Curso 2

Programación de Controladores (PLC) Modicon (Schneider)

Teoría y ejemplos prácticos desarrollados

Mayo 2020

Rolf Dahl-skog Stade
rolfds@gmail.com

Capitulo 1: conceptos generales

Programación orientada a objetos

Lenguajes de programación

La familia de controladores Modicon

El hardware M340

Direccionamiento de señales

Capitulo 2: *desarrollo de un ejercicio básico para aprender ...*

Usar el software Unity Pro

Crear un proyecto y configurar el hardware

Creación de variables (tipos de variables)

Programación, en tareas y secciones (secuencia de ejecución)

Buscar entre las librerías de clases (Asistente de FFB)

Distintas maneras de conectarse a un controlador

Cargar, o descargar, programas al controlador.

Forzar discretas y análogas.

Persistencia de estados y valores (al reiniciar).

Capitulo 3: *desarrollo de un ejercicio mas completo para aprender...*

Creación de estructuras de datos (DT)

Crear y programar una nueva clases (FB)

Uso de arreglos de estructuras (DT)

Lógicas de alarmas, lógicas registro de eventos, etc

Capitulo 4: *Comunicaciones*

Comunicación serial como esclavo

Configurar una red Ethernet

Comunicación serial como maestro

Pruebas de comunicación. *Usar aplicación ModbusTest*

Seguridad en la comunicación, como evitar *que la lógica pueda ser vulnerada por la comunicación*.

Capitulo 5: *Scantime & watchdog*

Conceptos, tipos de tareas.

Ciclos de ejecución de tareas.

Errores y problemas comunes.

Apendice: *Herramientas para trabajar*

Software Unity Pro, de Schneider.

Software ModbusTest2.exe

Programas de ejercicios y ejemplos.

El script python "ModbusTest2", como funciona este software por dentro.

Capitulo 1.

Programación Orientada a Objetos

Hace ya mucho tiempo que un programa dejo de ser, solo una lista de instrucciones. La programación ahora es modular, y esto es mucho mas que dividir el problema en partes independientes, es encapsular datos y operaciones relacionados dentro de un modulo, ocultando lo micro para hacer mas visible lo macro. La modularidad permite, además de reutilizar códigos, aumenta significativamente la legibilidad de los programas.

Para comprender la programación en lenguajes de programación modernos, es necesario entender los conceptos básicos de "Programación Orientada a Objetos:

Que es una “función”
Que es una “clase”
que es un ”objeto”
que es una “propiedad”
que es un “método”

Función

Una función es un programa, que recibe valores, y devuelve como resultado otros valores.

Clase

Una clase es la definición de un tipo de objeto, su definición incluye sus características, sus “propiedades” (atributos), y sus “métodos” (comportamientos).

Objeto

Un objeto es una entidad (una variable) de una clase (tipo) determinada.
Es normal tener muchos objetos iguales de cada clase, cada uno con su propiedad identidad (tag), y cada uno con sus propias propiedades (valores).

Propiedad

Cada "propiedad" es un atributo, o característica, de un tipo de dato determinado.

Método

Los "métodos", de una clase, son sus funciones "publicas" (que son accesibles, desde fuera de la clase).

Todos los lenguajes de programación ya vienen con muchas clases predefinidas.
Ejemplos de “clases” que ya existen en los controladores programables:

Bool (verdadero o falso)
Int (entero con signo)
Uint (entero sin signo)
Real
String (cadena de caracteres)

TON (Timer On Delay)
CTU (contador ascendente)
AnalogInput
DiscreteInput
DiscreteOutput
etc...

Ejemplos de "objetos" que podemos crear en un controladores programable:

X	tipo Int	valor 47
Y	tipo Uint	valor 35
Z	tipo Uint	valor **-**19
A	tipo Real	valor 23**,**8
M2	tipo String	valor "mensaje dos"
retardo3	tipo TON	valor preset 3 seg

Si llamamos al "método" Len (largo) del objeto M2, nos devuelve el valor 11, esto ocurre por que todos los objetos de la clase **String**, tienen un método Len, que cuenta cuantos caracteres tienen de largo.
M2**.**Len ---> 11

Cada clase tiene, propiedades de lectura y escritura (entradas), propiedades de solo lectura (salidas), y propiedades privadas (variables internas invisibles), y además cada clase tiene una lógica interna (código de programación) que sera invisibles desde afuera, que define sus comportamientos. Una clase una vez definida, la podemos usa en tantos objetos como necesitamos, pero sin importarnos como esta programada internamente.

Por ejemplo, si creamos un objeto de nombre "retardo3" del tipo **TON** (**T**imer **On** Delay) este tendrá todas las características, propiedades y métodos (comportamientos) de la clase TON. Una propiedad **In**, una propiedad **Out**, una propiedad **Preset**, etc y se comportara como esta definido en la clase TON.
Para medir un tiempo, solo escribimos en las propiedades de lectura/escritura (entradas) y leemos las propiedades de solo lectura (salidas), del objeto de la clase TON, pero no nos importa como esta programada internamente la clase TON.

Una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Y son objetos, todos los elementos (datos) de un clase (tipo) que los define, creados con un identificador (tag) que los identifica.
Al comprender esto, nos damos cuenta de que, **también podemos definir y crear una nueva clase**, que tenga las características que necesitamos.

Por ejemplo:
definimos la **clase "motor"** con las siguientes propiedades de lectura y escritura (entradas): partir, parar, estado, y propiedades de solo lectura (salidas): correr, falla, y con los siguientes métodos: partir, parar, etc
Luego podemos crear todos los "motor" que necesitamos, y todos se comportaran iguales, por que todos comparten la misma programación.

Otro ejemplo:
definimos la **clase "válvula"** con las siguientes propiedades: abierta, cerrada, posición, salida, enFalla, etc. y con los siguientes métodos: abrir, cerrar, parar.
Luego podemos crear todas las "válvula" que necesitamos, y todos se comportaran iguales, por que todos comparten la misma programación.

Otro ejemplo:
si el proyecto requiere 400 alarmas, en vez de emprender la larga y rutinaria tarea de programar la lógica de cada una de las 400 alarmas, podemos definir una **clase "alarma"**, con el comportamiento (lógica) y características (propiedades) que necesitamos. Y luego crear un arreglo (lista) de 400 elementos de esta clase "alarma" que definimos.

Con esta forma modular de programar, reutilizamos código, podemos tener guardadas nuestras propias librerías de clases.
Al ocultar parte internas del código, obtenemos programas mas legibles, ocultando la menudencia insignificante y resaltando lo importante.

Estos mismos conceptos se aplican para todos los lenguajes de programación modernos: Python, Java, C#, C++, etc

Lenguajes de programación

Un lenguaje de programación, son reglas gramaticales para escribir instrucciones o secuencias de órdenes, en forma de algoritmos, de manera que puedan ser entendidas por la maquina, con el fin de lograr que la maquina tenga el comportamiento deseado.
La historia de los PLC se remonta a finales de la década de 1960, y se crearon como un reemplazo de los paneles de control cableados con relés.
El primer PLC fue desarrollado por la división automatización de General Motors.
Modicon es la abreviatura de **MO**dular **DI**gital **CON**troler.

Estos primeros PLC se programaban en lenguaje "Lista de instrucciones".
Luego para facilitar el mantenimiento de estos, a personas que no eran programadores, sino electricista, se crearon herramientas para representar el lenguaje "Lista de instrucciones" como un diagrama eléctrico, a este lenguaje gráfico se le llamo "Ladder" (escalera).
El lenguaje "Ladder" no es mas que la representación gráfica del lenguaje "Lista de instrucciones".

Hoy en día las **normas IEC 61131** de la Comisión Electrotécnica Internacional, define los estándares para los **PAC** (**C**ontroladores de Automatización **P**rogramables) antes llamados PLC (Controladores Lógicos).

Como ejemplo, a continuación, se muestra el mismo programa en distintos lenguajes estándares.

Lista de instrucciones (Instruction List)

```
LD %I0.2.0
OR %Q0.4.0
AND %I0.2.1
ANDN %M3
ST %Q0.4.0
LD %IW0.3.0
GT 5600
ST %M3
```

Diagrama Escalera (Ladder Diagram)

El lenguaje LADDER, es un lenguaje de programación gráfico muy popular, por que como está basado en los clásicos esquemas de control eléctricos con reles, es fácil de entender para un técnico eléctrico. Por lo que es más adecuado para controlar variables discretas (boleanas) pero es difícil manipular las variables analógicas y expresar operaciones aritméticas. Tiene un soporte muy limitado

para las matrices y bucles, resultando a menudo en la duplicación de código. Es muy limitado para la programación orientada a objetos.

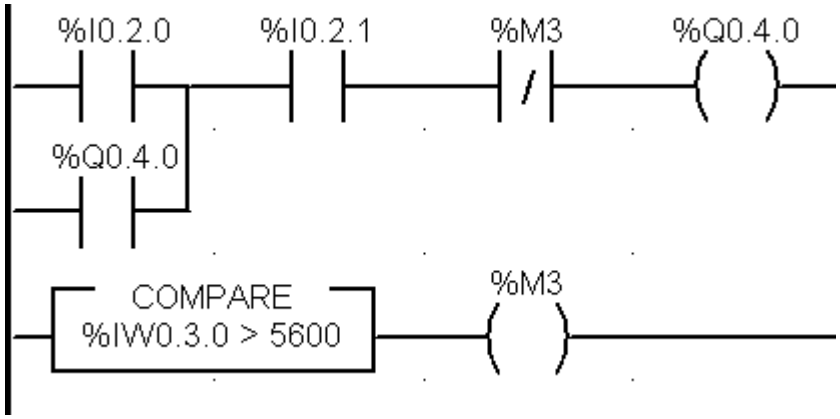
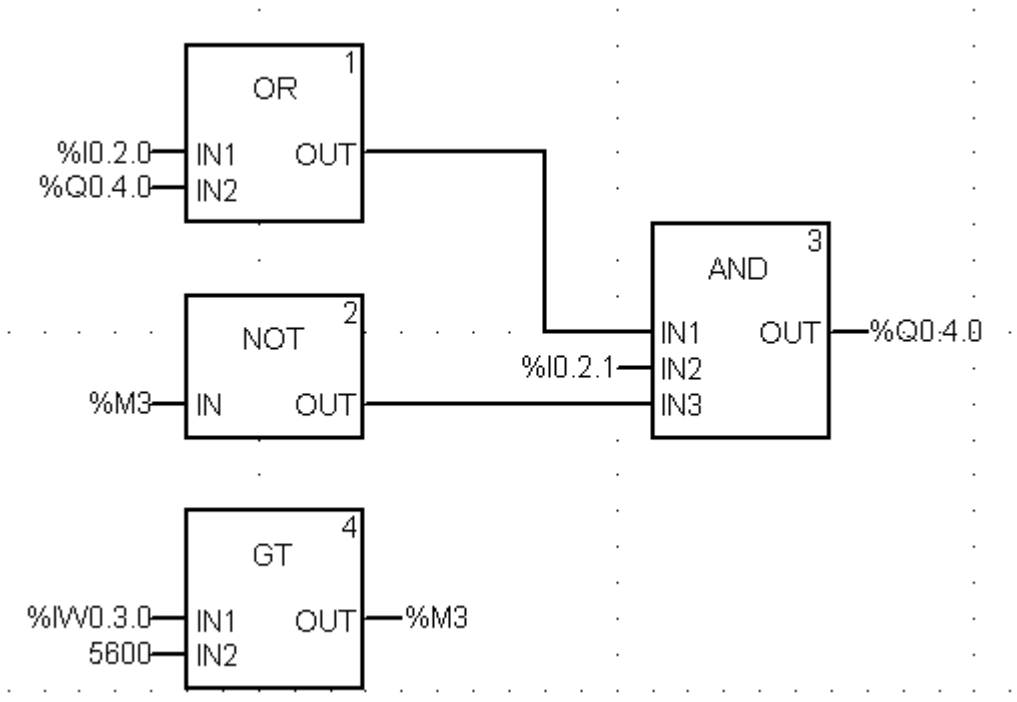


Diagrama de bloques de funciones (Function Block Diagram)

El lenguaje Function Block, se vuelve difícil de seguir cuando el programa es muy grande. No tiene un soporte para matrices y bucles, resultando a menudo en la duplicación de código. Es muy limitado para la programación orientada a objetos.



Texto estructurado (Structured Text)

El Texto Estructurado tiene todas las ventajas de un lenguaje de programación moderno, lo que permite una programación modular funcional y orientada a objetos.
Su sintaxis es clara y fácil de aprender, es muy parecido a Pascal, Python, y Basic.

```
Partir := %I0.2.0;
Parar := %I0.2.1; (* normal cerrado *)
Nivel := %IW0.3.0; (* analoga *)

MotorBba := (Partir or MotorBba)
  and Parar and not AltoNivel;

AltoNivel := Nivel > 5600; (* 56,0 % *)

%Q0.4.0 := MotorBba;
```

Es muy fácil insertar comentarios dentro del código, permitiendo lograr programas bien documentados.

Controladores Modicon

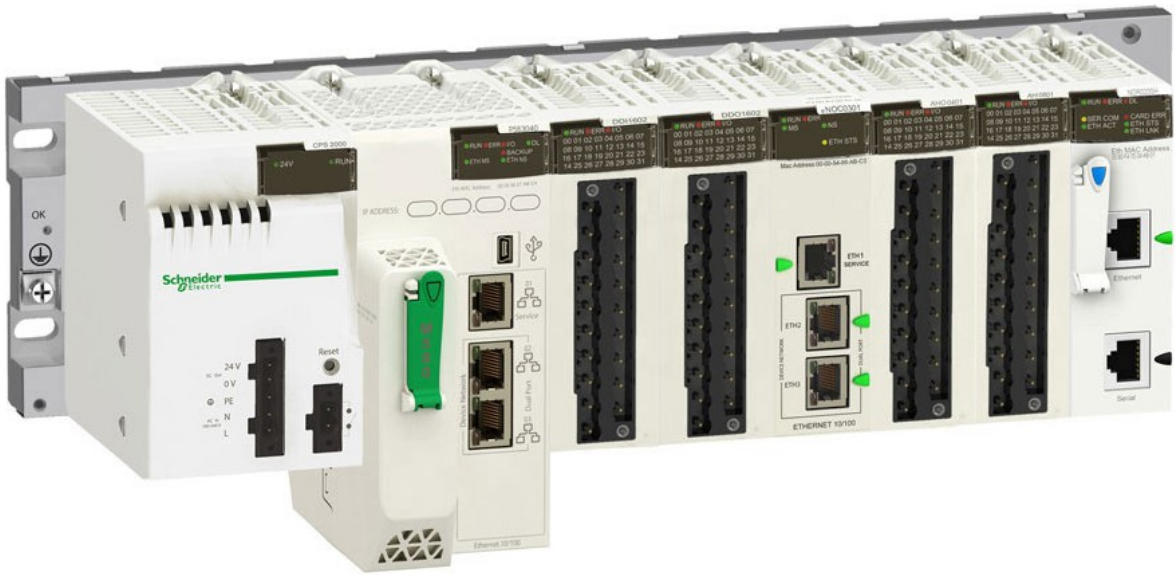
Los controladores Modicon, actualmente perteneciente a Schneider, tienen tres lineas de productos:

1.- Los **M200**, son las **PLC** básicos, son para maquinas pequeñas. M221, M241, M251, M262, etc

2.- Los **M340**, son los **PAC** de gama alta, para automatización de procesos y maquinas complejas.
Utilizan los módulos entrada/salida de la linea X80.

3.- Los **M580**, son los **PAC**, de tope de gama, con Ethernet integrada en el núcleo, para el control de plantas de proceso.

Utilizan los mismos módulos entrada/salida de la linea X80.
Control distribuido sobre red Ethernet redundante
Procesadores redundantes.



Los siguientes capítulos de este curso de capacitación, se refieren específicamente a los **M340**.

El hardware Modicon M340

Procesadores M340

Los M340 tienen varios modelos de procesador, diferenciándose por su capacidad y tipos de puertos de comunicación. Todos los procesadores, tiene un puerto USB para programarlo.

En la imagen se muestra procesador P34-2020, que tiene un puerto Ethernet (marca verde) y un puerto serial (marca negro).

Al lado izquierdo tiene la tarjeta de memoria donde se guarda el programa. Cada vez que el procesador se apaga, cuando vuelve a ser energizado toma el programa desde esta tarjeta de memoria. El procesador no usa batería de respaldo.

No sacar, ni cambiar la tarjeta de memoria con el procesador energizado.

Racks X80 Modicon

Existen varios modelos de racks compatibles con los módulos X80, con 4, 6, 8 o 12 slots.
Los BM**X**-XBP… que tienen solo el bus de comunicación serial (X Bus).
Los BM**E**-XBP… que además tienen un bus de comunicación Ethernet.
Ejemplos:
BMX-XBP0800 es un rack de 8 ranuras, con bus serial.
BMX-XBP1200 es un rack de 12 ranuras, con bus serial.
Cada rack requiere un modulo fuente de alimentacion.
También hay racks especiales para fuentes de alimentación redundantes.

Un mismo modelo de rack puede ser usado como rack principal, si tiene un modulo procesador en su ranura 0, puede ser un rack de extensión si esta conectado a otro, por medio de un "kit de extensión", como el BMX-XBE2005.

Fuentes de alimentación

Existen fuentes de alimentación de distintas potencias y voltajes, para ser montadas en los racks X80. Y también existen fuentes de alimentación redundantes.
Ejemplo: BMXCPS3020 alimentación de 20 a 48 volt continuos, 32 watt.

Módulos Entrada/Salida X80

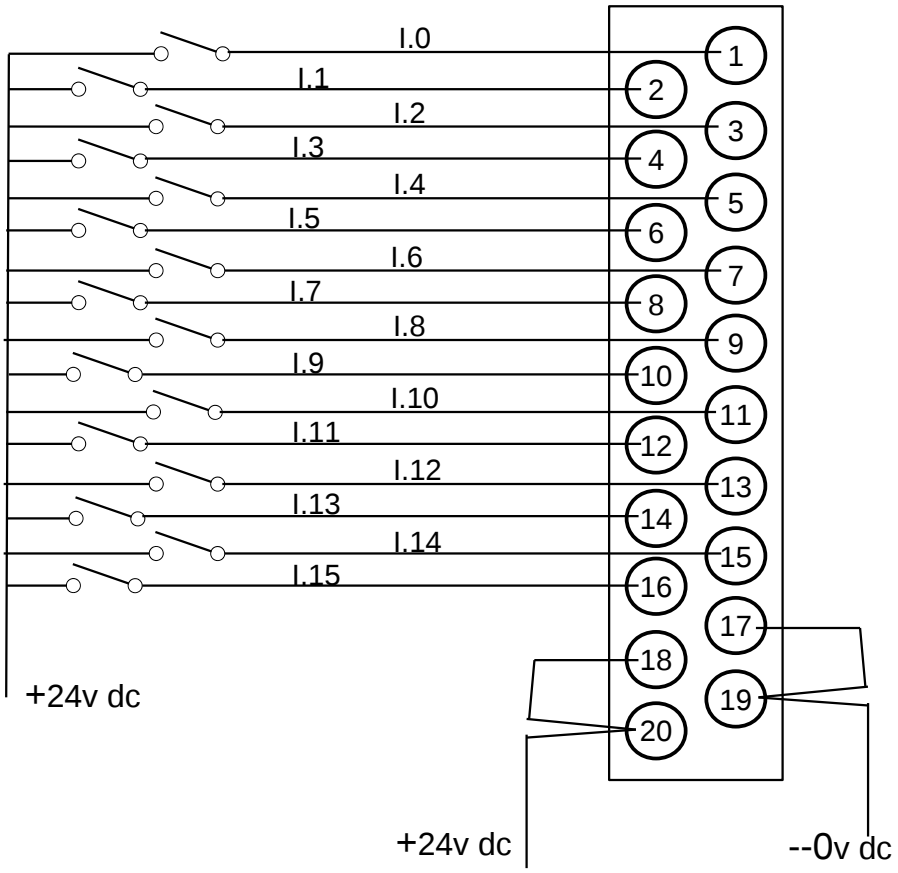
Existen módulos X80 para todos los tipos de entradas, salidas, o comunicación. Los mismos módulos pueden ser usados en racks BMX o en racks BME, en sistemas M340 o M580. Aunque algunos módulos especiales, requieren un bus ethernet (racks BME).

Todos los módulos se pueden quitar y poner con tensión y con el procesador en Run, excepto la fuente de alimentación y el mismo procesador.

Módulos entradas discretas

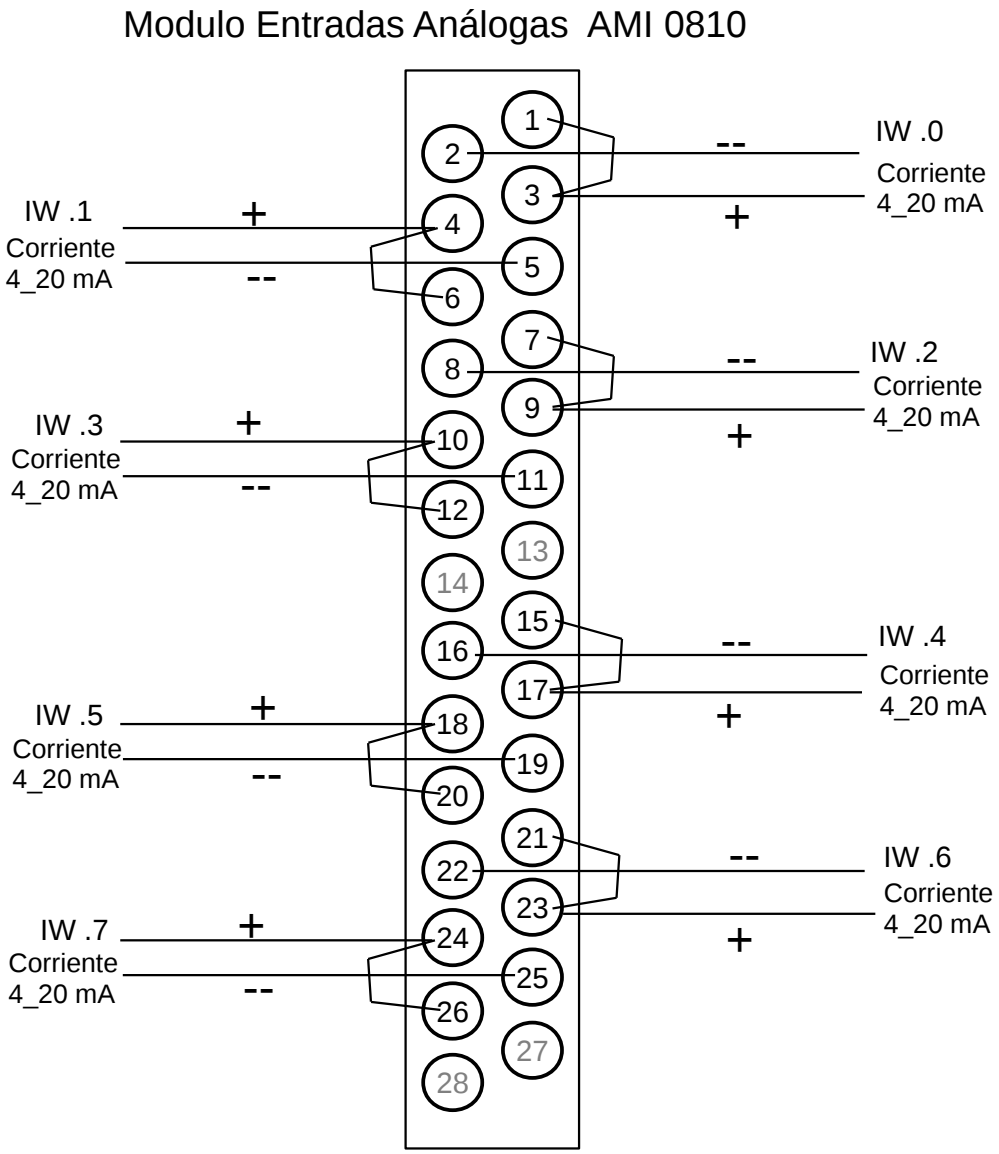
Existen varios modelos de módulos de entradas discretas, uno muy usado es el modelo **BMX-DDI1602** de 16 entradas discretas de 24 volt continuos.
Este requiere un bloque conector **BMX-FTB2010** de 20 pines.

Modulo Entradas Discretas DDI 1602



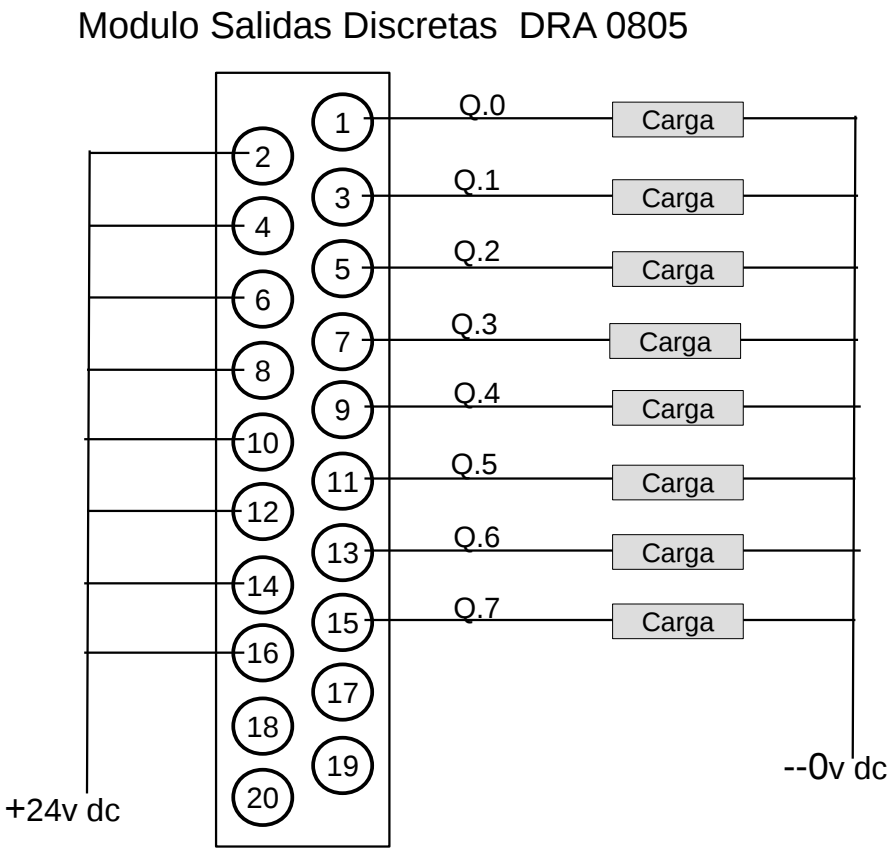
Módulos de entradas análogas

Existen varios modelos de módulos de entradas análoga, uno muy usado es el modelo **BMX-AMI0810** de 8 entradas análogas aisladas. Cada canal puede ser conectado para medir 0 a 20 mili-Amperes (puente conectado), o puede ser para medir 0 a 5 volt (sin el puente).
Este modelo tiene aislación galvánica entre canales, por lo que puede tolerar lazos de control alimentados desde distintas fuentes.
Este requiere un bloque conector **BMX-FTB2800** de 28 pines.



Módulos salidas discretas

Existen varios modelos de módulos de salidas discretas, por triac, por transistor, de común positivo, de común negativo, etc. El modelo que usaremos es el **BMX-DRA0805** de 8 salidas por rele.
Este requiere un bloque conector **BMX-FTB2010** de 20 pines.



Direccionamiento de señales

Direcciones de entradas y salidas:

Todas las direcciones están formadas por %tipo.rack.slot.canal
Por ejemplo:
%I.0.2.3 = Entrada discreta en rack 0, slot 2, canal 3 (cuarto canal).
%IW.0.3.1 = Entrada análoga en rack 0, slot 3, canal 1 (segundo canal).
%Q.0.4.2 = Salida discreta en rack 0, slot 4, canal 2 (tercer canal).
Address "0.0.1" = segundo puerto de comunicaciones, en rack 0, slot 0 (CPU).
%QW.1.5.3 = Salida analoga en rack 1, slot 5, canal 3 (cuarto canal).
Address "0.6.0" = primer puerto, en rack 0, slot 6 (modulo de comunicacion).

Direcciones de comunicación Modbus:

Discretas de lectura o escritura.

%M0	=	Coil 0:0001
%M1	=	Coil 0:0002
%M2	=	Coil 0:0003
%M3	=	Coil 0:0004
%M4	=	Coil 0:0005
%M5	=	Coil 0:0006
%M6	=	Coil 0:0007
%M7	=	Coil 0:0008
%M8	=	Coil 0:0009
%M9	=	Coil 0:0010
%M10	=	Coil 0:0011
%M11	=	Coil 0:0012

etc …

Análogas de lectura o escritura, son "Word", enteros de 16 bits. Por defecto se asume que son enteros CON signo, es decir su valor puede ser desde -32768 (2 elevado a 15) hasta +32767 (2 elevado a 15 menos 1), el bit 16 se usa como signo.

%MW0	=	Holding Register 4:0001
%MW1	=	Holding Register 4:0002
%MW2	=	Holding Register 4:0003
%MW3	=	Holding Register 4:0004
%MW4	=	Holding Register 4:0005
%MW5	=	Holding Register 4:0006
%MW6	=	Holding Register 4:0007
%MW7	=	Holding Register 4:0008

etc …

Capitulo 2:

En este capitulo vamos a desarrollar un ejercicio muy básico, que nos va a permitir cubrir los siguientes temas:

Usar el software Unity Pro

Crear un proyecto y configurar el hardware

Creación de variables (tipos de variables)

Programación, en tareas y secciones (secuencia de ejecución)

Buscar entre las librerías de clases (Asistente de FFB)

Distintas maneras de conectarse a un controlador

Cargar, o descargar, programas al controlador.

Forzar discretas y análogas.

Persistencia de estados y valores (al reiniciar).

Definición del ejercicio a desarrollar:

Supongamos que tenemos las siguientes señales:

HS-01 = Entrada discreta, botón pulsador partir.

HS-02 = Entrada discreta, botón pulsador parar.

LSLL-03 = Entrada discreta, interruptor de bajo nivel en pozo de agua.

XS-06 = Salida discreta, motor de la bomba de agua.

LT-04 = Entrada análoga, nivel en estanque de agua-

PT-05 = Entrada análoga, presión de descarga de bomba de agua.

XA-01 = Salida discreta, alarma bajo nivel en pozo de agua

XA-02 = Salida discreta, falla de la bomba de agua

La lógica, para este ejercicio, seria la siguiente:

El botón HS-01 hace partir la bomba, si no hay motivo que la detenga.

El botón HS-02 hace parar la bomba.

El LSLL detiene la bomba, si su señal no llega, y también activa alarma XA-01.

El LT-04 hace partir la bomba, cuando al nivel esta bajo 48%.

El LT-04 hace parar la bomba, cuando al nivel esta sobre 91%.

Alarma XA-02 se activa si la presión PT-05 es menor a 2,1 K/cm2, después de 5 segundos, de activar el motor de la bomba.

Un Ejemplo, muy básico, muy simple.

El hardware necesario

Un rack BMX-XBP0800

Una fuente BMX-CPS-2000
Este modelo de fuente se alimenta con 220 volt, y puede entrega 24 volt, que se van a usar para alimentar las señales.

Un procesador BMX- P34-2020, en el slot 0 del rack.

Un modulo BMX-DDI1602, en el slot 2 del rack.

Un modulo BMX-AMI0810, en el slot 3 del rack.

Un modulo BMX-DRA0805, en el slot 4 del rack.

En la imagen, se ven también módulos de comunicación, en los slot 6 y 7, estos no los vamos a usar por ahora; se usaran en otros capítulos de esta misma capacitación.

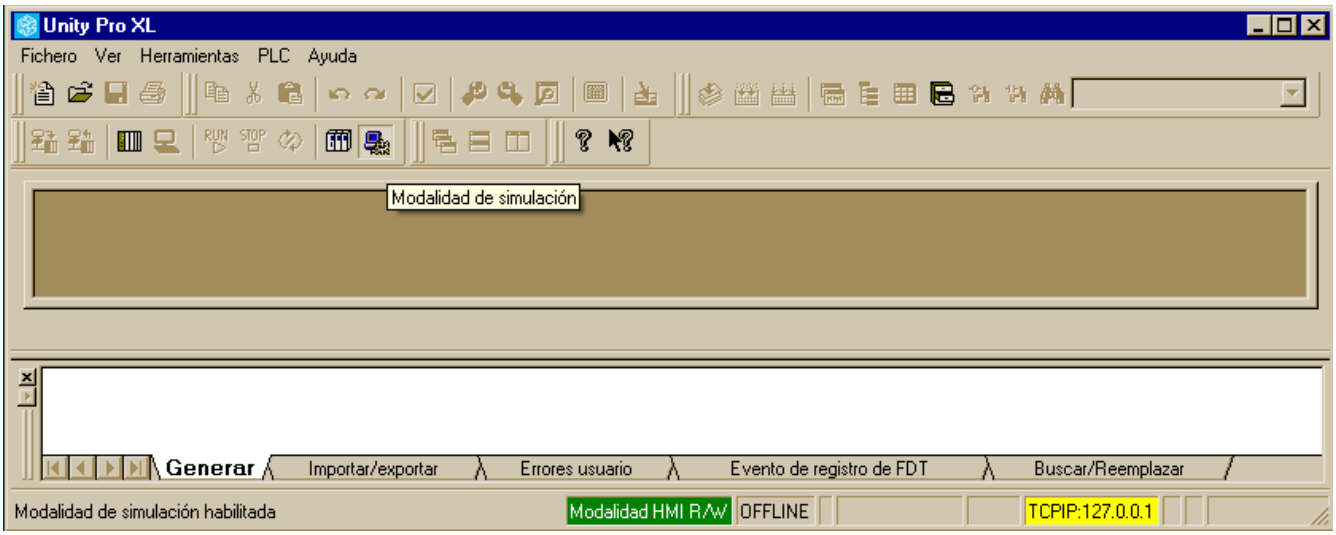
El software Unity Pro

Necesitamos software "Unity Pro XL" (versión 8 o superior)

Vamos a usar una maquina virtual
que tiene instalada la versión 8 "Unity Pro XL".

> En el Anexo (al final) hay mas información sobre donde están las herramientas, material, y programas de ejemplo, para este curso.

Abre la maquina virtual y en el escritorio esta el icono (el cubo azul), que abre el "Unity Pro XL".

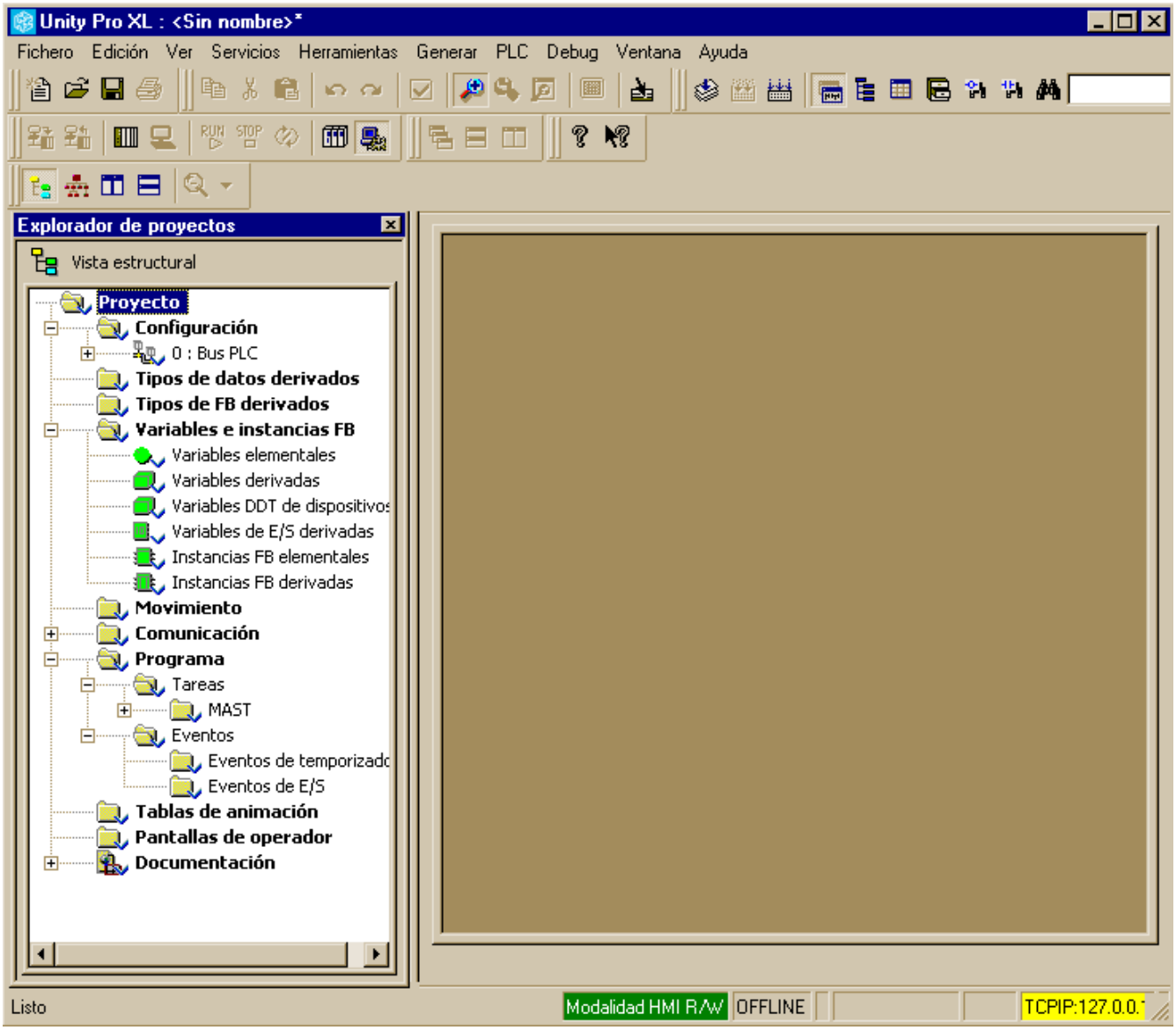


En la parte inferior vemos un área blanca, es la ventana de resultados, que nos muestra si hay errores al ejecutar una compilación o carga. Si tenemos poco espacio podemos ocultarla, mientras no la necesitamos. Menú → Ver → Ventana de resultados.

Lo primero que vemos es unos botones "modalidad simulación" y otro botón "modalidad estándar". Por defecto esta seleccionado el modo simulación, esto permite probar la lógica en un plc virtual, pero no permite conectarse a un plc real.

Al apuntar sobre un botón muestra para que sirve. Todas estas funciones también están ordenadas en los menús.

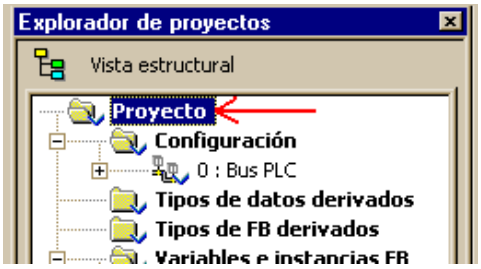
Selecciona en menú Archivo → Nuevo. Selecciona el procesador **P34-2020**, y bastidor (rack) XBP0800.



Ahora ya tenemos nuestro primer programa, aun vacío y sin nombre.
El menú creció con muchas mas funciones, y apareció la ventana "explorador del proyecto".

Si tenemos poco espacio, esta ventana la podemos ocultar o mostrar, rápidamente presionando el botón "explorador del proyecto"

Seleccionar con el botón derecho del ratón, sobre donde dice "proyecto", y en el menú desplegable, selecciona propiedades, para darle un nombre al tu nuevo programa o aplicación.



Carpetas en el explorador del proyecto:

Configuración
Permite definir los componentes de hardware y su configuración.

Tipos de datos derivados (DDT)
Los "**D**erived **D**ata **T**ype" (DDT) son conjuntos de datos, formado por: un arreglo de datos del mismo tipo, o un arreglo de una estructura de datos.
Se crean en la tabla de edición de datos, en la pestaña tipos DDT.

Tipos de FB derivados (DFB)
Los "**D**erived **F**unction **B**lock" (DFB) son bloques de funciones programados por el usuario, es decir cada DFB es una nueva **clase** que creamos, o agregamos.
Se crean en la tabla de edición de datos, en la pestaña tipos DFB.
Se pueden cargar **librerías** de clases, o importar/exportar clases individuales.

Variables e instancias FB

Variables elementales ← objetos de tipos predefinidas. Por ejemplo variables tipo: Bool, Integer, Real, etc

Variables derivadas ← objetos de estructuras o arreglos de datos (DDT). Por ejemplo: un arreglo de Boleanos, un arreglo de Integer, etc.

Variables DDT de dispositivos. ← objetos de tipos de datos, definidos por un modulo o dispositivo especial.

Variables E/S derivadas. ← objetos de tipos definidos por la comunicación con otros dispositivos.

Instancias FB elementales ← objetos de clases predefinidas. Por ejemplo: timers, contadores, PID, etc.

Instancias FB derivadas ← objetos de nuevas **clases** agregadas.

Comunicación
Aquí se definen las redes de comunicación.

Programa
Tareas Mast
Tareas Fast
Tareas por Eventos

En estas carpetas están las secciones que forman el programa, dentro de la carpeta “Mast” las que forman el programa principal, de ejecución cíclica, ordenadas por orden de ejecución.

Dentro de la carpeta “Fast” pequeños programas de ejecución periódica, que necesitan ejecutarse cada tiempo exacto..

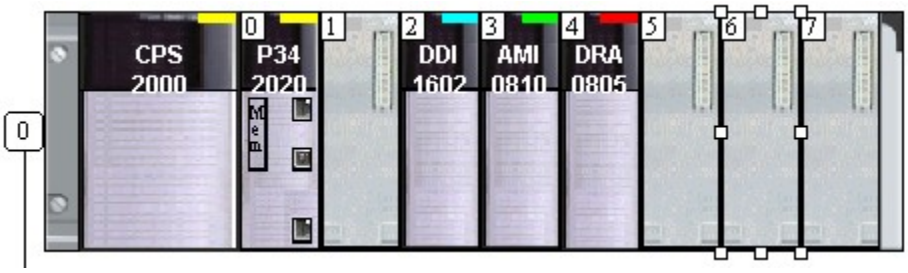
Dentro de la carpeta “Eventos” pequeños programas que se ejecutan, por ejemplo por el evento de una entrada de un modulo contador rápido.

Tablas de animación

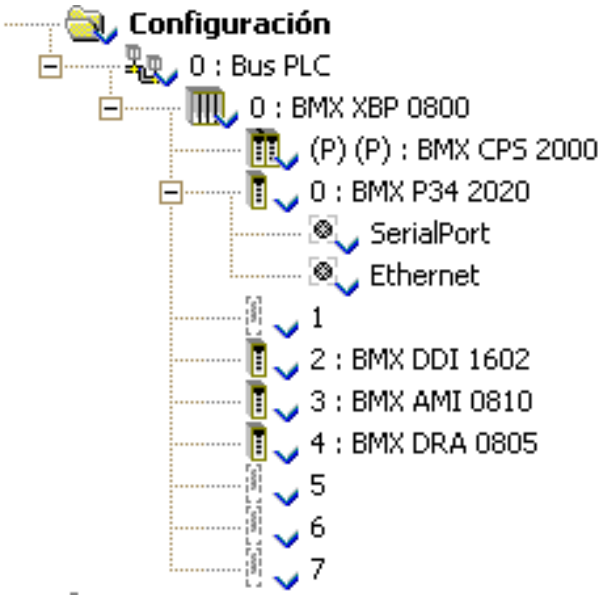
Podemos crear tablas con las variables que nos interesa visualizar, cuando estamos conectados en linea.

Configurar el hardware:

En el explorador del proyecto abrimos la carpeta configuración.
Se despliega como árbol todo el hardware, que inicialmente solo tiene el rack y el procesador. Si hacemos doble click sobre el rack, se abre una ventana que nos permite seleccionar y agregar los módulos en cada slot.



Dejamos el rack 0 (el único) de esta forma.



Configuraciones en el procesador (cpu):
Click derecho sobre el procesador (cpu) para abrir sus propiedades.

En la pestaña configuración, marcar la opción "Inicio automático de la ejecución". Si esta opción no esta seleccionada , cada vez que se apaga y re-enciende el plc, este quedara en modo Stop, esperando hasta que alguien le diga Run.

En la misma pestaña configuración, esta la cantidad de memoria asignada para cada tipo de datos. Si el programa es muy grande, con muchos datos, puede que se necesite hacer algunos ajustes, aquí.

Configuración de salida discretas:
Click derecho sobre cada modulo de salidas discretas.
Donde dice modalidad de retorno: Si se selecciona "mantener", las salidas mantendrán su valor cuando la CPU pase a STOP, la CPU se apague, o este modulo pierda comunicación con la CPU. Si se selecciona "retorno", las salidas cambiaran al valor de retorno, que debe ser definido por cada canal. 0 apagar, 1 encender.

Configuración de entradas análogas:
Click derecho sobre cada modulo de entradas análogas.
Por cada canal hay que definir su configuración.
Utilizado: habilitar o deshabilitar el canal.
Símbolo: se completa automáticamente cuando lo asignamos a una variable.
Rango: Elegir entre las opciones (0 a 10v, 0 a 20mA, 4 a 20mA, 1 a 5v, etc).
Escala: por defecto 0% = 0 y 100% = 10000. la falla por bajo rango -8%; y la falla por sobre rango en 108% (se pueden ajustar, o deshabilitar)
Filtro: permite filtrar señales ruidosas, con un filtro pasa bajos.

De la misma manera, si hay módulos de salida análoga, también hay que revisar sus rangos y valor de retorno.

Creación de las variables

Abrir la tabla de edición de variables, y crear las que necesitamos:

Nombre	Tipo	Dirección	Comentario
HS01	Ebool	%I0.2.1	Botón pulsador partir
HS02	Ebool	%I0.2.2	Botón pulsador parar
LSLL03	Ebool	%I0.2.8	Interruptor bajo nivel en pozo agua.
XS06	Ebool	%Q0.4.2	Motor de la bomba de agua.
LT04	Int	%IW0.3.0	Nivel en estanque de agua
LT04_H	Bool		Alto nivel en estanque
LT04_L	EBool	%M2	Bajo nivel en estanque
PT05	Real		Presión descarga bomba (0 a 5 K/cm²)
PSH05	Real		Setting presión descarga (2,5 K/cm²)
PT05_H	Bool		Hay presión de descarga
XA01	EBool	%Q0.4.5	Alarma bajo nivel en pozo
XA02	EBool	%Q0.4.4	Alarma falla de la bomba de agua
Bomba	Bool		Para la lógica de la bomba

Todas las entradas y salidas análogas deben ser tipo Int, entero con signo. Por defecto tienen un rango de 0 a 100000.

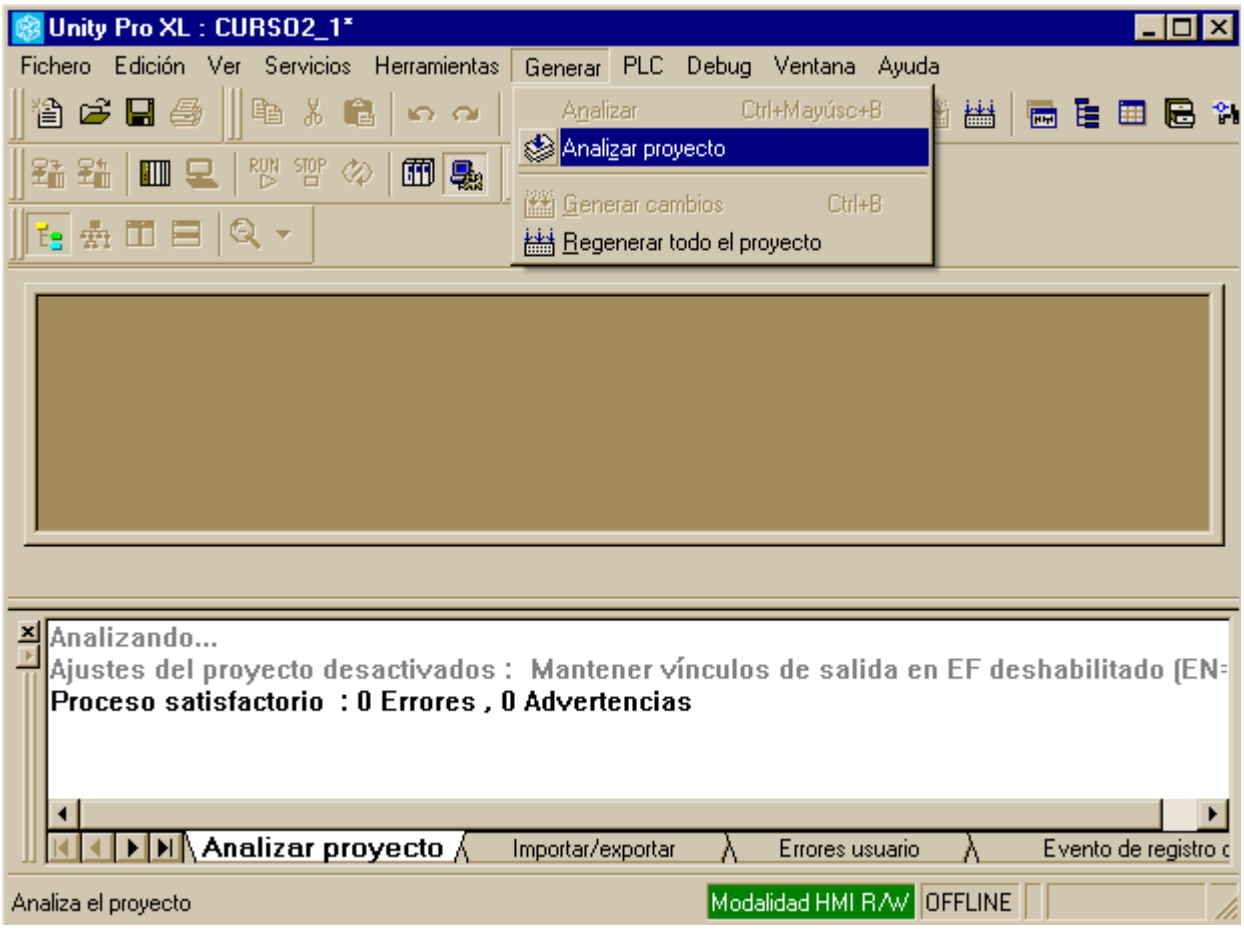
Todas las entradas y salidas discretas deben ser tipo **E**Bool. La diferencia entre EBool y Bool, es que EBool es una discreta forzable, y las Bool son discretas que no se pueden forzar.

Una variable puede no tener dirección física, ni tampoco dirección Modbus, en ese caso es una variable interna, y tampoco es forzable.

Para la segunda alarma necesitamos un retardo de 5 segundos, para esto, en la misma tabla de edición de variables, en la pestaña "bloques de funciones", creamos un objeto de la clase TON.

Nombre	Tipo	Comentario
Retardo1	TON	Timer para la alarma 2

Ahora para revisar si todo va bien, en menú → ver → activar Ventana de resultados, y luego, hacemos un menú → generar → **Analizar proyecto**



Cada vez que hacemos cambios, podemos llamar a "Analizar proyecto" y nos muestra donde hay errores o inconsistencias.

Doble click, sobre el mensaje en rojo, te lleva a donde esta el error.

Programación de tareas

En el explorador de proyecto, abrimos la carpeta Programa → Tareas → MAST → Secciones. Click derecho sobre Secciones, "Nueva seccion".
Nombre "analogas", lenguaje ST, protección ninguna.

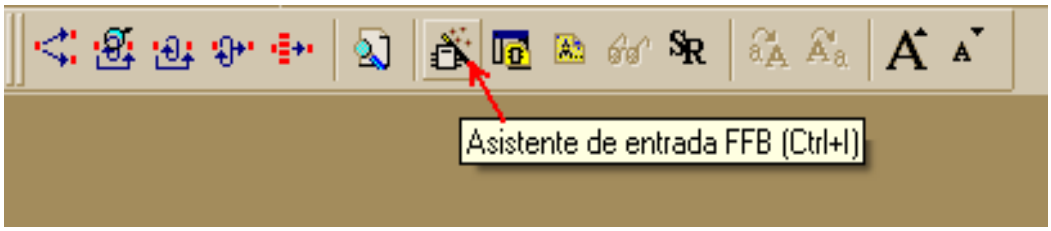
En la ventana de la sección "**analogas**" escribimos lo siguiente.

```
LT04_H  :=  LT04 > 9100;  (* si mayor 91% *)
LT04_L  :=  LT04 < 4800;  (* si menor 48% *)
PT05  :=  INT_TO_REAL( IN:=%IW0.3.2);
PT05  :=  (PT05 / 2000.0);  (* de rango 10000 a rango 5 K/cm² *)
PSH05  :=  2.5;  (* Setting presión descarga 2,5 K/cm² *)
PT05_H  :=  PT05 > PSH05;  (* hay presión de descarga *)
```

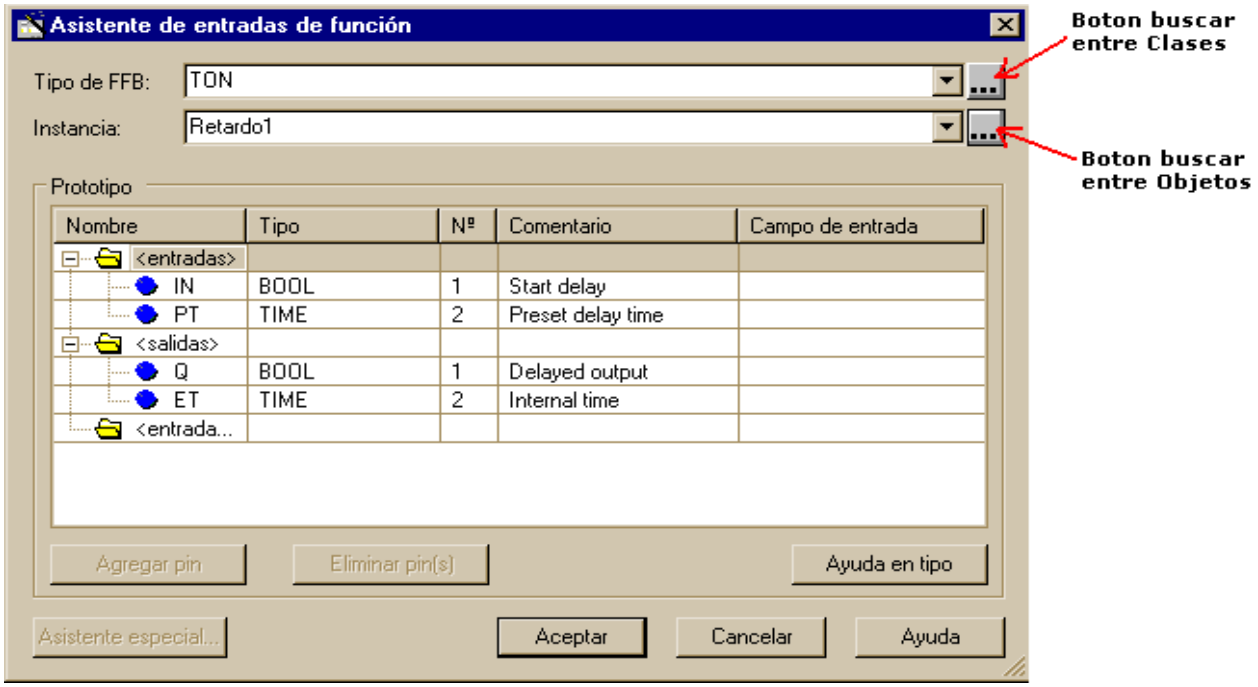
Cada sentencia, u orden; puede tener una o varias lineas, pero siempre se debe marcar su final con un **;**

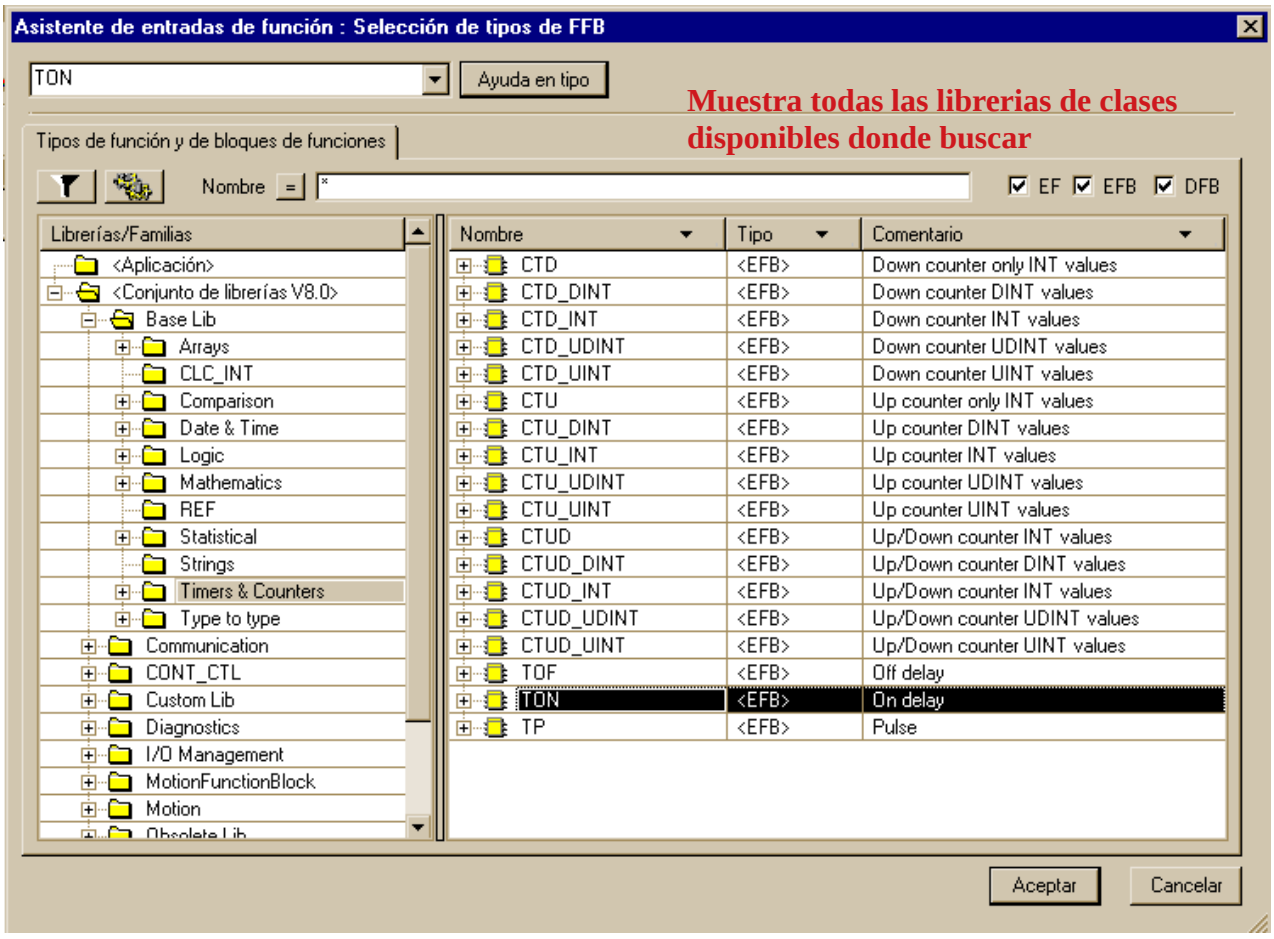
Lo de color azul es el código ejecutable del programa, lo de color verde son comentarios. El compilador ignora todo lo que esta entre (* y *).
Podemos meter comentarios entremedio del código, para explicarlo.

En la segunda linea usamos la **función** "INT_TO_REAL", por que la entrada análoga es un valor entero, y necesitamos convertirlo a un valor real (con decimales).
Por que no se puede hacer operaciones aritméticas, ni tampoco comparar, entre valores de distintos tipos. Fíjate que en la linea tres se divide por 2000**,**0 (un real), si se dividiera por 2000 (un entero) daría error tipos distintos.



Cuando seleccionamos la ventana del programa, en la barra de herramientas se habilitan varios botones que nos facilitan el ingresar el código de programa. El botón "**Asistente de entrada FFB**" es de gran ayuda para buscar y seleccionar entre las funciones disponibles. Nos permite navegar por todas las librerías de clases, incluyendo las que importamos, y las clases que nosotros hemos creado.


Asistente de entradas de función
Tipo de FFB:
TON
Instancia:
Retardo1
Prototipo
Nombre
Tipo
Nº
Comentario
Campo de entrada
<entradas>
IN
BOOL
1
Start delay
PT
TIME
2
Preset delay time
<salidas>
Q
BOOL
1
Delayed output
ET
TIME
2
Internal time
<entrada...
Agregar pin
Eliminar pin(s)
Ayuda en tipo
Asistente especial...
Aceptar
Cancelar
Ayuda
Boton buscar entre Clases
Boton buscar entre Objetos


Asistente de entradas de función : Selección de tipos de FFB
TON
Ayuda en tipo
Muestra todas las librerias de clases disponibles donde buscar
Tipos de función y de bloques de funciones
Nombre
=
*
EF
EFB
DFB
Librerías/Familias
<Aplicación>
<Conjunto de librerías V8.0>
Base Lib
Arrays
CLC_INT
Comparison
Date & Time
Logic
Mathematics
REF
Statistical
Strings
Timers & Counters
Type to type
Communication
CONT_CTL
Custom Lib
Diagnostics
I/O Management
MotionFunctionBlock
Motion
Obsolete Lib
Nombre
Tipo
Comentario
CTD
<EFB>
Down counter only INT values
CTD_DINT
<EFB>
Down counter DINT values
CTD_INT
<EFB>
Down counter INT values
CTD_UDINT
<EFB>
Down counter UDINT values
CTD_UINT
<EFB>
Down counter UINT values
CTU
<EFB>
Up counter only INT values
CTU_DINT
<EFB>
Up counter DINT values
CTU_INT
<EFB>
Up counter INT values
CTU_UDINT
<EFB>
Up counter UDINT values
CTU_UINT
<EFB>
Up counter UINT values
CTUD
<EFB>
Up/Down counter INT values
CTUD_DINT
<EFB>
Up/Down counter DINT values
CTUD_INT
<EFB>
Up/Down counter INT values
CTUD_UDINT
<EFB>
Up/Down counter UDINT values
CTUD_UINT
<EFB>
Up/Down counter UINT values
TOF
<EFB>
Off delay
TON
<EFB>
On delay
TP
<EFB>
Pulse
Aceptar
Cancelar

Podríamos poner todo el código en una sola sección, pero se prefiere separarlo para hacerlo mas ordenado.

Luego agregamos otra sección, de nombre "bomba", lenguaje ST.

En la ventana de la sección "**bomba**" escribimos lo siguiente.

```
Bomba := ( HS01  or  LT04_L  or  Bomba)   (* parte con botón O bajo nivel*)
  and  not  HS02       (* para con botón parar *)
  and  not  LT04_H       (* para con alto nivel *)
  and  LSLL03;      (* para si falta interruptor de nivel *)

XS06  :=  Bomba;
```

En español lo anterior equivale a:

La bomba funcionara si ***(****esta botón partir* ***O*** *bajo nivel* ***O*** *ya esta andando****)***
Y *si* ***No*** *esta botón parar*
Y *si* ***No*** *esta alto nivel*
Y *si esta interruptor de nivel****.***

En la ultima linea se asigna el valor de "Bomba" al canal de salida.

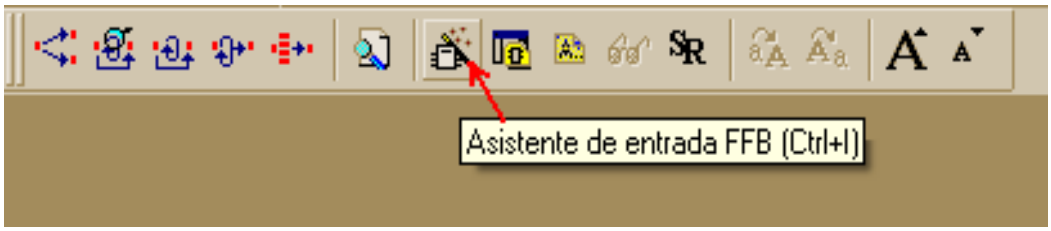
El orden en que muestran las secciones del programa, en el explorador de proyecto, es el orden en que se ejecutan. En algunos casos este orden , o secuencia de ejecución es importante.
Para cambiar el orden, arrastra la sección en el explorador de proyecto.

Luego agregamos otra sección, de nombre "alarmas", lenguaje ST.

En la ventana de la sección "**alarmas**" escribimos lo siguiente.

```
(* alarma 1,  si no llega señal se interruptor LSLL03 *)
XA01 :=  not  LSLL03;
(* alarma 2,  si no hay presión de descarga de la bomba
   cuando la bomba lleva mas de 5 segundos corriendo *)
Retardo1( IN :=  (Bomba  and not  PT05_H ),
          PT := t#5s,
          Q  =>  XA02 );
```

Aquí usamos el elemento del la clase TON (Timer On Delay), que creamos antes.



Usa el botón "Asistente de entrada" para agregar el objeto Retardo1.

Retardo1 tiene dos entradas: **In** de tipo bool, que es lo que lo activa, y la entrada PT (preset) del tipo time, donde le ajustamos su tiempo; y tiene una salida Q, del tipo bool, donde nos entrega un resultado.

Ahora es bueno revisar si todo va bien, en menu → ver → activar Ventana de resultados, y luego, hacemos un menu → generar -> **Analizar proyecto**

Distintas maneras de conectarse a un controlador

Hay dos formas de conectarse a los procesadores modelo BMX-P34-2020.

1. Por conexión directa a su puerto USB
2. Por conexión a su puerto de red Ethernet

1. Conexión directa a su puerto USB

Se necesita un cable USB estandar, que por el lado del computador tenga un conector tipo A y por el lado del PLC tenga en conector tipo mini-B.
Si usas Unity Pro Xl dentro de una maquina virtual, recuerda pasarle el dispositivo "BMX CPU", a la maquina virtual.
En el menú → plc → seleccionar modalidad estándar (No simulación).
En el menú → plc → seleccionar establecer dirección.
Al lado izquierdo de la ventana que se abre, donde dice PLC, poner como dirección **sys** y como medio **USB**. Presionar el botón "comprobar conexión".
Si no hay error, ahora puedes conectarte seleccionando menú → plc → Conectar.

2. Conexión a su puerto de red Ethernet

Los procesadores (CPU) tiene puertos seriales y Ethernet, y ambos usan el mismo conector RJ45. Para distinguirlos el Ethernet tiene una marca verde, y el serial tiene una marca negro.

La dificultad es saber cual es la dirección **IP** que esta usando.

La dirección IP que tiene un procesador M340 modelo BMX-P34-2020, depende de lo que tenga configurado en su programación, y también de la posición de dos selectores ubicados por debajo del mismo modulo.

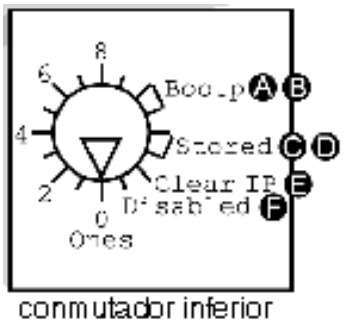
Un procesador P34-2020 nuevo, que no tenga ningún programa siempre tendrá la dirección IP por defecto, que se calcula según su dirección MAC, que viene escrita sobre el mismo, por que no tiene ninguna programada.

Un procesador P34-2020 que si tenga un programa, puede usar la IP que se le configuro en el programa, o puede usar la dirección IP por defecto, que se calcula según su dirección MAC. Dependiendo de la posición de los selectores ubicados por debajo del mismo modulo.

Dirección IP por defecto:
La IP por defecto se calcula como 84.x.x.x usando los tres últimos números de su dirección MAC, que esta escrita en el panel frontal del procesador debajo de su panel de visualización.
Las direcciones Ip se ingresan en decimal , pero las direcciones MAC siempre están en hexadecimal.

Por ejemplo si su MAC es 00-80-F4-10-88-C1
10 hexadecimal = 16 decimal
88 hexadecimal = 136 decimal
C1 hexadecimal = 193 decimal
Por lo tanto su ip por defecto seria 84.16.136.193

Los 2 conmutadores rotatorios de la parte trasera del módulo proporcionan una forma de seleccionar una dirección IP.
La selección en el conmutador inferior de un parámetro no numérico (BOOTP, STORED, CLEAR IP, DISABLED) hace que la configuración del conmutador superior no tenga importancia.



conmutador inferior

BOOTP: (A o B) obtiene una dirección IP de un servidor

STORED: (C o D) usa la ip configurada en el programa.

CLEAR IP: (E) usa la IP por defecto (en base a su direccion mac).

DISABLED: (0) deshabilita las comunicaciones Ethernet.

Una vez conocida su direccion IP, en el menú → plc → seleccionar modalidad estándar (No simulación).

Luego en el menú → plc → seleccionar establecer dirección.
Al lado izquierdo de la ventana que se abre, donde dice PLC, poner como dirección **su IP** y como medio **TCPIP**. Presionar el botón "comprobar conexión".

Para conectarte seleccionando menú → plc → Conectar.

Nota:
Los puertos seriales Modicon usan, el mismo conector RJ45 de ocho pines, que los puertos Ethernet. Para distinguirlos, los que tienen una marca **color verde son Ethernet**, y los que tienen una marca color negro son seriales.

Amigos, en la esquina de la sala de reuniones, del grupo Electrónica, sobre una mesita, les dejo este kit de pruebas para practicar.

Los que no tengan acceso a este equipo, pueden usar Unity Pro en modo simulación.

En el menú → plc → seleccionar **modalidad simulación**.

De esta manera se conectan y cargan el programa en un plc virtual, que les permite probar toda la lógica, y se comporta igual que un plc real, con la sola excepción de que no puede emular el funcionamiento de los puertos de comunicación.

Cargar o descargar programas

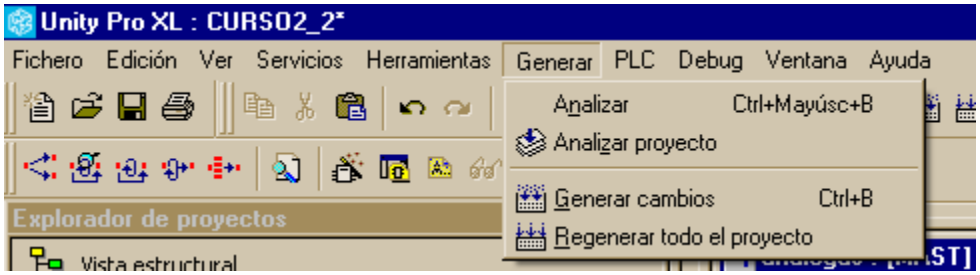
Al conectarse a un plc (menú→ plc → conectar), pueden ocurrir varias situaciones posibles.

El programa en el computador es igual al que hay en el PLC.
El programa en el computador es distinto al que hay en el PLC.
El computador no tiene programa y el plc si tiene.
El computador tiene un programa y el plc no tiene ninguno.
El programa en el computador no esta "generado" (no compilado).

El menú → plc, tiene opciones para cubrir todas estas situaciones.

Se puede copiar hacia el PLC, o también copiar desde el PLC.

Solo se puede cargar hacia el plc un programa que ya ha sido analizado y generado.



Menú → Generar → Analizar :
Solo la sección actualmente seleccionada.
Menú → Generar → Analizar proyecto :
Todo, configuración y lógicas.
Menú → Generar → Generar cambios :
Los cambios realizados en linea (conectado).
Menú → Generar → Regenerar todo el proyecto :
Los cambios realizados desconectado, significa detener y volver a cargar el programa hacia el plc, para poder volver a ver lógica en linea.

Solo se puede ver en linea, un programa que se igual al que hay en el plc.

Si lo modificaste (estando desconectado), vuelve atrás con el archivo de respaldo, o lee el programa desde el plc. Si modificas un programa sin estar conectado, para cargarlo al plc se deberá detener el plc.

Si modificas un programa estando conectado, no es necesario detener el plc, se usa menú → generar → "generar cambios", con lo que se actualiza el programa en el plc, sin necesidad de detener el plc. Recuerda Fichero → Guardar.

Cada vez que cargas un nuevo programa hacia el plc, primero te pide confirmar si estas seguro que quieres detener el plc. Y después de terminar la carga, **el plc**

se quedara detenido. Hacer **menú → plc → Ejecutar**, para volver a ponerlo en Run.

Al ver el programa en linea :

Al ejecutar el programa, y verlo en linea, las variables booleanas, cuando están en **falso** se ven color **rojo**, o cuando están en **verdadero** se ven color **verde**.

Las variables análogas se muestran en color amarillo. Para ver los valores análogos, click derecho sobre la variable, y seleccionar "Nueva ventana de inspección". Click derecho sobre una "ventana de inspección", para cerrarla.

Abre la sección "analogas", seleccionar PT05 y presionar botón lentes.


```
LT04_H  :=  LT04 > 9100;   (* si mayor 91% *)
LT04_L  :=  LT04 < 4800;   (* si menor 48% *)

PT05    :=  INT_TO_REAL( IN:=%IW0.3.2);
PT05    :=  (PT05 / 2000.0);   (* de rango 10000 a rango 5 K/cm² *)
PSH05   :=  2.5;   (* Setting presión descarga 2,5 K/cm² *)
PT05_H  :=  PT05 > PSH05;   (* hay presión de descarga *)
```

PT05
3.362

El nivel LT04 esta normal, por esto LT04-H esta apagado, y LT04-L también esta apagado.
A PT05 se le agrego una "ventana de inspección" que muestra se valor actual, 3,362 K/cm2. Como PT05 esta mayor a 2,5 el PT05-H esta activado.

Abre la sección “bomba”

```
Bomba := ( HS01   or   LT04_L   or   Bomba)    (* parte x botón O bajo nivel*)
  and   not   HS02          (* para con botón para *)
  and   not   LT04_H          (* para con alto nivel *)
  and   LSLL03;          (* para si falta interruptor de nivel *)

XS06   :=   Bomba;
```

Vemos que Bomba esta activada, los botones partir, parar no están activados.

La señal LSLL03 esta forzada en verdadero, por eso se muestra verde con un rectángulo.

Y en la sección “alarmas”

```
(* alarma 1,  si no llega señal se interruptor LSLL03 *)
XA01 :=   not   LSLL03;

(* alarma 2,  si no hay presión de descarga de la bomba
    cuando la bomba lleva mas de 5 segundos corriendo *)
Retardo1( IN :=   (Bomba   and not   PT05_H ),
               PT := t#5s,
               Q   =>   XA02 );
```

La alarma 1 (XA01) esta apagada por que LSLL03 esta forzado en verdadero.

La alarma 2 (XA02) esta apagada por que Bomba esta activada y PT05-H también esta activado.

Forzar discretas y análogas.

Se puede forzar desde la sección del programa, o también desde una "**Tabla de animación**".
En el explorador del proyecto, click derecho sobre la carpeta tablas de animación, y seleccionar, nueva tabla de animación. Ponle un nombre original si vas a usar varias tablas.

Nombre	Valor	Tipo	Comentario
HS01	0	EBOOL	boton pulsador partir
HS02	0	EBOOL	boton pulsador parar
LSLL03	F1	EBOOL	Int bajo nivel pozo
LT04	6500	INT	nivel en estanque de agua
LT04_H	0	BOOL	alto nivel en estanque
LT04_L	0	EBOOL	bajo nivel en estanque
%IW0.3.2	5100	INT	
PT05	2.55	REAL	presion descarga bomba
PT05_H	1	BOOL	hay presion descarga bba
Bomba	1	BOOL	
XS06	1	EBOOL	motor bomba agua
%M27	0	EBOOL	
%MW4	0	INT	

A una tabla le puedes agregar las variables que te interesan, ingresándolas por su nombre o una dirección.
Te muestra su valor actual, y ahí esta una de las formas de forzar. En la imagen de ejemplo, se ve que FSLL03 esta forzada en 1.

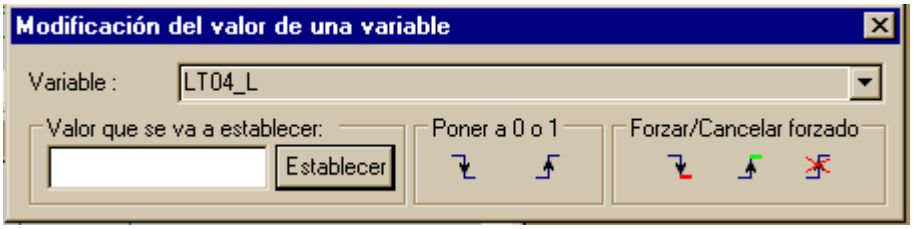
Las variables solo se pueden forzar si cumplen estas dos condiciones:

- Ser discreta del tipo **E**bool
- Tener dirección física (entrada/salida) **o** dirección Modbus.

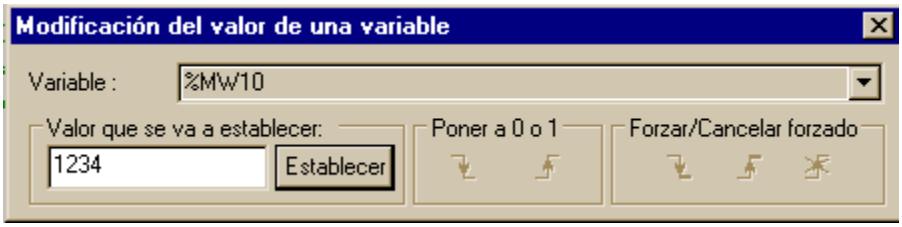
En la imagen de ejemplo, LT04-H no se puede forzar, pero LT04-L si se puede forzar.

La dirección Modbus %M27 (coil 0:0028) si se puede forzar, aunque ni siquiera hemos creado esta variable, por que todas las direcciones Modbus discretas ($\%M_X$) por definición son tipo Ebool.

La otra forma de forzar es, al ver el programa en linea, encontramos la variable y hacemos click derecho sobre esta, y se despliega un menú en el que esta la opción "Modificar valor ..."



En la imagen LT04 que es Ebool y tiene dirección Modbus, podemos cambiarle su valor, y también podemos forzarla a mantener el valor.

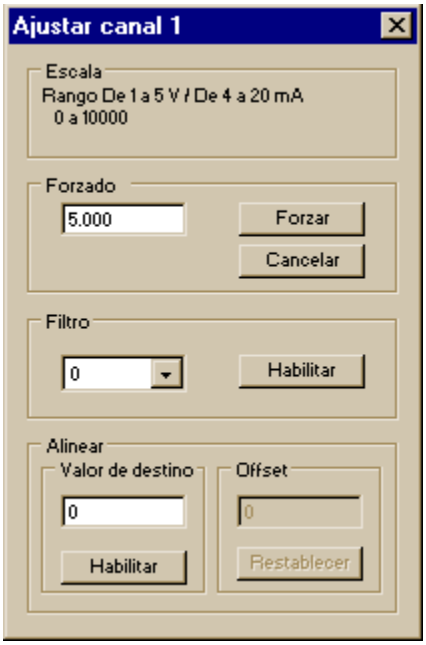


En la imagen %MW10 que es una dirección Modbus análoga, tipo entero, podemos cambiarle su valor, pero no podemos forzarla a mantener el valor.

Siempre que hay variables forzadas (*en el programa*), se muestra un **F** de color rojo en la barra de estado, abajo a la izquierda.

Al hacer click derecho sobra la **F** muestra una tabla con todo lo que esta forzado.

Las variables **análogas** no se pueden forzar (*en el programa*), aunque hay un truco para forzar una entrada análoga en el modulo de entrada, en vez de en el programa. Si embargo, esta opción es peligrosa, por que en ese caso la barra de estado no muestra que hay algo forzado en el programa, (es como forzar en el mismo instrumento), y podríamos no darnos cuenta de que la señal no es real.



En el explorador del proyecto, abrimos configuración, desplegamos el árbol del hardware, buscamos el modulo de entrada análoga y lo abrimos. Al estar en linea, además de la pestaña configuración, aparece otra pestaña "Depuración". Que muestra una tabla con el estado de todos los canales de esta tarjeta.

Hacemos doble clik sobre la columna F de la linea del canal que nos interesa, y aparece el cuadro … *(ver imagen)*

El valor que ingresamos debe ser un entero, por que es en el rango (en cuentas) del conversor análogo digital. En la imagen ingresamos 5000 para obligar a este canal a mantenerse en 50,00 %

Persistencia de estados y valores.

Muchos creen, que al apagar y volver a encender el plc, todos los estados, y valores, vuelven a comenzar en cero. Esta mito, totalmente falso, se origina por que muchos, acostumbrados a programar en Ladder, un lenguaje que dibuja circuitos eléctricos, creen que el programa se comportara como un circuitos eléctrico de relés al cortar la alimentación eléctrica.
En realidad el lenguaje "Ladder" no es mas que la representación gráfica del lenguaje "Lista de instrucciones".

Los controladores programables (PLC, PAC) cuando se apagan y vuelven a encender, no solo conservan el programa que tenían, si no que también conservan, todos los estados discretos y valores análogos que tenían, antes de apagarse. Lo que estaba andando, continuará andando. La secuencia que iba a medio camino, seguirá a medio camino.

Este comportamiento puede ser muy peligroso, si es un controlador de un sistema de seguridad, es por esto que algunos pocos equipos especialmente diseñados para seguridad, en su partida borran todos sus estados y valores. Pero la gran mayoría de los plc, de todas las marcas y fabricantes, no lo hace automáticamente.

No gustaría que en un reinicio del controlador, algunos valores, como los setting de alarmas se mantengan, pero que otros valores, como los estados de la lógica se reinicien, etc. para que el proceso se reinicie en forma segura.

Como controlar que se reinicia y que se conserva:

En todos los plc o PAC, de distintos fabricantes, existen muchas variables de sistema, estas son señales especiales, de diagnostico del sistema, en los Modicon, por ejemplo:

%S0	coldstart	Reinicio en frio (apagar → encender)
%S1	warmstart	Reinicio en caliente (Run → Stop → Run)
%S13	1rstscanrun	Primer ciclo después de cambiar a Run

Usando estas señales podemos definir que variables deben reiniciarse a 0 y cuales deben reiniciarse a un valor determinado.

Por ejemplo, supongamos que debemos asegurar que siempre parta en estas condiciones:

SD Rele Shutdown en apagado.
TSH Setting alta temp en 120 °C

En las secciones del programa, agregamos una sección de nombre "reinicio", y la desplazamos al primer lugar, para que sea la primera en ejecutarse. Y dentro de "reinicio" ponemos:

```
If %S13 then
    SD := False;
    TSH := 120.0;
end_if;
```

Lo que esta entre "If" y "end_if" **solo** se ejecuta una vez, cuando el controlador cambia su estado a Run. Esto se conoce como firstscan.

Capitulo 3:

Desarrollo de un ejercicio mas completo, aplicando todos los conceptos anteriores. Clases, estructuras, funciones, etc.

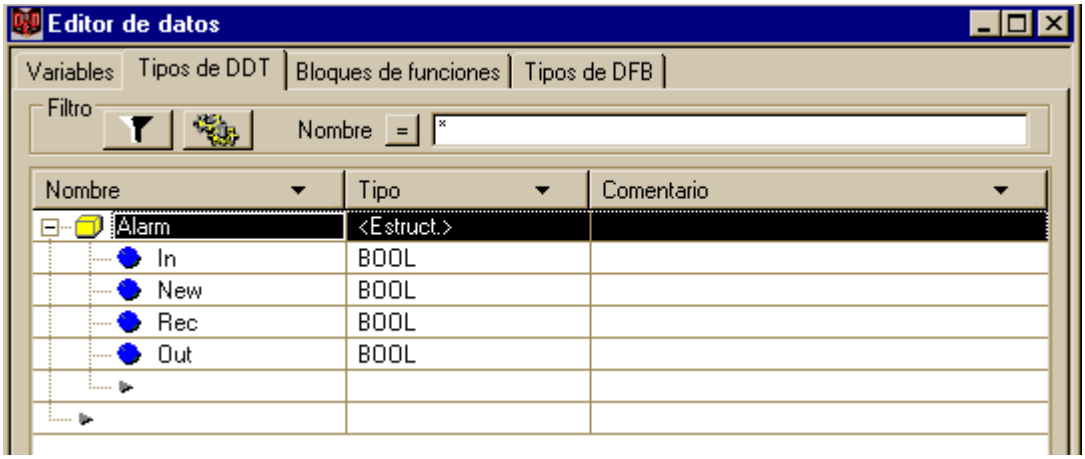
Vamos a crear un nuevo proyecto, con el mismo hardware que en el ejercicio anterior, tal como se enseño en el capitulo 2. Pero este va a tener otros elementos distintos y otros programas distintos.
Si al mismo ejercio anterior, le haces un guardar como … ya tienes la mitad del nuevo proyecto. Borra todo lo que hay en la carpeta secciones de programa, y también borra todo lo que hay en la tabla de variables. Y dile regenerar todo.
Ahora podemos comenzar con el ejercicio 2.

El problema a resolver:
Vamos a usar 8 entradas discretas, 4 conectadas a botones pulsadores y 4 conectadas a interruptores, y también 4 entradas análogas.
Vamos a usar solo 1 salida discreta, el relé "shutdown".
Vamos a tener muchas alarmas, y cada vez que se apaga el relé "shutdown", queremos saber cual de las alarmas fue la culpable de la caída del relé.

Como tenemos muchas alarmas, que deben comportarse todas iguales, vamos a crear una estructura de datos (un tipo DDT) de nombre "alarm". Y luego vamos a crear un arreglo de esta estructura "alarm". Entonces podemos usar un Loop For, para que todas las alarmas usen la misma lógica.

Crear una estructura:

Abre la tabla de variables, y selecciona la **pestaña Tipos DDT** ….

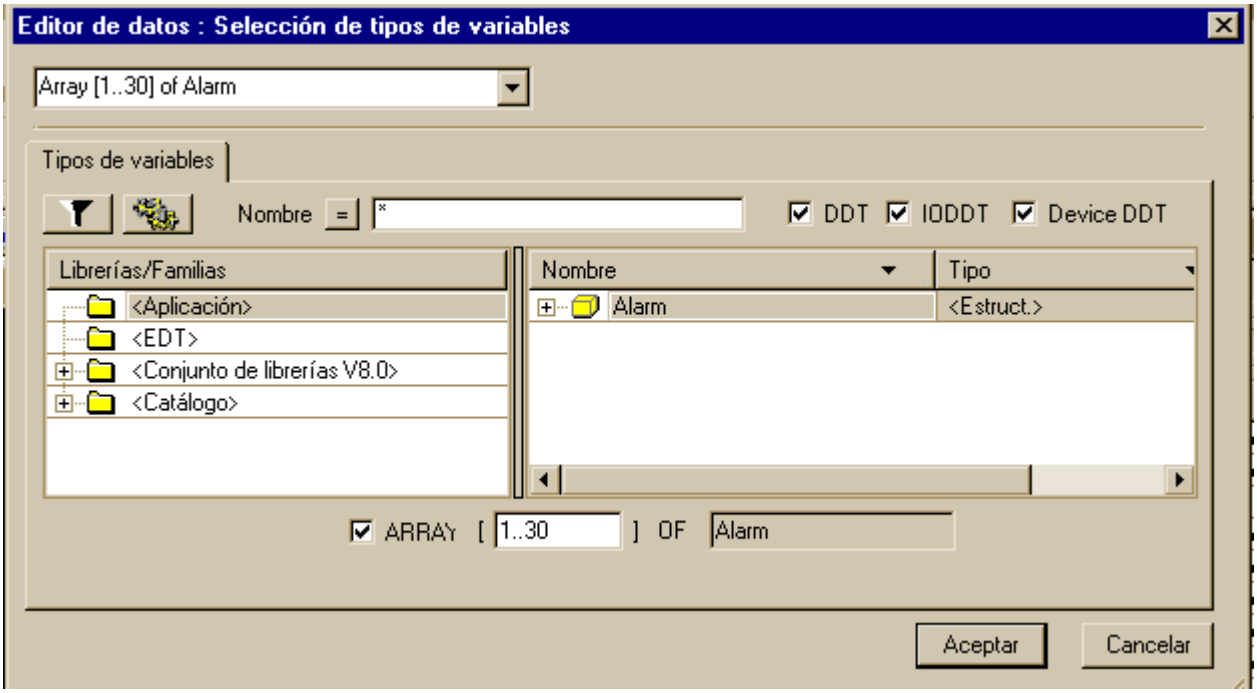


Y crea la **estructura** "Alarm" que se muestra en la imagen.

En la tabla de variables, crea las siguientes:

Nombre	Tipo	Dirección	Comentario
AlarmResumen	Bool		Resumen alarmas
AlarmNueva	Bool		Alarma sin reconocer
Reconocer	Bool		Reconocer las alarmas
i	Int		indice
Alarma	**Array[1..30] Of Alarm**		

Para **crear el arreglo** de "Alarmas", en la columna tipo, toca en el botón  para abrir una ventana que muestra todos los tipos disponibles (ver imagen). Selecciona la carpeta "Aplicación" para ver los tipos de datos definidos en la misma aplicación, y ahí esta la estructura, que recién creaste.



En la parte inferior **marca donde dice ARRAY**, y define el tamaño, 1..30, para que sea un arreglo, de ese tipo.

Ya creaste rápidamente 120 variables booleanas que se necesitan para la lógica de las 30 alarmas, ordenadas en un arreglo.

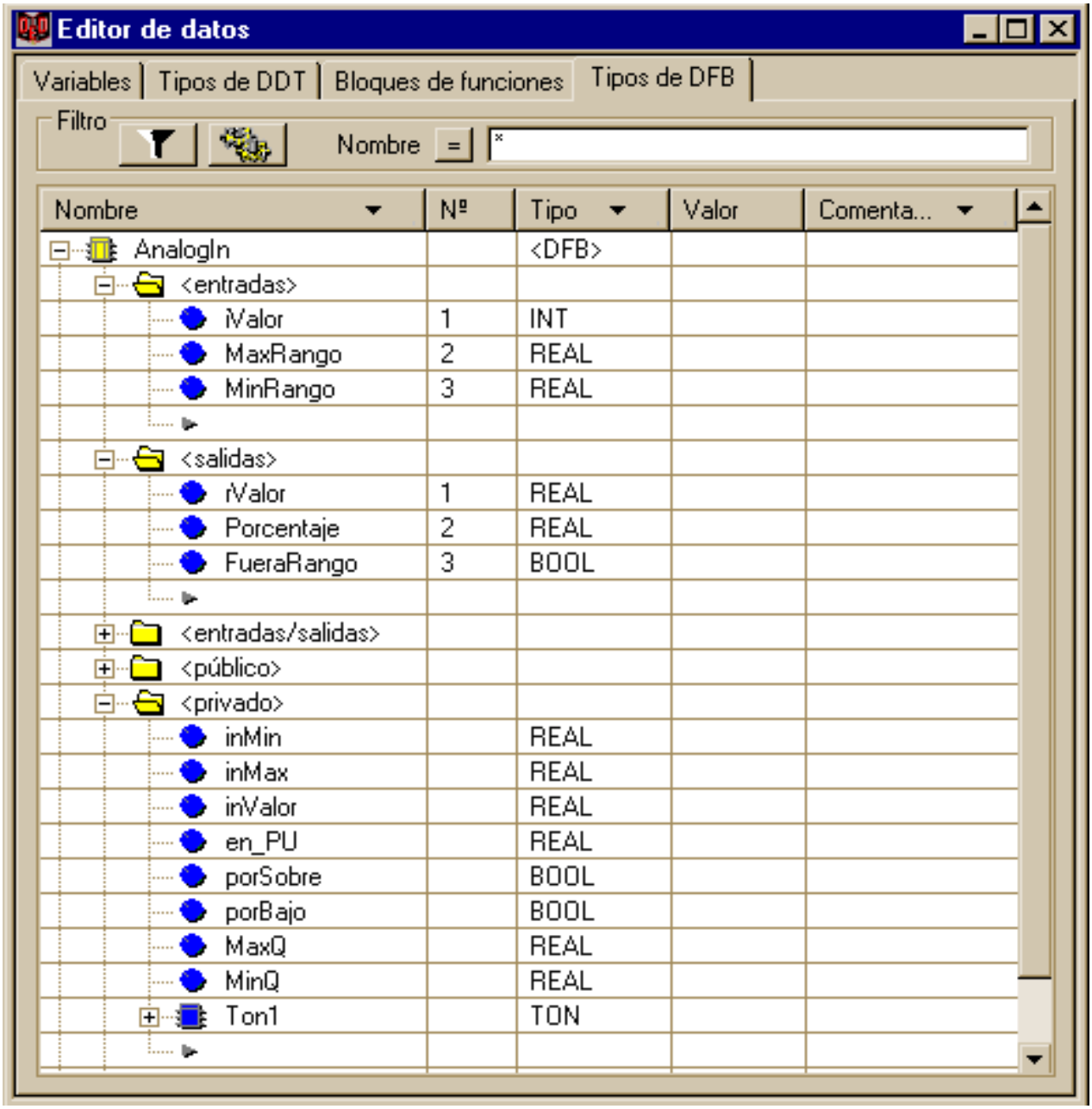
Ahora continua agregando las demás variables

Nombre	Tipo	Dirección	Comentario
BotRecon	Ebool	%I0.2.0	Boton Reconocer alarmas
BotReset	Ebool	%I0.2.1	Boton pulsador Partir
PararLoc	Ebool	%I0.2.2	Boton pulsador parar local
PararRem	Ebool	%I0.2.3	Bboton pulsador parada remota
switch1	Ebool	%I0.2.8	Interruptor 1
switch2	Ebool	%I0.2.9	Interruptor 2
switch3	Ebool	%I0.2.10	Interruptor 3
switch4	Ebool	%I0.2.11	Interruptor 4
ReleSD	Ebool	%Q0.4.2	Rele Shutdown
Nivel	Real		Valor en %
Presion	Real		Valor en K/cm2
Temp	Real		Valor en °C
Flujo	Real		Valor en mtr3/h
Nivel_H	Bool		Alarma alto
Nivel_HH	Bool		Alarma muy alto
Nivel_Bad	Bool		Alarma fuera de rango
Presion_L	Bool		Alarma bajo
Presion_LL	Bool		Alarma muy bajo
Presion_Bad	Bool		Alarma fuera de rango
Temp_H	Bool		Alarma alto
Temp_HH	Bool		Alarma muy alto
Temp_Bad	Bool		Alarma fuera de rango
Flujo_L	Bool		Alarma bajo
Flujo_LL	Bool		Alarma muy bajo
Flujo_Bad	Bool		Alarma fuera de rango

Crear un clase:

Como tenemos varias señales análogas, y cada una hay que evaluarla, y llevarla a un rango de ingeniería, vamos a crear una clase "AnalogIn", que haga el mismo trabajo igual para cada una.

En la tabla de variables, selecciona la **pestaña tipos DFB**, y crea una de nombre "AnalogIn", que tenga la siguiente estructura.



Nombre	Nº	Tipo	Valor	Comenta...
AnalogIn		<DFB>		
<entradas>				
iValor	1	INT		
MaxRango	2	REAL		
MinRango	3	REAL		
<salidas>				
rValor	1	REAL		
Porcentaje	2	REAL		
FueraRango	3	BOOL		
<entradas/salidas>				
<público>				
<privado>				
inMin		REAL		
inMax		REAL		
inValor		REAL		
en_PU		REAL		
porSobre		BOOL		
porBajo		BOOL		
MaxQ		REAL		
MinQ		REAL		
Ton1		TON		

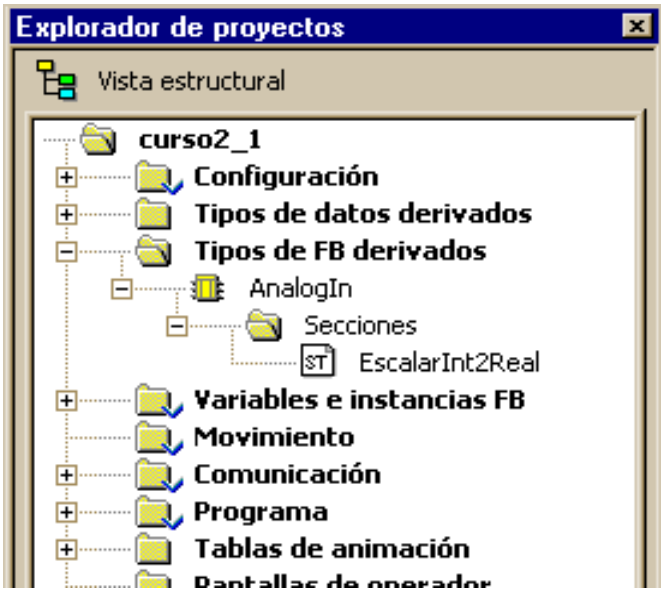
En la carpeta "entrada" los datos que recibirá: el valor en cuentas desde una entrada análoga, y el rango que se le va a asignar.

En la carpeta “salida” los datos que devuelve: el valor en el rango, el valor en porcentaje, y la indicación de que esta fuera de rango.

En la carpeta “privado”: las variables internas que usa su código interno.

Programar el código de la clase:

Si vemos ahora en el explorador del proyecto, apareció dentro de la carpeta **Tipos de FB derivados**, la nueva clase “AnalogIn” que creamos.



Dentro de la carpeta Secciones, de esta clase AnalogIn, crea una nueva sección de nombre “EscalarInt2Real” en lenguaje ST

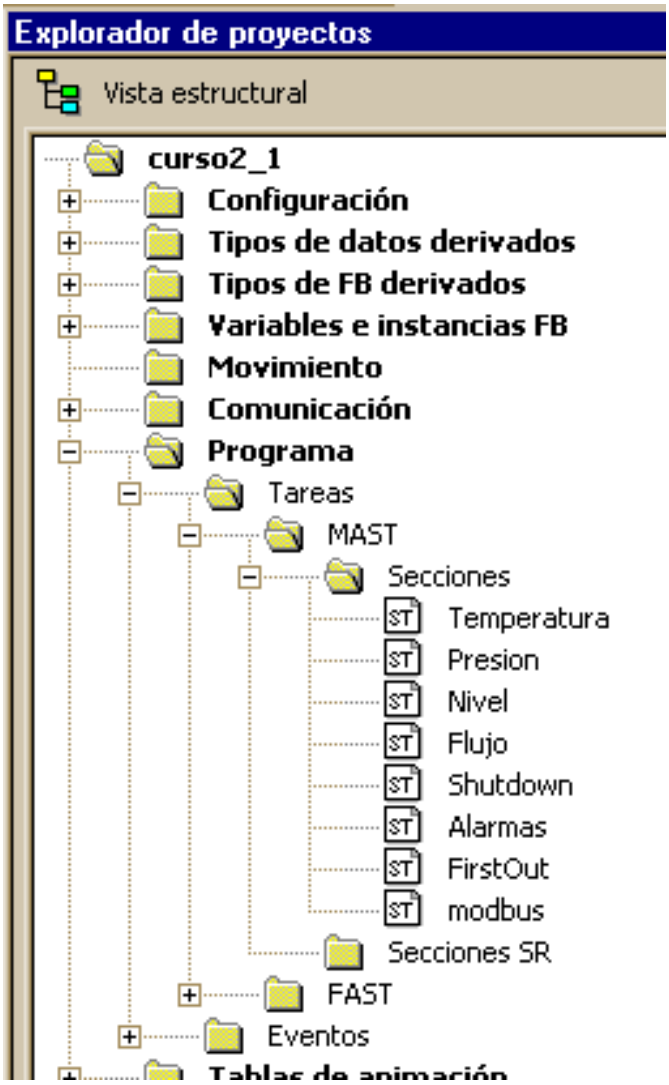
Ahora vamos a ingresar el código dentro de esta sección, que es la lógica que se ejecutara para todas las señales análogas.
Usando las variables “privadas” de esta clase.

```
(*Escala de valor entero a valor real *)

(* rango entrada:  de 0 a 10000  *)
inMin := 0.0;          (*  4 mA *)
inMax := 10000.0;      (*  20 mA *)

(* valores mayores a 32766 son valores negativos,
 cuando hay menos de  4 mA *)
If  iValor < 32766 then
  inValor := INT_TO_REAL (IN := iValor);
Else
  inValor := INT_TO_REAL (IN := ( iValor - 32767));
end_if;

(* para prevenir division por cero *)
If  (MaxRango - MinRango) =  0.0  then
```

```
  MinQ := 0.0;          MaxQ := 100.0;
Else
  MinQ := MinRango;        MaxQ := MaxRango;
end_if;

(* en_PU = en Por Unidad *)
en_PU := (inValor - inMin) / (inMax - inMin);

Porcentaje := en_PU * 100.0;

rValor := (((MaxQ - MinQ) * en_PU) + MinQ);

(* si esta mas de 103 % del rango *)
If  en_PU > 1.03  then  porSobre := True;   end_if;
If  en_PU < 1.0  then   porSobre := False;   end_if;

(* si esta menos de -5 % del rango *)
If  en_PU < -0.05  then  porBajo := True;   end_if;
If  en_PU > 0.0  then   porBajo := False;   end_if;

(* Si esta por sobre el rango, o por bajo el rango,
 por mas de 5 segundos, entonces activar alarma 'Fuera de Rango' *)
Ton1 (IN := (porSobre OR porBajo), PT := T#5s, Q => FueraRango);
```

Secciones del programa principal:

En el explorador del proyecto, en la carpeta programa, dentro de la tarea “Mast” (master) crear las secciones, tipo ST, ordenadas como se muestra en la imagen.



Tenemos que crear un elementos de la clase "AnalogIn", para cada señal análoga conectada.

En el editor de datos, ir a la **pestaña "Bloques de funciones"** y crear:

Nombre	Tipo	Comentario
AnalogIn_0	AnalogIn	para señal de canal 0
AnalogIn_1	AnalogIn	para señal de canal 1
AnalogIn_2	AnalogIn	para señal de canal 2
AnalogIn_3	AnalogIn	para señal de canal 3
AnalogIn_4	AnalogIn	para señal de canal 4
AnalogIn_5	AnalogIn	para señal de canal 5
AnalogIn_6	AnalogIn	para señal de canal 6
AnalogIn_7	AnalogIn	para señal de canal 7

Ahora veremos lo simple que quedan los programa al usar estos "bloque de función" AnalogIn.

Programa "Temperatura"

```
AnalogIn_0 ( iValor := %IW0.3.0 ,    (* modulo 3 canal 0 *)
        MaxRango := 150.0 ,     (* grados Celsius *)
        MinRango := 0.0 ,
        rValor => Temp ,
        FueraRango => Temp_Bad ) ;
(* muy alta  temperatura *)
If Temp > 130.0 then Temp_HH := True;  end_if ;
If Temp < 127.0 then Temp_HH := False;  end_if ;
(* alta  temperatura *)
If Temp > 110.0 then Temp_H := True;  end_if ;
If Temp < 107.0 then Temp_H := False;  end_if ;
```

Programa "Presion"

```
AnalogIn_2 ( iValor := %IW0.3.2 ,
        MaxRango := 5.0 ,      (* K/cm2 *)
        MinRango := 0.0 ,
        rValor => Presion ,
        FueraRango => Presion_Bad ) ;
(* baja  presion *)
If Presion < 2.0 then Presion_L := True;  end_if ;
If Presion > 2.15 then Presion_L := False;  end_if ;
(* muy baja  presion *)
If Presion < 1.5 then Presion_LL := True;  end_if ;
If Presion > 1.65 then Presion_LL := False;  end_if ;
```

Programa "Nivel"

```
AnalogIn_4 ( iValor := %IW0.3.4 ,
        MaxRango := 100.0 ,    (* % *)
        MinRango := 0.0 ,
        rValor => Nivel ,
        FueraRango => Nivel_Bad ) ;
(* muy alto  nivel *)
If Nivel > 78.0 then Nivel_HH := True;  end_if ;
If Nivel < 74.5 then Nivel_HH := False;  end_if ;
(* alto  nivel *)
If Nivel > 45.0 then Nivel_H := True;  end_if ;
If Nivel < 41.5 then Nivel_H := False;  end_if ;
```

Programa "Flujo"

```
AnalogIn_6 ( iValor := %IW0.3.6 ,
        MaxRango := 2750.0 ,
        MinRango := 0.0 ,
        rValor => Flujo ,
        FueraRango => Flujo_Bad ) ;
(* bajo  flujo *)
If Flujo < 320.0 then Flujo_L := True ;  end_if ;
If Flujo > 345.0 then Flujo_L := False ;  end_if ;
(* muy bajo  flujo *)
If Flujo < 200.0 then Flujo_LL := True ;  end_if ;
If Flujo > 230.0 then Flujo_LL := False ;  end_if ;
```

Programa "Shutdown"

```
ReleSD := ( BotReset  or  ReleSD )
   (* condiciones que apagan el rele: *)
  and  not  PararLoc      (* boton parada local *)
  and  not  PararRem      (* boton parada remota *)
  and  not  Temp_HH       (* muy alta temperatura *)
  and  not  Presion_LL      (* muy baja presion *)
  and  not  Nivel_HH      (* muy alto nivel *)
  and  not  Flujo_LL      (* muy bajo flujo *)
  and  switch1      (* interruptor 1 abierto *)
  and  switch2      (* interruptor 2 abierto *)
  and  switch3      (* interruptor 3 abierto *)
  and  switch4 ;      (* interruptor 4 abierto *)
```

Programa “Alarmas”

Asignamos la condición que dispara cada alarma.
Las que no se usan, están deshabilitadas asignándole False.
Y luego esta la lógica para todas las alarmas.

```
Alarma[1].In := PararLoc ;     (* boton parada local *)
Alarma[2].In := PararRem ;    (* boton parada remota *)
Alarma[3].In := Temp_H ;      (* alta temperatura *)
Alarma[4].In := Temp_HH ;     (* muy alta temperatura *)
Alarma[5].In := Temp_Bad ;    (* temperatura fuera de rango *)
Alarma[6].In := Presion_L ;    (* baja presion *)
Alarma[7].In := Presion_LL ;   (* muy baja presion *)
Alarma[8].In := Presion_Bad ;   (* presion fuera de rango *)
Alarma[9].In := Nivel_H ;      (* alto nivel *)
Alarma[10].In := Nivel_HH ;    (* muy alto nivel *)
Alarma[11].In := Nivel_Bad ;   (* nivel fuera de rango *)
Alarma[12].In := Flujo_L ;     (* bajo flujo *)
Alarma[13].In := Flujo_LL ;    (* muy bajo flujo *)
Alarma[14].In := Flujo_Bad ;    (* flujo fuera de rango *)
Alarma[15].In := not switch1 ;   (* interruptor 1 abierto *)
Alarma[16].In := not switch2 ;   (* interruptor 2 abierto *)
Alarma[17].In := not switch3 ;   (* interruptor 3 abierto *)
Alarma[18].In := not switch4 ;   (* interruptor 4 abierto *)
Alarma[19].In := not ReleSD ;   (* rele shutdown apagado *)
Alarma[20].In := False ;
Alarma[21].In := False ;
Alarma[22].In := False ;
Alarma[23].In := False ;
Alarma[24].In := False ;
Alarma[25].In := False ;
Alarma[26].In := False ;
Alarma[27].In := False ;
Alarma[28].In := False ;
```

```
 Alarma[29].In := False ;
 Alarma[30].In := False ;
(*
 Alarma[x].In   <--  condicion que dispara la alarma
 Alarma[x].New   -->  que no ha sido reconocida aun.
 Alarma[x].Rec   -->  ya fue reconocida.
 Alarma[x].Out   -->  alarma esta presente.

_____________________________________________
 Logica para todas las alarmas:*)
AlarmNueva := False ;
AlarmResumen := False ;
Reconocer := ( BotRecon  or  %M500 ) ;
For  i  :=  1  to  30  do
   If (Alarma[i].In  or  Alarma[i].New)  and  not  Alarma[i].Rec  then
      Alarma[i].New := True ;
   else
      Alarma[i].New := False ;
   end_if;
   If Alarma[i].New  then  AlarmNueva := True;  end_if;
   If (Alarma[i].In  or  Alarma[i].New)
   and  (Reconocer  or  Alarma[i].Rec)  then
      Alarma[i].Rec := True;
   else
      Alarma[i].Rec := False;
   end_if;
   If (Alarma[i].New  or  Alarma[i].Rec)  then
      Alarma[i].Out := True;
   else
      Alarma[i].Out := False;
   end_if;
   If Alarma[i].Out  then  AlarmResumen := True;  end_if;
End_for ;
%M500 := False;
(*_____________________________________________*)
```

```
%Q0.4.5 := AlarmNueva;    (* Rele que activa bocina de alarma *)
%Q0.4.7 := AlarmResumen;    (* Rele resumen de alarma *)
```

Las salida**s** Alarma[x]**.**Out muestran cuales alarmas están activada. Estas se asignan a una dirección Modbus, para poder ser leídas por un DCS, o pantalla local.
La dirección %M500 (Modbus coil 0:0501) permite reconocer las alarmas desde la pantalla local.

Programa "FirstOut"

Si queremos registrar el motivo de la caída del rele, inmediatamente **después de** las secciones "Shutdown" y "Alarmas", debe ejecutarse esta sección "FirstOut".

```
(* Registrar causa de caida del rele *)
 If not ReleSD and not grabado then;
   (* cual de condiciones que apagan el rele estan presentes: *)
    %M100 := PararLoc;    (* boton parada local *)
    %M101 := PararRem;    (* boton parada remota *)
    %M102 := Temp_HH;     (* muy alta temperatura *)
    %M103 := Presion_LL;  (* muy baja presion *)
    %M104 := Nivel_HH;    (* muy alto nivel *)
    %M105 := Flujo_LL;    (* muy bajo flujo *)
    %M106 := not switch1; (* interruptor 1 abierto *)
    %M107 := not switch2; (* interruptor 2 abierto *)
    %M108 := not switch3; (* interruptor 3 abierto *)
    %M109 := not switch4; (* interruptor 4 abierto *)
    %M110 := False;
    %M111 := False;
    %M112 := False;
    %M113 := False;
    %M114 := False;
```

```
   %M115 := False;

  (* desplazar historia, olvidando el mas antiguo *)
   %MW19:= %MW18; %MW18:= %MW17; %MW17:= %MW16;
   %MW16:= %MW15; %MW15:= %MW14; %MW14:= %MW13;
   %MW13:= %MW12; %MW12:= %MW11; %MW11:= %MW10;
  (* poner en historia el ultimo *)
   %MW10.0 := %M100;    %MW10.8 := %M108;
   %MW10.1 := %M101;    %MW10.9 := %M109;
   %MW10.2 := %M102;    %MW10.10 := %M110;
   %MW10.3 := %M103;    %MW10.11 := %M111;
   %MW10.4 := %M104;    %MW10.12 := %M112;
   %MW10.5 := %M105;    %MW10.13 := %M113;
   %MW10.6 := %M106;    %MW10.14 := %M114;
   %MW10.7 := %M107;    %MW10.15 := %M115;
   grabado:= True;
end_if;

If ReleSD then
   grabado:= False;
end_if;
```

Estados %M100 al %M110 son motivo de ultima caída del relé.

Análogos %MW10 al %MW19 son historia de últimos motivos. Donde cada bit, de cada %MW representa las causas en ese evento.

(ejemplo %MW10**.**3 es bit 3 de análoga %M10)

Capitulo 4:

Un controlador no es un ente aislado, tiene que comunicarse con sus ayudantes y también tiene que responder a sus jefes. Para esto tiene sus distintos puertos de comunicación.
El mismo controlador (plc) puede ser "esclavo" (servidor) de otros sistemas superiores. Y al mismo tiempo puede ser "maestro" (cliente) de otros controladores subordinados.

Un controlador (CPU) M340 tiene varios puertos de comunicación. La desventaja de usar un puerto de comunicaciones del mismo controlador (CPU) es que estos puertos **no** son aislados. Deberían usarse solo para comunicar con otros dispositivos en el mismo lugar (monitor Bently, controlador Woodward, etc). Si usamos estos para comunicar con dispositivos remotos, como un DCS, la red de la planta, etc, cualquier diferencia de voltajes entre la conexión a tierra local y la conexión a tierra remota podría quemar el controlador (CPU).

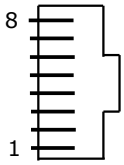
Podemos agregar módulos de comunicación, con mas puertos, en cualquier ranura del rack.

Comunicación serial como esclavo

Cuando se actúa como esclavo, basta solo con configurar los parámetros de conexión, y tener los datos en la tabla de direcciones. Todos los datos con dirección %Mx y %MWx podrán ser lee idos, o escritos por comunicación.

Continuando con el ejercicio anterior, ir a explorador de proyecto → Configuración, y agregarle, en la ranura 6 un modulo NOM200
El **modulo de comunicación NOM-0200** tiene 2 puertos seriales aislados. El primero puede ser RS232 o puede ser RS485, el segundo puerto siempre es RS485.

Conector RJ45 en NOM0200



RS485:
pin 4 ---- Data 0
pin 5 ---- Data 1
pin 8 ---- com 0v

Los puertos seriales Modicon usan, el mismo conector RJ45 de ocho pines, que los puertos Ethernet. Para distinguirlos, los que tienen una marca color verde son Ethernet, y los que tienen una marca color negro son seriales.

Abre la configuración del modulo NOM200. Y selecciona **canal 0**, y configúralo como:

Función = conexión Modbus.
Modo = **Esclavo**

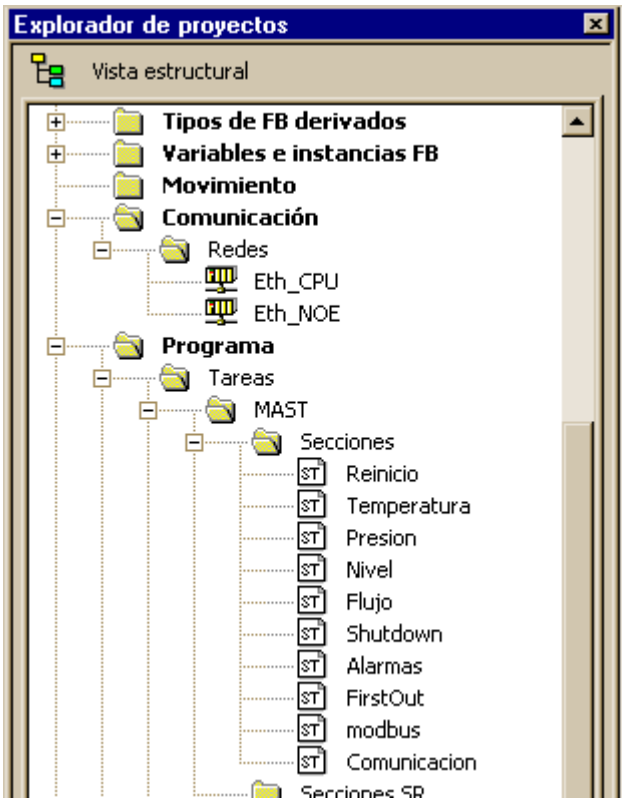
Esclavo numero = **2**
Linea física = RS485
Velocidad = 19200 bit/s
Datos = RTU (8 bits)
Parada = 1 bits
Paridad = Ninguna

selecciona **canal 1**, y configúralo como:
Función = conexión Modbus.
Modo = **Maestro**
Velocidad = 19200 bit/s
Datos = RTU (8 bits)
Parada = 1 bits
Paridad = Ninguna

Configurar una red Ethernet

Continuando con el ejercicio anterior, ir a Explorador de proyecto, Configuración, y agregarle, en la ranura 7 un modulo NOE0100
El **Modulo de comunicación _NOE-0100** tiene 1 puertos Ethernet aislado.

Ahora en el Explorador del proyecto, dentro de la carpeta Comunicación, en la carpeta Redes, crear nueva red:
Tipo = Ethernet
Nombre = "**Eth_CPU"**
Configura esta como
Modelo = CPU2030
Ip = **192.168.0.200**



Crear también otra red:
Tipo = Ethernet
Nombre = "**Eth_NOE"**
Configura esta como:
Modelo = NOE0100
Ip = **192.168.0.201**

Abre la configuración de la CPU, y en su puerto Ethernet, poner:
Función = Eth Tcp Ip
Conexión = "Eth_CPU"

Abre la configuración del modulo NOE, poner:
Función = Eth Tcp Ip
Conexión = "Eth_NOE"

Ahora solo falta disponer en una tabla, todos los datos en las direcciones.

Todas las direcciones %Mx y %MWx podrán ser lee idos, o escritos por comunicación.

Agrega una nueva sección al final del programa.

Programa "modbus"

```
(* Asigna valores a direcciones
 que pueden ser leidas por comunicacion *)

(* estados de condiciones actuales *)
%M0 := ReleSD;      (* rele shutdown *)
%M1 := PararLoc;      (* boton parada local *)
%M2 := PararRem;     (* boton parada remota *)
%M3 := Temp_H;      (* alta temperatura *)
%M4 := Temp_HH;     (* muy alta temperatura *)
%M5 := Presion_L;      (* baja presion *)
%M6 := Presion_LL;      (* muy baja presion *)
%M7 := Nivel_H;     (* alto nivel *)
%M8 := Nivel_HH;       (* muy alto nivel *)
%M9 := Flujo_L;      (* bajo flujo *)
%M10 := Flujo_LL;     (* muy bajo flujo *)
%M11 := not switch1;      (* interruptor 1 abierto *)
%M12 := not switch2;       (* interruptor 2 abierto *)
%M13 := not switch3;       (* interruptor 3 abierto *)
%M14 := not switch4;       (* interruptor 4 abierto *)
%M15 := False;
%M16 := False;
%M17 := False;
%M18 := Com2ok;      (* estado comunicacion serial *)
%M19 := False;
```

```
(* estados de alarmas *)
%M20 := AlarmNueva;      (* Existe alarma sin reconocer *)
%M21 := Alarma[1].Out;      (* boton parada local *)
%M22 := Alarma[2].Out;      (* boton parada remota *)
%M23 := Alarma[3].Out;      (* alta temperatura *)
%M24 := Alarma[4].Out;      (* muy alta temperatura *)
%M25 := Alarma[5].Out;      (* temperatura fuera de rango *)
%M26 := Alarma[6].Out;      (* baja presion *)
%M27 := Alarma[7].Out;      (* muy baja presion *)
%M28 := Alarma[8].Out;       (* presion fuera de rango *)
%M29 := Alarma[9].Out;       (* alto nivel *)
%M30 := Alarma[10].Out;      (* muy alto nivel *)
%M31 := Alarma[11].Out;       (* nivel fuera de rango *)
%M32 := Alarma[12].Out;       (* bajo flujo *)
%M33 := Alarma[13].Out;      (* muy bajo flujo *)
%M34 := Alarma[14].Out;      (* flujo fuera de rango *)
%M35 := Alarma[15].Out;      (* interruptor 1 abierto *)
%M36 := Alarma[16].Out;      (* interruptor 2 abierto *)
%M37 := Alarma[17].Out;      (* interruptor 3 abierto *)
%M38 := Alarma[18].Out;       (* interruptor 4 abierto *)
%M39 := Alarma[19].Out;       (* rele shutdown apagado *)
%M40 := Alarma[20].Out;
%M41 := Alarma[21].Out;
%M42 := Alarma[22].Out;
%M43 := Alarma[23].Out;
%M44 := Alarma[24].Out;
%M45 := Alarma[25].Out;
%M46 := Alarma[26].Out;
%M47 := Alarma[27].Out;
%M48 := Alarma[28].Out;
%M49 := Alarma[29].Out;
%M50 := Alarma[30].Out;
```

```
(* valores analogas, en rango 0 a 10000 *)
%MW0 := %IW0.3.0;       (* Temperatura*)
%MW1 := %IW0.3.1;
%MW2 := %IW0.3.2;      (* Presion *)
%MW3 := %IW0.3.3;
%MW4 := %IW0.3.4;       (* Nivel *)
%MW5 := %IW0.3.5;
%MW6 := %IW0.3.6;      (* Flujo *)
%MW7 := %IW0.3.7;

(* estados %M100 al %M110 son motivos de ultima caida del rele
 analogos %MW10 al %MW19 son historia de ultimos motivos
 en programa FirstOut *)

(* analogos %MW20 al %MW23 son lo leido por comunicacion
 desde el esclavo2,   en programa Comunicacion *)
```

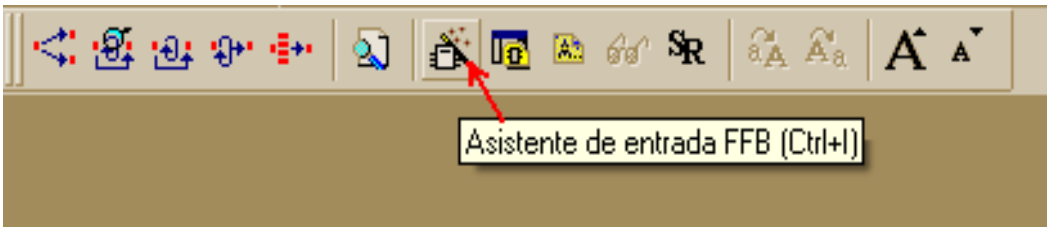
Comunicación serial como maestro

Actuar como "maestro" (o cliente) significa que debemos programar, el control de los mensajes de comunicación. Debemos definir que, y cuando enviar ordenes al "esclavo" (o servidor), desde el que queremos leer datos (o escribir datos).

Dentro de las librerías de clases, ya existen varias que sirven para comunicación, cada una con su descripción y ayuda.

Para leer datos desde otro equipo por un puerto de comunicación, se necesita usar la función **READ-VAR**.

Recuerda el botón "Asistente de entrada FFB" es de gran ayuda para buscar y seleccionar entre las funciones disponibles.



La función **READ-VAR** tiene las siguientes entradas/salidas:

ADR <-- recibe un "arreglo tipo address".
OBJ <-- recibe un string, con el tipo de dato que leer
NUM <-- recibe un entero, el índice del primer dato a leer
NB <-- recibe un entero, cantidad de datos a leer
GEST <--> Un arreglo de 4 enteros de 16 bits (memoria de gestion)
RECP --> entrega una tabla con los datos leidos.

El "arreglo tipo address", lo obtenemos de la funcion ADDM , que recibe un string, con una dirección, y nos devuelve un "arreglo tipo address". Esto se explica en el ejemplo.

Cada vez que se llama a la función READ-VAR, se ejecuta un consulta (solo una), por el puerto de comunicación. Necesitamos saber, si llega una respuesta, o se cumplió el tiempo máximo, y solo entonces volver a llamar a la función READ-VAR para que repita el proceso, ejecutando nuevamente la consulta.

Esto se entiende mejor con un ejemplo, para programar el ejemplo crea las siguientes variables:

Com2ok (tipo Bool) Si comunicación fue tuvo éxito o falló.

Com2Leer (tipo Bool) volver a llamar a READ-VAR cuando termina la comunicación anterior.

Podrían tener cualquier nombre, las llame ..2.. solo por que en este ejemplo se va a leer del esclavo #2.

Esclavo2 (tipo String) para ingresar la dirección como texto.

TablaGestion2 es un arreglo de 4 enteros, ARRAY[0..3] OF INT

Com2Read es un arreglo donde poner los datos que se reciben, por supuesto el tamaño y tipo de dato debe corresponder con lo que se se espera recibir. Para este ejemplo ARRAY[0..3] OF INT.

La función READ-VAR necesita se le asigne un espacio de memoria para trabajar, este es el que llama "**Tabla de gestion**". En la ayuda de este FB, esta explicado el significado de los bits de esta tabla. Todos sus valores se pueden leer, y algunos se pueden escribir, para controlar esta función READ-VAR.

Por ejemplo, el tercer entero de la tabla (TablaGestion**[2]**) es el valor de time-out, o tiempo máximo que espera por una respuesta, en décimas de segundo. Si a TablaGestion**[2]** le escribimos, por ejemplo, 17 el tiempo máximo lo establecemos en 1,7 segundos.

Otro dato interesante es, el primer bit del primer entero de la tabla de gestión (TablaGestion**[0].**0) si esta en True indica que aun se esta trabajando en la comunicación, si esta en False indica que ya se ha terminado la comunicación.

Otro dato interesante es, el primer bit del segundo entero de la tabla de gestión (TablaGestion**[1].**0) si esta en True indica que la comunicación anterior termino en falla o error, si esta en False indica que la comunicación anterior tuvo exito.

Esto se entiende mejor con un ejemplo.
Agrega una nueva sección de programa, al final de la tarea Mast.

Programa "Comunicacion"

```
(* Lee por comunicacion serial como Master-rtu *)

Esclavo2 := '0.6.1.2';  (* <--- desde donde leer *)
(* rack 0, modulo 6, puerto serial 1, hacia esclavo modbus 2

Leer palabras  %MW100 hasta %MW103  desde esclavo2
y copiar lo recibido en Com2Read *)
```

```
if (TablaGestion2[0].0 = False) then
   (* TablaGestion2[0].0 es primer bit del primer entero de la tabla de gestion2,
   si esta en True aun se esta trabajando en la comunicacion,
   si esta en False ya se ha terminado la comunicacion *)
        if (TablaGestion2[1].0 = True) then Com2ok:= False;  end_if;
        if (TablaGestion2[1].0 = False) then Com2ok:= True;  end_if;
       (* TablaGestion2[1].0 es primer bit del segundo entero de la tabla de gestion2,
       si esta en True la comunicacion termino en Error,
       si esta en False la comunicacion termino en Exito *)
      Com2Leer:= True;
else;
   Com2Leer:= False;
end_if;

if Com2Leer then
   TablaGestion2[2]:= 5;   (*timeout = 5 * 100mSeg = 500 mSeg *)
   TablaGestion2[3]:= 0; (* largo en bytes datos resividos *)
   READ_VAR(ADR:= ADDM(Esclavo2),  (* <--- desde donde leer *)
       OBJ:= '%MW', (* tipo Analog Out *)
       NUM:= 100, (* indice del primer dato *)
       NB:= 4, (* cantidad de datos *)
       GEST:= TablaGestion2,
       RECP=> Com2Read ); (* guardar en tabla Com2Read *)
end_if;

(* Nota:  En ajustes del proyecto,  habilitar "permitir matrices dinamicas"  para poder usar
READ_VAR *)

(* Tabla Com2Read es lo recibido por comunicacion *)
%MW20 := Com2Read[0];
%MW21 := Com2Read[1];
%MW22 := Com2Read[2];
%MW23 := Com2Read[3];
```

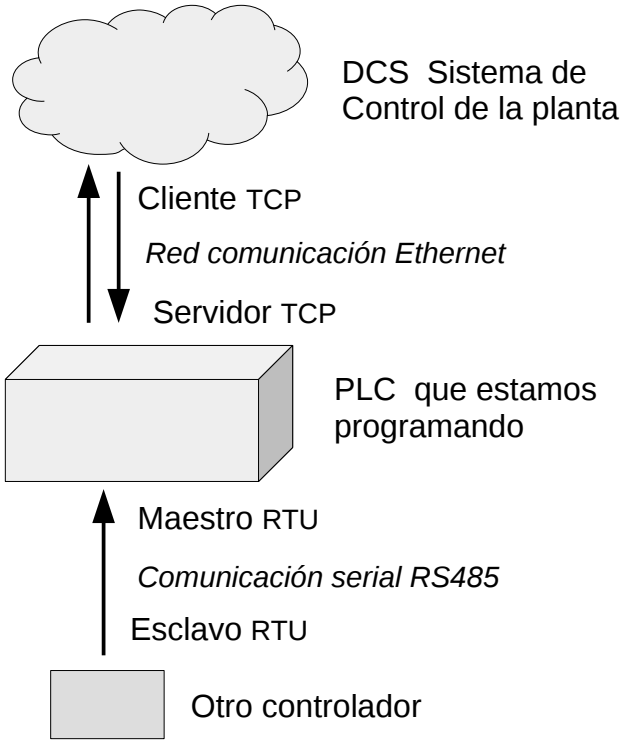
En este ejemplo solo estamos leyendo valores análogos, si también hay que leer estados discretos, habría que hacer una secuencia: primero leer análogos después leer discretos, y solo entonces repetir.

Para facilitar esto, y también para hacer mas eficiente la comunicación, es una practica muy difundida, que el esclavo envíe los estados discretos como los bits de un entero. De esta manera solo habría que leer un paquete de datos enteros, para obtener tanto discretos como análogos.

Un ejemplo real de esto, son los rack 3500 del sistema de monitores de vibraciones **Bently Nevada** de GE.
 Estos solo entregan valores enteros (%MW) donde los valores análogos, como vibracion, gap, setting, etc están en rango 0 a 65535. Y todos los estados de cada canal cambien están en un entero, en el que, bit 0 significa “not ok”, bit 1 significa “alert”, bit 2 significa “danger”, bit 3 significa “bypass”, etc.

Pruebas de comunicación

Para realizar pruebas de comunicación, vamos a usar un esquema típico, como el que se muestra en la siguiente imagen



Conexión Ethernet con el DCS

El PLC que estamos programando, por cualquiera de sus puertos Ethernet, es servidor, esto significa que entrega datos y recibe ordenes desde el DCS.

El DCS se puede simular usando la aplicación "ModbusTest" corriendo en tu propio computador.
Te puedes conectar, por el Ethernet de la cpu (192.168.0.200) o por la ethernet del modulo NOE (192.168.0.201), es igual. Puedes usar un cable normal, conectado directo, o a través de un switch.

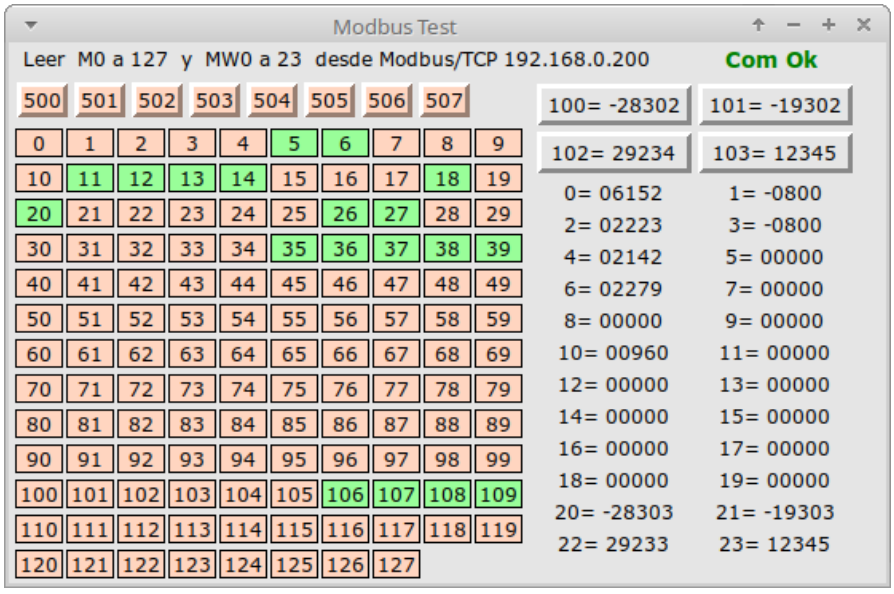
Para probar la comunicación recomiendo usar la aplicación "ModbusTest", que se entrega junto con este curso. En el Anexo (al final) hay mas información sobre las herramientas para este curso, como conseguirlas y como instalarlas.

Este DCS (simulado) va a leer discretos desde el %M0 hasta el %M127, va a leer análogos desde el %MW0 hasta el %MW23. Y también nos permite enviar ordenes (discretos) escribiendo en %M500 y siguientes, o enviar setting escribiendo en %MW100 y siguientes.

El archivo "config.txt" para la aplicación ModbusTest seria:

```
[Modbus]
ip = 192.168.0.200
port = 502
timeout = 0.2

[Digital Read Only]
init = 0
count = 128

[Analog Read Only]
init = 0
count = 24

[Digital Read And Write]
init = 500
count = 8

[Analog Read And Write]
init = 100
count = 4
```

Al ejecutar la aplicación ModbusTest vemos la ventana de la imagen.



Los botones en la parte superior izquierda, son las discretas que se pueden leer y escribir.

Debajo, a la izquierda, las luces que muestran los estados de las discretas que esta leyendo. En rojo si están en 0, en verde si están en 1.

Ve en el programa del plc el significado de cada dirección.

Los botones en la parte superior derecha, son las análogas que se pueden leer y escribir.

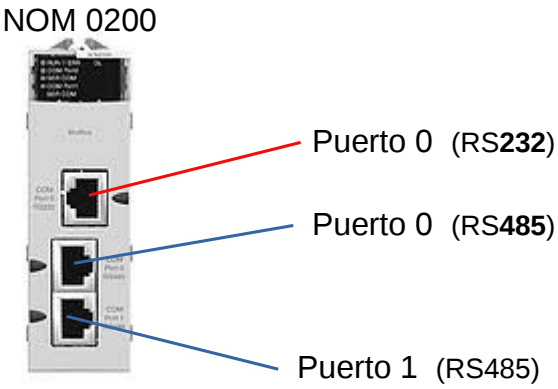
Las listas en la parte inferior derecha son las análogas que se están leyendo.

Comunicación serial maestro - esclavo

Como no tenemos otro controlador para que haga de esclavo, vamos a usar el mismo, también como el esclavo. Aunque esta conexión no tiene ninguna utilidad practica, sirve para probar las comunicaciones.

El canal 0 del modulo NOM0200 es esclavo #2 (responde).

El canal 1 del modulo NOM0200 es maestro (pregunta).



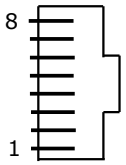
Si corremos el programa, Menú → PLC → Ejecutar, o con el botón Run, vemos en el modulo NOM0200 que el **led "SER COM 1"** esta intermitente. Por que el maestro esta pidiendo datos al esclavo 2, pero como no recibe ninguna respuesta, después de cumplirse el tiempo máximo, vuelve a repetir la pregunta.

Para que el "esclavo 2" tenga algún valor cambiante en los datos que va a enviar, en alguna parte del programa agrega:

```
(*__datos__de__esclavo_#_2_______*)
%MW100 := %SW18 ;
%MW101 := (%SW18 +9000) ;
%MW102 := (%SW18 -8000) ;
```

El maestro lee desde el esclavo # 2, las palabras %MW100 hasta %MW103, y los valores que recibe como respuesta los copia en %MW20 hasta %MW23.

Conector RJ45 en NOM0200



RS485:
pin 4 ---- Data 0
pin 5 ---- Data 1
pin 8 ---- com 0v

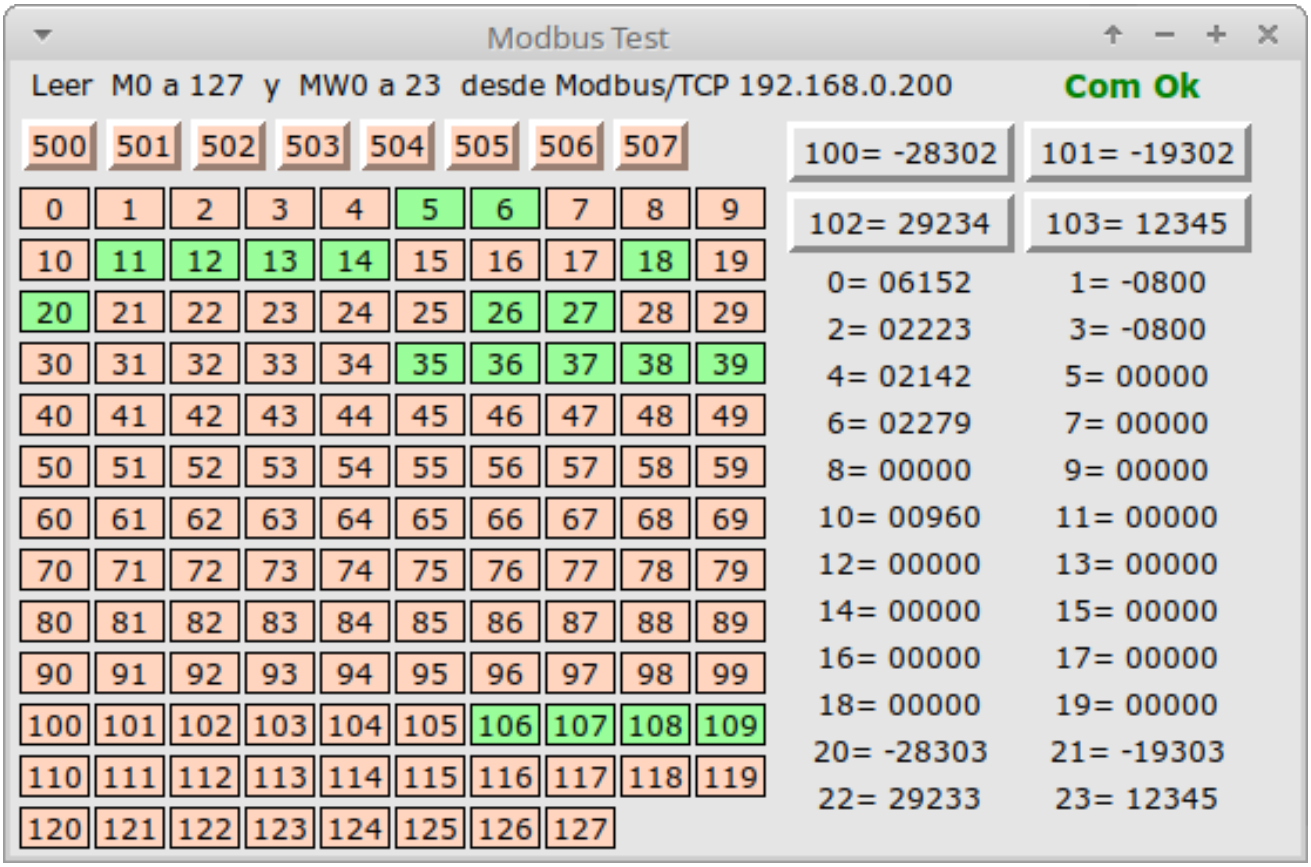
Conecta el pin 4 con el pin 4 del otro puerto, y el pin 5 con el pin 5 del otro puerto RS485. Para que el maestro, pueda comunicarse con el esclavo # 2.

Ahora ambos, led "SER COM **0**" y el led "SER COM **1**" están encendidos, por que uno esta preguntando y el otro esta respondiendo.

Volviendo a la ventana de la aplicación ModbusTest se observa que:

Si esta conectado el cable de comunicación serial, %MW20 hasta %MW23 tiene siempre los mismos valores que %MW100 hasta %MW103. Además esta en 1 (verde) %M18 (comunicación con esclavo 2 ok).
Prueba escribir un valor sobre %MW103, y el mismo valor se copiara hacia %MW23, por la comunicación serial.

Se **des**conectamos el cable de comunicación serial, %MW20 hasta %MW23 mantendrán su ultimo valor, aunque en %MW100 hasta %MW103 los valores continúan cambiando. Además esta en 0 (rojo) %M18 (falla de comunicación con esclavo 2).



Seguridad en la comunicación

Además de las técnicas de filtrado por ip (que se puede configurar en el plc), y de la necesaria separación de redes, cortafuegos, etc hay algunas practicas, muy recomendables, para evitar que el funcionamiento de la lógica sea vulnerable por la comunicación.

Todas las direcciones %Mx y %MWx podrán ser leeidas, o **escritas** por comunicación, por lo tanto es muy importante que las variables que se usan en la lógica no deben tener dirección Modbus.

En el ejemplo anterior, ver programa sección "Shutdown", la variable ReleSD no tiene dirección, para que no pueda ser escrita por comunicación.

En programa sección "Modbus" hay una linea %M0 := ReleSD; para que cualquiera leyendo el valor de la dirección %M0 puede observar el valor actual de ReleSD. Pero si alguien escribe sobre la dirección %M0 no tiene ningún efecto sobre la lógica.

Lo mismo pasa con las análogas, en la linea %MW2 := %IW0.3.2; copiamos el valor de la tercera entrada análoga a la dirección %MW2. La dirección física %IW0.3.2 se usa en la lógica pero no es accesible por comunicación.
Cualquiera leyendo el valor de la dirección %MW2 puede observar el valor actual de esa entrada análoga. Pero si alguien escribe sobre la dirección %MW2 no tiene ningún efecto por que no se usa en la lógica.

Para recibir un comando por comunicación, es una mala practica suponer que el que escribe un valor en una dirección, también va a limpiar esa dirección después.

Por ejemplo, supongamos existe una variable "parar" con una dirección %M314. Si alguien escribe un 1 en la dirección %M314, además de provocar una parada, podría quedar permanentemente con la orden de parar.

En el ejemplo anterior, ver programa sección "Alarmas",

```
Reconocer := ( BotRecon  or  %M500 ) ;
(* …. todo el resto de la logica de las alarmas …   y despues *)
%M500 := False;
```

Si alguien escribe 1 en la dirección %M500, tendrá el mismo efecto que presionar el botón "Reconocer alarmas". Pero inmediatamente después de realizar esto en la lógica, automáticamente la dirección %M500 es limpiada volviéndola a 0.

Entonces no importa si el que envío la orden, olvida volver a escribir 0, la orden "Reconocer alarmas" no puede quedarse permanente y la lógica solo reacciona al cambio desde 0 hacia 1.

Para recibir un setting por comunicación, es una mala practica suponer que el valor que se recibe es valido.

```
If  (%MW123 > 999) and (%MW123 < 3001) then
   SettingH := %MW123 ;
end_if ;
%MW123 := 0;
```

Solo si la dirección %MW123 tiene un valor en el rango valido, entre 1000 y 3000, este valor es aceptado como valor para SettingH, y después la dirección %MW123 es limpiada.
En este ejemplo: la variable "SettingH" no debe tener dirección Modbus.

Capitulo 5:

Scantime & watchdog, conceptos, tipos de tareas, ciclos de ejecución de tareas, errores y problemas comunes.

Watchdog

Todos los controladores programables (PLC), tienen incorporado un mecanismo de seguridad, conocido como watchdog. En palabras simples es un contador de tiempo, que reinicia (apaga) el controlador (PLC) si el tiempo que demora en ejecutar su programación (software) es mayor a un valor máximo. Esta es una protección del sistema, por si el software se bloquea, por algún error de programación, loop infinito, división por cero, etc.
En un PLC esto evita que la maquina o proceso pueda seguir corriendo, sin supervisión (sin protección) si el software se bloquea.

El ajuste de valor de watchdog, debe considerar, el tiempo máximo de ejecución de todas las tareas, en la peor condición, mas un margen de seguridad. Pero también debe considerar cuanto es el máximo tiempo de retardo, admisible para un sistema de seguridad. Por ejemplo: tener un watchdog de 1400 mili Seg, en el control de un horno, es aceptable, un segundo mas o menos no va ha hacer ninguna diferencia significativa. Pero, tener un watchdog de 1400 mili Seg, en un compresor centrifugo, no es bueno, por que en segundo pueden pasar muchas cosas, y puede que el sistema de seguridad, sea demasiado lento para proteger este proceso.

Scan-time

Se denomina scan time al tiempo que demora en ejecutarse una "Tarea". Si el tiempo que demora es mayor, al máximo tiempo permitido, se produce una falla de watchdog. Que provoca un reinicio del procesador (shutdown).

Concepto de "Tarea"

En un controlador programable (PLC), se entiende por "Tarea", todo el ciclo de trabajo que comprende :

1. Leer valores desde las señales que vienen del proceso (presiones, temperaturas, estados, etc). Alimentar el programa con estos nuevos valores.
2. Procesar el programa, recalculando nuevos valores para cada variable, según los nuevos datos.
3. Escribir resultados hacia las señales que van hacia el proceso (cambiar apertura de válvulas, estado de reles y motores, etc)

Cada tarea tiene su propio watchdog.

Interval-time

Se denomina "interval time" al tiempo real, transcurrido entre cada nueva ejecución, de la misma "Tarea".

Si el tiempo real transcurrido entre cada ejecución de una tarea, supera su propio watchdog, se produce un evento alarma de watchdog, aunque el retraso pueda deberse al exceso de tiempo usado por otra "tarea".

Ciclos de ejecución, en controladores monotarea

En controladores antiguos, solo existe una "Tarea" programable, que siempre se ejecuta cíclicamente. Esto es volverla a iniciarla, inmediatamente cuando la misma termine.

En este caso, una detención por falla de watchdog, es cuando el "scan time", de la única "tarea", supera el máximo tiempo permitido, en el ajuste del watchdog.

Ciclos de ejecución, en controladores multitarea

En controladores modernos, "multitarea", el mismo controlador tiene varias "Tareas" programadas, las cuales tienen distintas prioridades, y distintos ciclos de trabajo, y las tareas se turnan entre ellas, compitiendo por el uso del procesador.

Tareas Ciclicas

La tarea del tipo cíclica, es la que vuelve a comenzar, inmediatamente después de que termina. Por definición la tarea cíclica, siempre es la de menor prioridad, por que de otra forma nunca dejaría espacio para las otras tareas.

Por lo mismo, solo puede haber una tarea cíclica, que normalmente se divide en "secciones" de programa.

Las "secciones", de la tarea, ejecutan en un orden determinado, que debe ser definido en el diseño del programa.

La tarea también incluye "subrutinas", que son partes de programa, que solo se ejecutan, si son necesarias, al ser llamadas desde una sección de programa.

El scan-time de la tarea ciclica (**la tarea principal**), puede ser muy variable dependiendo de lo que esta ocurriendo en el proceso. Por ejemplo: si todo esta normal y estable, podría demorar 9 mili Segundos, pero cuando ocurre una perturbación grave, y hay que activar alarmas, registrar eventos, e iniciar acciones de detención. La misma "Tarea" programada podría demorar por ejemplo 100 mili Seg. El ajuste del watchdog, debe dejarse en un valor mucho mas alto, considerando siempre la peor condición.

La mayor parte de la programación debería estar en la "Tarea" cíclica.

Tareas Periódicas

Una tarea del tipo periódica, es la que se inicia cada determinado tiempo. En algunos sistemas se les conoce también como "Fast Task" (tareas rápidas)

Las “Tareas” periódicas existe solo para ejecutar acciones que necesitan un tiempo determinado entre ejecuciones.
Una tarea periódica interrumpe, la ejecución de la tarea principal, y solo después de que esta termina, puede continua la tarea principal. Por esto una tarea periódica, debe ser rápida

Por ejemplo: cada 50 mili Seg solo leer y registrar el valor de la presión. El programa que analiza esas variaciones de presión debe estar en otra tarea.

Puede haber mas de una tarea periódica, con distintos periodos de tiempo. Por eso, una tarea periódica debe tener definida una prioridad, por que podría haber otras tareas periódicas (o por eventos), y también se interrumpen y posponen entre ellas.

Tareas por Eventos

Una tarea por evento, es la que se ejecuta inmediatamente, cuando ocurre un cambio en un señal rápida. Esto es posible solo en sistemas en que exista, módulos de hardware de señales rápidas.
Por ejemplo: Si existiera un módulos de hardware contador, en una cinta transportadora de botellas, esta señal, podría disparar la ejecución de una tarea rápida que solo incrementa la cuenta de botellas que pasan.
Una tarea por evento, debe ser rápida, por que interrumpe y pospone la ejecución de las otras tareas para ejecutarse.
Al igual que a las tareas periódicas una tarea por evento debe tener definida una prioridad, por que, también se interrumpen y posponen con otras tareas.

Problemas comunes

En muchas aplicaciones, la programación del controlador (PLC) tienen problemas de diseño, que provocan fallas, y detenciones no deseadas, por errores de programación (fallas de software), al no entender los conceptos anteriores.
En varias aplicaciones se ha mal entendido el concepto de prioridad, y se a puesto la gran mayoría de la programación, en una tarea periódica (“Fast Task”), en lugar de colocarla en la tarea ciclica (“Main Task”). Por que erróneamente se cree que debe tener mas prioridad. Cuando lo que se debe priorizar es, el orden de ejecución de las secciones, dentro de la tarea cíclica (“Main Task”).

Task Overlap

Fallas de “overlap” (error de traslapo), se produce en una tarea periódica, cuando a la tarea le corresponde iniciarse de nuevo, y no puede hacer lo, por que aun no termina su ejecución anterior.
Es una alarma de watchdog, que puede conducir a una falla de watchdog, y reiniciar el procesador (shutdown, parar el proceso).

Los "overlap" hacen que la tarea se posponga, hasta la siguiente oportunidad, aumentando su "interval time" el tiempo real, transcurrido entre cada nueva ejecución.
Con el peligro, de que si el "interval time" de la tarea, supera el "watchdog" de la misma tarea, se gatilla una falla por watchdog.
Los frecuentes errores de "overlap" se deben a una mala programación, donde se a puesto un gigantesco programa en una tarea que debería ser rápida, y se le exige ejecutarse dentro de un periodo de tiempo acotado.

Por ejemplo: donde todo el calculo de los algoritmos anti-surge se hace en una tarea periódica, en vez de, ejecutar solo la lectura de valores de presión y flujo en la tarea periódica (cada tiempo fijo), y el calculo de los algoritmos anti-surge ejecutarlos en la tarea cíclica (no periódica).

CPU redundantes

Existe el mito, o creencia de que tener CPU redundante protege contra los problemas de software. En realidad no, esta redundancia solo sirve si la falla de la CPU (watchdog) se debe a problemas del hardware, de la electrónica. Ya que si el motivo del watchdog fue el software, la CPU de reserva, que esta sincronizada con la primera, solo repetirá el mismo problema de software, en el otro hardware, y la maquina o proceso igual se detendrá (shutdown).

Anexos:

Herramientas para trabajar, material y programas de ejemplo, para este curso.

Software Unity Pro

Para esta capacitación vamos a usar la versión 8 de "Unity Pro XL".
Esta funciona igual que las versiones mas actuales, para Windows 10, pero cabe en una maquina virtual mas pequeña, que no requiere un súper computador para correr la maquina virtual.

Desde la pagina www.virtualbox.org descarga virtual box.
Instálalo en tu computador.

Descarga esta maquina virtual Modbus8.ova

Abre VirtualBox y en menú → archivo → **Importar servicio virtualizado**.
Selecciona el archivo Modbus8**.**ova que copiaste, y sigue las instrucciones, para crear tu copia de esta maquina virtual.

Abre la maquina virtual "Modbus8" el usuario es Administrador, la contraseña es `password`.

Programas de los ejercicios y ejemplos.

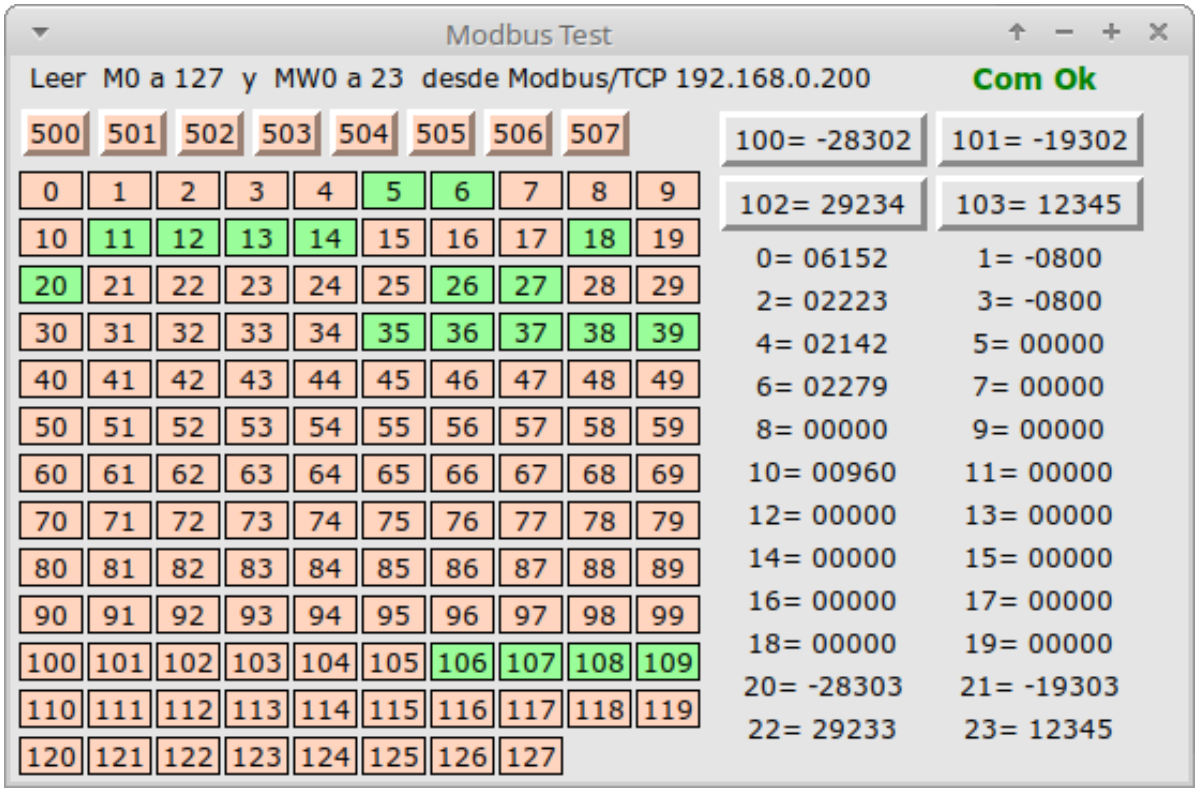
https://drive.google.com/drive/folders/10sVEll3GTozX9C1jK2e-fwgJxSrkydeo?usp=sharing

Software **ModbusTest2**.exe

Esta pequeña aplicación, es una herramienta para probar la comunicación con un servidor Modbus/TCP.

Fue creada solo para fines de prueba, con el curso de capacitacion, de programacion de PLC. Es peligroso usarla para escribir datos hacia un PLC controlando un proceso real; no lo hagas, a menos que estes muy seguro de lo que estas haciendo.

Al ejecutar la aplicación vemos la ventana de la imagen.



En la esquina superior derecha, si se logra comunicar con el PLC dice **Com Ok**, en cambio si no logra comunicación dirá **Sin Com**

Los **botones** en la parte superior izquierda, son las discretas que se pueden leer y escribir.

Debajo, a la izquierda, las luces que muestran los estados de las discretas que esta leyendo. En rojo si están en 0, en verde si están en 1.

Los **botones** en la parte superior derecha, son las análogas que se pueden leer y escribir.

Las listas en la parte inferior derecha son los valores de las direcciones análogas que se están leyendo.

La configuración la lee, cuando se inicia, desde el archivo "config.txt" que debe estar en la misma carpeta sobre la que se esta ejecutando la aplicación.
Esta puede ser cambiada con cualquier editor de texto.

Un ejemplo de "config.txt"

```
[Modbus]
ip = 192.168.0.200
port = 502
timeout = 0.2

[Digital Read Only]
init = 0
count = 128

[Analog Read Only]
init = 0
count = 24

[Digital Read And Write]
init = 500
count = 8

[Analog Read And Write]
init = 100
count = 4
```

¿ Como obtener una copia de esta herramienta ?

Alternativa 1 **el camino del impaciente**, solo quieres usarla, y no te interesa aprender. Descargar desde osdn.net/users/rolfds/pf/ModbusTest/ una versión compilada solo para Windows de 64bit.

Alternativa 2 **el camino del autodidacta**, si además de usarla, quieres aprender como funciona, y usarla en Windows, Linux y Mac por favor continua leyendo lo que sigue después, por que es software libre, y puedes copiarlo, usarlo y modificarlo, bajo los términos de la licencia GNU General Public License, versión 3.
Rolf Dahl-skog Stade. *(Abril 2020)*

ModbusTest2 esta programado en lenguaje python.

¿Que es python? Python es un lenguaje de programación multiplataforma y de código abierto (libre), diseñado para ser fácil de leer.
es.wikipedia.org/wiki/Python

Antes de que te pierdas en muchos vídeos de youtube, que solo te van a confundir, **consejos básicos necesarios** que no aparecen en los muchos cursos que hay en Internet :

1.- Ir a la pagina www.**python**.org y descarga el instalador de la actual versión estable de python **3**. No desde la tienda de aplicaciones de Microsoft.

Importante: NO lo instales en la ruta que propone por defecto windows. Selecciona instalación personalizada, y ponlo en una ruta simple (corta), como por ejemplo en `c:\python` esto te facilitará la vida despues.

2.- Un programa en este lenguaje, es un archivo de texto plano, sin formato. Puede ser creado, o modificado, con cualquier editor de texto, sino fuera por que este lenguaje no usa **;** { } ni ningún otro carácter extraño para delimita las instrucciones, para esto solo usa la indentacion o sangría de las lineas.

Si agregas o eliminas espacios al comienzo de una linea, o cambias espacios por tabulación, o cambias tabulación por espacios, fácilmente puedes dañar el programa. *Esto es motivo de confusión, lo veo igual pero a mi no me funciona.* Por eso es bueno que puedas ver estos caracteres, y cualquier editores de texto no los muestra. Es muy recomendable usar un editor de texto que si muestre los caracteres espacio y tabulación, y muestre la indentacion.

Usa Geany que además tiene resaltado de código, plegado automático, y varias utilidades mas. También es libre. www.geany.org
Dentro de las preferencias de Geany, habilitar la opción: Editor -> Mostrar espacios en blanco.

3.- El programa que escribes, solo es un archivo de texto con extension **.**py no es un ejecutable. Para ejecutarlo debes pasarselo al interprete de lenguaje, que lo analiza, convierte y ejecuta.
Si con tu explorador de archivos al hacer click derecho, abrir, sobre `tuarchivo.py` no pasa nada, o te dice que no reconoce el tipo de archivo.
Falta enseñarle a tu computador, que hacer con estos archivos, y la ruta hasta el interprete de python.
Como establecer las rutas, depende de que cada version de sistema operativo.

Al menos, enséñale a Geany en que ruta instalaste python …

Geany → menu → construir → establecer comando de construncion, y frente al boton "python" escribe: `c:\laRutaDondeEsta\python.exe "%f"`
Para que al presionar ejecutar, llame al interprete de python y le pase la dirección de tu archivo.

4.- Cualquier aplicación de utilidad, es mucho mas que **un** solo archivo.

Generalmente son muchos archivos, que se importan, como módulos o librerías (colecciones de archivos). Algunos de estos módulos, los mas usados, ya se instalan por defecto, al instalar el lenguaje python, si necesitas otros debes agregarlos a tu sistema.

En **PyPi** (Python Package Index) pypi.org encuentras cientos de estos módulos, libres para descargarlos y usarlos

Si en tu programa pones `import tkinter` todos los archivo de la librería tkinter ahora también son parte de tu programa.

Si en tu programa pones `import pymodbus` probablemente te dirá "Error, no se encuentra este modulo". Muy pocas personas usan una librería Modbus, esta librería hay que instalar primero.

5.- ¿ Como agregar una librería ?

Cuando instalaste python, también se instalo una aplicación que se llama **pypi** que es justamente para eso, instalar paquetes desde pypi.org

En un terminal (consola de comandos) ingresa:

`pip install pymodbus`	para agregar la librería Modbus
`pip uninstall PIL`	quita la librería PIL
`pip install Pillow`	agregar librería de manipulación de imágenes
`pip list`	muestra que paquetes ya has agregado
`pip help`	

6.- Si la aplicación no funciona, como encuentro que falla ?

Python siempre envía todos los mensajes de error hacia el terminal (consola de comandos). Ejecútala desde un terminal, y te mostrara donde esta cualquier problema que encuentre.

Si lo ejecutas con Geany el terminal continuará abierto, incluso aunque tu programa falle y se caiga, para que puedas ver en que linea del programa esta el error.

Mi primera ventana

Usando Geany menu → archivo → nuevo. Escribe (o copia) ...

```
from tkinter import *     # Carga módulo de widgets estándar
ventana = Tk()
ventana.geometry('300x200')  # anchura x altura
ventana.configure(bg = 'beige')
ventana.title('Mi ventana')
boton = Button(ventana, text='Salir', command=quit).pack(side=BOTTOM)
ventana.mainloop()
```

menu → archivo → Guardar ...

menu → construir → ejecutar

Felicitaciones. Creaste tu primera ventana !!!

Ahora puedes abrir ModbusTest2**.**py u otros ejemplos, y comenzar a jugar.

https://python-para-impacientes.blogspot.com/p/indice.html

https://python-para-impacientes.blogspot.com/p/tutorial-de-tkinter.html

https://docs.python-guide.org/

https://learnxinyminutes.com/docs/es-es/python-es/

https://docs.python.org/3/ documentación oficial

https://docs.python.org/3/library/tkinter.html

https://devfreebooks.github.io/python/