APUNTES DE INTRODUCCIÓN A LOS PLCs

CURSO DE ACTUALIZACIÓN PROFESIONAL

Walter Giovannini, Andrés Azar, Rafael Canetti, Pablo Belzarena, Agustín Rodríguez, Javier Román

Instituto de Ingeniería Eléctrica

Facultad de Ingeniería

Universidad de la República

Versión Nº11, agosto de 2016

CAPÍTULO 1: INTRODUCCIÓN

1 Historia de los PLC

El primer controlador programable se introduce a fines de la década del 60, con el objetivo de sustituir la lógica de relé. Hasta ese momento, la lógica de relé era la única forma de implementación de los procesos lógicos necesarios para el funcionamiento de máquinas como calderas industriales, máquinas textiles, etc.

La primer compañía que introdujo el controlador programable fue Bedford Associated. Propuso un controlador programable llamado Modular Digital Controller. Esta denominación fue sintetizada en MODICON, nombre que dura hasta nuestro días.

Otros realizaban esta lógica usando computadoras, como la PDP-8 de Digital Research, con costos bastante elevados. De esta forma, se introducía un elemento de gran potencia que podía sustituir lógicas de relés en forma muy eficiente. Estos nuevos controladores basados en computadoras eran fácilmente programables y además tenían la ventaja de tener una interfaz gráfica con el operador.

En la década de los 70, una empresa emergente en el mercado de los controladores programables, Allen Bradley, denominó a sus controladores por la sigla "PLC" (Programmable Logic Controller). Este nombre se universalizó rápidamente entre los demás fabricantes. Allen Bradley se transformaría luego en una de las más grandes compañías de fabricación de PLC.

Como sustitutivo de la lógica de relé, podemos pensar al PLC como una CPU con entradas y salidas, para mover relés y recibir señales de llaves. Las llaves pueden provenir de detectores de proximidad, de un contacto auxiliar de un contactor de potencia, etc.

El primer cambio cualitativo que se da con la introducción de los PLCs es la posibilidad de reprogramación de los mismos. Cualquier cambio en la secuencia del proceso de un circuito con lógica de relé era tan costoso y engorroso que en muchos casos resultaba más económico cambiar todo el tablero de mando.

La sustitución de relés mecánicos por relés virtuales, es decir, aquellos que sólo existían dentro del programa del PLC, permitió un salto enorme en la utilización de estos equipos dentro de complejos procesos industriales.

Obviamente las salidas de los PLCs actúan sobre relés mecánicos o directamente sobre contactores, pero su número baja dramáticamente.

En las figuras se muestra un ejemplo sencillo de aplicación de un PLC. La Figura 1 que sigue muestra un circuito de arranque de un motor de trifásica.

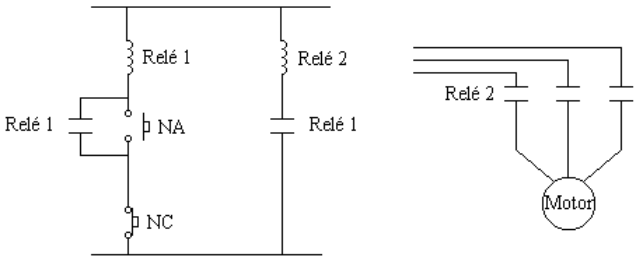


Figura 1

La utilización de un PLC en esta aplicación transformaría la topología anterior en la de la Figura 2.

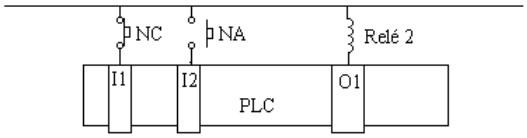


Figura 2

Los contactos se cablean a las entradas I1 e I2 del PLC, y el relé 2 se cablea a la salida O1 del PLC. La lógica del circuito se implementa en software dentro del PLC, de forma que el relé 1 pasa a ser un relé virtual interno al PLC.

Supongamos ahora que se quiere agregar un nuevo botón de arranque. En el circuito de la Figura 1, la configuración recableada quedaría:

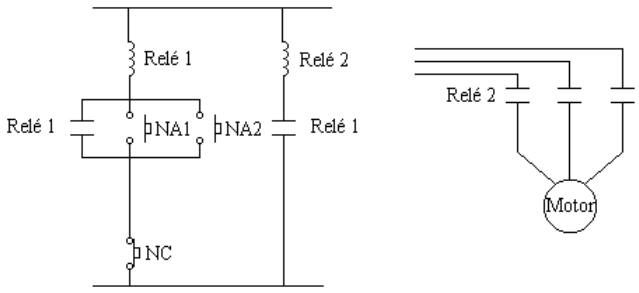


Figura 3

En el circuito de la Figura 2, esto se resuelve simplemente cableando NA2 a una nueva entrada digital del PLC, y cambiando el software:

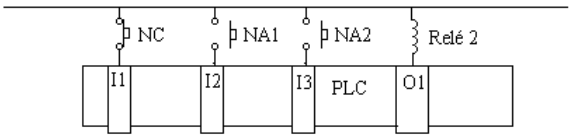


Figura 4

Este ejemplo sencillo muestra dos ventajas de la utilización de un PLC: la sustitución de la lógica de elementos discretos por el software del PLC, y la flexibilidad con que pueden realizarse nuevas conexiones. Un circuito más complejo, que involucrara timers en el arranque, condiciones de seguridad, y encadenara otros procesos, sólo requeriría cablear más entradas y reescribir el software.

A mediados de los 70, la introducción de los microprocesadores y los microcontroladores permitió un salto enorme en el hardware de los PLCs. Los primeros microcontroladores utilizados por la industria de fabricación de los PLCs fueron los AMD 2901 y 2903.

En 1973, Modicon desarrolla el protocolo de comunicaciones MODBUS. En lo posterior, el uso de este protocolo sería muy difundido. Desafortunadamente, la falta de estandarización ha sembrado desorden en el terreno de las comunicaciones entre PLCs. Muchos fabricantes han adoptado el uso de protocolos propietarios, lo que ha dificultado en gran medida la tarea de comunicar PLCs de distinto fabricante.

Es decir, los PLCs no sólo tenían entradas/salidas digitales sino además entradas salidas de comunicaciones. Las comunicaciones tuvieron una importancia grande en la descentralización de los sistemas: se sustituyó el cableado de la planta a un único PLC central, por un sistema de varios PLCs, distribuidos a lo largo de la planta y comunicados entre sí. Esto permitió ahorrar en el costo del cableado, que en muchos sistemas centralizados es del orden del costo de los equipos.

Además de las comunicaciones se introdujeron entradas salidas analógicas, cuya resolución evolucionó desde 8 bits hasta 12 a 16 bits en la actualidad. Esto permitió incursionar no sólo en la lógica digital sino además en la posibilidad de realizar control dentro de estos equipos.

En la década de los 80 se introdujo la posibilidad de programar los PLC por medio de un PC. Este método facilitó la programación, que hasta entonces se hacía con terminales manuales de uso bastante engorroso.

El uso intensivo de microcontroladores como el 8048 y el 8051 de Intel y la línea 68000 de Motorola en la década de los 80 resultó en la etapa más fructífera de los PLC.

Estos microcontroladores ya se usaban en forma masiva dentro de la industria de los electrodomésticos. Hoy día es difícil que un electrodoméstico de cierta complejidad, como puede ser una lavadora, no utilice un microcontrolador. Los microcontroladores sustituyeron los programadores con levas, que en el caso de las lavadoras reemplazaban a la lógica secuencial.

Los programadores con levas se componían esencialmente de un tambor que giraba. Este tambor tenía pistas de cobre sobre su superficie. Estas pistas realizaban la secuencia de

lavado al ser movido el cilindro por un motor a muy baja velocidad. La aplicación de estos programadores a la industria tuvo una vida muy efímera, ya que no podían competir con los PLC en cuanto a reprogramabilidad. La única justificación de su uso resultó en su muy bajo costo.

En la década de los 90, la introducción gradual de nuevos protocolos de comunicaciones y la modernización del hardware (en especial el hardware de comunicaciones, con la introducción de CPU tan potentes como el 386 de Intel y la línea 68000 de Motorola) permitió la estandarización de las comunicaciones y del lenguaje de programación.

En particular, la CPU de los PLCs del laboratorio es de la línea PowerQUICC de Freescale, y funciona a 100MHz@32bits.

Los lenguajes de programación fueron normalizados bajo la norma IEC 61131-3. Esta norma define 5 lenguajes: Ladder Logic (LD), Instruction List (IL), Structured Text (ST), Sequential Function Chart (SFC), Function Block Diagram (FBD).

La intención de mantener un mercado cautivo a través de los lenguajes propietarios, y el desinterés de los usuarios por realizar las inversiones necesarias para los cambios, llevaron a que esta norma no se utilizara de forma amplia. La excepción a lo anterior son los fabricantes europeos de PLC, quienes contribuyeron a la redacción de la norma de forma más activa, y la utilizan de forma intensiva.

Hoy día existen fábricas de PLC prácticamente todos los países desarrollados: Modicon, Allen Bradley y General Electric en Estados Unidos, Siemens en Alemania, Telemécanique en Francia, Omrom en Japón, etc. Además, si bien el software utilizado para la programación de los equipos es propio de cada compañía, el lenguaje utilizado es acorde a la norma IEC61131-3.

2 Dentro de un PLC

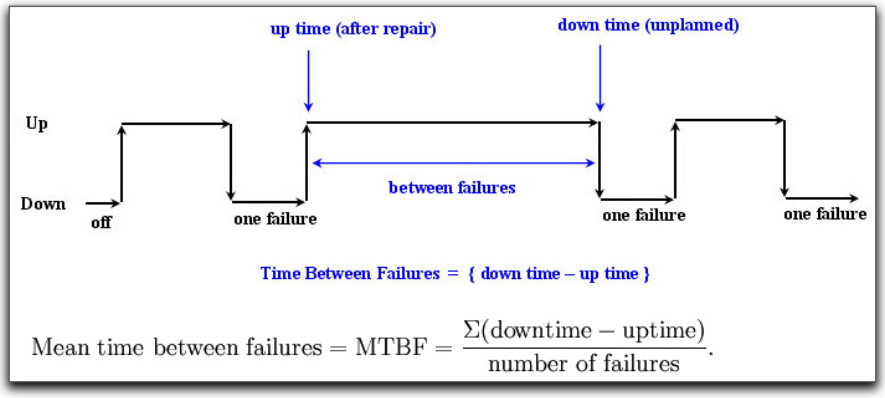
Pero qué es un PLC, cuáles son las entradas/salidas y cómo están constituidos internamente. En la sección anterior, esto fue introducido de forma muy rudimentaria.

Como cualquier computadora, un PLC se compone de una CPU y un módulo de entradas/salidas.

En cuanto a las diferencias con el PC, un PLC presenta una cantidad bastante menor de recursos que un PC. En particular, un PLC no tiene interfaz gráfica. Por otro lado, se supone que el hardware de un PLC es de mejor calidad y está mejor probado que el de un PC, debido a su mayor simpleza.

El sistema operativo de un PLC no admite muchos errores, debido a la función de control de procesos que el PLC tiene. Si por algún motivo un PLC se “cuelga“, se puede crear una situación de extrema gravedad en una fábrica. Pensemos en una refinería de petróleo o una central nuclear, campos pioneros en el uso de PLC y computadoras en todas sus formas.

Lo anterior no quiere decir que un PLC no falle. Simplemente, se supone que diversos factores como la mortalidad infantil de componentes, las pruebas a las que se somete antes de entrar en servicio y la simplicidad del hardware y del software, hacen que un PLC sea más confiable que un PC. Hay casos en que el proceso es llevado por un PC pero necesariamente para comunicarse con el mundo necesita de PLC. Los fabricantes de PLC indican esta característica de robustez con el parámetro "tiempo medio entre fallas" (por sus siglas en inglés MTBF). El PLC utilizado en los laboratorios del curso por ejemplo indica un MTBF de 170 años.



2.1 La CPU del PLC

La CPU se compone de un procesador y memoria. Como fue mencionado, la potencia y el fabricante del procesador varía, según las prestaciones solicitadas al equipo.

Actualmente, los microprocesadores utilizados en la CPU tienden a ser de uso general. Existen PLCs que utilizan en el CPU microprocesadores tan poderosos como el 80386 o 80486 de Intel o la línea 68000 de Motorola. Sin embargo, la mayoría utilizan procesadores menos poderosos.

Físicamente, un PLC tiene tres memorias distintas: una memoria ROM para el sistema operativo, una memoria RAM para los datos, y una memoria Flash para el programa de usuario.

2.2 Entradas / salidas del PLC

El número de entradas/salidas de un PLC, que es acotado, se puede aumentar a través de módulos de expansión o de unidades remotas. De esta forma, físicamente las entradas y salidas lógicas de un PLC se encuentran en el mismo PLC, en módulos de expansión y en unidades remotas.

En general, las entradas y salidas físicas de un PLC tienen bornes comunes respectivos. De esta forma, se disminuye el tamaño de los PLCs, limitado principalmente por el tamaño de las borneras.

2.2.1 Entradas del PLC

Las entradas del PLC se conectan a las entradas físicas del proceso. Las entradas tienen un borne común, por lo que se debe asegurar que las señales que se conecten estén referidas a un único potencial.

Las entradas de un PLC pueden ser digitales o analógicas.

Entradas analógicas

Desde el punto de vista lógico, una entrada analógica consta de un acondicionador de señal, seguido de un conversor A/D. El conjunto de entradas analógicas posibles es amplio, e incluye señales de termocuplas, señales resistencias de platino, estándar 4-20 mA, estándar 1-10 V, comunicación HART, etc.

Normalmente, la precisión de los conversores A/D es de 12 bits.

Entradas digitales

Estas entradas pueden ser señales de switches, sensores de posición, contactos auxiliares de contactores, etc.

Las entradas son acopladas ópticamente a la CPU de forma disminuir la probabilidad de daños en caso de fallas. Normalmente, la aislación es del orden de 1,5 kV_{ef}.

La rotura de una entrada digital es prácticamente imposible de reparar, por lo que el usuario deberá de prever un 10 o 20% de entradas adicionales, para cubrir eventuales ampliaciones del sistema o eventuales roturas de entradas debidas cableados erróneos.

En los PLCs de laboratorio, los umbrales que el definen “0” y el “1” lógicos, son respectivamente 5 y 15V. En la actualidad se encuentran PLCs que admiten entradas de alterna de una amplia gama de tensión (llegan a 220 VAC) a efectos de simplificar el cableado.

La frecuencia de una señal de entrada está limitada por el tiempo de ciclo del PLC. Para casos de entradas muy rápidas, se cuenta con contadores hardware adicionales, que la CPU interroga cíclicamente.

2.2.2 Salidas del PLC

Las salidas del PLC permiten actuar sobre el mundo exterior. Las salidas de un PLC pueden ser digitales o analógicas.

Salidas analógicas

Desde el punto de vista lógico, una salida analógica consta de un conversor D/A seguido de un acondicionador de señal.

Las salidas analógicas pueden salidas de corriente o salidas de tensión. Generalmente, la precisión del conversor D/A es de 12 bits.

Salidas digitales

Pueden ser transistores o relés directamente. Las salidas digitales presentan aislación galvánica, del orden de 1,5 kV_{ef}. Normalmente, las salidas tienen borne común.

2.3 Relés internos o virtuales

La lógica de relés dispone a los relés en un arreglo, en el que unos relés energizan a otros, hasta llegar al accionamiento de dispositivos que actúan directamente sobre el proceso, como motores, solenoides etc.

Con el PLC, todos los relés intermedios se sustituyen por los relés internos del PLC. Los relés internos son relés simulados por el software de usuario que se ejecuta en el PLC. Estos relés internos son los que efectivamente permiten la simplificación de la lógica de relés.

El lenguaje LD, que veremos más adelante, tiene reglas sintácticas tales que un programa escrito en LD se hace muy semejante a un diagrama de lógica de relés.

2.4 Timers

Uno de los elementos más usados en la lógica de relés son los *timers* discretos, es decir, cajas con un potenciómetro (para fijar el valor de un tiempo) y un relé de salida.

En un PLC se pueden tener hasta cientos de timers. Un timer se activa por un contacto interno o virtual del PLC, o por entradas exteriores.

Los tipos de timers de uso más extendido son el DELAY ON y el DELAY OFF.

Los DELAY ON son aquellos cuya salida es activada un tiempo programable después activada la entrada.

Los DELAY OFF son aquellos cuya salida es *desactivada* un tiempo programable después de activada de la entrada.

En la programación de un PLC se utilizan además una gran cantidad de bloques funcionales adicionales. Entre estos, tenemos contadores, monoestables, etc. que serán explicados más adelante con mayor detalle.

2.5 Diagrama de operación de un PLC

La operación de un PLC se basa en un *sistema operativo*. Cada un tiempo prefijado por el programador, denominado *tiempo de ciclo*, se inicia un nuevo *ciclo* del sistema operativo Figura 5. En cada ciclo, el sistema operativo ejecuta secuencialmente las tres tareas que siguen:

1) Actualización de las entradas físicas y de comunicaciones.
2) Ejecución de programa de usuario.
3) Actualización de las salidas físicas y de comunicaciones.

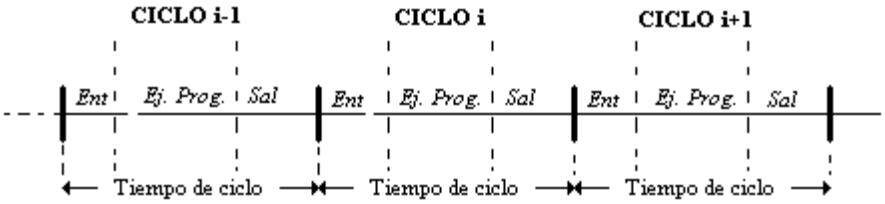


Figura 5

De esta forma no hay cambios inesperados durante la ejecución del programa: la actualización de entradas/salidas se realiza en forma secuencial con el programa, pero no en el momento de ejecución.

Los tiempos de ciclo más comunes son 10, 50, o 100 mseg. Para aplicaciones de control y monitoreo de las redes eléctricas, que requieren de un tiempo de ciclo mucho menor que los 20 mseg. de la red, hay PLC más rápidos, con tiempo de ciclo mucho menor que 1 mseg.

2.6 Tiempo de respuesta

Un aspecto de gran importancia es el tiempo de respuesta de un PLC. Definimos por tiempo de respuesta al tiempo que transcurre desde que cambia una entrada, hasta que cambia la salida que corresponda.

Normalmente, el mínimo tiempo de respuesta es un ciclo de PLC, debido a la forma de ejecución del sistema operativo. En un peor caso, el tiempo de respuesta es de dos tiempos de ciclo. Esto sucederá en caso que la entrada cambie inmediatamente después de terminar la tarea de actualización de entradas (Figura 6).

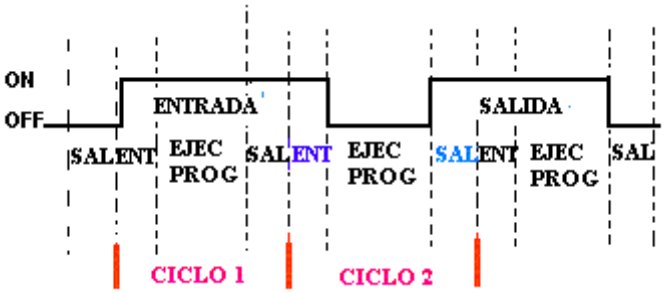


Figura 6

Se deduce que el ancho mínimo de un pulso de entrada para que pueda ser reconocido por el PLC es un tiempo de ciclo. En la Figura 7, el pulso 3 no será visto por el PLC.

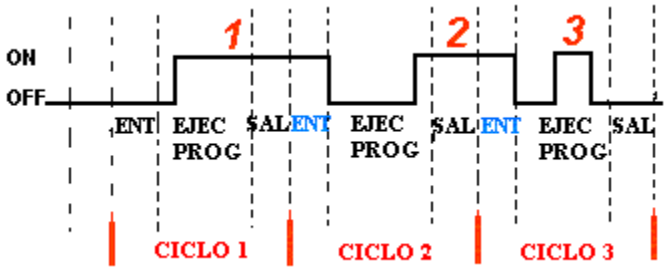


Figura 7

Para el caso que se utilicen entradas rápidas - por ejemplo, en aplicaciones de contadores en los que los pulsos a contar sean mucho más rápidos que el tiempo de ciclo del PLC- se dispone de entradas especiales. Una entrada rápida puede llegar a los 10 KHz.

En caso que una entrada necesite una atención especial se dispone también de interrupciones, análogas a las de cualquier procesador.

2.7 Configuración del PLC

El funcionamiento del PLC viene dado por su configuración. En general, el usuario configura al PLC desde un PC o una consola. Los parámetros de la configuración quedan almacenados en memoria no volátil del PLC.

En general, una configuración determina, entre otros:

1) el tiempo de ciclo
2) si la unidad maestra, esclava o *stand-alone* (capítulo “Comunicaciones”).

3 Entorno de operación del PLC

En general, un PLC funciona en uno de 3 modos de operación:

1) aislado
2) en bus "de datos"
3) formando parte de uno o más buses de comunicaciones

Un sistema con un PLC operando en bus de datos consta de un PLC y un conjunto de extensiones de entrada/salida anexadas. El PLC accede a las extensiones a través del bus de datos. Las extensiones se direccionan así como periféricos del CPU del PLC.

En general, las extensiones pueden ser digitales o analógicas. Las extensiones digitales son aquellas que tienen entradas/salidas digitales, y las extensiones analógicas son aquellas que tienen entradas/salidas analógicas.

Un PLC conectado a un bus de comunicaciones forma parte de un sistema distribuido, en el que pueden haber otros PLCs, otros PCs, y unidades remotas de entrada/salida. El bus de comunicaciones consta en general de uno o dos pares trenzados y un cable de referencia, compartidos por todas las unidades. En general, la comunicación se realiza a

través de protocolos maestro –esclavo. En un protocolo de este tipo, una unidad prefijada, denominada unidad maestra, se comunica cíclicamente con todas las demás unidades en el bus, denominadas unidades esclavas. Este tipo de arquitectura *descentralizada* (denominada comúnmente como Distributed Control System, DCS) permite implementar sistemas de control instalando la unidad central en el panel de control, y las unidades de entrada / salida en la vecindad de los sensores y actuadores.

De forma análoga a las extensiones, distinguimos las unidades remotas de entrada/salida en unidades digitales y unidades analógicas. Las unidades digitales tienen entradas/salidas digitales, y las unidades analógicas tienen entradas/salidas analógicas.

Desde el punto de vista del usuario, las entradas/salidas de las unidades del sistema se mapean como datos (ver capítulo 2) de la memoria del PLC maestro (en caso de que el maestro sea un PLC y no un PC), en una zona denominada memoria de entrada / salida. Quien se encarga de realizar la transferencia de datos con las demás unidades para actualizar el contenido de esta memoria es el sistema operativo del PLC maestro. Esta actualización se realiza ciclo a ciclo del sistema operativo del PLC maestro, de forma transparente al usuario. Los datos de entrada se actualizan en el ciclo de actualización de entradas del PLC, y las salidas de las unidades esclavas se actualizan en el ciclo de actualización de salidas del PLC. Desde el punto del programa de usuario, resulta transparente si el sistema está comunicado a través de un bus de datos o a través de un bus de comunicaciones.

CAPÍTULO 2: TIPOS DE DATOS

1 La memoria del PLC

La memoria del PLC contiene los programas y los datos necesarios para la operación del PLC. La memoria se divide en tres partes:

1) Memoria no accesible al usuario: esta parte de memoria contiene los programas y datos del sistema operativo del PLC.
2) Memoria de programa de usuario y de configuración: esta parte de memoria sólo puede ser cambiada utilizando una unidad de programación.
3) Memoria accesible: esta memoria es de lectura/escritura. Contiene el espacio de entradas, el de salidas, y el de datos.

La memoria accesible se divide en un espacio de memoria de entrada/salida, y un espacio de memoria de lectura escritura.

El espacio de entrada/salida del PLC se mapea en la memoria de entrada/salida. Este espacio se actualiza en los intervalos de entrada/salida del ciclo del PLC.

2 Los tipos de datos del PLC

Las localidades de la memoria accesible se agrupan formando datos. El PLC maneja 4 tipos de datos: datos binarios, palabras, palabras dobles, flotantes.

2.1 Bits

Un dato del tipo binario (bit) consiste de un dígito binario. Puede tomar uno de los dos valores TRUE o FALSE. Estos datos se utilizan en operaciones lógicas. En el programa suelen indicarse como BOOL o con el prefijo X.

2.2 Palabras

Son números de dos bytes. Toman valores enteros en el rango –32524 a + 32524. El tipo palabra es similar al tipo *int* de los lenguajes corrientes. Se utilizan en operaciones matemáticas (producto, suma, resta, división *entera*) y operaciones de comparación. En el programa las palabras suelen indicarse como WORD o con el prefijo W y los enteros como INTEGER compartiendo el mismo prefijo que las palabras.

2.3 Palabras dobles

Son números de 4 bytes con signo. Las operaciones que se les puede aplicar son similares a las de palabras. El tipo palabra doble se denomina *long* en los lenguajes corrientes.

Pueden tomar valores de un rango muy amplio (-2^{31} a 2^{31}), lo que permite utilizarlos en divisiones racionales con buena precisión. Esto puede ser necesario si no se cuenta con el tipo *flotante* y se quiere hacer una conversión de unidades de medida. Existen funciones que convierten palabra en palabra doble.

En el programa suelen indicarse como DWORD, DOUBLE WORD o con el prefijo DW. Además pueden definirse los enteros dobles o DOUBLE INTEGER compartiendo el mismo prefijo DW.

2.4 Flotantes

Son representaciones de números reales. En general, las representaciones son de 4 o de 8 bytes, dependiendo del formato. Por ejemplo, el formato IEEE 754 utiliza 4 bytes para representar números reales en el rango $\pm 1.1754944 \times 10^{-38}$ a $\pm 3.4028238 \times 10^{+38}$.

Actúan como operandos de funciones analíticas habituales, como seno, coseno, etc.

En el programa suelen indicarse como FLOAT o REAL.

3 Variables y constantes

Un dato puede ser una *variable* o una *constante*.

El valor de una constante no puede ser modificado desde el programa de usuario. Su valor se fija cuando se define la constante (en tiempo de programación).

El valor de una variable puede ser modificado desde el programa de usuario. En principio, no tiene un valor determinado al comenzar a correr el programa.

4 Direcciones de los datos

La forma más común de referirse a un dato del PLC es a través de su dirección. Una dirección puede referir a una entrada, una salida o un dato en memoria.

Las variables siempre tienen direcciones accesibles al usuario. Sin embargo, hay dos tipos de constantes: el grupo de las constantes con direcciones accesibles y el grupo de las constantes sin direcciones accesibles (definidas en los PLC del laboratorio con el prefijo #).

Desde el punto de vista sintáctico, una dirección se compone (pág. 292 de B.Morriss) de un código de dos letras, seguido de uno o más campos de número. En general, la primer letra es una de las siguientes:

I ó E	para una dirección de memoria de entrada
Q u O	para una dirección de memoria de salida
M	para una dirección interna de memoria de datos

y la segunda letra del código es una de las siguientes:

X	bit	(se supondrá bit en el caso que no haya segunda letra)
	B	byte (8 bits)
	W	word (16 bits)
	D	double word (32 bits)
	L	long word (64 bits)

4.1 Direcciones de datos binarios

Las referencias a datos binarios internos son de la forma:

M XX.YY.

El valor máximo de XX depende del PLC. YY toma valores enteros en el rango 0 a 15. Ejemplo: M 01.14.

Las entradas/salidas digitales del PLC son variables binarias. En los PLC del laboratorio, las entradas se refieren por I XX.YY y las salidas por O XX.YY.

En los PLC del laboratorio, tenemos también dos bits constantes pre definidos, referidos como TRUE y FALSE.

4.2 Direcciones de palabras

Las palabras internas se refieren por:

MW XX, YY,

XX e YY dependen del PLC.

Las entradas salidas de conversores A/D son palabras. En los PLC del laboratorio, las entradas de conversores se refieren por IW XX.YY y las salidas OW XX.YY.

En los PLC del laboratorio, tenemos también palabras constantes, que se refieren por KW XX.YY. Un grupo de constantes KW definen la configuración del sistema.

4.3 Direcciones de palabras dobles

Las palabras dobles internas se refieren por:

MD XX, YY

XX e YY dependen del PLC.

En los PLC del laboratorio, tenemos también palabras dobles constantes, que se refieren por KD XX.YY.

4.4 Resumen para los PLCs del laboratorio

La siguiente tabla vale para los PLCs del laboratorio:

Espacio en memoria del dato	Tipo del dato	Denominación

Memoria de Entrada	Binario	I XX,YY
	Palabra	IW XX,YY
	Palabra Doble	--
Memoria de datos (internos)	Binario	M XX,YY
	Palabra	MW XX,YY
	Palabra Doble	MD XX,YY
Memoria de Salida	Binario	O XX,YY
	Palabra	OW XX,YY
	Palabra Doble	--

5 Memoria correspondiente a extensiones y unidades remotas

Como se dijo en el capítulo 1, desde el punto de vista del usuario, las entradas/salidas de las unidades del sistema, se mapean como datos de la memoria de entrada/salida del PLC central.

Consideremos un sistema con un PLC central y unidades remotas o extensiones. Referida al ciclo del PLC central, la operación del sistema transcurre de la siguiente manera:

1) En la etapa de actualización de entradas: transferencia de los valores de las entradas de cada una de las unidades remotas o extensiones a la memoria de entrada que corresponde en el PLC central. Como fue dicho en el capítulo 1, esta transferencia puede ser vía bus de datos o vía bus de comunicaciones, y es manejada a nivel de sistema operativo.
2) Ejecución de programa de usuario: el programa de usuario utiliza los valores cargados en 1), junto con los demás datos de la memoria, para actualizar la memoria de salida.
3) En la etapa de actualización de salidas: transferencia desde memoria de salida del PLC central a las salidas de las extensiones o las unidades remotas. A cada unidad se transfiere la zona de memoria que le corresponde. Una vez más, esta transferencia puede ser vía bus de datos o vía bus de comunicaciones, y es manejada a nivel de sistema operativo.

La correspondencia entre las entradas/salidas de las unidades remotas o extensiones y la memoria de entrada/salida del PLC central depende del fabricante. En los PLCs del laboratorio, la zona de memoria correspondiente a una extensión depende de la ubicación física de la extensión, y la zona de memoria correspondiente a una unidad remota del bus CS-31 se configura por hardware en la unidad remota.

CAPÍTULO 3: LENGUAJE LADDER

1 Introducción

Como vimos en el capítulo 1, los PLCs fueron introducidos en el mercado con el objetivo de sustituir en la industria las lógicas de control basadas en relés.

Los sistemas de control basados en lógica cableada de relés presentaban dos problemas: mantenimiento y flexibilidad. Por un lado, requerían de un costo de mantenimiento importante (fallas en dispositivos electro-mecánicos) y por otro lado cualquier modificación a la lógica requería recablear el tablero correspondiente.

Veamos un ejemplo: tenemos un motor que debe ser encendido si cualquiera de dos llaves se accionan. Ambas llaves operan sobre 24 Vdc y el motor es un motor de 220 VAC monofásico. Un diagrama muy simplificado del plano eléctrico de dicha instalación podría ser el de la Figura 8:

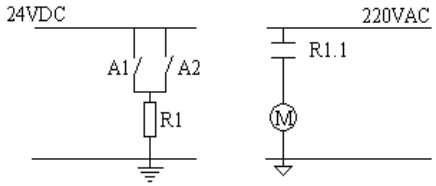


Figura 8

Si luego se resuelve que por ej., por razones de seguridad, ambas llaves deben estar ON para dar la señal de arranque al motor, se debe recablear el tablero para lograrlo. El plano eléctrico correspondiente podría ser como el de la Figura 9:

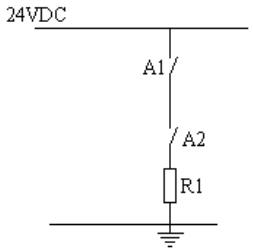


Figura 9

Cada cambio introducido obliga al recableado. Asimismo, a mayor número de relés, existe mayor probabilidad de contactos sucios, flojos, etc., o en el fin de su vida útil y por tanto mayor probabilidad de falla y mayores costos de mantenimiento.

En este contexto, la introducción de un PLC procura:

1) La eliminación de los relés intermedios.

2) Aumentar la flexibilidad del sistema, permitiendo la modificación de la lógica por software.

Este cambio trae aparejado un problema. Normalmente quienes realizan el mantenimiento de una planta industrial son electricistas muy entrenados en leer planos eléctricos y en detectar fallas en los tableros eléctricos. La sustitución del tablero por software hará necesaria la capacitación de los electricistas de la planta en el software.

Este hecho llevó a diseñar un lenguaje de programación que pudiera ser fácilmente leído y entendido por los electricistas de la planta con un mínimo entrenamiento: el lenguaje LADDER. Así, LADDER es un lenguaje cuya sintaxis es extremadamente similar a la de un plano eléctrico.

El plano eléctrico simplificado de la Figura 8, se vería en ladder como la Figura 10.

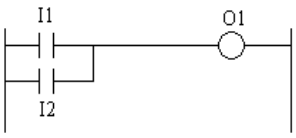


Figura 10

Y el PLC estaría cableado esquemáticamente por ejemplo como en la Figura 11:

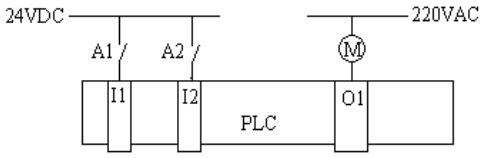


Figura 11

Los contactos de las llaves A1 y A2, que antes se cableaban a la bobina de un relé, se cablean ahora a entradas digitales de 24Vdc del PLC. Una salida del PLC (O1) se cablea al motor. La lógica que activa la salida para arrancar el motor, es realizada dentro del PLC. El PLC lee las entradas I1 e I2 (cableadas a A1 y A2) en cada scan del programa, copiando a una posición en memoria un 1 si entrada leída estaba activada y un cero en caso contrario. A su vez, en cada scan ejecuta la lógica programada y actualiza las salidas con el resultado correspondiente. Para que el sistema funcione como el de la Figura 9, alcanza sólo con modificar el software del PLC de forma que los dos contactos queden en serie, sin necesidad de recablear.

2 Estructura básica de un programa LADDER

El programa Ladder se ejecuta cíclicamente de arriba hacia abajo y de izquierda a derecha. En cada escalón (rung) se evalúan las condiciones lógicas a la izquierda del renglón. Si la serie de las condiciones lógicas del renglón es verdadera (lo que indica que hay conexión eléctrica de izquierda a derecha en el símil eléctrico) se actualizan las salidas ubicadas en el extremo derecho del escalón. En general cada escalón debe tener

al menos una salida en el extremo derecho, aunque es posible que no existan condiciones en la entrada (Figura 12).

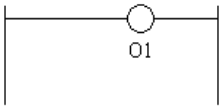


Figura 12

En este caso en cada scan del programa, O1 se setea incondicionalmente a 1 (el escalón es siempre verdadero).

3 Símbolos, direcciones y operaciones básicas

Previo a describir las diferentes instrucciones, se menciona que los diferentes fabricantes de PLCs difieren en la implementación y la nomenclatura de las instrucciones. En este curso se utiliza la nomenclatura asociada a los PLCs del laboratorio del curso, que en algunos aspectos es estándar y que se entiende fácilmente comprensible. Para cada fabricante particular se deberá ver en el correspondiente manual la forma en que se implementa cada instrucción.

Una entrada (contacto) se representa por el siguiente símbolo:

Figura 13

Una salida (bobina de un relé) se representa por el siguiente símbolo:

Figura 14

Cada símbolo tiene asociada una dirección en la memoria del PLC, en la que el PLC almacena el valor del bit correspondiente. Según el capítulo anterior, existen tres grandes áreas de datos en la memoria del PLC: entradas, salidas y de datos. Las entradas son bits de sólo lectura, que el PLC actualiza en cada scan copiando el estado de las entradas. Las salidas son bits en un área de lectura/escritura. El PLC lee los bits de salida en cada scan para actualizar el valor de las salidas. Los bits intermedios son variables de lectura/escritura, que podemos usar para construir la lógica del programa del PLC (equivalen a los relés auxiliares en la lógica cableada).

En el capítulo anterior se explica la forma de referir a las distintas variables según la dirección en la memoria del PLC. Normalmente, es posible dar nombre (LABELS) a las variables de forma de poder referirlas de manera más comprensible.

Las operaciones básicas que es posible realizar son el AND y el OR lógicos, representados en la Figura 15 y Figura 16.

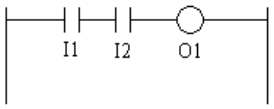


Figura 15

AND: se setea O1 si I1=1 y I2=1.

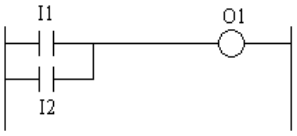


Figura 16

OR: Se setea O1 si I1=1 o I2=1.

4 Instrucciones básicas con bits

Direct Contact (o LOAD): es un contacto normalmente abierto, es decir es Verdadero si el bit está en 1 (Figura 17).

Figura 17

Inverted Contact (o LOADBAR): es un contacto normalmente cerrado, es decir es verdadero si el bit está en 0 (Figura 18).

Figura 18

Direct Coil (o OUT), es análogo a la bobina de un relé, es decir si el resultado del escalón (rung) en que se encuentra es Verdadero seteará el bit correspondiente a 1. Si el escalón en que se encuentra es falso, seteará el bit a 0 (Figura 19).

Figura 19

Inverted Coil (o OUTBAR) Si el resultado del escalón en que se encuentra es Verdadero, seteará el bit correspondiente a 0. Si el escalón es Falso, seteará el bit correspondiente a 1 (Figura 20).

Figura 20

Set Coil (o LATCH): Es una instrucción de salida de tipo retentivo que sólo puede setear un bit a 1. Es decir si el escalón es Verdadero seteará el bit correspondiente a 1 si el escalón es falso no hará nada. Usualmente es utilizada en conjunción con la instrucción Reset Coil (Figura 21).

Figura 21

Reset Coil (o UNLATCH): Es una instrucción de salida de tipo retentivo que sólo puede setear un bit a 0. Es decir si el escalón es Verdadero seteará el bit correspondiente a 0, si el escalón es falso no hará nada. Usualmente es utilizada en conjunción con la instrucción Set Coil (Figura 22).

Figura 22

Nótese la diferencia entre las intrucciones Set Coil y Direct Coil. Una vez que el bit fue seteado en 1 con Set Coil, sólo puede ser reseteado con una instrucción Reset Coil. El bit vuelve automáticamente a 0 si el rung es falso con la instrucción Direct Coil. Como ejemplo de lo anterior, consideremos una entrada del PLC conectada a un contador de pulsos electromagnético (un imán y un contacto que se cierra cuando el imán pasa por frente al contacto). Estos contadores habitualmente tienen rebotes cuando el contacto se cierra. Si estos rebotes tienen una duración tal que pueden ser vistos por el scan del PLC (y la entrada del PLC no filtra adecuadamente estos rebotes), el bit de entrada cambiará sucesivamente de 1 a 0 hasta que finalicen los rebotes. Usando una instrucción Direct Coil sobre un bit auxiliar para contar los pulsos, se contarán pulsos ficticios a causa de los rebotes (Figura 23 y Figura 24).

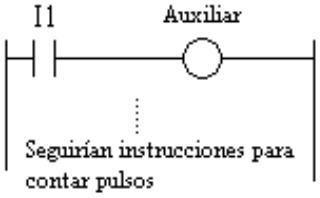


Figura 23

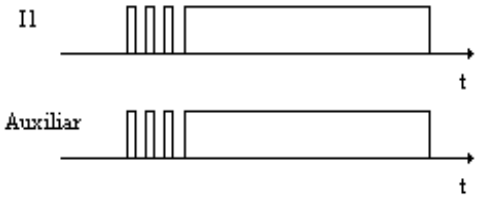


Figura 24

Una forma de solucionar este problema es usar un Set Coil sobre el bit auxiliar en lugar del Direct Coil, seguido de un Reset Coil cuando la entrada vuelva a 0 pero luego de temporizar para filtrar los rebotes. De esta forma se filtran los rebotes (Figura 25 y Figura 26).

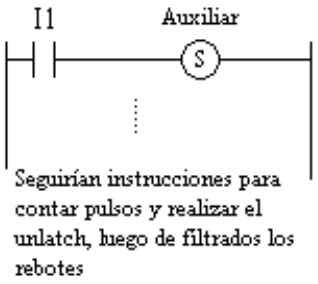


Figura 25

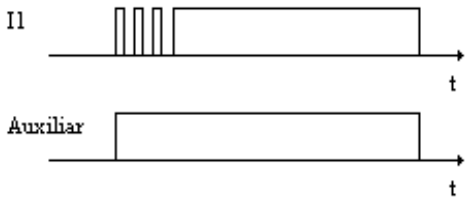


Figura 26

5 Timers y contadores

5.1 El bloque funcional

Desde el punto vista del programador, los contadores y timers son ejemplos de bloques funcionales. Un bloque funcional es un objeto gráfico que se representa por un rectángulo, con puntos de conexión de entradas, puntos de conexión de salidas y un identificador. El identificador describe función a la que refiere el bloque. Las entradas de la función se conectan a los puntos de conexión de entradas, en el borde izquierdo del rectángulo. Las salidas de la función se conectan a los puntos de conexión de salidas, en el borde derecho del rectángulo. La figura representa un bloque funcional que implementa la función FUN, con dos entradas y dos salidas:

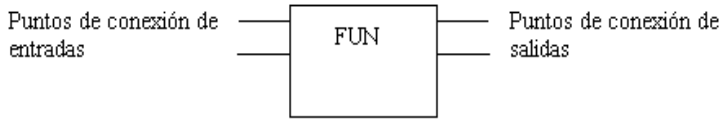


Figura 27

Las entradas y salidas son datos. El tipo de estos datos depende del bloque funcional. Los datos de entrada y de salida de un bloque se conectan por intermedio de líneas de conexión.

5.2 Contadores

Los contadores son instrucciones simples, pero cada fabricante utiliza diferentes formas de representarlos y de nombrarlos. Existen tres tipos básicos de contadores:

UP- Counters: en el PLC del laboratorio se denominan CTU
DOWN - Counters: el PLC del laboratorio no tiene; normalmente se denominan CTD
UP-DOWN Counters: en el PLC del laboratorio se denominan VRZ

El rango del contador depende del fabricante. En el caso de los PLCs del laboratorio, CTU cuenta de 0 a 32767 y VRZ cuenta de –32768 a 32767.

Para analizar la operación de un contador se considera el contador tipo UP CTU del PLC del laboratorio, que se muestra en la figura.

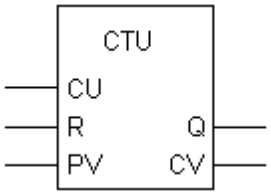


Figura 28

La cuenta actual del contador se mantiene en una variable interna denominada genéricamente “Acumulador”. La entrada tipo bit CU, que se denomina genéricamente “Pulso”, se conecta al tren de pulsos que se cuentan. La entrada tipo bit R, que se denomina genéricamente Reset, escribe 0 en el acumulador. La entrada tipo word PV determina el límite máximo de la cuenta. La salida tipo bit Q, genéricamente denominada “Done”, indica si la cuenta en el acumulador es mayor o igual que el límite máximo. La salida tipo palabra “CV” contiene la cuenta actual del contador.

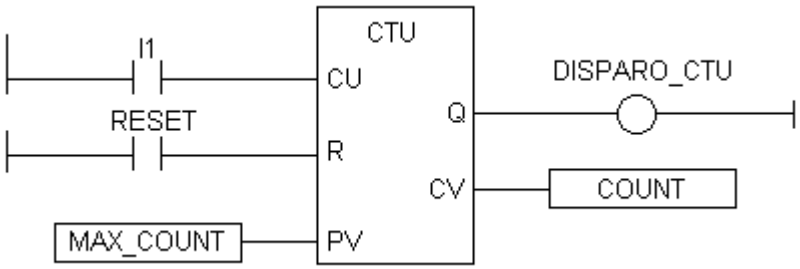


Figura 29

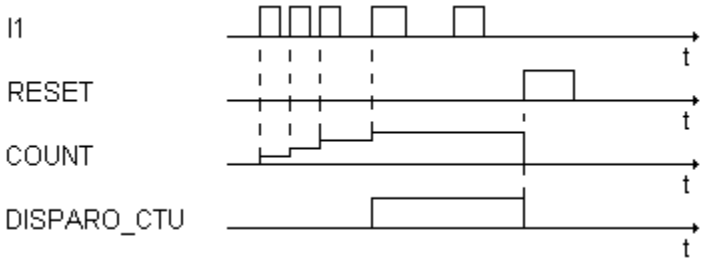


Figura 30

Diagrama de tiempos para un contador UP (Preset = MAX_COUNT = 4). Se observa que el acumulador se incrementa a intervalos de tiempo variables (que dependen de los pulsos en I1).

Un contador UP conectado según Figura 29, tiene un funcionamiento como el de la Figura 30. Los contadores UP_DOWN son similares pero tienen tres entradas: UP, DOWN y RESET.

5.3 Timers

Los Timers son instrucciones destinadas a esperar un cierto intervalo de tiempo antes de "hacer algo". Habitualmente los diferentes fabricantes proveen tres tipos básicos de timers (aunque con nomenclatura muy variada):

Timer On delay (retardo en el encendido): en el PLC del laboratorio se denominan ESV y TON. Luego que el rung en el que se encuentra pasa a verdadero durante x segundos, el timer pasa el bit de salida asociado a 1.

Timer Off delay (retardo en el apagado): en el PLC del laboratorio se denominan ASV y TOFF. Luego que el rung en el que se encuentra pasa a falso durante x segundos, el timer pasa el bit de salida asociado a 0.

Timer retentivos: no existen en el PLC del laboratorio. Los timers tipo "On delay" y "Off delay" se resetean si el rung correspondiente pasa respectivamente a falso (TON) o a verdadero (TOF), aún durante el período de conteo del retardo. Un timer retentivo cuenta el tiempo que el rung es verdadero, congelando la cuenta con cambios de verdadero a falso del rung. Esta propiedad hace que sea necesario además una entrada de reset.

Para analizar la operación de un timer se considera el timer On Delay ESV del PLC del laboratorio, que se muestra en la figura.

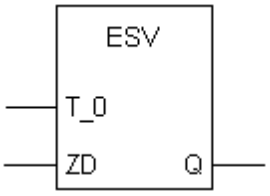


Figura 31

La cuenta del tiempo se mantiene en una variable interna denominada genéricamente "Acumulador". La entrada tipo bit T_0, que se denomina genéricamente "Habilitación", se conecta al pulso que el timer retarda. La entrada tipo dword ZD, que se denomina genéricamente "Preset", determina el valor que tiene que alcanzar el acumulador para que se ejecute la acción del timer. La salida tipo bit Q, genéricamente denominada "Done", indica la expiración del tiempo del retardo.

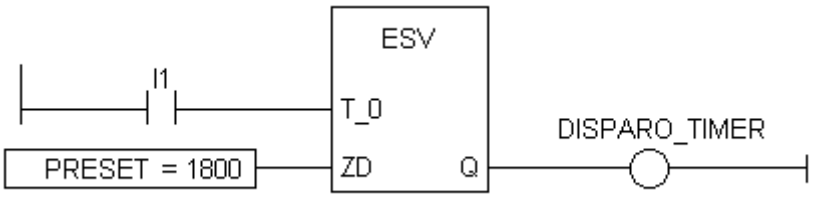


Figura 32

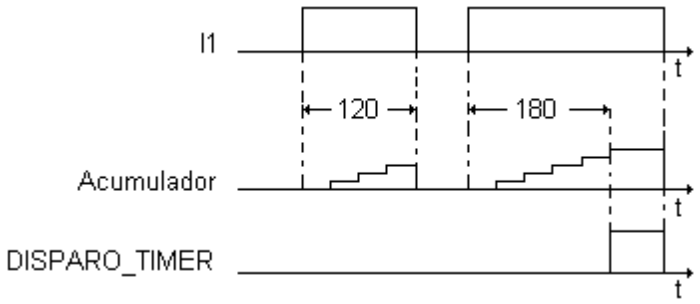


Figura 33

Diagrama de tiempos para un ESV, conectado según Figura 32. Obsérvese que el acumulador se incrementa a intervalos de tiempo iguales.

El comportamiento de ASV es similar. Si en el ejemplo anterior el timer es un ASV, retarda el pasaje de I1 de 1 a 0 si la entrada vale 0 por más de 180 s.

El diagrama de tiempos de un Timer retentivo conectado según la Figura 34 (denominado de forma genérica "RT"), se muestra en la Figura 35.

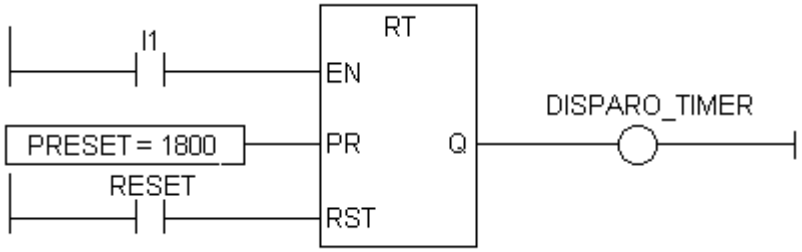


Figura 34

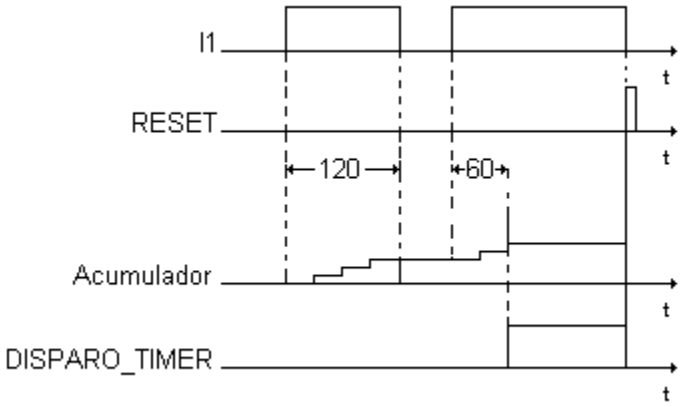


Figura 35

Es importante destacar que la cuenta del timer depende sólo del sistema operativo, mientras la cuenta del contador depende de la ejecución del programa. Esto es, *la cuenta del timer es independiente de la ejecución del* programa, mientras la cuenta del contador *sí depende de la ejecución del programa*.

5.4 Precisión de Timer

Como el tiempo de scan de algunos PLCs es del orden de los milisegundos, es importante tener en cuenta la precisión del timer si se desea contar tiempos de este orden.

Existen diferentes fuentes de error. Las más importantes son:

- Errores en la entrada: si la entrada que se conecta a la habilitación pasa a 1 durante la ejecución del programa, el programa no la verá hasta el próximo ciclo de ejecución del programa, luego de la actualización de las entrada / salida. El error será máximo e igual a 1 tiempo de ciclo

- Errores en la salida. Sucede lo mismo que en el caso anterior con la salida conectada a Q. Por ejemplo, si el timer vence mientras el PLC actualiza las salidas, no se

actualizará la salida retardada hasta que se ejecute nuevamente el programa, se vea el timer DONE, y se setee a 1 el bit correspondiente, y luego se actualice la salida. El error máximo es igual a (el timer vence inmediatamente después de la ejecución de la instrucción del timer) 1 tiempo de ciclo.

Combinando los dos errores anteriores, el error máximo desde el momento de transición de una entrada hasta el momento de transición de la salida temporizada correspondiente es 3 veces el tiempo de ciclo. Es decir por ej. que por más que un fabricante tenga timers con incrementos de 1 ms, un tiempo de ciclo de 5 ms puede llevar a un error en el tiempo de hasta 15 ms.

A los errores anteriores se suman los errores provenientes de filtrado en las entradas y del tiempo físico en que un contacto se cierra en una salida o un transistor se pone ON.

Algunos PLCs implementan además los llamados "high speed counters". Estos contadores son contadores "hardware" y por lo tanto no dependen de los tiempos de scan del PLC.

6 Shift register

Entre otras cosas, permiten almacenar eventos en un área de memoria por el desplazamiento o la rotación de los bits individuales del área de memoria. En la familia de PLCs que integran los PLCs del laboratorio la instrucción se denomina SHIFT, y se representa en la Figura 36. Se hace notar que esta instrucción no está disponible en los PLCs del laboratorio.

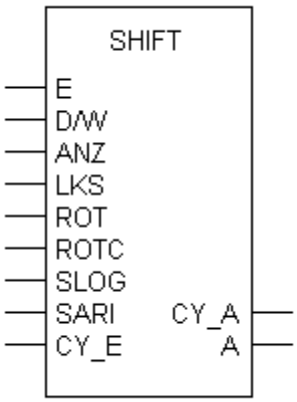


Figura 36

Los parámetros utilizados por la instrucción SHIFT del PLC del laboratorio son:

- E: Entrada tipo word o dword que contiene el operando que se desplaza o rota (en la aplicación de ejemplo al inicio de esta sección, zona de memoria donde se almacenan los eventos)
- D/W: Entrada tipo bit que especifica el tipo de dato de E
- ANZ: Entrada tipo word, que especifica el número de posiciones del desplazamiento o la rotación

- LKS: Entrada tipo bit, que especifica la dirección del desplazamiento o de la rotación
- ROT: Entrada tipo bit, que especifica que la operación es una rotación
- ROTC: Entrada tipo bit, que especifica que la operación es una rotación incluyendo la bandera CARRY
- SLOG: Entrada tipo bit, que especifica que la operación es un desplazamiento lógico
- SARI: Entrada tipo bit, que especifica que la operación es un desplazamiento aritmético
- CY_E: Entrada tipo bit, que especifica el valor inicial del bit CARRY FLAG. Este bit interviene en la operación de desplazamiento o rotación según se especifica más abajo.

- CY_A: Salida tipo bit, que contiene el valor del CARRY FLAG después de la ejecución de la operación
- A: Salida tipo word o dword que contiene el resultado de la operación de desplazamiento o rotación.

De acuerdo a la especificación más arriba, el bloque permite la ejecución de distintas operaciones de desplazamiento o rotación sobre las entradas E y CY_E. Las operaciones ROT y ROT_C se muestran en la Figura 37, y la operación SLOG en la Figura 38.

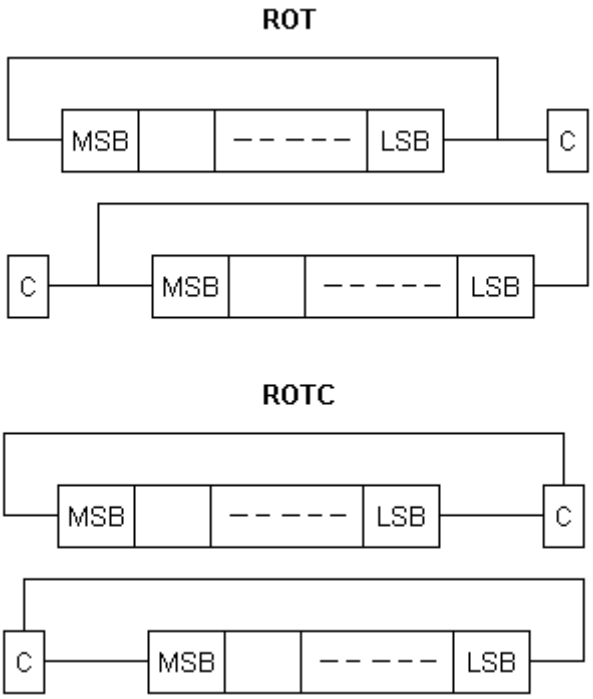


Figura 37

Figura 38

Aparte de shift registers de bits, algunos fabricantes proveen también de instrucciones de shift registers de palabras (16 bits por ej.), que permiten implementar buffers FIFO o LIFO con valores. Estas instrucciones permiten realizar filtros de datos con promedios deslizantes.

7 Instrucciones de control de flujo

Un número importante de fabricantes, aunque no todos, permite la implementación de instrucciones de control de flujo en un programa ladder. Así son comunes instrucciones tipo JSR (jump to subroutine) o instrucciones tipo GoTo, que en determinadas condiciones permiten no ejecutar (saltar) una parte del código ladder o implementar loops.

La instrucción JUMP disponible en el PLC del laboratorio es un salto condicional al valor TRUE de un bit. Se define por:

- Jump label: Posición del programa donde se salta
- Jump symbol: Símbolo asociado a la instrucción JUMP
- Bit que define el salto

La Figura 39 muestra el esquema de una instrucción de salto en el PLC del laboratorio.

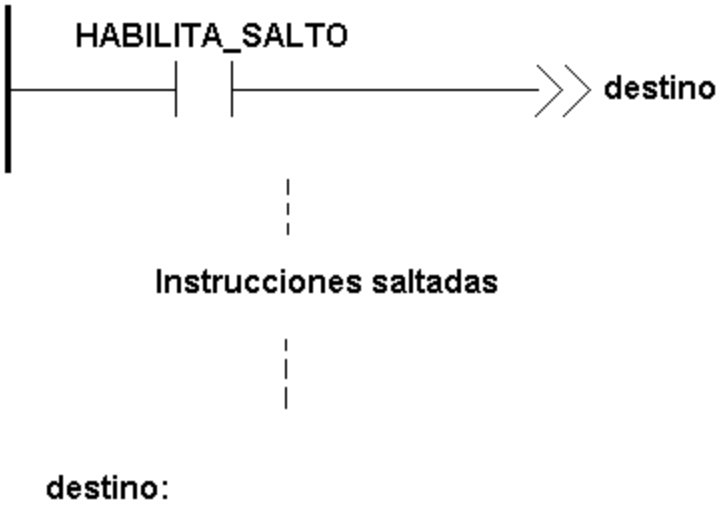


Figura 39

8 Otras instrucciones

Existen muchos tipos de instrucciones. La cantidad de instrucciones disponibles varía de un PLC a otro y de un fabricante a otro. Varias se estudian en otras partes de este curso. Desde el punto de vista de un programa Ladder se puede clasificar estas instrucciones en dos clases:

- Instrucciones de Entrada
- Instrucciones de Salida

Las palabras "entrada" y "salida" refieren en este contexto a la zona donde se ubican en el "rung" de un programa ladder. Se denomina instrucciones de entrada a aquellas que permiten evaluar si el "rung" es verdadero o falso e instrucciones de salida a aquellas que ejecutan acciones dependiendo del resultado de un rung.

Como ejemplo se mencionan los siguientes:

- Instrucciones de Entrada:
  - Instrucciones de Comparación: Igual (EQU), Mayor (GRT), etc. Tienen resultado verdadero o falso según las entradas tipo palabra sean iguales, una mayor que la otra, etc.

- Instrucciones de Salida:
  - Operaciones aritméticas o lógicas: ADD, AND, MUL, etc. Tienen entradas tipo palabra y salidas tipo palabra
  - Operaciones de movimiento de memoria: permiten copiar áreas de memoria
  - Funciones de control PID: implementan controladores PID
  - Funciones de comunicaciones: permiten el intercambio de mensajes entre PLCs.

9 Ejemplo

Veamos el siguiente ejemplo: se desea escribir un programa que controle el encendido - apagado de una bomba.

La bomba será encendida si:

1) Se pulsa el botón de arranque.
2) La protección térmica está deshabilitada.
3) Está abierto el botón de emergencia.
4) Está abierto el botón de parada.

Desde un tiempo T después del encendido, no puede haber ni sobre corriente ni baja corriente. Expresado de otra forma, desde un tiempo T después del arranque, la corriente I debe cumplir $I_{MIN} < I < I_{MAX}$, siendo I_{MIN} e I_{MAX} límites prefijados.

El motor de la bomba se apagará si:

1) Se pulsa el botón de parada.
2) Se cierra la protección térmica.
3) Se pulsa el botón de emergencia.
4) Los límites de corriente no son los correctos.

El programa LD que cumple lo anterior es el siguiente:

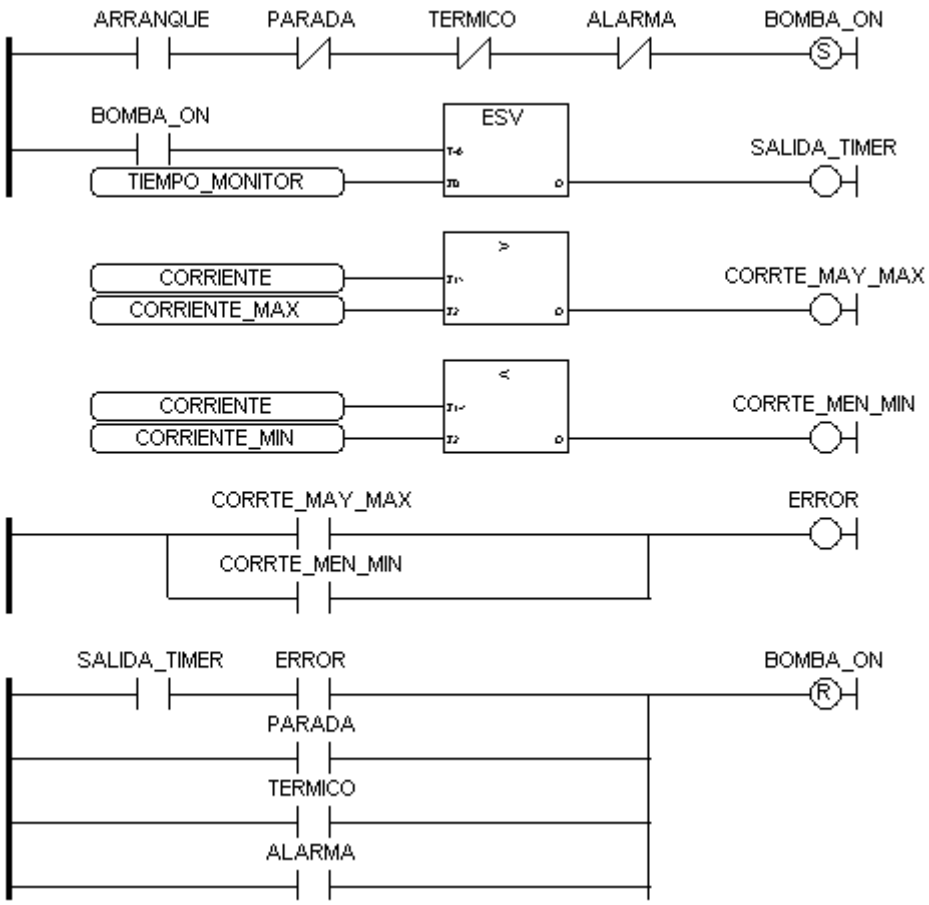


Figura 40

La Figura 41 muestra la conexión del sistema, y la relación entre los distintos elementos del circuito y las variables del programa.

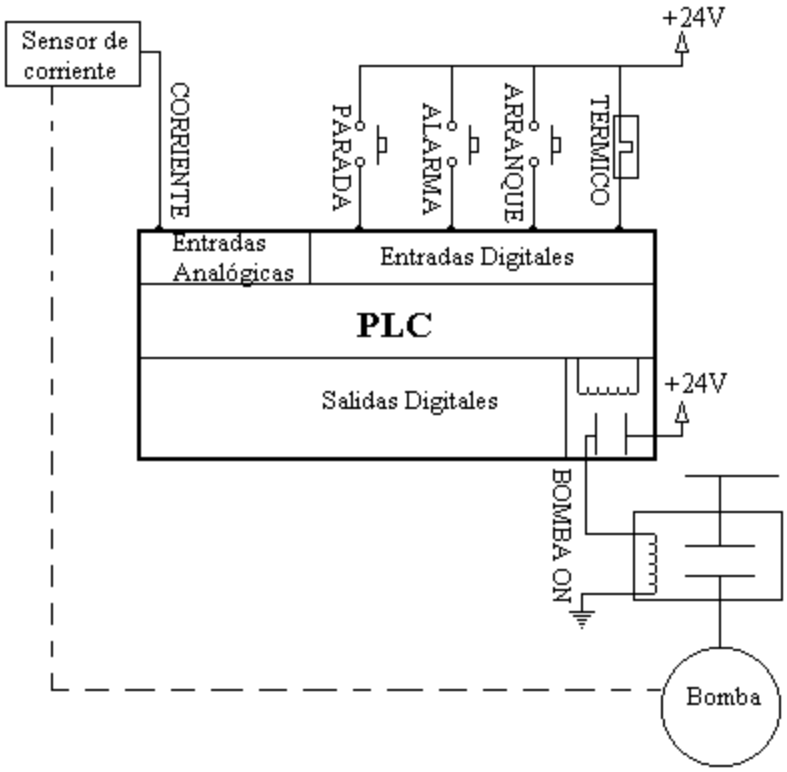


Figura 41

ALARMA, TERMICO, ARRANQUE y PARADA representan entradas tipo bit al PLC. BOMBA_ON es una salida tipo bit (relé) del PLC. CORRIENTE representa una entrada tipo IW (analógica) al PLC.

CAPÍTULO 4: INTRODUCCIÓN AL AMBIENTE DE DESARROLLO DE PROGRAMAS AUTOMATION BUILDER

El software que se utilizará para configurar el hardware, periféricos e interfaces de comunicación del PLC será el Automation Builder versión 1.2.

La programación de rutinas y variables se realizará en CoDeSys, software de uso libre que cumple con la norma IEC de programación de autómatas y que cuenta con una interfaz hacia el Automation Builder.

1 Automation Builder

El manejo de los proyectos se realiza desde la ventana principal, pudiendo crear un nuevo proyecto o abrir uno existente. El software cuenta con una herramienta de importación de proyectos de versiones anteriores. Se presenta a continuación la ventana de selección de proyectos:

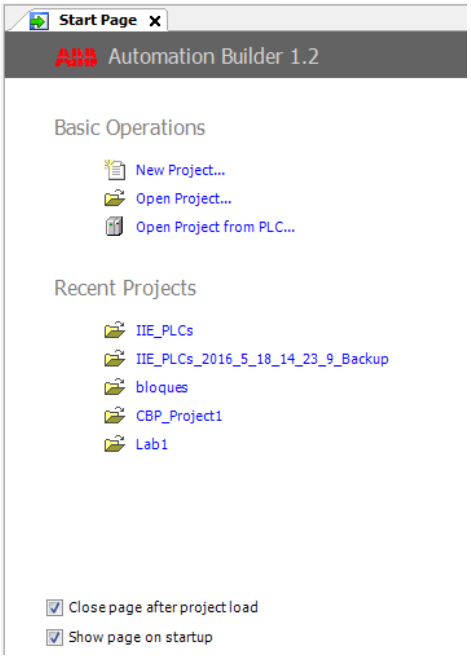


Figura 42

Al seleccionar un nuevo proyecto es necesario indicar si se incluirá un CPU. Para el caso de los laboratorios del curso seleccionar AC500. Luego se requiere al usuario ingresar el modelo del PLC a utilizar. En el caso de este curso se trabajará con la línea PM-554-ETH

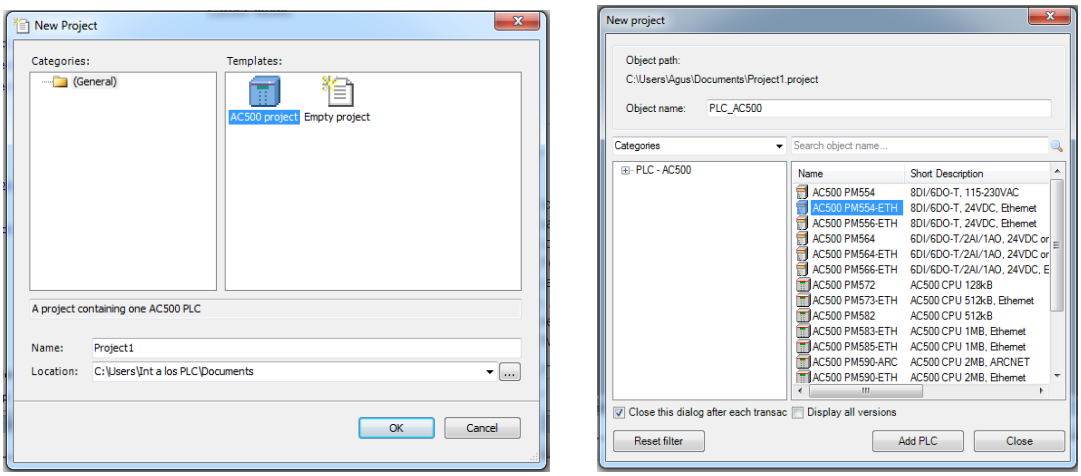


Figura 43

En la parte izquierda de la pantalla principal del Automation Builder se mostrará el controlador ingresado y las distintas interfaces de comunicación con las que cuenta el equipo. Para el caso del controlador seleccionado se cuenta con entradas y salidas integradas (OBIO, *on-board-input-ouput*), bus de conexión de módulos (IO_Bus) y Ethernet (ETH1).

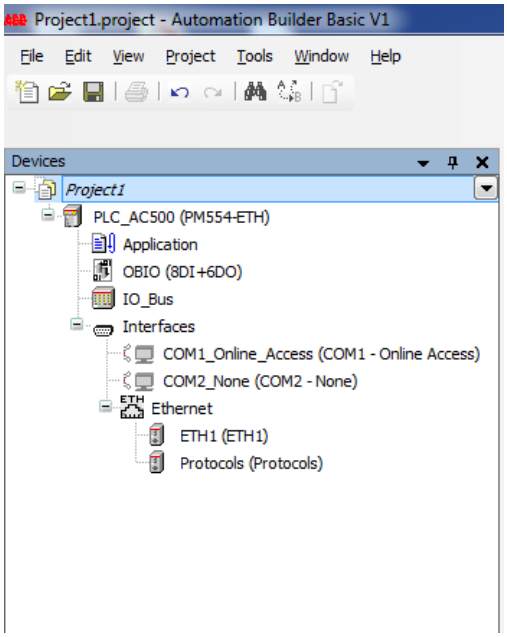


Figura 44

Se configurará la interfaz Ethernet para forzar una IP dentro del rango de trabajo del laboratorio, de forma que el PLC y la PC trabajen dentro de la misma subred.

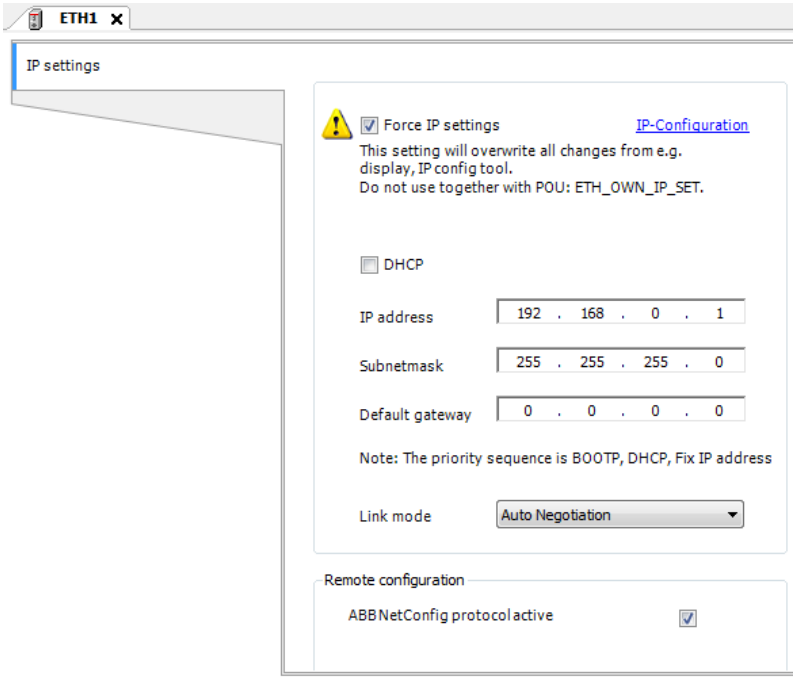


Figura 45

Se observa además que el equipo seleccionado cuenta con entradas y salidas integradas. El mapeo de direcciones de las mismas se puede observar en la pestaña OBIO de acuerdo a la siguiente figura. A modo de ejemplo, se observa que la primera entrada digital es mapeada a la dirección %IX4000.0.

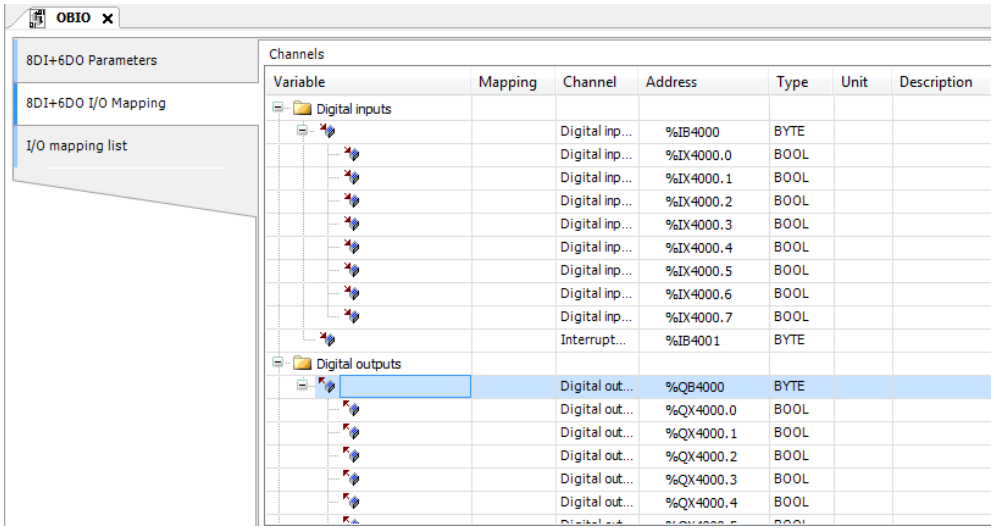


Figura 46

Para agregar el módulo analógico con el que se trabajará en el curso se debe hacer click derecho sobre el IO_Bus y seleccionar agregar objeto. En el laboratorio se trabajará con el módulo de entrada analógicas para sensores de temperatura tipo RTD denominado *AI562*.

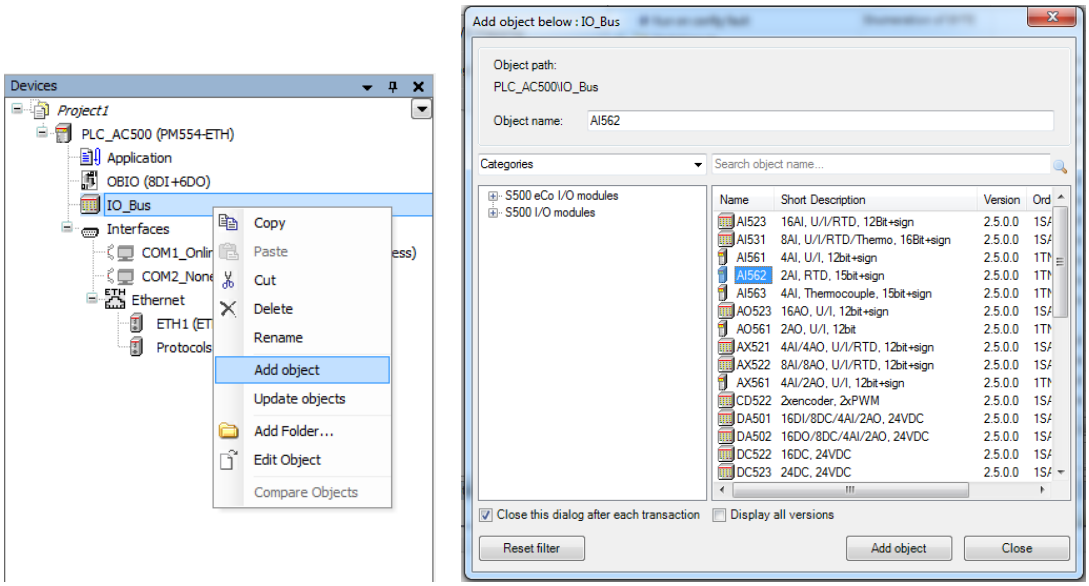


Figura 47

La carga de los datos de hardware e interfaces al PLC conectado via Ethernet puede realizarse a través del login del Automation Builder. Realizar un login a un PLC activo además permite observar errores y advertencias, así como reconocer y eliminar las mismas.

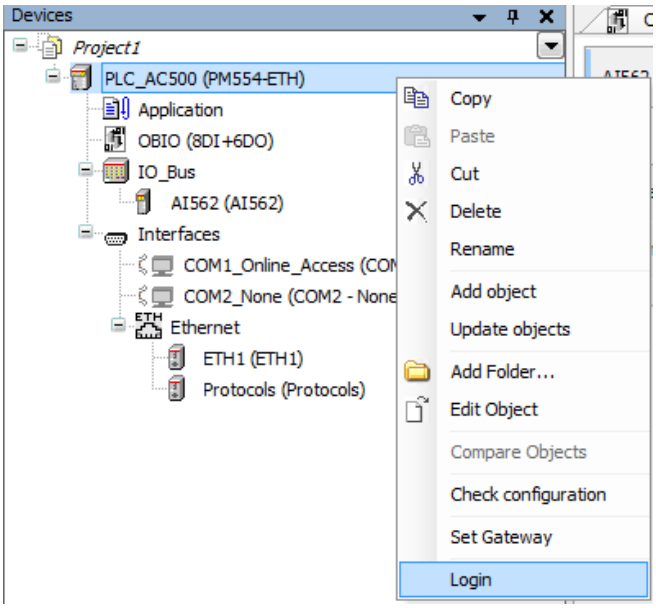


Figura 48

2 CoDeSys

La programación de las rutinas de automatización se realizará en el software CoDeSys. Para ingresar al mismo se debe hacer doble click sobre el ícono de Aplicación dentro de la ventana de interfaces del PLC en el Automation Builder.

Esta ventana está dividida en dos paneles: el panel izquierdo presenta cuatro pestañas denominadas “POUs”, “Data types”, “Visualizations” y “Resources”; el panel derecho se

presenta distintas ventanas según la selección en el panel izquierdo. Dentro de la carpeta POUs (*program-organization-units*) se muestran las distintas rutinas programadas. Por defecto el PLC corre la rutina PLC_PRG, desde la cual se pueden realizar llamados a otras rutinas como se muestra en la siguiente figura de ejemplo.

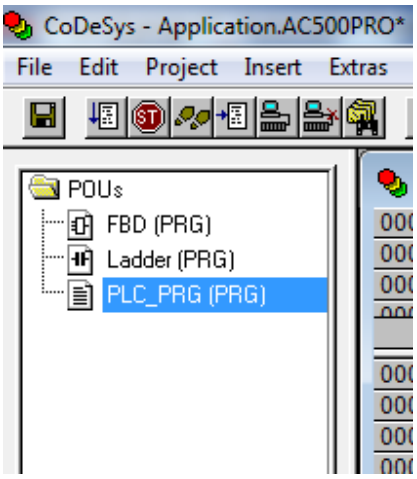


Figura 49

Haciendo click derecho sobre esta ventana se puede crear una nueva rutina, completando en la ventana de asistente de creación de POU el nombre, el tipo y el lenguaje a utilizar. El tipo de POU se utiliza para indicar si es una rutina de ejecuciones o si se está generando un bloque personalizado para reutilizar dentro de un rutina, como se mostrará más adelante.

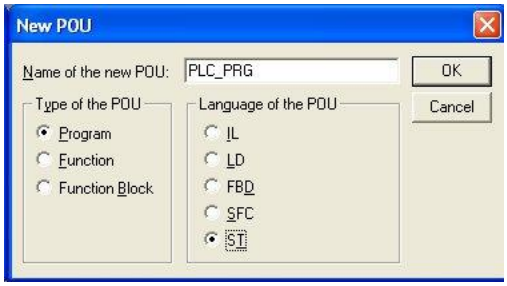


Figura 50

En la se muestra la ventana del proyecto por defecto. En el panel izquierdo aparece seleccionada la pestaña POUs, y en la lista de POUs aparece seleccionado el POU "PLC_PRG". En el panel derecho aparece la ventana de programación del POU "PLC_PRG". La sección superior corresponde a la declaración de variables del POU mientras la sección inferior corresponde al código del POU.

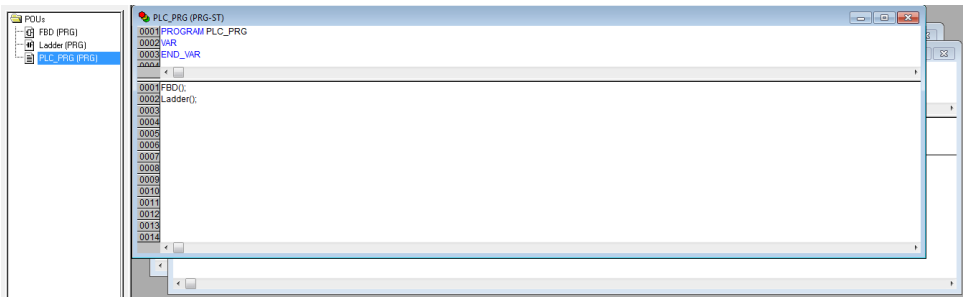


Figura 51

La barra de herramientas se ajusta automáticamente al lenguaje seleccionado. Por ejemplo, al crearse un nuevo POU en lenguaje ladder la barra de herramientas incluirá los contactos y bobinas de utilización más frecuentes, así como un bloque genérico que al seleccionarse abrirá el asistente de ingreso de objetos. Este permite recorrer las librerías de objetos instaladas en el CoDeSys, pudiendo implementar contadores, timers, etc.

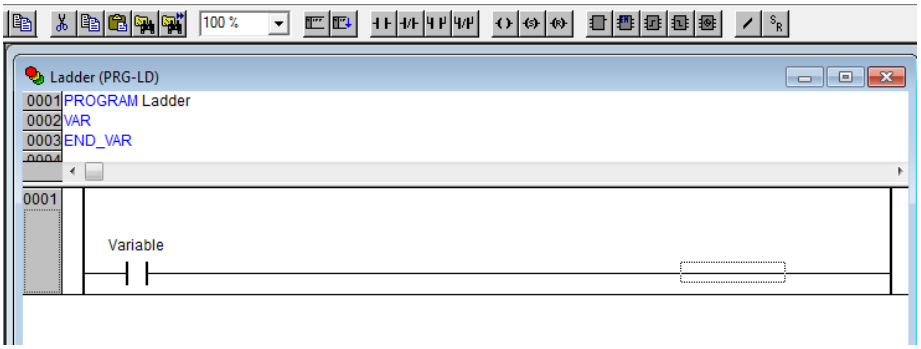


Figura 52

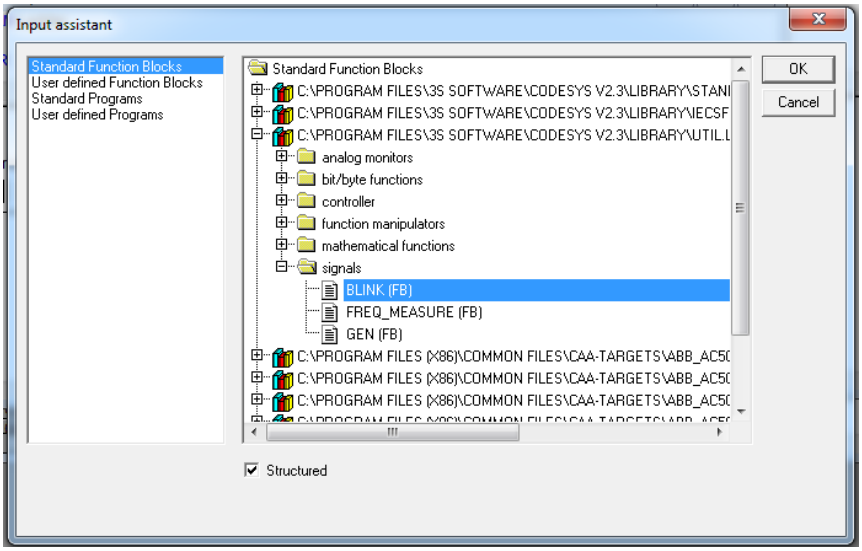


Figura 53

La configuración de las variables globales del proyecto y de las entradas/salidas configuradas en el hardware se realiza en Global_Variables dentro de la pestaña Recursos.

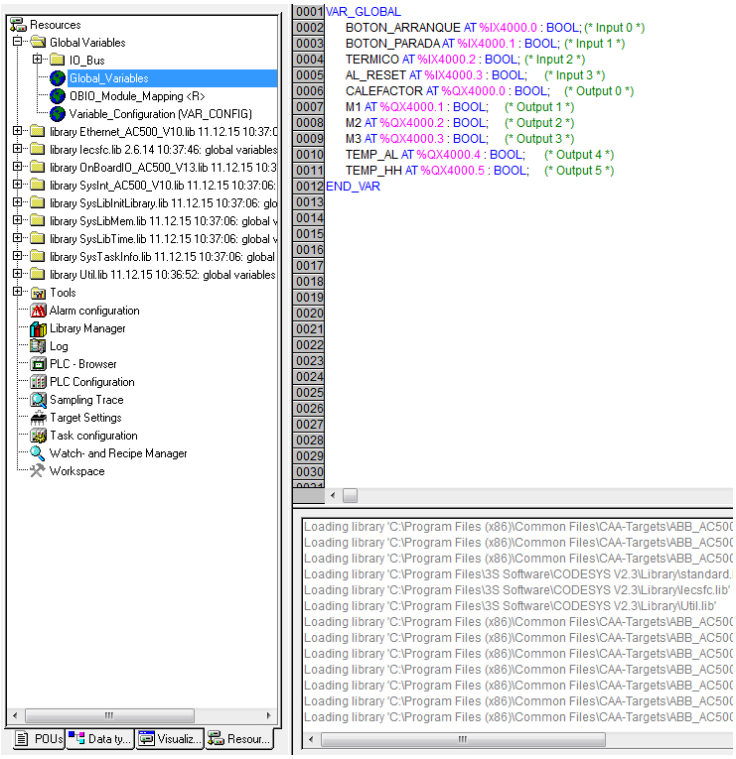


Figura 54

Finalizada la programación del proyecto, dentro del menú Proyecto se encuentra la opción Rebuild All que compilará el programa.

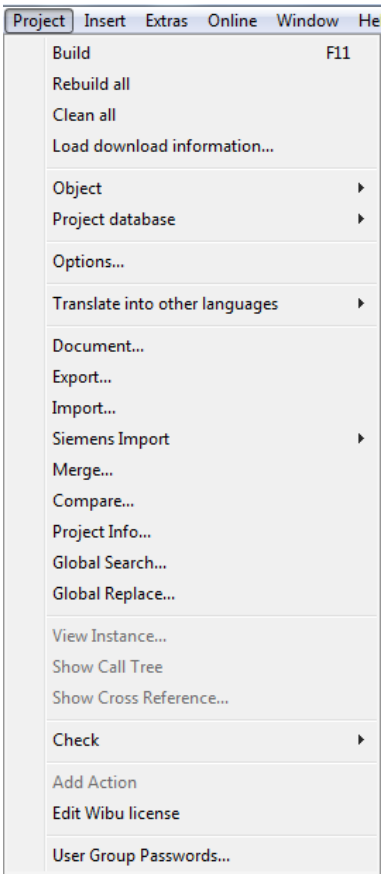


Figura 55

Al seleccionar Download o Login desde el menú Online se descargará el programa al PLC, dándole marcha al programa a través de la opción Run.

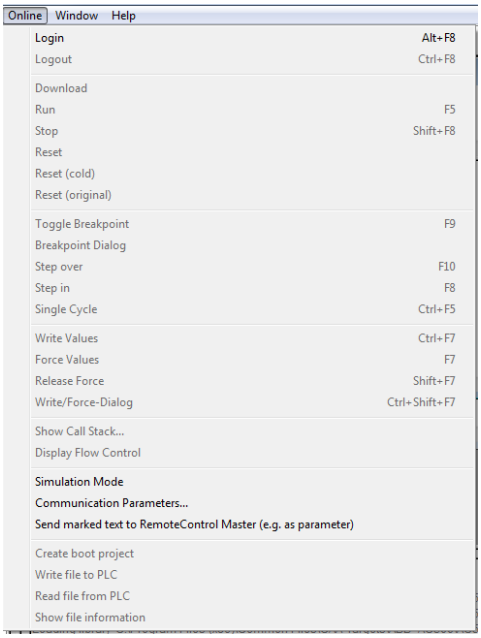


Figura 56

CAPÍTULO 5: LENGUAJE FBD

3 El bloque funcional. Estructura del programa en FBD.

El lenguaje FBD se basa enteramente en los bloques funcionales. La noción de bloques funcionales ya la introdujimos de forma breve en el capítulo 3, referido al lenguaje LD. En efecto, gran parte de las instrucciones en LD toman la forma de bloques funcionales.

Un bloque funcional se representa por un rectángulo, con puntos de conexión de entradas, puntos de conexión de salidas y un identificador. El identificador describe función a la que refiere el bloque. Las entradas de la función se conectan a los puntos de conexión de entradas, en el borde izquierdo del rectángulo. Las salidas de la función se conectan a los puntos de conexión de salidas, en el borde derecho del rectángulo.

La Figura 57

representa un bloque funcional que implementa la función FUN, con dos entradas y dos salidas:

Figura 57

Las entradas y salidas son datos. El tipo de estos datos depende del bloque funcional. Estos datos pueden ser datos de la memoria accesible, o directamente salidas de otros bloques funcionales. Es decir, está permitido conectar la salida de un bloque a la entrada de otro.

Los datos de entrada y de salida de un bloque se conectan por intermedio de líneas de conexión.

En general, un programa en FBD se ejecuta de arriba hacia abajo, y de izquierda a derecha.

Veamos el siguiente ejemplo: se desea escribir un programa que controle el encendido - apagado de una bomba.

La bomba será encendida si:

1) Se pulsa el botón de arranque.
2) La protección térmica está deshabilitada.
3) Está abierto el botón de emergencia.
4) Está abierto el botón de parada.

Desde un tiempo T después del encendido, no puede haber ni sobre corriente ni baja corriente. Expresado de otra forma, desde un tiempo T después del arranque, la corriente I debe cumplir $I_{MIN} < I < I_{MAX}$, siendo I_{MIN} e I_{MAX} límites prefijados.

El motor de la bomba se apagará si:

1) Se pulsa el botón de parada.
2) Se cierra la protección térmica.
3) Se pulsa el botón de emergencia.
4) Los límites de corriente no son los correctos.

El programa FBD que cumple lo anterior es éste:

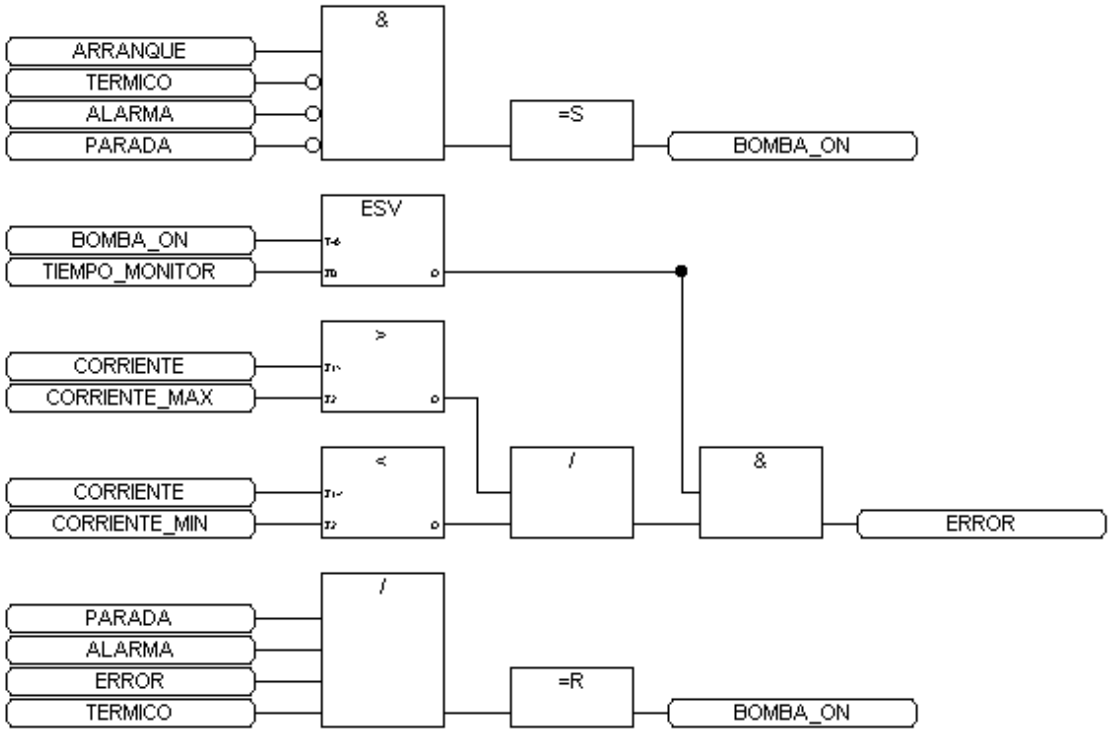


Figura 58

En el programa anterior, los tipos de datos son los siguientes:

Datos binarios:
Entradas digitales: ARRANQUE, TERMICO, EMERGENCIA, PARADA.
Bits internos: ERROR.
Salidas digitales: BOMBA_ON.
Palabras dobles: TIEMPO_MONITOR (constante).
Palabras:
Constantes: CORRIENTE_MAX y CORRIENTE_MIN.
Entradas analógicas: CORRIENTE.

El mismo programa en LD queda:

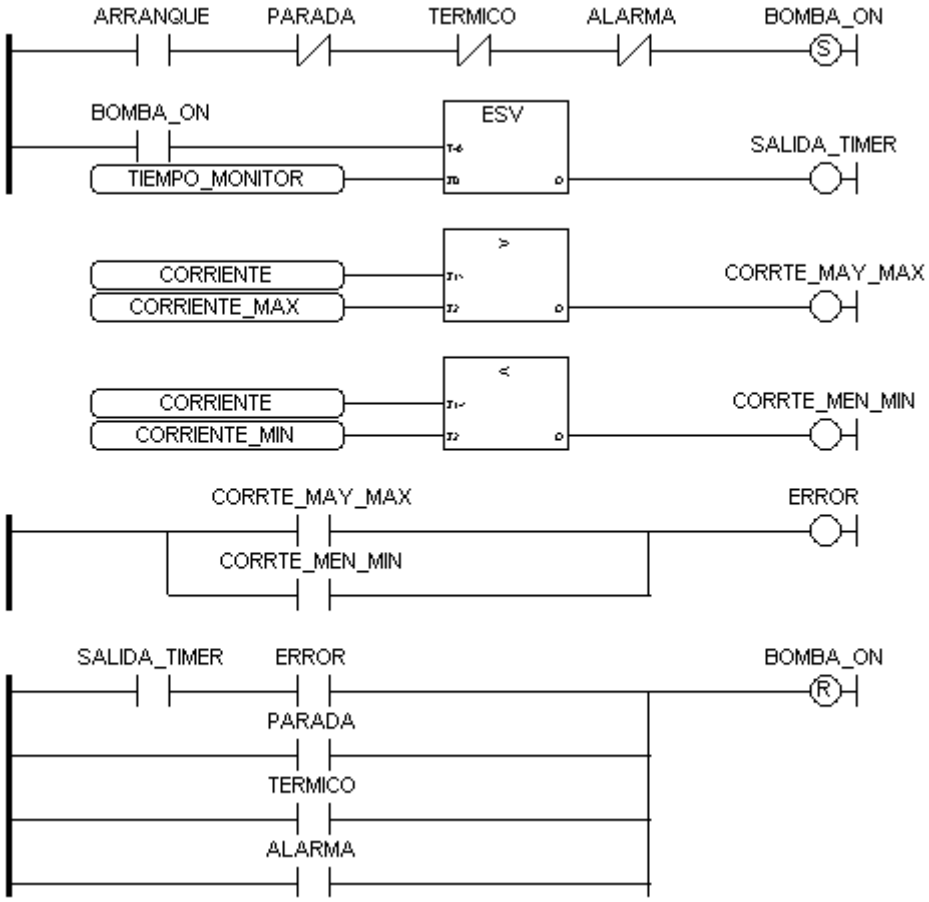


Figura 59

Aunque el tamaño de los programas resulta el mismo, el programa en FBD es más claro.

Las comparaciones entre palabras se realizan con el bloque de comparación "<", con dos entradas: una es CORRIENTE, que representa la corriente medida directamente de la salida analógica del convertidor de corriente alterna a voltaje de continua de la bomba, y la otra representa el límite de la corriente.

La salida del bloque de comparación es un bit, que indica si la corriente superó o no dicho valor. Se utiliza un timer para habilitar una señal de error a partir de la comparación de CORRIENTE con sus límites, a partir de un tiempo T desde el arranque. Es decir, si a partir del tiempo T desde el arranque se encuentra que CORRIENTE no está dentro de los límites prefijados se apagará la bomba.

Como los botones de arranque y parada se mantienen en 1 sólo cuando se pulsan, se utilizan las instrucciones "=S" e "=R" (equivalentes a las instrucciones LATCH y UNLATCH de LD). Esto hace que la variable de salida se mantenga en el valor que corresponde tras la activación de alguno de los botones.

4 Grupos de instrucciones

Las funciones que podemos realizar con los distintos tipos de datos son muy variadas. Para dar una idea de la cantidad de bloques de que se dispone, el equipo que utilizaremos en el laboratorio admite alrededor de 170 bloques.

Una forma de clasificar las funciones implementadas por los bloques funcionales es la siguiente:

a) Funciones binarias.
b) Funciones de timer.
c) Contadores.
d) Funciones de palabras simples y dobles: funciones de comparación, aritméticas, lógicas.
e) Funciones de control de programa.
f) Funciones de control.
g) Funciones de comunicación.
h) Funciones de conversión de formato (pack, unpack).
i) Funciones de alto nivel (multiplexor).
j) Funciones y de acceso a memoria.

En esta parte del capítulo pasaremos revista por los tipos de funciones más usados.
Los bloques funcionales descritos anteriormente son comunes a todos los PLC, de forma independiente del fabricante.

4.1 Funciones binarias

Las funciones binarias tienen entradas y salidas del tipo binario.

Entre las funciones binarias cuentan las funciones AND, OR, XOR, =S (LATCH) e =R(UNLATCH).

Se permite cambiar el número de entradas de ciertos bloques.

Una entrada a un bloque puede ser una variable negada (complemento a 1 de la variable). Para esto, se conecta la variable al punto de conexión que corresponda a través de una línea de conexión con negación booleana. En la Figura 60, la negación de VAR es una de las entradas al bloque FUN.

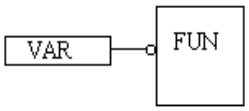


Figura 60

El ejemplo de programa de encendido/apagado de una bomba es una nueva muestra - que se añade a ejemplos vistos en el capítulo 2 - de la importancia de las instrucciones LATCH y UNLATCH. Estas instrucciones son útiles en casos en que las entradas que activan una salida no se mantienen a lo largo de todo el tiempo de encendido de la misma. Bastará que las entradas que activan la salida se mantengan durante un tiempo actualización de entradas (recordar el ciclo del PLC), para que la salida quede activa. Un caso típico de los anteriores se observa en el ejemplo de encendido/apagado de la bomba: la pulsación

de un botón se realiza durante un período de tiempo, de forma que al soltarlo la entrada asociada se desactiva. Sin embargo, el uso de una instrucción LATCH (SET) permitió mantener la salida en activa (o inactiva) tras la desactivación de la entrada.

De acuerdo al ciclo de programa del PLC, una variable de salida que se somete a sucesivos LATCH (SET) y UNLATCH (RESET) en un programa, cambiará su valor de acuerdo a la última instrucción de este tipo que
actuó sobre la variable durante el tiempo de ejecución del programa.

4.2 Funciones de Timer y Contadores

Este grupo incluye timers (*delay* OFF, *delay* ON, etc.), contadores (hacia arriba o hacia abajo, con cuentas de tipo palabra o tipo palabra doble), monoestables, etc.

4.3 Funciones de comparación de palabras y palabras dobles

En el ejemplo de encendido/apagado de una bomba ya se utilizó la función "<" de comparación de palabras. En general, el resultado de estas operaciones es un bit. El resultado será 1 si la igualdad o desigualdad del bloque es verdadera, y 0 si es falsa. De forma similar, hay bloques que realizan estas funciones sobre doble palabras.

4.4 Funciones aritméticas sobre palabras y doble palabras

En estas funciones, las salidas son del mismo tipo de datos que las entradas. Incluyen suma, resta, división y multiplicación, sobre palabras o palabras dobles. Los bloques más utilizados son la suma, la resta y la división. Si bien se dispone de funciones matemáticas de mayor nivel, como raíz cuadrada o seno, se utilizan raramente.

4.5 Funciones lógicas sobre palabras y palabras dobles

En general, ejecutan la función lógica que corresponde bit a bit de los operandos. Según la instrucción, el tipo del resultado puede ser bit (por ej., MASK) o el mismo de las entradas (palabra o palabra doble). Las funciones WAND, WOR y WXOR de palabras se corresponden (bit a bit) con el AND, OR y XOR, respectivamente, de un bit.

Asimismo, entre estas funciones cuentan distintos *shift* y *rotate* de palabras o palabras dobles.

4.6 Control de flujo en el programa

Una de las instrucciones más importantes de este grupo es el JUMP, similar a la sentencia de otros lenguajes. Se trata de un salto condicional. La instrucción JUMP se asocia a una condición booleana y una etiqueta (*label*). Si la condición es verdadera se continúa la ejecución del programa la línea a la que hace referencia la etiqueta; si es falsa se continúa la ejecución del programa en la sentencia siguiente a la JUMP.

No es de uso normal, ya que una buena estructura de programa evita el uso de esta sentencia.

Dos bloques funcionales muy importantes, que se refieren indirectamente al control de flujo del programa, son DI y DO.

En el caso DI se lee una entrada en forma inmediata dentro del ciclo del PLC. Análogamente DO escribe una salida de forma inmediata dentro del ciclo del PLC. Estos bloques cobran importancia en casos en que el ciclo del PLC es muy lento respecto del tiempo requerido para actualizar ciertas entradas o salidas.

4.7 Funciones de control PID

Entre las funciones asociadas a control, tenemos la función PID (control proporcional, integral y derivativo) y la PI (control proporcional e integral).

Las constantes del control se pueden fijar en forma externa, con un PC conectado al PLC. Las salidas de control pueden ser variadas, desde una salida analógica de 4-20 ma hasta una salida que varíe el ancho del pulso de una salida ON/OFF. Las salidas se verán con más detalles en el capítulo "El PLC como controlador".

El control de un proceso de variación muy lenta respecto del tiempo de ciclo del PLC requiere de un término integral de peso muy pequeño. Esto se logra ejecutando el bloque de control cada N ciclos del PLC. Para esto, existe una función de control de programa, denominada LZB.

Con el aumento de la capacidad de los PLCs, las funciones de control se han vuelto más sofisticadas, de forma que con el tiempo los PLCs han ido desplazando a los controles discretos.

4.8 Funciones de conversión de datos

En este grupo cuentan funciones que permiten pasar de un número en BCD a otro en binario o a la inversa.

Pero las más usuales son las de conversión de una palabra en una palabra doble. Un ejemplo de aplicación de estas funciones es la fijación del tiempo de un timer a través de una palabra. El hecho que este tiempo deba ser una palabra doble, hace necesaria la conversión de esta palabra en palabra doble.

Dos funciones muy útiles de este grupo son las funciones *pack* y *unpack*. *Pack* convierte un conjunto no ordenado de bits en una palabra o palabra doble, mientras *unpack* permite desempacar los bits. Son funciones muy utilizadas en comunicaciones, pues en muchos casos es más fácil transmitir palabras que bits individuales.

4.9 Funciones de comunicaciones

La puerta serie del PLC se puede utilizar de dos maneras. La primera es la forma habitual de uso (RS232 ASCII) , en la que se intercambian (transmiten y reciben) mensajes con un PC. La segunda consiste en aplicarla en comunicaciones a través del protocolo Modbus -que veremos más adelante- en ambientes industriales.

Existe una función (en los PLCs del laboratorio, SINIT), para fijar las constantes de la puerta serie, como ser velocidad, número de bits, paridad, etc. Asimismo, existe otra que permite la comunicación a través del protocolo Modbus.

5 LD o FBD

De los ejemplos anteriores, surgen dos diferencias entre un programa en FBD y uno en LD:

1) La implementación de las funciones lógicas de bits. En LD, un AND y un OR de bits se implementan como una serie y un paralelo de contactos, respectivamente. Mientras tanto, en FBD existen los bloques AND y OR.
2) La concatenación de bloques funcionales: en LD, no se puede conectar la salida de un bloque a la entrada de otro de forma directa, mientras que en FBD sí se puede.

En la Figura 61 se muestra un ejemplo de concatenación de bloques: la salida Torque de Control_de_Velocidad se realimenta a Control_de_Posicion, de forma que Control_de_Posicion pueda detectar una falla si el comando de un gran torque no hace decrecer el error de posición.

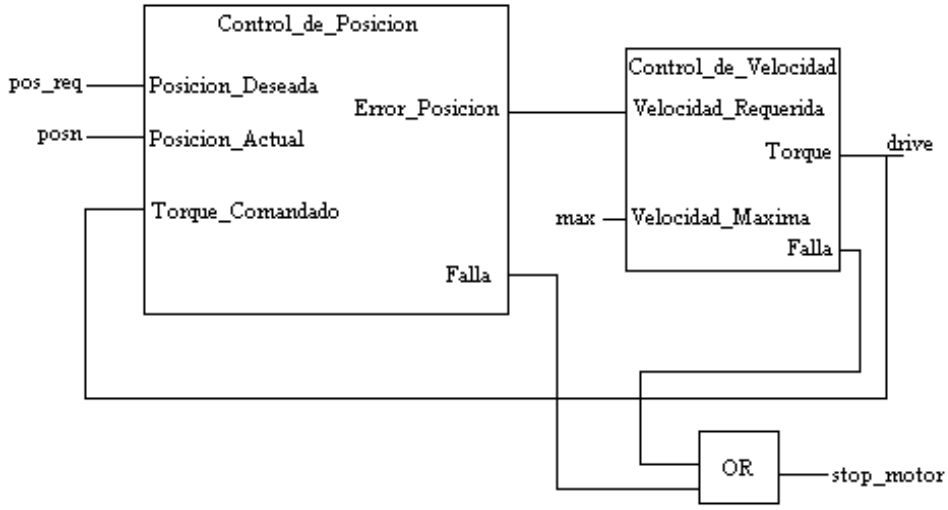


Figura 61

El programa en FBD de la Figura 61 tiene la apariencia de un loop infinito. La ejecución del programa transcurre de forma que la entrada Torque_Comandado de Control_de_Posicion en un *ciclo dado del PLC* coincide con la salida Torque de Control_de_Velocidad del *ciclo anterior*.

En principio, el lenguaje LD soporta los mismos bloques funcionales que FBD. Sin embargo, existe una diferencia entre los bloques LD y los bloques FBD: el estándar IEC 1131-3 incluye contactos de habilitación EN para todos los bloques en LD. De esta forma, en LD la función asociada al bloque no se ejecutará si la habilitación vale '0'.

La elección de uno u otro lenguaje depende en última instancia del lenguaje que resulte más claro al usuario, ya que desde el punto de vista de longitud de un programa, ninguno de los lenguajes presenta grandes ventajas respecto del otro.

CAPÍTULO 6: SINTONÍA DE UN CONTROLADOR CONTINUO

1 Introducción

Este capítulo trata la sintonía de un controlador continuo tipo PID. Como son necesarios algunos conceptos básicos de Control, se acompaña de un Apéndice donde se recuerdan estos conceptos. El lector familiarizado con el Control puede pasar por alto dicho Apéndice.

1.1 Un problema de Control

Un problema general de Control que queremos abordar se formula de la siguiente manera:

Se desea controlar un proceso, por ejemplo un proceso térmico, químico, etc.
No se conoce en detalle su comportamiento.
Se conoce un experimento, por ejemplo, la respuesta a escalón.

El problema se presenta de forma esquemática en la Figura 62.

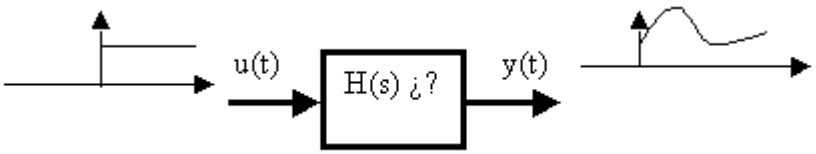


Figura 62

La respuesta al escalón consiste en registrar la salida y(t) del proceso cuando la entrada es u(t) = Y(t). Y(t) se denomina función "escalón", y se define según:

$$Y(t) = \begin{cases} 0 \text{ si } t < 0 \\ A \text{ si } t > 0 \end{cases}$$

Existen dos maneras básicas de diseñar un control (ver Apéndice): control en lazo abierto y control en lazo cerrado. Una estructura posible de control en lazo cerrado se muestra en la Figura 63.

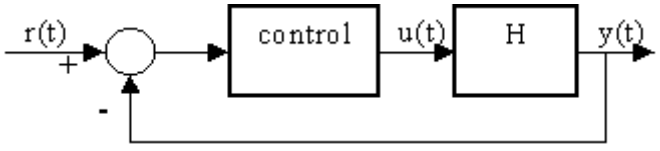


Figura 63

El problema consiste en encontrar una "dinámica apropiada" para el bloque del controlador. Lo primero que se debe tener claro es qué comportamiento se desea del sistema controlado. El comportamiento deseado puede ser por ejemplo:

que la respuesta del sistema a una entrada u(t) constante sea una salida constante muy parecida a la entrada
que la salida del sistema siga "lo más fielmente posible" a la entrada.

1.2 La necesidad de la estandarización. Respuesta a escalón.

La precisión del segundo objetivo de la sección anterior requiere de la respuesta a las siguientes preguntas:
¿Qué quiere decir "seguir fielmente"?
Si la entrada es por ejemplo un escalón ¿cómo medir el apartamiento? ¿cómo definir la velocidad con que el sistema "sigue a la entrada"?.

La necesidad de una especificación común a usuarios y diseñadores lleva a la especificación de un estándar.

Algunos parámetros de la respuesta a escalón representan una forma usual de especificar características de la respuesta transitoria o índices de desempeño en el dominio del tiempo. Para definir estos parámetros, se considera la respuesta a escalón de la Figura 64. Se define:

Mp - Sobretiro: exceso de la salida por arriba de su valor asintótico final (%)
$\mathbf{t_d}$ - Retardo. Tiempo que tarda la respuesta en alcanzar el 50% del valor final
$\mathbf{t_r}$ – Tiempo de levantamiento. Tiempo que tarda la respuesta en pasar del 10% al 90% del valor final.
$\mathbf{t_s}$ – Tiempo de asentamiento. Tiempo a partir del que la respuesta no se aparta más del 5% de su valor final.
$\mathbf{t_p}$ – Tiempo de pico. Tiempo en el que ocurre el valor máximo de la salida.

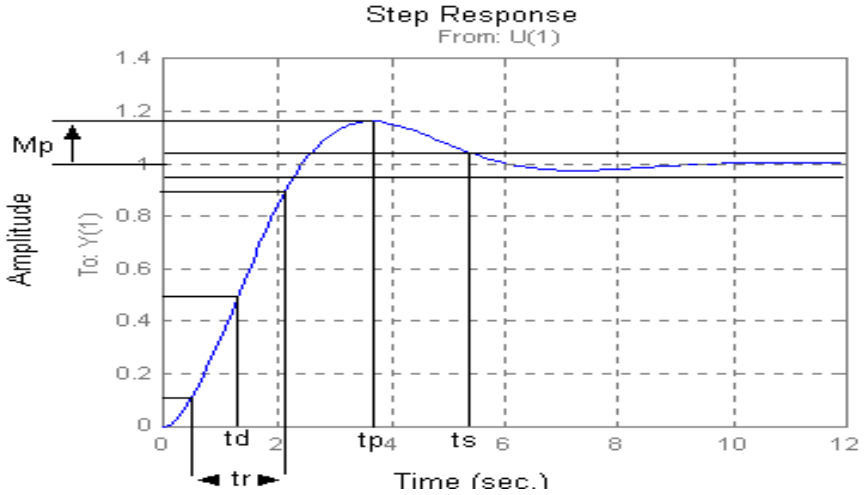


Figura 64

Las definiciones anteriores permiten especificar los conceptos "velocidad de respuesta" o "apartamiento" con la precisión requerida para un diseño manteniendo el sentido práctico requerido por la aplicación.

Resulta evidente que otros objetos del lazo de control deben estar estandarizados para facilitar la práctica industrial. Por ejemplo, se debe estandarizar:

la interfaz entre los sensores que miden la salida y(t) y el controlador
la interfaz entre los actuadores sobre la planta o proceso y el controlador (p.ej: 4-20mA, 0-10V, etc.).

Las estandarizaciones permiten integrar el sistema realimentado de control con componentes (sensores, actuadores, controladores) de diversos proveedores.

2 Un ejemplo, el controlador proporcional P

Veamos un primer ejemplo de diseño de un controlador.

Se considera un motor. La entrada es la tensión aplicada, y la salida la velocidad angular. Supongamos que su función de transferencia es

$$H(s) = \frac{1}{s+1}$$

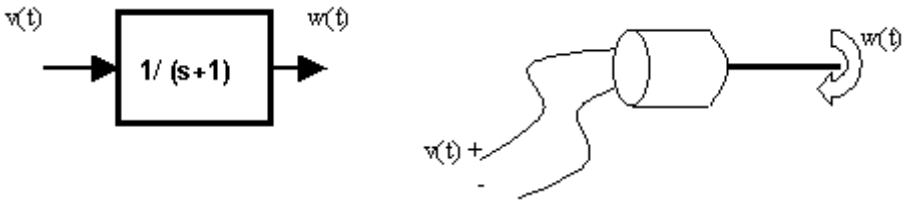


Figura 65

La Figura 66 muestra la respuesta a escalón del motor. La "velocidad" medida por su tiempo de asentamiento es de 3 segundos.

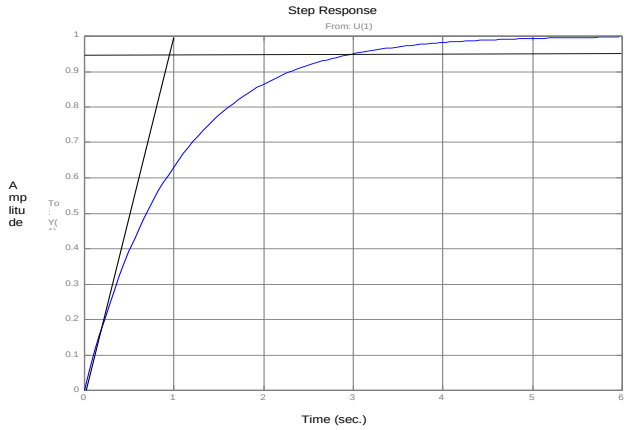


Figura 66

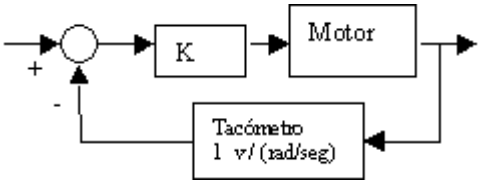


Figura 67

Nos proponemos diseñar un controlador para mejorar la velocidad de respuesta del sistema controlado bajando el *tiempo de asentamiento a 0.2 s*.

Para esto, se propone un sistema de control realimentado según la Figura 67. Nótese que la Figura 67 representa un caso particular de la Figura 63, donde el controlador es un amplificador de ganancia K. Esta estructura se denomina "estructura de realimentación unitaria con controlador Proporcional".

El diseño del controlador consiste en encontrar K tal que ts = 0.2 seg.

La transferencia del lazo cerrado es:

$$G(s) = \frac{K\,H(s)}{1 + K\,H(s)} = \frac{K}{s + K + 1}$$

Luego, como $t_S = 3 * \frac{1}{K+1}$, $\quad t_S < 0.2 \quad \Rightarrow \quad \boxed{K \geq 14}$

La Figura 68 muestra la respuesta a escalón del sistema realimentado, con K=14, y la compara con la respuesta a escalón original del motor.

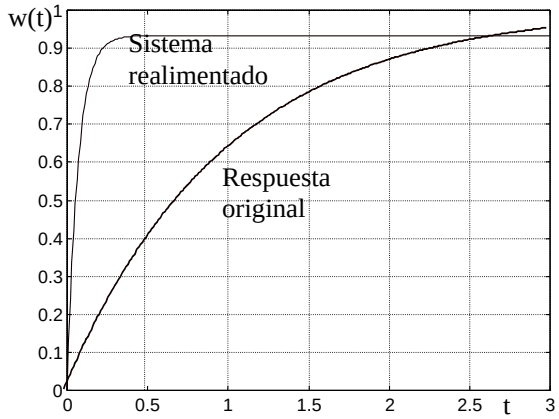


Figura 68

Se nota la mejora obtenida con el controlador proporcional. La velocidad de respuesta fue sensiblemente mejorada.

¿Cómo se consiguió esta mejora?

Para explicarlo, se considera la salida del controlador en la Figura 67 cuando r(t) es un escalón de 1V. El controlador recibe la señal e(t), diferencia entre la entrada a seguir y la velocidad medida, y entrega al motor una tensión K veces mayor que e(t). De esta forma al inicio la entrada u(t) llega a 14V, con lo cual se consigue una subida más rápida que en el sistema sin control. Conforme la velocidad aumenta, el voltaje de entrada al motor disminuye, hasta alcanzar un valor de régimen (Figura 69).

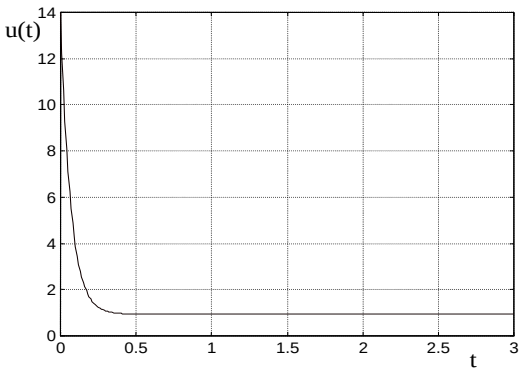


Figura 69

En resumen, se mejora la velocidad del sistema a costa de una señal de control (u(t)) mucho más alta.

De aquí resulta la primer limitación práctica de este controlador: si quisiéramos hacer un sistema controlado más rápido, deberíamos aumentar K pero posiblemente alcanzaríamos los límites físicos, ya sea del voltaje que el amplificador puede entregar, o del voltaje que el motor puede soportar.

3 Cuidado con las altas ganancias

El ejemplo que sigue muestra los inconvenientes que surgen de usar altas ganancias. Se formula de la misma forma que el ejemplo anterior, salvo que ahora no se conoce la función de transferencia H(s), sino que se conoce la respuesta a escalón del sistema a controlar. La respuesta a escalón se muestra en la Figura 70.

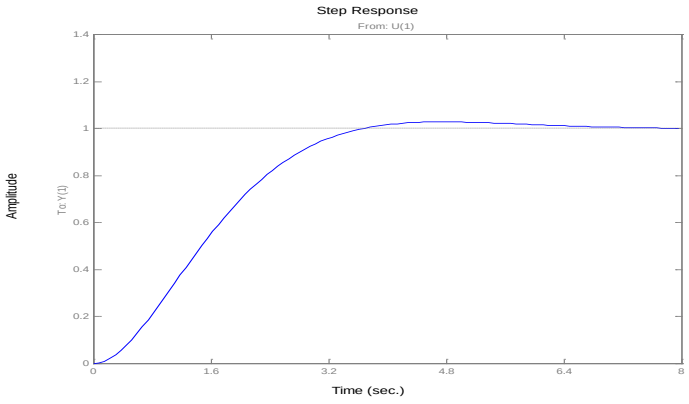


Figura 70

A partir de este experimento, se "estima" la H(s) desconocida (proceso) con $H_1(s)$.

$$H_1(s) \approx H(s)$$

$$H_1(s) = \frac{1}{s^2 + 1.5s + 1}$$

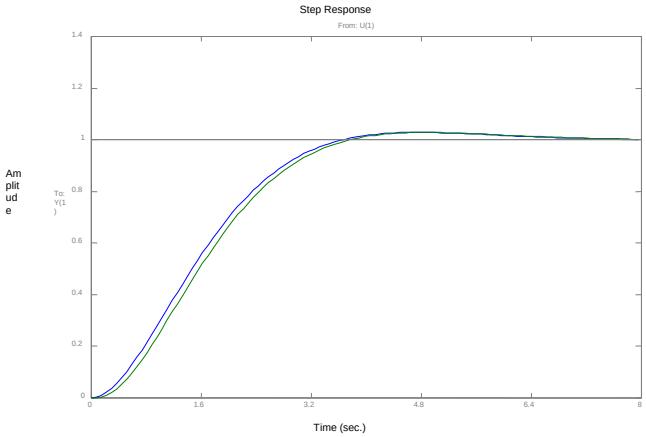


Figura 71

Nótese la semejanza de las respuestas a escalón del proceso real y el modelo estimado (Figura 71).

Utilizando H1 se diseña un control realimentado análogo al del ejemplo anterior. Para aumentar la velocidad de la respuesta al escalón se aumenta el valor de K. A continuación se contrasta el comportamiento esperado por simulaciones a partir de H1 con el comportamiento real (H(s)) para distintos valores de K:

K=14

Para K=14 se espera un comportamiento estable (Figura 72) con sobretiro alto. Sin embargo, el sistema real presenta un comportamiento bastante más oscilatorio (Figura 73).

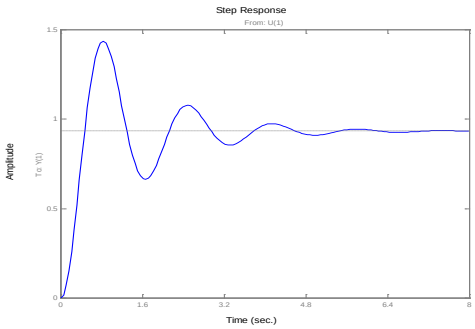


Figura 72

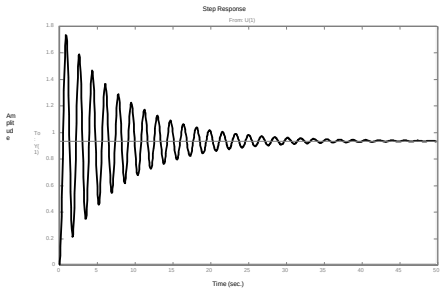


Figura 73

K=17.7

Para K=17.7, se espera un comportamiento como el de la Figura 74. Sin embargo, el sistema real (Figura 75) presenta un comportamiento inestable.

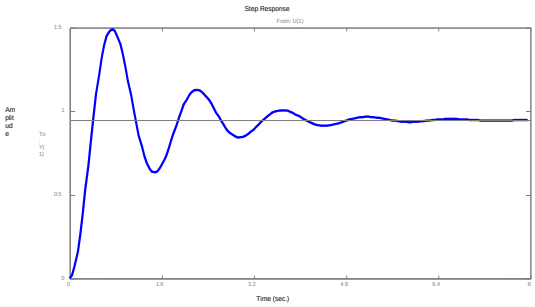


Figura 74

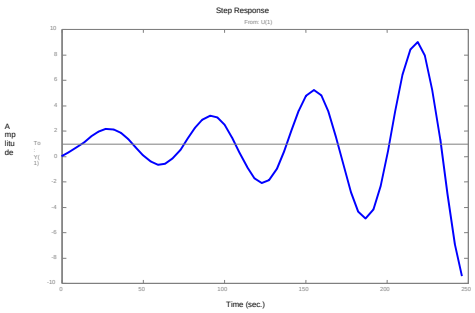


Figura 75

¿Cómo se explica el error en las predicciones?

El modelo estimado se comporta de acuerdo a:

$$H_1(s) = \frac{1}{s^2 + 1.5s + 1}$$

El sistema real se comporta de acuerdo a:

$$H(s) = \frac{1}{(1+0.1s)(s^2+1.5s+1)}$$

H_1 es similar a H(s), pero no idéntico. La aproximación no tiene en cuenta un polo muy rápido (en s = -10). Para valores relativamente bajos de ganancia del controlador, el error de la "aproximación" del modelo no es apreciable. Sin embargo, a ganancias suficientemente grandes, el sistema real realimentado se inestabiliza.

En la Figura 76 y la Figura 77 se muestra el lugar de las raices (ver anexo) del sistema realimentado del proceso real y de la aproximación, respectivamente. Para bajas ganancias se aprecia que los polos dominantes son semejantes. Sin embargo, para ganancias mayores que 5-7, ya se perciben diferencias. Para ganancias suficientemente altas, la ubicación de los polos del sistema real indica que el sistema inestable.

Una mala combinación: errores de modelado y altas ganancias.

Es importante mencionar que la existencia del polo en s = -10 se hubiera puesto de manifiesto a través de medición de la *respuesta en frecuencia* del proceso.

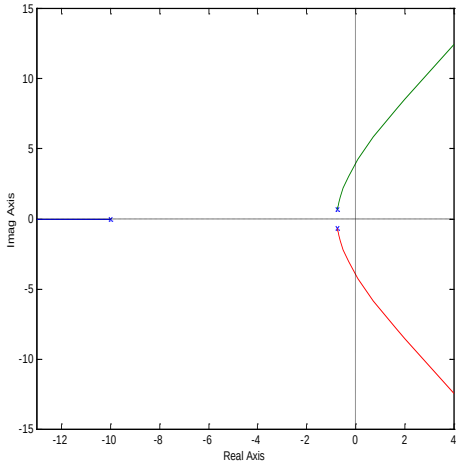


Figura 76

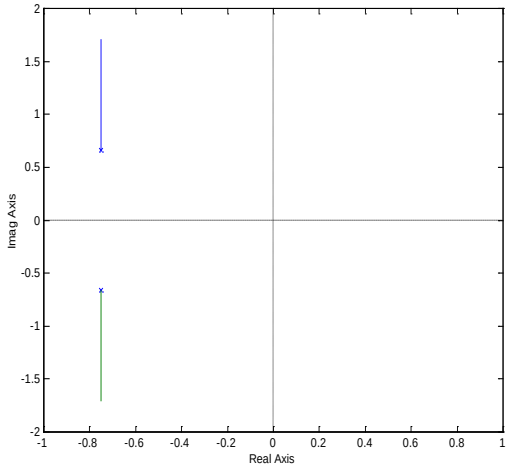


Figura 77

4 El Controlador PID

4.1 Definición

De la misma manera que se estandarizan las interfaces entre bloques del sistema de control realimentado, también la propia estandarización del controlador ha tenido una fuerte difusión en la industria por razones evidentes.

Se ha discutido en las secciones anteriores ejemplos con controlador proporcional. Otras clases de controladores con dinámicas más complejas son posibles. Una de ellas es el controlador PID, que resulta lo suficientemente complejo como para resolver gran cantidad de problemas de control industrial y lo suficientemente simple como para que su ajuste sea posible sin mucha sofisticación. Además, su acción es de fácil interpretación. El controlador PID se ha convertido en un "controlador estándar" de procesos continuos.

El controlador PID es un sistema lineal, cuya relación entrada salida es:

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt} \right]$$

y su función de transferencia:

$$U(s) = K\left[1 + \frac{1}{T_i s} + T_d s \right] E(s)$$

El nombre PID proviene de que su salida está compuesta por tres sumandos: uno Proporcional a la señal de error, otro proporcional a su Integral, otro proporcional a su Derivada.

Las tres acciones: P, I, D, pueden usarse simultáneamente o suprimiendo algunas de ellas. Las combinaciones usuales son: P, PI y PID.

Acción	Ecuación	Función de Transferencia
P (Proporcional)	$u(t) = K\, e(t)$	$U(s) = K\, E(s)$
I (Integral)	$u(t) = K/T_i \int e(t)\, dt$	$U(s) = K/T_i\ E(s)\ /s$
D (Diferencial)	$u(t) = K\, T_d\ de(t)/dt$	$U(s) = K\, T_d\ s\, E(s)$

Por ejemplo el controlador PI es:

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \right]$$

La sintonía del PID (o de sus casos particulares) consiste en encontrar valores adecuados para las constantes K, Ti y Td.

4.2 Sintonía. Regla de Ziegler Nichols.

Existen numerosas maneras de sintonizar el controlador PID. Por ejemplo, en el lugar de las raices, en el diagrama de Nyquist, etc.

En 1942, Ziegler y Nichols desarrollaron técnicas experimentales de sintonía de lazos de control como el de la Figura 78. Su sencillez, el basarse en datos experimentales, y el no necesitar el conocimiento de la función de transferencia H(s) de la planta a controlar las han hecho de enorme popularidad en la industria.

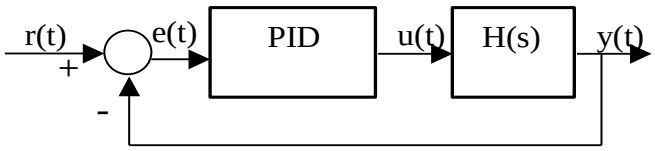


Figura 78

Veremos una de estas reglas: la de *respuesta a escalón*.

Los pasos que hay que seguir son:

Abrir el lazo de control de la Figura 78, y medir la respuesta a escalón de la planta H(s). Considerando el primer tramo de la respuesta a escalón de acuerdo a la Figura 79, determinar L y α.
Cerrar el lazo de control como en la Figura 78, eligiendo los valores de K, Ti, Td según la tabla, de acuerdo al modo de uso del controlador (P, PI, o PID).

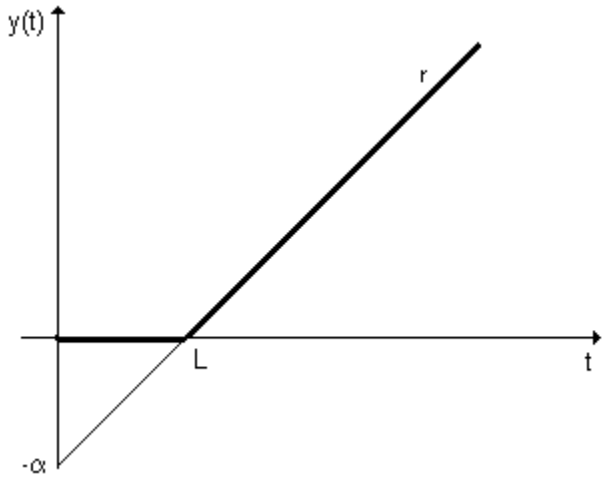


Figura 79

Controlador	K	T_i	T_d
P	$1/\alpha$		
PI	$0.9/\alpha$	3L	
PID	$1.2/\alpha$	2L	L/2

4.3 Aspectos de implementación

El PLC no implementa la relación entrada salida del controlador analógico de forma exacta. Por ejemplo, el manual del PLC del laboratorio indica que el controlador PI de relación entrada salida

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \right]$$

se aproxima por la relación:

$$Y(n) = \frac{KP}{100} * \frac{w - x}{TN/T} + YI(n-1) + \frac{KP}{100} * (w - x)$$

donde:

T = tiempo de ciclo del PLC
w = Setpoint
x = entrada
KP = coeficiente proporcional especificado como porcentaje
TN/T = Ti / T
YI(n-1) = salida del bloque integrador en el ciclo (n-1) del PLC
Y = salida (se corresponde con *u* en la fórmula analógica)

Las funciones w, x e Y se evalúan en el ciclo n del PLC.

La fórmula anterior representa una aproximación del término integral por la fórmula del rectángulo. En efecto:

$$\int_0^{nT}(w(t)-x(t))dt = \underbrace{\int_0^{(n-1)T}(w(t)-x(t))dt}_{\cong YI(n-1)} + \underbrace{\int_{(n-1)T}^{nT}(w(t)-x(t))dt}_{\cong (w(nT)-x(nT))*T} \cong YI(n-1)+(w(nT)-x(nT))*T$$

5 Conversión de unidades

Una cuestión no menor es la conversión de unidades. El problema radica en que el PLC trabaja sobre la base de un sistema de unidades diferente al de las señales físicas que entran y salen de la planta. Esto lleva a que se deba traducir las constantes determinadas para sintonizar el controlador - expresadas en el sistema de unidades de las señales físicas - a constantes equivalentes en unidades del PLC.

Analicemos de forma general el problema de sintonía de un PLC funcionando como controlador PI a partir de las reglas de Z-N. Utilizaremos la notación de la Figura 80.

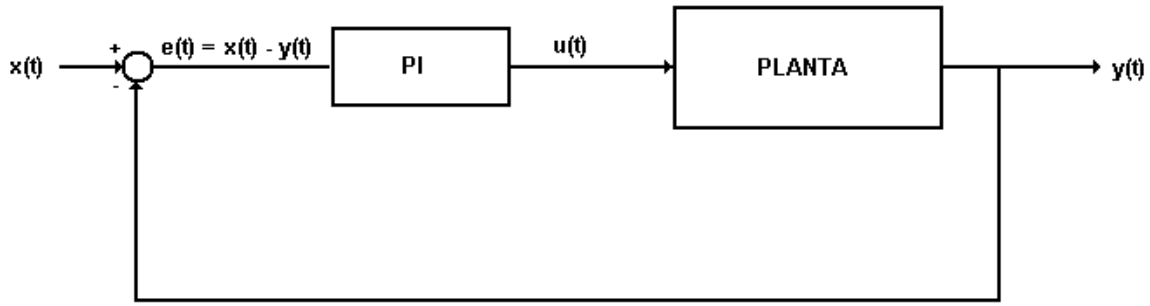


Figura 80

Se define Tscan = Tiempo de muestreo = Tiempo de ciclo del PLC.

Se denota la unidad de una señal a(t) por [a], y por [t] la unidad en que se expresa el tiempo.

Dada una señal a(t), se denota por ad[n] a la aproximación digital de a(t) en el instante n*Tscan. La unidad de ad[n] se denota por [ad].

En general, la aplicación de las reglas de Z-N resulta en constantes KP y Ti, cuyas unidades están dadas por:

$$[K_P] = \frac{[u]}{[e]}$$
$$[T_i] = \frac{[e]*[t]}{[u]}$$

Por claridad, y sin que esto implique una pérdida de generalidad, se supone una aproximación numérica a $\int_0^{n*Tscan} e(t)dt$ análoga a la de los PLCs del laboratorio:

$$\int_0^{n*Tscan} e(t)dt = YI(n-1) + T_{scan} * e(n)$$

donde YI(n-1) es el valor de la aproximación de la integral en el tiempo (n-1)*Tscan.

Manteniendo el sistema de unidades del sistema analógico, el controlador digital que aproxima el PI tiene la forma:

$$u_d[n] = K_P * e_d[n] + \frac{K_p}{T_i}\left(Y(n-1) + T_{scan} * e_d(n)\right)$$

donde

$$[ud] = [u].$$

La relación entrada-salida en el sistema de unidades del PLC será:

$$u_d^{PLC}[n] = K_p^{PLC} * e_d^{PLC}[n] + \frac{K_p^{PLC}}{T_i^{PLC}}\left(YI^{PLC}(n-1) + T_{scan} * e_d^{PLC}(n)\right)$$

Supongamos el sistema de unidades del PLC es tal que:

$$\begin{cases} u_d^{PLC} = M * u_d \\ e_d^{PLC} = N * e_d \end{cases}$$

Es claro entonces que:

$$YI^{PLC} = N * YI$$

Por lo tanto, $\left(K_P^{PLC}, YI^{PLC}\right)$ se relacionan con (Kp, YI) según:

$$\begin{cases} K_p^{PLC} * \frac{N}{M} = K_p \Rightarrow K_p^{PLC} = K_p * \frac{M}{N} \\ T_i^{PLC} = T_i \end{cases}.$$

Las relaciones anteriores expresan las constantes equivalentes en el sistema de unidades del PLC en función de las constantes del problema.

APÉNDICE – RECORDATORIO DE CONTROL

1 Sistemas

El objeto básico de la teoría de control es el *sistema*.

El sistema consiste una relación causa-efecto en el que la señal de salida "y" es función de la señal de entrada "u".

$$y = S[u]$$

La entrada "u" es una señal que se puede manipular, mientras la salida "y" es una señal que se puede observar.

Un sistema se puede representar como un diagrama de bloques en el que el sentido de las flechas indica la relación causa-efecto (Figura 81).

Figura 81

En lo que sigue se considera las señales u e y como funciones del tiempo u(t) e y(t), que representan magnitudes de interés (temperaturas, voltajes, presiones, concentraciones, etc.). El tiempo t es una variable continua y real.

2 Sistemas lineales e invariantes en el tiempo

Un sistema S se dice lineal si cualquier combinación lineal de entradas produce como salida la correspondiente combinación lineal de las salidas que se hubieran obtenido para cada entrada:

Si $\begin{cases} y_1(t) = S[u_1] \\ y_2(t) = S[u_2] \end{cases}$ entonces $S[\alpha_1 u_1 + \alpha_2 u_2] = \alpha_1 y_1 + \alpha_2 y_2 \quad \forall\ u_1, u_2, \alpha_1, \alpha_2.$

El sistema se dice invariante en el tiempo si cualquiera sea el retardo considerado, al retardar la entrada se obtiene la misma salida pero retardada en la misma cantidad:

Si $y(t) = S[u(t)]$ entonces $y(t+\tau) = S[u(t+\tau)] \quad \forall\ u, \tau.$

3 Función de Transferencia

Sea un sistema dinámico lineal e invariante en el tiempo, en tiempo continuo, que parte del reposo. Si la entrada y salida son funciones u:R→R, y:R→R, con Transformada de Laplace U(s) y Y(s) , entonces existe una función H(s), denominada *Función de Transferencia*, tal que:

$$H(s) = \frac{Y(s)}{U(s)}$$

La función de transferencia H(s) es la transformada de Laplace de la función de respuesta a impulso:

$$H(s) = L\{h(t)\}$$

Si además el sistema es de parámetros concentrados, H(s) es una función racional.

Figura 82

4 Control en lazo abierto y control en lazo cerrado.

Supongamos una planta H a controlar, y el problema de control formulado como "¿ cuál es la entrada u(t) que produce una salida y(t) de características deseadas ?".

De las distintas maneras de abordar este problema, hay dos formas básicas:

Precalcular la señal u, o sea, precalcular el conjunto de valores u(t) a aplicar a la planta ∀t. Este modo se denomina *control en lazo abierto*.

Generar la u(t) en forma dinámica tomando en cuenta la evolución real de la salida objetivo y(t). Este modo se denomina *control en lazo cerrado***.**

Ambos enfoques se representan gráficamente en la Figura 83 (esquemas (2) y (3)), en un problema típico de regulación de la salida de la planta (y(t)).

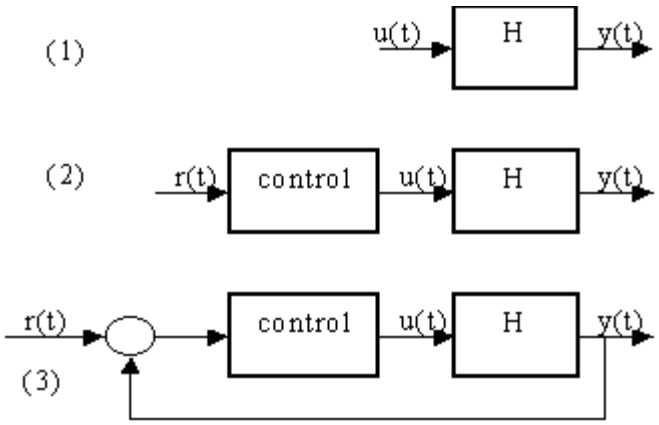


Figura 83

5 Respuesta en Frecuencia

Cuando un sistema lineal, invariante en el tiempo y estable es excitado con una entrada sinusoidal estacionaria, su salida en régimen ($t \to \infty$) se comporta como una señal sinusoidal de la misma frecuencia, amplificada y desfasada. Es decir, si:

$$u(t) = A * Y(t) * \text{sen}(\omega t)$$

entonces

$$y(t) \xrightarrow[t \to \infty]{} A * H(j\omega) * \text{sen}(\omega t + Arg(H(j\omega)))$$

El factor de amplificación y el defasaje dependen sólo de la frecuencia ω, y son el módulo y la fase de la función de transferencia evaluada en s = jω.

Se denomina *Respuesta en Frecuencia* a la función H(jω), que caracteriza en módulo y fase la respuesta asintótica del sistema frente a una entrada sinusoidal.

Dicha función tiene diversos modos de representación, más o menos convenientes para distintos procedimientos de diseño, entre ellos: el *Diagrama de Bode* y el *Diagrama de Nyqüist*

6 Diagrama de Bode

Es una manera de representar la respuesta en frecuencia H(jω) de un sistema de Función de Transferencia H(s). Consiste en dos gráficos: uno para representar la función módulo y otro para representar la función argumento.

- En el primero se grafica $20 \log |H(j\omega)|$ vs. log(ω). $0<\omega<\infty$.
- En el segundo se grafica $\angle H(j\omega)$ vs. log(ω). $0<\omega<\infty$.

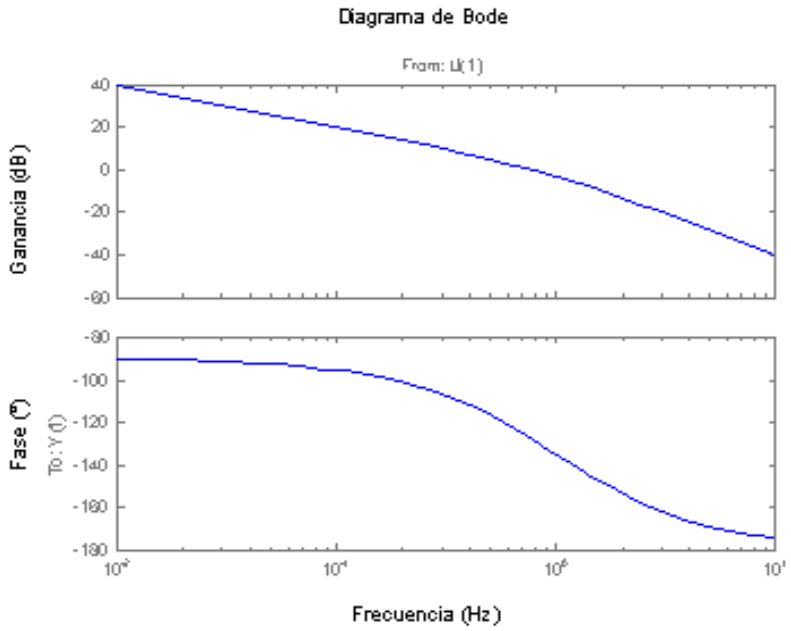


Figura 84

7 Diagrama de Nyqüist

Es una manera de representar la respuesta en frecuencia H(jω) de un sistema de Función de Transferencia H(s).

Se dibuja en el plano complejo la curva que describe la función H(jω) cuando ω varía desde -∞ hasta +∞ (jω recorre el eje imaginario).

Esta curva es simétrica respecto del eje real ya que

$$H(j\omega) = \overline{H(-j\omega)}$$

Por esta razón a veces se dibuja sólo la parte correspondiente a ω>0.

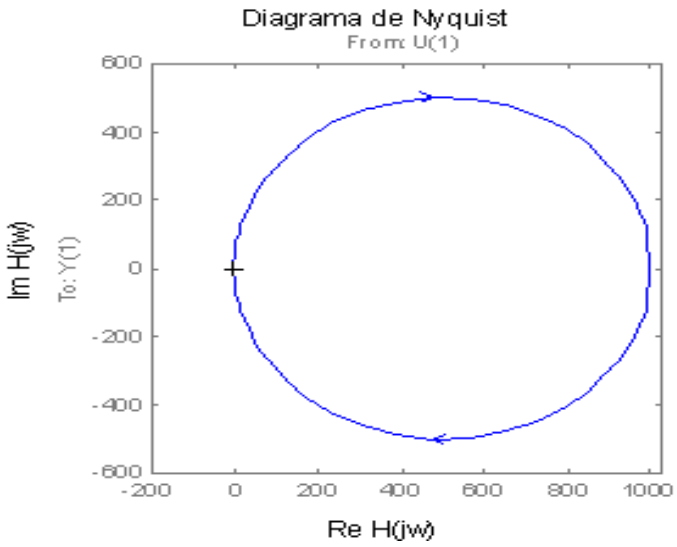


Figura 85

8 Márgenes de estabilidad

Al diseñar un sistema realimentado de control, la primer condición que debe satisfacerse es su estabilidad. Naturalmente surge la pregunta: *aún siendo estable el sistema nominal diseñado, ¿qué tanto pueden variar la planta o el controlador sin perder la estabilidad del lazo cerrado?*

Una manera estándar de estimar este margen es determinar cuánto pueden modificarse por separado la ganancia y la fase del lazo abierto sin que se pierda la estabilidad. Estas dos magnitudes se denominan *margen de ganancia* y *margen de fase* respectivamente.

Sea G(s) la función de transferencia de la planta. Para mantener la estabilidad del sistema en lazo cerrado, G(s) no puede multiplicarse por un factor mayor a MG, ni decrementar su fase en más de MF, donde:

Margen de Ganancia $MG = \dfrac{1}{|G(j\omega_C)|}$, siendo ω_C tal que $Arg(G(j\omega_C)) = -\pi$

Margen de Fase $MF = \pi + Arg(G(j\omega_C'))$, siendo ω_C' tal que $|G(j\omega_C')| = 1$.

9 Lugar de las raices (root-locus)

Es el lugar geométrico de los puntos del plano complejo que son solución (para algún valor de k) de la ecuación polinomial:

$$p(s) + k\, q(s) = 0 \qquad (1)$$

Donde p(s) es un polinomio de grado n, q(s) polinomio de grado m, ambos de coeficientes reales y k es un real que varía entre $+\infty$ y $+\infty$. La parte del lugar que corresponde a valores de k: $0 \le k < +\infty$ se denomina *lugar geométrico positivo*, y la parte correspondiente a $-\infty < k \le 0$ *lugar geométrico negativo.*

De (1) resulta que todos los puntos s que pertenecen al lugar de las raíces satisfacen:

$$\mathrm{Im}\left\{\frac{q(s)}{p(s)}\right\} = 0$$

Se define:

$$\begin{cases} \mathrm{Re}\left\{\dfrac{q(s)}{p(s)}\right\} < 0 = \text{Lugar geométrico positivo} \\ \mathrm{Re}\left\{\dfrac{q(s)}{p(s)}\right\} > 0 = \text{Lugar geométrico negativo} \end{cases}$$

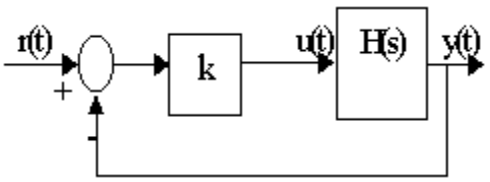


Figura 86

El lugar de las raíces (Figura 87) es el lugar de los polos del sistema H(s) = q(s)/p(s), realimentado unitariamente y con controlador proporcional k, como en la Figura 86.

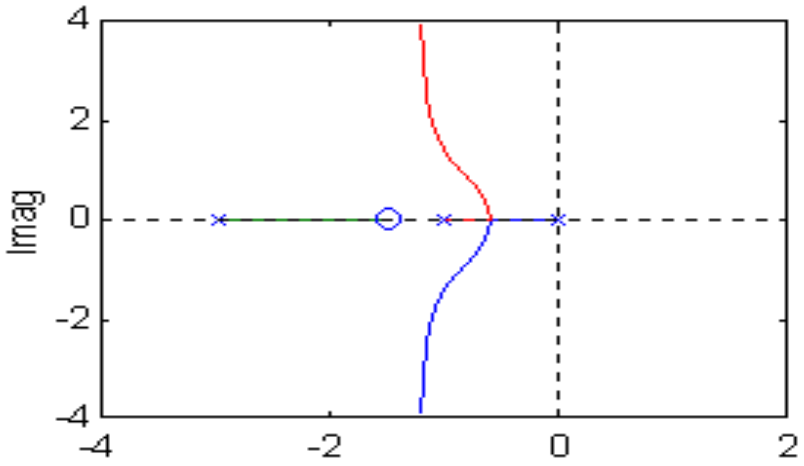


Figura 87

CAPÍTULO 7: LENGUAJE IL

1 Estructura de un programa en IL

El lenguaje IL, o *Instruction List*, es un lenguaje de bajo nivel.

Un programa en lenguaje IL consta de una lista de *instrucciones*. Cada instrucción debe comenzar al inicio de una línea. Una instrucción consta de:

1) un *operador*, que puede ser modificado con uno o más *modificadores*
2) uno o más *operandos*, según la operación, separados por comas (",")

De forma opcional se puede comenzar la línea de la instrucción por un *label* seguido de dos puntos (":"). Asimismo, de forma opcional se puede terminar la línea en un comentario, entre los caracteres "(*" y termina en "*)".

Se pueden introducir líneas vacías entre instrucciones, así como comentarios en líneas sin instrucciones.

2 El registro IL

Todas las instrucciones involucran al **registro IL** o **resultado actual**. De esta forma, cualquier operación transcurre de forma que:

1) si hay más de un operando, uno de los operandos es el valor del registro IL antes de la operación
2) el resultado de la operación se devuelve en el registro IL

Desde el punto de vista lógico, el registro IL involucrado en operaciones cuyo resultado es de tipo bit, es un bit, y el registro IL involucrado en operaciones cuyo resultado es de tipo palabra, es una palabra.

3 Ejemplos de instrucciones

Veamos ejemplos de instrucciones:

```
Label       Operador    Operando          Comentario

Inicio:     LD          BOTON                   (*BOTON -> registro IL"*)
ANDN        %I62.02                             (*!%I62.02 AND (registro
IL) ->
                                                registro IL*)
            ST          INICIO_MOTOR    (*registro IL -> INICIO_MOTOR*)
```

4 Modificadores de operadores

Existen dos modificadores:

N = negación booleana del operador (por ej. ORN %I62,00 se interpreta OR NOT I62,00)

C = ejecución condicional al valor del registro IL. Sólo se ejecuta la sentencia si el registro IL (resultado

actual) es TRUE (o tiene valor no nulo en caso de operaciones no booleanas). El modificador C se

puede combinar con N para ejecutar la instrucción condicionado a que el registro IL (resultado

actual) sea FALSE (nulo, para operaciones no booleanas).

5 Lista de instrucciones

La siguiente tabla se refiere a las operaciones que permiten los PLCs utilizados en el laboratorio. En lo que sigue, el registro IL (resultado actual) se denomina A.

Operador	Modificadores	Operadores	Descripción
LD	N	Variable, constante	Operando -> A
!	N	Variable, constante	Operando -> A
ST	N	Variable	A -> Operando
S		Variable	TRUE -> Operando
R		Variable	FALSE -> Operando
!BA 0		Nombre de bloque funcional	Llama la función del bloque funcional
CAL		Nombre de bloque funcional	Llama la función del bloque funcional
CAL_FB		Llamada a subrutina	
VTASK		Llamada a interrupción	
JMP	C N	Label	Salta a un label
AND	N	BOO	BOO AND A -> A
&	N	BOO	BOO AND A -> A
OR	N	BOO	BOO OR A -> A
/	N	BOO	BOO OR A -> A
ADD		Variable, constante	Operando + A -> A
+		Variable, constante	Operando + A -> A
SUB		Variable, constante	Operando - A -> A
-		Variable, constante	Operando - A -> A
MUL		Variable, constante	Operando * A -> A
*		Variable, constante	Operando * A -> A
DIV		Variable, constante	Operando / A -> A
:		Variable, constante	Operando / A -> A
GT		Variable, constante	Test: >
>		Variable, constante	Test: >

GE		Variable, constante	Test: >=
>=		Variable, constante	Test: >=
EQ		Variable, constante	Test: =
=?		Variable, constante	Test: =
LE		Variable, constante	Test: <=
<=		Variable, constante	Test: <=
LT		Variable, constante	Test: <
<		Variable, constante	Test: <
NE		Variable, constante	Test: <>
<>		Variable, constante	Test: <>

CAPÍTULO 8: SFC

1 Introducción

El lenguaje SFC es un lenguaje *gráfico* basado en operaciones *secuenciales*. En su forma más básica, un programa en lenguaje SFC consiste de una secuencia de *pasos* (*steps*) y *transiciones* (*transitions*).

La secuencia de pasos y transiciones deberá cumplir las siguientes reglas gráficas:

1) entre dos pasos debe haber al menos una transición
2) entre dos transiciones debe haber al menos un paso
3) debe existir al menos un paso inicial.

La ejecución del programa comienza en el *paso inicial*, que tiene un símbolo característico. Un paso, en particular el paso inicial, consiste de una secuencia de *acciones*.

Si el paso inicial es seguido de otro paso, la regla 1 establece que entre ambos pasos debe existir una *transición*. Una transición consta de un conjunto de sentencias, cuyo resultado es un *booleano*, esto es, puede tomar sólo uno de los dos valores TRUE o FALSE.

Para introducir la secuencia general de ejecución de un programa SFC, consideremos el programa de la Figura 88. El paso 1 se representa por un rectángulo con borde doble. Esto significa que el paso 1 es un paso inicial. El paso 2 es simplemente un rectángulo. Cada una de las transiciones se representa por una conexión cruzada por una línea horizontal.

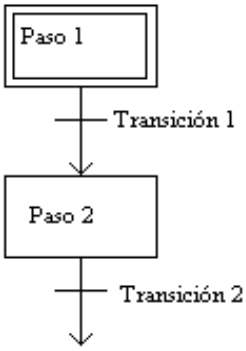


Figura 88

La secuencia de ejecución del programa de la Figura 88 se muestra en la Figura 89, y se explica a continuación.

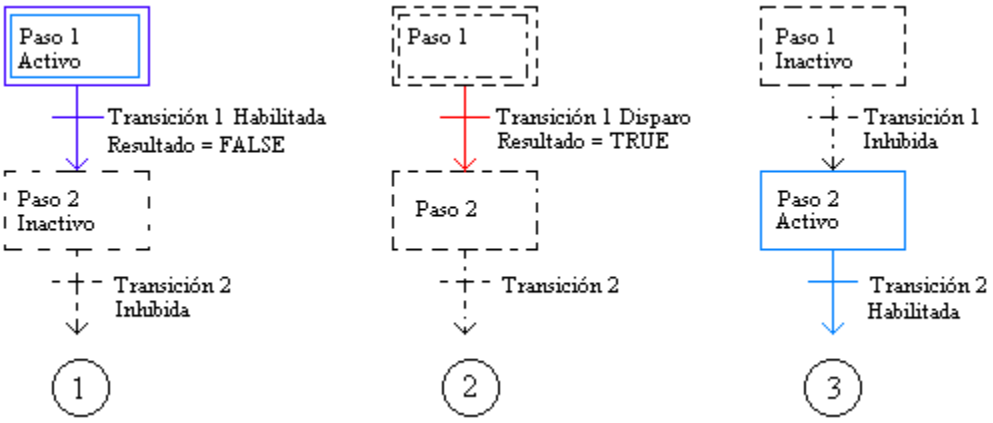


Figura 89

El PLC ejecutará cíclica y secuencialmente las *acciones* del paso 1 y las *sentencias* de la transición 1, hasta que el resultado de la transición 1 sea TRUE.

Cuando el resultado de la transición 1 pasa a ser TRUE, diremos que la transición 1 se *dispara*. Durante el intervalo de tiempo desde el inicio de la ejecución del programa hasta que la transición 1 se dispara, diremos que el paso 1 está en estado *activo* y la transición 1 está *habilitada*.

Tras el disparo de la transición 1, el paso 1 pasa a estado *inactivo*, la transición 1 queda *inhibida*. El paso 2 pasa a estado activo, y la transición 2 queda habilitada.

El paso 2 permanecerá en estado activo y la transición 2 habilitada en tanto la transición 2 no se dispare. Dicho de otra forma, se ejecutarán cíclica y secuencialmente las acciones del paso 2 y las sentencias de la transición 2, en tanto la transición 2 no se dispare. El disparo de la transición activará el paso 3 y habilitará la transición 3, y así sucesivamente.

1.1 Ejemplo

Veamos el programa de encendido / apagado de una bomba, presentado en el capítulo 5, escrito ahora en lenguaje SFC.

Por comodidad, re escribimos la especificación del problema:

La bomba será encendida si:

1) Se pulsa el botón de arranque.
2) La protección térmica está deshabilitada.
3) Está abierto el botón de emergencia.
4) Está abierto el botón de parada.

Desde un tiempo T después del encendido, no puede haber ni sobre corriente ni baja corriente. Expresado de otra forma, desde un tiempo T después del arranque, la corriente I debe cumplir $I_{MIN} < I < I_{MAX}$, siendo I_{MIN} e I_{MAX} límites prefijados.

El motor de la bomba se apagará si:

1) Se pulsa el botón de parada.
2) Se cierra la protección térmica.
3) Se pulsa el botón de emergencia.
4) Los límites de corriente no son los correctos.

La secuencia de pasos y transiciones del programa se muestra en la Figura 90.

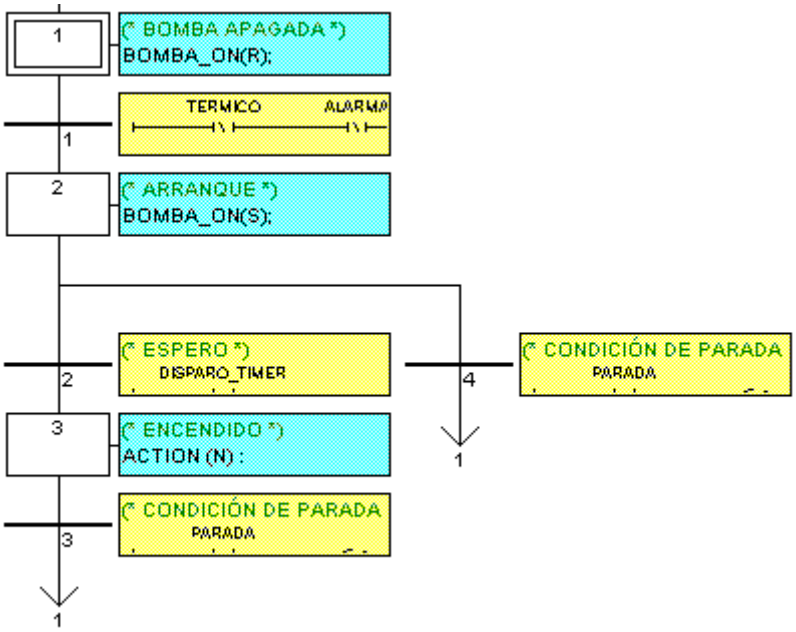


Figura 90

En la Figura 90 hay dos elementos nuevos, cuya interpretación aparece como bastante intuitiva: la bifurcación de transiciones (2 y 4) tras el paso 2 y las flechas que siguen a las transiciones 3 y 4.

La bifurcación que sigue al paso 2 se denomina divergencia simple. Tanto la transición 2 como la 4 quedan habilitadas al activarse el paso 2. La primera ellas que se dispare, guiará la ejecución del programa por su rama.

Las flechas que siguen a las transiciones 3 y 4, no son otra cosa que saltos al paso 1, como indica el 1 al lado de cada una de las flechas.

Para analizar el programa en SFC de la Figura 90, re escribimos las especificaciones en forma de secuencia de *estados*. Procedemos, suponiendo que la bomba está inicialmente apagada. En el análisis que sigue, una variable precedida del caracter "!" se interpreta como la variable negada.

1) En estado "BOMBA APAGADA", mantengo la bomba apagada (BOMBA_ON = 0). La condición de arranque es

   (!TERMICO AND !ALARMA AND !PARADA AND ARRANQUE).

Es decir, en tanto esta condición no se cumple, me mantengo en el estado "BOMBA APAGADA".

2) Cuando la condición en 1) sea verdadera, paso a un estado transitorio, "ARRANQUE", en el que enciendo la bomba (BOMBA_ON = 1). En el estado "ARRANQUE" pueden suceder dos cosas:

   a) (TERMICO OR ALARMA OR PARADA) = 1. En este caso vuelvo al estado "BOMBA APAGADA".
   b) Pasan 5 segundos sin que pase la condición en a). En este caso paso al estado ENCENDIDO.

3) En el estado ENCENDIDO, mantengo (BOMBA_ON) en tanto se mantenga la condición

   (ERROR OR TERMICO OR ALARMA OR PARADA) = 0,

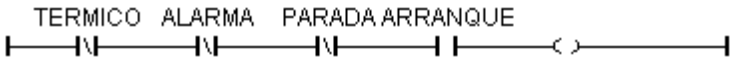
siendo ERROR = (CORRIENTE > CORRIENTE_MAX) OR (CORRIENTE < CORRIENTE_MIN).

La secuencia de pasos y transiciones del programa (Figura 90) se deriva de forma inmediata del análisis anterior. Las acciones en cada estado, así como las condiciones de transición de un estado a otro, determinan las *acciones* de cada uno de los pasos, y las *condiciones* que disparan las transiciones.

PASO 1 (*BOMBA APAGADA*):

BOMBA_ON(R); (*Resetea BOMBA_ON cuando se activa el paso*)

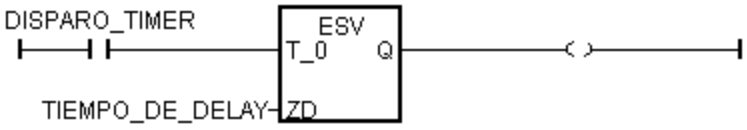
TRANSICIÓN 1:



PASO 2 (*ARRANQUE*)

DISPARO_TIMER; (*DISPARO_TIMER = 1 mientras que paso 2 activo, y 0 mientras paso 2 inactivo*)
BOMBA_ON(S); (*Setea BOMBA_ON cuando se activa el paso*)

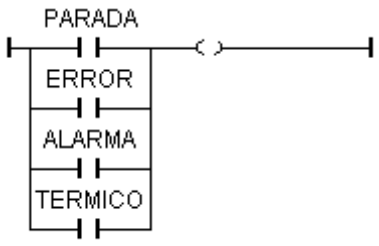
TRANSICIÓN 2 (*ESPERO*)



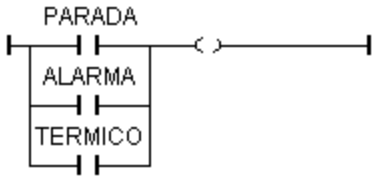
PASO 3 (*ENCENDIDO*)

```
ACTION (N) :        (*Acción tipo N: todas las sentencias hasta END_ACTION se
ejecutarán en cada ciclo de ejecución de las acciones del  paso 3*)
                LD      CORRIENTE
                GE      CORRIENTE_MAX
                ST      ERROR
                LD      CORRIENTE_MIN
                GE      CORRIENTE
                ST      ERROR_MIN
                LD      ERROR_MIN
                OR ERROR
                ST ERROR    (*ERROR = (CORRIENTE > CORRIENTE_MAX) OR
                             (CORRIENTE < CORRIENTE_MIN) *)
END_ACTION;
```

TRANSICIÓN 4 (*CONDICIÓN DE PARADA*)



TRANSICIÓN 3 (*CONDICIÓN DE PARADA PRECIPITADA*)



2 Componentes de SFC

Los componentes básicos del lenguaje SFC son los *pasos,* los *pasos iniciales*, las *transiciones* y los *saltos a pasos.*

2.1 El paso y el paso inicial

Un paso se representa por un rectángulo (Figura 91). A cada paso se le asigna un número y un comentario. El *número* junto con el *comentario* asociado al paso conforman el *nivel 1 de programación del paso.*

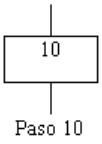


Figura 91

Un paso tiene dos estados posibles: estado *activo* o estado *pasivo* (inactivo). En general, el estado de un paso será *activo* desde el disparo de una transición cualquiera que le precede (conectada al paso) hasta el disparo de una transición transición cualquiera que le sigue (conectada al paso).

Un *paso* consiste de una secuencia de *acciones,* que se ejecutan cíclica y secuencialmente durante el estado activo del paso. Las acciones conforman el *nivel 2 de programación del paso.*

De esta forma, cada paso forma un bucle junto con la transición que le sigue, cuya condición de salida es el disparo de la transición.

Las acciones del paso pueden ser de tres tipos: *boolean, pulse* o *non-stored*.

Una acción tipo *boolean* relaciona el estado de una variable con el estado del paso.

Una acción tipo *pulse* se ejecuta *una sola* vez durante el estado *activo* del paso.

Una acción tipo *non-stored* se ejecuta en cada ciclo de ejecución de las acciones del estado.

Dentro de un paso, cada acción debe terminar en ";".

Un programa SFC comienza a ejecutarse en el *paso inicial*, representado por un rectángulo de borde doble (Figura 92). Esto es, el paso inicial pasa a estado activo cuando el programa comienza a ejecutarse.

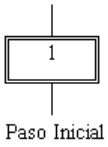


Figura 92

2.2 La transición

Las transiciones se representan por conexiones cruzadas por una pequeña barra horizontal (Figura 93). A cada transición se le asigna un número y un comentario. El *número* junto con el *comentario* asociado a la transición conforman el *nivel 1 de programación de la transición.*

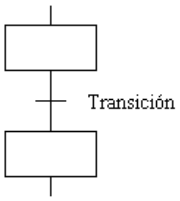


Figura 93

Una transición tiene dos estados posibles: habilitada o no habilitada. Una transición estará habilitada sólo si todos los pasos unidos a ella están en estado *activo*.

Una transición es una secuencia de sentencias, cuyo resultado es un *booleano*. Estas sentencias conforman el *nivel 2 de programación de la transición.* Con la transición habilitada, las sentencias se evaluarán de forma cíclica, junto con las acciones de los pasos activos.

Una vez habilitada, no se activará el paso siguiente a la transición en tanto la transición no se *dispare*. La condición de disparo de la transición es que el *resultado* de las sentencias en la transición sea TRUE.

Una transición puede ser programada en IL o en QUICK LD (no en ambos lenguajes simultáneamente). Si se programa en QUICK LD, constará de un único rung, y el *resultado* será el almacenado en el *coil* que termina el *rung*. Si se programa en IL, el *resultado* será el *valor del registro IL* al terminar la ejecución de las sentencias en la transición.

En el caso que la programación de una transición sea en IL, la no referencia al *registro IL* en *por lo menos una* sentencia de la transición, llevará a *errores en tiempo de ejecución* del programa.

2.3 Los saltos a pasos

El símbolo de salto es simplemente una conexión terminada en flecha (Figura 94). El salto se puede utilizar para indicar una conexión entre una *transición* y un *paso*. No se permiten saltos de pasos a transiciones. El símbolo del salto debe tener un número de referencia al paso del salto.

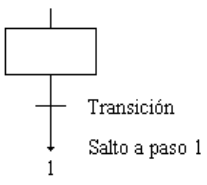


Figura 94

3 Las divergencias y convergencias

Las convergencias y divergencias pueden ser de dos tipos. Divergencias/Convergencias simples (OR) o dobles (AND).

3.1 Convergencias/Divergencias Simples

Una divergencia simple es la conexión de *un paso* a *más de una transición* (Figura 95). Una convergencia simple es la conexión de más de una transición a un paso (Figura 95).

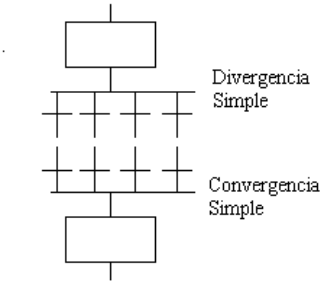


Figura 95

La actividad del paso de una divergencia simple habilita *todas* las transiciones de la divergencia simple.

El disparo de *una* transición cualquiera de una convergencia simple *activa* al paso de la convergencia simple.

3.2 Convergencias/Divergencias Dobles

Una divergencia doble es la conexión de *una transición* a *más de un paso* (Figura 96). Una divergencia doble es la conexión de más de un paso a una transición (Figura 96).

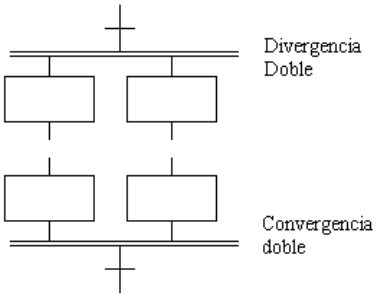


Figura 96

El disparo de la transición de una divergencia doble activa todos los pasos de la divergencia doble.

Una transición de una convergencia doble está habilitada si *todos* los pasos de la convergencia doble están activos.

4 Macros

Un macro es una representación única de un grupo único de pasos y transiciones.

5 Acciones en los pasos

Como fue dicho anteriormente, existen tres tipos de acciones posibles en un paso: *boolean*, *pulse* o *non-stored*.

5.1 Acción tipo *boolean*

Una acción tipo *boolean* relaciona el estado de una variable con el estado del paso. Los distintos tipos de acciones tipo *boolean* son:

var; asigna la señal de actividad del paso a la variable "var"
/var; asigna la negación de la señal de actividad del paso a la variable "var"
var(S); pone "var" en 1 cuando la señal de actividad del paso pasa a 1
var(R); pone "var" en 0 cuando la señal de actividad del paso pasa a 1

5.2 Acción tipo *pulse*

Una acción tipo *pulse* se ejecuta *una sola* vez cada vez que se *activa* el paso. Sintaxis:

```
ACTION (P):
    (* Instrucciones IL *)
END_ACTION;
```

5.3 Acción tipo *non-stored*

Una acción tipo *non-stored* se ejecuta en cada ciclo de ejecución de las acciones del estado. Sintaxis:

```
ACTION (N):
    (* Instrucciones IL *)
END_ACTION;
```

CAPÍTULO 9: HERRAMIENTAS DE DIAGNÓSTICO

Cualquier PLC tiene códigos de error para identificar errores de programación y errores de operación. Cuando algo está mal, aparecen en el monitor códigos de error, ya sea en forma de código o en forma de mensaje, según el sistema.

Asimismo, cualquier PLC tiene LEDs en el módulo CPU y en el de Entrada/Salida, y en el de alimentación. En general, un LED rojo indica un problema y un LED verde indica funcionamiento correcto. El parpadeo de un LED significa a menudo que hay una función activa o que el módulo está esperando por algo. La interpretación correcta de los LEDs de diagnóstico puede ahorrar gran cantidad de tiempo a la hora de la puesta en funcionamiento del sistema.

De forma general, se puede clasificar los errores de un PLC en: errores fatales, errores no fatales y errores de programación o configuración.

Frente a un error fatal, el PLC abandona el modo de ejecución y entra en el modo falla. Un error fatal puede ser consecuencia de la detección de una componente fallida en la prueba de arranque del PLC, o al momento de utilizar la componente durante la ejecución del programa. Ciertos problemas de programación o configuración pueden ser también causa de errores fatales.

La constatación de un error no fatal no hace que el PLC no abandone el modo de ejecución. Las causas de un error no fatal pueden ser: baja tensión en la batería de respaldo, ciertos errores de configuración o programación. El PLC indica errores no fatales a través de bits de error o a través de la escritura de códigos en la memoria. El programa de usuario es responsable de leer los códigos de la memoria, y actuar en consecuencia.

Los errores de configuración o programación no pueden ser detectados automáticamente por el PLC. Los debe detectar el usuario, utilizando su capacidad de resolución de problemas.

CAPÍTULO 10: COMUNICACIONES

Como se dijo en la introducción del curso, los PLC tienen la posibilidad de comunicarse con otros dispositivos digitalmente.

Un bus muy interesante que soportan los PLC del laboratorio es el bus Profibus. Un par de hilos conectados a un puerto de comunicaciones que se adiciona, permite a estos PLCs comunicarse con dispositivos remotos, que pueden ser módulos de entradas/salidas analógicas, digitales, e inclusive otros PLCs. Esto permite descentralizar el sistema, instalando los módulos cerca de los sensores y los actuadores, con el consiguiente ahorro de costo en el cableado general.

Asimismo, se vio como los PLCs del laboratorio se comunican con las extensiones a través de un bus paralelo, que permite mayor velocidad de comunicaciones.

Como Profibus, existen gran cantidad de protocolos de software y hardware que permiten la comunicación entre PLCs del mismo suministrador, así como entre PLCs de distinta procedencia.

La comunicación se extiende no sólo a PLCs, sino también a instrumentos de campo, como controladores, medidores de pH, caudalímetros, transmisores de presión, balanzas, medidores de nivel, etc.

En el campo de la transmisión de señales en un sistema distribuido industrial, se presentan dos opciones:

- Conexión directa del PLC a las entradas/salidas de los instrumentos (principalmente 4.20 mA en *lazos de corriente*).

- Comunicación digital entre dispositivos vía un bus serie.

La comunicación digital presenta muchas ventajas frente a la conexión directa. A saber:

- Permite bajar los costos de cableado como ya mencionamos.

- Permite transmitir gran cantidad de información por un mismo cable (señal a medir, señales auxiliares, diagnósticos, etc.)

- Permite configurar un control (fijando constantes, como el set-point, el valor de la derivada, integral, etc.) o cualquier otro instrumento como los mencionados en forma remota.

- No hay pérdida de precisión de la señal analógica. Normalmente, la resolución de los conversores A/D (D/A) de los instrumentos que mencionamos es mayor a 12 o 14 bits, mientras la resolución de los conversores de los PLCs es de 8 a 12 bits. Por tanto, la conexión de la señal analógica al conversor A/D (D/A) del PLC redundaría en una pérdida de precisión respecto del conversor del instrumento. Esta pérdida de precisión podría resultar inaceptable.

Como ejemplo, consideremos un medidor de nivel de un tanque de petróleo, que trabaja con 20 bits de precisión. La conexión de su salida analógica 4-20mA a un conversor de 16 bits de un PLC resultaría en una pérdida de precisión: la precisión del instrumento es del orden del millón de cuentas, mientras la precisión del conversor A/D de 16 bits es de sólo 65536 cuentas. Esto es, la precisión bajó 16 veces. Utilizando un enlace de comunicaciones, se pueden transmitir los 20 bits en un paquete de comunicaciones, manteniendo la precisión original.

No está de más señalar que la salida o entrada 4-20mA de instrumentos analógicos es un estándar, al que se llegó luego de muchos años. Llevó un buen tiempo lograr que tanto un caudalímetro como un indicador de pH tuvieran salidas 4-20 mA. Sin embargo, la tendencia actual es la sustitución de las entradas/salidas 4-20 mA de los instrumentos por módulos de comunicación. Excepción a esta tendencia de sustitución son las válvulas de control, que se accionan neumáticamente a través transductores corriente eléctrica/presión, con entrada 4-20 mA y salida normalizada en el rango 3 psi a 15 psi; por más que existan válvulas con comunicaciones, su elevado costo hace que en la gran mayoría de las aplicaciones se utilicen válvulas 4-20 mA.

El gran problema en el campo de las comunicaciones digitales de los PLCs es que por ahora no existe un único protocolo estándar, sino que hay varios protocolos utilizados en la industria. Algunos propietarios de un fabricante y otros abiertos utilizados por varios fabricantes.

Los protocolos propietarios presentan un gran obstáculo a la estandarización. Muchos fabricantes de instrumentación industrial utilizan protocolos propietarios, lo cual hace imposible el diálogo entre dispositivos de distinto fabricante en una red de comunicaciones. La selección de un PLC y un dispositivo de distintos fabricantes, obliga a la utilización del estándar 4-20 mA en detrimento de la línea de comunicaciones.

Hoy día existe un número elevado de protocolos abiertos - Modbus, Profibus, Foundation Fieldbus, HART, AS-i, DeviceNet, CAN (utilizado en la industria automotriz), etc., que difieren no sólo en el protocolo de software, sino también en la capa física.

Aunque no exista un estándar de comunicaciones, se ha hecho extendido el uso del protocolo Modbus, utilizado para la comunicación entre el PLC y el PC en el laboratorio. Como se indicó en la introducción, este protocolo fue desarrollado por Modicon. Debido a que Modbus fue el primer protocolo establecido, la gran cantidad de aplicaciones que lo utilizaban obligó a muchos fabricantes a implementarlo. Otro protocolo utilizado extensivamente hoy en día es Profibus.

El protocolo de comunicación digital HART utiliza modulación FSK por la línea analógica 4-20 mA de dos hilos, según el estándar Bell103/113. Al igual que el Modbus, este protocolo permite la conexión de diversos instrumentos, comunicándose en régimen de maestro-esclavo. Al configurarse en modo de comunicación digital, todas las líneas quedan en reposo de 4 mA. Los bits se transmiten superpuestos a esta señal en forma de tonos de voltaje: una frecuencia de 1200 Hz significa "0" y una frecuencia de 2200Hz significa "1" (Figura 97).

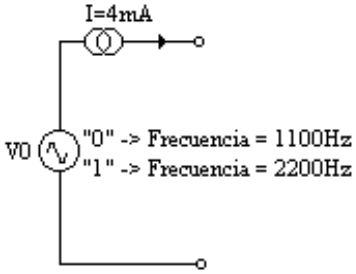


Figura 97

El protocola HART puede ser utilizado en forma individual en cada instrumento conectado por su cable de 4..20 mA. Si bien esto no resulta en un ahorro del cableado (se mantiene un cable por instrumento) posibilita la combinación de ambos esquemas: una señal principal transmitida en forma convencional por 4..20 mA sumada a una señal digital sobreimpuesta (FSK) para transmisión de otras variables del instrumento, diagnósticos y configuración. La señal de 4..20 mA es un mecanismo más robusto y confiable que la comunicación digital en ambientes industriales y por esto el protocolo HART es elegido muchas veces por sobre otros.

Hoy en día la gran mayoría de los protocolos son maestro-esclavo. En un protocolo de este tipo, la estación que oficia de maestro de la red interroga a los PLC remotos e instrumentos, procesando la información que recibe. El ciclo de interrogación del maestro a los esclavos se denomina en inglés *polling*.

La Figura 98 muestra un ciclo de comunicaciones típico en un sistema maestro – esclavo, en un sistema con un maestro y dos esclavos. En cada paso, la unidad transmisora aparece en gris, y las estaciones receptoras en estado de recepción en blanco. Transcurrido un intervalo de tiempo dado desde el último ciclo de *polling*, durante el cual la línea permaneció en reposo, el maestro inicia un nuevo ciclo de *polling*, enviando un mensaje al esclavo 1, y pasando a continuación a estado de recepción. Ambos esclavos escuchan el mensaje, pero sólo el esclavo 1 responde, ya que fue el destinatario del mensaje del maestro. Una vez recibida la respuesta del esclavo 1, el maestro interroga al esclavo 2, pasando a continuación a estado de recepción. Una vez más, ambos esclavos reciben el mensaje, pero sólo el esclavo 2 lo contesta, por haber sido el destinatario. Recibida la respuesta del esclavo 2, la línea pasa a estado de reposo, hasta que el maestro determine el inicio de un nuevo ciclo de *polling*.

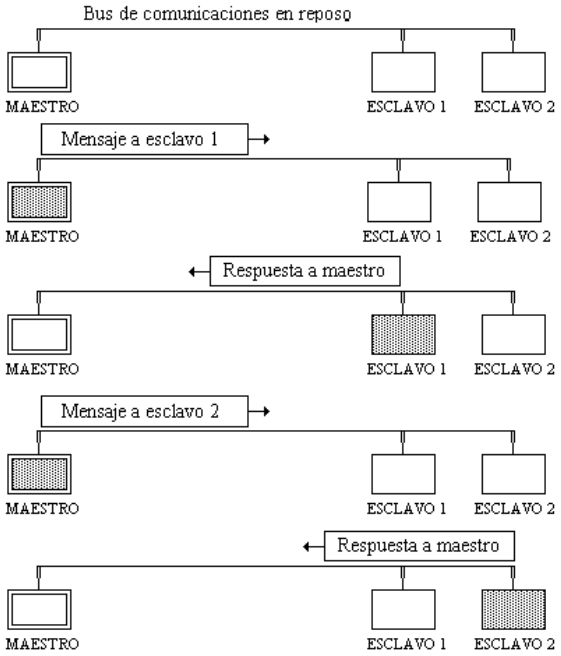


Figura 98

Se puede ver que a *baja carga*, un protocolo del tipo maestro - esclavo resulta en un tiempo de transmisión más lento del que se obtendría de un protocolo en el que cada estación transmitiera el dato por iniciativa propia. En lo anterior, definimos el tiempo de transmisión como el tiempo desde que la estación transmisora tiene nuevo dato válido hasta que la estación receptora lo recibe.

En los hechos, cuando los tiempos involucrados en los procesos son de muchos minutos, se fija el tiempo entre interrogaciones (*polling)* del maestro en varios minutos, de forma de no sobrecargar a los PLCs de la red con las comunicaciones. Este aspecto aumenta la ineficiencia del protocolo.

El PLC puede actuar en doble función de maestro/esclavo, a través de puertas de comunicaciones independientes. Este hecho permite:

- utilizar la interfaz gráfica de un PC para desplegar variables del PLC. Para esto, se conecta un PLC como esclavo de un PC y como maestro de una red de comunicaciones de PLCs e instrumentos y sensores.

- distribuir físicamente la red de PLCs e instrumentación de forma de bajar el costo de cableado cumpliendo la performance datos – tiempo de una aplicación dada.

1 Redes Físicas

Los protocolos de comunicaciones mencionados anteriormente necesitan de una conexión física. Aunque vimos que el protocolo HART utiliza la línea de 4-20 mA del instrumento, en general la gran mayoría de los protocolos maestro esclavo en red utilizan RS 485 de 2 hilos (Figura 99) o RS 422 de 4 hilos.

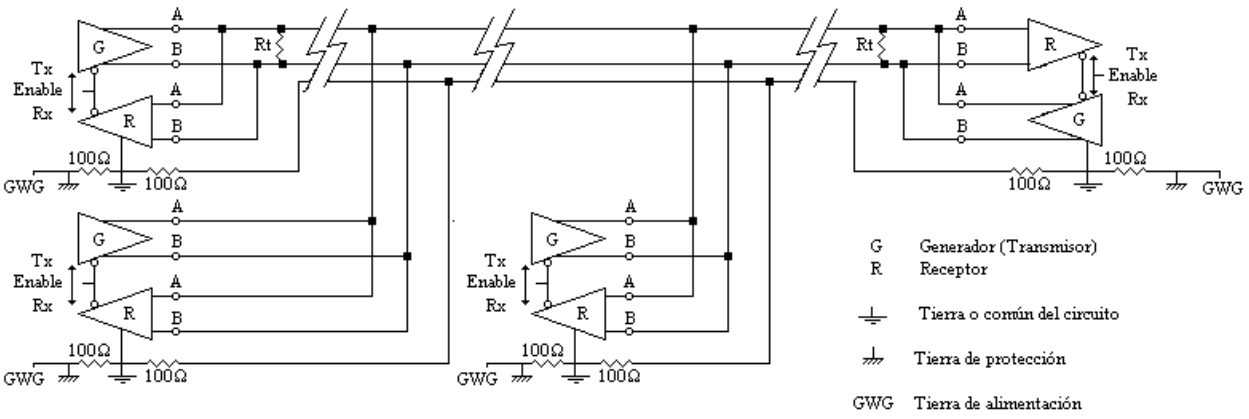


Figura 99

La Figura 99 muestra una aplicación típica de una red multidrop RS-485. Las resistencias Rt representan resistencias de terminación de línea, y se colocan sólo en los dos extremos.

Las comunicaciones en RS485 o RS422 son ejemplos de redes multidrop. Se dice que una red es multidrop o de difusión, cuando todos los dispositivos de la red están conectados entre sí por una sola línea de comunicaciones. En ambos casos, en reposo el transmisor de cada instrumento está en estado flotante, mientras que las recepciones están todas activas. Los niveles de tensión cuando el sistema está en reposo son determinados por el maestro de la red, o eventualmente por resistencias de terminación conectadas una a tierra y la otra a la fuente positiva. Cuando el maestro envía un mensaje a un dispositivo, todos los dispositivos lo reciben, pero sólo el dispositivo referido habilita la transmisión y envía los datos. De esta forma, nunca hay más de un transmisor activo.

La tensión de alimentación más común para una línea RS 485 es 5V. Sin embargo, la tensión de alimentación puede llegar a 12V. En cualquier caso, un voltaje diferencial mayor o igual que 0.2V determinará el 1 o el 0 lógicos, según la polaridad (Figura 100). Esto es, el 1 y el 0 lógicos no dependen de la tensión respecto a la tierra del circuito; dependen de la tensión diferencial entre las líneas (Figura 100).

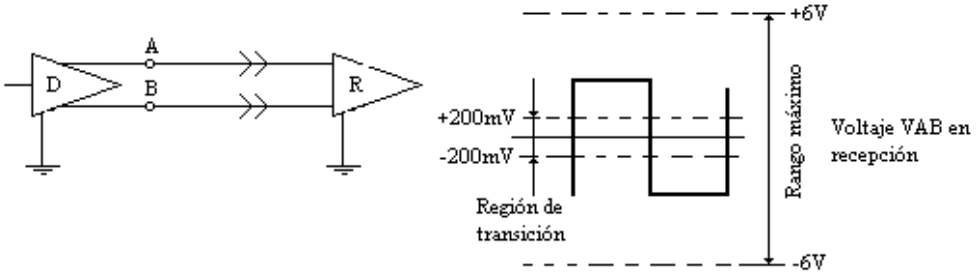


Figura 100

RS 485 y RS 422 presentan gran inmunidad al ruido, debido a la transmisión en modo diferencial y a la utilización de pares trenzados.

Como el modo común de los drivers 485 no puede ser mayor a 12 voltios, se sobrentiende que las tierras digitales de transmisores y receptores deben estar unidas.

Las uniones de ambas tierras traen problemas en líneas largas, donde las tierras de ambos equipos son distintas. La diferencia de potencial entre ambas puede generar fuertes corrientes, que a su vez inducen tensiones en las líneas de comunicaciones. Estas tensiones, denominadas *modo común*, no afectan la tasa de error (siempre y cuando no superen los límites de modo común tolerados por los drivers), ya que no cambian la diferencia de voltaje entre las líneas. En los hechos, cuando se presenta un levantamiento significativo de una línea respecto del potencial de tierra, es de tal magnitud que produce rupturas de las aislaciones.

El número máximo de dispositivos que se puede conectar a una línea 485 bajo Modbus está normalizado a 32.

Las distancia máxima que se puede cubrir directamente con un par trenzado es del orden de 5000 pies (alrededor de 1600m). Esta distancia es lo suficientemente grande como para cubrir el campo de aplicaciones dentro de una planta industrial. Sin embargo, hay instalaciones en Uruguay que duplican esa distancia con tasa de errores despreciable.

Hay que destacar además que en sustitución de los pares de cobre se está utilizando la fibra óptica. La misma permite un aumento enorme del ancho de banda - que redunda en aumento de velocidad -, así como inmunidad al ruido eléctrico y a descargas atmosféricas. Esto último hace que se pueda llevar conjuntamente con líneas de potencia simplificando el cableado.

Desde el punto de vista de confiabilidad, un protocolo maestro esclavo como el que describimos, tiene un inconveniente importante: la falla de un transmisor o un cortocircuito en la línea puede dejar a toda la red fuera de servicio.

Aunque los drivers 485 son de alta confiabilidad, los problemas expuestos en el párrafo anterior hacen que en algunos casos sumamente especiales se prefiera adoptar una configuración en estrella. En esta configuración, de alto costo, el PLC central tiene tantas bocas de comunicaciones como dispositivos hay en la red.

A nivel de redes de comunicaciones, la opción a la red multidrop o de difusión es la comunicación punto a punto. Un ejemplo típico de hardware para comunicaciones punto a punto es el conocido RS-232, cuya distancia máxima es alrededor de 30-40 metros. Hay normas que especifican los niveles de tensión en este protocolo que, a diferencia del 485 si es referido a tierra. Una tensión menor a –3 voltios se interpreta como un 0 lógico y una mayor a +4 como un 1. Hay también especificaciones de este protocolo que permiten alcanzar distancias mayores, utilizando cables blindados y drivers de hardware de mayor potencia.

2 Descripción del protocolo Modbus

Estudiaremos en detalle el protocolo Modbus, por ser uno de los más difundidos.

Se trata de un protocolo maestro esclavo. Según la forma de transmitir los datos, se distinguen dos variantes: una transmite caracteres ASCII, y la otra se basa en un protocolo binario o RTU.

La transmisión de caracteres ASCII utiliza sólo 128 de los 256 valores posibles. El protocolo binario RTU, de uso más generalizado, utiliza 256 valores posibles disponibles. Por lo tanto, el protocolo RTU duplica la velocidad del ASCII.

La utilización de RTU presenta un inconveniente: la mayoría de los modems utilizan caracteres de control en la comunicación, por lo que pueden no funcionar en el modo Modbus RTU, debido a que el protocolo utiliza los 256 valores de byte posibles. Para estos casos se presentan dos opciones:

utilizar un módem diseñado para el protocolo Modbus RTU
configurar al módem en modo transparente. No todos los modems permiten esta configuración.

El maestro del Modbus tiene la iniciativa. Envía un mensaje al esclavo y a continuación espera pasivamente la respuesta. El maestro genera mensajes de reintento de comunicación en los casos que no existe respuesta dentro de un tiempo dado, o que recibe mensaje de respuesta incorrecto (corrompido por ruido).

El mensaje genérico enviado por el maestro es:

Número de esclavo (1 byte)	Código de función (1 byte)	Texto de mensaje (N bytes)	CRC 16 (2 bytes)

En RTU, el número de esclavo puede tomar valores de 1 a 255, por lo que tenemos un máximo de 255 estaciones remotas (RTU).

El segundo byte indica la función sobre el esclavo. La función puede ser una de las siguientes:

leer n bits (función 01)
leer n palabras (función 03)
escribir un bit (función 05)
escribir una palabra (función 06)
escribir n bits (función 0F)
escribir n palabras (función 10)

Aunque existen más funciones, en general las anteriores son las más usadas.

De forma genérica, el campo que hemos denominado texto de mensaje, se compone de:

dirección inicial de lectura o escritura en la memoria del equipo con el que el maestro inicia la comunicación (2 bytes)
número de direcciones de lectura /escritura, a partir de la dirección inicial
en caso de escritura, los datos de escritura

El mensaje termina en un CRC de 2 bytes. El CRC permite asegurar la integridad del mensaje con una probabilidad muy alta. Se calcula por intermedio de un polinomio que incluye todos los bytes trasmitidos.

Una forma alternativa de asegurar la integridad del mensaje, es el método más sencillo del *check-sum*. El *check-sum* se genera a partir de la suma algebraica de todos los bytes comenzando por el inicial. Para este carácter de control se utiliza un solo byte.

La respuesta del esclavo depende del mensaje del maestro.

Veamos un caso concreto: para la lectura de N palabras (función 03), el maestro envía:

SLAVE (1 byte)	03	ADH ADL	N	CRC

N denota el número de palabras a ser leídos a partir de la dirección (ADH ADL). Es claro que ADH denota el byte alto, y ADL el byte bajo de la dirección inicial.

El mensaje recibido por el maestro es bastante similar al enviado, salvo que contiene los datos. Está formado por: el número de esclavo, el código de función solicitada por el maestro, el número de bytes de datos, los datos y el CRC.

En caso que el esclavo reciba un mensaje cuyo CRC verifique, pero de contenido erróneo, responderá con un mensaje de error. El segundo byte del mensaje de error es el OR de 80H con uno de los códigos de error que siguen:

error 01 error en el código de función
error 02 error de dirección
error 03 error de datos.

Es de destacar que la comunicación del PLC como esclavo Modbus es independiente de la existencia de un programa de usuario (depende sólo del sistema operativo). Los únicos requisitos para la lectura o escritura de variables del PLC son

conocer las direcciones a las que se puede acceder (obtenidas de datos del fabricante)

Configurar el número de esclavo en el PLC (en los PLCs del laboratorio, este número se fija en la ventana de configuración del PLC, desde el ambiente de desarrollo).

Es de destacar que las funciones de Modbus son orientadas a palabras o a bits. Por tanto, la lectura o escritura de palabras dobles o flotantes transcurre como lectura o escritura de palabras. La conversión que corresponda quedará a cargo del programa de aplicación que procesa los datos recibidos o transmitidos. En los programas supervisorios de PC, estas conversiones se hacen de forma transparente al usuario. Esto permite el despliegue de los datos en unidades de ingeniería, lo que hace una interfaz de usuario muy amigable.

Otra aplicación interesante del protocolo Modbus es la lectura o escritura de registros de un controlador que forma parte de una red de dispositivos Modbus. Pueden existir hasta

150 comandos Modbus sobre un simple controlador. Enumeramos algunos ejemplos a continuación:

lectura de la variable a controlar
escritura y lectura del set-point
escritura y lectura de la salida analógica del control
escritura y lectura de banda proporcional (inversa del valor de Kp visto en control)
escritura y lectura del tiempo de integración
escritura y lectura de la acción derivativa
escritura y lectura de sí el set-point es remoto o local
lectura de un set-point remoto

Para redes con intervalos de tiempo entre *polling* sucesivos demasiado prolongados, se ha implementado una variante de este protocolo, donde el esclavo toma la iniciativa de la comunicación para informar al maestro de una alarma. Los casos de colisión de mensajes de esclavos se resuelven con tiempos de retransmisión aleatorios como en una red Ethernet

Los drivers de hardware condicionaron al Modbus a una baja velocidad de transmisión (máximo 19200 baudios). Debido a esto, Modicon implementó un protocolo llamado Modbus Plus, a 1 megabit por segundo, con los correspondientes drivers de hardware y software para soportar ésta velocidad. Este protocolo incluye además funciones más avanzadas. que las vistas anteriormente.

Sintetizando, el protocolo Modbus se comporta de forma que el maestro ve las memorias accesibles de los dispositivos de la red como propias.

El sistema SCADA que sirve de ejemplo en el capítulo siguiente (utilizado también en el laboratorio 2), se comunica con los PLC del laboratorio a través del protocolo Modbus.

2.1 Ejemplo de comunicación MODBUS

La Figura 101 muestra un ejemplo de bloque funcional que implementa la comunicación Modbus entre dos PLCs. El bloque se ejecuta en el PLC maestro, y gobierna una comunicación con el PLC esclavo (SLAV).

Las entradas y salidas de este bloque representan una interfaz con el sistema operativo.

Si al ejecutar la función MODBUS (en la etapa de ejecución de programa del ciclo del PLC) FREI = TRUE, el sistema operativo inicia la transmisión de la función FCT al esclavo SLAV. ADDR representa la dirección inicial a la que refiere la función FCT en el esclavo, y NB el número de direcciones referidos a partir de la dirección inicial. DATA representa la dirección inicial a la que refiere el *bloque* MODBUS en el maestro. La entrada TIME representa un timeout de transmisión: si al cabo de TIME milisegundos el esclavo no respondió, el sistema operativo retransmite. Es claro que el tiempo TIME tiene que ser *mayor* que el tiempo de respuesta del esclavo para que este bloque funcione.

En cuanto a las salidas, el sistema operativo pone RDY en 1 terminada la comunicación. El sistema operativo informa de los errores de comunicaciones a través de la bandera ERR. Si ERR = 1, ERN contiene el código del error que aconteció.

Con las entradas al bloque tal como aparecen en la Figura 101, la ejecución del bloque en el PLC maestro, causará que el sistema operativo del maestro lea las 4 direcciones a partir de la 992 (I62.00...03 en los PLCs del laboratorio) del esclavo "SLAVE_1K" (en este caso SLAVE_1K = 1), y escriba los datos leídos a partir de la dirección O62.02 de su memoria. Así, en este caso O62.02 indica al sistema operativo del maestro la dirección de comienzo del buffer de recepción.

La conexión del bit READY a RDY y a FREI causará el inicio de una transmisión al ciclo siguiente de finalizada la transmisión anterior.

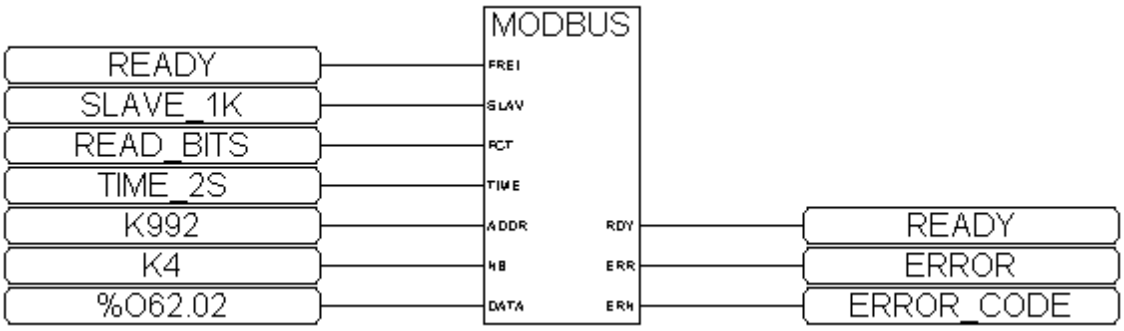


Figura 101

3 MODBUS/TCP

3.1 Breve introducción a las redes TCP/IP

Se explican los conceptos básicos de la arquitectura de capas, así como las funciones de las distintas capas del protocolo TCP/IP sobre la base de un ejemplo.

3.1.1 El planteo del problema

Se considera una red de comunicaciones, esto es, un conjunto de nodos conectados por enlaces de comunicación. Un nodo puede ser un PLC, un PC, etc.

El esquema de comunicaciones que se considera en lo que sigue distingue a los nodos de la red en clientes y servidores.

Un nodo servidor ejecuta un programa servidor. Un programa servidor escucha la red en forma permanente, respondiendo a los mensajes recibidos de acuerdo a un protocolo.

Un nodo cliente ejecuta un programa cliente. Un programa cliente es el encargado de iniciar la comunicación con un servidor de acuerdo a un protocolo.

Un nodo puede ser simultáneamente cliente y servidor.

En lo que sigue se presentan los elementos que intervienen en las comunicaciones entre nodos de la red cuando el protocolo de transporte de la de red es TCP/IP.

3.1.2 Especificación de un protocolo de aplicación

El envío de datos entre cliente y servidor requiere de un conjunto de mensajes tipo "prepárese para recibir los datos xxx", "envíeme los datos xxx", "mensaje conteniendo datos xxx", etc. La especificación de cada uno de estos mensajes y su codificación determina un protocolo de comunicaciones. Se supone en lo que sigue un protocolo basado en 5 mensajes:

WRITE_DATA (01): solicitud de transmisión de datos al servidor (de cliente a servidor)
RCPT_READY (02): listo para recibir de datos (de servidor a cliente)
READ_DATA (03): solicitud de datos (de cliente a servidor)
DATA (04): datos y dirección de comienzo (de servidor a cliente o de cliente a servidor)
END_TRANSFER (05): transferencia terminada (de servidor a cliente, o de cliente a servidor)

Se utiliza una palabra para codificar los mensajes.

Un protocolo cliente - servidor como el anterior se denomina "protocolo de aplicación".

3.1.3 El caso de una LAN Ethernet 10BaseT

Se considera en primer instancia el caso en que los nodos clientes y servidores se conectan a través de una red LAN Ethernet 10BaseT.

Una red Ethernet consta de nodos y *hubs*. Cada nodo se conecta a una *boca* de un *hub* través de cinco cables: un par trenzado "Tx", un par trenzado "Rx" y una referencia. Estos cinco cables terminan en ambos extremos en conectores RJ-45. Un hub tiene varias bocas, que se conectan a nodos o a otros hubs. El resultado es una configuración estrella de conexiones a un hub. El hub se comporta simplemente como un repetidor de difusión: transmite todo dato entrante por cualquiera de las bocas a todas las demás.

Un ejemplo simple de red LAN Ethernet se muestra en la Figura 102. La red de la Figura 102 consiste de dos *hubs* con 8 bocas cada uno, dos PCs y dos PLCs. Las conexiones están representadas por líneas terminadas en puntos negros.

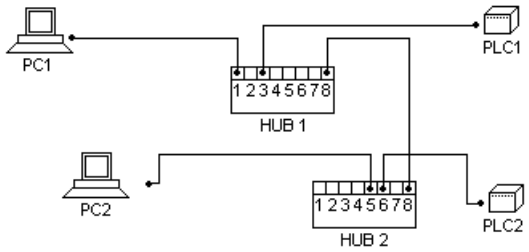


Figura 102

Al igual que el medio físico RS 485, el medio físico Ethernet es un medio de *difusión*: todos los nodos reciben los datos transmitidos a través de la red, aunque sólo el nodo aludido los procesa. En efecto, de acuerdo a las consideraciones anteriores, los datos transmitidos a la red son retransmitidos a todos los nodos de la LAN por cada uno de los hubs. Sin embargo, a diferencia de un medio físico RS 485, en un medio Ethernet cada nodo puede comenzar una transmisión por iniciativa propia, sin necesidad del comando de un maestro. Este hecho determina la existencia de colisiones, es decir, de la transmisión simultánea de nodos. El hardware de los nodos del medio Ethernet permite a los nodos detectar la existencia de una colisión. En caso de colisión, los nodos cuyas transmisiones hacen colisión retransmiten después de un tiempo aleatorio.

El mecanismo de detección de colisión en Ethernet se basa en el tiempo de propagación de los pulsos en el medio Ethernet. Un nodo que transmite un bit a través del cable determina que no existe colisión si tras volver el cable al reposo, permanece en reposo por un tiempo mayor a un tiempo dado T0. Cualquier dato detectado en el cable en un tiempo menor que T0 desde el reposo indica colisión.

El tiempo T0 viene dado por el tiempo máximo que un bit demora atravesar la distancia entre dos nodos cualesquiera de la red. Debido a la velocidad de propagación finita de los bits en un cable (dada por la velocidad de la luz en el cable), T0 impone una cota a la distancia máxima entre transmisor y receptor.

En Ethernet 10BaseT, el número de segmentos no puede superar 5, y cada segmento puede medir hasta 100 metros. Esta cota viene dada por el ancho de banda del cable.

Las comunicaciones de datos a través de un medio Ethernet se guían por el protocolo IEEE 802.3, que establece soluciones a los problemas expuestos en los párrafos anteriores y a muchos otros. El protocolo Ethernet establece la existencia de una identificación única de cada nodo de la red, que referiremos por "dirección LAN".

La resolución de una comunicación entre dos programas a través de una red LAN Ethernet según el protocolo de aplicación establecido en la sección anterior, requiere de la implementación de un programa servidor, de los bloques funcionales asociados al cliente del protocolo de aplicación, y de los bloques funcionales asociados al protocolo Ethernet.

Bloques funcionales asociados al cliente del protocolo de aplicación:

Bloque READ (Figura 103): lee BUFFER_SIZE palabras en el cliente desde la dirección de memoria REMOTE_BUFFER del servidor LAN_ADDR. Los datos se reciben en el buffer que comienza en BUFFER_ADDR.

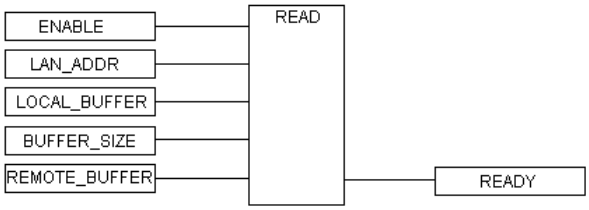


Figura 103

Bloque WRITE (Figura 105): escribe BUFFER_SIZE palabras a partir de la dirección de memoria REMOTE_BUFFER del servidor LAN_ADDR. El buffer de transmisión del cliente comienza en BUFFER_ADDR.

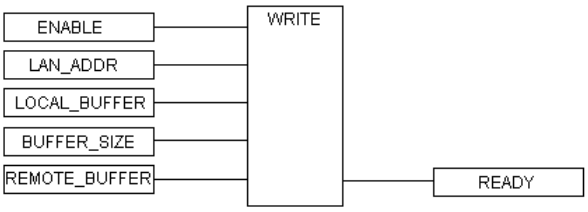


Figura 104

Bloques funcionales asociados al protocolo Ethernet:

La comunicación a través de la LAN Ethernet se basa en los siguientes bloques:

Bloque LAN_RCV (Figura 105): ENABLE habilita la recepción de un bloque de datos desde la dirección LAN_ADDR de la LAN. El buffer de recepción tiene tamaño BUFFER_SIZE y comienza en la dirección BUFFER_ADDR.

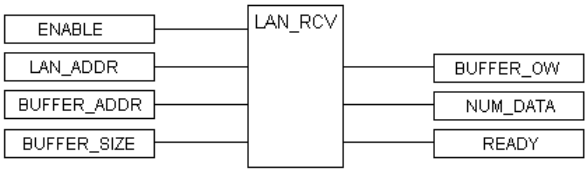


Figura 105

Bloque LAN_TR (Figura 106): ENABLE habilita la transmisión de un bloque datos a la dirección LAN_ADDR de la LAN. El buffer de transmisión tiene BUFFER_SIZE datos y comienza en la dirección BUFFER_ADDR.

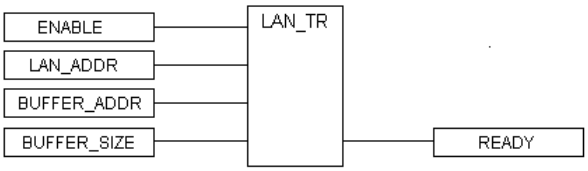


Figura 106

Bloque LAN_LISTEN (Figura 107): utilizado por el programa servidor para escuchar a la red. Si ENABLE = 1, el SO recibe cualquier mensaje que lleva por destino la dirección LAN del servidor al buffer de recepción de tamaño BUFFER_SIZE que comienza en BUFFER_ADDR. LAN_ADDR contiene la dirección LAN del origen del mensaje recibido.

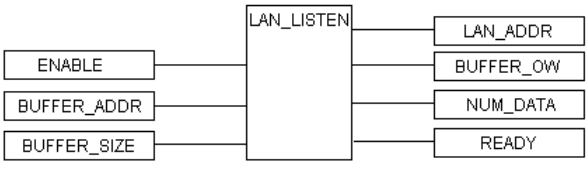


Figura 107

¿Por qué se implementan de forma separada los bloques que resuelven la comunicación Ethernet de los bloques asociados al protocolo de aplicación? La respuesta es clara: la interfaz de Ethernet es independiente del protocolo de la aplicación. Por tanto, ésta separación posibilita la reutilización de los bloques que resuelven la comunicación Ethernet por una aplicación distinta a la aplicación de la sección anterior.

Un esquema de implementación del bloque funcional WRITE se muestra en la Figura 108. Los comentarios se escriben en negrita. Por brevedad, se cometen algunos abusos de sintaxis.

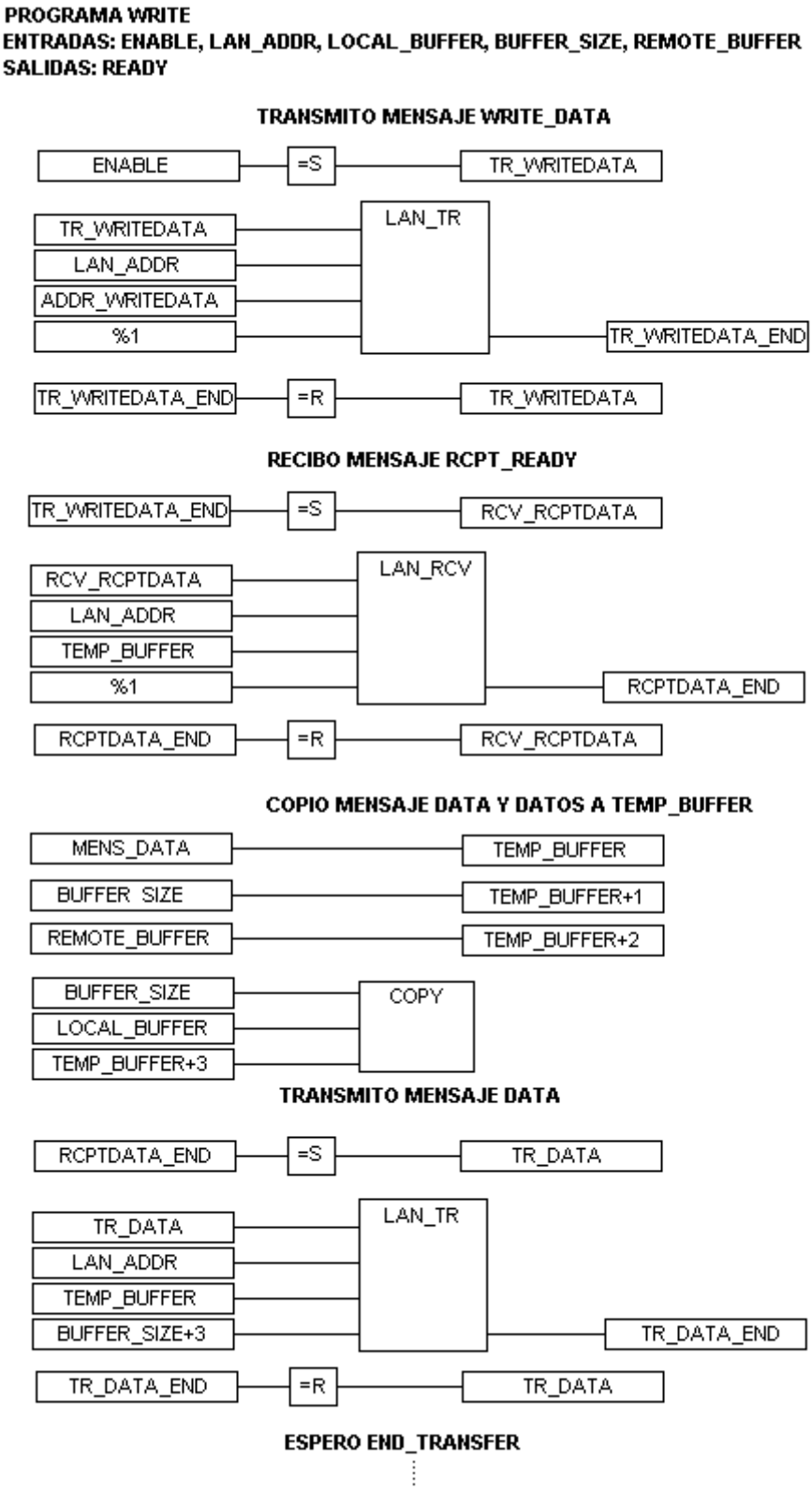


Figura 108

Un esquema de implementación posible del programa servidor se muestra en la Figura 109.

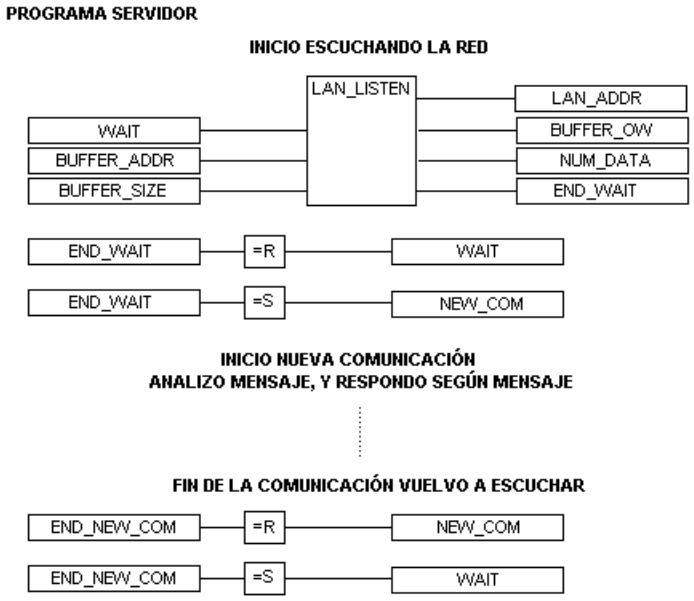


Figura 109

La transmisión de cada mensaje del protocolo de aplicación transcurre según sigue:

En el extremo transmisor, el bloque LAN_TR encapsula el mensaje del protocolo de aplicación en BUFFER_ADDR en una trama de datos Ethernet. El encapsulamiento consiste en agregar encabezado y cola Ethernet al mensaje de aplicación.

En el extremo receptor, el bloque LAN_RCV hace la operación inversa de LAN_TR, devolviendo en BUFFER_ADDR el mensaje de aplicación enviado por el extremo transmisor.

La Figura 110 muestra el proceso de transmisión de un mensaje de aplicación WRITE_DATA de cliente a servidor. El programa servidor recibe en BUFFER_ADDR el mensaje WRITE_DATA transmitido por el programa cliente. La forma en que el mensaje se transmite a través del medio Ethernet resulta totalmente transparente a los programas cliente y servidor.

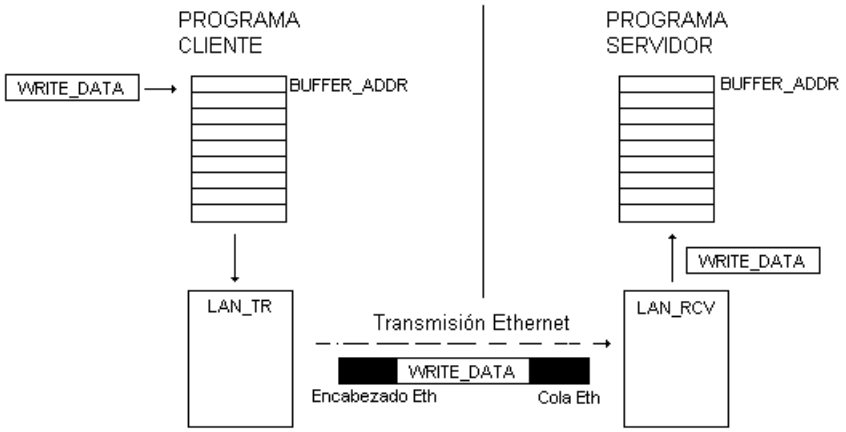


Figura 110

La arquitectura anterior se conoce en la teoría de comunicaciones como "arquitectura de capas". En una comunicación entre dos nodos según una arquitectura de capas, cada nodo se comporta como una pila de N capas. La capa i (i = 1..N) de un nodo se comunica con la capa i de otro nodo en un protocolo común, utilizando los servicios de las capas (i-1) respectivas. La capa i accede los servicios de la capa (i-1) a través de la interfaz entre ambas capas. La forma en que la capa (i-1) hace llegar los datos de la capa i de un extremo al otro es transparente a la capa i. La Figura 111 esquematiza una comunicación en 4 capas.

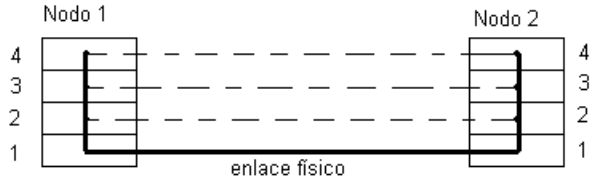


Figura 111

Las comunicaciones expuestas en ésta sección se basan en una arquitectura de tres capas: capa de aplicación, capa Ethernet, y capa física. La capa de aplicación cliente se comunica con la capa de aplicación servidora a través del protocolo de aplicación, utilizando los servicios de la capa Ethernet accedidos a través de los bloques LAN_TR, LAN_RCV y LAN_LISTEN. La capa Ethernet en un extremo se comunica con la capa Ethernet en el otro utilizando los servicios de la capa física, accedidos a través de una interfaz física. La capa física transmite una trama de datos Ethernet de un extremo a otro.

3.1.4 Resolución del problema sobre una red más complicada: el protocolo IP

Resuelto el problema en que la red es una LAN, se analiza una red más complicada. En la nueva topología, mostrada en la Figura 112, el nodo PC2 de la LAN se comunica a través de enlaces serie a los PLCA y PCA.

Se desea transmitir un mensaje desde el nodo cliente PLC1 de la LAN al nodo servidor PLCA. La situación ya no es tan simple, ya que el nodo PLCA no forma parte de la LAN, ni de ninguna LAN, y por tanto no puede referirse por una dirección Ethernet.

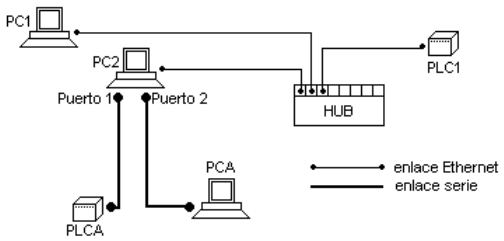


Figura 112

La solución al problema consiste en utilizar un protocolo que permita identificar unívoca y uniformemente los nodos de la red. Esto es, se busca un protocolo que define direcciones de forma independiente de la red. El esquema debe ser independiente de que la comunicación sea a través una LAN, de una red de enlaces serie, o de cualquier otro tipo de enlace. El protocolo que resuelve este problema es el protocolo IP, que asigna a cada nodo una dirección IP. Esta dirección debe ser única a lo largo de toda la red. Por tanto, cada servidor y cada cliente de la red tiene una única dirección IP que lo identifica a los efectos de cualquier comunicación.

Una dirección IP consta de 32 bits, y se denota por "X1.X2.X3.X4", donde los Xi son conjuntos de 8 bits, referidos en lenguaje IP como "octetos" (no se utiliza la nomenclatura byte, ya que un byte no siempre representa 8 bits).

¿Cómo llega un paquete de datos a destino a través de la red IP? Volviendo al ejemplo, el nodo PC1 debe conocer que el camino para llegar al nodo "A" pasa por el nodo PC2. A su vez, el nodo PC2 debe pasar el paquete al nodo "A" a través del puerto serie 1. Para solucionar este problema, el protocolo IP establece que en cada nodo debe existir una tabla que permita al nodo determinar dónde enviar un paquete de datos para llegar a un destino dado. Esta tabla se denomina "tabla de ruteo". Una entrada genérica de la tabla de ruteo se interpreta como sigue: "para llegar a la dirección IP X1.X2.X3.X4, transmitir a nodo Y1.Y2.Y3.Y4".

La resolución de la comunicación a través de la red de la Figura 112 requiere de modificar los programas cliente y servidor respecto del desarrollo inicial, de forma que accedan la interfaz IP y no la interfaz LAN. Esto es, las transmisiones de mensajes de aplicación no se hacen más a direcciones LAN, sino a direcciones IP.

El protocolo IP aparece en una capa intermedia entre la capa de la LAN y la capa de aplicación. A su vez, la capa IP utiliza los servicios de Ethernet y de transmisión por puerto serie para llevar los paquete a destino. De esta forma, los protocolos de transmisión por puerto serie y Ethernet constituyen una capa inferior a la capa IP.

En lo que sigue se detallan las modificaciones requeridas por la red de la Figura 112, respecto de la implementación para una LAN Ethernet:

Modificaciones en los bloques asociados al protocolo de aplicación:

En programas cliente y servidor del protocolo de aplicación se sustituye el acceso a LAN por el acceso a la red IP:

READ y WRITE sustituyen:
LAN_RCV por RCV_IP
LAN_TR por TR_IP
El programa servidor sustituye:
LAN_LISTEN por IP_LISTEN
LAN_RCV por RCV_IP
LAN_TR por TR_IP

Bloques RCV_IP, TR_IP y LISTEN_IP:

Bloque RCV_IP (Figura 113): ENABLE habilita la recepción de un bloque de datos desde la dirección de IP_ADDR. El buffer de recepción tiene tamaño BUFFER_SIZE y comienza en la dirección BUFFER_ADDR.

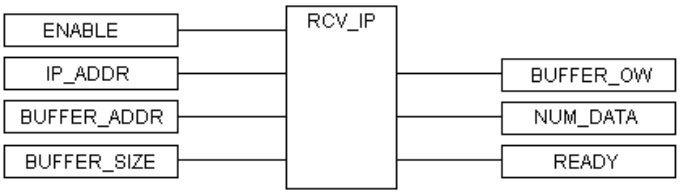


Figura 113

Bloque TR_IP (Figura 114): ENABLE habilita la transmisión de un bloque datos a la dirección IP_ADDR. El buffer de transmisión tiene BUFFER_SIZE datos y comienza en la dirección BUFFER_ADDR.

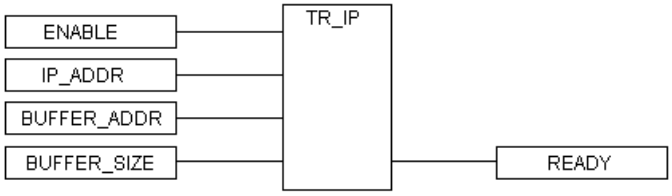


Figura 114

Una implementación posible del bloque TR_IP viene dada según sigue:
Se construye un paquete IP a partir de los datos en BUFFER_ADDR e IP_ADDR = IP_DEST, según la Figura 115:

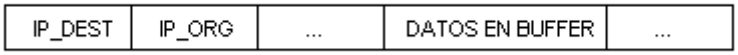


Figura 115

Se encuentra en la tabla de ruteo el nodo vecino por el cual se alcanza a IP_DEST. Se denomina NODO_VECINO al nodo que resulta de la búsqueda.
Si NODO_VECINO se alcanza a través de la LAN, se ejecuta LAN_TR a LAN_ADDR del NODO_VECINO, pasando el paquete IP en el buffer.

Si NODO_VECINO se alcanza a través de un puerto serie, se ejecuta TX_232 al puerto serie que corresponde, pasando el paquete IP en el buffer.

Bloque LISTEN_IP (Figura 116): utilizado por el programa servidor para escuchar a la red. Si ENABLE = 1, el SO recibe cualquier mensaje que lleva por destino la dirección IP del servidor al buffer de recepción de tamaño BUFFER_SIZE que comienza en BUFFER_ADDR. IP_ADDR contiene la dirección IP de origen del mensaje recibido.

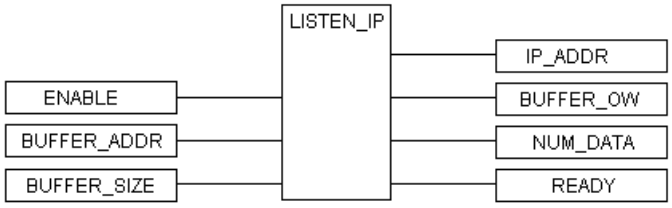


Figura 116

Bloques funcionales asociados a la transmisión de una trama de datos por el puerto serie:

Bloque RX_232 (Figura 117): ENABLE habilita la recepción de un bloque de datos por el puerto serie SER_PORT. El buffer de recepción tiene tamaño BUFFER_SIZE y comienza en la dirección BUFFER_ADDR.

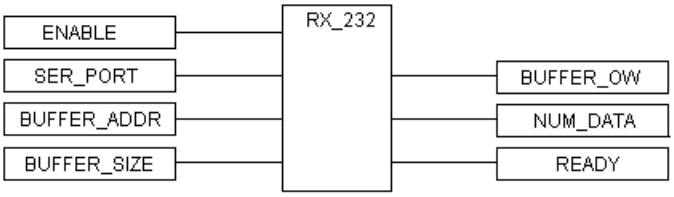


Figura 117

Bloque TX_232 (Figura 118): ENABLE habilita la transmisión de un bloque datos por el puerto serie SER_PORT. El buffer de transmisión tiene BUFFER_SIZE datos y comienza en la dirección BUFFER_ADDR.

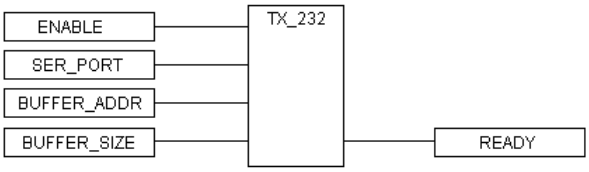


Figura 118

Modificaciones al procesamiento de entradas de comunicaciones del SO:

El mecanismo de ruteo de IP recae necesariamente sobre los sistemas operativos de los dispositivos conectados a la red. El sistema operativo de cada dispositivo conectado a la red ejecuta de forma permanente funciones tipo LISTEN a cada uno de los puertos de comunicaciones conectados a red (Ethernet o Serie).

La Figura 119 muestra una implementación de la actualización de entradas de la red IP de un PLC con puertos serie y puerto Ethernet conectados a la red IP.

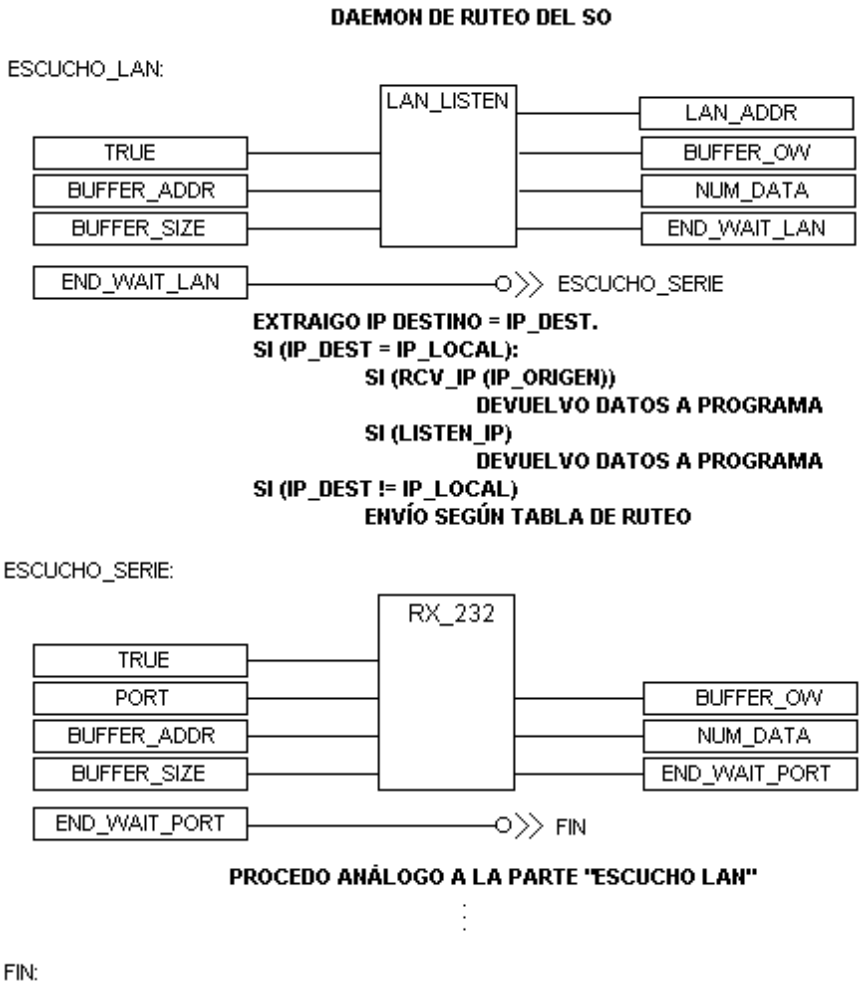


Figura 119

La Figura 120 representa la comunicación entre los nodos PLC1 y PLCA de la red IP de la Figura 112. Las cuatro capas que intervienen se denominan "capa de aplicación", "capa IP" o "capa de red", "capa de enlace de datos" y "capa física". Se observa una diferencia entre la capa de aplicación y las demás: la capa de aplicación se comunica punta a punta, mientras las demás se comunican con el "nodo vecino".

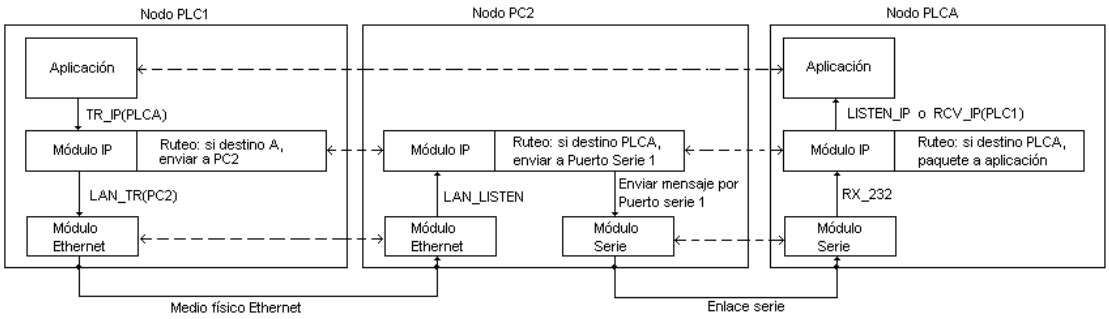


Figura 120

3.1.5 Problemas con el tamaño de los mensajes: fragmentación

Los mensajes tipo DATA tienen tamaño variable. El hecho de que cada capa tiene un tamaño máximo de mensaje transmitido posible determina la necesidad de fragmentación de mensajes. La limitación en el tamaño se debe por ejemplo a límites en los tamaños de buffer de transmisión y recepción, o a restricciones impuestas por los encabezados (el largo máximo de un paquete IP, dado por el campo del encabezado que define el largo, es 64 KB).

La fragmentación de una capa se hace de forma transparente a la capa de arriba. En el extremo transmisor la capa transmisora fragmenta los mensajes y en el extremo receptor la capa correspondiente los reconstruye.

Se hace notar que la implementación de la Figura 119 requiere de modificaciones para contemplar la fragmentación de mensajes: el daemon de ruteo del SO no puede pasar un mensaje de aplicación a la capa de aplicación en tanto no haya recibido todos los fragmentos (la información de fragmentación se encuentra en el encabezado del paquete IP). Este hecho requiere de la existencia de un buffer intermedio en memoria en el que se almacenan los fragmentos de un mensaje de aplicación en tanto el mismo no se haya completado.

3.1.6 Mensajes perdidos en la red: el protocolo TCP

Las medidas tomadas frente a pérdidas de mensajes dependen de la capa. La capa de enlace de datos es en general "orientada a conexión": existe una conexión entre el extremo transmisor y el extremo receptor que regula la transmisión de datos. En general el módulo de capa de enlace en el nodo receptor solicita el re envío de mensajes al módulo correspondiente en el nodo transmisor en caso de error a la recepción. Sin embargo, éste no es el caso de la capa IP, que se comporta como el correo postal ofreciendo un servicio de mejor "esfuerzo". Un módulo IP de un nodo no necesariamente recibe notificación de parte del nodo vecino sobre la recepción o no de un paquete de datos.

El hecho de que el protocolo IP es no confiable hace posible un escenario según sigue. Se supone una red según la Figura 121, donde el nodo B está en estado de congestión, esto es, su buffer de procesamiento de paquetes está saturado. El escenario se ilustra en la Figura 122. Para transferir un conjunto de datos al nodo C, el nodo PC1 envía tres mensajes denominados M1, M2 y M3.

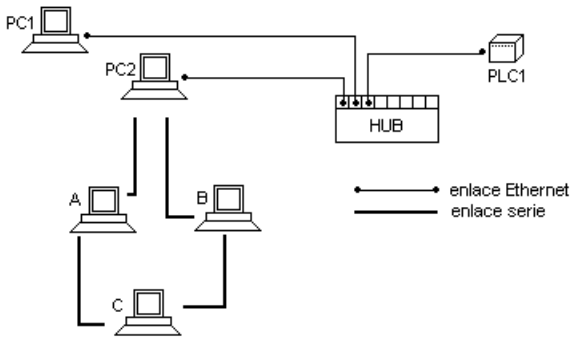


Figura 121

El módulo IP en PC1 agrega encabezado y cola a M1 y M3, y los envía a C. Por otro lado, fragmenta el M2 en dos paquetes M21 y M22, y ejecuta "TR_LAN" al nodo PC2 con cada uno de los paquetes.

Las tablas de ruteo del nodo PC2 indican que puede llegar a C a través de A o de B. Se supone que PC2 transmite el paquete M21 a través de A y el paquete M22 a través de B. El paquete M21 llega a C a través de A. Sin embargo, de acuerdo a lo que establece el protocolo IP en caso de congestión, el paquete M22 es descartado por el nodo B.

La capa IP del nodo C recibe el paquete M21 a través de A, pero no el paquete 22, descartado por B. La capa IP en C es incapaz de reconstruir el mensaje M2, por lo que al cabo de un tiempo descarta el paquete M21. El resultado es que la aplicación servidora del nodo C recibe sólo los mensajes 1 y 3.

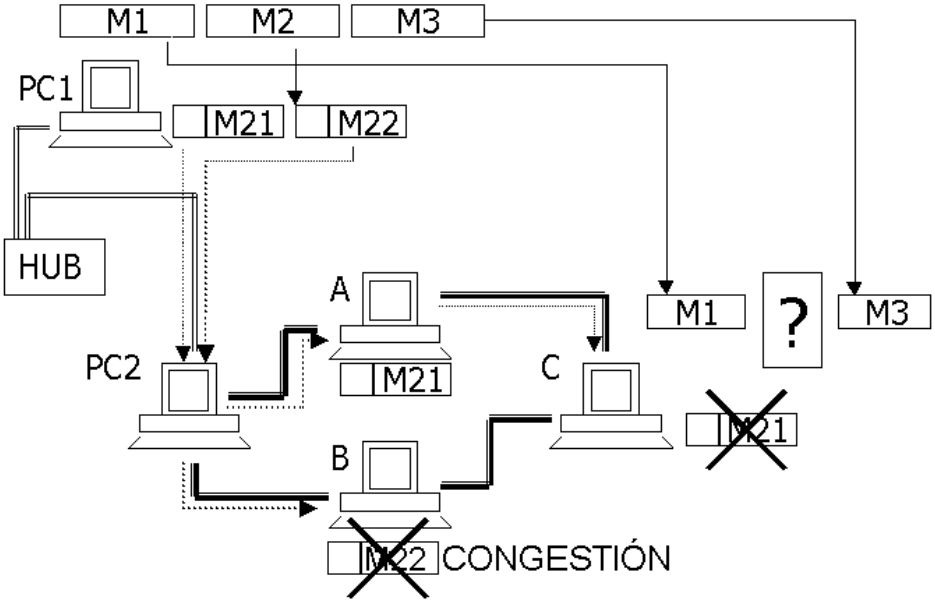


Figura 122

Esta situación errónea se resuelve cambiando el protocolo de la aplicación, incorporando un esquema de numeración y un esquema de reconocimiento basado en mensajes tipo "recibí mensaje número N". La recepción en un extremo de un mensaje "recibí mensaje número N" se interpreta como que el otro extremo recibió todos los mensajes desde el último mensaje reconocido hasta el mensaje número N. Si un extremo envía su mensaje N y transcurre más de un tiempo dado sin reconocimiento del mensaje N, el extremo re envía el mensaje.

Sin embargo, el esquema de reconocimiento es independiente del protocolo de aplicación. De acuerdo a los criterios expuestos en “El caso de una LAN Ethernet 10BaseT”, se separan las nuevas funciones de reconocimiento extremo a extremo de la aplicación y se incorporan a un módulo con una interfaz a la aplicación del tipo "envío mensaje a destino IP". Estas funciones forman parte del protocolo TCP. Se tiene por tanto una nueva capa, situada debajo de la capa de aplicación y encima de la capa IP, que se comunica extremo a extremo.

3.1.7 Distinción de instancias de una aplicación cliente en un nodo: puerto de origen TCP

¿Qué sucede si dos aplicaciones iguales en un mismo nodo cliente identificado con una dirección IP, envían simultáneamente un mensaje al mismo servidor? ¿Cómo distingue la aplicación servidora los mensajes de una aplicación de los de la otra?

La solución a este problema requiere de un canal virtual de comunicaciones para cada aplicación. Cada canal se identifica con un número único. Antes de iniciar la aplicación cliente una transferencia de datos al servidor, ejecuta la función de la capa TCP "iniciar canal de comunicaciones a IP destino". Esta función desencadena el intercambio de una serie de mensajes entre la capa TCP del nodo de origen y la capa TCP del nodo de destino, que permiten a la capa TCP de origen reserva un número especial para la comunicación entre las aplicaciones, denominado "puerto". Los mensajes TCP correspondientes a esta transacción son:

mensaje de solicitud de reserva recursos para una comunicación nueva (cliente a servidor)
mensaje de canal de comunicaciones establecido con puerto N (servidor a cliente)

El reconocimiento de la capa de transporte de destino lleva al establecimiento del canal. Establecido el canal, todo mensaje de la capa de transporte de origen asociado a la comunicación incluye el número de puerto registrado.

3.1.8 Distinción entre aplicaciones servidoras en un mismo servidor: puerto de destino TCP

El mismo problema planteado en "Distinción de instancias de una aplicación cliente en un nodo: puerto de origen TCP" para un cliente existe del lado del servidor. ¿Cómo distingue un cliente una aplicación servidora de otra aplicación servidora en un mismo nodo? De forma análoga a las aplicaciones clientes, la distinción de dos aplicaciones servidoras requiere de la asociación a cada aplicación servidora a un número único. Cuando una aplicación servidora comienza a escuchar la red, pide a la capa TCP que le asigne un número, denominado también "puerto". En definitiva, toda aplicación cliente que se comunica a una aplicación servidora dada lo hace al IP de destino y al puerto de la aplicación servidora.

Por tanto, el canal de comunicaciones queda determinado por cuatro números: el puerto y la dirección IP de origen de la comunicación, y el puerto y la dirección IP de destino.

3.1.9 Acceso al puerto de la aplicación servidora

Para que una aplicación cliente pueda comunicarse con una aplicación servidora el puerto asignado a la aplicación servidora tiene que estar convenido de antemano. De esta forma, el puerto en que escucha la aplicación servidora pasa a ser parte del protocolo. En este sentido, la especificación del protocolo de aplicación ejemplo en "Especificación de un protocolo de aplicación" se completa especificando por ejemplo al puerto 603 como el puerto en que la aplicación escucha. De esta forma el cliente inicia una comunicación al puerto 603 para acceder al servicio del ejemplo.

Por tanto, el puerto de origen puede ser cualquier número, el de destino no.

3.2 Introducción a MODBUS/TCP

TCP/IP es el protocolo de transporte más extendido en Internet. Se basa en una arquitectura de capas, y provee de un transporte confiable de datos entre máquinas. Por otro lado, Ethernet se ha transformado en el estándar de facto de redes empresariales, y ha madurado al punto que hoy día el costo de implementar una red Ethernet se ha equiparado al costo de los buses de campo.

Ya que las redes empresariales utilizan TCP/IP sobre Ethernet, la utilización a nivel de planta industrial de TCP/IP sobre Ethernet permite la integración de la red de la corporación (Intranet) con la red de la planta.

Las razones expuestas en los párrafos anteriores explican que la tendencia hoy día sea que TCP/IP sobre Ethernet se convierta en estándar de redes industriales.

En marzo de 1999 Schneider Electric (MODICON) publicó la versión 1.0 de la especificación abierta MODBUS/TCP, la adaptación de MODBUS a las redes TCP/IP. Este protocolo combina la red física Ethernet, el estándar de redes TCP/IP, y la enormemente difundida representación de datos MODBUS. En la especificación se recalca lo simple que resulta implementar este protocolo en un dispositivo con zócalos TCP/IP.

3.3 Diferencias entre MODBUS/TCP y MODBUS

El MODBUS es un protocolo maestro-esclavo. El dispositivo configurado como maestro interroga al dispositivo configurado como esclavo.

MODBUS/TCP es un protocolo cliente-servidor; el cliente interroga al servidor. Lo nuevo es que, con el software necesario, un dispositivo puede ser *simultáneamente* cliente en una o más conexiones MODBUS/TCP y servidor en una o más conexiones MODBUS/TCP.

La comunicación MODBUS/TCP entre cliente y servidor comienza con el establecimiento por parte del cliente de una conexión TCP al puerto 502 del servidor. Establecida la conexión, la comunicación consiste de mensajes de pedido de parte del cliente, y respuestas del servidor. El cliente y el servidor intercambian tramas MODBUS/TCP con el formato general de la Figura 123 ([1]).

Figura 123

La trama MODBUS/TCP tiene ciertas diferencias con la trama MODBUS RTU:

[1] El código de función junto con el campo de datos (si existe) forman un PDU (Protocol Data Unit), independiente del protocolo de transporte (en este caso TCP). De esta forma, en el caso general, se podría transmitir el PDU sobre cualquier protocolo de transporte, agregando un encabezado y una cola (para chequeo de errores) apropiados.

La dirección de esclavo MODBUS se reemplaza por un byte "Identificador de la unidad" ("Unit Identifier") en el encabezado MODBUS/TCP. El "Identificador de la unidad" se utiliza en aplicaciones con puentes, routers y gateways, que comunican dispositivos MODBUS/TCP con dispositivos MODBUS bajo una *única* dirección IP.

El encabezado MODBUS/TCP incluye información de largo del mensaje, que permite al receptor determinar los límites del mensaje, eventualmente dividido en varios paquetes durante la transmisión. La existencia de reglas que permiten determinar el largo de forma explícita o implícita, junto con el CRC32 que efectúa Ethernet, minimizan la probabilidad de no detectar una trama corrompida.

No incluye CRC, por las razones expuestas en el punto anterior.

La estructura de la porción conformada por el código de función y los datos es idéntica a la de la trama MODBUS.

La especificación abierta de MODBUS/TCP define las distintas funciones. La especificación de cada función incluye la especificación del mensaje de pedido (request) del cliente y del mensaje de respuesta (response) del servidor. De forma análoga a MODBUS, los mensajes se diseñan de forma que el recipiente pueda determinar el final del mensaje: si la función es de largo fijo, el largo se determina por el código de función, mientras que si es de largo variable, se incluye una cuenta de bytes (byte count) en el campo de datos del mensaje. Por ejemplo, el mensaje "Read multiple registers" (FC = 3) tiene la siguiente forma:

Request
Byte 0: FC = 03
Byte 1-2: Reference number
Byte 3-4: Word count (1-125)

Response
Byte 0: FC = 03
Byte 1: Byte count of response (B=2 x word count)
Byte 2-(B+1): Register values

3.4 Descripción del encabezado MODBUS/TCP

El encabezado tiene 7 bytes de largo, e incluye 4 campos:

Identificador de transacción: se utiliza para identificar una transacción. El servidor MODBUS/TCP copia el identificador de transacción del pedido en la respuesta.
Identificador de protocolo: Utilizado para multiplexado a nivel de aplicación ([1]). El protocolo MODBUS se identifica por el valor 0.
Largo: Es una cuenta de bytes de los campos que siguen (Identificador de unidad y campos de datos).
Identificador de unidad: Este campo se utiliza en el ruteo interno de un sistema. La aplicación típica es la de una subred de esclavos MODBUS o MODBUS+, comunicados

entre sí por una línea serial, y a Ethernet TCP/IP a través de un gateway. El campo lo fija el cliente en el mensaje de pedido. El servidor lo debe copiar a la respuesta.

La siguiente tabla describe cada uno de los campos:

Campo	Largo	Descripción	Cliente	Servidor
Identificador de transacción	2B	Identificación de la transacción de Pedido/Respuesta MODBUS	Iniciado por el cliente	El servidor lo copia en la respuesta
Identificador de protocolo	2B	0 = Protocolo MODBUS	Iniciado por el cliente	El servidor lo copia en la respuesta
Largo	2B	Número de bytes que siguen	Iniciado por el cliente (pedido)	Iniciado por el servidor(respuesta)
Identificador de unidad	1B	Identificador de esclavo remoto, comunicado a través de línea serie u otro bus	Iniciado por el cliente	El servidor lo copia en la respuesta

3.5 Tipos de datos a los que se orienta el protocolo MODBUS/TCP

Al igual que MODBUS, el MODBUS/TCP se orienta bits y palabras (16 bits). De todas formas, cualquier dato que se pueda convertir a un arreglo de palabras de 16 bits puede transportarse por MODBUS/TCP.

3.6 Ejemplo de red MODBUS / TCP

En la Figura 124 se muestra un ejemplo de red MODBUS/TCP. A continuación se describen las funciones de los distintos dispositivos.

Los dispositivos A, B y D son dispositivos MODBUS/TCP, con función tanto de servidor como cliente. Por ejemplo, el dispositivo A puede actuar como cliente para bajar datos del dispositivo F, y a su vez puede actuar como servidor para recibir alarmas del dispositivo D o B. Esto es una clara ventaja sobre los dispositivos tradicionales MODBUS/RTU, donde los dispositivos esclavos deben esperar pacientemente el interrogatorio del maestro, aún en estado de alarma. Cada uno de los dispositivos A, B y D tiene una dirección IP que los identifica en la red Ethernet TCP/IP.

Los dispositivos E y F son esclavos MODBUS/RTU tradicionales. Se comunican a través de un bus RS-485 con un gateway MODBUS/TCP-MODBUS. Este gateway se encarga de que los dispositivos E y F aparezcan en la red MODBUS/TCP como servidores MODBUS/TCP. La limitación, inherente a MODBUS/RTU, es que los dispositivos E y F sólo pueden cumplir la función de servidores MODBUS/TCP, y no de clientes. Desde el punto de vista de la red MODBUS/TCP, la subred de los dispositivos E y F se ve bajo un único IP, que es el IP del gateway. Dentro de esta subred, cada de los dispositivos se identifica por el campo de identificador de unidad.

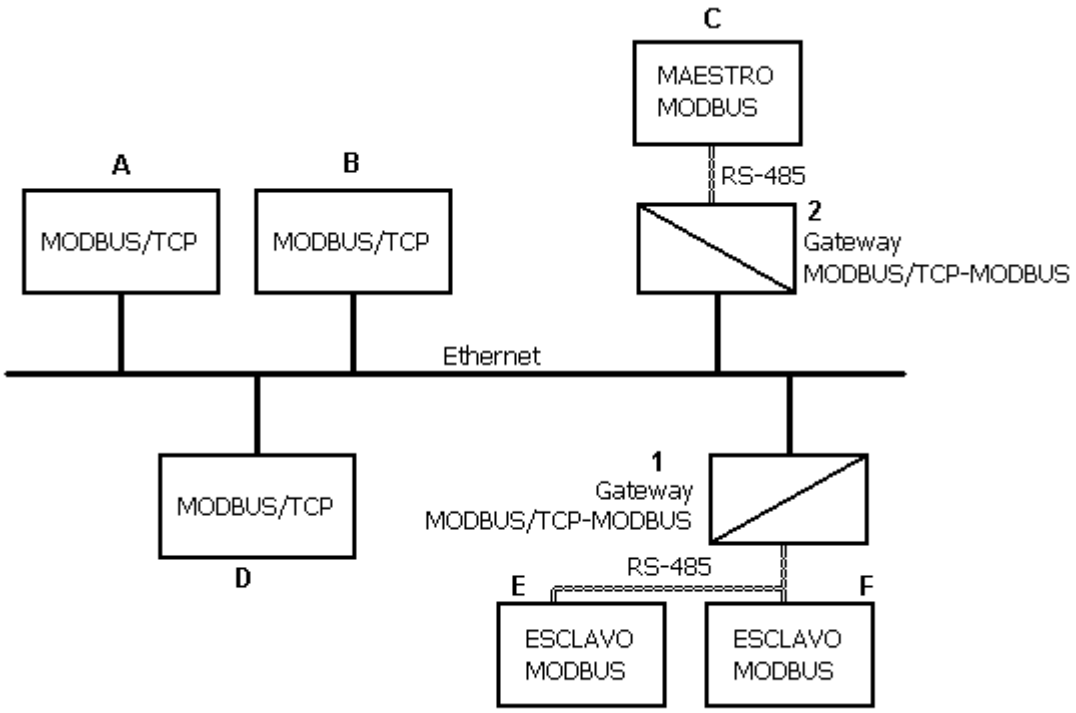


Figura 124

Por último, el dispositivo C es un maestro MODBUS/RTU tradicional, comunicado al Gateway 2 MODBUS/TCP-MODBUS a través de RS-485. Una vez más, el Gateway 2 se encarga de que el dispositivo C aparezca en la red MODBUS/RTU como un cliente MODBUS/TCP. El protocolo MODBUS/RTU obliga a que el dispositivo C tenga sólo función cliente MODBUS/TCP, y no servidor. Desde el punto de vista de la red MODBUS/TCP, la subred del dispositivo C se ve bajo un único IP, que es el IP del Gateway 2. Al dispositivo C le corresponderá un identificador de unidad dado.

CAPÍTULO 11: STRUCTURED TEXT

El lenguaje Structured Text (ST) es el quinto definido por la norma IEC 1131-3. Se trata de un lenguaje de texto de alto nivel que toma elementos de los lenguajes C y Pascal, con sintaxis muy similar a la de Pascal.

Comenzando la descripción del último lenguaje de la norma IEC 1131, se hace notar que no es fácil elegir el lenguaje adecuado para escribir un programa. Una elección posible es escribir el programa en FBD, que resulta más claro, utilizando bloques funcionales programados en ST.

ST es un lenguaje pequeño, con sólo 40 palabras claves y 10 tipos de sentencias. Muchos fabricantes de PLC proveían de lenguajes de texto alto nivel previo al advenimiento del estándar IEC 1131-3. Estos lenguajes influenciaron el diseño de ST. Sin embargo, ningún de lenguaje de este tipo ha resultado tan difundido como el LD. Como resultado existen menos implementaciones de ST, y generalmente casi todas están basadas en el estándar IEC 1131-3.

Un programa en ST consiste en un conjunto de sentencias. Cada sentencia contiene expresiones válidas ST.

1 Operadores de expresiones

Una expresión resulta en un único valor, y se compone de operadores y operandos. Los operandos pueden ser valores literales, variables o invocaciones a funciones.

La tabla que sigue muestra los operadores de ST en orden de precedencia descendente:

Operación	**Símbolo**
Paréntesis	()
Invocación a función	Identificador(lista de argumentos)
Exponente	**
Negación	-
Complemento	NOT
Multiplicación	*
División	/
Módulo	MOD
Suma	+
Resta	-
Comparación	<, >, <=, >=
Igualdad	=
Desigualdad	<>
AND booleano	&
AND booleano	AND
XOR booleano	XOR
OR booleano	OR

1.1 El operador cast

El lenguaje ST define el operador *cast*, ya definido en C y Pascal. El *cast* convierte una variable de un tipo de dato a otro, de acuerdo a reglas predefinidas. Por ejemplo, en caso de efectuarse una multiplicación de enteros donde se sabe a priori que el resultado puede ser un número mayor que máximo entero representable, se hace necesario hacer un pasaje de ambas variable a Real, obteniéndose un resultado tipo Real, no restringido al rango de –32767 a + 32767 de un entero.

1.2 Los operadores booleanos

Si se considera Var1, Var2 y Var3 como boolenas, la sentencia

```
Var1 := Var2 AND Var3
```

asigna (Var2 AND Var3) a Var1, y se ejecuta de forma análoga al bloque AND del lenguaje FBD. En ST resulta más fácil formar expresiones booleanas complejas que en FBD:

```
Var1 := (Var1 AND Var2) OR Var 3;
```

Lo anterior llevaría más de un bloque en FBD. Se hace notar la precedencia de los operadores debida a los paréntesis: primero se ejecuta (Var1 AND Var2), y luego el OR con Var3. Este tipo de precedencia es análoga a los lenguajes de alto nivel ya mencionados.

2 Sentencias

Se definen los siguientes tipos de sentencias: comentario, asignación, invocación a bloque funcional, retorno, sentencias de selección y sentencias de iteración.

2.1 Comentario

Los comentarios se sitúan entre los símbolos ‘(*’ y ‘*)’. Por ejemplo:

```
(* Este es un comentario *)
```

2.2 Asignación

La asignación permite la transferencia del valor de una expresión válida a una variable. El operador ‘:=’ se utiliza como operador de asignación. Por ejemplo:

```
A := B;          (*asigna el valor de la variable B
                   a la variable A*)

C := C+1;        (*incrementa en 1 la variable C*)
```

```
     VAR :=  7;          (*asigna el número 7 a la variable
VAR*)

     VAR := VAR * 10;    (*asigna VAR*10 a VAR*)

     Y := COS(X) + 12; (*X e Y son tipo Real*)
```

Se hace notar que el operador de asignación es idéntico al operador asignación de Pascal.

2.3 Invocación a bloque funcional

Como se menciona en secciones anteriores, el lenguaje ST se especifica como parte del estándar IEC 1131. De acuerdo a este estándar, la invocación de un bloque funcional requiere de la definición de una instancia del bloque funcional. La instancia se define en alguna de las definiciones de variables de los distintos niveles jerárquicos del programa. Esta organización de programa se describe en detalle en el capítulo que sigue.

Una vez declarada la instancia del bloque funcional, el bloque funcional se invoca a través del nombre de la instancia bloque funcional, seguido de una lista de asignaciones de entradas entre paréntesis. No se requiere de un orden particular en las asignaciones de entradas, y no se requiere la lista completa de asignaciones. Si no se asigna una entrada en la invocación de un bloque funcional, se utiliza el valor previo de la entrada, o el valor inicial en caso que sea la primer invocación al bloque.

Cada salida de un bloque funcional se refiere por una dirección en formato nombre_bloque_funcional.nombre_salida.

Como ejemplo, las siguientes invocaciones son válidas para una instancia denominada “FUN” de un bloque funcional con entradas IN1, IN2, e IN3, y salidas OUT1, OUT2 y OUT3:

```
FUN(IN1:= 1, IN2:=B, IN3:= %IW2.3);

FOO := FUN.OUT1;

BAR := FUN.OUT2;

FUN(IN2:=FOO, IN1:= BAR);

BAZ := FUN.OUT1 * 5;
```

2.4 Retorno

La sentencia RETURN se puede utilizar para terminar una función o programa escrito en ST. En el siguiente ejemplo, la función termina si A>B:

```
IF A > B THEN
     RETURN;
END_IF
```

La ejecución de la instrucción RETURN causa la terminación de la ejecución del programa correspondiente en el ciclo dado.

2.5 Sentencias de selección

ST incluye dos tipos de sentencias de selección: la sentencia IF..THEN..ELSE..END_IF y la sentencia CASE..OF..ELSE..END_CASE.

2.5.1 La instrucción IF

La instrucción IF ejecuta instrucciones de acuerdo al valor de una expresión booleana. Una expresión booleana es aquella que tiene como resultado TRUE o FALSE, es decir un '1' o un '0'.

Sintaxis:

```
IF <Expresión booleana 1> THEN
     <Instrucciones IF>
{ ELSIF <Expresión booleana 2> THEN
     <Instrucciones ELSE_IF 1>
     ....................................
  ELSIF <Expresión booleana n> THEN
     <Instrucciones ELSE_IF n-1>
  ELSE
     <Instrucciones ELSE> }
END_IF;
```

La sección entre llaves {} es opcional.

Si la Expresión booleana 1 retorna TRUE sólo se ejecutan las <Instrucciones IF>, continuando la ejecución del programa después del END_IF.

La construcción de secuencias de control de flujo como ésta en un lenguaje distinto de ST se hace bastante complicada.

En el ejemplo que sigue, se asigna el valor 7 a la variable B siempre y cuando A = 3:

```
IF A = 3 THEN
     B := 7;
END_IF
```

2.5.2 La instrucción CASE

La instrucción CASE permite la ejecución condicionada al valor de una variable de acuerdo a la sintaxis que sigue:

```
CASE  <Var1> OF

     <Valor1>:                    <Instrucción 1>
     <Valor2>:                    <Instrucción 2>
     <Valor3, Valor4, Valor5>:    <Instrucción 3>
     <Valor6 .. Valor10>:         <Instrucción 4>
          ...
     <Valor n>:                   <Instrucción n>

ELSE <Instrucción ELSE>
END_CASE;
```

Si la variable en <Var1> tiene el valor <Valor i>, se ejecuta la instrucción <Instrucción i>, i=1..n. Si la variable en <Var1> no tiene ninguno de los valores indicados, se ejecuta la <Instrucción ELSE>.

Si se ejecuta una misma instrucción para un conjunto de valores de la variable, se escriben los valores separados por comas.

Si se ejecuta una misma instrucción en un rango de valores de la variable, se escribe el valor inicial y el valor final separados por dos puntos uno después del otro.

Ejemplo:

```
CASE INT1 OF
     1:   BOOL1 := TRUE;
          BOOL3 := FALSE;
     2:   BOOL2 := FALSE;
          BOOL3 := TRUE;
     10:  BOOL4 := TRUE;
          BOOL5 := TRUE;
ELSE
     BOOL6 := NOT BOOL1;
     BOOL7 := BOOL1 OR BOOL2;
END_CASE
```

Si la variable INT1 no coincide con 1, 2 o 10 se ejecuta la instrucción ELSE. Es posible sustituir una sentencia CASE por una tipo IF y ELSE_IF, a costa de una pérdida de claridad. La sentencia ‘CASE’ es similar al ‘switch’ del lenguaje ‘C’.

La utilización de una instrucción ‘CASE’ resulta útil en el caso de una máquina secuencial.

2.6 Sentencias de iteración

ST incluye tres tipos de sentencias de iteración: la sentencia FOR..DO, la sentencia WHILE..DO, y la sentencia REPEAT..UNTIL.

La sentencia EXIT se utiliza para finalizar la ejecución de un bucle dado que se verifica una condición.

2.6.1 El bucle FOR

El bucle FOR se utiliza para programar procesos que se repiten.

Sintaxis:

```
INT_Var: INT;

FOR <INT_Var>:=<VALOR_INICIAL> TO <VALOR_FINAL> {BY
<Paso>}
DO
     <Instrucciones>
END_FOR;
```

La parte entre llaves {} es opcional.

Las <Instrucciones> se ejecutan mientras que el contador <INT_Var> no sea mayor que <VALOR_INICIAL>. Esta condición se verifica *antes* de la ejecución de las <Instrucciones>, por lo que las <Instrucciones> no se ejecutan si <VALOR_INICIAL> es mayor que <VALOR_FINAL>.

Cada vez que se ejecutan las <Instrucciones>, se incrementa el valor de la variable Int_Var en el valor de "Paso". El valor de defecto de <Paso> es 1.

Ejemplo:

```
FOR Contador:=1 TO 5 BY 1 DO
     Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

En caso que el valor inicial de Var1 sea 1, al final del bucle su valor será 32.

NOTA: se debe prever que el valor final esté en el rango del tipo de Contador. Por ejemplo, si un Contador tipo Int se quiere incrementar hasta el valor 60000, se obtiene un bucle infinito, ya que Contador tiene restringido su valor en el rango -32768... +32767.

En el ejemplo que sigue, el valor de Var1 a la salida del bucle FOR será 8.

```
FOR Contador:=1 TO 5 BY 1 DO
     Var1:=Var1*2;
     IF Var1 > 5 THEN
          EXIT;
     END_IF
END_FOR;
```

2.6.2 El bucle WHILE

A diferencia del bucle FOR, en el bucle WHILE la condición de terminación puede ser cualquier expresión booleana. El bucle termina cuando la condición es falsa.

Sintaxis:

WHILE <Expresión booleana>
    <Instrucciones>
END_WHILE;

Las <Instrucciones> se ejecutan cíclicamente en tanto la <Expresión booleana> devuelve TRUE. Las <Instrucciones> nunca se ejecutan si <Expresión Booleana> es FALSE en la primer evaluación. Por otro lado, si <Expresión Booleana> nunca toma valor FALSE, las <Instrucciones> se repiten indefinidamente, resultando en un bucle infinito.

NOTA: el programador debe eliminar la posibilidad de un bucle infinito. Esto se logra por ejemplo con una sentencia que incrementa o decrementa un contador en <Instrucciones>.

Ejemplo:

```
WHILE Contador<>0 DO
     Var1 := Var1*2;
     Contador := Contador – 1;
END_WHILE
```

En cierto sentido, los bucles WHILE y REPEAT (descrito a continuación) son más poderosos que el bucle FOR, ya que no requieren de conocer el número de ciclos antes de la ejecución. Por tanto, en algunos casos representan la única solución. Sin embargo, un bucle FOR es preferible si el número de ciclos de bucle es conocido, pues garantiza que no haya bucles infinitos si se toman las precauciones ya mencionadas.

2.6.3 El bucle REPEAT

El bucle REPEAT se diferencia del bucle WHILE en que la condición de salida se verifica después de ejecutar el bucle, y no antes. Esto significa que el bucle siempre se va a ejecutar al menos una vez, independientemente de la condición de salida.

Sintaxis:

```
REPEAT
     <Instrucciones>
UNTIL <Expresión booleana>
END_REPEAT;
```

Las <Instrucciones> se ejecutan hasta que la <Expresión booleana> devuelve TRUE.

Las <Instrucciones> se ejecutan una sola vez si la <Expresión booleana> devuelve TRUE en la primer evaluación.

Las <Instrucciones> se repiten indefinidamente si la <Expresión booleana> no toma valor TRUE. Esto causa un bucle infinito. El programador debe eliminar toda posibilidad de un bucle infinito. Esto se logra por ejemplo con una sentencia que incrementa o decrementa un contador en <Instrucciones>.

Ejemplo:

```
REPEAT
     Var1 := Var1*2;
     Contador := Contador - 1;
UNTIL
     Contador := 0
END_REPEAT;
```

2.6.4 Bucles Infinitos

La posibilidad de generar un bucle infinito está siempre presente al programar un PLC. Cómo se comportará el PLC en estas condiciones no es previsible y puede diferir de fabricante en fabricante.

3 Ejemplo

Se considera el siguiente ejemplo: se desea escribir un programa que controle el encendido - apagado de una bomba.

La bomba será encendida si:

1) Se pulsa el botón de arranque.
2) La protección térmica está deshabilitada.
3) Está abierto el botón de emergencia.
4) Está abierto el botón de parada.

Desde un tiempo T después del encendido, no puede haber ni sobre corriente ni baja corriente. Expresado de otra forma, desde un tiempo T después del arranque, la corriente I debe cumplir $I_{MIN} < I < I_{MAX}$, siendo I_{MIN} e I_{MAX} límites prefijados.

El motor de la bomba se apagará si:

1) Se pulsa el botón de parada.
2) Se cierra la protección térmica.
3) Se pulsa el botón de emergencia.
4) Los límites de corriente no son los correctos.

El programa ST que cumple lo anterior es el siguiente:

```
(* Inicialización *)
IF (NOT CICLOINI_FLAG)
THEN
     ESTADO_APAGADO := TRUE;
     ESTADO_TRANSITORIO := FALSE;
     ESTADO_ENCENDIDO := FALSE;
     CICLOINI_FLAG := TRUE;
END_IF;


TIMER1 (E := ESTADO_TRANSITORIO, T := DELAY );
TIMEOUT_TIMER1 := TIMER1.A;

IF (ESTADO_APAGADO AND ARRANQUE) THEN
     ESTADO_TRANSITORIO := TRUE;
     BOMBA_ON := TRUE;
     ESTADO_APAGADO := FALSE;
ELSIF (ESTADO_APAGADO AND (NOT ARRANQUE)) THEN
          BOMBA_ON := FALSE;
ELSIF (ESTADO_TRANSITORIO AND (TERMICO OR ALARMA OR
PARADA)) THEN
          ESTADO_TRANSITORIO := FALSE;
          BOMBA_ON := FALSE;
          ESTADO_APAGADO := TRUE;
ELSIF (ESTADO_TRANSITORIO AND TIMEOUT_TIMER1)
THEN
          ESTADO_TRANSITORIO := FALSE;
          ESTADO_ENCENDIDO := TRUE;
ELSIF (ESTADO_ENCENDIDO AND (TERMICO OR ALARMA OR
PARADA     OR     (CORRIENTE     >     CORRIENTE_MAX)
OR(CORRIENTE<CORRIENTE_MIN)))
THEN
          ESTADO_ENCENDIDO := FALSE;
          BOMBA_ON := FALSE;
          ESTADO_APAGADO := TRUE;
END_IF;
```

La declaración de variables de este programa es:

```
PROGRAM PLC_PRG
VAR
     DELAY AT %MD4000.1: TIME  := T#5s;
     TIMER1: ESV;
     ESTADO_APAGADO AT %MX0.0: BOOL;
     ESTADO_TRANSITORIO AT %MX0.1: BOOL;
     ESTADO_ENCENDIDO AT %MX0.2: BOOL;
     TIMEOUT_TIMER1 AT %MX0.3: BOOL;
     CORRIENTE_MAX AT %MW3001.0: INT := 120;
     CORRIENTE_MIN AT %MW3001.1: INT := 50;
END_VAR
```

CAPÍTULO 12: ARQUITECTURA DEL PLC SEGÚN IEC 61131-3

La arquitectura del PLC considerada en lo que va del curso consiste de una máquina que ejecuta cíclicamente un programa a intervalos regulares de tiempo, con tres fases: lectura de entradas, ejecución de programa y actualización de salidas.

Este régimen cíclico de ejecución puede alterarse sólo por una interrupción, causada por un evento externo, como puede ser el cambio de estado de una entrada física. La interrupción detiene la ejecución del programa principal y llama a una rutina de atención a la interrupción. Completada la rutina de interrupción, el PLC retoma la ejecución del programa en la misma sentencia donde había sido interrumpido, completando el ciclo de la forma habitual. Aunque no fueron utilizadas, los PLCs del laboratorio presentan la posibilidad de definir interrupciones.

El continuo avance del hardware y del software del PLC llevó a que la norma IEC 61131-3 definiera una arquitectura más avanzada que la considerada hasta el momento en el curso.

La nueva arquitectura del PLC define una programación jerárquica basada en 4 niveles: configuración, recursos, tareas y programas.

La configuración define los recursos, los datos compartidos por todos los recursos, y los datos accesibles desde el exterior del PLC.

Cada recurso define las tareas asociadas, los datos compartidos por todos los programas, los datos del recurso accesibles desde el exterior del PLC, y los programas que ejecutan las tareas del recurso.

La tarea incorpora los conceptos de prioridad y de condición de disparo.

1 Tareas

La declaración de una tarea incluye los siguientes elementos:

- Condición de disparo de la tarea: período de tiempo (Interval) o evento (single)
- Prioridad de la tarea
- Programas asociados a la tarea

Los programas se organizan sobre la base de unidades de programa denominadas POU (Program Organizational Units). Una POU puede ser:

- Un programa (propiamente dicho)
- Un bloque funcional
- Una función o un procedimiento

La condición de disparo de una tarea desencadena la ejecución de los programas de la tarea. Puede ser un intervalo periódico de tiempo expresado en milisegundos o un evento. Si la condición de disparo es un período de tiempo, decimos que la tarea tiene un ciclo

asociado, análogo al ciclo de un PLC. En cada ciclo de tiempo, la tarea se activa ejecutando el programa o los programas asociados.

Definimos la prioridad de la tarea como un número de 1 a n, donde 1 indica la prioridad más alta. Una tarea con prioridad 1 no puede ser interrumpida por otra tarea de prioridad igual o menor. Una tarea de prioridad 2 puede ser interrumpida por una de mayor prioridad, es decir, de prioridad 1. Si el programador asigna la misma prioridad a todas las tareas del recurso, el recurso ejecuta las tareas en forma secuencial, sin interrupción de una sobre la otra.

Una tarea debe tener asociado como mínimo un programa, que puede ser escrito en cualquiera de los lenguajes estudiados en el curso (definidos por el estándar IEC 61131-3).

El siguiente es un ejemplo de configuración de tareas, que define dos tareas hipotéticas asociadas al programa de control del Laboratorio 2.

```
Task Configuration:
    TAREA1(PRIORITY:=1,INTERVAL:=T#100ms)
        PWM
    TAREA2(PRIORITY:=2,INTERVAL:=T#1sec);
        CONTROL
```

En este ejemplo, TAREA1 es una tarea con prioridad 1 que maneja el relé del calentador a través del programa PWM, y TAREA2 una tarea de menor prioridad, con un tiempo de ciclo mayor, que realiza el control a través del programa CONTROL. Nótese como la nueva arquitectura simplifica la programación, evitando la necesidad de programación adicional para ejecutar el bloque de control cada n ciclos.

2 Variables globales, locales y variables de configuración

En lo que va del curso, cada variable se asocia a un lugar de memoria fijo. La arquitectura definida en la norma IEC 61131-3 permite la declaración de variables locales, globales o de configuración.

Las variables locales son análogas a las variables automáticas de los lenguajes de alto nivel. Se definen en un POU. No tienen una dirección de memoria asignada, y su alcance es sólo el POU, esto es, su tiempo de vida se limita a la ejecución del POU y no se pueden acceder desde el exterior del POU. Dos POU distintas pueden definir variables locales con el mismo nombre sin que esto implique conflicto ninguno, ya que las variables definidas existen sólo en el contexto de ejecución del POU que corresponde.

Las variables globales se definen al nivel de configuración, de recursos y de programas. Las variables globales definidas en un recurso son accesibles desde cualquier tarea del recurso, permitiendo un método muy cómodo de relacionar tareas. En el ejemplo del control de la sección anterior, la única relación necesaria entre las tareas 1 y 2 es la variable global “salida_del_control”, que define el ciclo de trabajo del PWM de la tarea 1.

El conjunto de variables globales incluye las variables con dirección de memoria asignada. Estas variables se denominan "variables representadas de forma directa". La dirección puede referir a una localidad del espacio de entrada, de salida o de memoria de datos intermedios. Las variables representadas de forma directa tienen la virtud de ser accesibles desde el exterior del PLC por un sistema SCADA.

Para que una variable sea accesible desde el exterior del PLC, se debe definir en la sección VAR_ACCESS de la configuración.

Las variables de configuración del PLC (definidas al nivel del recurso) permiten definir parámetros de configuración como los ya mencionados en el curso:

dirección MODBUS del PLC
modo de funcionamiento MODBUS del PLC
áreas de datos que no deben ser perdidos en caso de un corte de energía
parámetros de hardware, como el tipo de E/S analógicas (4-20 mA, 0 –10V, etc.)
otros

El formato de una declaración de variable es:

NombreVar [AT %DirecciónVar] : TipoDeDato [:= ValorInicial];

donde:

NombreVar = nombre de la variable

"AT %DirecciónVar" define la variable como una variable representada de forma directa. La "DirecciónVar" se compone de un *código de dos letras* y luego un *número*.

La primer letra debe ser:
I para una dirección de entrada
Q para una dirección de salida
M para una dirección de memoria interna
S para el estado del PLC (reservado para estándares futuros)

La segunda letra debe ser:

X bit (la no presencia de segunda letra también significa bit)
B byte (8 bits)
W word (16 bits)
D double word (32 bits)
L long word (64 bits)

No hay restricción sobre el formato numérico.

TipoDeDato = uno de los tipos de datos reconocidos por el estándar IEC 61131-3. Se definen más abajo.

Valor inicial: valor inicial de la variable

Un ejemplo de declaración de variable es:

motor_1 AT %OW42 : INT := 35

3 Los POU

Hay tres tipos de POUs:

- Programas
- Bloques funcionales
- Funciones o procedimientos

Los programas son las unidades de programa que se ejecutan al dispararse una tarea. Un programa puede llamar a funciones y a *instancias* de *templates* de bloques funcionales. Sin embargo, un programa no puede llamar a otro programa.

Una función acepta múltiples argumentos, aunque el resultado es una sola variable de retorno, definido en la definición de la función. Por ejemplo, la función FuncionEjemplo, declarada por "FUNCTION FuncionEjemplo: WORD", retorna tipo WORD. Cualquier POU puede llamar funciones. Las funciones no tienen memoria, esto es, no tienen variables *permanentes* asociadas – ver más abajo.

Un bloque funcional se define sobre la base de un *template* de bloque funcional. El *template* de un bloque funcional es el programa que define el bloque. Para utilizar un template de bloque, una tarea debe declarar una *instancia* del template de bloque.

Un bloque funcional se distingue de una función en dos aspectos: permite más de una salida, y tiene asociadas variables *permanentes*, esto es, variables que conservan el valor entre un llamado al bloque y otro. Este hecho hace que la utilización de un bloque funcional requiera de la definición de una instancia del t*emplate* del bloque.

4 Tipos de variables

El conjunto de tipos de variables definido es mucho más extenso que el conjunto de los PLCs del laboratorio. A continuación se muestra una lista de los tipos posibles.

INT	Entero con signo de 16 bits. Los prefijos S, D, o L cambian el tamaño a 8 bits, 32 bits, 64 bits.
BOOL	Bit
BYTE	8 bits
WORD	Palabra de 16 bit
DWORD	Palabra de 32 bits
LWORD	Palabra de 64 bits
REAL	Punto Flotante 32 bits
LREAL	Punto Flotante 64 bits
TIME,DATE, TIME_OF_DAY, DATE_AND_TIME	
STRING	string de caracteres ASCII

Estructura de datos contenida en instancia de FB	var_name = nombre FB
Tipo de datos derivado	Definido por el usuario. Sintaxis para definir tipo: TYPE ... END_TYPE. Puede ser un ARRAY, un conjunto de nombres alfanuméricos, un intervalo de números, estructura de datos, composición de tipos derivados (ver ejemplos)

Ejemplos de tipos de datos derivados:

una declaración simple de variable:
TYPE my_real: LREAL;
END_TYPE
una declaración simple de variable seguida de dos números definiendo límites:
TYPE my_real: LREAL (-4.3, +4.3);
END_TYPE
conjunto de nombres alfanuméricos
TYPE mode_switch (load, run, unload);
END_TYPE
Un arreglo de datos del mismo tipo, en una o más dimensiones. Ejemplo, ARRAY de 4x3 *integers*.
TYPE my_array
ARRAY [1..4,1..3] OF INT;
END_TYPE
Estructura de datos que consiste de distintos elementos de distintos tipos de datos. Palabras clave: STRUCT y END_STRUCT. Ejemplo:
TYPE my_struct
STRUCT
switch : BOOL
speed : INT
position : my_array
END_STRUCT
END_TYPE
Composición de *tipos derivados*. Por ejemplo, un *struct* que contenga otro *struct*.

Cada campo de una variable de un *tipo derivado* se accede "var_name.campo"
1)

CAPÍTULO 14: SISTEMAS SUPERVISORIOS (SCADA)

5 Definiciones generales

El objetivo de un sistema SCADA (Supervisory Control And Data Acquisition) es el control y supervisión de una planta (por "planta" debe entenderse la acepción que se le da a esta palabra en la teoría de control). Dicha planta puede ser una red de distribución de energía eléctrica, una red de saneamiento, un proceso industrial, etc.

Un sistema SCADA consta de 5 grupos de componentes:

1) La instrumentación de campo
2) Las estaciones remotas
3) La red de comunicaciones
4) La estación central de supervisión
5) El software que se ejecuta en la estación central de supervisión

En general, las componentes del sistema SCADA se conectan como en la Figura 125.

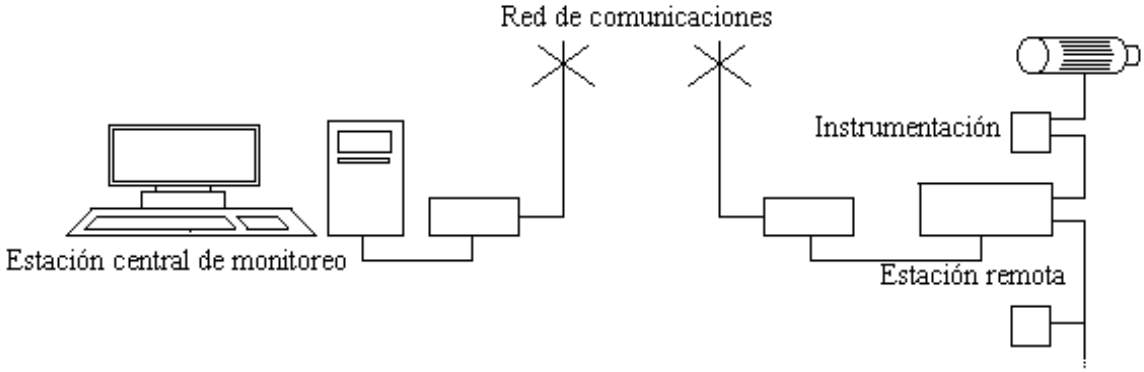


Figura 125

5.1 La instrumentación de campo

La *instrumentación de campo* consta de *sensores* y *actuadores* distribuidos en la planta.

Los *sensores* convierten magnitudes físicas, como caudal, velocidad, voltajes, corrientes, etc., en magnitudes eléctricas (voltajes o corrientes) que pueden ser leídas por la *estación remota*. En general, los sensores consisten de transductores y de los dispositivos de acondicionamiento de señales.

Como la palabra lo dice, los *actuadores* actúan sobre los equipos de la planta. La salida de un actuador puede abrir o cerrar un interruptor, generar la señal de entrada a un comando analógico para controlar la velocidad de un motor, etc.

5.2 Las estaciones remotas

Los instrumentos de campo situados en la planta a supervisar y controlar, se conectan a las *estaciones remotas.* Las estaciones remotas se instalan en las vecindades de los

equipos supervisados o controlados. En general, una estación remota es un PLC (Programmable Logic Controller) o una RTU (Remote Terminal Unit). Las estaciones remotas adquieren las señales digitales o analógicas generadas por los sensores, y actúan sobre los actuadores.

La estación remota puede ser una unidad modular, esto es, una unidad que forma parte de un sistema de módulos remotos. Los módulos pueden ser PLCs, módulos de entrada/salida digital, módulos de entrada/salida analógica, etc. Este sistema se comunica en un plano distinto del plano del sistema SCADA.

5.3 La red de comunicaciones

La *red de comunicaciones* es el medio para transferir información de una localidad a otra. Puede ser una línea telefónica, de telefonía celular, de radio, o cables.

El cable se utiliza normalmente en fábricas. No es práctico en sistemas que cubren grandes áreas geográficas por el alto costo y el gran trabajo de instalación.

El uso de líneas telefónicas es una opción de menor costo en sistemas que cubren áreas geográficas. Para conexiones *on-line* se pueden usar líneas telefónicas dedicadas, caras, mientras que para conexiones que requieren de actualizaciones a intervalos regulares se pueden usar líneas de discado comunes.

Para casos que los sistemas remotos no sean accesibles por líneas telefónicas, una solución económica es el uso de señales de radio.

Las líneas de telefonía celular se usan cada vez más hoy en día. Consisten en establecer un enlace de comunicación de datos de SCADA sobre la red telefonía celular.

5.4 La estación central de supervisión

La *estación central de supervisión* (en inglés, Central Monitoring Station, CMS) es la unidad *maestra* del sistema SCADA. Tiene dos funciones:

1) Recibir la información de las estaciones remotas y generar las acciones de acuerdo a los eventos detectados.
2) Función de interfaz hombre – máquina, entre el sistema al operador (MMI - Man Machine Interface).

Las CMS de un sistema pueden estar dispuestas en red, para compartir la información de los sistemas
SCADA.

5.5 El Software de la Unidad Central

El software de la unidad central de supervisión brinda una ventana al proceso que transcurre en la planta, permitiendo al personal de la planta el acceso a los datos a través de una interfaz amigable. Un análisis más detallado del software de la unidad central se da en 2.

5.6 Ejemplo de un sistema SCADA

En la Figura 126 se muestra un ejemplo de un sistema SCADA para distribución de agua. La unidad remota del tanque elevado se conecta directamente a la estación central de supervisión vía radio. Las unidades remotas en las estaciones de bombeo 1 y 2, se conectan a la estación central de supervisión vía radio, a través de un repetidor de radio.

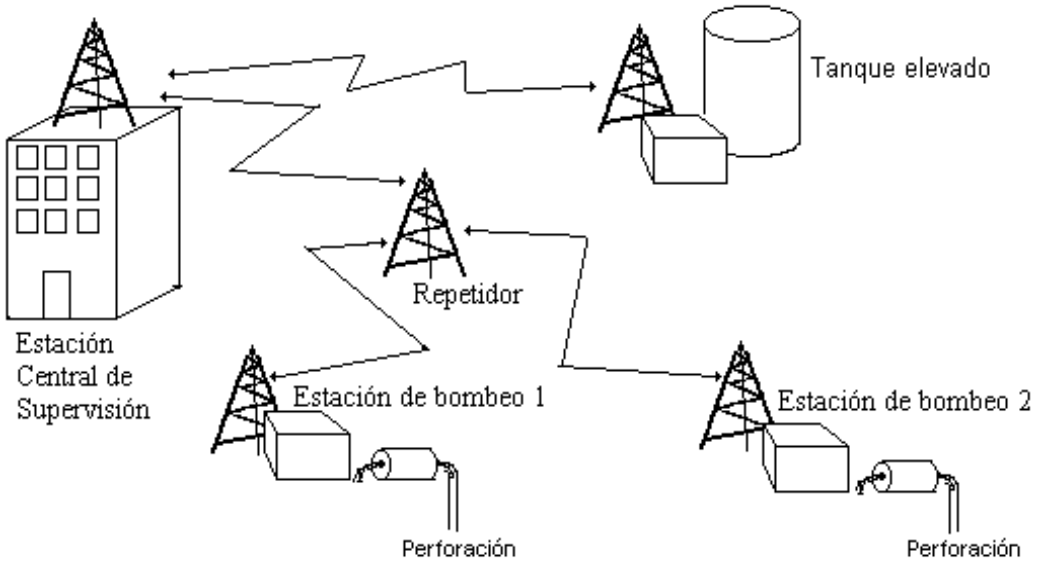


Figura 126

6 El Software de la Unidad Central de Supervisión

6.1 Descripción de las capas del software

El software de la unidad central se denomina de la misma forma que el sistema: "sistema SCADA" o "SCADA". Debido a esto, referiremos tanto al software como al sistema físico por esta nomenclatura, de forma indiferente.

El software de la unidad central es multitarea. Las distintas funciones que el programa realiza, se distribuyen verticalmente en capas, de forma que cada capa brinda servicios a la capa de arriba. El acceso de una capa A a los servicios de la capa inferior B, es a través de la *interfaz de* las capas A y B.

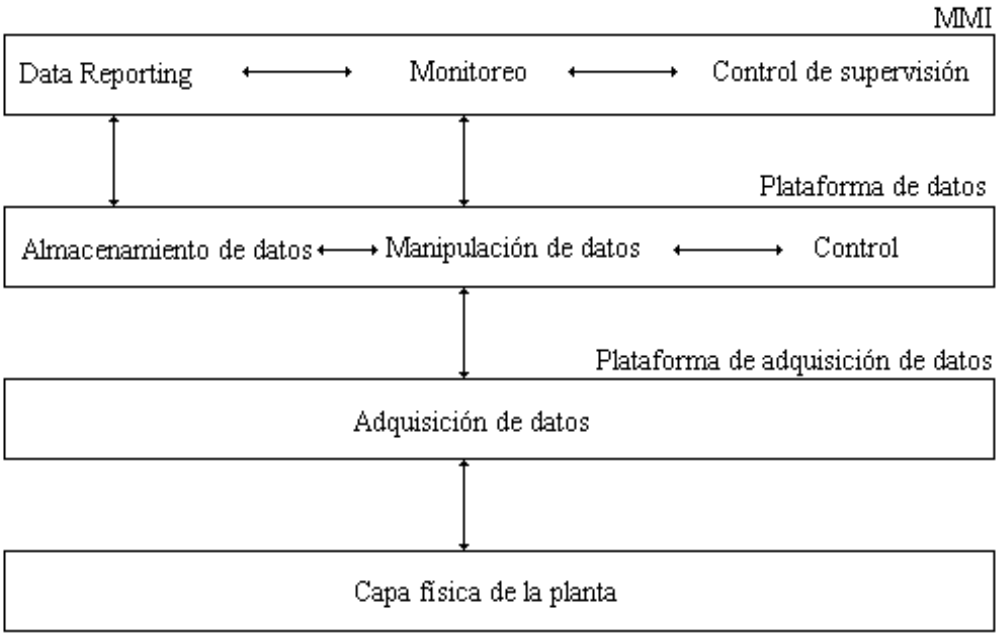


Figura 127

El esquema de capas se muestra en la Figura 127. Cada capa realiza varias *funciones*. Las flechas verticales describen interfaces entre funciones de capas adyacentes, y las flechas horizontales describen interfaces entre funciones de una misma capa. El conjunto de flechas verticales entre dos capas describen la interfaz entre las capas. La bidireccionalidad de las flechas implica que el acceso de la función que accede a la interfaz (en el caso vertical, la función de la capa superior, y en caso horizontal, cualquiera de las dos funciones) es de lectura escritura.

Se observa que hay tres capas: Plataforma de adquisición de datos, Plataforma de datos, y MMI (Man Machine Interface).

6.1.1 La plataforma de adquisición de datos

La *Plataforma de adquisición de datos* realiza la *adquisición de datos*. La adquisición de datos consiste del manejo de entrada / salida de los datos de la planta. Para esto, utiliza la interfaz de capa física, que no es otra cosa que la red de comunicaciones.

6.1.2 La plataforma de datos

La *Plataforma de datos* realiza las funciones de manipulación, control y almacenamiento de datos.

La *manipulación de datos* consiste de la transformación, en uno u otro sentido, de los datos de la interfaz a los datos en un formato interpretable por otras aplicaciones de la Plataforma de datos, así como por aplicaciones de la capa MMI.

El *control* consiste de la aplicación automática de algoritmos que ajustan valores del proceso de forma de mantenerlos entre límites prefijados.

El *almacenamiento de datos* consiste en muestrear y guardar en archivos la evolución de cualquier dato concreto del sistema (histórico). Las frecuencias de muestreo las especifica el operador.

Una función adicional de esta capa es la función de *alarma*, que consiste del reconocimiento y reporte inmediato de eventos excepcionales.

6.1.3 La MMI

La MMI ejecuta la interfaz de usuario, realizando las funciones de monitoreo, control de supervisión y *data reporting.*

El *monitoreo* consiste del despliegue de datos en tiempo real.

El *control de supervisión* permite al operador cambiar *set points* y otros valores claves directamente desde la estación central.

El *Data reporting* permite al usuario exportar los archivos de datos almacenados a archivos de formatos útiles para el análisis posterior. Asimismo, el usuario puede cargar las muestras del dato desde los archivos y desplegar la evolución del dato a partir de cualquier momento.

6.2 La arquitectura básica del software

El software que se ejecuta en la unidad de datos es multitarea. La adquisición de datos es realizada por la tarea *I/O driver*. Las funciones de la plataforma de datos, exceptuando el almacenamiento de datos, las realiza la tarea *SAC* (Scan, Alarm, Control). La interfaz entre el I/O driver y el SAC se denomina *Driver Image Table*. La interfaz entre la SAC y la capa MMI se denomina *Base de datos (Database)*. El flujo de datos se representa en la Figura 128.

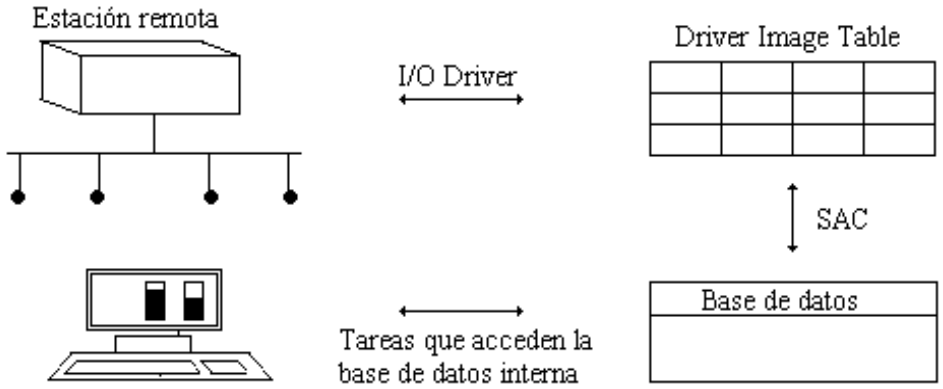


Figura 128

6.2.1 I/O Driver (Plataforma de adquisición de datos)

La adquisición de datos tiene dos aspectos, el hardware de adquisición y el software. El hardware puede ser simplemente un puerto serie de un PC (RS232 o a través de un conversor RS485) o una tarjeta que se instala en un slot del equipo de supervisión. Esta tarjeta habitualmente propietaria, se encarga de la interfaz con el bus de campo.

La tarea que realiza la adquisición de datos es el *I/O driver*. El I/O driver controla el hardware de adquisición y se comunica con la unidad remota, a través del protocolo de comunicaciones. La interfaz del I/O driver (plataforma de adquisición de datos) con la plataforma de datos es el *Driver Image Table*.

El *Driver Image Table* se puede pensar como un conjunto de casilleros. Cada casillero se denomina *poll record*. Cada *poll record* puede contener un único dato de la estación remota, o un conjunto de datos contiguos en la memoria de la estación remota. Los *poll record* se configuran por el usuario. Así, podemos visualizar el Image Table como un imagen de ciertos sectores de la memoria de la estación remota.

El *Driver Image Table* se actualiza cíclicamente. Generalmente el I/O driver lleva a cabo la adquisición de datos para actualizar la Image Table a través del *polling* de las unidades remotas. El intervalo de tiempo entre dos actualizaciones consecutivas se denomina *poll time*.

En algunos sistemas más sofisticados es posible configurar los PLCs de forma que la Image Table reciba mensajes "no solicitados", para registrar eventos raros que deben informarse inmediatamente.

6.2.2 El SAC (Plataforma de datos)

Tiene las siguientes funciones:

- Lee los datos del *Driver Image Table* (interfaz con la plataforma de adquisición de datos).
- Traduce los datos a formato de la base de datos.
- Compara los datos con límites de alarma, y genera mensajes de alarma.
- Ejecuta la lógica de control.
- Detección de excepciones.
- Cumple con los pedidos de escritura de datos en el *Driver Image Table* (datos a ser transmitidos a la unidad remota). Estos pedidos corresponden a las tareas de la misma capa y de capas superiores.

El procesamiento de la tarea SAC puede ser de formas: cíclico, basado en excepciones, o procesamiento una única vez (*One-Shot*).

La *base de datos* es el corazón del sistema. La base de datos es una representación de proceso, creada uniendo bloques de lógica de control de proceso.

La base de datos la crea el usuario. Consiste de bloques y *chains*. Un bloque es un conjunto de código de control de proceso que lleva a cabo una tarea específica. En general hay dos tipos de bloques:

Los bloques primarios leen o escriben datos en el *Driver Image Table*, o realizan alguna función específica.
Los bloques secundarios manipulan los datos que se les pasa.

Los bloques primarios manejan toda la entrada salida con el *Driver Image Table*.

Un *chain* es una serie de bloques conectados que se crean para controlar y supervisar. Por ejemplo, un bucle de control particular que consiste de leer un dato, aplicarle una fórmula estándar y luego enviar el resultado a la estación remota, podría consistir de un Bloque de Entrada Analógica, un Bloque de Cálculo, y un Bloque de Salida Analógica.

Por otra parte, en caso que se requiera leer datos y hacer cálculos para ser utilizados de aplicaciones de software en capas superiores, se crea un bucle de supervisión conectando un Bloque de Entrada Analógica y un Bloque de Cálculo.

6.2.3 Tareas del MMI

La MMI es la capa que ejecuta la ventana al proceso. Consta de una aplicación gráfica que contiene las herramientas necesarias para desarrollar interfaces gráficas que permitan a los operadores interactuar con datos en tiempo real. Las aplicaciones gráficas acceden información de la base de datos.

El usuario configura el o los mímicos de su aplicación y asocia los elementos gráficos con las variables de la base de datos de la capa inferior. De esta forma el usuario puede tener una visión simple y fácil de entender del proceso animando objetos de acuerdo con el valor de dichas variables. Esta interfaz permite visualizar y también modificar valores del proceso, por ej. el setpoint de un PID en la planta.

Otra tarea de esta capa es la generación de reportes. La base de datos del SCADA almacena las variables con una frecuencia configurable y durante un tiempo configurable. Por ejemplo puede almacenar el valor de una temperatura cada 10 segundos y manteniendo 1 semana de historia en la base. Sobre estos datos es posible construir gráficos y reportes analizando la variación de los mismos en el período de almacenamiento.

7 Tópicos más avanzados

7.1 La interfaz entre la capas

En un principio, los sistemas SCADA eran sistemas propietarios cerrados. Si bien existía la estructura en capas, estaba oculta para el usuario.

Posteriormente, la necesidad de comunicación con "cualquier hardware" hizo que los fabricantes de sistemas SCADA separaran el I/O driver del resto de la aplicación, lo que permitía a un proveedor de hardware realizar el I/O Driver para un SCADA dado, y anexarlo al resto de la aplicación.

La separación del I/O driver trajo de la mano el problema de la estandarización de la interfaz, que se resolvió de dos maneras:

A través de una API (Application Programming Interface) que el fabricante de SCADA proveía, y que permitía integrar el driver al sistema.

A través de un protocolo DDE (Dynamic Data Exchange) que el driver debía respetar, para que el SCADA pudiera comunicarse con él.

En los últimos tiempos los diversos fabricantes de hardware y software de sistemas industriales se han venido esforzando en estandarizar una interfaz única para los diversos sistemas. Esta interfaz se conoce con el nombre de OPC (Ole for Process Control). La OPC permite comunicar dos aplicaciones en una arquitectura cliente servidor, a través de un protocolo especializado para este tipo de aplicaciones.

La necesidad de los usuarios de visualizar los datos de la base de datos del SCADA no desde la MMI sino desde otras aplicaciones, hicieron que muchos fabricantes de SCADA habilitaran formas de acceder su base de datos. A modo de ejemplo, hoy día se pueden establecer enlaces dinámicos entre las celdas de una hoja de cálculo y la base de datos.

7.2 Programación dentro del SCADA

Actualmente la tendencia de los sistemas SCADA es a la apertura del sistema hacia aplicaciones de usuario no previstas de forma estándar. Asimismo, hoy día muchos sistemas SCADA añaden a la posibilidad de configuración, la posibilidad de programación dentro del sistema SCADA. De esta forma se extiende el código del sistema. Entre las muchas formas realizar esto, la más común es embeber en el SCADA el lenguaje Visual Basic Scripts, de forma que el usuario puede programar y acceder a todos los datos internos del SCADA.

7.3 Consideraciones sobre los Sistemas SCADA en red

En sistemas de mayor porte es habitual encontrar más de una estación de supervisión., debido a dos razones:

- la necesidad de replicar los datos en más de un lugar físico
- la necesidad de distribuir la base de datos del SCADA, debido a razones de volumen de datos, a razones de distancias, o simplemente a razones operativas.

En estos casos, los terminales en los que se ejecuta el SCADA se disponen en una red LAN. Si bien este tipo de arquitectura es permitido por muchos sistemas, entre las muchas variantes de implementación que existen algunas pueden traer problemas. Por ejemplo, la arquitectura más simple de este tipo, aquella que copia la base de datos del SCADA en todos los terminales, puede llevar a fuertes pérdidas de performance de la red en sistemas con mucho volumen de información, debido al volumen de tráfico requerido en la LAN.

7.4 Conexión con los Sistemas de Información de la planta

La base de datos interna del SCADA es un conjunto de archivos en un formato propietario, diseñado de forma de optimizar la velocidad de escritura en la base. No está pensada para almacenar grandes volúmenes de información ni está estructurada para realizar análisis complejos de los datos almacenados. Las bases de datos que sí contemplan el almacenamiento de grandes volúmenes de información y las facilidades de análisis e integración de la información son las bases de datos relacionales (por ej. Oracle, MS SQLServer, Sybase, Access, etc.).
Es habitual que la información recolectada por un sistema SCADA deba integrarse con la información del sistema de información de la empresa. En muchos casos es necesario realizar análisis estadísticos o reportes más sofisticados que los del sistema SCADA, a partir de información que no se encuentra dentro de la base de datos del SCADA, o es necesario procesar información recolectada por el SCADA en períodos de tiempo mayores a los períodos en que resulta razonable mantener los datos en base interna del SCADA.

Las razones anteriores llevan a que deba trasladarse información de las variables de la base de datos del SCADA a la base de datos relacional que utilice la empresa. Las variables se trasladan en algunos casos directamente, y en otros con algún preprocesamiento. Algunos SCADAs proveen mecanismos para realizar esta operación. Los dos más frecuentes son:

1) Configuración del SCADA de forma que en determinadas condiciones el sistema realice la escritura de ciertos datos en una base de datos relacional.
2) La interfaz de programación descripta en 3.2.