
Institución
Universitaria
PROGRAMACIÓN DE CONTROLADORES LÓGICOS PROGRAMABLES CON LENGUAJE SCL
Juan Guillermo Mejía Arango

Programación de controladores lógicos programables con lenguaje SCL

Programación de controladores lógicos programables con lenguaje SCL

Juan Guillermo Mejía Arango

Mejía Arango, Juan Guillermo

Programación de controladores lógicos programables con lenguaje SCL / Juan Guillermo Mejía Arango

-- Medellín: Instituto Tecnológico Metropolitano, 2022.

155 p. -- (Teknik)

Incluye referencias bibliográficas

Controladores lógicos programables. 2. SCL (lenguaje de programación). I. Tit. II. Serie

629.8 SCDD 21 ed.

Catalogación en la publicación - Biblioteca ITM

Primera edición: noviembre de 2022

ISBN: 978-958-5122--73-4 (PDF)

Hecho en Medellín, Colombia

EDICIÓN

Fondo Editorial ITM

Calle 73 76ª-354

Telefono: (574) 604 440 5100
Ext. 5197-5382

catalogo.itm.edu.co

fondoeditorial@itm.edu.co

Instituto Tecnológico Metropolitano

Medellín, Colombia

EQUIPO EDITORIAL

Juliana Cardona Quiros

Directora editorial

Sebastián Vásquez Moreno

Profesional universitario FEITM

Catalina Ocampo Ocampo

Gustavo Otálvaro Ocampo

Editores de mesa

Martha Cecilia Caballero Jerez

Corrección de textos

Mauricio Raigosa Álvarez

Diseño y diagramación

Cómo citar:
Mejía Arango, J. G. (2022). *Programación de controladores lógicos programables con lenguaje SCL*. Fondo Editorial ITM.

CONTENIDO

Agradecimientos

A mi familia, esposa e hijos, que me inspiran con su dedicación y búsqueda permanente del conocimiento y exploración del mundo, animándome a seguir construyendo y dejando una huella en este trasegar por la vida.

A mis alumnos, destinatarios principales de este texto, que con su interés hacen que todos los días aparezcan nuevas ideas y retos, llenando de sentido mi vida.

A la familia ITM, que pone a nuestra disposición herramientas para que construyamos a través de los sueños y deseos de superación.

Juan Guillermo Mejía Arango

Introducción

En este libro se hace una introducción a la programación de controladores lógicos programables, haciendo uso del lenguaje conocido como SCL (Structured Control Language), bajo el software de programación TIA PORTAL de Siemens, incorporando los conceptos teóricos y desarrollo de problemas de aplicación.

Dentro de los lenguajes de programación normalizados para la programación de los controladores lógicos programables (PLC) se tienen: el lenguaje de contactos (KOP), el de diagrama de funciones (FUP), el de lista de instrucciones (AWL), el S7-GRAPH y el SCL. Los primeros cuatro son de uso general, son más utilizados en los cursos introductorios de automatización, por lo que se dispone de varias fuentes de consulta para estudiarlos y practicarlos. En cuanto al lenguaje SCL, su uso ha sido menos frecuente y la disponibilidad de material de consulta ha sido más limitada, especialmente en su aplicación. No obstante, el SCL ofrece grandes posibilidades de programación cuando se requieren algoritmos más complejos en los que se tenga que utilizar mayor cantidad de fórmulas, procesar mucha información en forma de datos y plantear funciones matemáticas más complejas. Esta publicación pretende facilitar la comprensión del lenguaje SCL y su dominio para aplicarlo a soluciones de automatización con los PLC.

La estrategia que se desarrolla a lo largo del texto consiste en el planteamiento teórico de los temas que se consideraron más relevantes, seguido del planteamiento de problemas que son un reflejo de situaciones que se presentan en la industria para el control y automatización de procesos productivos y de fabricación. Para cada problema se sugiere una solución desde el punto de vista del autor, que pretende ante todo llevar a la práctica los temas explicados; no obstante, la solución a dichos problemas puede tener otros caminos que posiblemente conduzcan a un algoritmo más práctico y sencillo. Las soluciones presentadas han sido verificadas en el software de TIA PORTAL. Parte de los problemas se ha solucionado para PLC de la familia S7-300 y otros para PLC de la familia S7-1200.

Se tiene como finalidad que el lector de este libro pueda adquirir los conocimientos necesarios para dominar los componentes esenciales del lenguaje SCL, con el fin de usarlos en la solución de los problemas reales que se puedan encontrar en el entorno industrial. Sería preferible que el lector ya tenga unos conceptos previos del entorno de programación en TIA PORTAL, puesto que no es tratado en profundidad en este libro. Entonces, se requiere que previamente se domine la configuración de hardware, se conozca el proceso para el ingreso de un programa, la simulación, la conexión del PLC y discriminar entre la memoria de entradas, de salidas y marcas.

El libro se dividió en seis capítulos. En el primero se realiza una conceptualización general de la programación en SCL, destacando su significado, sus ventajas y descripción del entorno en STEP7. En el capítulo 2 se definen los componentes de un programa en SCL, explicando lo que son las expresiones, definiendo las palabras reservadas para programación, clasificando las áreas de memoria del PLC conformadas por entradas, salidas, marcas, los bloques de datos y la periferia de entrada y de salida. Por último, se explican los conceptos y tratamiento de los números, los tipos de datos y las variables.

En el capítulo 3 se trabaja con las instrucciones generales y de control, destacándose las instrucciones de asignación, de asignación con tiempo y fecha, las instrucciones de control IF, THEN, ELSIF, ELSE, END_IF, FOR, WHILE, REPEAT, CONTINUE, EXIT, GOTO y RETURN. El capítulo 4 está dedicado a los temporizadores presentados en el formato SCL, considerándose los más generalizados: el temporizador como impulso o TP, el temporizador con retardo a la conexión o TON, el temporizador con retardo a la desconexión o TOF y la función de contaje de tiempo. En el capítulo 5 se hace un tratamiento de los contadores en formato SCL. Finalmente, el capítulo 6 está dedicado a profundizar en el acceso a la información, haciendo un tratamiento especial de los bloques de datos, el direccionamiento indirecto de un programa en SCL, trabajando con el área de memoria de las bases de datos, el área de memoria de entradas, salidas y marcas, la programación en bloques FC y FB, usando su memoria interna para el almacenamiento y manejo de la información.

En los capítulos 3, 4 y 5 las aplicaciones se realizan utilizando especialmente la memoria de entradas, salidas y marcas, por medio de un direccionamiento “simbólico”, para que el lector se concentre en el entendimiento de las instrucciones SCL. En el capítulo 6 las aplicaciones se extienden a otras áreas de memoria disponibles en los bloques de datos y en el interior de los bloques FC y FB, con lo que se comprende que puede haber otras técnicas de acceso a la memoria, más eficientes, que potencian las aplicaciones que se pueden realizar con la programación en SCL.

01. CONCEPTOS GENERALES DE LA PROGRAMACIÓN EN SCL

1. Conceptos generales de la programación en SCL

1.1. ¿Qué es SCL?

SCL significa “Structured Control Language”, que está acorde con la norma IEC 1131-3 que define los lenguajes de programación para los controladores lógicos programables,donde se establece, entre otros, el lenguaje ST “Structured Text” (Öhman, 1995), teniendo este unas características especiales que lo identifican, como una redacción parecida al lenguaje de alto nivel de Pascal y otros como C y C++, pudiéndose utilizar instrucciones para programas con:

- Condicionales: IF
- Secuencias: CASE
- Rutinas repetitivas: FOR, WHILE Y REPEAT
- Funciones booleanas: AND, OR, entre otras

1.2. Ventajas del lenguaje SCL

La redacción de las rutinas ocupa menos espacio que la de otros lenguajes de programación como el de contactos, que permite el diseño de programas más complejos que requieran el uso de funciones matemáticas, procesamiento estadístico y el manejo de información, especialmente en bases de datos. En su programación se utilizan variables que se pueden enlazar con entradas y salidas, tanto digitales como análogas. El SCL posibilita la escritura de programas fáciles de entender, claros y concisos (Goderie, 2019). Es útil en aquellos programas donde se deben incluir rutinas de control más avanzadas que requieran de gran número de funciones matemáticas como controladores PID, inteligencia artificial o lógica difusa. Luego de tener conocimientos básicos en lenguajes de alto nivel, la asimilación del lenguaje SCL es relativamente fácil.

El manejo de datos es otra de las ventajas del lenguaje SCL; por ejemplo, se facilita el manejo de información en matrices, el filtrado de señales, el intercambio de información de las variables de proceso y aplicaciones de gestión de la información, entre otras tareas (Peciña, 2018). El lenguaje SCL se entiende con elementos propios del PLC como: entradas, salidas, temporizadores, contadores; además, se pueden combinar con bloques escritos en KOP y FUP (Siemens, 2018). La labor de copiar y pegar programas se facilita debido a que se manipula texto, mostrando errores de sintaxis a medida que se introduce el programa en TIA PORTAL.

1.3. Entorno del SCL con el STEP7

Para integrar el SCL al STEP 7 se requiere de un editor/compilador y un depurador. Un editor crea y edita los textos del programa SCL en programas para STEP7;

mientras se escribe el texto se depura la sintaxis y se advierte de errores por medio de colores. Por su parte, el depurador puede ejecutar el programa y encontrar errores en la lógica. Es posible, en este entorno, observar el desarrollo del programa en forma continua o paso a paso.

En STEP7 el programa está incluido en bloques que pueden ser ejecutados en diferentes secuencias y alternativas. Así, un programa completo del PLC puede estar incluido en subprogramas escritos en diferentes bloques. En la Figura 1 se muestra un programa escrito en SCL dentro de un bloque llamado FC1.

Figura 1.

Programa SCL escrito en un bloque FC1

```
Bloque FC1
1 //Ejemplo SCL
2 //Activar una salida con una entrada
3 IF “Entrada0”=1 THEN
4     “Salida0” := 1;
5 ELSE
6      “Salida0” := 0;
7 END_IF;
```

Fuente: elaboración propia, adaptado de TIA PORTAL.

Los programas en SCL se pueden escribir en diferentes tipos de bloques siguiendo el formato de subrutinas. En la Tabla 1 se muestran los tipos de bloques que comprenden un proyecto en STEP7.

Tabla 1.

Tipos de bloques asociados a un proyecto en STEP7

Encabezado	Nombre del bloque	Descripción
OB	Bloque de organización	Pertenecen al sistema operativo y se enlazan con el programa realizado por el usuario.
FC	Función	Bloque para alojar un programa genérico. No tienen asociadas directamente bases de datos.
FB	Bloque de función	Alojan programas y tienen asociadas bases de datos.
DB	Bloque de datos	Almacenan los datos generados en el proceso.
UDT	Tipo de datos de usuario	Almacenan datos cuyo tipo es definido por el usuario.

Fuente: elaboración propia, adaptado de TIA PORTAL.

Los bloques de organización (OB) permiten la ejecución de programas de acuerdo con el estado de funcionamiento del PLC: al iniciar, en forma cíclica, por interrupciones, entre otras. En la Tabla 2 se muestran los bloques de organización válidos para una CPU s/ 1200 de Siemens.

Tabla 2.

Bloques de organización en un PLC S71200 de Siemens

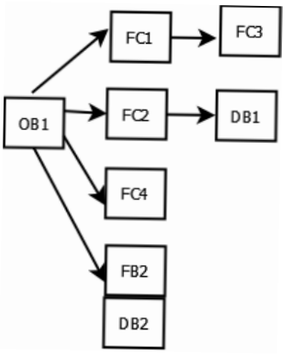
Bloques de organización STEP7	
Número	**Descripción**
OB1, o OB>123	OB de ciclo; no hay que llamarlo, se ejecutan en orden de numeración.
OB10 al OB17	Bloque de organización de alarma horaria. Se activan a una hora determinada o en forma periódica.
OB20 al OB23	OB de alarma de retardo. Interrumpen el procesamiento cíclico cuando se supera un tiempo de retardo definido.
OB30 al OB38	Los OB de interrupción cíclica inician el programa en intervalos periódicos, por ejemplo, cada 100 ms.
OB40 al OB47	Los OB de interrupción de hardware interrumpen el programa por una falla en el hardware.
OB55	OB de alarma de estado. Se activa cuando recibe una alarma de estado, por ejemplo, cuando un módulo de un esclavo cambia de estado.
OB56	Alarma de actualización. Cuando se recibe una alarma de actualización como cambiar un parámetro en un esclavo o dispositivo.
OB57	Alarma del fabricante o perfil luego de recibir una alarma del fabricante o perfil.
OB80	OB de error de tiempo; interrumpen el programa cuando se supera el tiempo máximo de ciclo definido en las propiedades de la CPU.
OB82	OB de alarma de diagnóstico; interrumpen el procesamiento cíclico cuando el módulo seleccionado para diagnóstico detecta un error.
OB83	El OB de presencia de módulo; se activa cuando detecta la incorporación o el retiro de un módulo.
OB86	Se activa cuando hay una falla en el rack.
OB100 al OB 102	OB de arranque; se ejecuta solamente durante el arranque de la CPU, luego se ejecuta el OB1 de ciclo.

Fuente: elaboración propia, adaptado de TIA PORTAL.

En la Figura 2 se presenta un ejemplo de llamadas entre bloques de un programa de STEP7. En él se observa que del OB1 se llaman directamente los bloques de programa FC1, FC2, FC4 y FB2; del FC1 se llama el FC3; el FC2 invoca los datos del DB1, y el FB2 tiene asociados directamente los datos del DB2. Recuerde que en cada bloque de programación puede ir código de programa. El usuario es libre de definir en qué forma va a repartir el código de programa entre los diferentes bloques que tiene disponibles, por ejemplo, todo un programa puede estar codificado en el OB1; sin embargo, dependiendo de la complejidad se podría dificultar el seguimiento de este. La ejecución cíclica del programa empieza con la activación del tiempo de vigilancia, luego se actualiza la memoria imagen de proceso de las salidas, paso seguido se lee la memoria imagen de las entradas para, posteriormente, ejecutar el programa de usuario. El ciclo empieza nuevamente (Siemens, 2016).

Figura 2.

Ejemplo de llamadas entre bloques en el STEP7



Fuente: elaboración propia.

02. COMPONENTES DE UN PROGRAMA SCL

2. Componentes de un programa SCL

2.1.Expresiones

Devuelven un valor que se calcula durante la ejecución del programa y se componen de operandos y, por lo general, operadores. Un operando puede ser, por ejemplo, un número, una variable o una constante, mientras que los operadores pueden ser los que determinan una operación a realizar como suma, resta, multiplicación y división, entre otras.

- Las expresiones pueden ser aritméticas, lógicas o de comparación.
- Operadores de expresiones aritméticas: +, -, *, /, MOD (función módulo) y ** (potencia).
- Las expresiones de comparación están conformadas por: <,>, <=,>=, =, <> (diferente).
- Las expresiones lógicas incluyen: NOT, AND (&), OR, XOR.
- También se incluyen las operaciones de paréntesis () y asignación (:=).
- Algunos ejemplos de asignación de valor en SCL son:
- x := y; // A la variable x se le asigna el valor de la variable y.
- x + = y; // A la variable x se le suma el valor de y.
- x := x+y; // Equivale a la expresión anterior.

  Observe que las instrucciones terminan con punto y coma “;”.

  El símbolo “//” se utiliza para hacer comentarios (no es ejecutado por el programa).

Problema 1.1. Ejemplo escrito en TIA PORTAL para un PLC S7 1200. Se quiere realizar la suma de las variables “B” y “C” como real, el resultado se depositará en la variable “A”.

Solución: primero se declaran las variables en la tabla de variables (Figura 3). En este caso se emplearán áreas de memoria de marca.

Figura 3.

Ejemplo de declaración de variables en una tabla problema 1.1

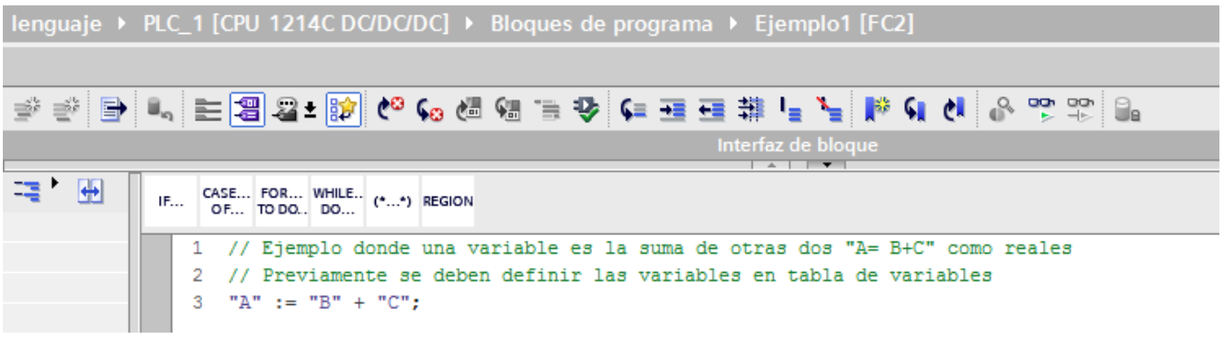
Tabla de variables_1

		Nombre	Tipo de datos	Dirección
1		A	Real	%MD20
2		B	Real	%MD24
3		C	Real	%MD28

Fuente: elaboración propia.

En la Figura 4 se muestra el algoritmo del ejemplo escrito en TIA PORTAL.

Figura 4.



Fuente: elaboración propia.

El programa correspondiente en texto es:

```
// Ejemplo donde una variable es la suma de otras dos “A= B+C” como reales
// Previamente se deben definir las variables en tabla de variables
“A” := “B” + “C”;
```

2.2 Palabras clave programación en SCL

Son palabras que están incluidas en el propio lenguaje de programación y no deben ser utilizadas para definición de variables. En SCL se tienen las siguientes palabras reservadas (Tabla 3)., definidas por Siemens, a las cuales se les agregó su uso común (Siemens, 2005);

Tabla 3.

Palabras reservadas en SCL

Palabra	Uso	Palabra	Uso
AND	Función AND	END_WHILE	Fin de la instrucción WHILE
ANY	Puntero que señala el inicio de un área de datos	EXIT	Instrucción para salir de un bucle
ARRAY	Variable tipo ARRAY	FALSE	Constante booleana predefinida: condición lógica no se cumple, valor igual a 0
BEGIN	Inicio de la sección de instrucciones de bloques lógicos o sección de inicialización de bloques de datos	FOR	Inicio de la instrucción FOR
BLOCK_DB	Parámetro para nombrar un DB	FUNCTION	Inicio de la función
BLOCK_FB	Parámetro para nombrar un FB	FUNCTION_ BLOCK	Inicio del bloque de función
BLOCK_FC	Parámetro para nombrar un FC	GOTO	Inicio de la instrucción GOTO
BLOCK_SDB	Parámetro para nombrar un SDB	IF	Inicio de la instrucción IF

Palabra	Uso	Palabra	Uso
BLOCK_SFB	Parámetro para nombrar un SFB	INT	Tipo de datos Entero
BLOCK_SFC	Parámetro para nombrar un SFC	LABEL	Etiqueta de salto
BOOL	Tipo de dato Booleano	MOD	Operador módulo
BY	Incremento del bucle FOR	NIL	Tipo de datos puntero cero
BYTE	Tipo de dato BYTE	NOT	Inversión lógica
CASE	Instrucción CASE	OF	Inicio de la especificación del tipo de datos/ de la sección de instrucciones de la instrucción CASE
CHAR	Tipo de dato simple	OK	Instrucción Comprobar validez
CONST	Inicio de la declaración de constante	OR	Función OR
CONTINUE	Instrucción para salir de un bucle	ORGANIZATION_ BLOCK	Inicio del bloque de organización
COUNTER	Tipo de parámetro para especificar un contador	POINTER	Tipo de datos puntero
DATA_BLOCK	Inicio del bloque de datos	REAL	Tipo de datos real
DATE	Tipo de datos fecha	REPEAT	Inicio de la instrucción REPEAT
DATE_AND_ TIME	Tipo de datos fecha y tiempo	RETURN	Instrucción RETURN
DINT	Tipo de datos doble entero	S5TIME	Tipo de datos S5TIME
DIV	Función dividir	STRING	Tipo de datos STRING
DO	Inicio de la sección de instrucciones de FOR y WHILE	STRUCT	Inicio de la especificación de una estructura, seguido de la lista de componentes
DT	Tipo de datos fecha y hora	THEN	Inicio de la sección de instrucciones de una instrucción IF
DWORD	Tipo de datos doble palabra	TIME	Tipo de datos simple para indicaciones horarias
ELSE	Rama alternativa en la instrucción IF y CASE	TIMER	Tipo de parámetro para especificar un temporizador
ELSIF	Condición alternativa de la instrucción IF	TIME_OF_DAY	Tipo de datos
EN	Operando de sistema del mecanismo EN/ENO	TO	Definición del valor final de una instrucción FOR
ENO	Operando de sistema del mecanismo EN/ENO	TOD	Tipo de datos fecha y hora
END_CASE	Fin de la instrucción CASE	TRUE	Constante booleana predefinida: condición lógica se cumple, valor diferente de 0
END_CONST		TYPE	Inicio del tipo de datos PLC
END_DATA_ BLOCK	Fin de la instrucción FOR	VAR	Inicio de un bloque de declaración
END_FOR	Fin del bloque de datos	VAR_TEMP	Inicio de un bloque de declaración
END_FUNC- TION	Fin de la función	UNTIL	Fin de la sección de instrucciones de una instrucción REPEAT
END_FUNC- TION_BLOCK	Fin del bloque de función	VAR_INPUT	Inicio de un bloque de declaración
END_IF	Fin de la instrucción IF	VAR_IN_OUT	Inicio de un bloque de declaración

Palabra	Uso	Palabra	Uso
END_LABEL		VAR_OUTPUT	Inicio de un bloque de declaración
END_TYPE	Fin del tipo de datos PLC	WHILE	Inicio de la instrucción WHILE
END_ORGA-NIZATION_ BLOCK	Fin del bloque de organización	WORD	Tipo de datos Word
END_REPEAT	Fin de la instrucción REPEAT	XOR	Función XOR
END_STRUCT	Fin de la especificación de una estructura	VOID	La función no tiene ningún valor de retorno
END_VAR	Fin de un bloque de declaración		

Fuente: elaboración propia a partir de Siemens, 2005.

2.3. Áreas de memoria del PLC

La CPU del PLC reconoce las siguientes áreas de memoria para su direccionamiento e indicación de tipo de dato.

2.3.1. Entradas

- I0.7 dirección de la imagen de entrada booleana del byte 0 y el bit 7.
- IB8 dirección de la imagen de entrada en forma de byte 8 (8 bits del 0 al 7).
- IW10 dirección de la imagen de entrada en forma de palabra 10 (bytes 10 y 11; 16 bits).
- ID16 dirección la imagen de entrada en forma de doble palabra 16 (bytes 16, 17,18 y 19; 32 bits).
- Para las siguientes áreas de memoria el direccionamiento es igual.

2.3.2. Salidas

- Q0.0 dirección de salida booleana.
- QB15 dirección de byte de salida 15 (8 salidas).
- QW20 dirección de palabra de salida (16 salidas).
- QD32 dirección de doble palabra de salida (32 salidas).

2.3.3. Marcas

- M10.5 dirección de una marca en forma de bit.
- MB20 dirección de marca en forma de byte.
- MW16 dirección de marca en forma de palabra.
- MD30 dirección de marca en forma de doble palabra.

2.3.4. Bloque de datos

- DB2.DBX2.4 dirección del bit 4, del byte 2, del bloque de datos 2.
- DB4.DBB8 dirección de todo el byte 8, del bloque de datos 4.
- DB1.DBW10 dirección de la palabra que empieza en el byte 10 y termina en el byte 11, del bloque de datos 1.
- DB2.DBD4 dirección de la doble palabra que empieza en el byte 4 y termina en el byte 7, del bloque de datos 2.

2.3.5. Periferia de entrada

- PIB8 entrada de periferia ubicada en el byte 8.
- PIW10 entrada de periferia como palabra.
- PID20 entrada de periferia en forma de doble palabra que ocupa los bytes 20, 21,22, y 23.

2.3.6. Periferia de salida

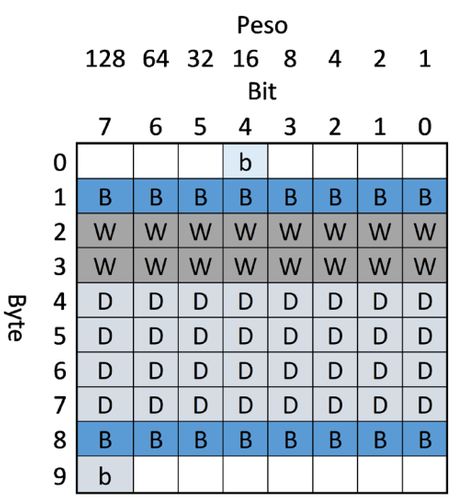
- PQB5
- PQW10
- PQD12

En la Figura 5 se muestra la configuración básica de un área de memoria, siendo b un bit, B un byte, W una palabra y D una doble palabra.

Si se toma en cuenta el peso, un bit solo puede tomar valores en decimal entre 0 y 1, un byte toma valores como entero entre 0 y 255, una palabra entre 0 y 65 535 y una doble palabra cubre un rango de 0 a 4 294 967 296.

Figura 5.

Configuración simbólica de una memoria



Fuente: elaboración propia.

2.4. Números

Pueden ser representados como enteros en formatos de INT y DINT, binario, octal, hexadecimal, como real y como cadena de caracteres.

- -1, 25, +85 son números enteros.
- 2#1010, 8#12, 16#A corresponden al número 10 en decimal representados como binario, octal y hexadecimal.
- 2.0, -4.6, +9.5, 955.0001, 3.5E11, 2.2E-5 corresponde a números reales.
- “pedro”, “calle 48”, “10-32-15-a” corresponden a cadenas de caracteres.

2.5.Tipos de datos

Pueden ser simples compuestos, definidos por el usuario o parámetros.

- Los simples son: BOOL, BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, TIME.
- DATE, TIME_OF_DAY, S5TIME .
- Entre los compuestos se tienen: DATE_AND_TIME, STRING, ARRAY y STRUCT.
- Los tipos de datos de parámetros están compuestos por: TIMER, BLOCK_FB, POINTER, ANY, COUNTER, BLOCK_FC, BLOCK_DB y BLOCK_SDB.

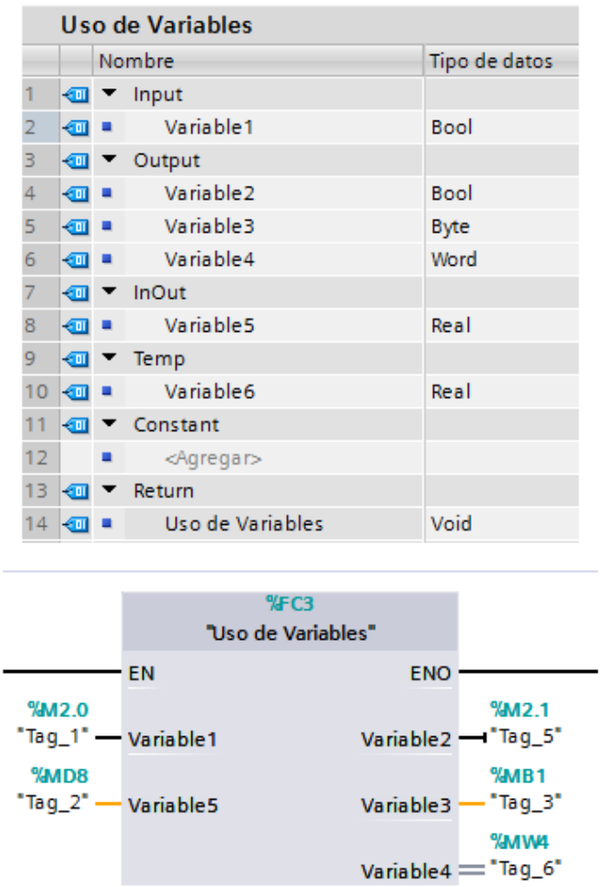
2.6.Variables

Su valor puede cambiar en el transcurso del programa y se tienen que declarar antes de utilizarlas en un bloque lógico o en un bloque de datos. Se pueden encontrar variables locales, de usuario globales y de las áreas de memoria de la CPU.

Los datos locales se declaran dentro de un bloque lógico FC, FB, y OB; solo se pueden utilizar dentro de ese bloque. Estos datos pueden ser estáticos, es decir, valores que se conservan en todos los recorridos de un bloque de función. Las variables temporales no ocupan área de memoria y solo conservan su valor en un recorrido del bloque. Los parámetros de bloque son variables de una función o un bloque de función a los cuales se les debe transferir su valor en cada llamada; por tanto, se pueden configurar como entrada, como salida o como entrada salida. En la Figura 6 se muestran variables declaradas como parámetros en un FC3; luego, por programa, se transfieren las variables desde un área de memoria de marcas a las variables locales declaradas en el FC3.

Figura 6.

Variables locales declaradas como parámetros y su transferencia con otras áreas de memoria

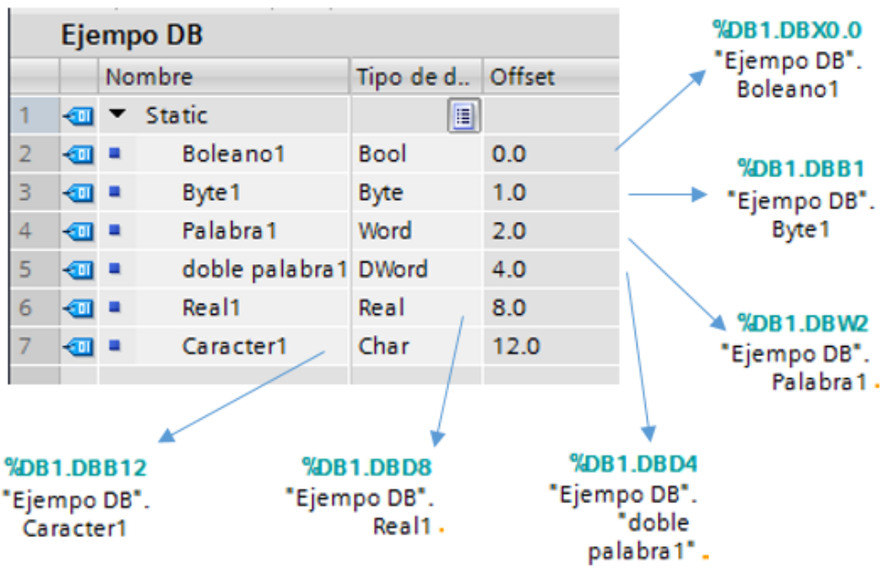


Fuente: elaboración propia.

Los datos globales o los datos de usuario pueden ser utilizados en cualquier área de programa y deben estar declaradas en un DB. En la Figura 7 se muestran variables globales declaradas en la base de datos 1 y cómo son invocadas desde diferentes partes del programa con la dirección y su representación simbólica.

Figura 7.

Variables globales declaradas en la base de datos invocadas desde otras áreas del programa



Fuente: elaboración propia.

Las áreas de memoria de la CPU se encuentran en la tabla de símbolos y pueden ser accedidas desde cualquier área de programa. En la Figura 8 se muestra una tabla de variables del PLC describiendo el nombre, el tipo de dato y la dirección.

Figura 8.

Variables del PLC

Variables PLC

	Nombre	Tabla de variables	Tipo de datos	Dirección
1	Entrada0	Tabla de variables estándar	Bool	%I0.0
2	Entrada1	Tabla de variables estándar	Bool	%I0.1
3	Entrada2	Tabla de variables estándar	Bool	%I0.2
4	Entrada3	Tabla de variables estándar	Bool	%I0.3
5	Entrada4	Tabla de variables estándar	Bool	%I0.4
6	Entrada5	Tabla de variables estándar	Bool	%I0.5
7	Salida0	Tabla de variables estándar	Bool	%Q0.0
8	Salida1	Tabla de variables estándar	Bool	%Q0.1
9	Salida2	Tabla de variables estándar	Bool	%Q0.2
10	Salida3	Tabla de variables estándar	Bool	%Q0.3
11	A	Tabla de variables_1	Real	%MD20
12	B	Tabla de variables_1	Real	%MD24
13	C	Tabla de variables_1	Real	%MD28

Fuente: elaboración propia.

03. INSTRUCIONES GENERALES Y DE CONTROL

3. Instrucciones generales y de control

3.1. Instrucciones de asignación

Sirven para llevar a una variable el valor de una constante o de otra variable; también se le puede asignar el valor que resulta de la evaluación de una expresión (Peciña Belmonte, 2018).

Ejemplos:

- Velocidad: =b*h/2;
- Acumulado: =3*a+2*b+c;

Se cuenta con una definición de variables en la tabla de símbolos y en una base de datos creada a propósito para ilustrar el ejemplo, véanse las Figuras 9 y 10. En la Figura 11 se muestran ejemplos de asignación de variables.

Figura 9.

Variables definidas en tabla de símbolos para el ejemplo de asignación de variables

Tabla de variables estándar

	Nombre	Tipo de datos	Dirección
1	Entrada1	Bool	%I0.0
2	Alarma	Bool	%M2.0
3	Dato1	Byte	%MB3
4	Dato2	Word	%MW4
5	Dato3	DWord	%MD6
6	Dato4	Real	%MD10

Fuente: elaboración propia.

Figura 10.

Variables definidas en base de datos para el ejemplo de asignación de variables

Mis_Datos1

	Nombre	Tipo de datos	Offset	Comentario
1	Static			
2	Resultado1	Bool	0.0	%DB1.DBX0.0
3	Resultado2	Bool	0.1	%DB1.DBX0.1
4	Resultado3	Byte	1.0	%DB1.DBB1
5	Resultado4	Word	2.0	%DB1.DBW2
6	Resultado5	DWord	4.0	%DB1.DBD4
7	Resultado6	Real	8.0	%DB1.DBD8

Fuente: elaboración propia.

Figura 11.

Ejemplos de asignación en lenguaje SCL en programa de TIA PORTAL

```
1  // ejemplos de asignación en SCL
2  //las asignaciones se harán en variables
3  //declaradas en una base de datos
4  //otras variables se declararon en
5  //Variables globales
6  "Mis_Datos1".Resultado1 := 1;
7  // Asigna al DB1.DBX0.0 el valor de 1
8  "Mis_Datos1".Resultado1 := FALSE;
9  // Asigna al DB1.DBX0.0 el valor de 0
10 %DB1.DBX0.1 := "Entrada1"; //Asigna al
11 //DB1.DBX0.1( el valor que tenga
12 //la entrada I0.0 (Entrada1)
13 %DB1.DBX0.1 := "Entrada1" AND "Alarma";
14 //Asigna al DB1.DBX0.1 el valor que resulta de
15 //la AND entre I0.0 (Entrada1) y M2.0 (Alarma)
16 "Mis_Datos1".Resultado3 := "Dato1";// asigna
17 // al DB1.DBB1 el valor de Dato1 (MB3)que
18 // es un byte
19 %DB1.DBW2 := "Dato2"; // Asigna a resultado4
20 // el valor de la palabra Dato2 (MW4)
21 "Mis_Datos1".Resultado5 := "Dato3";
22 // Asigna al DB1.DBD4 el valor de
23 // la doble palabra MD6 (Dato3)
24 "Mis_Datos1".Resultado6 := "Dato4";
25 // Asigna al DB1.DBD8 el valor de
26 // la variable real MD10 (Dato4)
```

Fuente: elaboración propia.

Se resaltan algunos aspectos de este código en SCL:

- Se debe recordar que el símbolo "//" se emplea para la introducción de comentarios útiles para documentar el programa y no se ejecuta como instrucción.
- La asignación se realiza por medio de los símbolos ":=" (dos puntos e igual).
- Se debe recordar que cada línea de instrucción termina con ";" (punto y coma).

En la línea 6 se hace una asignación booleana constante; obsérvese que se puede emplear un 0, un 1, la expresión False o la expresión True.

Obsérvese que, intencionalmente, algunas variables están entre comillas, indicando que son operandos simbólicos, es decir, que la variable recibe un nombre; otras están precedidas del símbolo "%", indicando que es un operando absoluto; la variable se deposita en un área de memoria y la tendencia es emplear notación simbólica. Las variables "Mis_Datos1".Resultado1 y "Entrada1" son símbolos, mientras que %DB1.DBX0.0, I0.0 son direcciones que están haciendo alusión a las mismas áreas de memoria.

En la línea 13 la evaluación de la función AND entre I0.0 (Entrada1) y la alarma (M2.0) son depositadas en la base de datos 1, variable Resultado2; DB1.DBX0.1.

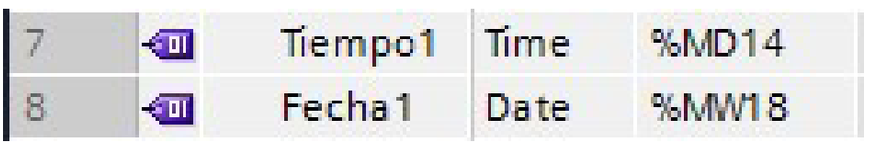
En la línea 16 se hace una asignación en forma de byte; en la línea 19 se hace asignación en forma de palabra, mientras que en las líneas 21 y 24 se hace la asignación en forma de doble palabra y real.

3.2. Asignaciones con tiempo y fecha

También se pueden realizar operaciones de asignación con variables en formato de tiempo y fecha. En la Figura 12 se declaran las variables que se van a utilizar; se distingue que la variable Tiempo1, en formato Time, se deposita como doble palabra en MD14, mientras que la variable Fecha1, en formato Date, se almacena en MW18 como palabra.

Figura 12.

Variables creadas para ejemplo de asignación de variables en formato de tiempo y hora



7	Tiempo1	Time	%MD14
8	Fecha1	Date	%MW18

Fuente: elaboración propia.

En la línea 31 se asigna en la variable Tiempo1 un valor constante de 1 día, 2 horas, 15 minutos, 20 segundos y 8 ms. En la línea 34 se hace la asignación de una fecha como constante (Figura 13).

Figura 13.

Programa de ejemplo asignación de variables en formato de tiempo y hora

```
29  //ejemplo de asignación de variables en
30  //formato de tiempo y hora
31  "Tiempo1" := Time#1D_2H_15M_20S_8MS; // se asigna en Tiempo1
32  // un valor en formato tiempo de 1 día, 2 horas, 15 minutos
33  // 20segundos y 8 ms
34  "Fecha1" := Date#2019-08-06; // Se asigna en Fecha1 una
35  //la fecha de agosto 6 de 2019
```

Fuente: elaboración propia.

3.3. Instrucciones de control

No todos los algoritmos de lógica permiten una solución puramente secuencial en un solo camino. A medida que los programas se vuelven más exigentes se debe

recurrir a instrucciones que permitan la bifurcación de acuerdo con una condición dada o instrucciones que permitan la repetición de una actividad un número de veces requerido. En SCL se tienen las instrucciones de control, de selección, de repetición y de salto. No obstante, en este tipo de control se debe tener cuidado con la terminación del programa o la subrutina, ya que estas suelen empezar y terminar con una palabra clave, especialmente al terminar las instrucciones, cuando se suele olvidar la última palabra clave, dando lugar a errores; esto se dificulta especialmente cuando se usan ciclos anidados (Goderie, 2019).

3.3.1. Instrucción IF

Es una instrucción de selección que permite tomar un camino entre dos alternativas presentadas, de acuerdo con la evaluación de condición resultante que puede ser Falso o Verdadero. Si se cumple una condición, entonces se hace algo; de lo contrario, se hace otra cosa. Suele ir acompañada de las instrucciones THEN, ELSIF, ELSE y END_IF.

3.3.2. Instrucción THEN

Permite que se ejecuten unas instrucciones si la condición evaluada es verdadera. La instrucción combinada IF… THEN permite ejecutar la o las instrucciones si se cumple la condición del IF; de lo contrario, se sigue con la ejecución de instrucciones que están luego de IF (Peciña, 2018).

3.3.3. Instrucción ELSIF

Se ejecuta si la anterior condición no se cumple, evaluando una nueva condición, por lo tanto, permite evaluar varias condiciones. Si la condición 1 no se cumple, entonces evalúe la condición 2 y si esta no se cumple, siga con la condición 3.

3.3.4. Instrucción ELSE

Se ejecuta cuando las condiciones anteriores no se cumplen. Se puede omitir. Entonces, si ninguna de las sentencias anteriores se cumplió, siga con las instrucciones que hay luego de ELSE. La combinación de la instrucción IF… THEN… ELSE evalúa la condición; si se cumple se ejecuta lo que abarca THEN; de lo contrario, se ejecutan las instrucciones luego de ELSE.

3.3.5. Instrucción END_IF

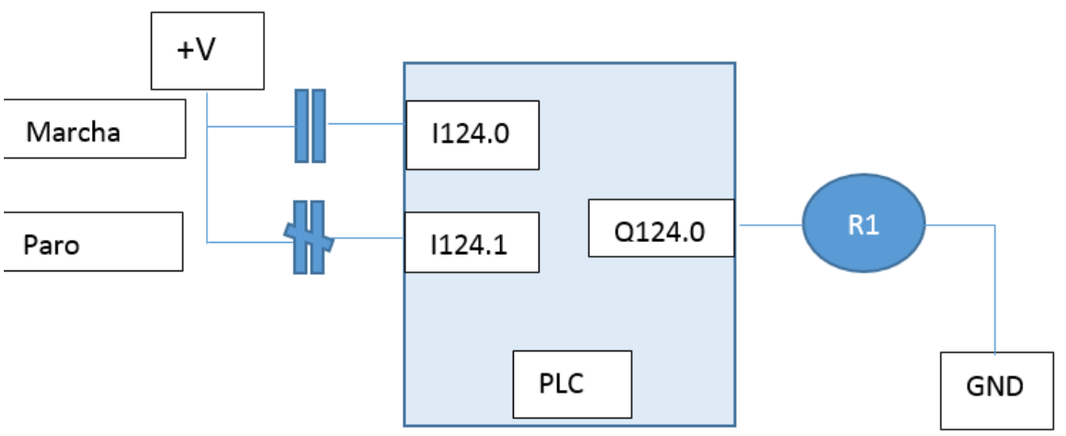
Se requiere para cerrar la instrucción IF, permite la continuación del programa y debe terminar con “;”.

Problema 3.1. Se requiere llevar a cabo la puesta en marcha con enclavamiento de un motor. El pulsador de marcha es normalmente abierto (NA) y el pulsador

de paro es normalmente cerrado (NC). En la Figura 14 se muestra una posible conexión del PLC para esta aplicación. R1 será el encargado de activar el motor en forma indirecta. La señal +V, por lo general, corresponde a un voltaje de 24 V, lo mismo que la salida Q124.0 conectada a R1.

Figura 14.

Esquema de conexiones en el PLC para un arranque de motor con enclavamiento definido en el problema 3.1



Fuente: elaboración propia.

En la Tabla 4 se muestran los símbolos definidos en el PLC para las variables involucradas en un PLC S7-300.

Tabla 4.

Símbolos utilizados en el problema 3.1

Marcha1	%I124.0
Paro1	%I124.1
Motor1	%Q124.0

Fuente: elaboración propia.

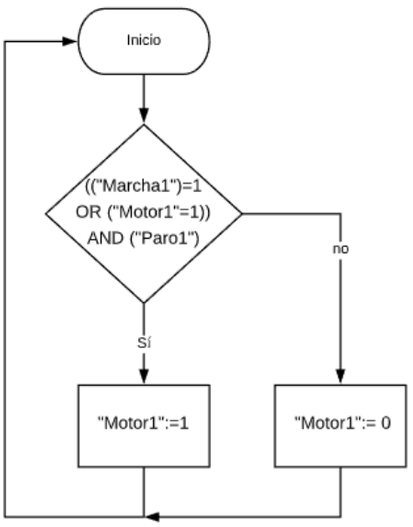
El algoritmo de control en SCL se muestra a continuación. Sabiendo que "Paro1" normalmente es verdadero por ser un contacto NC, entonces, la variable "Motor1" se pone en verdadero (1) al pulsar "Marcha1", quedando un estado de memoria enclavado debido a la realimentación que se hace con la función OR de "Motor1". Para detener el funcionamiento se presiona "Paro1", con lo que "Paro1" va a falso y la evaluación de la función AND resulta en falso, desactivando "Motor1"; se pierde el estado de memoria. Primero se realiza una función OR entre "Marcha1"

y “Motor1”, y al resultado se le hace una función AND con “Paro1”. En caso de que la evaluación anterior sea verdadera (1), entonces “Motor1” se activa; en caso contrario, “Motor1” va a Falso (0). La salida del motor, que en este caso es Q124.0, se activa en caso de que “Motor1” sea verdadero (1 lógico). La solución de este problema en diagrama de flujo se muestra en la Figura 15. El algoritmo en SCL se muestra a continuación y se evidencia la sencillez del programa. Hay que tener en cuenta que el efecto de la instrucción “Motor1”:=1 es la función SET.

```
IF ((“Marcha1”)=1 OR (“Motor1”=1)) AND (“Paro1”)=1 THEN
    “Motor1”:=1; //Se activa si se cumple la condición anterior
ELSE              // En caso contrario se desactiva la salida
    “Motor1”:= 0;
END_IF;
```

Figura 15.

Diagrama de flujo para solucionar problema 3.1



Fuente: elaboración propia.

Problema 3.2. Solucione el problema anterior usando funciones de SET y RESET. Una solución puede ser:

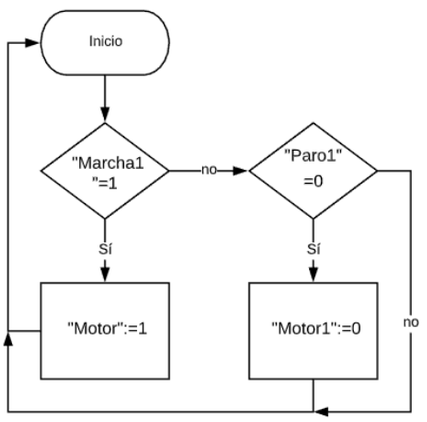
```
   IF “Marcha1” THEN
    “Motor”:=1;     //Función SET
END_IF;

IF “Paro1” =0 THEN // Tener en cuenta que Paro es NC, cuando se pulsa este botón
// la variable Paro se pone en falso
    “Motor1”:=0;    //Función RESET
END_IF;
```

Las instrucciones SET o RESET se logran luego de evaluar una condición que, de ser verdadera, permite la activación de cada una de ellas, según sea el caso. La función SET hace que a la variable manipulada se le asigne un 1 lógico, mientras que la función RESET hace que a esta misma variable se le asigne un 0 lógico.

Como se puede observar, se han mostrado dos formas de consultar el estado de una variable. Por ejemplo, la instrucción IF “Marcha1”=1 pregunta si se está dando esta condición; con la instrucción IF “Marcha1” se logra el mismo resultado. En muchos problemas es recomendable realizar un diagrama de flujo antes de proceder a realizar el código en SCL, tal como se muestra en la Figura 16.

Figura 16.

Diagrama de flujo para solucionar problema 3.2

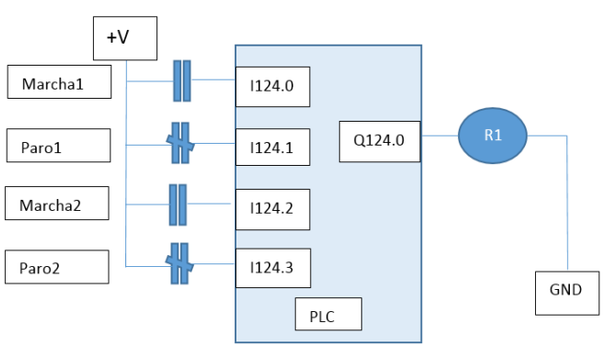


Fuente: elaboración propia.

Problema 3.3. Active o desactive un motor desde dos lugares diferentes. En la Figura 17 se muestra el plano de conexiones del PLC y se agregan dos entradas. En la Tabla 5 aparecen dos nuevas variables declaradas en la tabla de símbolos, que corresponden a otro pulsador de marcha, normalmente abierto (NA) y otro de paro, normalmente cerrado (NC).

Figura 17.

Conexiones del PLC para el problema 3.3



Fuente: elaboración propia.

Tabla 5.

Nuevas variables para definir en la tabla de símbolos en problema 3.3

Marcha2	%Q124.2
Paro2	%I124.3

Fuente: elaboración propia.

Se supone que los pulsadores de “Marcha1” y “Paro1” se encuentran en un lugar, mientras que “Marcha2” y “Paro2” se encuentran en uno diferente. Se resalta la forma en la que se puede realizar anidamiento de instrucciones utilizando paréntesis. Simplemente, se debe realizar una evaluación OR entre las entradas “Marcha1”, “Marcha2” y la salida “Motor1”; el resultado se evalúa con una AND que involucra a “Paro1” y “Paro2”.

El siguiente es el algoritmo de control en SCL:

```
IF ((“Marcha1”) = 1 OR (“Marcha2”) = 1 OR (“Motor1” = 1)) AND ((“Paro1”)
= 1 AND (“Paro2”=1)) THEN

    “Motor1” := 1;

ELSE

    “Motor1” := 0;

END_IF;
```

Problema 3.4. Puesta en marcha, con enclavamiento, de un motor con marcha NA y paro NC; agregue una alarma intermitente de 1 Hz, mientras el motor está en funcionamiento.

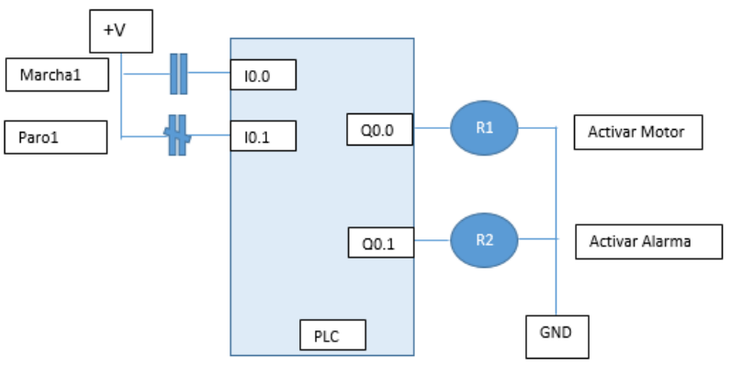
Si se trabaja con un PLC S7300, S71200 o S71500, previamente se debe conFigurar la marca M0.5 como oscilador de 1 Hz, desde la configuración de hardware del PLC. Otra alternativa es simularlo por software. En la Figura 18 se detallan las conexiones del PLC, se agrega una salida para la activación de la alarma por medio de R2; con R1 se supone que se activa el motor. El algoritmo de control en SCL es el siguiente:

```
IF ((“Marcha1” = 1) OR (“Motor1” = 1)) AND (“Paro1” = 1)  THEN
    “Motor1” := 1;
    “Alarma1” := (“Oscilador_1Hz” AND “Motor1”);
ELSE
    “Motor1” := 0;
END_IF;
```

Si “Motor1” deja de funcionar también se impedirá la activación de “Alarma1”.

Figura 18.

Plano de conexiones del PLC problema 3.4



Fuente: elaboración propia.

En la Tabla 6, de símbolos, se debieron adicionar las siguientes variables.

Tabla 6.

Nuevas variables para definir en la tabla de símbolos del problema 3.4

Alarma1	%Q124.0
Oscilador_1Hz	%M0.5

Fuente: elaboración propia.

Problema 3.5. Cree un programa que permita la detección de flancos de una variable de tipo booleano y registre el último estado obtenido.

En este caso la información se guardará en una base de datos llamada “Detector” (Base de datos1) que contiene las variables requeridas. Cuando en el programa aparece la Variable “Detector”.Señal significa que a la base de datos se le dio el nombre “Detector” y la variable es “Señal” (Tabla 7).

Tabla 7.

Variables guardadas en la base de datos “Detector” del problema 3.5

Nombre	Dirección	Tipo	Descripción
“Detector”.Señal	%DB1.DBX0.0	BOOL	Es la señal a la que se le quieren detectar los flancos
“Detector”.Flanco_Subida	%DB1.DBX0.1	BOOL	Es el testigo de que se dio un flanco de subida en el ciclo actual
“Detector”.Auxiliar_Subida	%DB1.DBX0.2	BOOL	Variable auxiliar de flanco de subida
“Detector”.Flanco_Bajada	%DB1.DBX0.3	BOOL	Es el testigo de que se dio un flanco de bajada en el ciclo actual
“Detector”.“Auxiliar_ Bajada”	%DB1.DBX0.4	BOOL	Variable auxiliar de flanco de bajada

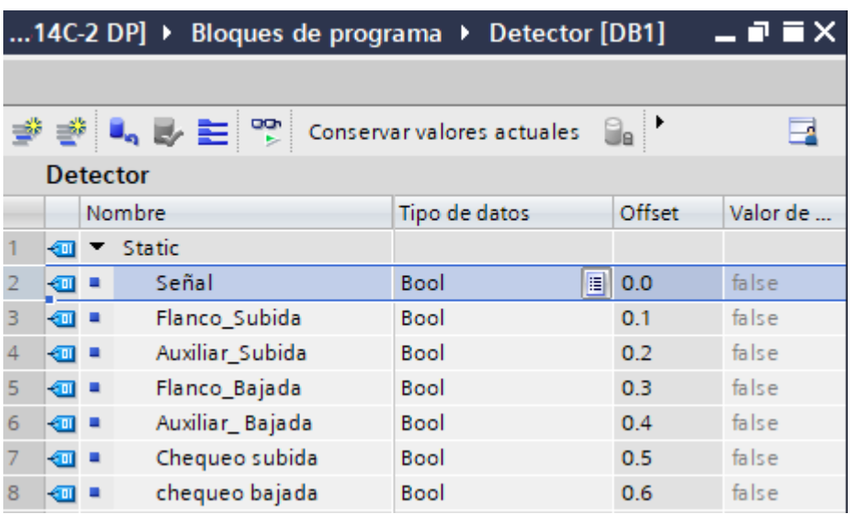
"Detector""Chequeo subida"	%DB1.DBX0.5	BOOL	Detectará si el último flanco de la señal fue de subida, sin importar el ciclo
"Detector""chequeo bajada"	%DB1.DBX0.6	BOOL	Detectará si el último flanco de la señal fue de bajada, sin importar el ciclo

Fuente: elaboración propia.

En la Figura 19 se muestra la apariencia de la base de datos en TIA PORTAL; se debe considerar que el efecto de detectar un flanco de subida o de bajada solo es efectivo en un ciclo, por lo tanto, la acción de control que se quiere realizar con estos flancos se debe efectuar en el mismo ciclo.

Figura 19.

Apariencia de la base de datos "Detector" del problema 3.5



Fuente: elaboración propia.

El programa sugerido para solución del problema 3.5 escrito en SCL es el siguiente. Se adicionan comandos de SET y RESET con el objetivo de memorizar el último flanco detectado.

```
//Detector flancos de subida.

// Este detector actúa cuando la señal está en 1 y el auxiliar de subida
estaba en cero

// luego el auxiliar va a 1 y para el próximo ciclo no se reconocerá flan-
co de subida.

// Quiere decir que este detector solo tiene efecto en un ciclo, para los
siguientes

//Detector de flanco de subida permanecerá en cero hasta que efectivamente
se dé un nuevo flanco de subida

"Detector".Flanco_Subida := "Detector".Señal AND NOT "Detector".Auxiliar_
Subida;
```

```
"Detector".Auxiliar_Subida := "Detector".Señal;

"Detector".Flanco_Bajada := NOT "Detector".Señal AND NOT "Detector"."Au-
xiliar_ Bajada";

"Detector"."Auxiliar_ Bajada" := NOT "Detector".Señal;



// La siguiente etapa sirve para verificar la detección de flancos

// No es requerido en el programa

// Como detector de flanco solo tiene efecto en un ciclo, entonces se uti-
lizaron funciones

// de SET y RESET para estar chequeando el último flanco detectado



IF "Detector".Flanco_Subida THEN

    "Detector"."Chequeo subida":= 1;

    "Detector"."chequeo bajada" := 0; //Para resetear el estado anterior

END_IF;

IF "Detector".Flanco_Bajada THEN

    "Detector"."Chequeo subida" := 0;

    "Detector"."chequeo bajada" := 1;

END_IF;
```

Problema 3.6. Se tienen nueve entradas digitales encargadas de manipular ocho salidas digitales. Cuando la entrada 9 está en cero cada entrada de la 1 a la 8 se encargará de energizar su propia salida. En caso de que la entrada 9 esté en uno, al activar la entrada 1 se deberán activar las entradas 1 a 4 y al activar las entradas 1 y 2 se energizarán, además, las salidas de la 5 a la 8.

En la Figura 20 se muestran las direcciones y símbolos para utilizar en la tabla de variables; se resalta que, por conveniencia, se llamó a las entradas digitales de la I124.0 a la I124.7; "Entrada_Grupo1" que corresponde al byte IB124, lo mismo ocurre con las salidas agrupadas en el byte de salida QB124; "Salida_Grupo1".

Figura 20.

Símbolos para utilizar en el problema 3.6

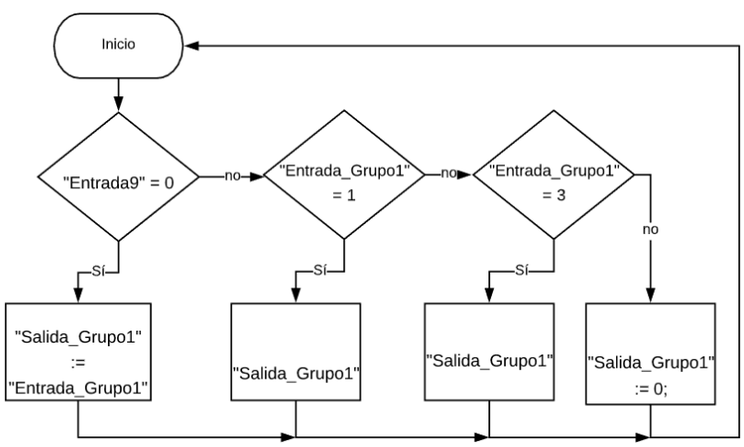
Entrada1	%I124.0	Salida1	%Q124.0
Entrada2	%I124.1	Salida2	%Q124.1
Entrada3	%I124.2	Salida3	%Q124.2
Entrada4	%I124.3	Salida4	%Q124.3
Entrada5	%I124.4	Salida5	%Q124.4
Entrada6	%I124.5	Salida6	%Q124.5
Entrada7	%I124.6	Salida7	%Q124.6
Entrada8	%I124.7	Salida8	%Q124.7
Entrada9	%I125.0		
Entrada_Grupo1	%IB124	Salida_Grupo1	%QB124

Fuente: elaboración propia.

En la Figura 21 se muestra el diagrama de flujo correspondiente al problema 3.6.

Figura 21.

Diagrama de flujo correspondiente al problema 3.6



Fuente: elaboración propia.

El algoritmo en SCL está definido en la siguiente forma:

```
IF “Entrada9” = 0 THEN
    “Salida_Grupo1” := “Entrada_Grupo1”;  // El byte de salida QB124 es
// el espejo del byte de entrada IB124 (8 bits)
ELSE   // Si entrada9 es diferente de 0
    IF “Entrada_Grupo1” = 1 THEN // y el bit 0 de “Entrada_Grupo1” es 1
salida  “Salida_Grupo1” := 15; // se activan los primeros 4 bits de la
    ELSE
trada    IF “Entrada_Grupo1” = 3 THEN //Si los dos primeros bits de la en-
// están activos, entonces todos los bits de salida se activarán
            “Salida_Grupo1” := 255;
        ELSE
```

```
        “Salida_Grupo1” := 0; // de lo contrario los 8 bits de salida es-
tán apagados
        END_IF; // hay 3 condicionales anidados que se deben terminar.

    END_IF;
END_IF;
```

Problema 3.7. Se tiene la posibilidad de hallar el área de figuras geométricas básicas: círculo, rectángulo y triángulo, según la posición de dos entradas digitales. Los datos para hallar el área se encuentran en posiciones de memoria diferentes de acuerdo con la figura geométrica. En la Figura 22 se muestran las variables, direcciones y tipo de variable para incluir en la tabla de símbolos. El diagrama de flujo correspondiente se muestra en la Figura 23.

Figura 22.

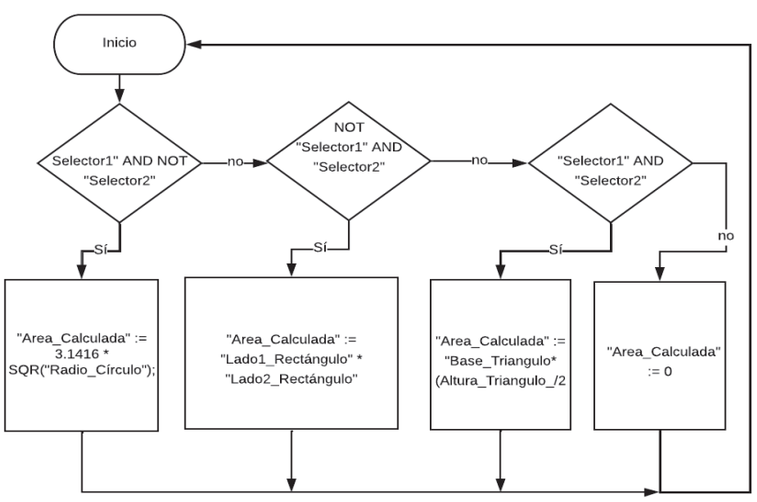
Símbolos para emplear en el problema 3.7

Variable	Dirección	Tipo
Selector1	%I124.0	Bool
Selector2	%I124.1	Bool
Radio_Círculo	%MD10	Real
Lado1_Rectángulo	%MD14	Real
Lado2_Rectángulo	%MD18	Real
Base Triangulo	%MD22	Real
Altura_Triangulo	%MD26	Real
Area_Calculada	%MD28	Real

Fuente: elaboración propia.

Figura 23.

Diagrama de flujo propuesto para solución del problema 3.7



Fuente:elaboración propia.

El siguiente es el código en SCL propuesto para dar solución al problema 3.7. Se resalta que se está consultando una variable negada NOT “Selector2”; las asignaciones son el resultado de la evaluación de funciones matemáticas simples.

```
IF “Selector1” AND NOT “Selector2” THEN //Hallar área de círculo
    “Area_Calculada” := 3.1416 * SQR(“Radio_Círculo”);
ELSE
    IF NOT “Selector1” AND “Selector2” THEN //Hallar área de rectángulo
        “Area_Calculada” := “Lado1_Rectángulo”*”Lado2_Rectángulo”;
    ELSE
        IF “Selector1” AND “Selector2” THEN //Hallar área de triángulo
            “Area_Calculada” := “Base_Triángulo” * “Altura Triángulo” / 2;
        ELSE
            “Area_Calculada” := 0; // en caso de que los selectores sean 0
        END_IF;
    END_IF;
END_IF;
```

Problema 3.8. Se está midiendo caudal por medio de un sistema de turbina que da una señal de frecuencia (0 a 225 Hz) para una entrada de 0 a 30 l/min, proporcionalmente, para muestras de caudal realizadas cada 2 segundos, supuestamente para las mismas condiciones; se han obtenido valores de caudal algo dispersos, oscilando entre picos máximos y mínimos más o menos constantes. Al llevar esta medición a un sistema de control se observa una respuesta muy oscilatoria. Para tratar de reducir el impacto de estos picos de medición en la respuesta de control se propone realizar un filtro a la señal de entrada, de tal forma que se obtengan valores más constantes para las mismas condiciones de flujo.

Para la solución del problema se propone estar encontrando el promedio de las últimas mediciones y tomar esta señal de medida del controlador (se puede decir que es un promedio móvil simple) donde se tienen en cuenta los datos más nuevos y lo viejos se van desechando; esto ayuda a disminuir el ruido de las mediciones. El número de datos para tomar depende de qué grado de sensibilidad se le quiera dar a la medición; entre más datos se tomen menos cambios tendrá la medida. No se debe abusar de este número, es decir, tomarlo muy alto porque esto podría significar pérdida de sensibilidad ante cambios reales de la señal medida o tendencias de esta (Salazar López, 2019). Para este programa también se considerará en la fórmula el valor anterior del promedio de caudal calculado previamente.

La fórmula que se propone es:

Ecuación 1.

Promedio móvil simple

Donde:

- PM: promedio móvil
- X(t): valor de la medida en un tiempo t
- Nn: número de elementos del numerador

Para el programa se tendrán en cuenta las variables que se muestran en la Tabla 8.

Tabla 8.

Variables para emplear en el problema 3.8

Nombre	Tipo	Dirección	Comentario
Carga de datos	Bool	%I124.0	Oscilador a 1 Hz
Detector de flanco	Bool	%M20.1	Para detectar flanco de subida
Auxiliar	Bool	%M20.0	Auxiliar para detectar flancos
Caudal	Real	%MD104	Señal actual del sensor de caudal
Caudal_prom	Real	%MD108	Caudal promedio calculado
Caudal-1	Real	%MD112	Caudal anterior
Caudal-2	Real	%MD116	
Caudal-3	Real	%MD120	
Caudal-4	Real	%MD124	
Caudal-5	Real	%MD128	Caudal más antiguo

Fuente: elaboración propia.

El programa propuesto en SCL para el problema 3.8 es el siguiente:

```
“Detector de flanco” := “Carga de datos” AND NOT “Auxiliar”;
“Auxiliar” := “Carga de datos”;
IF “Detector de flanco” THEN

    “Caudal-5” := “Caudal-4”;
    “Caudal-4”:= “Caudal-3”;
    “Caudal-3” := “Caudal-2”;
    “Caudal-2” := “Caudal-1”;
    “Caudal-1” := “Caudal”;

    “Caudal_prom”:=(“Caudal_prom”+”Caudal”+”Caudal-1”+”Caudal-2”+”Cau-
dal-3”+”Caudal-4”+”Caudal-5”)/7;


END_IF;
```

De esta forma, siempre que se detecta un pulso positivo de “Carga de datos” (M0.1. oscilador de 1Hz), un nuevo promedio será calculado. Los elementos para considerar son el promedio de caudal anterior, el valor de caudal del instante y el valor de caudal de las últimas cinco lecturas; entonces, en el denominador debe ir el número 7. El usuario tiene la posibilidad de quitar términos del numerados según su conveniencia en la sensibilidad de la medición de caudal. Nótese cómo en cada ejecución de la rutina, los datos más viejos van quedando en la cola y el que venía de último desaparece. En la Figura 24 está la apariencia del programa SCL en el software TIA PORTAL.

Figura 24.

Programa SCL en el software TIA PORTAL del problema 3.8

```
"Detector de flanco" := "Carga de datos" AND NOT "Auxiliar";
"Auxiliar" := "Carga de datos";
IF "Detector de flanco" THEN

    "Caudal-5" := "Caudal-4";
    "Caudal-4":= "Caudal-3";
    "Caudal-3" := "Caudal-2";
    "Caudal-2" := "Caudal-1";
    "Caudal-1" := "Caudal";
    "Caudal_prom":=("Caudal_prom"+"Caudal"+"Caudal-1"
    +"Caudal-2"+"Caudal-3"+"Caudal-4"+"Caudal-5")/7;

END_IF;
```

Fuente: elaboración propia.

3.3.6. Instrucción CASE

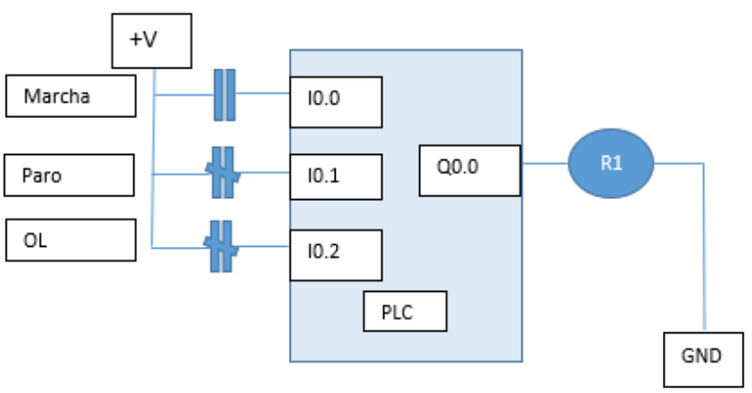
Da la posibilidad de escoger entre varias opciones; es una instrucción de selección que permite la ramificación del programa en n posibilidades, siendo n una variable entera (Martínez León, 2017). En el siguiente ejemplo se ejecutará una sentencia diferente dependiendo de si el contenido de “Nombre de Variable” (un entero) es 1, 2, 3, 4 u otro valor. La estructura general de la instrucción CASE en SCL es:

```
CASE _Nombre de Variable_ OF  // la variable es un entero
    1:  // Se pone la sentencia en caso de que la variable sea 1
        “Variable 1”:=1 ;
    2..4:  // Se pone sentencia en caso de que la variable sea 2, 3, o 4
       “Variable 1”:=4;
7,9,11:  // Se pone sentencia en caso de que la variable sea 7,9,11
      “Variable 2 “:=10  ;

    ELSE  // Se pone la sentencia en caso de que la variable tenga un valor diferente
     “Variable 3 “:=8    ;
END_CASE;
```

Problema 3.9. Considérese la puesta en marcha de un motor; se dispone como entradas los botones de marcha (NA), de paro (NC), y un térmico (NC). Como salida se tiene la activación de un relé para poner en funcionamiento el motor. En la Figura 25 se muestra el esquema de conexiones del PLC (S7-1200). Como estrategia se diseñará una matriz en la que se muestran las combinaciones de entradas y salidas que harán que la salida esté activa, esta matriz puede ser llamada tabla de verdad (Gil Sánchez et al., 2019). En la Tabla 9 se pueden apreciar las posibles combinaciones aplicables al problema 3.9.

Figura 25.

Diagrama de conexiones para el problema 3.9



Fuente: elaboración propia.

Con cada combinación de las variables se puede conformar un número (conversión de binario a decimal); según la tabla, solo tres números hacen que el motor permanezca en marcha: 7, 14 y 15. En la columna de "Número" se pone un número que resulta de la conversión de binario a decimal de la combinación de las variables "Marcha", "Paro", "OL" y "R_Motor". Este número decimal será el que se utilice en el programa SCL para la distribución realizada por la instrucción CASE. En la columna "Activar motor (+1)" se pone un "Sí" en las combinaciones o números en que se debe activar el motor.

Tabla 9.

Combinaciones posibles para que la salida del motor esté activa, problema 3.9

Número	Variable				Activar motor (+1)
	Marcha	Paro	OL	R_Motor	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	

6	0	1	1	0	
7	0	1	1	1	Sí
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	Sí
15	1	1	1	1	Sí

Fuente: elaboración propia.

La tabla de símbolos propuesta se muestra en la Tabla 10.

Tabla 10.

Variables para utilizar para el problema 3.9

Nombre	Dirección	Tipo
Esp_Marcha	%M3.3	Bool
Esp_OL	%M3.1	Bool
Esp_Paro	%M3.2	Bool
Esp_R_Motor	%M3.0	Bool
Estados	%MW2	INT
Marcha	%I0.0	Bool
OL	%I0.2	Bool
Paro	%I0.1	Bool
R_Motor	%Q0.0	Bool

Fuente: elaboración propia.

Por conveniencia del manejo de la tabla, la imagen de los estados de las entradas y la salida se lleva a la memoria de marcas. En MW2 se almacena la variable que contiene el número entero que se va a consultar, donde el byte 3 es el menos significativo de MW2; recuerde que MW2 está conformado por el byte 2 y el byte 3 (Siemens, 2005). El programa en SCL propuesto es el siguiente; las primeras cuatro instrucciones llevan las entradas y salidas a la imagen que se propone.

```
“Esp_Marcha”:=”Marcha”;
“Esp_Paro” := “Paro”;
“Esp_OL” := “OL”;
“Esp_R_Motor” := “R_Motor”;
CASE “Estados” OF
    7:        “R_Motor” := 1;
    14..15:        “R_Motor” := 1;
    ELSE
        “R_Motor” := 0;
END_CASE;
```

Problema 3.10. Se dispone de un botón de marcha (NA), un botón de paro (NC) y tres motores. Diseñe el arranque en secuencia de los tres motores, uno por uno, cada vez que se presiona "Marcha". Considere que para dar una nueva marcha primero se tiene que soltar el pulsador; la primera vez que se suelte "Marcha" se registrará el evento en una marca (M1), la segunda vez se registrará en M2. La matriz de entradas y salidas, teniendo en cuenta estos dos eventos, se muestra en la Tabla 11; para resumir, solo se mostrarán las combinaciones que mantendrán activa alguna salida, M1 y M2. También, con cada combinación de variables se puede conformar un número decimal. La columna "#" contiene los números que corresponden a la conversión a decimal del número binario formado por la combinación de las columnas que cubren el campo "Variable"; observe que este campo está compuesto por entradas, salidas y marcas. Este número decimal será el que se utilice en el programa SCL para la distribución realizada por la instrucción CASE. En las columnas del campo "Activar (+1)" se especifican las salidas y marcas que se deben activar con cada combinación.

Tabla 11.

Matriz de combinaciones activación del motor en problema 3.10

#	Variable							Activar (+1)				
	M	**P**	**Q1**	**M1**	**Q2**	**M2**	**Q3**	**Q1**	**M1**	**Q2**	**M2**	**Q3**
96	1	1	0	0	0	0	0	Sí				
112	1	1	1	0	0	0	0	Sí				
48	0	1	1	0	0	0	0	Sí	Sí			
56	0	1	1	1	0	0	0	Sí	Sí			
120	1	1	1	1	0	0	0	Sí	Sí	Sí		
124	1	1	1	1	1	0	0	Sí	Sí	Sí		
60	0	1	1	1	1	0	0	Sí	Sí	Sí	Sí	
62	0	1	1	1	1	1	0	Sí	Sí	Sí	Sí	
126	1	1	1	1	1	1	0	Sí	Sí	Sí	Sí	Sí
127	1	1	1	1	1	1	1	Sí	Sí	Sí	Sí	Sí
63	0	1	1	1	1	1	1	Sí	Sí	Sí	Sí	Sí

Fuente: elaboración propia.

Los símbolos para utilizar en el problema 3.9 se muestran en la Tabla 12.

Tabla 12.

Variables para utilizar en el problema 3.10

Nombre	Dirección	Tipo
Marcha	%I0.0	Bool
Paro	%I0.1	Bool
Motor1	%Q0.0	Bool
Esp_Motor1	%M3.4	Bool
Esp_Motor3	%M3.0	Bool
Esp_Marcha	%M3.6	Bool

Esp_Paro	%M3.5	Bool
Estados	%MW2	Int
Motor2	%Q0.1	Bool
Motor3	%Q0.2	Bool
M2	%M3.1	Bool
Esp_Motor2	%M3.2	Bool
M1	%M3.3	Bool

Fuente: elaboración propia.

El programa en SCL para la solución al problema 3.10 se muestra a continuación. Por guardar un orden, los estados de CASE se ponen de menor a mayor, aunque no correspondan con el orden de secuencia de operación del motor.

```
"Esp_Marcha":="Marcha";// Se crea un espejo de Marcha, Paro, Motor1,

"Esp_Paro" := "Paro";// Motor2 y Motor3 en la memoria de marca MW2

"Esp_Motor1" := "Motor1";// MB3 es el byte menos significativo

"Esp_Motor2" := "Motor2";
"Esp_Motor3" := "Motor3";
CASE "Estados" OF
    48:// Este caso se da cuando Paro y Motor1 están activos y se
        //soltó el pulsador
        //Recuerde que Paro es NC
        "Motor1" := 1;
        "M1" := 1;// Registra que se suelta el Pulsador primera vez
    56:// Paro, Motor1 y M1 están activos
        "Motor1" := 1;
        "M1" := 1;

    60:// Paro, Motor1, M1 y Motor2 están activos
        "Motor1" := 1;
        "M1" := 1;
        "Motor2" := 1;
        "M2" := 1; // Registra que se suelta el Pulsador segunda vez
    62: //Paro, Motor1, M1, Motor2 y M2 están activos
        "Motor1" := 1;
        "M1" := 1;
        "Motor2" := 1;
        "M2" := 1;

    63: //Paro, Motor1, M1, Motor2, M2 y Motor3 están activos
        "Motor1" := 1;
        "M1" := 1;
        "Motor2" := 1;
        "M2" := 1;
        "Motor3" := 1;
```

```
    96: // Cuando se da Marcha por primera vez
        “Motor1” := 1; // Arranca el Motor1
    112: //Marcha, Paro y Motor1 están activos
        “Motor1” := 1;

    120:// Cuando se da Marcha por segunda vez
        “Motor1” := 1;
        “M1” := 1;
        “Motor2” := 1; // Arranca el Motor1
    124: //Paro, Motor1, M1 y Motor2 están activos
        “Motor1” := 1;
        “M1” := 1;
        “Motor2” := 1;

    126,127: // Se activa la Marcha por tercera vez
       “Motor1” := 1;
       “M1” := 1;
       “Motor2” := 1;
       “M2” := 1;
       “Motor3” := 1;

   ELSE // En un caso diferente se apagan las salidas, M1 y M2
        “Motor1” := 0;
        “M1” := 0;
        “Motor2” := 0;
        “M2” := 0;
        “Motor3” := 0;
END_CASE;
```

Observe que algunos casos repiten la activación de sus salidas: (96,112), (48,56), (120,124), (60,62) y (63,126,127); eso sugiere que el programa puede quedar más corto, tal como se muestra a continuación:

```
“Esp_Marcha”:=”Marcha”;// Se crea un espejo de Marcha, Paro, Motor1,
“Esp_Paro” := “Paro”;// Motor2 y Motor3 en la memoria de marca MW2
“Esp_Motor1” := “Motor1”;// MB3 es el Byte menos significativo
“Esp_Motor2” := “Motor2”;
“Esp_Motor3” := “Motor3”;

CASE “Estados” OF
    48,56:// Este caso se da cuando Paro y Motor1 están activos y se
        //soltó el pulsador
        //Recuerde que Paro es NC
        “Motor1” := 1;
        “M1” := 1;// Registra que se suelta el Pulsador primera vez

    60,62:// Paro, Motor1, M1 y Motor2 están activos
        “Motor1” := 1;
```

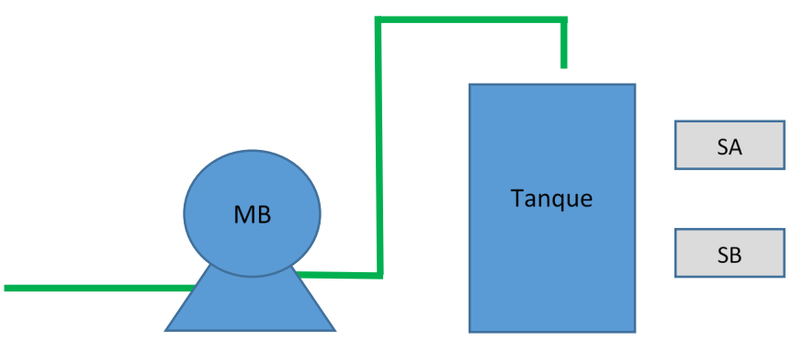
```
        "M1" := 1;
        "Motor2" := 1;
        "M2" := 1; // Registra que se suelta el Pulsador segunda vez

    63,126,127: //Paro, Motor1, M1, Motor2, M2 y Motor3 están activos
        "Motor1" := 1;
        "M1" := 1;
        "Motor2" := 1;
        "M2" := 1;
        "Motor3" := 1;
    96,112: // cuando se da Marcha por primera vez
        "Motor1" := 1; // Arranca el Motor1

    120,124:// cuando se da Marcha por segunda vez
        "Motor1" := 1;
        "M1" := 1;
        "Motor2" := 1; // Arranca el Motor1

   ELSE // En un Caso diferente se apagan las salidas, M1 y M2
        "Motor1" := 0;
        "M1" := 0;
        "Motor2" := 0;
        "M2" := 0;
        "Motor3" := 0;
END_CASE;
```

Problema 3.11. Se dispone de un tanque al cual se le controla su nivel por medio de una motobomba (MB) que impulsa líquido a la entrada de este. El nivel debe permanecer entre dos valores determinados por dos sensores, uno de baja (SB) y otro de alta (SA). El sistema dispone de pulsadores de marcha (M) y paro (P). Cuando el nivel está muy bajo, sensor SB desactivado, la motobomba se debe energizar y solo se apaga cuando el nivel llega a activar el sensor SA (nivel alto).

En la Figura 26 se muestra la disposición del tanque, la motobomba y los sensores. Las conexiones del PLC se sugieren en la Figura 27.

Figura 26.

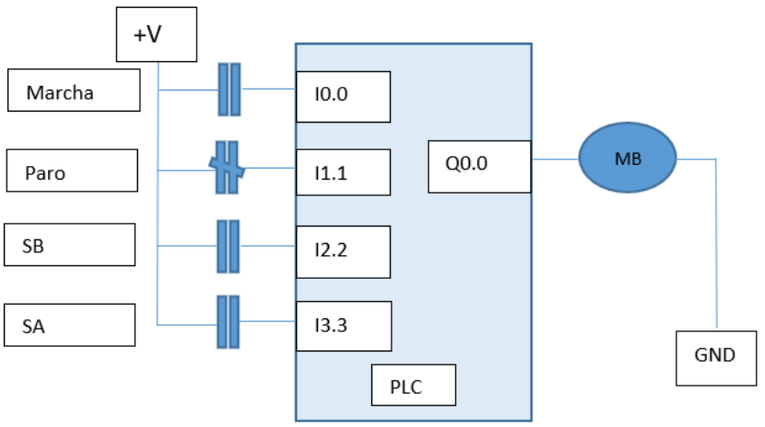
Disposición del circuito hidráulico del problema 3.11



Fuente: elaboración propia.

Figura 27.

Conexiones del PLC problema 3.11



Fuente: elaboración del propia.

En la Tabla 13 se muestran las combinaciones que hacen que la motobomba se energice. Se incluye una marca (Mar) cuya función es registrar el evento cuando Marcha fue presionado. Con cada combinación de las variables (número binario) se forma un número decimal (ver campo de "Variable") que es utilizado en la función CASE para activar "Mar" y "MB" del campo "Activar (+1)".

Tabla 13.

Matriz de combinaciones para activar la motobomba del problema 3.11

#	Variable						Activar (+1)	
	M	P	SB	SA	Mar	MB	Mar	MB
18	0	1	0	0	1	0	1	1
19	0	1	0	0	1	1	1	1
27	0	1	1	0	1	1	1	1
48	1	1	0	0	0	0	1	1
50	1	1	0	0	1	0	1	1
51	1	1	0	0	1	1	1	1
56	1	1	1	0	0	0	1	1
60	1	1	1	1	0	0	1	1
59	1	1	1	0	1	1	1	1
26	0	1	1	0	1	0	1	0
30	0	1	1	1	1	0	1	0
31	0	1	1	1	1	1	1	0
58	1	1	1	0	1	0	1	0
62	1	1	1	1	1	0	1	0
63	1	1	1	1	1	1	1	0

Fuente: elaboración propia.

En la Tabla 14 se detallan los símbolos utilizados en el programa; Esp_MB, Esp_Marcha y Esp_Paro se utilizan para formar una palabra MW2 con los estados de las entradas y salidas.

Tabla 14.

Variables para utilizar en el problema 3.11

Nombre	Tipo de dato	Dirección
Marcha	Bool	%I0.0
Paro	Bool	%I0.1
MB	Bool	%Q0.0
Esp_MB	Bool	%M3.0
Esp_Marcha	Bool	%M3.5
Esp_Paro	Bool	%M3.4
Estados	Int	%MW2
Mar	Bool	%M3.1
Esp_SA	Bool	%M3.2
Esp_SB	Bool	%M3.3
SB	Bool	%I0.2
SA	Bool	%I0.3

Fuente: elaboración propia.

El programa en SCL para la solución del problema 3.11 se muestra a continuación:

```
“Esp_Marcha”:=”Marcha”;// Se crea un espejo de Marcha, Paro, MB,
“Esp_Paro” := “Paro”;// y Marca en la memoia de marca MW2
“Esp_MB” := “MB”;// MB3 es el byte menos significativo
“Esp_SA” := “SA”;
“Esp_SB” := “SB”;

CASE “Estados” OF
    18,19,27,48,50,51,59,56,60:// Recuerde que Paro es NC
        “MB” := 1;
        “Mar” := 1;

    26,30,31,58,62,63://
        “MB” := 0;
        “Mar” := 1;//

   ELSE // En un Caso diferente se apagan las salidas MB y la Marca
         “MB” := 0;
        “Mar” := 0;//
END_CASE;
```

3.3.7. Instrucción FOR

Es una instrucción de repetición que permite que se ejecute una serie de instrucciones mientras una condición se mantenga; en este caso, la condición se da mientras se mantenga una variable dentro de un rango. Los rangos de la variable deben ser del mismo tipo, entero o doble entero (Siemens, 2005).

Esta instrucción se debe aplicar en instrucciones para ejecutar en el PLC que se requieran realizar en forma repetitiva durante un lapso muy corto; de lo contrario, pueden causar problemas de fallo en la ejecución del programa de PLC. Por ejemplo, no se recomienda suspender la normal ejecución del programa esperando que alguna condición se cumpla. La instrucción FOR debe ir acompañada del rango que debe cubrir la variable de control; es decir, los valores mínimo y máximo que se van a considerar y el incremento, especificando valor y sentido (incremento o decremento).

La instrucción FOR se leería: "Para un valor inicial de la variable de chequeo hasta el valor final, con incrementos/decrementos de (número o expresión), haga o ejecute las instrucciones incluidas"; luego, un "contador" incrementa o decrementa las veces definidas para cada bucle para aproximarse al límite definido (mínimo o máximo). Si el contador de la variable de control aún está dentro del rango, la instrucción se ejecuta nuevamente; de lo contrario, se sale de la ejecución del bloque.

Un ejemplo de la estructura FOR es el siguiente; el significado de las variables utilizadas se muestra en la Tabla 15.

```
FOR "Sumador" := 1 TO 40 DO // La Altura se recalcula mientras Sumador
    "Altura" := "Altura" - 8 // esté entre 1 y 40 con incrementos de 1
    ; // Si sumador está entre 1 y 40 se repite el ciclo
END_FOR; // Si Sumador es superior a 40 se sale del ciclo

FOR "Contador" := "Valor_inicial" TO "Valor_Final" BY "Incrementos" DO
    // Acá los límites y los incrementos se dan por variables
    // Es posible también utilizar expresiones
    "Sentencia" := "Sentencia" + 5;
END_FOR;
```

Tabla 15.

Variables utilizadas para el ejemplo de la estructura FOR

Variable	Tipo	Dirección	Significado
Contador	Int	%MW30	Para llevar una cuenta
Valor_inicial	Int	%MW36	Para inicializar el contador
Valor_Final	Int	%MW38	Para indicar el límite final de la ejecución de los ciclos

Incrementos	Int	%MW40	Indica las veces que incrementa el contador por cada ciclo que se ejecuta
Sentencia	Int	%MW42	Instrucción que se quiere ejecutar mientras se mantenga dentro del ciclo
Altura	Real	%MD44	
Sumador	Dint	%MD48	Para llevar otra cuenta

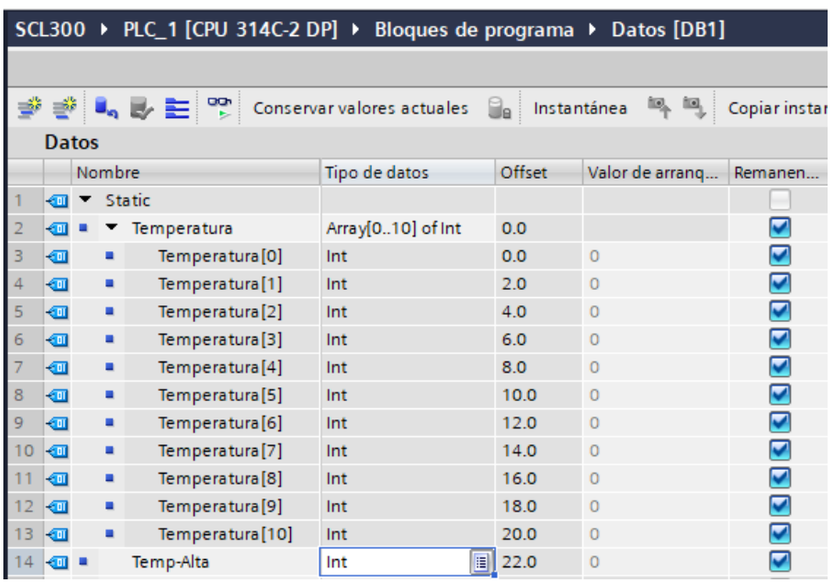
Fuente: elaboración propia.

Problema 3.12. En una base de datos se está almacenando el valor de temperatura tomada en once puntos diferentes de un proceso. Se requiere almacenar, en una variable ubicada en la misma base de datos, el valor de la temperatura más alta de las once.

Observe que este es un proceso repetitivo que se puede ejecutar en forma continua sin esperar a que se dé alguna condición externa al ciclo una vez ha comenzado; la única condición requerida para salir de la rutina se alcanza con el valor extremo del "contador de ciclos". Para este caso, se genera una base de datos que contenga un "Array" de 11 elementos como entero correspondiente a las temperaturas medidas y se le agrega una variable donde se almacenará la temperatura más alta; (Figura 28). También se genera un puntero en la tabla de variables en formato de entero que hará las veces de contador (Figura 29).

Figura 28.

Variables para emplear en la base de datos para solución del problema 3.12

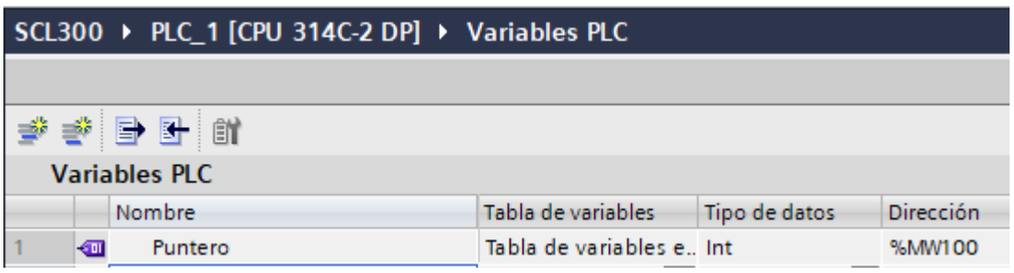


	Nombre	Tipo de datos	Offset	Valor de arranq...	Remanen...
1	Static				
2	Temperatura	Array[0..10] of Int	0.0		
3	Temperatura[0]	Int	0.0	0	
4	Temperatura[1]	Int	2.0	0	
5	Temperatura[2]	Int	4.0	0	
6	Temperatura[3]	Int	6.0	0	
7	Temperatura[4]	Int	8.0	0	
8	Temperatura[5]	Int	10.0	0	
9	Temperatura[6]	Int	12.0	0	
10	Temperatura[7]	Int	14.0	0	
11	Temperatura[8]	Int	16.0	0	
12	Temperatura[9]	Int	18.0	0	
13	Temperatura[10]	Int	20.0	0	
14	Temp-Alta	Int	22.0	0	

Fuente: elaboración propia.

Figura 29.

Puntero definido en una tabla de variables para solución del problema 3.12



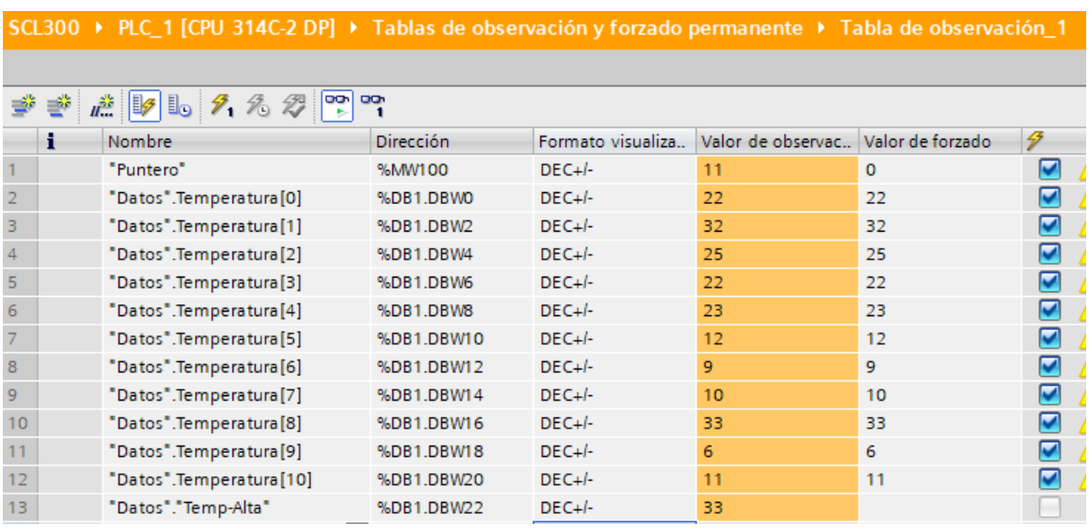
Fuente: elaboración propia.

El programa en SCL para la solución del problema 3.12 se muestra a continuación. La clave para su solución consiste en estar actualizando, en cada ciclo, el registro que contiene la temperatura más alta. En cada iteración se compara el contenido del registro indicado por "Puntero" con el registro de la variable "Temp-Alta". De esta forma, "Puntero" hace las veces de contador de ciclos y se incrementa en uno cada vez que se llega al final del ciclo FOR. El resultado de la simulación se muestra en la Figura 30 y se puede observar que la temperatura más alta es de 33 °C, ubicada en el puntero 8.

```
"Datos"."Temp-Alta" := 0;// Inicialmente el valor de la temperatura más
alta se refresca
FOR "Puntero" := 0 TO 10 DO //11 ciclos
    IF "Datos".Temperatura["Puntero"]>"Datos"."Temp-Alta" THEN
        "Datos"."Temp-Alta" := "Datos".Temperatura["Puntero"];
    END_IF;
END_FOR;
```

Figura 30.

Resultado de la simulación realizada para el problema 3.12



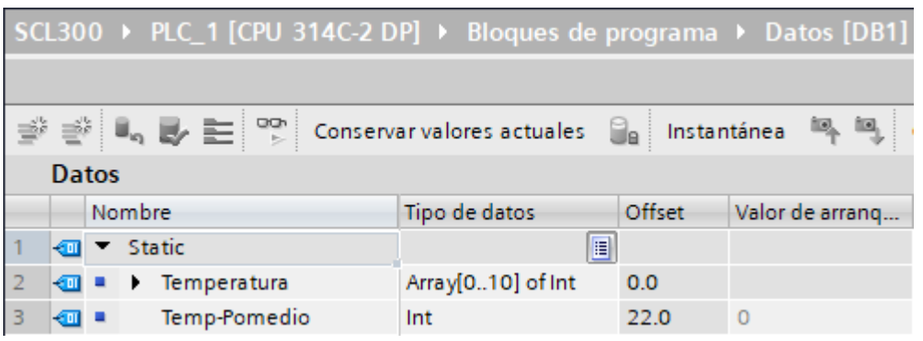
Fuente: elaboración propia.

Problema 3.13. En una base de datos se está almacenando el valor de temperatura tomada en once puntos diferentes de un proceso. Se requiere almacenar, en una variable ubicada en la misma base de datos, el valor promedio de las temperaturas. Se deben descartar aquellos valores de temperatura iguales a cero.

En la Figura 31 se muestran las variables "Temperatura" y "Temp-Promedio"; la primera, en un arreglo de Array de once elementos como entero con índices que van del 0 al 11, usada para almacenar el valor de las temperaturas. La segunda es una variable simple en formato de entero en la que se va a depositar el valor de la temperatura promedio. Se debe crear una variable llamada "Puntero" en MW10 como entero, encargada de definir el valor del índice del Array de la variable "Temperatura".

Figura 31.

Variables en base de datos para el problema 3.13

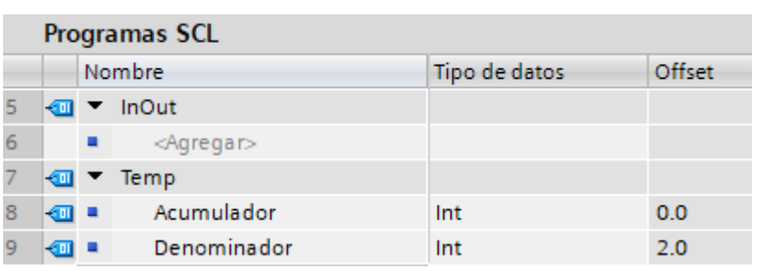


Fuente: elaboración propia.

Se van a utilizar dos variables locales temporales, #Acumulador y #Denominador, usadas para realizar operaciones intermedias que no se requieren almacenar. En #Acumulador se llevará la cuenta de la suma de las temperaturas, mientras que en #Denominador se llevará la cuenta de las temperaturas diferentes de 0 (Figura 32).

Figura 32.

Variables locales temporales en base de datos para el problema 3.13



Fuente: elaboración propia.

Se presenta a continuación un programa en SCL que responde a la solución del problema. Para realizar el promedio por once ciclos continuos se va sumando el valor de cada una de las temperaturas y el resultado se almacena en #Acumulador; la variable #Denominador se incrementa en 1 en cada ciclo de la rutina FOR, siempre y cuando el valor de la temperatura del ciclo actual sea diferente de 0; de esta forma, se están descartando aquellos valores de temperatura iguales a 0. Cuando se terminan los ciclos, se realiza la división de la variable #Acumulador entre la variable #Denominador, con lo que se obtiene la variable "Temp-Promedio" en la base de datos.

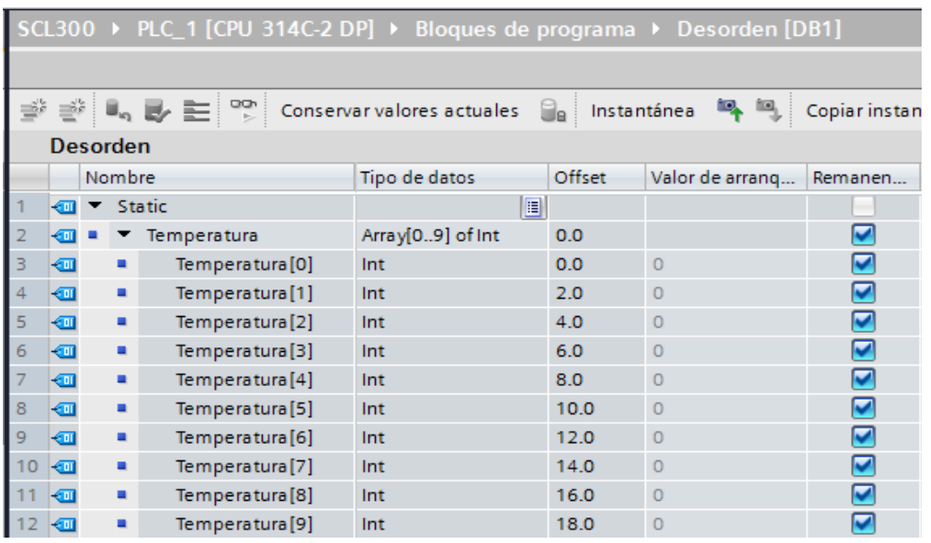
```
#Acumulador := 0;
#Denominador := 0;
FOR "Puntero" := 0 TO 10 DO
    #Acumulador := #Acumulador + "Datos".Temperatura["Puntero"];
    IF "Datos".Temperatura["Puntero"]=0 THEN
        #Denominador := #Denominador;
    ELSE
        #Denominador := #Denominador + 1;
    END_IF;

END_FOR;
"Datos"."Temp-Pomedio" := #Acumulador / #Denominador;
```

Problema 3.14. Se tiene el registro de diez valores de temperatura en una base de datos; se quiere tener en otra base de datos el valor de estas variables en orden ascendente. La actualización de esta nueva base de datos se debe dar siempre que se tiene el flanco positivo de una entrada digital (I124.0). La actualización consiste en transferir los datos de la primera tabla a la segunda; la organización en la segunda tabla se hace en forma automática.

Para la solución se crea una base de datos "Desorden" donde se almacena el valor de las temperaturas que se deben ordenar (Figura 33). Se tiene un arreglo o Array para la variable "Temperatura" compuesto por diez índices del 0 al 9 como entero.

Figura 33.

Base de datos de las temperaturas en desorden de problema 3.14



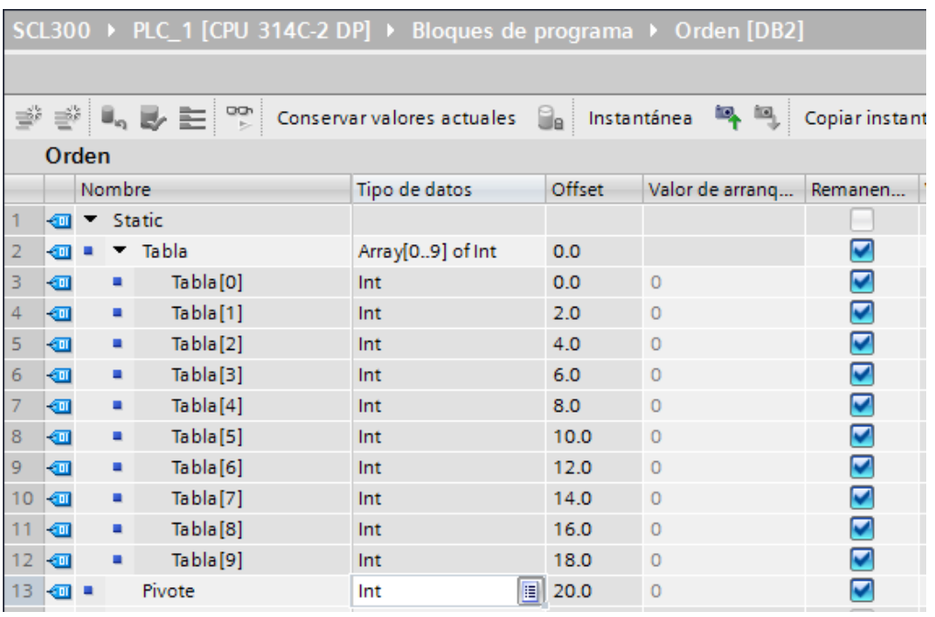
	Nombre	Tipo de datos	Offset	Valor de arranq...	Remanen...
1	Static				
2	Temperatura	Array[0..9] of Int	0.0		
3	Temperatura[0]	Int	0.0	0	
4	Temperatura[1]	Int	2.0	0	
5	Temperatura[2]	Int	4.0	0	
6	Temperatura[3]	Int	6.0	0	
7	Temperatura[4]	Int	8.0	0	
8	Temperatura[5]	Int	10.0	0	
9	Temperatura[6]	Int	12.0	0	
10	Temperatura[7]	Int	14.0	0	
11	Temperatura[8]	Int	16.0	0	
12	Temperatura[9]	Int	18.0	0	

Fuente: elaboración propia.

Se crea otra base de datos llamada “Orden” donde se pondrán las temperaturas en orden ascendente en un Array de diez elementos denominado “Tabla”. Observe que se agregó una variable llamada “Pivote”, como entero, que se utilizará para el intercambio de datos cuando se esté averiguando por el dato menor (Figura 34).

Figura 34.

Base de datos donde se colocan las temperaturas en forma ascendente para el problema 3.14



	Nombre	Tipo de datos	Offset	Valor de arranq...	Remanen...
1	Static				
2	Tabla	Array[0..9] of Int	0.0		
3	Tabla[0]	Int	0.0	0	
4	Tabla[1]	Int	2.0	0	
5	Tabla[2]	Int	4.0	0	
6	Tabla[3]	Int	6.0	0	
7	Tabla[4]	Int	8.0	0	
8	Tabla[5]	Int	10.0	0	
9	Tabla[6]	Int	12.0	0	
10	Tabla[7]	Int	14.0	0	
11	Tabla[8]	Int	16.0	0	
12	Tabla[9]	Int	18.0	0	
13	Pivote	Int	20.0	0	

Fuente: elaboración propia.

En la Tabla 16 se declaran las variables y los apuntadores I y J se usan para tener control sobre las variables que componen los Array de “Temperatura” y “Tabla”. El detector de flanco se utiliza para que las variables del Array “Tabla” solo se actualicen con el flanco positivo de I124.0.

Tabla 16.

Variables para el problema 3.14

Nombre	Tipo	Dirección	Comentario
J	Int	%MW10	Apuntador J
I	Int	%MW12	Apuntador I
Carga de datos	Bool	%I124.0	Actualizar tabla
Detector de flanco	Bool	%M20.1	Para detectar flanco de subida
Auxiliar	Bool	%M20.0	Auxiliar para el flanco

Fuente: elaboración propia.

El programa se divide en dos partes: la primera transfiere los datos de la base de datos “Desorden” a la base de datos “Orden” al detectarse el flanco positivo de I124.0; entonces, bajo un condicional IF se entra en el ciclo FOR que se ejecuta 10 veces para transferir cada dato del Array “Temperatura” al Array “Tabla” en forma consecutiva, lo que es controlado por el apuntador “I”. La segunda parte del programa consiste en dos rutinas FOR anidado, controlados por los apuntadores “I” y “J”. En las iteraciones más anidadas, lo que se trata de hacer es ir dejando el valor más bajo de las temperaturas en las posiciones cuyo apuntador es inferior; entonces, si una temperatura de una posición más adelante es menor que la temperatura de la posición actual se procede a realizar el intercambio de valores para que queden en orden; es en este momento cuando se utiliza el “Pivote” para que guarde provisionalmente el valor más bajo. Esto se hace por diez ciclos, pero no asegura que las temperaturas hayan quedado en perfecto orden. La iteración exterior posibilita asegurarse de que todas las posiciones de memoria donde se alojan las temperaturas sean sometidas al mismo proceso, desde la cero hasta la diez.

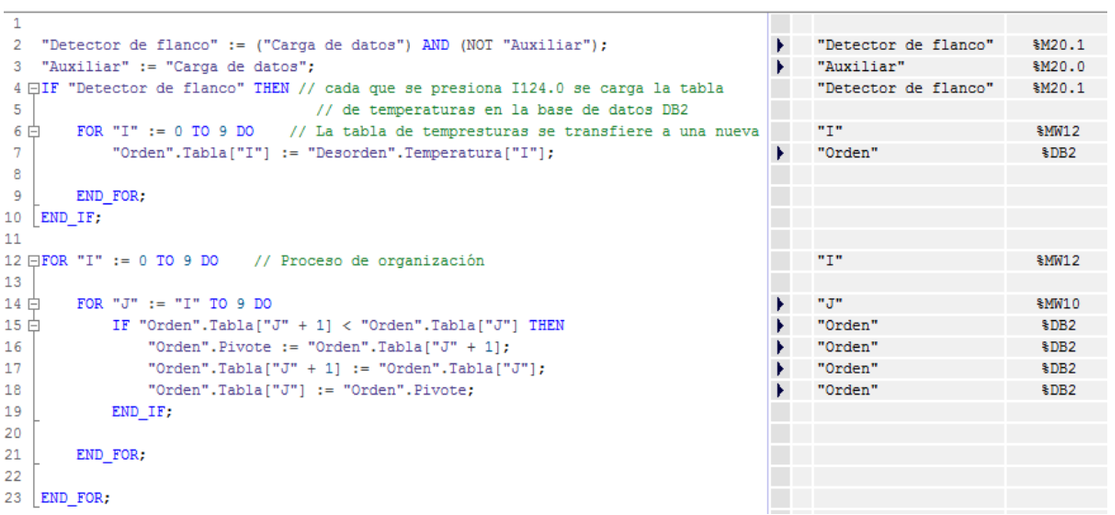
El algoritmo en SCL para el problema 3.14 se muestra a continuación; se puede asegurar que la tabla queda totalmente organizada luego de que se han realizado varias ejecuciones del programa, es decir, si el programa está dentro de una subrutina esta debe ejecutarse varias veces. Esto se debe a que, en la subrutina anidada, la ejecución no empieza desde 0, sino desde “I”, pudiendo quedar en las posiciones más bajas valores de temperatura que no corresponden a los más bajos, requiriéndose nuevas ejecuciones del ciclo para que se posicionen adecuadamente, en especial cuando los valores bajos de temperatura quedaron ocupando las posiciones más altas en la disposición original, por lo que no fueron incluidas en las primeras iteraciones.

```
“Detector de flanco” := (“Carga de datos”) AND (NOT “Auxiliar”);

“Auxiliar” := “Carga de datos”;

IF “Detector de flanco” THEN // cada que se presiona I124.0 se carga la tabla

                    // de temperaturas en la base de datos DB2

    FOR “I” := 0 TO 9 DO    // La tabla de temperaturas se transfiere a
una nueva tabla

        “Orden”.Tabla[“I”] := “Desorden”.Temperatura[“I”];

    END_FOR;

END_IF;

FOR “I” := 0 TO 9 DO    // Proceso de organización

    FOR “J” := “I” TO 9 DO
        IF “Orden”.Tabla[“J” + 1] < “Orden”.Tabla[“J”] THEN
            “Orden”.Pivote := “Orden”.Tabla[“J” + 1];
            “Orden”.Tabla[“J” + 1] := “Orden”.Tabla[“J”];
            “Orden”.Tabla[“J”] := “Orden”.Pivote;
        END_IF;

    END_FOR;
END_FOR;
```

En la Figura 35 se muestra la apariencia del programa en TIA PORTAL.

Figura 35.

Apariencia del programa en TIA PORTAL del problema 3.14



Fuente: elaboración propia.

En la Figura 36 se muestra el resultado de la simulación; aparecen en la columna de forzado los valores de temperatura en desorden. En la columna de observación se pueden ver los datos completamente organizados luego de correr varias veces la subrutina.

Figura 36.

Simulación del programa del problema 3.14

i	Nombre	Dirección	Formato visualiza..	Valor de observac..	Valor de forzado
	"Desorden".Temperatura[0]	%DB1.DBW0	DEC	45	45
	"Desorden".Temperatura[1]	%DB1.DBW2	DEC	85	85
	"Desorden".Temperatura[2]	%DB1.DBW4	DEC	22	22
	"Desorden".Temperatura[3]	%DB1.DBW6	DEC	45	45
	"Desorden".Temperatura[4]	%DB1.DBW8	DEC	2	2
	"Desorden".Temperatura[5]	%DB1.DBW10	DEC	7	7
	"Desorden".Temperatura[6]	%DB1.DBW12	DEC	9	9
	"Desorden".Temperatura[7]	%DB1.DBW14	DEC	9	9
	"Desorden".Temperatura[8]	%DB1.DBW16	DEC	32	32
	"Desorden".Temperatura[9]	%DB1.DBW18	DEC	1	1
	"Orden".Tabla[0]	%DB2.DBW0	DEC+/-	1	
	"Orden".Tabla[1]	%DB2.DBW2	DEC+/-	2	
	"Orden".Tabla[2]	%DB2.DBW4	DEC+/-	7	
	"Orden".Tabla[3]	%DB2.DBW6	DEC+/-	9	
	"Orden".Tabla[4]	%DB2.DBW8	DEC+/-	9	
	"Orden".Tabla[5]	%DB2.DBW10	DEC+/-	22	
	"Orden".Tabla[6]	%DB2.DBW12	DEC+/-	32	
	"Orden".Tabla[7]	%DB2.DBW14	DEC+/-	45	
	"Orden".Tabla[8]	%DB2.DBW16	DEC+/-	45	
	"Orden".Tabla[9]	%DB2.DBW18	DEC+/-	85	
	"Orden".Pivote	%DB2.DBW20	DEC+/-	85	
	"J"	%MW10	DEC+/-	10	
	"I"	%MW12	DEC+/-	10	

Fuente: elaboración propia.

En forma alternativa se presenta un programa que requiere solamente de un llamado de la subrutina para dejar completamente organizada la tabla. El cambio simplemente consistió en dejar el inicio del ciclo FOR anidado desde 0 y no desde "I" como fue planteado en el algoritmo anterior. Esto podría tener incidencia para tablas demasiado grandes. En este caso, el número de iteraciones sería de 10 veces de "J" por cada iteración de "I", o sea, 100 veces. Piense en una tabla que contenga 100 registros; serían 10 000 veces que se tienen que repetir las operaciones, aumentando considerablemente el tiempo de ejecución de un solo ciclo.

```
"Detector de flanco" := ("Carga de datos") AND (NOT "Auxiliar");
"Auxiliar" := "Carga de datos";
IF "Detector de flanco" THEN // cada que se presiona I124.0 se carga la
tabla
                            // de temperaturas en la base de datos DB2
    FOR "I" := 0 TO 9 DO    // La tabla de temperaturas se transfiere a
una nueva tabla
        "Orden".Tabla["I"] := "Desorden".Temperatura["I"];

    END_FOR;
END_IF;

FOR "I" := 0 TO 9 DO    // Proceso de organización
```

```
    FOR “J” := 0 TO 9 DO
        IF “Orden”.Tabla[“J” + 1] < “Orden”.Tabla[“J”] THEN
            “Orden”.Pivote := “Orden”.Tabla[“J” + 1];
            “Orden”.Tabla[“J” + 1] := “Orden”.Tabla[“J”];
            “Orden”.Tabla[“J”] := “Orden”.Pivote;
        END_IF;

    END_FOR;
END_FOR;
```

3.3.8. Instrucción WHILE

También es de repetición. Las instrucciones se repiten mientras se cumpla una condición que determina su ejecución (Peciña Belmonte, 2018). La condición de evaluación es una expresión que puede ser de tipo Booleano, verdadero o falso, resultante de una operación lógica o una comparación. La forma directa de salir del ciclo WHILE es con la instrucción END_WHILE, pero también se pueden utilizar las instrucciones CONTINUE o EXIT (Hernández Corssi, 2019). El bucle WHILE también permite el anidamiento.

El siguiente es un ejemplo de la estructura WHILE, que significa que las instrucciones que se ponen luego de “DO” se ejecutan mientras la condición se esté cumpliendo (Contador<10). Las instrucciones deben incluir una que tenga control sobre el resultado del condicionante para facilitar la salida de la subrutina WHILE, pudiendo ser un contador o el resultado de otra variable modificada.

```
WHILE “Contador” <10 DO
    “Area”:= “Area”+5;
    “contador”:=Contador+1;
END_WHILE;
```

Problema 3.15. Se dispone de un sensor de presión con rango de 0 a 80 PSI, conectado a un circuito electrónico con una salida de 0 a 10 VDC proporcionales a la presión que se está midiendo. Eventualmente, se debe realizar una calibración por software en el PLC para garantizar que las lecturas de presión sean adecuadas. Realice un programa para que, a partir de una tabla de calibración de 10 registros de voltaje y presión, se corrija la ecuación con que se debe hallar la presión, dados futuros voltajes de entrada.

Solución: en una base de datos se registrarán los 10 valores de voltaje y los 10 valores de presión arrojados por un instrumento patrón. Luego de dar una orden se debe realizar una regresión lineal para ajustar una línea recta, utilizando el método de mínimos cuadrados para encontrar la nueva ecuación con la que se calculará la presión hasta una nueva calibración (De la Puente Viedma, 2018).

Las ecuaciones por utilizar en la regresión, para ajuste de la recta por mínimos cuadrados, serán las que se presentan a continuación. La ecuación 2 describe la línea recta donde "a_0" representa la intercepción con el eje de las abscisas y "a_1" la pendiente. En este caso "y" corresponde a la variable presión y "V" a la variable voltaje del problema 3.15. En la ecuación 3 se muestra el desarrollo que permite hallar el coeficiente "a_1", que utiliza la técnica de minimizar la suma de los cuadrados de los residuos (ecuación normal) donde "n" corresponde al número de muestras; en este caso 10, "v_i" y "p_i" son las parejas de valores de cada muestra de voltaje y presión. La ecuación 4 determina la forma para hallar "a_0" utilizando la misma técnica: "p_{prom}" es la media de la variable presión y "v_{prom}" es la media de la variable voltaje. En las ecuaciones 5 y 6 se describen las fórmulas para hallar "p_{prom}" y "v_{prom}"

Ecuación 2.

Que describe la línea recta del problema 3.15

$$y = a_0 + a_1 * V$$

Ecuación 3.

Hallar coeficiente por método de suma de los mínimos cuadrados de los residuos

$$a_1 = \left(n * \sum v_i * p_i\right) - \left(\sum v_i * \sum p_i\right) / \left(n * \sum v_i^2 - \left(\sum v_i\right)^2\right)$$

Ecuación 4.

Hallar coeficiente "a_0" por método de suma de los mínimos cuadrados de los residuos

$$a_0 = Pprom + a_1 * vprom$$

Ecuación 5.

Fórmula para encontrar el valor medio de la presión

$$p_{prom} = \left(\sum p_i\right) / n$$

Ecuación 6.

Fórmula para encontrar el valor medio del voltaje

$$v_{prom} = \left(\sum v_i\right) / n$$

En la Tabla 17 se muestra un ejemplo de lo que sería el proceso de calibración y de cálculo de la ecuación correspondiente. La columna "Muestra" indica cada una de las pruebas que se realizaron, en las columnas Voltaje(i) y Presión(i) se registran los valores respectivos de las lecturas realizadas, en V*P se realiza la multiplicación de las dos variables, y en V^2 se obtiene el cuadrado de la variable voltaje para cada lectura. Al final de cada columna se realiza la sumatoria de todas las muestras y los resultados.

Tabla 17.

Cálculos realizados para cada par de lecturas del problema 3.15

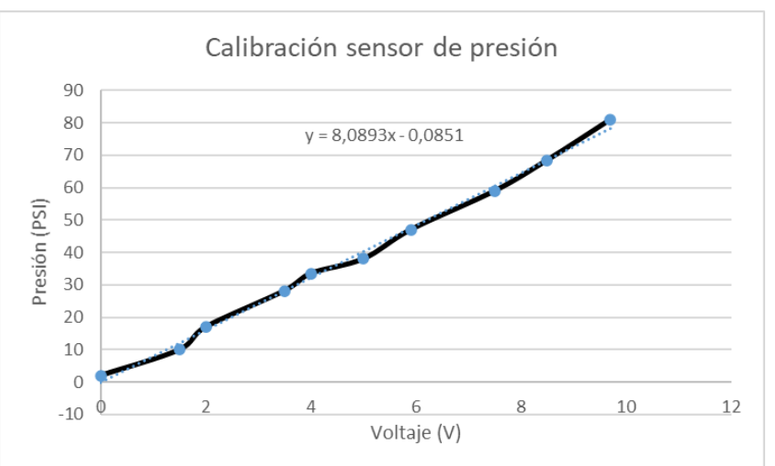
Muestra	Voltaje(i)	Presión(i)	V*P	V^2
1	0	2	0	0
2	1.5	10	15	2.25
3	2	17	34	4
4	3.5	28.2	98.7	12.25
5	4	33.5	134	16
6	5	38	190	25
7	5.9	47	277.3	34.81
8	7.5	59	442.5	56.25
9	8.5	68.5	582.25	72.25
10	9.7	81	785.7	94.09
Sumatoria	47.6	384.2	2559.45	316.9

Fuente: elaboración propia.

En la Figura 37 se muestra el resultado del ejemplo corrido en Excel. Observe que aparece una curva con los datos reales de calibración y una con la tendencia obtenida por medio de correlación lineal, arrojando la ecuación y = 8,0893x – 0,0851.

Figura 37.

Recta de calibración para el problema 3.5 construida en Excel



Fuente: elaboración propia.

Para solucionar el problema en el PLC, primero se crea una base de datos llamada “Datos” donde se declaran las variables de la Tabla 18. Observe que se crearon dos Array en formato real, de diez elementos cada uno, para la variable presión y para la variable voltaje. También se crearon variables que irán realizando la sumatoria de las diferentes columnas definidas en la tabla 17.

Tabla 18.

Variables en base de datos para el problema 3.15

Nombre	Tipo de datos	Nombre	Tipo de datos
Static		PresCal	Array[1..9] of Real
Presión	Real	PresCal[1]	Real
Voltaje	Real	PresCal[2]	Real
Contador	Int	PresCal[3]	Real
Calibración	Bool	PresCal[4]	Real
AuxiliarFlanco	Bool	PresCal[5]	Real
DetectorFlanco	Bool	PresCal[6]	Real
VolCal	Array[1..9] of Real	PresCal[7]	Real
VolCal[1]	Real	PresCal[8]	Real
VolCal[2]	Real	PresCal[9]	Real
VolCal[3]	Real	PresCal[10]	Real
VolCal[4]	Real	SumaVol	Real
VolCal[5]	Real	SumaPres	Real
VolCal[6]	Real	SumaV*P	Real
VolCal[7]	Real	SumaV*V	Real
VolCal[8]	Real	Volprom	Real
VolCal[9]	Real	PresProm	Real
VolCal[10]	Real	a0	Real
		a1	Real

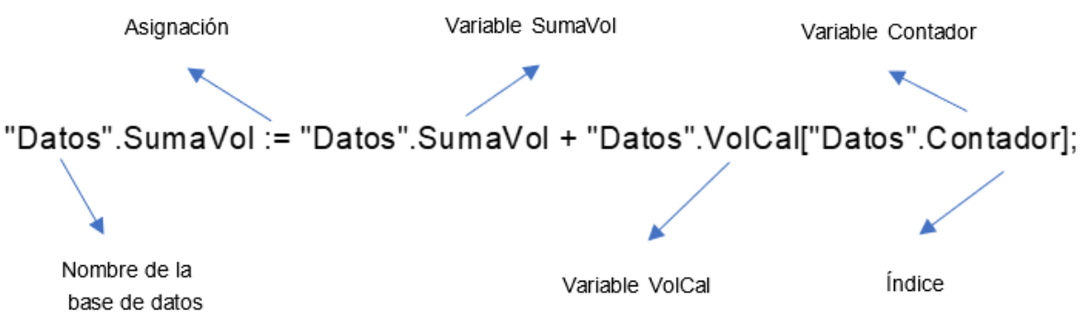
Fuente: elaboración propia.

Además, en la base de datos están depositadas las variables “Presión” y “Voltaje” donde finalmente deberá llegar el valor de voltaje y el algoritmo dará como salida la presión correspondiente a ese voltaje de acuerdo con la ecuación ajustada. La variable “Contador” se crea para delimitar las muestras con las que se realizará la regresión lineal. La entrada booleana “Calibración” sirve para recalcular la fórmula de regresión cada vez que se detecta un flanco de subida. ”Auxiliar flanco” y “Detector flanco” se utilizan para detectar el flanco de subida de la variable “Calibración”. En el arreglo “VolCal” se depositan los nueve valores de voltaje utilizados para la calibración; lo mismo se tiene configurado en el arreglo “PresCal” para los valores de presión. Las variables “SumaVol”, “SumaPres”, “SumaV*P”, “SumaV*V”, “Volprom”, “PresProm”, “a_0” y “a_1” se utilizan para desarrollar las fórmulas del algoritmo.

El siguiente es el programa en SCL para dar solución al problema 3.15. Se puede dividir en tres partes. La primera se encarga de detectar un flanco positivo para proceder a la calibración de la tabla. En caso de no ocurrir este flanco, se sigue con la tercera parte que consiste en estar encontrando el valor de la presión de acuerdo con el valor que da la variable "Voltaje" y los valores de los coeficientes "a_0" y "a_1" hallados en la parte 2 del programa. La segunda parte, que es la más extensa, es controlada por un ciclo WHILE (se repite diez veces) que contiene un "Contador" antes de cada incremento; se actualizan con los valores de las variables encargadas de llevar la cuenta de las sumas con los valores indicados en las variables cuyo índice corresponde al mismo "Contador". A modo de ejemplo, se detalla una de las instrucciones de esta segunda sección del programa en la Figura 38.

Figura 38.

Detalle de una instrucción del problema 3.15



Fuente: elaboración propia.

```
"Datos".DetectorFlanco := "Datos".Calibración AND NOT "Datos".AuxiliarFlanco ;
"Datos".AuxiliarFlanco := "Datos".Calibración;
IF "Datos".DetectorFlanco THEN
    "Datos".Contador := 1;
    WHILE "Datos".Contador <= 10 DO
     "Datos".SumaVol := "Datos".SumaVol + "Datos".VolCal["Datos".Contador];
     "Datos".SumaPres := "Datos".SumaPres + "Datos".PresCal["Datos".Contador];
     "Datos"."SumaV*V" := "Datos"."SumaV*V" + "Datos".VolCal["Datos".Conta-
dor] * "Datos".VolCal["Datos".Contador];
        "Datos"."SumaV*P" := "Datos"."SumaV*P" + "Datos".VolCal["Datos".
Contador] * "Datos".PresCal["Datos".Contador];
        "Datos".Contador := "Datos".Contador + 1;
    END_WHILE;
    "Datos".Volprom := "Datos".SumaVol / 10;
    "Datos".PresProm := "Datos".SumaPres / 10;
    "Datos".a1 := (10 * "Datos"."SumaV*P" - "Datos".SumaVol * "Datos".Suma-
Pres) / (10 * "Datos"."SumaV*V" - "Datos".SumaVol * "Datos".SumaVol);
    "Datos".a0 := "Datos".PresProm - ("Datos".a1 * "Datos".Volprom);
END_IF;
"Datos".Presión :="Datos".a0 +("Datos".a1*"Datos".Voltaje);
```

En la Figura 39 se muestra la tabla de simulación para poner a prueba el algoritmo del problema 3.15. Así, cuando el voltaje es 2 V, la presión calculada es de 16,09353 PSI.

Figura 39.

Tabla de simulación para el problema 3.15

Datos.a0	%DB1.DBD116	Número en coma flotante	#¡CAMPO!		False
Datos.a1	%DB1.DBD120	Número en coma flotante	8.089.301		False
Datos.Presión	%DB1.DBD0	Número en coma flotante	1.609.353		False
Datos.Voltaje	%DB1.DBD4	Número en coma flotante	2.0	2.0	True
Datos.Contador	%DB1.DBW8	DEC+/-	11		False
Datos.Calibración	%DB1.DBX10.0	BOOL	FALSE	FALSE	True
Datos.AuxiliarFlanco	%DB1.DBX10.1	BOOL	FALSE		False
Datos.DetectorFlanco	%DB1.DBX10.2	BOOL	FALSE		False
Datos.VolCal	P#DB1.DBX12.0				False
Datos.VolCal[1]	%DB1.DBD12	Número en coma flotante	0.0	0.0	True
Datos.VolCal[2]	%DB1.DBD16	Número en coma flotante	1.5	1.5	True
Datos.VolCal[3]	%DB1.DBD20	Número en coma flotante	2.0	2.0	True
Datos.VolCal[4]	%DB1.DBD24	Número en coma flotante	3.5	3.5	True
Datos.VolCal[5]	%DB1.DBD28	Número en coma flotante	4.0	4.0	True
Datos.VolCal[6]	%DB1.DBD32	Número en coma flotante	5.0	5.0	True
Datos.VolCal[7]	%DB1.DBD36	Número en coma flotante	5.9	5.9	True
Datos.VolCal[8]	%DB1.DBD40	Número en coma flotante	7.5	7.5	True
Datos.VolCal[9]	%DB1.DBD44	Número en coma flotante	8.5	8.5	True
Datos.VolCal[10]	%DB1.DBD48	Número en coma flotante	9.7	9.7	True
Datos.PresCal	P#DB1.DBX52.0				False
Datos.PresCal[1]	%DB1.DBD52	Número en coma flotante	2.0	2.0	True
Datos.PresCal[2]	%DB1.DBD56	Número en coma flotante	10.0	10.0	True
Datos.PresCal[3]	%DB1.DBD60	Número en coma flotante	17.0	17.0	True
Datos.PresCal[4]	%DB1.DBD64	Número en coma flotante	28.2	28.2	True
Datos.PresCal[5]	%DB1.DBD68	Número en coma flotante	33.5	33.5	True
Datos.PresCal[6]	%DB1.DBD72	Número en coma flotante	38.0	38.0	True
Datos.PresCal[7]	%DB1.DBD76	Número en coma flotante	47.0	47.0	True
Datos.PresCal[8]	%DB1.DBD80	Número en coma flotante	59.0	59.0	True
Datos.PresCal[9]	%DB1.DBD84	Número en coma flotante	68.5	68.5	True
Datos.PresCal[10]	%DB1.DBD88	Número en coma flotante	81.0	81.0	True

Fuente: elaboración propia.

3.3.9. Instrucción REPEAT

Es de repetición y permite la ejecución de una serie de instrucciones hasta que se da una instrucción lógica de interrupción UNTIL. La instrucción REPEAT se diferencia de la instrucción WHILE en que en la primera se ejecutan las órdenes por lo menos una vez, ya que la condición se evalúa al final; por el contrario, en REPEAT la evaluación se realiza al comienzo y, por lo tanto, se puede salir antes de ejecutar la instrucción (Peciña Belmonte, 2018).

La ejecución actual de un ciclo REPEAT se puede interrumpir por medio de las instrucciones CONTINUE, o también se puede salir definitivamente del ciclo por medio de la instrucción EXIT. En el siguiente ejemplo en SCL se describe la estructura de un ciclo REPEAT. Considere que "Nueva_Temperatura" corresponde a la dirección MD10 como real, "Temperatura" corresponde a MD14 y "Bloqueo_Sensor" es la marca M2.0. De esta forma, se estará actualizando en cada ciclo con el valor de "Nueva_Temperatura" hasta que "Bloqueo_Sensor" se ponga en verdadero. Recuerde que se debe evitar que el programa quede en la ejecución indefinida de este bucle, situación que puede provocar un fallo en el PLC.

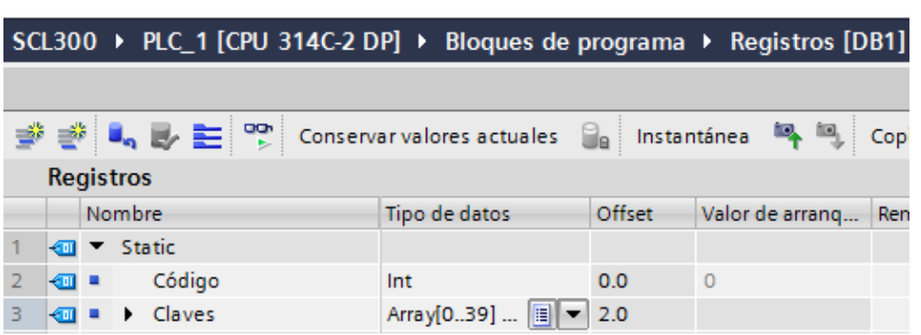
```
REPEAT "Nueva_Temperatura":= "Temperatura";
UNTIL "Bloqueo_Sensor"
END_REPEAT;
```

Problema 3.16. En una base de datos se tiene un registro 40 códigos de acceso a un área supervisada. Para tal fin, una persona introduce un código y solicita la validación por medio de un pulsador. En caso de que el código esté registrado, se activa una puerta de entrada que es cerrada por la misma persona por medio de un pulsador ubicado en la parte interior. En caso de que el código no esté registrado, se activa una alarma hasta que se presiona el mismo pulsador desde la parte interior del área.

En la Figura 40 se muestra la base de datos creada "Registros". Observe que la variable "Claves" en realidad es un Array de cuarenta registros como entero. Más adelante en ella se deben guardar los registros permitidos. En la variable "Código" se depositará la clave ingresada por cada usuario que quiera entrar.

Figura 40.

Base de datos registros para el problema 3.16

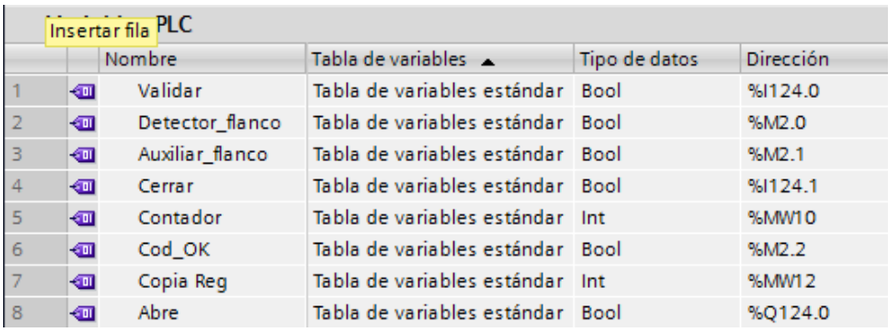


Fuente: elaboración propia.

En la Figura 41 aparecen las demás variables que se van a utilizar para dar solución al problema 3.16. "Validar" será activado por el usuario que quiere poner a prueba un nuevo código, "Cerrar" se utiliza por la persona que ingresa para cerrar la puerta, "Contador" se utiliza como apuntador de cada uno de los códigos registrados como válidos, "Cod_OK" es una marca que genera el mismo programa cuando un código ha sido validado como correcto, "Copia Reg" se usa para tener una copia del código, y "Abre" es la salida para activar la apertura de la puerta; se debe agregar "Alarma" para la salida Q124.1.

Figura 41.

Variables PLC para el problema 3.16



	Nombre	Tabla de variables	Tipo de datos	Dirección
1	Validar	Tabla de variables estándar	Bool	%I124.0
2	Detector_flanco	Tabla de variables estándar	Bool	%M2.0
3	Auxiliar_flanco	Tabla de variables estándar	Bool	%M2.1
4	Cerrar	Tabla de variables estándar	Bool	%I124.1
5	Contador	Tabla de variables estándar	Int	%MW10
6	Cod_OK	Tabla de variables estándar	Bool	%M2.2
7	Copia Reg	Tabla de variables estándar	Int	%MW12
8	Abre	Tabla de variables estándar	Bool	%Q124.0

Fuente: elaboración propia.

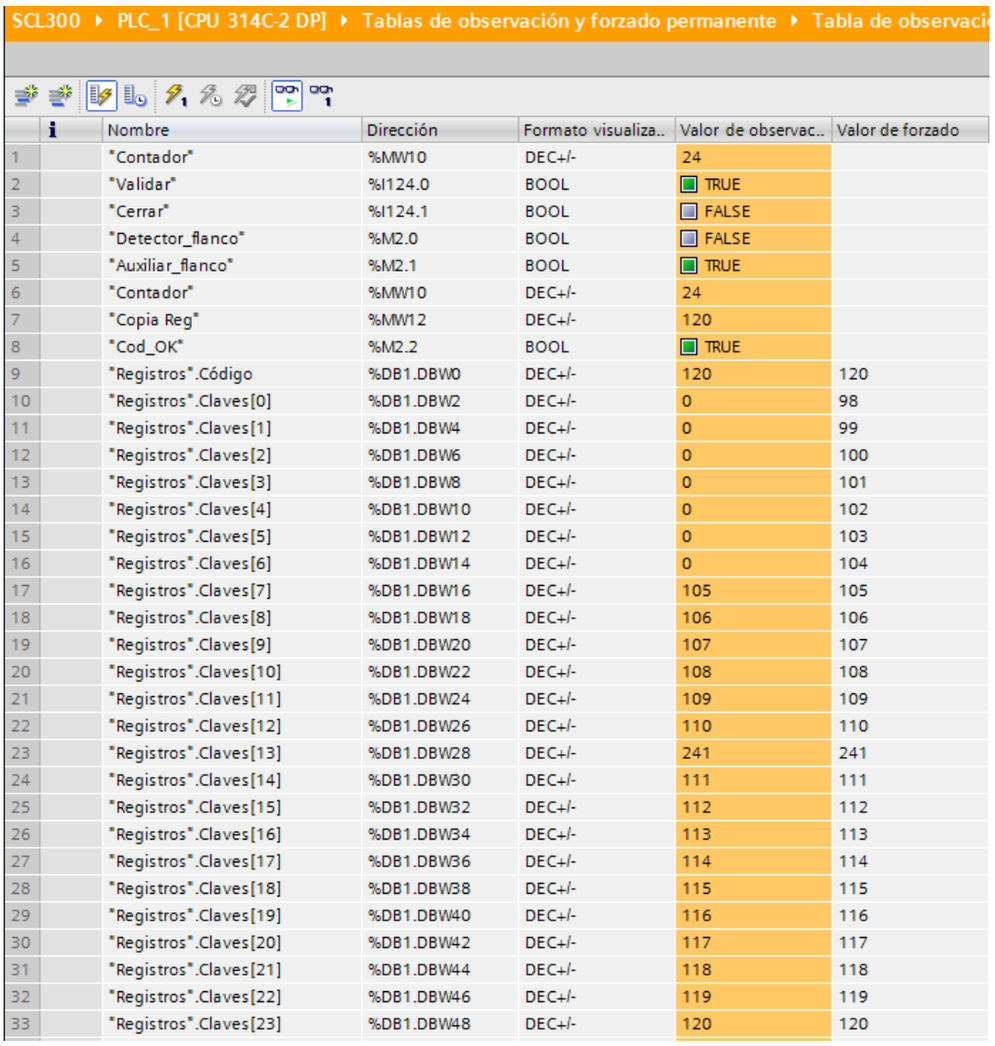
El programa sugerido en SCL se muestra a continuación. Previamente, el usuario debió ingresar un "Código"; luego, al detectarse el flanco positivo de "Validar", se reinicia "Contador", se entra en el ciclo REPEAT y se mantendrá allí mientras no se haya dado la condición de que se encontró un código igual al ingresado ("Registros".Código = "Registros".Claves["Contador"-1]) o el contador ya supera el número 39, indicando que ya se leyeron los 40 registros. Lo que se hace en el ciclo REPEAT es ir verificando si el código ingresado coincide con el número de la variable apuntada por "contador"; de ocurrir esta coincidencia, se activa "Cod_OK y "Copia Reg" es actualizado con el código que fue encontrado (el apuntado por el "Contador"); de lo contrario, solo se procede a incrementar el contador para permitir hacer la próxima verificación con un registro más alto. Ya fuera del ciclo REPEAT, si "Cod_ OK" es verdadero, entonces se activa "Abre" que significa activar Q124.0 para abrir la puerta. Luego, si se pulsa el botón de "Cerrar", entonces se desenergiza Q124.0 (cierra la puerta) y se reinician "Cod_OK", "Alarma", "Contador" y "Copia Reg". En el caso de que el contador haya alcanzado el valor de 40, significa que el código ingresado no coincidió con ninguna de los que se tienen registrados, dando lugar a que se active la "Alarma" Q124.1 y "Copia Reg" queda cargado con el registro ingresado por el usuario; así permanece hasta que alguien en el interior pulse "Cerrar" (I124.1).

```
"Detector_flanco" := "Validar" AND NOT "Auxiliar_flanco";
"Auxiliar_flanco" := "Validar";
IF "Detector_flanco" THEN
    "Contador" := 0;
    REPEAT
        IF "Registros".Código = "Registros".Claves["Contador"] THEN
            "Cod_OK" := 1;
            "Copia Reg" := "Registros".Claves["Contador"];
        END_IF;

        "Contador" := "Contador" + 1;
    UNTIL
dor"-1] "Contador"=40 OR "Registros".Código = "Registros".Claves["Conta-
    END_REPEAT;
END_IF;
IF "Cod_OK" THEN
    "Abre" := 1;
END_IF;
IF "Cerrar" THEN
    "Abre" := 0;
    "Cod_OK" := 0;
    "Alarma" := 0;
    "Contador" := 0;
    "Copia Reg" := 0;

END_IF;
IF "Contador"= 40 THEN
    "Alarma" := 1;
    "Copia Reg" := "Registros".Código;
END_IF;
```

Se muestra el resultado de una simulación donde en la variable "Registros". Código de la base de datos se ingresó el código 120 y el algoritmo lo encontró (en la posición 23), luego de intentar con 24 códigos diferentes, provocando que la señal para apertura de la puerta se activara. Todo esto, luego de pulsar la entrada correspondiente a "Validar". Esta misma señal se apagará cuando se dé la orden de cerrar la puerta, "Cerrar" (Figura 42).

Figura 42.

Simulación para el problema 3.16



SCL300 ▸ PLC_1 [CPU 314C-2 DP] ▸ Tablas de observación y forzado permanente ▸ Tabla de observaci

	i	Nombre	Dirección	Formato visualiza..	Valor de observac..	Valor de forzado
1		"Contador"	%MW10	DEC+/-	24	
2		"Validar"	%I124.0	BOOL	TRUE	
3		"Cerrar"	%I124.1	BOOL	FALSE	
4		"Detector_flanco"	%M2.0	BOOL	FALSE	
5		"Auxiliar_flanco"	%M2.1	BOOL	TRUE	
6		"Contador"	%MW10	DEC+/-	24	
7		"Copia Reg"	%MW12	DEC+/-	120	
8		"Cod_OK"	%M2.2	BOOL	TRUE	
9		"Registros".Código	%DB1.DBW0	DEC+/-	120	120
10		"Registros".Claves[0]	%DB1.DBW2	DEC+/-	0	98
11		"Registros".Claves[1]	%DB1.DBW4	DEC+/-	0	99
12		"Registros".Claves[2]	%DB1.DBW6	DEC+/-	0	100
13		"Registros".Claves[3]	%DB1.DBW8	DEC+/-	0	101
14		"Registros".Claves[4]	%DB1.DBW10	DEC+/-	0	102
15		"Registros".Claves[5]	%DB1.DBW12	DEC+/-	0	103
16		"Registros".Claves[6]	%DB1.DBW14	DEC+/-	0	104
17		"Registros".Claves[7]	%DB1.DBW16	DEC+/-	105	105
18		"Registros".Claves[8]	%DB1.DBW18	DEC+/-	106	106
19		"Registros".Claves[9]	%DB1.DBW20	DEC+/-	107	107
20		"Registros".Claves[10]	%DB1.DBW22	DEC+/-	108	108
21		"Registros".Claves[11]	%DB1.DBW24	DEC+/-	109	109
22		"Registros".Claves[12]	%DB1.DBW26	DEC+/-	110	110
23		"Registros".Claves[13]	%DB1.DBW28	DEC+/-	241	241
24		"Registros".Claves[14]	%DB1.DBW30	DEC+/-	111	111
25		"Registros".Claves[15]	%DB1.DBW32	DEC+/-	112	112
26		"Registros".Claves[16]	%DB1.DBW34	DEC+/-	113	113
27		"Registros".Claves[17]	%DB1.DBW36	DEC+/-	114	114
28		"Registros".Claves[18]	%DB1.DBW38	DEC+/-	115	115
29		"Registros".Claves[19]	%DB1.DBW40	DEC+/-	116	116
30		"Registros".Claves[20]	%DB1.DBW42	DEC+/-	117	117
31		"Registros".Claves[21]	%DB1.DBW44	DEC+/-	118	118
32		"Registros".Claves[22]	%DB1.DBW46	DEC+/-	119	119
33		"Registros".Claves[23]	%DB1.DBW48	DEC+/-	120	120

Fuente: elaboración propia.

3.3.10. Instrucción CONTINUE

Es una instrucción de salto. Cuando se da esta condición se interrumpe la ejecución de un bucle. Es una instrucción que permite verificar la condición de un bucle en ejecución (FOR, WHILE o REPEAT) para terminar la ejecución de la iteración actual y continuar con la próxima. De esta forma, la instrucción CONTINUE finaliza la ejecución de una iteración de un bucle, pero no la ejecución del bucle en sí. La instrucción CONTINUE salta (no ejecuta) las instrucciones que existan después de ella, en la iteración de un bucle.

En el siguiente ejemplo se presenta la estructura general de la instrucción CONTINUE en un ciclo FOR donde se puede asumir “Corriente” en MW10 como entero, “P” en MD14, “R” en MD18 y “Corriente_Real” en MD22 como reales. “Tem-

peratura" es un Array en una base de datos (Datos) de 20 elementos como real. El resultado del proceso será que el Array de temperaturas se actualizará para los valores de corriente mayores a 7, de acuerdo con la fórmula especificada, donde la instrucción SQR permite elevar al cuadrado un número. Si la condición IF es menor que 7, el programa continuará en la siguiente iteración dentro del ciclo FOR.

```
FOR "Corriente" := 0 TO 20 DO
    IF ("Corriente" < 7) THEN
        CONTINUE;
    END_IF;
    "Corriente_Real" := INT_TO_REAL("Corriente");
    "Datos".Temperatura["Corriente"] := "P" * SQR("Corriente_Real")*"R";
END_FOR;
```

Problema 3.17. Se tiene una base de datos de diez registros. Cada vez que se hace una solicitud desde una entrada externa, se deben limpiar (dejar en 0) los primeros cinco.

Se inicia creando una base de datos, declarando un Array de diez registros (0 a 9) en formato real como se muestra en la Tabla 19; se adiciona una variable llamada "Contador", en formato entero, que se utilizará como índice.

Tabla 19.

Base de datos creada para el problema 3.17

Nombre	Tipo de datos
Static	
Registros	Array[0..9] of Real
Registros[0]	Real
Registros[1]	Real
Registros[2]	Real
Registros[3]	Real
Registros[4]	Real
Registros[5]	Real
Registros[6]	Real
Registros[7]	Real
Registros[8]	Real
Registros[9]	Real
Contador	Int

Fuente: elaboración propia.

Las demás variables por utilizar se muestran en la Tabla 20, declaradas en variables PLC. La variable "Limpiar" corresponde a una entrada digital que indicará el momento en que se debe realizar el proceso de limpieza.

Tabla 20.

Variables PLC creadas para el problema 3.17

Variable	Tipo	Dirección
Limpiar	Bool	%I0.0
Detector_Flanco	Bool	%M2.0
Auxiliar_Flanco	Bool	%M2.1

Fuente: elaboración propia.

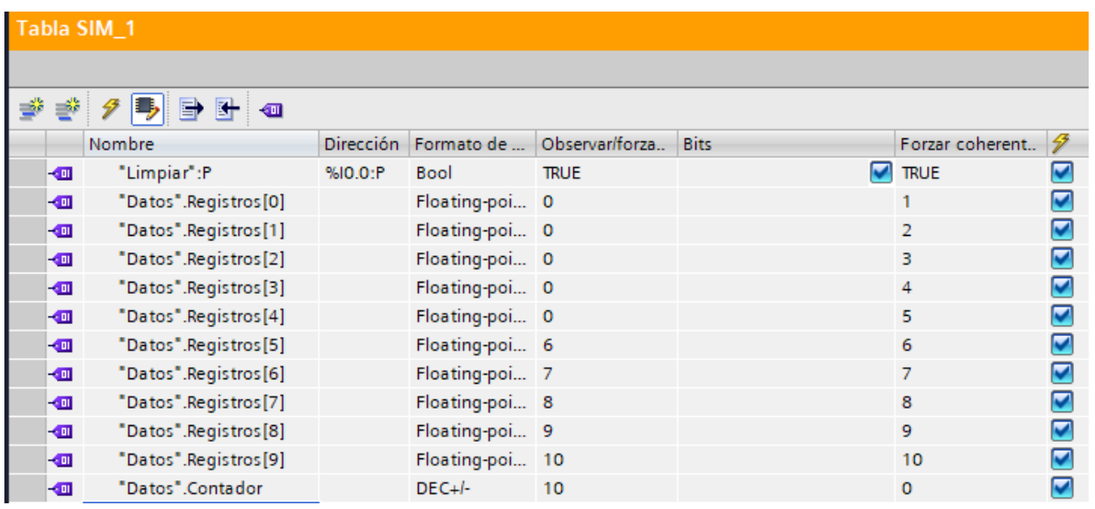
El programa en SCL que se propone para la solución del problema 3.19 se muestra a continuación. Cada vez que se llega a la instrucción "Continue", se pasa a ejecutar el próximo ciclo FOR. De esta forma, mientras "Contador" sea menor que 5 se procederá a la limpieza del registro correspondiente; los demás se dejarán como están.

```
"Detector_Flanco" := "Limpiar" AND NOT "Auxiliar_Flanco"; // Detecta flanco positivo
"Auxiliar_Flanco" := "Limpiar";
IF "Detector_Flanco" THEN // Al detectar flanco se inicia la limpieza
    "Datos".Contador := 0;
    FOR "Datos".Contador := 0 TO 9 DO
        ;
        IF "Datos".Contador >= 5 THEN
            CONTINUE;// Se deja el registro igual a partir Contador=5
        END_IF;
        "Datos".Registros["Datos".Contador] := 0; // Se limpia el registro actual
    END_FOR;
END_IF;
```

El resultado de la simulación para un PLC S7-1200 se muestra en la Figura 43. Se observa que luego de activar "Limpiar" los primeros cinco registros quedan con un valor de 0. En los demás quedan los valores precargados.

Figura 43.

Simulación del algoritmo para el problema 3.17



Fuente: elaboración propia.

Problema 3.18. Se tiene una base de datos de diez registros y se debe, cada vez que se activa una entrada, sacar el promedio de estos registros. Se deben considerar solo los que son diferentes a 0.

En la Figura 44 se muestra la base de datos creada. Se compone de un Array de diez registros en formato Real donde se depositarán los valores que se quieren promediar, una variable "Contador" encargada de manejar los índices de los registros, y una variable "Promedio" en la que se depositará el promedio de los valores. En "Denominador" se llevará el conteo de los registros que son diferentes a 0 y en "Numerador se lleva la sumatoria del contenido de todos los registros.

Figura 44.

Bloque de datos creado para almacenar las variables del problema 3.18

Fuente: elaboración propia.

En una tabla de variables PLC se declaran variables adicionales (Tabla 21).

Tabla 21.

Variables adicionales para el problema 3.18

Variable	Tipo	Dirección	Comentario
Promediar	Bool	%I0.0	Entrada que activa el cálculo del promedio
Detector_Flanco	Bool	%M2.0	Marca para detector de flanco de subida
Auxiliar_Flanco	Bool	%M2.1	Marca para detector de flanco de subida

Fuente: elaboración propia.

El programa escrito en SCL se muestra a continuación. Se acepta la rutina al detectar el flanco de subida de la variable “Promediar” (I0.0). Se inicializan las variables “Denominador”, “Promedio” y “Numerador” ubicadas en la base de datos llamada “Datos”. Luego se entra en un ciclo FOR que se repite 10 veces; sin embargo, la función “Continue” facilita que solo se realicen cálculos cuando el contenido del registro correspondiente es diferente de 0. Las operaciones consisten en llevar la cuenta de registros con contenido diferente de 0 “Denominador”, realizar la sumatoria de estos mismos registros “Numerador” e ir calculando el promedio de los registros indagados.

```
“Detector_Flanco” := “Promediar” AND NOT “Auxiliar_Flanco”;
“Auxiliar_Flanco” := “Promediar”; // Al detectar flanco positivo
IF “Detector_Flanco” THEN // se entra en el ciclo para promediar
       “Datos”.Denominador := 0; // Se inicializan variables en 0
    “Datos”.Promedio := 0;
    “Datos”.Numerador := 0;

    FOR “Datos”.Contador:=0 TO 9 DO
        IF “Datos”.Registros[“Datos”.Contador] = 0.0 THEN
            CONTINUE; // Se descartan los registros que contienen 0.0
        END_IF;
        “Datos”.Denominador := “Datos”.Denominador + 1; // Llevar la
cuenta de registros con contenido diferente de 0
        “Datos”.Numerador:= “Datos”.Numerador + “Datos”.Registros[“Da-
tos”.Contador]; // Sumatoria de contenido de los registros
        “Datos”.Promedio := “Datos”.Numerador / “Datos”.Denominador; //
Se va calculando el promedio
    END_FOR;
END_IF;
```

En la Figura 45 se muestra el resultado de la simulación donde se puede apreciar que dos de los registros contienen el número 0, por lo que, luego de la simulación, la variable “Denominador” contabiliza 8, la variable numerador queda en 212 y “Promedio” en 26,5.

Figura 45.

Simulación correspondiente al problema 3.18

	Nombre	Dirección	Formato de ...	Observar/forza..	Bits	Forzar coherent..	
	"Promediar":P	%I0.0:P	Bool	TRUE	☑	TRUE	☑
	"Datos".Registros[0]		Floating-poi...	20		20	☑
	"Datos".Registros[1]		Floating-poi...	30		30	☑
	"Datos".Registros[2]		Floating-poi...	45		45	☑
	"Datos".Registros[3]		Floating-poi...	32		32	☑
	"Datos".Registros[4]		Floating-poi...	4		4	☑
	"Datos".Registros[5]		Floating-poi...	0		0	☑
	"Datos".Registros[6]		Floating-poi...	12		12	☑
	"Datos".Registros[7]		Floating-poi...	40		40	☑
	"Datos".Registros[8]		Floating-poi...	29		29	☑
	"Datos".Registros[9]		Floating-poi...	0		0	☑
	"Datos".Contador		DEC+/-	10		0	☐
	"Datos".Promedio		Floating-poi...	26,5		0	☐
	"Datos".Denominador		Floating-poi...	8		0	☐
	"Datos".Numerador		Floating-poi...	212		0	☐

Fuente: elaboración propia.

Problema 3.19. Se tiene una base de datos de diez registros que deben tener valores entre 50 y 150. Cuando el número ingresado en cada registro quede por fuera de este rango se debe reemplazar por el mínimo permitido o por el máximo permitido, según sea la situación.

Se creó la base de datos que se muestra en la Figura 46, dejando un Array para los diez registros y una variable llamada “Contador” para controlar la ejecución de un LOOP FOR.

Figura 46.

Base de datos para el problema 3.

Datos

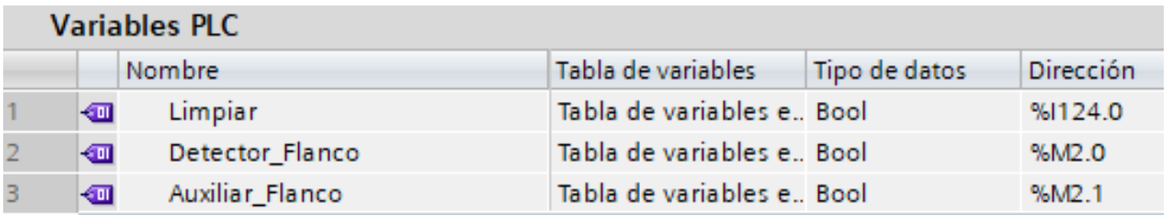
	Nombre	Tipo de datos	Offset
1	Static		
2	Registros	Array[0..9] of Real	0.0
3	Registros[0]	Real	0.0
4	Registros[1]	Real	4.0
5	Registros[2]	Real	8.0
6	Registros[3]	Real	12.0
7	Registros[4]	Real	16.0
8	Registros[5]	Real	20.0
9	Registros[6]	Real	24.0
10	Registros[7]	Real	28.0
11	Registros[8]	Real	32.0
12	Registros[9]	Real	36.0
13	Contador	Int	40.0

Fuente: elaboración propia.

Las variables adicionales se muestran en la Figura 47 donde la variable “Limpiar”, I124.0, es la encargada de corregir la tabla.

Figura 47.

Variables adicionales para el problema 3.19



Variables PLC

	Nombre	Tabla de variables	Tipo de datos	Dirección
1	Limpiar	Tabla de variables e..	Bool	%I124.0
2	Detector_Flanco	Tabla de variables e..	Bool	%M2.0
3	Auxiliar_Flanco	Tabla de variables e..	Bool	%M2.1

Fuente: elaboración propia.

Se propone el algoritmo en SCL que se muestra a continuación. Se utiliza un ciclo FOR donde, en cada iteración, se determina si el número correspondiente al apuntador dado por “Contador” se encuentra dentro del rango 45 a 150, caso en el cual se continuará con el siguiente registro; de lo contrario, se procede a hacer el ajuste al extremo del rango que corresponde.

```
“Detector_Flanco” := “Limpiar” AND NOT “Auxiliar_Flanco”;
“Auxiliar_Flanco” := “Limpiar”;
IF “Detector_Flanco” THEN

    FOR “Datos”.Contador := 0 TO 9 DO
        IF “Datos”.Registros[“Datos”.Contador] > 50.0 AND “Datos”.Regis-
tros[“Datos”.Contador] < 150 THEN
            CONTINUE;
        END_IF;
        IF “Datos”.Registros[“Datos”.Contador] < 50.0 THEN
            “Datos”.Registros[“Datos”.Contador] := 50.0;
        END_IF;
        IF “Datos”.Registros[“Datos”.Contador] > 50.0 THEN
            “Datos”.Registros[“Datos”.Contador] := 150.0;
        END_IF;

    END_FOR;
END_IF;
```

En la Figura 48 se muestra el resultado de la simulación para el problema 3.19. Del registro 0 al 9 se llevan diferentes valores; al registro 1 se le asignó un valor de 170, pero el programa lo ajustó a 150; lo mismo pasa con los registros 5 y 8. En los registros 2 y 9 se introdujeron valores menores a 50 y el programa los ajusta a 45.

Figura 48.

Simulación para comprobar solución del del problema 3.19

Nombre	Dirección	Formato visualiza..	Valor de observac..	Valor de forz..
"Datos".Registros[0]	%DB1.DBD0	Número en coma...	67.0	67.0
"Datos".Registros[1]	%DB1.DBD4	Número en coma...	150.0	170.0
"Datos".Registros[2]	%DB1.DBD8	Número en coma...	50.0	45.0
"Datos".Registros[3]	%DB1.DBD12	Número en coma...	128.0	128.0
"Datos".Registros[4]	%DB1.DBD16	Número en coma...	143.0	143.0
"Datos".Registros[5]	%DB1.DBD20	Número en coma...	150.0	167.0
"Datos".Registros[6]	%DB1.DBD24	Número en coma...	87.0	87.0
"Datos".Registros[7]	%DB1.DBD28	Número en coma...	94.0	94.0
"Datos".Registros[8]	%DB1.DBD32	Número en coma...	150.0	190.0
"Datos".Registros[9]	%DB1.DBD36	Número en coma...	50.0	12.0
"Datos".Contador	%DB1.DBW40	DEC+/-	10	

Fuente: elaboración propia.

3.3.11. Instrucción EXIT

Es una instrucción de salto que sale de un bucle en forma inmediata y permanente, independientemente de cualquier condición. De esa forma se puede interrumpir en cualquier punto la ejecución de un bucle FOR, WHILE o REPEAT sin considerar las condiciones que se tengan, continuando el proceso después del bucle que se está ejecutando.

En el siguiente ejemplo se presenta la estructura general de la Instrucción EXIT en un ciclo FOR donde se puede asumir: “Corriente” en MW10 como entero, “P” en MD14, “R” en MD18 y “Corriente_Real” en MD22 como reales. “Temperatura” es un Array en una base de datos (Datos) de 20 elementos como real. El resultado del proceso será que el Array de temperaturas se actualizará para los valores de corriente mayores o iguales a 7, de acuerdo con la fórmula especificada, donde la instrucción SQR permite elevar al cuadrado un número. Cuando la condición IF es menor que 7 las iteraciones del bucle FOR no se vuelven a ejecutar, siguiendo el programa después de END_FOR.

```
FOR “Corriente” := 20 TO 0 BY (-1) DO
    IF (“Corriente” < 7) THEN
        EXIT;
    END_IF;
    “Corriente_Real” := INT_TO_REAL(“Corriente”);
    “Datos”.Temperatura[“Corriente”] := “P” * SQR(“Corriente_Real”)*”R”;
END_FOR;
```

Problema 3.20. En una base de datos de 20 registros se deposita el valor de códigos de productos fabricados durante un día. Se quiere, eventualmente, identificar si un producto en especial ha sido fabricado y que se devuelva la primera posición de memoria encontrada dentro la base de datos que coincida con el código del producto.

En la Figura 49 se muestra la base de datos del problema 3.20. En el Array "Producto", de 20 registros en formato entero, se depositan los valores de códigos de los productos fabricados que serían válidos. Se crearon otras tres variables: "Contador", "Registro" y Código". "Contador" contendrá el índice de la posición de memoria que se está consultando del Array; en "Registro" se guardará el índice de la posición de la base de datos que coincide con el "Código", y este último será el valor del número que se quiere buscar en los elementos del Array "Producto".

Figura 49.

Base de datos creada para el problema 3.20

Datos

	Nombre	Tipo de datos	Offset
1	Static		
2	Producto	Array[0..19] of Int	0.0
3	Contador	Int	40.0
4	Registro	Int	42.0
5	Código	Int	44.0

Fuente: elaboración propia.

Las variables adicionales para el problema 3.20 se presentan en la Figura 50 donde "Limpiar" corresponde a la entrada digital I124.0 que iniciará la rutina búsqueda de la posición de memoria que coincida en el "Código".

Figura 50.

Variables adicionales para el problema 3.20

Tabla de variables estándar

	Nombre	Tipo de datos	Dirección
1	Limpiar	Bool	%I124.0
2	Detector_Flanco	Bool	%M2.0
3	Auxiliar_Flanco	Bool	%M2.1

Fuente: elaboración propia.

El algoritmo sugerido en SCL es el siguiente. Cuando se detecta el Flanco positivo de “Limpiar” se entra en un ciclo WHILE que se ejecutará mientras “Contador” sea menor o igual a 19, para lo cual previamente se debe iniciar en 0. En cada iteración de WHILE se verifica si el número del Array de “Producto”, direccionado por el índice que corresponde a “Contador”, coincide con el número que es depositado en “Código”, caso en el cual se sale definitivamente del ciclo WHILE al ejecutar la instrucción EXIT; de lo contrario, se sigue con la próxima iteración luego de incrementar el contador en 1. Al salir del ciclo WHILE, la variable “Registro” se cargará con el valor del contador que coincidió con el código buscado.

```
“Detector_Flanco” := “Limpiar” AND NOT “Auxiliar_Flanco”; // Cada vez que se presiona
“Auxiliar_Flanco” := “Limpiar”; // Limpiar, se inicia la búsqueda de un
IF “Detector_Flanco” THEN // código coincidente
    “Datos”.Contador := 0;
    WHILE “Datos”.Contador <= 19 DO
        IF “Datos”.Producto[“Datos”.Contador] = “Datos”.Código THEN
            EXIT;
        END_IF;
        “Datos”.Contador := “Datos”.Contador + 1;
    END_WHILE;
    “Datos”.Registro := “Datos”.Contador;
END_IF;
```

El resultado de una simulación se muestra en la Figura 51. El número buscado es 29, que fue localizado en la posición del Array “Producto” direccionada por “Registro”; en este caso, con el índice 9.

Figura 51.

Resultado de simulación del problema 3.20

Nombre	Dirección	Formato visualiza..	Valor de observac..	Valor d
"Datos".Producto[0]	%DB1.DBW0	DEC+/-	6	6
"Datos".Producto[1]	%DB1.DBW2	DEC+/-	7	7
"Datos".Producto[2]	%DB1.DBW4	DEC+/-	3	3
"Datos".Producto[3]	%DB1.DBW6	DEC+/-	1	1
"Datos".Producto[4]	%DB1.DBW8	DEC+/-	45	45
"Datos".Producto[5]	%DB1.DBW10	DEC+/-	7	7
"Datos".Producto[6]	%DB1.DBW12	DEC+/-	1	1
"Datos".Producto[7]	%DB1.DBW14	DEC+/-	2	2
"Datos".Producto[8]	%DB1.DBW16	DEC+/-	5	5
"Datos".Producto[9]	%DB1.DBW18	DEC+/-	29	29
"Datos".Producto[10]	%DB1.DBW20	DEC+/-	25	25
"Datos".Producto[11]	%DB1.DBW22	DEC+/-	23	23
"Datos".Producto[12]	%DB1.DBW24	DEC+/-	17	17
"Datos".Producto[13]	%DB1.DBW26	DEC+/-	15	15
"Datos".Producto[14]	%DB1.DBW28	DEC+/-	12	12
"Datos".Producto[15]	%DB1.DBW30	DEC+/-	99	99
"Datos".Producto[16]	%DB1.DBW32	DEC+/-	88	88
"Datos".Producto[17]	%DB1.DBW34	DEC+/-	77	77
"Datos".Producto[18]	%DB1.DBW36	DEC+/-	66	66
"Datos".Producto[19]	%DB1.DBW38	DEC+/-	55	55
"Datos".Contador	%DB1.DBW40	DEC+/-	9	
"Datos".Registro	%DB1.DBW42	DEC+/-	9	0
"Datos".Código	%DB1.DBW44	DEC+/-	29	29

Fuente: elaboración propia.

3.3.12. Instrucción GOTO

Es una instrucción de salto que permite continuar el procesamiento del programa en la posición indicada por una etiqueta. Se recomienda no utilizarla porque rompe con la estructura de un programa escrito en SCL. La etiqueta debe estar en el mismo bloque (Siemens, 2005).

La estructura general de la instrucción GOTO se muestra en el siguiente ejemplo. Dependiendo del resultado del condicional, el programa puede seguir su ejecución en Meta1, Meta2, Meta3 o Meta4 para hacer que se prenda una alarma de alta o una alarma de baja, según el resultado de la comparación de la variable “Temperatura”.

```
IF “Temperatura” > 120 THEN // Temperatura en MD10 REAL
    GOTO Meta1;
ELSIF “Temperatura” < 40.0 THEN
    GOTO Meta2;
END_IF;
GOTO Meta3;
```

```
Meta1:
“Alarma_Alta” := 1; //Alarma_Alta en Q0.0
GOTO Meta4;
Meta2:
“Alarma_Baja” := 1; //Alarma_Baja en Q0.0
GOTO Meta4;
Meta3:
“Alarma_Alta” := 0;
“Alarma_Baja” := 0;
Meta4:;
```

3.3.13. Instrucción RETURN

Es una instrucción de salto que permite salir de un bloque, ya sea un OB, un FB o un FC para regresar al bloque que invocó.

Problema 3.21. Una receta para la fabricación de un producto se da por la mezcla de tres productos diferentes. Se pide elaborar un programa para que se pueda determinar la cantidad de litros de cada producto de acuerdo con una de cuatro recetas seleccionadas.

Las variables por usar se muestran en la Tabla 22.

Tabla 22.

Variables para utilizar en el problema 3.21

Nombre	Tipo	Dirección
Producto A	Real	%MD10
Producto B	Real	%MD14
Producto C	Real	%MD18
Selector Receta	Int	%MW22

Fuente: elaboración propia.

El algoritmo en SCL puede ser: con la instrucción CASE se evalúa cuál receta ejecutar de acuerdo con lo indicado por la variable “Selector” 1, 2 o 3; cada instrucción GOTO desvía el programa al sitio indicado por las etiquetas Receta1, Receta2, o Receta3; de no incluirse el 1, 2 o 3 en la instrucción CASE, entonces, otra instrucción GOTO desvía el programa a la posición marcada por la etiqueta Receta4. Según la etiqueta a la que se salte, se da una composición diferente para cada producto. Observe que es necesario realizar un nuevo salto cada vez que se tiene lista la composición de cada receta por medio de la instrucción GOTO Termina, que lleva hasta la instrucción RETURN que hace que el bloque regrese al que lo invocó (un FC, un OB o un FB). En lugar de utilizar GOTO Termina se pudo haber utilizado directamente la instrucción RETURN para salir del programa.

```
CASE “Selector Receta” OF
    1:  GOTO Receta1;
    2:  GOTO Receta2;
    3:  GOTO Receta3;

    ELSE
        GOTO Receta4;// En caso de que no se seleccione 1, 2, 3 o 4
END_CASE;

Receta1:
“Producto A” := 30;
“Producto B” := 50;
“Producto C” := 20;
GOTO Termina;
Receta2:
“Producto A” := 20;
“Producto B” := 40;
“Producto C” := 40;
GOTO Termina;
Receta3:
“Producto A” := 70;
“Producto B” := 10;
“Producto C” := 20;
GOTO Termina;
Receta4:
“Producto A” := 40;
“Producto B” := 40;
“Producto C” := 20;
Termina: RETURN;
```

04. TEMPORIZADORES

4. Temporizadores

Los temporizadores, a la vez que los contadores, sirven para ayudar a controlar la ejecución del programa de acuerdo con el estado actual de un temporizador o un contador. Desde el punto de vista lógico y secuencial se pueden activar procesos, salidas o eventos cuando el temporizador o contador llega a cierto valor. Por ejemplo, se puede mantener encendida una resistencia de calefacción durante un lapso de 20 minutos marcados por un temporizador, pero es posible que interese encender un sistema de agitación a partir de tan solo 5 minutos de calentamiento. De otra forma, es posible ejecutar operaciones que dependan del tiempo que lleva un determinado proceso. Por ejemplo, se puede querer abrir una válvula proporcionalmente a medida que transcurre el tiempo de ejecución del proceso. Los temporizadores y contadores de los PLC ofrecen estas posibilidades.

Con SCL se pueden utilizar los temporizadores para producir retardos, generar trenes de pulsos, temporizar o medir el tiempo de un proceso. Hay varias funciones normalizadas de temporización que se explicarán a continuación.

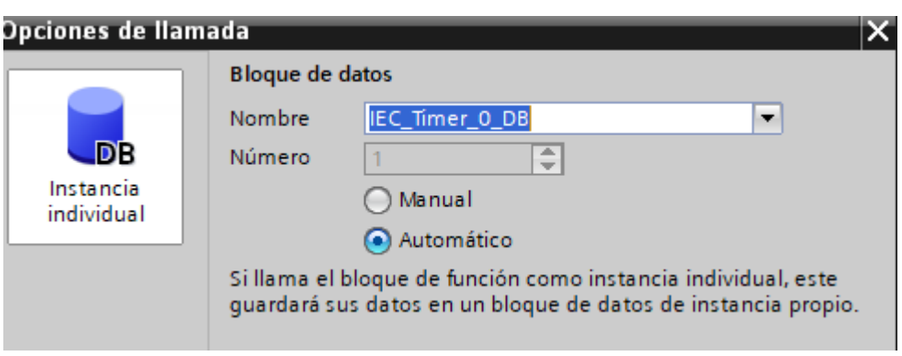
4.1. Temporizador como impulso (TP)

Permite activar una salida durante un tiempo programado al final del cual apagará nuevamente su salida; solo requiere de un impulso (flanco positivo) para activarlo. Es reconocido como un temporizador al trabajo debido a que empieza a temporizar una vez recibe su señal de activación.

Al invocar este temporizador en TIA PORTAL, desde el área de instrucciones básicas, se debe asignar una base de datos donde se almacenará información relevante para el temporizador (Figura 52).

Figura 52.

Generación de base de datos para almacenar información de un temporizador

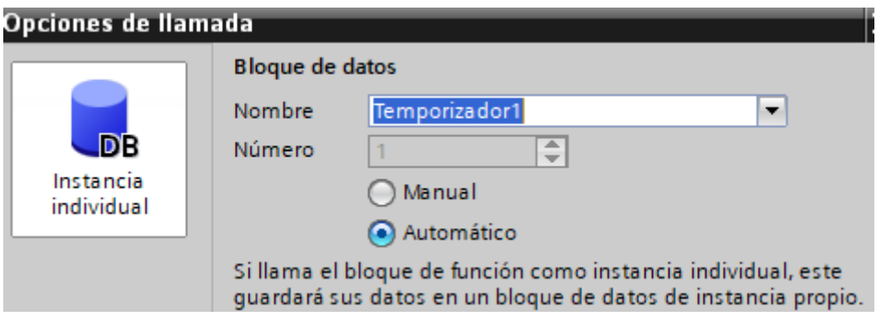


Fuente: elaboración propia.

A la base de datos asignada al temporizador creado se le puede dar un nombre que los distinga; en la Figura 53 donde se le da el nombre de Temporizador1.

Figura 53.

Asignación de nombre para la base de datos del temporizador

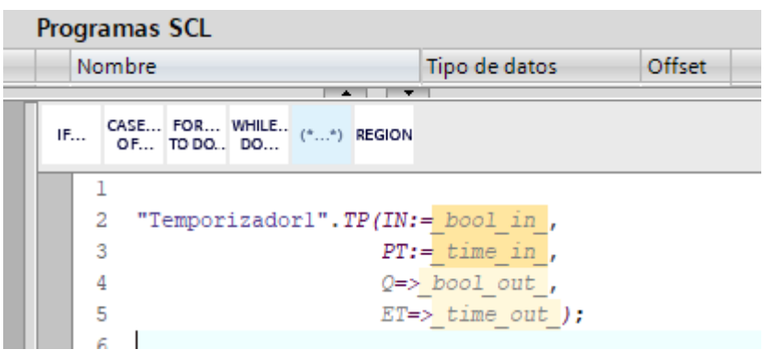


Fuente: elaboración propia.

La estructura del programa en SCL para el temporizador al impulso TP se muestra en la Figura 54 donde "Temporizador1" corresponde a la base de datos destinada para el temporizador; TP corresponde al tipo de temporizador como impulso; bool_in es la señal digital que pondrá en marcha el temporizador; PT es el tiempo de temporizador en formato Time; bool_out es la salida que se activa de acuerdo con el estado del temporizador, y time_out es un área de memoria en formato Time donde se deposita el valor de la cuenta ascendente del temporizador.

Figura 54.

Apariencia en TIA PORTAL del temporizador TP en SCL



Fuente: elaboración propia.

El programa en SCL para el temporizador TP es el siguiente:

```
"Temporizador1".TP(IN:=_bool_in_,
                   PT:=_time_in_,
                   Q=>_bool_out_,
                   ET=>_time_out_);
```

"Temporizador1".TP: permite invocar el Temporizador como impulso (TP) cuyos valores y controles se almacenan en la base de datos llamada "Temporizador1". Lo que va entre paréntesis permite controlar el temporizador.

IN:=_bool_in_: instrucción para activar el temporizador, bool_in puede ser cualquier entrada o marca booleana, por ejemplo, I124.0 o M10.5.

PT:=_time_in_: valor de la temporización en formato TIME.

Q=>_bool_out_: en la base de datos se activa esta posición de memoria cuando el temporizador inicia su cuenta; puede ser invocada en varias partes del programa haciendo referencia a esta memoria de la base de datos.

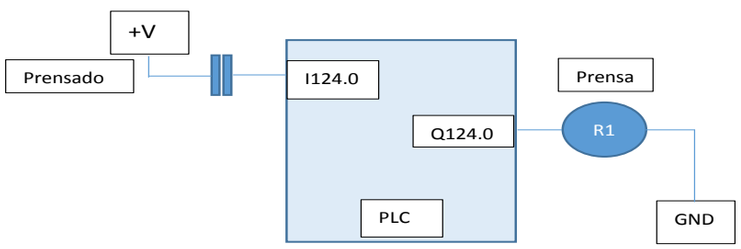
ET=>_time_out_: área de memoria donde se guarda el tiempo actual de temporización en formato Time.

Problema 4.1. La activación de una prensa hidráulica se debe dar por 5 segundos siempre que se presiona un pulsador de prensado.

Las conexiones del PLC se muestran en la Figura 55.

Figura 55.

Diagrama de conexiones del PLC para el problema 4.1



Fuente: elaboración propia.

Los símbolos utilizados para el problema 4.1 se presentan en la Tabla 23.

Tabla 23.

Símbolos para el problema 4.1

Nombre	Tipo	Dirección
Prensado	Bool	I124.0
Prensa	Bool	Q124.0

Fuente: elaboración propia.

El programa en SCL es el siguiente, luego de invocar este temporizador en TIA PORTAL desde el área de instrucciones básicas.

```
"Temporizador1".TP(IN:="Prensado",
                   PT:=T#5s, // 5 segundos en formato time
                   Q=>"Prensa");
```

Obsérvese que en este caso la instrucción ET=>_time_out_ fue suprimida, puesto que no es requerida.

Para verificar el funcionamiento simplemente active el pulsador “Prensado”; entonces, la salida “Prensa” se activará por 5 segundos.

Conviene ahora aclarar el significado del formato TIME (IEC), el cual se debe interpretar en milisegundos, pero puede ser expresado en términos de días, horas, minutos, segundos y milisegundos. El tiempo máximo aceptable es de T#-10d_20h_30m_20s_630ms o TIME#10d_20h_30m_20s_630ms o simplemente 2147483647 milisegundos en formato decimal que requiere de un espacio de memoria de 32 bits (doble palabra) para almacenarlo. Esto sugiere que, en el ejemplo anterior, el tiempo asignado al temporizador de 5 segundos puede ser manipulado desde algún área de memoria de 32 bits, por ejemplo, MD10. Para esto se crea la nueva variable Tiempo MD10 en formato Time (Tabla 24).

Tabla 24.

Tabla de variables para alternativa del problema 4.1

Nombre	Tipo	Dirección
Prensado	Bool	I124.0
Prensa	Bool	Q124.0
Tiempo	Time	MD10

Fuente: elaboración propia.

Se ajusta el programa SCL en la instrucción correspondiente, luego de invocar este temporizador en TIA PORTAL desde el área de instrucciones básicas.

```
“Temporizador1”.TP(IN:=”Prensado”,
                   PT:=”Tiempo”,
                   Q=>”Prensa”);
```

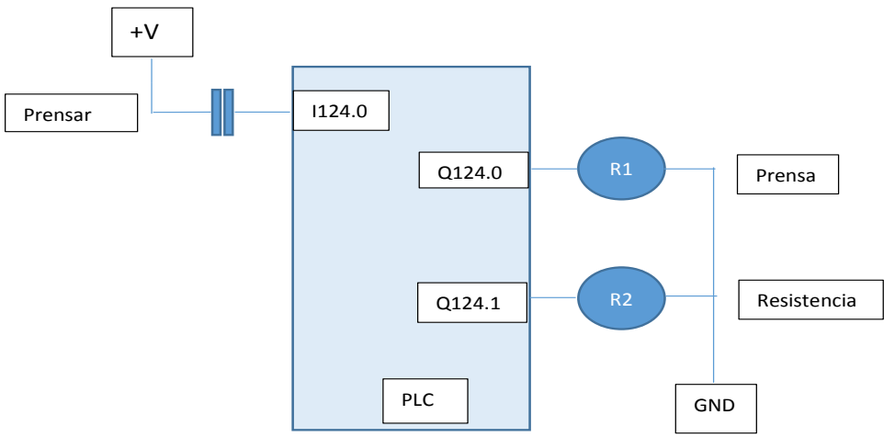
Al ejecutar el programa se observa que el tiempo del temporizador se puede alterar desde MD10, entrando el retardo en milisegundos; si se quieren 10 segundos se ingresa el número 10 000 en MD10.

Problema 4.2. Luego de pulsar un botón de prensado, se activa una salida que energiza una prensa hidráulica durante un tiempo (t) estipulado en una marca memoria del PLC en segundos. Una resistencia eléctrica para calentamiento se debe activar entre t/4 y 3t/4.

En la Figura 56 se muestran las conexiones del PLC.

Figura 56.

Diagrama de conexiones del PLC para el problema 4.2



Fuente: elaboración propia.

Las variables para utilizar en el problema 4.2 se muestran en la Tabla 25.

Tabla 25.

Variables del problema 4.2

Nombre	Tipo	Dirección
Prensado	Bool	I124.0
Prensa	Bool	Q124.0
Resistencia	Bool	Q124.1
Tiempo	Time	MD10
Ajuste	Time	MD14
T_Avance	Time	MD18

Fuente: elaboración propia.

El programa en SCL propuesto para la solución del problema se muestra más adelante. Es posible que se piense en la utilización de dos temporizadores para resolver este problema; sin embargo, aunque se requiere activar dos salidas diferentes que dependen de distintos tiempos de programación, fue posible el uso de un solo temporizador. Para esto es necesario incluir la instrucción ET=>"T_Avance" que permite almacenar el tiempo transcurrido (ET) en la variable "T_Avance" para realizar operaciones de comparación y tomar decisiones. La variable "Tiempo" se obtiene de multiplicar "Ajuste" por mil para llevar el tiempo a milisegundos. Al invocar el temporizador TP, cuya información se guarda en la base de datos "Temporizador1", se queda a la espera del flanco positivo de "Prensado", con lo que

inmediatamente se activa la prensa durante el tiempo guardado en “Tiempo”. Mediante un condicional IF se evalúa si el tiempo transcurrido en el temporizador ET, que es almacenado en “T_Avance”, se encuentra en el rango comprendido entre t/4 y 3t/4, circunstancia en la cual se activa la “Resistencia”.

```
“Tiempo” := “Ajuste” * 1000; // hallar el tiempo en milisegundos
“Temporizador1”.TP(IN:=”Prensado”,
                   PT:=”Tiempo”,
                   Q=>”Prensa”,
                   ET=>”T_Avance”);
IF “T_Avance” >(“Tiempo”/4) AND “T_Avance” <(3*”Tiempo”/4) THEN
    “Resistencia” := 1;
ELSE
    “Resistencia” := 0;

END_IF;
```

4.2. Temporizador con retardo a la conexión (TON)

Este temporizador permite generar un retardo antes de activar su salida, la cual se mantendrá en alto mientras la señal de activación esté presente; también es reconocido como temporizador al trabajo.

Al invocar este temporizador TON en TIA PORTAL, desde el área de instrucciones básicas, aparece la siguiente estructura SCL:

```
“IEC_Timer_0_DB”.TON(IN:=_bool_in_,
                     PT:=_time_in_,
                     Q=>_bool_out_,
                     ET=>_time_out_);
```

“IEC_Timer_0_DB” es el nombre de la base de datos para el temporizador.

TON es el tipo de temporizador invocado, con retardo a la conexión.

_bool_in_ es una señal digital que va a la entrada IN del temporizador para activarlo.

_time_in_ es donde se deposita el tiempo a temporizar en formato Time y que es cargado en PT.

_bool_out_ corresponde a la salida digital que se quiere activar una vez se llegue al tiempo de temporización; es tomado de la señal Q del temporizador.

_time_out_ es una señal que se toma de ET y va a indicar el tiempo transcurrido en ms.

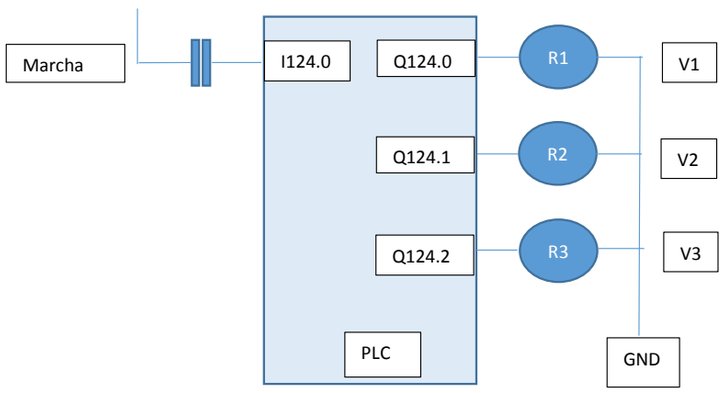
Problema 4.3. Se dispone de tres electroválvulas que permiten el paso de ingredientes diferentes a un tanque donde se realiza una mezcla de los productos. El

ingreso de cada ingrediente al tanque se debe realizar en forma temporizada y secuencial, luego de presionar un botón de marcha.

La Figura 57 presenta las conexiones del PLC, compuesto por una entrada digital y tres salidas también digitales.

Figura 57.

Diagrama de conexiones del PLC para el problema 4.3



Fuente: elaboración propia.

Las variables para emplear en el problema 4.3 se muestran en la Tabla 26. Los tiempos programados van en "Ajuste1", "Ajuste2", "Ajuste3", en formato Time.

Tabla 26.

Variables empleadas para el problema 4.3

Nombre	Tipo	Dirección
Marcha	Bool	%I124.0
V1	Bool	%Q124.0
V2	Bool	%Q124.1
V3	Bool	%Q124.2
Ajuste1	Time	%MD10
Ajuste2	Time	%MD14
Ajuste3	Time	%MD18
Ciclo_ON	Bool	%M2.0
Listo_T1	Bool	%M2.1
Listo_T2	Bool	%M2.2
Listo_T3	Bool	%M2.3

Fuente: elaboración propia.

El programa en SCL se muestra a continuación. El primer condicional IF se propone para que el ciclo solamente se realice una vez, luego de pulsar “Marcha” (no se debe dejar presionado); bajo esta condición “Ciclo_ON” se pone a 1. Con esta condición cumplida, el “Temporizador_1” se activa. Al finalizar el tiempo de temporización guardado en “Ajuste1”, se activa “Listo_T1”; esta última señal pone en funcionamiento el “Temporizador_2” que finalmente activará “Listo_T2”. Lo mismo sucederá con el “Temporizador_3” para activar finalmente “Listo_T3”; de esa forma se tiene la secuencia retardo1, “Listo_T1”, retardo2, “Listo_T2”, retardo3, “Listo_T3”. La instrucción “V1” := (“Ciclo_ON” AND NOT “Listo_T1”) hace que se active la válvula 1 solo mientras que el temporizador 1 está temporizando; las válvulas 2 y 3 se activan secuencialmente bajo el mismo concepto. Observe que solo hay una válvula abierta al tiempo. Finalmente, con “Listo_T3” activado se desactiva “Ciclo_ON” y se debe pulsar nuevamente “Marcha” para una nueva secuencia. Alternativamente, se pudo colocar un detector de flanco para “Marcha”.

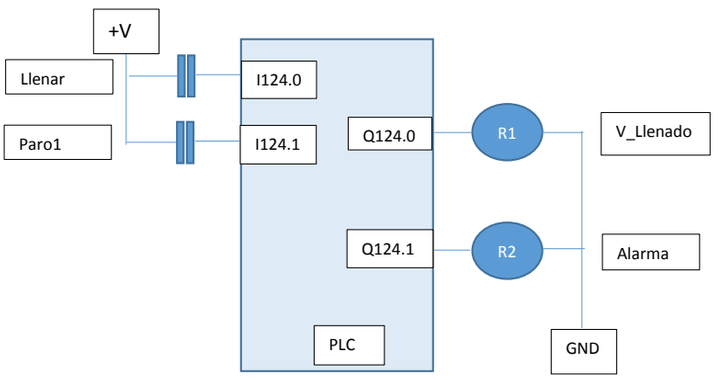
```
IF “Marcha” THEN //Identifica cuando se da marcha
    “Ciclo_ON” := 1;// Asegúrese de no dejar marcha presionado
END_IF; // Para que no se repita el ciclo al terminar el tercer temporizador
“Temporizador_1”.TON(IN:=”Ciclo_ON”,// El temporizador 1 se activa al dar marcha
                     PT:=”Ajuste1”,
                     Q=>”Listo_T1”);

“Temporizador_2”.TON(IN := “Listo_T1”,// El temporizador 2 inicia cuando
                     PT := “Ajuste2”,// el temporizador 1 finaliza
                     Q => “Listo_T2”);

“Temporizador_3”.TON(IN := “Listo_T2”,// El temporizador 3 inicia cuando
                     PT := “Ajuste3”,// el temporizador 2 finaliza
                     Q => “Listo_T3”);
“V1” := (“Ciclo_ON” AND NOT “Listo_T1”);// solo una válvula energizada
“V2” := (“Listo_T1” AND NOT “Listo_T2”);
“V3” := (“Listo_T2” AND NOT “Listo_T3”);
IF “Listo_T3” THEN  // Para reiniciar el ciclo
    “Ciclo_ON” := 0;
END_IF;
```

Problema 4.4. Se dispone de un recipiente al que se le quiere agregar un líquido por medio de una electroválvula. Al presionar un pulsador de llenado se energiza la bobina de la válvula, la cual se apaga cuando un sensor detecta que el recipiente está lleno. Sin embargo, hay un tiempo máximo de 10 segundos para el llenado; cuando se supere este tiempo se debe apagar la válvula y activar una alarma. No se debe permitir un nuevo llenado hasta que no se presione un pulsador de reconocimiento.

La Figura 58 muestra las conexiones del PLC; se dispuso de un pulsador para llenar, uno de paro, una salida para la electroválvula y otra salida para la alarma.

Figura 58.

Diagrama de conexiones del PLC para el problema 4.4



Fuente: elaboración propia.

En la Tabla 27 se representan las variables para utilizar en la solución del problema 4.4.

Tabla 27.

Variables requeridas para el problema 4.4

Nombre	Tipo	Dirección
Llenar	Bool	%I124.0
V_Llenado	Bool	%Q124.0
Alarma	Bool	%Q124.1
Sobretiempo	Bool	%M2.0
Reinicio	Bool	%I124.1
Sensor	Bool	%I124.2

Fuente: elaboración propia.

Para la solución en SCL se utiliza un primer condicional IF que permite el llenado, siempre y cuando no esté activa la señal de “Sobretiempo”. La segunda condición IF hace que se apague la electroválvula, bien sea porque “Sensor” ya detectó el llenado completo o bien porque se está tardando mucho tiempo en llenar.

La instrucción “Temporizador”.TON(IN:=(“V_Llenado” OR “Sobretiempo”) AND NOT “Reinicio”, tiene la función de activar el temporizador por medio de “V_Llenado”, dejarlo sostenido cuando se alcanza el “Sobretiempo” y ocurrida esta condición solo se puede deshabilitar el temporizador por medio de la señal “Reinicio”. La alarma estará activa siempre que esté activo “Sobretiempo”. Entonces, la clave para que no se dé error consiste en no dejar que el temporizador alcance

su cuenta programada de 10 s. Se deja al lector la tarea de terminar el programa, incluyendo el paro y sin entorpecer la acción de bloqueo, cuando esto sucede.

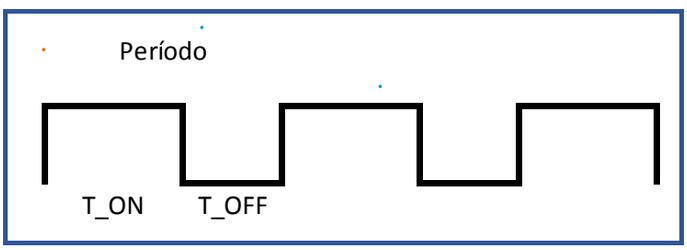
```
IF “Llenar” AND NOT “Sobretiempo” THEN // Cuando Sobretiempo se activa
hay problema
    “V_Llenado” := 1; // que causa un bloqueo para V_Llenado
END_IF;
IF “Sensor” OR “Sobretiempo” THEN
    “V_Llenado” := 0;
END_IF;
“Temporizador”.TON(IN:=(“V_Llenado” OR “Sobretiempo”) AND NOT “Reinicio”,
                   PT:=T#10s,// Sobretiempo deja memoria en el temporizador
                   Q=>”Sobretiempo”);// puede ser desbloqueado con Reinicio
“Alarma” := “Sobretiempo”;
```

Al realizar la simulación se observará que, luego de pulsar llenado, se activa la válvula hasta que el sensor se activa. En caso de que el sensor no detecte el llenado del recipiente, se apaga la válvula y se activa la alarma debido a que se alcanzó el tiempo de temporización. Solo es posible activar la válvula y desbloquear el temporizador cuando se oprime reinicio.

Problema 4.5. Genere una señal cuadrada a la que se le puedan configurar el período y el ciclo de dureza, es decir, la relación entre tiempo alto y tiempo bajo (Figura 59).

Figura 59.

Tren de pulsos esperado para el problema 4.5



Fuente: elaboración propia.

Para la generación del tren de pulsos se proponen dos temporizadores (TON), aunque se pueden dar otras configuraciones. Cuando se activa el pulsador “Generar” (I124.0) empieza a trabajar el primer temporizador; al terminar su cuenta se activa el segundo temporizador (el primero sigue activo); cuando este segundo temporizador termina reinicia el ciclo (ambos temporizadores se apagan).

Las variables por utilizar se muestran en la Tabla 28, a la cual se le agrega su descripción para mejor entendimiento.

Tabla 28.

Variables para el problema 4.5

Nombre	Tipo	Dirección	Descripción
Generar	Bool	%I124.0	Permite que se genere el tren de pulsos
SeñalT1	Bool	%M2.0	Señal de salida del temporizador1
SeñalT2	Bool	%M2.1	Señal de salida del temporizador2
Pulso	Bool	%M2.2	Permite observar el tren de pulsos
Periodo_Seg	Real	%MD10	El usuario ingresa el período deseado
Dureza%	Real	%MD14	El usuario ingresa el % del ciclo de dureza
T_ON	Real	%MD18	Se calcula el tiempo del pulso alto
T_OFF	Real	%MD22	Se calcula el tiempo del pulso bajo
Ajuste_T1	DInt	%MD26	Se convierte a formato entendible para Time
Ajuste_T2	DInt	%MD30	Se convierte a formato entendible para Time

Fuente: elaboración propia.

Para calcular los tiempos de ajuste se usarán operaciones en formato real, recurriendo a las siguientes fórmulas.

Ecuación 7.

$$Tom = Periodo * \frac{Dureza}{100}$$

Ecuación 8.

$$Tof = Periodo - Ton$$

El ajuste del temporizador 1 y el ajuste del temporizador 2 se obtienen convirtiendo Ton y Tof a doble entero y dividiéndolo por mil para que los valores queden en formato Time en milisegundos. En el programa propuesto en SCL, “IEC_Timer_0_DB” es el nombre para el primer temporizador y “IEC_Timer_0_DB_1” para el segundo. “Ajuste_T1” y “Ajuste_T2” corresponden a los tiempos de programación de cada uno de los temporizadores. “Generar” AND NOT “SeñalT2” pone en funcionamiento el temporizador 1, “Generar” AND “SeñalT1” pone en funcionamiento el temporizador 2. La instrucción “Pulso” := NOT “SeñalT1” AND “Generar” provoca la caída del pulso cuando el temporizador 1 está activo.

```
“IEC_Timer_0_DB”.TON(IN:=”Generar” AND NOT “SeñalT2”,
                     PT:=”Ajuste_T1”,
                     Q=>»SeñalT1»);

“IEC_Timer_0_DB_1”.TON(IN:=”Generar” AND “SeñalT1”,
                   PT:=”Ajuste_T2”,
                   Q=>”SeñalT2”);// Esta señal reinicia los dos temporizadores
“Pulso” := NOT “SeñalT1” AND “Generar”;
```

```
"T_ON" := "Periodo_Seg" * "Dureza%" / 100; //Así se calculan el Ton y el Tof
"T_OFF" := "Periodo_Seg" -"T_ON";
"Ajuste_T1" := REAL_TO_DINT("T_ON") * 1000; // Instrucción para convertir
de Real a Dint
"Ajuste_T2" := REAL_TO_DINT("T_OFF") * 1000;
```

Para simular el programa ingrese el tiempo correspondiente al período en MD10 en formato real y el ciclo de dureza (valor entre el 0 % y el 100 %) en MD14. Entonces, el Ton y Toff pueden ser observados en MD18 y MD22, respectivamente, en segundos. Los tiempos de ajuste de los temporizadores en milisegundos se observan en MD26 y MD30. Obsérvese que en este programa se utilizó una nueva instrucción de conversión:

REAL_TO_DINT("T_ON") y REAL_TO_DINT("T_OFF").

Este algoritmo resulta de mucha utilidad en aquellas aplicaciones que requieren de señales PWM (modulación de ancho de pulso) para el control de procesos que no requieran una sensibilidad mayor a la de los milisegundos, que es la que admiten los temporizadores explicados. Además, se deberá tener en cuenta la velocidad de respuesta de las salidas utilizadas en un PLC específico, es decir, si son salidas por relé, transistor o salidas especiales de tipo rápido.

4.3. Temporizador con retardo a la desconexión (TOF)

Este temporizador, cuando es activado, también activa su salida; en el momento que se retira la señal de activación del temporizador, este empezará a ejecutar el tiempo programado, al final del cual apagará su salida correspondiente. Es conocido, comúnmente, como temporizador al reposo porque empieza a temporizar cuando se retira su señal de activación.

La estructura en SCL del temporizador es la siguiente:

```
"IEC_Timer_0_DB".TOF(IN:=_bool_in_,
                     PT:=_time_in_,
                     Q=>_bool_out_,
                     ET=>_time_out_);
```

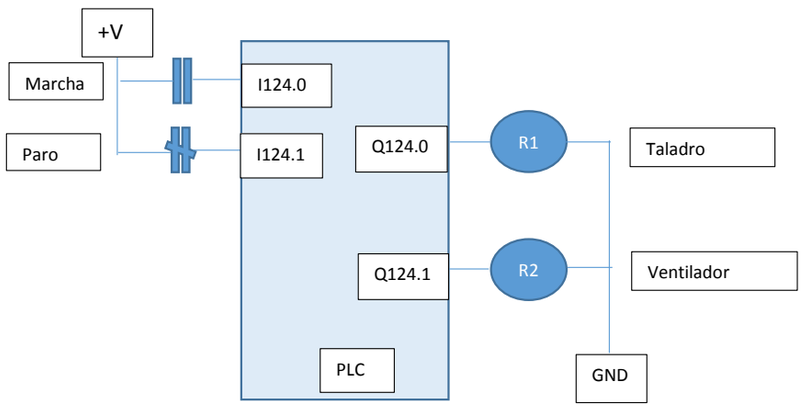
"IEC_Timer_0_DB" es el nombre que se le dará a la base de datos que contiene al temporizador; TOF significa que se configura el temporizador con retardo a la desconexión, _bool_in_ es la señal que activa el temporizador, _time_in_ es el tiempo de programación de temporizador en formato Time, _bool_out_ es la señal digital de salida del temporizador, y _time_out_ corresponde al área de memoria donde se deposita el valor actual de la cuenta del temporizador en formato Time.

Problema 4.6. Se dispone de una máquina taladradora que requiere de ventilación mientras está trabajando y de un tiempo adicional de 30 segundos luego de que es apagada.

El diagrama de conexiones se muestra en la Figura 60.

Figura 60.

Diagrama de conexiones del PLC para el problema 4.6



Fuente: elaboración propia.

La Tabla 29 describe las variables para utilizar en la solución del problema 4.6.

Tabla 29.

Variables para el problema 4.6

Nombre	Tipo	Dirección
Marcha	Bool	%I124.0
Paro	Bool	%I124.1
Taladro	Bool	%Q124.0
Ventilador	Bool	%Q124.1

Fuente: elaboración propia.

El programa en SCL sugerido se muestra a continuación:

```
IF “Marcha” THEN //Otra forma de presentar un SET RESET
    “Taladro” := 1;// Este sería el SET
ELSIF NOT “Paro” THEN //Evalúa la condición alternativa
    “Taladro” := 0; // Este sería el RESET
END_IF;

“Temporizador_Reposo”.TOF(IN:=”Taladro”,// El ventilador entrará al mismo
tiempo encendido
                          PT:=T#30S,// que el taladro, pero permanecerá
                          Q=>”Ventilador”); // por 30 segundos más
```

El programa propone un sistema de marcha del taladro con las funciones SET y RESET.

IF “Marcha” THEN “Taladro” := 1; el SET hace que el taladro se active al dar “Marcha”.

ELSIF NOT “Paro” THEN “Taladro” := 0; al dar paro se activa la función RESET y se apaga el taladro.

“Temporizador_Reposo” es el nombre que se le dio a la base de datos del temporizador.

IN:=”Taladro” es la instrucción que activa el temporizador, el cual provoca que se prenda inmediatamente el ventilador y siga activo por 30 segundos después de que el temporizador es desactivado.

Al simular recuerde que paro es un contacto normalmente cerrado. Al dar marcha se activan el taladro y el ventilador; cuando se da paro el taladro se desactiva, pero el ventilador continúa funcionando por treinta segundos.

4.4. Contaje de tiempo

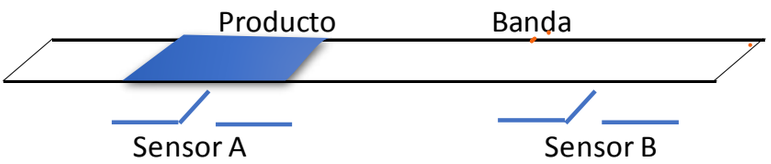
Se mencionó que una de las aplicaciones de los temporizadores es la de contar el tiempo. Por ejemplo, puede ser la duración de un evento o el tiempo de funcionamiento efectivo de una máquina. En TIA PORTAL el tiempo de avance del temporizador se puede tomar de la parte de la instrucción del temporizador ET=>. Los temporizadores se pueden utilizar para medir tiempos cuando no se requiere de mucha precisión. La ejecución del programa tiene alguna influencia sobre el tiempo de respuesta del temporizador.

Problema 4.7. Se necesita saber el tiempo que requiere un producto que pasa por una banda transportadora desde un punto A hasta un punto B. En caso de demorarse más de 15 segundos se debe activar una alarma por 5 segundos.

Se considerará que hay un espaciamiento adecuado de la circulación de productos, de tal forma que la secuencia permitida consiste en que se active un sensor en el punto A y luego otro sensor en el punto B. Luego de esto, se puede volver a activar el sensor A; es decir, el sensor A no se puede activar dos veces consecutivas sin activar primero el sensor B (Figura 61).

Figura 61.

Sistema de la banda del problema 4.7



Fuente: elaboración propia.

En la Tabla 30 se muestran las variables para utilizar en el problema 4.7.

Tabla 30.

Variables del problema 4.7

Variable	Tipo	Dirección	Comentario
Sensor_A	Bool	%I124.0	Detecta entrada de producto
Sensor_B	Bool	%I124.1	Detecta salida
Alarma	Bool	%Q124.0	Activar cuando excede tiempo A a B
Midiendo	Bool	%M2.0	Señal que indica que se está midiendo el tiempo
T_Recorrido	Time	%MD10	Tiempo medido
Conto	Bool	%M2.1	Señal que indica que se alcanzó el tiempo máximo
T_Avance	Time	%MD14	Registra el tiempo que duró el recorrido
T_Alarma	Time	%MD18	Cuando se excedió el tiempo
Flanco_Subida	Bool	%M2.2	Para detectar flanco de subida del detector B
Aux_Subida	Bool	%M2.3	Se usa como auxiliar

Fuente: elaboración propia.

A continuación, se muestra el programa sugerido en SCL:

```
IF “Sensor_A” THEN //Empieza la etapa de medida
    “Midiendo” := 1;
END_IF;
“IEC_Timer_0_DB”.TON(IN:=”Midiendo”,
               PT:=T#15s, //tiempo máximo para pasar de A a B
               Q=>”Conto”, // Registra si se alcanzó el tiempo máximo
               ET=>”T_Recorrido”); //va registrando el tiempo transcurrido

“Flanco_Subida” := “Sensor_B” AND NOT “Aux_Subida”;//Si pasa antes del
tiempo máximo
“Aux_Subida” := “Sensor_B”; // se guarda el tiempo y se apaga el primer
temporizador

IF “Flanco_Subida” THEN
    “T_Avance” := “T_Recorrido”;// Guardar el tiempo
    “Midiendo” := 0; // Reiniciar contador 1

END_IF;
```

```
"IEC_Timer_0_DB_1".TP(IN:="Conto", // se activa la alarma por 5 segundos
                      Q=>"Alarma",
                      PT:=T#5s,
                      ET=>"T_Alarma");
```

La etapa de medición del tiempo se inicia con la línea de instrucción IF "Sensor_A" THEN haciendo que se active la variable "Midiendo", con lo que se consigue activar el temporizador "IEC_Timer_0_DB" programado con 15 segundos. Si el objeto llega al punto B antes de 15 segundos, entonces, se reinicia el contador; de lo contrario, quiere decir que la variable "Conto" se activa, lo que provoca que también lo haga una alarma durante 5 segundos, controlada por un segundo temporizador "IEC_Timer_0_DB_1". La parte de la instrucción del primer temporizador ET=>"T_Recorrido" permite guardar el tiempo de avance del primer temporizador.

Para la simulación primero se activa y desactiva el sensor A, entonces, el primer temporizador empieza a trabajar durante 15 segundos; al concluir este tiempo se activa la alarma por 5 segundos (temporizador 2). Cuando se detecta sensor B se reinicia el temporizador. Si el sensor B se activa antes de los 15 segundos, no se pone en funcionamiento el temporizador 2 y tampoco la alarma. El tiempo que demora el transporte queda registrado en T_Avance.

Cuando se requiere más precisión en la medición del tiempo se debe recurrir al reloj del sistema del PLC, al uso de bloques especiales del PLC con esta aplicación o a la utilización del programa haciendo uso de los bloques de programación que trabajan por interrupciones.

Problema 4.8. Se desea saber las horas de funcionamiento de un compresor (horómetro) sin tener en cuenta los tiempos de parada. La actualización del horómetro se debe realizar cada minuto; además, se debe estar indicando el número de ciclos realizados por el compresor y el tiempo de funcionamiento del último ciclo.

En este problema se propone recurrir a la fecha y hora que registra el sistema del PLC en el momento de puesta en marcha del compresor y en el momento que se dé su parada. Se propone usar una base de datos como se describe en la Tabla 31 donde se muestran las variables para utilizar.

Tabla 31.

Variables de una base de datos llamada "HOR" para el problema 4.8

Base de datos 1 llamada en el programa "Hor"			
Nombre	**Dirección**	**Tipo**	**Comentario**
Ciclo_Run	%DB1.DBX0.0	BOOL	Indica si el ciclo está en proceso
Reset	%DB1.DBX0.1	BOOL	Reinicia el contador de ciclos y el horómetro
Flanco_Subida	%DB1.DBX0.2	BOOL	Detectar flanco de subida
Auxiliar_Subida	%DB1.DBX0.3	BOOL	Auxiliar flanco de subida
Flanco_Bajada	%DB1.DBX0.4	BOOL	Detectar flanco bajada
Auxiliar_ Bajada	%DB1.DBX0.5	BOOL	Auxiliar flanco bajada
Cuenta_Ciclos	%DB1.DBW2	INT	Contador de ciclos
Horometro	%DB1.DBD4	REAL	Acumulador de horas
Tiempo_Ult_Ciclo	%DB1.DBD8	REAL	Tiempo del último ciclo
Fecha_Hora_Inicio	P#DB1.DBX12.0	DATE_AND_TIME	Fecha y hora del inicio del último ciclo
Fecha_Hora_Final	P#DB1.DBX20.0	DATE_AND_TIME	Fecha y hora del final del último ciclo
Guardar_Fecha	P#DB1.DBX28.0	DATE_AND_TIME	Corresponde a la última fecha y hora guardadas
Ret_Val_D	%DB1.DBW36	INT	Ret_Val de función
Tiempo_Ciclo	%DB1.DBD38	Time	Sale de la comparación de las fechas y tiempos guardados

Fuente: elaboración propia.

El programa en SCL se detalla a continuación con los correspondientes comentarios. Se inicia con una parte para detectar los flancos de subida y bajada de la variable "Ciclo_Run". Cuando se detecta un flanco de subida, de activación del compresor, se procede a incrementar el contador de ciclos, capturar la fecha y hora de inicio del ciclo, guardar esta fecha e inicializar el tiempo de duración del ciclo que comienza.

Mientras el ciclo está en proceso, el compresor debe estar trabajando; se calcula el tiempo del ciclo actual haciendo la diferencia entre la última fecha leída y la que se había guardado. Cada vez que pasan 60 segundos se actualizan el tiempo del último ciclo y del horómetro. Al ser estos dos últimos reales, se debe hacer la conversión correspondiente y dividir por 3 600 000 para convertirlos en horas.

En el flanco de bajada se apaga el compresor, se leen la fecha y hora finales, se calcula el tiempo remanente desde el último minuto del ciclo que terminó, y se actualizan el tiempo del último ciclo y del horómetro.

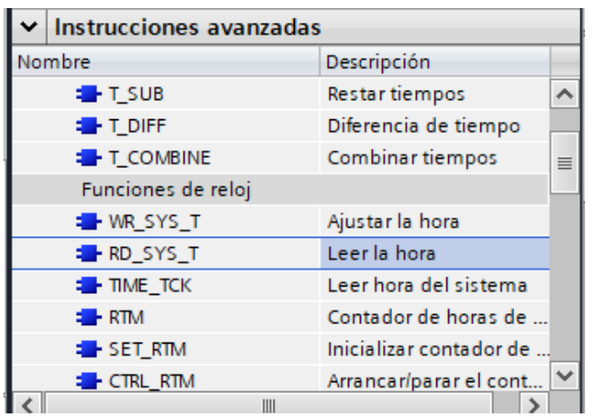
```
//Etapa para detectar flancos
"Hor".Flanco_Subida := "Hor".Ciclo_Run AND NOT "Hor".Auxiliar_Subida;
"Hor".Auxiliar_Subida := "Hor".Ciclo_Run;
```

```
"Hor".Flanco_Bajada := NOT "Hor".Ciclo_Run AND NOT "Hor"."Auxiliar_ Bajada";
"Hor"."Auxiliar_ Bajada" := NOT "Hor".Ciclo_Run;

//Etapa para caso de flanco positivo de activación del compresor
//Su función es incrementar el contador de ciclos, capturar la fecha y hora
//de inicio del ciclo, guardar esta fecha e inicializar el tiempo de duración
//del ciclo que comienza
IF "Hor".Flanco_Subida THEN
    "Hor".Cuenta_Ciclos := "Hor".Cuenta_Ciclos + 1;
    "Hor".Ret_Val_D:= RD_SYS_T("Hor".Fecha_Hora_Inicio);
    "Hor".Guardar_Fecha := "Hor".Fecha_Hora_Inicio;
    "Hor".Tiempo_Ult_Ciclo := 0.0;
END_IF;

//Mientras el ciclo está en proceso, compresor trabajando
//Se calcula el tiempo del ciclo actual, haciendo la diferencia entre la última fecha
//leída y la que se había guardado. Cada vez que pasan 60 segundos se actualiza el tiempo
//del último ciclo y del horómetro. Al ser estos dos úlimos reales se debe hacer la
//conversión correspondiente y dividir por 3 600 000 para convertirlo en horas
IF "Hor".Ciclo_Run THEN
    "Hor".Ret_Val_D:=RD_SYS_T("Hor".Fecha_Hora_Final);
    "Hor".Tiempo_Ciclo := "Hor".Fecha_Hora_Final - "Hor".Guardar_Fecha;
    IF TIME_TO_DINT("Hor".Tiempo_Ciclo) >= 60000 THEN
        "Hor".Tiempo_Ult_Ciclo := "Hor".Tiempo_Ult_Ciclo +
DINT_TO_REAL(TIME_TO_DINT("Hor".Tiempo_Ciclo)) / 3600000;
        "Hor".Horometro := "Hor".Horometro +
DINT_TO_REAL(TIME_TO_DINT("Hor".Tiempo_Ciclo)) / 3600000;
        "Hor".Guardar_Fecha := "Hor".Fecha_Hora_Final;
    END_IF;
END_IF;

// En el flanco de bajada, se apaga el compresor
// Se leen la fecha y hora finales, se calcula el tiempo remanente desde el últi-
mo minuto del ciclo
// que terminó, se actualizan el tiempo del último ciclo y del horómetro

IF "Hor".Flanco_Bajada THEN
    "Hor".Ret_Val_D:= RD_SYS_T("Hor".Fecha_Hora_Final);
    "Hor".Tiempo_Ciclo := "Hor".Fecha_Hora_Final - "Hor".Guardar_Fecha;
    "Hor".Tiempo_Ult_Ciclo := "Hor".Tiempo_Ult_Ciclo +
DINT_TO_REAL(TIME_TO_DINT("Hor".Tiempo_Ciclo)) / 3600000;
    "Hor".Horometro := "Hor".Horometro +
DINT_TO_REAL(TIME_TO_DINT("Hor".Tiempo_Ciclo)) / 3600000;
END_IF;

//Borrado de acumuladores cuando se da Reset
IF "Hor".Reset THEN
    "Hor".Cuenta_Ciclos := 0;
    "Hor".Horometro := 0.0;
END_IF;
```

La instrucción “Hor”.Ret_Val_D:= RD_SYS_T(“Hor”.Fecha_Hora_Inicio) se obtiene de una función especial del TIA PORTAL que aparece en funciones avanzadas, como se muestra en la Figura 62. Esta función permite leer la fecha y la hora actuales del reloj de la CPU. La parte de la función “Hor”.Ret_Val_D corresponde al espacio de memoria donde se quiere depositar un código de error que se presente durante la ejecución de la instrucción; se guarda en formato de entero, en una palabra. (“Hor”.Fecha_Hora_Inicio) es el área de memoria donde se quiere guardar la fecha y hora capturadas, por lo que este espacio debe estar en formato de DATE_AND_TIME, y ocupa un espacio 8 bytes.

Figura 62.

Lugar en TIA PORTAL donde se toma la instrucción para leer la hora

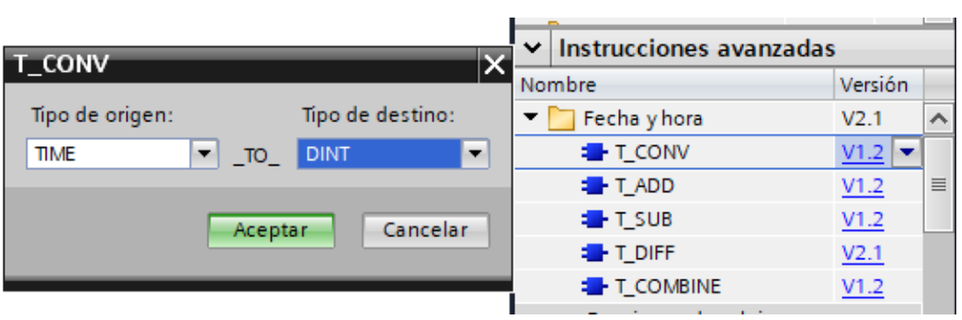


Fuente: elaboración propia.

La instrucción DINT_TO_REAL(TIME_TO_DINT(“Hor”.Tiempo_Ciclo)) / 3600000, en realidad, es un proceso de conversiones para llevar a formato REAL el valor que se encuentra en la variable “Tiempo_Ciclo”, que está en formato de TIME. Entonces, se pasa de TIME a DINT y luego a REAL. Finalmente, se divide por 3 600 000 para que el resultado se interprete en horas. En la Figura 63 se puede observar por dónde se puede acceder a esta instrucción en TIA PORTAL, utilizando la función T_CONV por instrucciones avanzadas; entonces, aparece una ventana donde se debe especificar el tipo de dato origen y destino.

Figura 63.

Acceder a una instrucción para convertir formatos de tiempos para el problema 4.8

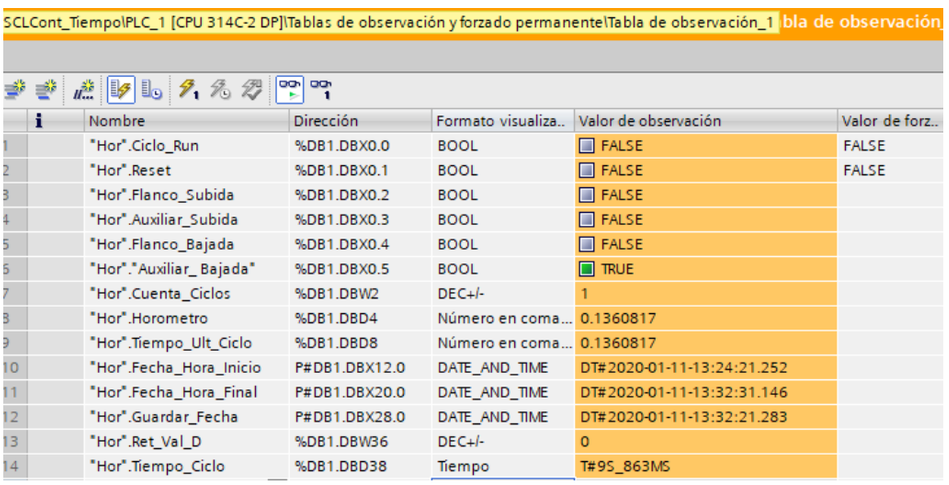


Fuente: elaboración propia.

A continuación, se muestran en la Figura 64 y en la Figura 65 los resultados de la ejecución de un primer ciclo y un segundo ciclo de trabajo de la máquina. Se destaca que el primer ciclo el horómetro y el tiempo del último ciclo son iguales.

Figura 64.

Resultado de la ejecución de un primer ciclo del programa del problema 4.8

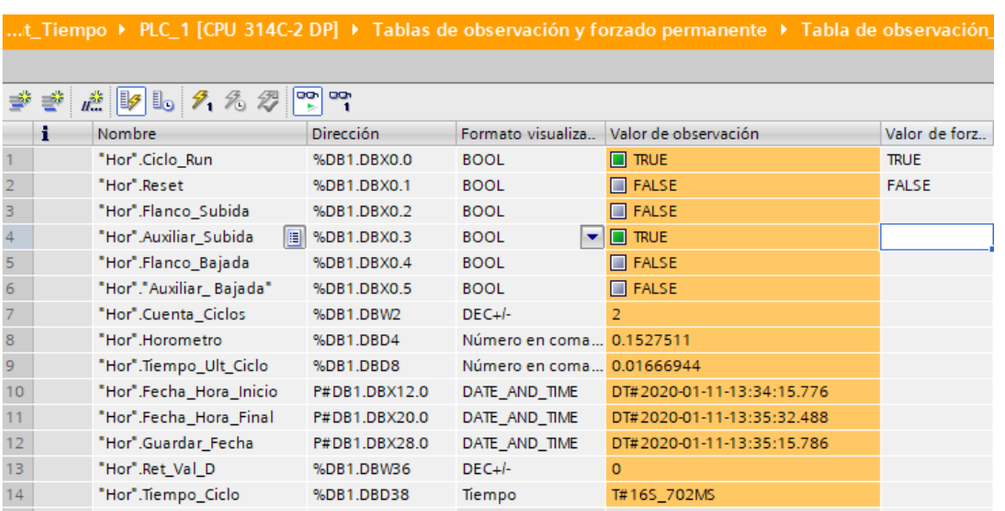


	Nombre	Dirección	Formato visualiza..	Valor de observación	Valor de forz..
1	"Hor".Ciclo_Run	%DB1.DBX0.0	BOOL	FALSE	FALSE
2	"Hor".Reset	%DB1.DBX0.1	BOOL	FALSE	FALSE
3	"Hor".Flanco_Subida	%DB1.DBX0.2	BOOL	FALSE	
4	"Hor".Auxiliar_Subida	%DB1.DBX0.3	BOOL	FALSE	
5	"Hor".Flanco_Bajada	%DB1.DBX0.4	BOOL	FALSE	
6	"Hor"."Auxiliar_ Bajada"	%DB1.DBX0.5	BOOL	TRUE	
7	"Hor".Cuenta_Ciclos	%DB1.DBW2	DEC+/-	1	
8	"Hor".Horometro	%DB1.DBD4	Número en coma...	0.1360817	
9	"Hor".Tiempo_Ult_Ciclo	%DB1.DBD8	Número en coma...	0.1360817	
10	"Hor".Fecha_Hora_Inicio	P#DB1.DBX12.0	DATE_AND_TIME	DT#2020-01-11-13:24:21.252	
11	"Hor".Fecha_Hora_Final	P#DB1.DBX20.0	DATE_AND_TIME	DT#2020-01-11-13:32:31.146	
12	"Hor".Guardar_Fecha	P#DB1.DBX28.0	DATE_AND_TIME	DT#2020-01-11-13:32:21.283	
13	"Hor".Ret_Val_D	%DB1.DBW36	DEC+/-	0	
14	"Hor".Tiempo_Ciclo	%DB1.DBD38	Tiempo	T#9S_863MS	

Fuente: elaboración propia.

Figura 65.

Resultado de la ejecución de un segundo ciclo del programa del problema 4.8



	Nombre	Dirección	Formato visualiza..	Valor de observación	Valor de forz..
1	"Hor".Ciclo_Run	%DB1.DBX0.0	BOOL	TRUE	TRUE
2	"Hor".Reset	%DB1.DBX0.1	BOOL	FALSE	FALSE
3	"Hor".Flanco_Subida	%DB1.DBX0.2	BOOL	FALSE	
4	"Hor".Auxiliar_Subida	%DB1.DBX0.3	BOOL	TRUE	
5	"Hor".Flanco_Bajada	%DB1.DBX0.4	BOOL	FALSE	
6	"Hor"."Auxiliar_ Bajada"	%DB1.DBX0.5	BOOL	FALSE	
7	"Hor".Cuenta_Ciclos	%DB1.DBW2	DEC+/-	2	
8	"Hor".Horometro	%DB1.DBD4	Número en coma...	0.1527511	
9	"Hor".Tiempo_Ult_Ciclo	%DB1.DBD8	Número en coma...	0.01666944	
10	"Hor".Fecha_Hora_Inicio	P#DB1.DBX12.0	DATE_AND_TIME	DT#2020-01-11-13:34:15.776	
11	"Hor".Fecha_Hora_Final	P#DB1.DBX20.0	DATE_AND_TIME	DT#2020-01-11-13:35:32.488	
12	"Hor".Guardar_Fecha	P#DB1.DBX28.0	DATE_AND_TIME	DT#2020-01-11-13:35:15.786	
13	"Hor".Ret_Val_D	%DB1.DBW36	DEC+/-	0	
14	"Hor".Tiempo_Ciclo	%DB1.DBD38	Tiempo	T#16S_702MS	

Fuente: elaboración propia.

Problema 4.9. Se requiere saber las horas de funcionamiento de un motor, y que un registro adicional indique el tiempo parcial de la última hora.

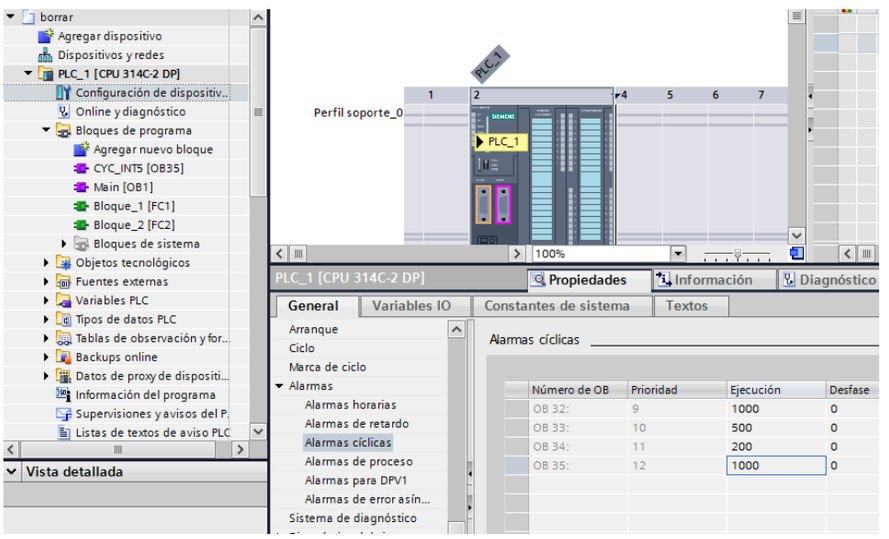
En este caso, para resolver el problema con una CPU 314C2DP, se recurrirá a los bloques de organización de alarma cíclica del PLC que trabajan por interrupciones. Estos bloques permiten ejecutar programas a espacios equidistantes. El núme-

ro de estos depende de la CPU por lo que se debe consultar el respectivo catálogo. En la CPU 314C2DP se dispone del OB35, que viene configurado de fábrica, para que se ejecute cada 100 ms, pero se ajustará para que lo haga cada segundo (este será el tiempo de actualización del horómetro); también, se pudo haber utilizado el OB32 que viene configurado con 1 segundo.

Para configurar o, más bien, para cambiar la configuración del tiempo de ejecución del OB35, se debe entrar al menú de configuración de dispositivos y, por propiedades de la CPU, se buscan las alarmas cíclicas y se procede a cambiar el tiempo de ejecución del bloque en milisegundos; en este caso, se ajustará 1000 ms (1 s) en el OB35, tal como se muestra en la Figura 66.

Figura 66.

Configuración de la alarma cíclica del OB35 para el problema 4.9



Fuente: elaboración propia.

La Tabla 32 muestra las variables para utilizar. En este problema se utilizó el área de memoria de marcas, pero recuerde que se pueden utilizar otras áreas como, por ejemplo, los bloques de datos.

Tabla 32.

Variables empleadas en el problema 4.9

Variable	Tipo	Dirección	Comentario
Señal_Motor	Bool	%M2.0	Señal que indica que el motor está energizado
Reset_Horómetro	Bool	%M2.1	Señal para poner el horómetro en cero
Contador	DInt	%MD22	Su valor se incrementa cada segundo, cada vez que pasan 3600 segundos se pone en cero
Horómetro	DInt	%MD26	Registra las horas, se actualiza cada hora
Parcial	Time	%MD30	Registra en formato de tiempo, el tiempo parcial menor a 1 hora

Fuente: elaboración propia.

El programa debe ser ingresado en el bloque de organización correspondiente, en este caso el OB35; no hay necesidad de invocarlo desde el OB1 ni de otro bloque de función. Así, durante la ejecución normal del programa, cada segundo se hará una interrupción para ejecutar lo que aquí está programado. Se debe tener en cuenta que la ejecución del programa introducido en OB35 no dure más de 1 s; de lo contrario, se producirá un error de ejecución. El programa SCL para ingresar en el OB35 es el siguiente:

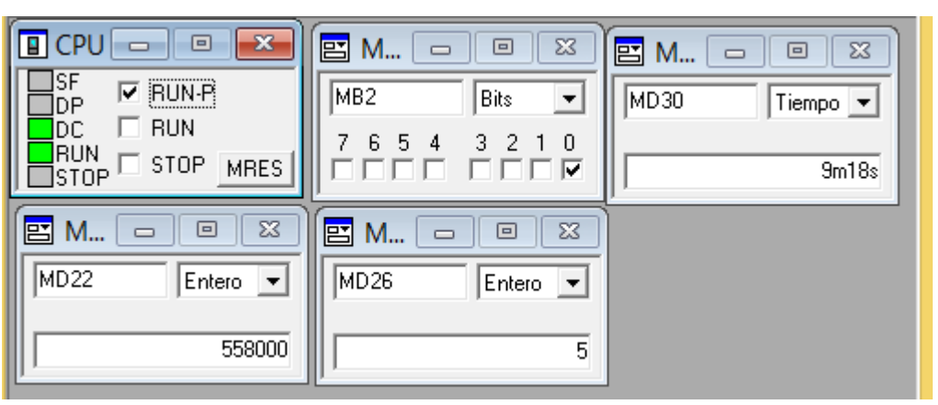
```
IF "Señal_Motor"  THEN
    "Contador" := "Contador" + 1000;// Se incrementa cada 1000 ms (1 s)
    "Parcial" := DINT_TO_TIME("Contador"); // se pasa a formato tiempo
    IF "Contador" >= 3600000 THEN //recuerde que el tiempo está en ms
        "Contador" := 0; // Se reinicia si pasó una hora
        "Horómetro" := "Horómetro" + 1; // se incrementa cada hora
    END_IF;
END_IF;

IF "Reset_Horómetro" THEN  //Resetear el horómetro
    "Contador" := 0;
    "Parcial" := t#0s;
    "Horómetro" := 0;
END_IF;
```

Durante la ejecución del programa se observará que MD22 (Contador) se incrementa en 1000 cada segundo, en MD26 (Horómetro) se muestra el tiempo en horas de funcionamiento del motor y en MD30 (Parcial) se registra el tiempo parcial en formato de tiempo. Cada vez que se quita la señal M2.0 (Motor encendido) o se apaga la CPU los registros se detienen, pero no pierden la cuenta al prenderse el motor o encender la CPU.

En la Figura 67 se puede ver un ejemplo de la simulación. El motor lleva funcionando 5 horas, con un tiempo parcial de 9 minutos y 18 segundos, que corresponden a 558 000 ms.

Figura 67.

Simulación del horómetro del problema 4.9



Fuente: elaboración propia.

Problema 4.10. Se dispone de una máquina mezcladora de varios productos. El operario debe asignar el tiempo del proceso de acuerdo con valores preestablecidos en el Departamento de Producción; se debe garantizar que el tiempo de proceso sea el tiempo efectivo mientras el motor del mezclador está activo, es decir, no se deben tener en cuenta los tiempos debidos al paro de la máquina para preparación del proceso.

Para este problema se utilizará un PLC S7-1200, el cual tiene incluido para su programación en TIA PORTAL un temporizador "TONR": sirve para acumular tiempo dentro de un período predefinido. En el PLC S7-300 se puede utilizar una instrucción similar, pero que debe ser llamada desde un bloque de organización con tiempos equidistantes. Como se observa, este temporizador se pudo utilizar para los horómetros que se han propuesto en otros problemas. El valor máximo del tiempo preestablecido, al ser de tipo Time (32 bits), es de T#10d_20h_30m_20s_630ms, que en total suman algo más de 240 horas.

En la Tabla 33 se detallan las variables para utilizar.

Tabla 33.

Variables empleadas en el problema 4.10

Variable	Tipo	Dirección	Comentario
Motor_Mezcla	Bool	%M2.0	Señal que hace que el temporizador esté en funcionamiento
Reset	Bool	%I0.0	Señal para reiniciar el temporizador
T_Proceso	Time	%MD10	Informa el tiempo que se tiene programado para el proceso en formato Time. Se calcula
Horas_Proceso	DInt	%MD14	Operario indica horas proceso
Minutos_Proceso	DInt	%MD18	Operario indica minutos proceso
Segundos_Proceso	DInt	%MD22	Operario indica segundos proceso
T_Agregado	DInt	%MD26	Se calcula el tiempo requerido de proceso en ms
T_Recorrido	Time	%MD30	El temporizador informa tiempo recorrido del proceso en formato Time
A_Proceso listo	Bool	%Q0.0	Alarma que indica que el proceso ha concluido

Fuente: elaboración propia.

A continuación, se muestra el programa propuesto en SCL. Para hallar el "T_Agregado" se convierten todos los tiempos a milisegundos; así, las horas se multiplican por 3 600 000, los minutos por 60 000 y los segundos por 1000. "IEC_Timer_0_DB".TONR es la parte de instrucción que invoca un temporizador tipo TONR y guarda sus datos en una base de latos llamada "IEC_Timer_0_DB", "Motor_Mezcla" es la variable que hace que el temporizador esté en funcionamiento, "T_Proceso" es el tiempo programado para el funcionamiento de la máquina, y "T_Recorrido" es el tiempo acumulado por el temporizador mientras está en funcionamiento.

```
“T_Agregado” := 3600000 * “Horas_Proceso” + 60000 * “Minutos_Proceso” +
“Segundos_Proceso” * 1000;
“T_Proceso” := DINT_TO_TIME(“T_Agregado”);// De las horas, minutos
y segundos dados por el operario
                      // se calcula el Tiempo de proceso en TIME
“IEC_Timer_0_DB”.TONR(IN:=”Motor_Mezcla”, // Se activa temporizador acumulador
listo”,R:=”Reset”);    PT:=”T_Proceso”,ET=>”T_Recorrido”,Q=>”A_Proceso
```

Para la simulación en un PLC real, se acude a la tabla de observación (Figura 68). Se ve que el operario ingresó 0h5m40s y luego de este tiempo se activó la alarma; la señal M2.0, que se supone indica si el motor está energizado, fue interrumpida en varias ocasiones; el tiempo acumulado se mantuvo.

Figura 68.

Tabla de observación para el problema 4.10

Nombre	Dirección	Formato visualiz...	Valor de observac..	Valor de forzado
"T_Proceso"	%MD10	Tiempo	T#5M_40S	
"Horas_Proceso"	%MD14	DEC+/-	0	0
"Minutos_Proceso"	%MD18	DEC+/-	5	5
"Segundos_Proceso"	%MD22	DEC+/-	40	40
"T_Agregado"	%MD26	DEC+/-	340000	
"T_Recorrido"	%MD30	Tiempo	T#5M_40S	
"Reset"	%I0.0	BOOL	FALSE	
"Motor_Mezcla"	%M2.0	BOOL	TRUE	TRUE
"A_Proceso listo"	%Q0.0	BOOL	TRUE	

Fuente: elaboración propia.

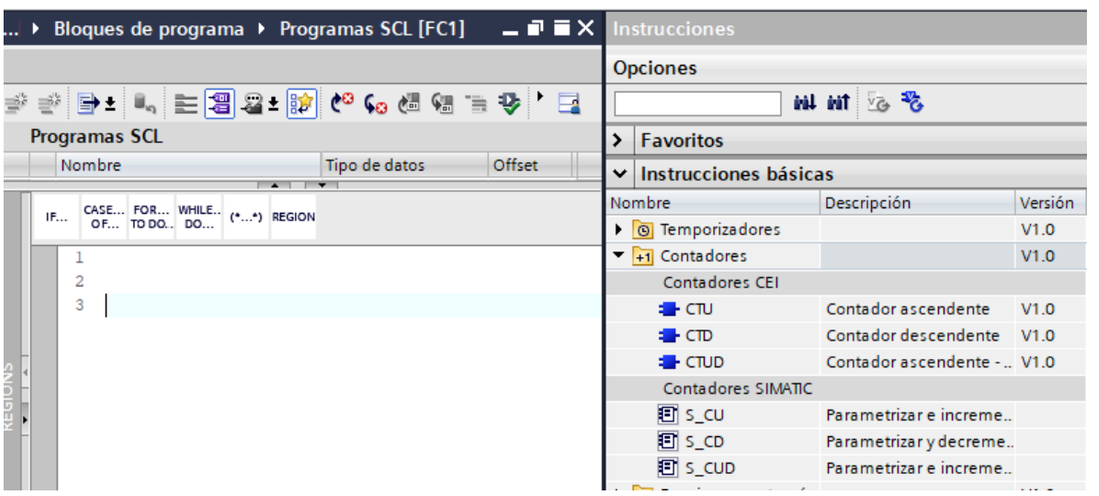
05. CONTADORES

5. Contadores

Los contadores permiten llevar la cuenta de eventos en el proceso, lo que posibilita la toma de decisiones dependiendo del valor se tenga almacenado. En TIA PORTAL, el contador más completo permite contar ascendente y descendentemente, preajustar un valor de cuenta, reiniciar el contador, activar una salida cuando se llega al valor preajustado o activar una salida cuando está en cero; también, se puede almacenar el valor de la cuenta para realizar cálculos y facilitar la toma de decisiones de control. No es necesario utilizar todas las funciones del contador. Para emplearlas se debe usar una base de datos propia a la cual se le puede asignar un nombre. La Figura 69 muestra el entorno de TIA PORTAL para cargar un contador.

Figura 69.

Entorno TIA PORTAL para cargar un contador

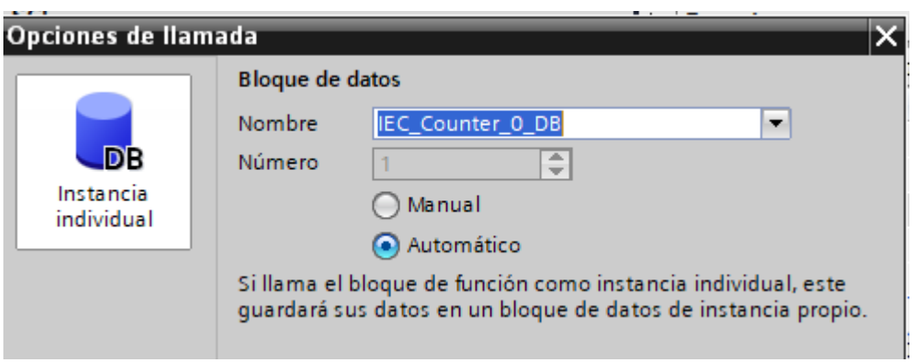


Fuente: elaboración propia.

El contador debe ser arrastrado al área de trabajo, pudiéndosele asignar un nombre a la base de datos correspondiente, tal como se muestra en la Figura 70.

Figura 70.

Asignación de una base de datos a un contador en TIA PORTAL

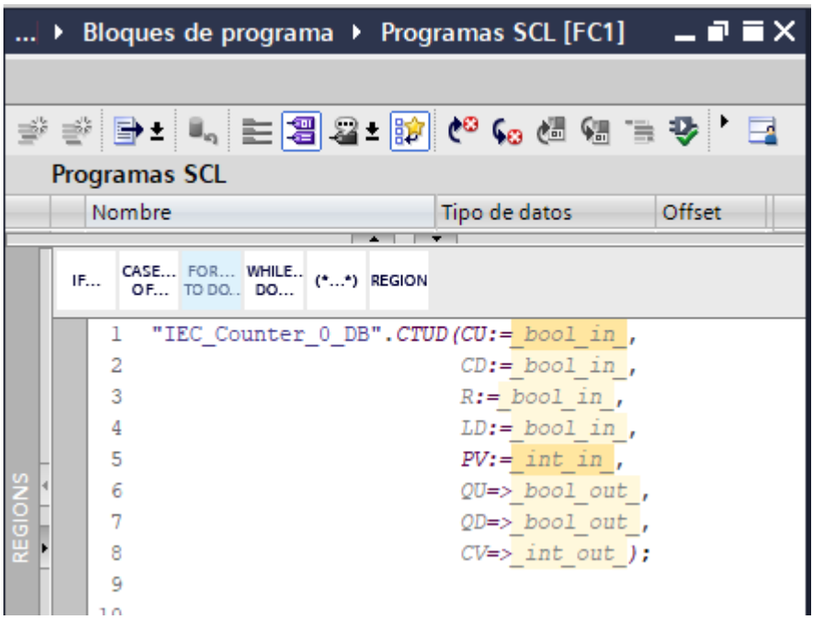


Fuente: elaboración propia.

Al dar aceptar aparece la siguiente estructura de un programa en SCL (Figura 71).

Figura 71.

Estructura de un contador en SCL



Fuente: elaboración propia.

El programa en SCL es el siguiente para un contador completo:

```
“IEC_Counter_0_DB”.CTUD(CU:=_bool_in_,
                        CD:=_bool_in_,
                        R:=_bool_in_,
                        LD:=_bool_in_,
                        PV:=_int_in_,
                        QU=>_bool_out_,
                        QD=>_bool_out_,
                        CV=>_int_out_);
```

“IEC_Counter_0_DB”.CTUD: corresponde al nombre de la base de datos del contador con la extensión CTUD, que pertenece a un contador ascendente y descendente.

CU:=_bool_in_: acá se indica señal booleana que recibirá los pulsos para cuenta ascendente.

CD:=_bool_in_: señal booleana para los pulsos de cuenta descendente.

R:=_bool_in_: señal para reiniciar el contador.

LD:=_bool_in_: señal booleana para preajustar un valor de contaje.PV:=_int_in_: acá se ingresa el valor de preajuste del contador en formato de entero (en una palabra). Cuando el contador alcanza o supera este valor su salida QU se coloca en 1.

QU=>_bool_out_: señal de salida del contador que se activa cuando se llega o supera el valor de preajuste.

QD=>_bool_out_: señal de salida del contador que se activa cuando el valor de cuenta llega o está por debajo de 0.

CV=>_int_out_: acá se almacena en formato de entero (Palabra) el valor actual de la cuenta puede tomar valores negativos.

Problema 5.1. Cree un algoritmo para llevar la cuenta de las cajas ubicadas dentro de una banda transportadora. Se dispone de un sensor de entrada de cajas, un sensor de salida de cajas y un botón para reiniciar el contador. Una lámpara verde se encenderá cuando en la banda haya menos de 4 cajas, una amarilla cuando haya entre 5 y 9 cajas y una roja cuando haya 10 o más cajas dentro de la banda.

Se proponen las variables en la Tabla 34 para solucionar el problema 5.1

Tabla 34.

Variables empleadas en el problema 5.1

Nombre	Tipo	Dirección	Comentario
Entra	Bool	%I124.0	Pulsos ascendentes
Sale	Bool	%I124.1	Pulsos descendentes
Reinicio	Bool	%I124.2	Lleva contador a 0
Verde	Bool	%Q124.0	Salida
Amarillo	Bool	%Q124.1	Salida
Rojo	Bool	%Q124.2	Salida
Cuenta	Int	%MW10	Valor del contador

Fuente: elaboración propia.

Se propone el siguiente algoritmo en SCL:

```
"IEC_Counter_0_DB".CTUD(CU:="Entra",
                        CD:="Sale",
                        R:="Reinicio",
                        PV:=10,
                        CV=>"Cuenta");

"Verde" := ("Cuenta"< 5);
"Amarillo" := ("Cuenta" < 10 AND "Cuenta" > 4  );
"Rojo" := "Cuenta" > 9;
```

Observe que no son requeridas todas las funciones del contador; PV es necesaria, aunque en este ejemplo no se utiliza. La primera parte del programa controla los parámetros del contador, que en este caso se llamó "IEC_Counter_0_DB". Para que el contador incremente su cuenta es necesario dar pulsos por medio de

"Entra"; para los de decremento los pulsos se dan con "Sale". La segunda parte del programa es utilizada para activar las salidas "Verde", "Amarillo" y "Rojo", de acuerdo con el valor actual del contador "Cuenta".

Problema 5.2. Se dispone de un depósito al cual le caen piezas desde una banda transportadora. Se desea conocer el peso aproximado del contenedor y el volumen ocupado a partir del número de piezas, su peso y volumen individual. Se dispone de un sensor que se activa por cada pieza que cae desde la banda. Dos alarmas se deben activar cuando se supere el peso o el volumen, previamente establecidos por el operario.

En la Tabla 35 se muestran las variables para utilizar.

Tabla 35.

Variables empleadas en el problema 5.2

Nombre	Tipo	Dirección	Observación
Sensor	Bool	%I0.0	Detecta las piezas que caen de la banda
Reset	Bool	%I0.1	Reinicia el contador
Pulsos	Int	%MW10	Los acumulados en el contador
Cuenta	Real	%MD12	Los pulsos convertidos a real
Peso_Unidad	Real	%MD16	El operario introduce este valor en Kg
Volumen_Unidad	Real	%MD20	El operario introduce este valor en m^3
Peso_Limite	Real	%MD24	El operario introduce este valor en Kg
Volumen_Limite	Real	%MD28	El operario introduce este valor en m^3
Peso_Acumulado	Real	%MD32	Multiplicación de cuenta por peso unitario
Volumen_Acumulado	Real	%MD36	Multiplicación de cuenta por volumen unitario
Alarma_Peso	Bool	%Q0.0	Al superar peso límite
Alarma_Volumen	Bool	%Q0.1	Al superar volumen límite

Fuente: elaboración propia.

Se propone el siguiente algoritmo en SCL:

```
"Cuenta_Piezas".CTU(CU:="Sensor",//contador ascendente
                    R:="Reset",
                    PV:=1000,//No interesa el valor
                    CV=>"Pulsos");
"Cuenta":=INT_TO_REAL("Pulsos");
"Peso_Acumulado":="Peso_Unidad"*"Cuenta";
"Volumen_Acumulado" := "Volumen_Unidad" * "Cuenta";
"Alarma_Peso" := "Peso_Acumulado" > "Peso_Limite";
"Alarma_Volumen" := "Volumen_Acumulado" > "Volumen_Limite";
```

El programa es simple; en este caso, se utiliza un contador ascendente (CTU). La primera parte del programa controla el contador al que se le dio el nombre de "Cuenta_Piezas"; la cuenta actual es registrada en la variable "Pulsos". En la segunda parte, el valor de contaje es convertido a real en "Cuenta", que es utilizada para

hallar el valor del “Peso_Acumulado” y el de “Volumen_Acumulado”. Con estos valores se activan las alarmas de acuerdo con el resultado de los comparadores.

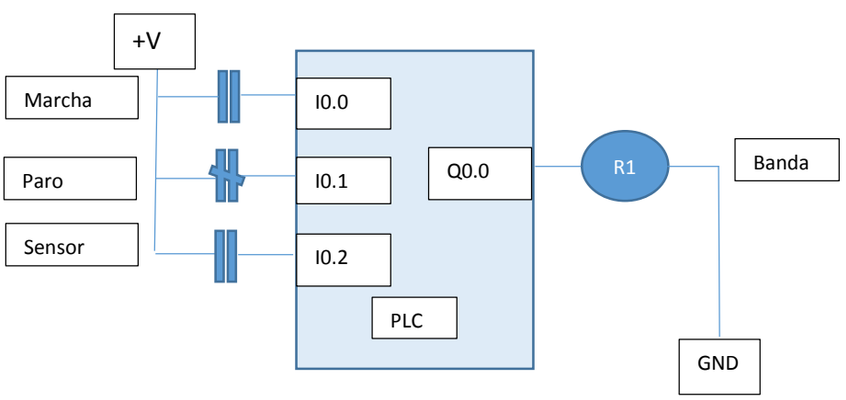
Para realizar la simulación se deben llenar los campos de peso unidad, volumen unidad, peso límite y volumen límite; luego, se observa lo que ocurre cuando dan los pulsos del sensor.

Problema 5.3. Cada vez que se da marcha se pone en funcionamiento una banda transportadora que va depositando paquetes en un contenedor. Al contabilizarse los paquetes preajustados por un operario, la banda se detiene y se debe dar marcha para un nuevo proceso. Si en el transcurso de un proceso se pulsa el botón de Paro, la cuenta acumulada hasta el momento no se debe perder.

El diagrama de conexiones del PLC se muestra en la Figura 72.

Figura 72.

Diagrama de conexiones para el problema 5.3



Fuente: elaboración propia.

La Tabla 36 muestra las variables requeridas en el problema 5.3.

Tabla 36.

Variables empleadas en el problema 5.3

Nombre	Tipo	Dirección	Descripción
Marcha	Bool	%I0.0	Poner en funcionamiento la banda, recargar el contador cuando la cuenta es 0
Paro	Bool	%I0.1	Parar la banda
Banda	Bool	%Q0.0	Salida del PLC para mover la banda
Preajuste	Int	%MW10	Introducido por el operario; número de paquetes que se van a contabilizar en cada proceso
Sensor	Bool	%I0.2	Cuenta los paquetes
Cuenta	Int	%MW12	La que lleva el contador
Estado_Cont	Bool	%M2.0	Se pone a 1 cuando la cuenta llega a 0

Detector flanco	Bool	%M2.1	Para detectar el momento en que el estado de contador pasa de 0 a 1
Auxiliar flanco	Bool	%M2.2	Necesario para detectar flanco positivo

Fuente: elaboración propia.

El algoritmo propuesto en SCL es el siguiente:

```
"Contador1".CTD(CD:="Sensor",// contador descendente
preajuste       LD:="Marcha" AND "Cuenta"=0 ,//momento en que se carga el
                PV:="Preajuste",// Número de paquetes que se quiere contabilizar
                Q=>"Estado_Cont",// se activa cuando la cuenta llega a 0
                CV=>"Cuenta"); //Se lleva el valor del contaje
IF "Marcha" THEN// Función SET
    "Banda" := 1;
END_IF;
"Detector flanco" := "Estado_Cont" AND NOT "Auxiliar flanco";
"Auxiliar flanco" := "Estado_Cont"; // Se requiere detector de flanco para apagar
IF NOT "Paro" OR "Detector flanco" THEN// Función RESET
    "Banda" := 0;//La banda se detiene cuando se da paro o cuando el contador llega a 0
END_IF;
```

En este ejemplo se decidió utilizar un contador descendente (CTD) y su salida Q para realizar la acción de control, la cual se pone a 1 cuando el contador llega a 0. Se deben considerar varios aspectos; el primero se refiere al apagado de la banda cuando se da paro o cuando la cuenta de los paquetes llega a cero, momento en el cual la salida Q del contador (estado contador) se pone a 1. Es importante determinar el momento en que se produce el cambio, por lo que se requiere detectar el flanco positivo de "Estado_Cont". El otro aspecto se refiere a la precarga del contador, la cual solo se puede dar cuando se presiona marcha y la cuenta está en cero. Cuando se para el proceso en el intermedio, y se da marcha nuevamente, el único efecto es el de poner en marcha la banda; la cuenta llevada por el contador se debe conservar.

Problema 5.4. Se quiere diseñar un horómetro para determinar el tiempo de funcionamiento de un motor. Para esto, se dispone de la señal de activación del motor y una de reset. En un registro adicional se debe indicar el tiempo transcurrido desde la última hora, el cual se llamará tiempo parcial.

Ya se presentó un ejemplo similar, pero ahora se trabajará con contadores ascendentes en cascada en una CPU S7-300, 314C 2DP, resultando dos aspectos importantes por resolver si se utiliza un bloque de organización de alarma cíclica (OB32). El primero se debe a que los contadores cuentan con el pulso de subida de una señal, lo que se solucionará, en este caso, haciendo que la señal de conteo se invierta cada vez que se ejecute el bloque de organización. Como para este ejemplo el OB32 se ejecuta cada segundo, entonces, el contador se incrementará cada dos segundos. También se podría configurar un tiempo de 500 ms en este bloque y

así el contador incrementaría cada segundo; o, si existe la posibilidad, se utiliza el OB33 que se ejecuta cada 500 ms, pero se dejará en el OB32 para fines de estudio. El segundo aspecto para resolver tiene que ver con el hecho de que los contadores normales cuentan hasta 32 767 que, si se toma en segundos y se traduce en tiempo, correspondería a 9h6m7s o, en nuestro caso, al doble de tiempo, 18h12m14s. En la mayoría de los casos se requiere de un horómetro de mayor capacidad de registro en horas, por lo que se requiere de otro contador. Entonces, se dejará un contador que cuente hasta 1800 (3600 segundos), se reinicie y se incremente la cuenta de un segundo contador que sirve para contabilizar las horas, con una capacidad de registro de 32 767 horas (aproximadamente 3,7 años).

La Tabla 37 muestra las variables para utilizar en el problema 5.4

Tabla 37.

Variables para emplear en el problema 5.4

Nombre	Tipo	Dirección	Descripción
Señal_Motor	Bool	%M2.0	El horómetro debe contar mientras esta señal este activa
Pulso	Bool	%M2.1	Para generar los pulsos que incrementan el primer contador. Tiene un período de 2 segundos
Cuenta1	Int	%MW10	Lleva la cuenta de los pulsos, incrementa cada dos segundos
Reset	Bool	%M2.2	Para reiniciar el horómetro y el tiempo parcial
Horómetro	Int	%MW12	Cuenta las horas de funcionamiento
Parcial	Time	%MD14	Registro de la fracción de hora en curso

Fuente: elaboración propia.

El programa en SCL propuesto para ingresar en el OB32 es el siguiente:

```
IF "Señal_Motor" THEN // Mantiene activo el horómetro
   "Pulso" := NOT "Pulso"; //Se utiliza para incrementar Cuenta1 con el pulso
END_IF;                    //positivo; la clave está en invertir la señal cada vez
                              //que se ejecuta OB32

"IEC_Counter_0_DB".CTU(CU := "Pulso", //Este es el contador de Cuenta1
                       R := ("IEC_Counter_0_DB".Q OR "Reset"),
                       PV := 1800, // Equivale a 3600 segundos
                       CV => "Cuenta1");

"IEC_Counter_0_DB_1".CTU(CU:="IEC_Counter_0_DB".Q,// este es el contador de horas
                         R:="Reset",
                         PV:=10, // No importa su valor <32767
                         CV=> "Horómetro");

"Parcial":=DINT_TO_TIME(INT_TO_DINT("Cuenta1"))*2*1000; // Calcular el tiempo
IF "Reset" THEN // parcial en milisegundos y se pasa a formato tiempo
    "Cuenta1" := 0; //Para reiniciar los contadores con Reset
    "Horómetro" := 0;
END_IF;
```

La instrucción “Pulso” := NOT “Pulso”; permite invertir la señal de pulso cada vez que se ejecuta OB32 (cada segundo).

La instrucción CU := “Pulso”, hace que el contador incremente cada dos segundos.

Con la instrucción R := (“IEC_Counter_0_DB”.Q OR “Reset”), se reinicia el contador cada que se presiona “Reset” o el contador alcanza su valor de 1800 (3600 segundos); observe que se utilizó la salida Q de la base de datos “IEC_Counter_0_DB”.

PV := 1800, es el valor de precarga del contador.

CV => “Cuenta1” es donde se lleva la cuenta del contador.

“IEC_Counter_0_DB_1”.CTU(CU:=”IEC_Counter_0_DB”.Q, otro contador cuenta cada que pasa una hora.

CV=> “Horómetro”, lleva la cuenta de horas de funcionamiento del motor.

Las siguientes instrucciones llevan el tiempo parcial menor a una hora. En la instrucción más anidada “Cuenta1” se multiplica por 2 y por mil para tener milisegundos en formato entero. Luego, se convierte a doble entero y, finalmente, se lleva a formato de TIME.

“Parcial”:=DINT_TO_TIME(INT_TO_DINT(“Cuenta1”))*2*1000; // Calcular el tiempo.

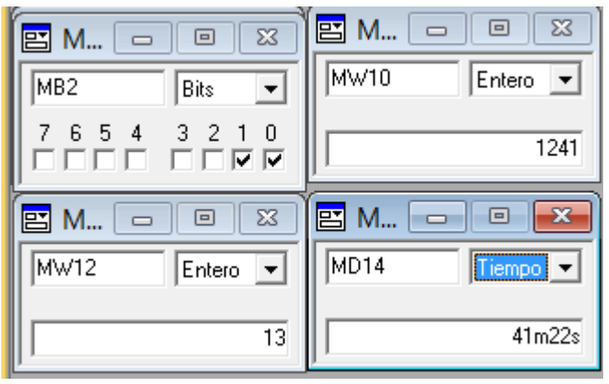
IF “Reset” THEN // parcial en milisegundos y se pasa a formato tiempo.

En la Figura 73 se puede ver el resultado de la simulación después de 13 horas de trabajo. Observe que la señal M2.0 está activa indicando que el motor está en funcionamiento, M2.1 es la señal que cambia de estado siempre que se ejecuta OB32, MW10 se incrementa en 1 cada 2 segundos, MW12 es el horómetro y MD14 corresponde al tiempo transcurrido de la última fracción de hora, incrementando cada 2 segundos. El tiempo en operación del motor es de 13h41m22s. Los 41m22s corresponden a la cuenta del tiempo parcial; multiplique por 2 el valor de MW10 para obtener los segundos equivalentes y convierta a minutos y segundos (divida por 60 para obtener minutos y el residuo lo multiplica por 60 para obtener la fracción en segundos). Si la CPU se detiene, el tiempo deja de contar y sigue en funcionamiento cuando se lleva nuevamente a RUN; lo mismo sucede con la señal de motor. Al activar M2.2 Reset tanto el contador como el horómetro y el registro de fracción de hora van a 0. Como se puede observar, este horómetro con contadores del PLC resultó algo más complejo que el realizado en un ejemplo anterior.

Aunque se han realizado varios ejemplos donde el programa principal queda incluido en el bloque de organización de interrupción, la recomendación es no cargarlo demasiado para no interferir con operaciones en otros bloques que también requieran de las interrupciones.

Figura 73.

Simulación del programa del problema 5.4



Fuente: elaboración propia.

06. ACCESO A LA INFORMACIÓN

6. Acceso a la información

Se profundizará ahora en diferentes formas de acceso a la información dentro de los bloques o entre ellos, refiriéndonos especialmente a los bloques de datos (DB), funciones (FC) y bloques de función (FB).

6.1. Bloques de datos

Aunque en el desarrollo de los temas anteriores se ha hecho uso de las bases de datos y algunos bloques especiales de programación, a continuación, se tendrá más detalle en su estructura, funcionamiento y uso, dando una mayor perspectiva de las variables que se manipulan. Luego de este capítulo, el lector se dará cuenta de que muchos de los ejemplos realizados pudieron tener otro tratamiento de variables para hacerlos más efectivos.

Los bloques de datos (DB) son áreas de memoria del PLC que sirven para manipular información que se genera en el proceso de ejecución de un programa. Se manejan dos tipos de datos: los globales y los de instancia. Los globales se pueden utilizar siempre que se hayan creado una base de datos y la respectiva variable. Los datos de instancia se generan automáticamente al configurar bloques de función (FB) que se explicarán más adelante.

A los datos globales se puede tener acceso desde cualquier parte del programa en forma absoluta, simbólica, indizada o estructurada.

Acceso absoluto. Se utiliza un identificador de operando y una dirección absoluta. El identificador de operando hace referencia al área de memoria y el tipo de dato; algunas áreas de memoria son las de imagen de las entradas, imagen de las salidas, marcas y bloques de datos, mientras que como ejemplos de tipos de datos se pueden mencionar el bit, el byte, la palabra y la doble palabra. Entonces, en el acceso absoluto de una base de datos se podrían tener los siguientes ejemplos:

%DB1.DBB8 hace referencia al byte 8 de la base de datos1.

%DB2.DBX10.3 hace referencia al bit 3 del byte 10 de la base de datos 2.

%DB2. DBW4 hace referencia a la palabra que empieza en el byte 4 y termina en el 5 de la base de datos 2.

%DB1.DBD2 hace referencia a la doble palabra que empieza en el byte 2 y termina en el 5 de la base de daros 1.

El signo “%” hace referencia a un identificador directo, utilizado cuando se quiere acceder a una dirección de memoria.

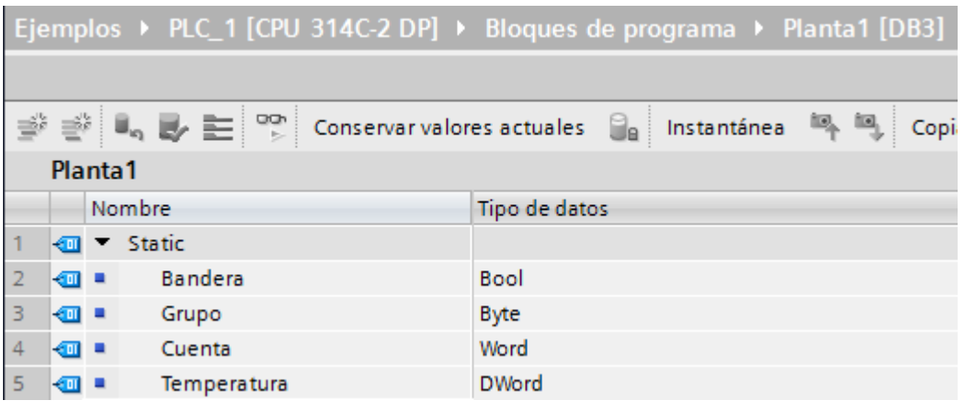
Acceso simbólico. Es un símbolo que identifica la variable de la dirección especificada; se presentan los siguientes ejemplos:

"Planta1".Temperatura donde "Planta1" corresponde al nombre de la base de datos y Temperatura es el nombre que se le dio a la variable.

En las siguientes imágenes se muestra una base de datos creada y la identificación de las variables en una tabla de observación donde se evidencian el direccionamiento absoluto y simbólico; véanse las Figuras 74 y 75.

Figura 74.

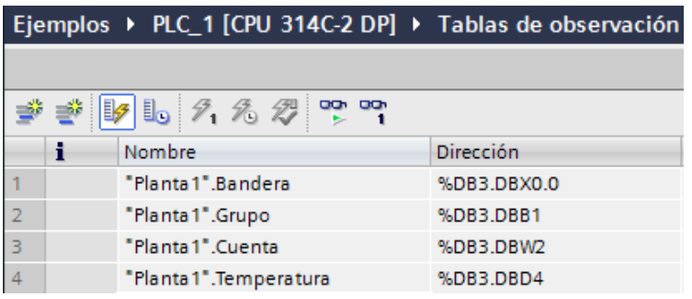
Ejemplo de una base de datos con las variables incluidas en ella



Fuente: elaboración propia.

Figura 75.

Variables de una base de datos vistas en una tabla de observación



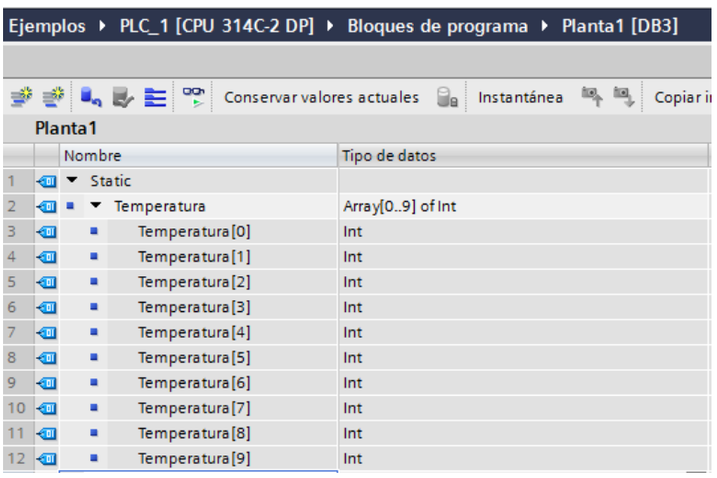
Fuente: elaboración propia.

Acceso indizado. Se logra mediante identificador de operando e índice de Array.

El Array es un tipo de dato donde se puede conformar un arreglo de un número determinado de variables que tienen un solo tipo de datos. En las Figuras 76 y 77 se muestran ejemplos de la creación de la base de datos con variables en un Array y la representación de las variables en la tabla de observación. Se aprecia que hay un espacio para el índice del Array. Como se puede ver es un arreglo de diez variables con el nombre de Temperatura y se distingue una de otra por un índice (elemento) que va de 0 a 9.

Figura 76.

Ejemplo de la definición de un Array en una base de datos



Fuente: elaboración propia.

Figura 77.

Tabla de observación mostrando las variables contenidas en un Array de una base de datos

Fuente: elaboración propia.

Los Array pueden ser unidimensionales, bidimensionales o de dimensión superior. En los Array unidimensionales, los elementos del Array van organizados en forma ascendente. Con los bidimensionales se puede conformar una tabla donde se distinguen las filas y las columnas, separados por coma. Los de orden superior hacen posible la ampliación de las dimensiones hasta un valor de 6. El índice de los Array puede ir de – 32 768 hasta 32 767; su límite se especifica con dos puntos y la dimensión con coma.

Ejemplos de Array:

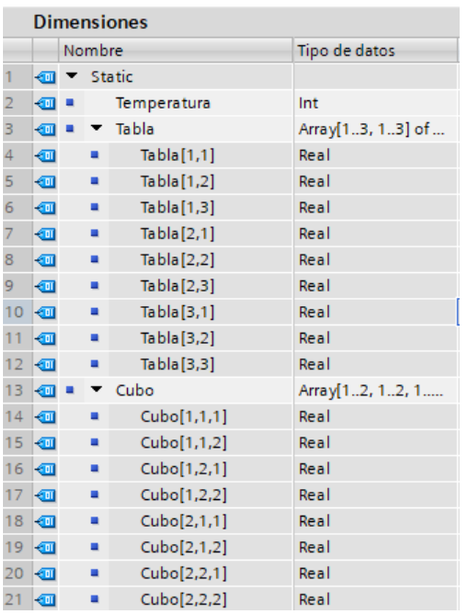
ARRAY[0..9] OF INT corresponde a un arreglo unidimensional de 10 elementos configurados como enteros.

ARRAY[1..50, 1..20] OF REAL corresponde a un arreglo bidimensional de 50 filas y 20 columnas en formato real.

En las Figuras 78 y 79 se muestran ejemplos de Array; se distingue el arreglo bidimensional en una variable llamada Tabla y uno tridimensional en un arreglo llamado Cubo.

Figura 78.

Ejemplo de un Array bidimensional llamado Tabla y otro tridimensional llamado Cubo



Dimensiones

	Nombre	Tipo de datos
1	Static	
2	Temperatura	Int
3	Tabla	Array[1..3, 1..3] of ...
4	Tabla[1,1]	Real
5	Tabla[1,2]	Real
6	Tabla[1,3]	Real
7	Tabla[2,1]	Real
8	Tabla[2,2]	Real
9	Tabla[2,3]	Real
10	Tabla[3,1]	Real
11	Tabla[3,2]	Real
12	Tabla[3,3]	Real
13	Cubo	Array[1..2, 1..2, 1.....
14	Cubo[1,1,1]	Real
15	Cubo[1,1,2]	Real
16	Cubo[1,2,1]	Real
17	Cubo[1,2,2]	Real
18	Cubo[2,1,1]	Real
19	Cubo[2,1,2]	Real
20	Cubo[2,2,1]	Real
21	Cubo[2,2,2]	Real

Fuente: elaboración propia.

Figura 79.

Tabla de observación que muestran un Array bidimensional y otro tridimensional

	Nombre	Dirección	Formato visualiza..
1	"Dimensiones".Temperatura	%DB3.DBW0	DEC+/-
2	"Dimensiones".Tabla[1, 1]	%DB3.DBD2	Número en coma...
3	"Dimensiones".Tabla[1, 2]	%DB3.DBD6	Número en coma...
4	"Dimensiones".Tabla[1, 3]	%DB3.DBD10	Número en coma...
5	"Dimensiones".Tabla[2, 1]	%DB3.DBD14	Número en coma...
6	"Dimensiones".Tabla[2, 2]	%DB3.DBD18	Número en coma...
7	"Dimensiones".Tabla[2, 3]	%DB3.DBD22	Número en coma...
8	"Dimensiones".Tabla[3, 1]	%DB3.DBD26	Número en coma...
9	"Dimensiones".Tabla[3, 2]	%DB3.DBD30	Número en coma...
10	"Dimensiones".Tabla[3, 3]	%DB3.DBD34	Número en coma...
11	"Dimensiones".Cubo[1, 1, 1]	%DB3.DBD38	Número en coma...
12	"Dimensiones".Cubo[1, 1, 2]	%DB3.DBD42	Número en coma...
13	"Dimensiones".Cubo[1, 2, 1]	%DB3.DBD46	Número en coma...
14	"Dimensiones".Cubo[1, 2, 2]	%DB3.DBD50	Número en coma...
15	"Dimensiones".Cubo[2, 1, 1]	%DB3.DBD54	Número en coma...
16	"Dimensiones".Cubo[2, 1, 2]	%DB3.DBD58	Número en coma...
17	"Dimensiones".Cubo[2, 2, 1]	%DB3.DBD62	Número en coma...
18	"Dimensiones".Cubo[2, 2, 2]	%DB3.DBD66	Número en coma...

Fuente: elaboración propia.

Problema 6.1. Una fábrica tiene tres plantas de producción; cada planta dispone de tres máquinas y a cada una se le debe hacer medición de temperatura, presión y nivel. Se tiene un sistema para que, siempre que se solicite, se inicie un proceso de almacenamiento de datos que se mandarán en forma secuencial cada 10 segundos.

Se supondrá que hay control para entrar los datos exactos por medio de una entrada digital (I124.0); en caso de dejar esta entrada siempre activa, el proceso de carga se reiniciará luego de guardarse la última variable, de la última máquina, de la tercera planta. El tiempo sugerido tiene como fin hacer un seguimiento a la simulación del proceso de carga y puede ser cargado según la necesidad. Se debe tener cuidado en caso de querer utilizar ciclos FOR o WHILE, puesto que, debido al tiempo de sincronización, se pueden provocar fallos relacionados con el desbordamiento de tiempo de ejecución de ciclo.

En la Tabla 38 se muestra parte de las variables para utilizar.

Tabla 38.

Variables para el problema 6.1

Nombre	Tipo de dato	Dirección	Observación
Habilitar	Bool	%I124.0	Habilita el proceso de almacenamiento de datos
Carga_Datos	Bool	%M2.0	Cada vez que se cumple el tiempo se almacena un nuevo dato
Detector_Flanco	Bool	%M2.1	Para detectar flancos positivos
Auxiliar_Flanco	Bool	%M2.2	Auxiliar flancos positivos
Dato_Nuevo	Real	%MD10	Dato para almacenar
Planta	Int	%MW14	Identificador de la planta 0, 1 o 2
Máquina	Int	%MW16	Identificador de la máquina 0, 1 o 2
Variable	Int	%MW18	Identificador de la variable 0, 1 o 2
T_Avance	Time	%MD20	Para observar el transcurso de tiempo entre cada lectura

Fuente: elaboración propia.

En la Tabla 39 se presenta la base de datos creada, que representa una matriz de tres por tres (3x3), para el almacenamiento de las demás variables. El índice a la derecha representa la variable, el del centro la máquina y el de la izquierda la planta.

Tabla 39.

Array tridimensional definido en la base de datos llamada Matriz

Base de datos				
Matriz [DB1]				
	Static Registros	Array[0..2, 0..2, 0..2] of Real		
1	Registros[0,0,0]	Real	Máquina1	Planta 0
2	Registros[0,0,1]	Real		
3	Registros[0,0,2]	Real		
4	Registros[0,1,0]	Real	Máquina2	
5	Registros[0,1,1]	Real		
6	Registros[0,1,2]	Real		
7	Registros[0,2,0]	Real	Máquina3	
8	Registros[0,2,1]	Real		
9	Registros[0,2,2]	Real		
10	Registros[1,0,0]	Real	Máquina1	Planta 1
11	Registros[1,0,1]	Real		
12	Registros[1,0,2]	Real		
13	Registros[1,1,0]	Real	Máquina2	
14	Registros[1,1,1]	Real		
15	Registros[1,1,2]	Real		
16	Registros[1,2,0]	Real	Máquina3	
17	Registros[1,2,1]	Real		
18	Registros[1,2,2]	Real		
19	Registros[2,0,0]	Real	Máquina1	Planta 2
20	Registros[2,0,1]	Real		
21	Registros[2,0,2]	Real		
22	Registros[2,1,0]	Real	Máquina2	
23	Registros[2,1,1]	Real		
24	Registros[2,1,2]	Real		
25	Registros[2,2,0]	Real	Máquina3	
26	Registros[2,2,1]	Real		
27	Registros[2,2,2]	Real		

Fuente: elaboración propia.

El programa sugerido en SCL es el siguiente:

```
IF "Habilitar" THEN

    "T_sincronismo".TON(IN := NOT "T_sincronismo".Q,
                        PT := T#10s,
                        Q => "Carga_Datos",
                        ET => "T_Avance");
  "Detector_Flanco" := "Carga_Datos" AND NOT "Auxiliar_Flanco";
  "Auxiliar_Flanco" := "Carga_Datos";

  IF "Carga_Datos" THEN
     "Matriz".Registros["Planta","Máquina","Variable"] := "Dato_Nuevo";
     "Variable" := "Variable" + 1;
```

```
    IF "Variable" >= 3 THEN
      "Variable" := 0;
        «Máquina» := «Máquina» + 1;
    ELSE
      GOTO Termina;
    END_IF;
    IF "Máquina" >= 3 THEN
        «Máquina» := 0;
      "Planta" := "Planta" + 1;
    ELSE
      GOTO Termina;
    END_IF;

              IF "Planta">=3 THEN
                  "Planta" := 0;
              ELSE
                  GOTO Termina;
              END_IF;

          END_IF;//

     END_IF;//
Termina:RETURN;
```

La actualización de datos se da siempre y cuando esté activo "Habilitar". Siendo así, cada 10 segundos se da un pulso para actualizar el próximo registro; esta actividad es controlada por el temporizador que se reinicia automáticamente. El detector de flancos hace que solo se actualicen datos con el flanco positivo de la salida del temporizador.

"Dato_Nuevo" es guardado en la posición de registros señalados por los índices "Planta", "Máquina" y "Variable". En la actualización de los índices se empieza con el incremento de "Variable", 1 a la vez, y se sale de la rutina por medio de las instrucciones GOTO y RETURN. Cuando este índice llega a 3 se posiciona en 0 y se sigue con el mismo proceso en los índices de "Máquina" y "Planta". Observe que la instrucción GOTO facilita que el programa no se quede esperando datos que demorarían en este caso 10 segundos, lo que provocaría un error del sistema.

6.2. Direccionamiento indirecto en SIMATIC S7-1200/1500

Se debe hacer uso de instrucciones especiales.

POKE_BOOL. Es una instrucción que permite escribir sobre un bit en un área de memoria que puede ser de una base de datos, entradas, salidas o marcas. Para esto, la instrucción dispone de un parámetro que es "AREA := 16#XX" (BYTE)

donde XX puede ser 81 para una entrada, 82 para una salida, 83 para una marca y 84 para una base de datos. Otro parámetro es “DBNUMBER := x” (DINT) donde x puede ser 0 para especificar que es una entrada, salida o marca; y 1 en adelante para especificar el número de una base de datos. El parámetro BYTEOFFSET (DINT) es la dirección donde se quiere escribir, tomándose los 16 bit menos significativos, y el parámetro BITOFFSET (INT) es el bit que se quiere modificar. Por último, VALUE (BOOL) es el bit que se quiere transferir.

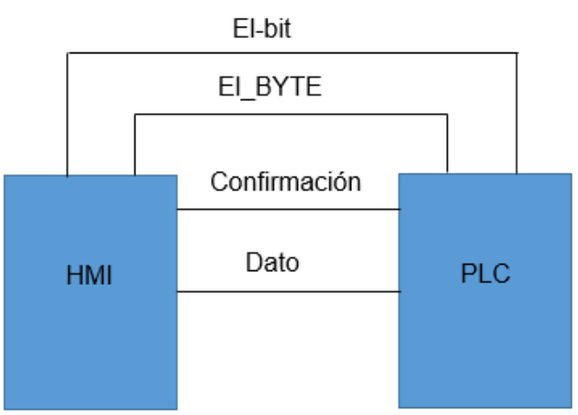
6.3. Área de memoria de base de datos

A continuación, se presenta una aplicación donde se procesa información en una base de datos que utiliza como una de sus instrucciones fundamentales “POKE_BOOL”.

Problema 6.2. En la Figura 80 se muestra una configuración de comunicación entre el PLC y una HMI donde se tiene un registro de alarmas de proceso. En una base de datos del PLC se tiene una variable de cuatro registros (Bytes) para almacenar el estado de 32 alarmas. Cada vez que en la pantalla se requiere almacenar una nueva alarma, da un flanco de subida en la señal “Confirmación” y pone el estado de la alarma correspondiente en la señal “Dato”; la posición de la alarma está determinada por las señales de “El_bit” y el “EL_BYTE”, respectivamente.

Figura 80.

Configuración de comunicación entre PLC y HMI del problema 6.2

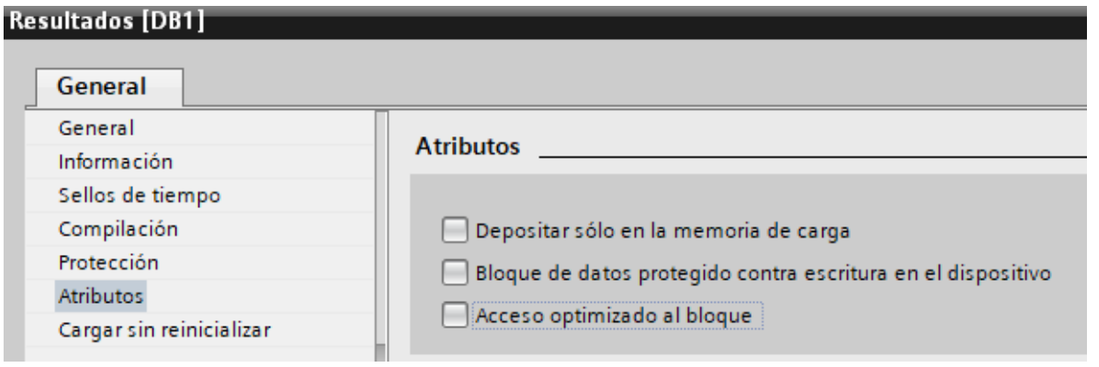


Fuente: elaboración propia.

Solución: se crea una base de datos que no tenga habilitada el área optimizada al bloque (Figura 81). Para esto puede acceder por propiedades cuando tenga la base de datos creada.

Figura 81.

Configuración de atributos de la base de datos 1 para permitir direccionamiento indirecto

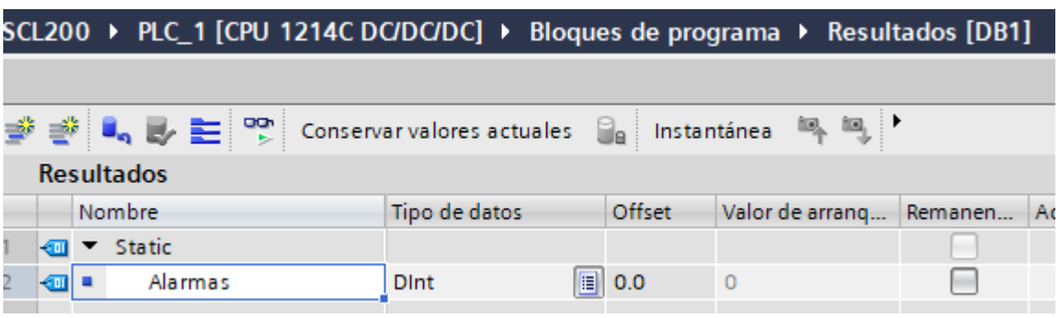


Fuente: elaboración propia.

En la base de datos se crean la variable o las variables para alarma; en este caso se crea una doble palabra para registrar las 32 alarmas a la vez. Si se desea, se puede hacer por bytes (declarando 4 registros de alarma) o por bits (declarando 32 alarmas por bits); (Figura 82).

Figura 82.

Variable de Alarmas creada en la base de datos 1 del problema 6.2



Fuente: elaboración propia.

En la Tabla 40 se observan las demás variables requeridas en el programa.

Tabla 40.

Otras variables definidas para el problema 6.2

Nombre	Tipo	Dirección	Comentario
Confirmación	Bool	%M2.0	Señal para cargar el dato
Dato	Bool	%M2.1	El dato para escribir en la base de datos
Detector flanco	Bool	%M2.2	Para cargar el dato con el flanco de subida
Aux_Flanco	Bool	%M2.3	Auxiliar
Area	Byte	%MB100	Escribir el Área en hexadecimal (puede ser directamente 16 #84 para base de datos)

Numero_DB	DInt	%MD102	Selecciona la base de datos (puede ser directamente 1, puesto que se trata de base de datos)
EL_BYTE	DInt	%MD08	Especifica el byte donde está la alarma (El byte de inicio)
El_bit	Int	%MW112	Especifica el bit de la alarma correspondiente

Fuente: elaboración propia.

El programa en SCL es el siguiente:

```
"Detector flanco" := "Confirmación" AND NOT "Aux_Flanco";
"Aux_Flanco" := "Confirmación";

IF "Detector flanco" THEN

    POKE_BOOL(area:="Area",
              dbNumber:="Numero_DB",
              byteOffset:="EL_BYTE",
              bitOffset:="El_bit",
              value:="Dato");
END_IF;
```

En la Figura 83 se muestra el resultado de la simulación en una tabla de observación. Se resalta en azul el byte 3, luego de que se activó el bit 5 debido a que se cumplen las siguientes condiciones: se tiene activo "Confirmación" para habilitar la carga de dato, se tiene seleccionada el área 16#84 que corresponde a la base de datos (se quiere acceder a registros de la base de datos), se seleccionó la base de datos 1, el Byte 3 y el bit 5 para cargarle el dato tipo BOOL disponible en "Dato", que en este caso es 1. La función utilizada es POKE_BOOL.

Figura 83.

Resultado de la simulación del problema 6.2

Nombre	Dirección	Formato visualiza..	Valor de observación	Valor de forzado
"Confirmación"	%M2.0	BOOL	TRUE	TRUE
"Dato"	%M2.1	BOOL	TRUE	TRUE
"Area"	%MB100	Hex	16#84	16#84
"EL_BYTE"	%MD108	DEC+/-	3	3
"El_bit"	%MW112	DEC+/-	5	5
"Numero_DB"	%MD102	DEC+/-	1	1
"Resultados".A..	%DB1.DBD0	Bin	2#0000_0000_0000_0000_0000_0000_0010_0000	

Fuente: elaboración propia.

6.4. Área de memoria de entradas, salidas y marcas

Se replantea ahora la aplicación del problema 6.2., accediendo a una memoria de marcas en lugar de una base de datos, se utiliza la misma instrucción "POKE_BOOL" pero la configuración de los parámetros es diferente. Luego, en otro problema se utiliza la instrucción "POKE" que posibilita manipular varios bits al mismo tiempo.

Problema 6.3. Supongamos que las alamas se están guardando en un área de marcas. Para esto se define la variable MD114 como doble entero “Alarmas”. El programa sería el mismo; solamente cambian los valores a forzar en la tabla de observación, teniendo en cuenta que el área de memoria es la 16#83 y la variable “Numero_DB” se pone en 0. La Figura 84 muestra la carga del bit 5 del byte 3 de la marca MD14; observe que MD108 se cargó con 117, que corresponde al tercer byte de la doble palabra MD114.

Figura 84.

Resultado de la simulación del problema 6.3

Nombre	Dirección	Formato visualiza..	Valor de observación	Valor de forzado
"Confirmación"	%M2.0	BOOL	TRUE	TRUE
"Dato"	%M2.1	BOOL	TRUE	TRUE
"Area"	%MB100	Hex	16#83	16#83
"EL_BYTE"	%MD108	DEC+/-	117	117
"El_bit"	%MW112	DEC+/-	5	5
"Numero_DB"	%MD102	DEC+/-	0	0
"Alarmas"	%MD114	Bin	2#0000_0000_0000_0000_0000_0000_0010_0000	

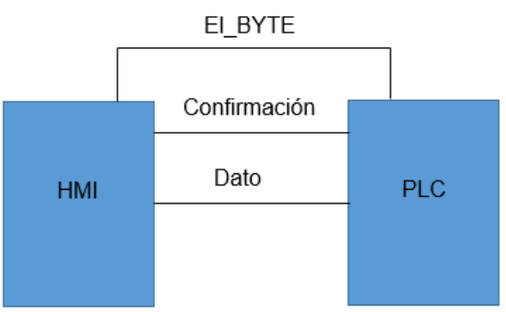
Fuente: elaboración propia.

POKE. Es una instrucción que permite escribir en una dirección de memoria y transferir varios bits a la vez; lo que cambia es el parámetro “VALUE” que se convierte en una secuencia de bits.

Problema 6.4. Modificando un poco el ejemplo anterior, se quiere mandar el estado de las alarmas (todas a la vez) hacia la base de datos; estas están depositadas en “Dato_Alarmas”, MD118 en formato DWORD. En la Figura 85 se muestra la nueva disposición del PLC y la HMI, puesto que ya no se requiere especificar el bit. En la Figura 86 se muestra que en la base de datos solo se requiere cambiar el formato de la variable “Alarmas” a DWORD.

Figura 85.

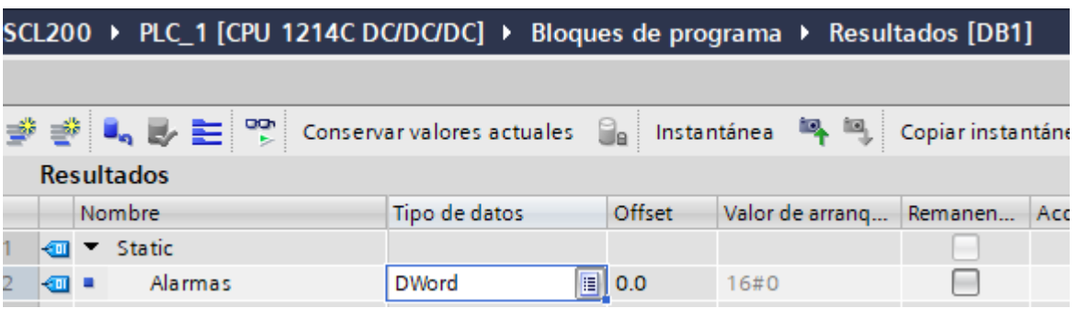
Configuración de la HMI y el PLC para el problema 6.4



Fuente: elaboración propia.

Figura 86.

Declaración de la variable "Alarmas" en la base de datos como DWord



Fuente: elaboración propia.

La Tabla 41 muestra las variables adicionales que se deben utilizar.

Tabla 41.

Variables adicionales para el problema 3.4

Nombre	Tipo	Dirección	Comentario
Confirmación	Bool	%M2.0	Señal para cargar el dato
Detector flanco	Bool	%M2.2	Para cargar el dato con el flanco de subida
Aux_Flanco	Bool	%M2.3	Auxiliar
Area	Byte	%MB100	Escribir el Área en hexadecimal (puede ser directamente 16 #84 para base de datos)
Numero_DB	DInt	%MD102	Selecciona la base de datos (puede ser directamente 1 puesto que se trata de base de datos)
EL_BYTE	DInt	%MD108	Especifica el byte donde está la alarma (el byte de inicio)
Dato_Alarmas	DWord	%MD118	Especifica el área de alarmas que se quiere transferir

Fuente: elaboración propia.

El programa en SCL es:

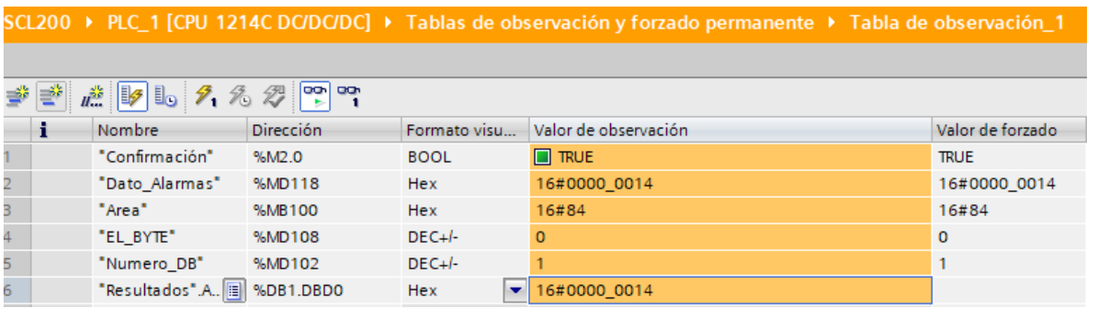
```
"Detector flanco" := "Confirmación" AND NOT "Aux_Flanco";
"Aux_Flanco" := "Confirmación";
IF "Detector flanco" THEN
      POKE(area:="Area",
        dbNumber:="Numero_DB",
        byteOffset:="EL_BYTE",
        value:="Dato_Alarmas");

END_IF;
```

Nota: es posible transferir un byte, dos bytes (WORD) o cuatro bytes (DWORD) a la vez.

El resultado de la simulación se muestra en la Figura 87 donde se observa que se transfirió el número 14 en hexadecimal de MD18 (Dato_Alarmas) a la base de datos en "Resultados".

Figura 87.

Resultado de la simulación del problema 6.4



Fuente: elaboración propia.

PEEK_BOOL. Es una instrucción que permite leer un bit de memoria desde un área que puede ser de una base de datos, entradas, salidas o marcas. Para esto, la instrucción dispone de un parámetro que es “AREA := 16#XX” (BYTE) donde XX puede ser 81 para una entrada, 82 para una salida, 83 para una marca y 84 para una base de datos. Otro parámetro es “DBNUMBER := x” (DINT) donde x puede ser 0 para especificar que es una entrada, salida o marca; y 1 en adelante para especificar el número de una base de datos. El parámetro BYTEOFFSET (DINT) es la dirección de donde se quiere leer, tomándose los 16 bits menos significativos y el parámetro BITOFFSET (INT) es el bit que se quiere leer. Por último, el resultado de la función se guarda en cualquier área de memoria en forma de bit.

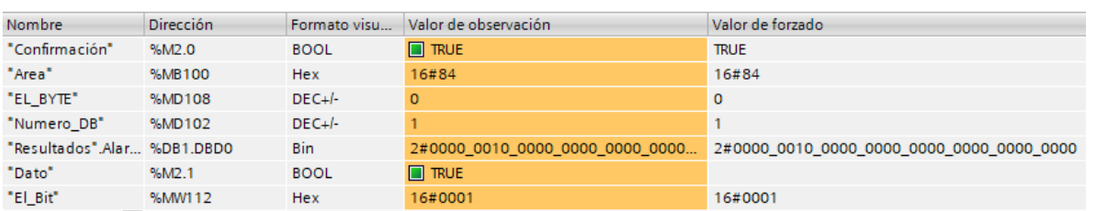
Problema 6.5. Se quiere cambiar el sentido de escritura de datos para pasar de un bit de la base de datos a la variable “Dato” M2.0. El programa en SCL se muestra a continuación:

```
“Detector flanco” := “Confirmación” AND NOT “Aux_Flanco”;
“Aux_Flanco” := “Confirmación”;

IF “Detector flanco” THEN

    “Dato” := PEEK_BOOL(area := “Area”, dbNumber := “Numero_DB”, byteOffset :=
“EL_BYTE”, bitOffset := “El_Bit”);
END_IF;
```

En la Figura 88 se muestra la simulación donde se transfiere el bit 1 del byte 0 de la base de datos 1 a la variable “Dato” M2.1.

Figura 88.

Simulación del problema 6.5



Fuente: elaboración propia.

PEEK. Es una instrucción que permite leer de una dirección de memoria y transferir varios bits a la vez; lo que cambia es el parámetro “VALUE” que consiste en una secuencia de bits.

Problema 6.6. Realice un programa para transferir información desde la base de datos hacia la Marca MD118.

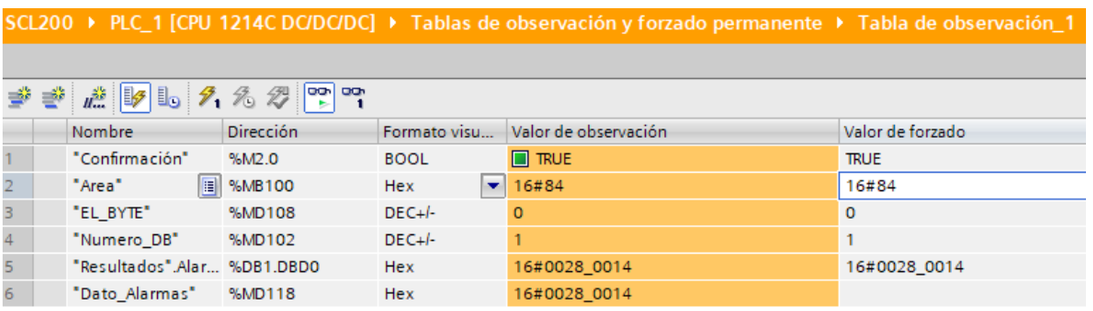
Para resolver el problema se debe tener en cuenta que a la instrucción PEEK se le agrega _DWORD para especificar el tipo de dato que se va a transferir.

```
“Detector flanco” := “Confirmación” AND NOT “Aux_Flanco”;
“Aux_Flanco” := “Confirmación”;
IF “Detector flanco” THEN
       “Dato_Alarmas” := PEEK_DWORD(area := “Area”, dbNumber := “Numero_
DB”, byteOffset := “EL_BYTE”);

END_IF;
```

En la Figura 89 se muestra cómo se transfirió el dato hexadecimal 16#0028_0014 disponible en la base de datos a la variable “Dato_Alarmas” en MD118.

Figura 89.

Resultado de simulación para el problema 6.5



Fuente: elaboración propia.

POKE_BLK. Es una instrucción para escribir en una zona de memoria, permitiendo transferir toda un área de memorias a otra; se pueden intercambiar datos entre memorias de entrada, salida, marcas o bases de datos. La instrucción POKE_BLK viene acompañada de los siguientes parámetros:

AREA_SRC := 16#XX (BYTE), especifica el área de origen donde XX puede ser 81 para una entrada, 82 para una salida, 83 para una marca y 84 para una base de datos.

DBNUMBER_SRC := x (DINT) donde x puede ser 0 para especificar que es una entrada, salida o marca y 1 en adelante para especificar el número de una base de datos.

BYTEOFFSET_SRC := b (DINT) donde b es el byte de la primera dirección de origen.

AREA_DEST := 16#XX (BYTE), especifica el área de destino.

DBNUMBER_DEST := x (DINT) donde x puede ser 0 para especificar que es una en-

trada, salida o marca y 1 en adelante para especificar el número de una base de datos.

BYTEOFFSET_DEST := b (DINT) donde b es el byte de la primera dirección de destino.

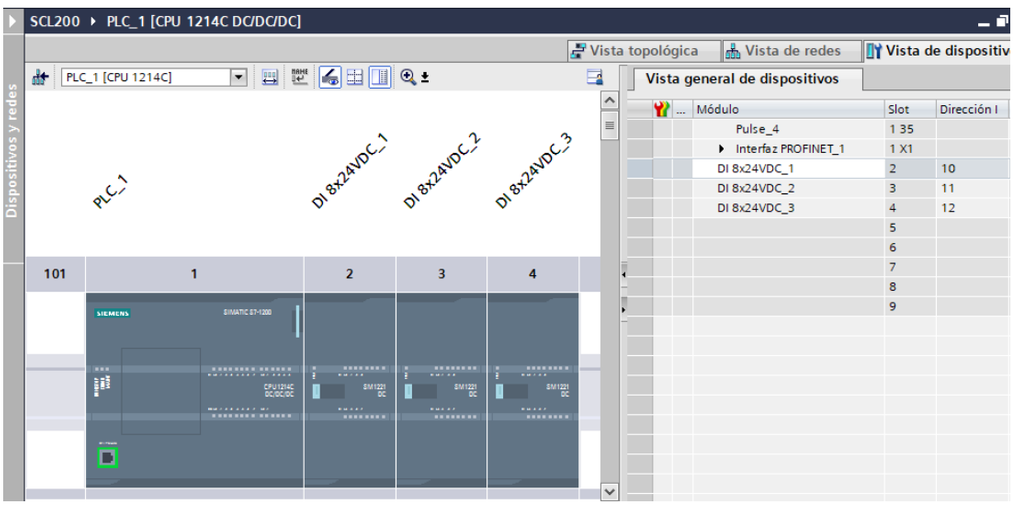
COUNT := n (DINT) donde n representa el número de bytes a transferir.

Problema 6.6. Se quiere transferir el estado de tres bytes de entrada, IB10, IB11, IB12 a áreas de memoria de marca que inician en los bytes 10, 20 o 30, el cual es fijado por una variable que se llama “Selector”.

El PLC 1200 deberá tener configurados tres módulos adicionales de entradas digitales de 8 bits cada uno, teniendo configuradas las direcciones 10, 11 y 12, como se muestra en la Figura 90.

Figura 90.

Configuración del hardware del PLC para el problema 6.6



Fuente: elaboración propia.

En Tabla 42 se especifican las variables para utilizar; observe que hay un área de la fuente de datos y tres áreas posibles para la transferencia de datos.

Tabla 42.

Variables para plantear el algoritmo del problema 6.6

Nombre	Tipo de dato	Dirección	Comentario
Entrada_0	Byte	%IB10	Fuente de datos
Entrada_1	Byte	%IB11	
Entrada_2	Byte	%IB12	
Marca_1_1	Byte	%MB10	Área de destino de datos 1
Marca_1_2	Byte	%MB11	
Marca_1_3	Byte	%MB12	
Marca_2_1	Byte	%MB20	Área de destino de datos 2
Marca_2_2	Byte	%MB21	
Marca_2_3	Byte	%MB22	

Marca_3_1	Byte	%MB30	Área de destino de datos 3
Marca_3_2	Byte	%MB31	
Marca_3_3	Byte	%MB32	
Selector	DInt	%MD2	Fija el inicio de la dirección destino

Fuente: elaboración propia.

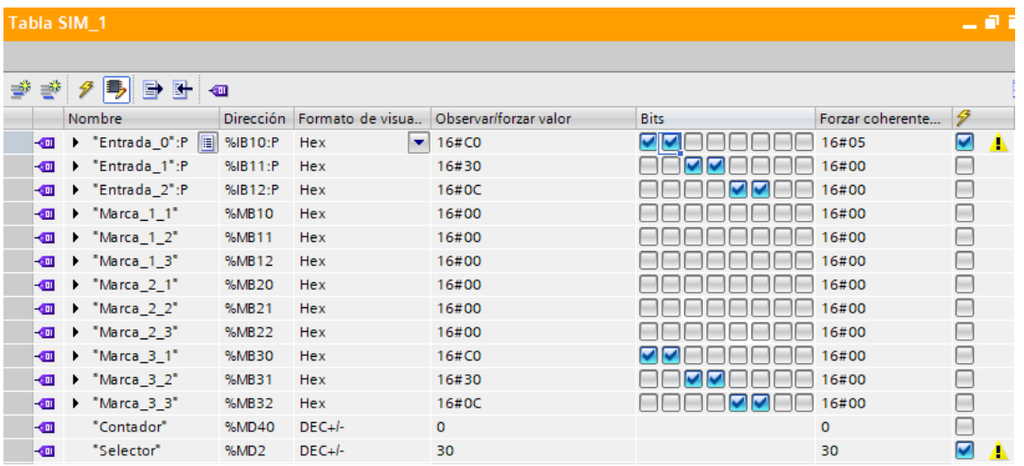
El programa en SCL es:

```
POKE_BLK(area_src:=16#81,
        dbNumber_src:=0, // porque no se utiliza base de datos para el origen
         byteOffset_src:=10, // Dirección de inicio de los bytes de entrada
         area_dest:=16#83, // el área de origen son entradas
        dbNumber_dest:=0, // porque no se utiliza base de datos para el destino
         byteOffset_dest:="Selector", // permite seleccionar el área de destino

         count:=3); // Solamente se van a transferir 3 bytes (podrían ser 50)
```

Luego de hacer la simulación en una tabla se obtienen los resultados de la Figura 91. Se observa que el selector apunta a la dirección 30. Entonces, los datos en las entradas digitales se transfieren a las direcciones MB30, MB31 y MB32.

Figura 91.

Simulación resultante para el problema 6.6



Fuente: elaboración propia.

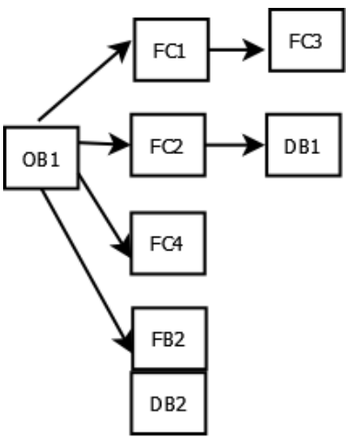
6.5. Bloques FC y FB

Hasta el momento, gran cantidad ejemplos presentados se ha limitado a tener un programa en FC1, el cual es llamado desde el OB1 y así se ejecuta el programa. La información generada se ha guardado en memoria de marcas (M) o base de datos (DB), pero, por supuesto, también se ha hecho uso de las áreas de memoria de entradas (I) y salidas (Q). Ahora nos concentramos en dos aspectos nuevos: el primero tiene que ver con realizar una programación por subrutinas y el segundo utilizando la memoria local de los bloques FC y FB.

La programación por subrutinas facilita dividir el programa en funciones específicas que se pueden utilizar según convenga al programador; esto permite tener un programa más modular y fácil de seguir. Para mejor entendimiento se hace referencia a la Figura 92 donde se muestra la dinámica de llamadas de las diferentes subrutinas.

Figura 92.

Programación por subrutinas



Fuente: elaboración propia.

En cada llamada de un bloque se genera una información que debe ser guardada en una de las áreas de memoria ya mencionadas, o guardada en forma local para la próxima llamada del bloque o, simplemente, utilizada en forma temporal por el bloque y, una vez se sale de él, esa información se pierde.

Este tipo de información se conoce con el nombre de datos locales y frecuentemente se citará como parámetros; se reconocerán porque sus variables estarán precedidas por el signo número “#”, a diferencia de las variables comunes que, como se ha podido observar, van dentro de comillas (“ “).

Los datos locales pueden ser estáticos, es decir, valores que se conservan en todos los recorridos de un bloque de función; y temporales, que no ocupan área de memoria y solo conservan su valor en un recorrido del bloque. Los parámetros de un bloque creado pueden ser transferidos o copiados de otras áreas de memoria, pudiendo ser de entrada, salida o entrada/salida.

A los FC no se les pueden asignar parámetros de tipo estático, mientras que los FB sí los tienen y por eso requieren de una base de datos de instancia (datos que solo pueden ser cambiados en el propio bloque). Siempre que se cree un FB se tendrá que asociar una base de datos.

6.6. Bloques (FC)

Se usan para funciones simples; tienen la posibilidad de usar parámetros que permiten la reutilización de códigos como bloques compactos, es decir, la función por realizar queda dentro del bloque, al cual solo hace falta asociarle las entradas y salidas que requiere para su funcionamiento. Para complementar, se puede decir que las FC programadas con parámetros se asemejan a circuitos electrónicos integrados donde todos aquellos de una misma referencia realizan la misma función, solo se requiere conectarles las entradas y salidas. Desde el punto de vista de programación se puede asociar con un objeto del cual se le pueden transferir las propiedades a otro objeto, pero que van a utilizar diferentes entradas y salidas.

6.7. Bloques de función (FB)

También permiten usar parámetros para la reutilización del código, pero se adiciona una base de datos en la cual se puede guardar gran cantidad de información propia del bloque y que se debe conservar entre llamadas de bloques. Se propondrán algunos problemas que faciliten el entendimiento de lo planteado.

Problema 6.7. Diseñe en una plantilla el arranque simple de un motor, que incluya la marcha, el paro (NC), una alarma y la salida para un motor. Luego, invoque esta función dos veces para controlar dos máquinas diferentes.

Por facilidad se presentará el código fuente de la plantilla creada en FC1; se observa cómo se crearon dos variables de entrada (locales), Arrancar : Bool y Parar : Bool; una variable de salida, Informar : Bool, y una de entrada salida, Activar : Bool. Note que las variables están acompañadas del signo #. Ahora, el programa deja ver dos partes: en la parte superior se presenta el área donde se definen las variables locales y en la parte inferior, luego de “Begin”, aparece el programa”.

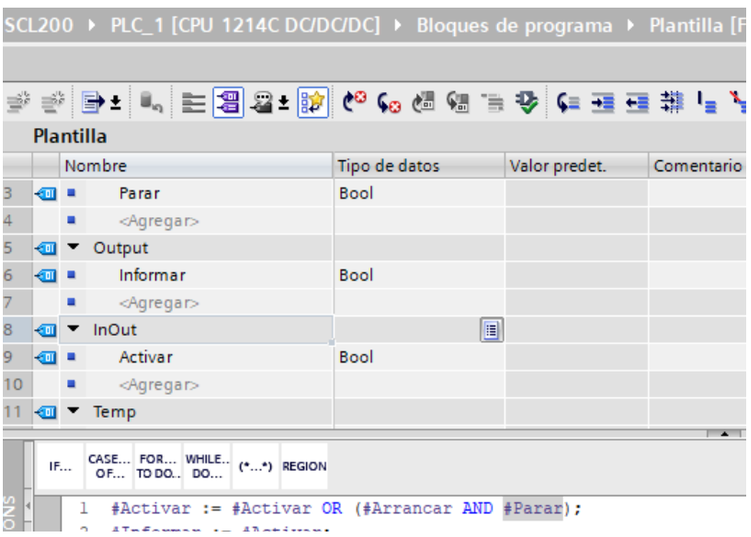
```
FUNCTION “Plantilla” : Void
{ S7_Optimized_Access := ‘TRUE’ }
VERSION : 0.1
   VAR_INPUT
      Arrancar : Bool;
      Parar : Bool;
   END_VAR

   VAR_OUTPUT
      Informar : Bool;
   END_VAR

   VAR_IN_OUT
      Activar : Bool;
   END_VAR
```

```
BEGIN
    #Activar := #Activar OR (#Arrancar AND #Parar);
    #Informar := #Activar;

END_FUNCTION
```

Una forma alternativa de presentar el programa se muestra en la Figura 93. Se observa en la parte superior del FC1 creado, llamado “Plantilla”, un área de parámetros donde se definen las variables locales mencionadas. En la parte inferior, aparece el programa del bloque que consiste básicamente en un sistema de marcha paro, definido con anterioridad. Se debe tener en cuenta que hasta el momento no se ha asociado ninguna entrada ni salida física del PLC; tampoco se ha hecho referencia a la memoria de marcas.

Figura 93.

Forma alternativa de presentar la declaración de variables locales y programa



Fuente: elaboración propia.

Para hacer efectivo el programa se crean las variables que se muestran en la Tabla 43; en ella se encuentran los elementos necesarios para activar dos máquinas en forma independiente.

Tabla 43.

Variables para desarrollo del problema 6.7

Nombre	Tipo	Dirección	Comentario
Marcha1	Bool	%I0.0	Arrancar motor 1
Paro1	Bool	%I0.1	Parar motor 1
Alarma1	Bool	%Q0.0	Alarma del motor 1
R_Motor1	Bool	%Q0.1	Salida motor 1
Marcha2	Bool	%I0.2	Arrancar motor 2

Paro2	Bool	%I0.3	Parar motor 2
Alarma2	Bool	%Q0.2	Alarma del motor 2
R_Motor2	Bool	%Q0.3	Salida motor 2

Fuente: elaboración propia.

La plantilla del FC1 será invocada dos veces desde otro bloque FC2 (llamado "Agrupar"), programado en SCL, donde es arrastrada FC1. El código para ingresar o completar es:

```
"Plantilla"(Arrancar:="Marcha1",
            Parar:="Paro1",
            Informar=>"Alarma1",
            Activar:="R_Motor1");
"Plantilla"(Arrancar:="Marcha2",
            Parar:="Paro2",
            Informar=>"Alarma2",
            Activar:="R_Motor2");
```

Observe que la plantilla puede ser invocada (llamada) las veces que se requiera; solo se tiene que realizar un programa. El resultado de la simulación se muestra en la Figura 94 donde se aprecia el funcionamiento de las dos máquinas luego de dar marcha en cada una.

Figura 94.

Simulación del programa para el problema 6.7

Nombre	Dirección	Format...	Observar/forzar valor	Bits	Forzar coherente...
"Marcha1":P	%I0.0:P	Bool	FALSE	☐	TRUE
"Paro1":P	%I0.1:P	Bool	TRUE	☑	TRUE
"Marcha2":P	%I0.2:P	Bool	FALSE	☐	TRUE
"Paro2":P	%I0.3:P	Bool	TRUE	☑	TRUE
"Alarma1"	%Q0.0	Bool	TRUE	☑	FALSE
"R_Motor1"	%Q0.1	Bool	TRUE	☑	FALSE
"Alarma2"	%Q0.2	Bool	TRUE	☑	FALSE
"R_Motor2"	%Q0.3	Bool	TRUE	☑	FALSE

Fuente: elaboración propia.

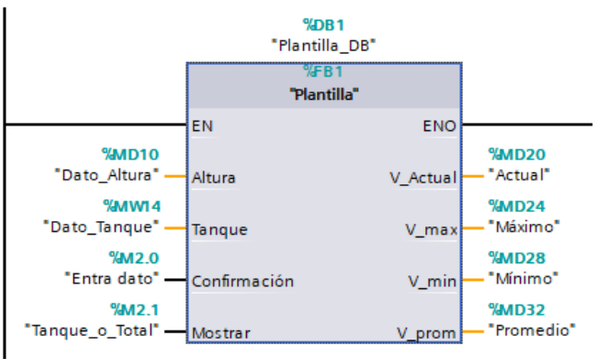
Problema 6.8. Suponga que en una empresa tienen "n" tanques idénticos provistos con sensores de altura para medir el nivel. Se pide realizar un algoritmo que esté en capacidad de informar el valor actual, el valor máximo, el valor mínimo y el valor promedio del volumen que se ha registrado en cada tanque o de los valores acumulados.

Como entradas físicas del sistema se tienen: un botón para confirmar la entrada de un dato nuevo de altura (M2.0), una variable que indica el tanque que corresponde al dato que se está entrando (MW14), una variable donde se deposita la altura (MD10) y un suiche para seleccionar lo que se quiere ver a la salida (M2.1). La capacidad de registro de lecturas tomadas para cada tanque es "5", en este caso. Como salidas del sistema se tienen: lectura actual (MD20), valor máximo de las últimas cinco lecturas (MD24), valor mínimo (MD28) y valor promedio (MD32).

La Figura 95 ilustra la presentación de la plantilla para las entradas y salidas del sistema; dentro del bloque está ubicado el algoritmo que se solicita.

Figura 95.

Plantilla de entradas y salidas del sistema del problema 6.8



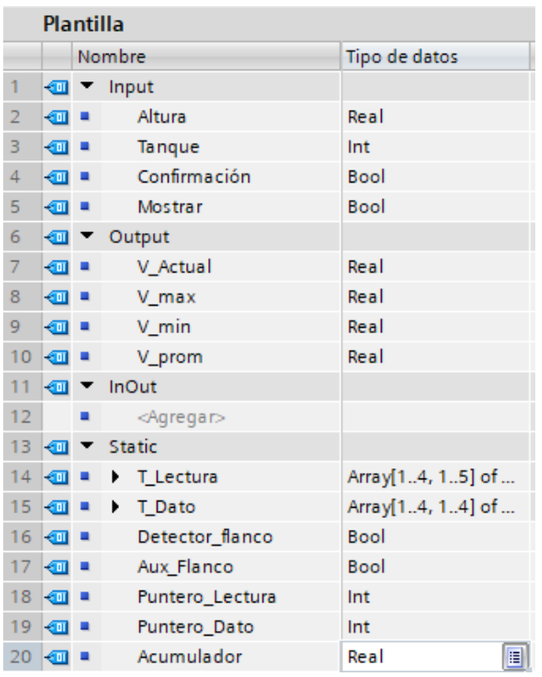
Fuente: elaboración propia.

Por facilidad, para hacer seguimiento al algoritmo se supondrá que solo son tres tanques; sin embargo, esto no afecta la estructura del programa. También se considera que por cada tanque se almacenarán los últimos cinco datos. Cada vez que se dé un pulso positivo en “Entrada dato”, el valor de “Dato_Altura” será procesado para el tanque indicado en “Dato_Tanque”, significando que todos los valores actuales, máximos, mínimos y promedios, serán actualizados para ese tanque. Lo mismo ocurre para los valores acumulados; los datos de los otros dos tanques quedan intactos. De acuerdo con la posición del suiche “Tanque_o_Total”, los datos de salida del bloque corresponderán a un tanque individual o al acumulado; con un 0 se muestran los datos individuales y con un 1 se muestran los datos totales. Considere que cada tanque es de tipo cilíndrico, colocado en forma vertical y con un radio de dos metros.

En esta ocasión todo el programa se realizará en un FB y se trabajará preferiblemente con parámetros (datos locales) que solo se utilizarán en el interior del bloque. El FB elaborado tiene el nombre de “Plantilla”. Dentro del FB se definirán las variables de tipo entrada, salida o estática; se incluyen variables estáticas que se usarán como índices, acumuladores o para detectar flancos, tal como se muestra en la Figura 96. Las variables estáticas se conservan entre llamadas de bloques.

Figura 96.

Parámetros definidos en el FB "Plantilla"



Plantilla

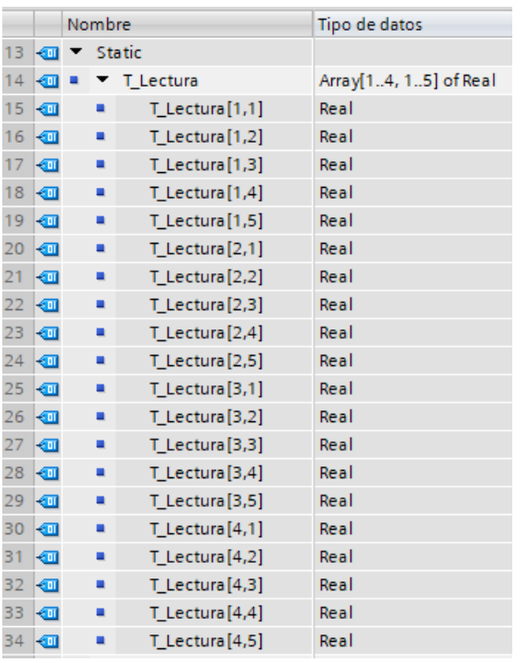
		Nombre	Tipo de datos
1		Input	
2		Altura	Real
3		Tanque	Int
4		Confirmación	Bool
5		Mostrar	Bool
6		Output	
7		V_Actual	Real
8		V_max	Real
9		V_min	Real
10		V_prom	Real
11		InOut	
12		<Agregar>	
13		Static	
14		T_Lectura	Array[1..4, 1..5] of ...
15		T_Dato	Array[1..4, 1..4] of ...
16		Detector_flanco	Bool
17		Aux_Flanco	Bool
18		Puntero_Lectura	Int
19		Puntero_Dato	Int
20		Acumulador	Real

Fuente: elaboración propia.

En la Figura 97 se muestra el despliegue del Array "T_Lectura" donde se conforma una matriz bidimensional de los tanques y las lecturas de los últimos cinco volúmenes procesados; debería ser de 3x5, pero se realiza de 4x5 para aprovechar la última fila para depositar los datos acumulados de los tres tanques. De T_Lectura [1,1] a T_Lectura [1,5] se guardan los cinco datos correspondientes al tanque 1. De T_Lectura [2,1] a T_Lectura [2,5] se guardan los cinco datos correspondientes al tanque 2. De T_Lectura [3,1] a T_Lectura [3,5] se guardan los cinco datos correspondientes al tanque 3. De T_Lectura [4,1] a T_Lectura [4,5] se guardan los cinco datos correspondientes a los valores acumulados.

Figura 97.

Despliegue del Array "T_Lectura"



	Nombre	Tipo de datos
13	Static	
14	T_Lectura	Array[1..4, 1..5] of Real
15	T_Lectura[1,1]	Real
16	T_Lectura[1,2]	Real
17	T_Lectura[1,3]	Real
18	T_Lectura[1,4]	Real
19	T_Lectura[1,5]	Real
20	T_Lectura[2,1]	Real
21	T_Lectura[2,2]	Real
22	T_Lectura[2,3]	Real
23	T_Lectura[2,4]	Real
24	T_Lectura[2,5]	Real
25	T_Lectura[3,1]	Real
26	T_Lectura[3,2]	Real
27	T_Lectura[3,3]	Real
28	T_Lectura[3,4]	Real
29	T_Lectura[3,5]	Real
30	T_Lectura[4,1]	Real
31	T_Lectura[4,2]	Real
32	T_Lectura[4,3]	Real
33	T_Lectura[4,4]	Real
34	T_Lectura[4,5]	Real

Fuente: elaboración propia.

En la Tabla 44 se muestra una interpretación de la matriz de la Figura 97. Los datos corresponden a volúmenes.

Tabla 44.

Distribución de los últimos cinco volúmenes calculados para cada tanque y el acumulado

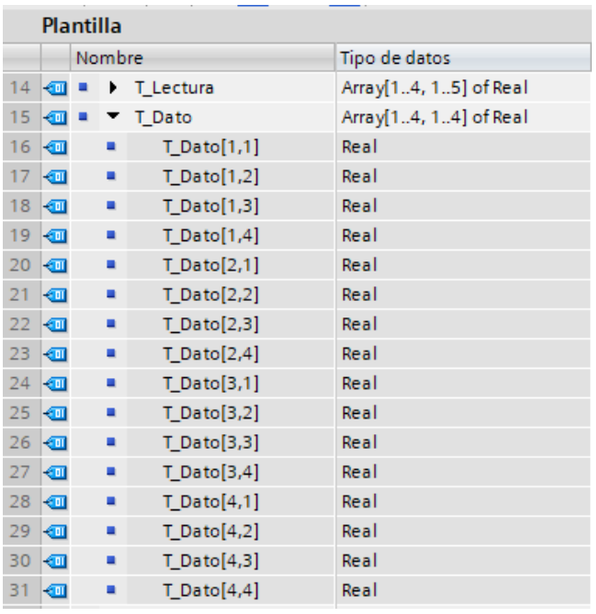
		Columna				
		1	2	3	4	5
Fila	1	Dato más nuevo tanque 1	Dato tanque 1	Dato tanque 1	Dato tanque 1	Dato más antiguo tanque 1
	2	Dato más nuevo tanque 2	Dato tanque 2	Dato tanque 2	Dato tanque 2	Dato más antiguo tanque 2
	3	Dato más nuevo tanque 3	Dato tanque 3	Dato tanque 3	Dato tanque 3	Dato más antiguo tanque 3
	4	Dato más nuevo acumulado	Dato acumulado	Dato acumulado	Dato acumulado	Dato más antiguo acumulado

Fuente: elaboración propia.

También se conforma una matriz "T_Dato" donde se depositan los datos calculados: valores actuales, valores máximos, valores mínimos, valores promedios; debería ser de 3x3 pero, igual que en el caso anterior, se adiciona una fila para depositar los valores correspondientes a los valores totales (Figura 98).

Figura 98.

Despliegue del Array "T_Dato"



Fuente: elaboración propia.

En la Tabla 45 se muestra la interpretación de la Figura 98 para depositar los datos calculados que se llevarán a la salida de la plantilla de valor actual, máximo, mínimo y promedio para cada tanque y el total.

Tabla 45.

Interpretación de la matriz "T_Dato" para el problema 6.8

		Columna			
		1	2	3	4
Fila	1	Volumen actual tanque 1	Volumen máximo tanque 1	Volumen mínimo tanque 1	Volumen promedio tanque 1
	2	Volumen actual tanque 2	Volumen máximo tanque 2	Volumen mínimo tanque 2	Volumen promedio tanque 2
	3	Volumen actual tanque 3	Volumen máximo tanque 3	Volumen mínimo tanque 3	Volumen promedio tanque 3
	4	Volumen actual total	Volumen máximo total	Volumen mínimo total	Volumen promedio total

Fuente: elaboración propia.

En la Tabla 46 se detallan las demás variables a utilizar.

Tabla 46.

Variables para utilizar en el problema 6.8

Nombre	**Tipo**	**Dirección**	**Observación**
Entra dato	Bool	%M2.0	Validar las entradas de las alturas
Dato_Altura	Real	%MD10	Altura del tanque
Dato_Tanque	Int	%MW14	Número del tanque
Tanque_o_Total	Bool	%M2.1	Selector del dato que se quiere ver
Actual	Real	%MD20	Valor de salida del volumen

Máximo	Real	%MD24	Volumen máximo a la salida
Mínimo	Real	%MD28	Volumen mínimo a la salida
Promedio	Real	%MD32	Volumen mínimo a la salida

Fuente: elaboración propia.

A continuación, se muestra en las Figuras 99, 100, 101, 102 y 103 la apariencia del algoritmo realizado en FB. Se incluyen los comentarios que dan más claridad.

Figura 99.

Rutina para actualizar las nuevas alturas del problema 6.8

```
//Cuando se solicita la carga  de una nueva altura para un tanque,
//priemro se respaldan los valores anteriores
// Son 5 valores, el dato más antiguo se descarta
#Detector_flanco := #Confirmación AND NOT #Aux_Flanco;
#Aux_Flanco := #Confirmación;

IF #Detector_flanco THEN

    FOR #Puntero_Lectura := 4 TO 1 BY -1 DO

        #T_Lectura[#Tanque, (#Puntero_Lectura + 1)] := #T_Lectura[#Tanque, (#Puntero_Lectura)];
    END_FOR;
    //Con los datos respaldados se actualiza el valor más nuevo
    #T_Lectura[#Tanque, (#Puntero_Lectura + 1)] := #Altura*3.1416*4;

    // Ahora se actualiza el volumen total (Se considera la ultima fila, la 4)
    // Priemro se respaldan los valores anteriores de la ultima fila
    // La última fila se reservó para almacenar los datos totales

    FOR #Puntero_Lectura := 4 TO 1 BY -1  DO
        #T_Lectura[4, (#Puntero_Lectura + 1)] := #T_Lectura[4, (#Puntero_Lectura)];
    END_FOR;
    // Luego se actualiza el valor total, sumando el vólumen nuevo de todos los tanques

    #T_Lectura[4, 1] := 0; // En esta dirección está el vólumen total
```

Fuente: elaboración propia.

Figura 100.

Rutina que incluye el cálculo del valor máximo del problema 6.8

```
    // el cual se borra cada que se actualiza una altura
    // Ahora se halla el nuevo vólumen total
    FOR #Puntero_Lectura :=1 TO 3 DO // Lectura 3 tanques, se aprovecha el mismo puntero
        #T_Lectura[4, 1] := #T_Lectura[4, 1] + #T_Lectura[#Puntero_Lectura, 1];
        //Se suma la primera columna (valor actual) de cada fila (tanque)
    END_FOR;
    // Ahora se actualizará la tabla de datos (Valor actual, Valor máximo
    // Valor mínimo y Valor promedio

    //Valor actual, solo se requiere mover de una tabla a otra
    //Los valores a mover están en la primera columna de la tabla de origen

    FOR #Puntero_Lectura := 1 TO 4 DO // En la fila 4 están los totales
        #T_Dato[#Puntero_Lectura, 1] := #T_Lectura[#Puntero_Lectura, 1];
    END_FOR;

    //Valor máximo, se guardan  en la segunda columna de Datos
    // Se requiere otro puntero llamado puntero dato
    //
    FOR #Puntero_Dato:= 1 TO 4 DO // La tabla de datos tiene 4 filas
        //se trabaja con la segunda columna de la tabla destino (Valor máximo)
        //
        #T_Dato[#Puntero_Dato, 2] := 0;
        // Se inicializa en  para comparar
        FOR #Puntero_Lectura:= 1 TO 5 DO // 5 comparaciones
```

Fuente: elaboración propia.

Figura 101.

Rutina para encontrar valor mínimo del problema 6.8

```
            IF #T_Dato[#Puntero_Dato, 2]< #T_Lectura[#Puntero_Dato,#Puntero_Lectura] THEN
               // Se va guadando el dato mayor de las comparaciones
                #T_Dato[#Puntero_Dato, 2] := #T_Lectura[#Puntero_Dato, #Puntero_Lectura];
            END_IF;
            // Se repite 5 veces
        END_FOR;
        //Luego se sigue con la siguinete fila, tnaque para encontrar el máximo

    END_FOR;
    // Se terminó con la actualización del valor máximo



    //Valor mínimo, se guardan  en la tercera columna de Datos
    // Se requiere el puntero llamado puntero dato
    //
    FOR #Puntero_Dato := 1 TO 4 DO // La tabla de datos tiene 4 filas
        //se trabaja con la tercera columna de la tabla destino (Valor mínimo)
        //
        #T_Dato[#Puntero_Dato, 3] := #T_Lectura[#Puntero_Dato, 1];
        // Se inicializa con primer valor para comparar
        FOR #Puntero_Lectura := 1 TO 5 DO // 5 comparaciones
            IF #T_Dato[#Puntero_Dato, 3] > #T_Lectura[#Puntero_Dato, #Puntero_Lectura] THEN
                // Se va guadando el dato mayor de las comparaciones
                #T_Dato[#Puntero_Dato, 3] := #T_Lectura[#Puntero_Dato, #Puntero_Lectura];
```

Fuente: elaboración propia.

Figura 102.

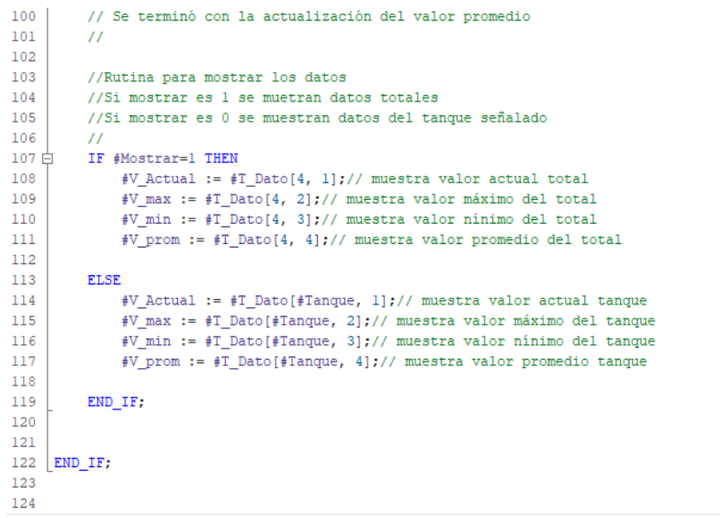
Rutina para encontrar valor promedio del problema 6.8

```
            END_IF;
            // Se repite 5 veces
        END_FOR;
        //Luego se sigue con la siguinete fila, tanque para encontrar el mínimo

    END_FOR;
    // Se terminó con la actualización del valor mínimo



    //Valor promedio, se guardan  en la cuarta columna de Datos
    // Se requiere el puntero llamado puntero dato
    //
    FOR #Puntero_Dato := 1 TO 4 DO // La tabla de datos tiene 4 filas
        //se trabaja con la cuarta columna de la tabla destino (Valor promedioo)
        //
        #Acumulador := 0;
        // Se inicializa suma
        FOR #Puntero_Lectura := 1 TO 5 DO // 5 comparaciones
            #Acumulador := #Acumulador + #T_Lectura[#Puntero_Dato, #Puntero_Lectura];
        END_FOR;
                //Luego se sigue con la siguinete fila, tanque para encontrar el mínimo
        #T_Dato[#Puntero_Dato, 4] := #Acumulador / 5;
    END_FOR;
    // Se terminó con la actualización del valor promedio
```

Fuente: elaboración propia.

Figura 103.

Rutina para mostrar datos



```
    // Se terminó con la actualización del valor promedio
    //

    //Rutina para mostrar los datos
    //Si mostrar es 1 se muetran datos totales
    //Si mostrar es 0 se muestran datos del tanque señalado
    //
    IF #Mostrar=1 THEN
        #V_Actual := #T_Dato[4, 1];// muestra valor actual total
        #V_max := #T_Dato[4, 2];// muestra valor máximo del total
        #V_min := #T_Dato[4, 3];// muestra valor nínimo del total
        #V_prom := #T_Dato[4, 4];// muestra valor promedio del total

    ELSE
        #V_Actual := #T_Dato[#Tanque, 1];// muestra valor actual tanque
        #V_max := #T_Dato[#Tanque, 2];// muestra valor máximo del tanque
        #V_min := #T_Dato[#Tanque, 3];// muestra valor nínimo del tanque
        #V_prom := #T_Dato[#Tanque, 4];// muestra valor promedio tanque

    END_IF;


END_IF;
```

Fuente: elaboración propia.

El algoritmo en SCL es el siguiente:

```
//Cuando se solicita la carga  de una nueva altura para un tanque,
//primero se respaldan los valores anteriores
// Son 5 valores, el dato más antiguo se descarta
#Detector_flanco := #Confirmación AND NOT #Aux_Flanco;
#Aux_Flanco := #Confirmación;

IF #Detector_flanco THEN

    FOR #Puntero_Lectura := 4 TO 1 BY -1 DO

        #T_Lectura[#Tanque, (#Puntero_Lectura + 1)] := #T_Lectura[#Tanque,
(#Puntero_Lectura)];
    END_FOR;
    //Con los datos respaldados se actualiza el valor más nuevo
    #T_Lectura[#Tanque, (#Puntero_Lectura + 1)] := #Altura*3.1416*4;

    // Ahora se actualiza el volumen total (Se considera la última fila, la 4)
    // Primero se respaldan los valores anteriores de la última fila
    // La última fila se reservó para almacenar los datos totales

    FOR #Puntero_Lectura := 4 TO 1 BY -1  DO
        #T_Lectura[4, (#Puntero_Lectura + 1)] := #T_Lectura[4, (#Puntero_Lectura)];
    END_FOR;
    // Luego se actualiza el valor total, sumando el volumen nuevo de todos los tanques

    #T_Lectura[4, 1] := 0; // En esta dirección está el volumen total
    // el cual se borra cada que se actualiza una altura
    // Ahora se halla el nuevo volumen total
    FOR #Puntero_Lectura :=1 TO 3 DO // Lectura 3 tanques, se aprovecha el mismo puntero
```

```
        #T_Lectura[4, 1] := #T_Lectura[4, 1] + #T_Lectura[#Puntero_Lectura, 1];
        //Se suma la primera columna (valor actual) de cada fila (tanque)
    END_FOR;
    // Ahora se actualizará la tabla de datos (Valor actual, Valor máximo
    // Valor mínimo y Valor promedio

    //Valor actual, solo se requiere mover de una tabla a otra
    //Los valores a mover están en la primera columna de la tabla de origen

    FOR #Puntero_Lectura := 1 TO 4 DO // En la fila 4 están los totales
        #T_Dato[#Puntero_Lectura, 1] := #T_Lectura[#Puntero_Lectura, 1];
    END_FOR;

    //Valor máximo, se guardan en la segunda columna de Datos
    // Se requiere otro puntero llamado puntero dato
    //
    FOR #Puntero_Dato:= 1 TO 4 DO // La tabla de datos tiene 4 filas
máximo) //se trabaja con la segunda columna de la tabla destino (Valor
        //
        #T_Dato[#Puntero_Dato, 2] := 0;
        // Se inicializa para comparar
        FOR #Puntero_Lectura:= 1 TO 5 DO // 5 comparaciones
            IF #T_Dato[#Puntero_Dato, 2]< #T_Lectura[#Puntero_Dato,#Pun-
tero_Lectura] THEN
        // Se va guadando el dato mayor de las comparaciones
          #T_Dato[#Puntero_Dato, 2] := #T_Lectura[#Puntero_Dato, #Punte-
ro_Lectura];
            END_IF;
        // Se repite 5 veces
        END_FOR;
el máximo //Luego se continúa con la siguiente fila, tanque, para encontrar

    END_FOR;
    // Se terminó con la actualización del valor máximo
          //Valor mínimo, se guardan en la tercera columna de Datos
    // Se requiere el puntero llamado puntero dato
    //
    FOR #Puntero_Dato := 1 TO 4 DO // La tabla de datos tiene 4 filas
mínimo) //se trabaja con la tercera columna de la tabla destino (Valor
        //
        #T_Dato[#Puntero_Dato, 3] := #T_Lectura[#Puntero_Dato, 1];
        // Se inicializa con primer valor para comparar
        FOR #Puntero_Lectura := 1 TO 5 DO // 5 comparaciones
          IF #T_Dato[#Puntero_Dato, 3] > #T_Lectura[#Puntero_Dato, #Pun-
tero_Lectura] THEN
        // Se va guardando el dato mayor de las comparaciones
        #T_Dato[#Puntero_Dato, 3] := #T_Lectura[#Puntero_Dato, #Puntero_Lectura];
            END_IF;
       // Se repite 5 veces
        END_FOR;
```

```
el mínimo//Luego se continúa con la siguiente fila, tanque, para encontrar

    END_FOR;
    // Se terminó con la actualización del valor mínimo
   //Valor promedio, se guardan en la cuarta columna de Datos
    // Se requiere el puntero llamado puntero dato

    FOR #Puntero_Dato := 1 TO 4 DO // La tabla de datos tiene 4 filas
promedio)//se trabaja con la cuarta columna de la tabla destino (Valor
        //
        #Acumulador := 0;
        // Se inicializa suma
        FOR #Puntero_Lectura := 1 TO 5 DO // 5 comparaciones
            #Acumulador := #Acumulador + #T_Lectura[#Puntero_Dato, #Puntero_Lectura];
        END_FOR;
        //Luego se continúa con la siguiente fila, tanque, para encontrar el mínimo
        #T_Dato[#Puntero_Dato, 4] := #Acumulador / 5;
    END_FOR;
    // Se terminó con la actualización del valor promedio

    //Rutina para mostrar los datos
    //Si mostrar es 1 se muestran datos totales
    //Si mostrar es 0 se muestran datos del tanque señalado
    //
    IF #Mostrar=1 THEN
        #V_Actual := #T_Dato[4, 1];// muestra valor actual total
        #V_max := #T_Dato[4, 2];// muestra valor máximo del total
        #V_min := #T_Dato[4, 3];// muestra valor mínimo del total
        #V_prom := #T_Dato[4, 4];// muestra valor promedio del total

    ELSE
        #V_Actual := #T_Dato[#Tanque, 1];// muestra valor actual tanque
        #V_max := #T_Dato[#Tanque, 2];// muestra valor máximo del tanque
        #V_min := #T_Dato[#Tanque, 3];// muestra valor mínimo del tanque
        #V_prom := #T_Dato[#Tanque, 4];// muestra valor promedio tanque

    END_IF;
 END_IF;
```

Las siguientes figuras presentan la simulación del proceso en una tabla SIM. En el caso de la Figura 104, es la simulación para el tanque 2, mostrando que la altura es de 4 m; recuerde que el radio del tanque es de 2 m y el valor actual del volumen es de 50,2656 m^3, que coincide con el volumen máximo de las últimas cinco lecturas, mientras que el mínimo es de 6 m^3.

Figura 104.

Tabla SIM para simulación del programa del problema 6.8

Nombre	Dirección	Formato de visua..	Observar/forzar
"Plantilla_DB".Altura		Floating-point nu...	4
"Plantilla_DB".Tanque		DEC+/-	2
"Plantilla_DB".Confirmación		Bool	FALSE
"Plantilla_DB".Mostrar		Bool	FALSE
"Plantilla_DB".V_Actual		Floating-point nu...	50,2656
"Plantilla_DB".V_max		Floating-point nu...	50,2656
"Plantilla_DB".V_min		Floating-point nu...	6
"Plantilla_DB".V_prom		Floating-point nu...	28,10624

Fuente: elaboración propia.

En la Figura 105 se observan los datos de todos los tanques, incluyendo los históricos; se puede constatar que, para el tanque 2, el valor promedio de los cinco volúmenes calculados es 28,10624 m^3, el valor máximo es 50,2656 m^3 y el mínimo es 6 m^3. El acumulado actual de los tres tanques es 662,2656 m^3.

Figura 105.

Lecturas para los tres tanques y el acumulado

Nombre	Dirección	Formato de visua..	Observar/forzar valor
"Plantilla_DB".T_Lectura[1,1]		Floating-point nu...	45
"Plantilla_DB".T_Lectura[1,2]		Floating-point nu...	45
"Plantilla_DB".T_Lectura[1,3]		Floating-point nu...	45
"Plantilla_DB".T_Lectura[1,4]		Floating-point nu...	45
"Plantilla_DB".T_Lectura[1,5]		Floating-point nu...	1
"Plantilla_DB".T_Lectura[2,1]		Floating-point nu...	50,2656
"Plantilla_DB".T_Lectura[2,2]		Floating-point nu...	25,1328
"Plantilla_DB".T_Lectura[2,3]		Floating-point nu...	25,1328
"Plantilla_DB".T_Lectura[2,4]		Floating-point nu...	34
"Plantilla_DB".T_Lectura[2,5]		Floating-point nu...	6
"Plantilla_DB".T_Lectura[3,1]		Floating-point nu...	567
"Plantilla_DB".T_Lectura[3,2]		Floating-point nu...	567
"Plantilla_DB".T_Lectura[3,3]		Floating-point nu...	567
"Plantilla_DB".T_Lectura[3,4]		Floating-point nu...	128
"Plantilla_DB".T_Lectura[3,5]		Floating-point nu...	128
"Plantilla_DB".T_Lectura[4,1]		Floating-point nu...	662,2656
"Plantilla_DB".T_Lectura[4,2]		Floating-point nu...	637,1328
"Plantilla_DB".T_Lectura[4,3]		Floating-point nu...	637,1328
"Plantilla_DB".T_Lectura[4,4]		Floating-point nu...	646
"Plantilla_DB".T_Lectura[4,5]		Floating-point nu...	646

Fuente: elaboración propia.

En la Figura 106 se muestra la información en la base de datos plantilla en el área de la matriz T_Dato.

Figura 106.

Información en la base de datos plantilla en el área de la matriz T_Dato

Nombre	Dirección	Formato de visua..	Observar/forzar
"Plantilla_DB".T_Dato[1,1]		Floating-point nu...	45
"Plantilla_DB".T_Dato[1,2]		Floating-point nu...	45
"Plantilla_DB".T_Dato[1,3]		Floating-point nu...	1
"Plantilla_DB".T_Dato[1,4]		Floating-point nu...	36,2
"Plantilla_DB".T_Dato[2,1]		Floating-point nu...	50,2656
"Plantilla_DB".T_Dato[2,2]		Floating-point nu...	50,2656
"Plantilla_DB".T_Dato[2,3]		Floating-point nu...	6
"Plantilla_DB".T_Dato[2,4]		Floating-point nu...	28,10624
"Plantilla_DB".T_Dato[3,1]		Floating-point nu...	567
"Plantilla_DB".T_Dato[3,2]		Floating-point nu...	567
"Plantilla_DB".T_Dato[3,3]		Floating-point nu...	128
"Plantilla_DB".T_Dato[3,4]		Floating-point nu...	391,4
"Plantilla_DB".T_Dato[4,1]		Floating-point nu...	662,2656
"Plantilla_DB".T_Dato[4,2]		Floating-point nu...	662,2656
"Plantilla_DB".T_Dato[4,3]		Floating-point nu...	637,1328
"Plantilla_DB".T_Dato[4,4]		Floating-point nu...	645,7062

Fuente: elaboración propia.

La Figura 107 destaca la información obtenida por fuera de la plantilla que es depositada en el área de memoria de marcas, principalmente.

Figura 107.

Información obtenida fuera de la plantilla memoria de marcas

Nombre	Dirección	Formato de visua..	Observar/forzar
"Plantilla_DB".Puntero_Lectura		DEC+/-	6
"Plantilla_DB".Puntero_Dato		DEC+/-	5
"Entra dato"	%M2.0	Bool	FALSE
"Dato_Altura"	%MD10	Floating-point nu...	4
"Dato_Tanque"	%MW14	DEC+/-	2
"Actual"	%MD20	Floating-point nu...	50,2656
"Máximo"	%MD24	Floating-point nu...	50,2656
"Mínimo"	%MD28	Floating-point nu...	6
"Promedio"	%MD32	Floating-point nu...	28,10624
"Tanque_o_Total"	%M2.1	Bool	FALSE

Fuente: elaboración propia.

Índice de figuras

Índice de tablas

Referencias

De la Puente Viedma, C. (2018). *Estadística descriptiva e inferencial.* Ediciones IDT.

Gil Sánchez, L.; Ibáñez Civera, J.; García Breijo, E. (2019). *Problemas de electrónica digital*. Universidad Politécnica de Valencia.

Goderie, J. (2019). *SCL-T Template programming for Siemens SCL* (Tesis de *maestría*, Delft University of Technology).

Hernández Corssi, N. A. (2019). *Diseño y simulación de un proceso industrial utilizando controlador SIMATIC S7-1200 con TIA PORTAL versión 14* (Trabajo de grado, Universidad Técnica Federico Santa María).

Martínez León, J. F. (2017). *Elaboración de material docente para el control de procesos discretos* (Trabajo de grado, Universidad de Jaén).

Öhman, M. y Johansson, S. (1995). *Prototype Implementation of the PLC Standard IEC 1131-3* (Tesis de maestría, Lund Institute of Technology). http://lup.lub.lu.se/student-papers/record/8848864

Peciña Belmonte, L. (2018). *Programación de controladores avanzados SIMATIC S7 1500 con TIA Portal AWL y SCL*. 2.ª ed. Marcombo.

Salazar López, B. (2019). *Promedio simple. Pronóstico de la demanda*. https://www.ingenieriaindustrialonline.com/pronostico-de-la-demanda/promedio-simple/

Siemens (2005). *SIMATIC S7-SCL V5.3 para S7-300/40*0. Manual.

Siemens (2016). *SIMATIC S7-1200 Easy Book*. Manual.

Siemens (2018). Módulo TIA Portal 051-201. *Programación en lenguajes de alto nivel con SCL y SIMATIC S7-1200. Documentación didáctica para cursos de formación*. Digital Factory, DF FA.

Teknik
Ingeniería y Tecnología

Programación de controladores lógicos programables con lenguaje SCL

Fuentes tipográficas: *Charter roman* para texto corrido, en 14 puntos, para títulos en *Galvji Bold* 16 puntos y subtítulos en *Galvji* 15 puntos.

Teknik

Ingeniería y Tecnología

En este libro se realiza una introducción a la programación de controladores lógicos programables haciendo uso del lenguaje conocido como SCL (Structured Control Language), bajo el software de programación TIA PORTAL de Siemens, incorporando los conceptos teóricos y desarrollo de problemas de aplicación. Como estrategia, se acompaña la mayoría de los capítulos con problemas resueltos adaptados a situaciones que se pueden encontrar en la industria·

FONDO EDITORIAL ITM

Alcaldía de Medellín