PREPRÁCTICA #3

Tema: Introducción a diagrama de bloques de funciones y texto estructurado.

1. Objetivos

1.1. Objetivo general

Desarrollar una aplicación de control de velocidad utilizando Connected Components Workbench para la realización de aplicaciones industriales, utilizando programación en diagrama de bloques de funciones y texto estructurado en el PLC micro850.

1.2. Objetivos específicos

1. Analizar el funcionamiento de la programación diagrama de bloque de funciones para la realización diferentes tipos de conexiones de bloques de instrucciones y texto estructurado en el entorno de programación.
2. Utilizar bloques temporizadores, contadores y comparadores en el entorno de grafico de bloques de funciones y texto estructurado.
3. Elaborar una aplicación con el Micro850 para el control y supervisión de las variables en la interfaz HMI

¿Qué actividades realizaran?

1. Tabular ventajas, desventajas o diferencias entre diagrama de bloques de funciones, texto estructurado con respecto a lenguaje de escalera.
2. Realizar y simular la aplicación del PowerFlex de la prepráctica anterior, pero en lenguaje de bloques.
3. Realizar y simular la aplicación del PowerFlex de la prepráctica anterior, pero en texto estructurado.

¿Como lo realizaran?

Programas por utilizar:

- Connected Components Workbench -CCW (se recomienda versión 13)
- RsLinx Classic
- Simulador micro850

¿Cuáles son los entregables de la prepráctica?

Realizar una infografía de manera individual/grupal (pareja) de las actividades a desarrollar en formato A3 (horizontal o vertical), el cual debe tener los siguientes elementos:

- Un tema claro y conciso: Es decir, comunicar un tema específico de manera clara y fácil de entender.
- Datos relevantes y precisos: La información presentada en la infografía debe ser precisa y relevantes a su tema.
- Diseño atractivo y llamativo: La infografía debe ser visualmente atractiva y llamar la atención del espectador.
- Estructura fácil de seguir: La información debe presentarse en una estructura clara y fácil de seguir para que el espectador no tenga problemas para entender su contenido.
- Fuentes y referencias: La infografía debe incluir las fuentes y referencias utilizadas para obtener la información presentada en ella.

Nota: Todas las prepráctícas serán entregados en el aula virtual en PDF, hasta la semana de la práctica, las faltas ortográficas serán penalizadas, así como la copia ya sea con otros reportes o de internet.

Material de apoyo

Seguir el siguiente orden para la realización de la prepráctica.

- Video 1: Introducción a diagrama de bloques de funciones en CCW
- Video 2: Realizar diagrama de bloques de funciones basado en diagrama escalera en CCW - Parte1
- Video 3: Realizar diagrama de bloques de funciones basado en diagrama escalera en CCW - Parte1
- Video 4: Configurar un PanelView en CCW
- Video 5: Descargar una pantalla HMI en un PanelView 800 utilizando CCW
- Video 6:Conexiones físicas para el control de un motor utilizando un PowerFlex 4M, PLC micro850 y PanelView 800.
- Video 7: Implementación de control de motor utilizando PowerFlex 4M, PLC micro850 y PanelView 800.
- Video 8: Texto estructurado en CCW

Anexo

Diagrama de bloque de funciones

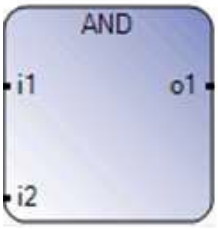
- Contiene bloques de instrucciones con variables de entrada y variables de salida. Las salidas de los bloques de instrucciones se conectan a las entradas de otros bloques de instrucciones mediante líneas de conexión.
- Cada variable de entrada debe estar conectada a una entrada de bloque y tener el tipo de datos correcto para dicha entrada. Una entrada de bloque puede ser un valor literal, una variable interna, una variable de entrada o una variable de salida de otro bloque.
- Cada variable de salida debe estar conectada a una salida de bloque y tener el tipo de datos correcto para dicha salida. Una salida de bloque puede ser una variable interna, una variable de salida o el nombre de un bloque (solo en el caso de funciones). Si una salida es el nombre de una función editada, representa la asignación del valor de retorno de la función, y el valor de la salida se devuelve al programa de llamada.
- Las conexiones están orientadas de izquierda a derecha, por lo que transportan los datos en esa dirección. La conexión izquierda y la conexión derecha deben tener el mismo tipo de datos.
- Un único punto de la conexión de la izquierda puede conectarse a varias conexiones de la derecha, denominadas divergencias, para difundir información a varios puntos. Todas las conexiones deben ser del mismo tipo de datos.

Bloque de funciones

Los bloques de funciones están representados por un cuadro que muestra el nombre de la instrucción y la versión abreviada de los nombres de parámetro y el nombre de la instancia se muestra en cursiva.

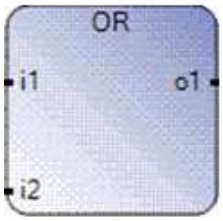
Funciones

Operación AND



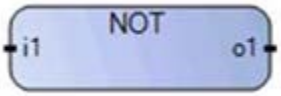
Parametro	Tipo de parámetro	Tipo de datos	Descripción
i1	Entrada	Bool	Valor en tipo de datos booleanos.
I2	Entrada	Bool	Valor en tipo de datos booleanos.
O1	Salida	Bool	Resultado de la operación booleana AND de los valores de entrada

Operación OR



Parámetro	Tipo de parámetro	Tipo de datos	Descripción
i1	Entrada	Bool	Valor en tipo de datos booleanos.
I2	Entrada	Bool	Valor en tipo de datos booleanos.
O1	Salida	Bool	Función booleana OR de los términos itroducidos.

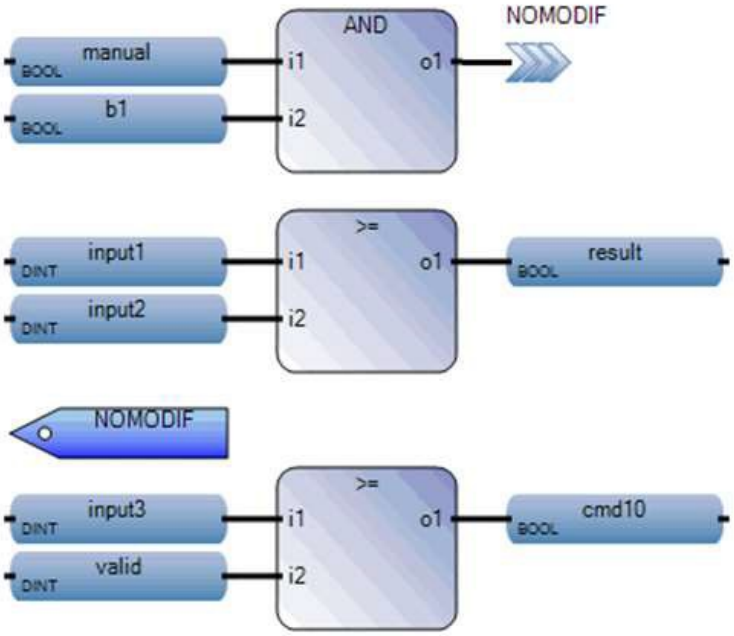
Operación NOT



Parámetro	Tipo de parámetro	Tipo de datos	Descripción
i1	Entrada	Bool	Cualquier expresión compleja o valor booleano.
O1	Salida	Bool	Verdadero cunado IN es Falso. Falso cuando IN es verdadero.

Etiquetas

Las etiquetas se utilizan como un destino para las instrucciones de salto o para controlar la ejecución del diagrama. Si la línea de conexión a la izquierda del símbolo de salto tiene el estado booleano Verdadero, la ejecución del programa salta directamente después del símbolo de etiqueta correspondiente. Ningún otro objeto se puede conectar a la derecha de un símbolo de salto. Las etiquetas no se conectan con otros elementos.



Variables

Las variables se usan para almacenar y procesar información. Utilice el cuadro de herramientas de FBD para agregar una variable a un programa de FBD.

Arrastre el elemento de “variable” al editor de lenguaje para abrir el selector de variables.

Ejemplo:

Marcha y paro de un sistema.

Structured Text

Consiste en una serie de instrucciones, donde es determinado como uno de los lenguajes de programación de alto nivel. En dicho lenguaje pueden ser ejecutados condicionales ("IF..THEN..ELSE") o lazos (“WHILE..DO”).

Ejemplo:

```
IF value < 7 THEN
    WHILE value < 8 DO
        value := value + 1;
    END_WHILE;
END_IF;
```

Existen operaciones disponibles en texto estructurado, donde la norma IEC 61131-3 describe todos los operadores estándar.

Operación	Símbolo
Función call	Function name (parameter list)
Exponenciación	EXPT

Negativo Construir el complemento	- NOT
Multiplicación División Modulo	* / MOD
Suma Resta	+ -
Comparación	<,>,<=,>=
Igual que No es igual a que	= <>
Booleano AND	AND
Booleano XOR	XOR
Booleano OR	OR
Único Comentario Múltiples comentarios	// /* start comment*/ or (* start comment*)
Estado de asignación	variable:= expresión;
Estados de selección	IF, THEN, ELSE, CASE...
Estados de interacción	FOR, WHILE, REPEAT...
Estados de control	RETURN, EXIT...

Elementos básicos y sentencias en texto estructurado

1. *<u>Asignación</u>*

Ítem	Descripción
Símbolo	:=
Significado	Asigna una variable a una expresión
Sintaxis	<variable> := <any_expression> ;
Operador	La variable deber ser una interna o externa y la expresión debe tener un mismo tipo de dato.

Ejemplo:

```
(* ST program with assignments *)

(* variable <<= variable *)
bo23 := bo10;


(* Variable <<= expression *)

bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;


(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

2. *IF THEN ELSE*

Ítem	Descripción
Nombre	IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF
Significado	Ejecuta una serie de listas de declaraciones de texto estructurado. Una selección es hecho acordando con el valor de una expresión booleana.

Sintaxis	IF <Boolean_expression> THEN
	    <statement> ;
	    <statement> ;
	    ...
	ELSIF <Boolean_expression> THEN
	    <statement> ;
	    <statement> ;
	    ...
	ELSE
	    <statement> ;
	    <statement> ;
	    ...

Ejemplo:

```
(* ST program using IF statement *)

IF manual AND not (alarm) THEN
level := manual_level;
bx126 := bi12 OR bi45;
ELSIF over_mode THEN
level := max_level;
ELSE
level := (lv16 * 100) / scale;
END_IF;

(* IF structure without ELSE *)
If overflow THEN
alarm_level := true;
END_IF;
```

3. *LAZO WHILE*

Tipo	Descripción

Nombre	WHILE ... DO ... END_WHILE
Significado	Es la estructura de iteraciones para un grupo de declaraciones de texto estructurado. La condición o expresión es evaluada antes de cada iteración.
Sintaxis	WHILE <Boolean_expression> DO <statement> ; <statement> ; ... END_WHILE ;

Importante: No iterar dentro un lazo demasiadas veces en un solo escaneo.
El controlador no ejecuta otras instrucciones en la rutina hasta que complete el lazo. Una falla importante ocurre cuando al completar el ciclo lleva más tiempo que el watchdog timer para la tarea.

Ejemplo:

```
(* ST program using WHILE statement *)
cnt := 0;
WHILE (_IO_EM_DI_01) DO

    cnt := cnt + 1;
    If cnt>10 THEN
        EXIT;
    END_IF;

END_WHILE;
```

4. LAZO FOR

Tipo	Descripción
Nombre	FOR ... TO ... BY ... DO ... END_FOR BY es opcional. Si no es especificado, el paso del incremento es 1.
Significado	Ejecuta un número limitado de iteraciones usando un índice (variable entera).
Sintaxis	FOR <index> := <mini> TO <maxi> BY <step> DO <statement> ; <statement> ; END_FOR;
Operando	Index: Variable interna (integer) incrementada en cada iteración. Mini: Valor inicial (integer) para el índice antes de la

	primera iteración. Maxi: Valor máximo (integer) permitido para Index. Step: Incremento por cada interacción

Ejemplo:

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
    END_IF;
END_FOR;
```

5. *CASE*

Tipo	Descripción
Nombre	CASE ... OF ... ELSE ... END_CASE
Significado	Ejecuta una de las condiciones. La selección es realizada de acuerdo con una expresión de tipo entero.
Sintaxis	CASE <integer_expression> OF <value> : <statement1> ; <statement2> ; <statementsN> <value> : <statements> ; <value>, <value> : <statements>; ... ELSE <statements> ; END_CASE;

Ejemplo:

```
(* ST program using CASE statement *)

CASE error_code OF 255: err_msg := 'Division by zero';
    fatal_error := TRUE;
    1: err_msg := 'Overflow';
    2, 3: err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
END_CASE;
```

Estructura de temporizadores en texto estructurado

- *TON*

```
TON_1(
```

void **TON_1**(BOOL IN, TIME PT)
Type : TON, On-delay timing

```
1  MaxTime := T#3s;
2  TON_1(in, MaxTime);
3  output := TON_1.Q;
4  elapse := TON_1.ET;
```

- *TOF*

```
TOF_1(
```

void **TOF_1**(BOOL IN, TIME PT)
Type : TOF, Off-delay timing

```
1  MaxTime := T#3s;
2  TOF_1(in, MaxTime);
3  output := TOF_1.Q;
4  elapse := TOF_1.ET;
```

- *TONOFF*

```
TONOFF_1(
```

void **TONOFF_1**(BOOL IN, TIME PT, TIME PTOF)
Type : TONOFF, Delay an output-on(true), then delay an output-off(false).

```
1  OnDelay := T#3s;
2  OffDelay := T#5s;
3  TONOFF_1(in, OnDelay, OffDelay);
4  output := TONOFF_1.Q;
5  elapse := TONOFF_1.ET;
```

- *TP*

```
TP_1(
```

void **TP_1**(BOOL IN, TIME PT)
Type : TP, Pulse timing

```
1  MaxTime := T#3s;
2  TP_1(in, MaxTime);
3  output := TP_1.Q;
4  elapse := TP_1.ET;
```

Estructura de contadores en texto estructurado

- *CTD*

```
CTD_1(
```

void **CTD_1**(BOOL CD, BOOL LOAD, DINT PV)
Type : CTD, Down counter

```
1  InitialValue := 10;
2  CTD_1(cd, load, InitialValue);
```

(*ST Equivalence: CTD1 is an instance of block *)

CTD1(trigger,load_cmd,100);

underflow := CTD1.Q;

result := CTD1.CV;

- *CTU*

```
CTU_1(
```

void **CTU_1**(BOOL CU, BOOL RESET, DINT PV)
Type : CTU, Up counter

```
1  MaximumValue := 10;
2  CTU_1(cu, reset, MaximumValue);
```

(* ST Equivalence: CTU1 is an instance of CTU block*)

```
CTU1(trigger,NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```

- *CTUD*

```
cu := TRUE;
cd := TRUE;
reset := FALSE;
load := FALSE;
pv := 10;
CTUD_1(cu, cd, reset, load, pv);
```

```
CTUD_1(
```

void **CTUD_1**(BOOL CU, BOOL CD, BOOL RESET, BOOL LOAD, DINT PV)
Type : CTUD, Up-down counter

(* ST Equivalence: We suppose CTUD1 is an instance of block*)

```
CTUD1(trigger1, trigger2, reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
```

Bloque SCALER

```
SCALER_1(
```

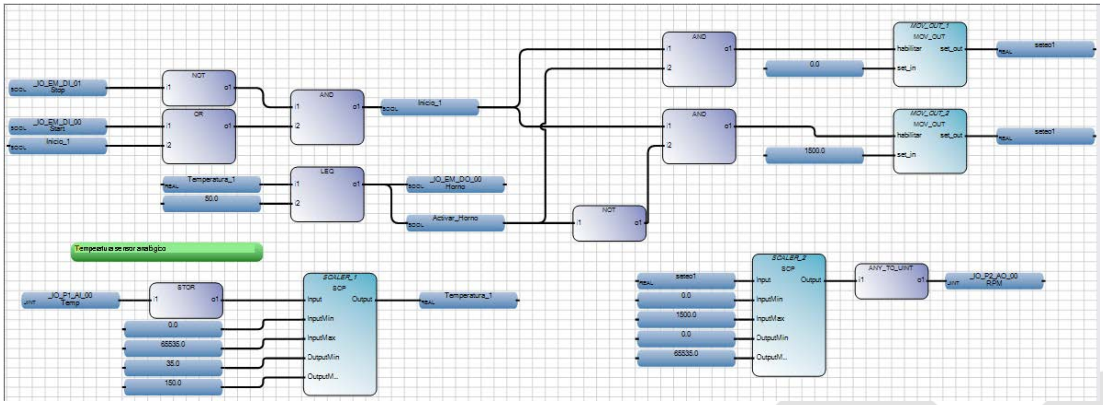
void **SCALER_1**(REAL Input, REAL InputMin, REAL InputMax, REAL OutputMin, REAL OutputMax)
Type : SCALER, Scale input value according to output range.

```
input := 10.0;
InputMin := 5.0;
InputMax := 15.0;
OutputMin := 1.0;
OutputMax := 10.0;
SCALER_1(input, InputMin, InputMax, OutputMin, OutputMax);
output := SCALER_1.Output;
```

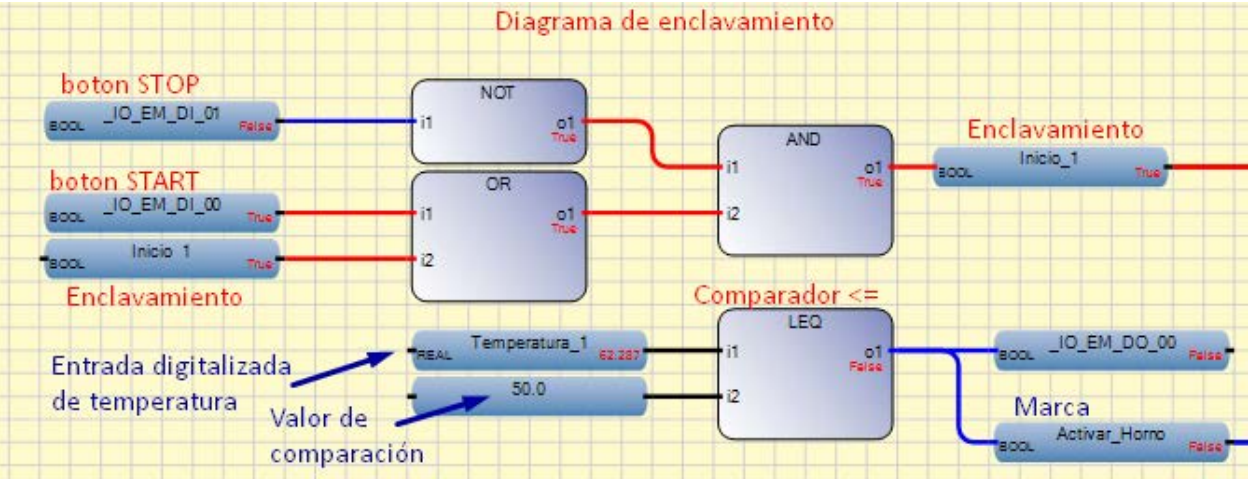
(* ST equivalence: SCALER1 is an instance of SCALER block *)

```
SCALER1(Signal_In, 4.0, 20.0 , 0.0 , 150.0 ) ;
Out_Temp := SCALER1.Output ;
```

Anexos

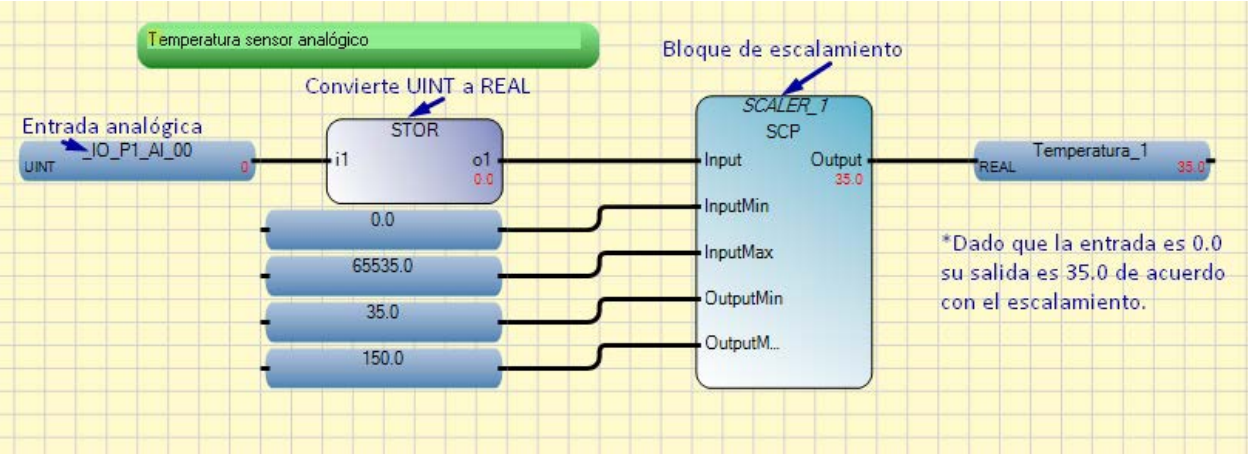


A continuación, se explica cada sección en funcionamiento.

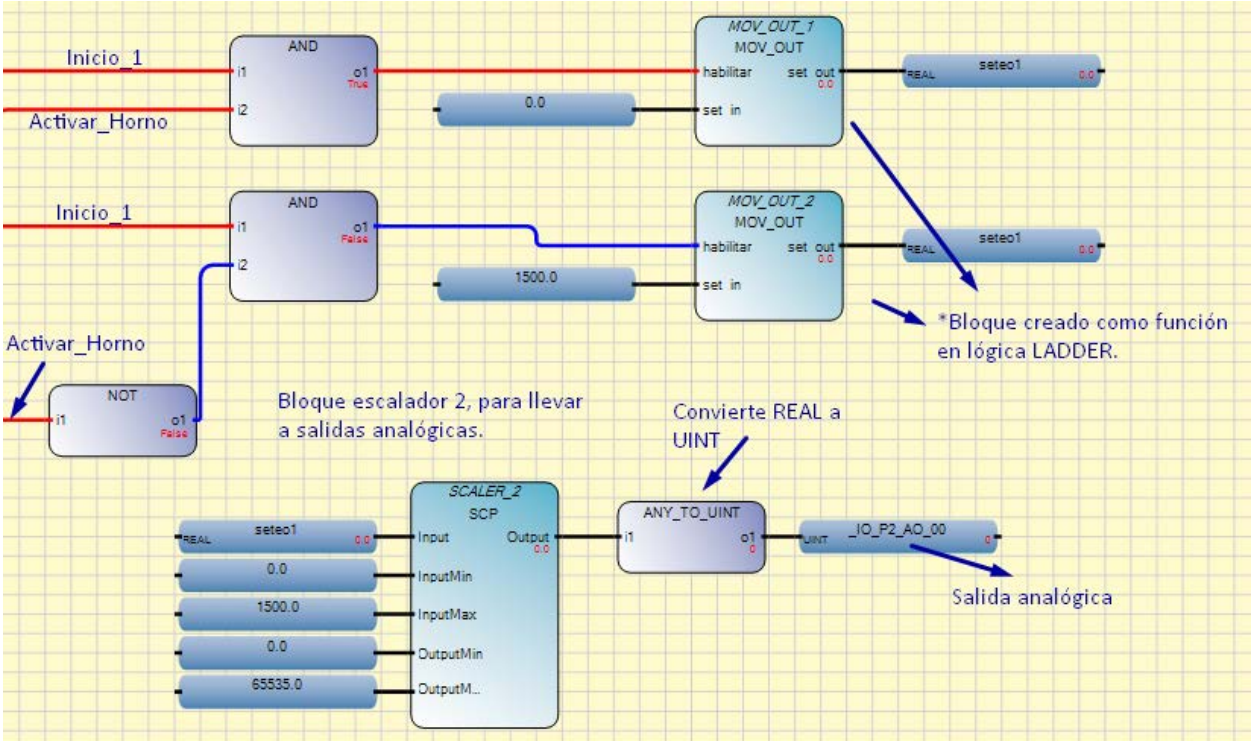


Primero tenemos, la sección de marcha/paro, donde se coloca los botones Start, Stop y una variable de enclavamiento llamada Inicio_1. Se usa una compuerta OR para el enclavamiento (puesta en paralelo), y se usa una compuerta AND para el botón de parada (Puesta en serie).

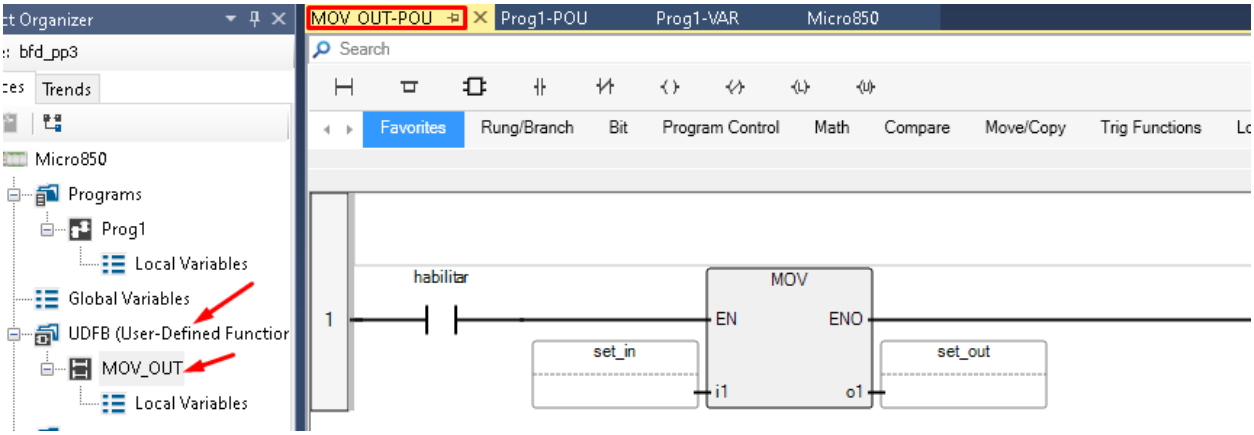
Segundo, tenemos un bloque comparador (menor o igual), el cual se usa para comparar si la temperatura que ingresa como entrada analógica es menor que 50°C, y con ello se activa la marca Activar_Horno y la salida de la variable Horno. La entrada de temperatura es digitalizada en los siguientes bloques, es decir, convertida de tipo dato UINT a REAL y esa se usa en este bloque comparador.



Tercero, se describe el escalamiento de la señal analógica, para ello se usa el bloque variable para especificar la dirección de tipo analógica (UINT), luego como ocurría con LADDER se debe convertir a tipo REAL para poder usarse con el bloque SCALER, por ello se usa el bloque STOR que convierte de UINT a REAL. Luego, se le da los valores mínimos y máximos del escalamiento, y se coloca una marca llamada temperatura_1 para usarla en los demás bloques. El rango de entrada puesto que es tipo REAL va de 0.0 a 65535.0, la salida está en un rango de 35.0 a 150.0.



Cuarto, se realiza una relación con compuertas AND, si Inicio_1 y Activar_Horno están activadas, colocamos un valor de 0.0 en la marca seteo1. El bloque llamado MOV_OUT es realizado en LADDER dado que, se necesita el bloque MOV para asignar una variable a otra, pero con una habilitación (que se active solo cuando queramos). El diagrama LADDER se muestra a continuación:



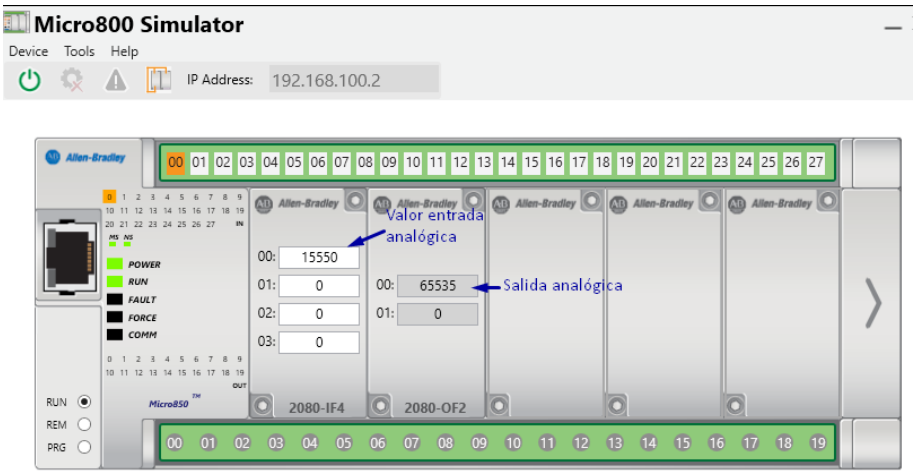
Es importante recalcar que cuando se programa con bloques funcionales, se puede crear cualquier función que deseemos en otro tipo de lenguaje (texto estructurado, LADDER). Es así, como se ha creado este bloque para luego usarlo en la programación. Este bloque solo contiene el bloque MOV con una habilitación. El bloque MOV necesita la habilitación, el valor a asignar y el valor de salida que se asignó.

Sí Inicio_1 esta activado y Activar_Horno está desactivado, se asigna en la marca seteo1 el valor de 1500.0.

Además, también se presentan los bloques que permiten visualizar en el módulo de salidas analógicas OF2, que básicamente es similar a lo visto con LADDER, consiste en realizar un escalado de la variable REAL de la marca seteo1, en un rango de entrada de 0.0 a 1500.0, hasta un rango de salida REAL (0.0 a 65535.0) para luego convertir esta variable tipo REAL a tipo UINT con el bloque ANY_TO_UINT. Finalmente, se asigna la variable de salida analógica.

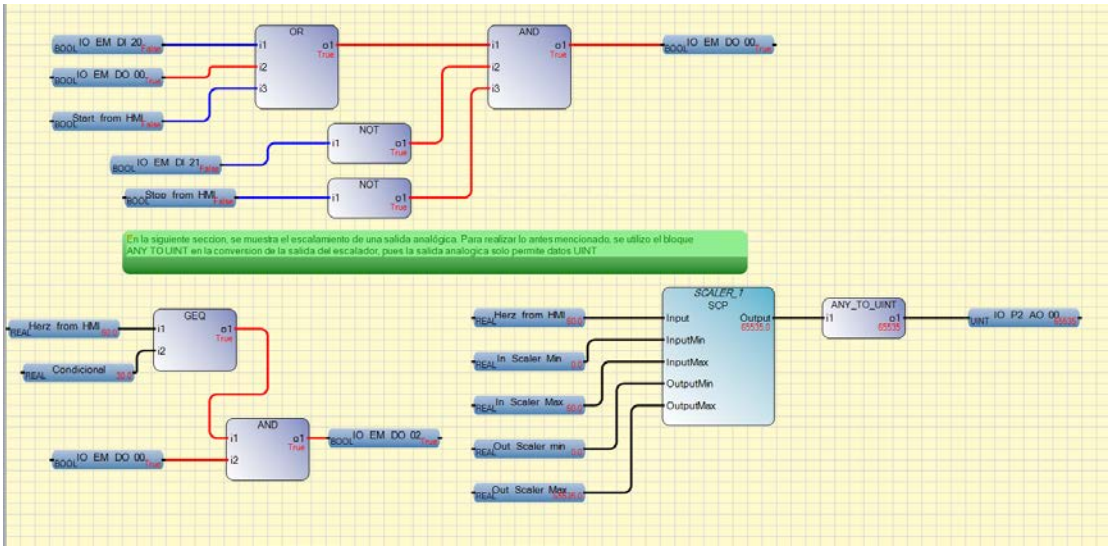
*La salida analógica representa las RPM.

En la siguiente sección se muestra pruebas de funcionamiento, para ello se simula y se coloca como entrada analógica el valor de 15550.0 en A0_00, que representa el valor para el cual su salida es máxima. Se visualizan los resultados en el simulador del Micro800:



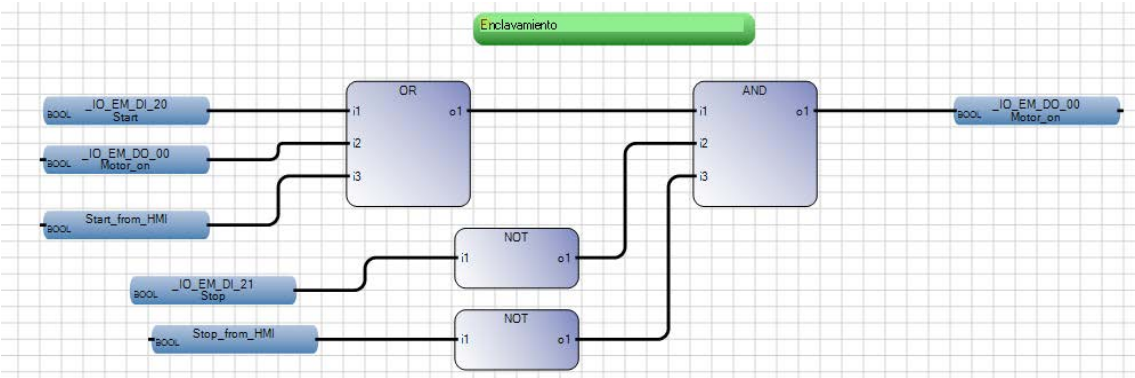
Pruebas de funcionamiento.

Analizar y comentar el proyecto en diagrama de bloques realizado en el software Connected Component Workbench. Además, realice los cambios necesarios en la programación para descargarlo en el PLC micro850 y controlar la velocidad, paro, marcha y cambio de giro en el variador PowerFlex 4M desde un PanelView 800.

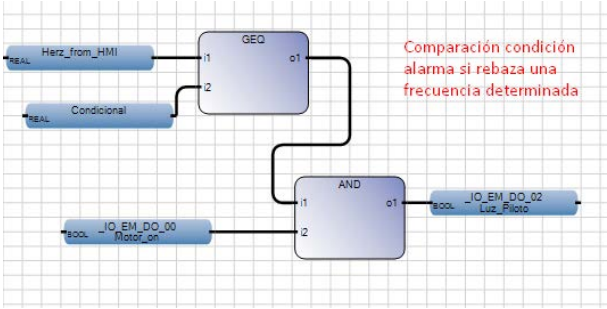


Descargar los archivos de la prepráctica para completar el ejercicio propuesto.

Análisis de programación base.



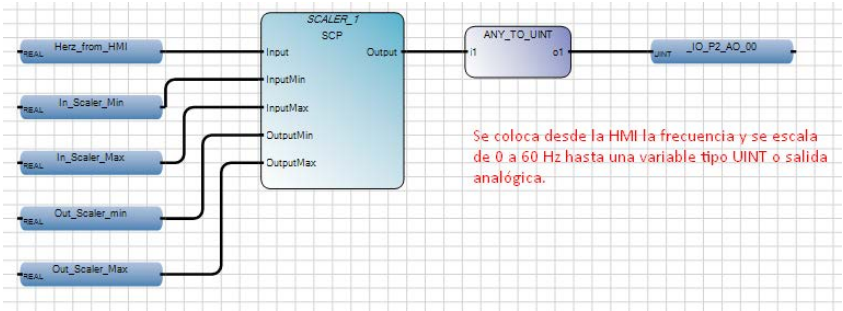
Se realiza el enclavamiento a partir de la entrada START, tanto del tablero como de HMI, si cualquiera de ellos se activa (a través de la compuerta OR), y el botón de STOP no está presionado, es decir, aquí es normalmente abierto (N.O), entonces se da la activación del motor.



Esta sección, nos compara a través del bloque GEQ (mayor o igual), y si la frecuencia de entrada es mayor o igual a 30 entonces se activa una luz piloto. Se presentan las variables locales usadas en este proyecto:

Scope: Prog2 Filter...

Name	Alias	Data Type	Dimension	Project Value	Initial Value
In_Scaler_Max		REAL			60.0
In_Scaler_Min		REAL			0.0
Out_Scaler_Max		REAL			65535.0
Out_Scaler_min		REAL			0.0
SCALER_1		SCALER		...	...
Condicional		REAL			30.0
New.					



Se usa el bloque SCALER para acondicionar la señal de salida hacia una variable de salida analógica y entrada del variador de frecuencia, con el valor REAL de 0.0 tendríamos 0V, y con 65535.0 tendríamos 10V que se conectaría a la entrada del variador o pin 13 de acuerdo con el datasheet del PowerFlex 4M.

Luego, si abrimos la sección de la pantalla HMI observamos primero la siguiente configuración de tags o etiquetas que se usan para vincular las variables a usar en la programación:

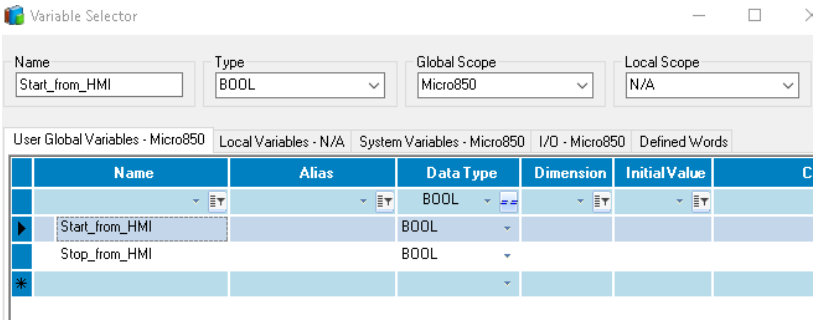
Tag Editor | Prog2-POU | Micro850 | Start Page

External | Memory | System | Global Connections

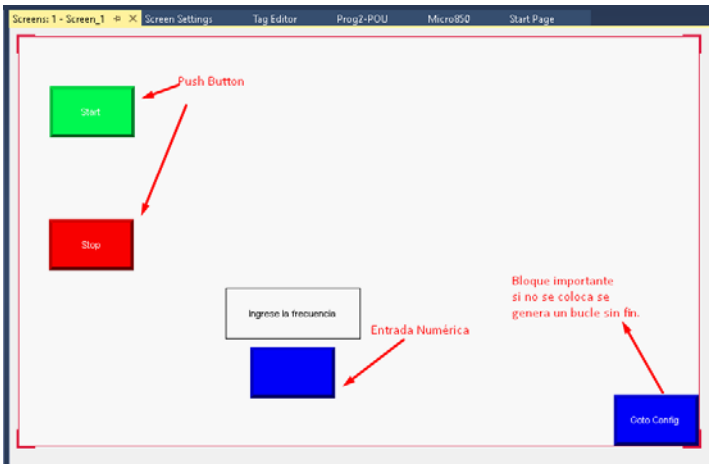
Add | Delete | Undo | Redo | Import | Export | << Typical

Tag Name	Data Type	Address	Controller	Description	Data Entry - Min	Data Entry -	Access
Start	Boolean	Start_from_HMI	PLC-1				Read/Write
Stop	Boolean	Stop_from_HMI	PLC-1				Read/Write
Herz	Real	Herz_from_HMI	PLC-1		-9999999	9999999	Read/Write

Se realiza etiquetado de los botones que se van a colocar desde la HMI, y se las vincula con las variables creadas en la programación.

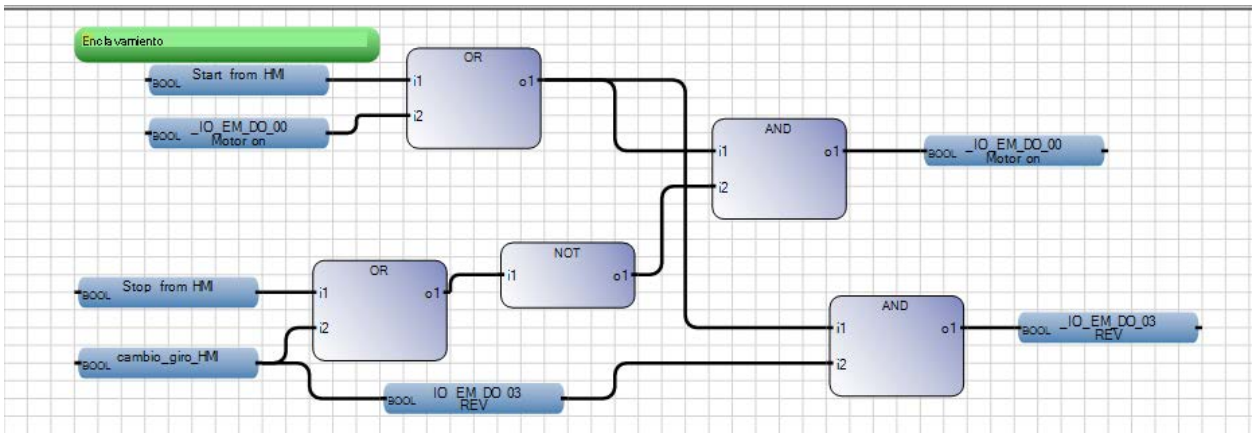


Se observa que son las mismas variables globales que se habían configurado anteriormente. Luego se muestra la pantalla HMI y sus configuraciones:

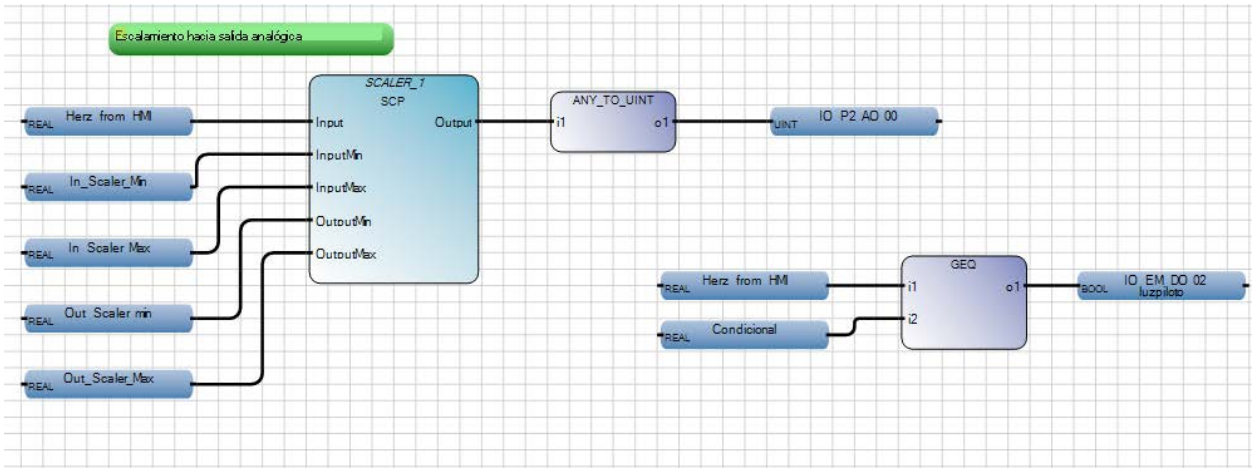


Luego, en pantallas tenemos agregado 2 botones pulsadores, y una entrada numérica para colocar valores desde la HMI y redireccionar dicho valor a la dirección de la variable asociada del controlador. Lo importante de estos bloques de programación es que podemos colocar la frecuencia desde la HMI.

Modificación del proyecto para descargarlo en el PLC.



Las modificaciones mostradas se realizaron con el fin de lograr enviar una señal que permita controlar el cambio de giro, es por esto que la inicialización del sistema y el encendido del motor ingresan a un or cuya salida ingresa a dos compuertas de tipo and dónde se comparan en la primera con la inactividad del stop o la señal de cambio de giro y en la segunda con la ejecución del motor en reverse, logrando con esto enclavar tanto la señal de encendido del motor como el reverse.



El uso scaler nos permite condicionar la señal de salida (digital) a una variable analógica por lo que mantiene la misma programación y escala definida antes de realizar las modificaciones.

Micro850-VAR | Settings | Tag Editor | Screen Settings | Screen

Scope: Micro850 Filter...

Name	Alias	Data Type
_IO_EM_DO_00	Motor_on	BOOL
_IO_EM_DO_01		BOOL
_IO_EM_DO_02	luzpiloto	BOOL
_IO_EM_DO_03	REV	BOOL

A las variables usadas dentro de la programación se les otorga un alias que nos permitan reconocer el funcionamiento del sistema a partir de palabras claves.

Para el HMI:

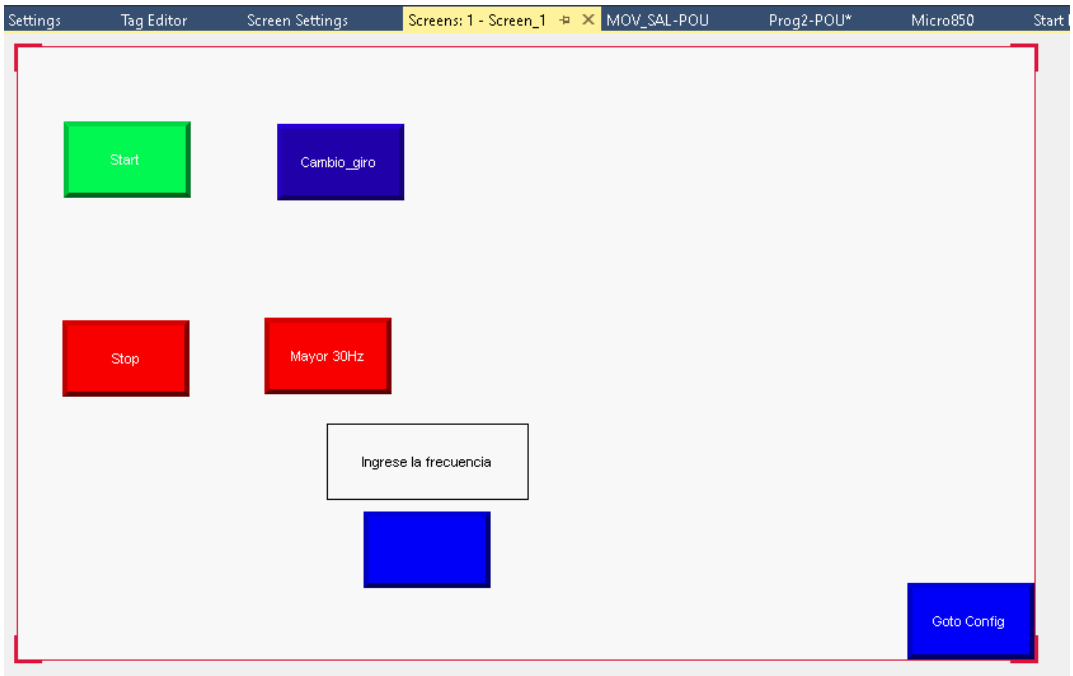
Settings | Tag Editor | Screens: 1 - Screen_1 | MOV_SAL-POU | Prog2-POU*

External | Memory | System | Global Connections

Add | Delete | Undo | Redo | Import | Export

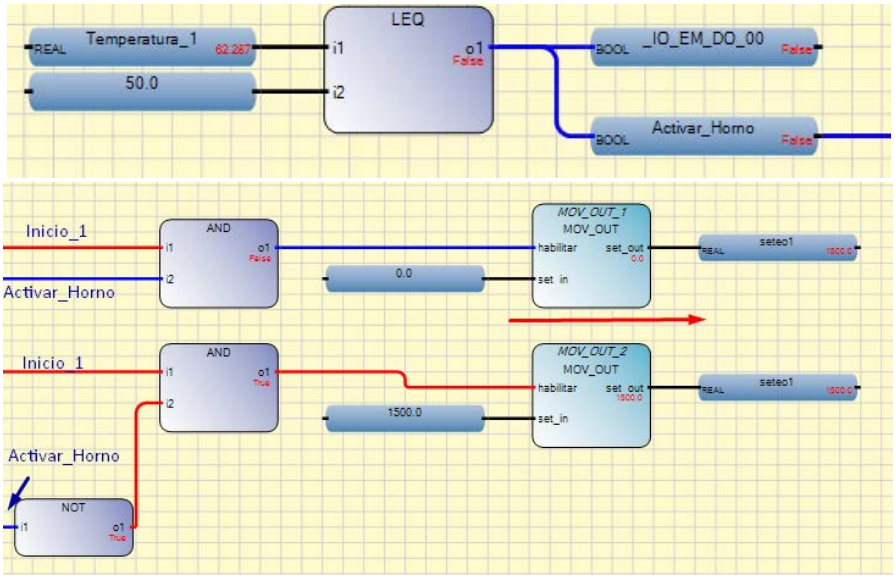
Tag Name	Data Type	Address	Controller	Description
Start	Boolean	Start_from_HMI	PLC-1	
Stop	Boolean	Stop_from_HMI	PLC-1	
Herz	Real	Herz_from_HMI	PLC-1	
girar_motor	Boolean	cambio_giro_ ...	PLC-1	
Piloto	Boolean	Luz_piloto	PLC-1	

Los botones presentes en el HMI necesitan ser etiquetados correctamente de manera que se los identifique según su función. En este caso solo se aumentan las variables creadas a partir de los cambios realizados.



Finalmente se pueden visualizar los nuevos botones (cambio de giro, mayor 30Hz) y desde esta pantalla manipular el funcionamiento del proyecto creado, sin embargo, se deben tomar en cuanta las limitaciones del programa usado para la simulación ya que no se nos permite simular el HMI como tal. El funcionamiento del resto de botones sigue siendo el mismo que se analizó antes de realizar los cambios.

Cuando introducimos otro valor, como por ejemplo 15550 obtendremos los siguientes resultados:



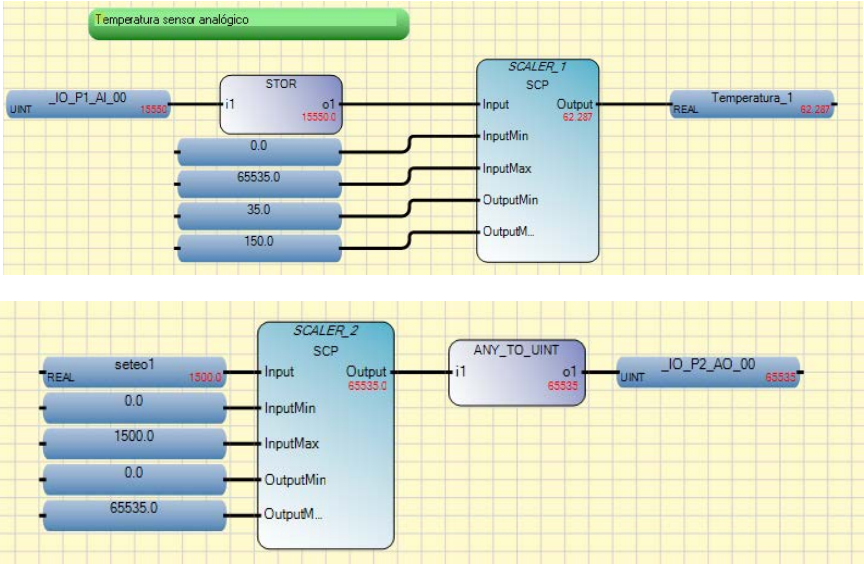
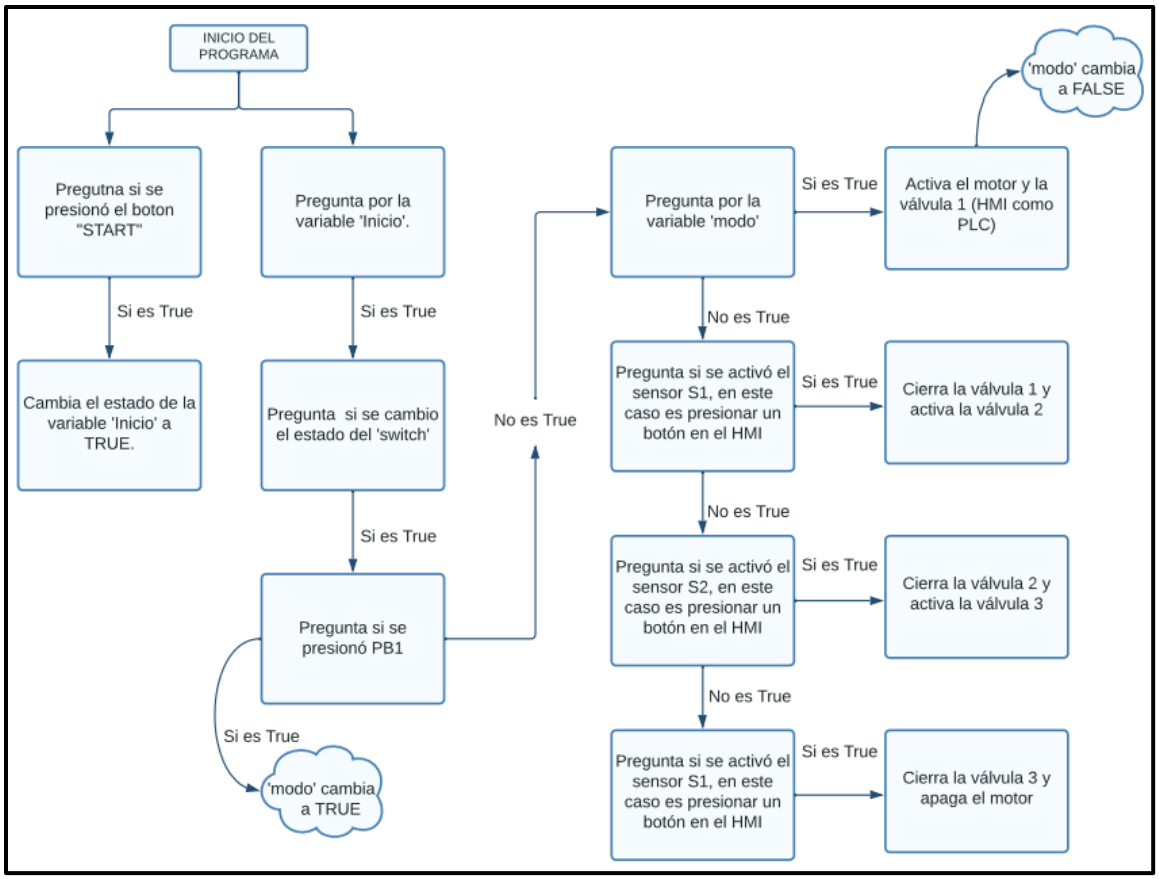


Ilustración 1. Pruebas de funcionamiento.

Analizar el proyecto en texto estructurado realizado en el software Connected Component Workbench. Además, describa el funcionamiento del proceso automatizado con la interfaz de la figura 1.

```
IF start OR _IO_EM_DI_00 THEN
    inicio:= TRUE;
ELSIF INICIO THEN
    IF switch OR _IO_EM_DI_08 THEN
        IF PB1 OR _IO_EM_DI_09 THEN
            MODO:=TRUE;
            ELSIF MODO THEN
                M:=TRUE;
                _IO_EM_DO_03:=TRUE;
                SOL1:=TRUE;
                _IO_EM_DO_00:=TRUE;
                MODO:=FALSE;
                ELSIF S1 OR _IO_EM_DI_02 THEN
                    SOL1:=FALSE;
                    _IO_EM_DO_00:=FALSE;
                    SOL2:=TRUE;
                    _IO_EM_DO_01:=TRUE;
                ELSIF S2 OR _IO_EM_DI_03 THEN
                    SOL2:=FALSE;
                    _IO_EM_DO_01:=FALSE;
                    SOL3:=TRUE;
                    _IO_EM_DO_02:=TRUE;
                ELSIF S3 OR _IO_EM_DI_04 THEN
                    SOL3:=FALSE;
                    _IO_EM_DO_02:=FALSE;
                    M:=FALSE;
                    _IO_EM_DO_03:=FALSE;
        END IF;
```

Descargar los archivos de la práctica para completar los ejercicios propuestos.




```
IF start OR _IO_EM_DI_00 THEN
    inicio:= TRUE;
```

Esta parte del código nos indica si la tecla Start (del HMI) o la entrada “DI00” han sido presionadas se enciende el botón de inicio

```
ELSIF INICIO THEN
```

En caso de que la condición anterior no sea válida, se analiza si se presionó la tecla Inicio, en caso de que sea así, realiza las siguientes acciones dependiendo de la validez de los condicionales

```
IF switch OR _IO_EM_DI_08 THEN
```

En caso de que la tecla inicio este activa y se haya activado el switch o la entrada “DI08”, se realizara las siguientes acciones dependiendo de la validez de los condicionales

```
IF PB1 OR _IO_EM_DI_09 THEN
    MODO:=TRUE;
ELSIF MODO THEN
    M:=TRUE;
    _IO_EM_DO_03:=TRUE;
    SOL1:=TRUE;
    _IO_EM_DO_00:=TRUE;
    MODO:=FALSE;
ELSIF S1 OR _IO_EM_DI_02 THEN
    SOL1:=FALSE;
    _IO_EM_DO_00:=FALSE;
    SOL2:=TRUE;
    _IO_EM_DO_01:=TRUE;
ELSIF S2 OR _IO_EM_DI_03 THEN
    SOL2:=FALSE;
    _IO_EM_DO_01:=FALSE;
    SOL3:=TRUE;
    _IO_EM_DO_02:=TRUE;
ELSIF S3 OR _IO_EM_DI_04 THEN
    SOL3:=FALSE;
    _IO_EM_DO_02:=FALSE;
    M:=FALSE;
    _IO_EM_DO_03:=FALSE;
END_IF;
```

En caso de que “Inicio” este encendido y el “SWITCH” o la entrada “DI08” estén activas, como primera condición pregunta si “PB1” o la entrada “DI09” están activas, en caso de que se cumpla esta condición activa Modo, por otro lado, si no se cumple la primera condición, la segunda condición pregunta si Modo esta activo, en caso de que se cumpla se activa “M”, “SOL1”, se activan las salidas “DO03”, “DO00” y se desactiva Modo. En caso de que no se cumpla la primera ni segunda, la tercera condición pregunta por “S1” o por la entrada “DI02”, en caso de estar activo, se activa “SOL2”, la salida “DO01” y se desactivan “SOL1” y la salida “DO00”. En caso de que no se cumplan las 3 condiciones, la cuarta condición pregunta por “S2” o por la entrada “DI03”, en caso de estar activo, se activa “SOL3” Y LA SALIDA “DO02”, se desactiva “SOL2” y la salida “DO01”, por último, si no se cumplen estas 4 condiciones, pregunta si “S3” o la entrada “DI04” están activas, en caso de que se cumpla se desactiva “SOL3”, “M” y las salidas “DO02”, “DO03”. Esta parte del código nos activa o desactiva las válvulas de cada tanque.

```
ELSIF PB1 OR _IO_EM_DI_09 THEN
    OK:=TRUE;
```

En caso de que “Inicio” este encendido y no se cumpla la condición del “SWITCH” o la entrada “DI08” pregunta si “PB1” o la entrada “DI09” están activas, en caso de que se cumpla activa “OK”.

```
ELSIF OK THEN
    m:=TRUE;
    _IO_EM_DO_03:=TRUE;
    sol1:=TRUE;
    _IO_EM_DO_00:=TRUE;
    OK:=FALSE;
    sol3:=FALSE;
    _IO_EM_DO_02:=FALSE;
```

En caso de que “Inicio” este encendido y no se cumpla la condición del “SWITCH” o la entrada “DI08” estén activas, ni la condición “PB1” o la entrada “DI09” estén activas, pregunta si “OK” esta activa, en caso de que se cumpla activa “M”, “SOL1”, las salidas “DO03”, “DO00” y desactiva “OK”, “SOL3” y la salida “DO02”.

```
ELSE
    TP_1(m,T#10s10ms);
    done:=TP_1.ET;
```

En caso de que “Inicio” este encendido y no se cumpla la condición del “SWITCH” o la entrada “DI08” estén activas, ni la condición “PB1” o la entrada “DI09” estén activas, no “OK” esté activa crea un Temporizador de Pulso, con Enable conectado a “M” y un tiempo de 10s10ms, la salida del temporizador de pulso lo asigna a “DONE” y realiza las siguientes acciones

```
IF DONE>T#10S THEN
    sol1:=FALSE;
    _IO_EM_DO_00:=FALSE;
    sol2:=TRUE;
    _IO_EM_DO_01:=TRUE;
    DONE:=T#0S;
    IF sensor2>150 THEN
        sol2:=FALSE;
        _IO_EM_DO_01:=FALSE;
        sol3:=TRUE;
        _IO_EM_DO_02:=TRUE;
        IF s3 OR _IO_EM_DI_04 THEN
            m:=FALSE;
            _IO_EM_DO_03:=FALSE;
            sol3:=FALSE;
            _IO_EM_DO_02:=FALSE;
        END_IF;
    END_IF;
END_IF;
```

Como primera condición: Si Done es mayor a 10 segundos, activa “SOL2”, la salida “DO01”, desactiva “SOL1” la salida “DO00” y asigna a Done el valor de 0 segundos.

Si se cumple la primera condición, la segunda condición es si sensor2 es mayor a 150 activa “SOL3”, la salida”DO02”, desactiva “SOL2” y la salida “DO01”. Por último, si se cumplen las 2 primeras condiciones pregunta si “S3” o la entrada “DI04” están activas, si se cumple desactiva “M”, “SOL3”, las salidas “DO02” y “DO03”.

```
IF STOP OR _IO_EM_DI_01 THEN
    INICIO:=FALSE;
    M:=FALSE;
    _IO_EM_DO_03:=FALSE;
    SOL1:=FALSE;
    _IO_EM_DO_00:=FALSE;
    MODO:=FALSE;
    OK:=FALSE;
    SOL2:=FALSE;
    _IO_EM_DO_01:=FALSE;
    SOL3:=FALSE;
    _IO_EM_DO_02:=FALSE;
END_IF;
```

Esta última sección nos pregunta si se ha presionado “STOP” o la entrada “DI01”, en caso de que se cumpla desactiva todas las variables y salidas del sistema