



OpenBAS

BUILDING
AUTOMATION
SYSTEM

Programming Tutorials

LT-6148 Rev. 1
January 2018

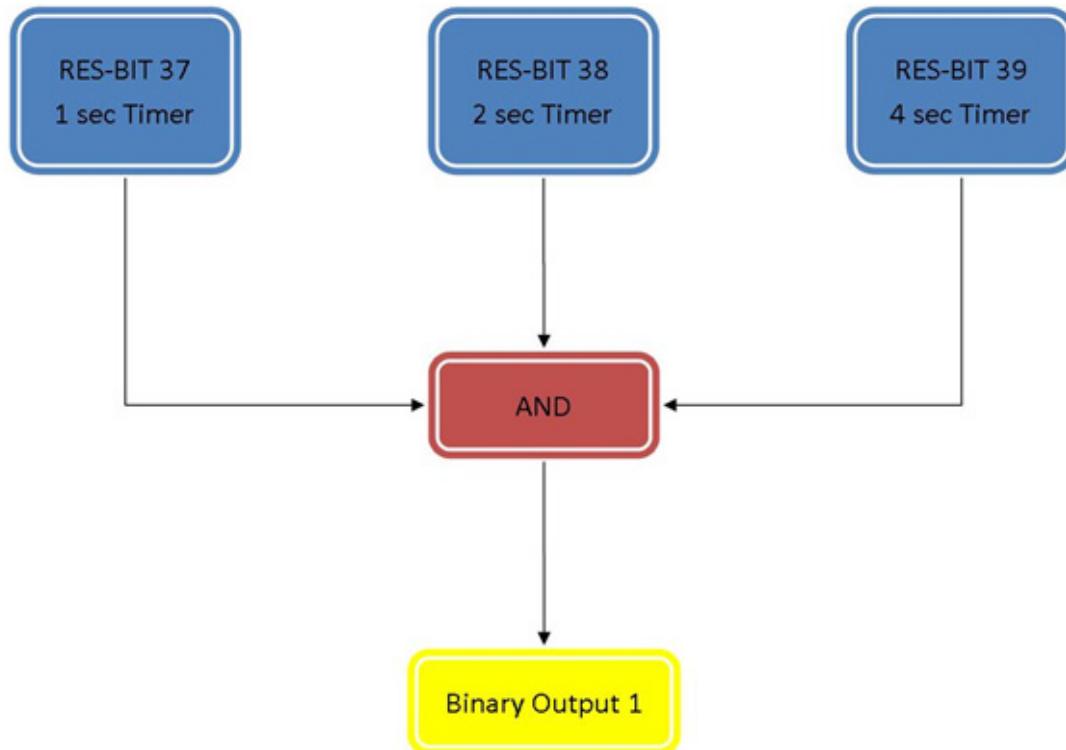
Table of Contents

1.0	Basic PLC Tutorial (AND, Output Assignment)	4
2.0	PLC Tutorial (Totalizer, K_BYTEx)	13
3.0	PLC Tutorial - Timer instruction	24
4.0	Example: PLC Tutorial (Average, Compare)	33
5.0	Example PLC Tutorial (Lighting Groups, Jump)	45
6.0	Script Programming Tutorial (Generic Application)	58
7.0	Script Programming Tutorial (Fan and Coil Applications)	71

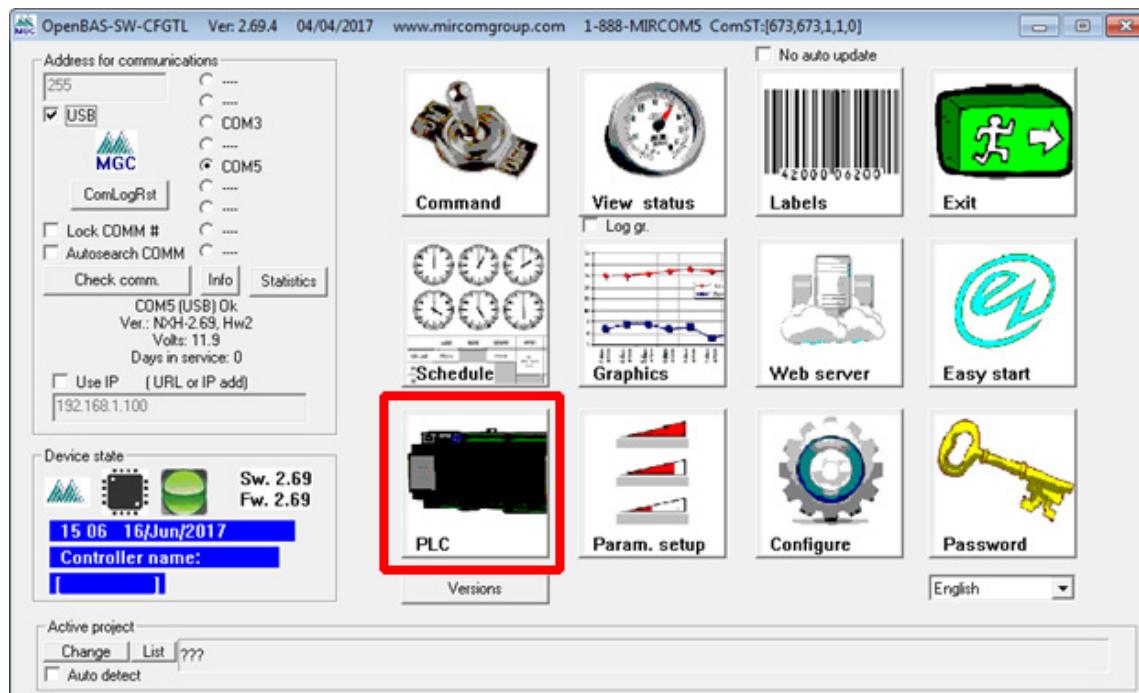
1.0 Basic PLC Tutorial (AND, Output Assignment)

This tutorial will outline an example using PLC logic controlled register bits, fixed timers and Output Assignments and Logic Instructions. The program will turn on and off a binary output as a result of an AND instruction.

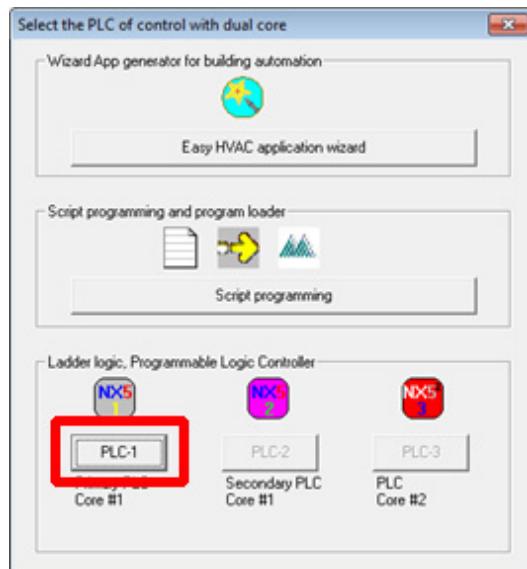
1.1 Block Diagram



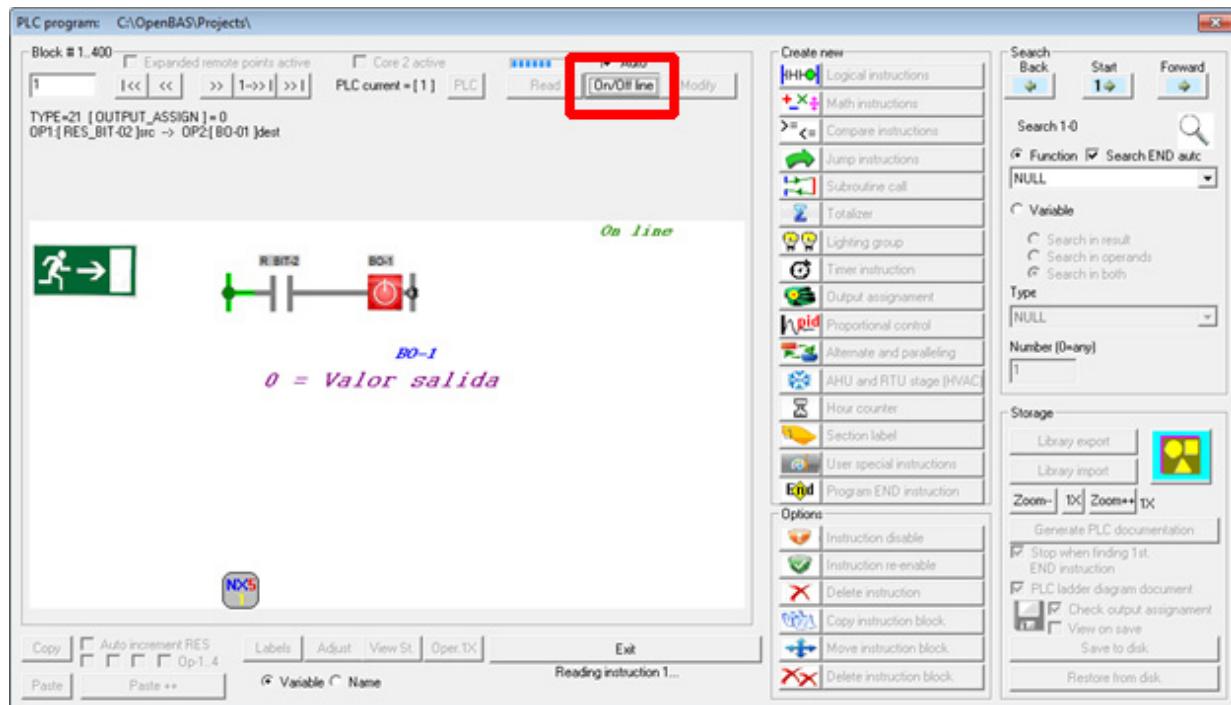
1. Select the PLC icon from the main screen of the OpenBAS software.



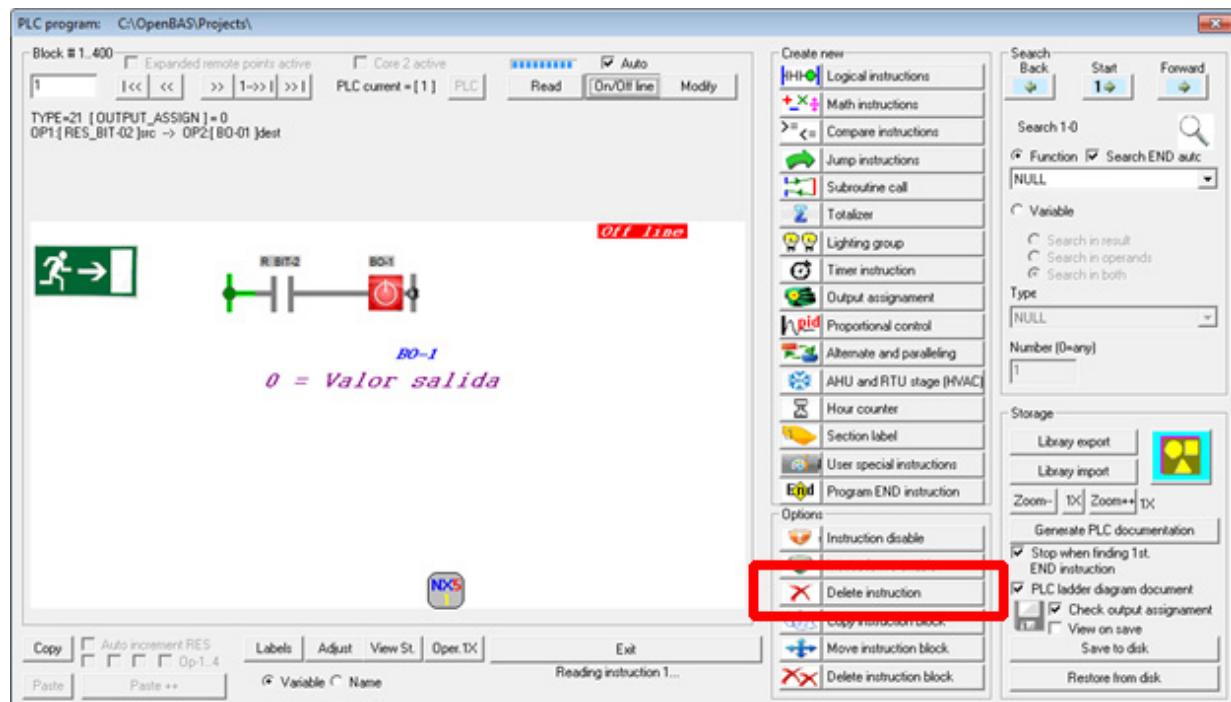
2. Select "PLC 1" as it is the only option. PLC's 2 and 3 are enabled if a dual core is installed.



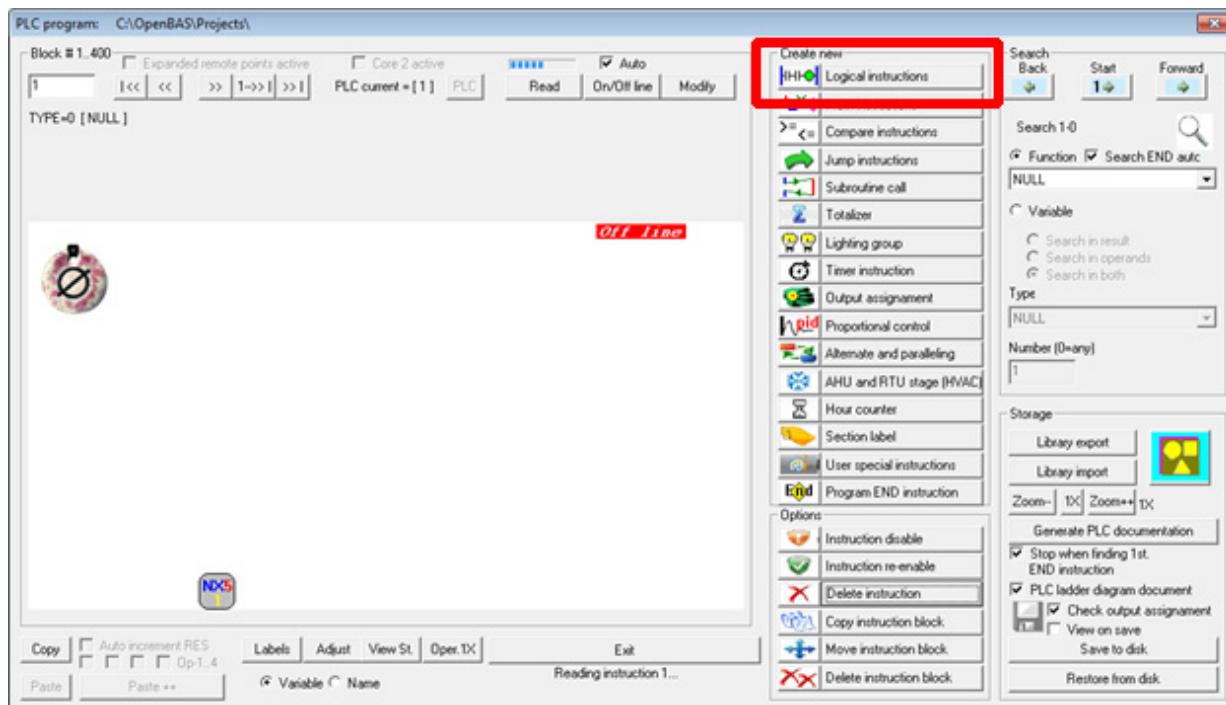
3. Press the “On/Off line” button to enable editing the logic.



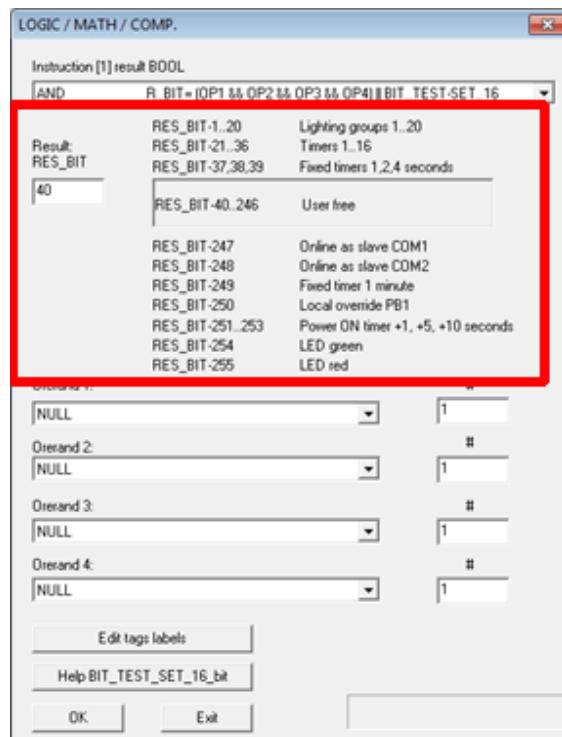
4. The functions on the right side are no longer greyed out and are usable. To begin programming, select the “Delete instruction” button to clear any outstanding logic in this block.



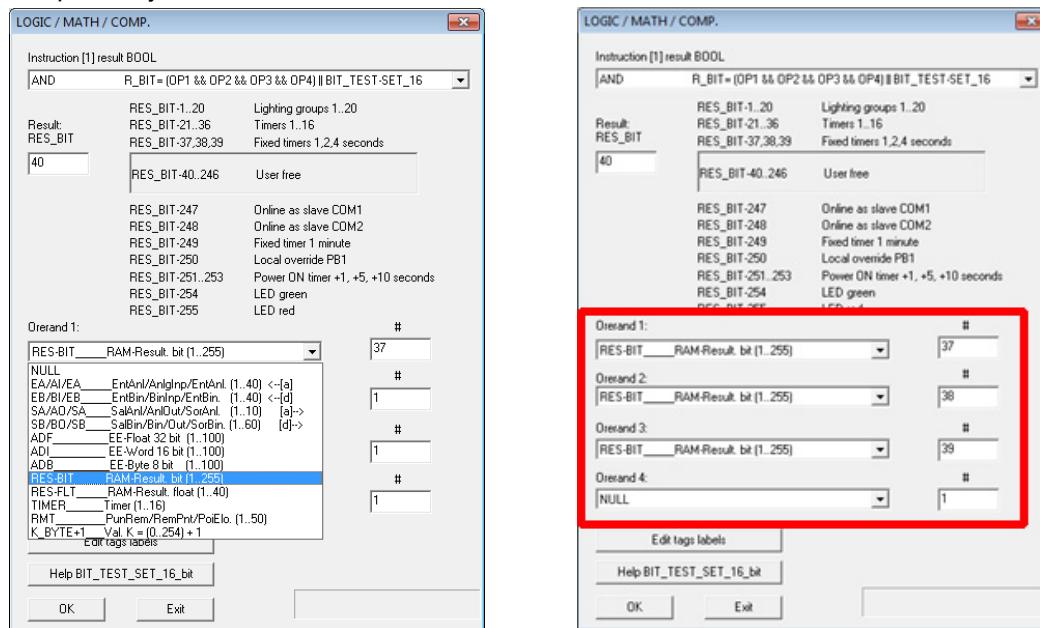
5. Select the “Logical Instruction” button to create an “AND” instruction.



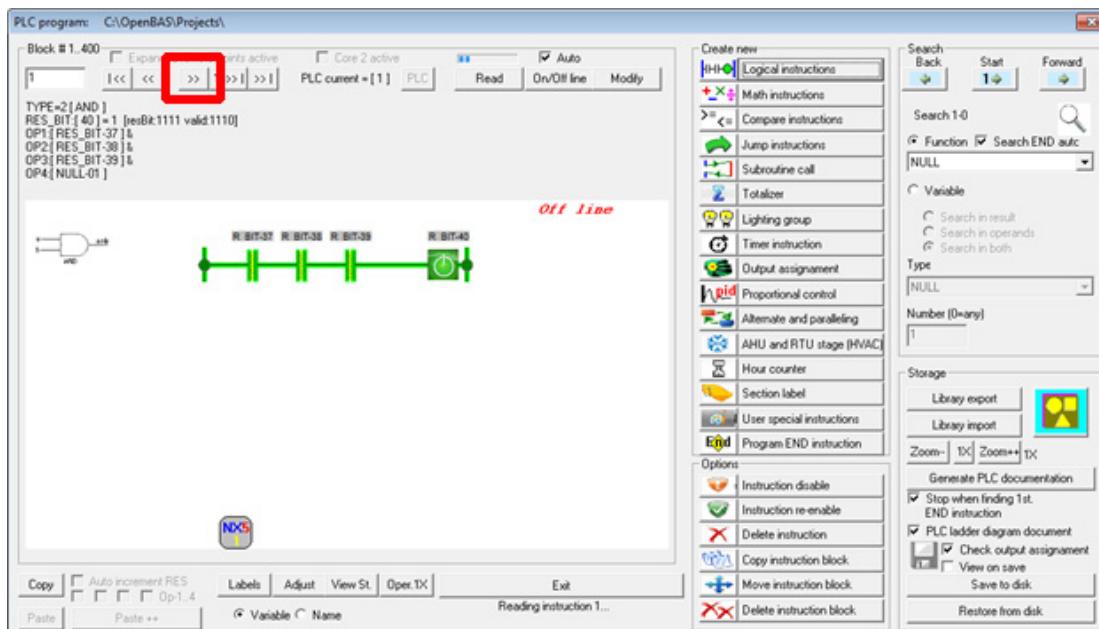
6. As shown in the dialog box below, the logic instructions involves an instruction (in this case AND), a result bit (RES_BIT) with a table of available bits and up to four operands for the instruction. In this case change the RES_BIT will be anyone of the User free bits (40 -246).



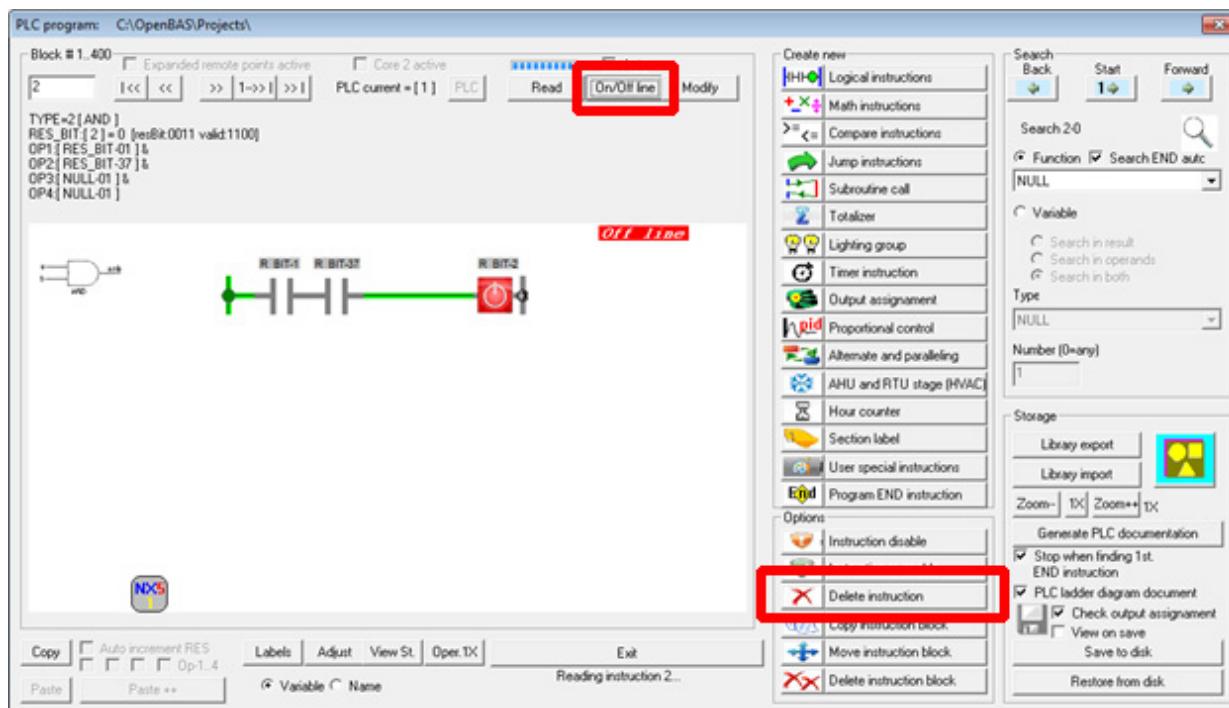
7. There are three fixed timers available located at bits 37, 38, 39. To select them, use the operand dropdown menu and select RES-BIT and change the number to 37, 38, or 39 respectively.



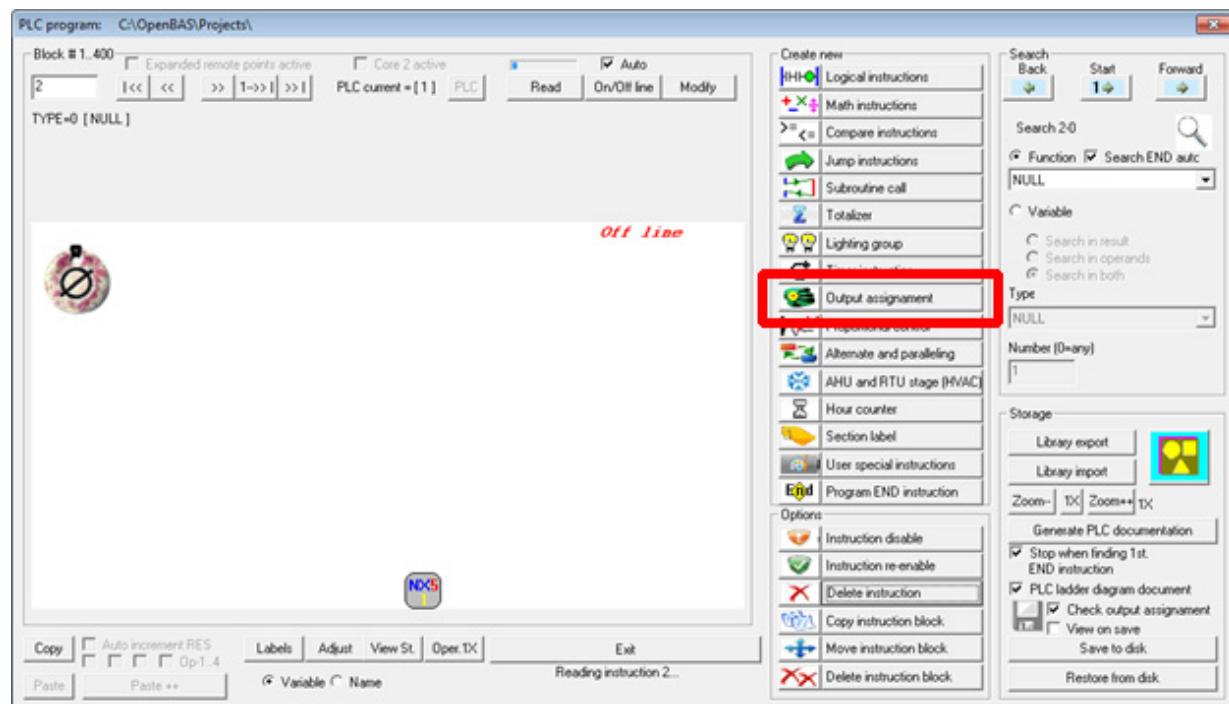
8. As shown below, the three timer bits are in an AND instruction with the R-BIT-40 as the output. Next move on to the next PLC block by pressing the “>>” button.



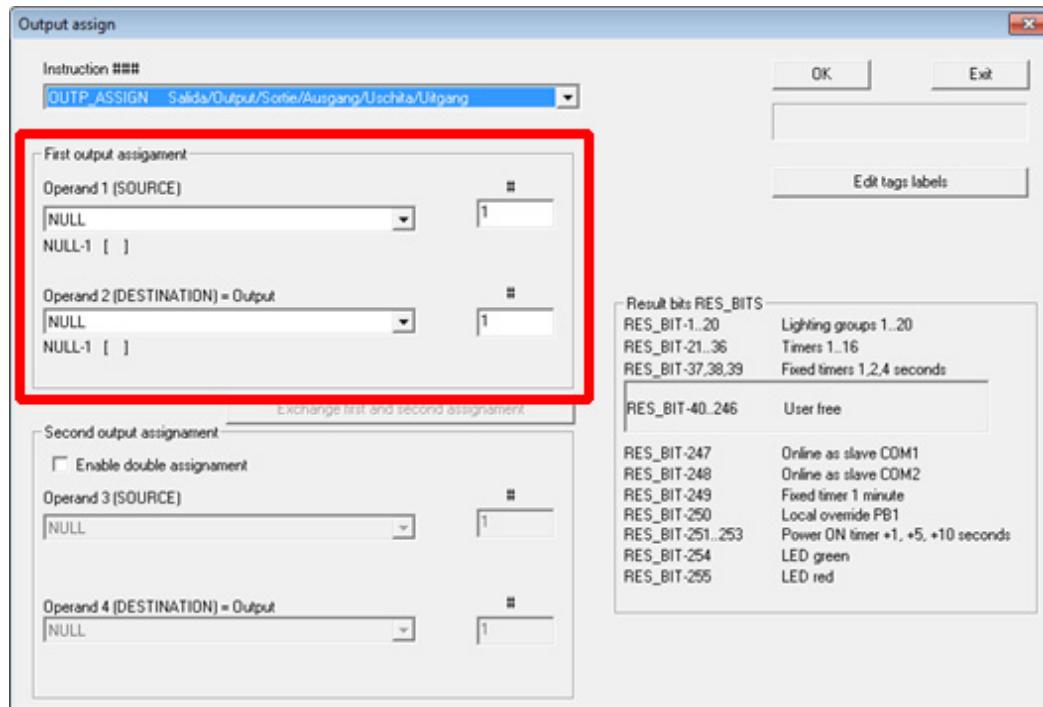
9. Again press the “On /Off line” to enable the functions and then delete any outstanding PLC logic by pressing the “Delete instruction” button.



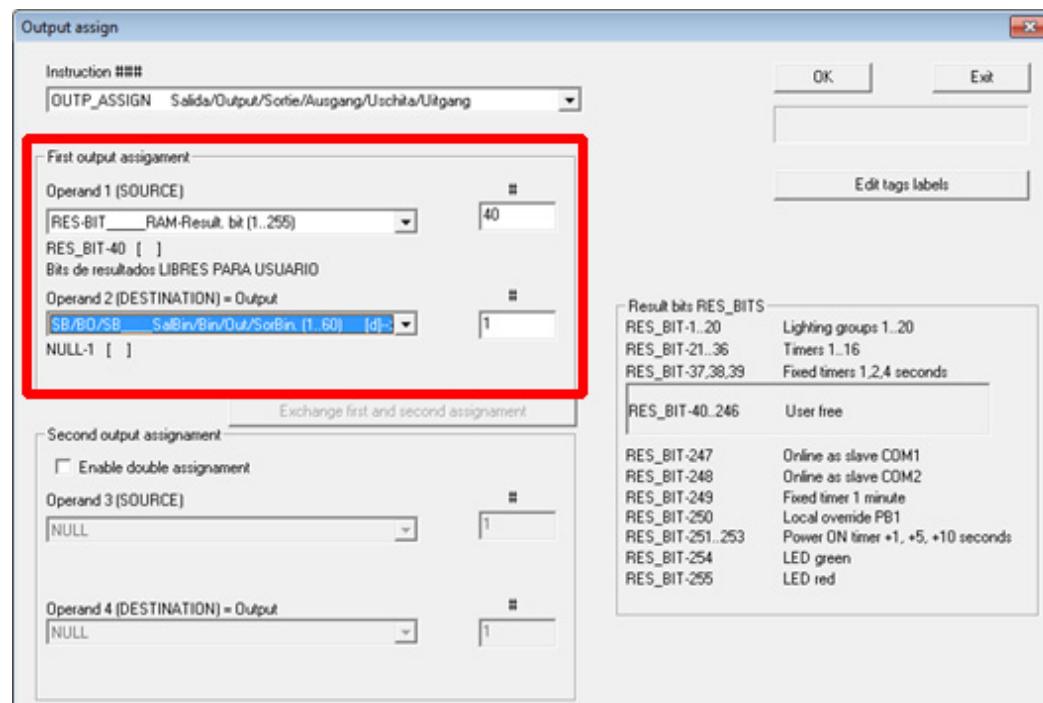
10. Select the “Output assignment” button to link the logic to an output.



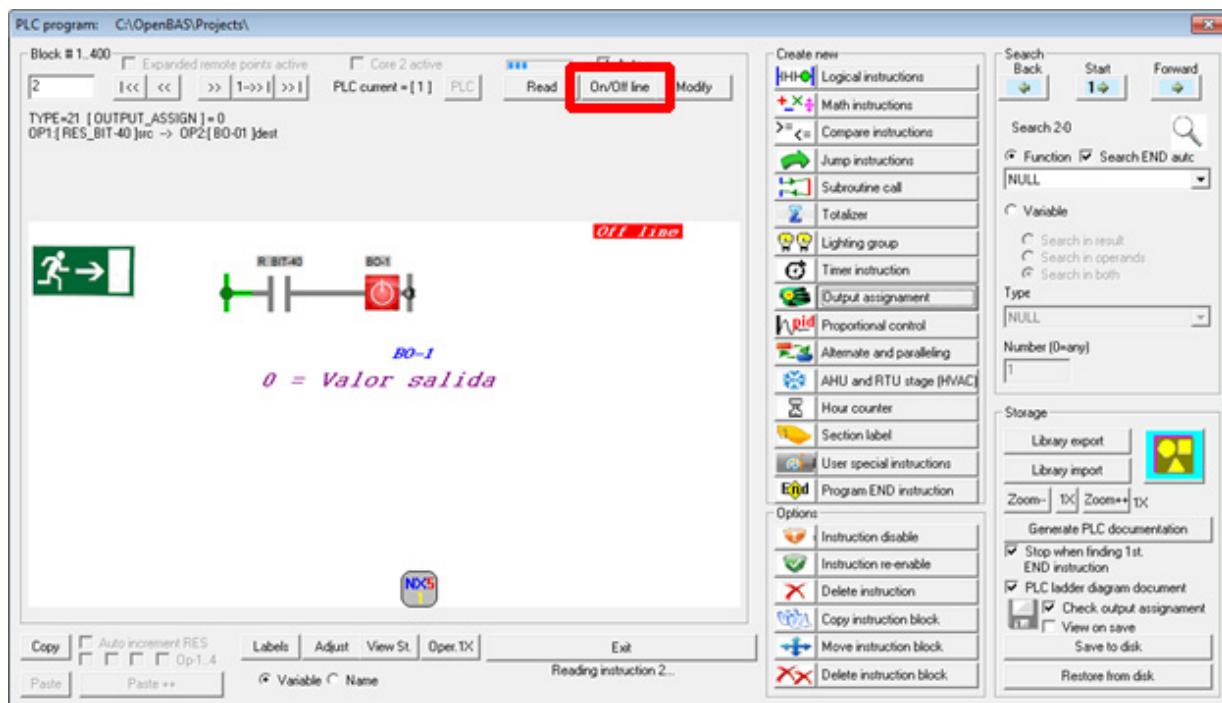
11. The Output assign dialog box again has three main parts: instruction, source and output. The instruction is by default at OUTP_ASSIGN but the source and output need to be set.



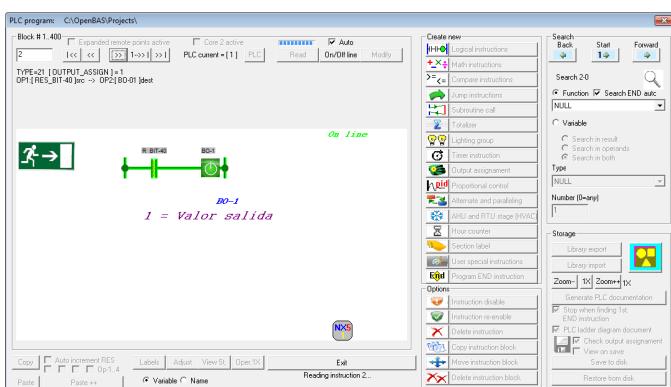
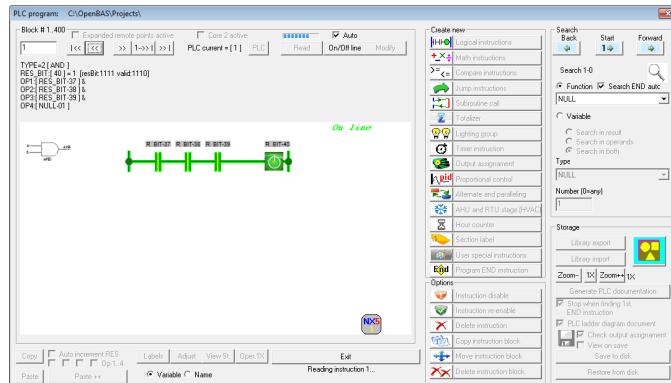
12. Select RES-BIT for the source and select 40 as the number. This makes the output from the previous block as the source for the output. For the output select Binary Out and select any available number. In this case 1.



13. The logic is now complete. Press the “On/Off line” to turn on the logic.



14. The binary output now will only turn on now when all three of the timers are on at the same time as they pulse on their different lengths.



1.2 PLC Ladder Diagram

```

+-----< OpenBAS >-----+
| PLC-1: 06/07/2017      11:10:37 hrs.
| Ver.   [NX5-2.69, Hw2]  ID: [NX522.69]  NX5-WIN: 2.69
|          www.mircomgroup.com
| Pgm.   C:\OpenBAS\Projects\
+-----[ START ]-----+
+-----<INSTR#-001>-----+
| RES_BIT-37      RES_BIT-38      RES_BIT-39      RES_BIT-40
| [---] [---] [---] [---] (R)
+-----<INSTR#-002>-----+
| ASSIGN OUTPUTS
| RES_BIT-40      BO-1
| [---] [---] ( R )
+-----<INSTR#-003>-----+
| END
+-----< OpenBAS >-----+
[FIN]

```

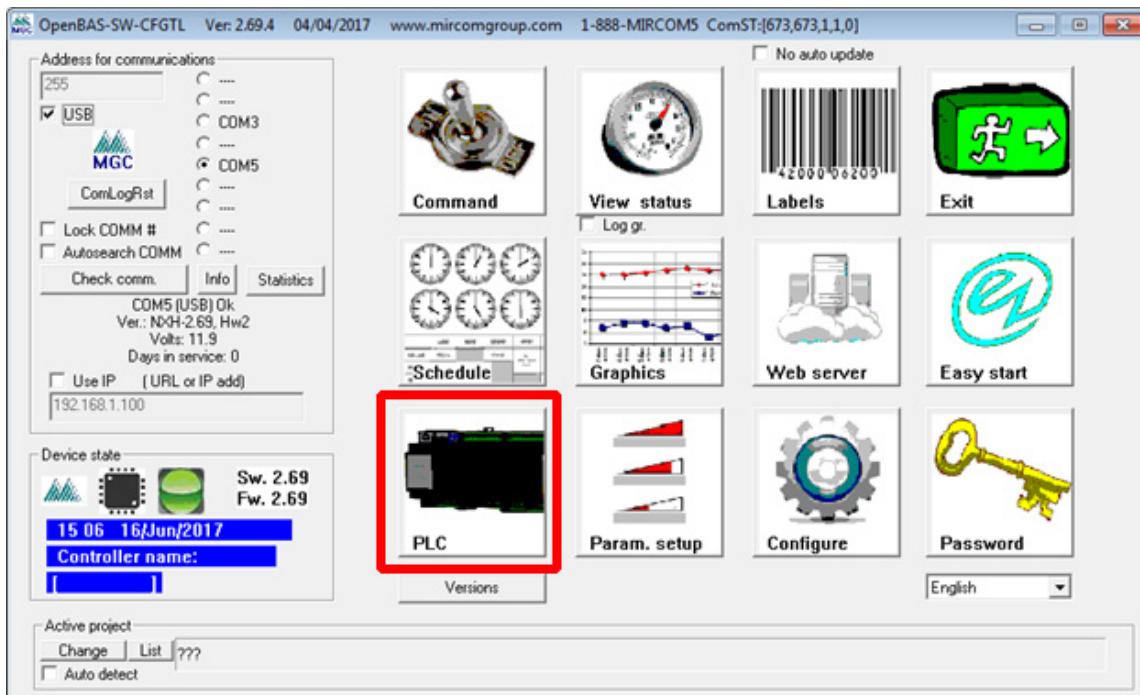
2.0 PLC Tutorial (Totalizer, K_BYT)

This tutorial will outline an example that builds on the first tutorial; “Basic PLC Tutorial”. The program will use the “Totalizer” function to count the number of pulses in a minute from the “AND” function. The first output will be on when no pulses have occurred, the second output after one pulse and so on until after seven pulses turns on the eighth output.

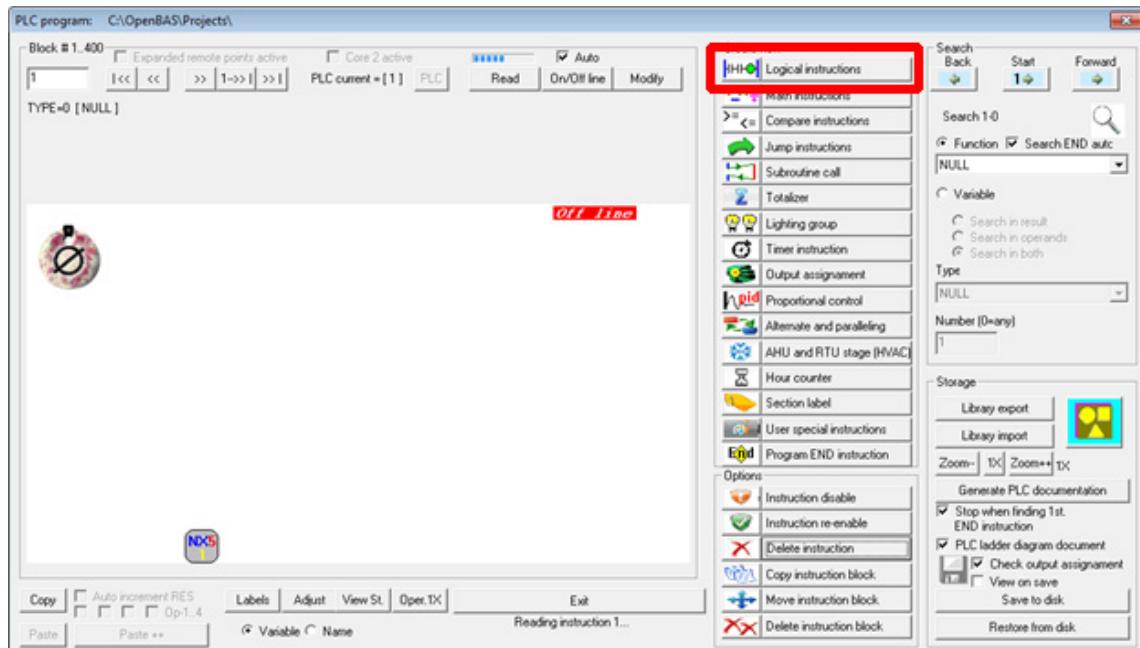
2.1 Block Diagram



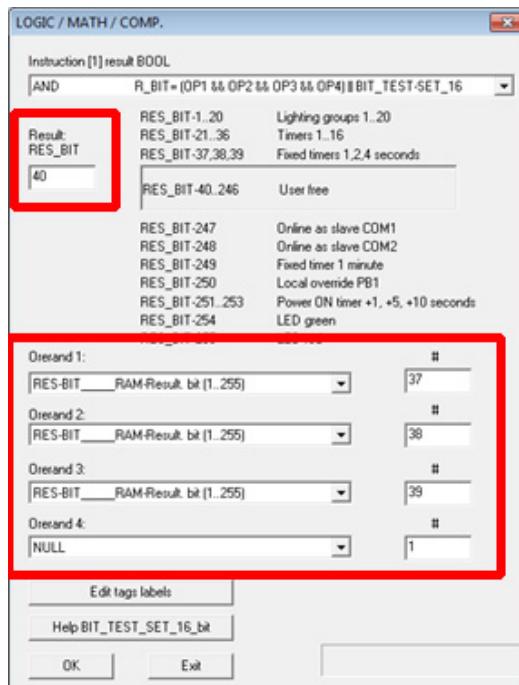
1. Select the PLC icon from the main screen of the OpenBAS software and select "PLC 1" to get to the PLC editor.



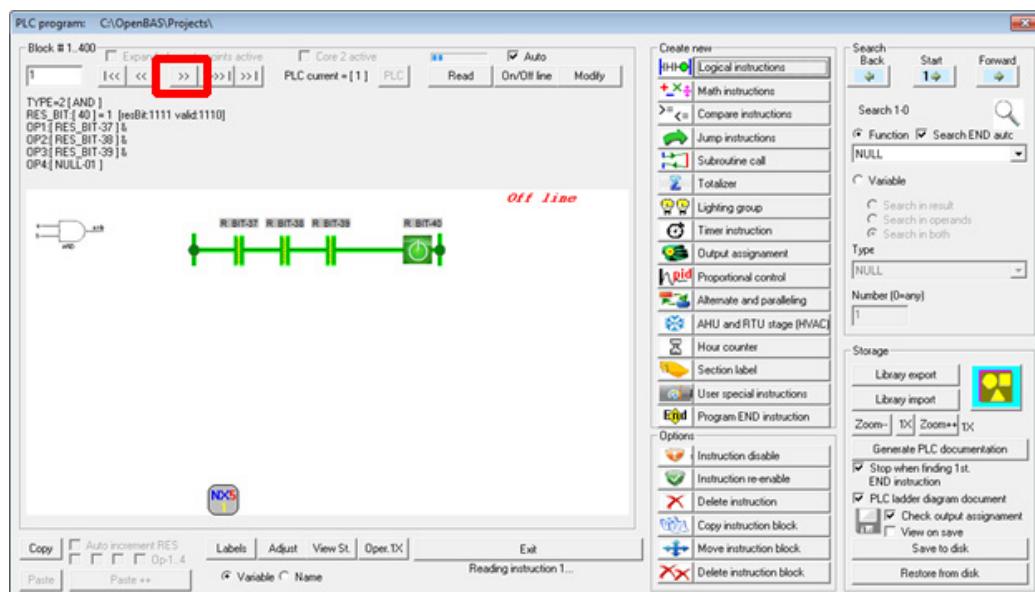
2. Clear any existing logic in the block and select the "Logical Instruction" button to create an "AND" instruction.



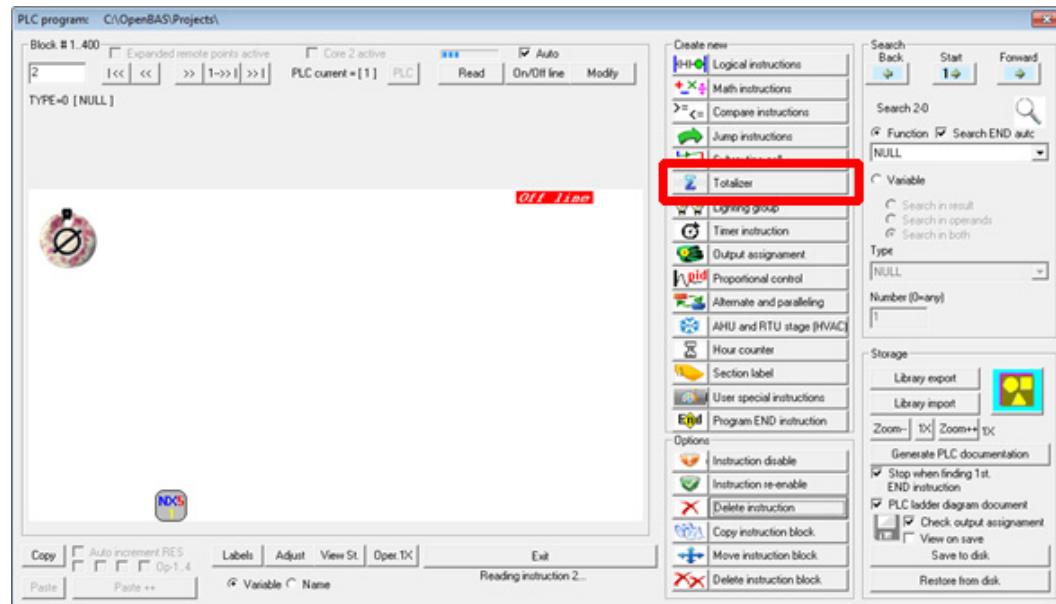
3. There are 3 fixed timers available at bits 37, 38 and 39. As shown in the image below, the result of the “AND” function of the three timers will be located at the free user #40 bit.



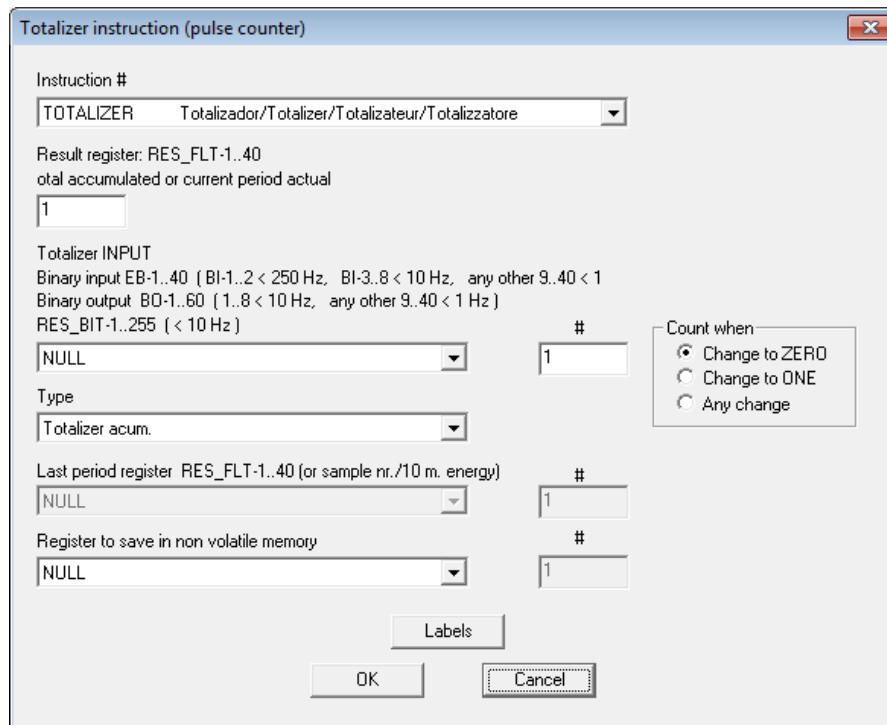
4. As shown below, the three timer bits are in an AND instruction with the R-BIT-40 as the output. Next move on to the next PLC block by pressing the “>>” button.



5. Clear any existing logic in the block and press the “Totalizer” button to create the counter.

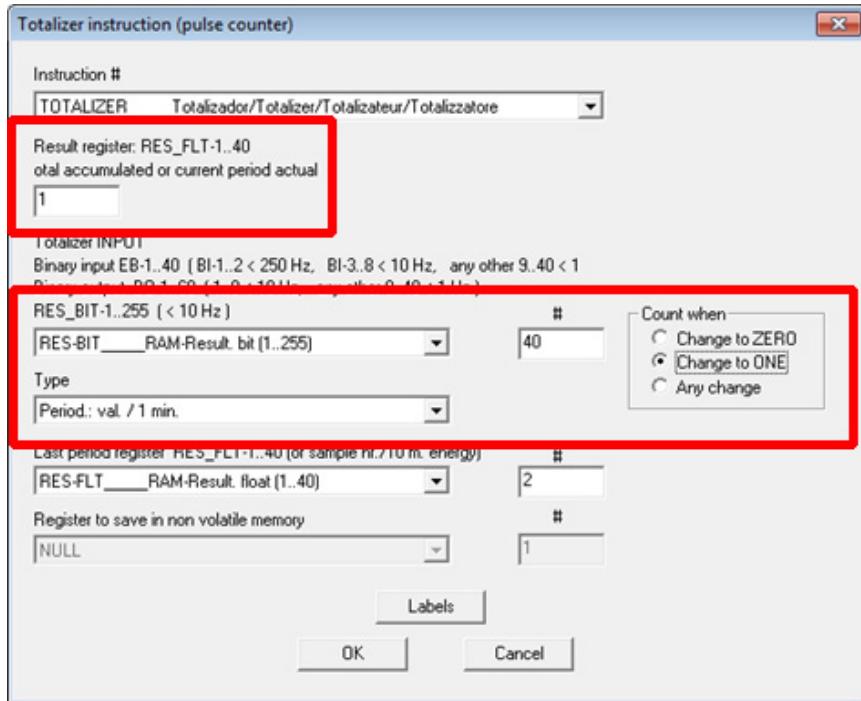


6. The software will alert the user to the allowed inputs and press “OK” to advance. The Totalizer includes a result register which has to be a float and then four input parameters. The first being the main inputting pulse being counted. The Type box which specifies the duration, two additional boxes for storing the output and a “Count when” box.

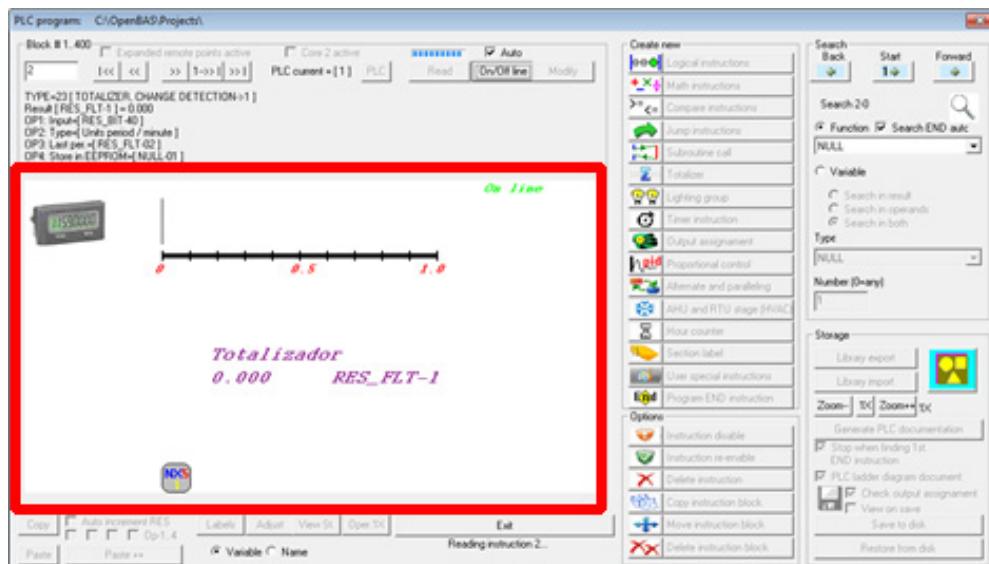


7. The most important portions of the dialog box are highlighted below. At the top is the output of the function which will be stored at the first register float. The second menu selects the input to the function which in this case the output to the “AND” function from the previous block. The next area allows you select when the totalizer count will increment which can be either “Change to ZERO” or “Change to ONE”. Next the type

drop-down box allows you to select the period on which the totalizer will count. The last two boxes should be filled out as follows.

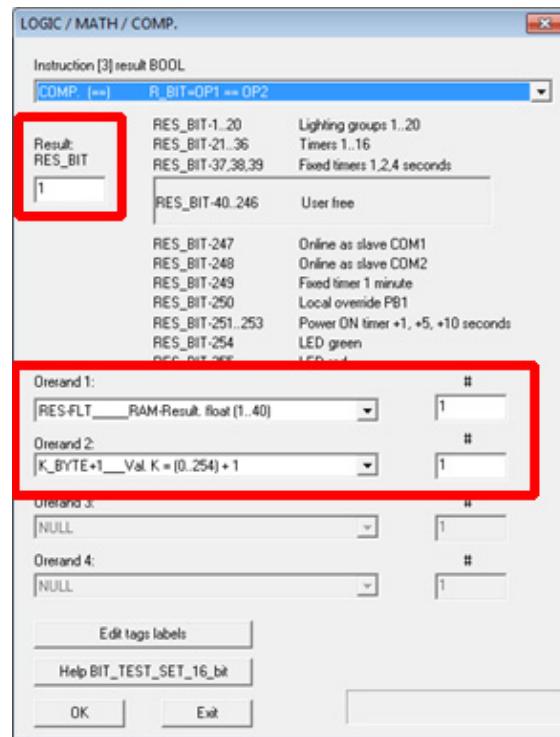


- The resulting graphic in the software shows the current count of the function and a number line displaying the data.

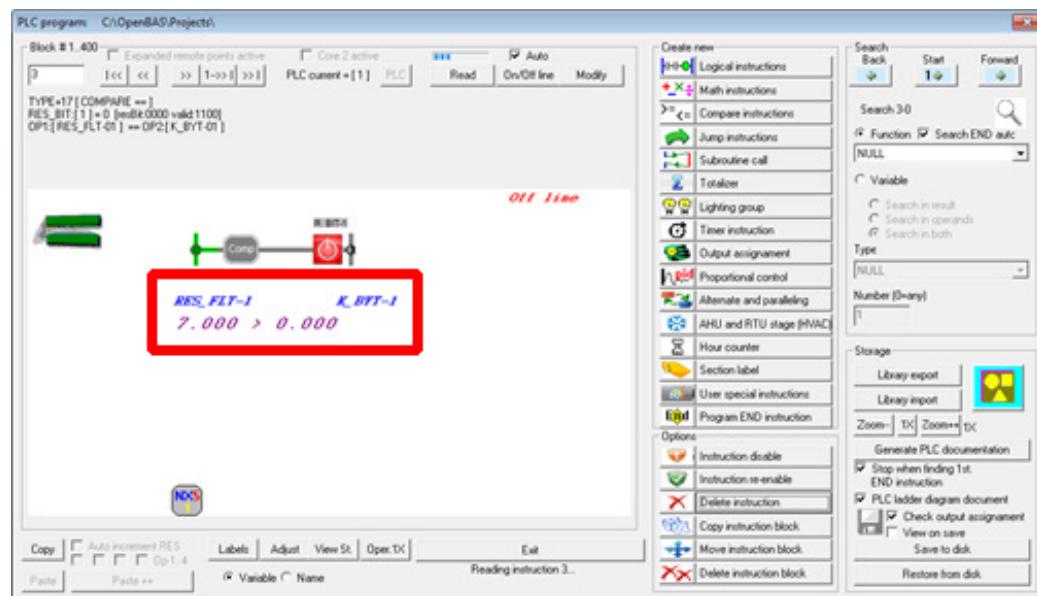


- Advance to the next PLC block, clear any existing logic and press the “Compare instructions” button. Select the “COMP. (==)” from the instruction drop-down menu. For the result register any free bit can be used but for simplicity use bit 1. For the first operand use the output of totalizer function which is register float 1. For the second

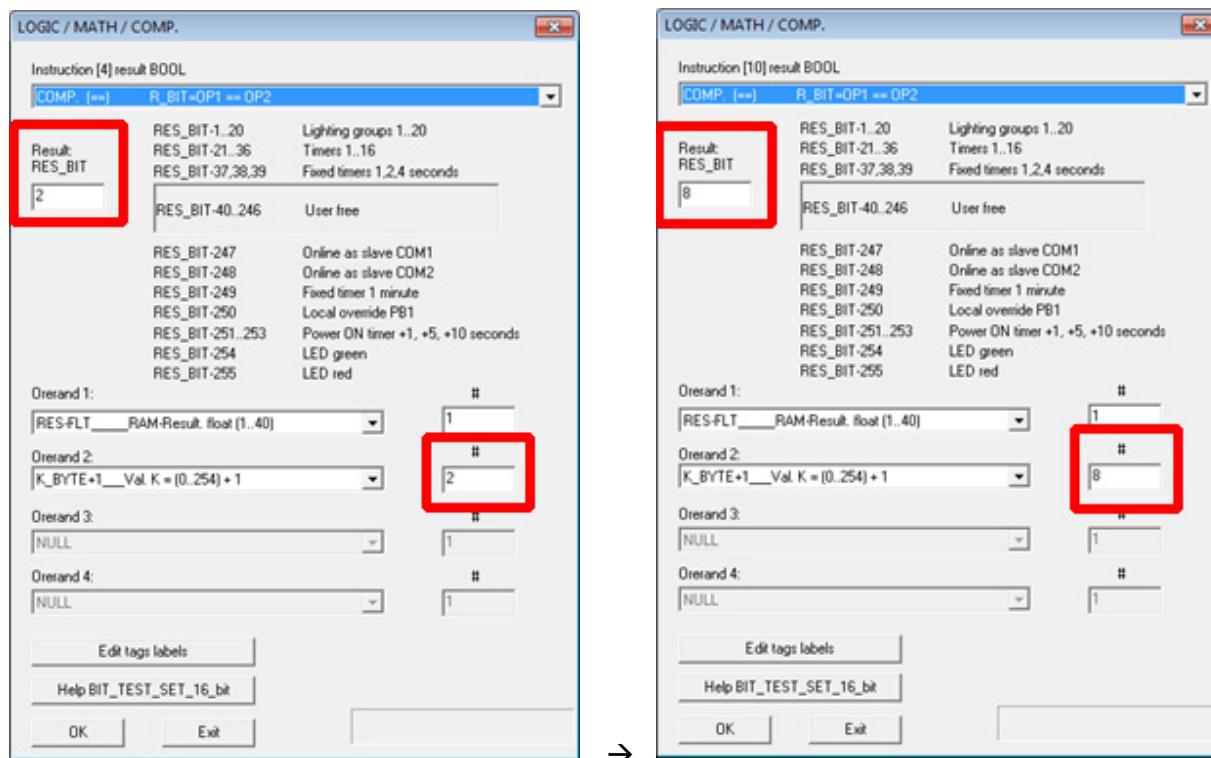
operand select the “K_BYT” option which allows you to pick a value with which to compare. The value for the first compare should be 1 as shown.



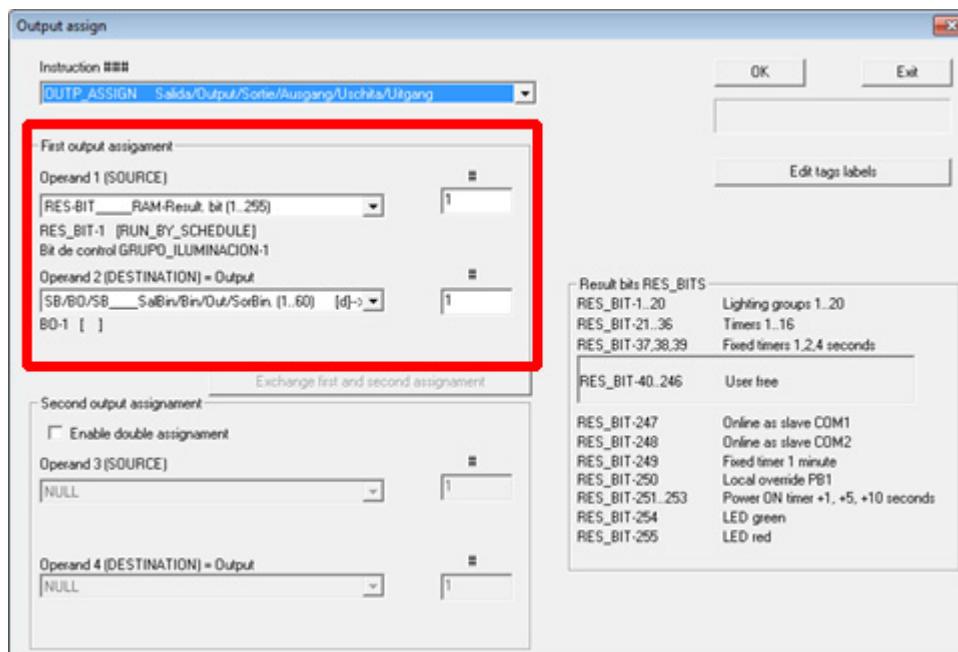
10. The K_BYT will be 1 less than the entered value as shown below.



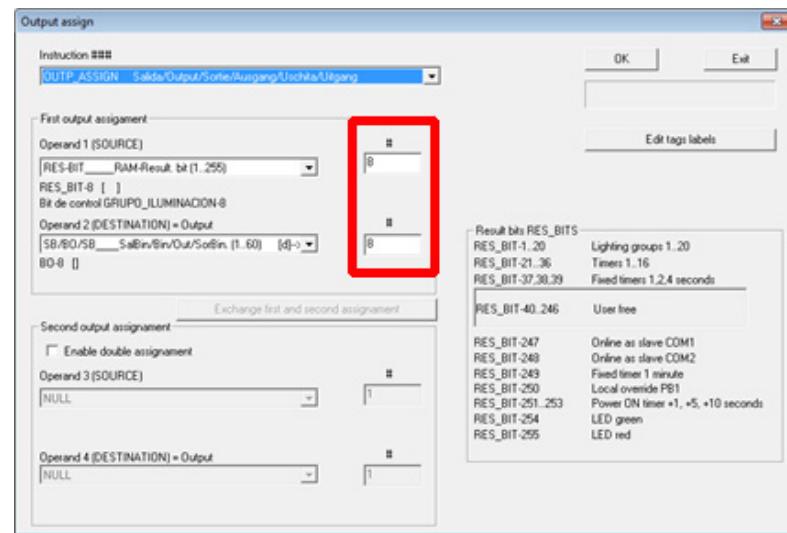
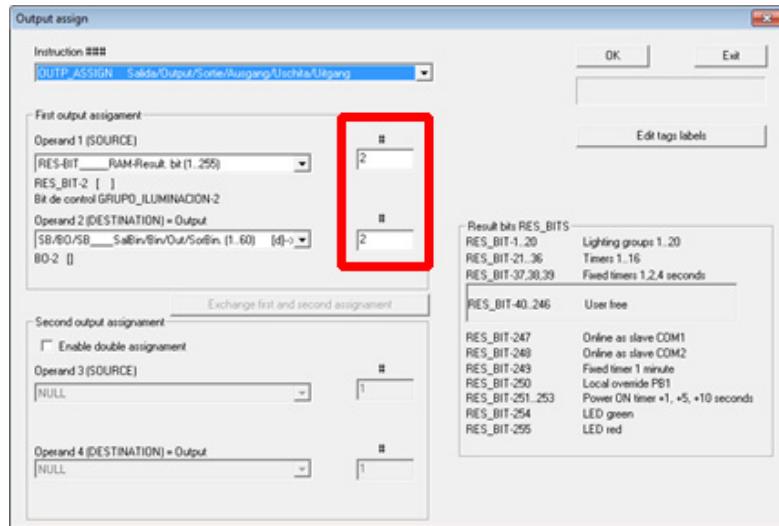
11. Repeat step 9 seven more times to add all the compare functions. Each time incrementing the output register bit and the K_BYT value be one as shown.



12. Next advance to the next PLC block and press the “Output assignment” button to assign the register bit outputs from the previous blocks to binary outputs. As shown below, the source is the first output from the comparisons. The output is the corresponding binary output.

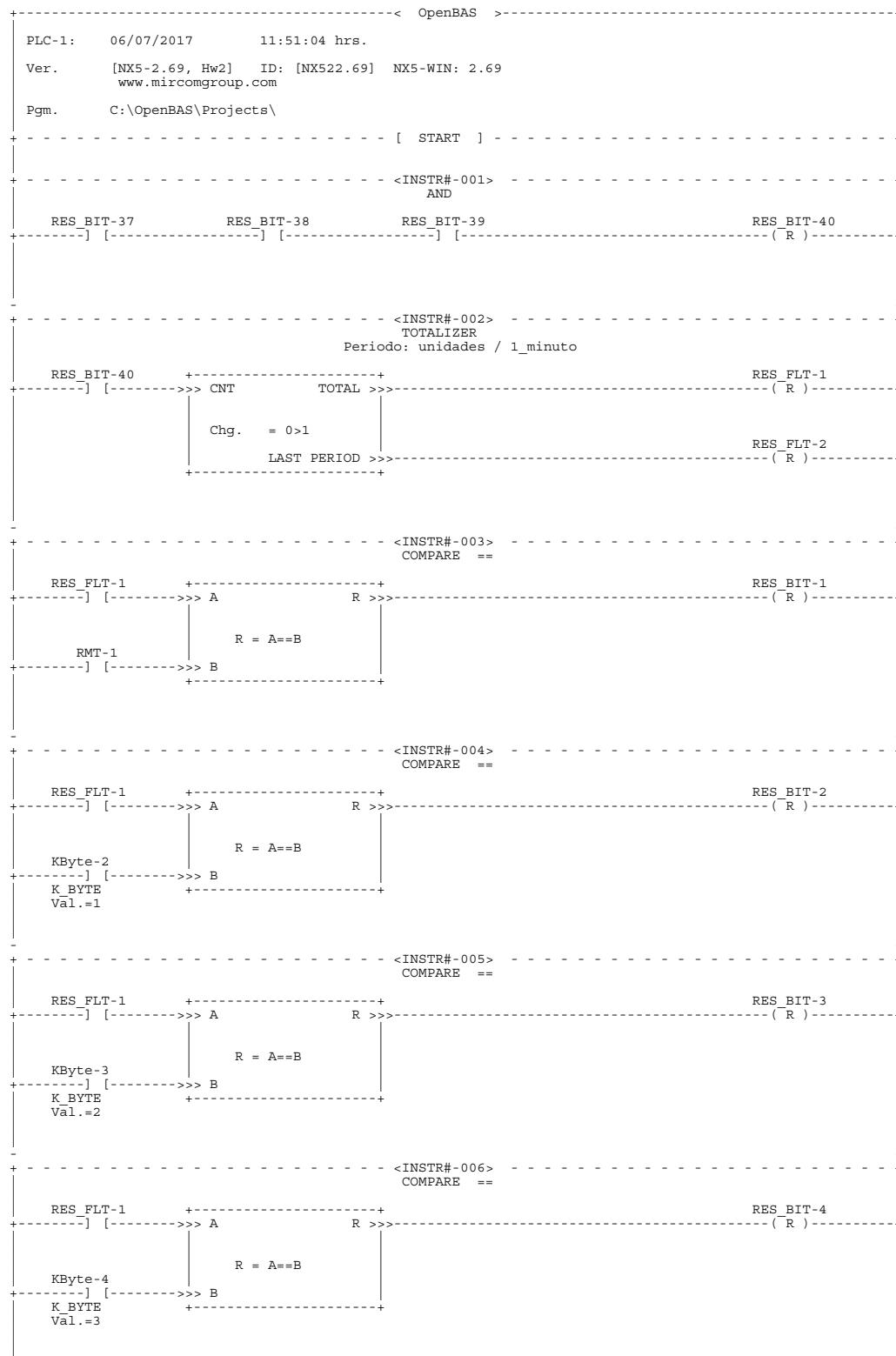


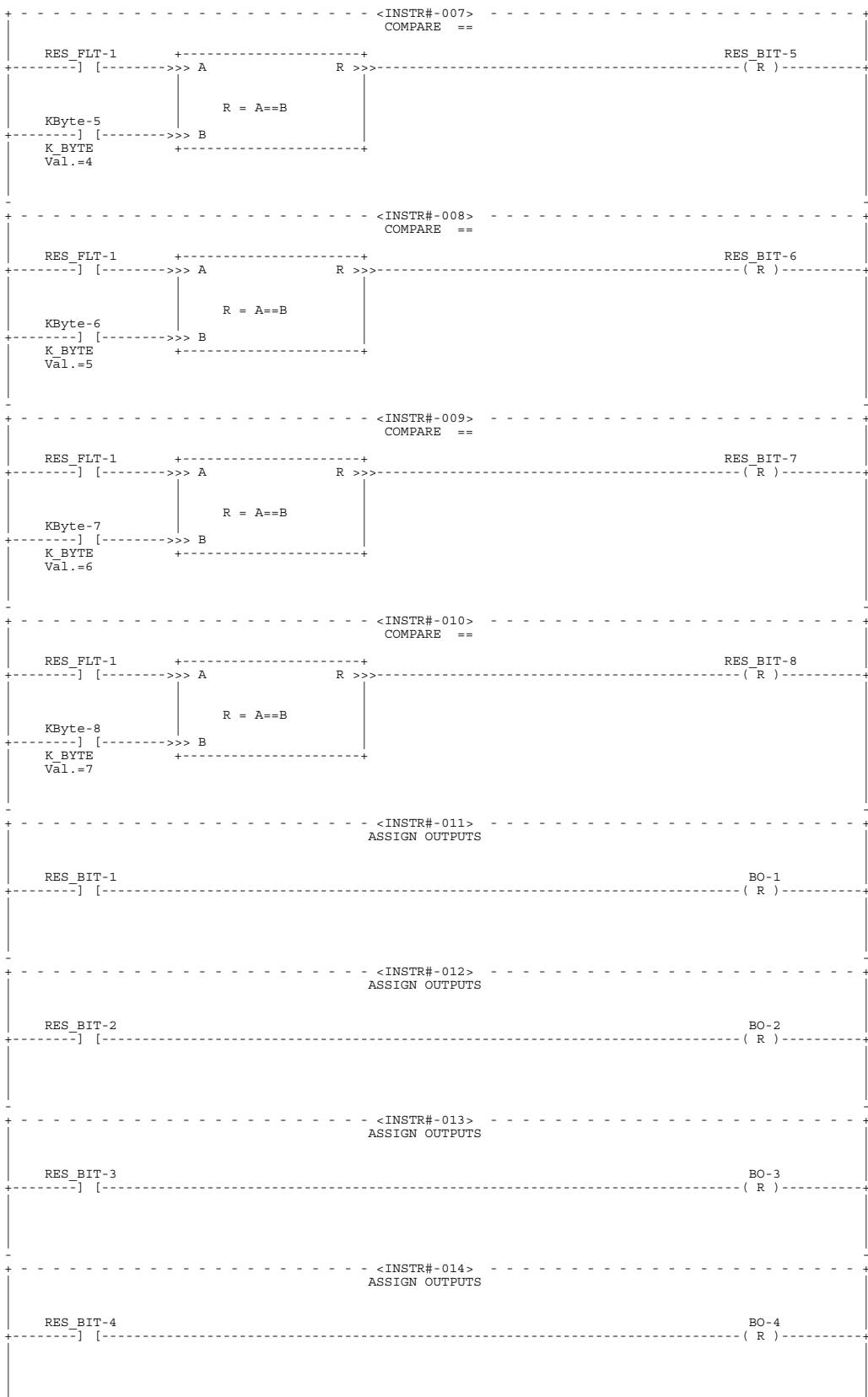
13. Repeat the previous step, 7 more times to set up all the remaining binary outputs. Each time incrementing the register bit and binary out numbers as shown.



14. The program is now complete. The program will act as a counter which counts from 1 to 8, resetting every 1 minute. Only one output is active at a time.

2.2 PLC Ladder Diagram





```

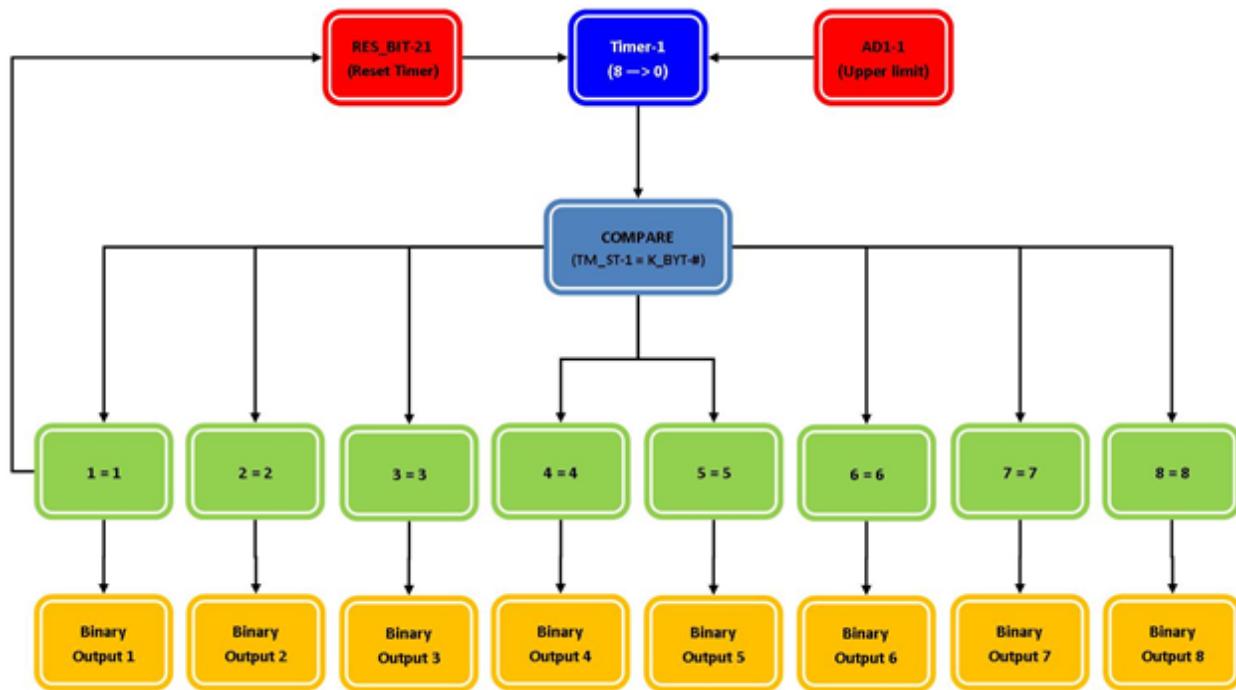
+-----<INSTR#-015>-----+
|   RES_BIT-5 [-----] -----+-----+
|                         ASSIGN OUTPUTS
|                         BO-5 ( R ) -----+
|-----+
+-----<INSTR#-016>-----+
|   RES_BIT-6 [-----] -----+-----+
|                         ASSIGN OUTPUTS
|                         BO-6 ( R ) -----+
|-----+
+-----<INSTR#-017>-----+
|   RES_BIT-7 [-----] -----+-----+
|                         ASSIGN OUTPUTS
|                         BO-7 ( R ) -----+
|-----+
+-----<INSTR#-018>-----+
|   RES_BIT-8 [-----] -----+-----+
|                         ASSIGN OUTPUTS
|                         BO-8 ( R ) -----+
|-----+
+-----<INSTR#-019>-----+
|                         END
+-----< OpenBAS >-----+
|-----+
[FIN]

```

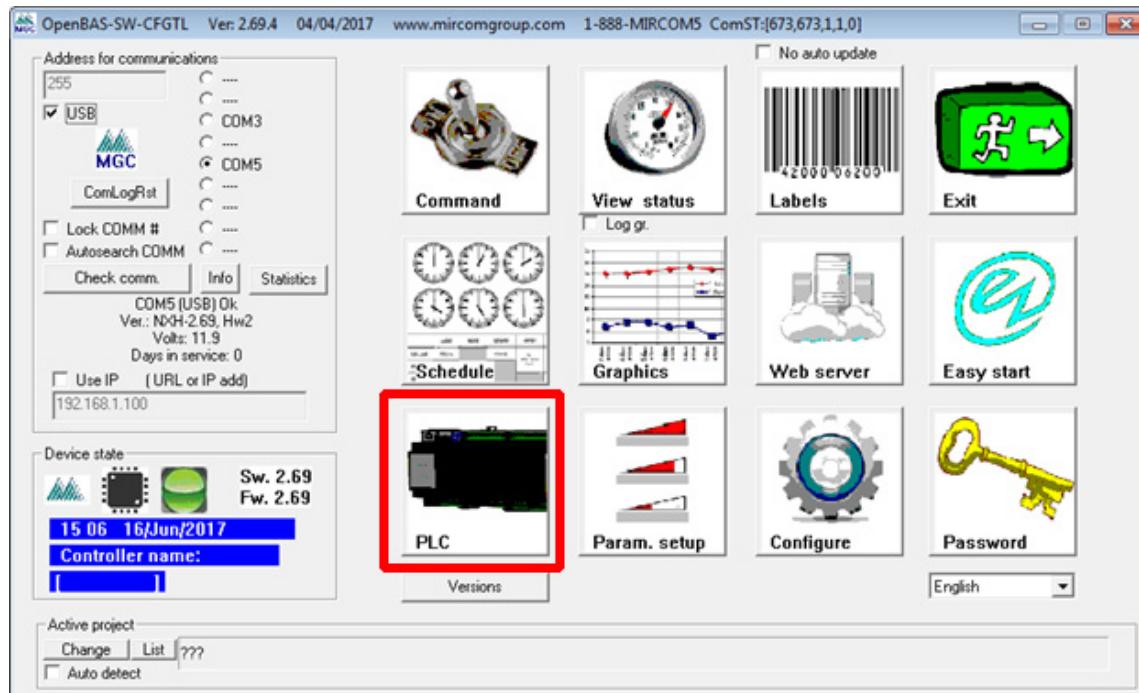
3.0 PLC Tutorial - Timer instruction

This tutorial will outline an example using the “Timer instruction” button in the PLC editor. Using the “Timer instruction” function, the program will provide a countdown from an inputted number and activate binary outputs accordingly. This counting down from 9 will loop infinitely as long as the program is running.

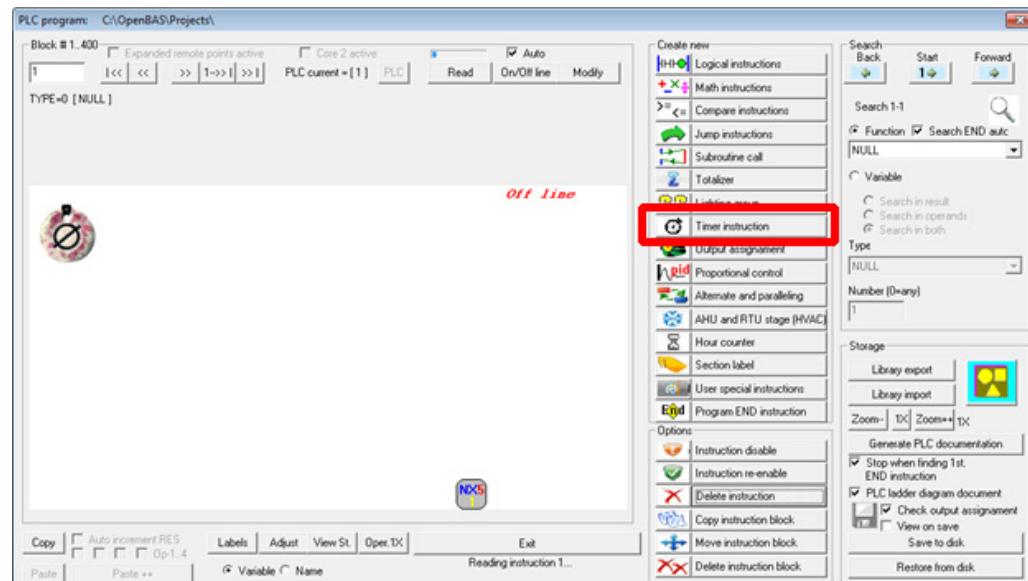
3.1 Block Diagram



1. Select the PLC icon from the main screen of the OpenBAS software and select "PLC 1" to get to the PLC editor.

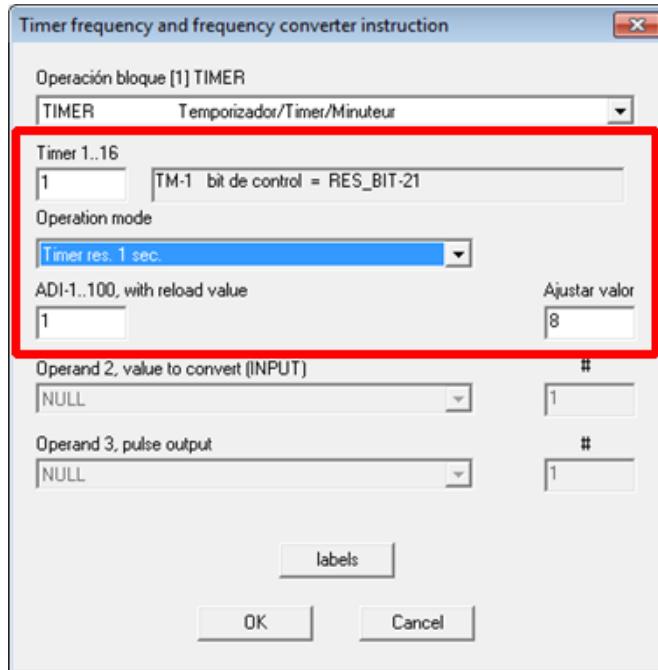


2. Clear any existing logic in the block and press the "Timer instructions" button to create the countdown timer.

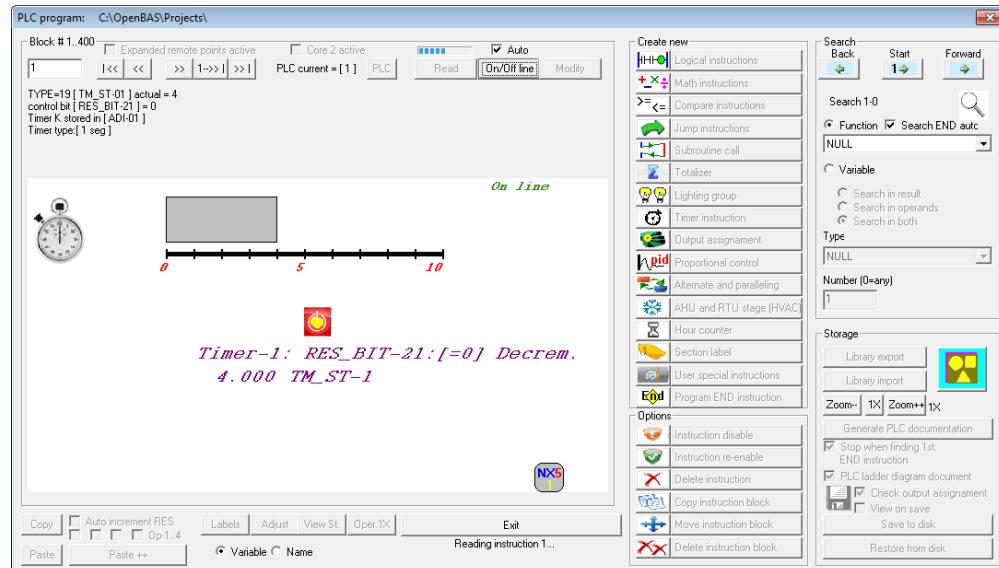


3. The Timer dialog menu's first part is the selecting the timer bit. The timer bit can be between 1 and 16 and the value stored there is the current state of the timer. The timer has a corresponding control bit which controls when the timer is running. Below that is the Operation mode which in this case is "Timer res. 1 sec" which acts as a countdown decrementing every 1 second. The last section is the value stored in EEPROM to which

the timer returns when activated by the control bit. In this case use 8, stored in the first ADI register.

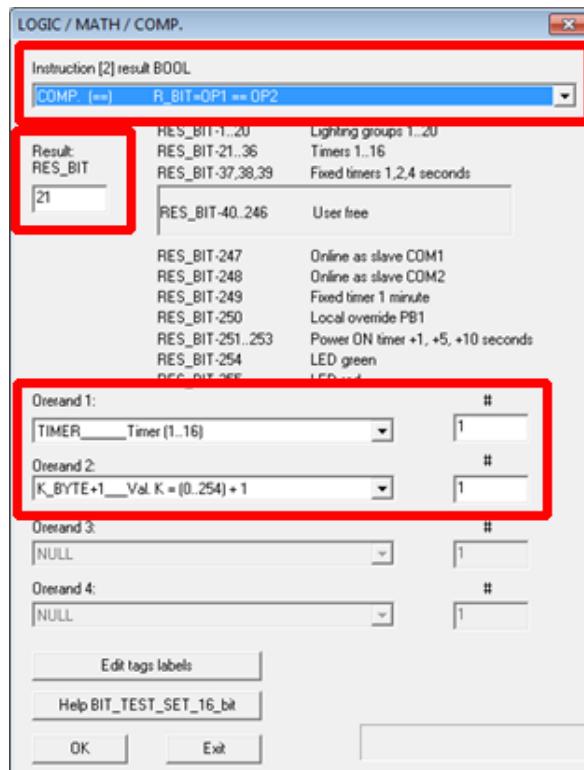


- As shown below the graphic shows the current value of the timer on a number line.

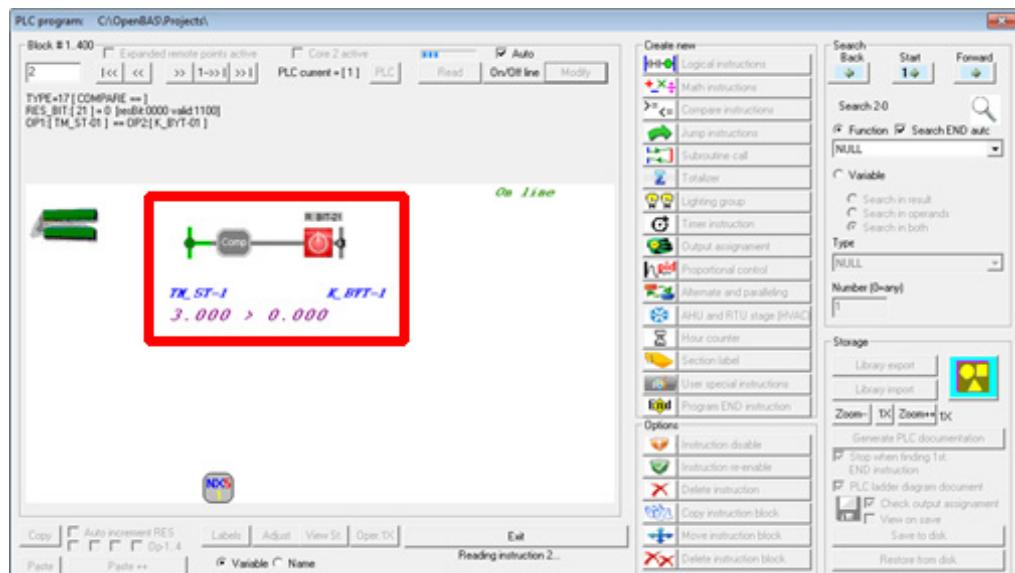


- Advance to the next PLC block and clear any existing logic. Press the “Compare instructions” button to bring up the below menu. In the top dropdown menu select the “==” comparison. For operand 1, select the timer 1 output and compare that to operand 2

set as the K_BYT value 1. For the result bit, select register bit 21 which is the control bit to the timer.

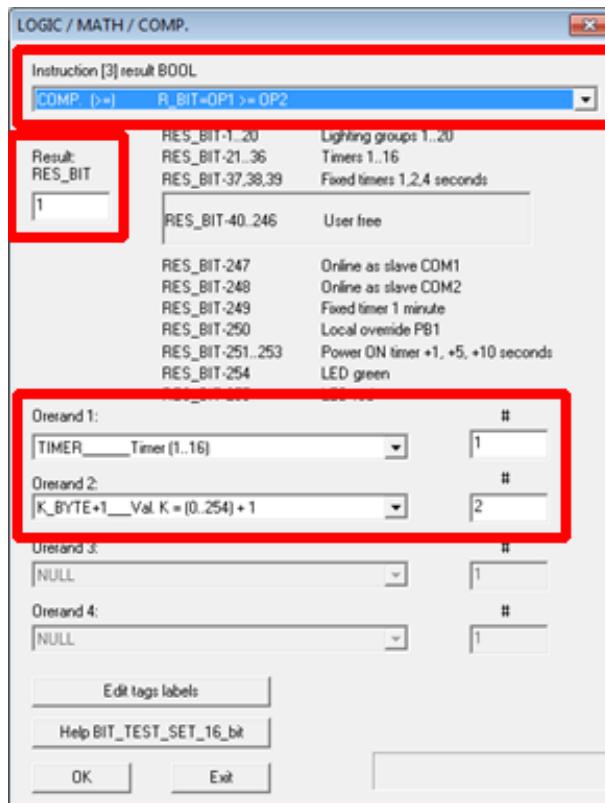


6. This means that the control bit will be turn on when the timer's current state is equal to zero. Therefore the control bit will be activated when the timer counts all the way down. When the control bit is active the timer is set to its upper limit value. Thus the timer will loop infinitely.

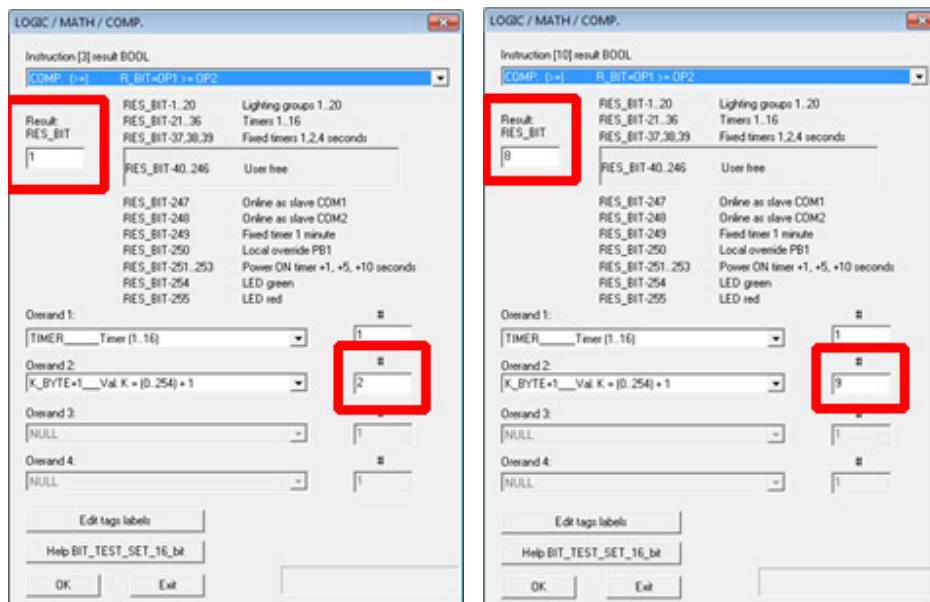


7. Advance to the next PLC block and clear any existing logic. Press the "Compare instructions" button to bring up to menu below. The goal of this comparison is to be

active any time the timer state is greater than or equal to the given value which in this case is 1. The result of this comparison is to be stored in the first register bit.

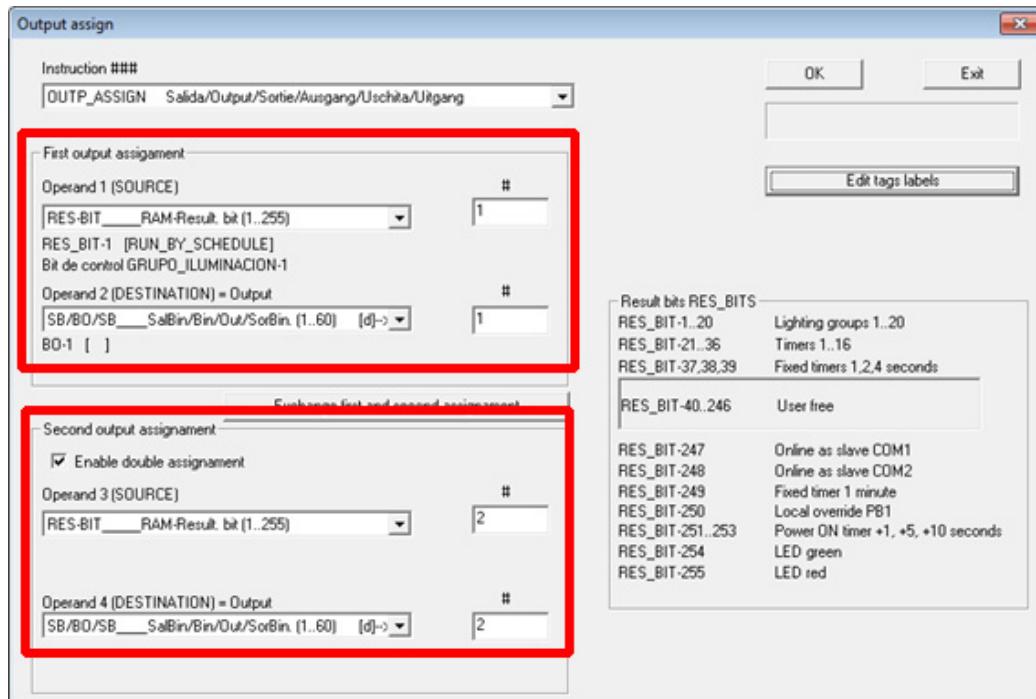


8. Repeat the previous step 7 more times to create a comparison for values 1 to 8. Each time incrementing the output register bit and the K_BYT value.

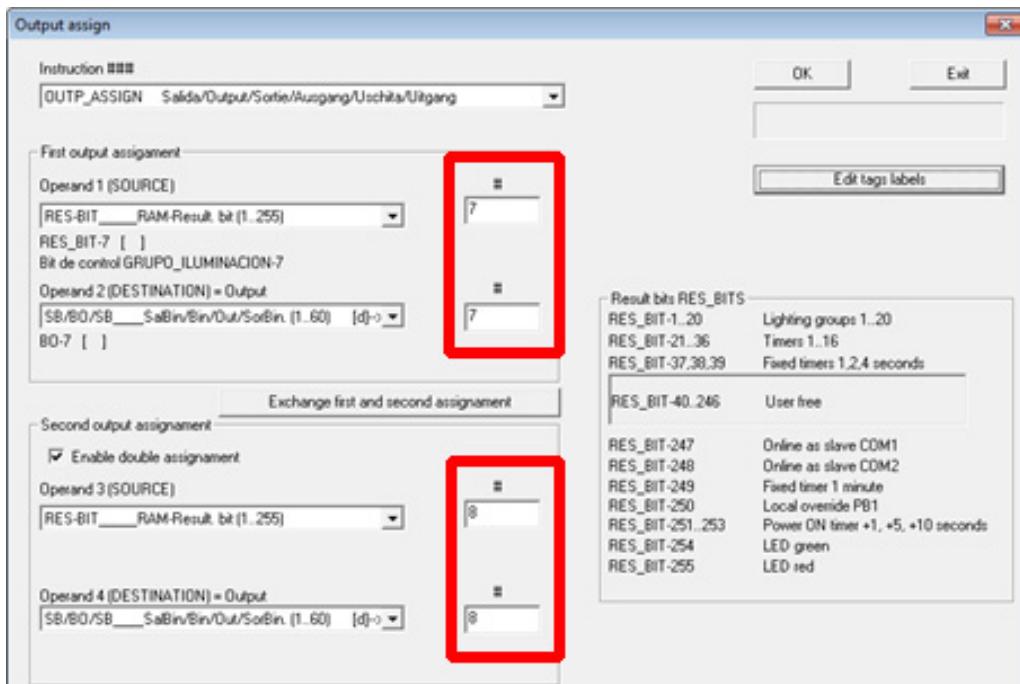


9. Advance to the next PLC block and clear any existing logic. Press the “Output assignment” button to assign the binary outputs. In this case enable double assignment

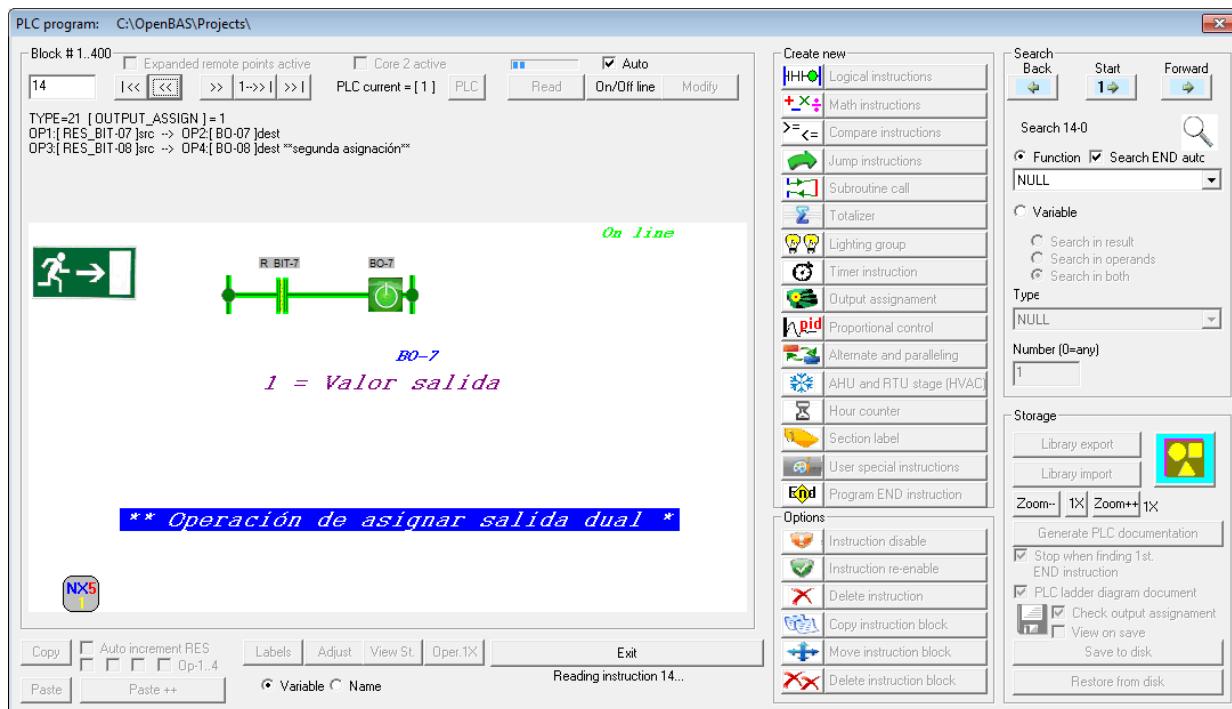
to cut in half the number for PLC blocks needed. As shown below, the source for each output is the corresponding register bit.



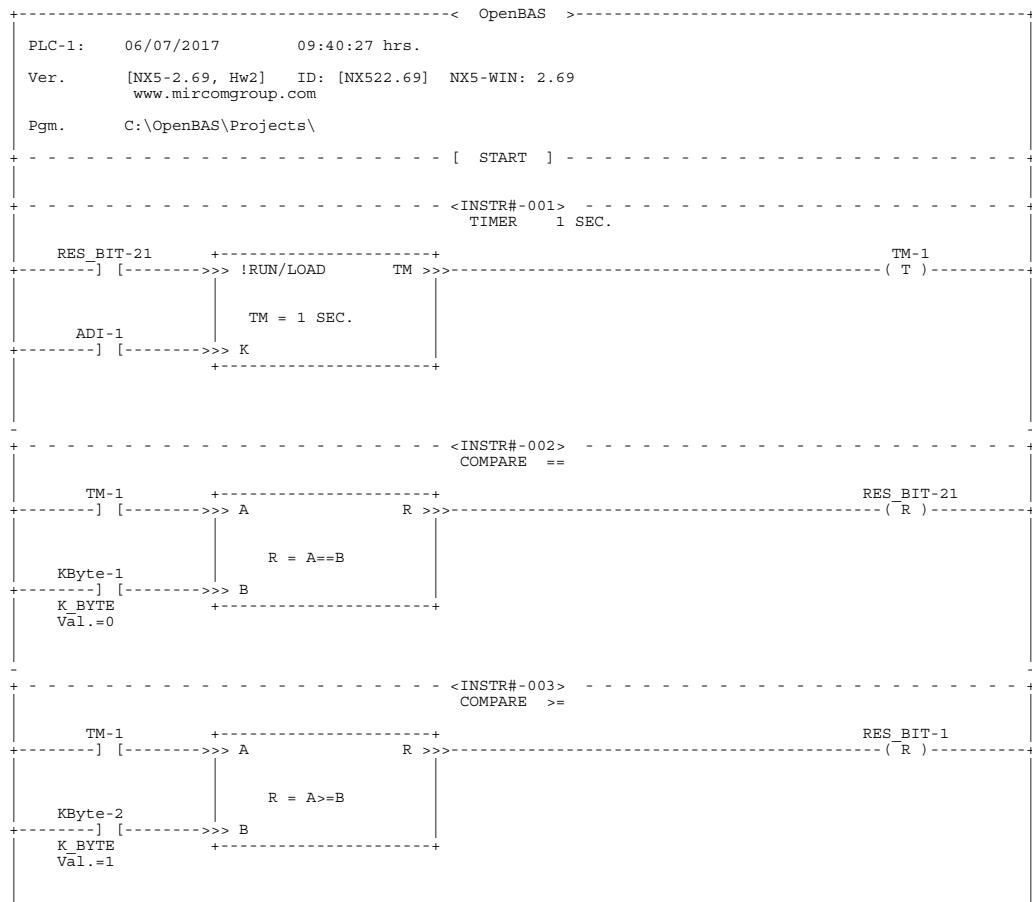
10. Repeat the previous step 3 more time to assign outputs 3-8. Each time incrementing the register bit and outputs.

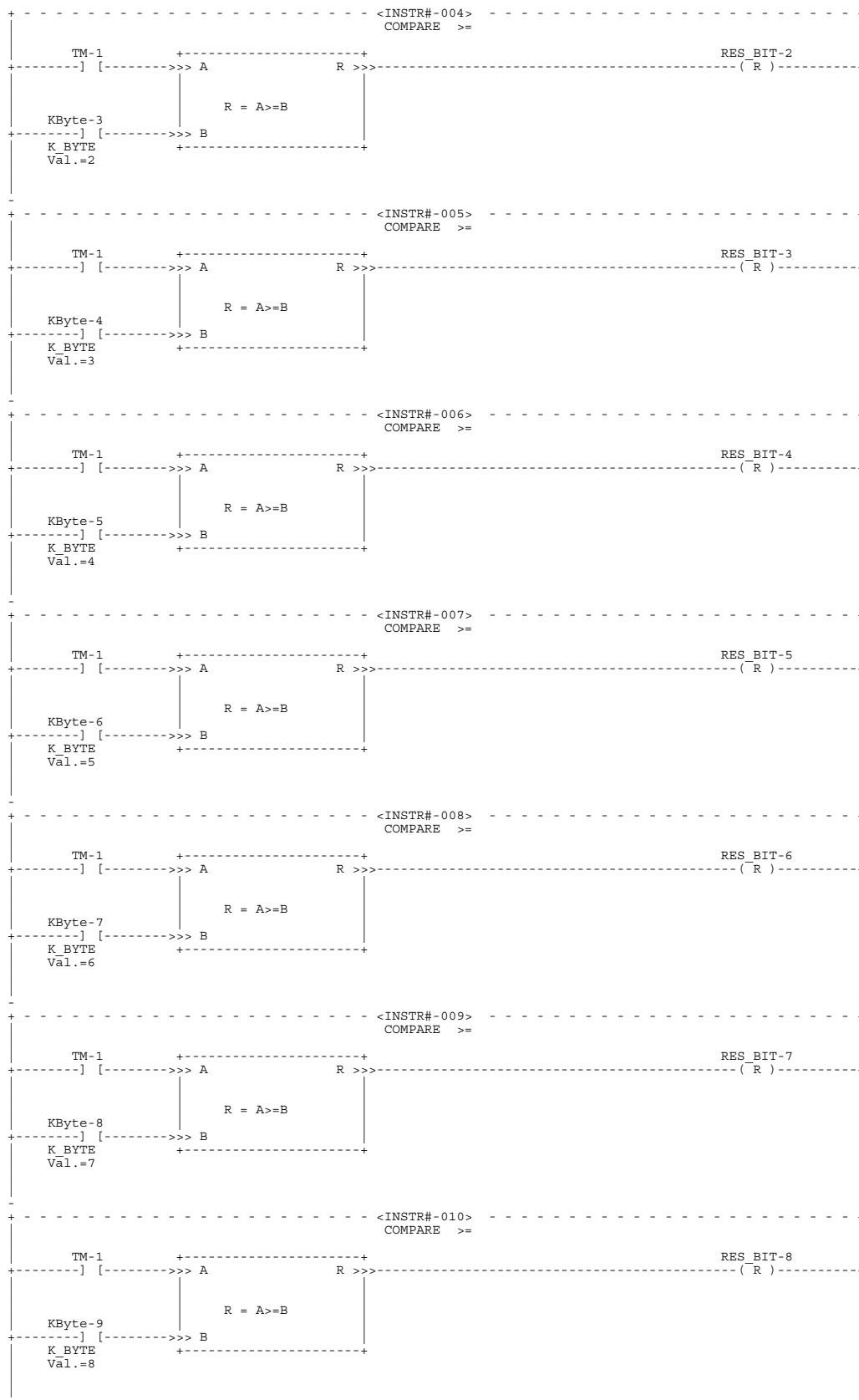


11. The program is now complete.



3.2 PLC Ladder Diagram





```

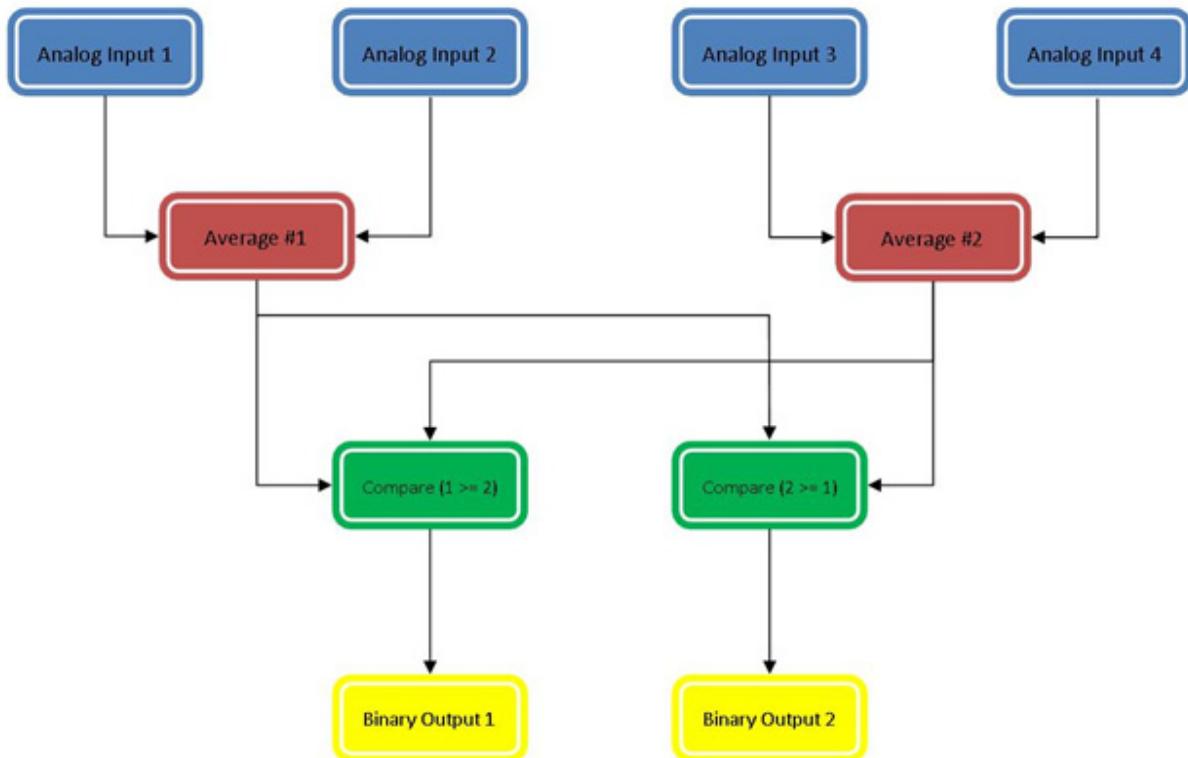
+-----<INSTR#-011>-----+
| ASSIGN OUTPUTS (DUAL)
|
RES_BIT-1 [-----] ( # 2 ) BO-1 ( R )
|
RES_BIT-2 [-----] ( # 2 ) BO-2 ( R )
HEATING_V ALVE
|
+-----<INSTR#-012>-----+
| ASSIGN OUTPUTS (DUAL)
|
RES_BIT-3 [-----] ( # 2 ) BO-3 ( R )
|
RES_BIT-4 [-----] ( # 2 ) BO-4 ( R )
|
+-----<INSTR#-013>-----+
| ASSIGN OUTPUTS (DUAL)
|
RES_BIT-5 [-----] ( # 2 ) BO-5 ( R )
|
RES_BIT-6 [-----] ( # 2 ) BO-6 ( R )
|
+-----<INSTR#-014>-----+
| ASSIGN OUTPUTS (DUAL)
|
RES_BIT-7 [-----] ( # 2 ) BO-7 ( R )
|
RES_BIT-8 [-----] ( # 2 ) BO-8 ( R )
|
+-----<INSTR#-015>-----+
| END
< OpenBAS >
-
```

[FIN]

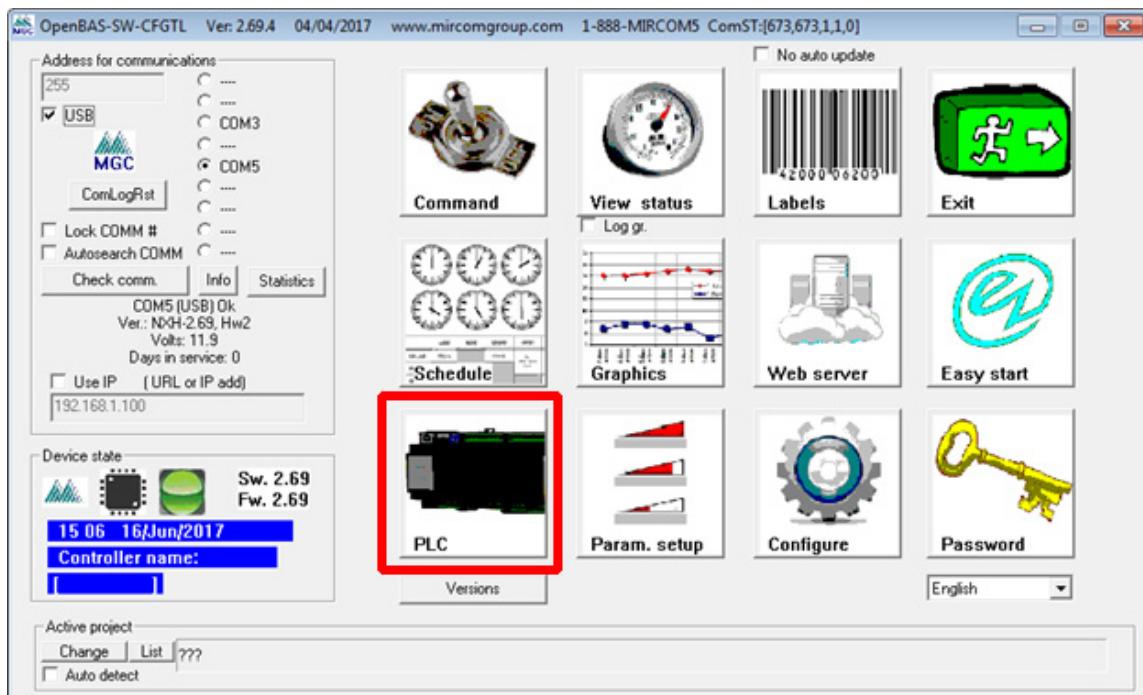
4.0 Example: PLC Tutorial (Average, Compare)

This tutorial will outline an example using PLC logic controlled “Average” and “Compare” functions to use analog inputs to interact with binary outputs. The program will average two analog inputs and compare that to the average of another to analog inputs to see which output to turn on.

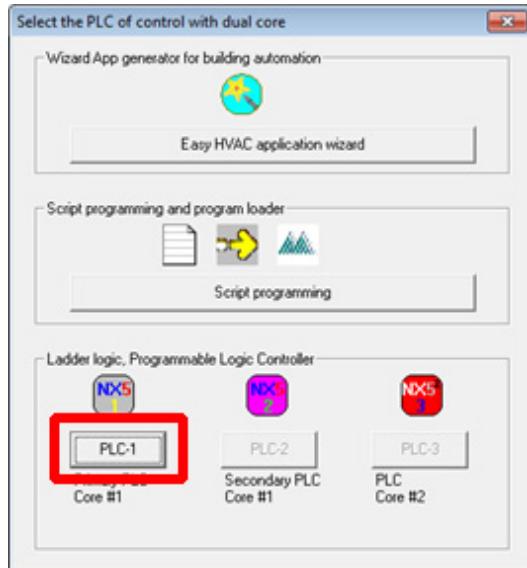
4.1 Block Diagram



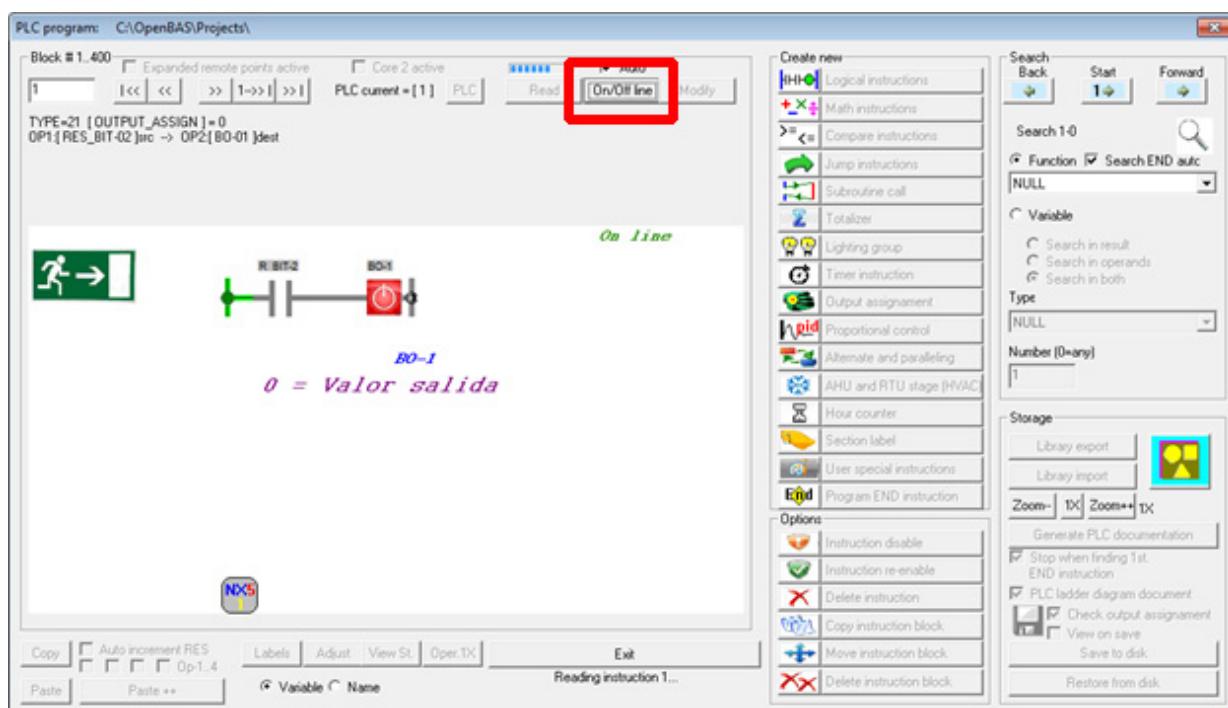
1. Select the PLC icon from the main screen of the OpenBAS software.



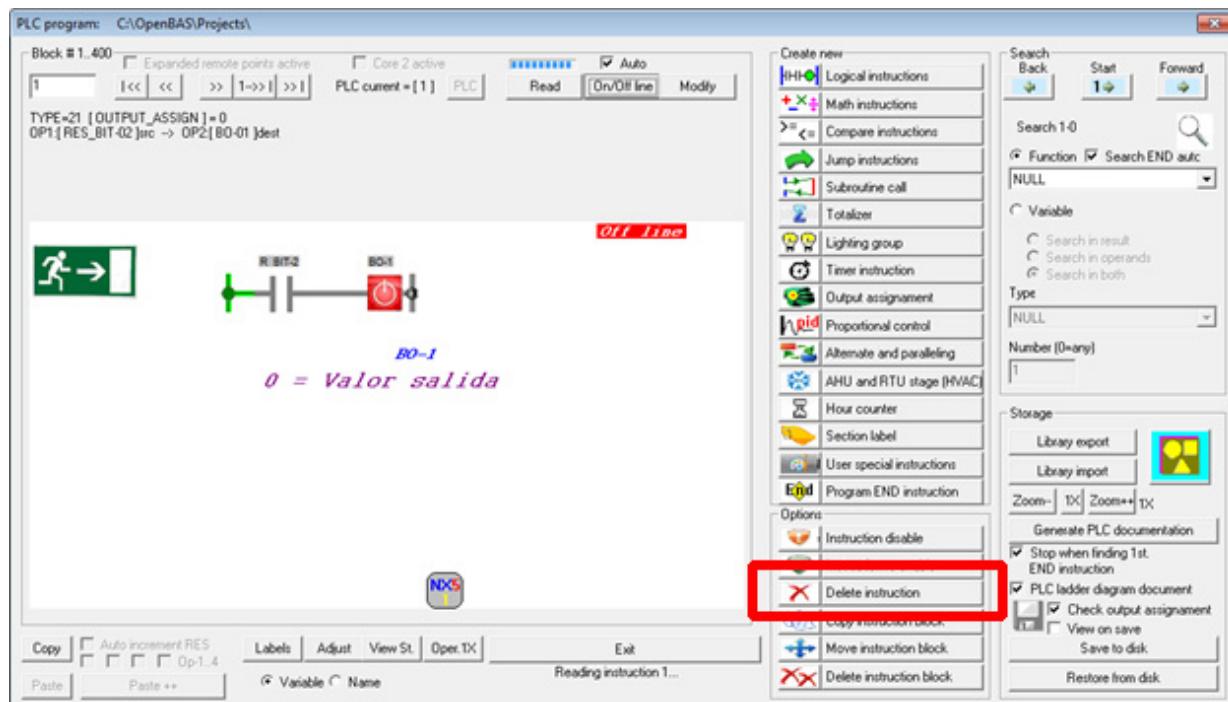
2. Select "PLC 1" as it is the only option. PLC's 2 and 3 are enabled if a dual core is installed.



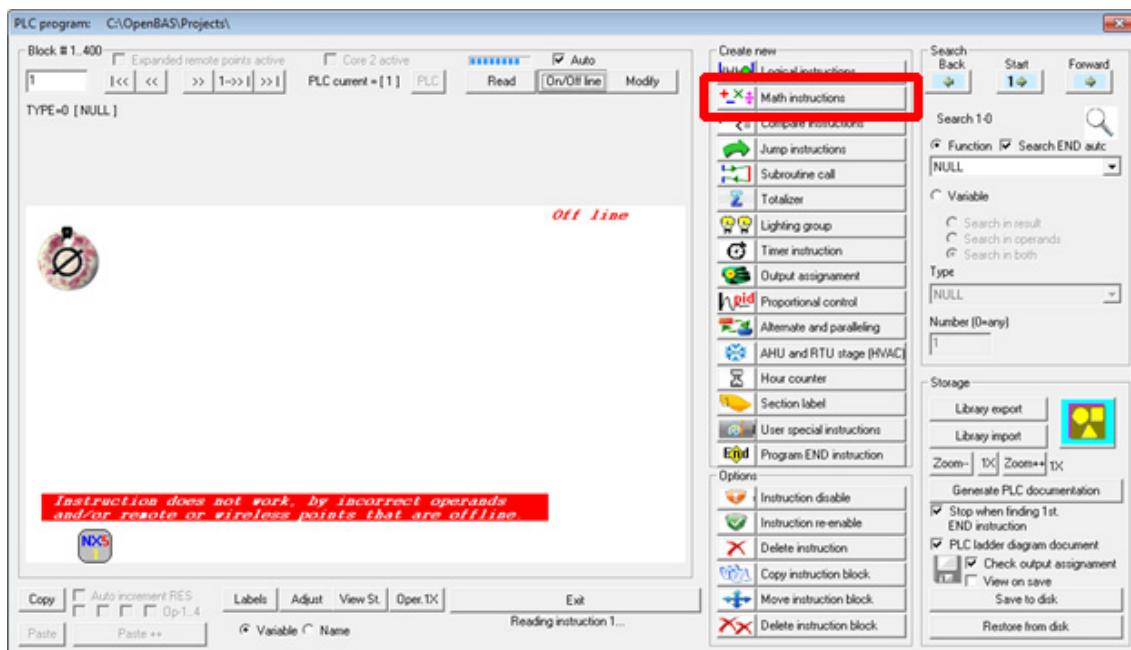
3. Press the “On/Off line” button to enable editing the logic.



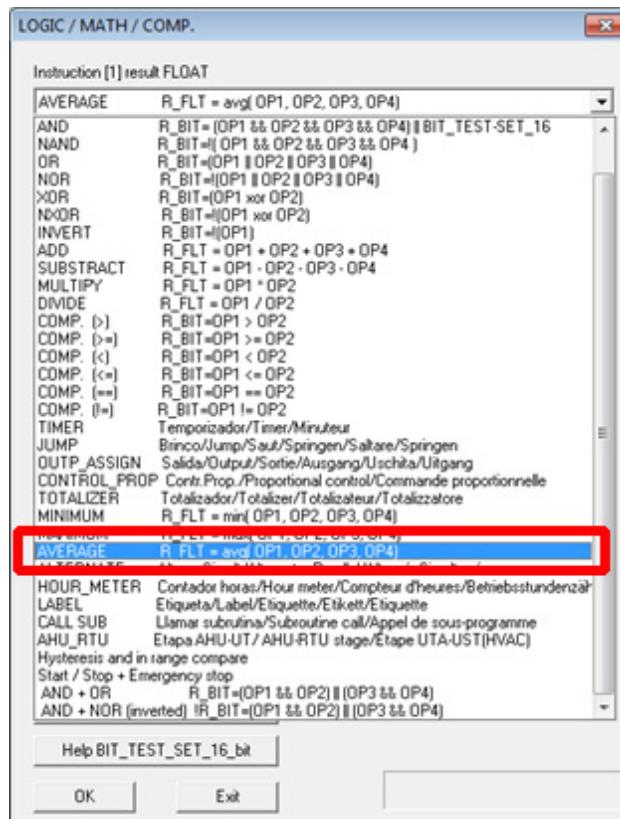
4. The functions on the right side are no longer greyed out and are usable. To begin programming, select the “Delete instruction” button to clear any outstanding logic in this block.



5. Select the “Math instructions” button to create the first average function.

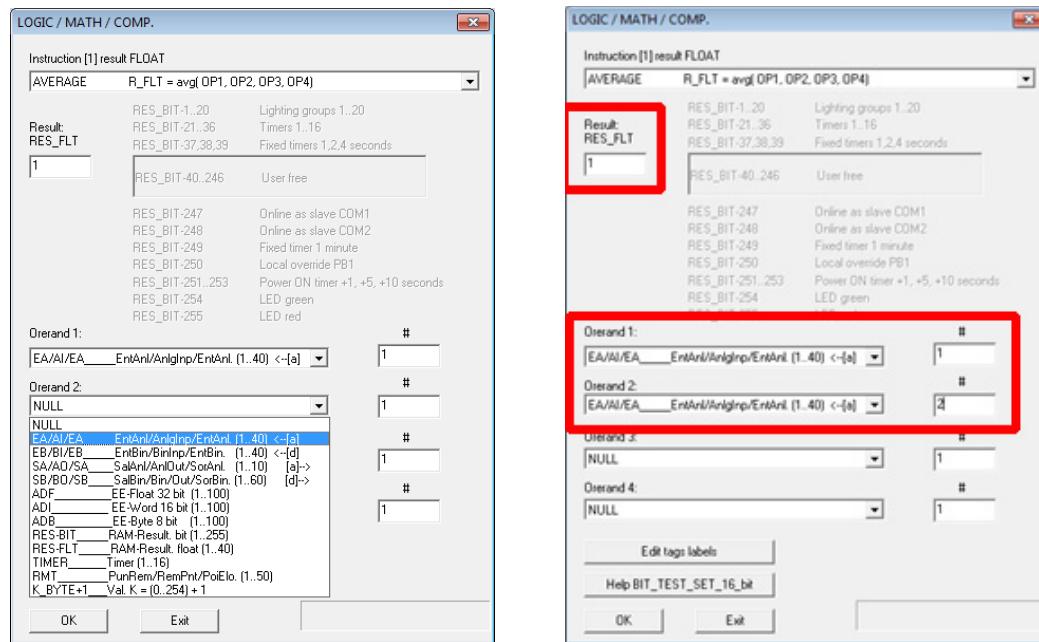


6. In the instruction drop down menu, select the average function.

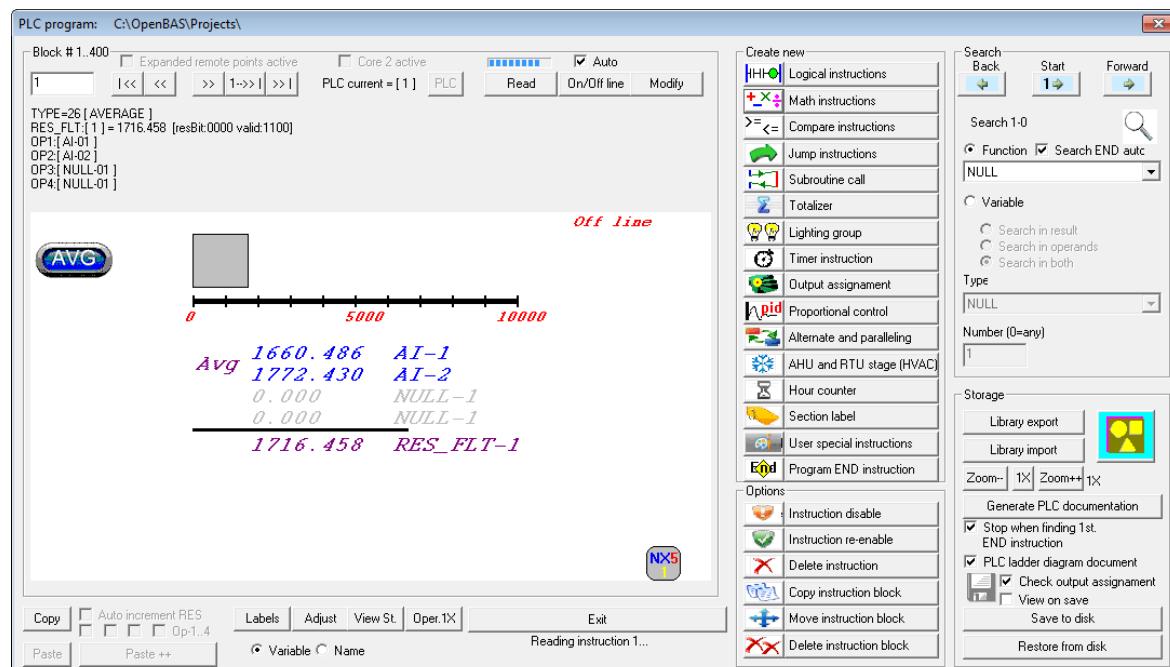


7. In the “AVERAGE” function dialog box there are two parts. The first part is the result, which is stored in a float register opposed to a bit register. The second part is the four

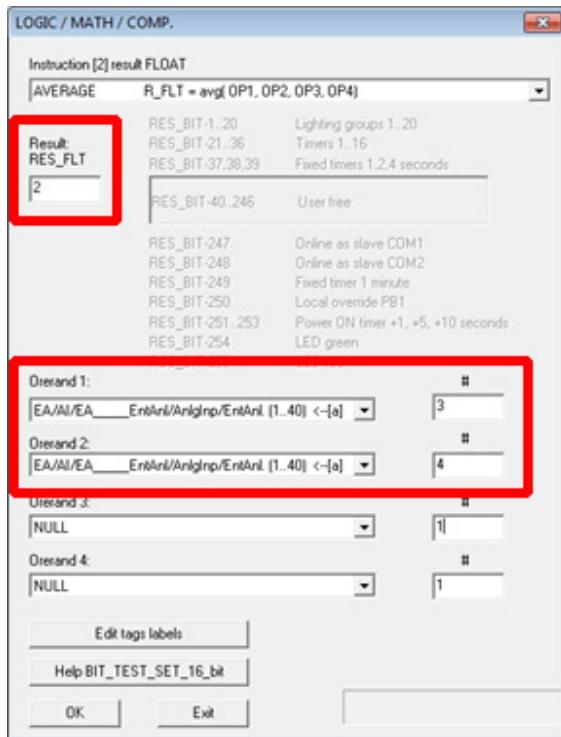
available operands for the average. In this case, select the analog input option for both operand 1 and 2. Select numbers 1 and 2 for the analog input numbers.



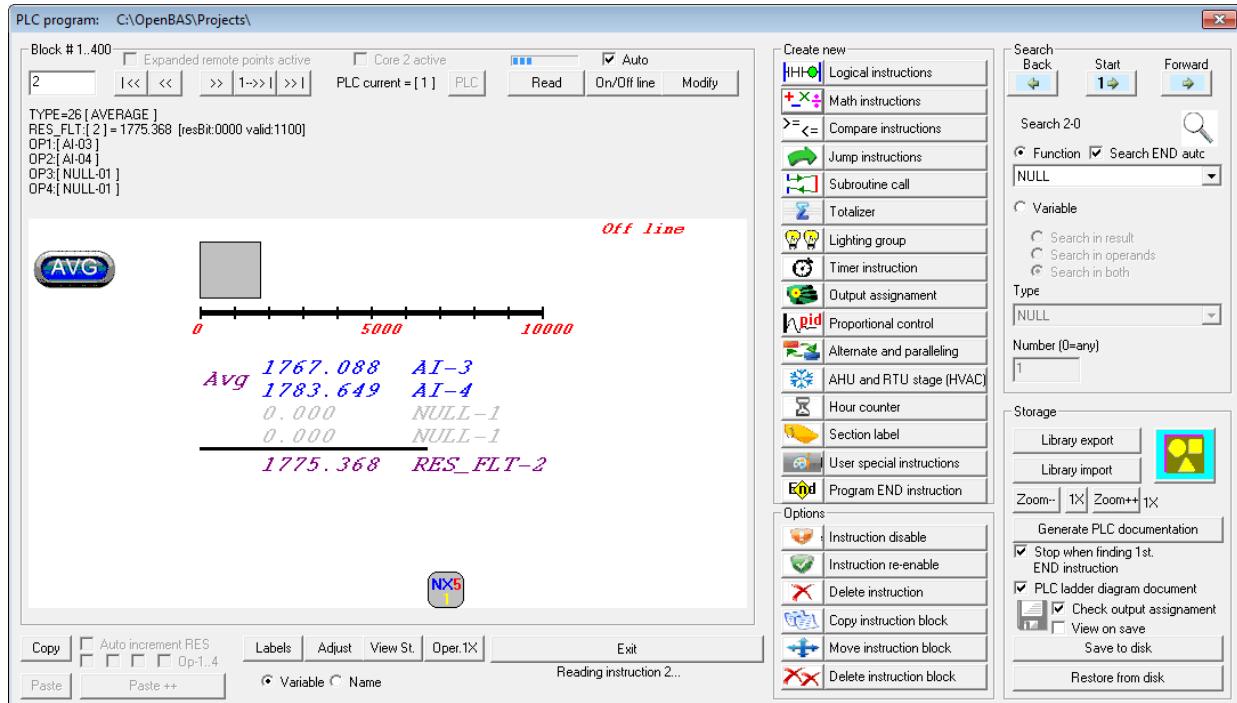
8. The result on the screen shows both the analog input values and the average stored in the float as well as a graphic showing the value on a number line.



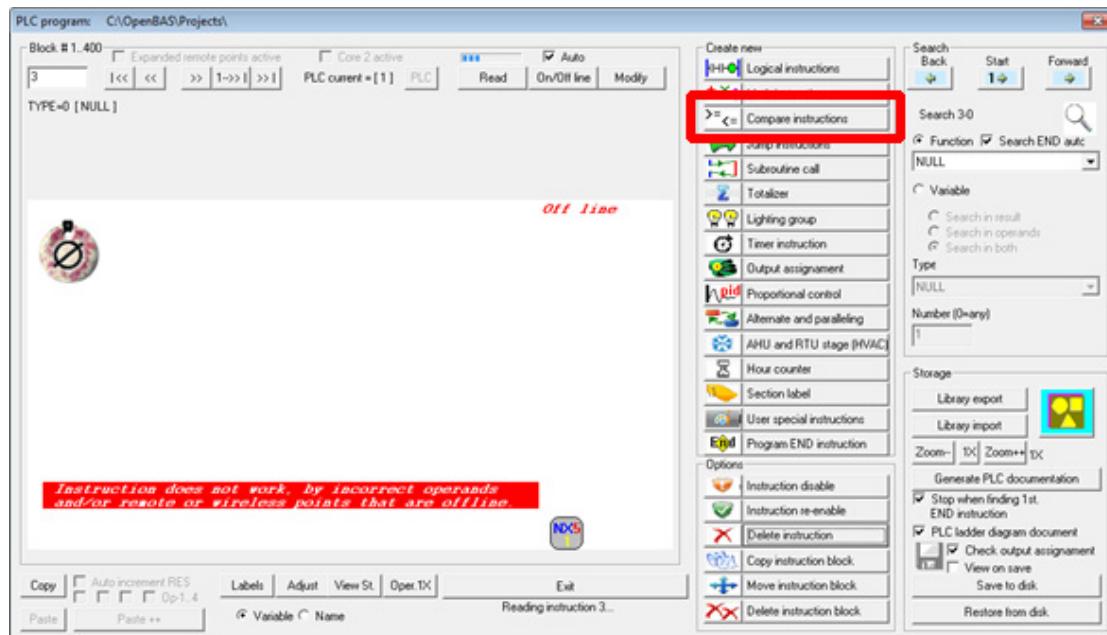
9. Press the “>>” button to advance to the next PLC block and repeat steps 4 -7 to create the second “AVERAGE” function with the values as shown.



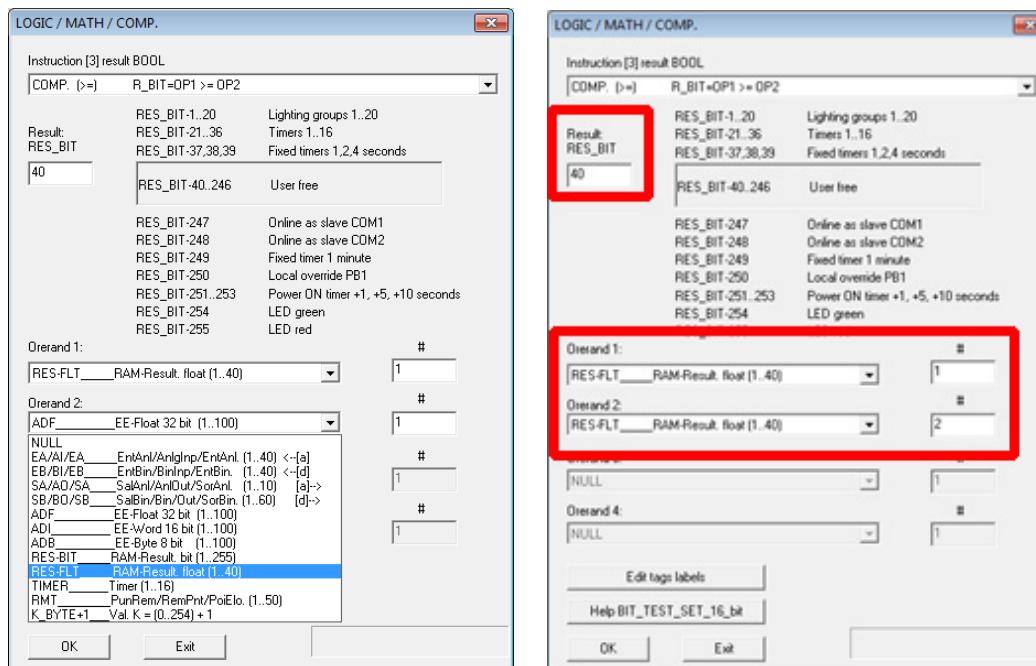
10. The result should be as shown below.



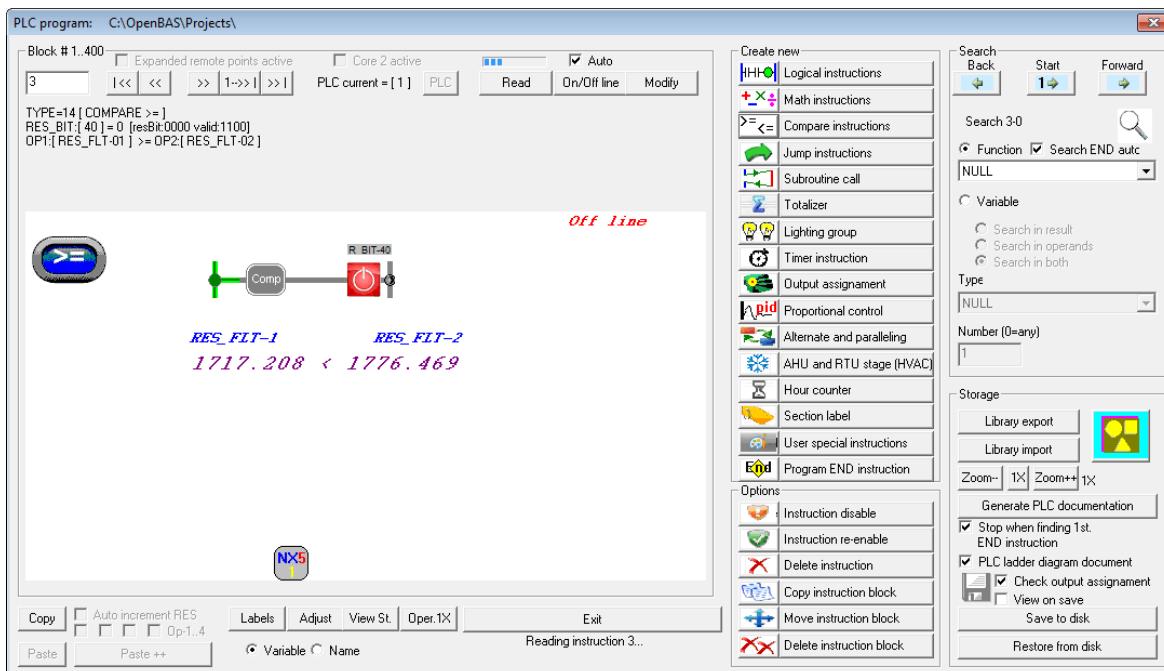
11. Press the “>>” button to advance the next PLC block and use the “Delete instruction” to clear any existing logic. This clears the way to press the “Compare instructions” button to set up the comparison function.



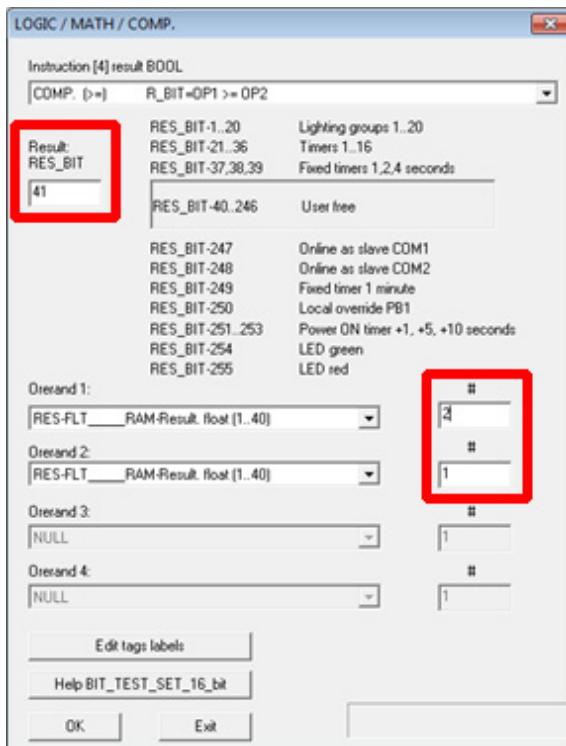
12. The dialog box that appears has three sections. The instruction dropdown box is already set to comparison. The result section allows the section a register bit to store the output. In this case use a free bit (40-246). The operands 1and 2 will be the output of the previous “AVERAGE” functions. Set the operands to RES-FLT with the numbers 1 and two as shown.



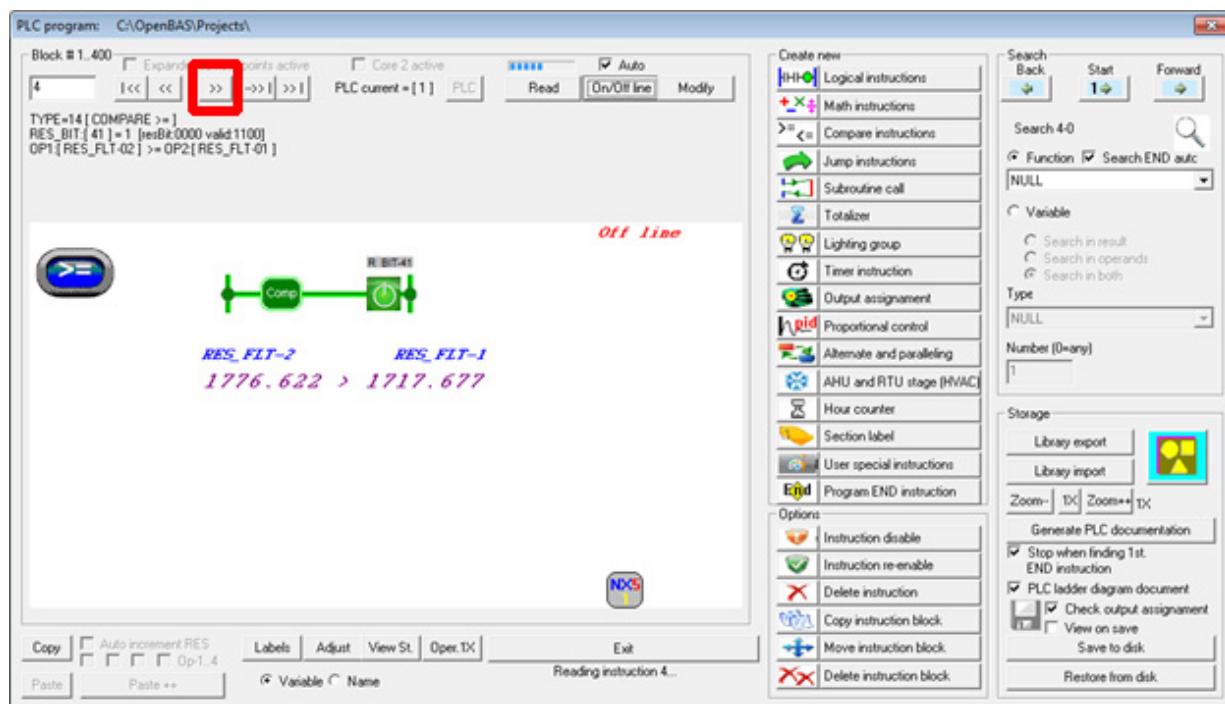
13. The result should be as shown below. The register bit 40 will be activated when average of inputs 1 and 2 is greater than the average of inputs 3 and 4.



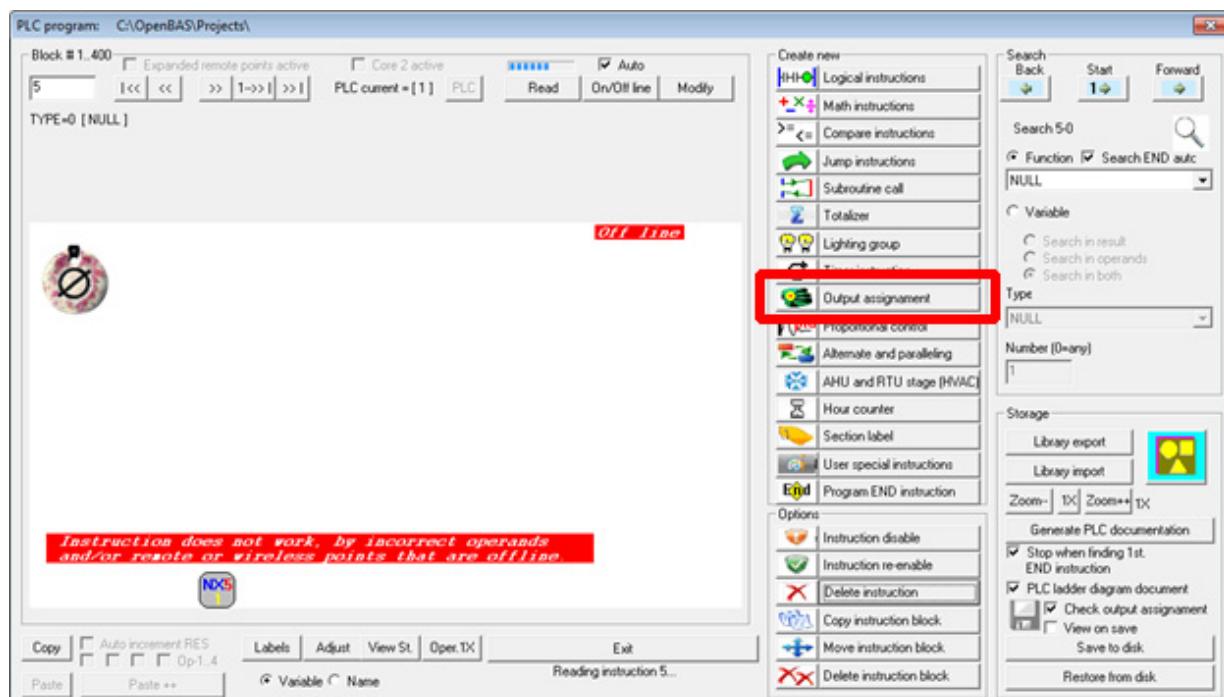
14. Repeat steps 11 and 12 to create a comparison that will be active when inputs 3 and 4 is greater than 1 and 2. As shown below, the only difference is having register float 2 comes before register float 1 and a new output register bit is selected.



15. The result should be similar to as shown below. The comparison is effectively the inverse of the previous comparison. Press the “>>” button to advance to the next PLC block.

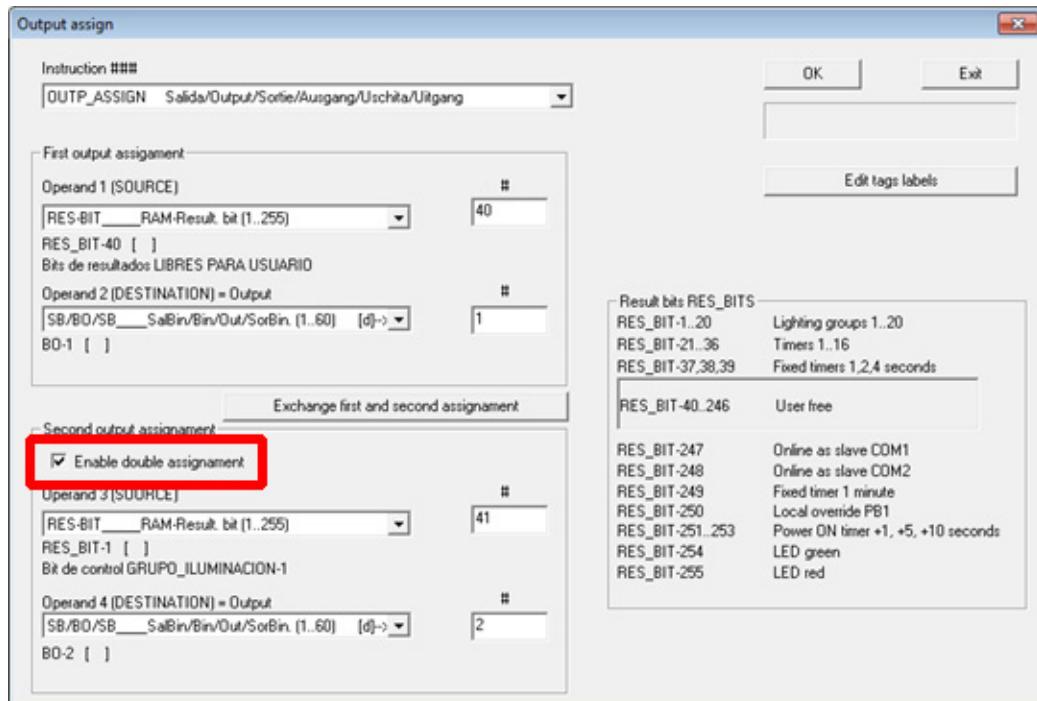


16. Press the “Delete instruction to clear any existing logic and press the “Output assignment” button to control our outputs.

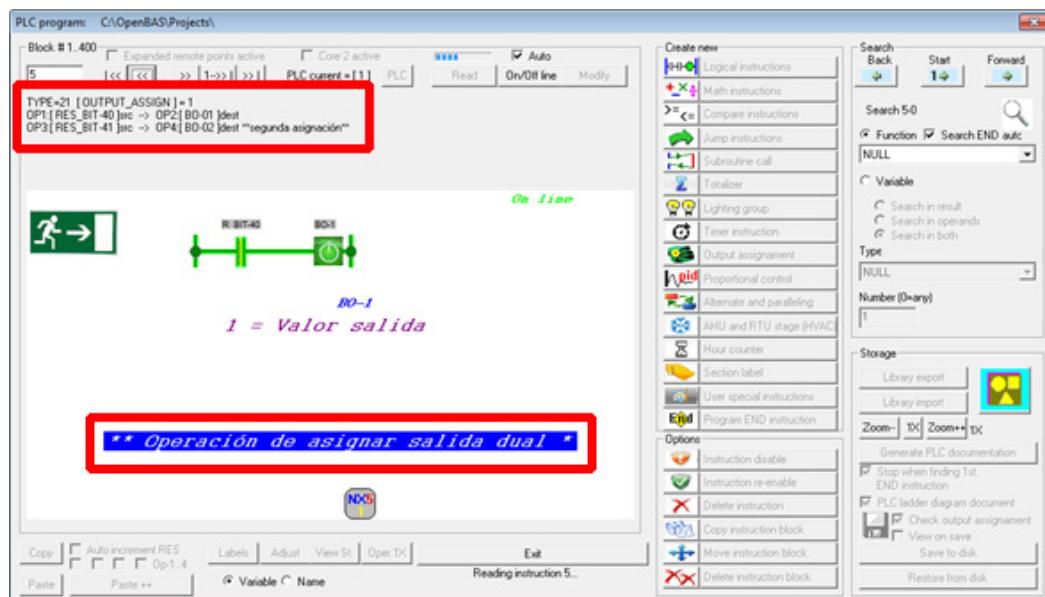


17. In the dialog box a source and output must be selected. In this case select register bit 40 for the source which was the output for the first comparison. Select binary output 1 for the output. Next select the “Enable double assignment” check box to allow assigning

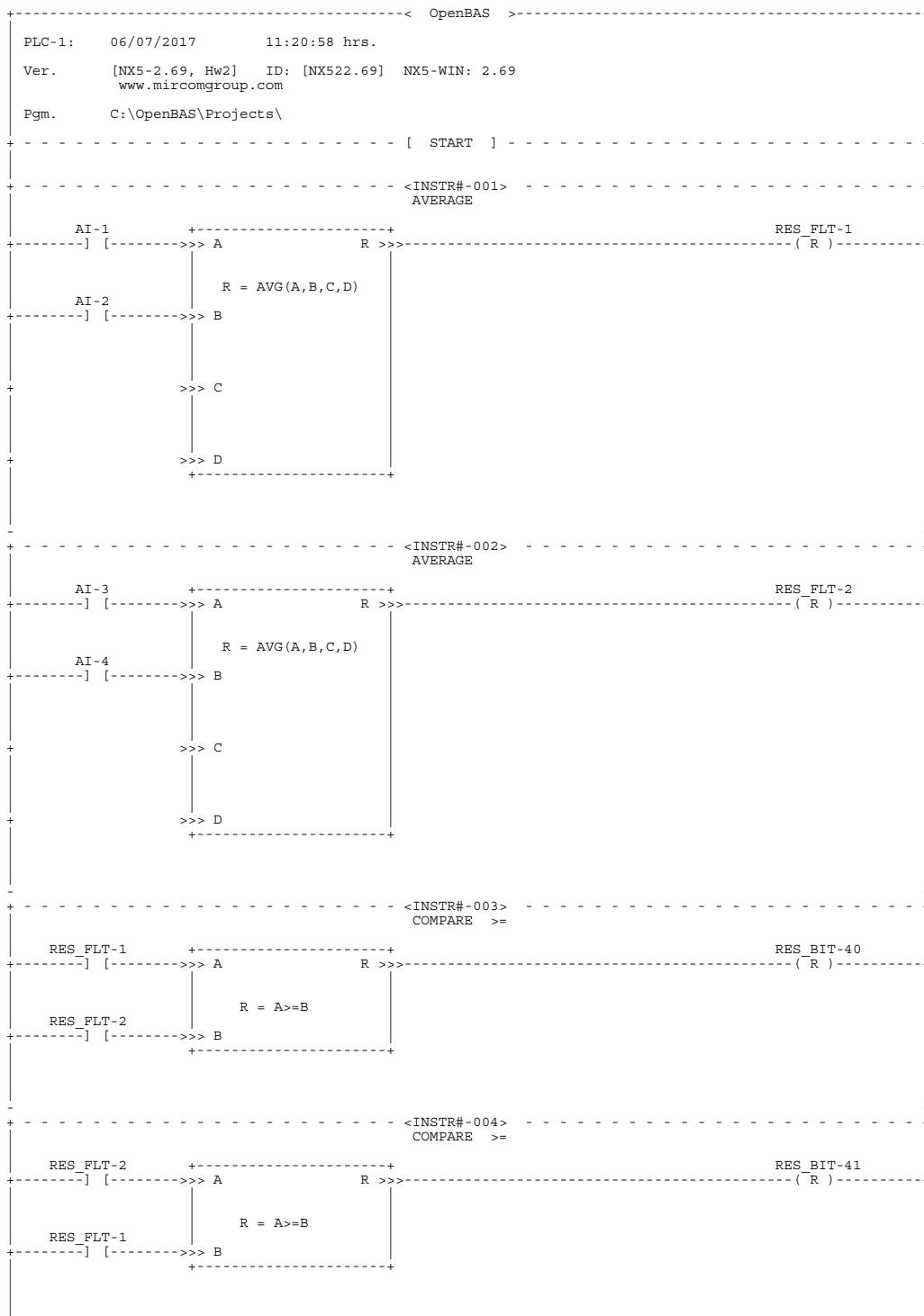
another output. The operand and source are the same just with different numbers as shown.



18. While only one output is shown, both are active as shown in the highlighted text below. The program is now complete. Binary output 1 will be active if inputs 1 and 2 average to be greater than the average of 3 and 4. Output 2 will be on if the opposite is true.



4.2 PLC Ladder Diagram

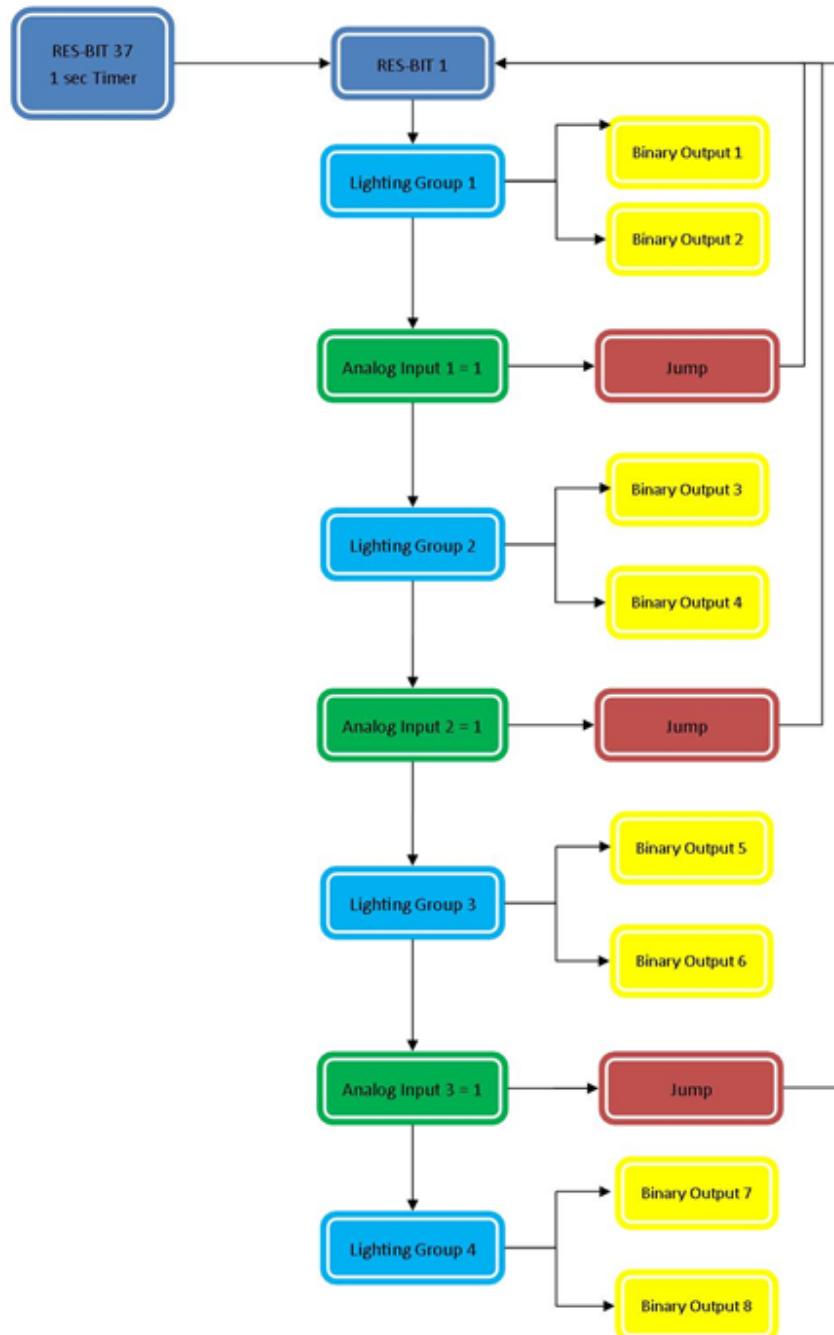


```
+-----<INSTR#-005>-----+
| ASSIGN OUTPUTS (DUAL)
+-----[-----RES_BIT-40-----]-----+
|                               BO-1
|                               ( R )
+-----[-----RES_BIT-41-----]-----+
|                               ( # 2      )
|                               )           BO-2
|                               ( R )
+-----<INSTR#-006>-----+
| END
| OpenBAS >
+-----+
[FIN]
```

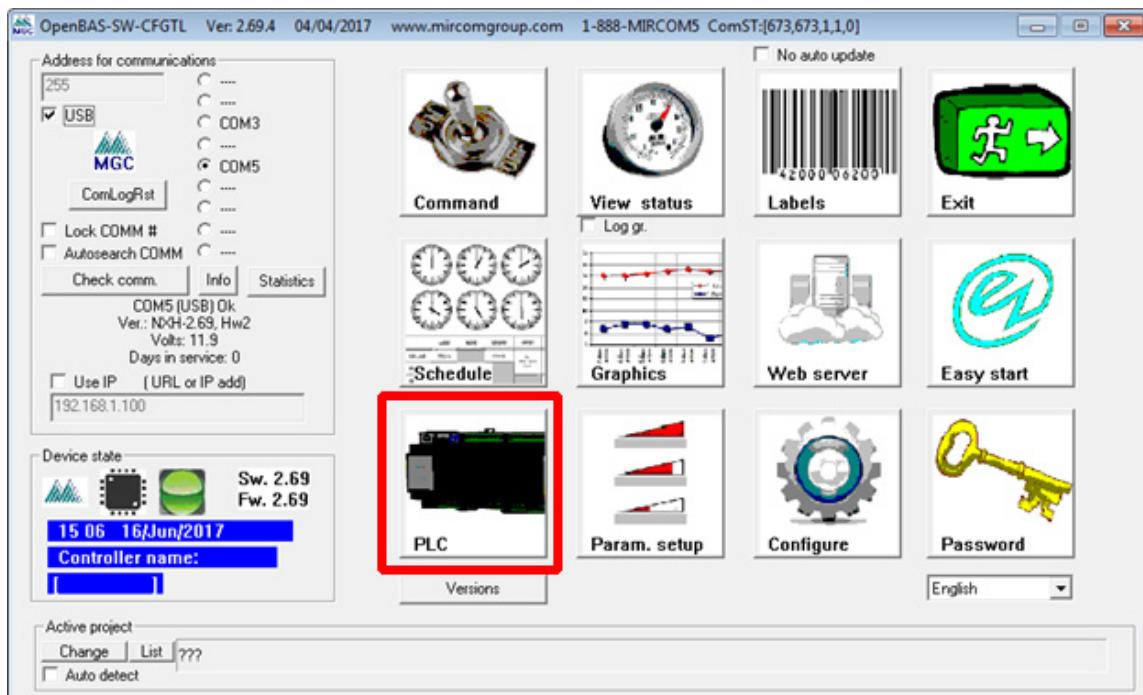
5.0 Example PLC Tutorial (Lighting Groups, Jump)

This tutorial will outline an example using PLC logic controlled “Lighting Groups” and “Jump” functions. The program will use lighting groups to control groups of outputs together and Jump functions to skip ahead in the program.

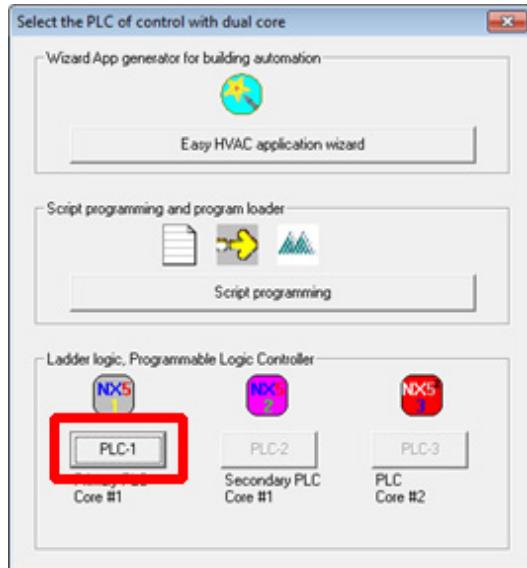
5.1 PLC Block Diagram



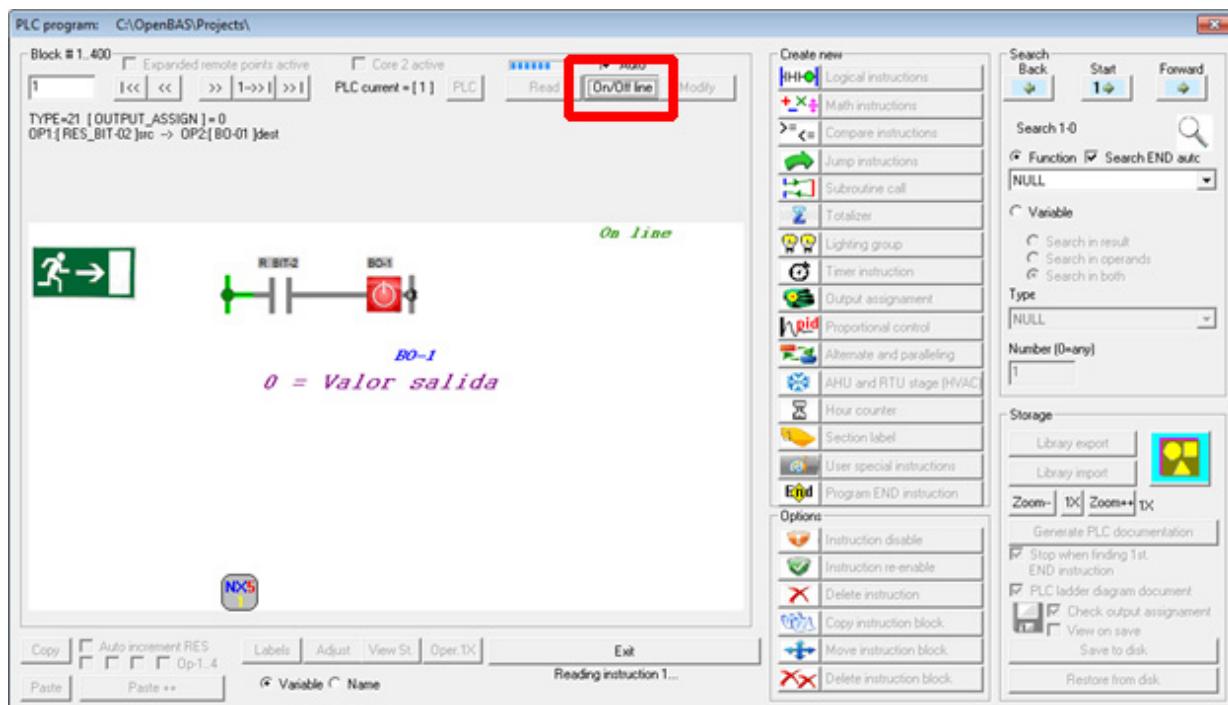
1. Select the PLC icon from the main screen of the OpenBAS software.



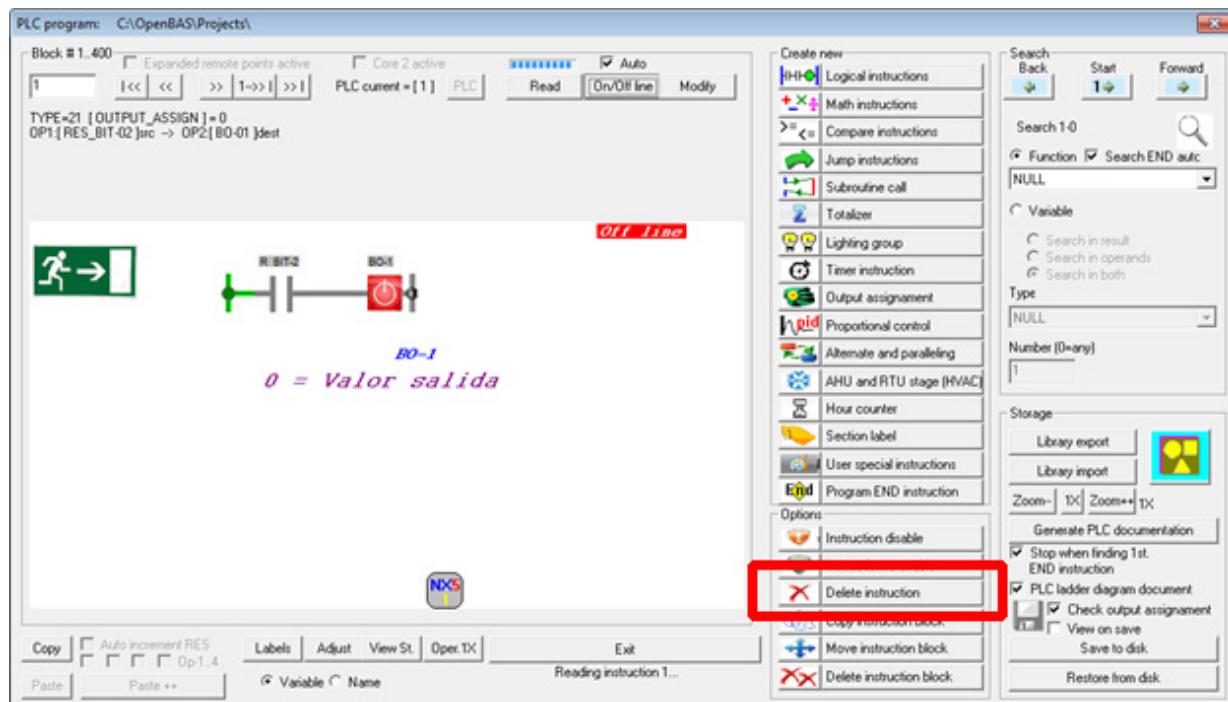
2. Select "PLC 1" as it is the only option. PLC's 2 and 3 are enabled if a dual core is installed.



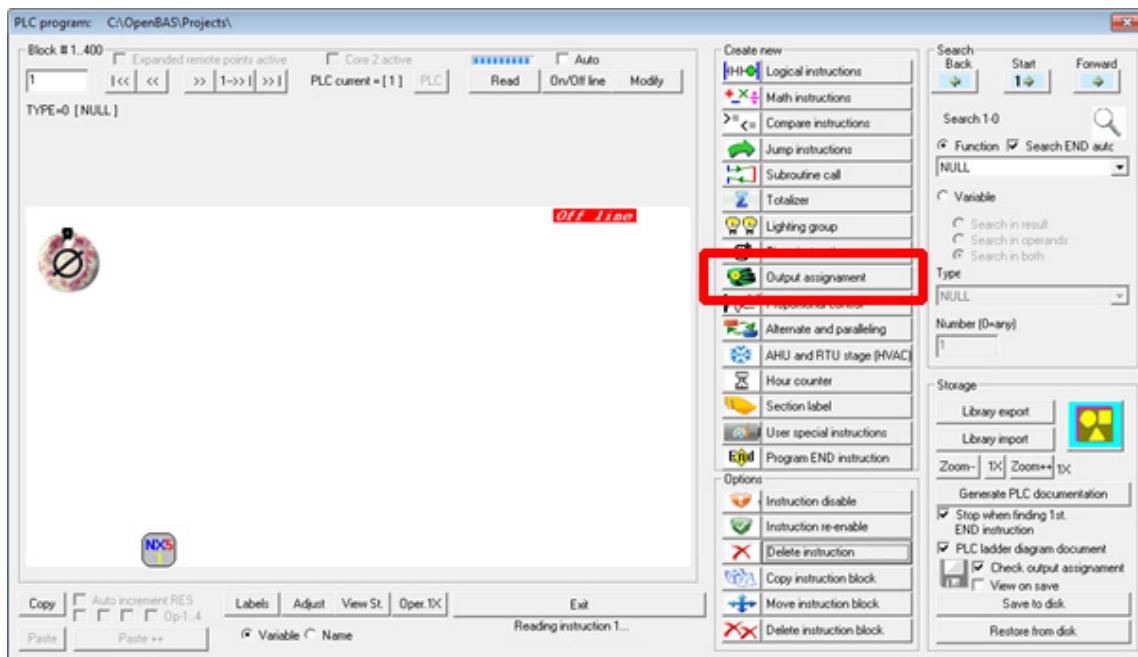
3. Press the “On/Off line” button to enable editing the logic.



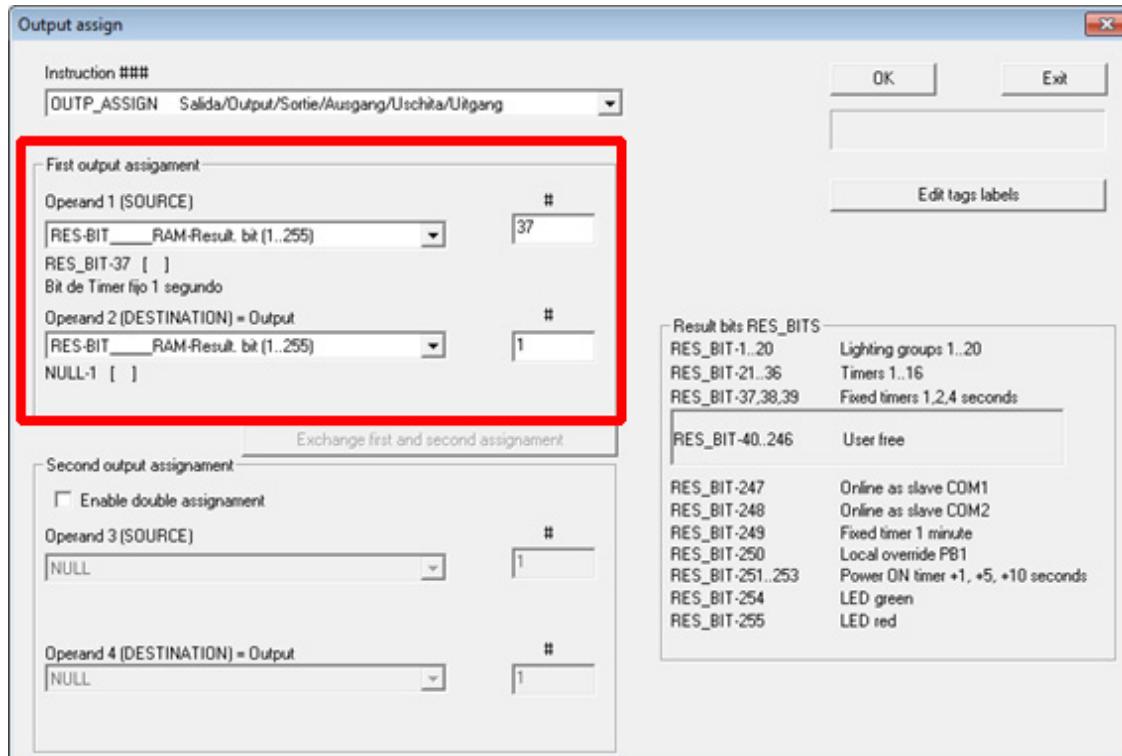
4. The functions on the right side are no longer greyed out and are usable. To begin programming, select the “Delete instruction” button to clear any outstanding logic in this block.



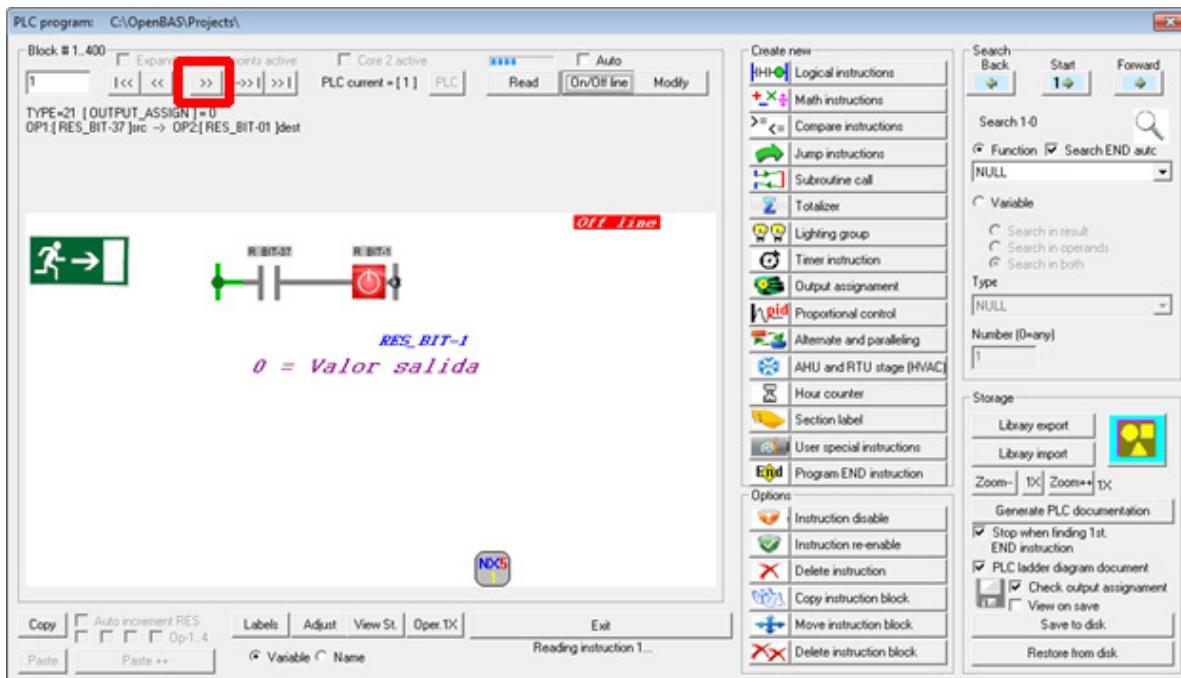
5. Press the “Output assign” to set up the timer for the program.



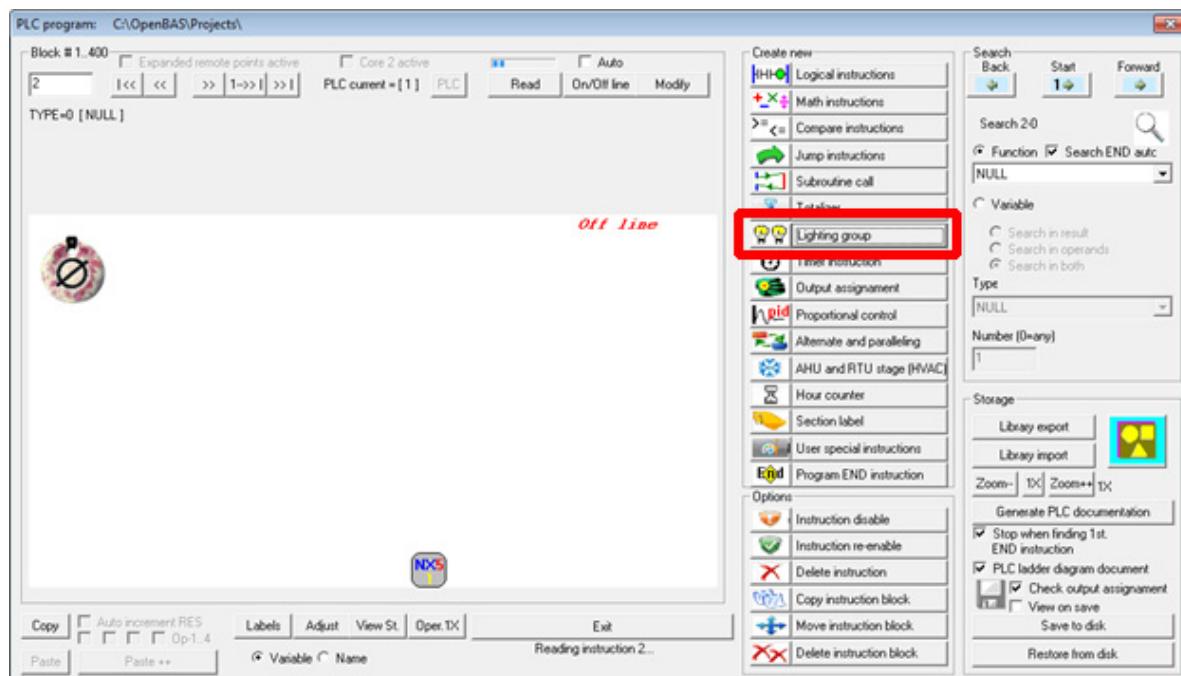
6. For the source, select register bit 37 which is the 1 second fixed timer. For the output, select register bit 1 which is the first bit dedicated to lighting group control. This effectively sets the lighting group bit equal to the timer.



7. Press the “>>” button to advance to the next PLC block.

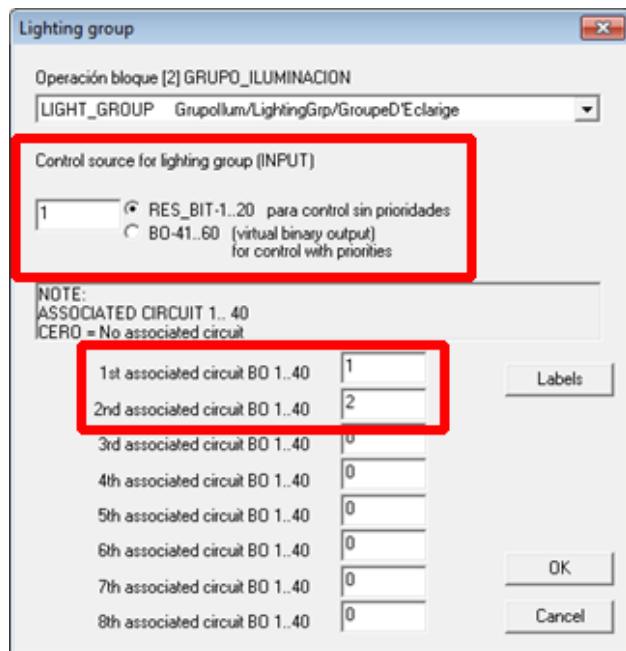


8. Press the “Delete instruction” button if there is any outstanding logic in the block then press the “Lighting Group” button to set up the first lighting group.

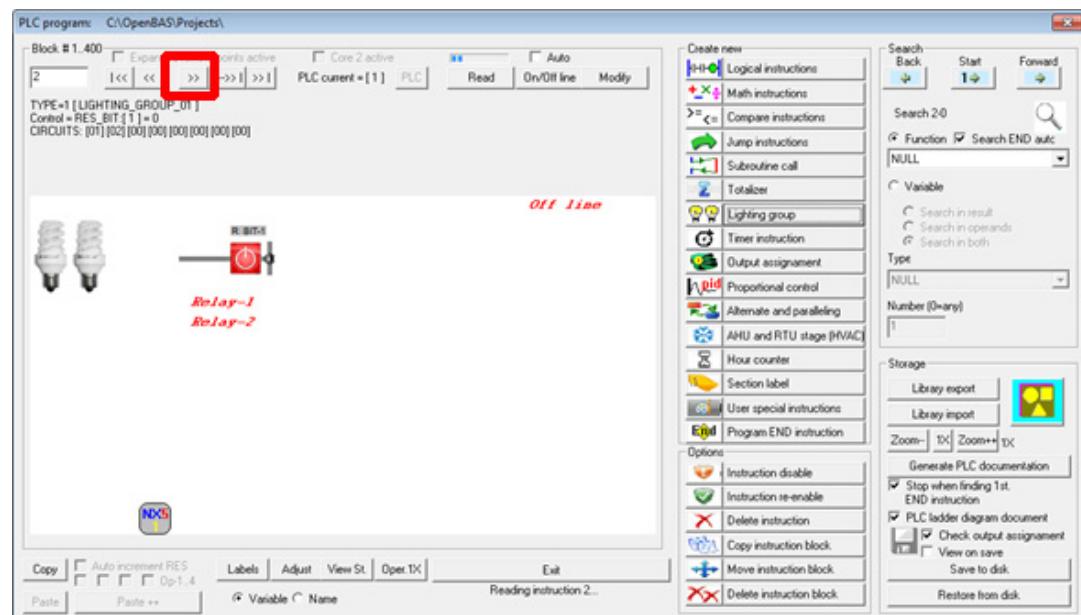


9. In the dialog box there are two main groups. The control source for the lighting group which can be either 1 of the 20 available register bits dedicated to lighting groups or virtual binary outputs. In this case use register bit 1 which was set up as a timer

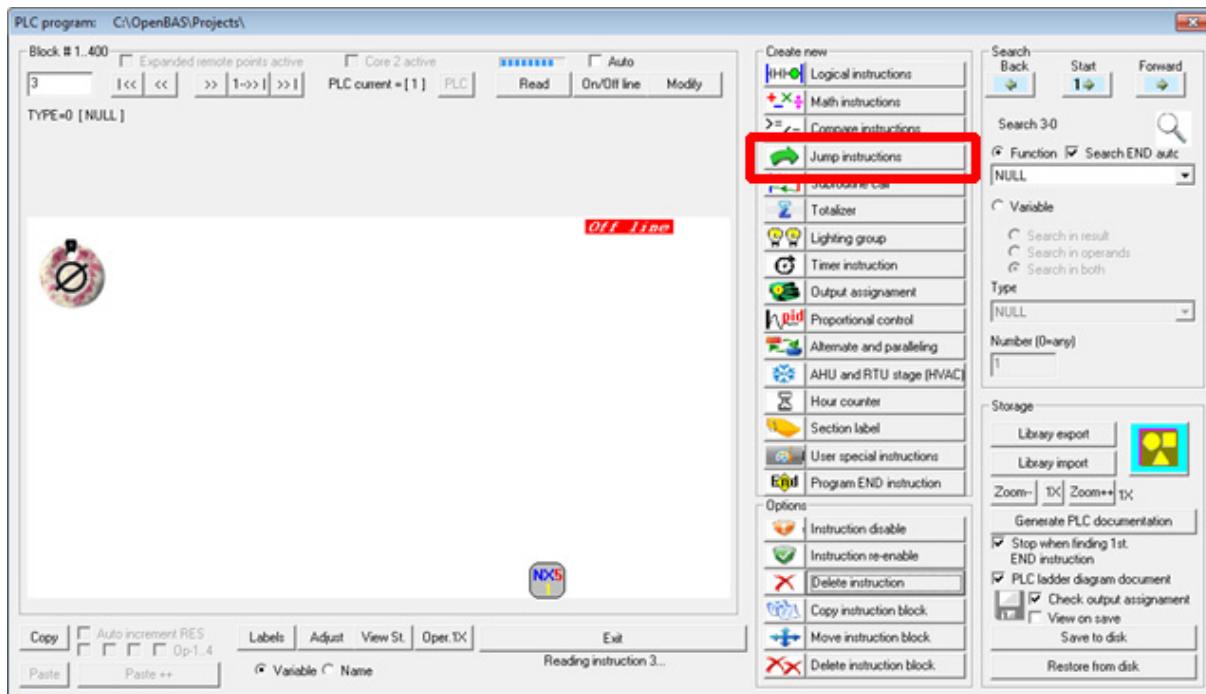
previously. In the circuits section, any circuit labeled 0 is not included and any circuit given a number will have that number as its name and will be added to the group.



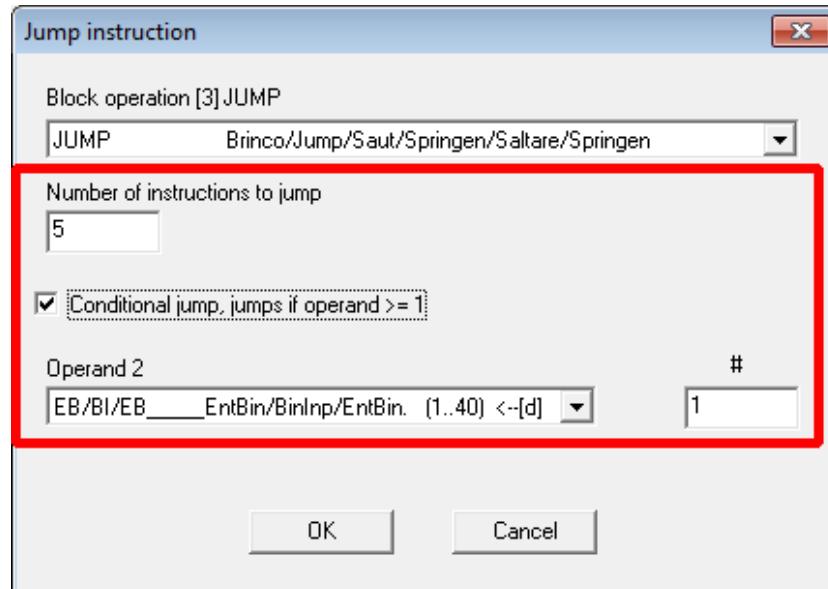
- As shown on the screen, the two relays controlled by the register bit are shown. Press the “>>” to advance to the next PLC block.



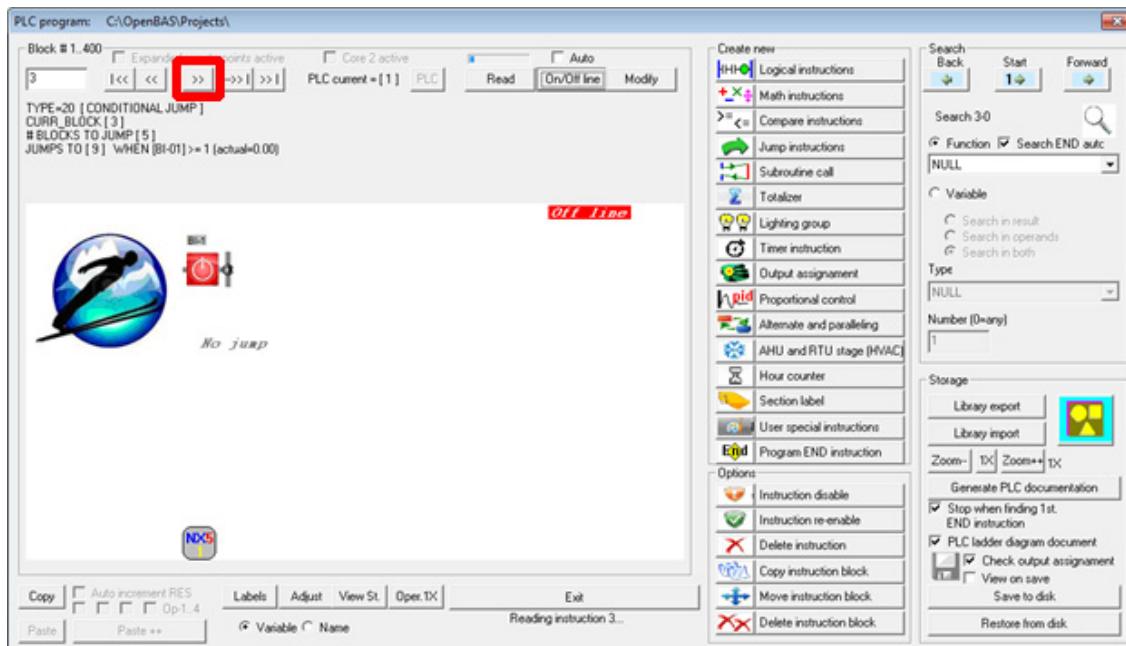
11. Press the “Delete instruction” button to clear any outstanding logic and then press the “Jump instructions” button to make the first jump.



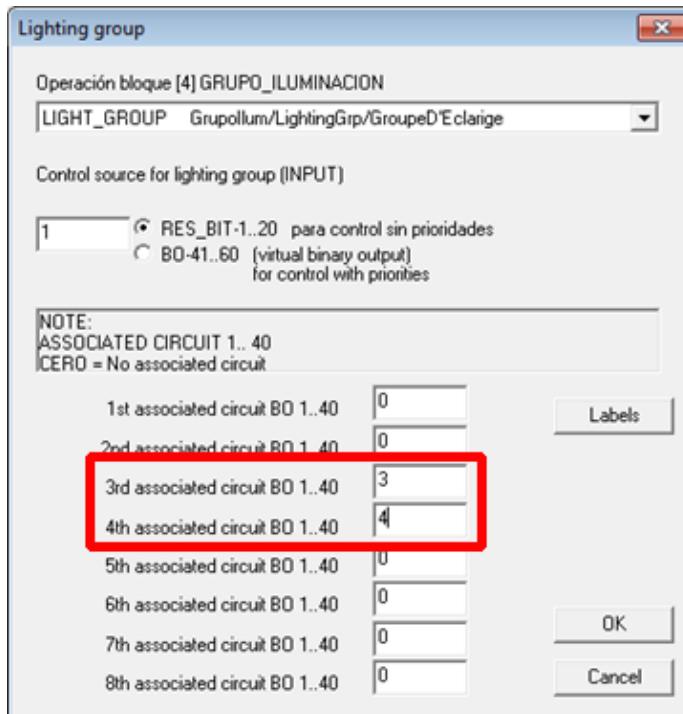
12. In the dialog box there are 3 important parts. First for “Number of instructions to jump” indicates how many PLC blocks to skip. In this case put 5 as this will skip to the end of the program. The “Conditional jump, if operand >= 1” check box allows for making the jump conditional based on the operand. Make sure this box is selected. The operand section is how the conditionality is controlled. In this case select binary input 1.



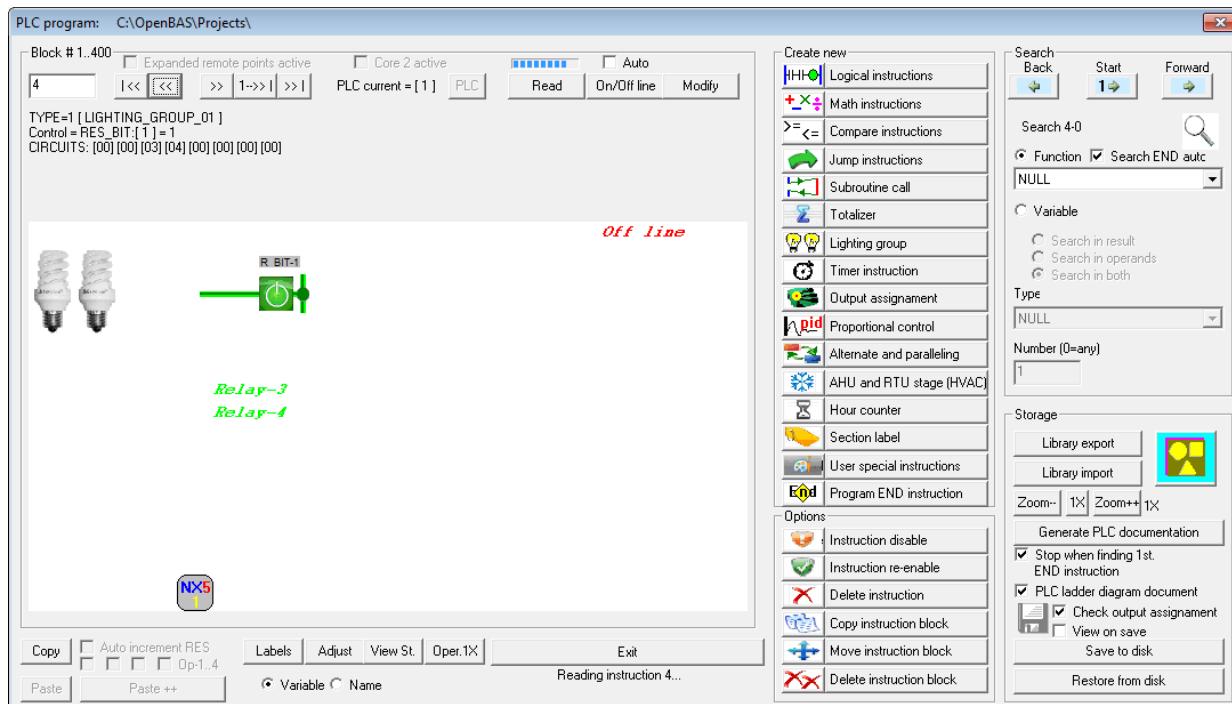
13. As shown, the jump graphic shows the status of the condition of the jump. Press this “>>” button to advance to the next PLC block.



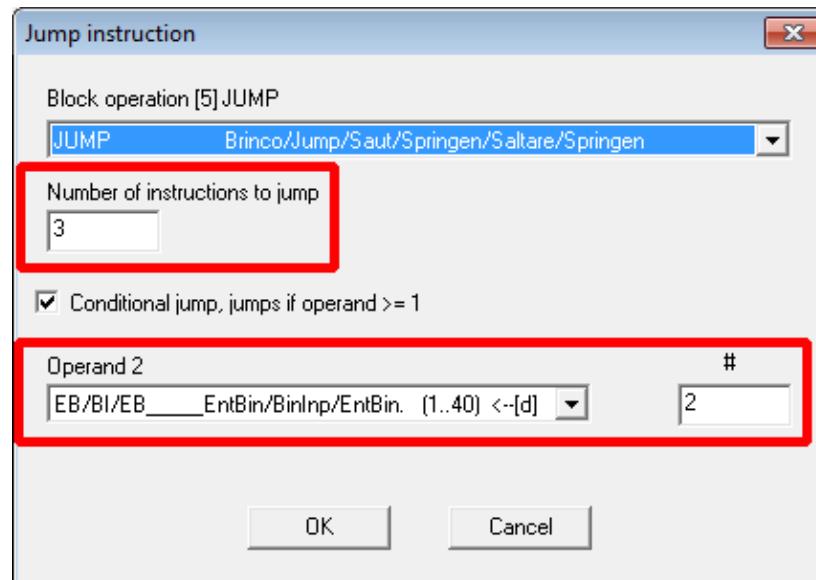
14. Just as previously done, create a lighting group by clearing any existing logic and pressing the “Lighting group” button. This lighting group will control output 3 and 4 as shown below.



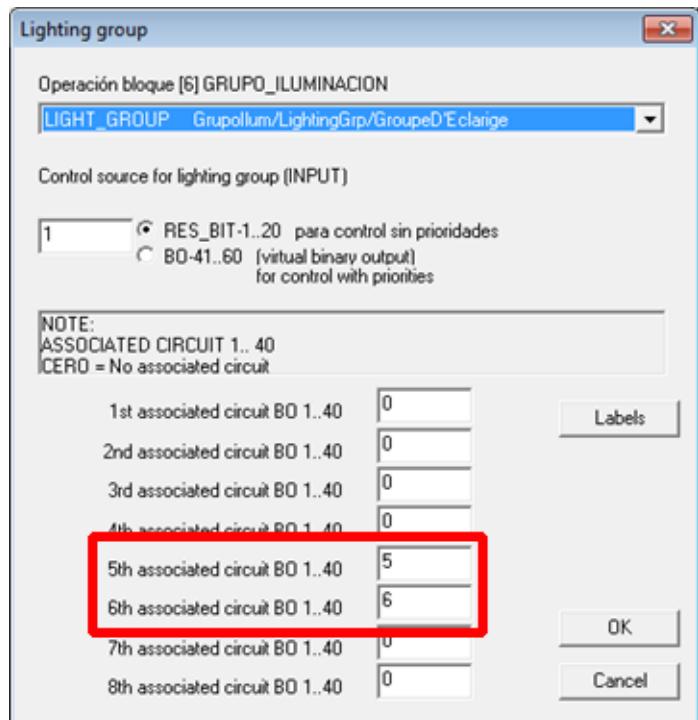
15. Again the lighting group graphic shows the state of the lighting group and all the relays in it.



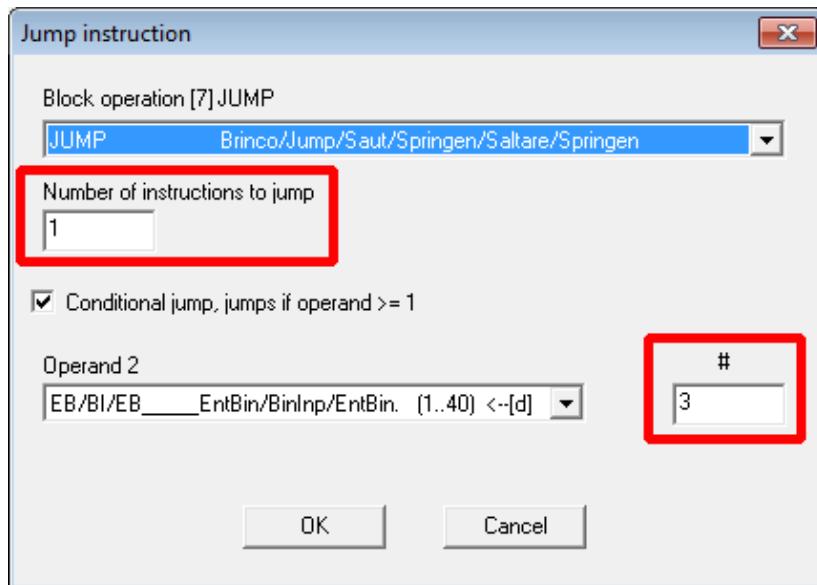
16. To make the next "Jump" follow the same steps of advancing to the next PLC block, clear any existing logic and press the "Jump instructions". This jump will jump 3 instructions this time to get to the end of the program and uses binary input 2 as the operand instead of input 1.



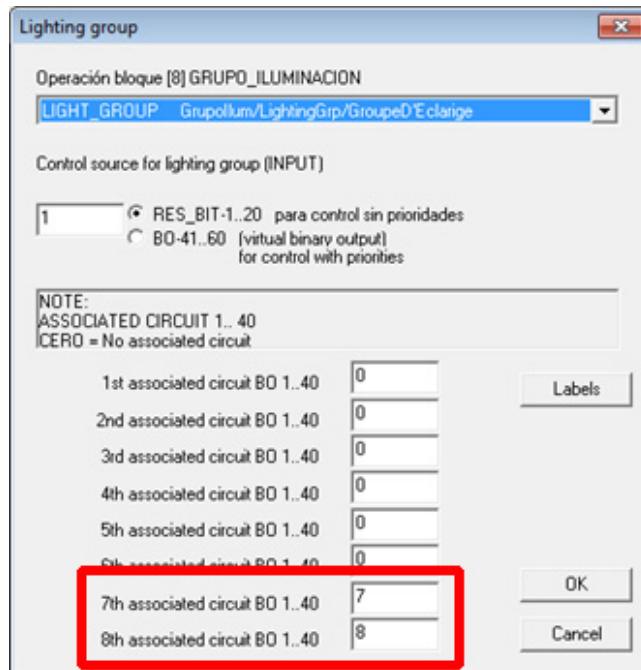
17. To create the next lighting group, repeat steps 8 -10 to create the lighting groups as shown.



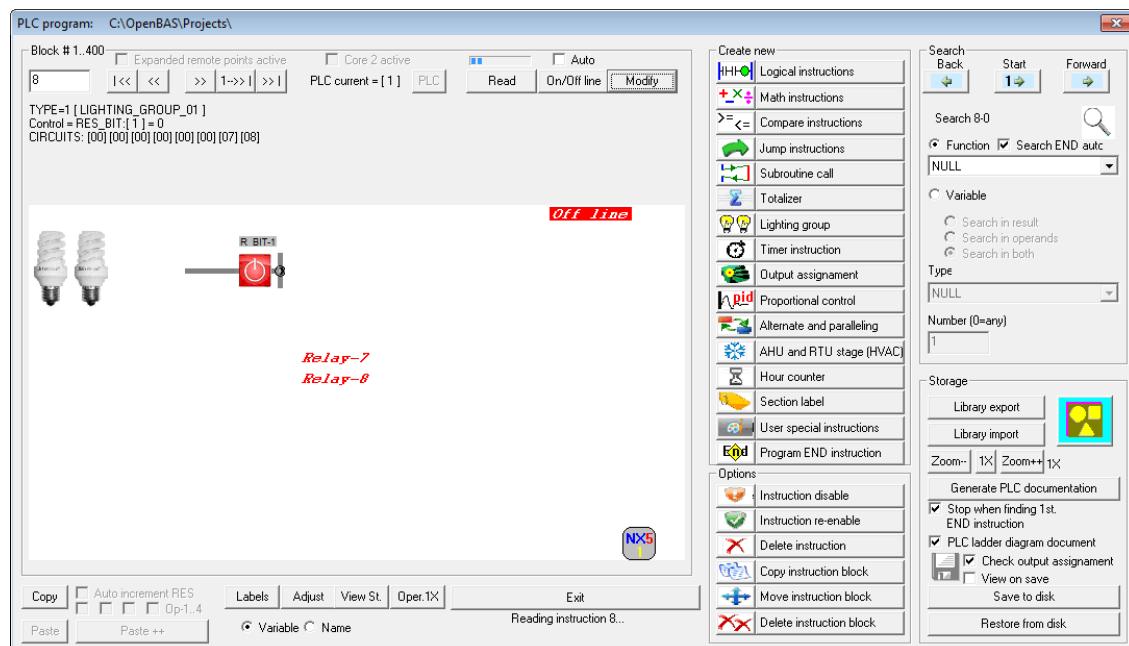
18. To make the final jump, repeat step 16 with the parameters shown in dialog box below.



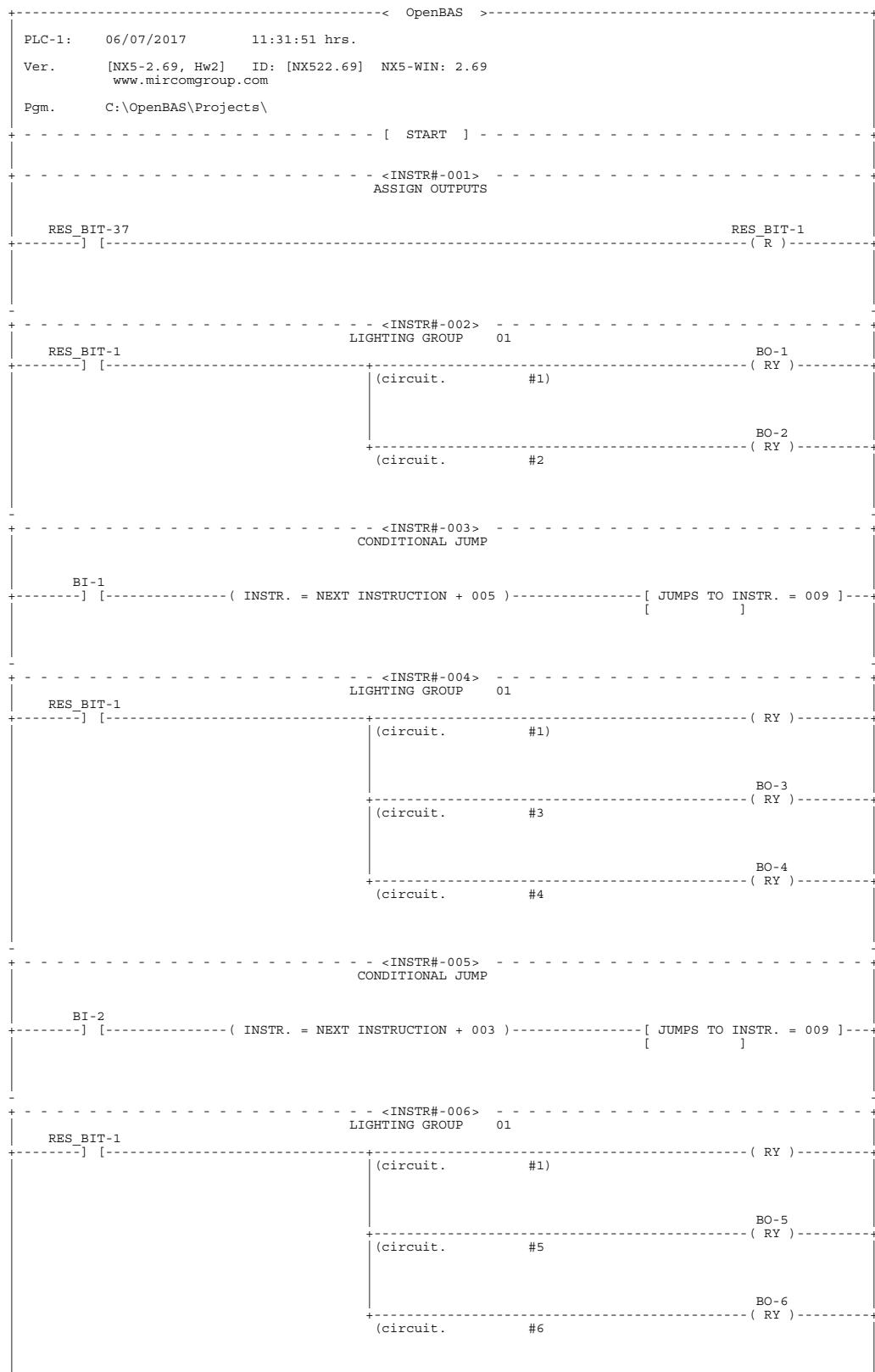
19. Finally to create the last lighting group, repeat steps 8 -10 again with the parameters in the dialog box below.



20. The program is now complete. When no binary inputs are on, all eight outputs will flash in accordance to the 1 second fixed timer. If the input is on then all logic blocks after the first lighting group will be skipped. As a result outputs 1 and 2 will continue flash but outputs 3-8 will be frozen its previous state. Input 2 allows two more outputs to be unaffected and so on for the next input.



5.2 PLC Ladder Diagram



```

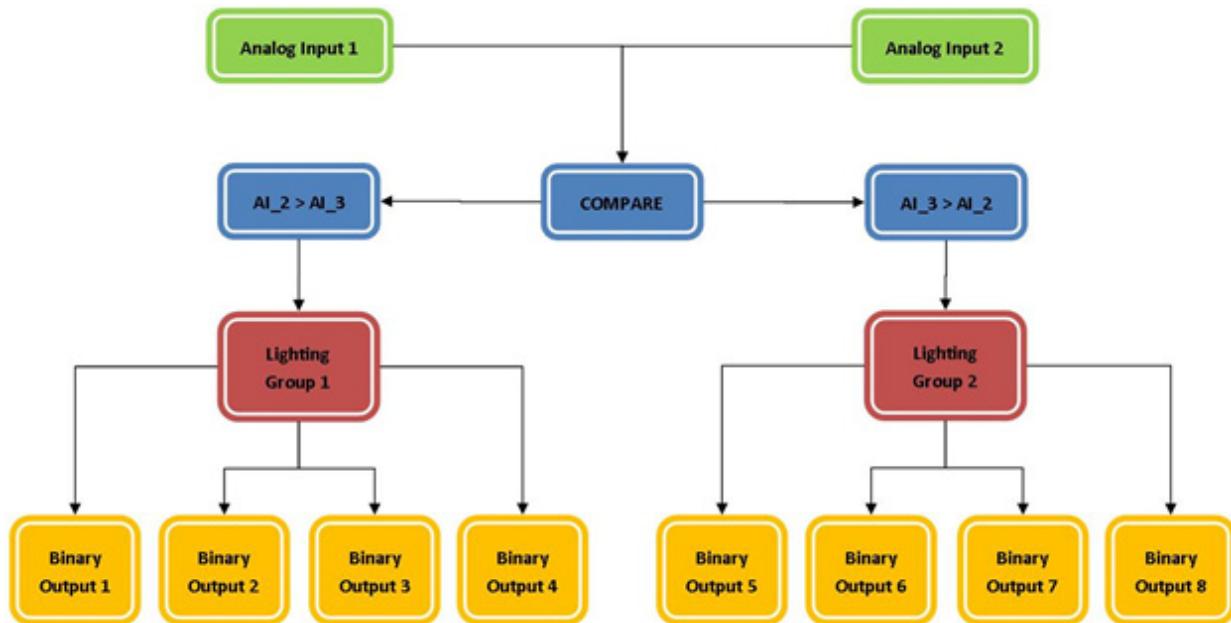
+-----<INSTR#-007>-----+
| BI-3 [----- ( INSTR. = NEXT INSTRUCTION + 001 ) ----- [ JUMPS TO INSTR. = 009 ] -----+
|-----+
|
+-----<INSTR#-008>-----+
| RBS_BIT-1 [----- LIGHTING GROUP 01
|-----+
| (circuit. #1) ( RY )
| (circuit. #7) ( RY )
| (circuit. #8) ( RY )
|
+-----<INSTR#-009>-----+
|-----+
| (program continues)-----< END OpenBAS >-----+
|-----<INSTR#-010>-----+
|-----< END OpenBAS >-----+
|
[FIN]

```

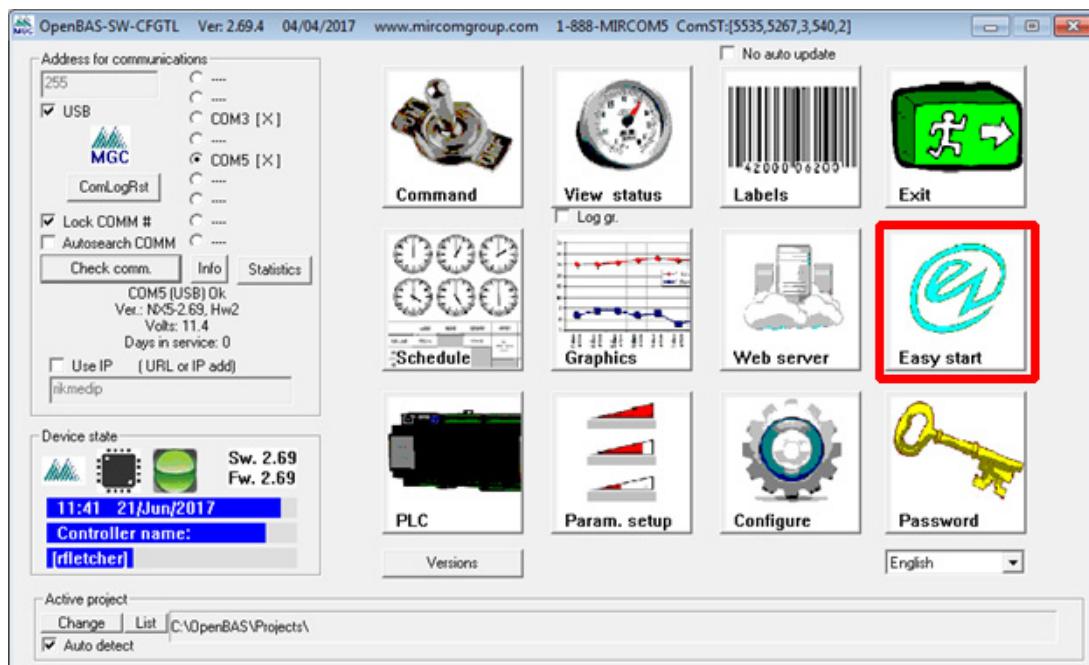
6.0 Script Programming Tutorial (Generic Application)

This tutorial will outline how to use the eZ HVAC wizard using the generic application option to create a simple script programming from scratch. This program will compare the values inputted through analog inputs 2 and 3 therefore turning on the corresponding lighting group to display the result of the comparison.

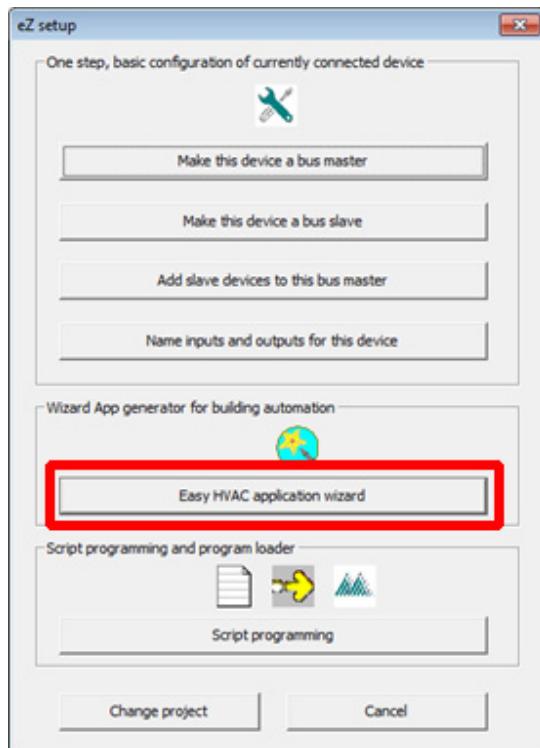
6.1 Block Diagram



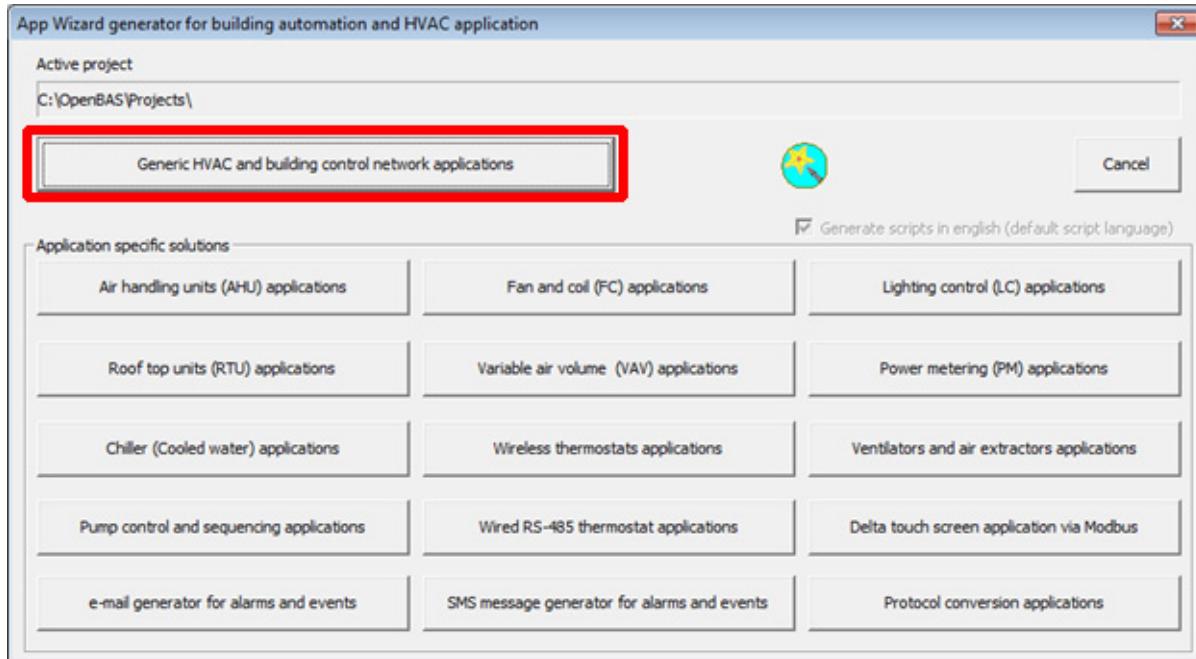
1. Select the “Easy start” button from the main screen of the OpenBAS software



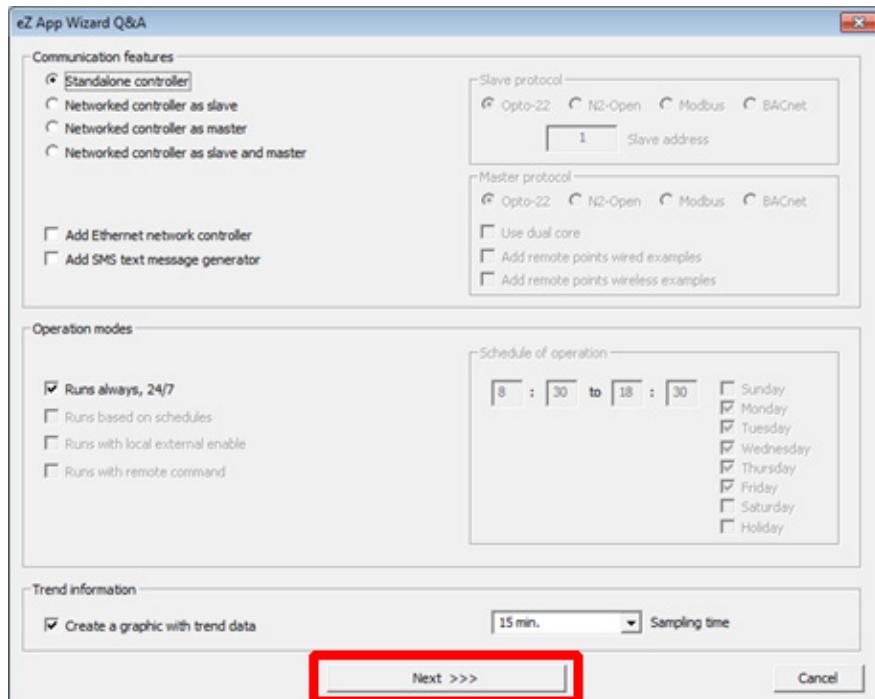
2. Select the “Easy HVAC application wizard” button from the dialog box



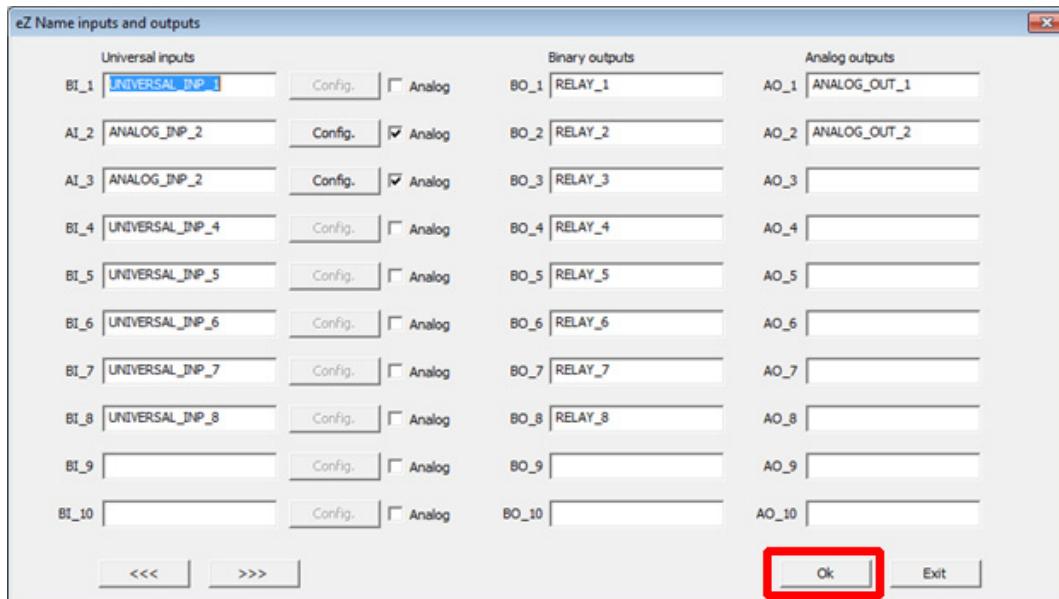
3. Press the “Generic HVAC and building control network applications” button to open a default script compiler template the defines inputs and outputs for general use opposed to a specific application.



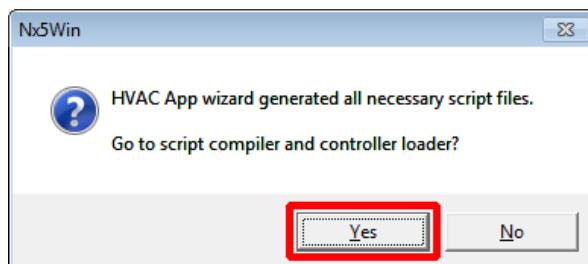
4. The wizard will present the user with the option to set schedules and or define master/slave relationship. Press “Next >>>” to continue.



5. The dialog box will present to the option to label any of the analog and binary inputs and outputs. Press “Ok” to advance.



6. The wizard is ready to generate the script files for the project. Press “Yes” to go to the script editor.



7. The wizard opens the generated script file which defines all the inputs and outputs with their corresponding label as defined by the previous dialog boxes. At the bottom of the script is the labelled section for adding the main body of the program.

```

script_1 - Notepad
File Edit Format View Help
// ENGLISH
// English is default script compiler language, by ez App wizzard by

/////////////////////////////////////////////////////////////////////////
// Generic Application for HVAC generated by HVAC APP wizard

/////////////////////////////////////////////////////////////////////////
DEFINE RES_BIT_40 = RUN_MODE

/////////////////////////////////////////////////////////////////////////
// RUN mode selected 24/7 (always on).
RUN_MODE = ON

/////////////////////////////////////////////////////////////////////////
// Universal inputs
DEFINE BI_1 UNIVERSAL_INP_1
DEFINE AI_2 ANALOG_INP_2
DEFINE AI_3 ANALOG_INP_3
DEFINE BI_4 UNIVERSAL_INP_4
DEFINE BI_5 UNIVERSAL_INP_5
DEFINE BI_6 UNIVERSAL_INP_6
DEFINE BI_7 UNIVERSAL_INP_7
DEFINE BI_8 UNIVERSAL_INP_8

/////////////////////////////////////////////////////////////////////////
// Binary outputs (relays)
DEFINE BO_1 RELAY_1
DEFINE BO_2 RELAY_2
DEFINE BO_3 RELAY_3
DEFINE BO_4 RELAY_4
DEFINE BO_5 RELAY_5
DEFINE BO_6 RELAY_6
DEFINE BO_7 RELAY_7
DEFINE BO_8 RELAY_8

/////////////////////////////////////////////////////////////////////////
// Analog outputs
DEFINE AO_1 ANALOG_OUT_1
DEFINE AO_2 ANALOG_OUT_2

/////////////////////////////////////////////////////////////////////////
// Analog input configuration
AI_CONFIGURATION AI_2 = 4 AI_CALIBRATION = 10.000000
AI_CONFIGURATION AI_3 = 4 AI_CALIBRATION = 10.000000

/////////////////////////////////////////////////////////////////////////
// Add trends to create graphics
TREND UNIVERSAL_INP_1
TREND ANALOG_INP_2 INTERVAL_MINUTES = 15
TREND ANALOG_INP_3 INTERVAL_MINUTES = 15
TREND UNIVERSAL_INP_4
TREND UNIVERSAL_INP_5
TREND UNIVERSAL_INP_6
TREND UNIVERSAL_INP_7
TREND UNIVERSAL_INP_8

/////////////////////////////////////////////////////////////////////////
// Check if control is supposed to run
// if not, turn off all outputs
IF RUN_MODE != OFF THEN JUMP PROG_START
RELAY_1 = OFF
RELAY_2 = OFF
RELAY_3 = OFF
RELAY_4 = OFF
RELAY_5 = OFF
RELAY_6 = OFF
RELAY_7 = OFF
RELAY_8 = OFF
ANALOG_OUT_1 = 0
ANALOG_OUT_2 = 0
END // Nothing else to do until controller is put into RUN mode

/////////////////////////////////////////////////////////////////////////
// Program start, do whatever you want to,
// starting from this point...
[PROG_START]
// TO DO, add your logic here.

END

```

8. First the user must define register bits for use in controlling the lighting groups. To do this, define register bits 1 and 2 with a name to be used elsewhere in the script.

```

script_1 - Notepad
File Edit Format View Help
ENGLISH
// English is default script compiler language, by ez App wizzard by

/////////////////////////////////////////////////////////////////////////
// Generic Application for HVAC generated by HVAC APP wizard

DEFINE RES_BIT_40 = RUN_MODE
DEFINE RES_BIT_1 = GROUP_1
DEFINE RES_BIT_2 = GROUP_2

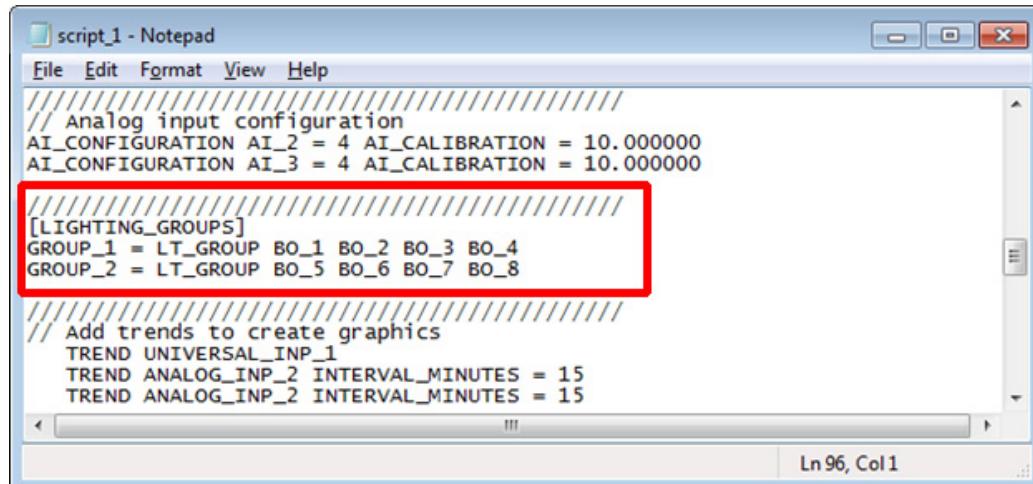
/////////////////////////////////////////////////////////////////////////
// RUN mode selected 24/7 (always on).
RUN_MODE = ON

/////////////////////////////////////////////////////////////////////////
// Universal inputs
DEFINE BI_1 UNIVERSAL_INP_1
DEFINE AI_2 ANALOG_INP_2
DEFINE AI_3 ANALOG_INP_3
DEFINE BI_4 UNIVERSAL_INP_4
DEFINE BI_5 UNIVERSAL_INP_5
DEFINE BI_6 UNIVERSAL_INP_6
DEFINE BI_7 UNIVERSAL_INP_7
DEFINE BI_8 UNIVERSAL_INP_8

```

9. The next step is to create lighting group using the script programming language. To create a lighting group, first give the group a name and set that equal to the lighting group function (LT_GROUP). Following the lighting group function the user defines all the outputs contained in that lighting group. As shown below, binary outputs 1-4 are in the first group and 5-8 are contained in the second group. By placing outputs in a lighting

group the programmer can control all the outputs in the group together with one command.



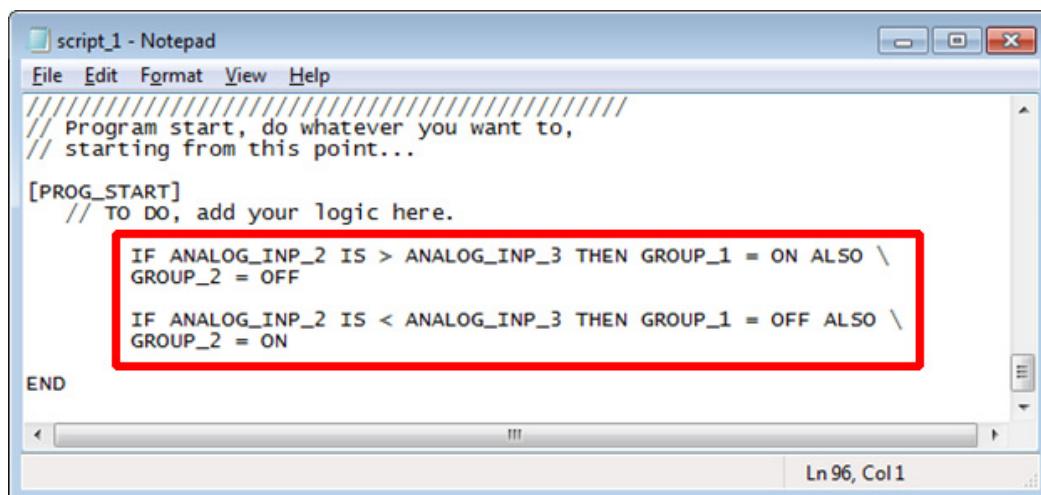
```
script_1 - Notepad
File Edit Format View Help
/////////////////////////////////////////////////////////////////
// Analog input configuration
AI_CONFIGURATION AI_2 = 4 AI_CALIBRATION = 10.000000
AI_CONFIGURATION AI_3 = 4 AI_CALIBRATION = 10.000000

/////////////////////////////////////////////////////////////////
[LIGHTING_GROUPS]
GROUP_1 = LT_GROUP BO_1 BO_2 BO_3 BO_4
GROUP_2 = LT_GROUP BO_5 BO_6 BO_7 BO_8

/////////////////////////////////////////////////////////////////
// Add trends to create graphics
TREND UNIVERSAL_INP_1
TREND ANALOG_INP_2 INTERVAL_MINUTES = 15
TREND ANALOG_INP_2 INTERVAL_MINUTES = 15

Ln 96, Col 1
```

10. The main body of the program is quite simple in this case. Two “if” statements are needed to make the comparisons between analog inputs 2 and 3. The first “if” statement compares if input 2 is greater than input 3. If that is the case then lighting group 1 is on and lighting group 2 is off. If input 3 is greater than input 2 then the opposite is true.



```
script_1 - Notepad
File Edit Format View Help
/////////////////////////////////////////////////////////////////
// Program start, do whatever you want to,
// starting from this point...

[PROG_START]
// TO DO, add your logic here.

    IF ANALOG_INP_2 IS > ANALOG_INP_3 THEN GROUP_1 = ON ALSO \
        GROUP_2 = OFF

    IF ANALOG_INP_2 IS < ANALOG_INP_3 THEN GROUP_1 = OFF ALSO \
        GROUP_2 = ON

END

Ln 96, Col 1
```

11. The completed script program should like as shown below.

```

script_1 - Notepad
File Edit Format View Help
ENGLISH
// English is default script compiler language, by ez App wizzard

/////////////////////////////////////////////////////////////////////////
// Generic application for HVAC generated by HVAC APP wizard
/////////////////////////////////////////////////////////////////////////

DEFINE RES_BIT_40 = RUN_MODE
DEFINE RES_BIT_1 = GROUP_1
DEFINE RES_BIT_2 = GROUP_2

// RUN mode selected 24/7 (always on).
RUN_MODE = ON

/////////////////////////////////////////////////////////////////////////
// universal inputs
DEFINE BI_1 UNIVERSAL_INP_1
DEFINE BI_2 ANALOG_INP_2
DEFINE BI_3 ANALOG_INP_3
DEFINE BI_4 UNIVERSAL_INP_4
DEFINE BI_5 UNIVERSAL_INP_5
DEFINE BI_6 UNIVERSAL_INP_6
DEFINE BI_7 UNIVERSAL_INP_7
DEFINE BI_8 UNIVERSAL_INP_8

/////////////////////////////////////////////////////////////////////////
// Binary outputs (relays)
DEFINE BO_1 RELAY_1
DEFINE BO_2 RELAY_2
DEFINE BO_3 RELAY_3
DEFINE BO_4 RELAY_4
DEFINE BO_5 RELAY_5
DEFINE BO_6 RELAY_6
DEFINE BO_7 RELAY_7
DEFINE BO_8 RELAY_8

/////////////////////////////////////////////////////////////////////////
// Analog OUTPUTS
DEFINE AO_1 ANALOG_OUT_1
DEFINE AO_2 ANALOG_OUT_2

/////////////////////////////////////////////////////////////////////////
// Analog input configuration
AT_CONFIGURATION AI_2 = 4 AI_CALIBRATION = 10.000000
AT_CONFIGURATION AI_3 = 4 AI_CALIBRATION = 10.000000

[LIGHTING_GROUP]
GROUP_1 = LT_GROUP BO_1 BO_2 BO_3 BO_4
GROUP_2 = LT_GROUP BO_5 BO_6 BO_7 BO_8

/////////////////////////////////////////////////////////////////////////
// Add trends to create graphics
TREND UNIVERSAL_INP_1
TREND ANALOG_INP_2 INTERVAL_MINUTES = 15

```



```

script_1 - Notepad
File Edit Format View Help
// Analog outputs
DEFINE AO_1 ANALOG_OUT_1
DEFINE AO_2 ANALOG_OUT_2

/////////////////////////////////////////////////////////////////////////
// Analog output configuration
AI_CONFIGURATION AI_1 = AI_CALIBRATION = 10.000000
AI_CONFIGURATION AI_3 = 4 AI_CALIBRATION = 10.000000

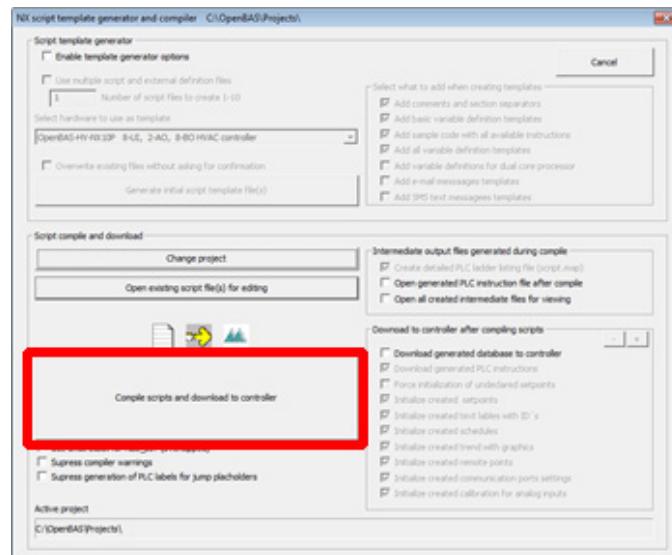
[LIGHTING_GROUP]
GROUP_1 = LT_GROUP BO_1 BO_2 BO_3 BO_4
GROUP_2 = LT_GROUP BO_5 BO_6 BO_7 BO_8

/////////////////////////////////////////////////////////////////////////
// Check if control is supposed to run
// if not, turn off all outputs
IF RUN_MODE != OFF THEN JUMP PROG_START
RELAY_1 = OFF
RELAY_2 = OFF
RELAY_3 = OFF
RELAY_4 = OFF
RELAY_5 = OFF
RELAY_6 = OFF
RELAY_7 = OFF
RELAY_8 = OFF
ANALOG_OUT_1 = 0
ANALOG_OUT_2 = 0
END // Nothing else to do until controller is put into RUN mode

/////////////////////////////////////////////////////////////////////////
// Program start, do whatever you want to,
// starting from this point...
[PROG_START]
// To do... add your logic here
IF ANALOG_INP_2 IS > ANALOG_INP_3 THEN GROUP_1 = ON ALSO \
GROUP_2 = OFF
IF ANALOG_INP_2 IS < ANALOG_INP_3 THEN GROUP_1 = OFF ALSO \
GROUP_2 = ON
END

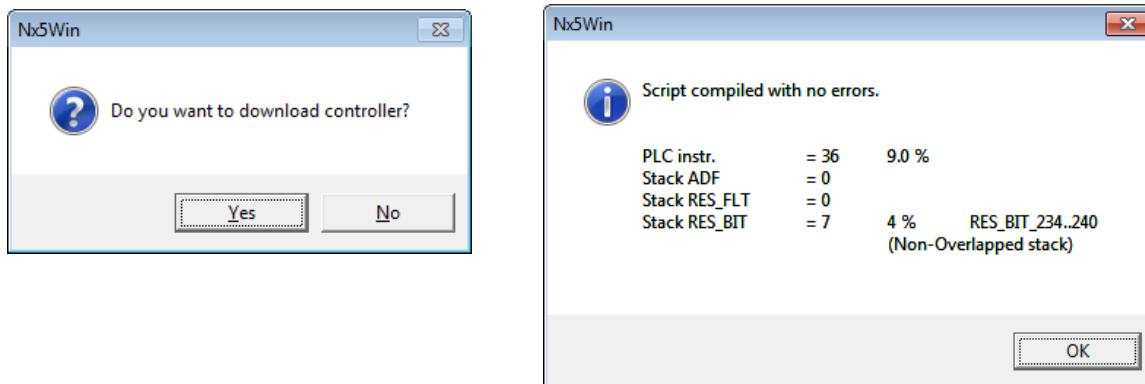
```

12. Save and close the script file to proceed to the below menu. Press the large “Compile scripts and download to controller” button.

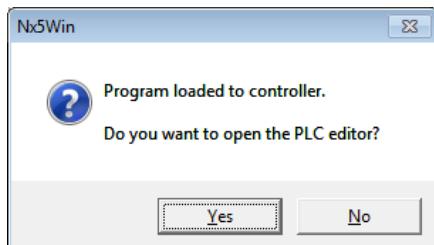


13. The software will alert the user to any errors should there be any. Pressing “Ok” will then ask the user again if they want to download to the controller. The software will begin

opening and closing a series of menus as it downloaded to the controller. Make sure to let the software do its job and don't any of the menus that appear.



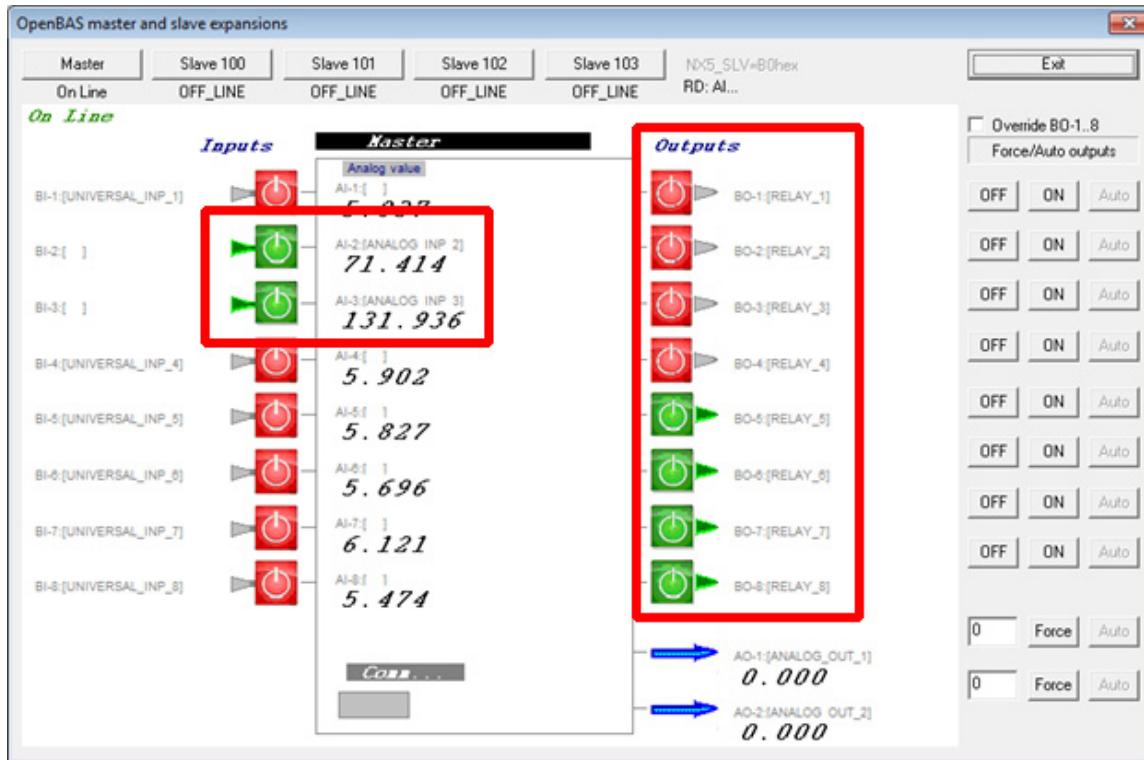
- Once the download is complete, the user will be presented with the option to open the PLC editor to view the PLC instructions created from the script program. The user can select either option to their liking at this stage.



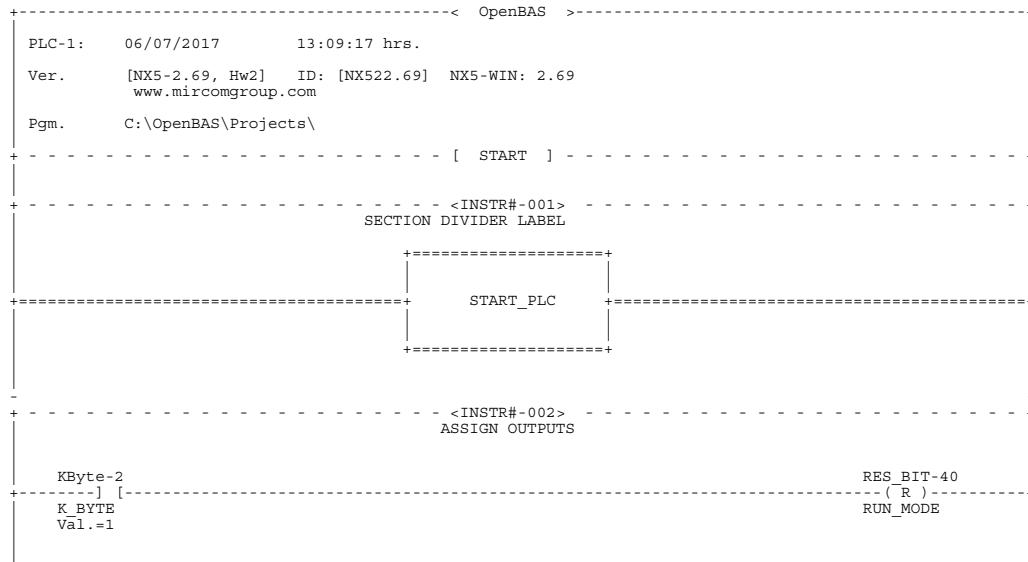
- Exit out of all the windows open until you reach the main OpenBAS screen. The best way to view the values of the inputs and outputs in action is to go to the “Command” window.

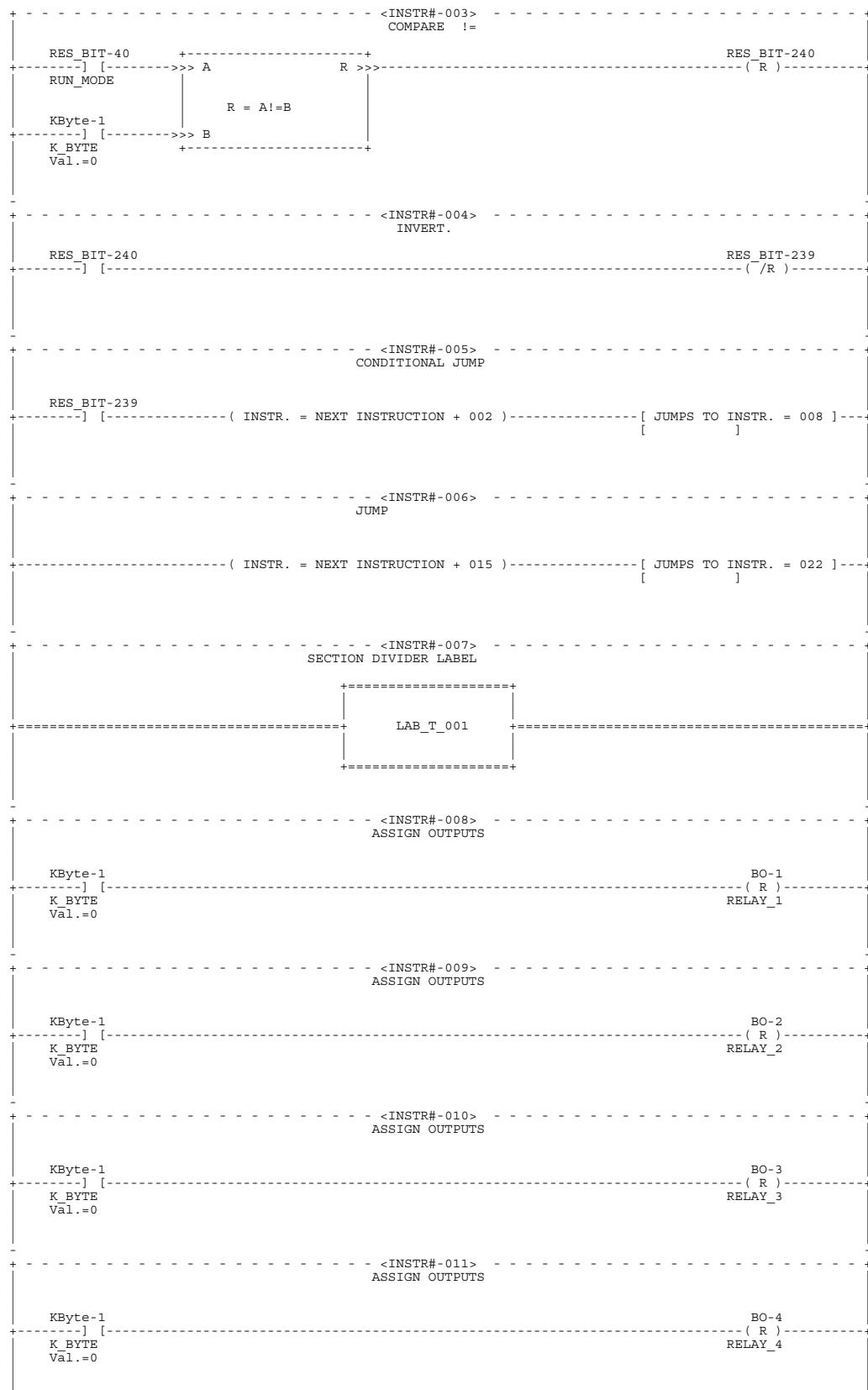


16. The command window allows the user to view all the values and states of the analog inputs and binary outputs. As shown below, analog input 3 is greater than input 2 and thus the second lighting group (outputs 5-8) is on.



6.2 PLC Ladder Diagram





```

+-----<INSTR#-012>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-013>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-014>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-015>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-016>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-017>-----+
| ASSIGN OUTPUTS
+-----[ KByte-1 [-----+
| K_BYTE
| Val.=0
+-----] ]-----+
+-----<INSTR#-018>-----+
| END
+---(program continues)---< OpenBAS >---+
+-----<INSTR#-019>-----+
| LIGHTING GROUP 01
+-----[ RES_BIT-1 [-----+
| GROUP_1 [-----+
| (circuit. #1)-----+
| (circuit. #2)-----+
| (circuit. #3)-----+
| (circuit. #4)-----+
+-----] ]-----+
+-----BO-5 ( R )-----+
| RELAY_5
+-----BO-6 ( R )-----+
| RELAY_6
+-----BO-7 ( R )-----+
| RELAY_7
+-----BO-8 ( R )-----+
| RELAY_8
+-----AO-1 ( R )-----+
| ANALOG_OU
| T_1
+-----AO-2 ( R )-----+
| ANALOG_OU
| T_2
+-----BO-1 ( RY )-----+
| RELAY_1
+-----BO-2 ( RY )-----+
| RELAY_2
+-----BO-3 ( RY )-----+
| RELAY_3
+-----BO-4 ( RY )-----+
| RELAY_4

```

```

+-----+ <INSTR#-020> -----+
|      RES_BIT-2
+-----] [----- GROUP_2
|      (circuit. #1)          BO-5
|      (circuit. #2)          ( RY )
|      (circuit. #3)          RELAY_5
|      (circuit. #4)          BO-6
|      (circuit. #5)          ( RY )
|      (circuit. #6)          RELAY_6
|      (circuit. #7)          BO-7
|      (circuit. #8)          ( RY )
|      (circuit. #9)          RELAY_7
|      (circuit. #10)         BO-8
|      (circuit. #11)         ( RY )
|      (circuit. #12)         RELAY_8
+-----+ <INSTR#-021> -----+
|      SECTION DIVIDER LABEL
+=====+ PROG_STAR +=====
+-----+ <INSTR#-022> -----+
|      COMPARE >
+-----] [----->>> A           R >>>-----+
|      ANALOG_IN             R = A>B             RES_BIT-238
|      P_2
+-----] [----->>> B           +
|      ANALOG_IN             R = A>B             ( -R )
|      P_3
+-----+ <INSTR#-023> -----+
|      INVERT.
+-----] [----- RES_BIT-237             RES_BIT-237
|      ( /R )-----+
+-----+ <INSTR#-024> -----+
|      CONDITIONAL JUMP
+-----] [-----( INSTR. = NEXT INSTRUCTION + 003 )-----+
|      [ JUMPS TO INSTR. = 028 ]
+-----+ <INSTR#-025> -----+
|      ASSIGN OUTPUTS
+-----] [----- KByte-2             RES_BIT-1
|      K_BYTE               ( R )-----+
|      Val.=1
+-----+ <INSTR#-026> -----+
|      ASSIGN OUTPUTS
+-----] [----- KByte-1             RES_BIT-2
|      K_BYTE               ( R )-----+
|      Val.=0
+-----+ <INSTR#-027> -----+
|      SECTION DIVIDER LABEL
+=====+ LAB_T_002 +=====
+-----+

```

```

+-----+ <INSTR#-028> -----+
|      |      COMPARE <
+-----+ [----->>> A      R >>>-----+
|      |      ANALOG_IN   +-----+
|      |      P_2          R = A<B
+-----+ [----->>> B      +-----+
|      |      ANALOG_IN   +
|      |      P_3
+-----+ <INSTR#-029> -----+
|      |      INVERT.
+-----+ [----- RES_BIT-236      RES_BIT-235
|      |      ( R )      ( /R )
+-----+
+-----+ <INSTR#-030> -----+
|      |      CONDITIONAL JUMP
+-----+ [-----( INSTR. = NEXT INSTRUCTION + 003 )-----+
|      |      [ JUMPS TO INSTR. = 034 ]
+-----+
+-----+ <INSTR#-031> -----+
|      |      ASSIGN OUTPUTS
+-----+ [----- KByte-1      RES_BIT-1
|      |      ( R ) -+
|      |      GROUP_1
|      |      K_BYTE
|      |      Val.=0
+-----+ <INSTR#-032> -----+
|      |      ASSIGN OUTPUTS
+-----+ [----- KByte_2      RES_BIT-2
|      |      ( R ) -+
|      |      GROUP_2
|      |      K_BYTE
|      |      Val.=1
+-----+ <INSTR#-033> -----+
|      |      SECTION DIVIDER LABEL
+-----+ ======+ LAB_T_003 ======+
+-----+
+-----+ <INSTR#-034> -----+
|      |      END
+---(program continues)---< OpenBAS >-----+
+-----+ <INSTR#-035> -----+
|      |      END
+---< OpenBAS >-----+

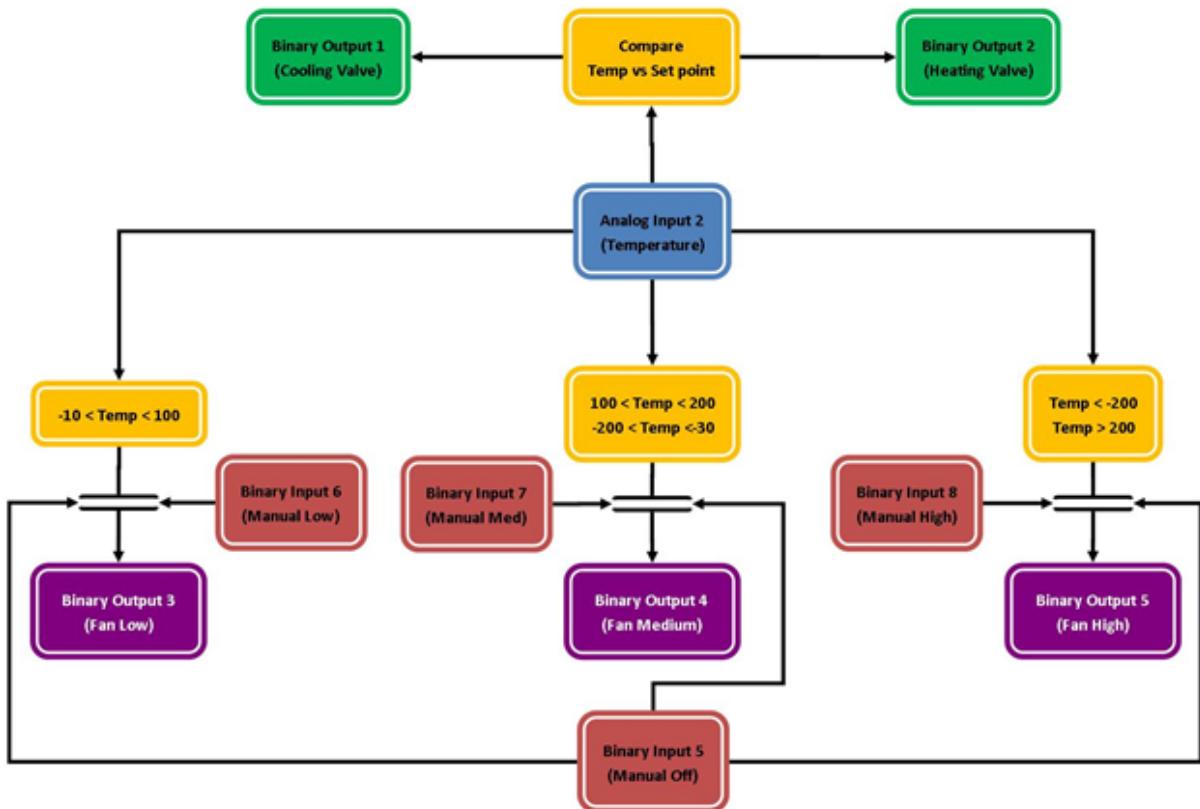
```

[FIN]

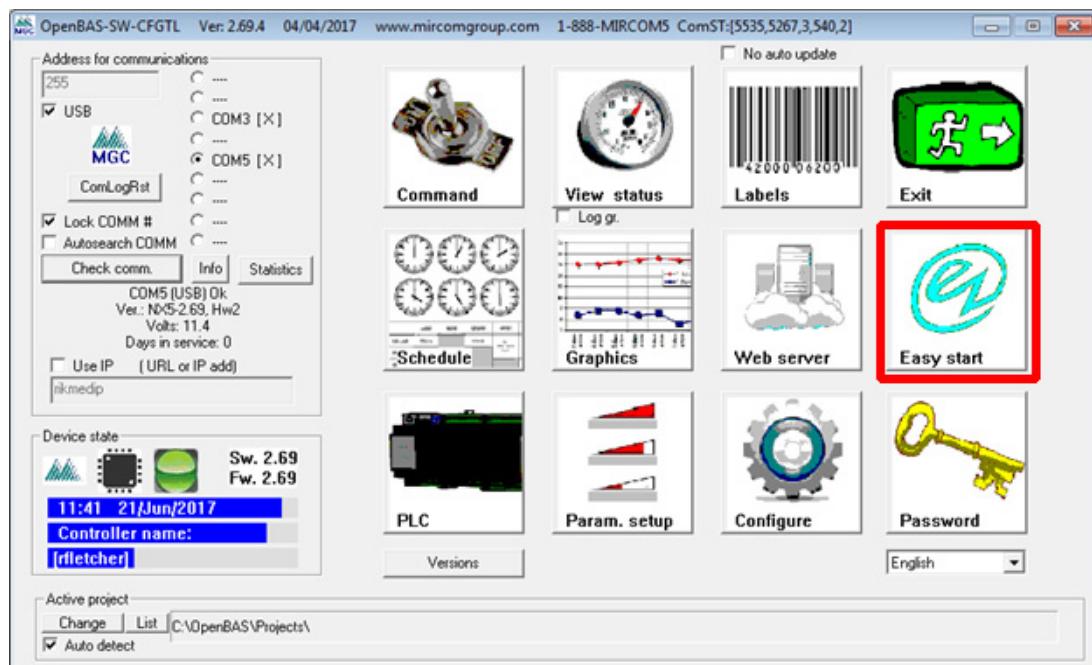
7.0 Script Programming Tutorial (Fan and Coil Applications)

This tutorial will outline how to use the eZ HVAC Wizard to create a template script that can be modified to add, remove or modify functionality to the need of the user. The script will use the default Fan and Coil Application script as the shell on which changes will be made to add reactive fan ramp up and manual overrides.

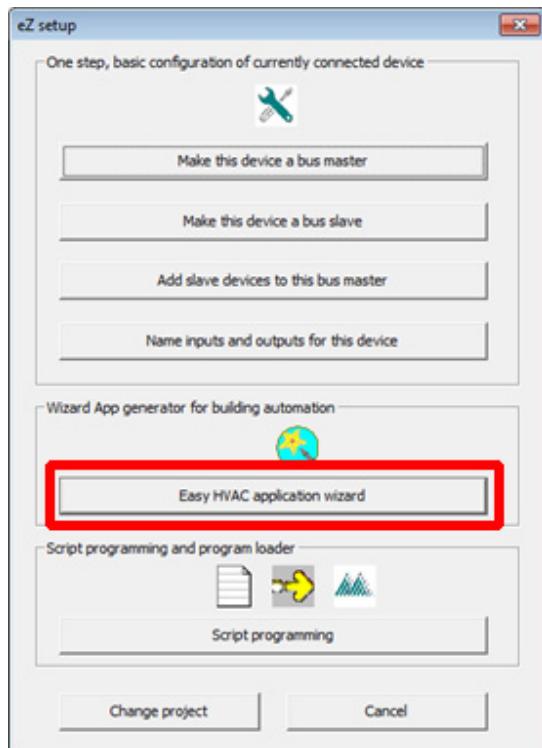
7.1 Block Diagram



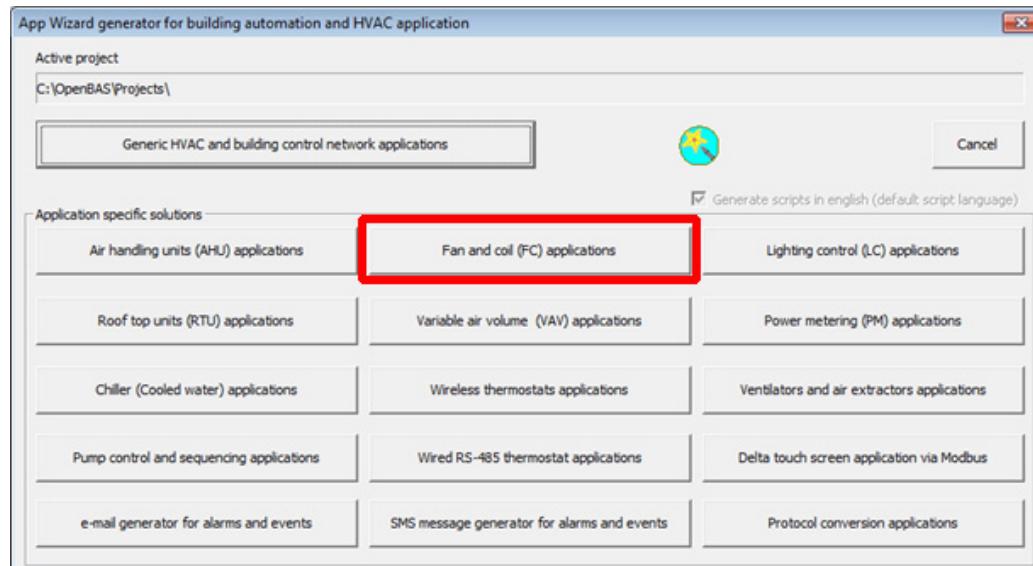
1. Select the “Easy start” button from the main screen of the OpenBAS software.



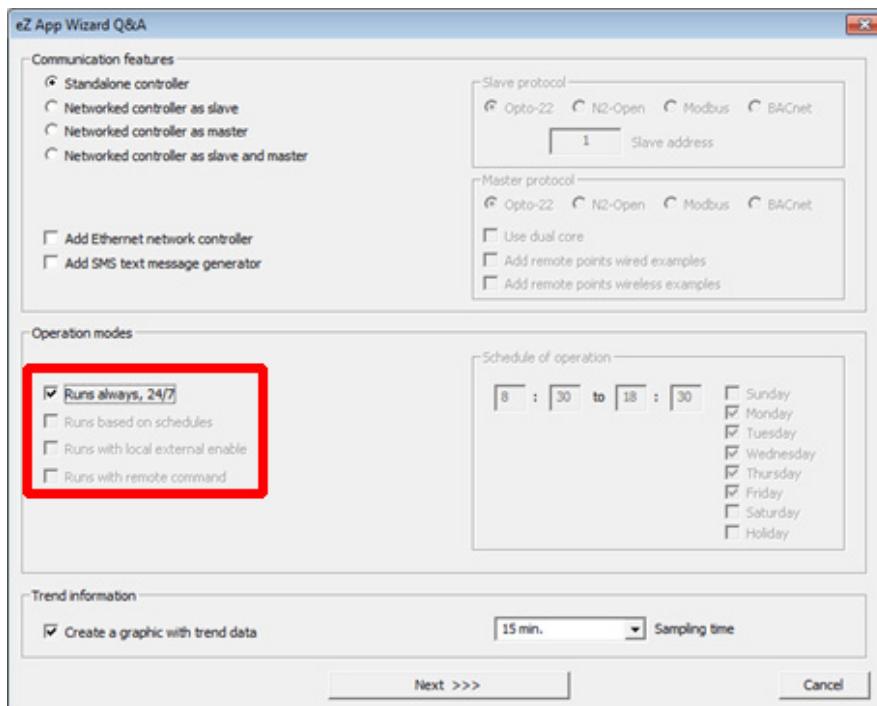
2. Press the “Easy HVAC application wizard” from the dialog box.



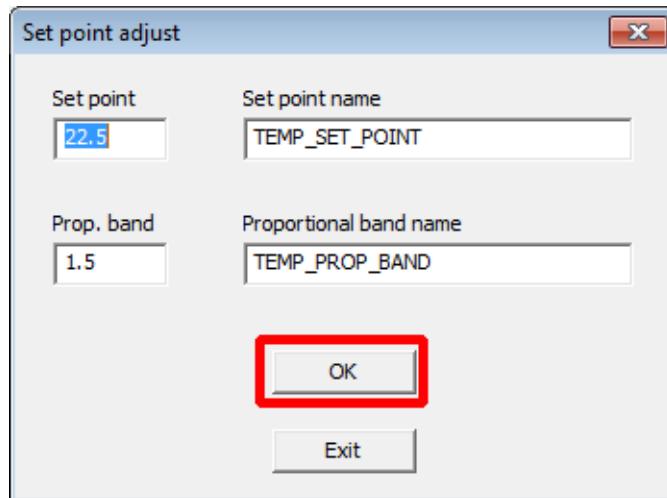
3. The App wizard presents lots of application specific options. In this case select the “Fan and coil (FC) applications” from the list of applications.



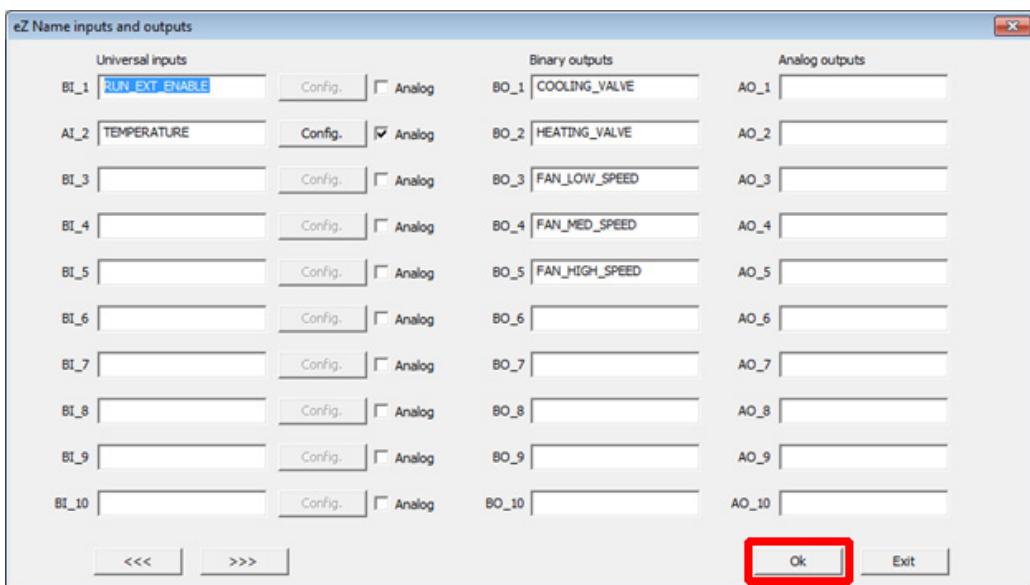
4. The wizard will present a bunch of options for customizing the fan and coil applications. For our purposes make sure “Runs always, 24/7” is selected and everything else is left default. Then press “Next >>”.



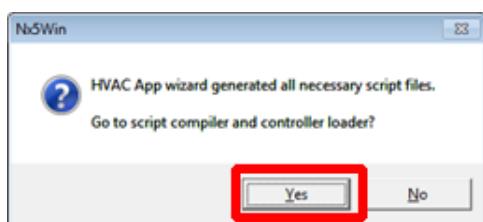
5. The wizard will next present a dialog box for selecting set point and proportional band. This can be set to anything but for our purpose leaves this default.



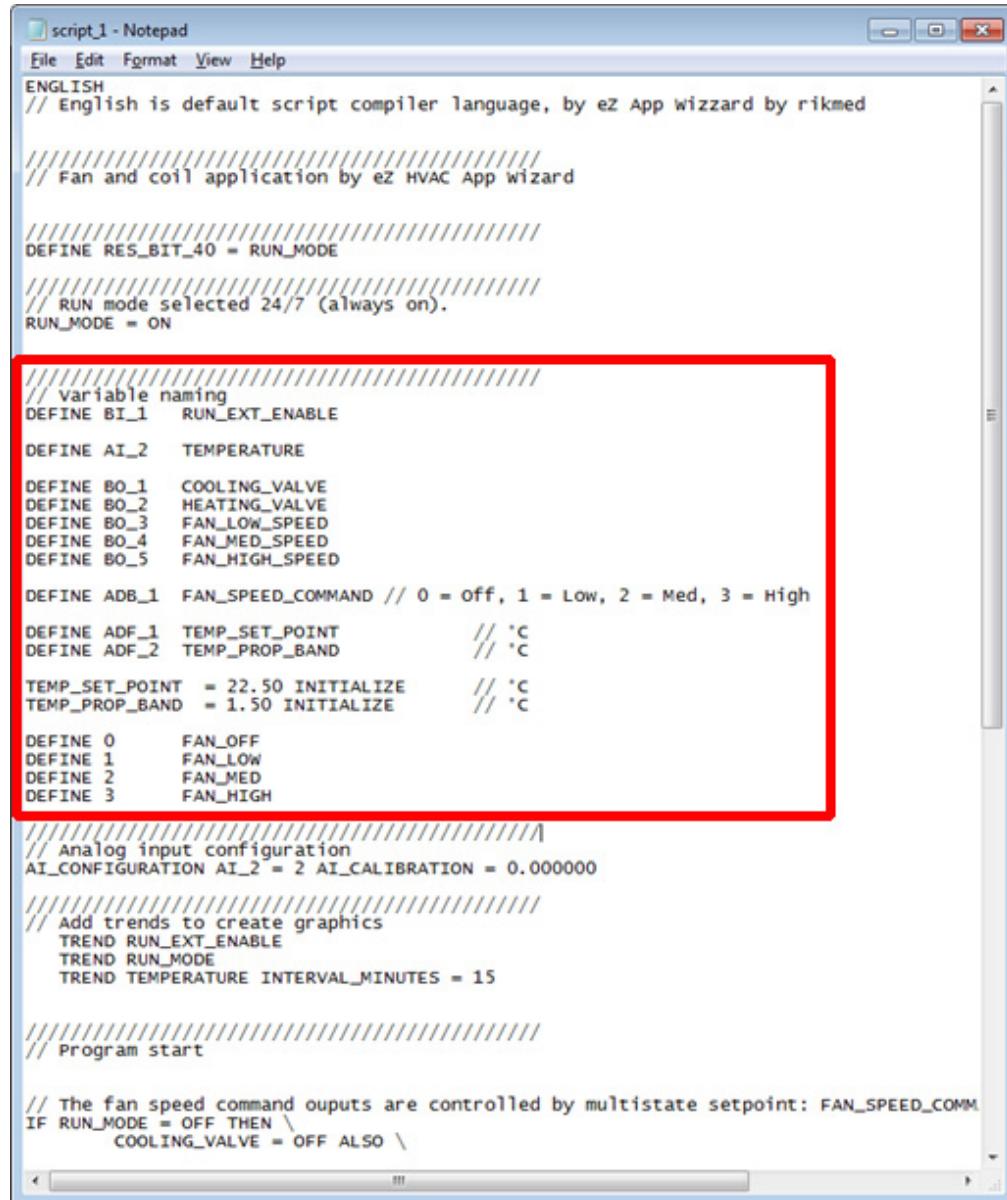
6. The wizard will then presents a menu to label all the inputs and outputs being used in the program. In our case simply leave these names as they are and press "Ok" to advance.



7. The wizard will tell the user that all the files have been created. Press "Yes" to go to the script.



8. The script will be opened and the file created contains all the script programming to fulfill the parameters specified in the previous dialog boxes. As highlighted below, all the variables for the inputs and outputs of the program are listed.



```
script_1 - Notepad
File Edit Format View Help
ENGLISH
// English is default script compiler language, by ez App wizzard by rikmed

///////////////////////////////
// Fan and coil application by ez HVAC App wizard

/////////////////////////////
DEFINE RES_BIT_40 = RUN_MODE

/////////////////////////////
// RUN mode selected 24/7 (always on).
RUN_MODE = ON

/////////////////////////////
//variable naming
DEFINE BI_1    RUN_EXT_ENABLE

DEFINE AI_2    TEMPERATURE

DEFINE BO_1    COOLING_VALVE
DEFINE BO_2    HEATING_VALVE
DEFINE BO_3    FAN_LOW_SPEED
DEFINE BO_4    FAN_MED_SPEED
DEFINE BO_5    FAN_HIGH_SPEED

DEFINE ADB_1   FAN_SPEED_COMMAND // 0 = Off, 1 = Low, 2 = Med, 3 = High
DEFINE ADF_1   TEMP_SET_POINT          // 'C
DEFINE ADF_2   TEMP_PROP_BAND         // 'C

TEMP_SET_POINT = 22.50 INITIALIZE
TEMP_PROP_BAND = 1.50 INITIALIZE

DEFINE 0      FAN_OFF
DEFINE 1      FAN_LOW
DEFINE 2      FAN_MED
DEFINE 3      FAN_HIGH

/////////////////////////////
// Analog input configuration
AI_CONFIGURATION AI_2 = 2 AI_CALIBRATION = 0.000000

/////////////////////////////
// Add trends to create graphics
TREND RUN_EXT_ENABLE
TREND RUN_MODE
TREND TEMPERATURE INTERVAL_MINUTES = 15

/////////////////////////////
// Program start

// The fan speed command ouputs are controlled by multistate setpoint: FAN_SPEED_COMM
IF RUN_MODE = OFF THEN \
    COOLING_VALVE = OFF ALSO \
```

9. In order to add manual override functionality additional binary inputs need to be declared in the variable naming section discussed before. As highlighted, binary inputs, 5-8 are given the names MANUAL_OFF, LOW, MED, HIGH.

```

script_1 - Notepad
File Edit Format View Help

/////////////////////////////
// Variable naming
DEFINE BI_1 RUN_EXT_ENABLE

DEFINE BI_5 MANUAL_OFF
DEFINE BI_6 MANUAL_LOW
DEFINE BI_7 MANUAL_MED
DEFINE BI_8 MANUAL_HIGH

DEFINE AI_2 TEMPERATURE

DEFINE BO_1 COOLING_VALVE
DEFINE BO_2 HEATING_VALVE
DEFINE BO_3 FAN_LOW_SPEED
DEFINE BO_4 FAN_MED_SPEED
DEFINE BO_5 FAN_HIGH_SPEED

DEFINE ADB_1 FAN_SPEED_COMMAND // 0 = off, 1 = Low, 2 = Med, 3 = High

DEFINE ADF_1 TEMP_SET_POINT      // °C
DEFINE ADF_2 TEMP_PROP_BAND     // °C

TEMP_SET_POINT = 22.50 INITIALIZE // °C
TEMP_PROP_BAND = 1.50 INITIALIZE // °C

DEFINE 0 FAN_OFF
DEFINE 1 FAN_LOW
DEFINE 2 FAN_MED
DEFINE 3 FAN_HIGH
    
```

10. At the bottom of the script is the main body of the program as shown.

```

script_1 - Notepad
File Edit Format View Help

// Program start

// The fan speed command outputs are controlled by multistate setpoint: FAN_SPEED_COMMAND
IF RUN_MODE = OFF THEN \
    COOLING_VALVE = OFF ALSO \
    HEATING_VALVE = OFF ALSO \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF ALSO \
END
// Nothing else to do until a RUN command is set to ON

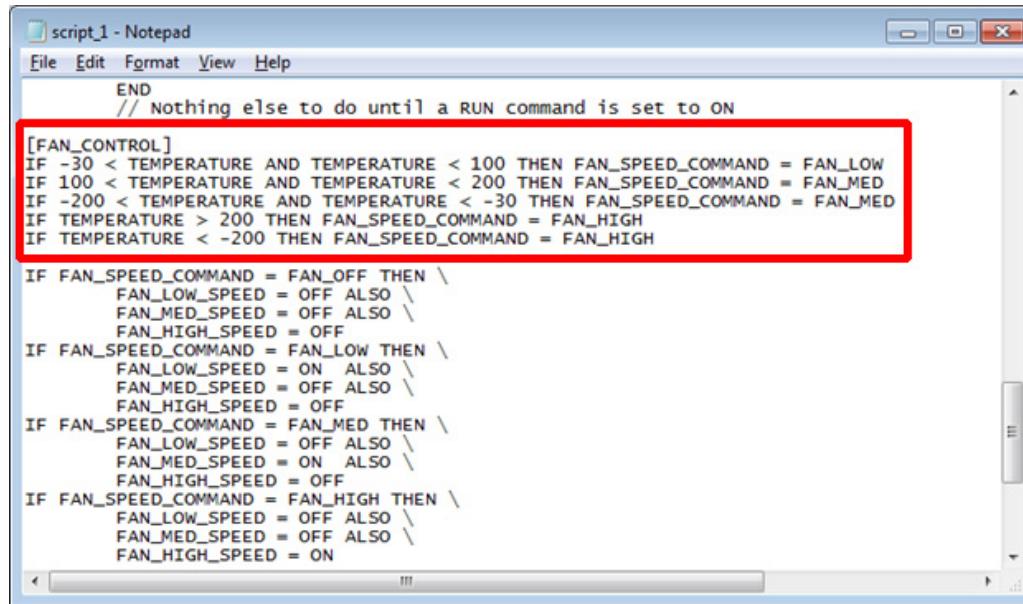
[FAN_CONTROL]
IF FAN_SPEED_COMMAND = FAN_OFF THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF \
IF FAN_SPEED_COMMAND = FAN_LOW THEN \
    FAN_LOW_SPEED = ON ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF \
IF FAN_SPEED_COMMAND = FAN_MED THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = ON ALSO \
    FAN_HIGH_SPEED = OFF \
IF FAN_SPEED_COMMAND = FAN_HIGH THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = ON \
// The valves will only operate if the fan is running

[VALVE_CONTROL]
// Cooling valve control
IF TEMP_SET_POINT > TEMP_PROP_BAND + TEMPERATURE AND FAN_SPEED_COMMAND != OFF THEN COOLING_VALVE = ON
IF TEMP_SET_POINT < TEMPERATURE OR FAN_SPEED_COMMAND = 0 THEN COOLING_VALVE = OFF

// Heating valve control
IF TEMP_SET_POINT IS < TEMP_PROP_BAND - TEMPERATURE AND FAN_SPEED_COMMAND IS NOT = OF THEN HEATING_VALVE = ON
IF TEMP_SET_POINT IS > TEMPERATURE OR FAN_SPEED_COMMAND = 0 THEN HEATING_VALVE = OFF
END
    
```

11. If no changes are made to the program that is created by the wizard then the fan speed would not be affected by the temperature. It could only be controlled by the user. Our goal is to make the fan speed be a function of how far the temperature is from the set point. As highlighted, if statements are used to set the FAN_SPEED_COMMAND to LOW, MED or HIGH. The numbers used to make the ranges for each speed are subjective. These are

the numbers used for this particular LEARN module but any numbers you like can be used.



```

script_1 - Notepad
File Edit Format View Help
END
// Nothing else to do until a RUN command is set to ON

[FAN_CONTROL]
IF -30 < TEMPERATURE AND TEMPERATURE < 100 THEN FAN_SPEED_COMMAND = FAN_LOW
IF 100 < TEMPERATURE AND TEMPERATURE < 200 THEN FAN_SPEED_COMMAND = FAN_MED
IF -200 < TEMPERATURE AND TEMPERATURE < -30 THEN FAN_SPEED_COMMAND = FAN_MED
IF TEMPERATURE > 200 THEN FAN_SPEED_COMMAND = FAN_HIGH
IF TEMPERATURE < -200 THEN FAN_SPEED_COMMAND = FAN_HIGH

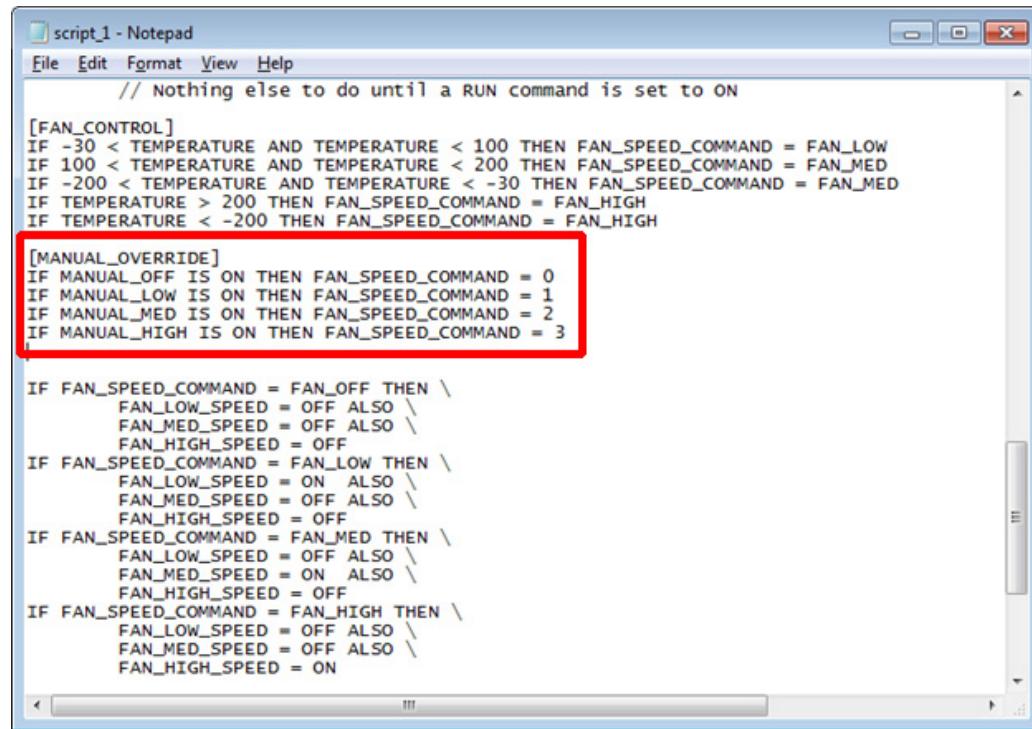
IF FAN_SPEED_COMMAND = FAN_OFF THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_LOW THEN \
    FAN_LOW_SPEED = ON ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_MED THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = ON ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_HIGH THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = ON

```

-200°C	-30°C	100°C	200°C	
FAN_HIGH_SPEED	FAN_MED_SPEED	FAN_LOW_SPEED	FAN_MED_SPEED	FAN_HIGH_SPEED
Set Point				
22°C				

12. The next task is to add manual override of the fan speed in order to allow the fan speed to be determined regardless of the temperature. As was defined in step 9, binary inputs 5-8 control the manual override variables. These set of if statements comes after the

previous group of statements and thus overrides whatever fan speed was determined above.



```

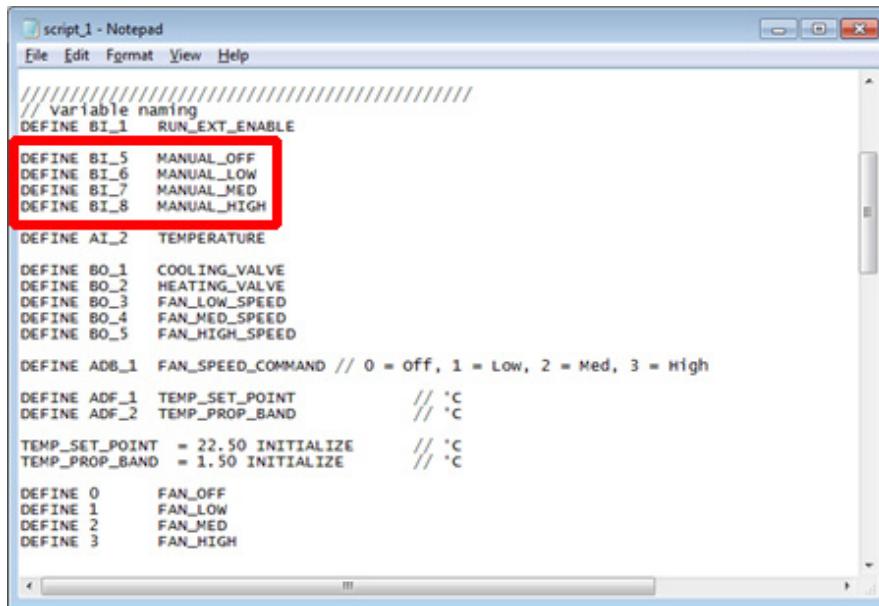
script_1 - Notepad
File Edit Format View Help
// Nothing else to do until a RUN command is set to ON

[FAN_CONTROL]
IF -30 < TEMPERATURE AND TEMPERATURE < 100 THEN FAN_SPEED_COMMAND = FAN_LOW
IF 100 < TEMPERATURE AND TEMPERATURE < 200 THEN FAN_SPEED_COMMAND = FAN_MED
IF -200 < TEMPERATURE AND TEMPERATURE < -30 THEN FAN_SPEED_COMMAND = FAN_MED
IF TEMPERATURE > 200 THEN FAN_SPEED_COMMAND = FAN_HIGH
IF TEMPERATURE < -200 THEN FAN_SPEED_COMMAND = FAN_HIGH

[MANUAL_OVERRIDE]
IF MANUAL_OFF IS ON THEN FAN_SPEED_COMMAND = 0
IF MANUAL_LOW IS ON THEN FAN_SPEED_COMMAND = 1
IF MANUAL_MED IS ON THEN FAN_SPEED_COMMAND = 2
IF MANUAL_HIGH IS ON THEN FAN_SPEED_COMMAND = 3

IF FAN_SPEED_COMMAND = FAN_OFF THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_LOW THEN \
    FAN_LOW_SPEED = ON ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_MED THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = ON ALSO \
    FAN_HIGH_SPEED = OFF
IF FAN_SPEED_COMMAND = FAN_HIGH THEN \
    FAN_LOW_SPEED = OFF ALSO \
    FAN_MED_SPEED = OFF ALSO \
    FAN_HIGH_SPEED = ON

```



```

script_1 - Notepad
File Edit Format View Help
////////////////////////////////////////////////////////////////
/// Variable naming
DEFINE BI_1    RUN_EXT_ENABLE

DEFINE BI_5    MANUAL_OFF
DEFINE BI_6    MANUAL_LOW
DEFINE BI_7    MANUAL_MED
DEFINE BI_8    MANUAL_HIGH

DEFINE AI_2    TEMPERATURE

DEFINE BO_1    COOLING_VALVE
DEFINE BO_2    HEATING_VALVE
DEFINE BO_3    FAN_LOW_SPEED
DEFINE BO_4    FAN_MED_SPEED
DEFINE BO_5    FAN_HIGH_SPEED

DEFINE ADB_1   FAN_SPEED_COMMAND // 0 = OFF, 1 = Low, 2 = Med, 3 = High
DEFINE ADF_1   TEMP_SET_POINT      // °C
DEFINE ADF_2   TEMP_PROP_BAND     // °C

TEMP_SET_POINT = 22.50 INITIALIZE
TEMP_PROP_BAND = 1.50 INITIALIZE

DEFINE 0    FAN_OFF
DEFINE 1    FAN_LOW
DEFINE 2    FAN_MED
DEFINE 3    FAN_HIGH

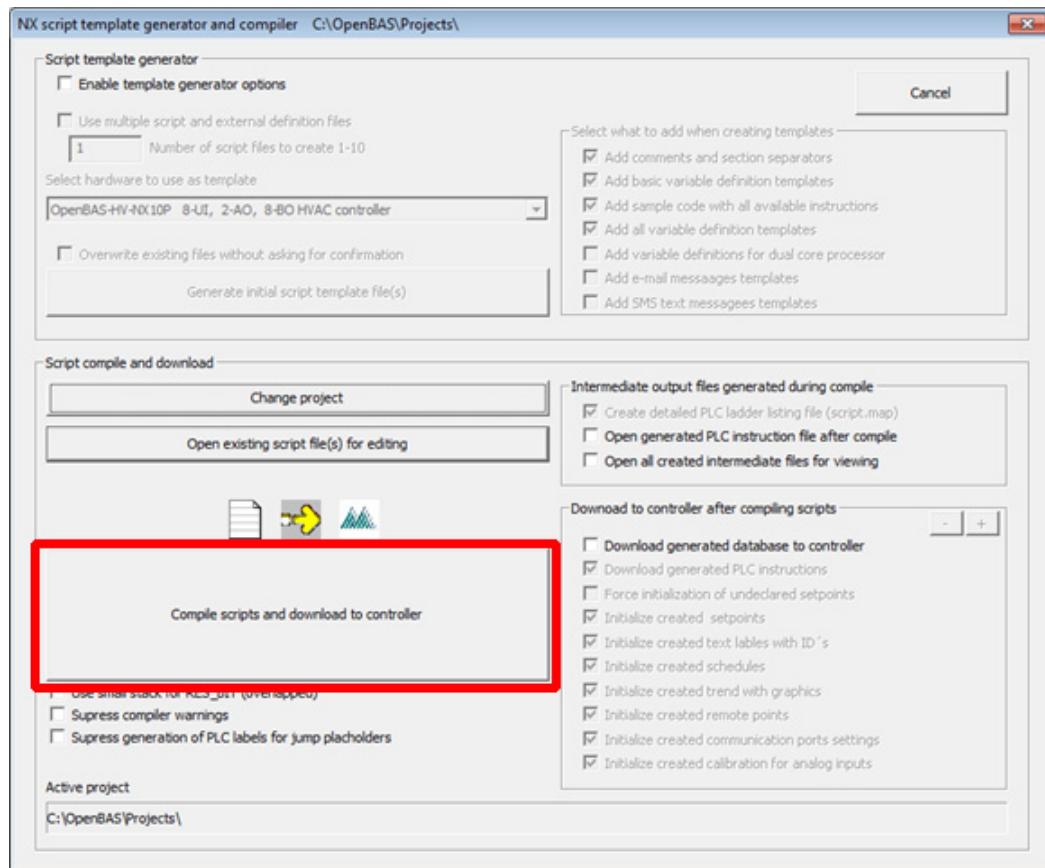
```

13. The main program of the script file should now look like the image below. Next, save and close the file.

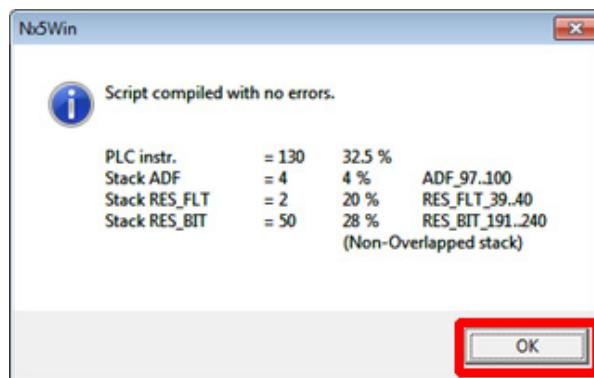
The screenshot shows a Windows Notepad window with the title "testScript - Notepad". The window contains a script written in a programming language, likely Python or a similar language, used for controlling fan and coil applications. The script includes comments, conditional statements (IF, THEN, END), and sections for fan control, manual override, valve control, and heating/cooling logic. The code is well-organized with proper indentation and syntax highlighting.

```
//////////  
// Program start  
  
// The fan speed command outputs are controlled by multistate setpoint: FAN_SPEED_COMM.  
IF RUN_MODE = OFF THEN \  
    COOLING_VALVE = OFF ALSO \  
    HEATING_VALVE = OFF ALSO \  
    FAN_LOW_SPEED = OFF ALSO \  
    FAN_MED_SPEED = OFF ALSO \  
    FAN_HIGH_SPEED = OFF ALSO \  
END  
// Nothing else to do until a RUN command is set to ON  
  
[FAN_CONTROL]  
IF -30 < TEMPERATURE AND TEMPERATURE < 100 THEN FAN_SPEED_COMMAND = FAN_LOW  
IF 100 < TEMPERATURE AND TEMPERATURE < 200 THEN FAN_SPEED_COMMAND = FAN_MED  
IF -200 < TEMPERATURE AND TEMPERATURE < -30 THEN FAN_SPEED_COMMAND = FAN_MED  
IF TEMPERATURE > 200 THEN FAN_SPEED_COMMAND = FAN_HIGH  
IF TEMPERATURE < -200 THEN FAN_SPEED_COMMAND = FAN_HIGH  
  
[MANUAL_OVERRIDE]  
IF MANUAL_OFF IS ON THEN FAN_SPEED_COMMAND = 0  
IF MANUAL_LOW IS ON THEN FAN_SPEED_COMMAND = 1  
IF MANUAL_MED IS ON THEN FAN_SPEED_COMMAND = 2  
IF MANUAL_HIGH IS ON THEN FAN_SPEED_COMMAND = 3  
  
IF FAN_SPEED_COMMAND = FAN_OFF THEN \  
    FAN_LOW_SPEED = OFF ALSO \  
    FAN_MED_SPEED = OFF ALSO \  
    FAN_HIGH_SPEED = OFF \  
IF FAN_SPEED_COMMAND = FAN_LOW THEN \  
    FAN_LOW_SPEED = ON ALSO \  
    FAN_MED_SPEED = OFF ALSO \  
    FAN_HIGH_SPEED = OFF \  
IF FAN_SPEED_COMMAND = FAN_MED THEN \  
    FAN_LOW_SPEED = OFF ALSO \  
    FAN_MED_SPEED = ON ALSO \  
    FAN_HIGH_SPEED = OFF \  
IF FAN_SPEED_COMMAND = FAN_HIGH THEN \  
    FAN_LOW_SPEED = OFF ALSO \  
    FAN_MED_SPEED = OFF ALSO \  
    FAN_HIGH_SPEED = ON  
  
// The valves will only operate if the fan is running  
  
[VALVE_CONTROL]  
  
// Cooling valve control]  
IF TEMP_SET_POINT < TEMP_PROP_BAND + TEMPERATURE AND FAN_SPEED_COMMAND != OFF THEN COOLING_VALVE = ON  
IF TEMP_SET_POINT > TEMPERATURE OR FAN_SPEED_COMMAND = 0 THEN COOLING_VALVE = OFF  
  
// Heating valve control]  
IF TEMP_SET_POINT IS > TEMP_PROP_BAND + TEMPERATURE AND FAN_SPEED_COMMAND IS NOT = OF  
IF TEMP_SET_POINT IS < TEMPERATURE OR FAN_SPEED_COMMAND = 0 THEN HEATING_VALVE = OFF  
  
END
```

14. Now back to the OpenBAS software. Here you have a number of options, most notably the “Compile scripts and download to controller” button. This is effectively uploading to the programmer. Press this button.

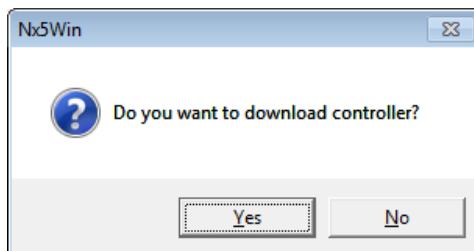


15. A pop-up will appear alerting the user to any errors in the script if they are present. In this case there are none and press “OK” to advance.

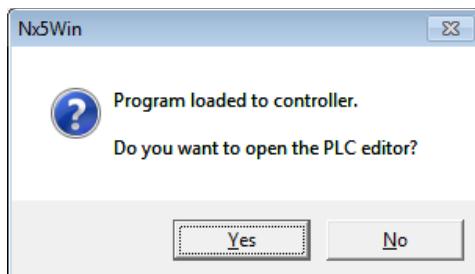


16. The program will ask you to download to the controller. Pressing “Yes” will kickstart a series of events as the software downloads to the controller. A number of dialog boxes

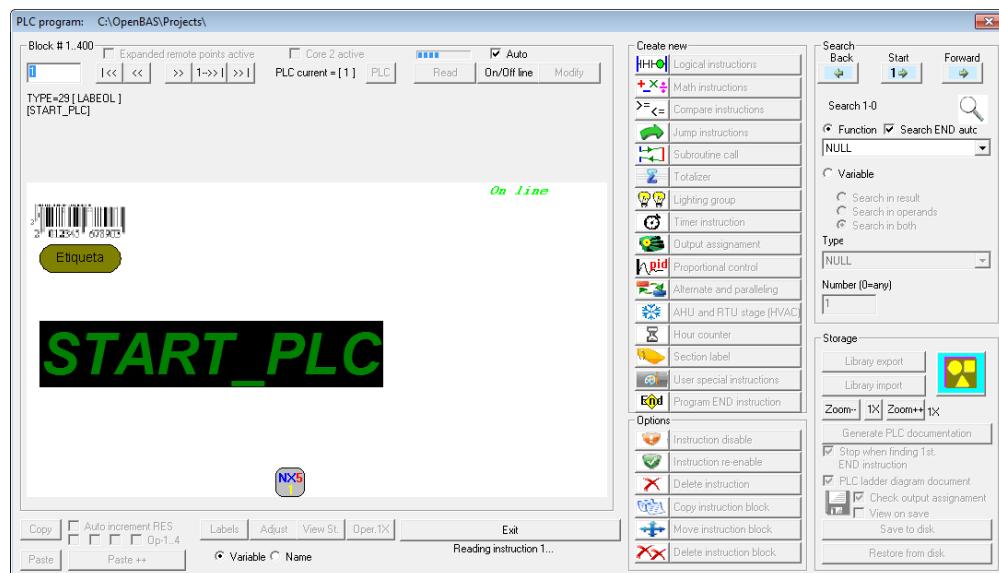
will pop-up and disappear as the parameters are filled. Let the software do its work and just wait for it to conclude.



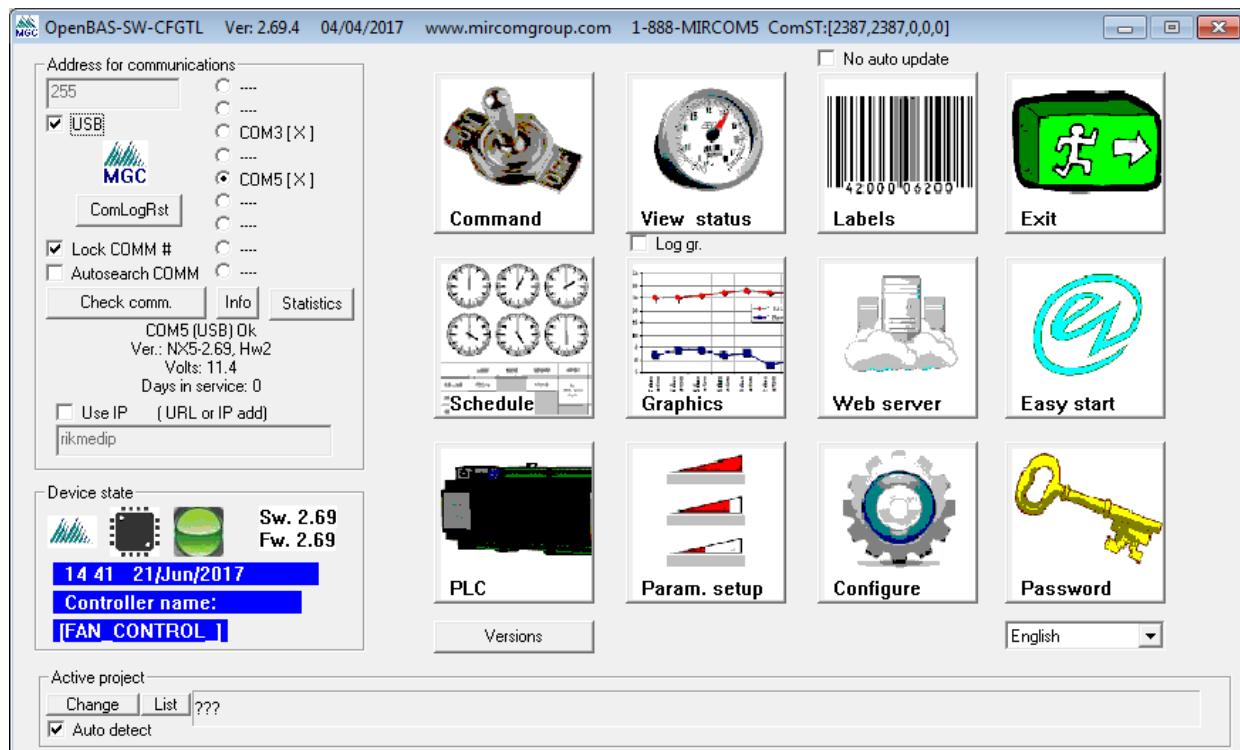
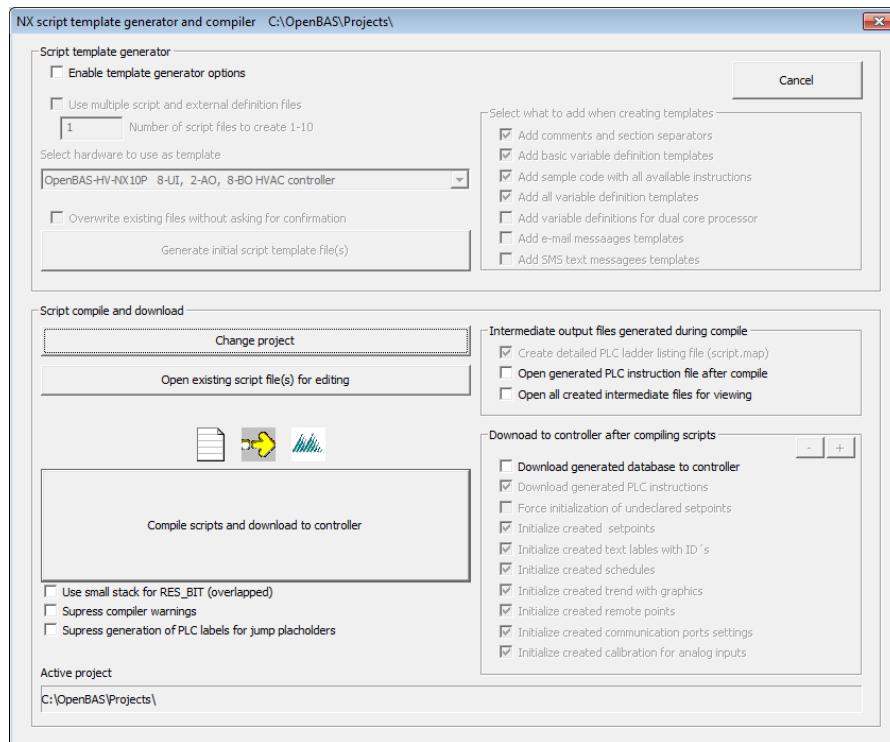
17. When the software is done uploading, the below dialog box will appear giving you the option to open the PLC editor to view the PLC logic that the script was turned into. The user can decide either way.



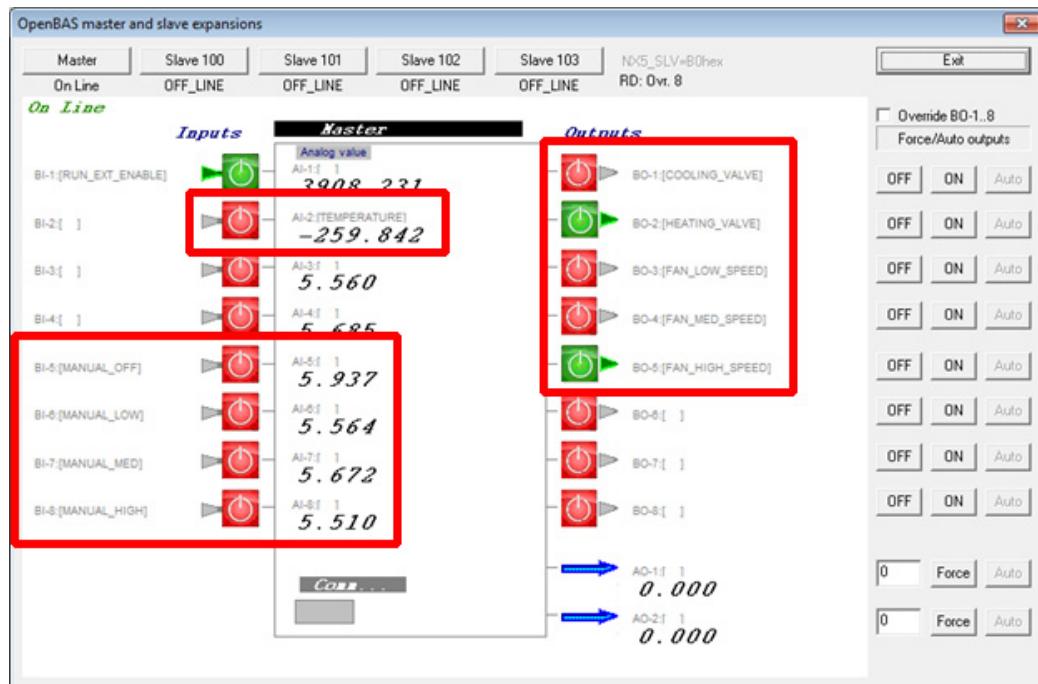
18. If you chose to view the PLC, the PLC window will appear and the user can press the “>>” button to scroll through the blocks. Simply close the window or press the “Exit” button whenever you choose.



19. Exiting the PLC returns you to the below window. Simply close every other window open until you return to the main OpenBAS software window and the programming is complete.

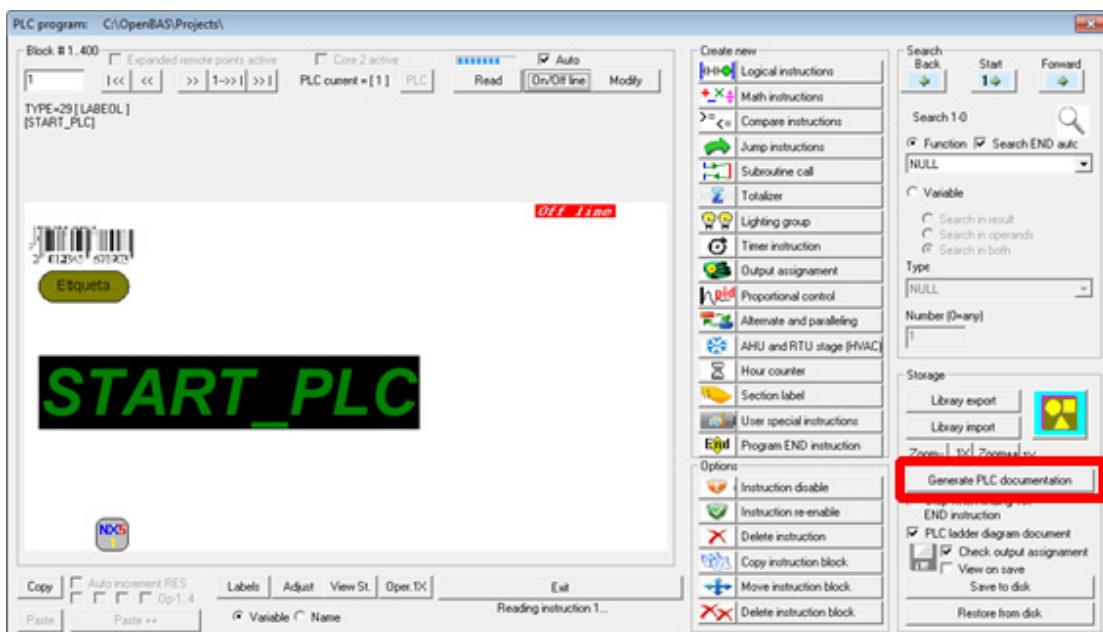


20. The best way to view the workings for the program is to go to the “Command” button on the main OpenBAS screen. This opens the below window that shows the inputs, outputs and their labels for the program allowing the user to watching all the values change with a change in input.

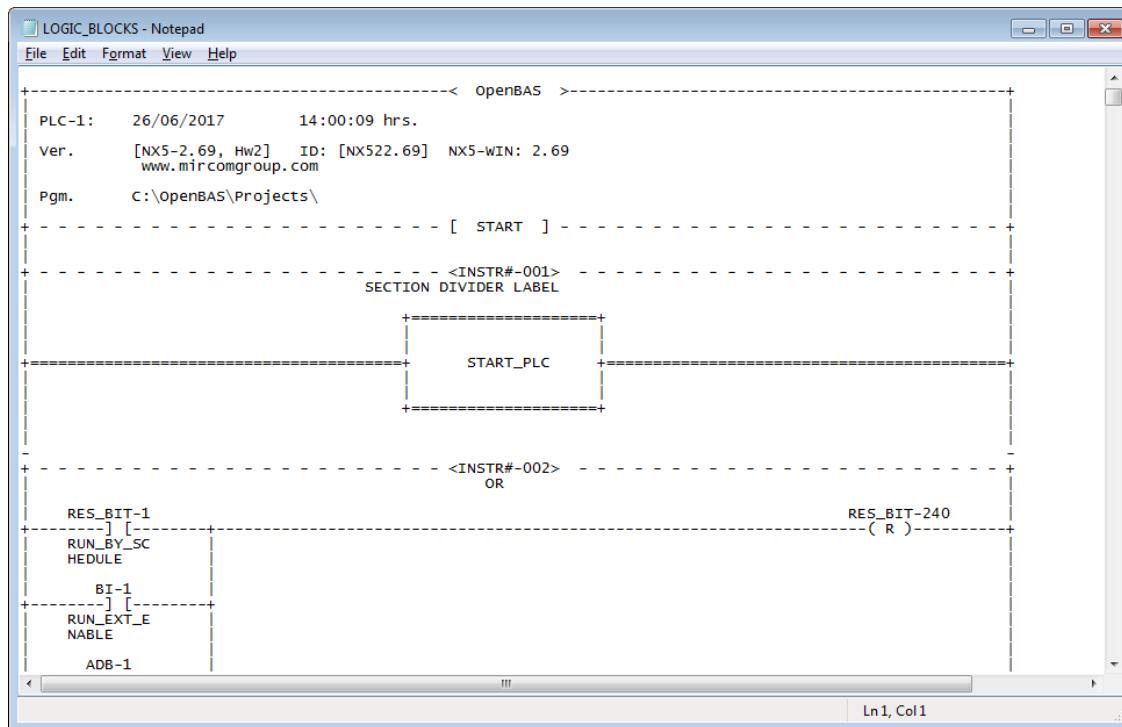


7.2 PLC Ladder Diagram

The PLC for this program is 131 blocks long and thus too long to reasonably include in this document. In order to view the Ladder diagram of this program go to the PLC editor in the software and press the “On/Off line” button to enable the “Generate PLC documentation” button



Pressing the “Generate PLC documentation” button opens a text file that contains the PLC Ladder diagram for the program as well as other information about the PLC.



```

LOGIC_BLOCKS - Notepad
File Edit Format View Help
+-----< OpenBAS >-----
PLC-1: 26/06/2017 14:00:09 hrs.
Ver. [NX5-2.69, Hw2] ID: [NX522.69] NX5-WIN: 2.69
www.mircomgroup.com
Pgm. C:\OpenBAS\Projects\
+----- [ START ] -----
+----- <INSTR#-001> -----
SECTION DIVIDER LABEL
+=====+
|          |
|          START_PLC          |
|          |          |
+=====+
+----- <INSTR#-002> -----
OR
RES_BIT-1
[ RUN_BY_SCHEDULE ]
[ RUN_EXT_ENABLE ]
ADB-1
RES_BIT-240
( R )

```



CANADA - Main Office
25 Interchange Way
Vaughan, ON L4K 5W3
Tel: (905) 660-4655
(888) 660-4655
Fax: (905) 660-4113

U.S.A
4575 Witmer Industrial Estates
Niagara Falls, NY 14305
Tel: (905) 660-4655
(888) 660-4655
Fax: (905) 660-4113

© Mircom 2018
Printed in Canada
Subject to change without prior notice
www.mircom.com