Griffith School of Engineering

Project 2

Wastewater Treatment System

Contents

SCADA Electrics Project 2: Wastewater Treatment Plant

1.0 INTRODUCTION

Each State and territories wastewater are treated using facilities known as wastewater treatment plants. Depending on municipal requirements and specifications each plant has several or more stages in which water is treated. Two of these main processes are known as *Primary Clarification* and *Aeration Basin* (Figure 1).

1.1 Primary Clarifier

In a *Primary Clarifier* tank, wastewater enters the tank after screening and grit removal eliminates large solids from the untreated water. A clarifier tank controls flow in and out of its system to ensure most organic matter, still present in the wastewater, has time to settle to the bottom of tank. This is also achieved by using a machine that turns within the tank to assist sediment settling (the matter known as sludge is removed from the bottom of clarifier tank). If wastewater flows to quickly into primary and aeration, without having enough time for most of the sludge to settle, it can negatively impact the next stages in the process. In addition, if the flowrate entering the tank is to slow it will also affect the system.

1.2 Aeration Basin

In an Aeration basin, numerous air blowers usually placed on bottom of basin create bubbles of oxygen throughout water. Microorganisms also known as returned activated sludge (RAS) need oxygen to survive while they breakdown left over matter within the effluent. RAS will multiply till there is no oxygen in the water hence added oxygen is needed to create a balanced environment (this process is known as aerobic digestion). If oxygen is to low the bacteria will not survive and if to high certain particulates, the RAS are trying to eat will grow, which defeats eliminating solids from wastewater.

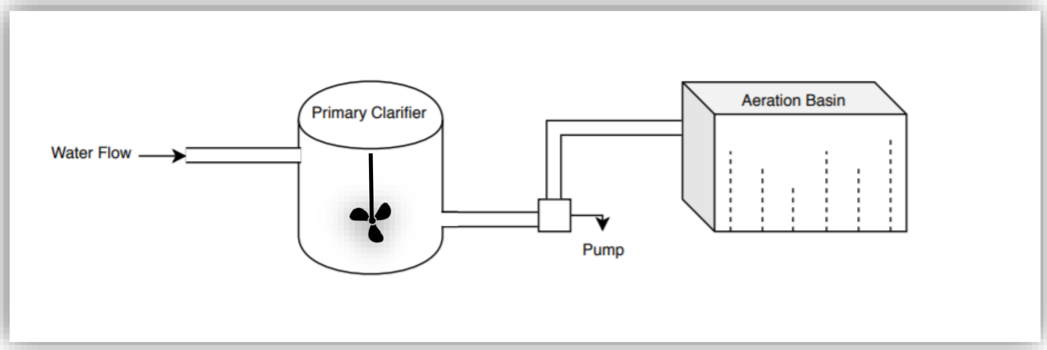


Figure 1. Primary Clarifier and Aeration Basin Diagram.

2. OBJECTIVE OF PROJECT

In this project, you are required to use the SCADA system to control and monitor *water level* and *machine speed* of a primary clarifier tank and *water* and *oxygen level* of an aeration basin tank within a wastewater treatment plant. In addition, you must implement a fault to occur when water level, machine speed and oxygen level is too high and when too low.

For example, when water level has passed the value you set for high water level an LED light will turn on indicating a water level fault has occurred. This means pump has been turned off. Once water level is back in range, using scroll bar and LED bar, press button under LED light to turn fault off (which activates pump). Remember the high and low values are pre-set in *Remote Connect* then used in your Figure Block Diagram (FBD) program.

Note: *Water level for both tanks can have the same analog input and output in SCADA because if a change in flowrate occurs in one tank a change in flowrate will occur in the other (pump will stop waterflow to both). Also, it is up to you to determine what low and high levels a fault should occur (Example: at 80% and at 20% a fault will occur).*

Most steps to set up this project are the same as previous project (Project 1) except you will be using Function Block Diagram (FBD) language instead of Ladder (LD) in **Logic Editor** and high and low limits will be changed. This will be explained in more detail under the heading: *Function Block Diagram* and *Project*.

3. Function Block Diagram

This section contains the basics of Function Block Diagram (FBD) logic which includes variable types, most common Function Blocks, *Object calling* with the differences between analog and digital input/output.

3.1 Review: Digital and Analog

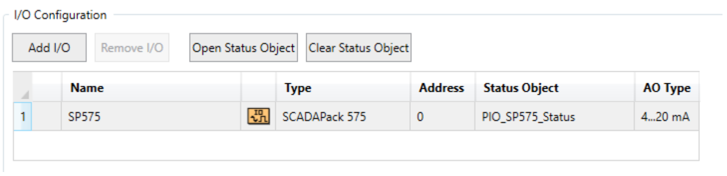
3.1.1 Digital Input (DI) and Digital Output (DO)

Both are Boolean variables that have two stages, TRUE or bit equal to 1 and False or bit equal to 0. The SCADApack contains 16 DIs and 8 DOs. The 16 DIs are divided in 8 buttons (DI1-DI8) and 8 switches (DI9-DI16). Buttons have a default Boolean state FALSE. When pressed the state goes to TRUE until it is released. Switches also have two stages however, they hold TRUE when high, and FALSE when low. The DOs (DO1-DO8) are represented by LEDs which on state FALSE will be off and on state TRUE will be on.

3.1.2 Analog Input (AI) and Analog Output (AO)

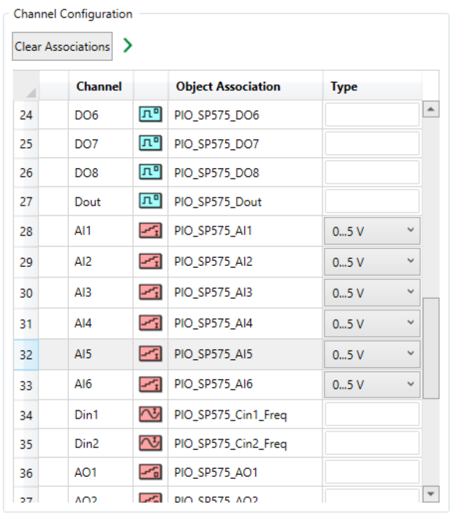
Both have a range of voltage or current variables. When creating your project remember to verify if the configuration of the *AO Type* (under object named SP575) in tab '**Configuration – SPx70 Controller'>Configuration>Physical I/O>Local** is 4 – 20 mA (Figure 2). After checking the AO Type, scroll the same window to the right where *Channel Configuration* is located. Search under *Channel* for all AI objects and verify if *Type* is set to 0…5V (Figure 3Figure 3).

Note: Remember that the usual configuration is to define the Analog IOs to 1…5V. This is to ensure there will be no damage to the control panel when using AI AND AO, because the sensor will read 0 V if it is faulty. This configuration disadvantage is losing resolution from the sensor reading.



	Name	Type	Address	Status Object	AO Type
1	SP575	SCADAPack 575	0	PIO_SP575_Status	4...20 mA

Figure 2. Configuring all Analog Outputs current to 4…20mA.

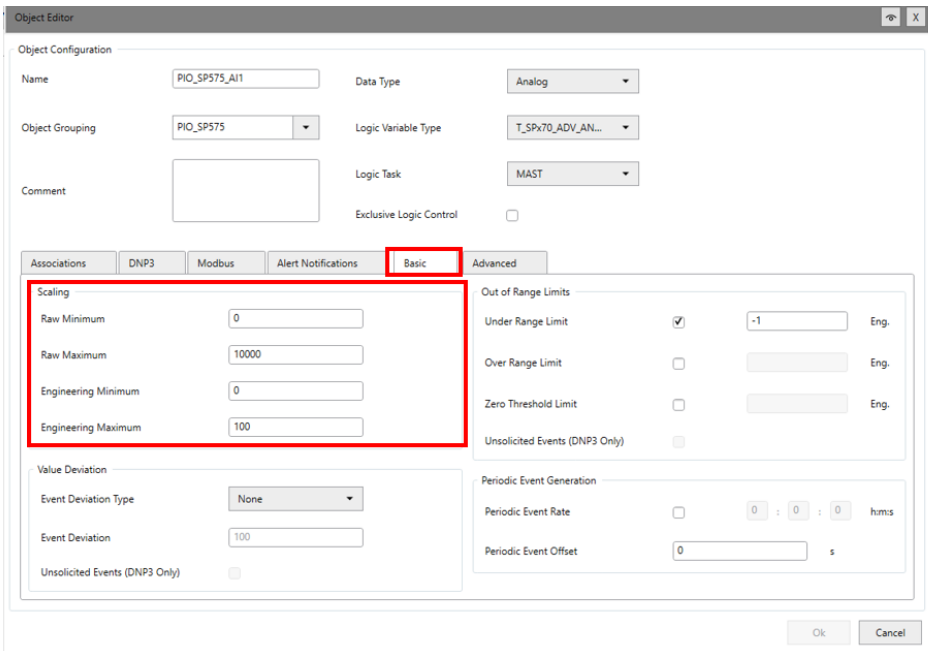


	Channel	Object Association	Type
24	DO6	PIO_SP575_DO6	
25	DO7	PIO_SP575_DO7	
26	DO8	PIO_SP575_DO8	
27	Dout	PIO_SP575_Dout	
28	AI1	PIO_SP575_AI1	0...5 V
29	AI2	PIO_SP575_AI2	0...5 V
30	AI3	PIO_SP575_AI3	0...5 V
31	AI4	PIO_SP575_AI4	0...5 V
32	AI5	PIO_SP575_AI5	0...5 V
33	AI6	PIO_SP575_AI6	0...5 V
34	Din1	PIO_SP575_Cin1_Freq	
35	Din2	PIO_SP575_Cin2_Freq	
36	AO1	PIO_SP575_AO1	
37	AO2	PIO_SP575_AO2	

Figure 3. Configuring all Analog Inputs to between 0…5V.

To set parameters for an Analog Input (AI) or an Analog Output (AO) you need to open its **Object Editor** window ‘Configuration – SPx70 Controller’>Configuration>Physical I/O>Local>Channel Configuration, double click object name or ‘Configuration – SPx70 Controller’>Objects>Object Configuration, double click on object name. Once **Object Editor** window is open (Figure 4) there will be options under *logic variable type*: T_SPx70_INT (integer), T_SPx70_DINT (double integer), T_SPx70_UINT (unsigned integer), T_SPx70_REAL and ADV_ANALOG. The default is T_SPx70_DINT, however it is recommended to use ADV_ANALOG. One advantage of this variable type is having access to more object variables with different logic types. An example is activating the *Alert Notifications* where you can enable four low limits (L1, L2, L3 and L4) and four high limits (H1, H2, H3 and H4), which the state variable for each is a BOOLEAN. Wherever a limit has been reached the state of this limit will be TRUE, otherwise it will be FALSE. Therefore, an AI state variable can be used in a logic structure along with a DI, like an AND gate. Remember the limit values depend on *Engineering Minimum* and *Engineering Maximum* defined under the tab *Basic* in **Object Editor**.

Note: *Analog Output (AO) default values should be: Engineering Maximum equal to 1000 and Raw Maximum to 100.*


Object Editor

Object Configuration

Name	PIO_SP575_AI1	Data Type	Analog
Object Grouping	PIO_SP575	Logic Variable Type	T_SPx70_ADV_AN...
Comment		Logic Task	MAST
		Exclusive Logic Control	☐

Associations | DNP3 | Modbus | Alert Notifications | Basic | Advanced

Scaling

Raw Minimum	0
Raw Maximum	10000
Engineering Minimum	0
Engineering Maximum	100

Out of Range Limits

Under Range Limit	✓	-1	Eng.
Over Range Limit	☐		Eng.
Zero Threshold Limit	☐		Eng.
Unsolicited Events (DNP3 Only)	☐		

Value Deviation

Event Deviation Type	None
Event Deviation	100
Unsolicited Events (DNP3 Only)	☐

Periodic Event Generation

Periodic Event Rate	☐	0 : 0 : 0	h:m:s
Periodic Event Offset	0		s

Ok Cancel

Figure 4. Object Editor Window to set Input and Output parameters.

3.2 Digital Gates/Blocks

Basic digital gates are OR, AND, NOT, XOR, ROTATE and SHIFT. These function blocks are found in the tab **EDIT>'FFB Input Assistant'** (Figure 5) or on tool bar with the symbol of a function block with a wand. The short key for the FFB Input Assistant is **CTRL+I**. The *Function Input Assistant* window will open (Figure 6), then to select a function block write in the text field next to *FFB type: OR*. It is possible to select the function block manually by clicking at the "**…**" at the end of *FFB type* text field, which will open another window named *FFB Type Selection* (Figure 7). Inside the *FFB Type Selection* select **Libset V11.1>Base Lib>Logic** to have access of all Logic Function Blocks.

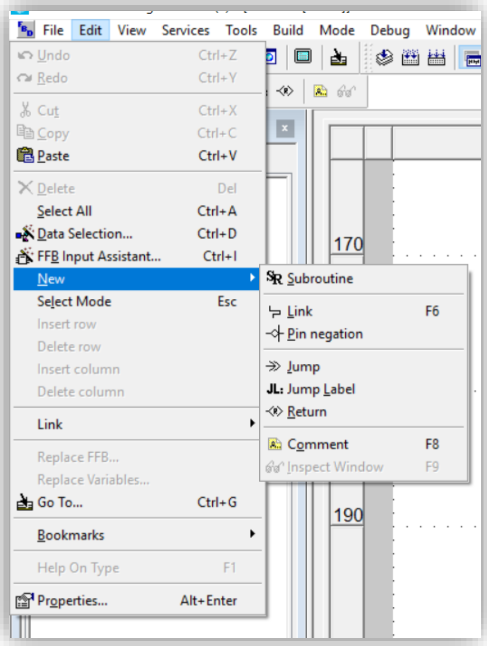


Figure 5. Selecting the FFB Input Assistant for accessing to the function blocks.

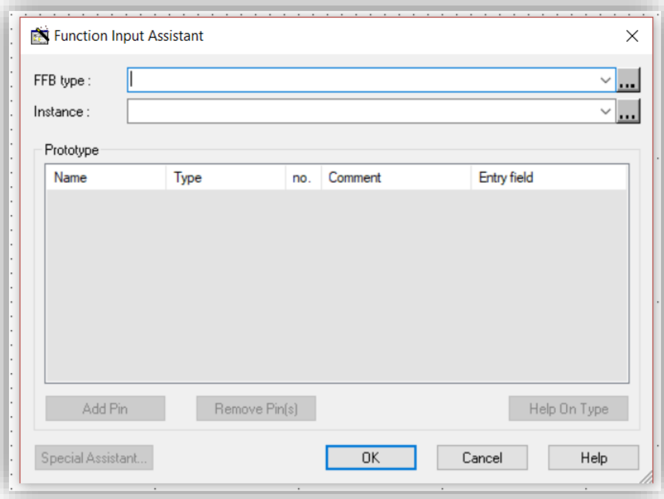
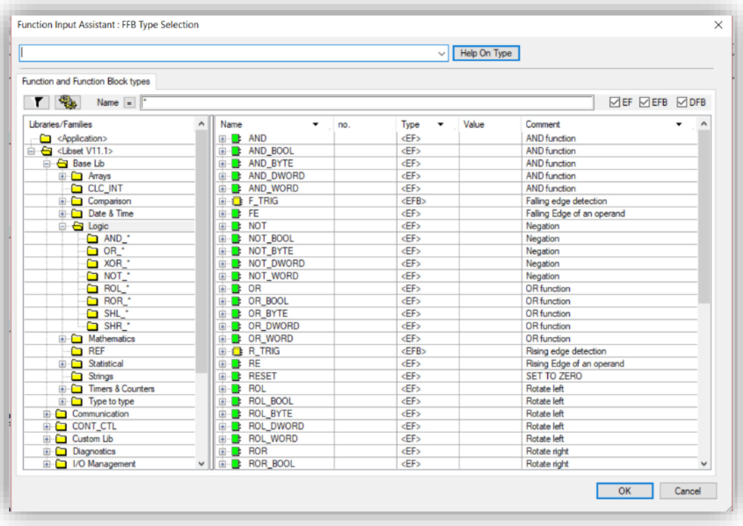


Figure 6. Function Input Assistant window.


Function Input Assistant : FFB Type Selection

Function and Function Block types

Name = * ☑EF ☑EFB ☑DFB

Libraries/Families
- <Application>
- <Libset V11.1>
  - Base Lib
    - Arrays
    - CLC_INT
    - Comparison
    - Date & Time
    - Logic
      - AND_*
      - OR_*
      - XOR_*
      - NOT_*
      - ROL_*
      - ROR_*
      - SHL_*
      - SHR_*
    - Mathematics
    - REF
    - Statistical
    - Strings
    - Timers & Counters
    - Type to type
  - Communication
  - CONT_CTL
  - Custom Lib
  - Diagnostics
  - I/O Management

Name	no.	Type	Value	Comment
AND		<EF>		AND function
AND_BOOL		<EF>		AND function
AND_BYTE		<EF>		AND function
AND_DWORD		<EF>		AND function
AND_WORD		<EF>		AND function
F_TRIG		<EFB>		Falling edge detection
FE		<EF>		Falling Edge of an operand
NOT		<EF>		Negation
NOT_BOOL		<EF>		Negation
NOT_BYTE		<EF>		Negation
NOT_DWORD		<EF>		Negation
NOT_WORD		<EF>		Negation
OR		<EF>		OR function
OR_BOOL		<EF>		OR function
OR_BYTE		<EF>		OR function
OR_DWORD		<EF>		OR function
OR_WORD		<EF>		OR function
R_TRIG		<EFB>		Rising edge detection
RE		<EF>		Rising Edge of an operand
RESET		<EF>		SET TO ZERO
ROL		<EF>		Rotate left
ROL_BOOL		<EF>		Rotate left
ROL_BYTE		<EF>		Rotate left
ROL_DWORD		<EF>		Rotate left
ROL_WORD		<EF>		Rotate left
ROR		<EF>		Rotate right
ROR_BOOL		<EF>		Rotate right

Figure 7. Inserting a variable on the input of a OR Function block.

As an example, we are going to create a simple code with three Digital Inputs and one Digital Output where an OR gate will be used. When selecting a common OR Function Block observe that the input and output accept ANY_BIT as variable type, which means that the system will automatically select the digital variable type according to the chosen variable type. If you need a specific variable type, select an appropriate Function Block, like OR_BOOL if all inputs and output are BOOLEAN (single bit).

Insert to your FBD code file an OR Function Block, then double click on the input *pin 1* to open a small text field (Figure 8) to select the input variable. On this field, you force a BOOLEAN value by typing 0 or 1. However, we want to use switches as Digital Inputs here, so select "…" to open the *FBD: Instance Selection* which contains all objects from the SCADApack. In this window, find the PIO_SP575_DI15 object, then open the object to access the object VALUE for which the variable type is a BOOLEAN (Figure 9). Next step is to associate Input *Pin 2* to PIO*_SP575_DI14.Value*. Then add a second OR Function Block, connecting one of its inputs to the output of the first OR Function Block. To link two pins, click on the **LINK** symbol on the tool bar or press **F6**. After this, select *PIO_SP575_DI13.Value* as the other input of the second OR and *PIO_SP575_DO7.Value* (*LED 7*) as its output. When the program is like Figure 10, build the code by clicking on the build image or on the tab **Build>Build Changes** or press **CTRL+B**, then *WRITE* it to your SCADApack (In Remote Connect) and connect the Logic Editor by pressing **CTRL+K** or going to the tab **Mode>Connect**.

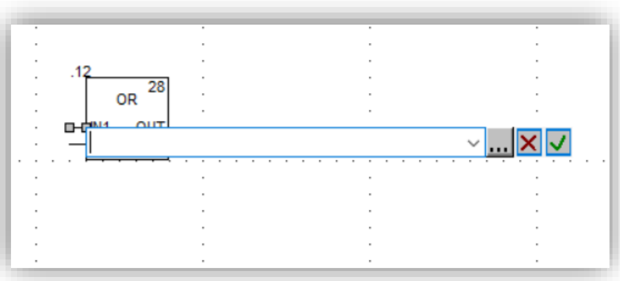
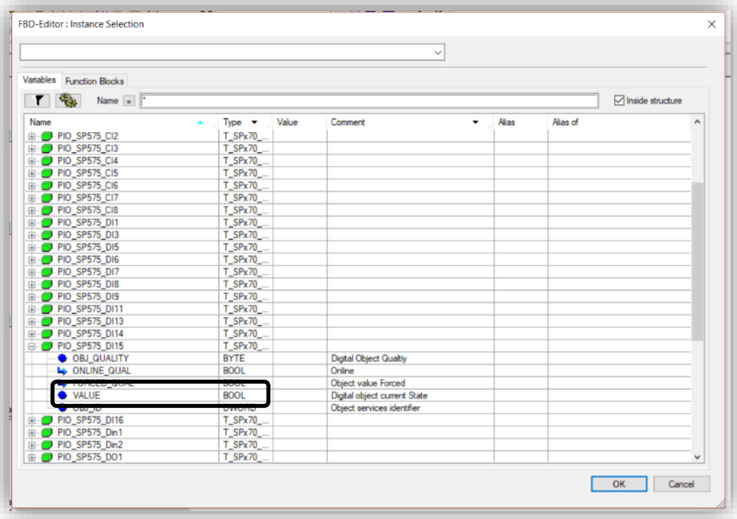


Figure 8. Inserting a variable on the input of an OR Function block.



FBD-Editor : Instance Selection

Variables | Function Blocks

Name = *

☑ Inside structure

Name	Type	Value	Comment	Alias	Alias of
PIO_SP575_CI2	T_SPx70_...				
PIO_SP575_CI3	T_SPx70_...				
PIO_SP575_CI4	T_SPx70_...				
PIO_SP575_CI5	T_SPx70_...				
PIO_SP575_CI6	T_SPx70_...				
PIO_SP575_CI7	T_SPx70_...				
PIO_SP575_CI8	T_SPx70_...				
PIO_SP575_DI1	T_SPx70_...				
PIO_SP575_DI3	T_SPx70_...				
PIO_SP575_DI5	T_SPx70_...				
PIO_SP575_DI6	T_SPx70_...				
PIO_SP575_DI7	T_SPx70_...				
PIO_SP575_DI8	T_SPx70_...				
PIO_SP575_DI9	T_SPx70_...				
PIO_SP575_DI11	T_SPx70_...				
PIO_SP575_DI13	T_SPx70_...				
PIO_SP575_DI14	T_SPx70_...				
PIO_SP575_DI15	T_SPx70_...				
OBJ_QUALITY	BYTE		Digital Object Qualtiy		
ONLINE_QUAL	BOOL		Online		
FORCED_QUAL	BOOL		Object value Forced		
VALUE	BOOL		Digital object current State		
OBJ_ID	DWORD		Object services identifier		
PIO_SP575_DI16	T_SPx70_...				
PIO_SP575_Din1	T_SPx70_...				
PIO_SP575_Din2	T_SPx70_...				
PIO_SP575_DO1	T_SPx70_...				

OK | Cancel

Figure 9. Object selection window.

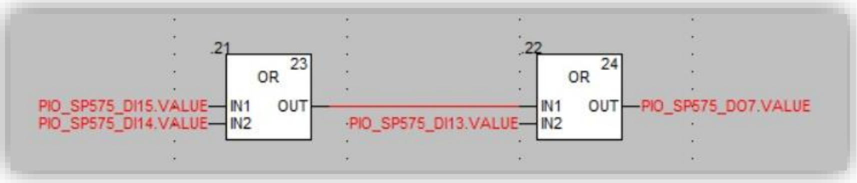


Figure 10. Example code.

Once the Logic Editor is connected to the SCADApack, all Digital Inputs and Outputs will have the colour red for BOOLEAN state equal to **FALSE**, and colour green equal to **TRUE** state. For this code, if any switch is up, the output (*LED 7*) will be **TRUE**. Figure 11 is an illustration of a scene when *Switch 6* and *Switch 7* (DI14 and DI15) are up. There is another way to write the same code using only one OR Function Block. Normally, each Function Block allows to have up to 32 inputs variables, so to open the other input pins simply click on the Function Block and drag down the Block Diagram. Figure 12 shows this program written using three inputs on an OR Block.

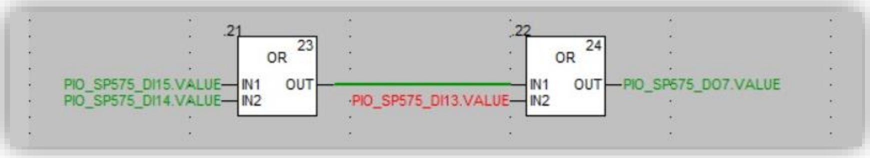


Figure 11. Example of the code running when two switches are on which makes the output TRUE.

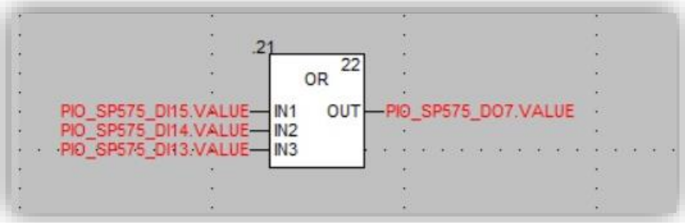
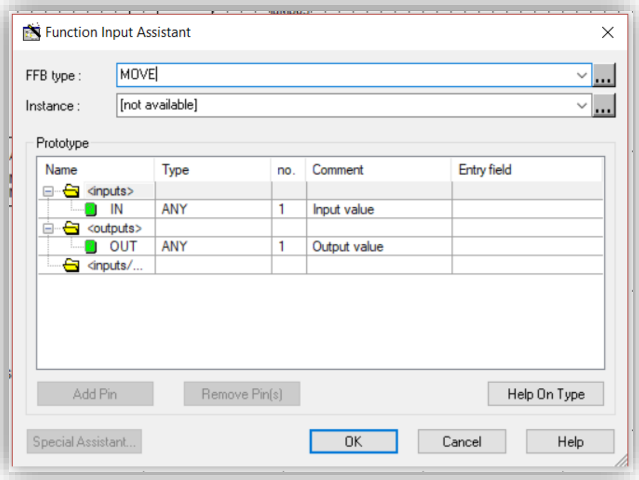


Figure 12. Same project using only one OR function block.

3.3 Basic Mathematic Functions

3.3.1 Move Function - Equal

MOVE Function Block is very useful and can receive *ANY* type of variable, digital or integer (Figure 13). MOVE means an equal mathematical operation, so it can directly connect an input to an output. If it is desired to make an LED follow the same state as a switch, it should be used as a MOVE Function Block as illustrated by Figure 14. To test this Function Block, add to code a MOVE Function Block that has as input *PIO_SP575_DI16.Value* (Switch 8) and an output *PIO_SP575_DO8.Value* (LED 8). Then, build and write the code into the SCADApack to observe the connection made.



FFB type : MOVE

Instance : [not available]

Prototype

Name	Type	no.	Comment	Entry field
<inputs>				
IN	ANY	1	Input value	
<outputs>				
OUT	ANY	1	Output value	
<inputs/...				

Figure 13. Move Function Block.

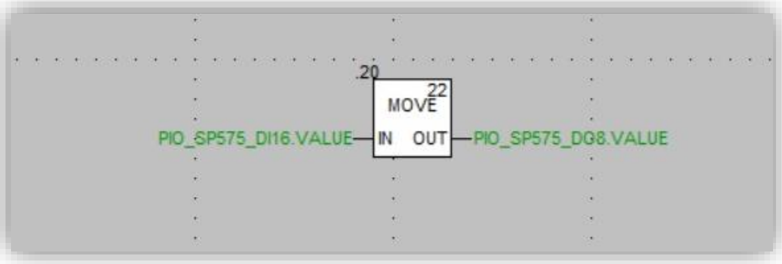
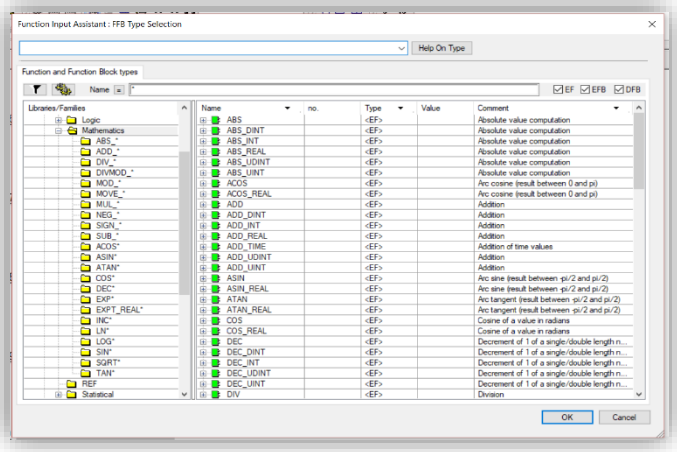


Figure 14. To make an Output follow the same state as an Input.

3.3.2 Basic Mathematics Functions

Basic Mathematic Function Blocks are the operations of: addition, subtraction, division and multiplication. These operations can be used to manage and adjust integer values for Analog Outputs. There are other Math Operations that can be used inside the logic, like Absolute value, Exponential, Logarithm, Square root, Sin, Cos, Tan, Arc Sin, Arc Cos and Arc Tan, however for this project and this course these operations are not necessary.

To access all Mathematic Function Blocks, open the *FBB Type Selection* window and go to the folder **Libset V11.1>Base Lib>Mathematics** (Figure 15). Again, there are Function Blocks that accept any variable type and others that are type specific, as an example, ADD and ADD_DINT.

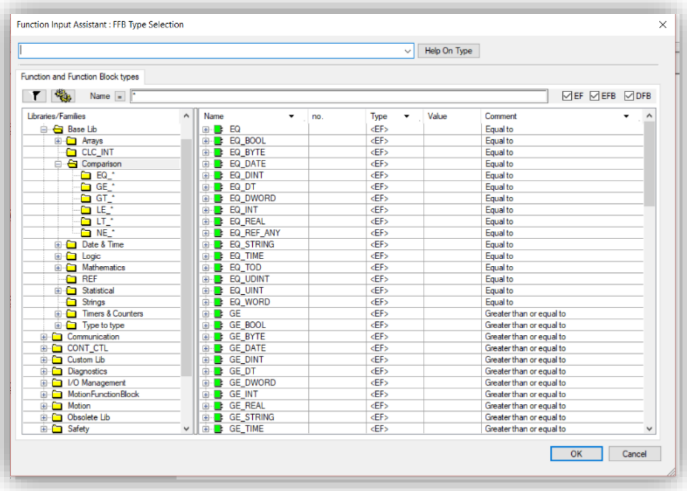


Name	no.	Type	Value	Comment
ABS		<EF>		Absolute value computation
ABS_DINT		<EF>		Absolute value computation
ABS_INT		<EF>		Absolute value computation
ABS_REAL		<EF>		Absolute value computation
ABS_UDINT		<EF>		Absolute value computation
ABS_UINT		<EF>		Absolute value computation
ACOS		<EF>		Arc cosine (result between 0 and pi)
ACOS_REAL		<EF>		Arc cosine (result between 0 and pi)
ADD		<EF>		Addition
ADD_DINT		<EF>		Addition
ADD_INT		<EF>		Addition
ADD_REAL		<EF>		Addition
ADD_TIME		<EF>		Addition of time values
ADD_UDINT		<EF>		Addition
ADD_UINT		<EF>		Addition
ASIN		<EF>		Arc sine (result between -pi/2 and pi/2)
ASIN_REAL		<EF>		Arc sine (result between -pi/2 and pi/2)
ATAN		<EF>		Arc tangent (result between -pi/2 and pi/2)
ATAN_REAL		<EF>		Arc tangent (result between -pi/2 and pi/2)
COS		<EF>		Cosine of a value in radians
COS_REAL		<EF>		Cosine of a value in radians
DEC		<EF>		Decrement of 1 of a single/double length n...
DEC_DINT		<EF>		Decrement of 1 of a single/double length n...
DEC_INT		<EF>		Decrement of 1 of a single/double length n...
DEC_UDINT		<EF>		Decrement of 1 of a single/double length n...
DEC_UINT		<EF>		Decrement of 1 of a single/double length n...
DIV		<EF>		Division

Figure 15. Mathematic Function Blocks.

3.4 Comparison Function

Comparison Function Blocks contain all comparison operations like equal to (**EQ**), greater than (**GT**), greater or equal than (**GE**), less than (**LT**), less or equal than (**LE**). The input of these comparison blocks can be numerical or digital values. However, the output is always BOOLEAN. Therefore, these Function Blocks are commonly used with **JUMP** which in other programming languages is equivalent to an *if statement*. Inside the *FFB Type Selection* window, select the folder **Libset V11.1>Base Lib>Comparison** (Figure 16) to have access to all comparison function blocks.



Name	no.	Type	Value	Comment
EQ		<EF>		Equal to
EQ_BOOL		<EF>		Equal to
EQ_BYTE		<EF>		Equal to
EQ_DATE		<EF>		Equal to
EQ_DINT		<EF>		Equal to
EQ_DT		<EF>		Equal to
EQ_DWORD		<EF>		Equal to
EQ_INT		<EF>		Equal to
EQ_REAL		<EF>		Equal to
EQ_REF_ANY		<EF>		Equal to
EQ_STRING		<EF>		Equal to
EQ_TIME		<EF>		Equal to
EQ_TOD		<EF>		Equal to
EQ_UDINT		<EF>		Equal to
EQ_UINT		<EF>		Equal to
EQ_WORD		<EF>		Equal to
GE		<EF>		Greater than or equal to
GE_BOOL		<EF>		Greater than or equal to
GE_BYTE		<EF>		Greater than or equal to
GE_DATE		<EF>		Greater than or equal to
GE_DINT		<EF>		Greater than or equal to
GE_DT		<EF>		Greater than or equal to
GE_DWORD		<EF>		Greater than or equal to
GE_INT		<EF>		Greater than or equal to
GE_REAL		<EF>		Greater than or equal to
GE_STRING		<EF>		Greater than or equal to
GE_TIME		<EF>		Greater than or equal to

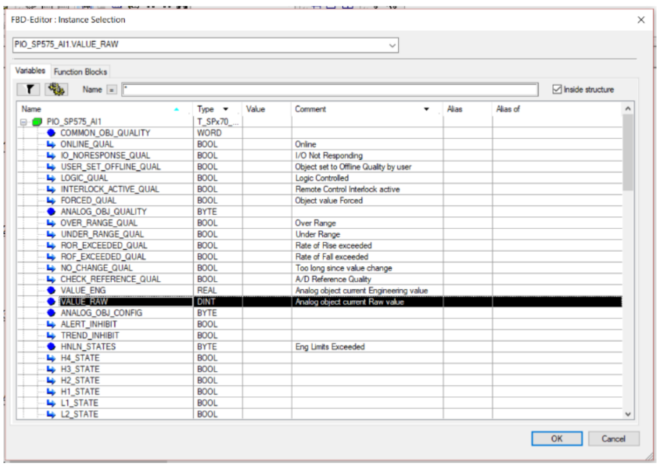
Figure 16. Comparison Function Block options.

In the **Logic Editor**, select a greater or equal than (GE). Insert on pin IN1 the desired analog scale variable and on pin IN2 the value to be compared. Be careful to follow this order, as the comparison follows the rule:

$$IN1 \geq IN2$$

Remember that the Engineering Value goes from 0 to 100, while Raw Value is from 0 to 1,000, as defined inside the analog object settings. Observing Figure 17, the Engineering scale is REAL value (it has one decimal number, in other words it is floating number), while the Raw scale is a double integer (DINT).

Note: The analog inputs usually are linear, however some SCADApack might have analog IOs that have an exponential behaviour.



PIO_SP575_AI1.VALUE_RAW

Name = *

Name	Type	Value	Comment	Alias	Alias of
PIO_SP575_AI1	T_SPx70_...				
COMMON_OBJ_QUALITY	WORD				
ONLINE_QUAL	BOOL		Online		
IO_NORESPONSE_QUAL	BOOL		I/O Not Responding		
USER_SET_OFFLINE_QUAL	BOOL		Object set to Offline Quality by user		
LOGIC_QUAL	BOOL		Logic Controlled		
INTERLOCK_ACTIVE_QUAL	BOOL		Remote Control Interlock active		
FORCED_QUAL	BOOL		Object value Forced		
ANALOG_OBJ_QUALITY	BYTE				
OVER_RANGE_QUAL	BOOL		Over Range		
UNDER_RANGE_QUAL	BOOL		Under Range		
ROR_EXCEEDED_QUAL	BOOL		Rate of Rise exceeded		
ROF_EXCEEDED_QUAL	BOOL		Rate of Fall exceeded		
NO_CHANGE_QUAL	BOOL		Too long since value change		
CHECK_REFERENCE_QUAL	BOOL		A/D Reference Quality		
VALUE_ENG	REAL		Analog object current Engineering value		
VALUE_RAW	DINT		Analog object current Raw value		
ANALOG_OBJ_CONFIG	BYTE				
ALERT_INHIBIT	BOOL				
TREND_INHIBIT	BOOL				
HNLN_STATES	BYTE		Eng Limits Exceeded		
H4_STATE	BOOL				
H3_STATE	BOOL				
H2_STATE	BOOL				
H1_STATE	BOOL				
L1_STATE	BOOL				
L2_STATE	BOOL				

Figure 17. Value Raw of Analog Input (AI1).

In Figure 18, we are setting the input pins IN1 as *PIO_SP575_AI1.VALUE_ENG* and IN2 as 30.0 (30% of the total range). And finally, set the output as *PIO_SP575_DO1.Value* (LED 1). BUILD the code and WRITE it to the SCADApack. Observe that the LED 1 will light up every time that the *Analog Input Engineering Value* is equal to or higher than 30.0.

Now, COPY and PASTE the previous block. Change the value of the *Input 2* to 50.0 and the output to PIO*_SP575_DO2.Value* (LED 2). Rebuild the code and rewrite it to the SCADApack.

Note: *You can change the Function Block to Less Than (LE) and observe the opposite result from the GE Function Block.*

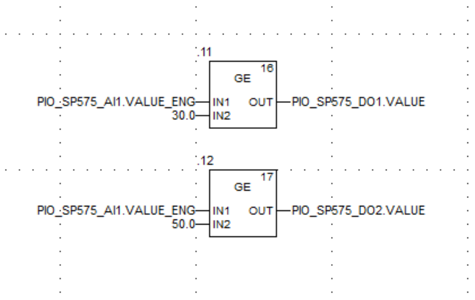
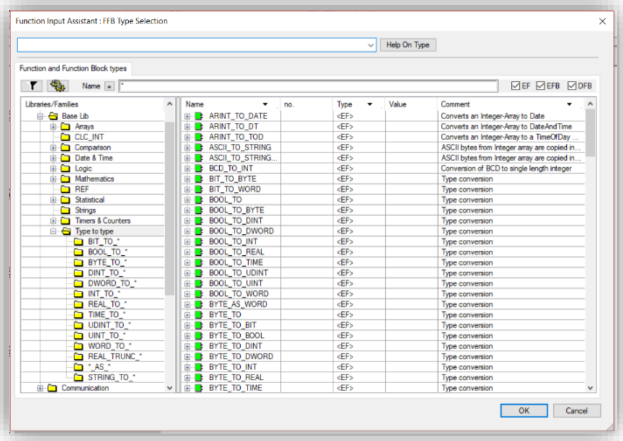


Figure 18. Greater or Equal Than Function Blocks.

3.5 Type Conversion Function

When an operation between Digital Values and Analog Values are needed, it is necessary to convert one variable type into another. For this propose, Type Conversion Function Blocks are used, located in **Libset V11.1>Base Lib>Type to type** (Figure 19).



Name	no.	Type	Value	Comment
ARINT_TO_DATE		<EF>		Converts an Integer-Array to Date
ARINT_TO_DT		<EF>		Converts an Integer-Array to DateAndTime
ARINT_TO_TOD		<EF>		Converts an Integer-Array to a TimeOfDay ...
ASCII_TO_STRING		<EF>		ASCII bytes from Integer array are copied in...
ASCII_TO_STRING...		<EF>		ASCII bytes from Integer array are copied in...
BCD_TO_INT		<EF>		Conversion of BCD to single length integer
BIT_TO_BYTE		<EF>		Type conversion
BIT_TO_WORD		<EF>		Type conversion
BOOL_TO		<EF>		Type conversion
BOOL_TO_BYTE		<EF>		Type conversion
BOOL_TO_DINT		<EF>		Type conversion
BOOL_TO_DWORD		<EF>		Type conversion
BOOL_TO_INT		<EF>		Type conversion
BOOL_TO_REAL		<EF>		Type conversion
BOOL_TO_TIME		<EF>		Type conversion
BOOL_TO_UDINT		<EF>		Type conversion
BOOL_TO_UINT		<EF>		Type conversion
BOOL_TO_WORD		<EF>		Type conversion
BYTE_AS_WORD		<EF>		Type conversion
BYTE_TO		<EF>		Type conversion
BYTE_TO_BIT		<EF>		Type conversion
BYTE_TO_BOOL		<EF>		Type conversion
BYTE_TO_DINT		<EF>		Type conversion
BYTE_TO_DWORD		<EF>		Type conversion
BYTE_TO_INT		<EF>		Type conversion
BYTE_TO_REAL		<EF>		Type conversion
BYTE_TO_TIME		<EF>		Type conversion

Figure 19. Type Conversion Function Block Folder.

3.6 Jump and Label

Function Block Diagram is a sequential programming language which means that it executes each line at a time. To avoid a certain part of the code, **JUMP** Function is used where the program will execute next line of the label associate with jump arrow. In a low-level language, **JUMP** is called a conditional statement and can be compared to an, *if statement* or *while statement* in a high-level language.

In this section, there are two code examples to introduce ways to use this function. First, a simple code that contains one MOVE and one AND Function Block. Add to your code the MOVE Function Block associating *PIO_SP575_DI16.Value* (Switch 8) to the Input and make the Output a **JUMP** (Call it Jump01). To create a Jump, go to the tab **Edit>New>Jump** (Figure 20) or search for the arrow symbol in the toolbars.

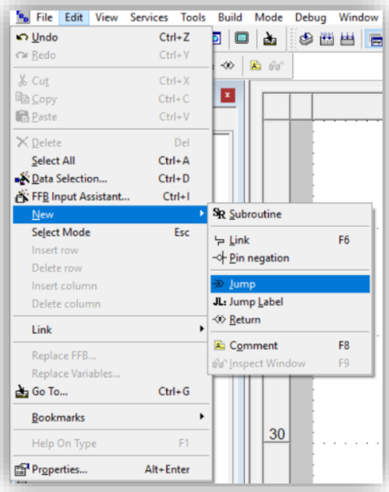


Figure 20. Location of Jump Function.

After, add the AND Function Block below the MOVE, associating *PIO_SP575_DI14.Value* (Switch 6) to *Input 1*, *PIO_SP575_DI15.Value* to Input 2 (Switch 7) and *PIO_SP575_DO8.Value* (LED 8) to output. The final step is to add the Jump Label, located under the AND Function Block by creating the Jump01 again in the tab **Edit>New>Jump**. **This element must be in a line where no other variable or block is in front of it, otherwise an error will occur in the code**. So, add the Jump Label below the AND Function Block, naming it Jump01 and double click on the Jump arrow, then in the jump window, it will be possible to select Jump01. Finally, BUILD the code and WRITE it to the SCADApack. Observe the results in Figure 21 to 24. When the *Switch 8 (DI16)* is on, the **JUMP** Function will be activated, and the program won't go through the AND Function Block. Therefore, while the Jump01 is ON, the *LED 8* (DO8) won't be on, even when both inputs are on TRUE state, Figure 22 However, if the **JUMP** is turned OFF, the LED will light up, Figure 23. Later, when the **JUMP** is activated again while the LED is on, and the Switches 6 and 7 go to FALSE state after, then the LED will continue ON, although both inputs are low state, Figure 24.

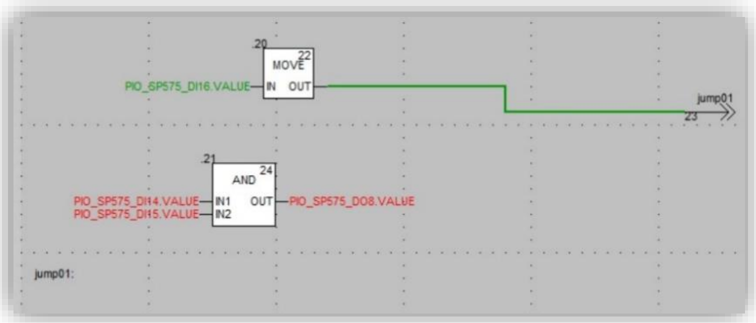


Figure 21. DI16 (switch8) is active, Jump Function is running. It will pass the AND Block.

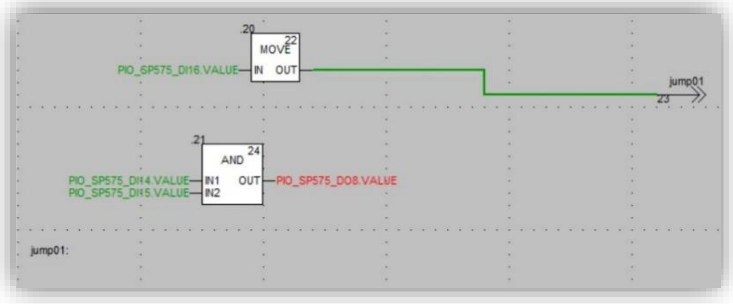


Figure 22. LED (DO8) still OFF even though both inputs on AND Block are set to TRUE, due to Jump still ON.

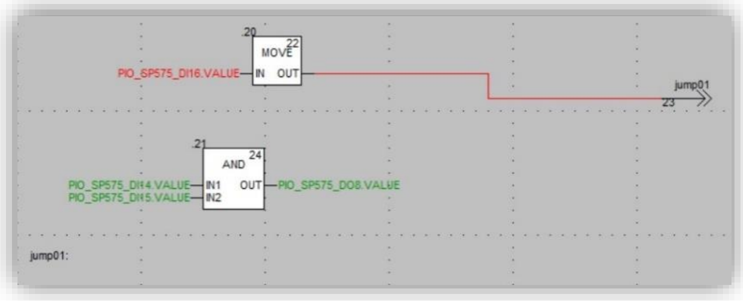


Figure 23. LED (DO8) will turn on when Jump has turned OFF and AND Block inputs are TRUE.

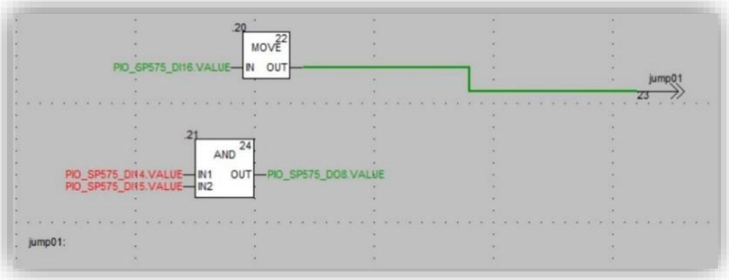


Figure 24. If LED (DO8) was ON before the Jump Function went ON, LED will stay ON (Even when inputs for AND Block are OFF).

Next code example, uses the previous code from the Comparison section (Figure 25). First of all, change the order of the GT Blocks in a way that the first block is a comparison to 50 and the second to 20. Then create a Jump Label at the first line called "Begin" and link a Jump to each block output, which associate to Jump01 and Jump02. After, add two MOVE Function Blocks. The first input set to '0' and as output the *PIO_SP575_AO1.VALUE_ENG*, and the second has an input '1' and a Jump function to "Begin" as output.

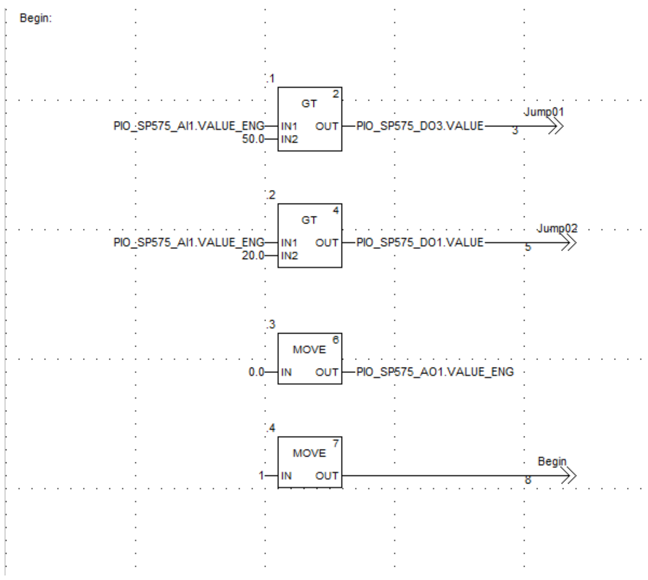


Figure 25. GT/Jump Function Block Code.

Next steps are setting the Jump logic. First, add the "Jump01" Label and two MOVE Function Blocks (Figure 26). On this part, both MOVE blocks have as input numerical value '7' and '1', respectively. And, as output, the *PIO_SP575_AO1.VALUE_ENG* and a Jump to "Begin". After, add the "Jump02" Label and one MOVE Function. The MOVE block will receive an input '3' and *PIO_SP575_AO1.VALUE_ENG* as output.

BUILD and WRITE the code to SCADApack. Observe, now that when the analog input is low none of the GT Blocks will activate the Jumps, so the analog output will be '0', then the MOVE block will jump to the 'begin' logic again. However, when one of the GT Blocks have a TRUE output, then it will execute the logic from the Jump label on its output.

Important: *The order of the GT Blocks matter a lot, because if the lower value comes first it will always be on and its jump will activate, ignoring the next code section that contains a higher value.*

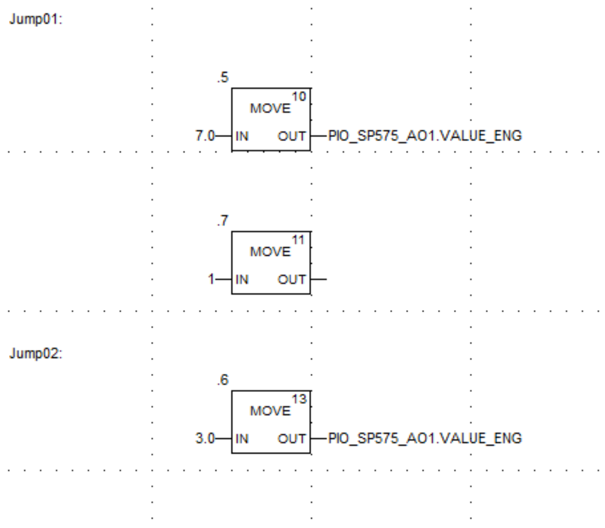

Jump01:
.5
MOVE 10
7.0 IN OUT PIO_SP575_AO1.VALUE_ENG
.7
MOVE 11
1 IN OUT
Jump02:
.6
MOVE 13
3.0 IN OUT PIO_SP575_AO1.VALUE_ENG

Figure 26. Jump Function Code.

4. Project

In this section you will develop the proposed project using Function Block Diagram Code (FBD) and control panel (Figure 27).

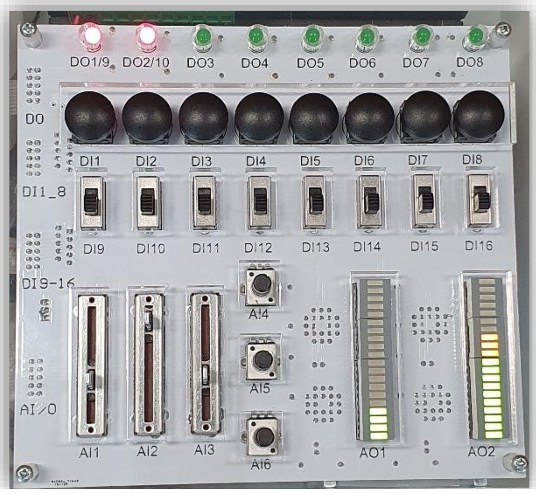


Figure 27. Control Panel connected to SCADAPack used for Project.

4.1 Step 1 – Water Level

You will need to display the water level for Tank 1 and Tank 2 via LED Bar (analog output, AO1). A Function Block (or a number of Function Blocks) is needed to connect one analog input (choose from AI1 to AI6) to an analog output (AO1) so water level can be viewed and controlled via the control panel (Figure 28).

Note: *A MOVE Function Block could work well here.*

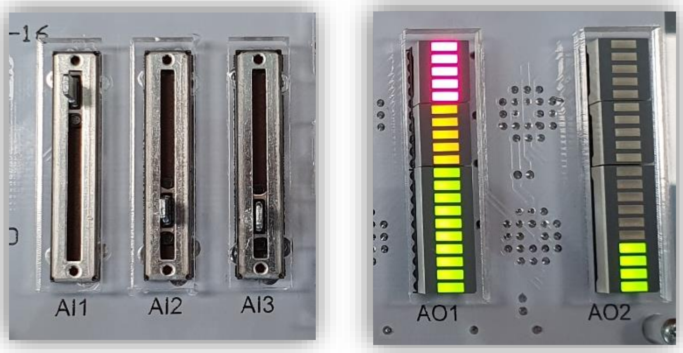


Figure 28. AI1 for analog input and AO1 analog output are connected.

Note: *Water level is same for Tank 1 and Tank 2.*

4.2 Step 2 – Tank 1 Machine Speed

Connect another analog input (choose again from AI2 to AI6) to analog output (AO2) so you are able to change machine speed variable value via LED Bar (Figure 29).

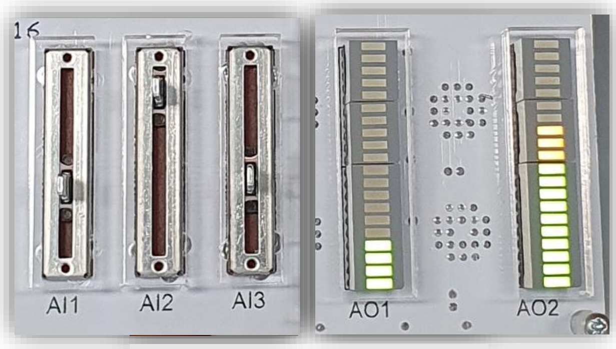


Figure 29. AI2 analog input for machine speed connected to AO2 analog output.

4.3 Step 3 – Tank 2 Oxygen Level

Implement an analog input you haven't used yet (from AI3 to AI6) and display oxygen level of Tank 2 on LED Bar 2 (AO2). There are only two LED Bar displays on the control panel. One is used for water level for both Tank 1 and Tank 2 (AO1) and the other LED Bar will need to be switched from machine speed on AO2 to oxygen level on AO2 and Vis versa.

4.4 Step 4 – Switching Analog Output

Create logic to use AO2 analog output to display:

- Machine speed in Tank 1
- Oxygen level in Tank 2.

Apply the following input variables and logic:

- Switches DI9 and DI11 represent the Machine speed in Tank 1 and Oxygen level in Tank 2, respectively.
- If none switch is activated, AO2 must be turned off or receive a value equal to zero.
- Whenever exclusively a single switch is ON, the AO2 must represent the correspondent input variable.
- In the case that both switches are activated, AO2 must be turned off or receive a value equal to zero.

Note: an XOR logic gate could execute the described logic to enable or disable coding sections (Figure 30).

Note: Applying the JUMP procedure might be an advantage for this type of logic.

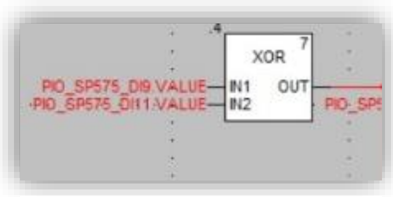


Figure 30. XOR Gate that can be used to change from one switch to another.

4.5 Step 5 – Faults

This step you will create possible faults of the system and a way to solve them. Tank 1 and Tank 2 should have separate *fault* LED outputs. This should indicate to the user which tank a fault has occurred. If a fault occurs in both tanks, for example when water level reaches high or low thresholds, one LED will be on for Tank 1 and another LED will be on for Tank 2. When a fault is rectified for either tank 1 or tank 2, there should be a button for each tank to turn off LEDs when fault has been fixed.

4.5.1 Fault on Water Level

The tank should have high- and low-level constraints. If the water is lower than say 20% or higher than 80%, plant functionality will be compromised, and a fault will occur on both tanks. Two *fault* LEDs will be turned ON to display to user a fault has occurred in both tanks. This also will suggest to the user that the pump connecting Tank 1 and Tank 2 will be turned off until the issue has been rectified.

Note: *Implement a button into code to turn fault LED OFF Tank 1 and another button for Tank 2. Ensure buttons cannot be pressed to turn LED OFF unless Tank 1 and Tank 2 levels (water level, machine speed and oxygen level) are back between the constraints that you set for water level, machine speed and oxygen level.*

4.5.2 Fault on Machine Speed

For Tank 1 a high- and low-level fault limit needs to be implemented for machine speed (similar to water level). As you scroll or rotate (depending on analog input used) LEDBar (AO2) should be connected to analog input.

The *fault* LED output for Tank 1 will be triggered ON to indicate a fault when high or low fault limits you set have been reached. You can use button (same button used for Tank 1 water level fault) when fault has been fixed and you need to turn fault LED OFF for Tank 1.

4.5.3 Fault on Oxygen

As above set high and low limit constraints for oxygen level in Tank 2. Connect an analog input (one you haven't chosen) to analog output (Bar2) and trigger *fault* LED for Tank 2 when a limit has been reached.

You will need to *switch* from Tank 1 to Tank 2 (Vis versa) to display machine speed and oxygen level on AO2.

Note*: If you use two switches to show levels of both Tank 1 (machine speed) and Tank 2 (oxygen level) on AO2, make LED above switch, turn ON. This will display to the user which tank you are looking at. By using an XOR logic gate for both switches you are only allowing one tank to be displayed on AO2.*

4.5.4 Other Faults

Another fault measure could be to code a manual fault switch for tank 1 and another manual fault switch for tank 2. This is like having an emergency shut down switch if unable to shut down via graphical user interface.

Note: Make LED above switch to turn ON/OFF when flicking switch.

5. Set up and Animate a MIMIC

Mimics are great for clients/employees to view company systems, like water level in a tank, graphically. The user-interface presents information in a graphical format that can show both static and dynamic representations of your system. In this project, you will design the mimic to move water level up and down in a primary clarifier tank and an aeration basin along with changing the colour of a pump to indicate when a fault has occurred. If pump stops, fault has been triggered it will turn red. If fault has been cleared pump will turn green. Figure 31, Illustrates a pump that is running and a water tank filling with water. You will also need to monitor water level, machine speed, oxygen level as well as outstation (RTU) communication status (Figure 32).

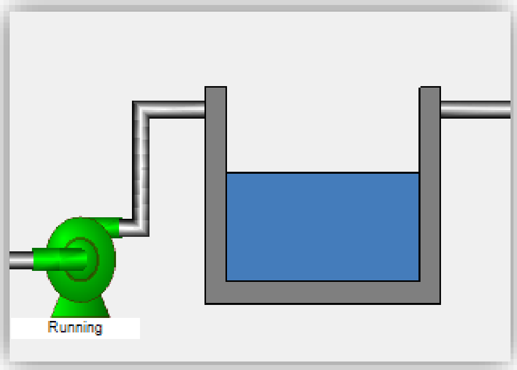
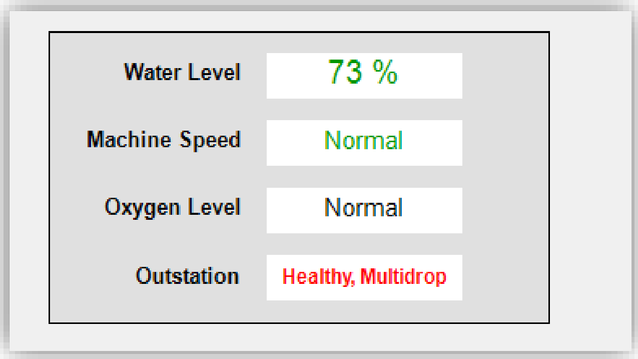


Figure 31. An example of a pump that is running and a water tank.



Water Level 73 %
Machine Speed Normal
Oxygen Level Normal
Outstation Healthy, Multidrop

Figure 32. Value and States of water level, machine speed, oxygen level and outstation (RTU) communication status.

5.1 Create Group Folders and Configure Points (Objects)

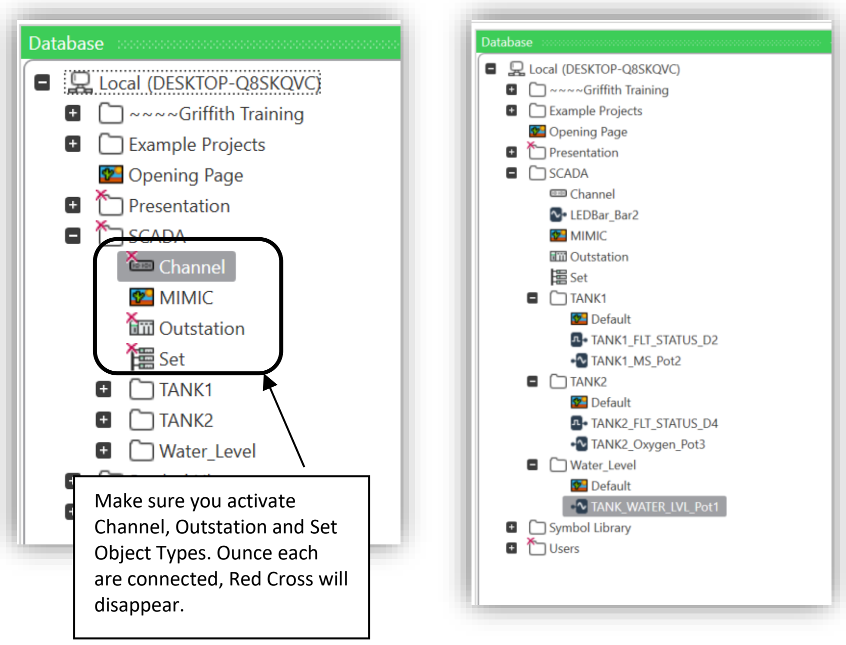
Once you are logged into ViewX, use *Groups* to organise your database so it is user friendly. To create a new group: *Right click* ***Local*** *under the* ***Database*** *bar>Select* ***Create New****>Select* ***Group****>Name the group>'Project_2' (in below Figures it will have 'SCADA')*. When new groups are created, a **Default** mimic will be created for each. You will be using the **Default** mimic under the *Project Group Name* you have given project 2.

Under *Project Group Name* proceed to make sub-groups. Create and name the following objects in each sub-group specified in Table 1 so **Database** will look similar to Figure 33:

Table 1. Parameters for sub-groups in ViewX.

Sub-Group	Object Type	Object Name
Project Group Name	DNP3 >> Direct Channel	Channel
Project Group Name	DNP3 >> Direct Outstation Set	Set
Project Group Name	DNP3 >> Generic >> Direct Outstation	Outstation
Project Group Name	Group	TANK1
Project Group Name	Group	TANK2
Project Group Name	Group	Water Level
TANK1	DNP3 >> Generic >> Binary Output Point	TANK1_FLT_STATUS_D2
TANK1	DNP3 >> Generic >> Analog Input Point	TANK1_MS_Pot2
TANK2	DNP3 >> Generic >> Binary Output Point	TANK2_FLT_STATUS_D4
TANK2	DNP3 >> Generic >> Analog Input Point	TANK2_Oxygen_Pot3
Water Level	DNP3 >> Generic >> Analog Input Point	TANK_WATER_LVL_Pot1

You can ADD, change any name or change the Input and Output Objects. This is a basic example of a mimic design. Remember it helps to match Object Names in Remote Connect with Object Names in ViewX.



Make sure you activate Channel, Outstation and Set Object Types. Ounce each are connected, Red Cross will disappear.

Figure 33. Database section should look similar to the above.

5.1.1 Establish Communications (Generic DNP3)

For all objects/points created in previous section, you will need to set the properties for each to enable RTU communications. The mimic you created in Project 1 has the same *Channel*, *Set* and *Outstation* object configurations.

5.1.1.1 Configure Channel Object

Double click Channel>Channel tab>Tick 'In Service'>Save. Channel is dynamically enabled. Configure the following settings (Table 2):

Table 2. Configure for Channel Object.

Connection Tab Field	Setting
Connection Type	Network
TCP/IP Type	UDP
Merge Connections	Tick
Independent Connections	Tick
Listen Port Enabled	Tick
Listen Port Number	20000
Accept Connection from Any Host	Tick
Scan Parameter Tab Field	**Setting**
Line Speed	10240000

Note: *ClearSCADA drivers use the Line Speed to calculate delays in response to sent messages. The setting entered can be the fastest speed as connection is via USB.*

5.1.1.2 Configure Set and Outstation Objects

Communications to RTU are established by both *Set* and *Outstation* object configuration. *Double click on Set object (Table 3).*

Table 3. Configure Set Object.

Tab	Property	Setting
Outstation Set	In Service	Tick
	Channel	*Group Name*.Channel
DNP3	DNP3 Local Address	30000

Double click on Outstation object (Table 4).

Table 4. Configure Outstation Object.

Tab	Section	Property	Setting
Outstation	N/A	In Service	Tick
		Outstation Set	*Group Name*.Set
DNP3	N/A	Address	RTU DNP3 address
	Integrity Polling	Interval	2s
Network	N/A	Network	Single Network
		Host Address	10.2.3.4
		Port	20000

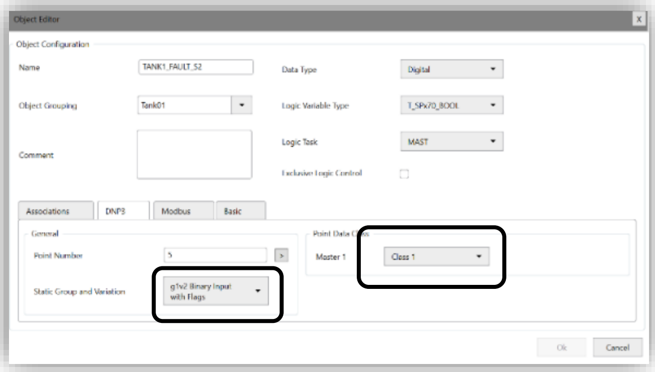
Once *Channel, Set* and *Outstation* have been configured and saved, the Red Cross on the top left of each object should have disappeared. To ensure RTU connection was successful and is working correctly you need to open the *Outstation View Status* dialog. *Right click on Outstation object under* ***Database****>Select View Status>from the menu the State field should display* ***Healthy, Multidrop****.* If communication state is different, view status of the other communication settings, *Set* and *Channel* to determine if these are in a healthy state.

5.1.2 Configure Outstation Points

DNP3 points need to be configured so that ClearSCADA can retrieve data from the RTU points and store them historically. You can also control RTU (outstation) points from corresponding DNP3 points on the mimic.

Note: ***The Group Name in below Figures say 'SCADA' but your Project Group Name should be different. Also, The Point Numbers shown may differ from your values. The Point Number you set in Remote Connect for an Input or Output should reflect the same number in ViewX for the same Input or Output.***

Also ensure Class, under **Object Editor**>DNP3 tab>Point Data Class is either Class 1, Class 2 or Class 3 as it cannot be Class 0 (Static). In addition, make sure Static Group and Variation drop down has Binary Input or Output WITH FLAG (Not without) (Figure 34).


Object Editor

Object Configuration

Name	TANK1_FAULT_S2	Data Type	Digital
Object Grouping	Tank01	Logic Variable Type	T_SPx70_BOOL
Comment		Logic Task	MAST
		Exclusive Logic Control	☐

Associations | DNP3 | Modbus | Basic

General
- Point Number: 5
- Static Group and Variation: g1v2 Binary Input with Flags

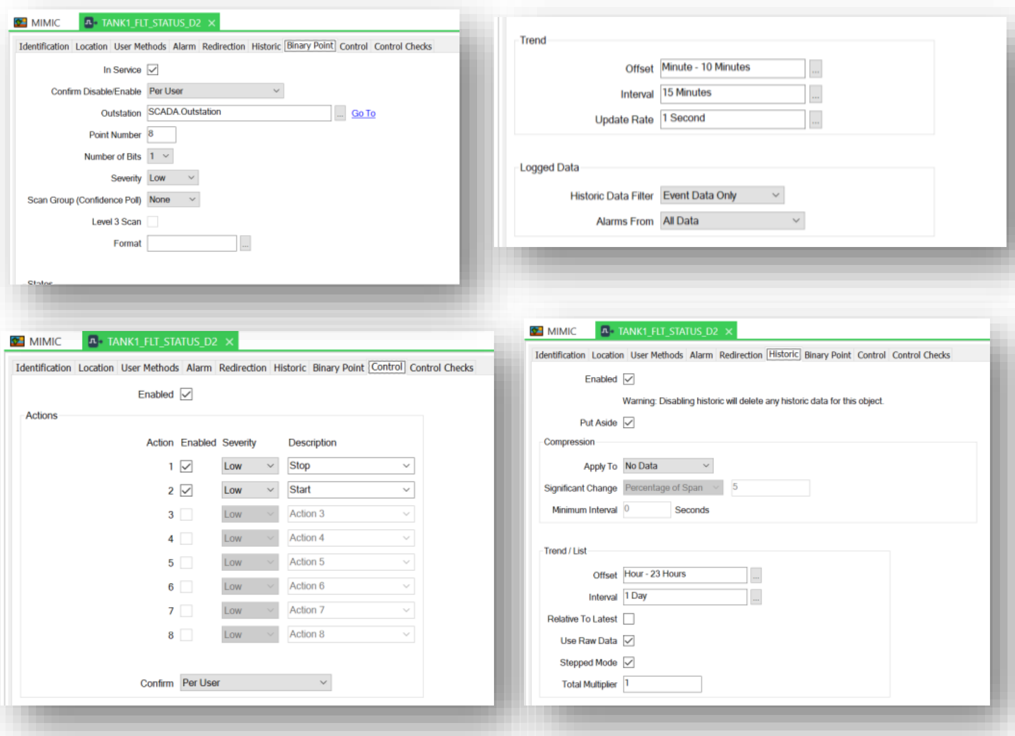
Point Data Class
- Master 1: Class 1

Figure 34. In Object Editor ensure Class is NOT Class 0 (static) and Static Group and Variation is WITH Flags.

5.1.2.1 Configure Binary Output Point (BOP)

A BOP is known as a **Control Point** which allows the user to send START and STOP commands to the RTU. This will turn ON and OFF the binary output on the RTU. You can also configure a BOP to RUN or STOP, pump operation on mimic from the control panel.

Configure the following properties for *Group Name>TANK1>TANK1_FLT_STATUS_D2 as shown in* Figure 35*.*



MIMIC
TANK1_FLT_STATUS_D2
Identification | Location | User Methods | Alarm | Redirection | Historic | Binary Point | Control | Control Checks

Binary Point
- In Service: ✓
- Confirm Disable/Enable: Per User
- Outstation: SCADA.Outstation (Go To)
- Point Number: 8
- Number of Bits: 1
- Severity: Low
- Scan Group (Confidence Poll): None
- Level 3 Scan: ☐
- Format:

Trend
- Offset: Minute - 10 Minutes
- Interval: 15 Minutes
- Update Rate: 1 Second

Logged Data
- Historic Data Filter: Event Data Only
- Alarms From: All Data

Control
- Enabled: ✓

Actions

Action	Enabled	Severity	Description
1	✓	Low	Stop
2	✓	Low	Start
3	☐	Low	Action 3
4	☐	Low	Action 4
5	☐	Low	Action 5
6	☐	Low	Action 6
7	☐	Low	Action 7
8	☐	Low	Action 8

Confirm: Per User

Historic
- Enabled: ✓
- Warning: Disabling historic will delete any historic data for this object.
- Put Aside: ✓

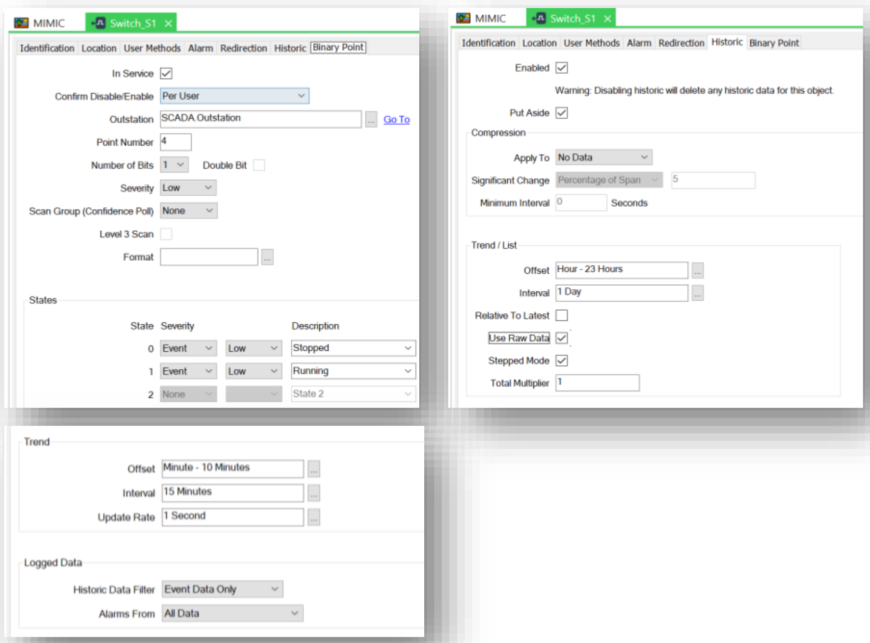
Compression
- Apply To: No Data
- Significant Change: Percentage of Span 5
- Minimum Interval: 0 Seconds

Trend / List
- Offset: Hour - 23 Hours
- Interval: 1 Day
- Relative To Latest: ☐
- Use Raw Data: ✓
- Stepped Mode: ✓
- Total Multiplier: 1

Figure 35. Configure binary output point.

5.1.2.2 Configure Binary Input Point (BIP)

A BIP is known as a **Status Point** which reports if, for example, a pump is running or has stopped. If a Binary Input Point is needed (for this mimic example it was not needed), Figure 36 shows how to configure properties for a Binary Input Point for *Group Name>TANK1>Switch_S1*



In Service ✓
Confirm Disable/Enable: Per User
Outstation: SCADA.Outstation ... Go To
Point Number: 4
Number of Bits: 1 Double Bit
Severity: Low
Scan Group (Confidence Poll): None
Level 3 Scan
Format:

States

State	Severity		Description
0	Event	Low	Stopped
1	Event	Low	Running
2	None		State 2

Trend
Offset: Minute - 10 Minutes
Interval: 15 Minutes
Update Rate: 1 Second

Logged Data
Historic Data Filter: Event Data Only
Alarms From: All Data

Enabled ✓
Warning: Disabling historic will delete any historic data for this object.
Put Aside ✓

Compression
Apply To: No Data
Significant Change: Percentage of Span 5
Minimum Interval: 0 Seconds

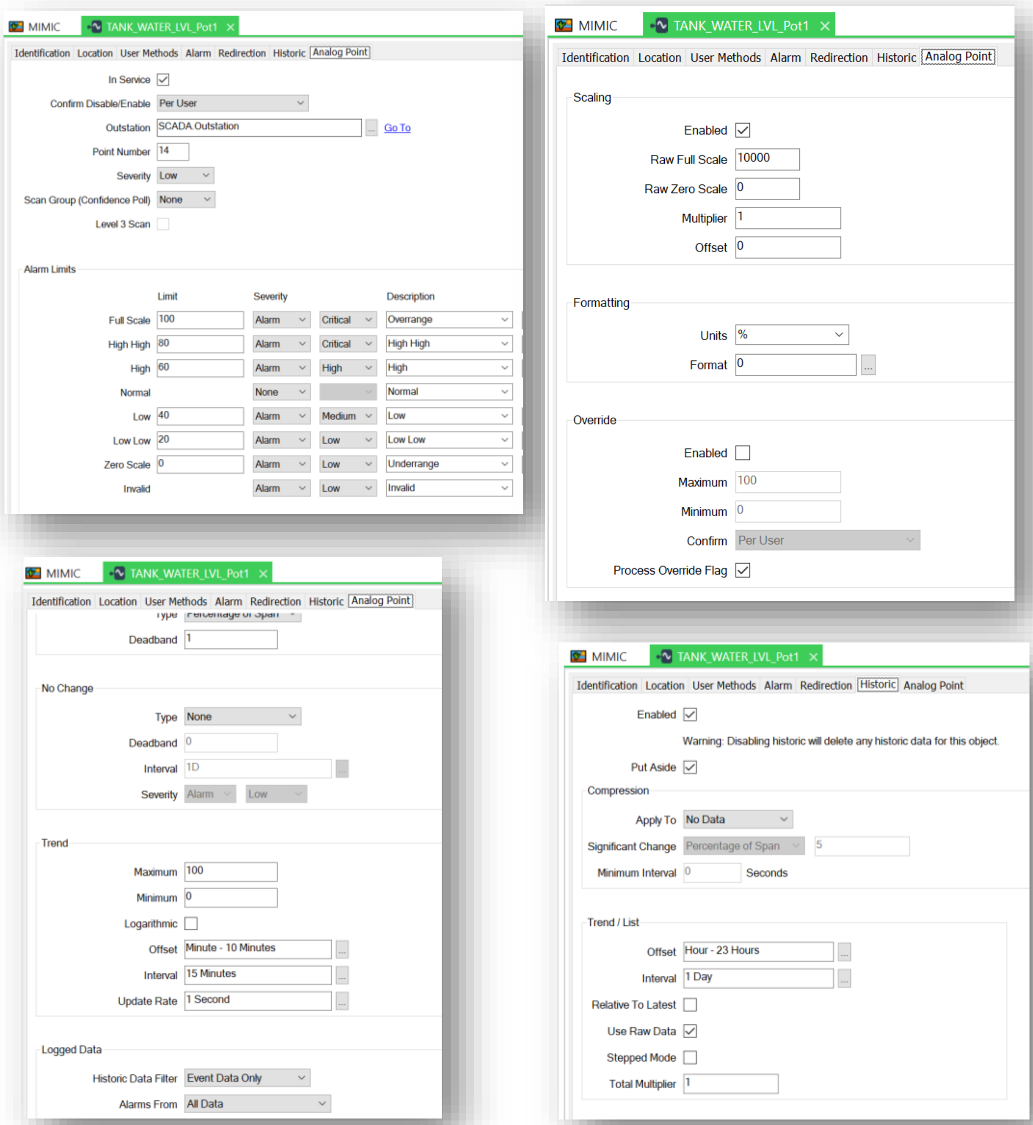
Trend / List
Offset: Hour - 23 Hours
Interval: 1 Day
Relative To Latest
Use Raw Data ✓
Stepped Mode ✓
Total Multiplier: 1

Figure 36. Configure properties for a binary input point.

5.1.2.3 Configure Analog Input Point (AIP)

An AIP is known as a **Level Point** It can, for example, report the level of the water in a tank numerically or graphically.

Configure properties for *Group Name>Water level>TANK_WATER_LVL_Pot1* are displayed in Figure 37.



In Service ✓
Confirm Disable/Enable: Per User
Outstation: SCADA.Outstation
Point Number: 14
Severity: Low
Scan Group (Confidence Poll): None
Level 3 Scan ☐

Alarm Limits

	Limit	Severity		Description
Full Scale	100	Alarm	Critical	Overrange
High High	80	Alarm	Critical	High High
High	60	Alarm	High	High
Normal		None		Normal
Low	40	Alarm	Medium	Low
Low Low	20	Alarm	Low	Low Low
Zero Scale	0	Alarm	Low	Underrange
Invalid		Alarm	Low	Invalid

Scaling
Enabled ✓
Raw Full Scale: 10000
Raw Zero Scale: 0
Multiplier: 1
Offset: 0

Formatting
Units: %
Format: 0

Override
Enabled ☐
Maximum: 100
Minimum: 0
Confirm: Per User
Process Override Flag ✓

Type: Percentage of Span
Deadband: 1

No Change
Type: None
Deadband: 0
Interval: 1D
Severity: Alarm Low

Trend
Maximum: 100
Minimum: 0
Logarithmic ☐
Offset: Minute - 10 Minutes
Interval: 15 Minutes
Update Rate: 1 Second

Logged Data
Historic Data Filter: Event Data Only
Alarms From: All Data

Enabled ✓
Warning: Disabling historic will delete any historic data for this object.
Put Aside ✓

Compression
Apply To: No Data
Significant Change: Percentage of Span 5
Minimum Interval: 0 Seconds

Trend / List
Offset: Hour - 23 Hours
Interval: 1 Day
Relative To Latest ☐
Use Raw Data ✓
Stepped Mode ☐
Total Multiplier: 1

Figure 37. Configure properties for analog input point.

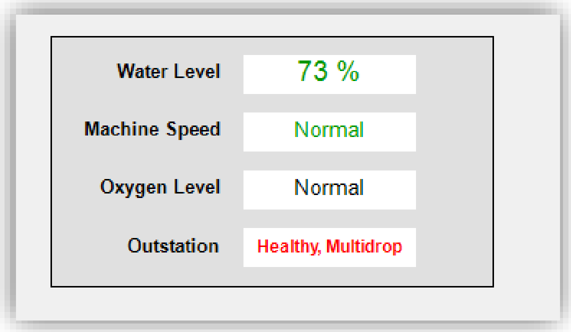
5.2 How to Display State of Tank 1 and Tank 2

Ensure mimic is in **Design Mode**. Drag the *Name* of ***TANK_WATER_LVL_Pot1*** from the **Database** bar to the mimic. From the context menu> Select *Value>Formatted Value*. **Place a text box (icon in *Graphics* tab) next to the Value and write *Water Level*.**

Test that the value changes when controlling the analog input on Control Panel. If values give you negative numbers and/or higher than 100%, you are able to change *Alarm Limits* values and/or *Raw Full Scale* under *Analog Point* tab when you double click on ***TANK_WATER_LVL_Pot1***. Adjust values until satisfied when scrolling.

Drag ***TANK1_MS_Pot2*** onto the mimic from **Database**. From the context menu>select *State*. Do the same for **TANK2_Oxygen_Pot3**. In addition, place Outstation State on mimic to ensure system is always *Healthy*.

A basic example is shown in Figure 38 (you are able to move boxes anywhere on your mimic).



Water Level	73 %
Machine Speed	Normal
Oxygen Level	Normal
Outstation	Healthy, Multidrop

Figure 38. Basic Example of state of system and tanks.

5.3 Add Pump to MIMIC

A range of symbols are included in **Symbol Library** on the **Database** Explorer Bar. One of these pre-built symbols is a pump which can be animated within ClearSCADA.

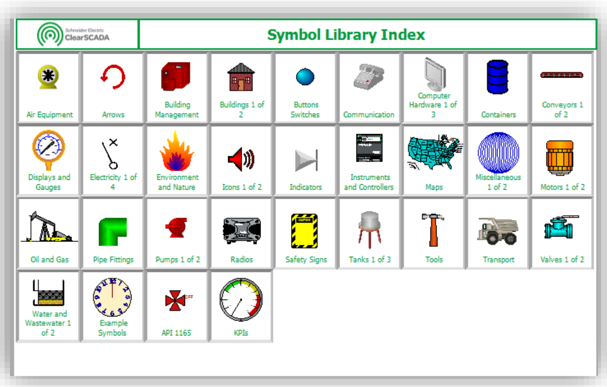


Figure 39. Symbol Library Index to find pump.

- Open **Symbol Library** mimic>**Symbol Library Index>**and switch to **Run Mode** .
- On **Symbol Library Index** mimic (Figure 39)>click *Pumps* group.
- Click the *pump symbol* named **Pump A1**.

Note: *The **Pump A1** mimic is highlighted in the **Database.*** **COPY THE HIGHLIGHTED PUMP A1 IN DATABASE AND PASTE INTO YOUR PROJECT 2 FOLDER AND USE YOUR COPIED VERSION.** *This is to ensure you don't alter the original version.*

- Once you copy and paste Pump A1 into your Project folder open the **Pump A1** mimic in Design mode

Note: *Click icon to place in **Run Mode** or **Design Mode**. When you see the grid dots, this is known as the **Design Mode**.*

- Right click *Pump Symbol* in the **Pump A1** mimic (Figure 40) and **Copy**. **Paste** into your mimic.
- Position *Pump Symbol* as required. You can find a *Pipe* drawing tool in *Graphics* tab to connect pump to tank.

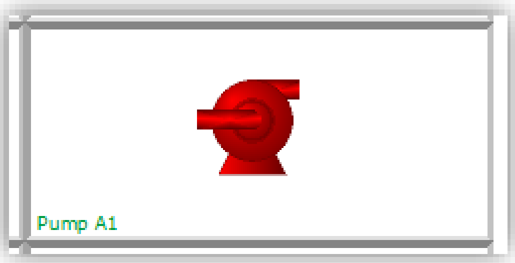


Figure 40. Pump A1 symbol.

Note: *To disable the snap to grid function, hold down the Alt key while you drag object.*

5.3.1 Animate Pump

One way to indicate what state the pump is in on your mimic would be to change the colour of the pump. An example would be to change pump colour to *Green* when running and *Red* if a fault occurs. To do this, the *flow chart editor* will be used.

The flow chart editor enables you to build complex animations in a graphical way. However, one simple way is a TRUE/FALSE Logic, dependant on the value of a single input. For example, if LED (D1) for *tank 1* is ON pump will be *Red* and *Stopped* when a fault has occurred, when LED (D1) is OFF pump is *Green* and *Running*, fault has been fixed.

- Right click on *pump* symbol>select *Animations*>Under *PolyFill* select *Fillcolor*> Select *Flowchart* button on left hand side. Figure 41 shows dialog box that opens.

Note: *The pump is made up of a multiple object, so the PolyFill group is used. You are able to control individual objects properties depending on what you would like the object to do.*

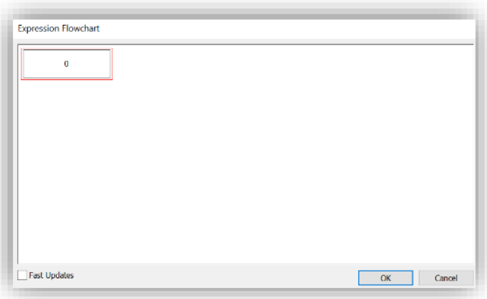


Figure 41. Flowchart dialog box to animate colour of pump.

- Right click inside the box outlined in Red>Select **Insert** from the menu.
- Double click the box with text **TRUE** > Browse to find the **Status** point and select the point > Press ok twice to close the windows.
- Double click the box at the right of the decision box to choose the colour green when TRUE (Figure 42).

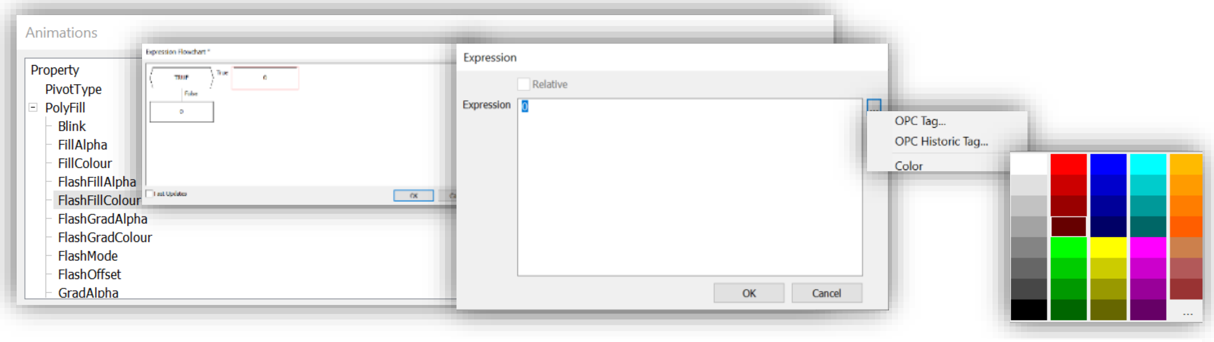


Figure 42. How to select colour when statement is TRUE.

Note: *To choose more colours, click on the three dots on the bottom right of the colour chart.*

- Specify red for the FALSE state. Close Flowchart Editor.

This expression will appear in the Animations dialog next to *Fillcolor*:

iif(".TANK1.TANK1_FLT_STATUS_D2.CurrentState", RGB(0,255,0), RGB(255,0,0))

You can also enter statements directly if you know the format of the statement and the results that you require.

Place the mimic into **Run Mode** and check if pump changes colour when **Status** point changes.

Note: The pump may need to have the **Gradient** colour adjusted as pump may not be solid in colour from previous steps.

To adjust the **Gradient** colour, you will need to repeat previous steps for *Fillcolor* but this time for *GradColor*. Ensure the colours are slightly darker than the existing animated colours.

Verify the entire pump animates as you would expect.

5.4 Water Tank and Water Level

In this section, you will use drawing tools to draw tank and basin on the **Group Name.Default** mimic When you double click on **Default** the *Graphics* tab (top left of ViewX) will be highlighted providing you with a number of drawing tools (Figure 44). In this example, a **Polygon** drawing tool was used for tanks (Figure 43).

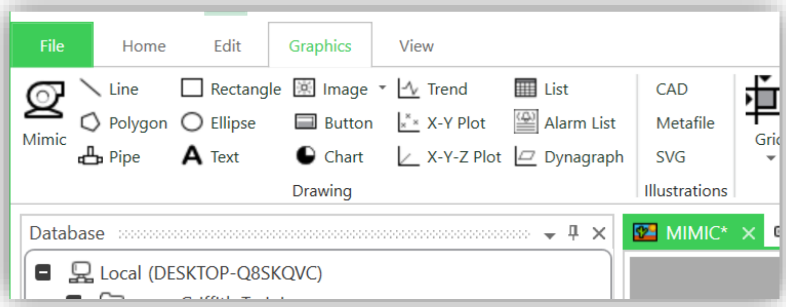


Figure 44. Drawing tools under Graphics.

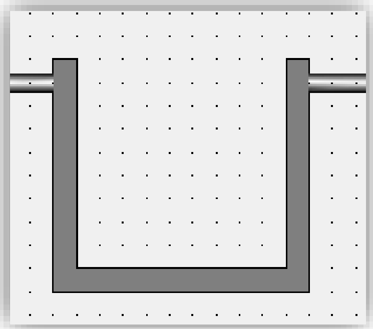


Figure 43. Tank example using Polygon.

5.4.1 Implement Size Animation for Water Level

Draw a square in your tank to represent liquid in the tank. Apply a *Fill* colour that represents water. *Double click on square/under **Polyline Properties** window/ go to Fill tab and change colour* (Figure 45). This square will be animated to change size from bottom to top. Animating the size of an object or shape is known as **Size Animation**. Select *square/Right click/Animations* (Figure 46).

SizeMax: Represents maximum size of square animation (The size of square drawn on mimic).
SizeMin: Associates the *minimum point value* of Input with minimum size of square.
SizeVal: Sets square size to current value of Analog Input (Between minimum and maximum value)

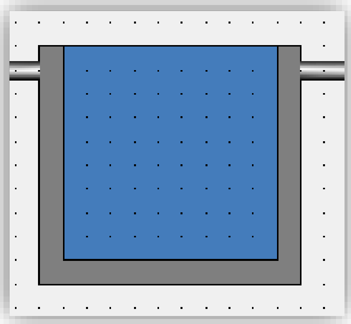
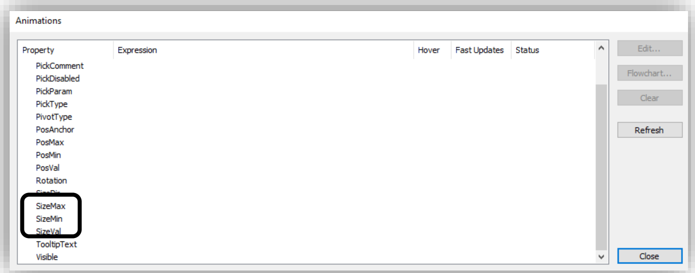


Figure 45. Fill colour in tank example to represent water.


Animations

Property	Expression	Hover	Fast Updates	Status
PickComment				
PickDisabled				
PickParam				
PickType				
PivotType				
PosAnchor				
PosMax				
PosMin				
PosVal				
Rotation				
SizeMax				
SizeMin				
SizeVal				
TooltipText				
Visible				

Figure 46. Animate the water in tank to move with analog input from control panel.

- Select ***SizeMax****>Edit* an Expression dialog opens.
- Select the browse button (…), from menu select **OPC Tag.**
- In **Select Tag** dialog go to tab *GroupName>WaterLevel folder>TANK_WATER_LVL_Pot1>$Config* (Figure 47).
- Select **FullScale** property tag, click *ok* to close dialog and return to animation window.
- Repeat for **SizeMin** –**ZeroScale** property tag in *$Config.*
- Repeat for **SizeVal–CurrentValue**. However, tag is not in $Config but still under point/object *TANK_WATER_LVL_Pot1.* Make sure you tick **Fast Update** check box in Expression dialog. This updates the client at a faster update rate.

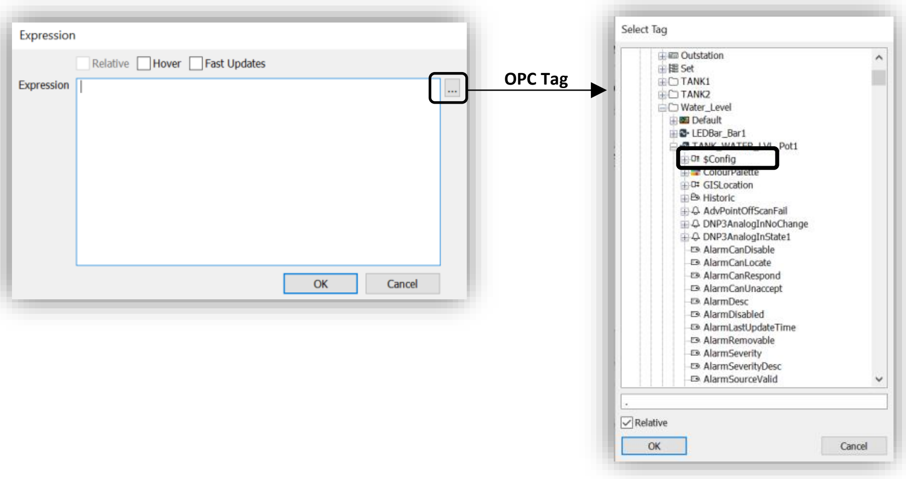
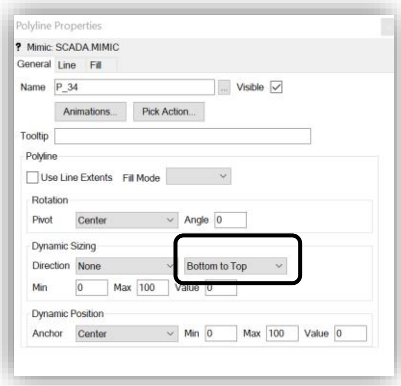


Figure 47. Find animation property to animate TANK_WATER_LVL_Pot1

- *Double click on water>General tab>Dynamic Sizing>Direction drop down menu>Select Bottom to Top* as shown in the following Figure 48.



Polyline Properties
Mimic: SCADA.MIMIC
General | Line | Fill

Name: P_34 — Visible: ✓
Animations... | Pick Action...
Tooltip:

Polyline
Use Line Extents: ☐ — Fill Mode:

Rotation
Pivot: Center — Angle: 0

Dynamic Sizing
Direction: None → Bottom to Top
Min: 0 — Max: 100 — Value: 0

Dynamic Position
Anchor: Center — Min: 0 — Max: 100 — Value: 0

Figure 48. Select Bottom to Top so water fills from Bottom to Top.

To check water level animation is working, press on icon to place mimic in Run Mode and verify if water level in mimic is adjusting with the Analog input from control panel.

5.5 Additional Options

- Animate a picture of a propeller and make it rotate.
- Change colour of the water when water reaches < 20% (Figure 49).

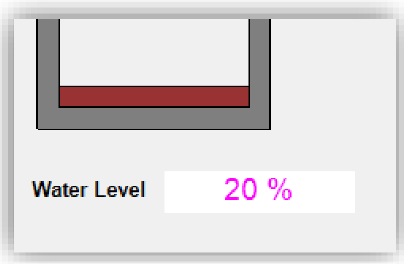


Figure 49. Change colour of water when it reaches 20% empty.

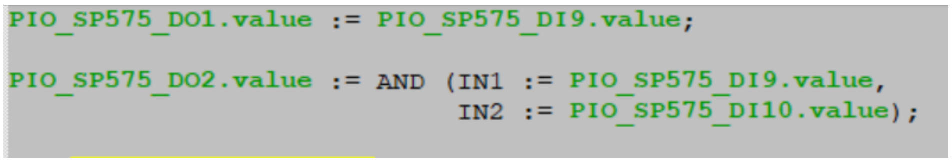
6. Structured Text

Structured Text (ST) is a programming language for PLCs/RTUs. For programmers and engineers who are used to high level programming languages, such as C, C++, Java and Python, ST (Structured Text) syntax may look familiar and the program might be easier to follow. However, it is a text description of Ladder diagram. The ST used in the Logic Editor is a simpler version in comparison to other examples found on the internet. The objective of this section is to translate the previous code made in Function Block Diagram into Structured Text.

6.1. Basic Logic

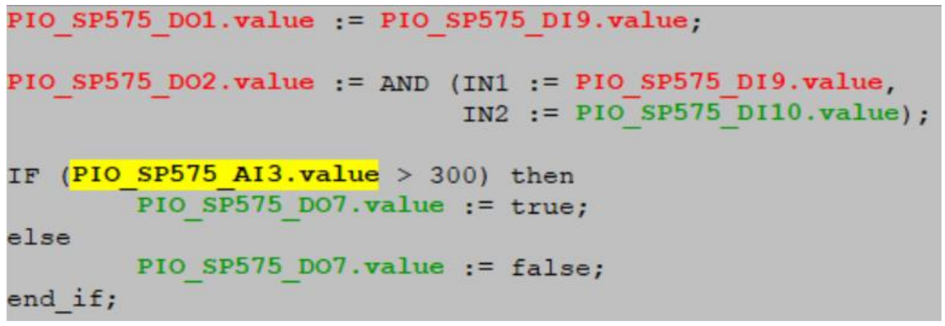
Structured Text code must have a “;” at the end of each line of code, like other high-programming languages. For association between two variables/data/object, the symbol “:=” is used, as illustrated on Figure 50 and Figure 51, where the variable on left is the output and variables on right are inputs. When using basic logic gates, such as AND, OR, XOR and NOT, again, the variable on the left is the output. However, for the inputs, brackets need to be placed after the logic gate name, associate each input with matching variable, using a “,” to separate each input, and be cautious of the maximum input value of 32.

Try the code from Figure 50 for testing. When connected to the SCADApack, both digital input and output variables will be coloured in green when state is HIGH (true) or red when state is LOW (false), Figure 51. As for analogue variables, the name will be highlighted in yellow.



```
PIO_SP575_DO1.value := PIO_SP575_DI9.value;

PIO_SP575_DO2.value := AND (IN1 := PIO_SP575_DI9.value,
                            IN2 := PIO_SP575_DI10.value);
```

Figure 50. Structured text basic code example when all inputs are true.

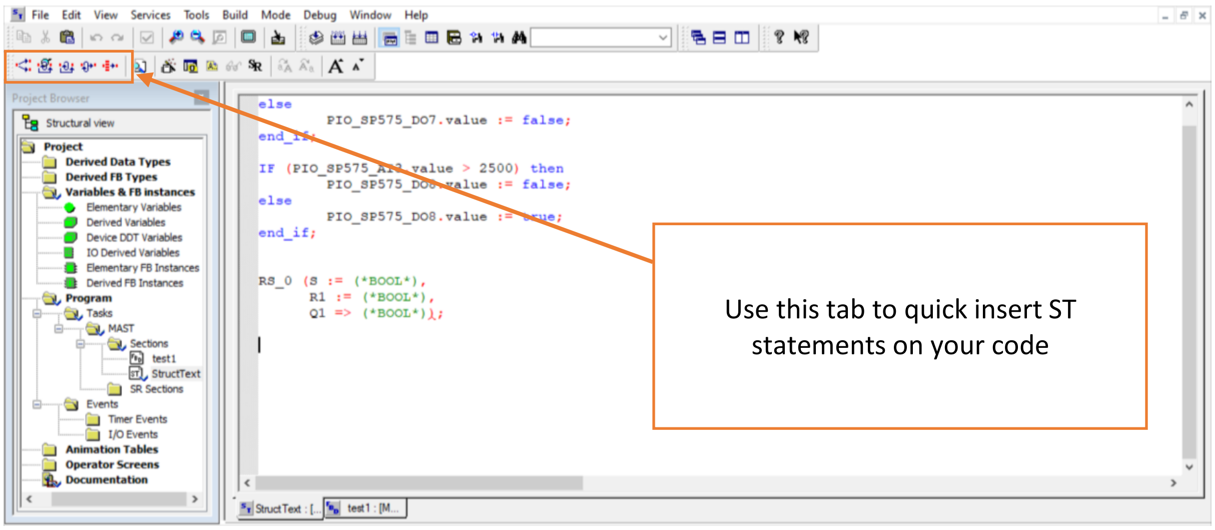


```
PIO_SP575_DO1.value := PIO_SP575_DI9.value;

PIO_SP575_DO2.value := AND (IN1 := PIO_SP575_DI9.value,
                            IN2 := PIO_SP575_DI10.value);

IF (PIO_SP575_AI3.value > 300) then
        PIO_SP575_DO7.value := true;
else
        PIO_SP575_DO7.value := false;
end_if;
```

Figure 51. Structured text basic code example when just one input is true. The analogue input is highlighted in yellow.

6.2. Statements

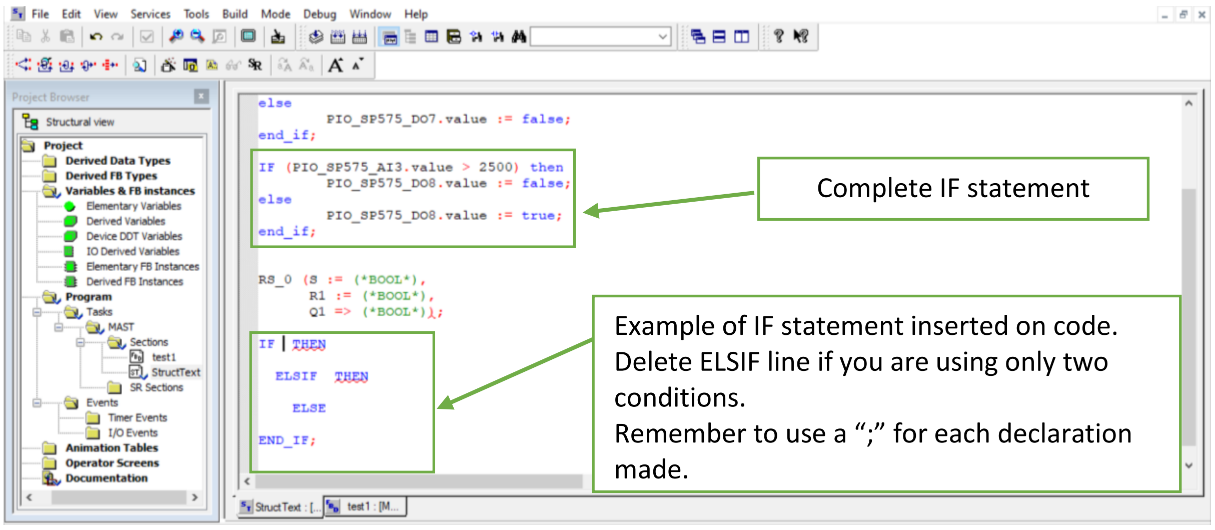
ST code has commonly used statements like IF, FOR, WHILE, REPEAT and CASE. The IF statement may be the only statement used to rewrite the project from Function Block Diagram into Structured Text. Because of that, the following example shows how to easily access statements on the Editor bar. Figure 52 shows the bar that has all statements for ST, while Figure 53 illustrates the statement placed on code. Observe that placing the selected statement does not mean that it will be fully filled. Variables and declarations should be entered by the user.



```
else
        PIO_SP575_DO7.value := false;
end_if;

IF (PIO_SP575_AI3.value > 2500) then
        PIO_SP575_DO8.value := false;
else
        PIO_SP575_DO8.value := true;
end_if;

RS_0 (S := (*BOOL*),
      R1 := (*BOOL*),
      Q1 => (*BOOL*));
```

Use this tab to quick insert ST statements on your code

Figure 52. Editor bar where statements are located.



```
else
        PIO_SP575_DO7.value := false;
end_if;

IF (PIO_SP575_AI3.value > 2500) then
        PIO_SP575_DO8.value := false;
else
        PIO_SP575_DO8.value := true;
end_if;

RS_0 (S := (*BOOL*),
      R1 := (*BOOL*),
      Q1 => (*BOOL*));

IF | THEN

  ELSIF  THEN

    ELSE

END_IF;
```

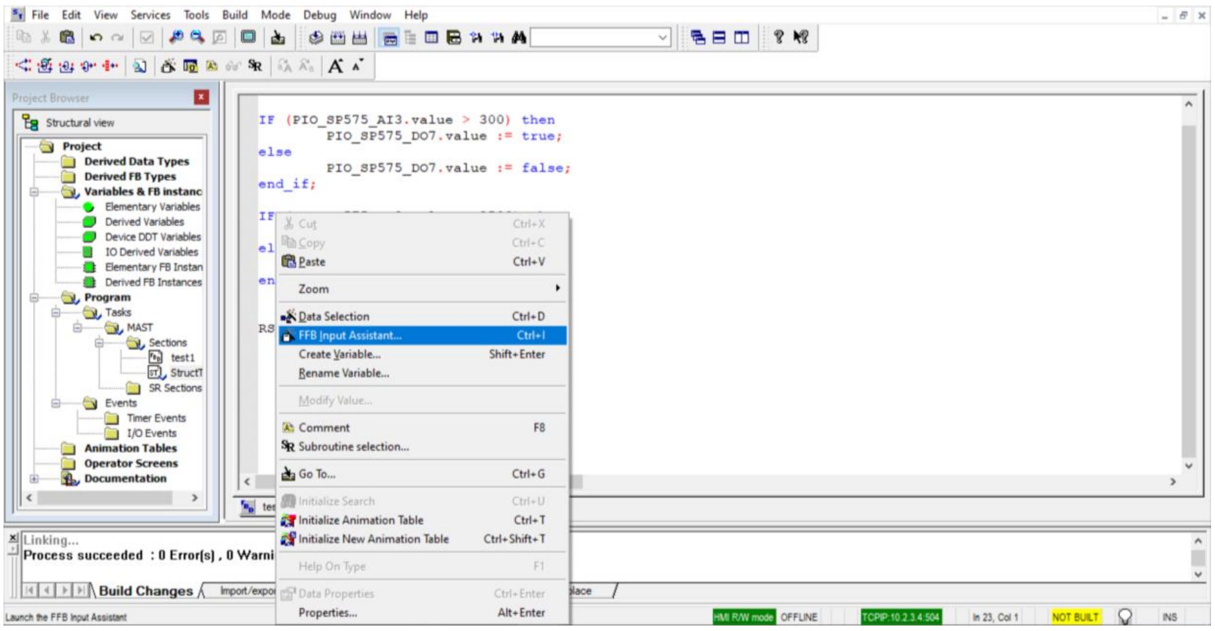
Complete IF statement

Example of IF statement inserted on code. Delete ELSIF line if you are using only two conditions.
Remember to use a ";" for each declaration made.

Figure 53. IF statement added to code from Editor bar.

6.3. Function Blocks

Another feature that Structured Text has, is the common library of functions, similar to Function Block Diagram and Ladder Diagram. Like FBD, there is an FFB Input assistant when the mouse right button is pressed on the code window, or on the Edit tab. The short cut on keyboard is "CTRL+I". Figures 54 to 57 demonstrate the process of selecting a function on FFB Input Assistant until placing function on code. Figure 57 shows that the function has a number after its name, as a single function can be used multiple times. Therefore, this number is a unique identification. Also, do not forget to replace inputs and outputs with the desirable objects/variables.



```
IF (PIO_SP575_AI3.value > 300) then
        PIO_SP575_DO7.value := true;
else
        PIO_SP575_DO7.value := false;
end_if;
```

```
Linking...
Process succeeded : 0 Error(s) , 0 Warni
```

Figure 54. After right clicking inside code window, select FFB Input Assistant.

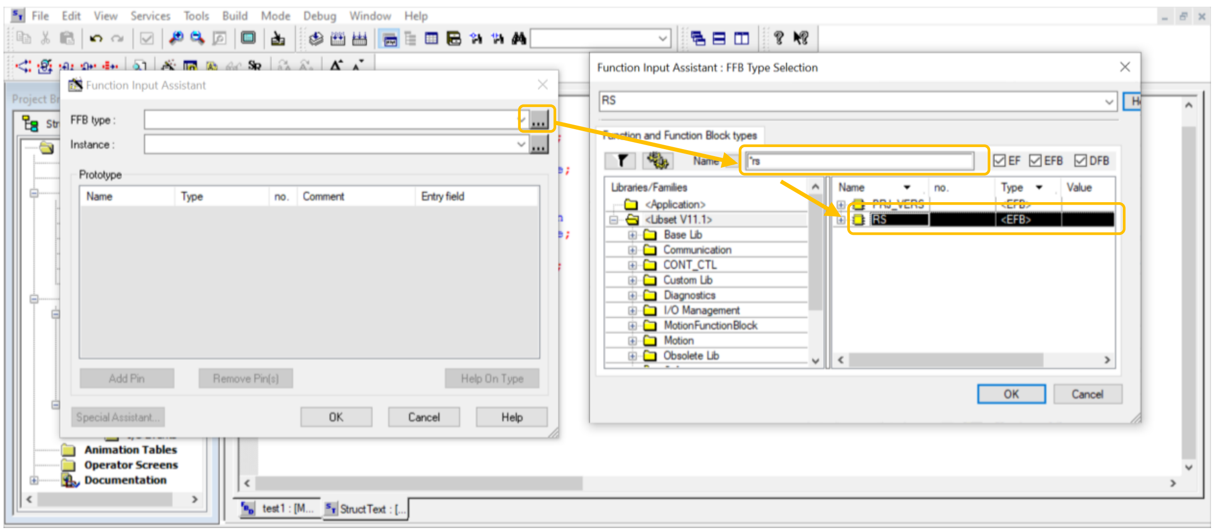
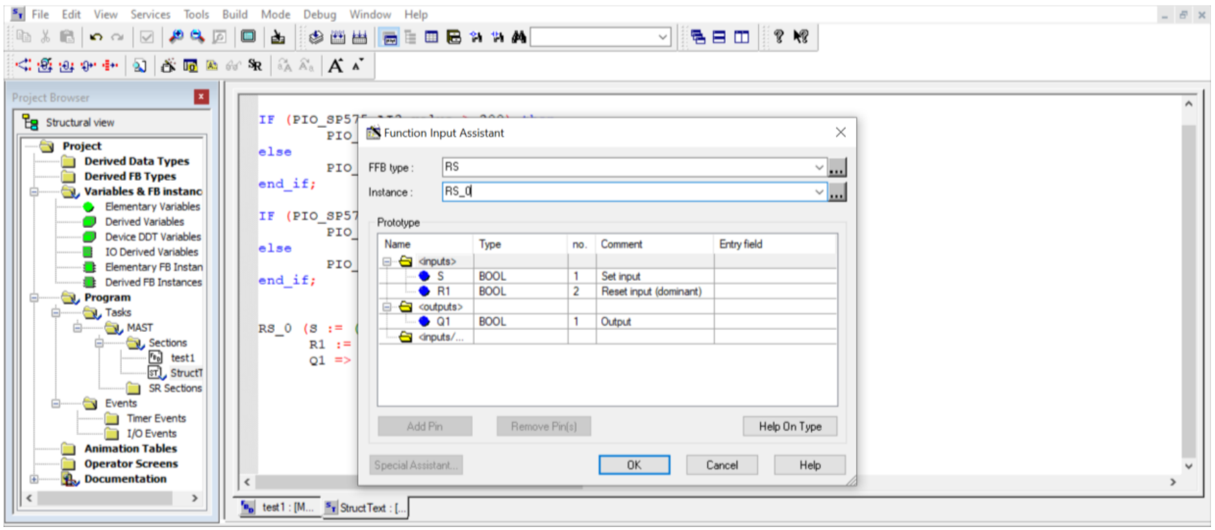


Figure 55. On FFB Input window select "..." from FFB type. The function library will open. Then, type on "name" text box the function or manually select it.



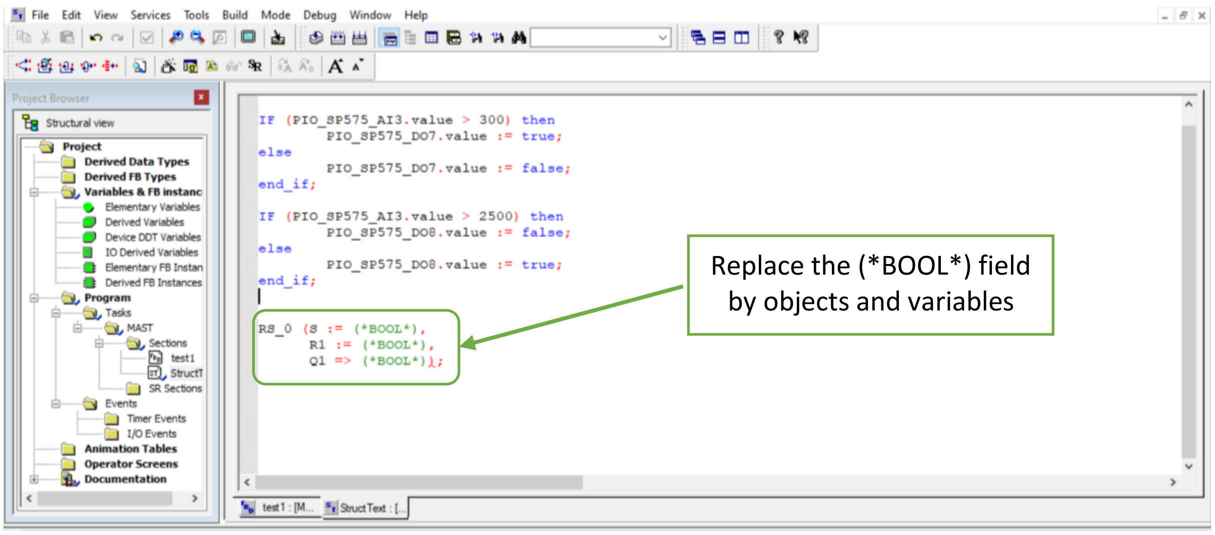
FFB type : RS

Instance : RS_0

Name	Type	no.	Comment	Entry field
<inputs>				
S	BOOL	1	Set input	
R1	BOOL	2	Reset input (dominant)	
<outputs>				
Q1	BOOL	1	Output	
<inputs/...				

```
RS_0 (S :=
     R1 :=
     Q1 =>
```

Figure 56. Selected function displayed on FFB Input Window.



```
IF (PIO_SP575_AI3.value > 300) then
        PIO_SP575_DO7.value := true;
else
        PIO_SP575_DO7.value := false;
end_if;

IF (PIO_SP575_AI3.value > 2500) then
        PIO_SP575_DO8.value := false;
else
        PIO_SP575_DO8.value := true;
end_if;

RS_0 (S  := (*BOOL*),
      R1 := (*BOOL*),
      Q1 => (*BOOL*));
```

Replace the (*BOOL*) field by objects and variables

Figure 57. Selected function is inserted on code.

6.4. Rewrite Project using Structured Text

Translate the Block Diagram into Structured Text code, by not using two codes in the same project, which could cause a conflict. Thus, save the FBD project in a different folder. Create a new project or copy previous project on a folder for ST only. On the Editor, insert a new ST section. Make sure that only this section exists in your project. With the basic logic descripted previously, rewrite this project.