Introducción a la Programación de controladores lógicos (P L C)

Este es el texto de un pequeño cursillo, que es una introducción a los conceptos básicos de la programación de controladores lógicos, aplicados al control o sistema de seguridad de una maquina o proceso.

Se refiere a los conceptos básicos de programación estructural y modular, y a las consideraciones básicas de seguridad funcional (SIL2).

Para mostrar la aplicación practica de estos conceptos, se presenta y analiza el mismo ejemplo ficticio, desarrollado para equipos de los principales fabricantes de PLC.   El mismo caso resuelto en un Triconex, en un Schneider y en un ControlLogix, para ver las diferencias entre distintas marcas y modelos de controlador lógico programable.

Rolf Dahl-Skog
rolfds@gmail.com

Índice de contenido

Controladores Lógicos Programables.....3

Consideraciones de seguridad.....4

- Sistema Instrumentado de Seguridad (SIS).....5
- Seguridad Funciona (Functional Safety) 5
- Niveles de seguridad (SIL).....5

Consideraciones de diseño.....7

Programación de un PLC.....8

- ¿Que es un algoritmo ?.....9
- Programación Estructurada......10
- Programación Modular.....11
- Tipos de variables......12
- Lenguajes de programación.....14
- Ejemplos de algoritmos usados en plc.....16
  - Algoritmo “First_Out”.....16
  - Algoritmo “Fan_In” 17
  - Algoritmo “Fan_Out” 18

Definición de un caso ficticio como ejemplo.....19

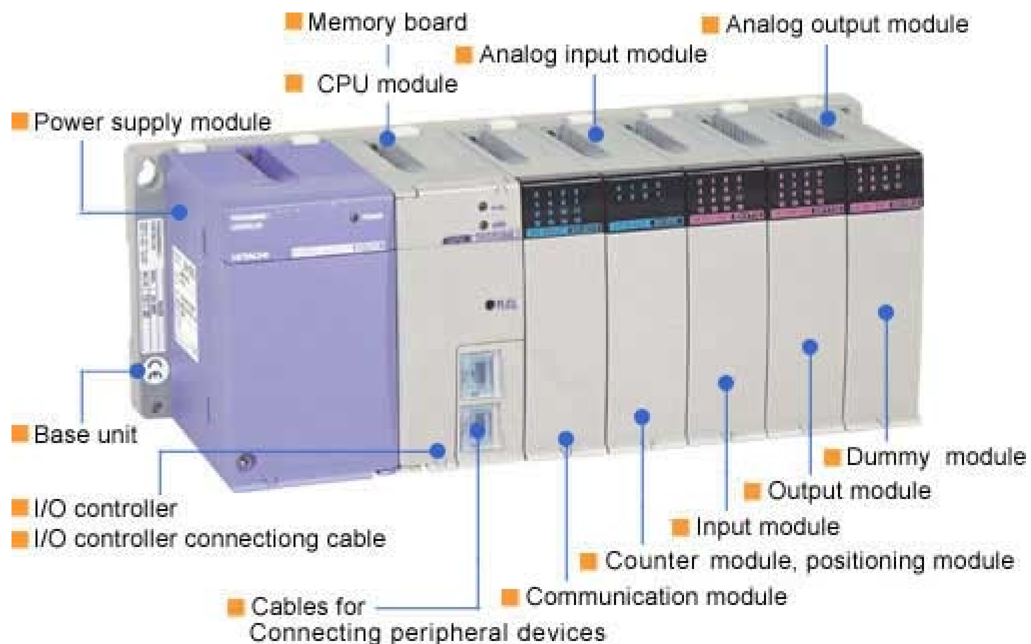
- Solución con Unity, de Schneider.....21
- Solución con Tristation, de Triconex.....27
- Solución con ControlLogix, de Rockwell.....33

Controladores Lógicos Programables.

Un Controlador Lógico Programable, o P.L.C. (**P**rogrammable **L**ogic **C**ontroller) es un computador especialmente diseñado para automatización industrial, para el control de una maquina o proceso industrial.

A diferencia de un computador tradicional, un PLC no tiene teclado, pantalla ni ratón, tampoco tienen disco duro ni Windows. Pero internamente si es un computador, con su hardware: procesadores, memoria, puertos de comunicación, etc. y con su software: un sistema operativo (que le llaman Firmware), y una programación, especifica para la aplicación o el caso particular en que se esta usando.

La principal diferencia entre un PC y un PLC, es que el PLC contienen múltiples canales para medir distintas señales provenientes de sensores instalados en la maquina o proceso que controlan. Y también tienen canales de salida de señal que actúan sobre la maquina o proceso que controlan.



Un PLC permite controlar o proteger un proceso industrial, posibilitando ademas las opciones de monitoreo y diagnostico de condiciones (alarmas), presentándolas en un HMI (Human-Machine Interface) o pantalla de operación, o presentándolas a una una red de control superior.

Un PLC es un ejemplo de control en tiempo real, pues reacciona automáticamente ante las condiciones variables que esta vigilando.

Un PLC puede ser parte de un sistema de control distribuido (DCS). O puede ser parte de un SCADA. O puede ser parte del sistema de seguridad.

Un DCS (Distributed Control System) . Es un sistema de control de un proceso (de una planta industrial) formado por una red de controladores. Un DCS esta orientado al control en tiempo real del proceso. En cambio se habla de un SCADA, cuando es una red orientada al monitoreo de equipos distantes (no control en tiempo real).

En cambio, un Sistema de seguridad, o SIS (Safety Instrumented System) es algo distinto a un sistema de control, es la protección para cuando el proceso (o la maquina) se sale de control. Por lo tanto: NO debe usarse el mismo PLC que se usa para el control de un proceso, como parte del sistema de seguridad (protección) del mismo proceso. Pues, **la finalidad de un Sistema de seguridad, es** justamente **llevar el proceso hacia una condición segura, cuando su control se sale de los limites de seguridad.**

Consideraciones de seguridad

La principal aplicación de un PLC en la industria petro-químicas y de sustancias químicas peligrosas, es formar parte de un sistema de seguridad (Safety Instrumented System) . Y por esto en el diseño de la solución adecuada para cada caso, siempre se debe analizarse también las necesidades de seguridad de cada caso, y también analizarse la integridad del diseño del mismo PLC desde el punto de vista de la seguridad.

La norma internacional IEC 61508, define los criterios de seguridad para el diseño, construcción y operación de sistemas eléctricos / electrónicos / electrónicos programables.

La norma internacional IEC 61511 esta basada en la IEC 61508, y define los criterios sobre la aplicación de **Sistemas Instrumentados de Seguridad** en las industrias de procesos, que se utiliza en industrias petro-químicas y de sustancias químicas peligrosas, entre otras.

La norma internacional IEC 61513 esta basada en la IEC 61508, y define los criterios sobre la aplicación de Sistemas Instrumentados de Seguridad en la industria nuclear.

Seguridad = Ausencia de riesgo inaceptable, de daño a las personas, de daños a la propiedad o al medio ambiente.

Sistema Instrumentado de Seguridad (SIS)

Un sistema instrumentado de seguridad (Safety Instrumented System) es un sistema que realiza funciones específicas para conseguir mantener el proceso dentro de condiciones seguras. Detectan cuando las condiciones del proceso son inaceptables o peligrosas, y toman automáticamente acciones correctivas.

La finalidad de un Sistema de seguridad (Safety Instrumented System), es llevar el proceso hacia una condición segura, cuando su control se sale de los limites de seguridad.

Los Sistema de seguridad, están separados y son independientes de los sistemas de control normal, aunque están compuestos por elementos similares, como sensores, procesadores de la lógica, actuadores , etc.

Seguridad Funciona (Functional Safety)

Estas normas definen una "Función de seguridad" (Functional Safety) como la detección de una condición potencialmente peligrosa, y la activación de un dispositivo de protección, o mecanismo correctivo para evitar las consecuencias del evento peligroso.

Las funciones de seguridad tienen el objetivo de reducir la probabilidad de eventos peligrosos que pueden causar daño.

Cada condición potencialmente peligrosa, que tenga el proceso (o la maquina) sera una función de seguridad distinta. Aun que varias funciones de seguridad pueden reaccionar activando el mismo dispositivo de protección.

La seguridad funcional se refiere a los sistemas activos que identifican una condición y reaccionan, para proteger de las consecuencias de esa condición. Se refiere a los sistemas instrumentados de seguridad; no se refieren a los mecanismos pasivos (por ejemplo una válvula de sobre presión).

Niveles de seguridad (SIL)

El correcto funcionamiento de un sistema de seguridad, requiere un correcto diseño para proporcionar la integridad y la confiabilidad requerida, por cada caso.

No todas las "funciones de seguridad", requieren el mismo nivel de seguridad, por que cada condición potencialmente peligrosa, tiene distintas probabilidades de ocurrencia, y también sus consecuencias tienen distintas magnitudes de daño potencial.

El "Nivel de Integración de Seguridad" o SIL (Safety Integrity Level) es el nivel de reducción de riesgo que se requiere para cada "Función de Seguridad". Siendo SIL-1 es el nivel menos seguro, y SIL-4 es el nivel mas seguro.

Para cada una de las "Función de Seguridad", del proceso o la maquina, debe determinarse el nivel de seguridad (SIL) requerido, analizando la probabilidad de ocurrencia de la condición peligrosa, y la magnitud del daño potencial de sus consecuencias.

SIL es entonces el nivel de seguridad requerido para cada determinada "Función de Seguridad"

Los requerimientos de seguridad (SIL) deben siempre determinarse a partir de un estudio de los riesgos de operación (**HAZOP**) del proceso o maquina correspondiente. Usando las técnicas de análisis que se mencionan en las normas IEC 61511 e IEC 61508.

Si después del **análisis de riesgos** de cada caso. Se define, que una determinada función de seguridad (una protección) debe ser SIL2 (o SIL3, etc) todos los elementos involucrados en esa función deben cumplir con el mismo criterio: Los sensores, transmisores, repetidores, actuadores, válvulas, etc y no solo el PLC. Para lograr esto, muchas veces se debe recurrir a sensores redundantes, o triple redundantes, o actuadores y válvulas redundantes, etc.

En el diseño del sistema de seguridad (SIS) debe considerarse el SIL de cada función de seguridad, y un análisis de fallos – efectos, y criticidad, de cada uno de los componentes del sistema.

Consideraciones de diseño

Las funciones de seguridad, de un sistemas de seguridad, se instalan para proteger a las personas, el medio ambiente y para la protección de activos de la empresa. Estas protecciones siempre deben activarse ante una condición peligrosa, pero solo en estas situaciones.

Mientras mayor el SIL requerido por una función de seguridad, deberán emplearse mas recursos tecnológicos para garantizar el funcionamiento de esta función de seguridad. Y mas complejo es el sistema, y mas cosas podrían fallar en él.

Si una función de seguridad se activa sin la presencia real de la situación peligrosa (por ejemplo, debido a una falla interna de algún componente, falla de instrumentación, etc.) provoca, detenciones innecesarias, que significan pérdidas económicas. El diseño debe incluir los auto diagnósticos y alarmas necesarios para advertir a tiempo las anormalidades. Usando redundancia y verificación de discrepancias si es necesario.

Como la mayoría de los PLC solo permite forzar señales de entrada o salida discretos, y hoy en día la mayoría de los instrumentos sensores no son discretos. El diseño de la lógica debería contemplar la forma de poder hacer mantenimiento de la instrumentación sin tener que detener el proceso productivo, incorporando bypass temporales de las protecciones, que no des habiliten la medición y alarmas.

La revisión y mantenimiento de los componentes del sistema instrumentado de seguridad, puede ser difícil, o hasta imposible de hacer sin detener el proceso, si en el diseño no se consideró el diagnostico y mantenimiento del sistema.

El Sistema resultante puede ser tan complejo como se imagine, o tan simple como lo que realmente se necesite. Un sistema de seguridad mal diseñado, no garantiza mas seguridad, solo dificultara su mantenimiento y la operación del proceso.

El comprar de equipos de mayor costo, certificados para seguridad, sin un buen **diseño integral** del sistema, puede que no aumente la seguridad, pero esta mayor complejidad tecnológica (y mayor inversión) si podría aumentara las detenciones innecesarias y las perdidas económicas, por no disponibilidad.

Programación de un PLC

Un programa de computadora es un conjunto de instrucciones que producirán la ejecución de una determinada tarea. En esencia, un programa es una respuesta predeterminada, a todas las posibles combinaciones de estados de la información que recibe.

El proceso de programación de un PLC es, por consiguiente, un proceso, en cuyo desarrollo se requieren cada una de las siguientes pasos:

1.- Definición y **análisis del problema**.

Este es el paso mas importante, tener claridad de que se necesita.

Este análisis debería incluir el resultado de un HAZOP.

1. 1 - ¿Qué resultados debe proporcionar el sistema? Que salidas, sobre que debe actuar.

1. 2 - ¿Qué datos se necesitan para determinar el resultado? Que debe medir o vigilar (Entradas)

1. 3 - ¿Como debe reaccionar ante perdida de información? Como debe reaccionar si no puede medir variables que necesita vigilar. (SIL.)

1. 4 - ¿Como debe reaccionar ante fallas de si mismo? Capacidad de auto diagnostico (SIL.)

2.- Definición de la arquitectura del hardware necesario (entradas/salidas, redundancia, auto diagnostico, etc.)

3.- Diseño de los algoritmos.

4.- Programación del código, en los lenguajes de programación.

5.- Depuración y verificación del programa (pruebas efectivas).

¿Que es un algoritmo ?

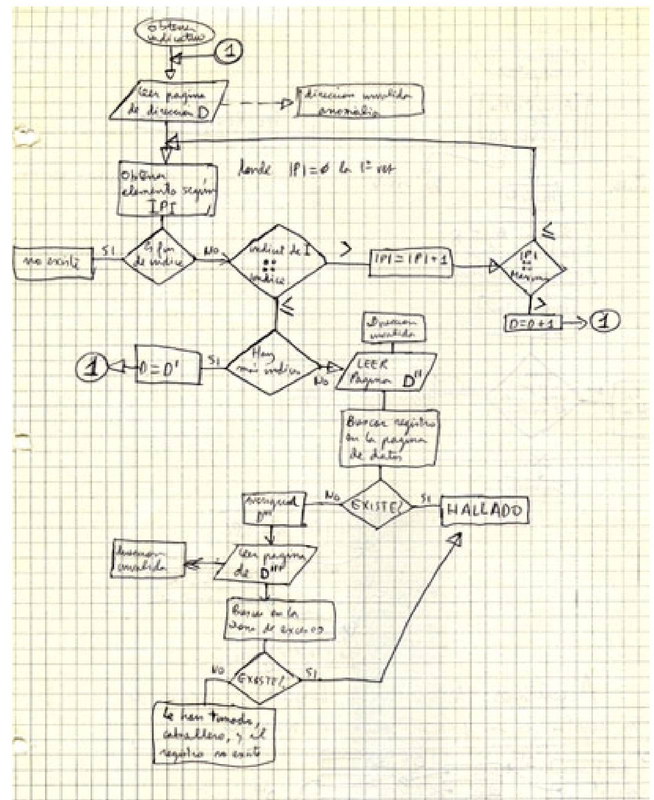
Un algoritmo es una secuencia de pasos (instrucciones o reglas) para llevar a cabo una tarea especifica.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan, como de la maquina que los ejecuta.   Diseñar un Algoritmo es diseñar un método de toma de decisiones.

Las características fundamentales que debe cumplir todo algoritmo son:

- La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida.
- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Ante las mismas condiciones de entradas, siempre debe obtener se el mismo resultado.
- Un algoritmo debe ser finito. Debe de tener un número finito de pasos. (En ninguna situación se puede quedar “pensado” en un ciclo infinito.

Todo programa de computador es un conjunto de algoritmos.



Ejemplo de algoritmo:

<u>Entradas</u>:

Botón de partida
Botón de parada
Estado del motor

<u>Salidas</u>:

Comando de marcha o detención hacia el motor

<u>Lógica del algoritmo</u>:

Si motor esta andando **O** esta activado botón de partida **Y** no esta activado botón de parada
Entonces   enviar comando de marcha
De lo contrario enviar comando detener

Programación Estructurada.

La programación estructurada es la forma clásica de programar en forma secuencial.   Podríamos decir que es lo contrario a la programación orientada a objetos (clases, eventos).

La programación estructurada utiliza solo 3 tipos de estructuras:

- Secuencias
- Selección (Instrucción condicional)
- Interacción (repetición condicional)

Cada una de estas estructuras puede a su vez contener otras de estas mismas estructuras, a esto se le llama "anidamiento".

Ejemplo:

```
Leer BotonPartir
Leer BotonParar
Si BotonPartir Y No BotonParar Entonces
      Motor = Activar
Sino
      Motor = Detener
FinDelSi
```

Otro ejemplo:

```
Leer x
Leer y
Si x > y Entonces
      Escribir x " es mayor que " y
Sino
      Escribir y " es mayor que " x
FinDelSi
```

Este ejemplo tiene un error.   ¿Lo viste?

Un programa es una respuesta predeterminada, a todas las posibles combinaciones de estados de la información que recibe.   ¿Que ocurre en este ejemplo si x es igual a y ?   También escribiría “y es mayor que x”.

Ventajas de la programación estructurada

-. Los programas son más fáciles de entender, pueden ser leídos de forma secuencial para entender la lógica.

-. La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.

-. Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los errores del programa se facilita debido a su estructura más sencilla y comprensible, por lo que los errores se pueden detectar y corregir más fácilmente.

-. Reducción de los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.

-. Los bloques de código son casi auto-explicativos, lo que reduce y facilita la documentación.

Inconvenientes de la programación estructurada

El principal inconveniente de la programación estructurada es que se obtiene un único bloque de programa, que cuando se hace demasiado grande, puede resultar problemático para el manejo de su código fuente por su gran extensión.

Esto se resuelve empleando, en forma conjunta tanto las técnicas de programación estructurada como las de **programación modular**.

En la actualidad la conjunción "Programación estructurada" y "programación modular" es la más utilizadas, en la programación de PLC, en la que los módulos tienen una **estructura jerárquica** en la que se pueden definir funciones dentro de funciones.

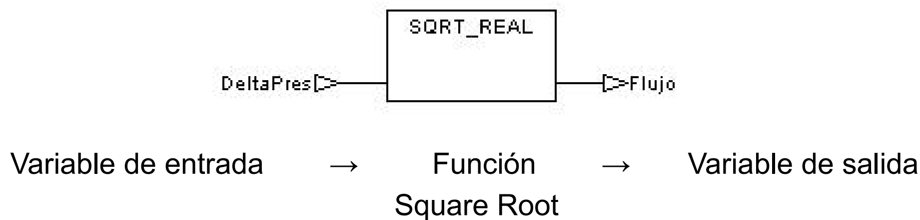
Programación Modular

La programación modular consiste en dividir un programa en módulos o sub-programas con el fin de hacerlo más legible y manejable.
Históricamente la programación modular, es una evolución de la programación estructurada, para solucionar problemas de programación mas grandes y complejos.

la programación modular, consiste en dividir un problema complejo en varios sub-problemas más simples, y estos a su vez en otros sub problemas más simples. Esto se hace hasta obtener sub problemas lo suficientemente simples como para poder ser resueltos fácilmente (divide y vencerás).

Cada modulo, es una parte de un programa, y es lo que en informática se conoce como "sub-rutina" y/ o "función".

Tradicionalmente se denomina “función” al sub programa que devuelve un resultado. Podemos alimentar una “funcion” con datos y esta nos entregará un resultado, sin necesitar tener que ocuparnos de su funcionamiento interno.



Variable de entrada → Función → Variable de salida
Square Root

Como ejemplo de función, en la imagen se muestra la función “SQRT-real” esta es una función ya existente que viene con el sistema, simplemente le damos un valor y nos devuelve otro valor que es la raiz cuadrada del anterior. De la misma manera podemos crear nuestras propias funciones para lo que necesitemos.

La ventaja de usar un sub programa que sea una “funcion” es que podemos aplicar muchas veces el mismo algoritmo para distintos datos, escribiendo una sola vez el código (algoritmo en lenguaje de programación), sin tener que escribir repetidas veces el mismo código.

Una subrutina o función, solo se ejecuta cuando es llamada desde dentro de otro programa, y hace que el programa principal se detenga, por que "le pasa" la ejecución a la sub rutina. El programa que llamo a la sub rutina, solo continuará su ejecución, cuando la sub rutina termina, y le “devuelve" el resultado.

Tipos de variables.

En programación, una variable es un espacio reservado en la memoria, que pueden cambiar de contenido a lo largo de la ejecución de un programa.

Las variables se representan con identificadores ("Tag") que hacen referencia al lugar de la memoria en donde se almacena un dato.

<u>Respecto a su ámbito</u>, un variable puede ser:

Variable Local: Cuando la misma sólo es accesible desde el sub programa al que pertenece, no pudiendo ser leída o modificada desde otro sub programa.

Variable Global: Cuando la misma es accesible desde todos los sub programas de la aplicación.

En un PLC todas sus entradas y salidas siempre son variables globales.

Las variables internas solo serán globales si específicamente se las define como tales. De lo contrario solo serán locales del sub programa dentro del cual están definidas.

La ventaja de usar variables locales, es que evitamos tener un gigantesco listado de variables globales, difícil de manejar.

Respecto a su tipo, un variable puede ser:

Tipo Logica: ("**booleana**") Verdadero / Falso.

Tipo **Entero**: un valor entero entre 0 y n.   Donde n depende de la cantidad de bytes con que trabaje el sistema, (y si usa o no un bit para representar el signo).
Por ejemplo: 2 bytes = 2x8 bits = 16bits ----- 2 elevado a 16 = 65536.   El valor maximo seria 65535

Tipo **Real**: o "coma flotante": El rango valido y su exactitud varia según la cantidad de bytes con que trabaje el sistema (cuociente + exponente).
Por ejemplo: 314,16 = 3,1416 x 10 elevado 2

Tipo caracteres: ("**String**") Es una secuencia de números enteros, que representa una secuencia de caracteres (letras o signos) de un determinado alfabeto.

Según el tamaño de memoria que usan, se acostumbra hablar de:

Bool = 1 bits
Byte = 8 bits
Word = 16 bit
Double Word = 32 bit
Integer = 2 bytes = 16 bits
Double Integer = 4 bytes
Real = 4 bytes
etc.

Lenguajes de programación

El standard internacional IEC 61131 define los siguientes lenguajes de programación para PLC:

- Instruction list (IL), *texto*
- Ladder (LD), *grafico*
- Function block diagram (FBD), *grafico*
- Structured text (ST), *texto*

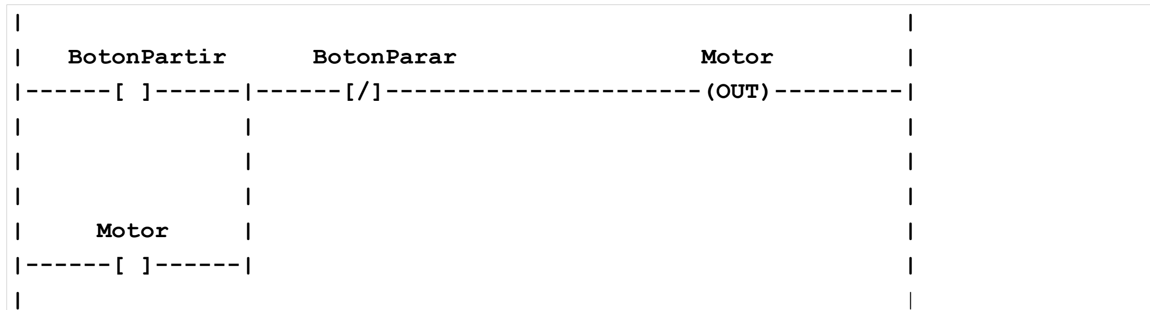
El **lenguaje "Instruction list"** no es mas que la representación en forma de texto del lenguaje gráfico "Ladder". Es el mas antiguo de los lenguajes de programación de PLC. Precursor del Ladder se usaba cuando los computadores aun no tenían capacidad gráfica.

Ejemplo en Instruction list

```
LD BotonPartir
OR Motor
AND NOT BotonParar
OUT Motor
```

El **lenguaje LADDER**, ("escalera"), es un lenguaje de programación gráfico muy popular, ya que está basado en los clásicos esquemas de control eléctricos con reles. De este modo, es muy fácil de entender para un técnico eléctrico.

El mismo ejemplo anterior en Ladder

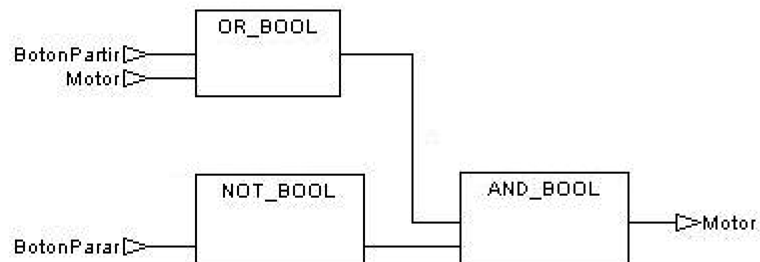


<u>**Desventajas** del lenguaje ladder</u>:

- Es el más adecuado para controlar los problemas, donde sólo las variables discretas (boleanas) son necesarias y/o donde son el problema principal de control.
- Es difícil manipular las variables analógicas y expresar las operaciones aritméticas.
- Tiene un soporte muy limitado para las matrices y bucles, resultando a menudo en la duplicación de código.

El **lenguaje "Function block diagram"** es un lenguaje gráfico, que describe una función entre las variables de entrada y variables de salida, como un conjunto de bloques elementales, que están conectados por lineas líneas de conexión.

El mismo ejemplo anterior en Function block diagram



El **lenguaje "Structured text"** es un lenguaje de alto nivel, que es estructurado en bloques y sintácticamente similar a Pascal.

El mismo ejemplo anterior en Structured text

```
IF ( BotonPartir OR Motor ) AND NOT BotonParar  THEN
      Motor := TRUE;
ELSE
      Motor := FALSE;
END_IF;
```

Ventajas del lenguaje Structured text :

- Soporta instrucciones aritmeticas complejas.
- Soporta instrucciones anidadas.
- Soporta ciclos de iteración (repeat-until, while-do)
- Soporta ejecución condicional (If-Then-Else, Case)
- Es fácil agregar comentarios que explican la lógica.
  Todo lo que esta entre **(*** y ***)** es comentario y no se ejecuta.

Ejemplos de algoritmos usados en plc

Algoritmo "First_Out"

Una maquina o proceso tiene, ademas de alarmas que no detienen, varias alarmas que detienen su funcionamiento. Cuando por alguna de estas se detiene la maquina, su detención produce alteraciones en el proceso, que provocan a su vez la activación de otras alarmas que también podrían haber causado la detención. El resultado es que cuando el operador llega a ver por que se detuvo la maquina, se encuentra con muchas alarmas activadas, y no sabe cual de ellas fue la primera que causo la detención.

```
(* Esta parte del programa registra cual es el motivo de la detencion de la
maquina.   Para que esto funcione, debe respetarse este orden de ejecucion de los
programas:
  1- Evaluar las entradas y calcular cuales Alarmas estan activadas.
  2- Decidir si maquina continua funcionando o hay que detenerla.
  3- Despues de decidir seguir o parar, ejecutar este algoritmo "First-Out"
  4- Mostrar los estados en el panel de Alarms o HMI.

ENTRADAS: Este algoritmo necesita recibir las siguientes señales.
 El arreglo TripAlarmNow[x] que contiene los estados actuales de todas las
condiciones que detienen.
 El arreglo AnalogNow[x] que contiene los valores actuales de las variables
analogas criticas
 La variable "Run" es si la maquina esta andando o detenida.

SALIDAS: Este algoritmo devolvera las siguientes señales.
 El arreglo TripAlarmLastSD[x] que contiene los estados que tenian todas las
condiciones que detienen, durante la ultima detencion
 El arreglo AnalogLastSD[x] que contiene los valores que tenian las variables
analogas criticas, durante la ultima detencion

Variables locales:
    YaGrabado del tipo Bool
 *)

IF NOT(Run) AND NOT(YaGrabado) THEN
    FOR x:= 1 TO 99 DO   (*Suponiendo que 99 es la dimension de los arreglos*)
        TripAlarmLastSD[x] := TripAlarmNow[x]
        AnalogLastSD[x] := AnalogNow[x]
    END_FOR;
    YaGrabado:= TRUE;
END_IF;
```

```
IF Run THEN
    YaGrabado:= FALSE;
END_IF;
```

Este algoritmo “First_out” permite tener una foto del estado de las variables criticas en el instante en que se detuvo la maquina, de manera de poder indicar cual fue la causante.

En el caso de un compresor basta con uno de estos algoritmos, por que la maquina tiene solo dos estados, pero, por ejemplo en un horno, en que hay válvulas de pilotos a gas, válvulas de quemadores de gas y válvulas de quemadores de petroleo. Habrá que tener un “First-Out” pilotos, un “First-Out” quemadores de gas, y un “First-Out” quemadores de petroleo, cada uno conectado a sus variables correspondientes.

Algoritmo “Fan_In”

Convierte 16 discretos en un valor entero entre 0 y 65535, usado para enviar por comunicación, los booleanos como un valor dentro de un arreglo de valores.

```
temp :=0
Si  bit00  entonces  temp := temp + 1
Si  bit01  entonces  temp := temp + 2
Si  bit02  entonces  temp := temp + 4
Si  bit03  entonces  temp := temp + 8
Si  bit04  entonces  temp := temp + 16
Si  bit05  entonces  temp := temp + 32
Si  bit06  entonces  temp := temp + 64
Si  bit07  entonces  temp := temp + 128
Si  bit08  entonces  temp := temp + 256
Si  bit09  entonces  temp := temp + 512
Si  bit10  entonces  temp := temp + 1024
Si  bit11  entonces  temp := temp + 2048
Si  bit12  entonces  temp := temp + 4096
Si  bit13  entonces  temp := temp + 8192
Si  bit14  entonces  temp := temp + 16384
Si  bit15  entonces  temp := temp + 32768
ValorOut := temp
```

Algoritmo “Fan_Out”

Es el algoritmo inverso del anterior, extrae 16 discretos desde un valor entero entre 0 y 65535. entregando un arreglo de booleanos "Alarma[x]

```
temp1 :=ValorIn;
Para  x  desde 16 hasta 1 repetir
    temp2 := ( 2 elevado a ( x-1 ) )
    Si  (temp1 / temp2) es menor que  1  entonces
            Alarma[x] = Falso
      De lo contrario
            Alarma[x] = Verdadero
            temp1 := temp1 - temp2;
Continuar con la siguiente repetición
```

En lenguaje de texto estructurado esto mismo es ...

```
temp1 :=ValorIn;
FOR x:= 16 TO 1 BY -1 DO
    temp2 := ( 2**(x-1));  (* 2 elevado a ... *)
    IF (temp1/temp2) < 1.0 THEN
        Alarma[x].a := FALSE;
      ELSE
        Alarma[x] := TRUE;
        temp1 := temp1 - temp2;
    END_IF;
END_FOR;
```

Definición de un caso ficticio como ejemplo

Para poder ver la aplicación practica de los conceptos de programación estructurada y modular, usando funciones y arreglos de variables locales, se plantea un caso ficticio, para resolverlo programan el mismo caso en PLC de distintos fabricantes.

Cada fabricante de PLC ofrece sus propias herramientas de programación (software), y cada modelo tiene sus propias características y particularidades.

En el desarrollo del ejemplo, no se detalla la configuración del hardware, ni las alarmas de auto diagnostico del mismo plc. Ya que la forma de hacer esto es distinta en cada modelo especifico de plc, y la lógica que se muestra aquí es la parte genérica, se podría aplicar a toda una linea de plc del mismo fabricante. La mayoría de los plc entregan variables de diagnostico de si mismos, y es el programador el que puede usar o no esta información en su programación.

<u>Planteamiento del problema:</u>

Supongamos que tenemos un compresor de gases reciproco (de pistón) movido por un motor eléctrico, y se necesita que tenga las siguientes alarmas y protecciones.

El motor se debe detener ante cualquiera de las siguientes condiciones:

- Botón de detención manual
- Baja presión de aceite de lubricación (medida por un transmisor de 4 a 20 miliamperes)
- Alta temperatura de descarga (medida por un transmisor de 4 a 20 miliamperes)
- Baja presión de succión (señal desde un switch de presión)
- Sobrecarga eléctrica (o falla eléctrica en general)
- Alta vibración (señal desde un switch de vibración)
- Alta diferencial succión descarga (señal desde un switch de presión diferencial)

El motor debe partir solo cuando una persona presione un botón de partida manual, siempre y cuando en ese momento estén cumplidos los siguientes requisitos:

- Ninguna de las condiciones que lo detienen
- Presión de aceite de lubricación tenga un valor normal (media por el transmisor)
- Temperatura de descarga tenga un valor normal (media por el transmisor)
- Exista presión de agua de enfriamiento (señal desde un switch de presión)

Debe tener un panel con luces de alarma que indique cada una de estas condiciones de alarma. Este debe funcionar como un panel de alarmas clásico. Esto es la luz de indicación de cada alarma debe parpadear hasta que la nueva alarma sea reconocida, con un botón reconocer.

- Alarma de detención por el botón de detención manual
- Alarma de muy baja presión de aceite
- Alarma de baja presión de aceite

Alarma de muy alta temperatura de descarga
Alarma de alta temperatura de descarga
Alarma de baja presión de succión
Alarma de falla eléctrica
Alarma de alta vibración
Alarma de alta diferencial succión descarga
Alarma de baja presión de agua de enfriamiento
Resumen alarma no reconocida (Bocina de alarma)
Resumen de condición de alarma presente (un rele a una alarma remota)

Debe ademas permitir que el estado de todas estas alarmas puedan ser leídas por comunicación digital desde un DCS.

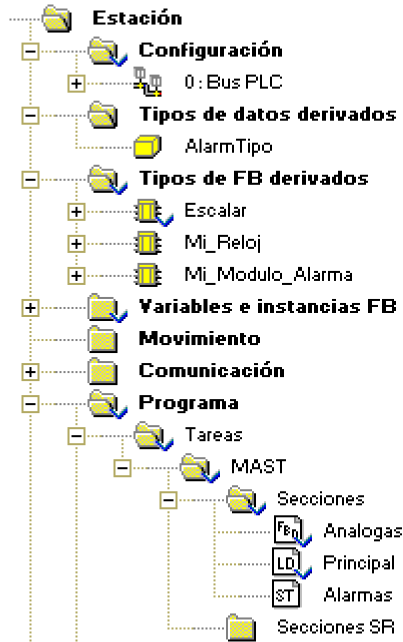
Se trata de un ejemplo típico de los mas simples.

El desarrollados que se muestra a continuación permiten dar un vistazo a las particularidades, de la programación de los principales fabricantes de plc, y ver como se aplica en ellos los conceptos de programación estructurada y modular. Las funciones, arreglos, ciclos, etc.

Solución con Unity, de Schneider

Para PLCs Modicon M340, Premium o Quamtum

El programa se divide en tres sub-programas: Analogas, Principal y Alarmas.

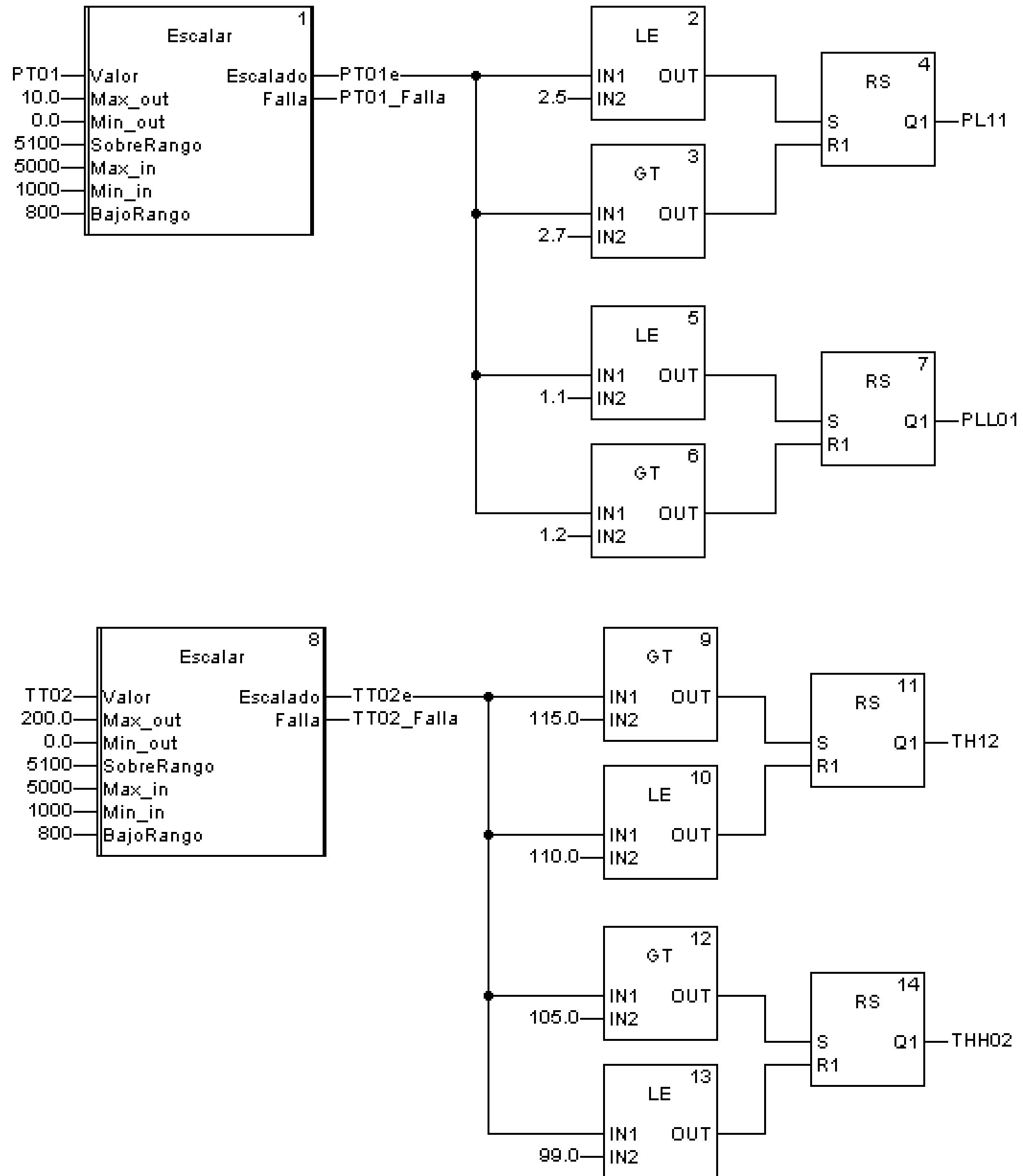


El primer programa que se ejecuta es "Analogas", donde se escalan las entradas analógicas y se comparan con las valores de alarma. Este programa a su vez usa la función "Escalar".

La **función "Escalar"** recibe la entrada análoga en rango de 1000 a 5000 mili volt y la escala al rango deseado. También levanta una alarma si la entrada esta fuera de su rango.

```
ValorIn := INT_TO_REAL(Valor);
MaxIn := INT_TO_REAL(Max_in);
MinIn := INT_TO_REAL(Min_in);

Escalado := ((((ValorIn - MinIn)/(MaxIn - MinIn))*(Max_Out - Min_Out))+ Min_Out) ;

IF (Valor > SobreRango) OR (Valor < BajoRango) THEN
    Falla := TRUE;
ELSE
    Falla := FALSE;
END_IF;
```

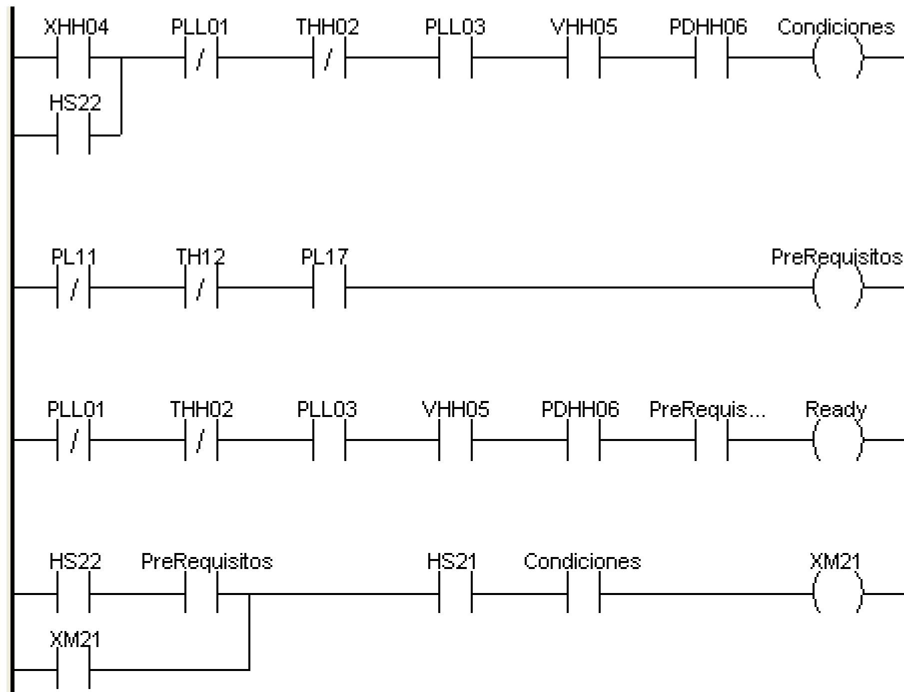
El **programa “Analogas”.**



El segundo programa que se ejecuta es “Principal”, en el se decide si mantener, detener o dar partida al motor principal.

HS21 es el boton de parar (normal cerrado).
HS22 es el boton de partir (normal abierto).
“Ready” es la indicación de que están todas las condiciones para partir.
XM21 el el comando de marcha o detención hacia el motor.

El **programa “Principal”**



Después se ejecuta el **programa “Alarmas”** donde se informa al operador los estados de las alarmas. El programa “Alarmas” usa un arreglo llamado “Alarm” (de dimensión 0 a 99) del tipo de “AlarmTipo” que es una estructura compuesta de ...

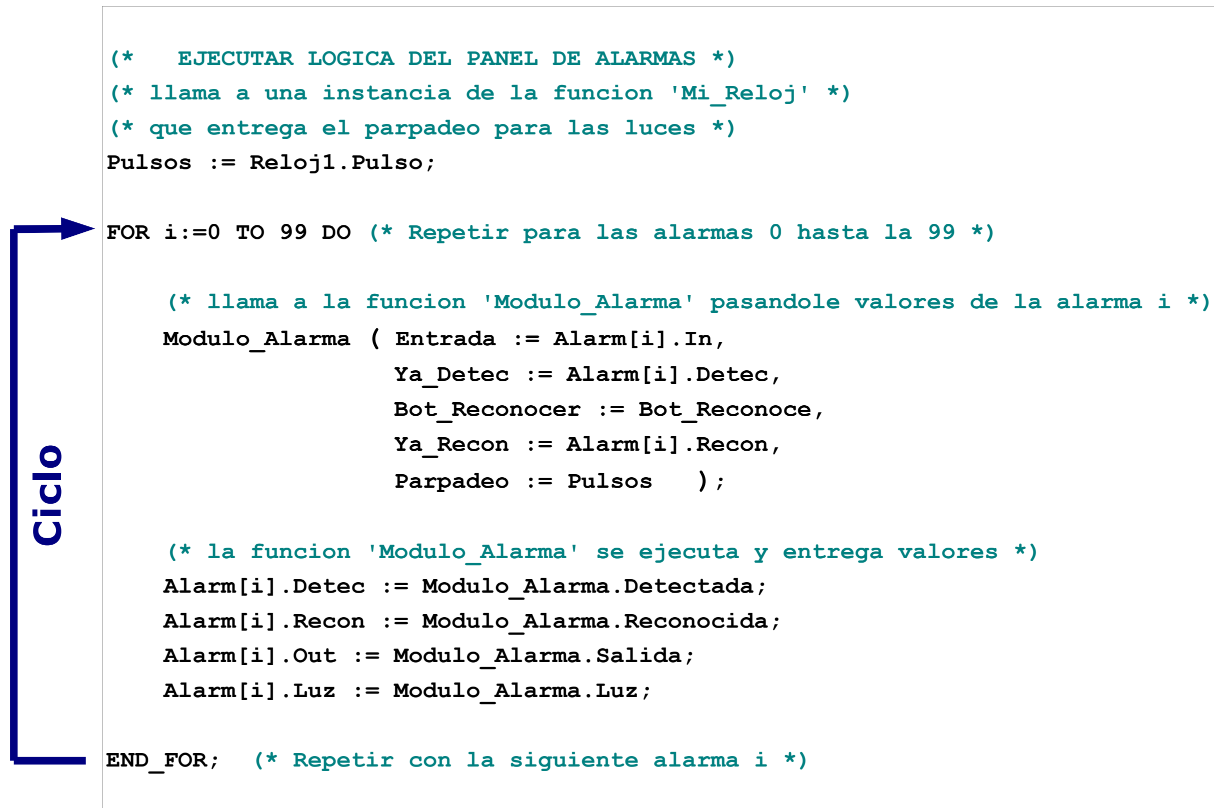
Nombre	Tipo
AlarmTipo	<Estruct.>
In	BOOL
Detec	BOOL
Recon	BOOL
Out	BOOL
Luz	BOOL

En la primera parte de este programa se asocian las condiciones y estados a las entradas de las alarmas.

```
(*_____________LEER__ESTADOS_____________*)
Bot_Reconoce := HS23;
Alarm[0].In := NOT(HS21); (*Detencion manual*)
Alarm[1].In := PLL01; (*muy baja presion aceite*)
Alarm[2].In := PL11; (*baja presion aceite*)
Alarm[3].In := THH02; (*muy alta temp descarga*)
Alarm[4].In := TH12; (*alta temp descarga*)
Alarm[5].In := NOT(PLL03); (*baja presion succion*)
Alarm[6].In := NOT(XHH04); (*falla electrica*)
```

```
Alarm[7].In := NOT(VHH05); (*alta vibracion*)
Alarm[8].In := NOT(PDHH06); (*alta diferencial succion descarga*)
Alarm[9].In := NOT(PL17); (*baja presion agua*)
Alarm[10].In := FALSE; (*Alarma disponible*)
Alarm[11].In := FALSE; (*Alarma disponible*)
Alarm[12].In := FALSE; (*Alarma disponible*)
Alarm[13].In := FALSE; (*Alarma disponible*)
Alarm[14].In := FALSE; (*Alarma disponible*)
(* se pueden seguir agregando hasta 99 alarmas*)
```

En la segunda parte de este programa se ejecuta la lógica de panel de alarmas, para esto se llama , por cada una de las alarmas, a la función “Modulo_Alarma”.



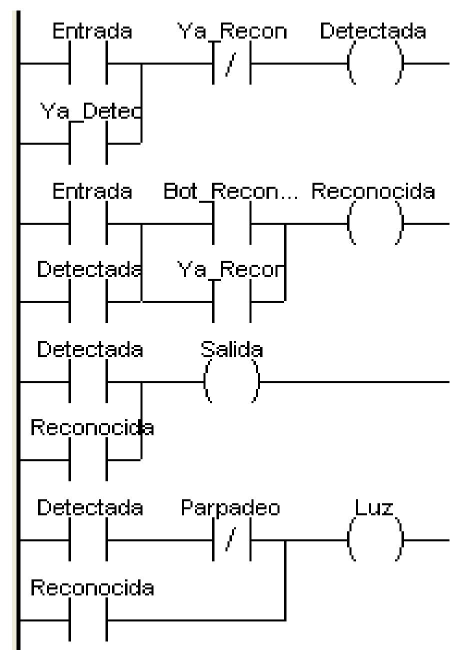

```
(*   EJECUTAR LOGICA DEL PANEL DE ALARMAS *)
(* llama a una instancia de la funcion 'Mi_Reloj' *)
(* que entrega el parpadeo para las luces *)
Pulsos := Reloj1.Pulso;

FOR i:=0 TO 99 DO (* Repetir para las alarmas 0 hasta la 99 *)

    (* llama a la funcion 'Modulo_Alarma' pasandole valores de la alarma i *)
    Modulo_Alarma ( Entrada := Alarm[i].In,
                    Ya_Detec := Alarm[i].Detec,
                    Bot_Reconocer := Bot_Reconoce,
                    Ya_Recon := Alarm[i].Recon,
                    Parpadeo := Pulsos   );

    (* la funcion 'Modulo_Alarma' se ejecuta y entrega valores *)
    Alarm[i].Detec := Modulo_Alarma.Detectada;
    Alarm[i].Recon := Modulo_Alarma.Reconocida;
    Alarm[i].Out := Modulo_Alarma.Salida;
    Alarm[i].Luz := Modulo_Alarma.Luz;

END_FOR;  (* Repetir con la siguiente alarma i *)
```

En la tercera parte de este programa se copia el resultado de la lógica de alarmas hacia las salidas correspondientes.

```
(*   ESCRIBIR RESULTADOS EN LUCES DE ALARMA *)
XA21 := Alarm[0].Luz; (*Detencion manual*)
XA01 := Alarm[1].Luz; (*muy baja presion aceite*)
XA11 := Alarm[2].Luz; (*baja presion aceite*)
XA02 := Alarm[3].Luz; (*muy alta temp descarga*)
XA12 := Alarm[4].Luz; (*alta temp descarga*)
XA03 := Alarm[5].Luz; (*baja presion succion*)
XA04 := Alarm[6].Luz; (*falla electrica*)
XA05 := Alarm[7].Luz; (*alta vibracion*)
XA06 := Alarm[8].Luz; (*alta diferencial succion descarga*)
```

```
XA17 := Alarm[9].Luz; (*baja presion agua*)
(*XA..  := Alarm[10].Luz; (*Alarma disponible*)
(*XA..  := Alarm[11].Luz; (*Alarma disponible*)
(*XA..  := Alarm[12].Luz; (*Alarma disponible*)
(*XA..  := Alarm[13].Luz; (*Alarma disponible*)
(*XA..  := Alarm[14].Luz; (*Alarma disponible*)
(*XA..  := Alarm[15].Luz; (*Alarma disponible*)

(*____________________________________________________________*)
(*  ESTADOS DE ALARMAS PARA COMUNICACION MODBUS *)
%M100 := Alarm[0].Out; (*Modbus 0:0101 = Alarma Detencion manual*)
%M101 := Alarm[1].Out; (*Modbus 0:0102 = Alarma muy baja presion aceite*)
%M102 := Alarm[2].Out; (*Modbus 0:0103 = Alarma baja presion aceite*)
%M103 := Alarm[3].Out; (*Modbus 0:0104 = Alarma muy alta temp descarga*)
%M104 := Alarm[4].Out; (*Modbus 0:0105 = Alarma alta temp descarga*)
%M105 := Alarm[5].Out; (*Modbus 0:0106 = Alarma baja presion succion*)
%M106 := Alarm[6].Out; (*Modbus 0:0107 = Alarma falla electrica*)
%M107 := Alarm[7].Out; (*Modbus 0:0108 = Alarma alta vibracion*)
%M108 := Alarm[8].Out; (*Modbus 0:0109 = Alarma alta diferencial succion descarga*)
%M109 := Alarm[9].Out; (*Modbus 0:0110 = Alarma baja presion agua*)
%M110 := Alarm[10].Out; (*Modbus 0:0111 = Alarma disponible*)
%M111 := Alarm[11].Out; (*Modbus 0:0112 = Alarma disponible*)
%M112 := Alarm[12].Out; (*Modbus 0:0113 = Alarma disponible*)
%M113 := Alarm[13].Out; (*Modbus 0:0114 = Alarma disponible*)
%M114 := Alarm[14].Out; (*Modbus 0:0115 = Alarma disponible*)
%M115 := Alarm[15].Out; (*Modbus 0:0116 = Alarma disponible*)
```

La función "**Modulo_Alarma**" contiene lo que se muestra en la imagen a la derecha.
Este es la misma lógica aplicada a cada una de las 99 alarmas.

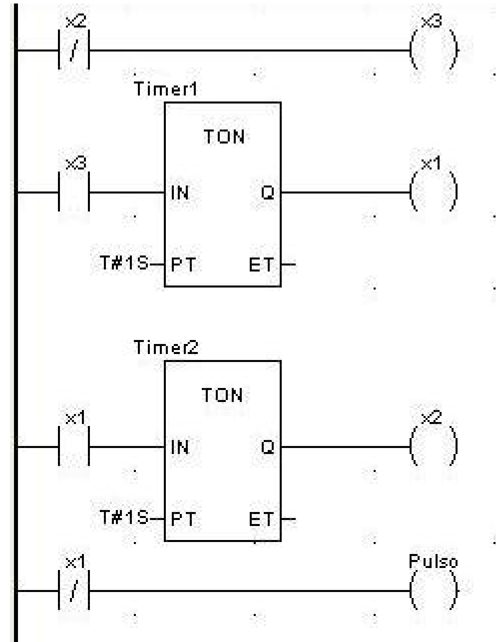


El usar un arreglo en conjunto con un ciclo condicional, permite en vez de escribir 99 veces la lógica para cada una de las 99 alarmas. Escribirla una sola vez y usarla 99 veces para las 99 alarmas.

El programa “Alarmas” también llama a la **función “Mi_reloj”** para generar el pulso con que parpadean las luces de las alarmas.

Nota que las variables x1, x2, x3, Timer1 y Timer2, no aparecen en el listado de variables de los demás programas, por que son variables locales (internas) de esta función “Mi_reloj”

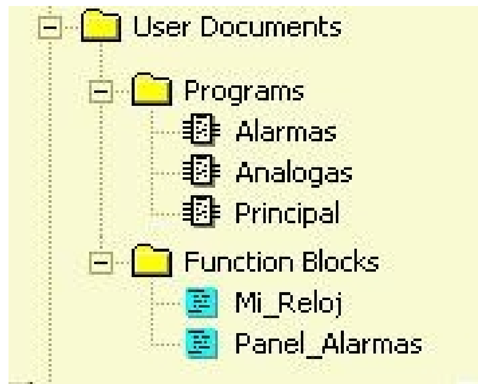
Esta función no tiene entradas, solo tiene una salida discreta llamada “Pulso”



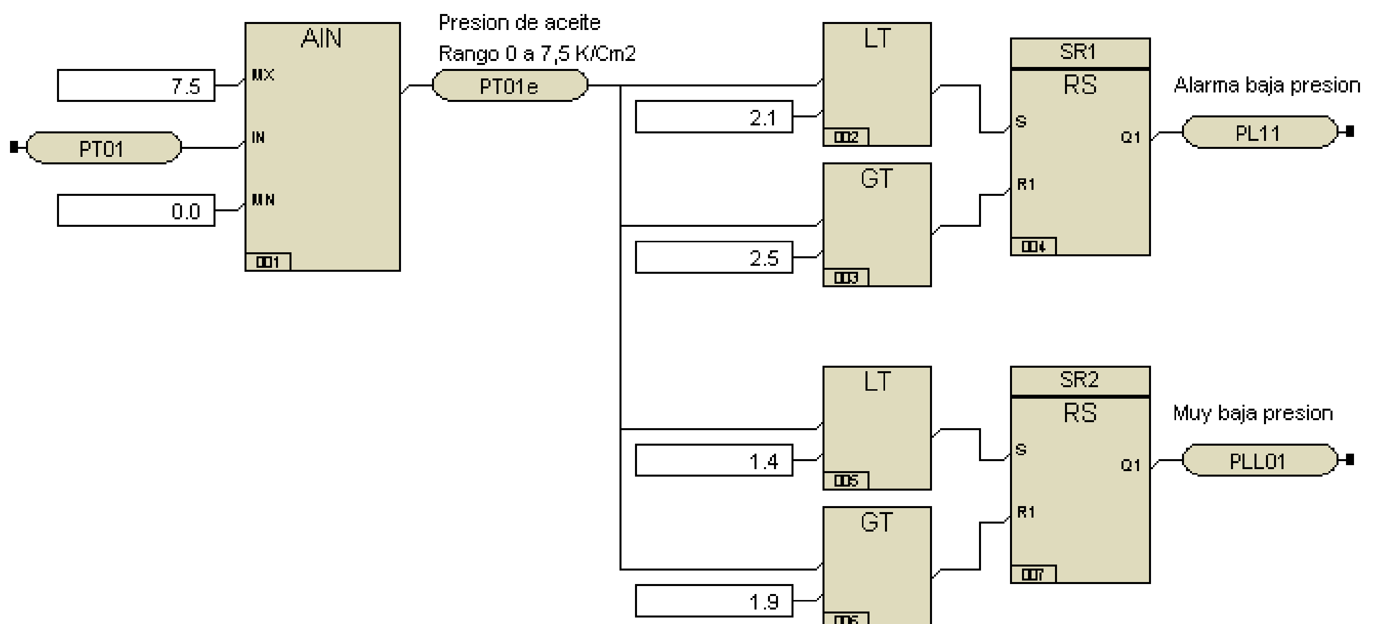
Solución con Tristation, de Triconex

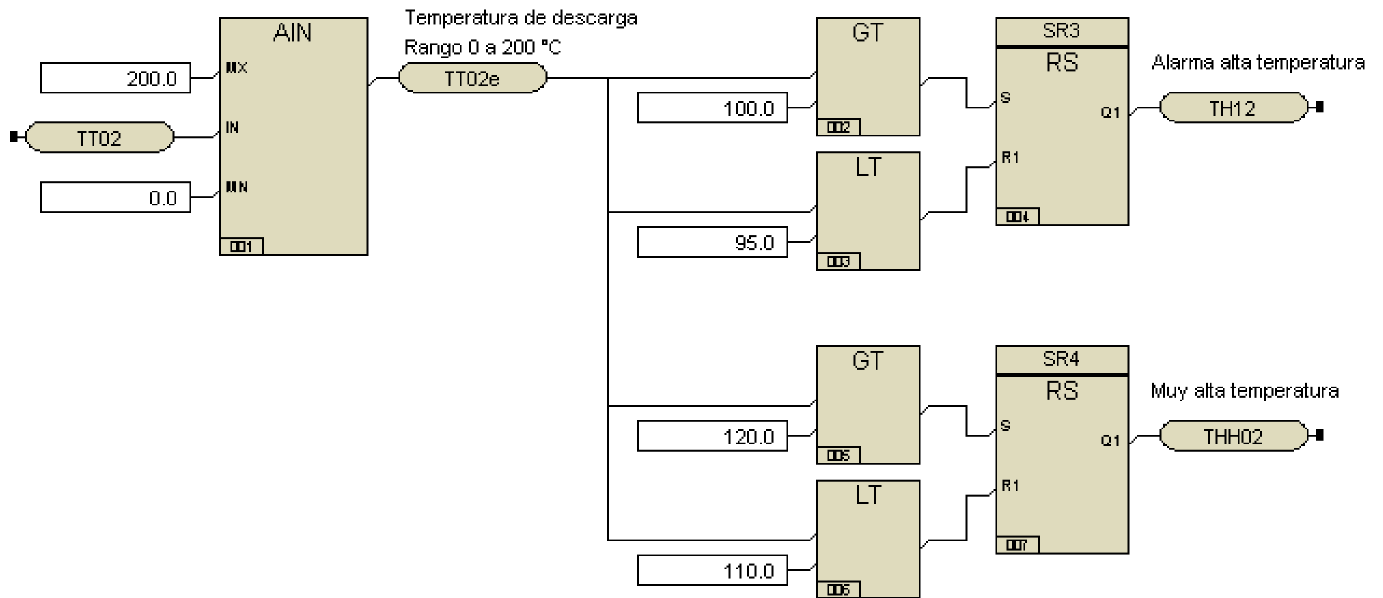
Para PLCs Triconex o Trident.

El programa se divide en tres subprogramas. Dentro de la configuracion del PLC se debe definir el orden de ejecución de estos: Primero se ejecuta el que se llama “Analogas”, donde se escalan las entradas analógicas y se comparan con las valores de alarma. Después se ejecuta el que se llama “Principal”, donde se decide si detener o no el motor. Después se ejecuta el que se llama “Alarmas” donde se informa al operador los estados de las alarmas.



El **programa “Analogas”** tiene lo siguiente:

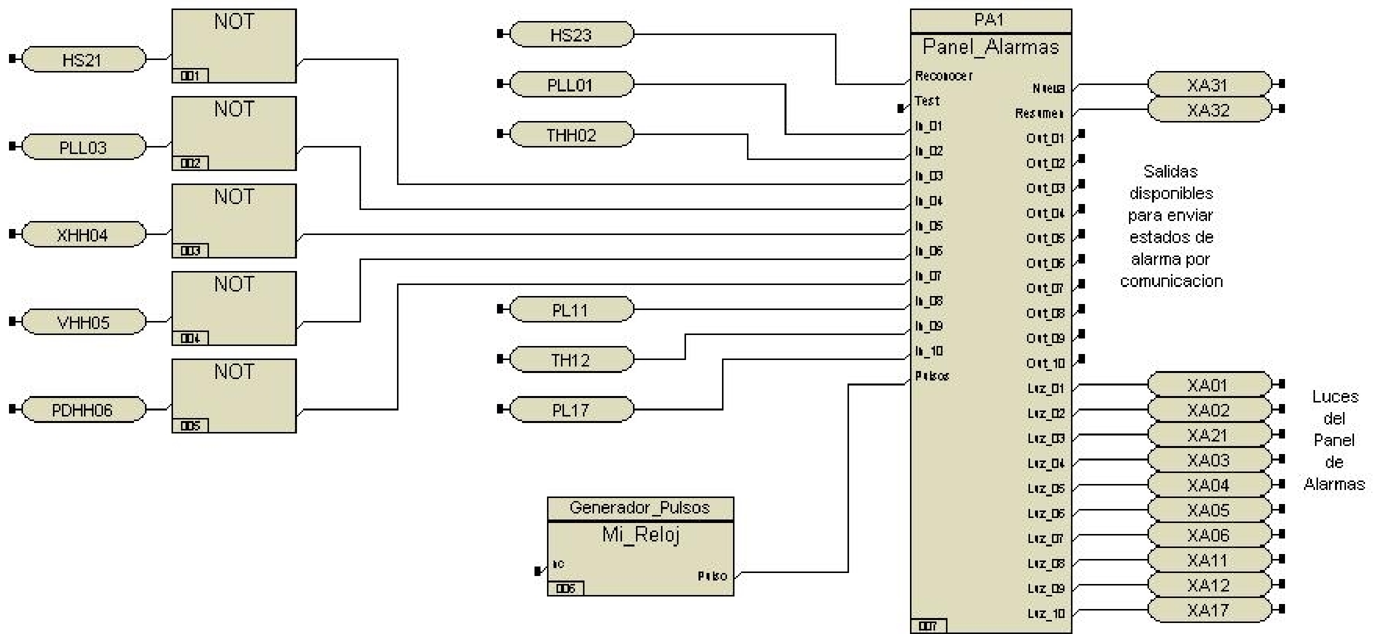




Luego el **programa "Principal"** que tiene lo siguiente:

Boton Partir
HS22
Detener
NOT
PL11
NOT
TH12
NOT
PL17
AND
Partir
Reset_Set
RS
Motor
XM21
Boton Parar
HS21
NOT
PLL01
THH02
NOT
PLL03
NOT
VHH05
NOT
PDHH06
Boton Partir
HS22
XHH04
OR
NOT
OR
Detener

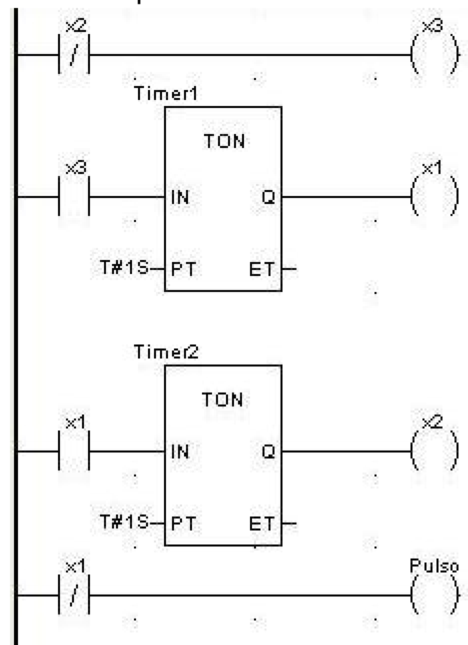
Por ultimo el **programa “Alarmas”** que tiene lo siguiente:



Este programa “Alarmas” llama a las siguientes funciones:

La **función “Mi_Reloj”** no tiene entradas, se usa para generar el pulso para las luces intermitentes.

```
FUNCTION_BLOCK Panel_Alarmas
(* Entradas y salidas de esta función *)
      VAR_INPUT
             (* No se necesitan entradas*)
      END_VAR
      VAR_OUTPUT
             Pulso : BOOL ;
      END_VAR
(* Variables Locales *)
      VAR
             Timer1 : TON;
             Timer2 : TON;
             x1, x2, x3 : BOOL ;
      END_VAR
(* Function Block Body *)
      IF x2 THEN x3:=FALSE; ELSE x3:=TRUE; END_IF;

      Timer1(IN:=x3,PT:=t#1s);
      x1 := Timer1.Q;

      Timer2(IN:=x1,PT:=t#1s);
      x2 := Timer2.Q;

      IF x1 THEN Pulso:=FALSE; ELSE Pulso:=TRUE; END_IF;
END_FUNCTION_BLOCK
```



Con el software Tristation (de Triconex) no existe manera de hacer un ciclo repetitivo. Esta limitación de Triconex es intencional, y debe ser para impedir que por errores de programación, pueda llegar a formarse, en alguna condición, un ciclo sin fin. De hecho los Triconex son los únicos certificados para aplicaciones en que se requiera SIL 3, y gestionan automáticamente su propio auto diagnostico y redundancia.

Por esto, la única forma de hacer la misma lógica para las 10 alarmas, es escribir 10 veces la misma lógica. Sin embargo como esta dentro de una función, si necesitamos mas alarmas podemos llamar varias veces a la misma función. Por ejemplo, si necesitáramos 40 alarmas, podríamos dentro del programa “Alarmas” poner 4 veces la función “Panel_Alarmas”.que tiene la lógica de 10 alarmas por cada vez.

La función “**Panel_Alarmas**” contiene la lógica de las alarmas.

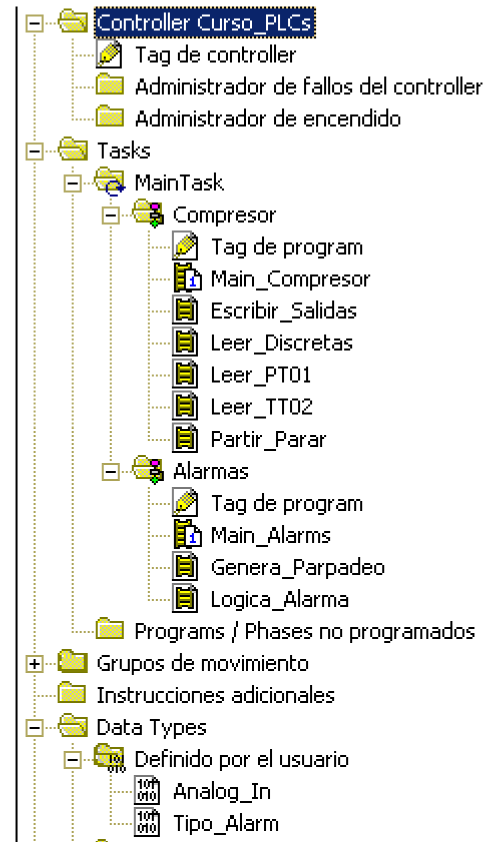
```
FUNCTION_BLOCK Panel_Alarmas

(* Entradas y salidas de esta funcion *)
    VAR_INPUT
      Reconocer, Test : BOOL;
      In_01, In_02, In_03, In_04, In_05, In_06, In_07, In_08, In_09, In_10 :BOOL;
      Pulsos :BOOL;
    END_VAR
    VAR_OUTPUT
      Nueva, Resumen : BOOL;
      Out_01, Out_02, Out_03, Out_04, Out_05, Out_06, Out_07, Out_08, Out_09, Out_10 :BOOL;
      Luz_01, Luz_02, Luz_03, Luz_04, Luz_05, Luz_06, Luz_07, Luz_08, Luz_09, Luz_10 :BOOL;
    END_VAR

(* Variables Locales *)
      VAR
      Det_01, Det_02, Det_03, Det_04, Det_05, Det_06, Det_07, Det_08, Det_09, Det_10 :BOOL;
      Rec_01, Rec_02, Rec_03, Rec_04, Rec_05, Rec_06, Rec_07, Rec_08, Rec_09, Rec_10 :BOOL;
      END_VAR


(* ALARMA  1 *)
IF (In_01 OR Det_01) AND NOT(Rec_01) THEN Det_01:=TRUE; ELSE Det_01:=FALSE; END_IF;
IF (In_01 OR Det_01) AND (Reconocer OR Rec_01) THEN Rec_01:=TRUE; ELSE Rec_01:=FALSE; END_IF;
IF (Det_01 OR Rec_01) THEN Out_01:=TRUE; ELSE Out_01:=FALSE; END_IF;
IF (Det_01 AND Pulsos) OR Rec_01 OR Test THEN Luz_01:=TRUE; ELSE Luz_01:=FALSE; END_IF;


(* ALARMA  2 *)
IF (In_02 OR Det_02) AND NOT(Rec_02) THEN Det_02:=TRUE; ELSE Det_02:=FALSE; END_IF;
IF (In_02 OR Det_02) AND (Reconocer OR Rec_02) THEN Rec_02:=TRUE; ELSE Rec_02:=FALSE; END_IF;
IF (Det_02 OR Rec_02) THEN Out_02:=TRUE; ELSE Out_02:=FALSE; END_IF;
IF (Det_02 AND Pulsos) OR Rec_02 OR Test THEN Luz_02:=TRUE; ELSE Luz_02:=FALSE; END_IF;
```

```
(* ALARMA  3 *)
IF (In_03 OR Det_03) AND NOT(Rec_03) THEN Det_03:=TRUE; ELSE Det_03:=FALSE; END_IF;
IF (In_03 OR Det_03) AND (Reconocer OR Rec_03) THEN Rec_03:=TRUE; ELSE Rec_03:=FALSE; END_IF;
IF (Det_03 OR Rec_03) THEN Out_03:=TRUE; ELSE Out_03:=FALSE; END_IF;
IF (Det_03 AND Pulsos) OR Rec_03 OR Test THEN Luz_03:=TRUE; ELSE Luz_03:=FALSE; END_IF;


(* ALARMA  4 *)
IF (In_04 OR Det_04) AND NOT(Rec_04) THEN Det_04:=TRUE; ELSE Det_04:=FALSE; END_IF;
IF (In_04 OR Det_04) AND (Reconocer OR Rec_04) THEN Rec_04:=TRUE; ELSE Rec_04:=FALSE; END_IF;
IF (Det_04 OR Rec_04) THEN Out_04:=TRUE; ELSE Out_04:=FALSE; END_IF;
IF (Det_04 AND Pulsos) OR Rec_04 OR Test THEN Luz_04:=TRUE; ELSE Luz_04:=FALSE; END_IF;


(* ALARMA  5 *)
IF (In_05 OR Det_05) AND NOT(Rec_05) THEN Det_05:=TRUE; ELSE Det_05:=FALSE; END_IF;
IF (In_05 OR Det_05) AND (Reconocer OR Rec_05) THEN Rec_05:=TRUE; ELSE Rec_05:=FALSE; END_IF;
IF (Det_05 OR Rec_05) THEN Out_05:=TRUE; ELSE Out_05:=FALSE; END_IF;
IF (Det_05 AND Pulsos) OR Rec_05 OR Test THEN Luz_05:=TRUE; ELSE Luz_05:=FALSE; END_IF;


(* ALARMA  6 *)
IF (In_06 OR Det_06) AND NOT(Rec_06) THEN Det_06:=TRUE; ELSE Det_06:=FALSE; END_IF;
IF (In_06 OR Det_06) AND (Reconocer OR Rec_06) THEN Rec_06:=TRUE; ELSE Rec_06:=FALSE; END_IF;
IF (Det_06 OR Rec_06) THEN Out_06:=TRUE; ELSE Out_06:=FALSE; END_IF;
IF (Det_06 AND Pulsos) OR Rec_06 OR Test THEN Luz_06:=TRUE; ELSE Luz_06:=FALSE; END_IF;


(* ALARMA  7 *)
IF (In_07 OR Det_07) AND NOT(Rec_07) THEN Det_07:=TRUE; ELSE Det_07:=FALSE; END_IF;
IF (In_07 OR Det_07) AND (Reconocer OR Rec_07) THEN Rec_07:=TRUE; ELSE Rec_07:=FALSE; END_IF;
IF (Det_07 OR Rec_07) THEN Out_07:=TRUE; ELSE Out_07:=FALSE; END_IF;
IF (Det_07 AND Pulsos) OR Rec_07 OR Test THEN Luz_07:=TRUE; ELSE Luz_07:=FALSE; END_IF;


(* ALARMA  8 *)
IF (In_08 OR Det_08) AND NOT(Rec_08) THEN Det_08:=TRUE; ELSE Det_08:=FALSE; END_IF;
IF (In_08 OR Det_08) AND (Reconocer OR Rec_08) THEN Rec_08:=TRUE; ELSE Rec_08:=FALSE; END_IF;
IF (Det_08 OR Rec_08) THEN Out_08:=TRUE; ELSE Out_08:=FALSE; END_IF;
IF (Det_08 AND Pulsos) OR Rec_08 OR Test THEN Luz_08:=TRUE; ELSE Luz_08:=FALSE; END_IF;


(* ALARMA  9 *)
IF (In_09 OR Det_09) AND NOT(Rec_09) THEN Det_09:=TRUE; ELSE Det_09:=FALSE; END_IF;
IF (In_09 OR Det_09) AND (Reconocer OR Rec_09) THEN Rec_09:=TRUE; ELSE Rec_09:=FALSE; END_IF;
IF (Det_09 OR Rec_09) THEN Out_09:=TRUE; ELSE Out_09:=FALSE; END_IF;
IF (Det_09 AND Pulsos) OR Rec_09 OR Test THEN Luz_09:=TRUE; ELSE Luz_09:=FALSE; END_IF;


(* ALARMA  10 *)
IF (In_10 OR Det_10) AND NOT(Rec_10) THEN Det_10:=TRUE; ELSE Det_10:=FALSE; END_IF;
```

```
IF (In_10 OR Det_10) AND (Reconocer OR Rec_10) THEN Rec_10:=TRUE; ELSE Rec_10:=FALSE; END_IF;
IF (Det_10 OR Rec_10) THEN Out_10:=TRUE; ELSE Out_10:=FALSE; END_IF;
IF (Det_10 AND Pulsos) OR Rec_10 OR Test THEN Luz_10:=TRUE; ELSE Luz_10:=FALSE; END_IF;


(*  RESUMEN  *)
IF Det_01 OR Det_02 OR Det_03 OR Det_04 OR Det_05 OR Det_06 OR Det_07
    OR Det_08 OR Det_09 OR Det_10 THEN Nueva:=TRUE; ELSE Nueva:=FALSE; END_IF;


IF Out_01 OR Out_02 OR Out_03 OR Out_04 OR Out_05 OR Out_06 OR Out_07
    OR Out_08 OR Out_09 OR Out_10 THEN Resumen:=TRUE; ELSE Resumen:=FALSE; END_IF;

END_FUNCTION_BLOCK
```

Solución con ControlLogix, de Rockwell

Para PLCs PLC-5, SLC500 o ControlLogix



ControlLogix permite varios programas y cada uno de estos contener a su vez sub programas.

Dentro de la configuración de la tarea (Task) se define en que orden se ejecutan los programas pertenecientes a la tarea.

Cada programa tiene definido cual es el único sub programa inicial, el que debe llamar a los otros sub programas, si se quiere que los otros sub programas se ejecuten.

De esta manera se puede usar un sub programa como si fuera una función, llamándolo varias veces, pasando le cada vez datos distintos, y leyendo cada vez el resultado.

Existen variables globales "Tag controller" dentro de las cuales están las entradas y salidas físicas junto con todas las variables de diagnostico del sistema. Y las variables que creamos para ser visibles por todos los programas.

Cada programa tiene sus propias variables locales visibles por todos sus sub programas, pero inaccesibles desde otros programas.

En este ejemplo los Tag del programa Compresor, y los Tag del programa Alarmas.

En este ejemplo se han definido dos tipos de estructuras de datos, los llamados "Analog_In" y los llamados "Tipo_Alarm".

En este ejemplo, se definen las variable locales (del programa "Compresor"") PT01 y TT02 ambas del tipo "Analog_in".

Esto permite tener agrupado, el rango, el valor de entrada, el valor de ingeniería y es estado del canal, en una misma variable. Ver la imagen siguiente.

Name	Value	Force Mas	Style	Data Type	Description
PDHH06	1		Decimal	BOOL	Dif Succion Descarga
PL01	0		Decimal	BOOL	Presion Aceite
PL17	1		Decimal	BOOL	Agua Enfriamiento
PLL01	0		Decimal	BOOL	Presion Aceite
⊟-PT01	{...}	{...}		Analog_In	Presion Aceite
PT01.Minimo	0.0		Float	REAL	Presion Aceite
PT01.Maximo	10.0		Float	REAL	Presion Aceite
PT01.Valor	3.75		Float	REAL	Presion Aceite
PT01.Volt_In	2.5		Float	REAL	Presion Aceite
PT01.OK	1		Decimal	BOOL	Presion Aceite
⊟-TT02	{...}	{...}		Analog_In	Temp Descarga
TT02.Minimo	0.0		Float	REAL	Temp Descarga
TT02.Maximo	200.0		Float	REAL	Temp Descarga
TT02.Valor	65.0		Float	REAL	Temp Descarga
TT02.Volt_In	2.3		Float	REAL	Temp Descarga
TT02.OK	1		Decimal	BOOL	Temp Descarga
VHH05	1		Decimal		
XHH04	1		Decimal		
XM21	1		Decimal		

Name	Value	Data Type
Alarm_Nueva	0	BOOL
Alarm_Presente	0	BOOL
⊟-Alarma	{...}	Tipo_Alarm[100]
⊞-Alarma[0]	{...}	Tipo_Alarm
⊞-Alarma[1]	{...}	Tipo_Alarm
⊟-Alarma[2]	{...}	Tipo_Alarm
Alarma[2].Entrada	0	BOOL
Alarma[2].Detectada	0	BOOL
Alarma[2].Reconocida	0	BOOL
Alarma[2].Salida	0	BOOL
Alarma[2].Luz	0	BOOL
⊞-Alarma[3]	{...}	Tipo_Alarm
⊞-Alarma[4]	{...}	Tipo_Alarm
⊞-Alarma[5]	{...}	Tipo_Alarm
⊞-Alarma[6]	{...}	Tipo_Alarm
⊟-Alarma[7]	{...}	Tipo_Alarm
Alarma[7].Entrada	0	BOOL
Alarma[7].Detectada	0	BOOL
Alarma[7].Reconocida	0	BOOL
Alarma[7].Salida	0	BOOL
Alarma[7].Luz	0	BOOL
⊞-Alarma[8]	{...}	Tipo_Alarm
⊞-Alarma[9]	{...}	Tipo_Alarm
⊞-Alarma[10]	{...}	Tipo_Alarm
⊞-Alarma[11]	{...}	Tipo_Alarm
⊞-Alarma[12]	{...}	Tipo_Alarm
⊞-Alarma[13]	{...}	Tipo_Alarm
⊞-Alarma[14]	{...}	Tipo_Alarm

En este ejemplo, se definen la variable global "Alarma" como un arreglo de datos del tipo "Tipo_Alarm". Ver la imagen de la derecha

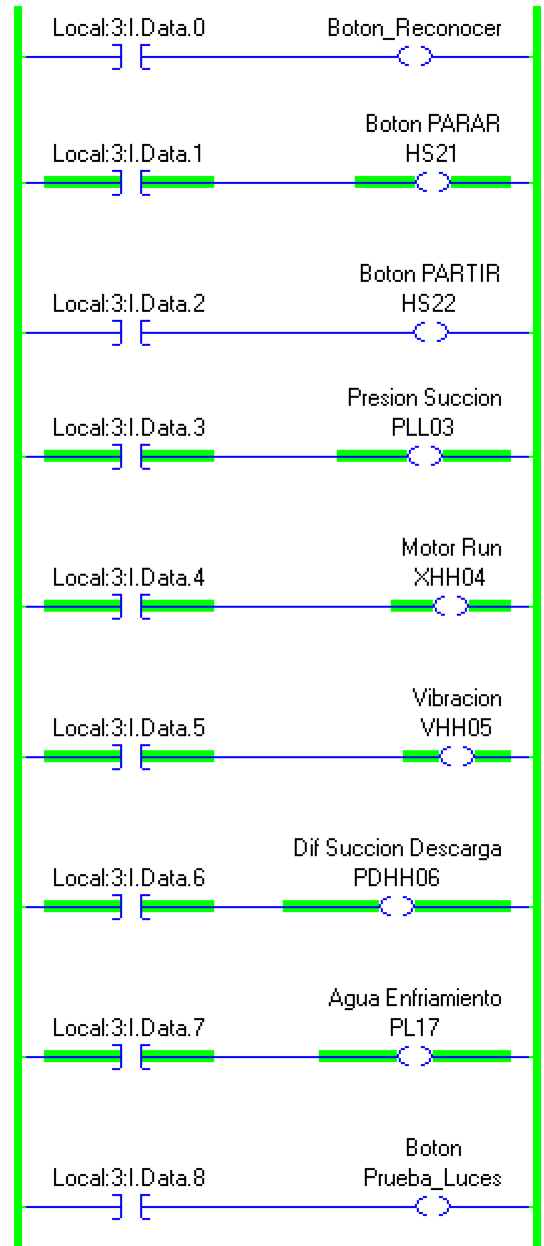
El arreglo de alarmas es variable global, por que aunque toda la logica y procesamiento de alarmas esta dentro del programa "Alarmas", las Alarma[x].Entrada serán activadas desde el programa "Compresor"

Esto permite separar la logica de funcionamiento del compresor, de la logica de procesamiento de las alarmas del mismo, tener una estructura mas ordenada, y no tener que crear tantas variables individuales.

Lo primero que se ejecuta es el **programa "Compresor"** y el único sub programa de este que se ejecuta automáticamente es "Main_compresor".   Todos los demás sub programas deben ser explicita mente llamados por la lógica

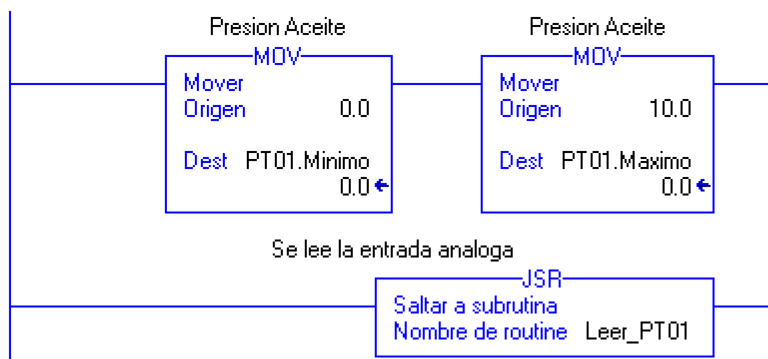
Lo primero que hace el "Main_compresor" es llamar al sub programa "Leer_discretas".

El sub programa "Leer_discretas" (imagen de la derecha) conecta los canales físicos de entrada, definidos en la configuración del hardware, con las variables discretas correspondientes.   Ver imagen de la derecha.



Cuando termina el sub programa "Leer_discretas" continua la ejecución del "Main_compresor"

Define el rango para la entrada análoga del transmisor de presión (PT...), y luego llama al sub programa "Leer_PT01"



El sub programa "Leer_PT01"

Conecta el canal fisico de entrada con la variable correspondiente.

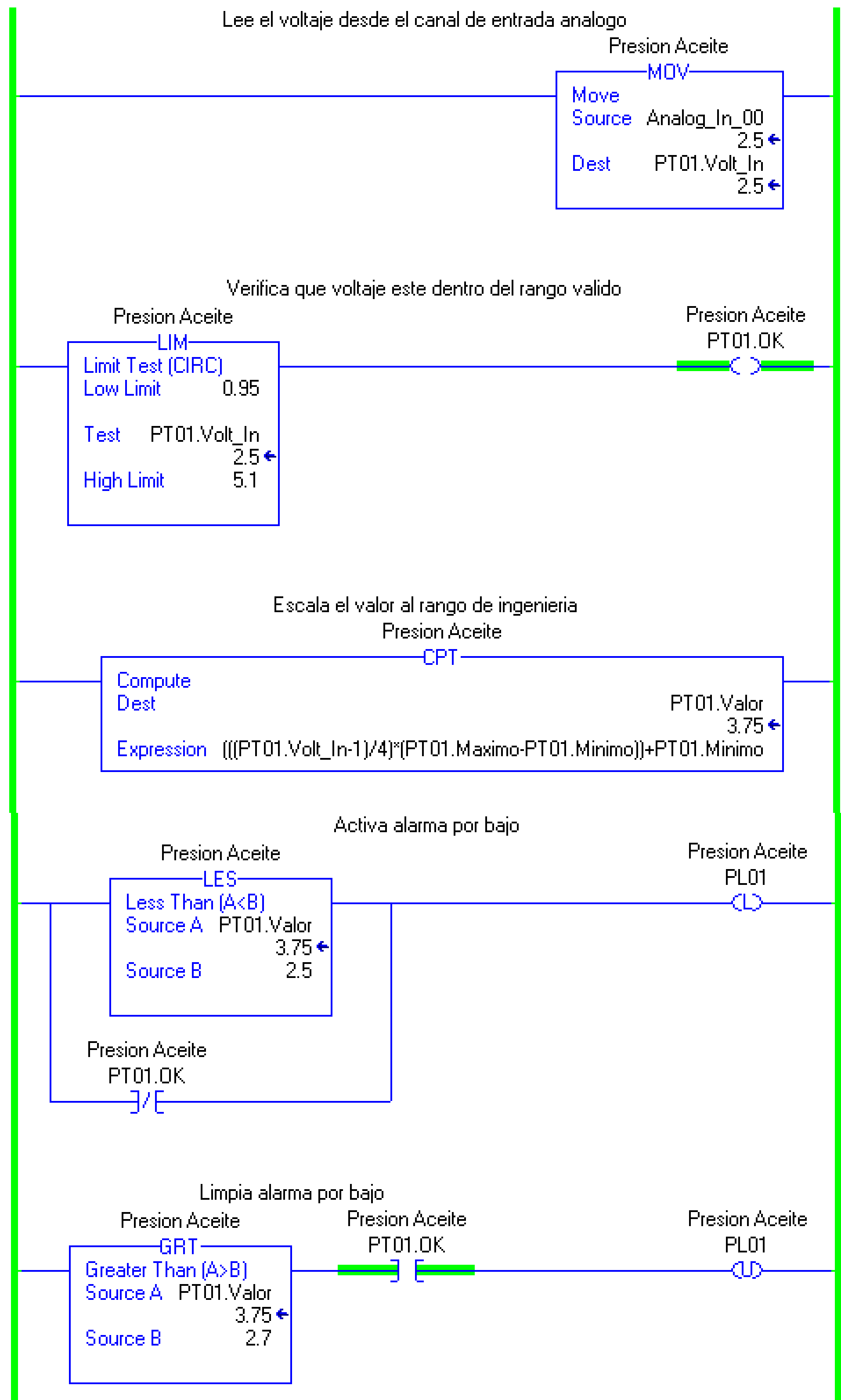
El canal analógico de entrada esta configurado para leer una señal de 0 a 20 miliamperes, como 0 a 5 volt.

Si se verifica que la señal es mayor a 0,95 volt y es menor a 5,1 volt, se considera que esta dentro del rango normal y se activa la variable PT01.OK

Se escala la señal al rango de ingeniería, anteriormente definido

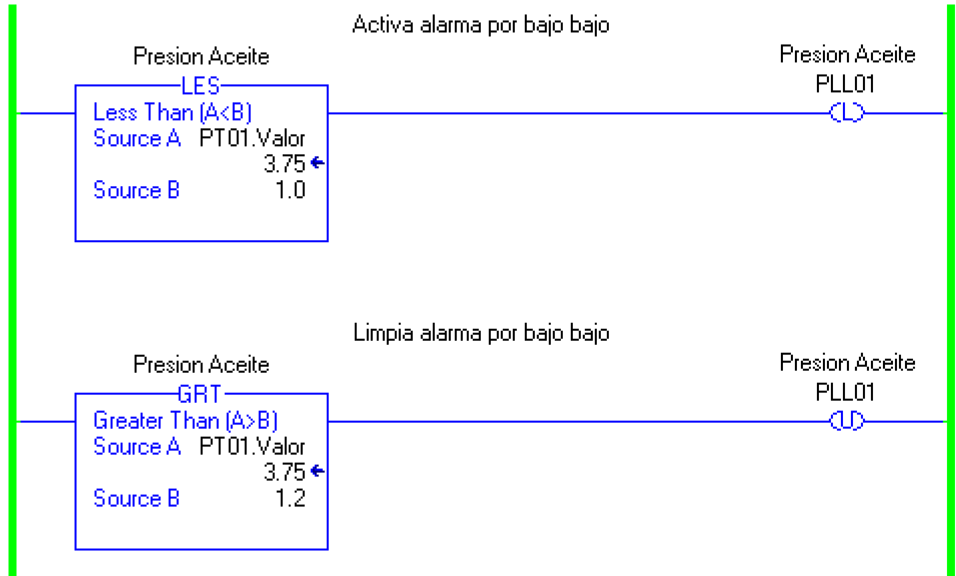
Se activa la alarma de baja presión si su valor es menor a …… O si la señal no esta OK

Se apaga la alarma de baja presión si su valor es mayor a …… Y si la señal si esta OK



Se activa la alarma de muy baja presión si su valor es menor a ……

Se apaga la alarma de muy baja presión si su valor es mayor a ……



Cuando termina el sub programa “Leer_PT01” continua la ejecución del “Main_compresor”, que continua definiendo el rango para la entrada análoga del transmisor de temperatura (TT...), y luego llama al sub programa “Leer_TT02”

Temp Descarga
MOV
Mover
Origen 0.0
Dest TT02.Minimo
0.0

Temp Descarga
MOV
Mover
Origen 200.0
Dest TT02.Maximo
0.0

Se lee la entrada analoga
JSR
Saltar a subrutina
Nombre de routine Leer_TT02

Lee el voltaje desde el canal de entrada analogo
Temp Descarga
MOV
Move
Source Analog_In_01
2.3
Dest TT02.Volt_In
2.3

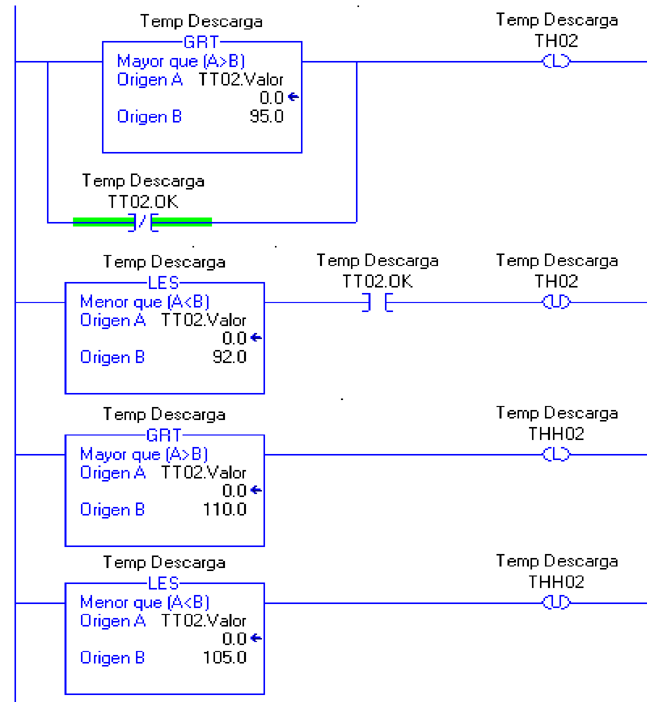
El sub “Leer_TT02” hace con la señal de temperatura, lo mismo que hizo “Leer_PT01” con la de presión.

Verifica que este dentro de su rango, y escalarla al rango de ingenieria.

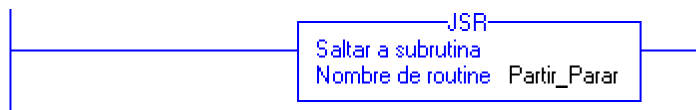
Luego la compara con los valores predefinidos de alarma, para activar o apagar las alarmas.

Verifica que voltaje este dentro del rango valido
Temp Descarga
LIM
Limit Test (CIRC)
Low Limit 0.95
Test TT02.Volt_In
2.3
High Limit 5.1
Temp Descarga
TT02.OK

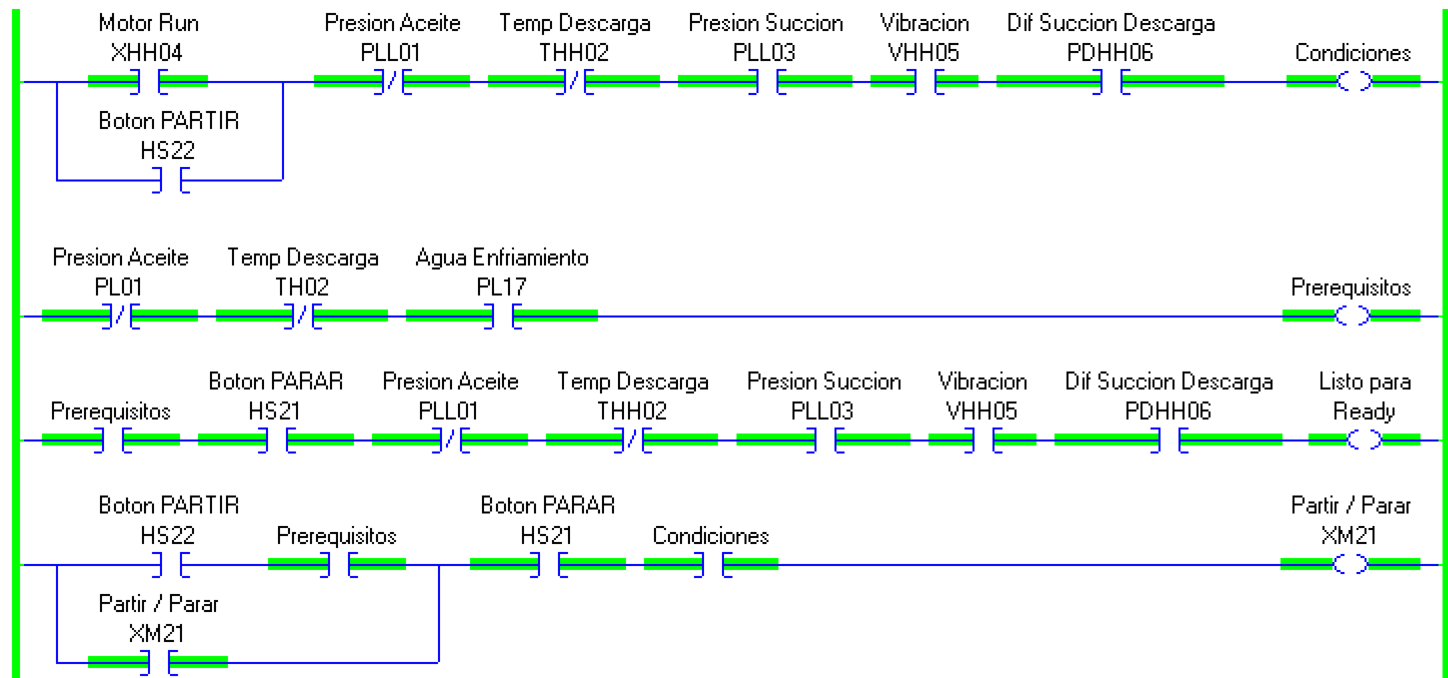
Escala el valor al rango de ingenieria
Temp Descarga
CPT
Compute
Dest TT02.Valor
65.0
Expression (((TT02.Volt_In-1)/4)*(TT02.Maximo-TT02.Minimo))+TT02.Minimo



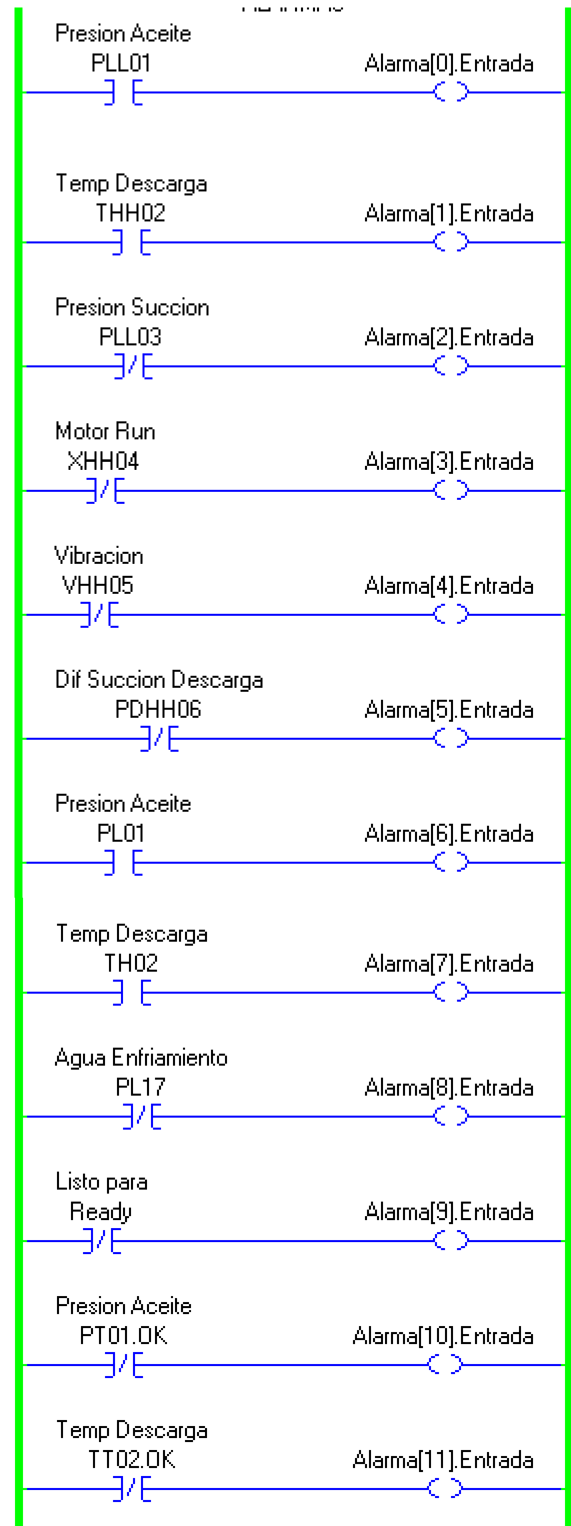
Cuando termina el sub programa “Leer_TT02” continua la ejecución del “Main_compresor” llamando al sub programa “Partir_parar”



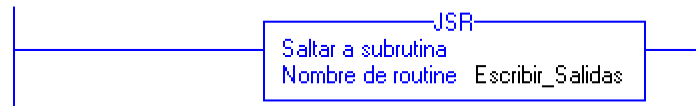
En “Partir_parar” se decide el estado que se enviara hacia el motor del compresor.



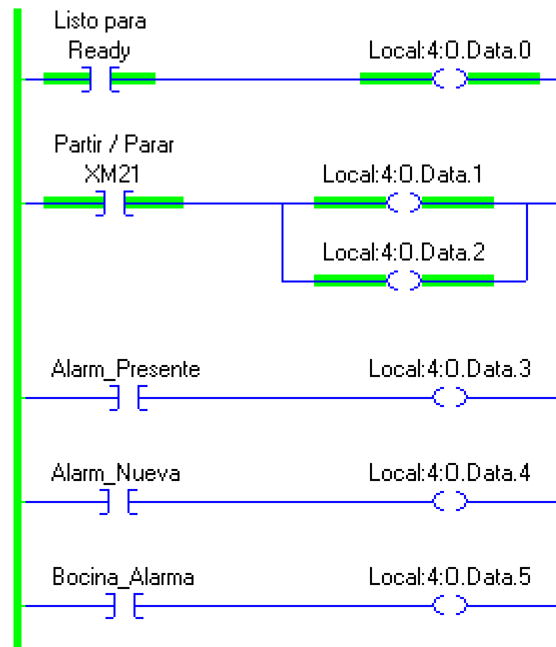
A continuación, en este mismo sub programa, están conectados los estados o condiciones, a las correspondientes alarmas que se activaran.



Cuando termina el sub programa “Partir_parar” continua la ejecución del “Main_compresor”   llamando al sub programa “Escribir_salidas”



En “Escribir_salidas” se conectan las variables discretas con los canales físicos de salida correspondientes, definidos en la configuración del hardware.

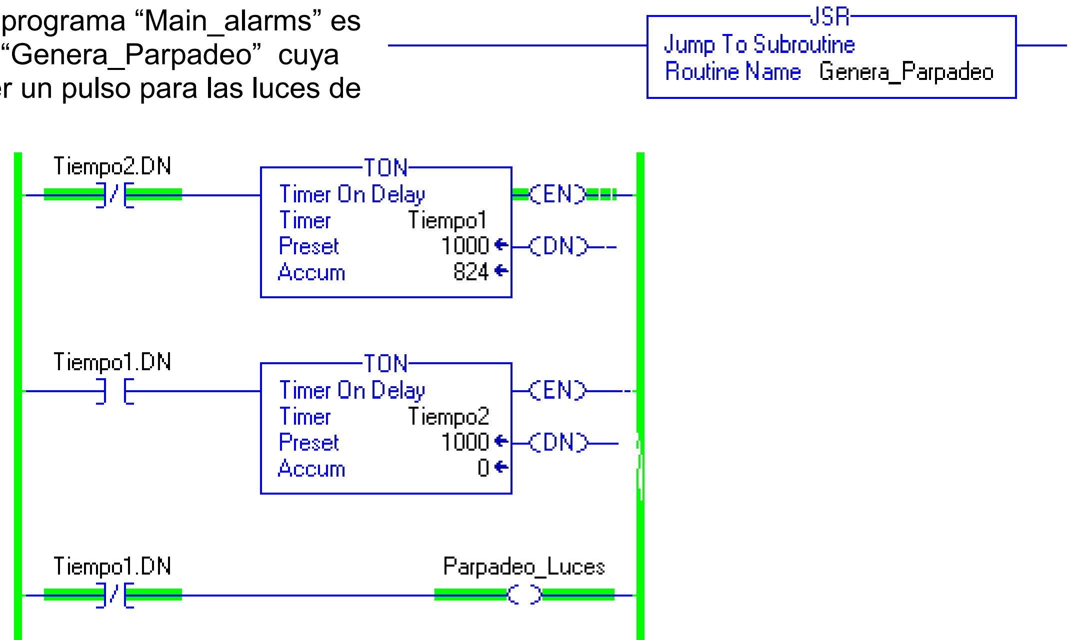


Note ce que la misma variable “Partir/Parar” puede conectarse a uno, a dos o a tres canales de salida, simultáneamente.   Esto podría usarse para aumentar la seguridad, ante la posibilidad de falla del canal de salida del plc, por ejemplo: Si la lógica conecta la variable de salida a a tres salidas físicas, cada una con un rele distinto. Y estos tres reles se alambran en una conexión 2 de 3 para activar el motor.   Las tres salidas tendrían que ser de módulos distintos del plc, por si falla un modulo completo.   Y tendría que haber tres entradas discretas monitoreando el estado de estos tres reles, para poder darse cuenta, y alarmar (falla de redundancia en plc), si estos no concuerdan entre si.

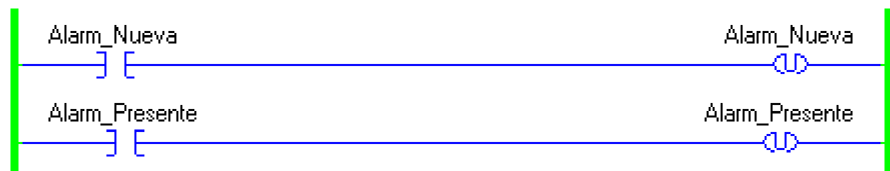
Los mismos criterios podrian usarse para señales de entradas redundantes al plc.

Después del programa “Compresor” se ejecuta es el **programa “Alarmas”** y el único sub programa de este que se ejecuta automáticamente es “Main_alarms”.   Todos los demás sub programas deben ser explicita mente llamados por la lógica

Lo primero que hace el programa “Main_alarms” es llamar al sub programa “Genera_Parpadeo” cuya única función es obtener un pulso para las luces de alarma.

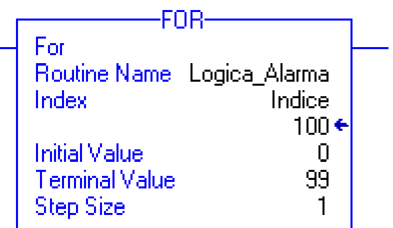


Luego el programa “Main_alarms” limpia (pone en falso) las variables que son resumen de alarmas.



Y después llama a ejecutar 100 veces el mismo sub programa, pero cada vez con datos distintos.

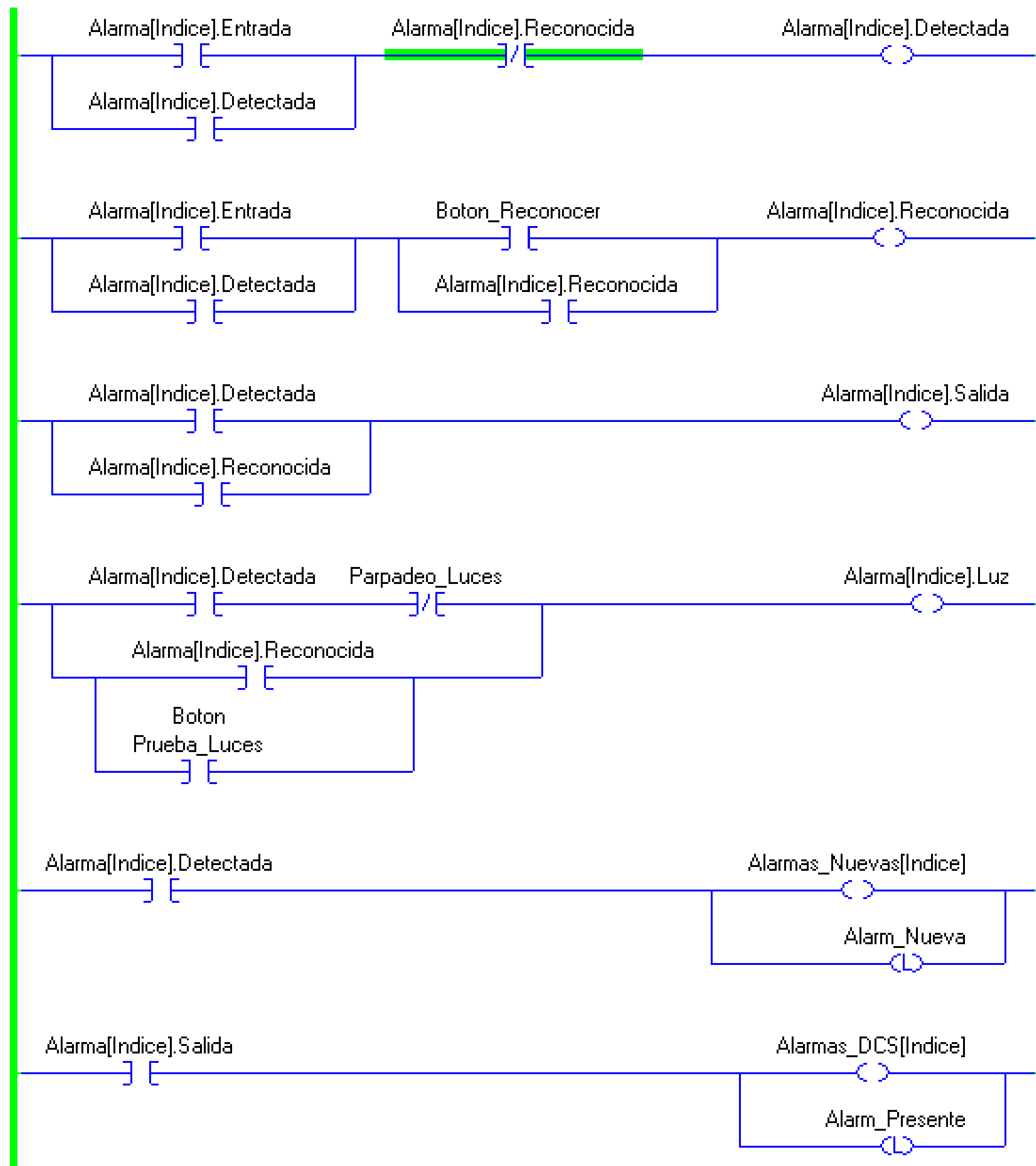
La primera vez con indice = 0, después con indice = 1, después con indice = 2, etc … Hasta indice = 99.



Esto permite ejecutar la misma lógica para la alarma 0, para la alarma 1, para la alarma 2 , etc...

Después de ejecutar esta misma lógica por cada una de las alarmas, verifica si alguna de ellas volvió a activar (poner en verdadero) las variables que son resumen de alarmas.

El sub programa “Logica_Alarma” contiene lo siguiente”



En la penúltima linea si la alarma esta activa y aun no ha sido reconosida, activa el resumen de alarma nueva.   Y también copia esta condición al arreglo de booleanos “Alarmas_Nuevas”.

En la última linea si la alarma esta presente, reconocida o no, activa el resumen de alarma presente.   Y también copia esta condición al arreglo de booleanos “Alarmas_DCS”.   Este arreglo podría ser enviado por comunicación a un DCS.

Si has leído hasta aquí,   gracias por tu paciencia.

Si crees que este documento le podría interesar a alguien mas,   reenvía se lo.

Cualquier sugerencia me pueden encontrar en   rolfds@gmail.com