

# Function Block Diagram Lab

## Objectives

---

### IN THIS LAB YOU WILL:

- **Receive a basic overview of Function Block Diagram programming**
- **Write logic using the Function Block Diagram language**
- **Download the logic to a PACSystems controller**
- **Interact with the logic using the QuickPanel display**



## Overview

### IEC-61131-3

\*IEC 61131-3 is an international standard first published in December 1993 by the IEC. This standard defines two graphical and two textual controller programming language standards:

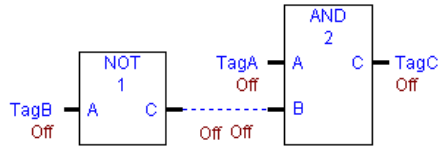

Textual	Graphical
<ul style="list-style-type: none"> <li>- Instruction List (IL)</li> <li>- Structured Text (ST)</li> </ul>	<ul style="list-style-type: none"> <li>- Function Block Diagram (FBD)</li> <li>- Ladder Diagram (LD)</li> </ul>
<pre>LD TagA ANDN TagB ST TagC</pre>	
<pre>TagC := TagA and not TagB;</pre>	

Figure 3-1. IEC Programming Languages Summary

The standard also defines both graphical and textual Sequential Function Chart elements to organize programs for sequential and parallel control processing. An example of an SFC is shown below:

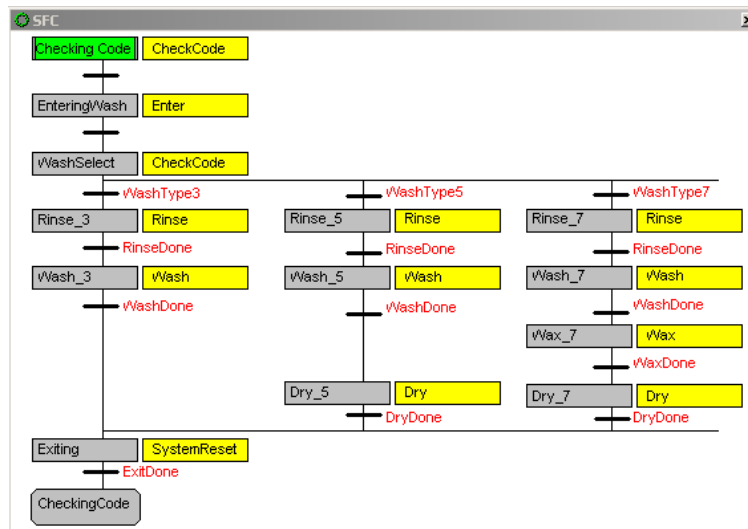


Figure 3-2. Sequential Function Chart (SFC) Example

\*Retrieved from "[http://en.wikipedia.org/wiki/IEC\\_61131-3](http://en.wikipedia.org/wiki/IEC_61131-3)"

## Function Block Diagram (FBD)

Function Block Diagram (FBD) is a user-created sheet or page of application logic consisting of interconnected graphical blocks that represent instructions and logic blocks. The FBD editor is a graphical, freeform editor for editing FBD logic.

### About Function Block Diagram Logic

Function Block Diagram (FBD) logic is one of four programming languages specified by the IEC 61131-3 standard. FBD is a process control language; most FBD instructions are not conditionally executed.

An FBD instruction is shown as a rectangle, with inputs entering from the left and outputs exiting on the right, as shown by the ADD instruction below.

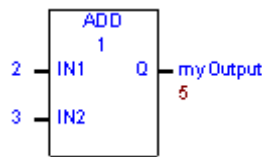


Figure 3-3. Example of FBD Instruction

In the example above, IN1 and IN2 are the input operands; Q is the output operand. The constants '2' and '3' are the input values for IN1 and IN2 respectively; myOutput is the output variable for Q with a current value of '5'.

The numeral 1 under the instruction name ADD inside of the FBD instruction rectangle denotes the execution order of the FBD logic on the page.

In its simplest form an FBD does the following:

- It accepts input from a variable, constant, or wire, which may be the output from another FBD instruction.
- It performs an operation by using an FBD instruction.
- It stores the result of the operation in an output variable or wire. You can use the result as input to another instruction or program.

#### **Input connection point must be only one of**

- Variable
- Wire
- Constant

#### **Output connection point must be only one of**

- Variable
- Wire

From an FBD block, you can call LD, ST, C, and other FBD blocks. These called blocks may be Program Blocks (PBs), Parameterized Subroutine Blocks (PSBs), or User Defined Function Blocks (UDFBs).

## Solve Order of FBD Instructions

The solve order of instructions in an FBD is determined by the following:

- The display position of the instruction on the FBD page (starting at upper left)
- Which instructions must be processed first so that the inputs to other FBD instruction can be resolved

The solve (or execution) order of each FBD instruction is indicated with a number under the Instruction name in the instruction rectangle. The solve order is determined as follows:

1. The FBD editor scans for instructions beginning at the top left corner of the diagram, proceeding left to right and then top to bottom until it finds the top left most pixel of an instruction. The first instruction that is encountered in this manner is executed first, as long as all of its inputs have been resolved.
2. If an instruction's inputs are not resolved, then logic will be checked until an Instruction's inputs are all resolved. As inputs are resolved, the logic will be executed.

### Example 1

When instructions are added to the FBD editor, they may appear as follows:

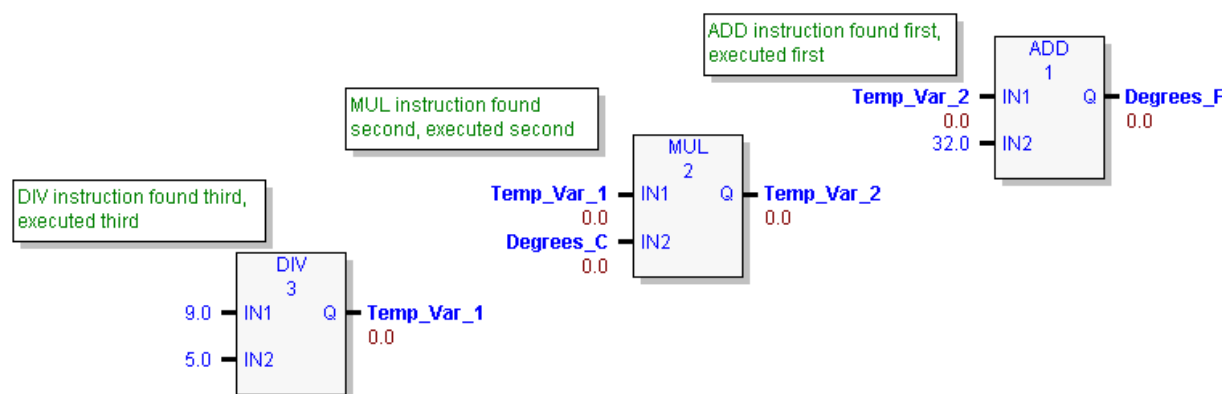


Figure 3-4. FBD Solve Order Example 1

In this case, as the instructions are encountered left-to-right, top-to-bottom, they are found the following order: ADD, MUL, and DIV. Since all inputs are resolved for each instruction, the execution order is defined.

## Example 2

If this logic is modified to “wire” the blocks together, then the following logic would result:

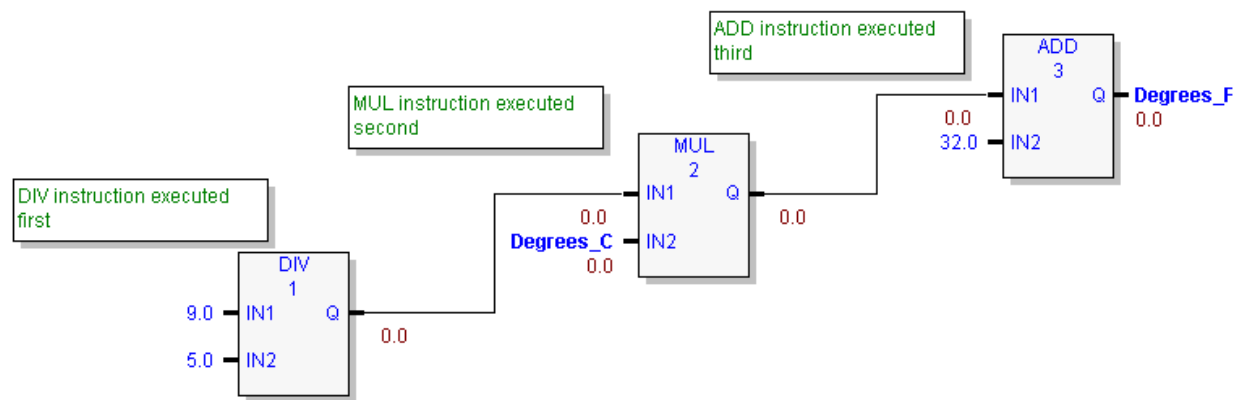


Figure 3-5. FBD Solve Order Example 2

The ADD instruction with a solve order of 3 (above) has the highest solve order because its input is connected by a wire to the output of the MUL instruction with a solve order of 2. The ADD instruction cannot have its input at connection point IN1 resolved until the MUL instruction’s solve order is resolved.

The MUL instruction (solve order of 2) cannot be resolved until the DIV instruction solve order is resolved, because a wire connects an input of the MUL instruction to the output of the DIV instruction.

The DIV instruction is first in the solve order because of the following:

- There are no input wires connected to it.
- There are no instructions in the FBD editor that are positioned higher whose inputs are not dependent on resolution of this instruction.

## Comments

Use the “Text” instruction to document your FBD with detailed comments to help yourself and others to understand the FBD. An example of a Text instruction is as follows:

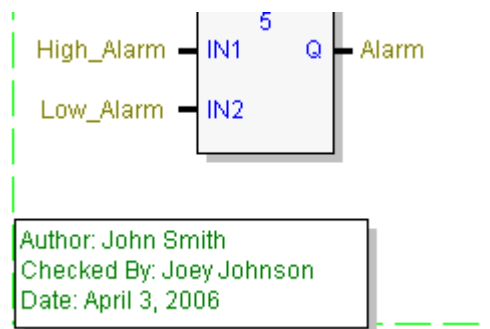
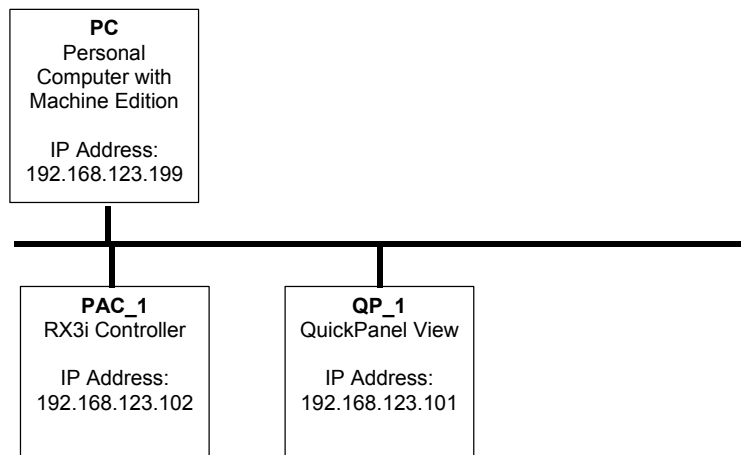


Figure 3-6. FBD Text Instruction Example

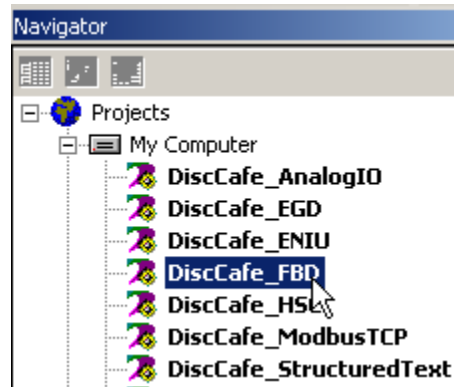
## Lab Exercise – Working with Function Block Diagrams

In this exercise you will examine some FBD logic.

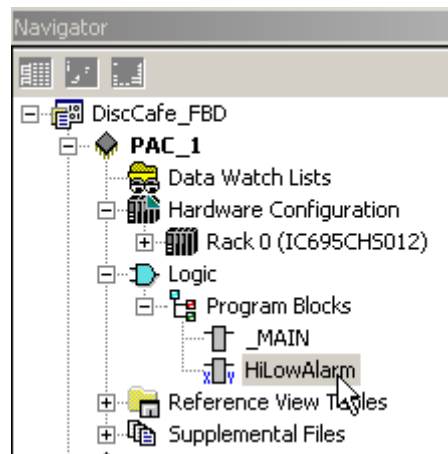
This lab assumes the following basic hardware configuration and setup. The IP Addresses defined below are used in the Project provided with this lab.



1. Begin by **double-clicking** on the Project named "DiscCafe\_FBD" in the Navigator window.



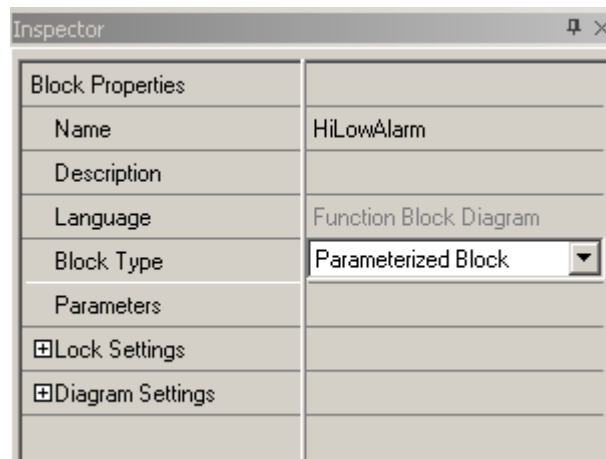
- Next click on the block labeled **HiLowAlarm** in the logic section of the **PAC\_1** target in the **Navigator** window.



- Once you have selected this block look at the **Inspector** window to view its' properties.

#### NOTE

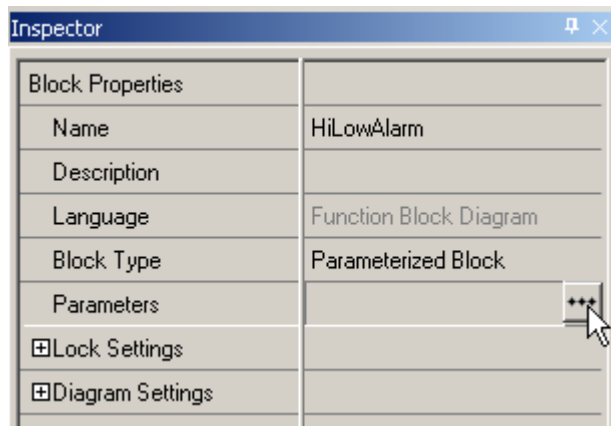
**If the Inspector window isn't open, right-click on the block and select Properties from the menu.**



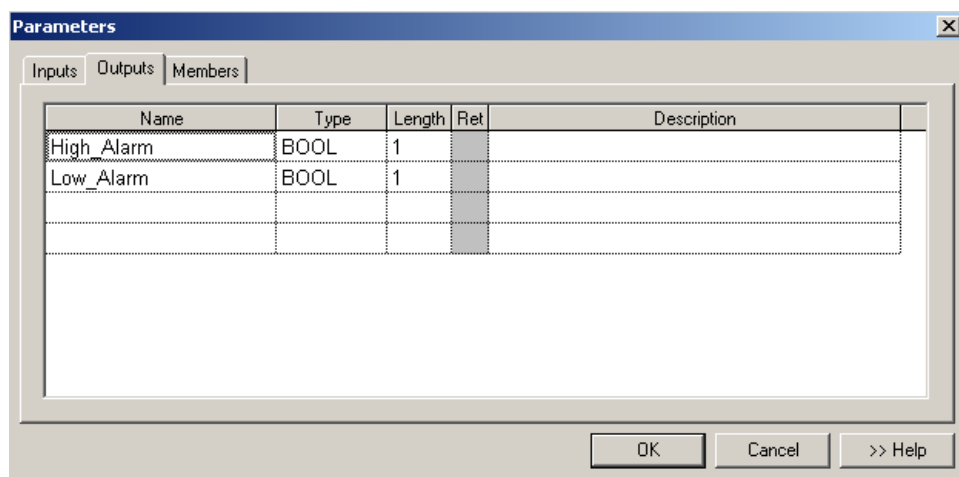
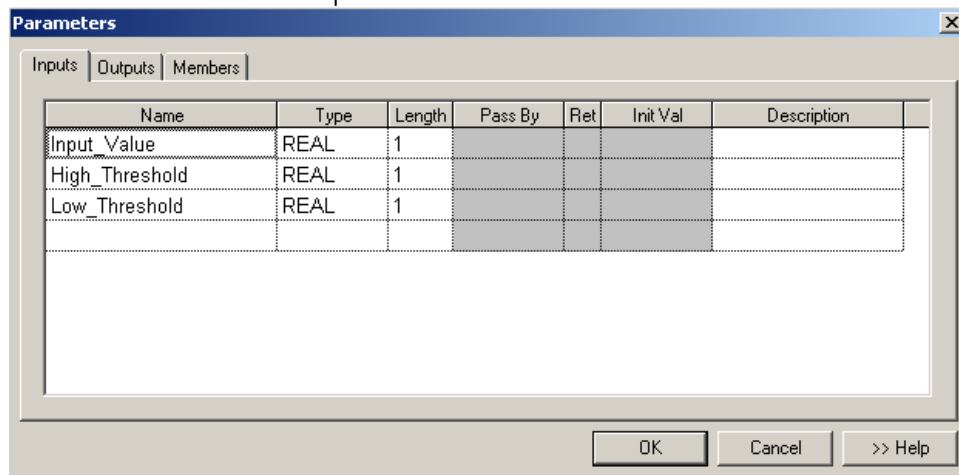
This is a **Parameterized Block (PSB)**. This is a program block that has input and output parameters associated with it. This block can be used over and over in the logic that you create whenever it is needed. A Parameterized Block takes the inputs provided, executes the logic inside the Block, and then updates the Block's outputs. It differs from a User Defined Function Block in that it does not have any Instance or Member variables that are retained from one call to another.



4. Next, click in the field next to the word **Parameters** and a button will appear.



5. Click on this button to view the parameters associated with this block.

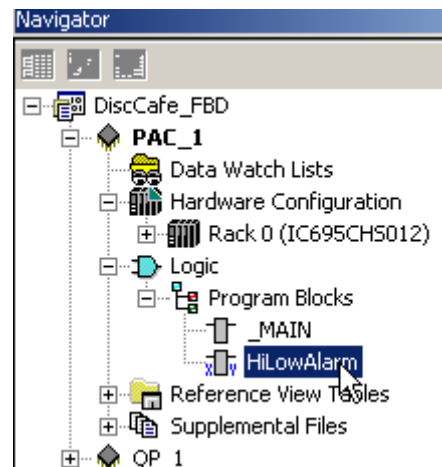


## NOTE

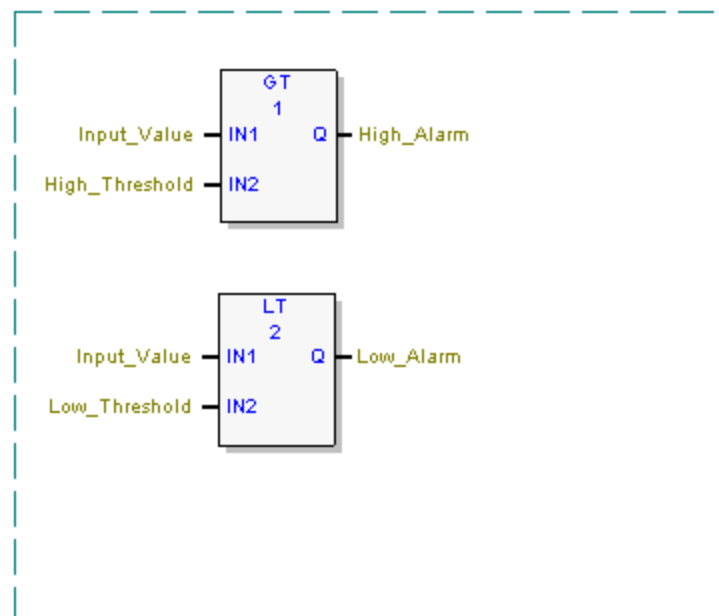
**There are 3 Input Parameters and 2 Output Parameters associated with this block. The block has no Member Variables.**

User Defined Function Blocks (UDFBs) have Member Variables as well as Input and Output Parameters. UDFBs are explained in the User Defined Function Block lab.

6. Next, **double-click** on the block to open it in the editor window.

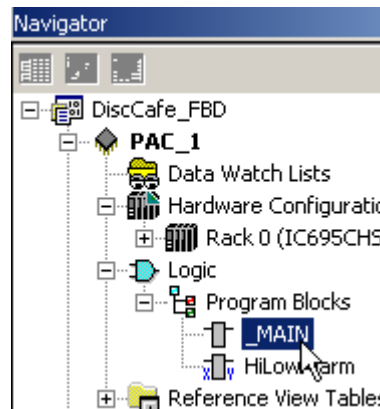


The following contents will be displayed for the HiLowAlarm block.

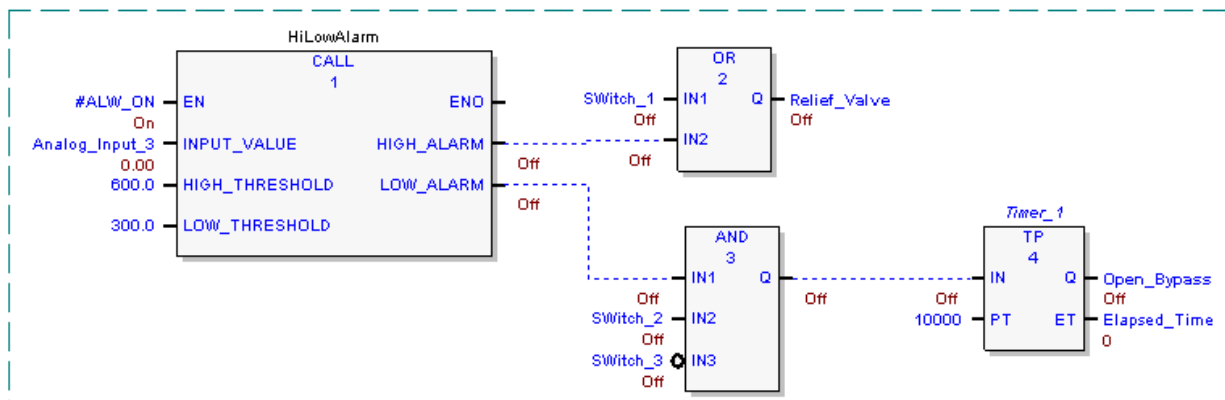


The logic in this block uses a **Greater Than (GT)** block and a **Less Than (LT)** block to generate **High and Low Alarms**. When this block is called users can enter the variables of their choice for the input and output parameters.

7. Next, **double-click** on the block labeled **\_MAIN**.



The contents of the **\_MAIN** FBD block are now displayed as follows. Please note that wires that are dashed represent **Boolean** values, where solid wires represent non-Boolean values such as Real and Integer values.



The logic for this block is explained in the Text block on the diagram.

This program Calls the Parameterized Block HiLowAlarm.

As the value of Analog\_Input\_3 changes high and low alarms are generated.

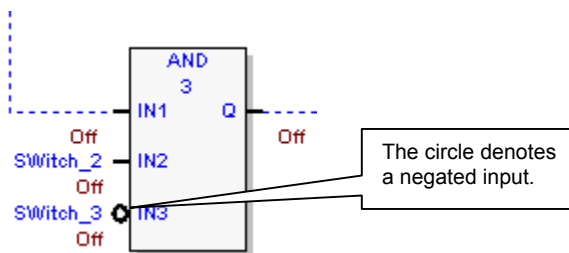
If a high alarm occurs or if Switch\_1 is on then the Relief\_Valve output of the OR block will be on.

If a low alarm occurs and if Switch\_2 is on and if Switch\_3 is not on (negated input on AND block) then pulse timer Timer\_1 will output a pulse that will open a bypass for 10 seconds.

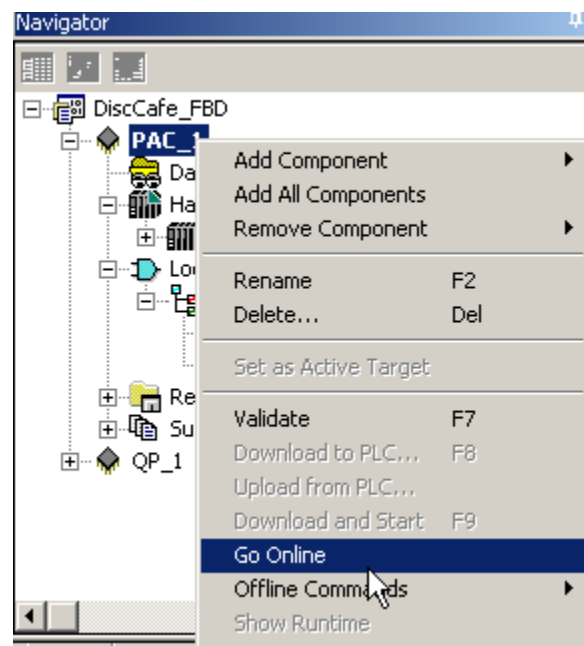
## NOTE

By default, the **\_MAIN** block is created as an LD block, but in this folder it has been defined as an FBD block. You can change the language of the **\_MAIN** block by deleting the **MAIN** block, then create a new block in the language that you desire (FBD language for example), and then renaming the new block **"MAIN"**.

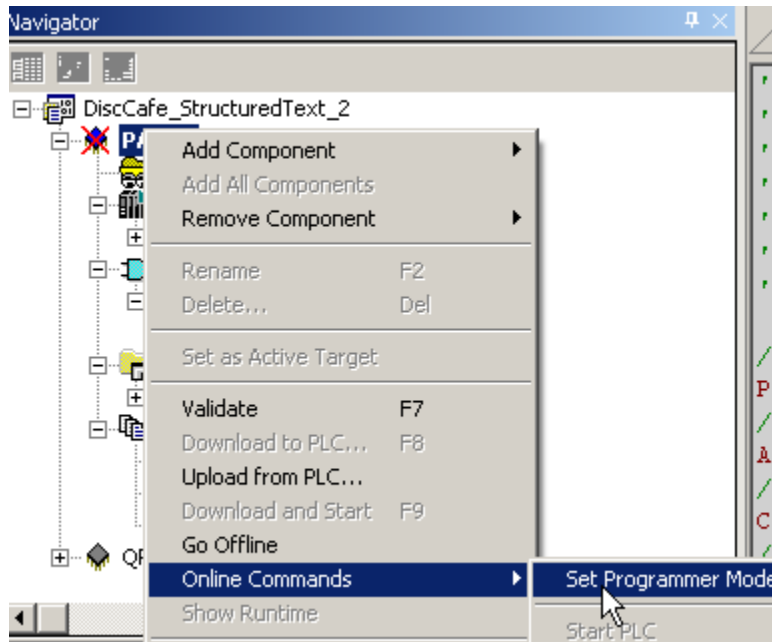
8. Observe that the **Switch\_3** input on the **AND** instruction has been **negated**.



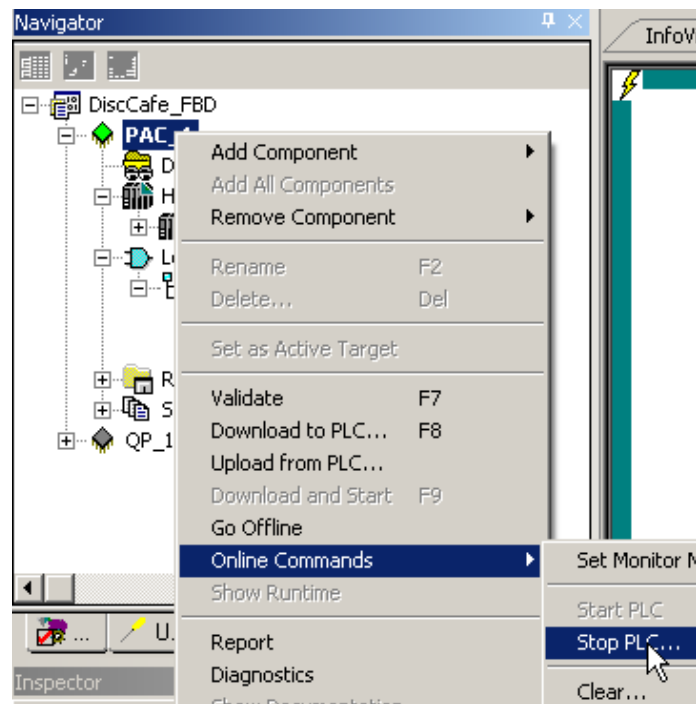
9. Next, **right-click** on the **PAC\_1** target and select **Go Online** from the menu, or press the **Online/Offline** toolbar button ⚡.



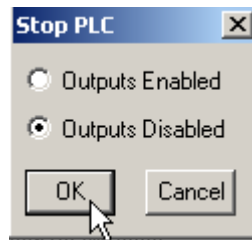
10. **Right-click** on the **PAC\_1** target and select **Online Commands, Set Programmer Mode** menu, or press the **Toggle Online Mode** toolbar button 🖱️.



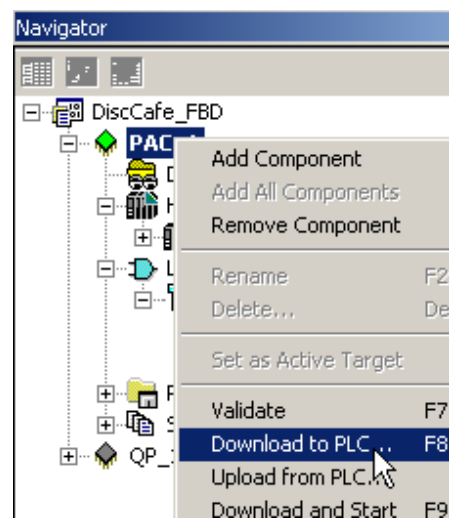
11. Right-click and select the **Online Commands, Stop PLC** menu, or press the **Stop Active Target** toolbar button ■.



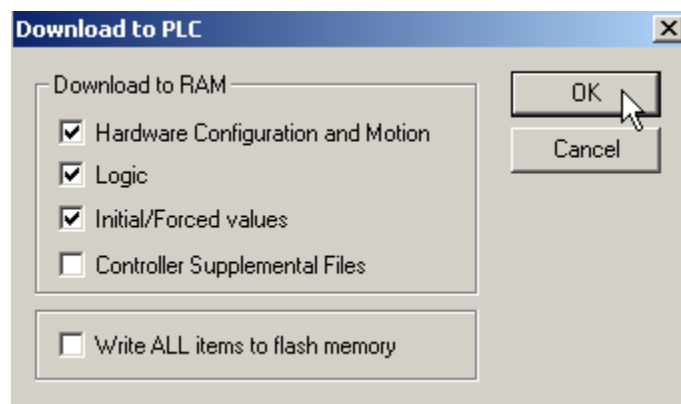
12. Click **OK**.



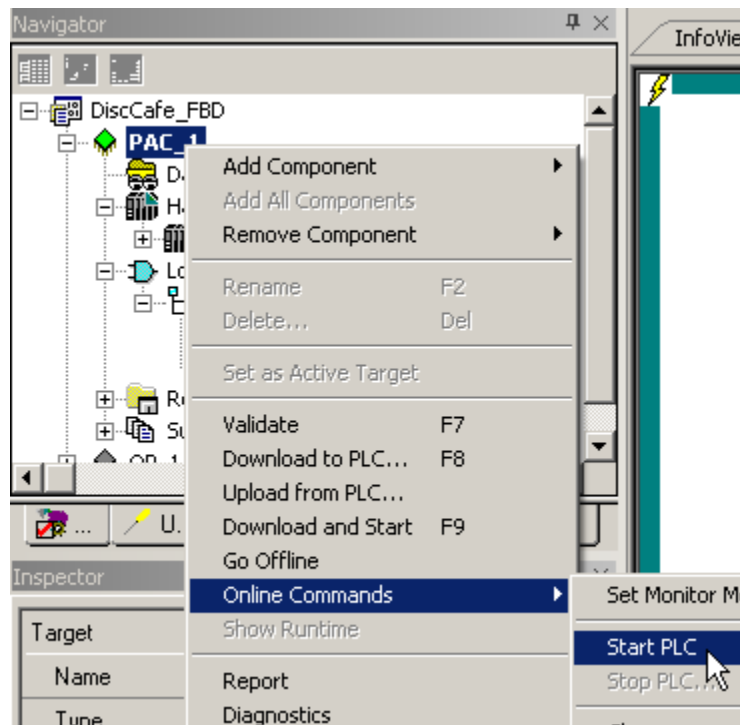
13. Right-click and select the **Download to PLC** menu, or press the **F8** function key, or press the **Download Active Target** toolbar button .



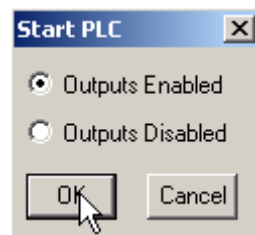
14. Make sure that the boxes are checked as shown and click **OK**.



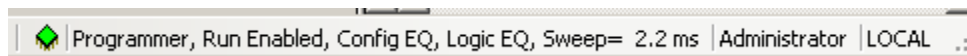
15. Right-click and go to the **Online Commands, Start PLC** menu, or press the **Start Active Target** toolbar button ►.



16. Click **OK**.

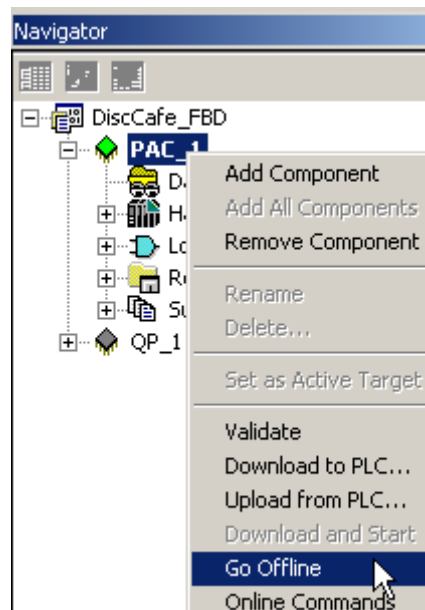


17. Check the information displayed on the Status Bar to verify that the PACSystems controller is in Run Mode, and that the Configuration and Logic are both Equal.

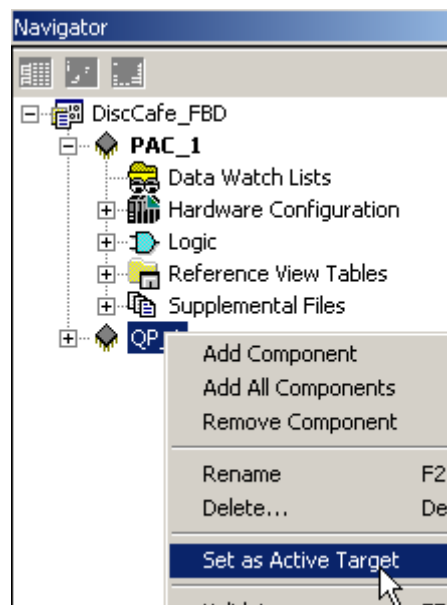


Once you have successfully downloaded and started the controller, you can use the QuickPanel to exercise the application. But first, you need to download the QuickPanel application to the demo unit.

18. Right-click on the **PAC\_1** target and select **Go Offline** from the menu, or press the **Online/Offline** toolbar button ⚡.

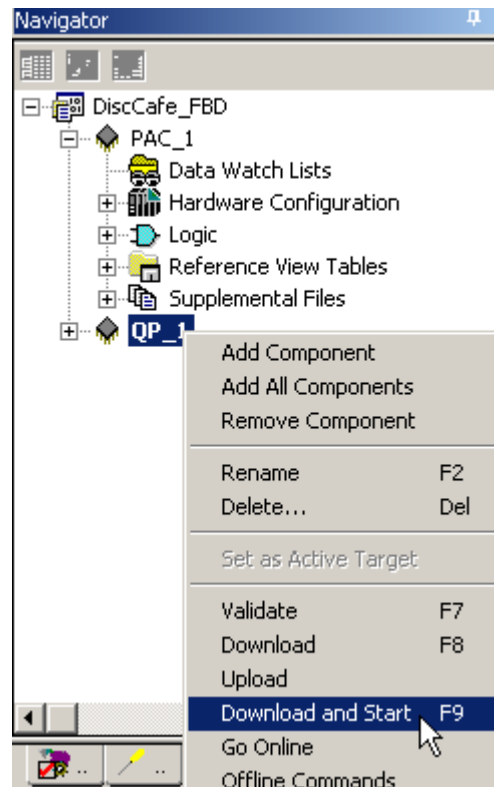


19. Right-click on the **QP\_1** target and select **Set as Active Target** from the menu.

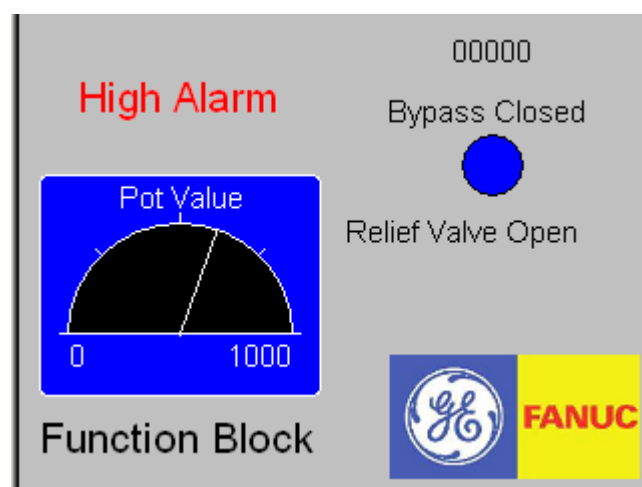


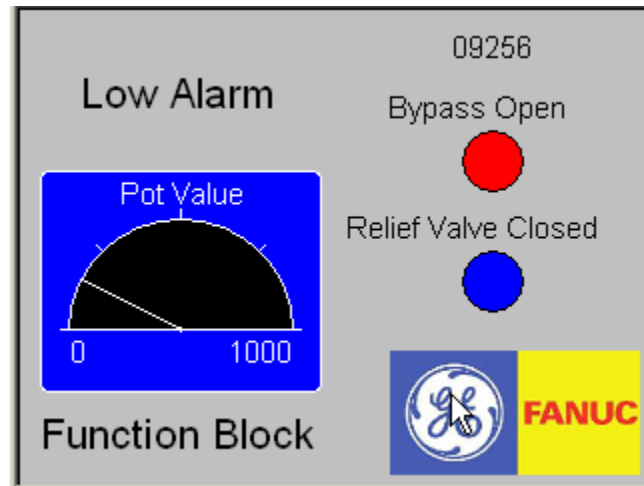


20. Right-click on the **QP\_1** target and select **Download and Start** from the menu, or press the **F9** function key, or press the **Download and Start Active Target** toolbar button .



21. After the **QP\_1** application has been successfully downloaded to the QuickPanel go to the QuickPanel screen on the PACSystems demo unit and continue as instructed below.
22. Move the **potentiometer** on the front of the **PACSystems demo unit** labeled **Analog Input** and observe the results on the QuickPanel screen.

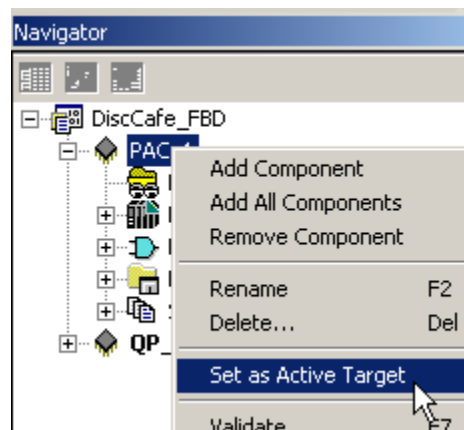




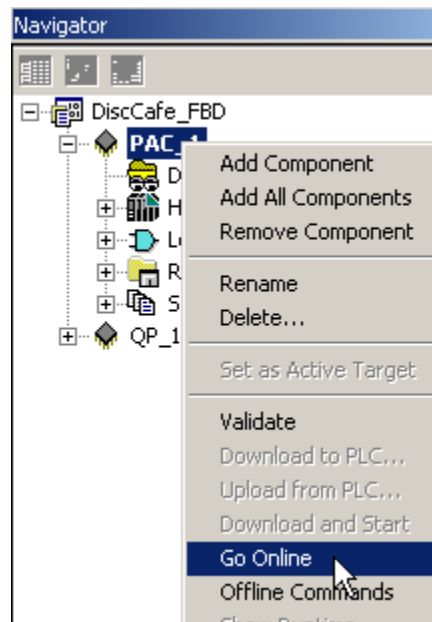
You can view the high and low alarms and see the elapsed time of the pulse timer and see the status of the bypass and relief valves.

If you want to examine the execution of the Function Block Diagram logic while you interact with the QuickPanel, you can go back online with the controller.

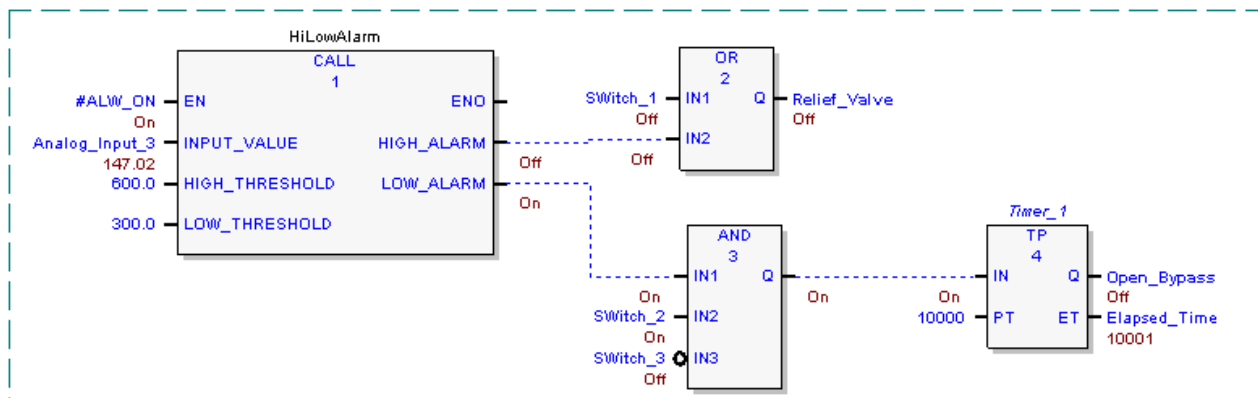
23. Right-click on the **PAC\_1** target and select **Set as Active Target** from the menu.



24. Right-click on the **PAC\_1** target and select **Go Online** from the menu, or press the **Online/Offline** toolbar button ⚡.



25. Next, right-click and select the **Online Commands, Set Programmer Mode** menu, or press the **Toggle Online Mode** toolbar button 🖱️.



## Review

In this lab, you have:

- Examined the configuration of a Hi Low Alarm Parameterized Block
- Seen how the Parameterized Block can be called in a logic block
- Examined a Function Block Diagram
- Downloaded FBD logic to a PACSystems controller
- Interacted with the logic using a QuickPanel operator interface