y recepción de comandos

2) Configuración del puerto serie

3) Monitorización de entradas digitales

4) Monitorización de salidas digitales

5) Monitorización de salidas analógicas

6) Monitorización de entradas analógicas

Con esta práctica por medio de un modo continuo visualizaba el estado de las entradas digitales, que podían ser cambiadas manualmente en el bastidor del propio autómata en cualquier momento a la vez este valor se hacía un cambio a un valor BCD y se proyectaba en la salida analógica 1, las salidas digitales reflejaban el contenido del área de memoria DM que previamente se avía guardado, la salida analógica 2 se podía modificar por medio de una barra y por ultimo las entradas analógicas se tendría que visualizar su valor bien codificado

dentro del rango permitido para que cuando hubiera un cambio en el potenciómetro del bastidor del autómata esta se reflejaría correctamente.

Todo esto se realizaba por medio de los comandos Host link, formando las cadenas de caracteres oportunas para la monitorización y control del autómata real, esto se realizaba todo en programación en C y utilizando el mismo programa que he utilizado para la creación de este proyecto y es el Builder 6. Lo único que en esta práctica las funciones que realice para el envió de los comandos o más bien para el envió/recepción carácter fue en ensamblador teniendo muy claro todos los registros de la UART para poder saber qué valor tendría que poner en cada registro y que valor tenía que visualizar y controlar para que hubiera una buena comunicación a diferencia de este proyecto con respecto a eso es que en el utilizo las funciones que me proporciona el propio sistema operativo (las APIs de Windows) asi facilitando más a la hora de programar y de poder realizar mayores aplicaciones con respecto a este tema. Ya que en este proyecto como he mencionado con antelación, lo que utilizo son unas clases creadas por mí mismo para la correcta comunicación.

5.1.1 Explicación específica de cada clase

T_API_COM

Es la clase que se encarga de la comunicación por medio de las API de comunicación de Windows, realizando las siguientes funciones:

- Abrir y cerrar puerto serie a utilizar según la necesidad de la aplicación
- configurar el puerto serie a utilizar según la necesidad de la aplicación
- Transmisión y recepción de caracteres o cadena de caracteres (Comandos Host link)

A Continuación se ilustra un trozo de codigo donde se declaran los Datos y Métodos utilizados en esta clase

```
// Declaración de la clase
class T_API_COM {

protected:
short NPuerto; // Número de puerto
bool bEstaAbierto; // ¿ Abierto ?
HANDLE hCom; // Manejador del puerto
HANDLE hComO; // Manejador del otropuerto
DCB sComCfg; // Estructura configuración
COMSTAT sComSta; // Estado del puerto
DWORD dwErr; // Errores del puerto
DWORD dwTamBufIn; // Tamaño del buffer de entrada
DWORD dwTamBufOut; // Tamaño del buffer de salida
COMMTIMEOUTS sTimOut; // Estructura de temporizaciones

public:
DWORD N_caracteres_recibidos;
DWORD N_caracteres_enviar;
T_API_COM(short=1); // Constructor con parámetro por defecto
~T_API_COM(); // Destructor (cerrará el puerto)
bool AbrirPuerto(); // Abrir puerto seleccionado
bool AbrirOtroPuerto(int iPort); // Abrir otro puerto
bool CerrarPuerto(); // Cerrar puerto seleccionado
  bool EstaAbierto(); // Estado actual del puerto
  HANDLE LeeHANDLE(); // Leer "manejador" del COM
 DCB ObtenerConfiguracion(); // Obtener configuración actual
//Modifica ciertos campos de la estructura DCB
  bool EstablecerConfiguracion(DCB sComCfg, DWORD dwVelocidad=9600,
                BYTE bParidad='N', BYTE bBitStop=1, BYTE bTrama=8);
  DWORD BytesEnBufferRx(); // Caracteres pendientes de lectura
  bool RecibeCaracter(BYTE& bDat); // Recibe un carácter
  bool TransmiteCaracter(BYTE bDat); // Transmite un carácter
// Transmitir una cadena de caracteres
  bool TransmitirCadena(char* pCadena);
// Recibir una cadena de caracteres
  bool RecibirCadena(char* pCad, DWORD dwLen);
  DWORD Get_N_caracteres_recibidos();
  DWORD Get_N_caracteres_enviar();
  bool BorrarTxRx();//Borra los buffer de transmicion y recepcion

};
```

T_PLC

Es la clase encargada de simular el comportamiento del Autómata CQM1, teniendo internamente sus correspondientes posiciones de memoria para el correcto comportamiento como un Autómata de la serie CQM1 de Omron, realizando las siguientes funciones:

- Posiciones de memoria (IR,LR,HR,PV,DM,AR,SV)

- Leer y escribir dichas posiciones de memoria

- Establecer los protocolos de un CQM1, simulando (estado de la CPU, modelo de CPU, El código de error del protocolo Host Link, protección contra escritura, Encendido o apagado del PLC, etc…)

A Continuación se ilustra un trozo de codigo donde se declaran los Datos y Métodos utilizados en esta clase

```
class T_plc : public T_API_COM {

      private:
      unsigned          short     estado     ;                  //
3=run,2=monitor,0=program
      bool power;
      unsigned  short  Error_CPU[2];  //guarda  codigo  de
error de CPU
      char* C_error;   //guarda el codigo de finalizacion de
respuesta de los comandos
      char* modelo;
      bool pin_DIP;  //Proteccion contra escritura
      bool Trama_larga;


      unsigned  short  IR[256];
      unsigned  short LR [64];
      unsigned  short HR [100];
      unsigned  short PV [512];
      bool           TC [512];
      unsigned  short DM [1024];
      unsigned  short AR [28];
      unsigned  short SV [512];
      unsigned  char   SV_prefijo[512];

   public:
       //constructores
       T_plc(void );  //sin parametros
       T_plc(int ,bool ,bool  ,char* );  //con parametros
       //Destructor
       ~T_plc();
       void  Set_estado(unsigned short);       // 0 monitor
                              //2 programa
                              //3 run
       unsigned  short   Get_estado   ();             //enviar
estado plc

 //------   Error del PLC --------
       // void Set_error(bool);
       // bool Get_error();

          //------   Error del PLC --------
 void Set_eCPU(unsigned short,unsigned short);
 unsigned short Get_eCPU(unsigned short);

 //------  Codigo de Finalizacion ------

 void Set_ComError(char*);
 char* Get_comError(void );

 //--------- Proteger escritura de memoria de usuario----
 void Set_DIP(bool);
 bool Get_DIP(void);

 //--------- Trama larga ----
 void Set_Trama_larga(bool);
 bool Get_Trama_larga(void);

//----------Memorias----------------
 void Set_IR(unsigned short, unsigned short);
 void Set_LR(unsigned short, unsigned short);
 void Set_HR(unsigned short, unsigned short);
 void Set_PV(unsigned short, unsigned short);
 void Set_TC(unsigned short, bool);
 void Set_DM(unsigned short, unsigned short);
 void Set_AR(unsigned short, unsigned short);
 void Set_SV(unsigned short, unsigned short);
 void Set_SV_prefijo(unsigned short, unsigned char);

 unsigned short Get_IR(unsigned short);
              unsigned short Get_LR(unsigned short);
              unsigned short Get_HR(unsigned short);
              unsigned short Get_PV(unsigned short);
              bool Get_TC(unsigned short);
              unsigned short Get_DM(unsigned short);
              unsigned short Get_AR(unsigned short);
              unsigned short Get_SV(unsigned short);
              unsigned char Get_SV_prefijo(unsigned short);
              int conv(char ) ; //---cambiar caracter a valor
              entero----------------------
              };
```

T_ERROR

Es la clase donde están implementados los posibles errores que pueden haber en los comandos Host link, es aquí donde cada comando vendrá a depurarse y analizar si dicho comando es correcto.

A Continuación se ilustra un trozo de codigo con los Datos y Métodos utilizados en esta clase

```
class T_Error:public T_plc{

    private:
        char* e_Comando;
    public:
        //constructores
        T_Error(void); //sin  parametros
        T_Error(char* );  //con parametros
        //Destructor
        ~T_Error();

      bool e00(char*); //Terminacion normal
      bool e01(char*);//No ejecutable en RUN
      bool e02(char*);//No ejecutable en MONITOR
      bool e03(char* ,unsigned int, unsigned int); //proteccion de
escritura
      bool e04(char*,unsigned int); //Direccion Excedida
      bool e0B(char*);//No ejecutable Program
      bool e13(char*);//Error de FCS
      bool e14(char*,unsigned int ); //Error de formato
 bool e15(char*, unsigned int , unsigned int ); //Error       de nº
datos de entrada
  bool e16(char*);//comando no existente
      bool e18(char*);//Error en la longitud de la trama
        bool e19(char*);//No ejecutable
        bool e21(void); //Error de CPU

        bool e23(char*);//Memoria usuario protegida
        bool eA3(char*);//cancelación por error FCS
        bool eA4(char*);//candelado debido al formato
        bool eA5(char*);// candelado debido al dato
        bool eA8(char*); candelado debido a la longitud
        unsigned long rango(char*,bool) ;
        void FCS(char* pCad1 )  ; //calculo de FCS


              };
```

T_COMANDOS

Es la clase que implementa todos los comandos Host link. Y estando en la última cadena de la jerarquía, hereda todas las funciones de las anteriores clases por eso con esta clase es por la cual se instancia el objeto de nombre (**oTplc**). Esta clase como he dicho crea las funciones de cada comando ósea recibe los comandos de la aplicación recibidos por las funciones de T_API_COM y pasa a depurarlos por medio de las funciones de T_ERROR pero claro teniendo en cuenta la situación del emulador con las funciones T_PLC, realizando la función o tarea a realizar por medio del comando si esta correcto y devolviendo una respuesta para que pueda ser tratada externamente ya sea por otra aplicación, SCADA, HDMI o cualquier otro dispositivo que entienda el protocolo Host link.

A Continuación se ilustra un trozo de codigo donde se declaran los Datos y Métodos utilizados en esta clase

```
class T_Comandos:public T_Error{

  private:
     char C_prueba[32];
     int estado1;//Estado del PLC
     bool ok ;
     unsigned int jj;  //para saber la longitud del comando recibido si es ok
   public:
       //constructores
       T_Comandos(char* );  //con parametros

       //Destructor
       ~T_Comandos();

     bool RR (char* pCad1);  //      LECTURA DE AREA IR/SR 335
     bool RL (char* pCad1);  //   LECTURA DE AREA LR 336
     bool RH (char* pCad1);  //   LECTURA DE AREA HR 336
     bool RC (char* pCad1);  // LECTURA DE PV 337
     bool RG (char* pCad1);  //  LECTURA DE ESTADO DE TC 337
     bool RD (char* pCad1);  //   LECTURA DE AREA DE DM 338
     bool RJ (char* pCad1);  //  LECTURA DE AREA DE AR 338
     bool WR (char* pCad1);  //   ESCRITURA DE AREA IR/SR 339
     bool WL (char* pCad1);  // ESCRITURA DE AREA LR 339
     bool WH (char* pCad1);  //   ESCRITURA DE AREA HR 340
     bool WC (char* pCad1);  //  ESCRITURA DE PV 340
     bool WG (char* pCad1);  //      ESCRITURA DE ESTADO DE TC 341
     bool WD (char* pCad1);  //  ESCRITURA DE AREA DM 342
     bool WJ (char* pCad1); //  ESCRITURA DE AREA AR 342
       bool Ralmu (char* pCad1);  //  LECTURA SV 1 343
       bool Walmu (char* pCad1);  //   CAMBIAR SV 1 346
       bool MS (char* pCad1);  //  LECTURA DE ESTADO
            348
     bool SC (char* pCad1);  //   ESCRITURA DE ESTADO 349
     bool MF (char* pCad1);  //   LECTURA DE ERROR 350
     bool KS (char* pCad1);  //  FORZADO A ON 351
     bool KR (char* pCad1);  //   FORZADO A OFF 352
     bool FK (char* pCad1);  //      FORZADOS MULTIPLES A ON/OFF 353
     bool KC (char* pCad1);  //    CANCELACION DE FORZADOS 354
     bool MM (char* pCad1);  //  LECTURA DE MODELO DE PLC 354
     bool TS (char* pCad1);  //   PRUEBA DE COMUNICACIONES 355
     bool QQ (char* pCad1);  //  COMANDO COMPUESTO 356
     bool XZ (char* pCad1); //  ABORTAR (sólo comando) 358
     bool aste_aste (char* pCad1);  //   INICIALIZAR (sólo comando) 358

     bool IC(char* pCad1);  //  Comando indefinido


     unsigned short valor_decimal(char* pCad1);   //Cambiar valor a decimal de un hexadecimal

     void trama(char*);

};
```

5.1.2 Explicación de ejecución del emulador

Cuando se ejecuta la aplicación o emulador de PLC lo primero que se hace es instanciar el objeto **(oTplc**), encargado de simular el comportamiento del CQM1 con sus correspondientes, memorias, métodos, clases, comunicaciones, comandos Host link, esto lo desarrollo asi debido a que tengo que simular como se sabe el comportamiento de la comunicación serie pero para que este comportamiento pueda ser más fiable de una manera real lo que realizo es guardar en posiciones de memoria de dicho objeto valores de situaciones de las memorias de un PLC y asi mantengo estos valores en todo momento para su posterior utilización cuando la aplicación lo necesite.

Hay configurar el puerto serie con el cual quiero establecer comunicación con el dispositivo externo que lo monitorizara o controlara, para eso tiene un interfaz gráfico ver siguiente figura, bastante intuitivo y fácil de utilizar, una vez seleccionada dicha configuración al darle a conectar automáticamente esta información es utilizada por las funciones de la clase (T_API_COM) para abrir el puerto con dicha configuración.

Una vez teniendo ya el puerto abierto correctamente con la configuración que quiero hacer la comunicación se activan dos temporizadores que serán los encargados de simular los ciclos del PLC.

En el timer principal (**Timer1**) es el encargado de simular correctamente la sincronización de comunicación, recepción y transmisión de datos en este caso los comandos Host link, lo realizado a través de un bucle que (do while) que lo que hace es esperar dato recibido o un cierto número de intentos de espera, si llegan datos o pasado estos intentos se sale del bucle y se pasa a la comprobación de los comandos por medio de una función [**bool compro(AnsiString *,DWORD,bool)];** creada en la propia aplicación con objetivo de comprobar y ejecutar dicho comando y si todo esto se realiza correctamente se devuelve un valor booleano para poder gestionar la respuesta a enviar (Trama de respuesta) ya sea verdadero, falso, trama norma o trama larga.

La función bool compro(AnsiString *,DWORD,bool)

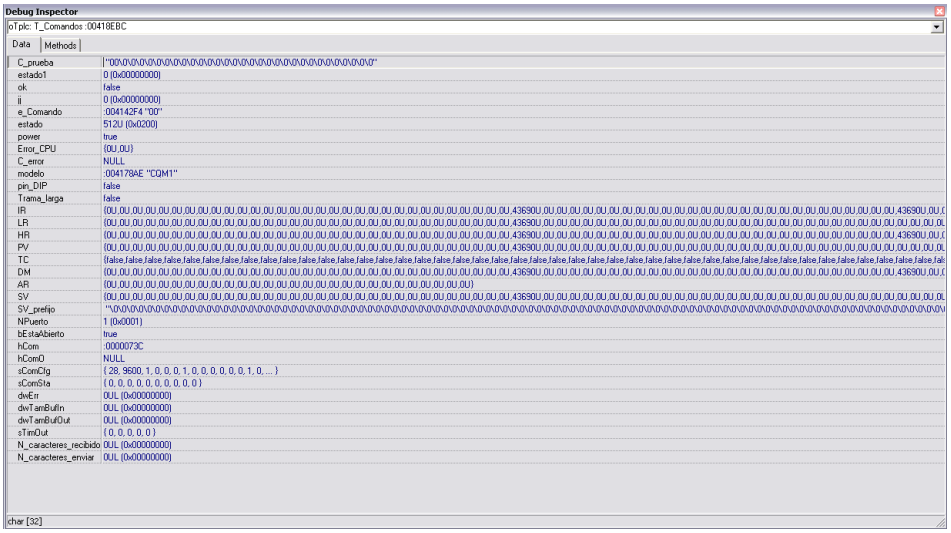
Esta función como he dicho con antelación es la encargada principal de gestionar correctamente los comandos (Envió - Transmisión).

Como los gestiona?

Esta función lo que espera es una cadena de caracteres en este caso los comandos, numero de caracteres recibidos y si la trama recibida es larga, con estos 3 parámetros la función lo que pasa a realizar es a buscar o comprobar que tipo de comando es, sabiendo que comando es, entra al bucle especifico de dicho comando y pasara a comprobar si el comando es correcto depurándose con los posibles errores que dicho comando pueda tener, esto lo realiza con la clase (T_Comandos) ya que esta clase pasa a utilizar las funciones creadas para comprobación de los errores por medio de la clase (T_Error), ya depurado los posibles errores y si hay alguno no hace nada que ya la aplicación se encargara de enviar la trama de respuesta correspondiente, sin embargo si todo sale con éxito se pasa a ejecutar la orden que le envía el comando, separando de las cadena de caracteres los canales o Nº de canales que hallan que leer o escribir y el dato si lo hubiera y en un correspondiente bucle se guardaría en su memoria correspondiente, devolviendo el formato o trama de respuesta que hay que enviar por medio de la aplicación.

5.2 Estructura de memoria del emulador

Para poder emular bien al plc real la idea que me surgió fue la de crearme el objeto (oTplc) mencionado anteriormente con todas las variables posibles que son las encargadas de simular la estructura de la memoria del plc como son las (IR, LR,SR,DM,TC, etc.), por que hago esto? Por la sencilla razón de que tengo que guardar los valores en los registros para futuras utilizaciones de los bits o palabras de un determinado canal en pocas palabras tengo que guardar la información. Como se puede observar en la imagen siguiente se ilustra todas las variables que emulan a casi todas las posiciones de memoria del plc.

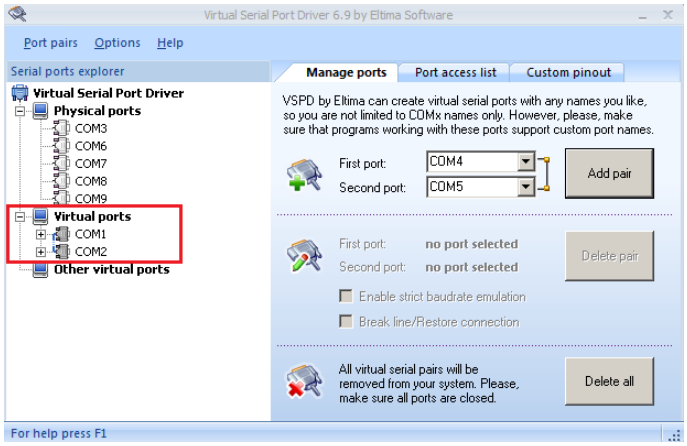


Por ejemplo si necesito ver el estado de la entrada IR 0.0 lo unique que tengo que hacer es utilizar el correspondiente método de lectura y leo el valor que hay en todo el canal y enmascaro para poder sacar como está el estado del IR 0.0 y si por lo contrario necesito cambiar el valor del IR 0.0 es igual lo único que hay que utilizar el método correspondiente de escribir, a partir de aquí ya todo es más sencillo escribir y leer posiciones de memoria, para poder obtener los datos necesarios a la hora de utilizarlos desde cualquier aplicación, claro siempre y cuando teniendo en cuenta más factores porque este proyecto está condicionado a muchas más condiciones.

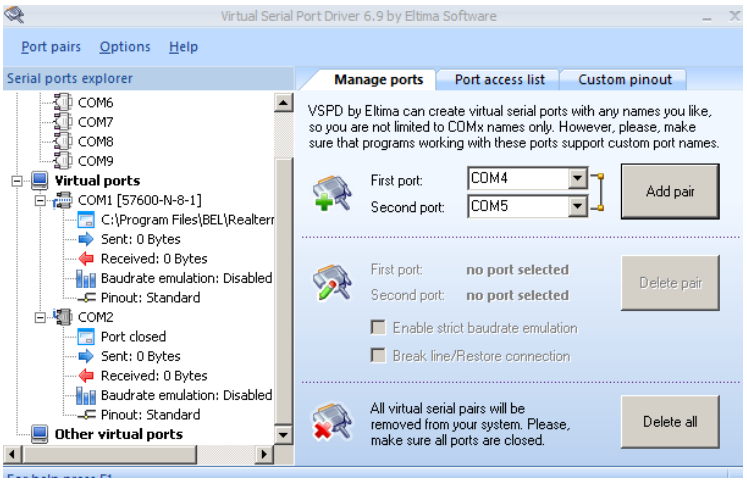
5.3 Como se realizaron todas las pruebas

Como he mencionada cree toda la estructura de memoria y métodos oportunos en un objeto a partir de aquí ya lo siguiente era ir depurando todos los posibles fallos, casos y funcionamiento correcto de los comandos HOSTLINK, tuve que utilizar 2 programas extras más encargadas de hacerme simulación de puerto serie virtuales y para poder enviar y recibir cadena de caracteres.

Tuve que simular puertos virtuales debido a que o en día los pc carecen de puerto serie real entonces hay que crearse virtuales para así poder establecer una correcta comunicación con dispositivos, el programa utilizado fue el **Configure Virtual Serial Port Drive**, es un programa muy sencillo de utilizar y la verdad que me sirvió de gran ayuda, como se puede observar en la imagen siguiente es muy fácil de usar y es solo crear par de puertos virtuales en este caso hay uno creado el COM1-COM2.

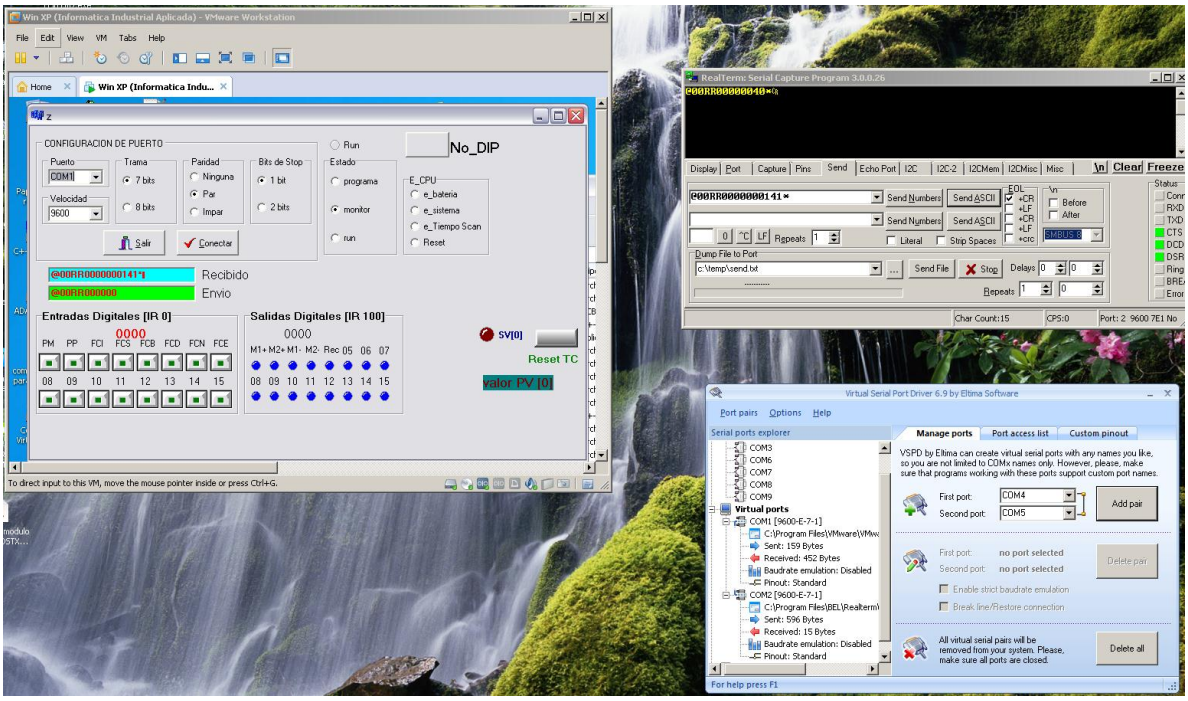


Se puede ver también cuando una aplicación abre un puerto, quien lo tiene abierto, como lo abre y se visualiza el envío y recepción de datos por dicho puerto, ver figura.



Aparte de lo dicho con anterioridad este programa también me facilita el envió de información desde la máquina virtual a la maquina real del pc y asu vez si tengo un convertidos USB→serie puedo enviar estos al exterior.

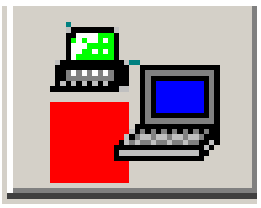
El otro programa utilizado es más conocido y fue mi gran herramienta de simulación de envío y recepción de comandos, ya que con este veo bien si mi emulador me respondía correctamente como deseaba, el programa es el **Realterm** por que fue mi gran herramienta, pues por que con este podía enviar los comandos oportunos que estaba depurando para su correcto funcionamiento y observa si mi programa del emulador respondía como yo deseaba, ya que el diseño de todos los métodos de las clases del emulador debía de ir uno a uno y simulando sus posibles errores, ya que esto no fue una tarea difícil pero si muy laboriosa, casi que podría decir que es lo fuerte de este proyecto.



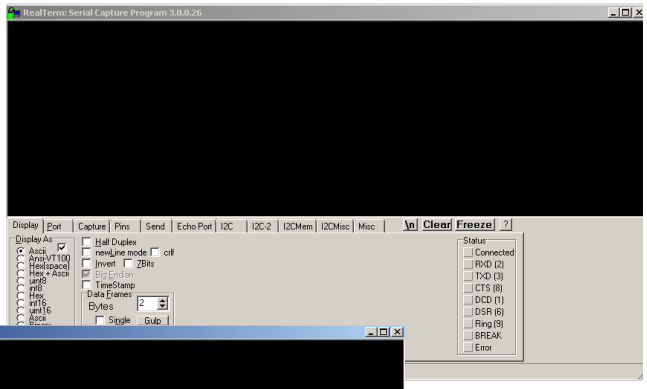
Como se observa en la figura anterior se ve como esta mmi emulador en la máquina virtual y recibe el dato enviado por parte del **realterm** y le devuelve la respuesta una vez verificado el comando y ejecutado, también se visualiza la comunicación que hay por medio del **Configure Virtual Serial Port Drive**, así tuve que estar haciendo para poder diseñar los métodos oportunos que tratarían o simularían a los comandos HOSTLINK, realizando la programación e ir depurando el programa paso a paso para analizar el correcto funcionamiento.

En la figura anterior lo que hace el **realterm** manda a leer un solo canal IR empezando por el 0 ósea solo lee ese canal, tiene que llevar su correspondiente detector de errores (FCS), ”*” y carácter retorno de carro << @00RR0000000141*/r >> todo esto está en la parte inferior del **realterm** en **sendASCIIS,** el emulador lo recibe como se observa en el cajetín de recibido, lo procesa, ejecuta y envía respuesta en cajetín de envío, enviando todo el comando respuestas de cómo está el canal **IR0** que luego se ve el valor recibido en el **realterm** en su pantalla principal en letras amarillas **<<** @00RR00000040*/r **>>**

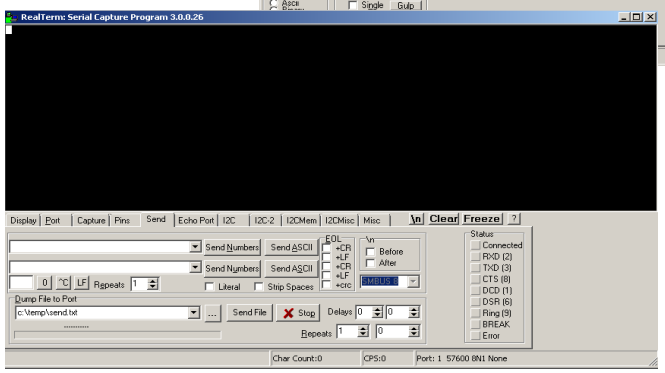
5.3.1 Realterm



RealTerm es una solución de software de terminal que permite a los administradores de red para capturar, gestionar o flujos binarios de depuración y otros datos. Su interfaz está destinado a ser lo más intuitivo posible, ya que cada función principal se detalla en su propia pestaña - por ejemplo, puede personalizar la pantalla entre varios tipos, tales como ASCII, ANSI, Hex, binario, INT8, ASCII + Hex y otros, cambiando así la forma en que los caracteres recibidos se visualizan en el terminal.



También puede modificar el puerto, la paridad y los bits de datos, por lo que para llegar a una configuración que mejor se adapte a sus necesidades.



RealTerm le permite especificar la ubicación del archivo TXT en la que se capturan los datos de entrada, con la posibilidad de insertar una marca de tiempo al comienzo de cada línea.

Las otras pestañas de la ventana principal de RealTerm se pueden usar para configurar el control de flujo por software, la conexión digital o de control de flujo de hardware.

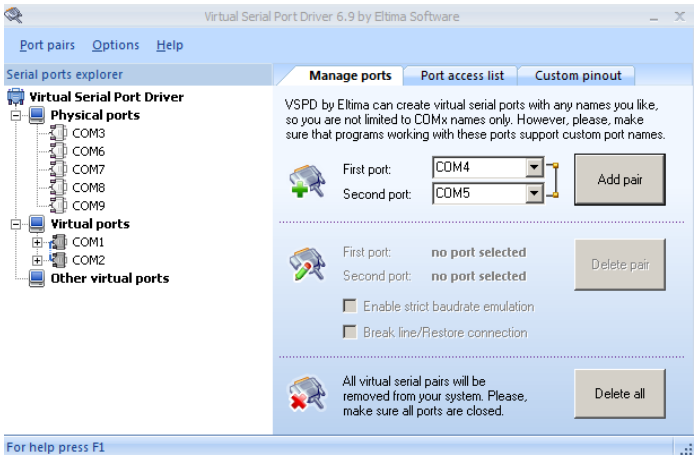
Uno de los beneficios de RealTerm es que viene con soporte para teclas de acceso rápido, que puede ayudar a ahorrar una gran cantidad de tiempo en que el manejo de los flujos de datos difíciles, lo que le permite centrarse más en el proceso de depuración que en el funcionamiento de la aplicación.

Con todo, RealTerm proporciona a los usuarios con numerosas funciones que les permiten depurar tipos binarios o de otro tipo de flujos de datos, sin embargo, se requieren habilidades avanzadas con el fin de aprovechar al máximo sus características.

Los novatos serán más probable ser abrumados por la gran cantidad de términos técnicos y las muchas configuraciones de la aplicación viene con, por lo que no van a apreciar su verdadero valor.

5.3.2 Configure Virtual Serial Port Driver

Es un programa que me permite crear puertos serie virtual, debido a la carencia de ellos en el ordenador, se pueden crear cualquier tipo de configuración de las líneas entre ellos, pero a mí lo único que me hacía falta era crearme dos puertos y que estuvieran puenteados entre ellos para poder simular la conexión del emulador y la aplicación que lo controlara.



6 ANEXOS

6.1 CODIGO

6.1.1 Emulador PLC

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include<math.h>
#include <time.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <windows.h>
#include "U_e1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "_GClass"
#pragma link "AbLED"
#pragma link "AbSwitch"
#pragma link "AbOpHour"
#pragma link "AbClock"
#pragma link "AbCBitBt"
#pragma resource "*.dfm"
#include "TAPICOM.H"


TForm1 *Form1;

T_Comandos oTplc("00") ;


DCB sComCfg1;
short DirBase;
bool comun_ok;
bool continuo=true;
int tiempo=0;
bool Nue_con= false;
unsigned short led_run,timer;
unsigned short ciclo = 0 ;//para simular los estados en el programa
de puente grua
bool marcha = 0; // simulacion de marcha en    puente grua

//----------Funciones------------
AnsiString calculo_FCS(AnsiString);
char envio_comando(AnsiString,bool);
char recepcion_comand(AnsiString *);
void activarE_Sdigitales(void);


bool compro( AnsiString *,DWORD,bool);
unsigned short estado;      //3 run, 2=monitor, 0=programa
unsigned short valor_memoria;//para guardar valor de la memoria
para simular un programa de automata
unsigned short valor_memoriaT,valor_memoriaTc;




//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
     estado=Estado->ItemIndex;
     estado=estado*2;
     if(estado>2) estado=3;          // 0 programa, 512 monitor, 768 run
     estado<<=8;
     oTplc.Set_estado(estado);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ConfigurarClick(TObject *Sender)
{
     BYTE bParidad;
     BYTE bBitStop;
     BYTE bTrama;

     switch (Paridad->ItemIndex){
       case 0: bParidad='N';
         break;
       case 1: bParidad='E';
         break;
       case 2: bParidad='O';
         break;
     }
     if (Bit_Stop->ItemIndex==0)  bBitStop=0;
     else bBitStop=2;

     if ( Trama->ItemIndex==0) bTrama=7;
     else bTrama=8;

     oTplc.AbrirOtroPuerto(Puerto->ItemIndex+1) ;
     sComCfg1= oTplc.ObtenerConfiguracion();


     if (oTplc.EstaAbierto()){
       oTplc.EstablecerConfiguracion(sComCfg1,1200*pow(2,Velocidad-
>ItemIndex), bParidad,bBitStop,bTrama);
                  //DCB sComCfg, DWORD dwVelocidad, BYTE bParidad, BYTE
bBitStop, BYTE bTrama){
     }
    else {
      ShowMessage("puerto no abierto" );
      Nue_con= true;
      Panel5->Visible=true;
    }
     Timer1->Enabled=true;

   //Activar las entradas y salidas digitales
      GroupBox1->Enabled=true;
      GroupBox2->Enabled=true;
    // Form1->Height=286;

}


//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
   int intentos=0,N_caracteres=0;
   led_run=oTplc.Get_estado()  ;
   led_run>>=8;
   char* car_recibido= new char[1];



  int i;
   char error,error1;
   bool ok_comando = false;
```

```
    bool trama_larga = true;
    AnsiString recibido,envio,aux_envio,E_digital,S_digital;
    DWORD N_caracteres_Recibidos;


     //------------recepcion de comandos---------------------------------

     do{
       if  ((led_run==3)&&  (Led_run->Checked==false))  Led_run->Checked=true;  //parpadeo del led de modo run
        else  Led_run->Checked=false;
        N_caracteres_Recibidos=oTplc.BytesEnBufferRx();
        error1= recepcion_comand(&recibido);
        intentos++;
        if(intentos>50) break;
       }while(error1==0);
       N_caracteres_Recibidos=oTplc.Get_N_caracteres_recibidos();
      if ((N_caracteres_Recibidos<128)||(recibido.SubString(N_caracteres_Recibidos-1,2)=="*\r")) trama_larga = false;      // si los caracteres recibidos son mas de 129 tiene que dividir la trama
      if(N_caracteres_Recibidos!=0)   Recibe->Text=recibido;

      IR_0->Caption=  E_digital.IntToHex(oTplc.Get_IR(0),4); //ver el valor de IR 0 para simulacion de programa    E/digitales
      IR_100->Caption=       S_digital.IntToHex(oTplc.Get_IR(100),4); //ver el valor de IR 0 para simulacion de programa  S/digitales

ok_comando=compro(&recibido,N_caracteres_Recibidos,trama_larga);  //pasa a comprobar todo y hace lo que tiene que hacer
      if (ok_comando){
        envio=recibido;
       if    (envio.SubString(6,2)=="11")    Programas->ItemIndex=0; //Activar temporizador de IOdigitales el Tim2
       if    (envio.SubString(6,2)=="12")    Programas->ItemIndex=1; //Activar temporizador de puente grua el Tim3
       if    (envio.SubString(6,2)=="13")    Programas->ItemIndex=2; //Activar temporizador de Temporizadores el Tim4
        N_caracteres=envio.Length();
       while(N_caracteres>131){  // si los caracteres a enviar son mas de 131 hay que dividir la respuesta
          aux_envio=envio.SubString(1,7);
          if              (envio.SubString(1,1)=="@"){              error= envio_comando(envio.SubString(1,127),true);
            envio=envio.SubString(128,N_caracteres);
          }
          else {
            error= envio_comando(envio.SubString(1,128),true);
            envio=envio.SubString(129,N_caracteres);
          }
          N_caracteres=envio.Length();
          oTplc.BorrarTxRx();

          do{
            error1=oTplc.RecibirCadena(car_recibido,131);
            oTplc.BorrarTxRx();
          }while (car_recibido[0]!='\r');
          if(N_caracteres<131){
            envio=envio.SubString(1,N_caracteres);
            error= envio_comando(envio,false);}

         Envio->Text=envio;
        }
        car_recibido=0;
        delete car_recibido;
        if((N_caracteres<131)&&(aux_envio=="")){
          if(trama_larga==true){
            error1=oTplc.TransmitirCadena("\r");
                        ok_comando=false;
            oTplc.Set_Trama_larga(trama_larga);
            do{
              N_caracteres_Recibidos=oTplc.BytesEnBufferRx()+5;        //se queda esperando la continuacion de la trama larga
              error1= recepcion_comand(&recibido);
              envio=envio.SubString(1,5);
              if (recibido.SubString(1,5)=="@00WR")  error1=0;
envio=envio+recibido;
              recibido=envio;
            }while(error1==0);
            if ((N_caracteres_Recibidos<128)||(recibido.SubString(N_caracteres_Recibidos-1,2)=="*\r")) trama_larga = false;
            ok_comando=compro(&recibido,N_caracteres_Recibidos,trama_larga);
            oTplc.Set_Trama_larga(trama_larga);
            if(ok_comando){
              envio=recibido;
              error= envio_comando(envio,false);
            }
            else{
              envio=recibido.SubString(1,5)+oTplc.Get_comError();
              error=envio_comando(envio,false);
            }
          }
          else error= envio_comando(envio,false);
        }
        Sleep(150);
        Envio->Text=envio;
      }
      else  if ((error1==1)&&(recibido!='\r')){    //envio error de formato si el comando malo e_14 y si no se recibe nada
          envio=recibido.SubString(1,5)+oTplc.Get_comError();
          N_caracteres=envio.Length();
          error=envio_comando(envio,false);
          Envio->Text=envio;
      }
    if(Programas->ItemIndex==2) Tipo_tim->Visible=true;
    else    Tipo_tim->Visible=false;
}
 //---------------------------------------------------------------------------
void __fastcall TForm1::EstadoClick(TObject *Sender)
{

     estado=Estado->ItemIndex;
     estado=estado*2;
     if(estado>2) estado=3;          // 0 programa, 512 monitor, 768 run
     estado<<=8;
     oTplc.Set_estado(estado);
 }
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
       activarE_Sdigitales();   //Activar Entradas y salidas digitales

    //------------pintar salidas digitales-------------------

     unsigned short salidas_100 =  oTplc.Get_IR(100);
      unsigned short auxiliar;
      auxiliar = salidas_100;
      if (salidas_100 & 1)    S_100_0->Checked=1;
      else S_100_0->Checked=0;
      if (salidas_100 & 2)    S_100_1->Checked=1;
       else S_100_1->Checked=0;
      if (salidas_100 & 4)    S_100_2->Checked=1;
       else S_100_2->Checked=0;
      if (salidas_100 & 8)    S_100_3->Checked=1;
       else S_100_3->Checked=0;
      if (salidas_100 & 16)    S_100_4->Checked=1;
       else S_100_4->Checked=0;
      if (salidas_100 & 32)    S_100_5->Checked=1;
       else S_100_5->Checked=0;
      if (salidas_100 & 64)    S_100_6->Checked=1;
       else S_100_6->Checked=0;
      if (salidas_100 & 128)    S_100_7->Checked=1;
       else S_100_7->Checked=0;
      if (salidas_100 & 256)    S_100_8->Checked=1;
       else S_100_8->Checked=0;
      if (salidas_100 & 512)    S_100_9->Checked=1;
       else S_100_9->Checked=0;
      if (salidas_100 & 1024)    S_100_10->Checked=1;
       else S_100_10->Checked=0;
      if (salidas_100 & 2048)    S_100_11->Checked=1;
                  else S_100_11->Checked=0;
                 if (salidas_100 & 4096)    S_100_12->Checked=1;
```

```
        else S_100_12->Checked=0;
       if (salidas_100 & 8192)    S_100_13->Checked=1;
        else S_100_13->Checked=0;
       if (salidas_100 & 16384)    S_100_14->Checked=1;
        else S_100_14->Checked=0;
       if (salidas_100 & 32768)    S_100_15->Checked=1;
        else S_100_15->Checked=0;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer3Timer(TObject *Sender)
{
    activarE_Sdigitales();  //Activar Entradas y salidas digitales
  //----------programam de puente grua ----------

    bool                pm=E_0->Checked,pp=E_1->Checked,fci=E_2-
>Checked,fcs=E_3->Checked;
    bool                fcb=E_4->Checked,fcd=E_5->Checked,fcn=E_6-
>Checked,fce=E_7->Checked;
    bool M1derecha = S_100_0->Checked, M2sube = S_100_1-
>Checked, M1izquierda = S_100_2->Checked;
    bool M2baja = S_100_3->Checked, rectificado = S_100_4-
>Checked;
    AnsiString S_digital;
    marcha = pm || marcha;     //marcha todo correcto
    marcha = (marcha) && !pp;
    if (pp==1) {oTplc.Set_IR(100,0);     //Apagar todas las salidas tras
pulsar PP
        S_100_0->Checked=0;
        S_100_1->Checked=0;
        S_100_2->Checked=0;
        S_100_3->Checked=0;
        S_100_4->Checked=0;
    }
    switch (ciclo){
      case 0: if((marcha == 1)&&(fci==1)&&(fcb==1) ){
              M2sube = 1;  //Activar el motor 2 +       //subir
              S_100_1->Checked= M2sube;
              oTplc.Set_IR(100,2);
              ciclo++;
             }

             break;
      case 1: if((marcha == 1)&&(fcs==1)&&(fcb==1) ){
              M2sube = 0;  //Desactivar el motor 2     // ir al FCD
              S_100_1->Checked= M2sube;
              M1derecha = 1;  //Activar el motor 1 +
              S_100_0->Checked = M1derecha;
              oTplc.Set_IR(100,1);
              ciclo++;
             }
             break;
      case 2: if((marcha == 1)&&(fcs==1)&&(fcd==1) ){
              M1derecha = 0;  //Desactivar el motor 1
              S_100_0->Checked = M1derecha;
              M2baja =1;                  //bajar
              S_100_3->Checked = M2baja;
              oTplc.Set_IR(100,8);
              ciclo++;
             }
             break;
      case 3: if((marcha == 1)&&(fci==1)&&(fcd==1) ){
              M2baja=0 ; //Desactivar el motor 2
              S_100_3->Checked  = M2baja;              //Desengrasado
espera 2s
              oTplc.Set_IR(100,0);

              ciclo++;
             }
             break;
       case 4: if((marcha == 1)&&(fci==1)&&(fcd==1) ){
              Sleep(2500);  // tiempo a esperar en desengrado 2s
              M2sube = 1;  //Activar el motor 2 +       //subir
S_100_1->Checked= M2sube;
oTplc.Set_IR(100,2);
ciclo++;
    }
    break;
case 5: if((marcha == 1)&&(fcs==1)&&(fcd==1) ){
      M2sube = 0;  //Desactivar el motor 2     // ir al FCD
      S_100_1->Checked= M2sube;
      M1derecha = 1;  //Activar el motor 1 +
      S_100_0->Checked = M1derecha;
      oTplc.Set_IR(100,1);
      ciclo++;
    }
    break;
case 6: if((marcha == 1)&&(fcs==1)&&(fcn==1) ){
      M1derecha = 0;  //Desactivar el motor 1
      S_100_0->Checked = M1derecha;
      M2baja =1;                  //bajar
      S_100_3->Checked = M2baja;
      oTplc.Set_IR(100,8);
      ciclo++;
    }
    break;
case 7: if((marcha == 1)&&(fci==1)&&(fcn==1) ){
      M2baja=0 ; //Desactivar el motor 2
      S_100_3->Checked = M2baja;        //Neutralizado 5s
      oTplc.Set_IR(100,0);
      ciclo++;
    }
    break;
case 8: if((marcha == 1)&&(fci==1)&&(fcn==1) ){
      Sleep(5500);  // tiempo a esperar por neutralizado 5s
      M2sube = 1;  //Activar el motor 2 +       //subir
      S_100_1->Checked= M2sube;
      oTplc.Set_IR(100,2);
      ciclo++;
    }
    break;
case 9: if((marcha == 1)&&(fcs==1)&&(fcn==1) ){
      M2sube = 0;  //Desactivar el motor 2     // ir al FCE
      S_100_1->Checked= M2sube;
      M1derecha = 1;  //Activar el motor 1 +
      S_100_0->Checked = M1derecha;
      oTplc.Set_IR(100,1);
      ciclo++;
    }
    break;
case 10: if((marcha == 1)&&(fcs==1)&&(fce==1) ){
      M1derecha = 0;  //Desactivar el motor 1
      S_100_0->Checked = M1derecha;
                M2baja =1;                  //bajar
                S_100_3->Checked = M2baja;
                oTplc.Set_IR(100,8);
      ciclo++;
    }
    break;
case 11: if((marcha == 1)&&(fci==1)&&(fce==1) ){
      M2baja=0 ; //Desactivar el motor 2
      S_100_3->Checked = M2baja;        //electrolisis 10s
      rectificado=1;     //Activar el rectificado
      S_100_4->Checked=rectificado;
      oTplc.Set_IR(100,16);
      ciclo++;

    }
    break;
case 12: if((marcha == 1)&&(fci==1)&&(fce==1) ){
      Sleep(10500);  // tiempo a esperar por electrolisis 10s
      rectificado=0;     //desactivar el rectificado
      S_100_4->Checked=rectificado;
      M2sube = 1;  //Activar el motor 2 +       //subir
      S_100_1->Checked= M2sube;
      oTplc.Set_IR(100,2);
      ciclo++;
    }
    break;
            case 13: if((marcha == 1)&&(fcs==1)&&(fce==1) ){
                  M2sube = 0;  //Desactivar el motor M2
                  S_100_1->Checked= M2sube;
                  M1izquierda = 1;
```

```
            S_100_2->Checked= M1izquierda;  // ir a izquierda
            oTplc.Set_IR(100,4);
            ciclo++;
           }
           break;
       case 14: if((marcha == 1)&&(fcs==1)&&(fcb==1) ){
            M1izquierda = 0;  //Desactivar el motor 1
            S_100_2->Checked = M1izquierda;
            M2baja =1;                  //bajar
            S_100_3->Checked = M2baja;
            oTplc.Set_IR(100,8);
            ciclo++;
           }
           break;

        case 15: if((marcha == 1)&&(fci==1)&&(fcb==1) ){
            M2baja =0;                  //Desactiar motor M2
            S_100_3->Checked = M2baja;
            oTplc.Set_IR(100,0);
            marcha = 0;
            ciclo = 0;
           }
           break;

   }
  IR_100->Caption= S_digital.IntToHex(oTplc.Get_IR(100),4);
            }
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer4Timer(TObject *Sender)
{
   unsigned char prefijo=oTplc.Get_SV_prefijo(0);  //obtener si TC0
es un temporizador o contador
    //------------Temporizador 0 -------------------------

    //------ a la conexion-----------------------------
  if((prefijo=='T')&&(E_8->Checked==true)&&(Tipo_tim-
>ItemIndex==0)){
   if((estado==768)||(estado==512)){
    oTplc.Set_IR(0,256);
    if (timer==0){
       valor_memoriaT=oTplc.Get_SV(0);      //precargar el valor de
SV en el PV al iniciar
       oTplc.Set_PV(0,valor_memoriaT);
       timer++;
    }
    if(valor_memoriaT>0) valor_memoriaT--;
    oTplc.Set_PV(0,valor_memoriaT);                  //Simulacion del
temporizador descontando
    Label5->Caption=oTplc.Get_PV(0);

    if(oTplc.Get_PV(0)==0){                          //AbLED1->Checked
^=true,timer=0;
      oTplc.Set_TC(0,true);   //activa el estado interno del TC0
      oTplc.Set_IR(100,1);   //Activar salida 100.0
      S_100_0->Checked=1;
      timer=0;
    }
   }
  }
   if((oTplc.Get_TC(0))&&(E_8->Checked==false)&&(Tipo_tim-
>ItemIndex==0)){
       oTplc.Set_TC(0,false);                  //para reiniciar el estado
interno del TC0
       oTplc.Set_IR(100,0);            //Desactivar la salida 100.0
       S_100_0->Checked=0;
       timer=0;
   }
   // ----------a la desconexion------
  if((prefijo=='T')&&(E_8->Checked==true)&&(Tipo_tim-
>ItemIndex==1)){
    oTplc.Set_IR(0,256);
    oTplc.Set_IR(100,1);
    S_100_0->Checked=1;
    if (timer==0){
       valor_memoriaT=oTplc.Get_SV(0);      //precargar el valor de
SV en el PV al iniciar
       oTplc.Set_PV(0,valor_memoriaT);
       timer++;
    }
    if(valor_memoriaT>0) valor_memoriaT--;
    oTplc.Set_PV(0,valor_memoriaT);                  //Simulacion del temporizador
descontando
    Label5->Caption=oTplc.Get_PV(0);

    if(oTplc.Get_PV(0)==0){
      oTplc.Set_TC(0,true);   //activa el estado interno del TC0
      oTplc.Set_IR(100,0);   //Activar salida 100.0
      S_100_0->Checked=0;
      Timer4->Enabled=false;
      timer=0;
    }

 }
 if((oTplc.Get_TC(0))&&(E_8->Checked==false)&&(Tipo_tim->ItemIndex==1)){
       oTplc.Set_TC(0,false);           //para reiniciar el estado interno del TC0
       oTplc.Set_IR(100,0);            //Desactivar la salida 100.0
       S_100_0->Checked=0;
       timer=0;
 }

 //------------contador 0 -------------------------
 if((prefijo=='C')&&(E_8->Checked==true)&&(Tipo_tim->ItemIndex==2)){

  if((estado==768)||(estado==512)){
    if (timer==0){
       valor_memoriaTc=oTplc.Get_SV(0);
       oTplc.Set_PV(0,valor_memoriaTc);
       timer++;
    }
    valor_memoriaTc--;
    E_8->Checked=false;
    oTplc.Set_PV(0,valor_memoriaTc);
    Label5->Caption=oTplc.Get_PV(0);
    if(oTplc.Get_TC(0)) {
       oTplc.Set_TC(0,false);   //para reiniciar el estado interno del TC0
       oTplc.Set_IR(100,0);            //Desactivar la salida 100.0
       S_100_0->Checked=0;
    }
    if(oTplc.Get_PV(0)==0){        //AbLED1->Checked ^=true,timer=0;
      oTplc.Set_TC(0,true);   //activa el estado interno del TC0
      oTplc.Set_IR(100,1);   //Activar salida 100.0
      S_100_0->Checked=1;
      timer=0;
    }
   }
  }
   if((oTplc.Get_eCPU(0)==0)&&(oTplc.Get_eCPU(0)==0))                E_CPU-
>ItemIndex=3;  //cambia el estado del seleccionador de errores
}
//---------------------------------------------------------------------------

void __fastcall TForm1::E_0MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=1;  // ver si la entrada 0.0 esta activada
    if(E_0->Checked==true) {
       aux |=1;    // Activa la entrada/salida con este OR
       oTplc.Set_IR(0,aux);
    }
    else {
       aux &=65534; // apaga la entrada/salida con este AND  y el valor FFFE
       oTplc.Set_IR(0,aux);
           }
}
                //---------------------------------------------------------------------------
                void    __fastcall    TForm1::E_1MouseDown(TObject    *Sender,
                TMouseButton Button,
                    TShiftState Shift, int X, int Y)
                {
                    unsigned short aux;
```

```
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=2;   // ver si la entrada 0.1 esta activada
    if(E_1->Checked==true) {
       aux |=2;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-2);//65533; // apaga la entrada/salida con este
AND  y el valor FFFE
       oTplc.Set_IR(0,aux);

    }
}
//---------------------------------------------------------------------------
void    __fastcall    TForm1::E_3MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
   unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=8;   // ver si la entrada 0.3 esta activada
    if(E_3->Checked==true) {
       aux |=8;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=65527; // apaga la entrada/salida con este AND  y el
valor FFFE
                    oTplc.Set_IR(0,aux);
                  }

             }
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_4MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=16;   // ver si la entrada 0.3 esta activada
    if(E_4->Checked==true) {
       aux |=16;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-16); // apaga la entrada/salida con este AND
y el valor FFFE
       oTplc.Set_IR(0,aux);
     }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_5MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=32;   // ver si la entrada 0.3 esta activada
    if(E_5->Checked==true) {
       aux |=32;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);
     }
    else {
       aux &=(65535-32); // apaga la entrada/salida con este AND
y el valor FFFE
       oTplc.Set_IR(0,aux);
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::E_6MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=64;   // ver si la entrada 0.3 esta activada
    if(E_6->Checked==true) {
       aux |=64;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-64); // apaga la entrada/salida con este AND  y el valor FFFE
       oTplc.Set_IR(0,aux);
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::E_7MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
   unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=128;   // ver si la entrada 0.3 esta activada
    if(E_7->Checked==true) {
       aux |=128;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-128); // apaga la entrada/salida con este AND  y el valor
FFFE
       oTplc.Set_IR(0,aux);
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::E_8MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=256;   // ver si la entrada 0.3 esta activada
    if(E_8->Checked==true) {
       aux |=256;     // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-256); // apaga la entrada/salida con este AND  y el valor
FFFE
       oTplc.Set_IR(0,aux);
     }

  if      ((E_8->Checked==false)&&(Programas->ItemIndex==2))          Timer4-
>Enabled=1;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::E_9MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
                    unsigned short aux;
                    valor_memoria=oTplc.Get_IR(0);
                    aux= valor_memoria;
                    valor_memoria &=512;   // ver si la entrada 0.3 esta activada
                    if(E_9->Checked==true) {
                       aux |=512;     // Activa la entrada/salida con este OR
                        oTplc.Set_IR(0,aux);
                     }
                    else {
                       aux &=(65535-512); // apaga la entrada/salida con este AND
```

```
y el valor FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_10MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=1024;  // ver si la entrada 0.3 esta activada
    if(E_10->Checked==true) {
       aux |=1024;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux  &=(65535-1024); // apaga la entrada/salida con este
AND  y el valor FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_11MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=2048;  // ver si la entrada 0.3 esta activada
    if(E_11->Checked==true) {
       aux |=2048;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux  &=(65535-2048); // apaga la entrada/salida con este
AND  y el valor FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_12MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=4096;  // ver si la entrada 0.3 esta activada
    if(E_12->Checked==true) {
       aux |=4096;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);
     }
    else {
       aux  &=(65535-4096); // apaga la entrada/salida con este
AND  y el valor FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_13MouseDown(TObject    *Sender,
TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=8192;  // ver si la entrada 0.3 esta activada
    if(E_13->Checked==true) {
       aux |=8192;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-8192); // apaga la entrada/salida con este AND  y el valor
FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_14MouseDown(TObject    *Sender,    TMouseButton
Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=16384;  // ver si la entrada 0.3 esta activada
    if(E_14->Checked==true) {
       aux |=16384;    // Activa la entrada/salida con este OR
                                  oTplc.Set_IR(0,aux);

                                }
    else {
       aux &=(65535-16384); // apaga la entrada/salida con este AND  y el valor
FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------

void    __fastcall    TForm1::E_15MouseDown(TObject    *Sender,    TMouseButton
Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=32768;  // ver si la entrada 0.3 esta activada
    if(E_15->Checked==true) {
       aux |=32768;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
    else {
       aux &=(65535-32768); // apaga la entrada/salida con este AND  y el valor
FFFE
        oTplc.Set_IR(0,aux);
      }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::E_2MouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    unsigned short aux;
    valor_memoria=oTplc.Get_IR(0);
    aux= valor_memoria;
    valor_memoria &=4;  // ver si la entrada 0.1 esta activada
    if(E_2->Checked==true) {
       aux |=4;    // Activa la entrada/salida con este OR
        oTplc.Set_IR(0,aux);

     }
                    else {
                       aux &=(65535-4);//65531; // apaga la entrada/salida con este
                AND  y el valor FFFE
                        oTplc.Set_IR(0,aux);
                      }
                }
                //---------------------------------------------------------------------------


                void __fastcall TForm1::Button1Click(TObject *Sender)
                {
                   bool dip ;
```

```
    dip^=oTplc.Get_DIP();
    oTplc.Set_DIP(dip);     // activar el DIP que es proteger memoria
de usuario
    if (dip==true) DIP->Caption="DIP";
    else DIP->Caption="NO_DIP";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::E_CPUClick(TObject *Sender)
{
     int aux = E_CPU->ItemIndex ;
     if (aux==0) oTplc.Set_eCPU(0,32);//se proboca un error en la
CPU y es el codigo con Error de bateria
     if (aux==1) oTplc.Set_eCPU(0,128);//se proboca un error en la
CPU y es el codigo con Error de sistema
     if(aux==2)  oTplc.Set_eCPU(1,16);//se proboca un error en la
CPU y es el codigo se excedio tiempo SCan
     if(aux==3)    oTplc.Set_eCPU(0,0),oTplc.Set_eCPU(1,0);//reset
de todos los posibles errores de CPU

}
//---------------------------------------------------------------------------

/*&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&*/

//----------------------        FUNCIONES      ------------------------------------
------
 AnsiString calculo_FCS(AnsiString cadena){

     char aux=0;
     int i;
     int j = cadena.Length();
     for (i=1;i<=j;i++){
         aux=aux^cadena[i];
      }
    return IntToHex(aux,2);
}

//----------------------Envio cadena de comandos al PLC-----------------
----------------
 char envio_comando(AnsiString envio,bool Trama_dividida){

     char error;
     AnsiString FCS;
     FCS = calculo_FCS(envio);//calculo_FCS(Ansistring aux)
     if (Trama_dividida==true) envio=envio+FCS+"\r";
     else envio=envio+FCS+"*\r";
     char acCad[133]="\0";
     char* pCad = new char [133];
     strcpy(acCad,envio.c_str());
     oTplc.TransmitirCadena(acCad);
     Sleep(35);    //Esperar respuesta del Adam

     delete pCad;
     return error;
 }

 //--------------------------Respuesta del PLC ------------------- -----------
-------
 char recepcion_comand(AnsiString *recibido){
    char error;
    char* pCad = new char [133];
    char R_comando[133]="\0";
    error=oTplc.RecibirCadena(pCad,133);    //32
    strcpy(R_comando, pCad);
    *recibido=R_comando;


  delete pCad;
  return error;
  }



 bool compro( AnsiString *comando,DWORD N_caracteres_re,bool Trama_larga){
  bool ok=false;
  AnsiString n_unidad,cabecera,trama,fcs,aux,cana_inicio,cana_final,dato;
  char* pCad1 = new char [133],pCad3[4];
  unsigned short* pCad2,aux_i,aux_f,Da_escr ;
  unsigned short canal_inicio,canal_final,N_canales,valor[1024];
  char R_comando[32]="\0";
  char C_final[2];
  unsigned long i,longitud,cociente,resto;
  trama=*comando;
  longitud=trama.Length();

//--------------------------COMPROBACION DE COMANDOS--------------------------
//*******************************************************************************
//**************************************** LECTURAS ******************************
  if (comando->SubString(1,5)=="@00RR") {        //Leer IR
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RR(pCad1);          //con un false en ok es que hay un error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();   // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
      if (aux=="00") {
        strcpy(pCad3,cana_inicio.c_str());
        //aux_i = oTplc.valor_decimal(pCad3);
        aux_i= cana_inicio.ToInt();
        aux_f=cana_final.ToInt();

        for(i=aux_i;i<(aux_i+aux_f);i++){
           valor[i]= oTplc.Get_IR(i);     //Leer el valor de IR
           aux=aux+aux.IntToHex(valor[i],4);
         }
        * comando=aux;
        *comando="@00RR"+*comando;
      }
   }
//*******************************************************************************
  else if(comando->SubString(1,5)=="@00RL") {     //Leer RL
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RL(pCad1);          //con un false en ok es que hay un error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
                cana_inicio= dato.SubString(1,4);
                canal_inicio=StrToInt(dato.SubString(1,4));
                cana_final= dato.SubString(5,4);
                canal_final= StrToInt(dato.SubString(5,4));
                pCad1=oTplc.Get_comError();         // Peticion de codigo de
              terminacion
                strcpy(C_final, pCad1);
                aux=C_final;
                  if (aux=="00") {
                  strcpy(pCad3,cana_inicio.c_str());
                  //aux_i = oTplc.valor_decimal(pCad3);
                  aux_i= cana_inicio.ToInt();
                  aux_f=cana_final.ToInt();
                  for(i=aux_i;i<(aux_i+aux_f);i++){
```

```
        valor[i]= oTplc.Get_LR(i);      //Leer el valor de LR
        aux=aux+aux.IntToHex(valor[i],4);
       }
      * comando=aux;
      *comando="@00RL"+*comando;
    }
  }
//*****************************************************************************
***
  else if(comando->SubString(1,5)=="@00RH") {    //Leer RH
         strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RH(pCad1);          //con un false en ok es que hay un
error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      //aux_i = oTplc.valor_decimal(pCad3);
      aux_i= cana_inicio.ToInt();
      aux_f=cana_final.ToInt();
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_HR(i);      //Leer el valor de HR
        aux=aux+aux.IntToHex(valor[i],4);
       }
      * comando=aux;
      *comando="@00RH"+*comando;
    }

  }
//*****************************************************************************
***
  else if (comando->SubString(1,5)=="@00RC") {    //Leer PV

    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RC(pCad1);          //con un false en ok es que hay un
error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      //aux_i = oTplc.valor_decimal(pCad3);
      aux_i= cana_inicio.ToInt();
      aux_f=cana_final.ToInt();
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_PV(i);      //Leer el valor de PV
        //aux=valor[i];
         aux=aux+aux.IntToHex(valor[i],4);
       }
      *comando=aux;
      *comando="@00RC"+*comando;
    }

  }
//*****************************************************************************
***
  else if (comando->SubString(1,5)=="@00RG") {//Leer TC (Estado
del temporizador)
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RG(pCad1);          //con un false en ok es que hay un
                error
                  if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      //aux_i = oTplc.valor_decimal(pCad3);
      aux_i= cana_inicio.ToInt();
      aux_f=cana_final.ToInt();
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_TC(i);      //Leer el valor de TC
        aux=aux+aux.IntToHex(valor[i],1);
       }
      * comando=aux;
      *comando="@00RG"+*comando;
    }

  }
//*****************************************************************************
  else if(comando->SubString(1,5)=="@00RD") {    //Leer DM
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RD(pCad1);          //con un false en ok es que hay un error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      //aux_i = oTplc.valor_decimal(pCad3);
      aux_i= cana_inicio.ToInt();
      aux_f=cana_final.ToInt();
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_DM(i);      //Leer el valor de DM
        aux=aux+aux.IntToHex(valor[i],4);
      }
       * comando=aux;
       *comando="@00RD"+*comando;
     }
  }

//*****************************************************************************
  else if (comando->SubString(1,5)=="@00RJ") {    //Leer AR
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.RJ(pCad1);          //con un false en ok es que hay un error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(1,4);
    canal_inicio=StrToInt(dato.SubString(1,4));
    cana_final= dato.SubString(5,4);
    canal_final= StrToInt(dato.SubString(5,4));
    pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      //aux_i = oTplc.valor_decimal(pCad3);
      aux_i= cana_inicio.ToInt();
      aux_f=cana_final.ToInt();
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_AR(i);      //Leer el valor de AR
        aux=aux+aux.IntToHex(valor[i],4);
       }
      * comando=aux;
```

```
        *comando="@00RJ"+*comando;

      }
    }
    //*******************************                                    ESCRITURA
    ************************************
    //*****************************************************************************
     else if (comando->SubString(1,5)=="@00WR") {    //Escribir IR
        if              ((Trama_larga==true)&&(trama.SubString(longitud-
    1,1)!="*")&&(!oTplc.Get_Trama_larga())){
          trama=trama.SubString(1,longitud-1);
          trama=trama+"*\r";
        }
        if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
            aux=canal_inicio;
            longitud=aux.Length();
            if (longitud==1) aux="000"+aux;
            if (longitud==2) aux="00"+aux;
            if (longitud==3) aux="0"+aux;
            if (longitud==4) aux=aux;
            trama.Insert(aux,6);
            aux=trama;
            longitud=trama.Length();
            trama=trama.SubString(1,longitud-4);
            trama= trama+calculo_FCS(trama);
            trama=trama+"*\r";
            oTplc.Set_Trama_larga(false);
         }


        strcpy(pCad1,trama.c_str());
        ok = oTplc.WR(pCad1);           //con un false en ok es que hay
    un error ;
        if (ok==false) goto error;
        strcpy(R_comando, pCad1);
        dato= R_comando;
        pCad1=oTplc.Get_comError();        // Peticion de codigo de
    terminacion
        strcpy(C_final, pCad1);
        aux=C_final;
        if (aux=="00") {
            canal_inicio=StrToInt(dato.SubString(1,4));
            cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
            dato= dato.SubString(5,dato.Length()-4);
            N_canales=dato.Length();
            N_canales=N_canales/4;
            aux=dato;
            int ii=0;;
            for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
               aux=dato.SubString((4*ii)+1,4);
               strcpy(pCad3,aux.c_str());
               Da_escr = oTplc.valor_decimal(pCad3);
               oTplc.Set_IR(canal_inicio+ii,  Da_escr )  ;
               ii++;
            }
           if (Trama_larga==true){
              canal_inicio=canal_inicio + N_canales;
              aux=aux.IntToHex(canal_inicio,4);
           }
            *comando="@00WR00";
           }

     }
    //*****************************************************************************
    ***
     else if (comando->SubString(1,5)=="@00WL") {    //Escribir LR
        if              ((Trama_larga==true)&&(trama.SubString(longitud-
    1,1)!="*")&&(!oTplc.Get_Trama_larga())){
          trama=trama.SubString(1,longitud-1);
          trama=trama+"*\r";
        }
        if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
                          aux=canal_inicio;
                          longitud=aux.Length();
                          if (longitud==1) aux="000"+aux;
                          if (longitud==2) aux="00"+aux;
    if (longitud==3) aux="0"+aux;
        if (longitud==4) aux=aux;
        trama.Insert(aux,6);
        aux=trama;
        longitud=trama.Length();
        trama=trama.SubString(1,longitud-4);
        trama= trama+calculo_FCS(trama);
        trama=trama+"*\r";
         oTplc.Set_Trama_larga(false);
     }
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.WL(pCad1);          //con un false en ok es que hay un error ;
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato= R_comando;
    pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
    if (aux=="00") {
        canal_inicio=StrToInt(dato.SubString(1,4));
        cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
        dato= dato.SubString(5,dato.Length()-4);
        N_canales=dato.Length();
        N_canales=N_canales/4;
        aux=dato;
        int ii=0;;
        for(i=canal_inicio;i<(canal_inicio+N_canales);i++){

           aux=dato.SubString((4*ii)+1,4);
           strcpy(pCad3,aux.c_str());
           Da_escr = oTplc.valor_decimal(pCad3);
           oTplc.Set_LR(canal_inicio+ii,  Da_escr )  ;
           ii++;
        }
       if (Trama_larga==true){
          canal_inicio=canal_inicio + N_canales;
          aux=aux.IntToHex(canal_inicio,4);
       }
        *comando="@00WL00";
       }
     }
    //*****************************************************************************
     else if (comando->SubString(1,5)=="@00WH") {    //Escribir HR
        if                          ((Trama_larga==true)&&(trama.SubString(longitud-
    1,1)!="*")&&(!oTplc.Get_Trama_larga())){
          trama=trama.SubString(1,longitud-1);
          trama=trama+"*\r";
        }
        if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
            aux=canal_inicio;
            longitud=aux.Length();
            if (longitud==1)aux="000"+aux;
            if (longitud==2)aux="00"+aux;
            if (longitud==3)aux="0"+aux;
            if (longitud==4) aux=aux;
            trama.Insert(aux,6);
            aux=trama;
            longitud=trama.Length();
            trama=trama.SubString(1,longitud-4);
            trama= trama+calculo_FCS(trama);
            trama=trama+"*\r";
            oTplc.Set_Trama_larga(false);
         }


        strcpy(pCad1,trama.c_str());
        ok = oTplc.WH(pCad1);          //con un false en ok es que hay un error ;
        if (ok==false) goto error;
        strcpy(R_comando, pCad1);
        dato= R_comando;
        pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
        strcpy(C_final, pCad1);
        aux=C_final;
        if (aux=="00") {
            canal_inicio=StrToInt(dato.SubString(1,4));
            cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
            dato= dato.SubString(5,dato.Length()-4);
            N_canales=dato.Length();
```

```
        N_canales=N_canales/4;
        aux=dato;
        int ii=0;;
        for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
           aux=dato.SubString((4*ii)+1,4);
           strcpy(pCad3,aux.c_str());
           Da_escr = oTplc.valor_decimal(pCad3);
           oTplc.Set_IR(canal_inicio+ii,  Da_escr )  ;
           ii++;
        }
       if (Trama_larga==true){
          canal_inicio=canal_inicio + N_canales;
          aux=aux.IntToHex(canal_inicio,4);
       }
         *comando="@00WH00";
      }

  }
//*****************************************************************************
***
  else if (comando->SubString(1,5)=="@00WC") {    //Escribir PV
     if              ((Trama_larga==true)&&(trama.SubString(longitud-
1,1)!="*")&&(!oTplc.Get_Trama_larga())){
       trama=trama.SubString(1,longitud-1);
       trama=trama+"*\r";
     }
     if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
         aux=canal_inicio;
        longitud=aux.Length();
        if (longitud==1)aux="000"+aux;
        if (longitud==2)aux="00"+aux;
        if (longitud==3)aux="0"+aux;
        if (longitud==4) aux=aux;
        trama.Insert(aux,6);
        aux=trama;
        longitud=trama.Length();
        trama=trama.SubString(1,longitud-4);
        trama= trama+calculo_FCS(trama);
        trama=trama+"*\r";
        oTplc.Set_Trama_larga(false);
     }

     strcpy(pCad1,trama.c_str());
     ok = oTplc.WC(pCad1);            //con un false en ok es que hay
un error ;
     if (ok==false) goto error;
     strcpy(R_comando, pCad1);
     dato= R_comando;
     pCad1=oTplc.Get_comError();        // Peticion de codigo de
terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
        canal_inicio=StrToInt(dato.SubString(1,4));
        cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
        dato= dato.SubString(5,dato.Length()-4);
        N_canales=dato.Length();
        N_canales=N_canales/4;
        aux=dato;
        int ii=0;;
        for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
           aux=dato.SubString((4*ii)+1,4);
           strcpy(pCad3,aux.c_str());
           Da_escr = oTplc.valor_decimal(pCad3);
           oTplc.Set_PV(canal_inicio+ii,  Da_escr )  ;
                       ii++;
                    }
                   if (Trama_larga==true){
                      canal_inicio=canal_inicio + N_canales;
          aux=aux.IntToHex(canal_inicio,4);
       }
         *comando="@00WC00";
      }

  }
//*****************************************************************************
***
  else if (comando->SubString(1,5)=="@00WG") {      //Escribir TC (Estado del
temporizador)
     AnsiString aux_WG;
     unsigned long  conta;
     if                          ((Trama_larga==true)&&(trama.SubString(longitud-
1,1)!="*")&&(!oTplc.Get_Trama_larga())){
       trama=trama.SubString(1,longitud-1);
       trama=trama+"*\r";
     }
     if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
         aux=canal_inicio;
        longitud=aux.Length();
        if (longitud==1)aux="000"+aux;
        if (longitud==2)aux="00"+aux;
        if (longitud==3)aux="0"+aux;
        if (longitud==4) aux=aux;
        trama.Insert(aux,6);
        aux=trama;
        longitud=trama.Length();
        trama=trama.SubString(1,longitud-4);
        trama= trama+calculo_FCS(trama);
        trama=trama+"*\r";
        oTplc.Set_Trama_larga(false);
     }
     longitud=trama.Length();
     aux_WG=trama.SubString(1,longitud-4);
     aux_WG=aux_WG.SubString(10,longitud);
     conta = aux_WG.Length();
     int g=1,gg=0;;
     do{
        aux_WG.Insert("000",g);
        g=g+4;
        gg++;
     }while(gg<conta);
     trama=trama.SubString(6,longitud);
     trama=trama.Delete(5,gg);
     longitud=trama.Length();
     aux_WG=aux_WG.Insert(trama.SubString(1,4),1);
     trama=trama.Delete(1,4);
     trama.Insert("@00WG"+aux_WG,1);
     strcpy(pCad1,trama.c_str());
     ok = oTplc.WG(pCad1);          //con un false en ok es que hay un error ;
     if (ok==false) goto error;
     strcpy(R_comando, pCad1);
     dato= R_comando;
     pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
        canal_inicio=StrToInt(dato.SubString(1,4));
        cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
        dato= dato.SubString(5,dato.Length()-4);
        N_canales=dato.Length();
        N_canales=N_canales/4;
        aux=dato;
        int ii=0;;
        for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
           aux=dato.SubString((4*ii)+1,4);
           strcpy(pCad3,aux.c_str());
           Da_escr = oTplc.valor_decimal(pCad3);
           oTplc.Set_TC(canal_inicio+ii,  Da_escr )  ;
           ii++;
        }
       if (Trama_larga==true){
                      canal_inicio=canal_inicio + N_canales;
                      aux=aux.IntToHex(canal_inicio,4);
                   }
         *comando="@00WG00";
      }
  }
//*****************************************************************************
  else if(comando->SubString(1,5)=="@00WD") {    //Escribir DM
     if                          ((Trama_larga==true)&&(trama.SubString(longitud-
1,1)!="*")&&(!oTplc.Get_Trama_larga())){
       trama=trama.SubString(1,longitud-1);
       trama=trama+"*\r";
     }
```

```
    if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
        aux=canal_inicio;
       longitud=aux.Length();
       if (longitud==1) aux="000"+aux;
       if (longitud==2) aux="00"+aux;
       if (longitud==3) aux="0"+aux;
       if (longitud==4) aux=aux;
       trama.Insert(aux,6);
       aux=trama;
       longitud=trama.Length();
       trama=trama.SubString(1,longitud-4);
       trama= trama+calculo_FCS(trama);
       trama=trama+"*\r";
        oTplc.Set_Trama_larga(false);
     }


    strcpy(pCad1,trama.c_str());
    ok = oTplc.WD(pCad1);          //con un false en ok es que hay un
error ;
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato= R_comando;
    pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
    if (aux=="00") {
       canal_inicio=StrToInt(dato.SubString(1,4));
       cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
       dato= dato.SubString(5,dato.Length()-4);
       N_canales=dato.Length();
       N_canales=N_canales/4;
       aux=dato;
       int ii=0;;
       for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
          aux=dato.SubString((4*ii)+1,4);
          strcpy(pCad3,aux.c_str());
          Da_escr = oTplc.valor_decimal(pCad3);
          oTplc.Set_DM(canal_inicio+ii,  Da_escr )  ;
          ii++;
       }
      if (Trama_larga==true){
         canal_inicio=canal_inicio + N_canales;
         aux=aux.IntToHex(canal_inicio,4);
      }
        *comando="@00WD00";
     }
  }
//*****************************************************************************
  else if (comando->SubString(1,5)=="@00WJ") {   //Escribir AR
     if                ((Trama_larga==true)&&(trama.SubString(longitud-
1,1)!="*")&&(!oTplc.Get_Trama_larga())){
       trama=trama.SubString(1,longitud-1);
       trama=trama+"*\r";
     }
     if((oTplc.Get_Trama_larga())&&(Trama_larga==false)){
        aux=canal_inicio;
       longitud=aux.Length();
                    if (longitud==1) aux="000"+aux;
                    if (longitud==2) aux="00"+aux;
                    if (longitud==3) aux="0"+aux;
                    if (longitud==4) aux=aux;
                    trama.Insert(aux,6);
       aux=trama;
       longitud=trama.Length();
       trama=trama.SubString(1,longitud-4);
       trama= trama+calculo_FCS(trama);
       trama=trama+"*\r";
        oTplc.Set_Trama_larga(false);
     }


    strcpy(pCad1,trama.c_str());
    ok = oTplc.WJ(pCad1);          //con un false en ok es que hay un
error ;
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato= R_comando;
    pCad1=oTplc.Get_comError();    // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
    if (aux=="00") {
       canal_inicio=StrToInt(dato.SubString(1,4));
       cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
       dato= dato.SubString(5,dato.Length()-4);
       N_canales=dato.Length();
       N_canales=N_canales/4;
       aux=dato;
       int ii=0;;
       for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
          aux=dato.SubString((4*ii)+1,4);
          strcpy(pCad3,aux.c_str());
          Da_escr = oTplc.valor_decimal(pCad3);
          oTplc.Set_IR(canal_inicio+ii,  Da_escr )  ;
          ii++;
       }
      if (Trama_larga==true){
         canal_inicio=canal_inicio + N_canales;
         aux=aux.IntToHex(canal_inicio,4);
      }
        *comando="@00WJ00";
     }
  }
//*****************************************************************************
  else if (comando->SubString(1,5)=="@00R#") {  //Leer SV
    strcpy(pCad1,comando->c_str())  ;
    ok = oTplc.Ralmu(pCad1);          //con un false en ok es que hay un error
    if (ok==false) goto error;
    strcpy(R_comando, pCad1);
    dato=R_comando;
    cana_inicio= dato.SubString(5,4);
    canal_inicio=StrToInt(dato.SubString(5,4));
    canal_final= 1;
    pCad1=oTplc.Get_comError();   // Peticion de codigo de terminacion
    strcpy(C_final, pCad1);
    aux=C_final;
     if (aux=="00") {
      strcpy(pCad3,cana_inicio.c_str());
      aux_i = oTplc.valor_decimal(pCad3);
      aux_f=canal_final;
      for(i=aux_i;i<(aux_i+aux_f);i++){
        valor[i]= oTplc.Get_SV(i); //Leer el valor seleccionado del  T/C
        aux=aux+valor[i];
       }
       * comando=aux;
       *comando="@00R#"+*comando;
    } }
//*****************************************************************************
  else if (comando->SubString(1,5)=="@00W#") {  //Escribir SV
     unsigned char prefijo[1];
     strcpy(pCad1,comando->c_str());
     ok = oTplc.Walmu(pCad1);          //con un false en ok es que hay un error ;
     if (ok==false) goto error;
     strcpy(R_comando, pCad1);
     dato= R_comando;
     pCad1=oTplc.Get_comError();   // Peticion de codigo de terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
        canal_inicio=StrToInt(dato.SubString(5,4));
        cana_inicio=cana_inicio.IntToHex(canal_inicio,4);
        dato= dato.SubString(9,dato.Length()-4);
        N_canales=1;
        int ii=0;;
        for(i=canal_inicio;i<(canal_inicio+N_canales);i++){
          aux=dato.SubString((4*ii)+1,4);
          Da_escr = StrToInt(aux.SubString(1,4));
          oTplc.Set_SV(canal_inicio+ii,  Da_escr )  ;
          ii++;
          if(comando->SubString(6,3)=="TIM")                    prefijo[0]='T',
oTplc.Set_SV_prefijo(canal_inicio,prefijo[0]);
          if(comando->SubString(6,3)=="CNT")                    prefijo[0]='C',
oTplc.Set_SV_prefijo(canal_inicio,prefijo[0]);
```

```
        }

        *comando="@00W#00";
      }
  }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00MS") {  //Leer estado
     strcpy(pCad1,comando->c_str())  ;
     ok = oTplc.MS(pCad1);          //con un false en ok es que hay un
error
     if (ok==false) goto error;
     pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
        unsigned short aux1=oTplc.Get_estado();
        aux1>>=8;
        if (aux1==0) aux= "0000" ;
        if (aux1==2) aux= "0200" ;
        if (aux1==3) aux= "0300" ;
        *comando=aux;
        *comando="@00MS"+*comando;
     }
  }
//*********************************************************************
  else  if  (comando->SubString(1,5)=="@00SC")  {      //Cambio de
estado
     strcpy(pCad1,comando->c_str())  ;
     ok = oTplc.SC(pCad1);          //con un false en ok es que hay un
error
     if (ok==false) goto error;
     dato=pCad1;
     pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
       if(dato=="00") oTplc.Set_estado(0);
       if(dato=="02") oTplc.Set_estado(512);
       if(dato=="03") oTplc.Set_estado(768);
     }
    *comando="@00SC00";
  }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00MF") {  //Lectura de error
     strcpy(pCad1,comando->c_str())  ;
     ok = oTplc.MF(pCad1);          //con un false en ok es que hay un
error
     if (ok==false) goto error;
     trama=pCad1;
     pCad1=oTplc.Get_comError();          // Peticion de codigo de
terminacion
     strcpy(C_final, pCad1);
     aux=C_final;
     if (aux=="00") {
        unsigned short aux1=oTplc.Get_eCPU(0);
        dato=IntToHex(aux1,2);
        aux1=oTplc.Get_eCPU(1);
        aux=IntToHex(aux1,2);
        aux.Insert(dato,5);
        if(trama=="01") oTplc.Set_eCPU(0,0), oTplc.Set_eCPU(1,0);
        *comando=aux;
        *comando="@00MF"+*comando;
     }
   }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00KS") {  //Forzado a set
     while(1) ; }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00KR") {  //Forzado a reset
     while(1) ; }
//*********************************************************************
***
  else  if  (comando->SubString(1,5)=="@00FK")  {        //Multiple
forzado set/reset
     while(1) ; }
//*********************************************************************
***
  else if (comando->SubString(1,5)=="@00KC") {      //cancelar el
forzado set/reset
     while(1) ; }
//*********************************************************************
***
  else if (comando->SubString(1,5)=="@00MM") {      //Lectura del
modelo PLC
     while(1) ;
     }
//*********************************************************************
***
  else if (comando->SubString(1,5)=="@00TS") {  //Prueba
   strcpy(pCad1,comando->c_str())  ;
  ok = oTplc.TS(pCad1);          //con un false en ok es que hay un error
  if (ok==false) goto error;
  strcpy(R_comando, pCad1);
  dato=R_comando;
  pCad1=oTplc.Get_comError();   // Peticion de codigo de terminacion
  strcpy(C_final, pCad1);
  aux=C_final;
   if (aux=="00") {
     aux=dato;
   }
   *comando=aux;
   *comando="@00TS"+*comando;
 }


//*********************************************************************
  else if (comando->SubString(1,5)=="@00QQ") {  //comando compuesto
     while(1) ; }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00XZ") {  //cancelar un comando
     while(1) ; }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00**") {  //Inicializar
                                    while(1) ; }
//*********************************************************************
  else if (comando->SubString(1,5)=="@00IC") {  //Indefinido
     while(1) ; }
//*********************************************************************
  else{
     oTplc.Set_ComError("14");
   }
  error:
  delete pCad1 ;

  return ok;
}
//---------------------------------------------------------------------------



void __fastcall TForm1::ProgramasClick(TObject *Sender)
{
      int programa=Programas->ItemIndex;
      switch (programa){
         case 0:   Timer2->Enabled=true;
                   Timer3->Enabled=false;
                   Timer4->Enabled=false;
                   ED0->Caption="00";
                   ED01->Caption="01";
                   ED02->Caption="02";
                   ED03->Caption="03";
                   ED04->Caption="04";
                   ED05->Caption="05";
                   ED06->Caption="06";
                   ED07->Caption="07";
                   ED08->Caption="08" ;
                   SD0->Caption="00";
                   SD1->Caption="01";
                   SD2->Caption="02";
                   SD3->Caption="03";
```

```
            SD4->Caption="04";
            break;
        case 1:  Timer2->Enabled=false;
            Timer3->Enabled=true;
            Timer4->Enabled=false;
            ED0->Caption="PM";
            ED01->Caption="PP";
            ED02->Caption="FCI";
            ED03->Caption="FCS";
            ED04->Caption="FCB";
            ED05->Caption="FCD";
            ED06->Caption="FCN";
            ED07->Caption="FCE";
            ED08->Caption="08" ;
            SD0->Caption="M1+";
            SD1->Caption="M2+";
            SD2->Caption="M1-";
            SD3->Caption="M2-";
            SD4->Caption="rect";


            break;
        case 2:  Timer2->Enabled=false;
            Timer3->Enabled=false;
            Timer4->Enabled=true;
            ED0->Caption="00";
            ED01->Caption="01";
            ED02->Caption="02";
            ED03->Caption="03";
            ED04->Caption="04";
            ED05->Caption="05";
            ED06->Caption="06";
            ED07->Caption="07";
            ED08->Caption="T/C" ;
            SD0->Caption="00";
            SD1->Caption="01";
            SD2->Caption="02";
            SD3->Caption="03";
            SD4->Caption="04";
            break;

    }

}
//---------------------------------------------------------------------------
 void activarE_Sdigitales(void){
      //-------ACtivar los interruptores segun las entradas ----------
   estado=oTplc.Get_estado();
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=1;   // ver si la entrada 0.0 esta activada
   if ( valor_memoria==1) {
      Form1->E_0->Checked=true;
     }
   else { Form1->E_0->Checked=false;
     }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=2;   // ver si la entrada 0.1 esta activada
   if ( valor_memoria==2) {
        Form1->E_1->Checked=true;}
   else {
        Form1->E_1->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=4;   // ver si la entrada 0.2 esta activada
   if ( valor_memoria==4) {
        Form1->E_2->Checked=true;}
   else {
        Form1->E_2->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=8;   // ver si la entrada 0.3 esta activada
   if ( valor_memoria==8) {
        Form1->E_3->Checked=true;}
   else {
        Form1->E_3->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=16;   // ver si la entrada 0.4 esta activada
   if ( valor_memoria==16) {
        Form1->E_4->Checked=true;}
   else {
        Form1->E_4->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=32;   // ver si la entrada 0.5 esta activada
   if ( valor_memoria==32) {
        Form1->E_5->Checked=true;}
   else {
        Form1->E_5->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=64;   // ver si la entrada 0.6 esta activada
   if ( valor_memoria==64) {
        Form1->E_6->Checked=true;}
   else {
        Form1->E_6->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=128;   // ver si la entrada 0.7 esta activada
   if ( valor_memoria==128) {
       Form1-> E_7->Checked=true;}
   else {
        Form1->E_7->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=256;   // ver si la entrada 0.8 esta activada
   if ( valor_memoria==256) {
        Form1->E_8->Checked=true;}
   else {
        Form1->E_8->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=512;   // ver si la entrada 0.9 esta activada
   if ( valor_memoria==512) {
        Form1->E_9->Checked=true;}
   else {
        Form1->E_9->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=1024;   // ver si la entrada 0.10 esta activada
   if ( valor_memoria==1024) {
        Form1->E_10->Checked=true;}
   else {
        Form1->E_10->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=2048;   // ver si la entrada 0.11 esta activada
   if ( valor_memoria==2048) {
        Form1->E_11->Checked=true;}
   else {
        Form1->E_11->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=4096;   // ver si la entrada 0.12 esta activada
   if ( valor_memoria==4096) {
        Form1->E_12->Checked=true;}
   else {
        Form1->E_12->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=8192;   // ver si la entrada 0.13 esta activada
   if ( valor_memoria==8192) {
        Form1->E_13->Checked=true;}
   else {
        Form1->E_13->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=16384;   // ver si la entrada 0.14 esta activada
   if ( valor_memoria==16384) {
        Form1->E_14->Checked=true;}
   else {
        Form1->E_14->Checked=false; }
   valor_memoria=oTplc.Get_IR(0);
   valor_memoria &=32768;   // ver si la entrada 0.15 esta activada
   if ( valor_memoria==32768) {
        Form1->E_15->Checked=true;}
   else {
        Form1->E_15->Checked=false; }
}
```

6.1.2 Supervisor y control de Entradas y salidas

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include<math.h>
#include "p1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "_GClass"
#pragma link "AbLED"
#pragma resource "*.dfm"
#include <Windows.h>

#include "TAPICOM.H"

TForm1 *Form1;

T_API_COM oTcom(1);
DCB sComCfg1;
short DirBase;
bool comun_ok;
bool continuo=true;
int tiempo=0,tiempo1_DM=0;
bool Nue_con= false;
int
contador_DM=0,contador_WDM=0,filas=1,Wfilas=1,DM_aver
=0,IR_aver=0;

TShape *oLED; // nos representa cualquier led de el bit
correspondiete



//----------Funciones----------------------

AnsiString calculo_FCS(AnsiString);
char envio_comando(AnsiString,bool);
char recepcion_comand(AnsiString *);
char inicializar_DM();
char inicializar_IR();
void pintaled(AnsiString,TShape *oLED1, int);
int conv(char );
int valor_decimal(AnsiString) ;
 unsigned short valor_decimal_otra(char* );


//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
  : TForm(Owner)
{
    BYTE bParidad;
    BYTE bBitStop;
    BYTE bTrama;
    switch (Paridad->ItemIndex){
      case 0: bParidad='N';
        break;
      case 1: bParidad='E';
        break;
      case 2: bParidad='O';
        break;
    }
    if (Bit_Stop->ItemIndex==0)  bBitStop=0;
    else bBitStop=2;

    if ( Trama->ItemIndex==0) bTrama=7;
    else bTrama=8;

    oTcom.AbrirOtroPuerto(Puerto->ItemIndex+1) ;
    sComCfg1= oTcom.ObtenerConfiguracion();

    if (oTcom.EstaAbierto()){
      oTcom.EstablecerConfiguracion(sComCfg1,1200*pow(2,Velocidad-
>ItemIndex), bParidad,bBitStop,bTrama);
                //DCB sComCfg, DWORD dwVelocidad, BYTE bParidad,
BYTE bBitStop, BYTE bTrama){
    }
    else {
      ShowMessage("puerto no abierto" );
      Nue_con= true;
      Panel5->Visible=true;
    }
       //"Horizontales"
    Rejilla->Cells[0][1] = "DM0000";
    Rejilla->Cells[0][2] = "DM0010";
    Rejilla->Cells[0][3] = "DM0020";
    Rejilla->Cells[0][4] = "DM0030";
    Rejilla->Cells[0][5] = "DM0040";
    Rejilla->Cells[0][6] = "DM0050";
    Rejilla->Cells[0][7] = "DM0060";
    Rejilla->Cells[0][8] = "DM0070";
    Rejilla->Cells[0][9] = "DM0080";
    Rejilla->Cells[0][10]= "DM0090";

    //"Verticales"
    Rejilla->Cells[1][0] = "      +0";
    Rejilla->Cells[2][0] = "      +1";
    Rejilla->Cells[3][0] = "      +2";
    Rejilla->Cells[4][0] = "      +3";
    Rejilla->Cells[5][0] = "      +4";
    Rejilla->Cells[6][0] = "      +5";
    Rejilla->Cells[7][0] = "      +6";
    Rejilla->Cells[8][0] = "      +7";
    Rejilla->Cells[9][0] = "      +8";
    Rejilla->Cells[10][0] = "     +9";


    //"Horizontales"
    R_TC->Cells[0][1] = "TC0000";
    R_TC->Cells[0][2] = "TC0010";
    R_TC->Cells[0][3] = "TC0020";
    R_TC->Cells[0][4] = "TC0030";
    R_TC->Cells[0][5] = "TC0040";
    R_TC->Cells[0][6] = "TC0050";
    R_TC->Cells[0][7] = "TC0060";
    R_TC->Cells[0][8] = "TC0070";
    R_TC->Cells[0][9] = "TC0080";
    R_TC->Cells[0][10]= "TC0090";

    //"Verticales"
    R_TC->Cells[1][0] = "      +0";
    R_TC->Cells[2][0] = "      +1";
    R_TC->Cells[3][0] = "      +2";
    R_TC->Cells[4][0] = "      +3";
    R_TC->Cells[5][0] = "      +4";
    R_TC->Cells[6][0] = "      +5";
    R_TC->Cells[7][0] = "      +6";
    R_TC->Cells[8][0] = "      +7";
    R_TC->Cells[9][0] = "      +8";
    R_TC->Cells[10][0] = "     +9";


    //"Horizontales"
    R_IR->Cells[0][1] = "IR0000";
    R_IR->Cells[0][2] = "IR0010";
    R_IR->Cells[0][3] = "IR0020";
    R_IR->Cells[0][4] = "IR0030";
    R_IR->Cells[0][5] = "IR0040";
    R_IR->Cells[0][6] = "IR0050";
    R_IR->Cells[0][7] = "IR0060";
```

```
    R_IR->Cells[0][8] = "IR0070";
    R_IR->Cells[0][9] = "IR0080";
    R_IR->Cells[0][10]= "IR0090";
    R_IR->Cells[0][11]= "IR0100";

    //"Verticales"
    R_IR->Cells[1][0] = "      +0";
    R_IR->Cells[2][0] = "      +1";
    R_IR->Cells[3][0] = "      +2";
    R_IR->Cells[4][0] = "      +3";
    R_IR->Cells[5][0] = "      +4";
    R_IR->Cells[6][0] = "      +5";
    R_IR->Cells[7][0] = "      +6";
    R_IR->Cells[8][0] = "      +7";
    R_IR->Cells[9][0] = "      +8";
    R_IR->Cells[10][0] = "     +9";



}

//---------------------------------------------------------------------------

 void __fastcall TForm1::FormCreate(TObject *Sender)
{

    char error,error1,error2,error3,error4;;
    int intentos=7;
    //AnsiString envio=Edit3->Text;
    AnsiString envio= "@00TS11";      //comando de prueba
para la comunicacion
                  // "@00WR00000001" // escribe un "1"  en
entrada 0.00 del automata
    AnsiString recibido;
    do{
      error2= envio_comando(envio,false);
      Sleep(200);
        //recepcion-----

       error3= recepcion_comand(&recibido);
       if (error3==0) break;
       intentos++;
     }while(intentos==10);
    if     ((recibido     ==     "@00TS1147*\r")&&(error3==0)){
//compruebo si llega la respuesta
       Form1->Edit4->Text=recibido;          //esperada segun
el comando enviado

       comun_ok=true;
       //Puerto->ItemIndex =Puerto->ItemIndex;
       Velocidad->ItemIndex =3;
       Panel5->Visible=true;
       char inici_DM = inicializar_DM();
       char inici_IR= inicializar_IR();
       Sleep(1000);
       ShowMessage("Leyendo DMs,TCs, IRs" );
       for(int g =0 ;g<10;g++)LeerDMs->Click();
       for(int g =0 ;g<10;g++)LeerTC->Click();
       for(int g =0 ;g<11;g++)LeerIR->Click();
       Panel5->Enabled=false;
    }
     if (comun_ok!=true) {
       ShowMessage("NO   hAY   COMUNICACION   CON
PLC");
        Panel5->Visible=true;
      }

}
//---------------------------------------------------------------------------
--------
void __fastcall TForm1::EnviarClick(TObject *Sender)
{
   char error,error1,error2,error3;
   AnsiString envio=Edit3->Text;
    Form1->Timer1->Enabled= false;
   int g=envio.Length();
   AnsiString recibido;
   short intentos =100;
   if (g<125) Tramalarga->Checked=false;
   if(envio=="") envio='\r';
   if(Tramalarga->Checked==false)
     error2= envio_comando(envio,false);
   else   error2= envio_comando(envio,true);

      Sleep(200);
         //recepcion-----
    //  do{
        error3= recepcion_comand(&recibido);
        Form1->Edit4->Text=recibido;
        intentos++;
     // } while ((error3!=0)&&(intentos<100));


}
//---------------------------------------------------------------------------
void __fastcall TForm1::SalirClick(TObject *Sender)
{
  exit(0);
}
//---------------------------------------------------------------------------


void __fastcall TForm1::ConfigurarClick(TObject *Sender)
{
    BYTE bParidad;
    BYTE bBitStop;
    BYTE bTrama;
    switch (Paridad->ItemIndex){
      case 0: bParidad='N';
        break;
      case 1: bParidad='E';
        break;
      case 2: bParidad='O';
        break;
    }
    if (Bit_Stop->ItemIndex==0)  bBitStop=0;
    else bBitStop=2;

    if ( Trama->ItemIndex==0) bTrama=7;
    else bTrama=8;

    oTcom.AbrirOtroPuerto(Puerto->ItemIndex+1) ;
    sComCfg1= oTcom.ObtenerConfiguracion();


    if (oTcom.EstaAbierto()){

oTcom.EstablecerConfiguracion(sComCfg1,1200*pow(2,Velocidad-
>ItemIndex), bParidad,bBitStop,bTrama);
                  //DCB   sComCfg,   DWORD   dwVelocidad,   BYTE
bParidad, BYTE bBitStop, BYTE bTrama){
       char error,error1,error2,error3,error4;;
    int intentos=7;

    AnsiString envio=  "@00TS11";       //comando de prueba para la
comunicacion
                  // "@00WR00000001" // escribe un "1"  en entrada 0.00
del automata
    AnsiString recibido;
    do{
      error2= envio_comando(envio,false);
      Sleep(200);
        //recepcion-----

       error3= recepcion_comand(&recibido);
       if (error3==0) break;
       intentos++;
     }while(intentos==10);
    if ((recibido == "@00TS1147*\r")&&(error3==0)){        //compruebo
si llega la respuesta
       Form1->Edit4->Text=recibido;                  //esperada  segun  el
comando enviado
```

```
        //Form1->Edit4->Text='\0';          //para limpiar el edit 4
        comun_ok=true;


        Velocidad->ItemIndex =3;
         char inici_DM = inicializar_DM();
         char inici_IR= inicializar_IR();
         Sleep(1000);
         ShowMessage("Leyendo DMs,TCs, IRs" );
         for(int g =0 ;g<10;g++)LeerDMs->Click();
         for(int g =0 ;g<10;g++)LeerTC->Click();
         for(int g =0 ;g<11;g++)LeerIR->Click();
         Panel5->Enabled=false;

    }
     if (comun_ok!=true) {
       ShowMessage("NO   hAY   COMUNICACION   CON
PLC");
        Panel5->Visible=true;
      }
    }
  else {
    ShowMessage("puerto no abierto" );
    Nue_con= true;
    Panel5->Visible=true;
  }



}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{

  if((comun_ok==true)&&(continuo==true)){
  int bcd1;
  unsigned int intentos =10000;
   char error,error1,error2,error3,error4,error5;
   unsigned int iPos=0,iPos1=0;
     TShape *oLED;
      AnsiString
inicial,n_canales,leer_DM,recibido,tiempo1,LED,cadena,cade
na1;
     AnsiString entr_di,recibido_1,recibido_Odig,recibido_11;
     char recibi[4],recibi_1[4];
     char *aux,*aux1;
     aux=&recibi[0];
     aux1=&recibi_1[0];



 //-----Leer entradas digitales y ir coloando su valor BCD en la
salida analogica 1
     n_canales="0001";
     entr_di="@00RR0000"+ n_canales;
    error2= envio_comando(entr_di,false);
    Sleep(200);
    if(error2!=0)
      error3= recepcion_comand(&recibido_1);

    if (error3==0){
      recibido_1=recibido_1.SubString(8,4);
       strcpy(recibi_1,recibido_1.c_str());
      LED="In_";
         for(int ii=0; ii<4;ii++){        //para cambiar caracter a
valor en entero
           aux1=&recibi_1[ii] ;
           iPos1=iPos1+ conv(*aux1)*pow(16,3-ii);      //valor a
escribir en decimal

          }
        cadena1=iPos1;
        l_dec->Caption=cadena1;
        l_Hex->Caption=recibido_1;
        pintaled(LED,oLED,iPos1);
```

```
        //----------escribir la salida analogica 1-------------

//

   unsigned int iPos11=0;
   char recibix[2];
   int recibexx[2];
   recibido_11=recibido_1.SubString(3,2);
   strcpy(recibix,recibido_11.c_str());
   for(int ii=0; ii<2;ii++){
            aux1=&recibix[ii];
           iPos11=iPos11+conv(*aux1)*pow(10,ii);    //valor a escribir del
caracter convertido a entero tal cual

          }

    int iposx;
    double bcd =iPos11/10.0;
   //----------numeros positivos-----------------

   if((iPos1<=153)&&(iPos11<100)){
      recibido_11=bcd;
      Out_A1->Position=iPos11;
      O_A1->Caption=recibido_11+"v";
      bcd =((bcd*2047)/10);
      iposx=bcd;
      recibido_11 = IntToHex(iposx,4) ;
      recibido_11="@00WR0101"+recibido_11;

      error2= envio_comando(recibido_11,false);
      Sleep(200);
      error3= recepcion_comand(&recibido_11);


    }
    //----------numeros negativos---------------
   else if((iPos1>=32768)&&(iPos11<100)){
      iposx=bcd;
      recibido_11= bcd;//~iposx;
      recibido_11.Insert("-",1);
      Out_A1->Position=~iPos11;
      O_A1->Caption=recibido_11+"v";
      bcd =((bcd*2047)/10);
      iposx=bcd;
      iposx=~iposx;
      iposx++;
      recibido_11 = IntToHex(iposx,4) ;
      recibido_11="@00WR0101"+recibido_11;
      error2= envio_comando(recibido_11,false);
      Sleep(200);
      error3= recepcion_comand(&recibido_11);

   }

   else if (iPos11>100) {
      recibido_11="0";
      Out_A1->Position=0;
      O_A1->Caption=recibido_11+"v";
      error3= recepcion_comand(&recibido_11);
      }
   //
              }



             }
            }
```

```
//--------tiempo de 0.5 segundos para ir leyendo las entradas
analogicas----

//-----------------Boton de Stop ciclo continuo----------------------
---------------------------------
  void __fastcall TForm1::StopClick(TObject *Sender)
{
                    continuo = false;
    Form1->Timer1->Enabled= false;
     Leer_EA->Checked=false;
}
//----------------boton de continual ciclo continuo-----------------
---------------------------------------

void __fastcall TForm1::ContinuoClick(TObject *Sender)
{
     continuo=true;
     Form1->Timer1->Enabled= true;

     Leer_EA->Checked=false;
}
//-------------       Barra salida Analogica 2-----------------------
-------------------------------

void __fastcall TForm1::Out_A2Change(TObject *Sender)
{
     Form1->Timer1->Enabled= false;

     char error,error1;
     int intentos =10000,iPosOA_22=0;
     double iPosOA_2;
     AnsiString B_OA2,recibido;
     iPosOA_2 = Out_A2->Position;
     iPosOA_2= iPosOA_2/10;
     B_OA2= iPosOA_2;
     iPosOA_22=iPosOA_2;
     iPosOA_22=(iPosOA_22*2047)/10;
     O_A2->Text =B_OA2+ "v" ;

     B_OA2=IntToHex(iPosOA_22,4);

     if(iPosOA_22<0) {
          B_OA2=B_OA2.SubString(5,4);
     }

     B_OA2="@00WR0102"+B_OA2;

     error= envio_comando(B_OA2,false);
     Sleep(200);
     error1= recepcion_comand(&recibido);

      Form1->Timer1->Enabled= true;

}
//---------------------------------------------------------------------



void __fastcall TForm1::EscribirClick(TObject *Sender)
{
     AnsiString TC;

     if  (Tipo_TC->ItemIndex==0)    TC="@00W#TIM  "+Edit1-
>Text+Edit2->Text;
     if  (Tipo_TC->ItemIndex==1)    TC="@00W#CNT  "+Edit1-
>Text+Edit2->Text;
                    Form1->Edit3->Text=TC;
                    Enviar->Click();

               }
               //-----------------------------------------------------------
----------

void __fastcall TForm1::LeerClick(TObject *Sender)
{
     AnsiString TC,TC1=Edit1->Text;
     TC="@00RC"+TC1+"0001";
     Form1->Edit3->Text=TC;
     Enviar->Click();
     TC=Form1->Edit4->Text;
     Edit5->Text=TC.SubString(8,4);
                    }
//---------------------------------------------------------------------



void __fastcall TForm1::LeerDMsClick(TObject *Sender)
{                    //Lee de 10 en 10 DM
     char  error2,error3;
     AnsiString envio,recibido;
     AnsiString DM;
     DM=contador_DM;
     if(contador_DM<10) DM="0"+DM;
    DM = "@00RD00" + DM + "0010" ;//Leer 10 canales de DM
     envio=DM;
     if(envio=="") envio='\r';

     short intentos =0;
     error2= envio_comando(envio,false );
     Sleep(300);
     //recepcion-----
     do{
        error3= recepcion_comand(&recibido);
        intentos++;
     } while ((error3!=0)&&(intentos<100));
     if(contador_DM<100){
       DM=recibido.SubString(8,recibido.Length());
       DM=DM.SubString(1,DM.Length()-4);
       Rejilla->Cells[1][filas] = DM.SubString(1,4);
       Rejilla->Cells[2][filas] = DM.SubString(5,4);
       Rejilla->Cells[3][filas] = DM.SubString(9,4);
       Rejilla->Cells[4][filas] = DM.SubString(13,4);
       Rejilla->Cells[5][filas] = DM.SubString(17,4);
       Rejilla->Cells[6][filas] = DM.SubString(21,4);
       Rejilla->Cells[7][filas] = DM.SubString(25,4);
       Rejilla->Cells[8][filas] = DM.SubString(29,4);
       Rejilla->Cells[9][filas] = DM.SubString(33,4);
       Rejilla->Cells[10][filas] = DM.SubString(37,4);
     }
     contador_DM=contador_DM+10;
     switch(contador_DM) {
        case 10:R2->Checked=true;
                R1->Checked=false;
                break;
        case 20:R3->Checked=true;
                R2->Checked=false;
                break;
        case 30:R4->Checked=true;
                R3->Checked=false;
                break;
        case 40:R5->Checked=true;
                R4->Checked=false;
                break;
        case 50:R6->Checked=true;
                R5->Checked=false;
                break;
        case 60:R7->Checked=true;
                R6->Checked=false;
                break;
        case 70:R8->Checked=true;
                R7->Checked=false;
                break;
        case 80:R9->Checked=true;
                R8->Checked=false;
                break;
        case 90:R10->Checked=true;
                R9->Checked=false;
                break;
```

```
        case 100:R1->Checked=true;
                 R10->Checked=false;
                 break;
      }

      filas++;
      if (filas ==11) filas =1;
      if (contador_DM ==100) contador_DM=0;
}
//---------------------------------------------------------------------------



void __fastcall TForm1::EscribirDMsClick(TObject *Sender)
{

      char  error2;
      int columna=1,canalDM=0;
      AnsiString envio,recibido,WDM;
      WDM=contador_WDM;
      if(contador_WDM<10) WDM="0"+WDM;
      WDM = "@00WD00" + WDM;//Escribir 10 canales de DM
      envio.Insert(Rejilla->Cells[1][Wfilas],1);
      envio.Insert(Rejilla->Cells[2][Wfilas],5);
      envio.Insert(Rejilla->Cells[3][Wfilas],9);
      envio.Insert(Rejilla->Cells[4][Wfilas],13);
      envio.Insert(Rejilla->Cells[5][Wfilas],17);
      envio.Insert(Rejilla->Cells[6][Wfilas],21);
      envio.Insert(Rejilla->Cells[7][Wfilas],25);
      envio.Insert(Rejilla->Cells[8][Wfilas],29);
      envio.Insert(Rejilla->Cells[9][Wfilas],33);
      envio.Insert(Rejilla->Cells[10][Wfilas],37);
      envio.Insert(WDM,1);
      error2= envio_comando(envio,false);
      Sleep(300);
      error2= recepcion_comand(&recibido);

      contador_WDM=contador_WDM+10;
      switch(contador_WDM) {
         case 10:W2->Checked=true;
                 W1->Checked=false;
                 break;
         case 20:W3->Checked=true;
                 W2->Checked=false;
                 break;
         case 30:W4->Checked=true;
                 W3->Checked=false;
                 break;
         case 40:W5->Checked=true;
                 W4->Checked=false;
                 break;
         case 50:W6->Checked=true;
                 W5->Checked=false;
                 break;
         case 60:W7->Checked=true;
                 W6->Checked=false;
                 break;
         case 70:W8->Checked=true;
                 W7->Checked=false;
                 break;
         case 80:W9->Checked=true;
                 W8->Checked=false;
                 break;
         case 90:W10->Checked=true;
                 W9->Checked=false;
                 break;
         case 100:W1->Checked=true;
                  W10->Checked=false;
                  break;
      }
      Wfilas++;
      if (Wfilas ==11) Wfilas =1;
      if (contador_WDM ==100) contador_WDM=0;

}
```

```
//---------------------------------------------------------------------------



void __fastcall TForm1::RejillaClick(TObject *Sender)
{
    //Escribir en la etiqueta el contenido de la celda
  AnsiString asContenido;
  asContenido = Rejilla->Cells[Rejilla->Col][Rejilla->Row];
}
//---------------------------------------------------------------------------



void __fastcall TForm1::LeerTCClick(TObject *Sender)
{
    char  error2,error3;              //Leer 10 TC del PV
      AnsiString envio,recibido;
      AnsiString DM;
      DM=contador_DM;
      if(contador_DM<10) DM="0"+DM;
     DM = "@00RC00" + DM + "0010" ;//Leer 10 canales de DM
      envio=DM;
      if(envio=="") envio='\r';

      short intentos =0;
      error2= envio_comando(envio,false);
      Sleep(300);
      //recepcion-----
      do{
         error3= recepcion_comand(&recibido);
         intentos++;
      } while ((error3!=0)&&(intentos<100));
      if(contador_DM<100){
        DM=recibido.SubString(8,recibido.Length());
        DM=DM.SubString(1,DM.Length()-4);
        R_TC->Cells[1][filas] = DM.SubString(1,4);
        R_TC->Cells[2][filas] = DM.SubString(5,4);
        R_TC->Cells[3][filas] = DM.SubString(9,4);
        R_TC->Cells[4][filas] = DM.SubString(13,4);
        R_TC->Cells[5][filas] = DM.SubString(17,4);
        R_TC->Cells[6][filas] = DM.SubString(21,4);
        R_TC->Cells[7][filas] = DM.SubString(25,4);
        R_TC->Cells[8][filas] = DM.SubString(29,4);
        R_TC->Cells[9][filas] = DM.SubString(33,4);
        R_TC->Cells[10][filas] = DM.SubString(37,4);
      }
      contador_DM=contador_DM+10;
      switch(contador_DM) {
         case 10:RT2->Checked=true;
                 RT1->Checked=false;
                 break;
         case 20:RT3->Checked=true;
                 RT2->Checked=false;
                 break;
         case 30:RT4->Checked=true;
                 RT3->Checked=false;
                 break;
         case 40:RT5->Checked=true;
                 RT4->Checked=false;
                 break;
         case 50:RT6->Checked=true;
                 RT5->Checked=false;
                 break;
         case 60:RT7->Checked=true;
                 RT6->Checked=false;
                 break;
         case 70:RT8->Checked=true;
                 RT7->Checked=false;
```

```
            break;
        case 80:RT9->Checked=true;
            RT8->Checked=false;
            break;
        case 90:RT10->Checked=true;
            RT9->Checked=false;
            break;
        case 100:RT1->Checked=true;
            RT10->Checked=false;
            break;
    }

    filas++;
    if (filas ==11) filas =1;
    if (contador_DM ==100) contador_DM=0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::paginasChange(TObject *Sender)
{
    AnsiString envioTC,recibidoTC;
    char  errorTC;
    contador_DM=0,contador_WDM=0,filas=1,Wfilas=1;
    Leer_EA->Checked=false;
    // --- inicializar los leds------
    //---- DMs---------------------
    R1->Checked=1;
    R2->Checked=0;
    R3->Checked=0;
    R4->Checked=0;
    R5->Checked=0;
    R6->Checked=0;
    R7->Checked=0;
    R8->Checked=0;
    R9->Checked=0;
    R10->Checked=0;
    W1->Checked=1;
    W2->Checked=0;
    W3->Checked=0;
    W4->Checked=0;
    W5->Checked=0;
    W6->Checked=0;
    W7->Checked=0;
    W8->Checked=0;
    W9->Checked=0;
    W10->Checked=0;
    //---- TCs---------------------
    RT1->Checked=1;
    RT2->Checked=0;
    RT3->Checked=0;
    RT4->Checked=0;
    RT5->Checked=0;
    RT6->Checked=0;
    RT7->Checked=0;
    RT8->Checked=0;
    RT9->Checked=0;
    RT10->Checked=0;
    WT1->Checked=1;
    WT2->Checked=0;
    WT3->Checked=0;
    WT4->Checked=0;
    WT5->Checked=0;
    WT6->Checked=0;
    WT7->Checked=0;
    WT8->Checked=0;
    WT9->Checked=0;
    WT10->Checked=0;
    //---- IRs---------------------
    RI1->Checked=1;
    RI2->Checked=0;
    RI3->Checked=0;
    RI4->Checked=0;
    RI5->Checked=0;
    RI6->Checked=0;
    RI7->Checked=0;
    RI8->Checked=0;
    RI9->Checked=0;
    RI10->Checked=0;
    WI1->Checked=1;
    WI2->Checked=0;
    WI3->Checked=0;
    WI4->Checked=0;
    WI5->Checked=0;
    WI6->Checked=0;
    WI7->Checked=0;
    WI8->Checked=0;
    WI9->Checked=0;
    WI10->Checked=0;
    oTcom.BorrarTxRx();
    //-------- activar el Tim4 de los tamporizadores en el PLC ----------
    if (paginas->TabIndex==2) {
        envioTC="@00TS13";
        errorTC = envio_comando(envioTC,false);
        Sleep(150);
        errorTC= recepcion_comand(&recibidoTC);
    }
    //----Activar el  Tim1 de los temporizadores de entradas y salidas
    if          ((paginas->TabIndex==0)||(paginas->TabIndex==1)||(paginas-
>TabIndex==3)) {
        envioTC="@00TS11";
        errorTC = envio_comando(envioTC,false);
        Sleep(150);
        errorTC= recepcion_comand(&recibidoTC);
    }


}
//---------------------------------------------------------------------------




void __fastcall TForm1::EscribirTCClick(TObject *Sender)
{
    char  error2;
    int columna=1,canalDM=0;
    AnsiString envio,recibido,WDM;
    WDM=contador_WDM;
    if(contador_WDM<10) WDM="0"+WDM;
    WDM = "@00WC00" + WDM;//Escribir 10 canales de TC   EL VALOR
PV
    envio.Insert(R_TC->Cells[1][Wfilas],1);
    envio.Insert(R_TC->Cells[2][Wfilas],5);
    envio.Insert(R_TC->Cells[3][Wfilas],9);
    envio.Insert(R_TC->Cells[4][Wfilas],13);
    envio.Insert(R_TC->Cells[5][Wfilas],17);
    envio.Insert(R_TC->Cells[6][Wfilas],21);
    envio.Insert(R_TC->Cells[7][Wfilas],25);
    envio.Insert(R_TC->Cells[8][Wfilas],29);
    envio.Insert(R_TC->Cells[9][Wfilas],33);
    envio.Insert(R_TC->Cells[10][Wfilas],37);
    envio.Insert(WDM,1);
    error2= envio_comando(envio,false);
    Sleep(300);
    error2= recepcion_comand(&recibido);

    contador_WDM=contador_WDM+10;
    switch(contador_WDM) {
        case 10:WT2->Checked=true;
            WT1->Checked=false;
            break;
        case 20:WT3->Checked=true;
            WT2->Checked=false;
            break;
        case 30:WT4->Checked=true;
            WT3->Checked=false;
            break;
        case 40:WT5->Checked=true;
            WT4->Checked=false;
            break;
        case 50:WT6->Checked=true;
            WT5->Checked=false;
```

```
            break;
        case 60:WT7->Checked=true;
            WT6->Checked=false;
            break;
        case 70:WT8->Checked=true;
            WT7->Checked=false;
            break;
        case 80:WT9->Checked=true;
            WT8->Checked=false;
            break;
        case 90:WT10->Checked=true;
            WT9->Checked=false;
            break;
        case 100:WT1->Checked=true;
            WT10->Checked=false;
            break;
    }
    Wfilas++;
    if (Wfilas ==11) Wfilas =1;
    if (contador_WDM ==100) contador_WDM=0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::LeerIRClick(TObject *Sender)
{
    char  error2,error3;            //Leer 10 IR del
    AnsiString envio,recibido;
    AnsiString DM;
    DM=contador_DM;
    if(contador_DM<10) DM="0"+DM;
    DM = "@00RR00" + DM + "0010" ;//Leer 10 canales de
DM
    if(contador_DM>99) DM.Delete(6,1);
    envio=DM;
    if(envio=="") envio='\r';

    short intentos =0;
    error2= envio_comando(envio,false);
    Sleep(300);
    //recepcion-----
    do{
       error3= recepcion_comand(&recibido);
       intentos++;
    } while ((error3!=0)&&(intentos<100));
    if(contador_DM<110){
      DM=recibido.SubString(8,recibido.Length());
      DM=DM.SubString(1,DM.Length()-4);
      R_IR->Cells[1][filas] = DM.SubString(1,4);
      R_IR->Cells[2][filas] = DM.SubString(5,4);
      R_IR->Cells[3][filas] = DM.SubString(9,4);
      R_IR->Cells[4][filas] = DM.SubString(13,4);
      R_IR->Cells[5][filas] = DM.SubString(17,4);
      R_IR->Cells[6][filas] = DM.SubString(21,4);
      R_IR->Cells[7][filas] = DM.SubString(25,4);
      R_IR->Cells[8][filas] = DM.SubString(29,4);
      R_IR->Cells[9][filas] = DM.SubString(33,4);
      R_IR->Cells[10][filas] = DM.SubString(37,4);

    }
    contador_DM=contador_DM+10;
    switch(contador_DM) {
       case 10:RI2->Checked=true;
            RI1->Checked=false;
            break;
       case 20:RI3->Checked=true;
            RI2->Checked=false;
            break;
       case 30:RI4->Checked=true;
            RI3->Checked=false;
            break;
       case 40:RI5->Checked=true;
            RI4->Checked=false;
            break;
       case 50:RI6->Checked=true;
            RI5->Checked=false;
            break;
       case 60:RI7->Checked=true;
            RI6->Checked=false;
            break;
       case 70:RI8->Checked=true;
            RI7->Checked=false;
            break;
       case 80:RI9->Checked=true;
            RI8->Checked=false;
            break;
       case 90:RI10->Checked=true;
            RI9->Checked=false;
            break;
       case 100:RI11->Checked=true;
            RI10->Checked=false;
            break;
       case 110:RI1->Checked=true;
            RI11->Checked=false;
            break;

    }

    filas++;
    if (filas ==12) filas =1;
    if (contador_DM ==110) contador_DM=0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EscribirIRClick(TObject *Sender)
{
     char  error2;
    int columna=1,canalDM=0;
    AnsiString envio,recibido,WDM;
    WDM=contador_WDM;
    if(contador_WDM<10) WDM="0"+WDM;
    WDM = "@00WR00" + WDM;//Escribir 10 canales de TC
    if(contador_WDM>99) WDM.Delete(6,1);
    envio.Insert(R_IR->Cells[1][Wfilas],1);
    envio.Insert(R_IR->Cells[2][Wfilas],5);
    envio.Insert(R_IR->Cells[3][Wfilas],9);
    envio.Insert(R_IR->Cells[4][Wfilas],13);
    envio.Insert(R_IR->Cells[5][Wfilas],17);
    envio.Insert(R_IR->Cells[6][Wfilas],21);
    envio.Insert(R_IR->Cells[7][Wfilas],25);
    envio.Insert(R_IR->Cells[8][Wfilas],29);
    envio.Insert(R_IR->Cells[9][Wfilas],33);
    envio.Insert(R_IR->Cells[10][Wfilas],37);
    envio.Insert(WDM,1);
    error2= envio_comando(envio,false);
    Sleep(300);
    error2= recepcion_comand(&recibido);

    contador_WDM=contador_WDM+10;
    switch(contador_WDM) {
       case 10:WI2->Checked=true;
            WI1->Checked=false;
            break;
       case 20:WI3->Checked=true;
            WI2->Checked=false;
            break;
       case 30:WI4->Checked=true;
            WI3->Checked=false;
            break;
       case 40:WI5->Checked=true;
            WI4->Checked=false;
            break;
       case 50:WI6->Checked=true;
            WI5->Checked=false;
            break;
       case 60:WI7->Checked=true;
            WI6->Checked=false;
            break;
       case 70:WI8->Checked=true;
            WI7->Checked=false;
            break;
       case 80:WI9->Checked=true;
            WI8->Checked=false;
```

```
                break;
            case 90:WI10->Checked=true;
                    WI9->Checked=false;
                    break;
            case 100:WI11->Checked=true;
                     WI10->Checked=false;
                     break;
            case 110:WI1->Checked=true;
                     WI11->Checked=false;
                     break;
          }
        Wfilas++;
        if (Wfilas ==12) Wfilas =1;
        if (contador_WDM ==110) contador_WDM=0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::R_TCClick(TObject *Sender)
{
          int columna= R_TC->Col;
          int fila=R_TC->Row;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit3Click(TObject *Sender)
{
            Form1->Timer1->Enabled= false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Leer_EAClick(TObject *Sender)
{
      continuo=false;
      Form1->Timer1->Enabled= false;

     unsigned int iPos11=0,iPos22=0,iPos33=0,iPos44=0;
     double V_an1=0,V_an2=0,V_an3=0,V_an4=0;

    unsigned int intentos =10000;
    char error,error1,error2,error3,error4,error5,error6,error7;
    AnsiString
In_Anal1,recibido_InAna1,In_Anal2,recibido_InAna2,
          In_Anal3,recibido_InAna3,In_Anal4,recibido_InAna4;
    In_Anal1="@00RR00010001";
    In_Anal2="@00RR00020001";
    In_Anal3="@00RR00030001";
    In_Anal4="@00RR00040001";

    error= envio_comando(In_Anal1,false);
    Sleep(200);
    error1= recepcion_comand(&recibido_InAna1);

    error2= envio_comando(In_Anal2,false);
    Sleep(200);
    error3= recepcion_comand(&recibido_InAna2);

    error4= envio_comando(In_Anal3,false);
    Sleep(200);
    error5= recepcion_comand(&recibido_InAna3);

    error6= envio_comando(In_Anal4,false);
    Sleep(200);
    error7= recepcion_comand(&recibido_InAna4);

     iPos11= valor_decimal(recibido_InAna1);
      V_an1=iPos11;
      if(V_an1>4048) V_an1=4048;              // los valores van
desde 30h=48d  hasta FD0h=4048d
      if(V_an1<48) V_an1=48;
      V_an1=((V_an1-48)*10)/4048;
      recibido_InAna1=V_an1;
      Inanl_1->Caption= recibido_InAna1.SubString(1,4)+"v";
      V_an1=V_an1*10;
ProgressBar1->Position= V_an1;

    iPos22= valor_decimal(recibido_InAna2);
     V_an2=iPos22;
     if(V_an2>4048) V_an2=4048;              // los valores van desde 30h=48d
hasta FD0h=4048d
     if(V_an2<48) V_an2=48;
     V_an2=((V_an2-48)*10)/4048;
     recibido_InAna2=V_an2;
     Inanl_2->Caption= recibido_InAna2.SubString(1,4)+"v";
     V_an2=V_an2*10;
     ProgressBar2->Position= V_an2;



    iPos33= valor_decimal(recibido_InAna3);
     V_an3=iPos33;
     if(V_an3>4048) V_an3=4048;              // los valores van desde 30h=48d
hasta FD0h=4048d
     if(V_an3<48) V_an3=48;
     V_an3=((V_an3-48)*10)/4048;
     recibido_InAna3=V_an3;
     Inanl_3->Caption= recibido_InAna3.SubString(1,4)+"v";
     V_an3=V_an3*10;
     ProgressBar3->Position= V_an3;



     iPos44= valor_decimal(recibido_InAna4);
     V_an4=iPos44;
     if(V_an4>4048) V_an4=4048;              // los valores van desde 30h=48d
hasta FD0h=4048d
     if(V_an4<48) V_an4=48;
     V_an4=((V_an4-48)*10)/4048;
     recibido_InAna4=V_an4;
     Inanl_4->Caption= recibido_InAna4.SubString(1,4)+"v";
     V_an4=V_an4*10;
     ProgressBar4->Position= V_an4;
     Leer_EA->Checked=false;


}
//---------------------------------------------------------------------------


void __fastcall TForm1::Leer_DMClick(TObject *Sender)
{
                    //--------leer Dms y ir pintandolas en los leds salidas
digitales----------

  char error,error1,error4,error5;
   unsigned int iPos=0,iPos1=0;
    TShape *oLED1;
     AnsiString
inicial,n_canales,leer_DM,recibido,tiempo1,LED,cadena,cadena1,recibido_O
dig;
     char recibi[4],recibi_1[4];
     char *aux,*aux1;
     aux=&recibi[0];
     aux1=&recibi_1[0];
    continuo=false;
    Form1->Timer1->Enabled= false;
     n_canales="0001";
     //inicial=IntToHex(tiempo,4);
     if(DM_aver<10){
          inicial="000";
          inicial=inicial+DM_aver;
     }
     else {
        inicial="00";
        inicial=inicial+DM_aver;
     }
     leer_DM="@00RD" + inicial + n_canales;
       error= envio_comando(leer_DM,false);
        Sleep(200);
```

```
        error1= recepcion_comand(&recibido);
        Sleep(200);
        LED="Out_";
        recibido=recibido.SubString(8,4);          //saco el dato
recibido del plc del Ansistring
        strcpy(recibi,recibido.c_str());//copiar a cadena de
caracteres los caracteres del stri

        for(int i=0; i<4;i++){
           aux=&recibi[i] ;
           iPos=iPos+  conv(*aux)*pow(16,3-i);         //valor a
escribir en decimal
             }
        cadena=iPos;
        O_Dec->Caption=cadena;
        O_Hex->Caption=recibido;
        pintaled(LED,oLED1,iPos);
        recibido="@00WR0100"+recibido;
        error4= envio_comando(recibido,false);
        Sleep(200);
        error5= recepcion_comand(&recibido_Odig);
        Sleep(200);


        Leer_DM->Checked=false;



}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit6Change(TObject *Sender)
{
          AnsiString DM = Edit6->Text;
          if(DM.Length()==2) DM_aver = StrToInt(DM) ;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit7Change(TObject *Sender)
{
        AnsiString DM = Edit7->Text;
        if(DM.Length()==4) IR_aver = StrToInt(DM) ;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
    AnsiString envio,recibido;
    char errorIR;
    char pCad3[4];
    unsigned short valor;
    Form1->Timer1->Enabled= false;
    envio=Edit7->Text;
    strcpy(pCad3,envio.c_str());
    valor = valor_decimal_otra(pCad3) ;
    if((valor>=0)&&(valor<=255)){
      if(valor>9) envio="@00RR00";
      if(valor>99) envio="@00RR0";
      if(valor<10) envio="@00RR000";
      envio=envio+valor;
      envio.Insert("0001",10) ;
      errorIR= envio_comando(envio,false);
      Sleep(200);
      errorIR= recepcion_comand(&recibido);
      Label25->Caption=recibido.SubString(8,4);


    }
    else  ShowMessage("valor fuera de rango");
    RadioButton1->Checked=false;

}
//---------------------------------------------------------------------------

/*&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
            &&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&*/

//---------------------     FUNCIONES   ---------------------------------------

//-------------------------calculo FCS  -------------------------------------------
-
AnsiString calculo_FCS(AnsiString cadena){

    char aux=0;
    int i;
    int j = cadena.Length();
    for (i=1;i<=j;i++){
       aux=aux^cadena[i];
     }
    return IntToHex(aux,2);
}

//--------------------Envio cadena de comandos al PLC----------------------
----------
 char envio_comando(AnsiString envio,bool tramalarga){

    char error;
    AnsiString FCS;
    FCS = calculo_FCS(envio);//calculo_FCS(Ansistring aux)
    if (tramalarga==false) envio=envio+FCS+"*\r";
    else  envio=envio+FCS+"\r";
    char acCad[133]="\0";
    char* pCad = new char [133];
    strcpy(acCad,envio.c_str());
    error = oTcom.TransmitirCadena(acCad);
    Sleep(35);   //Esperar respuesta del Adam
    delete pCad;
    return error;
 }

 //------------------------Respuesta del PLC ------------------- ----------------
--
 char recepcion_comand(AnsiString *recibido){
   char* pCad = new char [133];
   char R_comando[133]="\0";
   char error=1;
   bool Hay_cadenas=oTcom.RecibirCadena(pCad,133);
   strcpy(R_comando, pCad);
   *recibido=R_comando;
   if (Hay_cadenas==true) error=0;

 delete pCad;
 return error;
 }

//-------------Inicializacion de los 32 DM's -------------------------------
char inicializar_DM(){
 AnsiString envi_DM,aux,recibido,recibido1;         //esto solo funciona
en modo
 unsigned int intentos=10000;                  //programa del plc
  char error,error1,error2,error3;
 //unsigned short DM[32]={0};   //para poner a 0000 todos los datos de
las DMs

          unsigned  short   DM[]={0x0000,  0x0101,  0x0303,  0x0707,
```

```
0x0F0F, 0x1F1F, 0x3F3F, 0x7F7F,
   0xFFFF, 0xFEFE, 0xFCFC, 0xF8F8, 0xF0F0, 0xE0E0 ,
0xC0C0, 0x8080, 0x0000 , 0x8080,
   0xC0C0, 0xE0E0, 0xF0F0, 0xF8F8, 0xFCFC, 0xFEFE,
0xFFFF, 0x7F7F, 0x3F3F, 0x1F1F,
   0x0F0F, 0x0707, 0x0303, 0x0101};
 unsigned short *dm; //puntero para ir apuntando a los datos
del DM
  dm =&DM[0];  //apunto al primer dato de DM

    envi_DM="@00WD0000" ;       //primeros 16 DMs
     for (int i=0;i<16;i++){
       dm=&DM[i];
       aux=IntToHex(*dm,4);
       envi_DM = envi_DM + aux;
       }
       error= envio_comando(envi_DM,false);       //se envia la
trama completa
       Sleep(200);
       error1=  recepcion_comand(&recibido);      // se lee la
respuesta del plc

                    //segundos 16DMs
  if((recibido == "@00WD0053*\r")&&(error1==0)){
    envi_DM="@00WD0016" ;
      for (int i=16;i<32;i++){
      dm=&DM[i];
      aux=IntToHex(*dm,4);
      envi_DM = envi_DM + aux;
       }
  }
        error2= envio_comando(envi_DM,false);
        Sleep(200);
        error3=  recepcion_comand(&recibido1);     // se lee la
respuesta del plc

  return error;
}
//---------------------------------------------------------------------------
//funcion que pinta los leds
 void pintaled(AnsiString LED, TShape *oLED, int iPos){

   for (int i=1; i<17; i++){          //barrido de los componentes
hasta los 8
    oLED     =     (TShape*)Form1->FindComponent(LED     +
String(i));    // se ponen un nombre en comun "shape"  y se
cambia solo los nuemeros para poder barrer los 5
    if (iPos & (int)pow(2,i-1))        //segun las posicion de la
trackbar se hace el color

      oLED->Brush->Color = clLime;
    else
      oLED->Brush->Color = clGreen;

  }

  }
 //---------------cambiar caracter a valor entero----------------------
 int conv(char c) {
  int num=c;
 num=num-48;   //asci del 0
 if (num<=9)
 return num;
 if (c=='A')
 num = 10;
 if (c=='B')
num = 11;
 if (c=='C')
 num = 12;
 if (c=='D')
 num = 13;
 if (c=='E')
 num = 14;
 if (c=='F')
 num = 15;
return num;
}
//-------Cambiar valor a decimal --------------------------------
int valor_decimal(AnsiString recibido)  {

    unsigned int iPos1=0;
   char *aux,*aux1,recibi[4];
   aux=&recibi[0];
   recibido=recibido.SubString(8,4);
   strcpy(recibi,recibido.c_str());
   for(int ii=0; ii<4;ii++){
    aux1=&recibi[ii] ;
    iPos1=iPos1+  conv(*aux1)*pow(16,3-ii);       //valor a escribir en
decimal
     }
   return iPos1;
   }
//---------------------------------------------------------------------------

 //-------Cambiar valor a decimalIR --------------------------------
     unsigned short valor_decimal_otra(char* pCad1)  {
       AnsiString recibido=pCad1;
       unsigned int iPos1=0;
       char *aux,*aux1,recibi[4];
       aux=&recibi[0];
       strcpy(recibi,recibido.c_str());
       for(int ii=0; ii<4;ii++){
        aux1=&recibi[ii] ;
        iPos1=iPos1+ conv(*aux1)*pow(10,3-ii);      //valor a escribir en
decimal
       }
      return iPos1;
     }

//---------------------------------------------------------------------------
//-------inicializar los 5 primeros canales IR-------------------------

char inicializar_IR(void){
  AnsiString envi_IR,aux,recibido;         //esto solo funciona en modo

 char error;
unsigned short  DM[]={0xF0A1, 0x0060, 0x0800, 0x0A00, 0x0FD0 };
 unsigned short *dm; //puntero para ir apuntando a los datos del DM
 dm =&DM[0];  //apunto al primer dato de DM

   envi_IR="@00WR0000" ;       //primeros 16 DMs
    for (int i=0;i<5;i++){
      dm=&DM[i];
      aux=IntToHex(*dm,4);
      envi_IR = envi_IR + aux;
       }
      error=  envio_comando(envi_IR,false);          //se envia la trama
completa
                  Sleep(200);
                  error=  recepcion_comand(&recibido);     // se lee la
respuesta del plc

  return error;
 }
```

6.1.3 Aplicación Puente grua

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "TAPICOM.H"
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <dos.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "_GClass"
#pragma link "AbLED"
#pragma resource "*.dfm"
TForm1 *Form1;
T_API_COM oTcom(1);
DCB sComCfg1;

AnsiString calculo_FCS(AnsiString);
char envio_comando(AnsiString);
char recepcion_comand(AnsiString *);
int conv(char );
unsigned short valor_decimal(char* ) ;
char pCad3[4];
//---------------------------------------------------------------------------

int imagen=1;

__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{



}
//---------------------------------------------------------------------------


void __fastcall TForm1::ConectarClick(TObject *Sender)
{
oTcom.AbrirOtroPuerto(2) ;
sComCfg1= oTcom.ObtenerConfiguracion();


if (oTcom.EstaAbierto()){
oTcom.EstablecerConfiguracion(sComCfg1,9600,'E',2,7);
//DCB  sComCfg,  DWORD  dwVelocidad,  BYTE  bParidad,
BYTE bBitStop, BYTE bTrama){
}
else {
ShowMessage("puerto no abierto" );

}

char  error,error3;
AnsiString envio,recibido;
short intentos =0;
envio="@00TS12";
error= envio_comando(envio);
Sleep(300);
//recepcion-----
do{
error3= recepcion_comand(&recibido);
intentos++;
} while ((error3!=0)&&(intentos<100));
if (recibido.SubString(1,9)=="@00TS1244") {
Conectar->Visible=false;
B_pm->Visible=true;
B_pp->Visible=true;
BMarcha->Visible=true;
BParo->Visible=true;

}
Salir->Visible=true;
Image1->Visible=1;


}
//---------------------------------------------------------------------------
void __fastcall TForm1::BMarchaClick(TObject *Sender)
{

char  error;
AnsiString envio ,recibido;
unsigned short valor;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(200);
error= recepcion_comand(&recibido);
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
valor &=253;    //Apagar bit PP
valor |= 1;     //Activar bit PM
B_pp->Checked=false;
B_pm->Checked=true;
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(200);
error= recepcion_comand(&recibido);
Sleep(100);
if (imagen==39) imagen=1;// reiniciar el ciclo
Timer1->Enabled=1;

}

//---------------------------------------------------------------------------
void __fastcall TForm1::BParoClick(TObject *Sender)
{

char  error;
AnsiString envio ,recibido;
unsigned short valor;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(200);
error= recepcion_comand(&recibido);
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
valor &=254;    //Apagar bit PM
valor |= 2;     //Activar bit PP
B_pm->Checked=false;
B_pp->Checked=true;
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(200);
error= recepcion_comand(&recibido);
Sleep(100);
Timer1->Enabled=0;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{

char error;
AnsiString envio,recibido;
```

```
unsigned short valor,pp;
short intentos =0;
switch(imagen){
case 1: Image36->Visible=0;
Image1->Visible=1;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
//if(error!=0)      Timer1->Enabled=0,ShowMessage("No   hay
comunicacion");   //si no tiene respuesta
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No   hay   comunicacion");      //si   no   tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 20;                              //ACtivar el FCI y FCB
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 2: Image1->Visible=0;
Image3->Visible=1;     //Subir
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No   hay   comunicacion");      //si   no   tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 251;                             // desactivar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);

break;
case 3: Image3->Visible=0;
Image4->Visible=1;                  //Subir
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No   hay   comunicacion");      //si   no   tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 8;                               //Activar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 4: Image4->Visible=0;
Image5->Visible=1;                  //ir a la derecha    a FCD
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 239;                             // desactivar FCB
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 5: Image6->Visible=1;                  //ir a la derecha    a FCD
Image5->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 6: Image7->Visible=1;                  //ir a la derecha    a FCD
Image6->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 40;                              //Activar FCD
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
   Sleep(100);
break;
case 7:  Image8->Visible=1;
Image7->Visible=0;                  //Bajar
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
```

```
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 247;                              // desactivar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 8: Image9->Visible=1;
Image8->Visible=0;                         //Bajar
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 4;                                //Activar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 9: Sleep(2000);      // Desengrasado
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;

break;
case 10:  Image10->Visible=1;                  //subir
Image9->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 251;                              // desactivar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 11: Image11->Visible=1;                  //subir
Image10->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 8;                                //Activar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 12: Image12->Visible=1;                  //ir a la derecha    a
FCN
Image11->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 223;                              // desactivar FCD
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 13: Image13->Visible=1;                  //ir a la derecha    a
FCN
Image12->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 14: Image14->Visible=1;                  //ir a la derecha    a
FCN
Image13->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
```

```
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 64;                              //Activar FCN
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 15: Image15->Visible=1;                    // bajar
Image14->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 247;                             // desactivar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 16: Image16->Visible=1;                    // bajar
Image15->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 4;                               //Activar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 17: Sleep(5000);      //Decapado-Neutralizado
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;

break;
case 18: Image17->Visible=1;              //subir
Image16->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 251;                             // desactivar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 19: Image18->Visible=1;              //subir
Image17->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 8;                               //Activar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 20: Image19->Visible=1;                    //ir a la derecha    a
FCE
Image18->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 191;                             // desactivar FCN
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 21: Image20->Visible=1;                    //ir a la derecha    a
FCE
Image19->Visible=0;
```

```
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 22: Image21->Visible=1;                              //ir a la
derecha  a FCE
Image20->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 128;                              //Activar FCE
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 23:Image22->Visible=1;                          //bajar
Image21->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 247;                              // desactivar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 24: Image23->Visible=1;                          //bajar
Image22->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 4;                                //Activar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 25:Sleep(10000); //Electrolitico
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;

break;
case 26:  Image24->Visible=1;              //subir
Image23->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 251;                              // desactivar FCI
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 27: Image25->Visible=1;              //subir
Image24->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor |= 8;                                //Activar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 28: Image26->Visible=1;              // Ir a la izquierda
```

```
Image25->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
valor &= 127;                              // desactivar FCE
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 29:Image27->Visible=1;                // Ir a la izquierda
Image26->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 30: Image28->Visible=1;                // Ir a la izquierda
Image27->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 31: Image29->Visible=1;                // Ir a la izquierda
Image28->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");      //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 32: Image30->Visible=1;                // Ir a la izquierda
Image29->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 33: Image31->Visible=1;                // Ir a la izquierda
Image30->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 34: Image32->Visible=1;                // Ir a la izquierda
Image31->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
                strcpy(pCad3,recibido.SubString(8,4).c_str());
                valor = valor_decimal(pCad3 ) ;
                pp = valor&2;
                if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 35: Image33->Visible=1;                // Ir a la izquierda
Image32->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion");  //si no tiene respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;      // si esta pulsado PP parar
imagen++;
Sleep(100);
break;
case 36: Image34->Visible=1;                // Ir a la izquierda
Image33->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
```

```
Timer1->Enabled=0;
ShowMessage("No hay comunicacion"); //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;    // si esta pulsado PP parar
valor |= 16;        // activar FCB
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 37: Image35->Visible=1;          // Ir a la izquierda
Image34->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion"); //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;    // si esta pulsado PP parar
valor &= 247;       // desactivar FCS
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00") imagen++;
Sleep(100);
break;
case 38: Image36->Visible=1;          // Ir a la izquierda
Image35->Visible=0;
envio="@00RR00000001";
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if(error!=0){
Timer1->Enabled=0;
ShowMessage("No hay comunicacion"); //si no tiene
respuesta
}
strcpy(pCad3,recibido.SubString(8,4).c_str());
valor = valor_decimal(pCad3 ) ;
pp = valor&2;
if(pp==2) break;    // si esta pulsado PP parar
valor &=254;  //Apagar bit PM
valor |=4;  //Activar FCI
B_pm->Checked=false;
envio=IntToHex(valor,4);
envio="@00WR0000"+envio;
error= envio_comando(envio);
Sleep(300);
error= recepcion_comand(&recibido);
if (recibido.SubString(6,2)=="00")  imagen=39;
Sleep(100);
break;
}

}
//-------------------------------------------------------------------------


/*&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&*/

//-------------------- FUNCIONES ------------------------------------------


//-------------------------calculo FCS ---------------------------------------------
-
AnsiString calculo_FCS(AnsiString cadena){

char aux=0;
int i;
int j = cadena.Length();
for (i=1;i<=j;i++){
aux=aux^cadena[i];
}
return IntToHex(aux,2);
}

//---------------------Envio cadena de comandos al PLC----------------------
----------
char envio_comando(AnsiString envio){

char error;
AnsiString FCS;
FCS = calculo_FCS(envio);//calculo_FCS(Ansistring aux)
envio=envio+FCS+"*\r";
char acCad[133]="\0";
char* pCad = new char [133];
strcpy(acCad,envio.c_str());
oTcom.TransmitirCadena(acCad);
Sleep(35);  //Esperar respuesta del Adam

delete pCad;
return error;
}

//------------------------Respuesta del PLC ------------------- ----------------
-
char recepcion_comand(AnsiString *recibido){
char* pCad = new char [133];
char R_comando[133]="\0";
char error=1;
bool Hay_cadenas=oTcom.RecibirCadena(pCad,133);
strcpy(R_comando, pCad);
*recibido=R_comando;
if (Hay_cadenas==true) error=0;

delete pCad;
return error;
}


//--------------------------------------------------------------------------
//-------Cambiar valor a decimal ---------------------------------
unsigned short valor_decimal(char* pCad1)  {
AnsiString recibido=pCad1;
unsigned int iPos1=0;
char *aux,*aux1,recibi[4];
aux=&recibi[0];
strcpy(recibi,recibido.c_str());
for(int ii=0; ii<4;ii++){
```

```
aux1=&recibi[ii] ;
iPos1=iPos1+ conv(*aux1)*pow(16,3-ii);    //valor a escribir en
decimal
}
return iPos1;
}

//--------------cambiar caracter a valor entero----------------------
int conv(char c) {
int num=c;
num=num-48;    //asci del 0
if (num<=9)
return num;
if (c=='A')
num = 10;
if (c=='B')
num = 11;
if (c=='C')
num = 12;
if (c=='D')
num = 13;
if (c=='E')
num = 14;
if (c=='F')
num = 15;
return num;
}
```

6.1.4 TAPICOM

TAPICOM.H

```
//Fichero de declaración TAPICOM.h con la clase T_API_COM
//---------------------------------------------------------------------------
#ifndef TAPICOMH
#define TAPICOMH
//---------------------------------------------------------------------------
#include <windows.h>

// Declaración de la clase
class T_API_COM {

  protected:
    short NPuerto; // Número de puerto
    bool bEstaAbierto; // ¿ Abierto ?
    HANDLE hCom; // Manejador del puerto
    HANDLE hComO; // Manejador del otropuerto
    DCB sComCfg; // Estructura configuración
    COMSTAT sComSta; // Estado del puerto
    DWORD dwErr; // Errores del puerto
    DWORD dwTamBufIn; // Tamaño del buffer de entrada
    DWORD dwTamBufOut; // Tamaño del buffer de salida
    COMMTIMEOUTS sTimOut; // Estructura de temporizaciones

  public:
    DWORD N_caracteres_recibidos;
    DWORD N_caracteres_enviar;
    T_API_COM(short=1); // Constructor con parámetro por defecto
    ~T_API_COM(); // Destructor (cerrará el puerto)
    bool AbrirPuerto(); // Abrir puerto seleccionado
    bool AbrirOtroPuerto(int iPort); // Abrir otro puerto
    bool CerrarPuerto(); // Cerrar puerto seleccionado
    bool EstaAbierto(); // Estado actual del puerto
    HANDLE LeeHANDLE(); // Leer "manejador" del COM
    DCB ObtenerConfiguracion(); // Obtener configuración actual
//Modifica ciertos campos de la estructura DCB
    bool EstablecerConfiguracion(DCB sComCfg, DWORD
dwVelocidad=9600,
                BYTE bParidad='N', BYTE bBitStop=1, BYTE
bTrama=8);
    DWORD BytesEnBufferRx(); // Caracteres pendientes de
lectura
    bool RecibeCaracter(BYTE& bDat); // Recibe un carácter
    bool TransmiteCaracter(BYTE bDat); // Transmite un carácter
// Transmitir una cadena de caracteres
    bool TransmitirCadena(char* pCadena);
// Recibir una cadena de caracteres
    bool RecibirCadena(char* pCad, DWORD dwLen);
    DWORD Get_N_caracteres_recibidos();
    DWORD Get_N_caracteres_enviar();
    bool BorrarTxRx();//Borra los buffer de transmicion y recepcion

};
/*****************************************************************************
*
************************                                          PLC
****+*************************************
*****************************************************************************
*/
class T_plc : public T_API_COM {


      private:
      unsigned  short estado ;   // 3=run,2=monitor,0=program
      bool power;
      unsigned short Error_CPU[2]; //guarda codigo de error
de CPU
      char* C_error;    //guarda el codigo de finalizacion de
respuesta de los comandos
      char* modelo;
      bool pin_DIP;  //Proteccion contra escritura
                  bool Trama_larga;


    unsigned  short  IR[256];
    unsigned  short LR [64];
    unsigned  short HR [100];
    unsigned  short PV [512];
    bool          TC [512];
    unsigned  short DM [1024];
    unsigned  short AR [28];
    unsigned  short SV [512];
    unsigned  char   SV_prefijo[512];

  public:
      //constructores
      T_plc(void );  //sin parametros
      T_plc(int ,bool ,bool  ,char* );  //con parametros
      //Destructor
      ~T_plc();
      void  Set_estado(unsigned short);       // 0 monitor
                        //2 programa
                        //3 run
      unsigned short  Get_estado  ();        //enviar estado plc

     //------   Error del PLC --------
     // void Set_error(bool);
     // bool Get_error();

      //------   Error del PLC --------
      void Set_eCPU(unsigned short,unsigned short);
      unsigned short Get_eCPU(unsigned short);

      //------  Codigo de Finalizacion ------

      void Set_ComError(char*);
      char* Get_comError(void );

      //--------- Proteger escritura de memoria de usuario----
      void Set_DIP(bool);
      bool Get_DIP(void);

      //--------- Trama larga ----
      void Set_Trama_larga(bool);
      bool Get_Trama_larga(void);

     //----------Memorias---------------
      void Set_IR(unsigned short, unsigned short);
      void Set_LR(unsigned short, unsigned short);
      void Set_HR(unsigned short, unsigned short);
      void Set_PV(unsigned short, unsigned short);
      void Set_TC(unsigned short, bool);
      void Set_DM(unsigned short, unsigned short);
      void Set_AR(unsigned short, unsigned short);
      void Set_SV(unsigned short, unsigned short);
      void Set_SV_prefijo(unsigned short, unsigned char);
      unsigned short Get_IR(unsigned short);
      unsigned short Get_LR(unsigned short);
      unsigned short Get_HR(unsigned short);
      unsigned short Get_PV(unsigned short);
      bool Get_TC(unsigned short);
      unsigned short Get_DM(unsigned short);
      unsigned short Get_AR(unsigned short);
      unsigned short Get_SV(unsigned short);
      unsigned char Get_SV_prefijo(unsigned short);


      int conv(char ) ; //---cambiar caracter a valor entero----------

};
```

```
/*****************************************************************************
************************  ERRORES  ***********************************
                ******************************************************************************
                **/

                class T_Error:public T_plc{

                    private:
                        char* e_Comando;
                    public:
                        //constructores
                        T_Error(void); //sin  parametros
                        T_Error(char* );  //con parametros
                        //Destructor
                        ~T_Error();

                        bool e00(char*);
                        bool e01(char*);
                        bool e02(char*);
                        bool e03(char* ,unsigned int, unsigned int);
                        bool e04(char*,unsigned int);
                        bool e0B(char*);
                        bool e13(char*);

                        bool e14(char*,unsigned int );
                        bool e15(char*, unsigned int , unsigned int );
                        bool e16(char*);
                        bool e18(char*);
                        bool e19(char*);
                        bool e21(void);

                        bool e23(char*);
                        bool eA3(char*);
                        bool eA4(char*);
                        bool eA5(char*);
                        bool eA8(char*);
                        unsigned long rango(char*,bool) ;
                        void FCS(char* pCad1 )  ;

                };

                /*****************************************************************************
                **
                ************************                                    COMANDOS
                ***************************************
                *****************************************************************************
                **/
                // Declaración de la clase
                class T_Comandos:public T_Error{

                  private:
                                    char C_prueba[32];
                                    int estado1;//Estado del PLC
                                    bool ok ;
                    unsigned int jj;  //para saber la longitud del comando recibido si es ok
                  public:
                    //constructores
                    T_Comandos(char* );  //con parametros

                    //Destructor
                    ~T_Comandos();


                    bool RR (char* pCad1);  //     LECTURA DE AREA IR/SR 335
                    bool RL (char* pCad1);  //  LECTURA DE AREA LR 336
                    bool RH (char* pCad1);  //  LECTURA DE AREA HR 336
                    bool RC (char* pCad1);  // LECTURA DE PV 337
                    bool RG (char* pCad1);  // LECTURA DE ESTADO DE TC 337
                    bool RD (char* pCad1);  //  LECTURA DE AREA DE DM 338
                    bool RJ (char* pCad1);  // LECTURA DE AREA DE AR 338
                    bool WR (char* pCad1);  //  ESCRITURA DE AREA IR/SR 339
                    bool WL (char* pCad1);  // ESCRITURA DE AREA LR 339
                    bool WH (char* pCad1);  //  ESCRITURA DE AREA HR 340
                    bool WC (char* pCad1);  //  ESCRITURA DE PV 340
                    bool WG (char* pCad1);  //  ESCRITURA DE ESTADO DE TC 341
                    bool WD (char* pCad1);  // ESCRITURA DE AREA DM 342
                    bool WJ (char* pCad1); // ESCRITURA DE AREA AR 342
                    bool Ralmu (char* pCad1);  // LECTURA SV 1 343
                    bool Walmu (char* pCad1);  //  CAMBIAR SV 1 346
                    bool MS (char* pCad1);  // LECTURA DE ESTADO 348
                    bool SC (char* pCad1);  //  ESCRITURA DE ESTADO 349
                    bool MF (char* pCad1);  //  LECTURA DE ERROR 350
                    bool KS (char* pCad1);  // FORZADO A ON 351
                    bool KR (char* pCad1);  //  FORZADO A OFF 352
                    bool FK (char* pCad1);  //  FORZADOS MULTIPLES A ON/OFF 353
                    bool KC (char* pCad1);  //  CANCELACION DE FORZADOS 354
                    bool MM (char* pCad1);  // LECTURA DE MODELO DE PLC 354
                    bool TS (char* pCad1);  //  PRUEBA DE COMUNICACIONES 355
                    bool QQ (char* pCad1);  // COMANDO COMPUESTO 356
                    bool XZ (char* pCad1); // ABORTAR (sólo comando) 358
                    bool aste_aste (char* pCad1);  //  INICIALIZAR (sólo comando) 358
                    bool IC(char* pCad1);   // Comando indefinido

                    unsigned short valor_decimal(char* pCad1);      //Cambiar valor a
                decimal de un hexadecimal

                    void trama(char*);
                };
                #endif
```

TAPICOM.cpp

```
#include "TAPICOM.H"
#include <Windows.h>
#include <math.h>
#include <vcl.h>
#include "Prepara.H"
#pragma hdrstop
#include <iostream.h>
#include <conio.h >


//-----------------Constructor con parámetro por defecto-----------------
//*************************************************************************
**********
        T_API_COM::T_API_COM(short com){
          NPuerto=com;
          bEstaAbierto=false;
        }
//------------------- Destructor (cerrará el puerto)----------------------
//*************************************************************************
**********
        T_API_COM::~T_API_COM(){
          if (hCom!=INVALID_HANDLE_VALUE)
          CloseHandle(hCom);
        }
//------------------- Abrir puerto seleccionado---------------------------
//*************************************************************************
**********
        bool T_API_COM::AbrirPuerto(){
          char*  cNpuerto;
          cNpuerto= new char;
          char acConf[10]="COM";
          itoa(NPuerto, cNpuerto, 10);     //convertir entero a cadena
en base 10
          strcat(acConf, cNpuerto);
          hCom=CreateFile(acConf,
                    GENERIC_READ|GENERIC_WRITE,
                    0,0,OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,0);
          if (hCom!=INVALID_HANDLE_VALUE) bEstaAbierto=true;
          delete cNpuerto;
          return  bEstaAbierto;
        }
//----------------------- Abrir otro puerto-----------------------------
--
//*************************************************************************
**********
        bool T_API_COM::AbrirOtroPuerto(int iPort){
          NPuerto=iPort;
          CerrarPuerto();
          bEstaAbierto=AbrirPuerto();
          return  bEstaAbierto;
        }
//----------------------Cerrar puerto seleccionado-------------------------
---------------
//*************************************************************************
**********
        bool T_API_COM::CerrarPuerto(){
          if (hCom!=INVALID_HANDLE_VALUE) {
            if (CloseHandle(hCom)) bEstaAbierto=false;
          }
          return  bEstaAbierto;
        }
//---------------------- Estado actual del puerto-------------------------
-------------
//*************************************************************************
**********
        bool T_API_COM::EstaAbierto(){
          return bEstaAbierto;
        }
//---------------------- Leer “manejador” del COM-----------------
                    ----------------------
                    //*************************************************************************
                    ******************
        HANDLE T_API_COM::LeeHANDLE(){
          return (hCom);
        }
//----------------------- - Obtener configuración actual del puerto----------------
----------------------
//*************************************************************************************
*
        DCB T_API_COM::ObtenerConfiguracion(){
          GetCommState(hCom,&sComCfg);
          return sComCfg;
        }
//-------------------------Configurar puerto-----------------------------------
//******************Modifica ciertos campos de la estructura DCB
********************
        bool T_API_COM::EstablecerConfiguracion(DCB sComCfg, DWORD
dwVelocidad,
                                                BYTE bParidad, BYTE bBitStop, BYTE
bTrama){
          sComCfg=prepararconfi(sComCfg,      dwVelocidad,      bParidad,
bBitStop,  bTrama);
          bool GetConf=SetCommState(hCom,&sComCfg);
          return  GetConf;
        }
//----------------------- // Caracteres pendientes de lectura--------------------------
//*************************************************************************************
*
        DWORD T_API_COM::BytesEnBufferRx(){
          if( ClearCommError(hCom,&dwErr,&sComSta)){
            DWORD  iCar  =  sComSta.cbInQue;// numero de caracteres
pendientes de lectura
            return iCar;
          }
        }
//----------------------- Recibe un carácter-----------------------------------
//*************************************************************************************
*
        bool T_API_COM::RecibeCaracter(BYTE& bDat){
          sTimOut= ConfTimeOut(sTimOut);
          bool oKcar=false;
          if (SetCommTimeouts(hCom,&sTimOut)){
            DWORD  dwLen=BytesEnBufferRx() ;
            if(dwLen>0)
              oKcar=ReadFile(hCom,&bDat,dwLen,&dwTamBufIn,0);
          }
          return oKcar;
        }
//----------------------- Transmite un carácter--------------------------------
//*************************************************************************************
*
        bool T_API_COM::TransmiteCaracter(BYTE bDat){
          sTimOut= ConfTimeOut(sTimOut);
          DWORD dwLen;
          BYTE acCar[3] ="0";;
          acCar[0]=bDat;
          strcat(acCar,"\r\n");
          dwLen=strlen(acCar);       //longitud de la cadena  dando el nº de
caracteres
          bool oKcar=false;
          oKcar=WriteFile(hCom,acCar,dwLen,&dwTamBufOut,0);
          return oKcar;
        }
//----------------- Transmitir una cadena de caracteres-----------------------
//*************************************************************************************
*
        bool T_API_COM::TransmitirCadena(char* pCadena){
          sTimOut= ConfTimeOut(sTimOut);
          char acBuff[200]="\0";
          strcpy(acBuff,pCadena);
        //strcat(acBuff,"\n");
                    DWORD   dwLen=strlen(acBuff);// con esto se sabe cuantos
```

```
caracteres hay
        N_caracteres_enviar = dwLen;
        return WriteFile(hCom,acBuff,dwLen,&dwTamBufOut,0);
       }

//---------------------- Recibir una cadena de caracteres----------------------
---------
//**************************************************************************
**********
       bool   T_API_COM::RecibirCadena(char*   pCad,   DWORD
dwLen){
        sTimOut= ConfTimeOut(sTimOut);
        bool oKcar=false;
        char acBuf[133]="\0";
        DWORD  dwLen1=BytesEnBufferRx() ;   // con esto se sabe
cuantos caracteres hay
        N_caracteres_recibidos=dwLen1;
        if(dwLen1>0&&dwLen1<=dwLen){
         oKcar=ReadFile(hCom,acBuf,dwLen1,&dwTamBufIn,0);
         //acBuf[dwLen]='\0';
         strcpy(pCad,acBuf);
        }
        else  {
         *pCad=0;
         pCad=0;}
        return oKcar;
       }

 //---------------------- //lectura del numero de caracteres recibidos-------
-----------------------
//**************************************************************************
**********
       DWORD T_API_COM::Get_N_caracteres_recibidos(){
         return N_caracteres_recibidos;
       }
//---------------------- //lectura del numero de caracteres enviar----------
-------------------
//**************************************************************************
**********
       DWORD T_API_COM::Get_N_caracteres_enviar(){
         return N_caracteres_enviar;
       }

 //---------------------- //Borra los buffer de transmicion y recepcion------
-----------------------
//**************************************************************************
**********
       bool T_API_COM::BorrarTxRx(){
         bool                                          okBo=
PurgeComm(hCom,(PURGE_TXCLEAR||PURGE_TXCLEAR));
        return okBo;
       }

*/**************************************************************************
**********************
************************************                                  PLC
*************************************************
*****************************************************************************
**********************
*****************************************************************************
**********************/

//--------constructor sin parametros----------------------------------------
---------
       T_plc::T_plc(void){
       }
//--------constructorparametros---------------------------------------------
----------
       T_plc::T_plc(int   estado1,bool   v_Power,bool   v_error,char*
v_Modelo):T_API_COM(1){
        unsigned int dato=0;
        int i,dato1;
        AnsiString memoria;
        char c[4];
        estado = estado1;
        power=v_Power;
        if (v_error==false) Error_CPU[0] =0,Error_CPU[1] =0;
        modelo=v_Modelo;
        memoria = "0000";
        strcpy(c,memoria.c_str());
        for(i=0;i<4;i++){
         dato=dato+conv(c[i])*pow(16,3-i);
        }
        for(i=0;i<=131;i++){

if(i<=63)IR[i]=dato,LR[i]=dato,HR[i]=dato,PV[i]=dato,DM[i]=dato,
                SV[i]=dato;
         else if (i<=100)IR[i]=dato,HR[i]=dato,PV[i]=dato,DM[i]=dato,
                SV[i]=dato;
         else   IR[i]=dato,PV[i]=dato,DM[i]=dato,
                SV[i]=dato;
         if((i==31)&&(i<=63)){

IR[i]=43690,LR[i]=43690,HR[i]=43690,PV[i]=43690,DM[i]=43690,
               SV[i]=43690; }

         if(i==31) HR[i]=43690;
         if (i==59) IR[i]=43690,HR[i]=43690,DM[i]=43690;
         if (i==63) LR[i]=0;
         if (i==99) IR[i]=0,HR[i]=0,DM[i]=0;

        }
        IR[255]=43690;
        DM[1023]=43690;


       }
//-------------Destructor----------------------------------
       T_plc::~T_plc(){ }

//---------------      Metodos-----------------------------------------------
//---------------------------------------------------------------------

       void    T_plc::Set_estado(unsigned  short  aux){            //3  run,
2=monitor, 0=programa
        estado = aux;
       }


       unsigned short  T_plc::Get_estado (){          //enviar estado plc
        return estado;
       }

      void T_plc::Set_eCPU(unsigned short canal,unsigned short dato){
        unsigned short* p_dato;
        p_dato=&Error_CPU[canal];
        *p_dato=dato ;
       }

       unsigned short T_plc::Get_eCPU(unsigned short canal){
        return Error_CPU[canal];

       }

       //------  Error de comando ------
       void T_plc::Set_ComError(char* E_comando){
         C_error=E_comando;
       }
       char* T_plc::Get_comError(void){
         return C_error;
       }
       //--------- Proteger escritura de memoria de usuario----
       void T_plc::Set_DIP(bool b_DIP){
         pin_DIP=b_DIP;
       }
       bool T_plc::Get_DIP(void){
         return pin_DIP;
       }

       //--------- Trama larga ----
       void  T_plc::Set_Trama_larga(bool trama_larga){
         Trama_larga=trama_larga;
```

```
        }
        bool T_plc::Get_Trama_larga(void){
           return Trama_larga;
        }
 //-------------------- Escritura de memorias-------------------------------
-------------
        void  T_plc::Set_IR(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&IR[canal];
           *p_dato=dato ;
         }
        void  T_plc::Set_LR(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&LR[canal];
           *p_dato=dato ;
        }
        void  T_plc::Set_HR(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&HR[canal];
           *p_dato=dato ;
        }
        void  T_plc::Set_PV(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&PV[canal];
           *p_dato=dato ;
        }
        void T_plc::Set_TC(unsigned short canal, bool estado){
           bool* p_dato;
           p_dato=&TC[canal];
           *p_dato=estado;
        }
        void  T_plc::Set_DM(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&DM[canal];
           *p_dato=dato ;
        }
        void  T_plc::Set_AR(unsigned  short  canal,  unsigned  short
dato){
           unsigned short* p_dato;
           p_dato=&AR[canal];
           *p_dato=dato ;
        }
        void  T_plc::Set_SV(unsigned  short  canal,  unsigned  short
dato){
            unsigned short* p_dato;
           p_dato=&SV[canal];
           *p_dato=dato ;
        }
        void  T_plc::Set_SV_prefijo(unsigned  short  canal,  unsigned
char prefijo){
           SV_prefijo[canal]=prefijo;
        }

 //-------- Lectura de memorias ----------------------------------------
        unsigned short T_plc::Get_IR(unsigned short v_indx){
           return IR[v_indx];
        }
        unsigned short T_plc::Get_LR(unsigned short v_indx){
           return LR[v_indx];
        }
        unsigned short T_plc::Get_HR(unsigned short v_indx){
           return HR[v_indx];
        }
        unsigned short T_plc::Get_PV(unsigned short v_indx){
           return PV[v_indx];
        }
        bool T_plc::Get_TC(unsigned short v_indx){
           return TC[v_indx];
        }
        unsigned short T_plc::Get_DM(unsigned short v_indx){
           return DM[v_indx];
                        }
                        unsigned short T_plc::Get_AR(unsigned short v_indx){
                           return AR[v_indx];
                        }
                        unsigned short T_plc::Get_SV(unsigned short v_indx){
                           return SV[v_indx];
                        }
        unsigned char T_plc::Get_SV_prefijo(unsigned short canal){
             return SV_prefijo[canal];
        }

        //---------------cambiar caracter a valor entero----------------------
        int T_plc:: conv(char c) {
          int num=c;
          num=num-48;    //asci del 0
          if (num<=9)
          return num;
          if (c=='A')
          num = 10;
          if (c=='B')
          num = 11;
          if (c=='C')
          num = 12;
          if (c=='D')
          num = 13;
          if (c=='E')
          num = 14;
          if (c=='F')
          num = 15;
          return num;
        }

/*****************************************************************************
*********************  Errores ***********************************************
*****************************************************************************/
   //constructores
        T_Error::T_Error(void ){    //sin parametros

        }
        T_Error::T_Error(char*  pCad3):T_plc(0,true,false,"CQM1"){//T_plc(int
estado1,bool v_Power,bool  Verror,char* v_Modelo){
           e_Comando=pCad3;
        }
        //Destructor
        T_Error::~T_Error(){}
 //---------------- Metodos-----------------------------------------------
 //-----------------------------------------------------------------------
        bool T_Error::e00(char* pCad3){   // %%%%% Finalizacion normal
          bool e00=true;
          pCad3 ="00";
          this->Set_ComError(pCad3);
          return e00;
        }
        bool  T_Error::e01(char*  pCad3){        // %%%%% No ejecutable en
modo run
          bool e01;
          unsigned short aux=this->Get_estado();
          aux>>=8;
          if (aux == 3)  e01 = true  ;
          pCad3="01" ;
          if(e01==true)  this->Set_ComError(pCad3);
          return e01;
        }
        bool  T_Error::e02(char*  pCad3){        // %%%%% No ejecutable en
modo monitor
          bool e02;
          unsigned short aux=this->Get_estado();
          aux>>=8;
          if (aux == 2) e02 = true ;
          pCad3="02" ;
          if(e02==true)  this->Set_ComError(pCad3);
          return e02;
        }
        bool  T_Error::e03(char*  pCad3,  unsigned   int  D_inicio,  unsigned  int
D_fin ) {  // %%%%% Proteccion de escritura UM
          bool e03 = false;
```

```
        AnsiString aux = pCad3;
        unsigned int  aux1;
        aux=aux.SubString(6,4);
        aux1=StrToInt(aux);
        if ((aux1<D_fin)&&(aux1>D_inicio)) e03 = true;
        pCad3="03" ;
        if(e03==true)  this->Set_ComError(pCad3);
        return e03;
       }
      bool T_Error::e04(char* pCad3, unsigned int Max_direccion){
// %%%%% Direccion excedida
        AnsiString aux = pCad3;
        unsigned long  aux1;
        bool e04 = false;
        aux=aux.SubString(6,4);
        aux1=StrToInt(aux);
        if (aux1 >  Max_direccion) e04 = true;
        pCad3="04" ;
        if(e04==true)  this->Set_ComError(pCad3);
        return e04;
       }
      bool T_Error::e0B(char* pCad3){   // %%%%% No ejecutable
en modo programa
        bool e0B;
        unsigned short aux=this->Get_estado();
        if (aux == 0)  e0B = true ;
        pCad3="0B" ;
        if(e0B==true) this->Set_ComError(pCad3);
        return e0B;
       }
      bool T_Error::e13(char* pCad3){           // %%%%% El FCS
es falso
        bool e13 = false;
        AnsiString aux,aux1;
        aux = pCad3;
        aux1=aux;     //copia del comando
        int j= aux.Length() ;
        j=j-4;
        aux=aux.SubString(1,j);
        strcpy(pCad3,aux.c_str());
        FCS(pCad3);   //Obtengo el FCS correcto
        aux=aux1;
        j= aux.Length() ;
        j=j-3;
        aux=aux.SubString(j,2);
        if ( aux!=pCad3)  e13 = true;  // compara el FCS que saco
del dato con el FCS que me envian
        pCad3="13" ;
        if(e13==true)  Set_ComError(pCad3);
        return e13;
       }
      bool  T_Error::e14(char*  pCad3,unsigned  int  longitud){      //
%%%%% Error de formato
        unsigned long  aux1=longitud;
        short resto;
        bool e14=true;
        pCad3="14" ;
        resto=aux1%4;
        if(resto==0) {
          aux1=1;
          e14=false;
         }
        else aux1=0;
        if((e14==true)||(aux1==0))  Set_ComError(pCad3);
        return e14;
       }
      bool   T_Error::e15(char*   pCad3,   unsigned   int   D_min,
unsigned int D_max){  // %%%%% las Areas para leer y escribir son
malas
        AnsiString aux = pCad3,auxx;
        unsigned long  aux1,auxx1,resto,cociente;
        int longitud;
        longitud=aux.Length();
        bool e15 = false;
        if(aux.SubString(4,1)=='R'){
          auxx= aux.SubString(10,4);
          aux=aux.SubString(6,4);
          aux1=StrToInt(aux);
          auxx1=StrToInt(auxx)-1;
          if ((auxx1>D_max)||(aux1<D_min))e15 = true;
          pCad3="15" ;
          if(e15==true)  Set_ComError(pCad3);
         }
        else if (aux.SubString(4,1)=='W'){
          auxx=aux.SubString(10,longitud);
          longitud=auxx.Length();
          auxx=auxx.SubString(0,longitud-4);
          longitud=auxx.Length();
          cociente=longitud/4;
          resto=longitud%4;
          if(resto!=0)  Set_ComError("14");
          else{
            aux=aux.SubString(6,4);
            aux1=StrToInt(aux);
            if ((cociente>D_max)||(aux1<D_min))e15 = true;
            pCad3="15" ;
            if(e15==true)  Set_ComError(pCad3);
           }
         }
        else Set_ComError("14");
        return e15;
       }
      bool T_Error::e16(char* pCad3){          // %%%%% comando no
soportado(Leer SV, etc ->mal)

       }
      bool T_Error::e18(char* pCad3){          // %%%%% Error en la
longitud de la trama
        int Lon_max=131;
        bool e18 = false;
        AnsiString aux = pCad3;
        int j = aux.Length();
        if (j>Lon_max)  e18=true;
        pCad3="18" ;
        if(e18==true)  Set_ComError(pCad3);
        return e18;
       }
      bool T_Error::e19(char*pCad3 ){       // %%%%% No ejecutable
(para comando compuesto QQ)

                }
      bool T_Error::e21(void){          //  %%%%% Error de CPU
        unsigned short aux;
        bool e21=false;
        aux=Get_eCPU(0);
        if (aux!=0) e21 = true;
        else {aux=Get_eCPU(1);
            if (aux!=0) e21=true;
            }
        if (e21==true)  Set_ComError("21");
        return e21;
       }
      bool  T_Error::e23(void){     // %%%%% memoria de usuario
protegida contra escritura  y lectura
        bool e23=Get_DIP();
        if (e23==true) Set_ComError("23");
        return e23;
       }
      //----------------Errores con cadenas multimples ----------
----------
      bool T_Error::eA3(char* pCad3){ // %%%%% Abortado debido a
error de FCS en la transmision de dato

       }
      bool T_Error::eA4(char* pCad3){ // %%%%% cancelado debido
a un error de formato en la transmision

       }
      bool T_Error::eA5(char* pCad3){ // %%%%% cancelado debido
a un error de numero  de entrada de datos en transmision

       }
```

```
        bool   T_Error::eA8(char*pCad3){   //   %%%%%   cancelado
debido a error de longitud de trama en transmision de datos

        }
 //---------------Funciones---------------------------------------------------
 //---------------------------------------------------------------------------
        unsigned long T_Error::rango(char*pCad3,bool npalabra){
          int j=6;
          if (npalabra) j=10;
          AnsiString aux=pCad3;
          aux=aux.SubString(j,4);
          strcpy(pCad3,aux.c_str());
          unsigned long  a= StrToInt(pCad3);
          return a;
        }

        void T_Error::FCS(char* pCad1){
          AnsiString cadena;
          cadena = pCad1;
          char aux=0;
          int i;
          int j = cadena.Length();
          for (i=1;i<=j;i++){
            aux=aux^cadena[i];
          }

          cadena=IntToHex(aux,2);
          strcpy(pCad1,cadena.c_str());
        }

//***************************************************************************
***********
//--------------------------------Comandos-----------------------------------
-----
//***************************************************************************
***********

//------------Constructor-----------------------------------
        T_Comandos::T_Comandos(char* pCad1):T_Error(pCad1){
          strcpy(C_prueba, pCad1);
          ok = false;
        }
//--------------Destructor-----------------------------------
        T_Comandos::~T_Comandos(){ }

 //---------------- Metodos-----------------------------------------------------
 //-------------------------------lectura-----------------------------------------
        bool  T_Comandos::RR(char*  pCad1){          //%%%%%%%%
LECTURA DE AREA IR/SR 335
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (e00(pCad1)){          //Terminacion normal
            ok = true;
          }

          strcpy(pCad1,aux.c_str());
          if (e04(pCad1,6655)) {   //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e18(pCad1)) {     // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e14(pCad1,jj-9)) {     // %%%%% Error de formato
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e15(pCad1,0,255)) {  //las Areas para leer y escribir
son malas
            ok = false;
            goto salir;
          }
          if(e21()){          //Error de CPU
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e13(pCad1)) {     //El FCS es falso
            ok = false;
            goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
        }

//***************************************************************************
***********************
        bool  T_Comandos::RL  (char*  pCad1){           //%%%%%%%%
LECTURA DE AREA LR 336
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (e00(pCad1)){          //Terminacion normal
            ok = true;
          }
          strcpy(pCad1,aux.c_str());
          if (e04(pCad1,6655)) {   //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e18(pCad1)) {     // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e14(pCad1,jj-9)) {     // %%%%% Error de formato
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e15(pCad1,0,64)) {  //las Areas para leer y escribir son malas
            ok = false;
            goto salir;
          }
          if(e21()){          //Error de CPU
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e13(pCad1)) {     //El FCS es falso
            ok = false;
            goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
        }
//***************************************************************************
***********************
        bool  T_Comandos::RH  (char*  pCad1){           //%%%%%%%%
LECTURA DE AREA HR 336
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (e00(pCad1)){          //Terminacion normal
            ok = true;
          }
          strcpy(pCad1,aux.c_str());
          if (e04(pCad1,6655)) {   //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
```

```
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,99)) {  //las Areas para leer y escribir son
malas
          ok = false;
          goto salir;
        }
        if(this->e21()){         //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
       }
//*****************************************************************************
*****************************
      bool T_Comandos::RC (char* pCad1){        //%%%%%%%
LECTURA DE PV 337
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }

        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,511)) {  //las Areas para leer y escribir son
malas
          ok = false;
          goto salir;
        }
        if(e21()){         //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
       }
//*****************************************************************************
        ***********************************
              bool      T_Comandos::RG      (char*      pCad1){
        //%%%%%%% LECTURA DE ESTADO DE TC 337
      AnsiString aux;
      aux = pCad1;
      jj=aux.Length();
      if (e00(pCad1)){        //Terminacion normal
        ok = true;
      }

      strcpy(pCad1,aux.c_str());
      if (e04(pCad1,6655)) {  //Direccion excedida
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e18(pCad1)) {    // Error en la longitud de la trama
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e15(pCad1,0,511)) {  //las Areas para leer y escribir son malas
        ok = false;
        goto salir;
      }
      if(e21()){         //Error de CPU
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e13(pCad1)) {    //El FCS es falso
        ok = false;
        goto salir;
      }
      salir:
      strcpy(pCad1,aux.c_str());
      trama(pCad1); // obtener dato
      return ok;
     }
//*****************************************************************************
*****************
      bool T_Comandos::RD (char* pCad1){      //%%%%%%% LECTURA
DE AREA DE DM 338
      AnsiString aux;
      aux = pCad1;
      jj=aux.Length();
      if (e00(pCad1)){        //Terminacion normal
        ok = true;
      }
      strcpy(pCad1,aux.c_str());
      if (e04(pCad1,6655)) {  //Direccion excedida
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e18(pCad1)) {    // Error en la longitud de la trama
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
        ok = false;
        goto salir;
      }
      strcpy(pCad1,aux.c_str());
      if (e15(pCad1,0,6655)) {  //las Areas para leer y escribir son malas
        ok = false;
        goto salir;
      }
      if(this->e21()){         //Error de CPU
        ok = false;
```

```
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************************************************
**************************
      bool T_Comandos::RJ (char* pCad1){        //%%%%%%%
LECTURA DE AREA DE AR 338
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,27)) {  //las Areas para leer y escribir son
malas
          ok = false;
          goto salir;
        }
        if(e21()){        //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************escritura**************************
**************************************
//*****************************************************************************
**********************************
      bool T_Comandos::WR (char* pCad1){        //%%%%%%%
ESCRITURA DE AREA IR/SR 339
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e03(pCad1,252,255)) {  //las Areas para leer o escribir
son malas
          ok = false;
        goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,252)) {  //las Areas para leer o escribir son malas
          ok = false;
          goto salir;
        }
        if(e21()){        //Error de CPU
          ok = false;
          goto salir;
        }
        if(e23()){        //Error de memoria protegida DIP
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************************************************
*****************
      bool T_Comandos::WL (char* pCad1){              //%%%%%%%
ESCRITURA DE AREA LR 339
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        /* if (e03(pCad1,252,255)) {  //las Areas para leer o escribir son
malas
          ok = false;
          goto salir;
          } */

        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
```

```
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,63)) {  //las Areas para leer o escribir
son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {     //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*********************************************************************
**********************************
      bool      T_Comandos::WH      (char*      pCad1){
//%%%%%%% ESCRITURA DE AREA HR 340
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {   // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        /*if  (e03(pCad1,252,255))  {   //las  Areas  para  leer  o
escribir son malas
            ok = false;
            goto salir;
         } */
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {   //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {     // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {     // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,99)) {  //las Areas para leer o escribir
son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {     //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*********************************************************************
**********************************
      bool   T_Comandos::WC   (char*   pCad1){          //%%%%%%%
ESCRITURA DE PV 340
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {   // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        /*if (e03(pCad1,252,255)) {  //las Areas para leer o escribir son malas
           ok = false;
           goto salir;
         } */
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {   //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {     // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {     // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,511)) {  //las Areas para leer o escribir son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {     //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*********************************************************************
**********************************
      bool   T_Comandos::WG   (char*   pCad1){          //%%%%%%%
ESCRITURA DE ESTADO DE TC 341
        AnsiString aux;
        aux = pCad1;
        AnsiString aux_WG=aux;
        unsigned long  conta,longitud=aux_WG.Length(),longitud1;

        int g=1,gg=0;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {   // No ejecutable en modo run
          ok = false;
          goto salir;
        }
```

```
        strcpy(pCad1,aux.c_str());
        /*if (e03(pCad1,252,255)) {   //las Areas para leer o
escribir son malas
          ok = false;
          goto salir;
         } */
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {   //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,511)) {  //las Areas para leer o escribir
son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        aux_WG=aux_WG.SubString(10,longitud);
        longitud=aux_WG.Length();
        aux_WG=aux_WG.SubString(1,longitud-4);
        conta = aux_WG.Length();
        gg=conta/4;
        do{
           aux_WG.Delete(g,3);
           g++;
           gg--;
        }while(gg>0);
        aux_WG.Insert(aux.SubString(1,9),1);
        longitud=aux.Length();
        longitud1=aux_WG.Length();
        aux_WG.Insert(aux.SubString(longitud-
3,2),longitud1+1 );
        aux_WG=aux_WG+"*\r";
        strcpy(pCad1,aux_WG.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
       }
//**********************************************************************
************************************
       bool       T_Comandos::WD       (char*       pCad1){
//%%%%%%% ESCRITURA DE AREA DM 342
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        /*if (e03(pCad1,252,255)) {   //las Areas para leer o
escribir son malas
          ok = false;
          goto salir;
         } */
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {   //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e15(pCad1,0,6143)) {  //las Areas para leer o escribir son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        if(e23()){          //Error de memoria protegida DIP
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
       }
//**********************************************************************************
******************
       bool   T_Comandos::WJ   (char*   pCad1){              //%%%%%%%
ESCRITURA DE AREA AR 342
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        /*if (e03(pCad1,252,255)) {  //las Areas para leer o escribir son malas
          ok = false;
          goto salir;
         } */
        strcpy(pCad1,aux.c_str());
        if (e04(pCad1,6655)) {   //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
```

```
        if (e15(pCad1,0,27)) {  //las Areas para leer o escribir
son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*********************************************************************
*********************************
      bool      T_Comandos::Ralmu      (char*      pCad1){
//%%%%%%% LECTURA SV 1 343
        AnsiString aux,aux1;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        aux1=aux;
        aux1=aux1.Delete(6,4);
        strcpy(pCad1,aux1.c_str());
        if (e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }


if((aux.SubString(6,3)=="TIM")||(aux.SubString(6,3)=="CNT"))
aux=aux;
        else{ aux=aux+"e",jj=aux.Length();} //forzamos error
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }

        aux1=aux1.Insert("0001",10);
        strcpy(pCad1,aux1.c_str());
        if (e15(pCad1,0,511)) {  //las Areas para leer y escribir
son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }

        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
```

```
//*********************************************************************************
*****************
      bool   T_Comandos::Walmu(char*   pCad1){          //%%%%%%%
CAMBIAR SV 1 346
        AnsiString aux,aux1;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }

        aux1=aux;
          aux1=aux1.Delete(6,4);
          strcpy(pCad1,aux1.c_str());
          if (e04(pCad1,6655)) {  //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }

        if((aux.SubString(6,3)=="TIM")||(aux.SubString(6,3)=="CNT"))
aux=aux;
        else{ aux=aux+"e",jj=aux.Length();} //forzamos error
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }

        aux1=aux1.Insert("0001",10);
        strcpy(pCad1,aux1.c_str());
        if (e15(pCad1,0,511)) {  //las Areas para leer y escribir son malas
          ok = false;
          goto salir;
        }
        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        if(e23()){          //Error de memoria protegida DIP
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }


//*********************************************************************************
*****************
      bool T_Comandos::MS (char* pCad1){  //%%%%%%% LECTURA DE
ESTADO 348
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){        //Terminacion normal
          ok = true;
        }
```

```
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-9)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }


        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************************************************
**************************
      bool  T_Comandos::SC  (char*  pCad1){    //%%%%%%%
ESCRITURA DE ESTADO 349
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }


        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-7)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }


        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************************************************
**************************
      bool  T_Comandos::MF  (char*  pCad1){    //%%%%%%%
LECTURA DE ERROR 350
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e14(pCad1,jj-7)) {    // %%%%% Error de formato
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
      }
//*****************************************************************************
**************************
      bool T_Comandos::KS (char* pCad1){  //%%%%%%%% FORZADO A
ON 351

      }
//*****************************************************************************
**************************
      bool T_Comandos::KR (char* pCad1){  //%%%%%%%% FORZADO A
OFF 352

      }
//*****************************************************************************
**************************
      bool T_Comandos::FK (char* pCad1){  //%%%%%%%% FORZADOS
MULTIPLES A ON/OFF 353

      }
//*****************************************************************************
**************************
      bool  T_Comandos::KC  (char*  pCad1){        //%%%%%%%%
CANCELACION DE FORZADOS 354

      }
//*****************************************************************************
**************************
      bool T_Comandos::MM (char* pCad1){  //%%%%%%%% LECTURA
DE MODELO DE PLC 354

      }
//*****************************************************************************
**************************
      bool T_Comandos::TS (char* pCad1){  //%%%%%%%% PRUEBA DE
COMUNICACIONES 355
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (e00(pCad1)){          //Terminacion normal
          ok = true;
        }
        strcpy(pCad1,aux.c_str());
        if (e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }

        if(e21()){          //Error de CPU
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
```

```
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
       }
//*****************************************************************************
*****************************
       bool  T_Comandos::QQ  (char*  pCad1){     //%%%%%%%
COMANDO GENERICO 356

       }
//*****************************************************************************
*****************************
       bool  T_Comandos::XZ  (char*  pCad1){     //%%%%%%%
ABORTAR (sólo comando) 358

       }
//*****************************************************************************
*****************************
       bool T_Comandos::aste_aste  (char* pCad1){ //%%%%%%%
INICIALIZAR (sólo comando) 358

       }
//*****************************************************************************
*****************************
       bool  T_Comandos::IC(char*  pCad1){       //%%%%%%%
COMANDO INDEFINIDO

       }


//--------------Funciones-----------------------------------------------------
//----------------------------------------------------------------------------
//-------Cambiar valor a decimal de un hexadecimal----------------------
----------
       unsigned short T_Comandos::valor_decimal(char* pCad1)  {
         AnsiString recibido=pCad1;
         unsigned int iPos1=0;
         char *aux,*aux1,recibi[4];
         aux=&recibi[0];
         strcpy(recibi,recibido.c_str());
         for(int ii=0; ii<4;ii++){
           aux1=&recibi[ii] ;
           iPos1=iPos1+ conv(*aux1)*pow(16,3-ii);    //valor a escribir
en decimal
         }
         return iPos1;
       }


//--------------Extrae Trama --------------------------

       void T_Comandos::trama(char* pCad2){
         AnsiString aux;
         aux=pCad2;
         int j;
         j= aux.Length() ;
         j=j-9;
         aux=aux.SubString(6,j);
         strcpy(pCad2,aux.c_str());
       }
```

6.1.5 TCOMANDOS

TCOMANDOS.H

```
//---------------------------------------------------------------------------
#ifndef TCOMANDOSH
#define TCOMANDOSH
//---------------------------------------------------------------------------
#include <windows.h>

/*****************************************************************************
************************  PLC  ****+*************************************
*****************************************************************************/
class T_plc{


   private:
      int estado ;    // 3=run,2=monitor,0=program
      bool power;
      bool  e_CPU;
      char* C_error;
      char* modelo;
      bool pin_DIP;  //Proteccion contra escritura

      unsigned  short  IR[255];
      unsigned  short LR [63];
      unsigned  short HR [99];
      unsigned  short PV [511];
      unsigned  short TC [511];
      unsigned  short DM [6655];
      unsigned  short AR [27];
      unsigned  short SV1 [511];
      unsigned  short SV2 [511];
      unsigned  short SV3 [511];
   public:
       //constructores
       T_plc(void );  //sin parametros
       T_plc(int ,bool ,bool  ,char* );  //con parametros
       //Destructor
       ~T_plc();
       void  Set_estado(int);       // 0 run
                                    //2 monitor
                                    //3 programa
       int  Get_estado  ();        //enviar estado plc

      //------   Error del PLC --------
       void Set_error(bool);
       bool Get_error();

       //------  Codigo de Finalizacion ------

       void Set_ComError(char*);
       char* Get_comError(void );

       //--------- Proteger escritura de memoria de usuario----
       void Set_DIP(bool);
       bool Get_DIP(void);


      //----------Memorias----------------
       void Set_IR(unsigned short, unsigned short);
       void Set_LR(unsigned short, unsigned short);
       void Set_HR(unsigned short, unsigned short);
       void Set_PV(unsigned int, unsigned short*);
       void Set_TC(unsigned int, unsigned short*);
       void Set_DM(unsigned short, unsigned short);
       void Set_AR(unsigned short, unsigned short);
       void Set_SV1(unsigned int, unsigned short*);
       void Set_SV2(unsigned int, unsigned short*);
       void Set_SV3(unsigned int, unsigned short*);

       unsigned short Get_IR(unsigned short);
       unsigned short Get_LR(unsigned short);
       unsigned short Get_HR(unsigned short);
       unsigned short Get_PV(void);
       unsigned short Get_TC(void);
       unsigned short Get_DM(unsigned short);
       unsigned short Get_AR(unsigned short);
       unsigned short Get_SV1(void);
       unsigned short Get_SV2(void);
       unsigned short Get_SV3(void);

       int conv(char ) ; //---cambiar caracter a valor entero----------------------

};

/*****************************************************************************
************************  ERRORES  ***********************************
*****************************************************************************/

class T_Error:public T_plc{

    private:
       char* e_Comando;
    public:
       //constructores
       T_Error(void); //sin  parametros
       T_Error(char* );  //con parametros
       //Destructor
       ~T_Error();

       bool e00(char*);
       bool e01(char*);
       bool e02(char*);
       bool e03(char* ,unsigned int, unsigned int);
       bool e04(char*,unsigned int);
       bool e0B(char*);
       bool e13(char*);

       bool e14(char*,unsigned int );
       bool e15(char*, unsigned int , unsigned int );
       bool e16(char*);
       bool e18(char*);
       bool e19(char*);
       bool e21(void);

       bool e23(char*);
       bool eA3(char*);
       bool eA4(char*);
       bool eA5(char*);
       bool eA8(char*);
       unsigned long rango(char*,bool) ;
       void FCS(char* pCad1 )  ;
};

/*****************************************************************************
************************  COMANDOS  ***********************************
*****************************************************************************/
// Declaración de la clase
class T_Comandos:public T_Error{

  private:
      char C_prueba[32];
      int estado1;//Estado del PLC
```

```
      bool ok ;
      unsigned int jj;  //para saber la longitud del comando recibido si es
ok
   public:
        //constructores
        T_Comandos(char* );  //con parametros

        //Destructor
        ~T_Comandos();


      bool RR (char* pCad1);  //     LECTURA DE AREA IR/SR 335
      bool RL (char* pCad1);  //   LECTURA DE AREA LR 336
      bool RH (char* pCad1);  //   LECTURA DE AREA HR 336
      bool RC (char* pCad1);  //  LECTURA DE PV 337
      bool RG (char* pCad1);  //  LECTURA DE ESTADO DE TC 337
      bool RD (char* pCad1);  //   LECTURA DE AREA DE DM 338
      bool RJ (char* pCad1);  //  LECTURA DE AREA DE AR 338
      bool WR (char* pCad1);  //   ESCRITURA DE AREA IR/SR 339
      bool WL (char* pCad1);  //  ESCRITURA DE AREA LR 339
      bool WH (char* pCad1);  //   ESCRITURA DE AREA HR 340
      bool WC (char* pCad1);  //   ESCRITURA DE PV 340
      bool WG (char* pCad1);  //  ESCRITURA DE ESTADO DE TC 341
      bool WD (char* pCad1);  //  ESCRITURA DE AREA DM 342
      bool WJ (char* pCad1); //  ESCRITURA DE AREA AR 342
      bool Ralmu (char* pCad1);  //  LECTURA SV 1 343
      bool Rdolar (char* pCad1);  //   LECTURA SV 2 344
      bool Rporce (char* pCad1);  //  LECTURA SV 3 (Sólo PLCs CQM1)
345
      bool Walmu (char* pCad1);  //   CAMBIAR SV 1 346
      bool Wdolar (char* pCad1); //   CAMBIAR SV 2 346
      bool  Wporce  (char*  pCad1);   //     CAMBIAR  SV  3  (Sólo  PLCs
CQM1) 347
      bool MS (char* pCad1);  //  LECTURA DE ESTADO 348
      bool SC (char* pCad1);  //   ESCRITURA DE ESTADO 349
      bool MF (char* pCad1);  //   LECTURA DE ERROR 350
      bool KS (char* pCad1);  //  FORZADO A ON 351
      bool KR (char* pCad1);  //   FORZADO A OFF 352
      bool  FK  (char*  pCad1);   //   FORZADOS  MULTIPLES  A  ON/OFF
353
      bool KC (char* pCad1);  //   CANCELACION DE FORZADOS 354
      bool MM (char* pCad1);  //  LECTURA DE MODELO DE PLC 354
      bool TS (char* pCad1);  //   PRUEBA DE COMUNICACIONES 355
      bool RP (char* pCad1); //   LECTURA DE PROGRAMA 355
      bool WP (char* pCad1);  //   ESCRITURA DE PROGRAMA 356
      bool QQ (char* pCad1);  //  COMANDO COMPUESTO 356
      bool XZ (char* pCad1); //  ABORTAR (sólo comando) 358
      bool aste_aste (char* pCad1);  //   INICIALIZAR (sólo comando) 358
      bool IC(char* pCad1);   //  Comando indefinido

      unsigned  short  valor_decimal(char*  pCad1);     //cambiar  valor  a
decimal

      void trama(char*);
};
#endif
```

TCOMANDOS.cpp

```
include "TCOMANDOS.H"
#include <Windows.h>
#include <math.h>
#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h >


#include <stdio.h>




*/*********************PLC*************************************************
*****************************************************************************/
//--------sin parametros
  T_plc::T_plc(void){
  }
 //--------parametros
  T_plc::T_plc(int      estado1,bool      v_Power,bool      v_error,char*
v_Modelo){
       unsigned int dato=0;
       int i,dato1;
       AnsiString memoria;
      char c[4];
      estado = estado1;
      power=v_Power;
      e_CPU=v_error;
      modelo=v_Modelo;
      memoria = "ABCF";
      strcpy(c,memoria.c_str());
       for(i=0;i<4;i++){
         dato=dato+conv(c[i])*pow(16,3-i);
       }
       IR[i-4]=dato;
       LR[i-4]=dato;
       HR[i-4]=dato;
       PV[i-4]=dato;
       TC[i-4]=dato;
       DM[i-4]=dato;
       AR[i-4]=dato;
       SV1[i-4]=dato;
       SV2[i-4]=dato;
       SV3[i-4]=dato;



  }
//-------------Destructor----------------------------------
   T_plc::~T_plc(){ }

//--------------- Metodos-----------------------------------------------
//---------------------------------------------------------------------------

       void    T_plc::Set_estado(int  aux){           //3  run,  2=monitor,
0=programa
           estado = aux;
       }


       int  T_plc::Get_estado (){           //enviar estado plc
            return estado;

       }

       void T_plc::Set_error(bool v_Error){
            e_CPU=v_Error;
       }
       bool T_plc::Get_error(){
         return e_CPU;
       }

       //------  Error de comando ------

       void T_plc::Set_ComError(char* E_comando){
          C_error=E_comando;
       }
       char* T_plc::Get_comError(void){
          return C_error;

       }
       //--------- Proteger escritura de memoria de usuario----
       void T_plc::Set_DIP(bool b_DIP){
         pin_DIP=b_DIP;
       }
       bool T_plc::Get_DIP(void){
          return pin_DIP;
       }
 //--------------------- Escritura de memorias----------------------------------
--------
       void T_plc::Set_IR(unsigned short canal, unsigned short dato){
         unsigned short* p_dato;
         p_dato=&IR[canal];
         *p_dato=dato ;
        }
       void T_plc::Set_LR(unsigned short canal, unsigned short dato){
         unsigned short* p_dato;
         p_dato=&LR[canal];
         *p_dato=dato ;
       }
       void T_plc::Set_HR(unsigned short canal, unsigned short dato){
         unsigned short* p_dato;
         p_dato=&HR[canal];
         *p_dato=dato ;
       }
       void T_plc::Set_PV(unsigned int canal, unsigned short* dato){

       }
       void T_plc::Set_TC(unsigned int canal, unsigned short* dato){

       }
       void T_plc::Set_DM(unsigned short canal, unsigned short dato){
         unsigned short* p_dato;
         p_dato=&DM[canal];
         *p_dato=dato ;
       }
       void T_plc::Set_AR(unsigned short canal, unsigned short dato){
         unsigned short* p_dato;
         p_dato=&AR[canal];
         *p_dato=dato ;
       }
       void T_plc::Set_SV1(unsigned int canal, unsigned short* dato){

       }
       void T_plc::Set_SV2(unsigned int canal, unsigned short* dato){

       }
       void T_plc::Set_SV3(unsigned int canal, unsigned short* dato){

       }

 //-------- Lectura de memorias ----------------------------------------
       unsigned short T_plc::Get_IR(unsigned short v_indx){

           return IR[v_indx];
       }
       unsigned short T_plc::Get_LR(unsigned short v_indx){
           return LR[v_indx];
       }
       unsigned short T_plc::Get_HR(unsigned short v_indx){
           return HR[v_indx];
       }
```

```
unsigned short T_plc::Get_PV(void){

}
unsigned short T_plc::Get_TC(void){

}
unsigned short T_plc::Get_DM(unsigned short v_indx){
    return DM[v_indx];
}
unsigned short T_plc::Get_AR(unsigned short v_indx){
    return AR[v_indx];
}
unsigned short T_plc::Get_SV1(void){

}
unsigned short T_plc::Get_SV2(void){

}
unsigned short T_plc::Get_SV3(void){

}
//----------cambiar caracter a valor entero----------
int T_plc:: conv(char c) {
int num=c;
num=num-48;   //asci del 0
if (num<=9)
return num;
if (c=='A')
num = 10;
if (c=='B')
num = 11;
if (c=='C')
num = 12;
if (c=='D')
num = 13;
if (c=='E')
num = 14;
if (c=='F')
num = 15;
return num;
}


*/**********Errores**********
*****************************************************************************/
//constructores
    T_Error::T_Error(void ){   //sin parametros

    }
    T_Error::T_Error(char*
pCad3):T_plc(0,true,false,"CQM1"){//T_plc(int          estado1,bool
v_Power,bool  Verror,char* v_Modelo){
     e_Comando=pCad3;
    }
    //Destructor
    T_Error::~T_Error(){}
//---------- Metodos----------
//-----------------------------------------------------------------------
    bool   T_Error::e00(char*   pCad3){        //   %%%%%
Finalizacion normal

    bool e00=true;
    pCad3 ="00";
    this->Set_ComError(pCad3);
    return e00;

   }
   bool  T_Error::e01(char*  pCad3){        //  %%%%%  No
ejecutable en modo run

    bool e01;
    int aux=this->Get_estado();
    if (aux == 3)  e01 = true  ;
    pCad3="01" ;
    if(e01==true)  this->Set_ComError(pCad3);
    return e01;
   }
                bool  T_Error::e02(char*  pCad3){        //  %%%%%  No
        ejecutable en modo monitor
    bool e02;
    int aux=this->Get_estado();
    if (aux == 2) e02 = true ;
    pCad3="02" ;
    if(e02==true)  this->Set_ComError(pCad3);
    return e02;
   }
   bool  T_Error::e03(char*  pCad3,  unsigned   int  D_inicio,  unsigned int
D_fin ) {  //%%%%% Proteccion de escritura UM
     bool e03 = false;
     AnsiString aux = pCad3;
     unsigned int  aux1;
     aux=aux.SubString(6,4);
     aux1=StrToInt(aux);
     if ((aux1<D_fin)&&(aux1>D_inicio)) e03 = true;
     pCad3="03" ;
     if(e03==true)  this->Set_ComError(pCad3);
     return e03;
   }
   bool  T_Error::e04(char*  pCad3,  unsigned  int  Max_direccion){     //
%%%%% Direccion excedida

     AnsiString aux = pCad3;
     unsigned long  aux1;
     bool e04 = false;
     aux=aux.SubString(6,4);
     aux1=StrToInt(aux);
     if (aux1 >  Max_direccion) e04 = true;
     pCad3="04" ;
     if(e04==true)  this->Set_ComError(pCad3);
     return e04;
   }
   bool T_Error::e0B(char* pCad3){          // %%%%% No ejecutable en
modo programa
     bool e0B;
     int aux=this->Get_estado();
     if (aux == 0)  e0B = true ;
     pCad3="0B" ;
     if(e0B==true) this->Set_ComError(pCad3);
     return e0B;
    }
   bool T_Error::e13(char* pCad3){        // %%%%% El FCS es falso

     bool e13 = false;
     AnsiString aux,aux1;
     aux = pCad3;
     aux1=aux;    //copia del comando
     int j= aux.Length() ;
     j=j-4;
     aux=aux.SubString(1,j);
     strcpy(pCad3,aux.c_str());
     FCS(pCad3);   //Obtengo el FCS correcto
     aux=aux1;
     j= aux.Length() ;
     j=j-3;
     aux=aux.SubString(j,2);
     if ( aux!=pCad3)  e13 = true;  // compara el FCS que saco del dato
con el FCS que me envian
     pCad3="13" ;
     if(e13==true)  Set_ComError(pCad3);
     return e13;
    }
   bool T_Error::e14(char* pCad3,unsigned int longitud){    // %%%%%
Error de formato
     unsigned long  aux1=longitud;
     short resto;
     bool e14=true;
     pCad3="14" ;
     resto=aux1%4;
     if(resto==0) {
       aux1=1;
       e14=false;
     }
     else aux1=0;
     if((e14==true)||(aux1==0))  this->Set_ComError(pCad3);
```

```
          return e14;
        }
       bool T_Error::e15(char* pCad3, unsigned int D_min, unsigned
int D_max){  // %%%%% las Areas para leer y escribir son malas
          AnsiString aux = pCad3;
          unsigned long  aux1;
          bool e15 = false;
          aux=aux.SubString(6,4);
          aux1=StrToInt(aux);
          if ((aux1>D_max)||(aux1<D_min))e15 = true;
          pCad3="15" ;
          if(e15==true)  this->Set_ComError(pCad3);
          return e15;
       }
       bool T_Error::e16(char* pCad3){          // %%%%% comando
no soportado(Leer SV, etc ->mal)

       }

       bool T_Error::e18(char* pCad3){        // %%%%% Error en la
longitud de la trama

          int Lon_max=131;
          bool e18 = false;
          AnsiString aux = pCad3;
          int j = aux.Length();
          if (j>Lon_max)  e18=true;
          pCad3="18" ;
          if(e18==true)  this->Set_ComError(pCad3);
          return e18;
         }

       bool  T_Error::e19(char*pCad3  ){              // %%%%% No
ejecutable

       }

       bool T_Error::e21(void){            // %%%%% Error de CPU
          bool e21=this->Get_error();
          if (e21==true)  this->Set_ComError("21");
          return e21;
       }
       bool T_Error::e23(char* pCad3){    // %%%%% memoria de
usuario protegida contra escritura  y lectura
          bool e23=this->Get_DIP();
          if (e23==true)  this->Set_ComError("23");
          return e23;
       }
        //----------------Errores con cadenas multimples ----------
----------
       bool  T_Error::eA3(char*  pCad3){    // %%%%% Abortado
debido a error de FCS en la transmision de dato

       }
       bool  T_Error::eA4(char*  pCad3){  // %%%%% cancelado
debido a un error de formato en la transmision

       }
       bool  T_Error::eA5(char*  pCad3){   // %%%%% cancelado
debido a un error de numero  de entrada de datos en transmision

       }
       bool T_Error::eA8(char*pCad3){ // %%%%% cancelado
debido a error de longitud de trama en transmision de datos

       }
 //--------------Funciones-----------------------------------------
-----
 //-------------------------------------------------------------------
      unsigned       long       T_Error::rango(char*pCad3,bool
npalabra){
          int j=6;
          if (npalabra) j=10;
          AnsiString aux=pCad3;
          aux=aux.SubString(j,4);
          strcpy(pCad3,aux.c_str());
          unsigned long  a= StrToInt(pCad3);
                  return a;

                }

       void T_Error::FCS(char* pCad1){
       AnsiString cadena;
       cadena = pCad1;
       char aux=0;
       int i;
       int j = cadena.Length();
       for (i=1;i<=j;i++){
         aux=aux^cadena[i];
       }

       cadena=IntToHex(aux,2);
       strcpy(pCad1,cadena.c_str());
      }

//-------------------------Comandos---------------------------------
//*************************************************************************************
*

//-----------Constructor-----------------------------------
  T_Comandos::T_Comandos(char* pCad1):T_Error(pCad1){
       strcpy(C_prueba, pCad1);
       ok = false;

   }
//-------------Destructor----------------------------------------
   T_Comandos::~T_Comandos(){ }

 //---------------- Metodos-----------------------------------------------
 //-------------------------------lectura-----------------------------------------
       bool T_Comandos::RR(char* pCad1){       //%%%%%%% LECTURA
DE AREA IR/SR 335
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (this->e00(pCad1)){        //Terminacion normal
            ok = true;
          }

          strcpy(pCad1,aux.c_str());

          if (this->e04(pCad1,6655)) {  //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e15(pCad1,0,255)) {  //las Areas para leer y escribir son
malas
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){        //Error de CPU
           ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e13(pCad1)) {    //El FCS es falso
            ok = false;
            goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
```

```
        }
//*****************************************************************************
**************************
      bool  T_Comandos::RL  (char*  pCad1){          //%%%%%%%
LECTURA DE AREA LR 336
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());

           if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
           }


           strcpy(pCad1,aux.c_str());
           if (this->e15(pCad1,0,64)) {  //las Areas para leer y escribir
son malas
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
           }
           if(this->e21()){         //Error de CPU
            ok = false;
            goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e13(pCad1)) {    //El FCS es falso
             ok = false;
             goto salir;
           }
           salir:
           strcpy(pCad1,aux.c_str());
           trama(pCad1); // obtener dato
           return ok;
       }
//*****************************************************************************
**************************
      bool  T_Comandos::RH  (char*  pCad1){          //%%%%%%%
LECTURA DE AREA HR 336
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());

           if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
           }
}


strcpy(pCad1,aux.c_str());
if (this->e15(pCad1,0,99)) {  //las Areas para leer y escribir son malas
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
           }
           if(this->e21()){         //Error de CPU
            ok = false;
            goto salir;
           }
           strcpy(pCad1,aux.c_str());

           if (this->e13(pCad1)) {    //El FCS es falso
             ok = false;
             goto salir;
           }
           salir:
           strcpy(pCad1,aux.c_str());
           trama(pCad1); // obtener dato
           return ok;
       }
//*****************************************************************************
*****************
      bool T_Comandos::RC (char* pCad1){     //%%%%%%% LECTURA DE
PV 337

      }
//*****************************************************************************
*****************
      bool T_Comandos::RG (char* pCad1){     //%%%%%%% LECTURA DE
ESTADO DE TC 337

      }
//*****************************************************************************
*****************
      bool T_Comandos::RD (char* pCad1){     //%%%%%%% LECTURA DE
AREA DE DM 338
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());

           if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
           }


           strcpy(pCad1,aux.c_str());
           if (this->e15(pCad1,0,6655)) {  //las Areas para leer y escribir son
malas
             ok = false;
             goto salir;
           }
           strcpy(pCad1,aux.c_str());
           if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
           }
           if(this->e21()){         //Error de CPU
                                    ok = false;
                                    goto salir;
           }
           strcpy(pCad1,aux.c_str());

           if (this->e13(pCad1)) {    //El FCS es falso
```

```
            ok = false;
            goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
      }
//*****************************************************************************
**************************
      bool  T_Comandos::RJ  (char*  pCad1){          //%%%%%%%
LECTURA DE AREA DE AR 338
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());

          if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
          }

           strcpy(pCad1,aux.c_str());
           if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
          }

          strcpy(pCad1,aux.c_str());
           if (this->e15(pCad1,0,27)) {  //las Areas para leer y escribir
son malas
             ok = false;
             goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){         //Error de CPU
           ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());

          if (this->e13(pCad1)) {    //El FCS es falso
             ok = false;
             goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
      }
//***************************************escritura*****************************
**************************************
//*****************************************************************************
**************************************
      bool  T_Comandos::WR  (char*  pCad1){          //%%%%%%%
ESCRITURA DE AREA IR/SR 339
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());
           if (this->e01(pCad1)) {   // No ejecutable en modo run
             ok = false;
             goto salir;
          }

           strcpy(pCad1,aux.c_str());
           if (this->e03(pCad1,252,255)) {  //las Areas para leer o escribir son malas
             ok = false;
             goto salir;
           }

          strcpy(pCad1,aux.c_str());
          if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
          }

           strcpy(pCad1,aux.c_str());
           if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
          }

           strcpy(pCad1,aux.c_str());
           if (this->e15(pCad1,0,252)) {  //las Areas para leer o escribir son malas
             ok = false;
             goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){         //Error de CPU
           ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e13(pCad1)) {    //El FCS es falso
             ok = false;
             goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
      }
//*****************************************************************************
***********
      bool T_Comandos::WL (char* pCad1){       //%%%%%%% ESCRITURA DE
AREA LR 339
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
           if (this->e00(pCad1)){         //Terminacion normal
             ok = true;
           }

           strcpy(pCad1,aux.c_str());
          if (this->e01(pCad1)) {   // No ejecutable en modo run
             ok = false;
             goto salir;
          }

           strcpy(pCad1,aux.c_str());
          /*  if (this->e03(pCad1,252,255)) {   //las Areas para leer o escribir son
malas
             ok = false;
             goto salir;
           } */

          strcpy(pCad1,aux.c_str());
          if (this->e04(pCad1,6655)) {   //Direccion excedida
             ok = false;
             goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
             ok = false;
             goto salir;
          }
```

```
          strcpy(pCad1,aux.c_str());
          if (this->e15(pCad1,0,63)) {  //las Areas para leer o escribir son
malas
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){         //Error de CPU
           ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e13(pCad1)) {    //El FCS es falso
            ok = false;
            goto salir;
          }

          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
      }
//*************************************************************************
**********************
      bool  T_Comandos::WH  (char*  pCad1){            //%%%%%%%
ESCRITURA DE AREA HR 340
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (this->e00(pCad1)){         //Terminacion normal
            ok = true;
          }

          strcpy(pCad1,aux.c_str());
          if (this->e01(pCad1)) {  // No ejecutable en modo run
            ok = false;
            goto salir;
          }

          strcpy(pCad1,aux.c_str());
          /*if (this->e03(pCad1,252,255)) {  //las Areas para leer o escribir
son malas
            ok = false;
            goto salir;
          } */

          strcpy(pCad1,aux.c_str());
          if (this->e04(pCad1,6655)) {  //Direccion excedida
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
            ok = false;
            goto salir;
          }

          strcpy(pCad1,aux.c_str());
          if (this->e15(pCad1,0,99)) {  //las Areas para leer o escribir son
malas
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){         //Error de CPU
           ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e13(pCad1)) {    //El FCS es falso
            ok = false;
            goto salir;
          }
          salir:
          strcpy(pCad1,aux.c_str());
          trama(pCad1); // obtener dato
          return ok;
      }
//*************************************************************************
**********************
      bool T_Comandos::WC (char* pCad1){      //%%%%%%% ESCRITURA
DE PV 340

      }
//*************************************************************************
**********************
      bool T_Comandos::WG (char* pCad1){      //%%%%%%% ESCRITURA
DE ESTADO DE TC 341

      }
//*************************************************************************
**********************
      bool T_Comandos::WD (char* pCad1){      //%%%%%%% ESCRITURA
DE AREA DM 342
          AnsiString aux;
          aux = pCad1;
          jj=aux.Length();
          if (this->e00(pCad1)){         //Terminacion normal
            ok = true;
          }

          strcpy(pCad1,aux.c_str());
          if (this->e01(pCad1)) {  // No ejecutable en modo run
            ok = false;
            goto salir;
          }

          strcpy(pCad1,aux.c_str());
          /*if (this->e03(pCad1,252,255)) {  //las Areas para leer o escribir son
malas
            ok = false;
            goto salir;
          } */

          strcpy(pCad1,aux.c_str());
          if (this->e04(pCad1,6655)) {  //Direccion excedida
            ok = false;
            goto salir;
          }

          strcpy(pCad1,aux.c_str());
          if (this->e14(pCad1,jj-9)) {    // %%%%% Error de formato
            ok = false;
            goto salir;
          }

          strcpy(pCad1,aux.c_str());
          if (this->e15(pCad1,0,6143)) {  //las Areas para leer o escribir son
malas
            ok = false;
            goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e18(pCad1)) {    // Error en la longitud de la trama
            ok = false;
            goto salir;
          }
          if(this->e21()){         //Error de CPU
                        ok = false;
           goto salir;
          }
          strcpy(pCad1,aux.c_str());
          if (this->e13(pCad1)) {    //El FCS es falso
            ok = false;
            goto salir;
```

```
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
    }
//*****************************************************************************
*************************
    bool T_Comandos::WJ (char* pCad1){        //%%%%%%%
ESCRITURA DE AREA AR 342
        AnsiString aux;
        aux = pCad1;
        jj=aux.Length();
        if (this->e00(pCad1)){        //Terminacion normal
          ok = true;
        }

        strcpy(pCad1,aux.c_str());
        if (this->e01(pCad1)) {  // No ejecutable en modo run
          ok = false;
          goto salir;
        }

        strcpy(pCad1,aux.c_str());
        /*if (this->e03(pCad1,252,255)) {  //las Areas para leer o
escribir son malas
          ok = false;
          goto salir;
        } */

        strcpy(pCad1,aux.c_str());
        if (this->e04(pCad1,6655)) {  //Direccion excedida
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (this->e14(pCad1,jj-9)) {   // %%%%% Error de formato
          ok = false;
          goto salir;
        }

        strcpy(pCad1,aux.c_str());
        if (this->e15(pCad1,0,27)) {  //las Areas para leer o escribir
son malas
          ok = false;
          goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (this->e18(pCad1)) {    // Error en la longitud de la trama
          ok = false;
          goto salir;
        }
        if(this->e21()){        //Error de CPU
         ok = false;
         goto salir;
        }
        strcpy(pCad1,aux.c_str());
        if (this->e13(pCad1)) {    //El FCS es falso
          ok = false;
          goto salir;
        }
        salir:
        strcpy(pCad1,aux.c_str());
        trama(pCad1); // obtener dato
        return ok;
    }
//*****************************************************************************
*************************
    bool T_Comandos::Ralmu (char* pCad1){    //%%%%%%%
LECTURA SV 1 343

    }
//*****************************************************************************
*************************
    bool T_Comandos::Rdolar (char* pCad1){    //%%%%%%%
LECTURA SV 2 344

    }
//*****************************************************************************
*************
    bool T_Comandos::Rporce (char* pCad1){  //%%%%%%% LECTURA SV
3 (Sólo PLCs CQM1) 345

    }
//*****************************************************************************
*************
    bool T_Comandos::Walmu(char* pCad1){    //%%%%%%% CAMBIAR SV
1 346

    }
//*****************************************************************************
*************
    bool T_Comandos::Wdolar (char* pCad1){  //%%%%%%% CAMBIAR SV
2 346

    }
//*****************************************************************************
*************
    bool T_Comandos::Wporce (char* pCad1){  //%%%%%%% CAMBIAR SV
3 (Sólo PLCs CQM1) 347

    }
//*****************************************************************************
*************
    bool T_Comandos::MS (char* pCad1){    //%%%%%%% LECTURA DE
ESTADO 348

    }
//*****************************************************************************
*************
    bool T_Comandos::SC (char* pCad1){  //%%%%%%% ESCRITURA DE
ESTADO 349

    }
//*****************************************************************************
*************
    bool T_Comandos::MF (char* pCad1){    //%%%%%%% LECTURA DE
ERROR 350

    }
//*****************************************************************************
*************
    bool T_Comandos::KS (char* pCad1){  //%%%%%%% FORZADO A ON
351

    }
//*****************************************************************************
*************
    bool T_Comandos::KR (char* pCad1){  //%%%%%%% FORZADO A OFF
352

    }
//*****************************************************************************
*************
    bool T_Comandos::FK (char* pCad1){    //%%%%%%% FORZADOS
MULTIPLES A ON/OFF 353

    }
//*****************************************************************************
*************
    bool T_Comandos::KC (char* pCad1){  //%%%%%%% CANCELACION
DE FORZADOS 354

    }
//*****************************************************************************
*************
    bool T_Comandos::MM (char* pCad1){  //%%%%%%% LECTURA DE
MODELO DE PLC 354

    }
//*****************************************************************************
*************
    bool T_Comandos::TS (char* pCad1){  //%%%%%%% PRUEBA DE
COMUNICACIONES 355
```

```
        AnsiString aux,aux1;
        aux = pCad1;
       // if(!ok_FCS(pCad1)){ // compara el FCS que saco del dato con el FCS que me envian
        // ok = false;
        // goto salir ; }
        int j; ;
        j= aux.Length() ;
        j=j-9;
        aux1=aux.SubString(6,j);
        strcpy(pCad1,aux1.c_str());
        ok = true;
        salir:
        return ok;
    }
//*************************************************************************************************
    bool  T_Comandos::RP  (char*  pCad1){     //%%%%%%% LECTURA DE PROGRAMA 355

    }
//*************************************************************************************************
    bool  T_Comandos::WP  (char*  pCad1){     //%%%%%%% ESCRITURA DE PROGRAMA 356

    }
//*************************************************************************************************
    bool  T_Comandos::QQ  (char*  pCad1){     //%%%%%%% COMANDO GENERICO 356

    }
//*************************************************************************************************
    bool  T_Comandos::XZ  (char*  pCad1){     //%%%%%%% ABORTAR (sólo comando) 358

    }
//*************************************************************************************************
    bool T_Comandos::aste_aste  (char* pCad1){ //%%%%%%% INICIALIZAR (sólo comando) 358

    }
//*************************************************************************************************
    bool  T_Comandos::IC(char*  pCad1){       //%%%%%%% COMANDO INDEFINIDO

    }


//--------------Funciones-----------------------------------------------------
//----------------------------------------------------------------------------
//-------Cambiar valor a decimal -------------------------------
    unsigned short T_Comandos::valor_decimal(char* pCad1) {
     AnsiString recibido=pCad1;
     unsigned int iPos1=0;
     char *aux,*aux1,recibi[4];
     aux=&recibi[0];
     strcpy(recibi,recibido.c_str());
     for(int ii=0; ii<4;ii++){
      aux1=&recibi[ii] ;
      iPos1=iPos1+ conv(*aux1)*pow(16,3-ii);   //valor a escribir en decimal
     }
     return iPos1;
    }


//--------------Extrae Trama -------------------------

        void T_Comandos::trama(char* pCad2){
          AnsiString aux;
          aux=pCad2;
          int j;
          j= aux.Length() ;
          j=j-9;
          aux=aux.SubString(6,j);
          strcpy(pCad2,aux.c_str());
        }
```

6.2 Manuales técnicos

6.2.1 Comandos Host link

Con el siguiente enlace abre el archivo especifico del manual técnico original sobre los comando HOST LINK, que se encuentra en la misma carpeta donde se entrega este TFG.

Host link.pdf

6.2.2 Programación del PLC CQM1,CPM1,CPM1A,SRM1

Con el siguiente enlace abre el archivo especifico del manual técnico original sobre la programación de los PLCs PLC CQM1,CPM1,CPM1A,SRM1 que se encuentra en la misma carpeta donde se entrega este TFG.

Manual Programación CQM1-CPM-SRM.pdf

Guia raìda de programación

TFG Jordan Saavedra Felipe\GR_CQM1H_CJ1M.pdf

6.2.3 Manual. Borland C++ Builder

TFG Jordan Saavedra Felipe\McCraw Hill - Borland C++ Builder The Complete Reference.pdf

TFG Jordan Saavedra Felipe\Developers Guide C++Builder 6.pdf

6.2.4 Manuales de programación de C++ (Basico y Avanzado)

TFG Jordan Saavedra Felipe\cppbasico.pdf

TFG Jordan Saavedra Felipe\UN_2_C++Avanzado.pdf

6.2.5 Transparencias API COM (Windows)

TFG Jordan Saavedra Felipe\Win_API_32.pdf

TFG Jordan Saavedra Felipe\Transparencias API COM(2).pdf

PLIEGO DE CONDICIONES

7 DISPOSICIONES GENERALES

7.1 Objeto

El objeto del presente proyecto titulado "**Diseño de un Emulador de comunicaciones serie**" es la Culminación con éxito los estudios de Grado en Ingeniería Electrónica Industrial y Automática, y está enmarcado bajo la designación "Trabajo de Fin de Grado".

El autor del presente trabajo ha cursado los estudios en la Universidad de la Rioja, cumpliendo con las directrices especificadas por dicho centro en la normativa del trabajo de fin de grado en vigor en el curso 2016/2017.

El objeto de este Pliego de Condiciones es recoger y fijar las disposiciones técnicas, administrativas y económicas, y las normativas que ha de regir el programa para que cumpla con los objetivos estipulados.

Las características del diseño de este proyecto han sido descritas en detalle en la memoria de este proyecto y en sus correspondientes anexos y planos.

Las condiciones que se especifican en este documento tratan de cumplir con la calidad esperada para este proyecto. En caso de no realizarse según estas condiciones, el proyectista no se responsabilizará de los fallos o averías que puedan ocasionarse en su funcionamiento, y los problemas derivados repercutirán sobre terceras personas.

Cualquier modificación sobre lo establecido en los documentos de este sistema deberán ser aprobados por el proyectista.

7.2 Propiedad intelectual

Según el artículo 12 del reglamento de PFC de la ESCUELA TÉCNICA SUPERIOR de Ingeniería Industrial de La Rioja, aprobado por el Consejo de Gobierno del 15 de Abril de 2005, *"Podrá ser consultada la copia de la biblioteca si el autor y el Director de Proyecto Fin de Carrera así lo autorizan por escrito*".

7.3 Condiciones generales

Este proyecto se ajusta en su desarrollo a los reglamentos y disposiciones electrónicas vigentes. Atendiendo a esto una vez se haya aprobado por el Ministerio de Industria, tendrá carácter de obligado cumplimiento.

Una vez realizado el proyecto, se podrán realizar diversas modificaciones siempre bajo la supervisión del ingeniero o proyectista.

En caso de efectuarse alguna modificación, el correspondiente proyecto modificado se considera como parte integrante del proyecto definitivo y como tal, sujeto a las condiciones y especificaciones aprobadas por el ministerio.

La empresa adjudicataria suscribirá contrato ante notario donde se hará constar, a parte de los términos legales obligatorios, plazos de entrega y la conformidad con la sanción cuyo incumplimiento pueda acarrear.

7.3.1 Normativas y reglamentaciones

La realización del proyecto se regirá por la Ordenanza General de Seguridad e Higiene en el Trabajo del 7 de abril de 1970 y posteriores actualizaciones. Así mismo, se regirá por el Reglamento Electrotécnico de Baja Tensión, en el que se tendrán en cuenta las siguientes normas:

- UNE 20-514-82: Reglas de seguridad para los aparatos electrónicos

- UNE 157001:2002: sobre criterios generales para la elaboración de proyectos, que establece las consideraciones generales que permitan precisar las características que deben satisfacer los proyectos de productos obras y edificios, instalaciones, servicios o software, para que sean conformes al fin que están destinados.

- ISO/IEC 25000: conocida como SQuaRE (System and Software QualityRequirements and Evaluation), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software.

- ISO/IEC 25001 Planning and Management: establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

- Norma IEC 61131 es un conjunto de normas e informes técnicos publicados por la Comisión Electrotécnica Internacional con el objetivo de estandarizar los autómatas programables.

8 DEFINICIÓN Y ALCANCE

8.1 Objeto del pliego de condiciones

El presente pliego regirá en unión de las disposiciones que con carácter general y particular se indican, y tiene por objeto la ordenación de las condiciones técnico-facultativas que han de regir en la ejecución del presente Proyecto.
Se considerarán sujetas a las condiciones de este Pliego, todos los trabajos cuyas características y presupuestos, se adjuntan en las partes correspondientes del presente Proyecto. Se incluyen por tanto en este concepto los trabajos de programación para el emulador, así como la configuración de datos y algoritmos.

Si en el transcurso de los trabajos se hiciera necesario ejecutar cualquier clase de modificación o instalaciones que no se encuentren descritas en este Pliego de Condiciones, el Adjudicatario estará obligado a realizarlas con estricta sujeción a las órdenes que, al efecto, reciba del proyectista, y en cualquier caso, con arreglo a las reglas del buen arte constructivo.

El Ingeniero tendrá plenas atribuciones para sancionar la idoneidad de los sistemas empleados, los cuales estarán expuestos para su aprobación de forma que, a su juicio, los dispositivos que resulten defectuosas total o parcialmente, deberán ser retirados o sustituidos, sin que ello dé derecho a ningún tipo de reclamación por parte del Adjudicatario.

8.2 Documentos que definen el proyecto

El presente Pliego, conjuntamente con la Memoria, presupuesto, forman el presente proyecto. Los Diagramas de flujo constituyen los documentos que definen el proyecto en forma geométrica y cuantitativa, debido a la falta de necesidad de unos planos relevantes.

Los documentos que definen el proyecto y que la propiedad entregue al Contratista, pueden tener carácter contractual o meramente informativo.

Son documentos contractuales los Diagramas de flujo, Pliego de Condiciones, pruebas y Presupuestos Parcial y Total, que se incluyen en el presente Proyecto.

Los datos incluidos en la memoria y Anexos, así como la justificación de precios tienen carácter meramente Informativo.

Cualquier cambio en el planteamiento del sistema que implique un cambio sustancial respecto de lo proyectado deberá ponerse en conocimiento de la Dirección Técnica para que lo apruebe, si procede, y redacte el oportuno proyecto reformado.

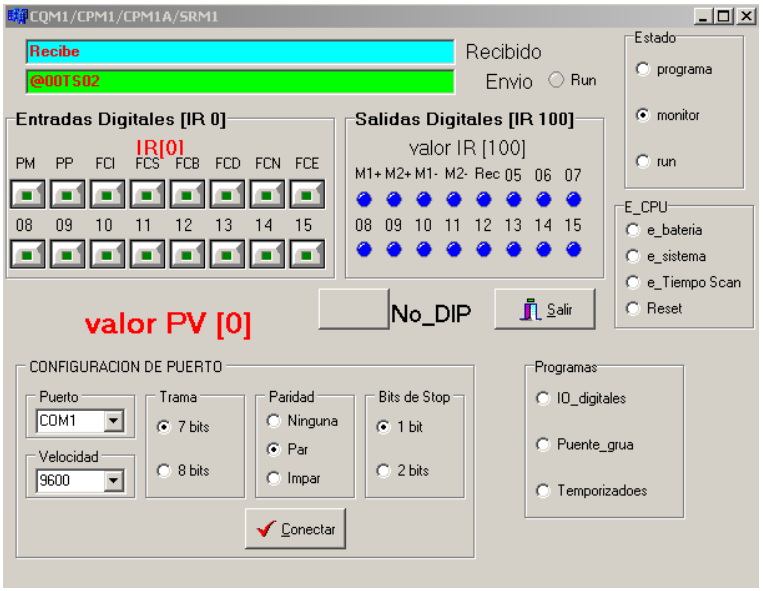
8.3 Compatibilidad entre documentos

En caso de incompatibilidad o contradicción entre los Planos y el Pliego, prevalecerá lo escrito en este último documento. Lo mencionado en el Pliego de Prescripciones Técnicas Particulares y omitido en los planos o viceversa, habrá de ser considerado como si estuviese expuesto en ambos documentos, siempre que la unidad de estudio este definida en uno u otro documento y figure en el Presupuesto.

9 ESPECIFICACIÓN DE USO DE SOFTWARE

9.1 Emulador de comunicaciones

El interfaz que simula el comportamiento de un plc de omron de las series (CQM1, CPM1, CPM1A y SRM1) es el que se puede observar en la figura. Consta de un entorno donde se puede programar el puerto serie por el cual se va a comunicar con el exterior, pulsadores y Leds que simulan las entradas digitales del canal (IR0) y salidas digitales del canal (IR100) debido a que esto es lo que yo considero más relevante a la hora de demostraciones , tiene 3 campos de selección uno para cambiar el estado del PLC ya sea run, monitor o programa, una selección de posibles errores de CPU (error de batería, error de sistema, error de tiempo scan) y por ultimo tiene una selección de programas internos que han sido implementados en su memoria para la correcta simulación de las diferentes aplicaciones, y por ultimo tiene dos cajetines uno azul turquesa y otro verde que es donde se podrá visualizar los comandos Hos tlink en la comunicación ya sea de recepción o transmisión.



Una cuando se ha configurado el puerto de como lo queremos utilizar al darle al botón conectar el automáticamente abre el puerto y comunica tal y como hemos especificado que lo abrirá, ya se queda esperando a que una aplicación externa se comunique con el para que pueda mandarle las órdenes oportunas y establezcan una comunicación.

Con los interruptores de la entrada digital del canal IR(0) se puede ir cambiando su estado no solo externo como se podrá ver si no también interno, por eso hay una etiqueta que muestra el valor del canal así con esto en todo el tiempo se puede visualizar el valor que hay guardado en el canal IR(0). Por el contrario los led de las salidas digitales no se observara ningún cambio hasta que no sea cargado un valor en ellas por vía aplicaciones externas.

Explicación de los paneles de selección:

Estos paneles están sobre todo para poder simular posibles cambios de alteraciones de funcionamiento del sistema ya que para poder simular un programa o un error del sistema y el estado de una CPU de un PLC, hay que provocar dichas alteraciones de funcionamiento para poder ver cómo responde el simulador y ver que no solo el sistema responde a funcionamiento correcto si no también da respuesta a funcionamientos incorrectos, como un error de batería o que se manda el comando de escritura estando el emulador en estado de run.

- Estado. Tiene la selección de los 3 estados importantes de cualquier PLC (run, monitor, programa). en modo run es para ejecución del programa a ejecutar no se puede modificar memoria (escribir) y solo se puede leer memoria, modo monitor es como una especie de modo run lo único que aquí este modo deja modificaciones de memoria este modo es el que se utiliza para todo aplicaciones SCADA. y el modo programa es para precargar el programa de la automatización que se quiera controlar sin ejecución alguna del dicho programa, este estado en este proyecto no tiene ninguna relevancia debido a que esto no es un proyecto de automatización entonces no hay ningún programa que se le quiera programar para después ejecutarlo, los programas ya están implementados en código (C++) de las diferentes aplicaciones que lo usaran, por eso digo que el estado programa no tienen ninguna importancia aquí y por lo cual no se utiliza, solo para cambiar internamente en memoria y saber en qué estado esta.

- Error de CPU: en este bloque simulo 3 posibles errores de los muchos que puede a ver en una CPU de un PLC, esto se hace para poder ver que el simulador responde a los diferentes comandos entando en error con el comando oportuno de error de sistema que le provoque.

- Selección de programa: este bloque es para seleccionar el programa que se quiera ejecutar, implementado internamente en C++ para una correcta demostración de respuesta del simulador a diferentes posibles aplicaciones de automatización.

-

9.2 Aplicaciones para supervisión y control del emulador de comunicaciones

9.2.1 Automatización de un puente-grúa

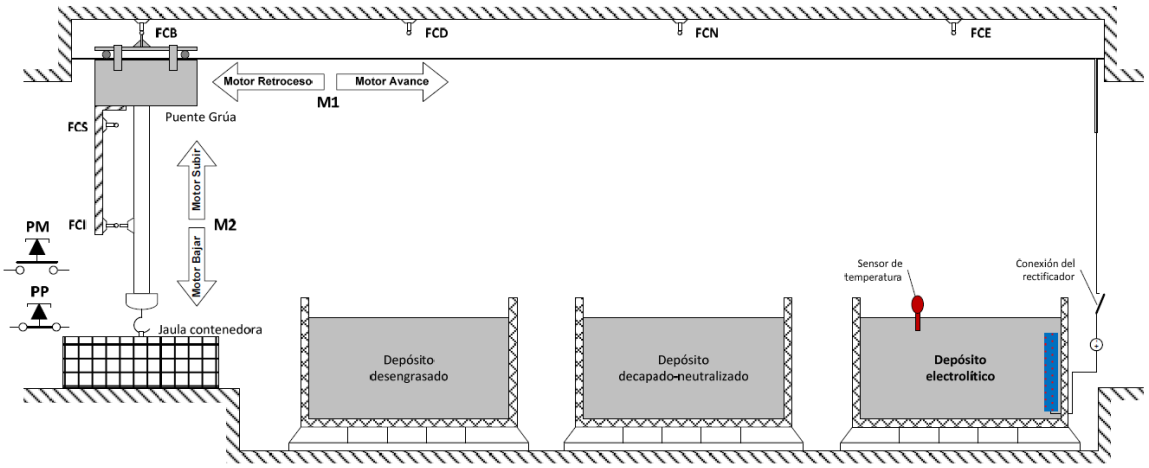
Se trata de automatizar una línea para el tratamiento de superficies de piezas metálicas con el fin de hacerlas resistentes a la oxidación.

El conjunto contara de un sistema para el transporte de piezas (puente-grúa) y otro de baños para el pre-tratamiento y electrolisis.

El puente-grúa dispone de dos motores: M1 para la translación longitudinal del carro de transporte y M2 para el movimiento vertical de la jaula contenedora de piezas. Ambos motores disponen de un arranque inversor para gobernar el sentido de movimiento requerido.

El sistema de baños lo forma 3 depósitos que posibilitan los tratamientos necesarios para el recubrimiento efectivo de las piezas. El primero de ellos realiza el desengrasado del material, el segundo elimina la capa de óxido activando su superficie y el tercero la deposición de la capa protectora.

En la siguiente figura se ilustra el proceso a automatizar.



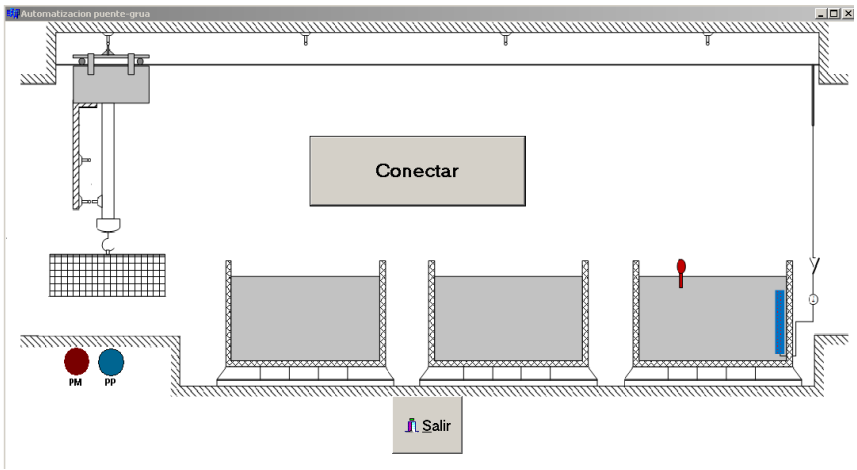
El sistema está en la situación de reposo o inicio de ciclo. El puente-grúa se encuentra en la "posición base", final de carrera base (FCB) y final de carrera inferior (FCI) activados los dos.

A partir de aquí se podría iniciar un ciclo de funcionamiento completo al actuar sobre el pulsador de marcha (PM). La secuencia de movimientos a realizar a partir de ese momento se describe a continuación:

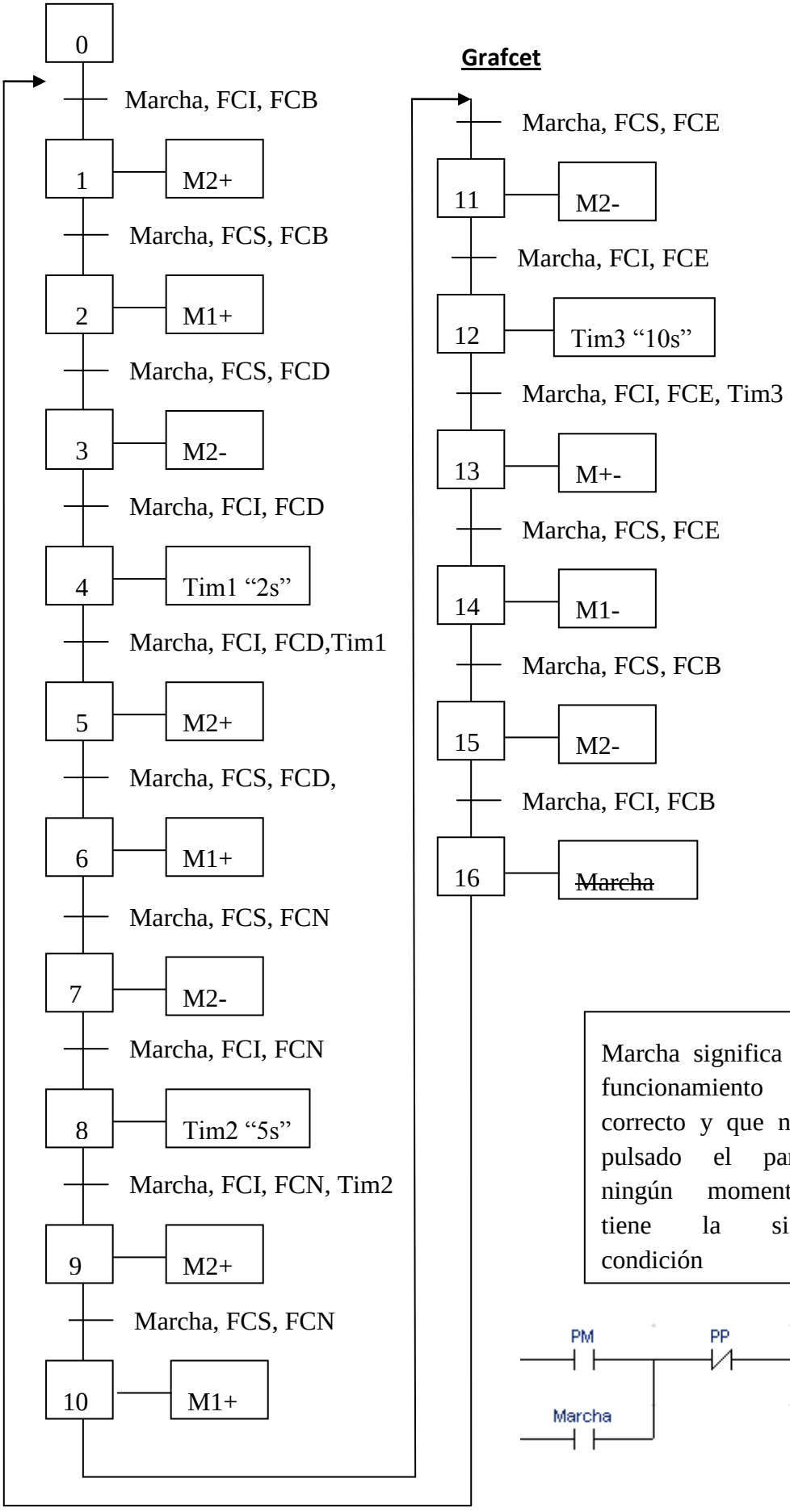
- Motor M2 sube la jaula con el material hasta llegar al final de carrera superior (FCS).
- Motor M1 desplaza el carro de transporte en sentido de avance hasta el puesto de desengrasado (FCD).

▪ Motor M2 baja la jaula hasta FCI momento en el que se produce la inmersión completa del material en el depósito de desengrasado. El tratamiento deberá mantenerse durante 2 segundos.
▪ Motor M2 sube la cadena del polipasto hasta el tope superior (FCS), una vez transcurrido el tiempo anterior.
▪ Motor M1 desplaza el carro hasta el puesto de decapado-neutralizado (FCN).
▪ Motor M2 baja el material al baño de eliminación de óxido y lo mantiene durante 5 segundos.
▪ Motor M2 sube la jaula hasta el tope superior (FCS) transcurrido el tiempo de tratamiento.
▪ Motor M1 avanza hasta el puesto de electrólisis (FCE).
▪ Motor M2 baja la cadena del polipasto hasta FCI provocando la inmersión en el baño de sales.
▪ Se activa el rectificador durante 10 segundos para la deposición de los iones metálicos en las piezas activadas catódicamente.
▪ Motor M2 sube la jaula hasta FCS transcurrido el tiempo de tratamiento.
▪ Motor M1 retrocede hasta la posición base (FCB) pasando por la vertical de los distintos puestos (FCN, FCD).
▪ Motor M2 baja el material ya tratado hasta FCI para su descarga manual.
El proceso podría volver a repetirse cargando sobre el gancho una nueva jaula con piezas y actuando nuevamente sobre el pulsador de marcha.

Si durante el funcionamiento normal descrito se actúa sobre el pulsador de paro (PP) el puente-grúa por seguridad o avería se quedara paralizado en el estado en el que esta y se mantendrá en dicho estado hasta que el proceso vuelva a ser rearmado por el pulsador de marcha (PM), continuando con las siguientes acciones.



Como se observa en la figura anterior así es el interface de la aplicación de la simulación de esta aplicación



Marcha significa que el funcionamiento es correcto y que no se a pulsado el paro en ningún momento. Y tiene la siguiente condición

Al ejecutar la aplicación saldrá una primera pantalla que tiene un botón que pone “conectar”, se abrirá el puerto COM2 y automáticamente envía una trama de prueba, que esperar a obtener respuesta por el emulador si está conectado correctamente, si es así la comunicación está establecida y se puede proceder a la marcha del proceso, de lo contrario saldrá la pantalla grafica con las imágenes del puente y depósitos y el botón nuevamente de conectar, para volver a intentar establecer comunicación por el COM2 y así sucesivamente.

La simulación de esta automatización, lo que realiza es el envío y recepción de comandos, la aplicación es la encargada de controlar los estados de dicha automatización, que quiero decir con esto, pues es muy sencillo la aplicación es la encargada de encender las entradas y salidas oportunas para ejecutar los procesos correctamente y que todo fluya con su transición adecuadamente y se ejecute correctamente. Ósea el programa de la automatización implementado en C++ esta todo en esta aplicación esta es la encargada de gestionar si pasa al siguiente estado o no según estén sus salidas o entradas previamente activas o desactivas correspondientemente para el paso del siguiente proceso.

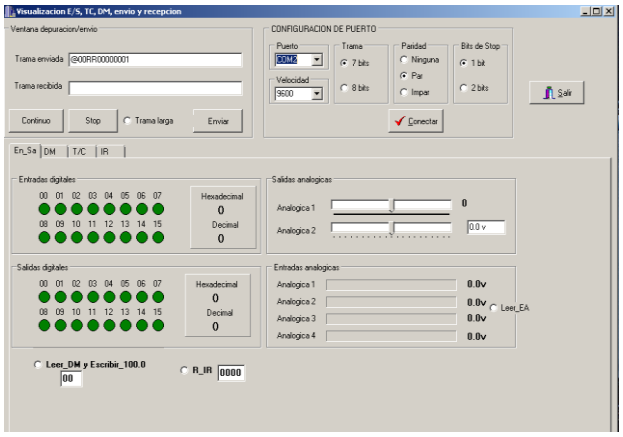
Al darle al botón de marcha “PM” si su estado de transición al siguiente pasó esta todo correcto inicia el arranque y movimiento del puente, subiendo la cesta y así seguirá hasta realizar todo el ciclo establecido en el enunciado.

9.2.2 Interface de visualización y control de la memoria interna del emulador

Este interface me parece muy importante y por eso lo he desarrollado, ya que es un entorno muy práctico y con el podemos visualizar muy bien el correcto funcionamiento de la comunicación, como el estado de las memorias más relevantes del sistema y en general casi lo más importante de un PLC. Ya que con el podemos enviar tramas de comandos Hostlink, ver los canales de entrada y salidas digitales, entradas y salidas analógicas ver las DMs y precargar los valores de los Temporizadores/contadores, todo esto al gusto de cómo lo quiera utilizar el usuario.

Tiene un bloque de configuración de puerto de comunicación, bloque de envío y recepción de comandos, y luego tiene 4 pestañas que serán las encargadas de visualizar y controlar los estados de las posiciones de memoria del emulador, todo con su correspondiente botones para el correcto funcionamiento.

Al iniciar la aplicación si hay una comunicación establecida con el emulador, este automáticamente carga valores de DMs, e IR, diseñado a si para su posterior utilización, luego lee 100 DM, 100 T/C, 110 IR, y rellena las tablas que se encuentran en las 3 pestañas siguientes. Establecida la comunicación con el emulador el bloque de comunicación se bloque para evitar posibles confusiones que pueda haber.



Control y visualización de entradas y salidas digitales y analógicas, tiene un botón de ciclo continuo que se encarga de empezar como su nombre lo dice en un ciclo continuo, el proceso que se ejecuta es el de estar leyendo las entradas digitales del emulador y pintar los leds (verdes) correspondientes, así se visualiza cuando hay cambios en las correspondientes entradas digitales, y a su vez este valor es cargado haciendo una conversión a "código BCD " oportuna, al canal de entrada analógica 1. Teniendo en cuenta también los valores negativos que serán obtenidos con la entrada (0.15). Un valor entero es configurado con las entradas (0.0, 0.1 ,0.2, 0.3) si se sobre pasa del número "9" la aplicación no hace nada, los valores decimales son configurados con las entradas (0.4, 0.5 ,0.6, 0.7) también se puede ver en la barra de estado en salidas analógica1 IR(101) como cambia su valor, haciendo su escalado correspondiente desde -10v a 10v.

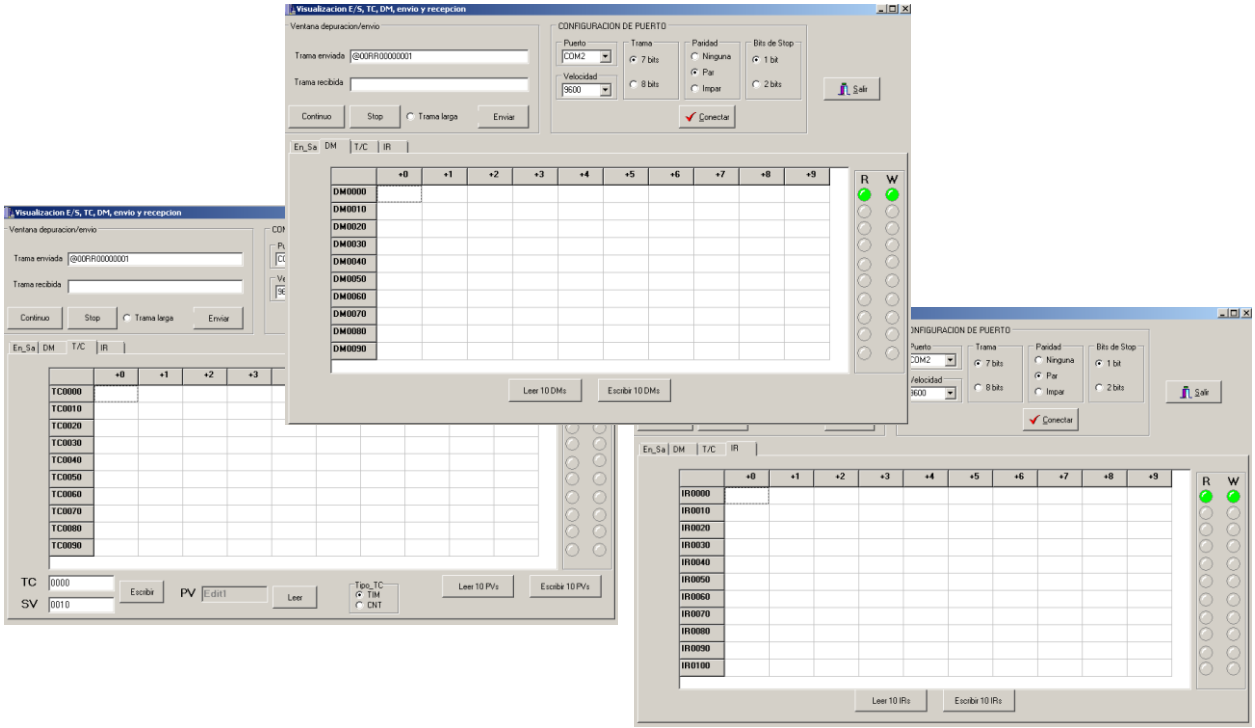
La salida analógica2 IR(102) es controlada con una barra que puede ser movida por el usuario, cambiando el valor y automáticamente se genera la trama oportuna que se enviara al emulador para que guarde ese valor en la IR(102), haciendo su escalado correspondiente desde 0 a 10 v.

En el bloque de salidas digitales se puede visualizar el valor que tiene los DMs que seleccionemos en el cajetín inferior, pulsando el clic que hay. Lo que hace esto es leer el DM oportuno y luego pintar los leds (verdes) y escribir en el emulador el valor del DM.

Las entradas analógicas solo puedo visualizarlas, ciclando en el clic “leer_EA”, veo su valor con un correspondiente escalado de 0 a 10v.

Y por último hay un cajetín en la zona inferior al lado de leer las DMs, que lo que ejecuta es la lectura de cualquier IR que necesitemos la información en ese momento.

Como se observa en las siguientes imágenes son pestañas donde se puede visualizar la memorias DMs, T/C, y IR de mi emulador y a su vez también puedo cambiar, solo es pinchar sobre la posición que quiero cambiar y tener el led de “W” posicionado en la fila que deseo escribir y le doy al botón escribir, el led pasara a la siguiente fila, para leer es solo tener posicionado el led ”R” en la fila que deseemos leer y se le da a leer, el led pasara a la siguiente fila igualmente. Y así sucesivamente.



La única pestaña un poco distinta de las 3 es la de los T/C que tiene una parte donde puede cargar el valor “SV” del temporizador/contador que seleccione individualmente y también puede ver el valor “PV”.

10 CONDICIONES ECONÓMICAS

10.1 Errores en el proyecto

En el caso de existir errores de diseño se dará cuenta al proyectista para dar con la solución al problema.

Además se prohibirá el uso de la aplicación por posibles daños que pueda ocasionar hasta que el error se vea subsanado.

10.2 Liquidación

Terminada la elaboración del proyecto se procederá a la liquidación, donde se incluyen los importes correspondientes a la elaboración del proyecto.

En el caso de que surgiera alguna posible modificación aprobada por la dirección del proyecto el contratante será el encargado de abonar este importe íntegramente.
Al suscribir el contrato el contratante deberá abonar al adjudicatario el 60% del total del presupuesto. Tras la entrega e instalación del software se deberá realizar un nuevo abono del 20% del total del presupuesto, que será condición necesaria para continuar con el proceso de diseño. En caso de no realizar el abono el proyectista queda exento de todos los deberes sus contractuales.

El 20% se quedará como garantía durante los 6 primeros meses a partir de la fecha de puesta en marcha. Si transcurrido este periodo de tiempo no se han advertido evidencias de defecto, se abonara la cantidad pendiente de entrega, y a partir de ese momento, se consideran completamente concluidos los compromisos entre las dos partes, a excepción del periodo de garantía.

10.3 Plazo de garantía

El emulador de comunicaciones posee un plazo de garantía total de 24 meses, a partir de la fecha de finalización del software. Esta garantía cubre la posible modificación de programa pertinente que sea necesaria, y un servicio técnico.

El plazo de garantía se extiende a 36 meses para el servicio técnico, la garantía quedará totalmente anulada en el caso de que el software sufra alguna manipulación por parte del cliente o haya sido manipulado por personas ajenas a nuestros Servicios Técnicos Oficiales. Se incluye Manual de Uso, para su correcta utilización.

No están incluida las reparaciones concernientes a averías debidas a causas accidentales: bloque de Windows, borrados de sistema, problemas software en el PC, etc. Siempre que se demuestre que su origen es independiente del normal funcionamiento del sistema del

emulador; tampoco estarán incluidas aquellas averías ocasionadas por actos de manipulación de código. En estos u otros supuestos de avería, el contratista estará obligado a suministrar a la parte contratante, antes de quince días, un presupuesto de reparación de averías.

De encontrarse mal funcionamiento en el momento de la a entrega, el codigo deberá de ser revisado sin coste alguno para el usuario en un plazo inferior a 48 horas por parte del servicio técnico.

10.4 Disposición final

Las partes contratantes, ingeniero Director y empresa cliente, se ratifican en el contenido del presente pliego de condiciones, que tiene igual validez, a todos los efectos, que una escritura pública, prometiendo su fiel cumplimiento.

10.5 Jurisdicción

Cuantas cuestiones, litigios o diferencias pudieran surgir durante o después de la programación, las partes se someterán a juicio de amigables componedores nombrados en número igual por ellas y presidido por el Ingeniero Director y, en último término, a los Tribunales de Justicia.

El Contratista es responsable de la ejecución del software en condiciones establecidas en el contrato y en los documentos que componen el Proyecto (la Memoria no tendrá consideración de documento del Proyecto).

10.6 Pagos de árbitros

El pago de impuestos y arbitrios en general, etc., cuyo abono debe hacerse durante el tiempo de ejecución de las obras por concepto inherente a los propios trabajos que se realizan correrá a cargo de la Contrata, siempre que en las condiciones particulares del Proyecto no se estipule lo contrario. No obstante, el Contratista deberá ser reintegrado del importe de todos aquellos conceptos que el Ingeniero Director considere justo hacerlo.

10.7 Causas de rescisión del contrato

Se considerarán causas suficientes de rescisión las que a continuación se señalan:

1. La muerte o incapacidad del Contratista.

2. La quiebra del Contratista.

En los casos anteriores, si los herederos o síndicos ofrecieran llevar a cabo la obra, bajo las mismas condiciones estipuladas en el contrato, el propietario puede admitir o rechazar el ofrecimiento, sin que en este último caso tengan aquel derecho a indemnización alguna.

3. Las alteraciones del Contrato por las causas siguientes:

a) La modificación del Proyecto en forma tal que presente alteraciones fundamentales del mismo, a juicio del Ingeniero Director y, en cualquier caso siempre que la valoración del presupuesto de ejecución , como consecuencia de estas modificaciones, represente, en más o menos del 10 por 100, como mínimo, de algunas unidades del Proyecto modificadas.

b) La modificación de unidades de programador, siempre que estas modificaciones representen variaciones en más o en menos, del 40 por 100, como mínimo de las unidades del proyecto modificadas.

4. El no dar comienzo la contrata a los trabajos dentro del plazo señalado en las condiciones particulares del proyecto.

5. El incumplimiento de las condiciones del contrato, cuando implique descuido o mala fe, con perjuicio de los intereses del software.

6. La terminación del plazo de ejecución del software, sin haberse llegado a ésta.

7. El abandono de instalación de software sin causa justificada.

8. La mala fe en la ejecución.

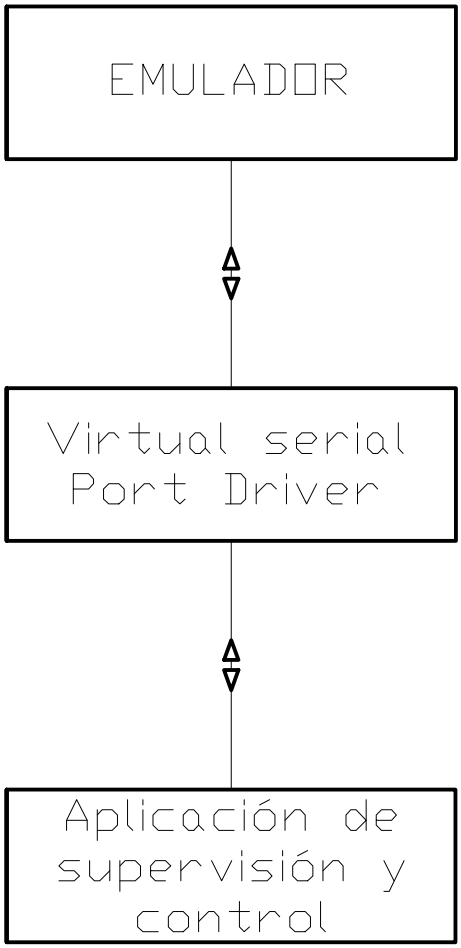
10.8 Disposición

Las dos partes contratantes, dirección técnica y empresa, ratifican el contenido del siguiente pliego de condiciones, el cual tiene igual validez, a todos los efectos, que una estructura publica, prometiendo fiel cumplimiento.

En Logroño a 4 de julio de 2017

Fdo. Felipe Jordan Saavedra

PLANOS



	FECHA	NOMBRE		ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en ingeniera electronica
Dibujado	29/06/2017	FELIPE JORDAN		
Comprobado				

ESCALAS	Estructura General	Numero 1
PROYECCION		REFERENCIA:
		Sustituye a
		Sustituido por

Utiliza sus constructores para inicializar las variables internas preseleccionadas.

Inicio

Instanciar objeto oTplc (T_Comandos)

Configurar puerto serie

¿Conectar al puerto puerto serie?

NO

SI

Abrir el puerto seleccionado y con la Configuración seleccionada

Mensaje "Puerto no abierto"

¿Puerto serie abierto?

NO

SI

Activación de temporizador (c++) (Timer1), encargado de la simulación de comunicación (Recepción/Transmisión) de comandos por el puerto, analizándolos y gestionándolos con sus correspondientes métodos y luego devuelve respuesta por el puerto (Comando respuesta)

Espera comandos en el Timer1

Como se utilizara el emulador

Activación de temporizador (c++) (Timer2), encargado de la simulación del ciclo SCAN PLC para poder visualizar las E/S, por medio de aplicación externa y un comando correspondiente.
Esta parte de programa es un subprograma o interrupción que se ejecutara cada vez que su valor es alcanzado, rompiendo el ciclo de ejecución del programa principal, y una vez ejecutado se vuelve

Activación de temporizador (c++) (Timer3), encargado de la simulación del ciclo SCAN PLC para poder visualizar, el programa de la aplicación del puente de grua, por medio de aplicación externa y un comando correspondiente.
Esta parte de programa es un subprograma o interrupción que se ejecutara cada vez que su valor es alcanzado, rompiendo el ciclo de ejecución del programa principal, y una vez ejecutado se vuelve

Activación de temporizador (c++) (Timer4), encargado de la simulación del ciclo SCAN PLC para poder visualizar, el programa de la aplicación del temporizador (conexión desconexión) y contador . por medio de aplicación externa y un comando correspondiente.
Esta parte de programa es un subprograma o interrupción que se ejecutara cada vez que su valor es alcanzado, rompiendo el ciclo de ejecución del programa principal, y una vez ejecutado se vuelve

	FECHA	NOMBRE		ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en ingeniera electronica
Dibujado	29/06/2017	FELIPE JORDAN		
Comprobado				

ESCALAS	Genérico	Numero 2
PROYECCION	EMULADOR DE COMUNICACIONES SERIE	REFERENCIA: Sustituye a Sustituido por

Inicio

Esperar dato

¿Hay dato recibido? □ intentos >50

NO

SI

Esto se ejecuta cada vez que salta e timer 1

Conocer Nº caracteres recibidos

Conocer si es Trama larga

Devuelve siempre la Trama de respuesta completa, ya sea el comando correcto o erróneo

Comprobar comando con la función (compro)

¿Comando Correcto?

SI

NO

¿Trama larga a enviar por PLC? Caracteres >131

SI

NO

Leer registro de Errores del PLC

Enviar Trama (Respuesta)

Restar caracteres enviados

Esperar delimitador '\r'

NO

¿Hay delimitador '\r'?

SI

Enviar Trama (Respuesta) continuación de la trama antes enviada

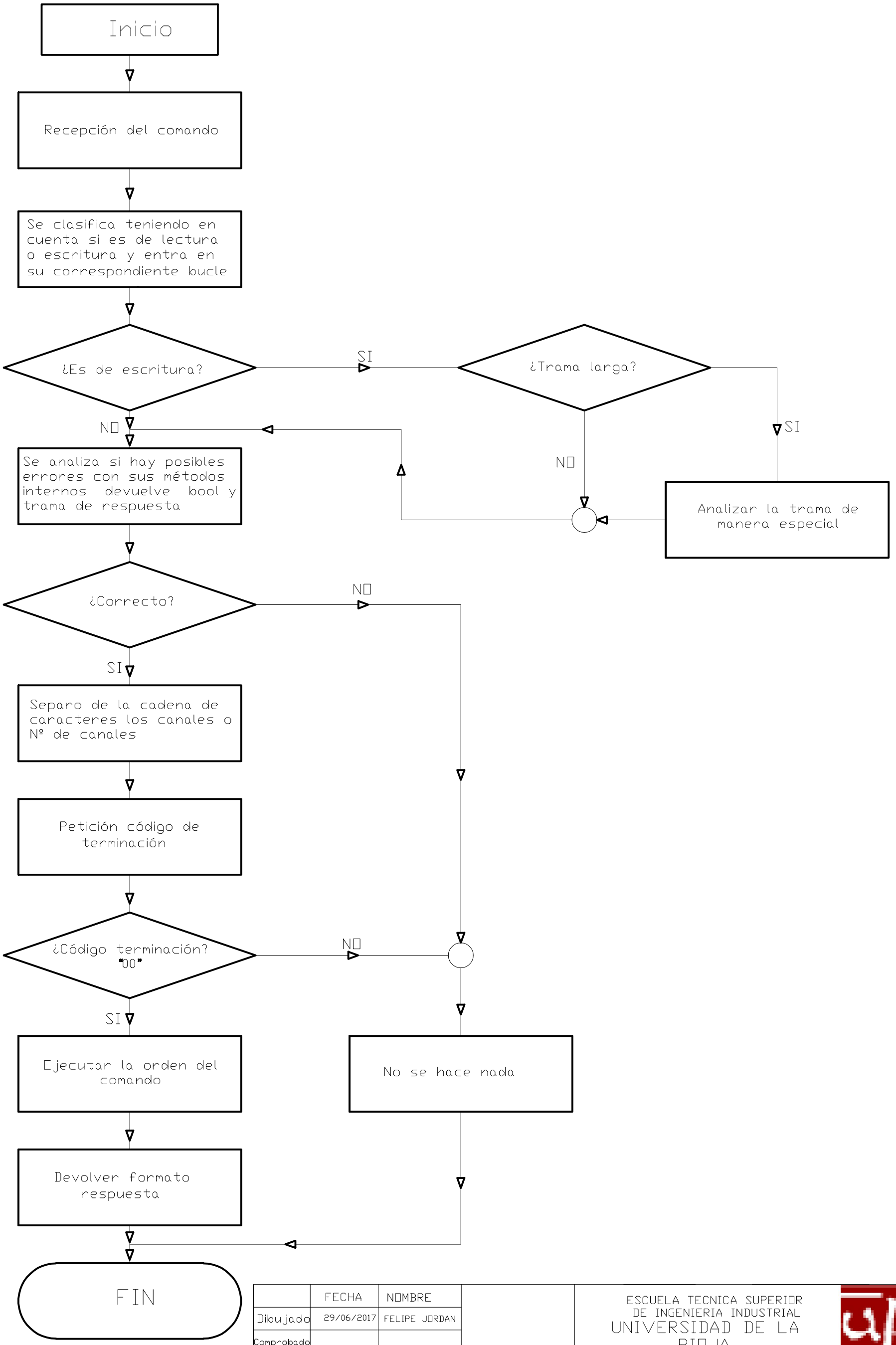
Restar caracteres enviados

Enviar Trama (Respuesta)

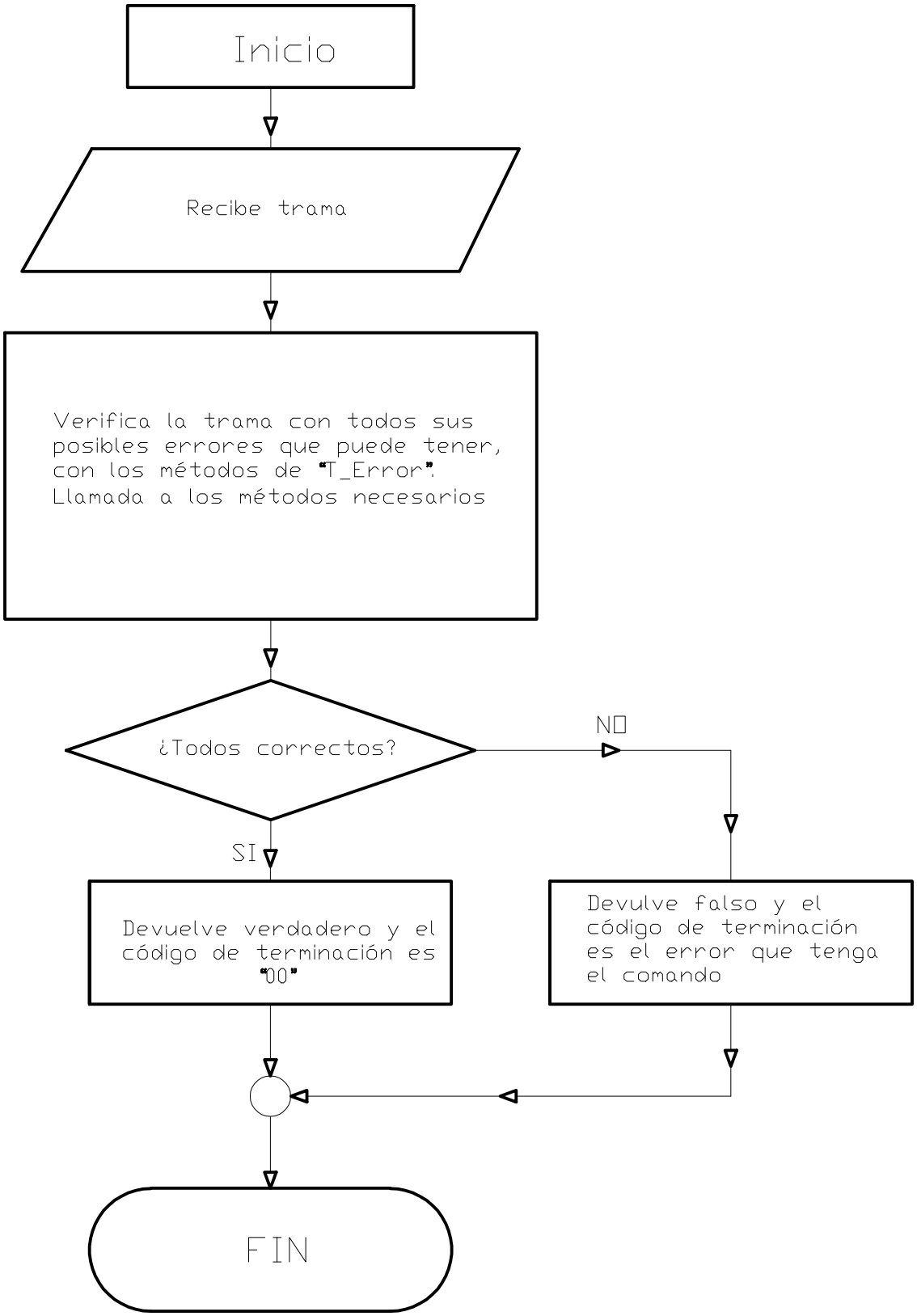
Enviar la respuesta de Error

FIN

	FECHA	NOMBRE		ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en ingeniera electronica	
Dibujado	29/06/2017	FELIPE JORDAN			
Comprobado					
ESCALAS	Timer1				Numero 3
PROYECCION	EMULADOR DE COMUNICACIONES SERIE				REFERENCIA:
					Sustituye a
					Sustituido por



	FECHA	NOMBRE		ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en ingeniera electronica	
Dibujado	29/06/2017	FELIPE JORDAN			
Comprobado					
ESCALAS	Función compro			Numero 4	
PROYECCION	EMULADOR DE COMUNICACIONES SERIE			REFERENCIA:	
				Sustituye a	
				Sustituido por	



	FECHA	NOMBRE		
Dibujado	29/06/2017	FELIPE JORDAN		
Comprobado				

ESCALAS	Ejecución del comando	Numero 5
PROYECCION	EMULADOR DE COMUNICACIONES SERIE	REFERENCIA: Sustituye a Sustituido por

PRESUPUESTO

Se ha desarrollado un minucioso presupuesto detallando todas las tareas de las que se compone el emulador, además del detalle de costo de tanto los recursos software como los recursos de mano de obra. Y por último el presupuesto resultante definitivo que se pagara por el software.

Se va a llevar a cabo un desglose de las tareas que se han realizado a lo largo de este TFG, lo que facilitara posteriormente un cálculo aproximado sobre su coste, debido a la complejidad de un trabajo de estas características se ha optado por desglosarlo en distintas fases, las cuales se van a comentar a continuación:

- Software necesario para el desarrollo
- Diseño
- Programación
- Depuración

16 PRESUPUESTOS PARCIALES

16.1 Software necesario para el desarrollo

Cuadro de precios de los software de desarrollo utilizados

software	Precio/ud (€)
C++ Builder 10.2 (profesional)	1802.90
Virtual serial port driver	137.95
Mware Workstation 12.5 Pro	274.95
Total	**2215.8 €**

16.2 Diseño

Este cuadro de precios hace referencia a los procedimientos específicos que se deben analizar con antelación para la creación del emulador.

Concepto	Unidades (Horas)	Precio/hora €	Sub Total €
Investigación técnica de los PLC CQM1/CPM1/CPM1A/SRM1	5	30	150
Investigación técnica de los comandos HOST LINK	5	30	150
Planteamiento del desarrollo de las comunicaciones vía puerto serie	16	45	720
Planteamiento de los interfaces	8	45	360
Planteamiento del desarrollo de la implementación de los comandos HOST LINK	24	45	1080
Planteamiento del desarrollo de la implementación de códigos de terminación (Errores)	24	45	1080
Planteamiento de las aplicaciones externas de control y supervisión	8	45	360
		Total	**3900 €**

Total 3900 €

16.3 Programación

Este cuadro de precios hace referencia a la creación de código de cada uno de las partes que componen el programa.

Concepto	Unidades (Horas)	Precio/hora €	Sub Total €
comunicación	8	30	240
Interfaces	2	30	60
Implementación de los comandos Host link	40	30	1200
Implementación de los códigos de error (códigos de terminación)	32	30	960
Programación de los programas de las aplicaciones externas para la supervisión y control	8	30	240
		Total	**2700 €**

16.4 Depuración

Este apartado hace referencia a los precios de la mano de obra que se invirtieron para la correcta depuración de los programas.

Concepto	Unidades (Horas)	Precio/hora €	Sub Total €
comunicación	12	40	480
Interfaces	2	40	80
comandos Host link	20	40	800
códigos de error (códigos de terminación)	15	40	600
aplicaciones externas para la supervisión y control	8	40	480
		Total	**2440 €**

16.5 Presupuesto total

Con los presupuestos parciales calculados anteriormente se realiza el cálculo del presupuesto total.

Resumen general del presupuesto		
Software necesario para el desarrollo	17.24 %	2215.8 €
Diseño	42.73 %	3900 €
Programación	21.01 %	2700 €
Depuración	18.99 %	2440 €

Presupuesto de ejecución material		**11255.8 €**
Gastos generales	13 %	1463.25 €
Beneficios industriales	6 %	675.34 €
Suma		13394.39 €
Impuestos añadidos IVA	21 %	2812.82 €
TOTAL CON IMPUESTOS		16207.21 €

El valor total del presupuesto del proyecto, impuestos incluidos, asciende a la cantidad de **DIESISEISMIL DOCIENTOS SIETE CON VENTIUN CÉNTIMOS.**

En Logroño a 4 julio de 2017

Fdo. Felipe Jordán Saavedra

Conclusiones y líneas futuras

17 CONCLUSIONES

En el presente TFG se ha conseguido cumplir los objetivos generales y específicos planteados al inicio del documento.

Se ha desarrollado una plataforma de prueba o emulador que consigue emular la comunicación serie basada en el protocolo Host link de OMRON con los PLCs CQM1, CPM1, CPM1A y SRM1.

Para facilitar la usabilidad del emulador se ha diseñado un interfaz gráfico con la que poder configurar adecuadamente el puerto y poder realizar simulaciones de programas de automatización, ver los comandos recibidos, ver las entradas y salidas digitales, simular el estado del emulador y provocar fallos del sistema.

Se ha realizado dos aplicaciones externas con las cuales se podrá controlar y monitorizar el emulador, para poder ver correctamente su funcionamiento ya que debido a que el emulador solo se encarga de recibir los comandos, verificar si son correctos, ejecutarlos internamente y mandar código de respuesta. Con estas dos aplicaciones se podrá ver mejor el funcionamiento por lo menos más intuitivo, aunque cabe resaltar que esto no es un TFG de automatización.

Con este proyecto como lo he dicho con anterioridad no se trata de un proyecto de automatización sino más bien de comunicación emulando la comunicación serie entre dos dispositivos utilizando el protocolo HOST LINK de OMRON, internamente le he implantado en código C++ trozos de programa para simular programas de aplicaciones de automatización para poder ver que funciona con las aplicaciones externas que lo controlan y supervisan.

Cabe destacar que todas las comunicaciones se establece menos o mas complejas de esta manera por eso me resulto muy interesante realizar este tema ya que para mi interés es muy importante.

Este TFG resulta más interesante en uso de educación debido a que es un emulador con lo cual no tiene ninguna aplicación a nivel con el mundo real, se puede enseñar a los nuevos estudiantes como se realiza ejecuta la comunicación con Host link y cómo se gestiona correctamente la memoria en los PLCs PLCs CQM1, CPM1, CPM1A y SRM1.

Y por finalizar quiero dar relevancia a que este TFG no sirve como uso real para ninguna aplicación industrial externa, pero si me a servido para poder conocer muy bien a fondo sobre las comunicaciones industriales en especial la del protocolo Host link, que aunque sea un protocolo de OMRON, todos los fabricantes usan su propio protocolo pero todos llevan una misma idea y utilizan la vía de transmisión serie.

18 LÍNEAS FUTURAS

Una vez desarrollado la implementación del emulador de pruebas, se pueden especificar una serie de líneas de crecimiento que permitan evolucionar en el proyecto. Se pueden destacar las siguientes propuestas.

- Terminar de implementar todos lo comando
- Corregir todos los posibles errores que hay en la comunicación
- Hacerlo más real teniendo en cuenta todas las configuración de un PLC
- Poder emular cualquier dispositivo por puerto serie ya sea otro PLC, un módulo de E/S Distribuidas, etc,…
- Poder interactuar como un dispositivo real, como por ejemplo establecer una comunicación con cx-programmer o con cx-supervisor .

BIBLIOGRAFIA

Manuales

Manual de programación CQM1-CPM1- CPM1-SRM

https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=20&ved=0ahUKEwj_3uTmpd7UAhUD0xoKHS2ZAik4ChAWCFQwCQ&url=ftp%3A%2F%2Fftp.scv.si%2Fvss%2Fcveto_fendre%2Fkrs1%2FProgramming%2520Manual%2520CQM1.pdf&usg=AFQjCNFcT6B_l4dyC2s2zJPtU_28Z-i2ug&cad=rja

Guia rápida de programación CQM1-CPM1- CPM1-SRM

https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&uact=8&ved=0ahUKEwjN7My-pN7UAhXMCBoKHbXzCzEQFghCMAU&url=https%3A%2F%2Fwww.tecnical.cat%2FPDF%2FOmron%2FPLC%2FCJ%2FGR_CQM1H_CJ1M.pdf&usg=AFQjCNEnim3I7J5RDX21aQUIer72zFq4cw

Programación C++

Aprenda C++ Básico

Aprenda C++ Avanzado

https://www.google.es/?gfe_rd=cr&ei=LnFSWb3rOZPY8gel9KsY&gws_rd=ssl#q=c%2B%2B+basico+tecnun

Recursos de software

https://www.embarcadero.com/es/app-development-tools-store/cbuilder

C++ Builder 10.2

https://www.eltima.com/es/purchase/vspdxp/

Virtual serial port driver

http://store.vmware.com/store/vmwde/es_ES/DisplayProductDetailsPage/ThemeID.29219600/productID.323427300

Mware Workstation 12.5 Pro