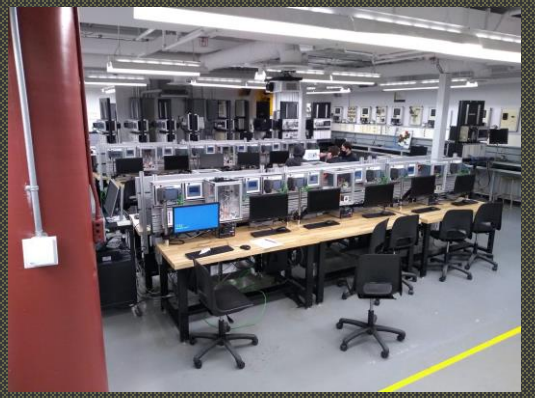
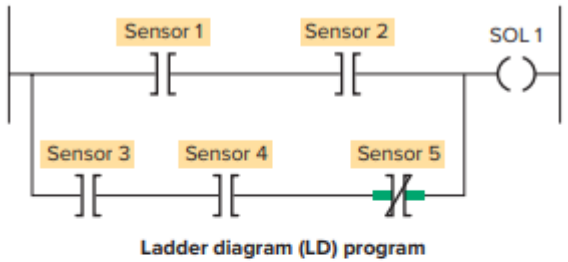


PROGRAMMABLE LOGIC CONTROLLERS
MENG 3500

HUMBER

STRUCTURED TEXT PROGRAMMING

- The Structured Text (ST) language is a high-level language whose syntax is like Pascal or C Basic languages.
- There are situations where employing ST programming offers numerous advantages compared to Ladder Logic and Function Block, particularly in scenarios that require real-time mathematical calculations.
- The practice of developing entirely in ST is lowered upon by most automation professionals. Often, ST is used by an engineer who is fresh out of school and is accustomed to modern, text-based programming languages.

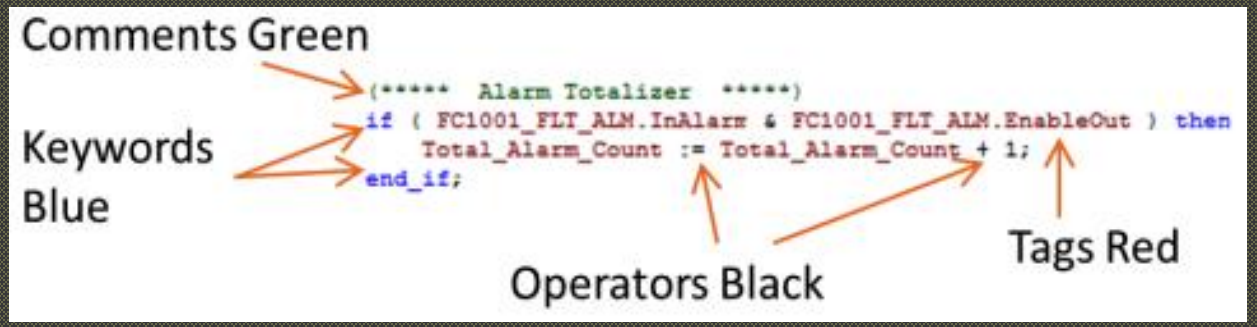


Ladder diagram (LD) program

```
IF Sensor_1 AND Sensor_2 THEN
        SOL_1 := 1;
ELSEIF Sensor_3 AND Sensor_4 AND NOT Sensor_5 THEN
        SOL_1 := 1;
END_IF;
```

Structured text (ST) program

Figure 5-18 PLC ladder and equivalent structured text program.



Austin Scott Learning RSLogix 5000 Programming

Frank D. Petruzella Programmable Logic Controllers 6th edition

HUMBER

STRUCTURED TEXT PROGRAMMING

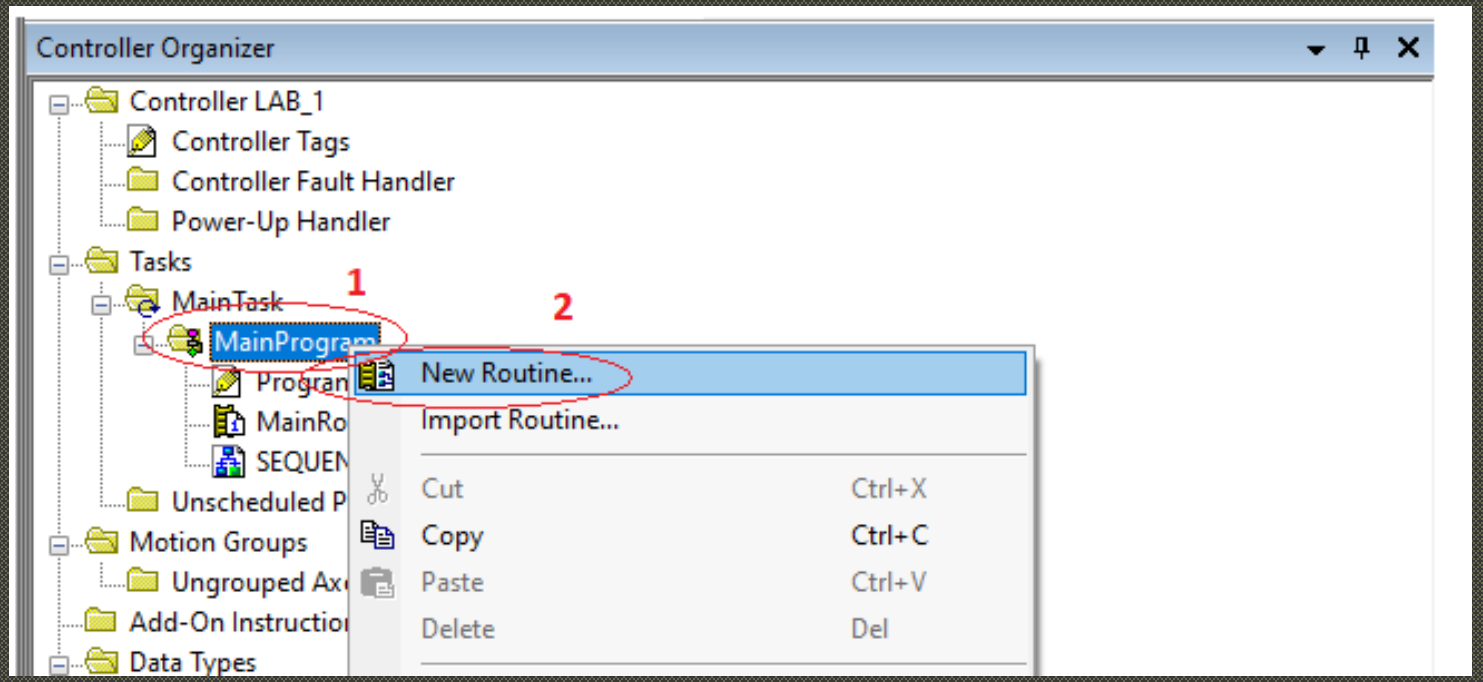
- The program below has two variables (tags), named Temp and Flow, and two outputs, named Pump and Green_Light. Temp, Flow, Pump, and Green_Light are all tags in a ControlLogix (CLX) project. This routine controls the pump, flow, and a green_light.
- The routine uses IF statements to make decisions. The first portion of logic compares the value of the Temp tag.
  - If Temp is greater than or equal to 100 and less than 200, then Pump is turned on, the Flow variable (tag) is set to a value of 45, and Green_Light is turned on.
  - Else if (ELSIF) Temp is less than 100, Pump is turned off, Flow is set to 20, and Green_Light is turned off.
  - If neither of those conditions is true, then the Else is true and Alarm_Light is turned on.

```
IF Temp >= 100 & Temp < 200 THEN
   Pump := 1; Flow := 45; Green_Light := 1;
ELSIF Temp < 100 THEN
   Pump := 0; Flow := 20; Green_Light := 0;
ELSE
   Alarm_Light := 1;
END_IF;
```

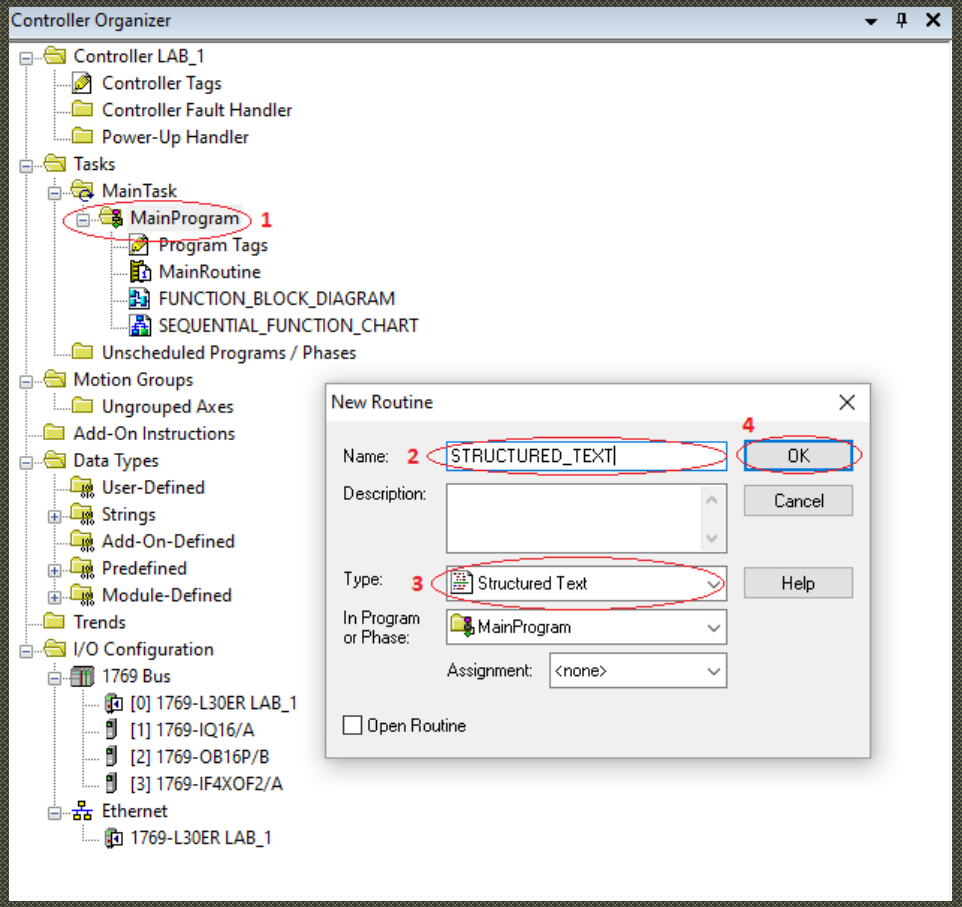
STRUCTURED TEXT PROGRAMMING

The logic created in a routine different from the main must be called from the Main Routine with a JSR (Jump to Subroutine) instruction. The main routine must be written in Ladder Logic.

First you must create the Subroutine for the Structured Text program and then call that subroutine from the Main routine.

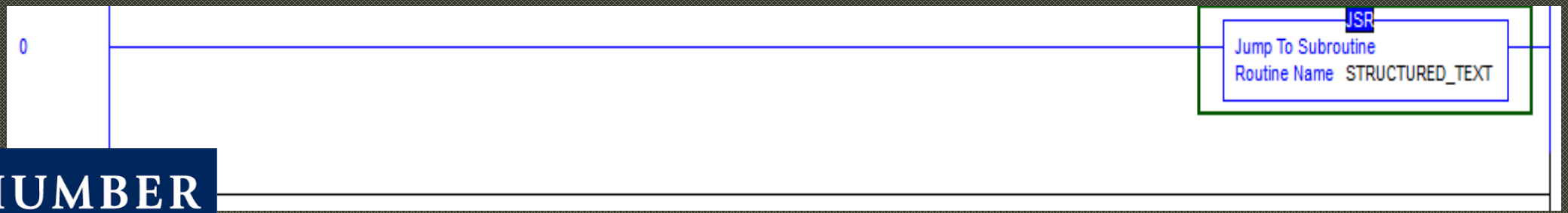


STRUCTURED TEXT PROGRAMMING



John Stenerson Programmable Logic Controllers with ControlLogix

CALL THE STRUCTURED_TEXT SUBROUTINE WITH JSR FROM THE MAIN ROUTINE



HUMBER

STRUCTURED TEXT PROGRAMMING

Assignment statements are used to assign a value to a tag.

Tag : = value or math expression;
Temp : = 115;

- All lines in a ST program must be terminated with a semicolon.
- A line may contain multiple statements separated by semicolons.

- The ‘ : = ‘ is the assignment operator.
- Tag on the left represents a variable, which can be Boolean, Single Integer, Integer, Double Integer, or Real.
- The expression on the right side of the equation is used to represent the value that will be assigned to the tag.

```
Temp := 212;
Temp := Var_1;
Var1 := Var_2 * 2;
```

Operator type	Operator	Maintains the last assigned value after a power loss
Retentive	:=	True
Non-retentive	[:=]	False

John Stenerson Programmable Logic Controllers with ControlLogix

STRUCTURED TEXT PROGRAMMING

CONDITIONAL STATEMENTS

Conditional statements allow selected statements to be executed when certain conditions are true.

Some of the conditional statements are:

IF THEN	If specific conditions are true
FOR DO	A specific number of times
WHILE DO	As long as a condition is true
REPEAT UNTIL	Until a condition is true
CASE OF	On the basis of a number

```
IF Sensor_1 THEN
Motor_1 : = 1;
Temp : = 150;
END_IF
```

```
IF Motor_On THEN
Red_Light : = 1;
ELSE
Green_Light : = 1;
END_IF;
```

```
IF Sensor_1 & Sensor_2 THEN
Pump : = 1; Heat_Coil : = 0;
ELSIF Sensor_3
THEN
Alarm : = 1;
END_IF;
```

```
IF Temp > = 100 & Temp < = 200 THEN
Pump : = 1; Flow : = 45; Green_Light : = 1;
ELSIF Temp < 100 & Temp > 50 THEN
Pump : = 0; Flow : = 20; Green_Light : = 0;
ELSE
Pump : = 0; Flow : = 0; Green_Light : = 0;
END_IF;
```

STRUCTURED TEXT PROGRAMMING

CONDITIONAL STATEMENTS

Conditional statements allow selected statements to be executed when certain conditions are true.

Some of the conditional statements are:

IF THEN	If specific conditions are true
FOR DO	A specific number of times
WHILE DO	As long as a condition is true
REPEAT UNTIL	Until a condition is true
CASE OF	On the basis of a number

```
FOR X : = 0 to 9 by 1
DO Temp[X] : = 99;
END_FOR;
```

```
WHILE ((CNT > 50) & (Temp < 200))
DO CNT : = CNT + 3;
END_WHILE;
```

```
REPEAT
CNT : = CNT + 2;
UNTIL ((CNT > 34) OR (Temp > 92))
END_REPEAT
```

During loop execution, the controller exclusively processes loop-related statements until completion. If the loop duration exceeds the task's watchdog timer (default 500 ms), a major fault occurs.
To address this, consider using alternative conditional statements like an IF-THEN statement if needed.

STRUCTURED TEXT PROGRAMMING

CONDITIONAL STATEMENTS

Conditional statements allow selected statements to be executed when certain conditions are true.

Some of the conditional statements are:

IF THEN	If specific conditions are true
FOR DO	A specific number of times
WHILE DO	As long as a condition is true
REPEAT UNTIL	Until a condition is true
CASE OF	On the basis of a number

```
CASE VAR_1 OF
1: Temp_1 : = 85; Pump_1 : = 1;
2: Temp_1 : = 105; Pump_1 : = 1;
3,4: Temp_1 : = 110; Pump_1 : = 1;
5..8: Temp_1 : = 115; Pump_1 : = 1;
9,11..15: Temp_1 : = 120; Pump_1 : = 1;
ELSE Alarm : = 1;
END_CASE;
```

- The value of tag Var_1 is used to decide which part of the logic to execute.
- Note the use of the ELSE at the end. If Var_1 has a value that is different from any of the cases specified, the ELSE portion of code will be run.

STRUCTURED TEXT PROGRAMMING

ARITHMETIC OPERATORS

Instruction	Operator	Optimal Data Type
Add	+	DINT, REAL
Subtract	–	DINT, REAL
Multiply	*	DINT, REAL
Exponent (X to the power of Y)	**	DINT, REAL
Divide	/	DINT, REAL
Modulo	MOD	DINT, REAL

All the standard arithmetic operators are available in ST programming.

Temp : = Var1 + 20; (* The tag Temp is assigned a value equal to Var1 plus 20. *)

RPM : = Speed / 60; (* RPM is assigned a value equal to Speed divided by 60 *)

Total_Cans : = Cases *12; (* Total_Cans is assigned a value equal to Cases multiplied by 12 *)

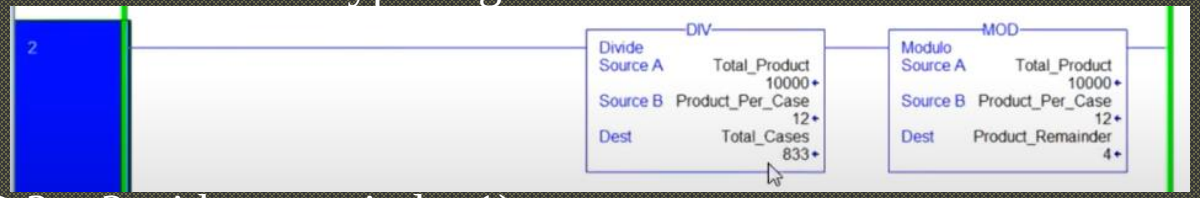
- The programmer must use correct Data Types when performing math operations: DINT, INT, REAL.

  Example**:** Answer : = 5/2;

  If the tag Answer is DINT type, the answer would be 2. Need to use REAL type to get 2.5

- Modulo can be used to find the remainder of a division.

  Example: Answer : = 5 MOD 2; (* Answer = 1 *) (5 MOD 2 = 2 with a remainder 1)



STRUCTURED TEXT PROGRAMMING

ARITHMETIC OPERATORS

Instruction	Operator	Optimal Data Type
Add	+	DINT, REAL
Subtract	–	DINT, REAL
Multiply	*	DINT, REAL
Exponent (X to the power of Y)	**	DINT, REAL
Divide	/	DINT, REAL
Modulo	MOD	DINT, REAL

For	Function	Optimal Data Type
Absolute value	ABS(numeric expression)	DINT, REAL
Arc cosine	ACOS(numeric expression)	REAL
Arc sine	ASIN(numeric expression)	REAL
Arc tangent	ATAN(numeric expression)	REAL
Cosine	COS(numeric expression)	REAL
Radians to degrees	DEG(numeric expression)	DINT, REAL
Natural log	LN(numeric expression)	REAL
Log base 10	LOG(numeric expression)	REAL
Degrees to radians	RAD(numeric expression)	DINT, REAL
Sine	SIN(numeric expression)	REAL
Square root	SQRT(numeric expression)	DINT, REAL
Tangent	TAN(numeric expression)	REAL
Truncate	TRUNC(numeric expression)	DINT, REAL

Numeric expression: An expression that calculates an integer or floating-point value.

For example, tag1+5

The TRUNC function typically takes a REAL input and returns a DINT.

STRUCTURED TEXT PROGRAMMING

Relational operators compare values or strings, yielding a BOOL result (1 for true, 0 for false), extensively used in decision-making.

RELATIONAL OPERATORS

Equal	=
Less than	<
Less than or equal	<=
Greater than	>
Greater than or equal	>=
Not equal	<>

```
IF (Input1 XOR Input2) THEN
    Output := TRUE;
ELSE
    Output := FALSE;
END_IF;
```

Inp1	Inp2	Inp1 XOR Inp2
1	1	0
1	0	1
0	1	1
0	0	0

LOGICAL OPERATORS

Logical AND	&, AND
Logical OR	OR
Logical exclusive OR	XOR
Logical Compliment	NOT

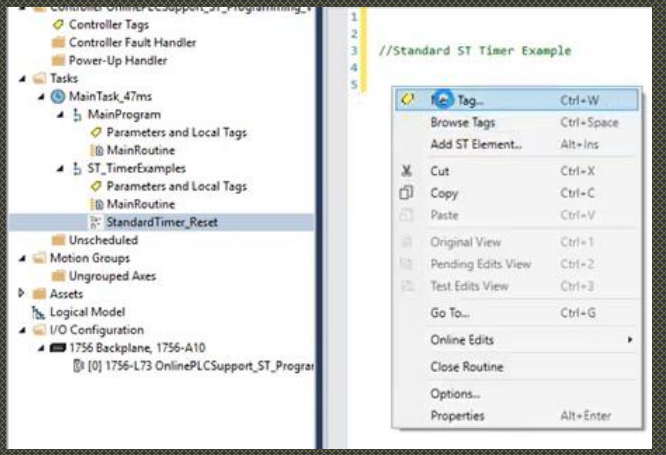
IF Sensor_1 THEN (* if Sensor_1=1 then *)

IF NOT Sensor_1 THEN (* if Sensor_1=0 then *)

IF Sensor_1 AND (TEMP < 150) THEN

STRUCTURED TEXT PROGRAMMING

TIMERS



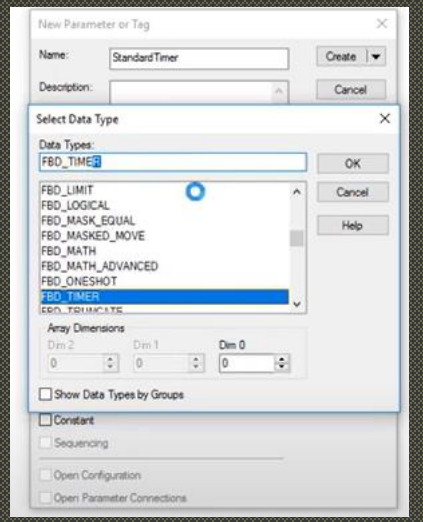
TONR (Timer On delay with Reset) can be used for time related tasks.

- Timer tag type must be FBD Timer type for ST.

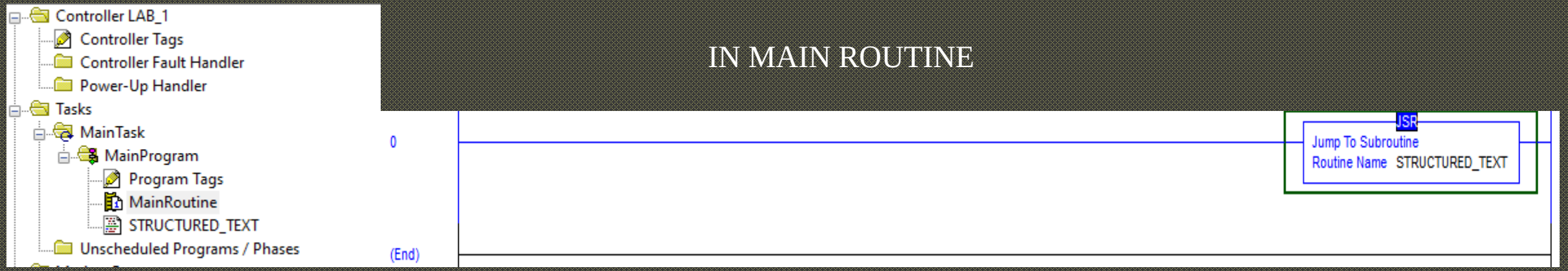
The example below shows how a timer can be used in ST programming.

- The first three lines are used to call the timer, set a PRE value, and enable the timer, i.e. start the timer timing.

```
TONR (Stop_Light_Timer);
Stop_Light_Timer.PRE : = 30000;
Stop_Light_Timer.TimerEnable : = 1;
IF (Stop_Light_Timer.ACC > 0 & Stop_Light_Timer.ACC < 10000) THEN
Green_1 : = 1;
ELSE
Green_1 : = 0;
END_IF
IF Stop_Light_Timer.ACC : = 30000 THEN
Stop_Light_Timer.Timer.Enable : = 0;
END_IF
```



STRUCTURED TEXT PROGRAMMING: DC MOTOR CONTROL



IN MAIN ROUTINE

IN STRUCTIRED TEXT SUBROUTINE

```
// START_PB-STOP_PB-RUN LOGIC FOR LIGHT AND DC MOTOR


RUN := (START_PB OR RUN) AND STOP_PB;

IF RUN THEN
    LIGHT := 1; MOTOR :=1;
ELSE
    LIGHT := 0; MOTOR := 0;
END_IF;
```

*Change the program:
Instead of LIGHT, use GREEN_LIGHT when MOTOR is ON and RED_LIGHT when the MOTOR is OFF.

HUMBER

STRUCTURED TEXT PROGRAMMING: SELF RESETTING TIMERS

```
//SELF RESETTING TIMER PROGRAM


TONR(TIMER_TAG);

    TIMER_TAG.PRE :=3000;

    IF NOT TIMER_TAG.DN THEN
    TIMER_TAG.TimerEnable := 1;

ELSE
    TIMER_TAG.TimerEnable := 0;
END_IF;


// SELF RESETTING TIMER PRGRAM 2

    TONR (TIMER_TAG2);

    TIMER_TAG2.PRE :=5000;

    TIMER_TAG2.TimerEnable := NOT TIMER_TAG2.DN;
```

STRUCTURED_TEXT MainProgram | FUNCTION_BLOCK_DIAGRAM MainProgram | SEQUENTIAL_FUNCTION_CHART MainProgram | MainRoutine MainProgram

Watch

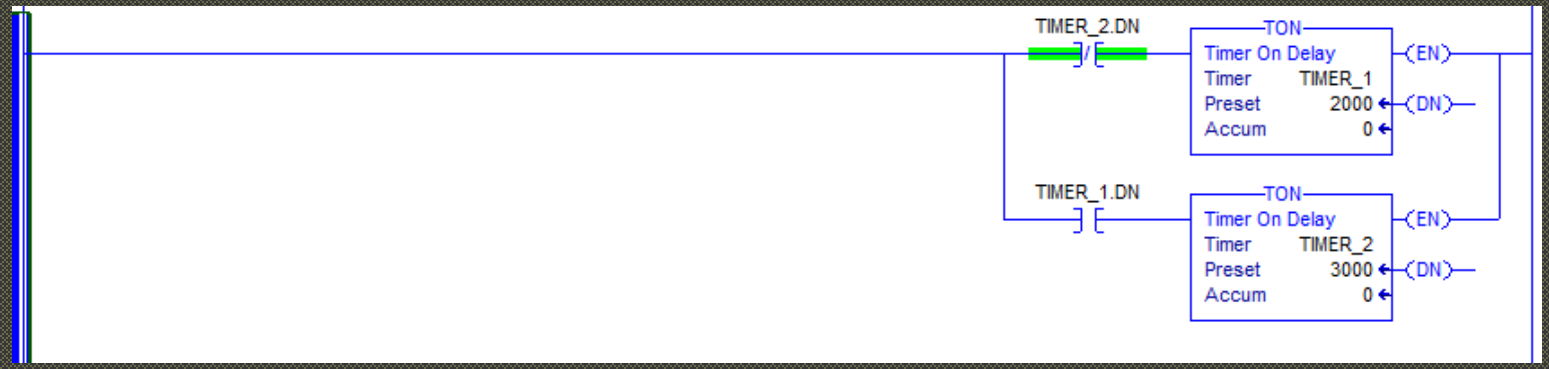
Current Routine

Name	Scope	Value	Force Mask	Description
TIMER_TAG	MainProgram	{...}	{...}	
TIMER_TAG.EnableIn	MainProgram	1		
TIMER_TAG.TimerEnable	MainProgram	1		
TIMER_TAG.PRE	MainProgram	3000		
TIMER_TAG.Reset	MainProgram	0		
TIMER_TAG.EnableOut	MainProgram	1		
TIMER_TAG.ACC	MainProgram	1110		
TIMER_TAG.EN	MainProgram	1		
TIMER_TAG.TT	MainProgram	1		
TIMER_TAG.DN	MainProgram	0		

STRUCTURED TEXT

HUMBER

STRUCTURED TEXT PROGRAMMING: CASCADING TIMERS



EQUIVALENT LADDER DIAGRAM

```
// CASCADING TIMERS

    TONR (TIMER_CAS1);
    TIMER_CAS1.PRE :=2000;

    TONR (TIMER_CAS_2);
    TIMER_CAS_2.PRE :=3000;

        TIMER_CAS1.TimerEnable := NOT TIMER_2.DN;
        TIMER_CAS_2.TimerEnable :=TIMER_1.DN;
```

STRUCTURED TEXT

Watch

Current Routine

Name	Scope	Value	Force Mask	Description
TIMER_CAS_2.PRE	MainProgram	3000		
TIMER_CAS_2.TimerEnable	MainProgram	0		
TIMER_CAS1	MainProgram	{...}	{...}	
TIMER_CAS1.EnableIn	MainProgram	1		
TIMER_CAS1.TimerEnable	MainProgram	1		
TIMER_CAS1.PRE	MainProgram	2000		
TIMER_CAS1.Reset	MainProgram	0		
TIMER_CAS1.EnableOut	MainProgram	1		
TIMER_CAS1.ACC	MainProgram	1032		
TIMER_CAS1.EN	MainProgram	1		
TIMER_CAS1.TT	MainProgram	1		
TIMER_CAS1.DN	MainProgram	0		

HUMBER

STRUCTURED TEXT PROGRAMMIMG: THREE LIGHTS CONTROL

```
(*THREE LIGHT PROGRAM: THE RED LIGHT STAYS FOR 8 SECONDS,
THE GREEN LIGHT STAYS FOR 8 SECONDS AND THE YELLOW STAYS ON FOR 4 SECONDS*)

RUN := (START_PB OR RUN) AND STOP_PB;

TONR (RED_TIMER);
RED_TIMER.PRE :=8000;

TONR (GREEN_TIMER);
GREEN_TIMER.PRE :=8000;


TONR (YELLOW_TIMER);
YELLOW_TIMER.PRE :=4000;

IF RUN THEN
    RED_TIMER.TimerEnable := NOT YELLOW_TIMER.DN;
    GREEN_TIMER.TimerEnable := RED_TIMER.DN;
    YELLOW_TIMER.TimerEnable := GREEN_TIMER.DN;
ELSE

    RED_TIMER.TimerEnable := 0;
    GREEN_TIMER.TimerEnable := 0;
    YELLOW_TIMER.TimerEnable := 0;
END_IF;

// FORMING THE OUTPUTS FOR THE LIGHTS
    RED_LIGHT := RED_TIMER.TT;
    GREEN_LIGHT := GREEN_TIMER.TT;
    YELLOW_LIGHT := YELLOW_TIMER.TT;
```

IN STRUCTURED TEXT SUBROUTINE

Change the program:

- Add pedestrian lights: WALK and DON'T WALK.
- DON'T WALK light stays ON solid during the first 4 seconds of the RED light and the last 2 seconds of the Yellow light.
- It flashes 0.5 ON and 0.5 OFF during second 4 seconds of the GREEN light and first 2 seconds of the Yellow light.
- WALK light stays ON solid during the first 4 seconds of the GREEN light.

STRUCTURED TEXT PROGRAMMING: IF-THEN-ELSE AND COMPARISON APPLICATIONS

IN STRUCTURED TEXT SUBROUTINE

```
// IF-THEN, COMPARISON INSTRUCTIONS APPLICATION
(* START_PB-STOP_PB-RUN ALLOWS TO OPERATE. 0-10V DC SIGNAL IS SENT TO THE ANALOG INPUT CHANNEL 0 (IN_CH0)
AND CONVERTED TO THE 0 - 32640 PLC UNITS BY THE ANALOG MODULE. WE SCALE THE 0 -32640 RANGE TO 0 - 1000
ENGINEERING UNITS USING THE FORMULA Y = K*IN_CH0. AS A RESULT, THE OUTPUT = 0.03*IN_CH0*)

OUTPUT := 0.03*IN_CH0;

RUN := (START_PB OR RUN) AND STOP_PB;
IF RUN THEN
IF OUTPUT >= 400 AND OUTPUT < 650 THEN
    LIGHT := 1; MOTOR := 1; ALARM_LIGHT := 0;
ELSIF OUTPUT < 400 THEN
    LIGHT := 1; MOTOR := 0; ALARM_LIGHT := 0;
ELSE
    LIGHT := 0; MOTOR := 0; ALARM_LIGHT := 1;
END_IF;
ELSE
LIGHT := 0; MOTOR := 0; ALARM_LIGHT := 0;
END_IF;
```

STRUCTURED TEXT PROGRAMMING: MOTOR CONTROL WITH A TIMER

IN STRUCTURED TEXT SUBROUTINE MOTOR WILL RUN FOR 10 SECONDS

```
TONR (TIMER);
TIMER.PRE := 10000;

RUN := (START_PB OR RUN) AND STOP_PB AND NOT TIMER.DN;
TIMER.TimerEnable := 1;
IF RUN THEN

    MOTOR := 1;
ELSE
    MOTOR := 0;
    TIMER.TimerEnable := 0;
END_IF;
```

STRUCTURED_TEXT
MainProgram

Watch

Current Routine

Name	Scope	Value	Force Mask	Description
TIMER	MainProgram	{...}	{...}	
TIMER.EnableIn	MainProgram	1		
TIMER.TimerEnable	MainProgram	1		
TIMER.PRE	MainProgram	10000		
TIMER.Reset	MainProgram	0		
TIMER.EnableOut	MainProgram	1		
TIMER.ACC	MainProgram	4078		
TIMER.EN	MainProgram	1		

STRUCTURED TEXT PROGRAMMING: COUNTERS AND ONE-SHOT RISE

```
// COUNTER OPERATION; ONE SHOT RISE INSTRUCTION IN ST


OSRI (ONE_SHOT); // ONE SHOT RISE INSTRUCTION SETTING
ONE_SHOT.InputBit := START_PB; // ONE SHOT RISE PULSE GENERATION FROM START_PB


CTUD(COUNTER); // COUNTER INSTRUCTION SETTING
IF ONE_SHOT.OutputBit THEN //ONE_SHOT.OUTPUTBIT IS THE GENERATED ONE-SCAN PULSE
COUNTER.Reset := 0;
COUNTER.CUEnable :=1;
ELSE
COUNTER.CUEnable :=0;
END_IF;
IF COUNTER.ACC >= 5 THEN
    COUNTER.Reset := 1; //RESETTING THE COUNTER WHEN THE ACCUMULATED VALUE IS EQUAL TO 5
END_IF;
```

STRUCTURED_TEXT
MainProgram

```
3  //One-Shot Example
4
5  //Control the one-shot based on the input status
6
7
```

New Tag... Ctrl+W
Browse Tags Ctrl+Space
Add ST Element... Alt+Ins
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Original View Ctrl+1
Pending Edits View Ctrl+2
Test Edits View Ctrl+3
Ctrl+G
Alt+Enter

Watch

Current Routine

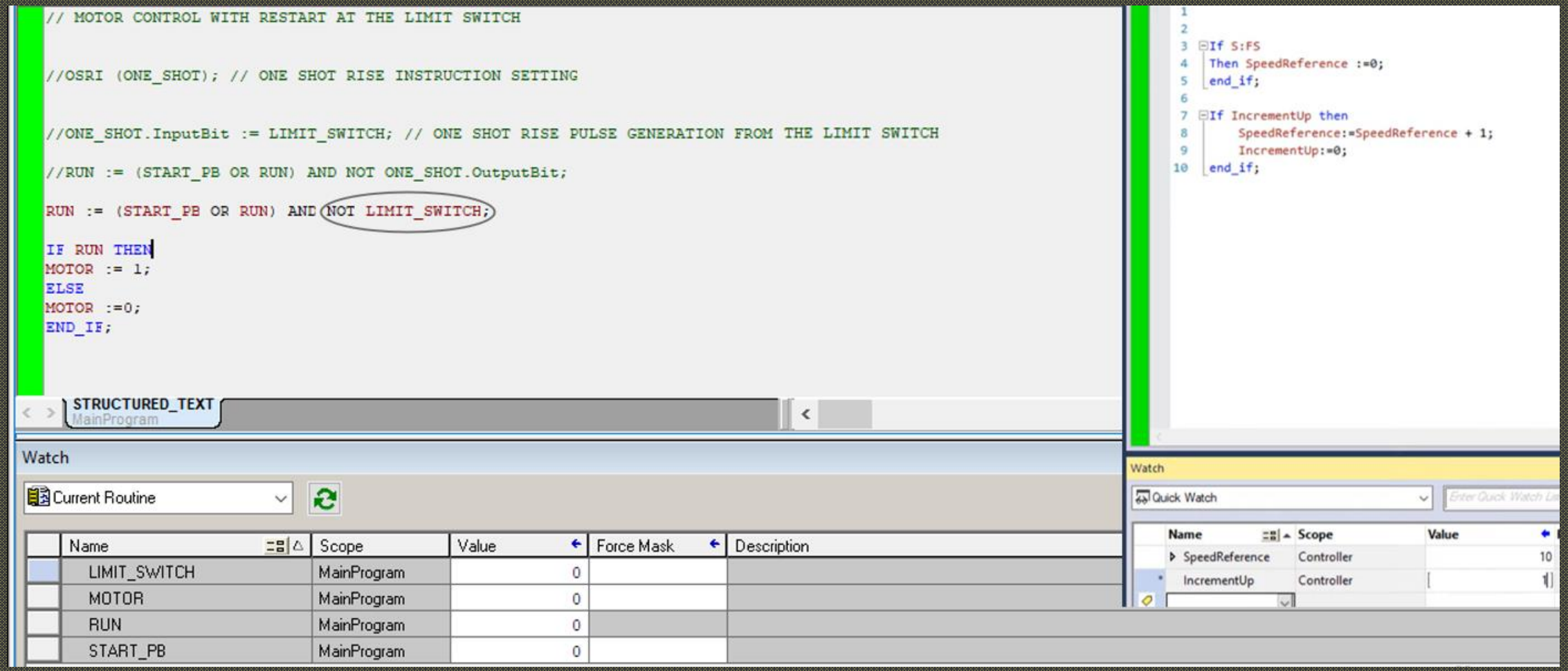
Name	Scope	Value	Force Mask	Description
COUNTER	MainProgram	{...}	{...}	
COUNTER.EnableIn	MainProgram	1		
COUNTER.CUEnable	MainProgram	0		
COUNTER.CDEnable	MainProgram	0		
COUNTER.PRE	MainProgram	0		
COUNTER.Reset	MainProgram	0		
COUNTER.EnableOut	MainProgram	1		
COUNTER.ACC	MainProgram	3		
COUNTER.CU	MainProgram	0		
COUNTER.CD	MainProgram	0		
COUNTER.DN	MainProgram	1		
COUNTER.OV	MainProgram	0		
COUNTER.UN	MainProgram	0		
COUNTER.ACC	MainProgram	3		
COUNTER.CUEnable	MainProgram	0		
COUNTER.Reset	MainProgram	0		
ONE_SHOT	MainProgram	{...}	{...}	
ONE_SHOT.InputBit	MainProgram	0		
ONE_SHOT.OutputBit	MainProgram	0		

New Tag
Name: OneShotCount_Trigger
Create
Description:
Cancel

Select Data Type
Data Types:
FBD_ONESHOT
OK
Cancel
Help
FBD_COUNTER
FBD_LIMIT
FBD_LOGICAL
FBD_MASK_EQUAL
FBD_MASKED_MOVE
FBD_MATH
FBD_MATH_ADVANCED
FBD_ONESHOT
Array Dimensions
Dim 2 Dim 1 Dim 0
0 0 0
Show Data Types by Groups
Constant
Sequencing
Open Configuration
Open Parameter Connections

HUMBER

STRUCTURED TEXT PROGRAMMING: – MOTOR CONTROL WITH RESTART AT THE LIMIT SWITCH



HUMBER

STRUCTURED TEXT PROGRAMMING: – MOTOR CONTROL WITH RESTART AT THE LIMIT SWITCH

```
// MOTOR CONTROL WITH RESTART AT THE LIMIT SWITCH WITH ONE SHOT RISE FROM THE LIMIT SWITCH


OSRI (ONE_SHOT); // ONE SHOT RISE INSTRUCTION SETTING


ONE_SHOT.InputBit := LIMIT_SWITCH; // ONE SHOT RISE PULSE GENERATION FROM THE LIMIT SWITCH

RUN := (START_PB OR RUN) AND NOT ONE_SHOT.OutputBit;

IF RUN THEN
MOTOR := 1;
ELSE
MOTOR :=0;
END_IF;
```

STRUCTURED_TEXT*
MainProgram

Watch

Current Routine

Name	Scope	Value	Force Mask	Description
LIMIT_SWITCH	MainProgram	0		
MOTOR	MainProgram	0		
ONE_SHOT	MainProgram	{...}	{...}	
ONE_SHOT.InputBit	MainProgram	0		
ONE_SHOT.OutputBit	MainProgram	0		
RUN	MainProgram	0		
START_PB	MainProgram	0		

Structured Text Example

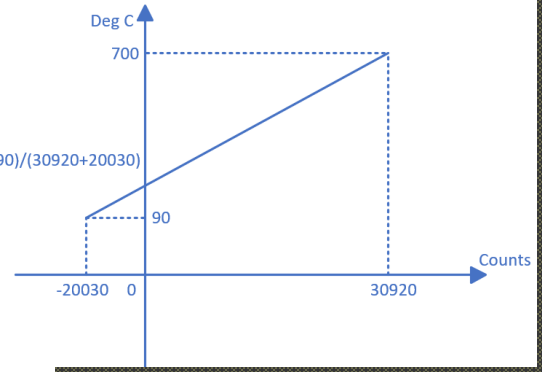
The TTT573 transducer measures temperature in the range of 90 ^{0}C to 700 ^{0}C and is connected to an CLX PLC analog input module having a 16-bit ADC whose output integer value corresponds to the lowest and highest sensor value as:

- -20030
- 30920

Convert the ADC output integer (TT573_MEAS) to the temperature, in ^{0}C and store the result in TT573_DEGC. Also, the ADC output integer should be limited to the proper range before doing the conversation.
Assume the following physical input and internal variable:

- Local:3:I.Ch0Data
- TT573_MEAS – raw temperature measurement, represents 90 ^{0}C - 700 ^{0}C for the ADC integer range indicated above
- TT573_DEGC – Current temperature in ^{0}C

```
1   (* Example 12.1 Conversion Example *)
2
3   (* Copyright (c) 2011 Dogwood Valley Press, LLC *)
4
5   TmpInt := TT573_MEAS;
6   IF (TmpInt < -20030) THEN
7     TmpInt := -20030;
8   END_IF;
9   IF (TmpInt > 30920) THEN
10    TmpInt := 30920;
11  END_IF;
12  TT573_DEGC := ((TmpInt + 20030) / 50950.0)*(700.0 - 90.0) + 90.0;
13
14
15
```



HUMBER

FUNCTION BLOCK DIAGRAM PROGRAMMING

INTRODUCTION

A function block diagram (FBD) is a graphical representation of process flow using simple and complex interconnecting blocks.

- A function block can take one or more inputs, make decisions or calculations and generate one or more outputs.
- A dash line indicates a Boolean signal path, and a solid line indicates an integer or real value.
- Connections between blocks and database tags are accomplished with **IREF** and **OREF** symbols.
- The **OCON** and **ICON** symbols are used to program connections between function blocks that are widely separated or on separate sheets.

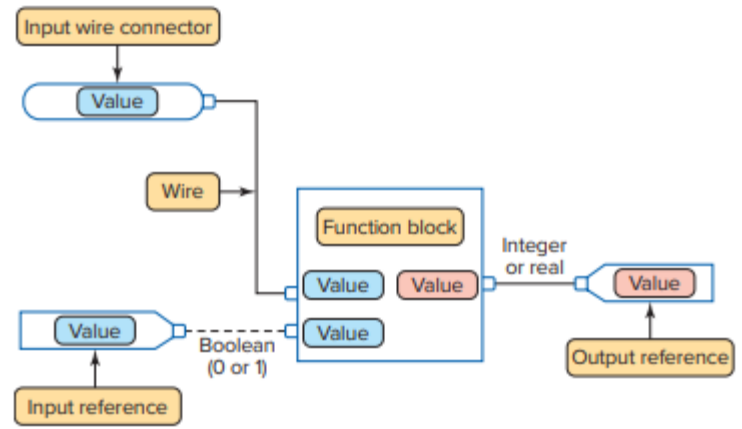


Figure 15-95 Structure of function block or routine.

FUNCTION BLOCK DIAGRAM PROGRAMMING

INTRODUCTION

- References represent tags that are linked to values stored in a controller's memory.
- Function blocks can be connected to other function blocks by connecting their outputs to the input of another function block using wires and pins.
- Wire connectors are used to create a path without using a wire.

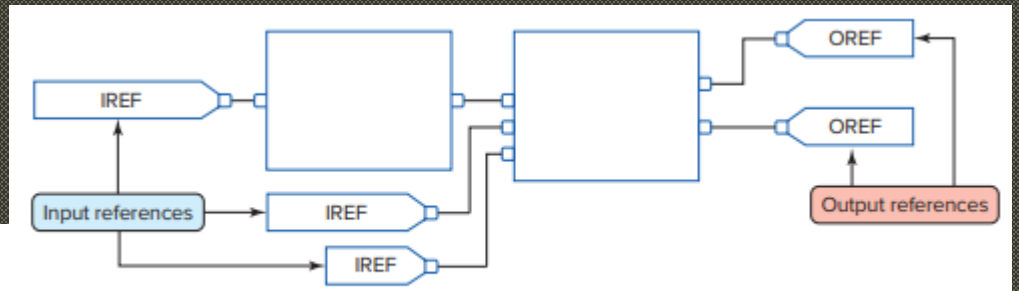


Figure 15-97 Input and output references.

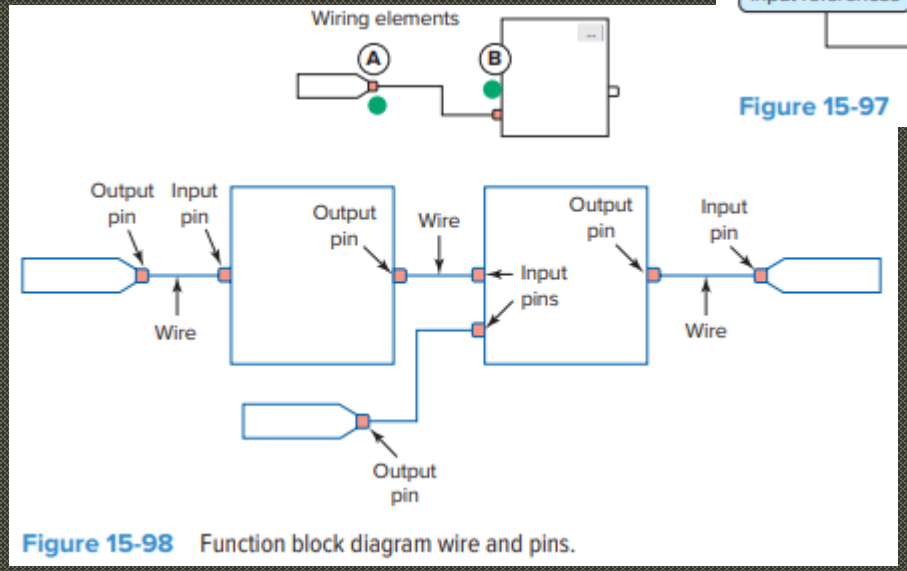


Figure 15-98 Function block diagram wire and pins.

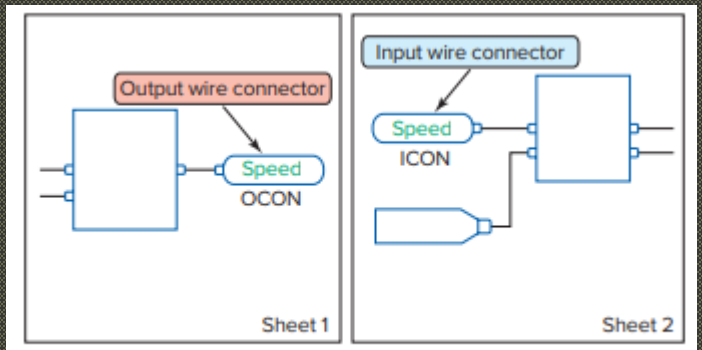


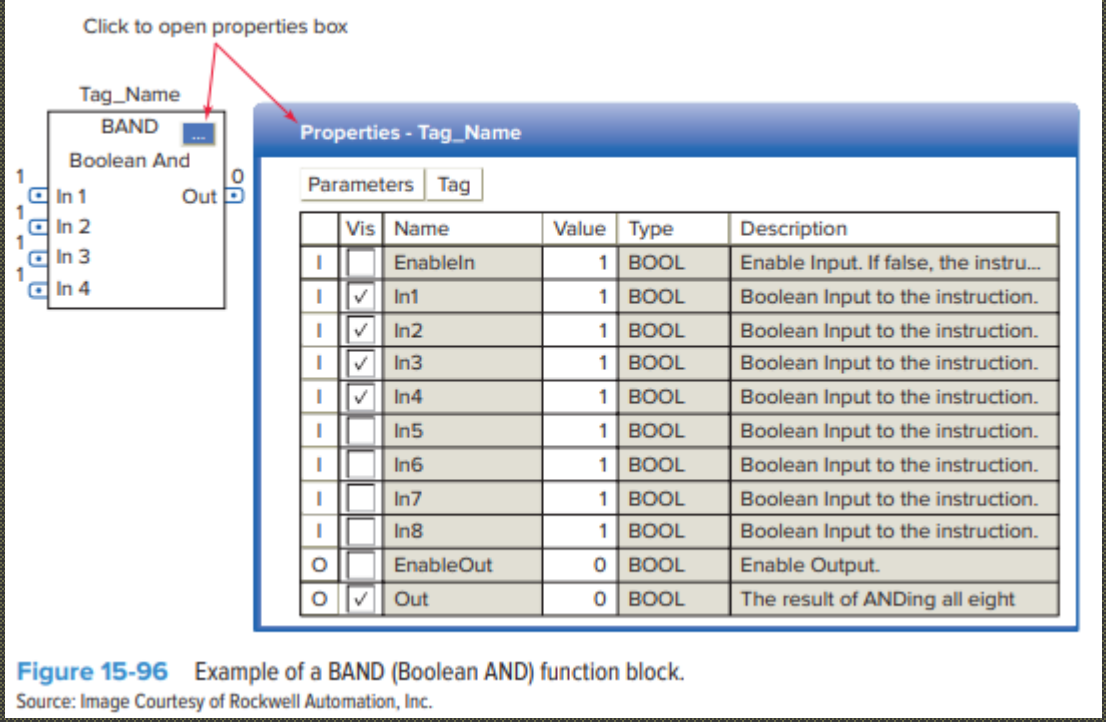
Figure 15-99 OCON and ICON wire connectors.

- An output wire connector, or OCON, sends a value or signal to an input wire connector, or ICON.
- Each output wire connector must have at least one corresponding input wire connector.
- Each output wire connector requires a unique tag name, and the corresponding input connector must have the same name.

FUNCTION BLOCK DIAGRAM PROGRAMMING

INTRODUCTION

There are different types of FBDs included in the programming software to perform various common tasks.
In addition, customized Add-On instructions can be created by the programmer for sets of commonly used logic.



	Vis	Name	Value	Type	Description
I	☐	EnableIn	1	BOOL	Enable Input. If false, the instru...
I	✓	In1	1	BOOL	Boolean Input to the instruction.
I	✓	In2	1	BOOL	Boolean Input to the instruction.
I	✓	In3	1	BOOL	Boolean Input to the instruction.
I	✓	In4	1	BOOL	Boolean Input to the instruction.
I	☐	In5	1	BOOL	Boolean Input to the instruction.
I	☐	In6	1	BOOL	Boolean Input to the instruction.
I	☐	In7	1	BOOL	Boolean Input to the instruction.
I	☐	In8	1	BOOL	Boolean Input to the instruction.
O	☐	EnableOut	0	BOOL	Enable Output.
O	✓	Out	0	BOOL	The result of ANDing all eight

Figure 15-96 Example of a BAND (Boolean AND) function block.
Source: Image Courtesy of Rockwell Automation, Inc.

- Inputs are shown entering from the left and outputs exiting on the right.
- The function block type is shown within the block.
- A tag name for the block is placed above it.
- The names of the inputs and outputs are shown within the block.
- The default view of the block has some but not all the input and output parameters visible when the box is placed into the program.
- The 1 and 0 next to the inputs and outputs identify the logical state of the input and output pins for the instruction.

FUNCTION BLOCK DIAGRAM PROGRAMMING

INTRODUCTION

- In every program scan, FBD blocks are assessed from the left side to the right, following the signal flow, until the final output is determined.
- If function blocks are not wired together, it does not matter which block executes first as there is no data flow between the blocks.
- The location of a block does not affect the order in which the blocks execute.

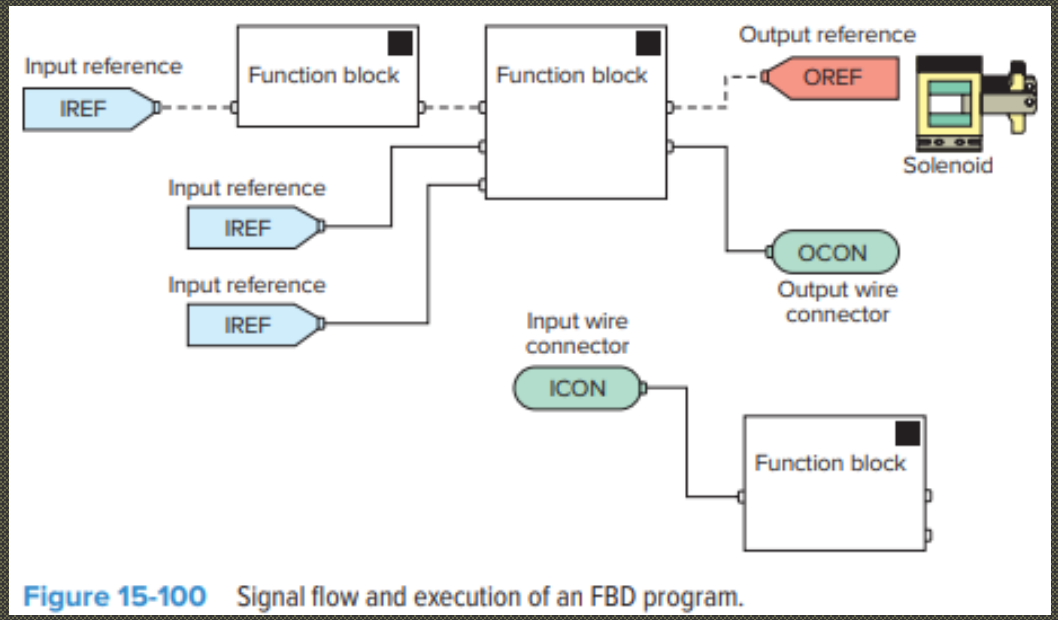
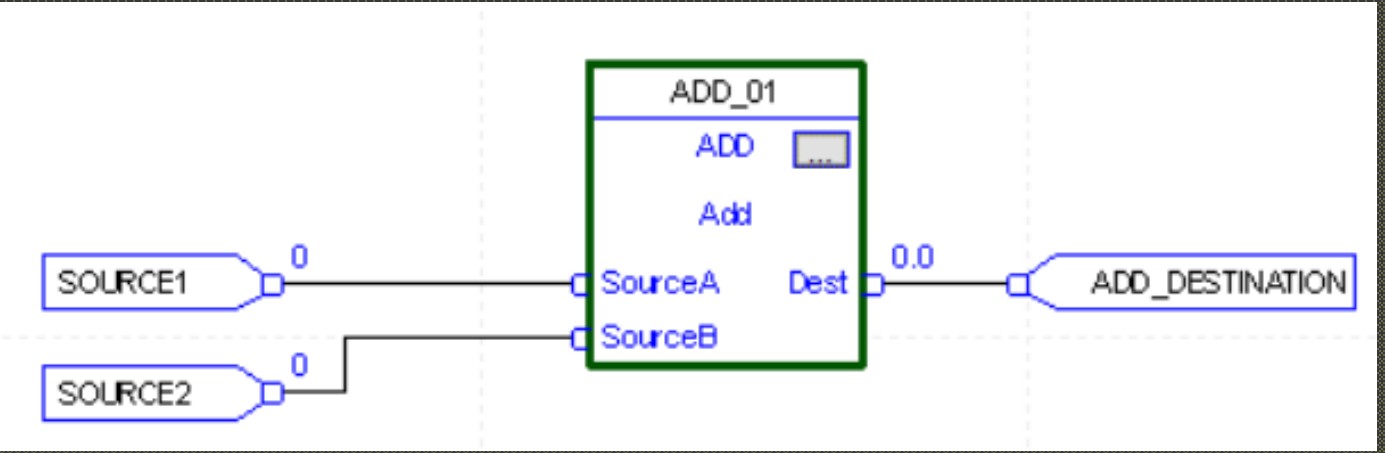


Figure 15-100 Signal flow and execution of an FBD program.

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Instruction Sample

- To use a function block instruction in the routine, a tag is created for the function block that has several tag members.
- In the case of **ADD** function block, ADD_01 is the default name (can change the name) that is automatically created when you put the ADD function block into the routine.



FUNCTOIN BLOCK DIAGRAM PROGRAMMING

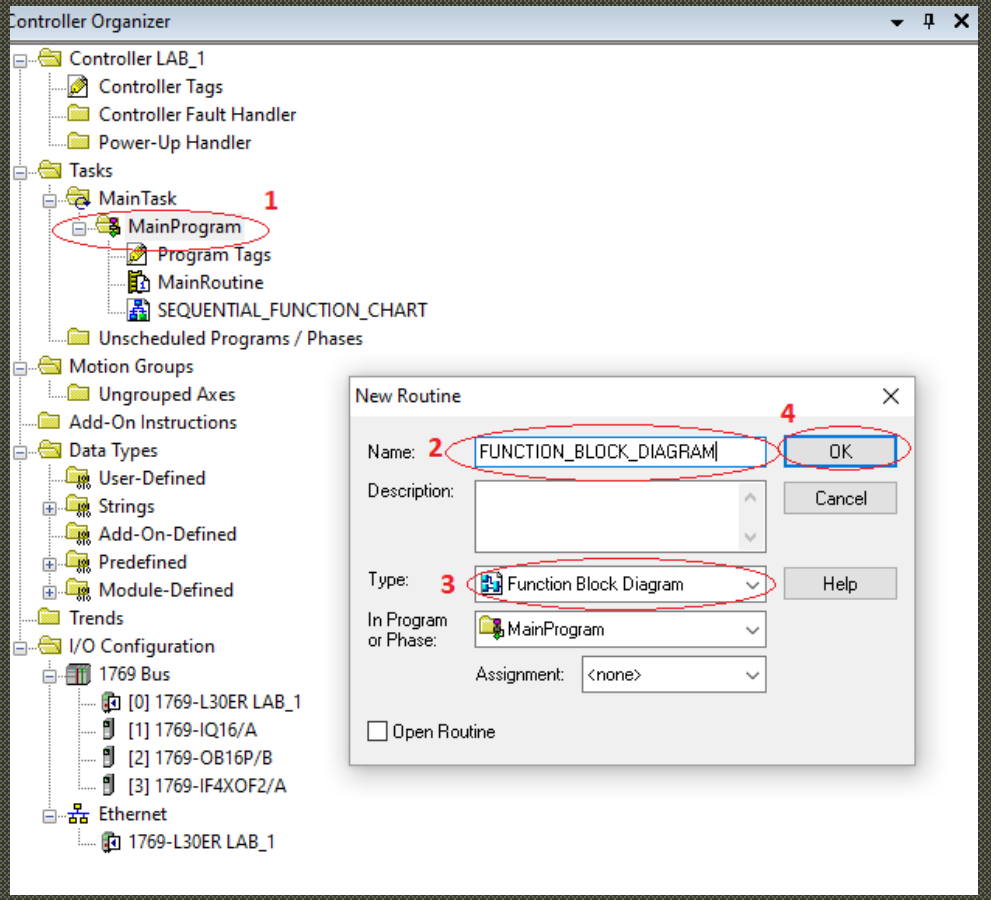
ADD Function Block

- Each function block tag has several tag members that can be used in logic.
- The function block tag members are used to store configuration and status information about the instruction. For example, ADD_01.EnableIn tag member could be used in logic to enable this function block.

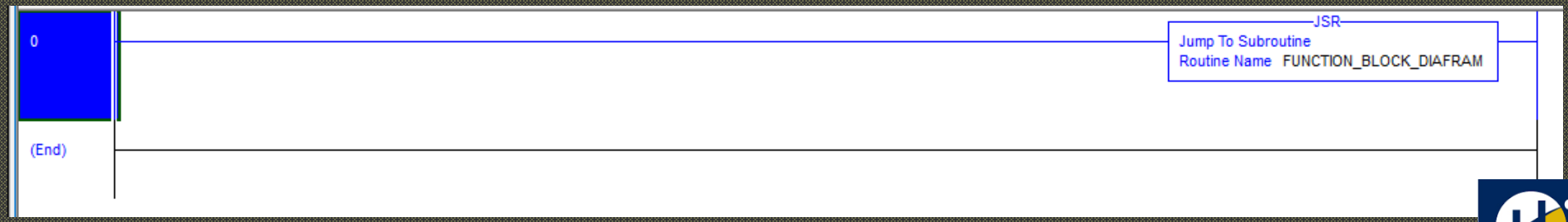
Name	Value	Force Mask	Style	Data T
ADD_01	{...}	{...}		FBD_M
ADD_01.EnableIn	1		Decimal	BOOL
ADD_01.SourceA	0.0		Float	REAL
ADD_01.SourceB	0.0		Float	REAL
ADD_01.EnableOut	0		Decimal	BOOL
ADD_01.Dest	0.0		Float	REAL
SOURCE1	0		Decimal	DINT
SOURCE2	0		Decimal	DINT
ADD_DESTINATION	0		Decimal	DINT

FUNCTION BLOCK DIAGRAM PROGRAMMING

- CREATE A FUNCTION_BLOCK_DIAGRAM SUBROUTINE



- CALL THE SUBROUTINE FROM THE MAIN ROUTINE

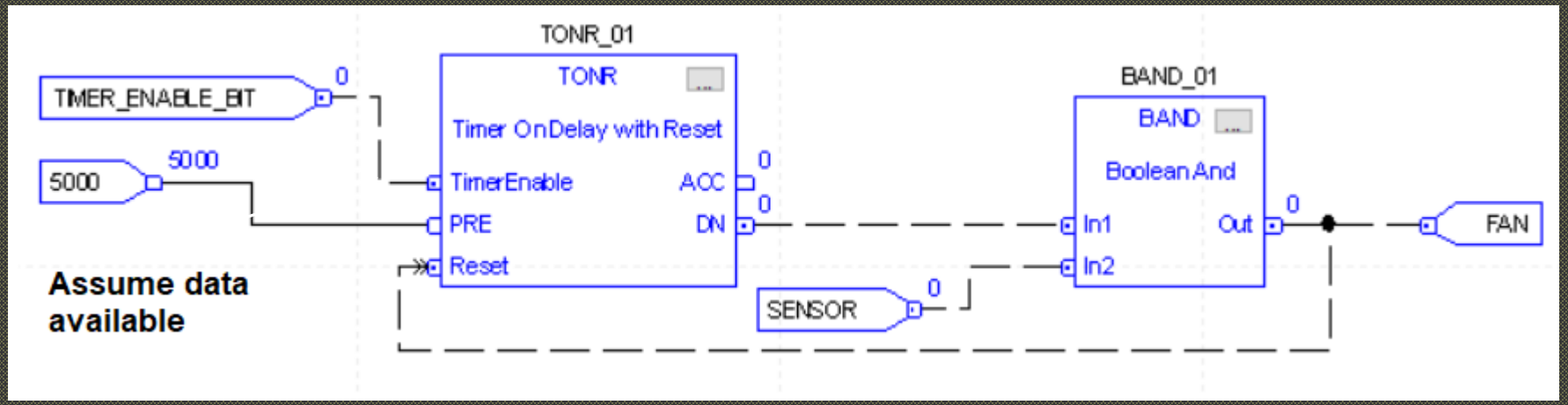


HUMBER

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Feedback

Feedback to a block is done by writing an output pin from a block to an input pin on the same function block.



HUMBER

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Function Block Elements and Diagrams

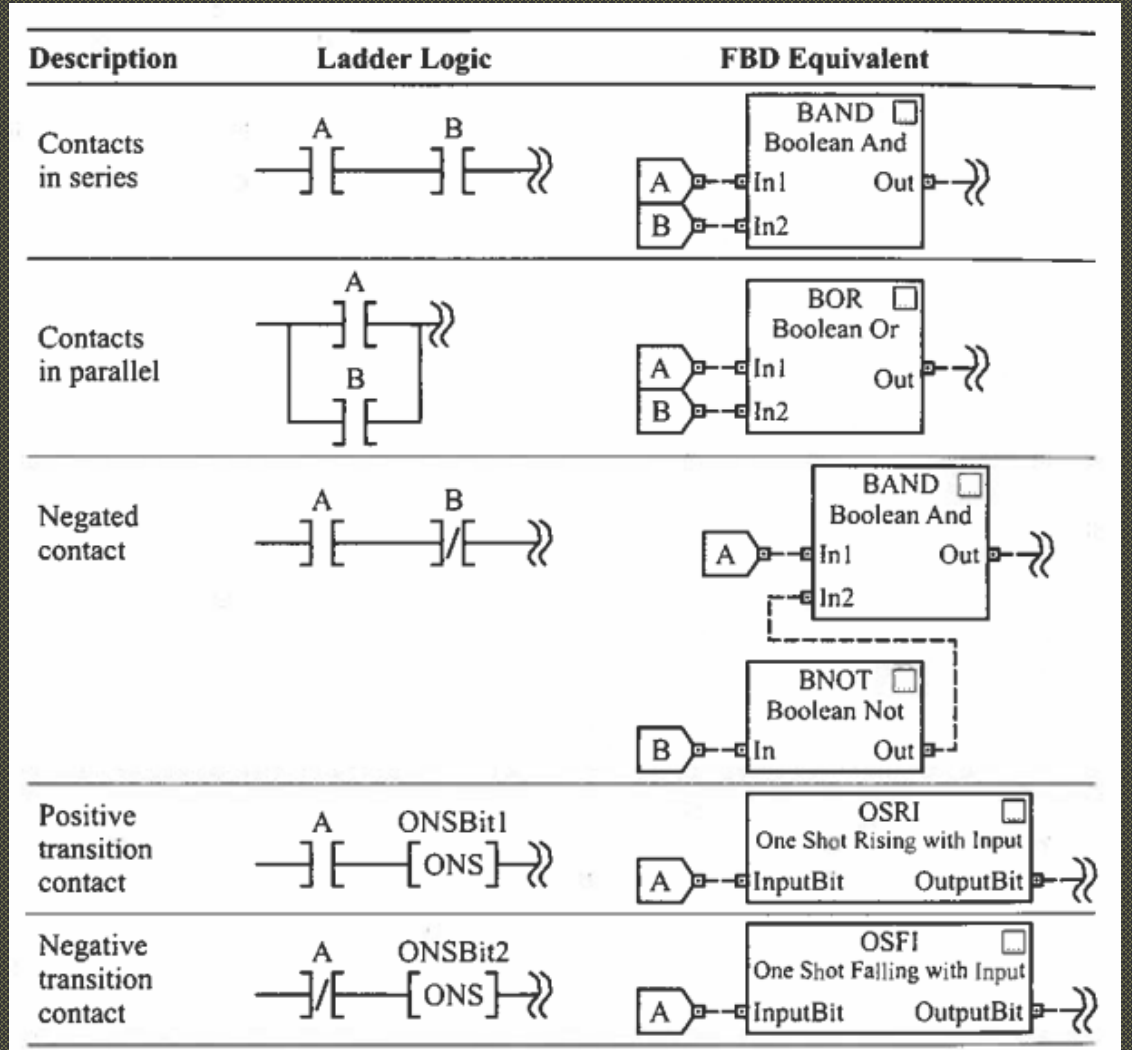


Figure 11.8. ControlLogix FBD equivalents to ladder logic contacts.

HUMBER

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Function Block Elements and Diagrams

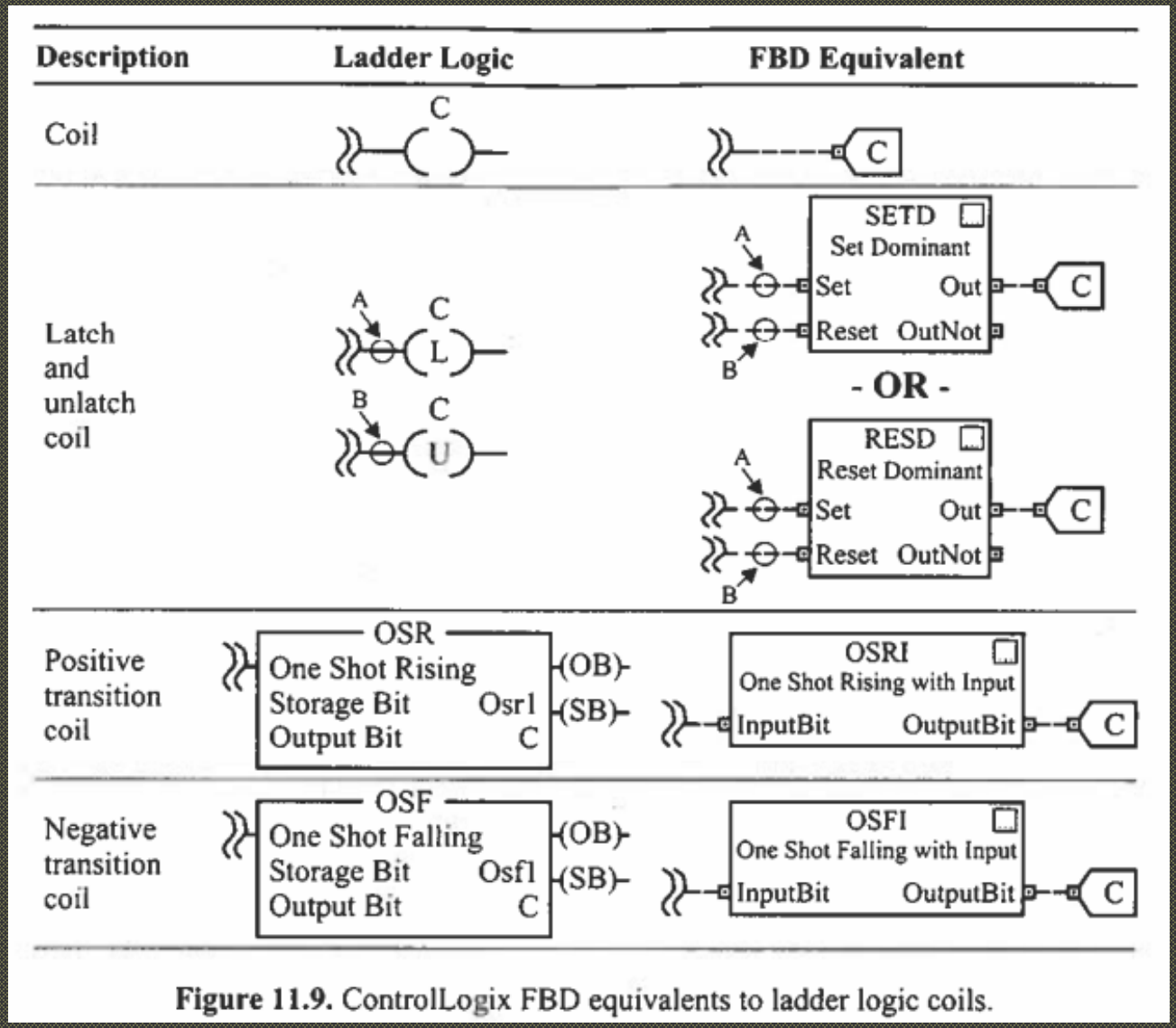


Figure 11.9. ControlLogix FBD equivalents to ladder logic coils.

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Function Block Elements and Diagrams

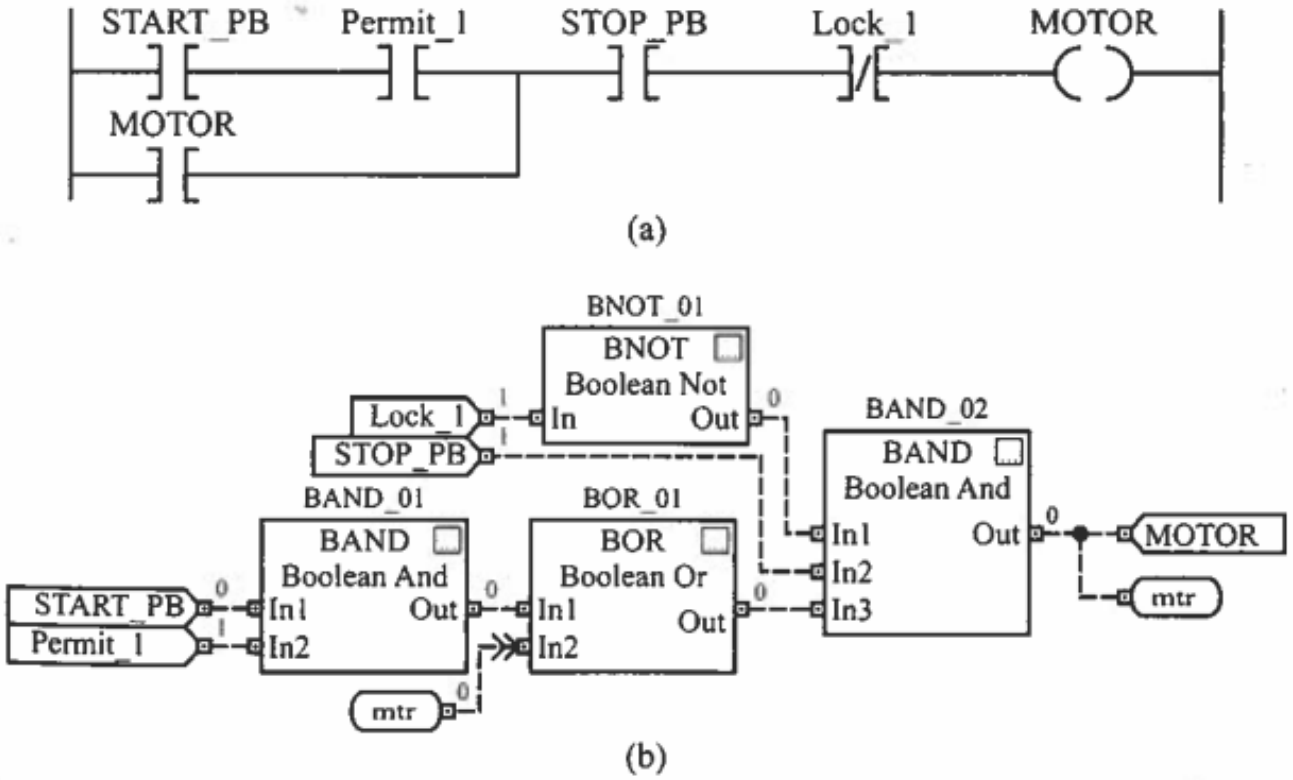


Figure 11.10. Start/stop logic for ControlLogix: *(a)* ladder logic; *(b)* equivalent FBD with implicit feedback.

HUMBER

FUNCTION BLOCK DIAGRAM

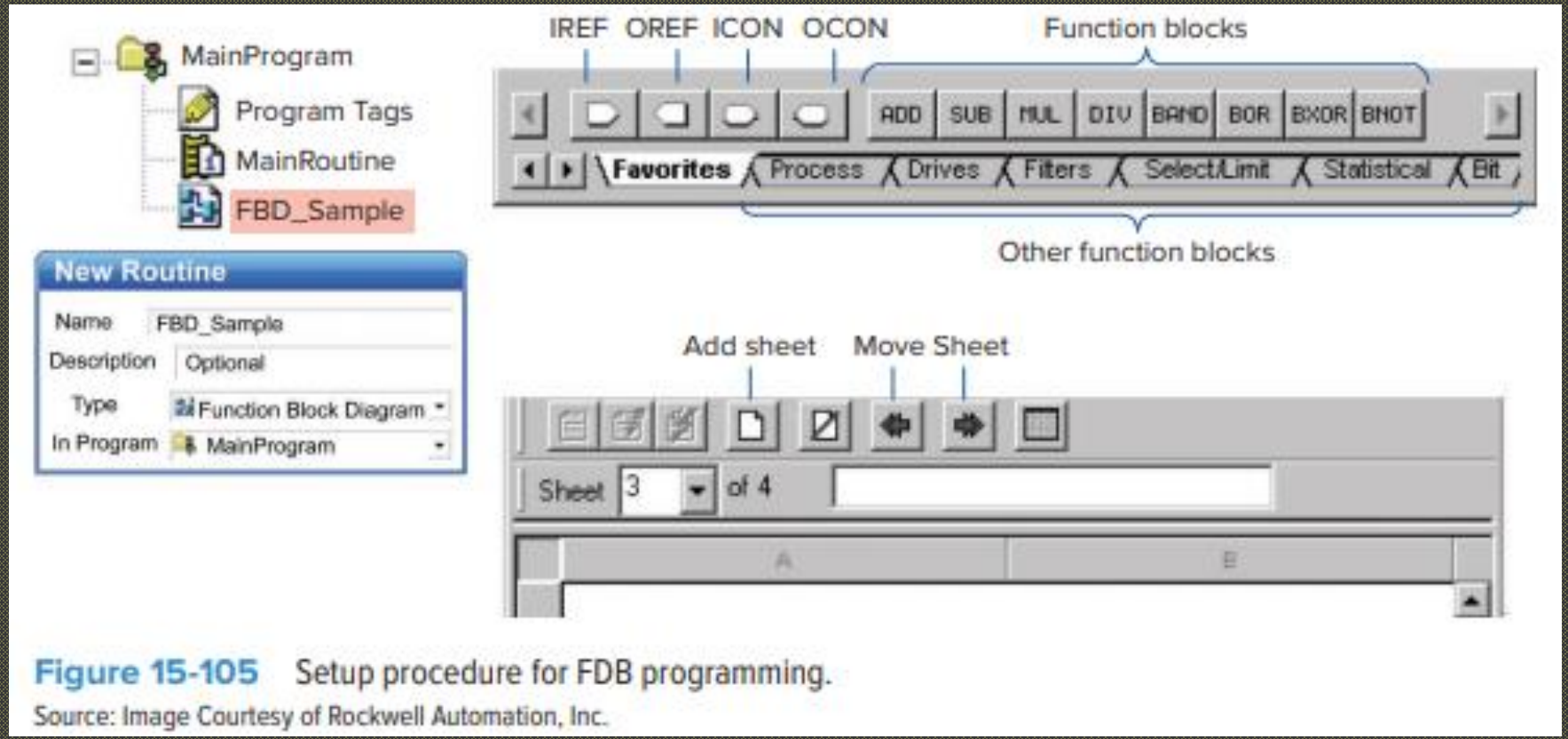
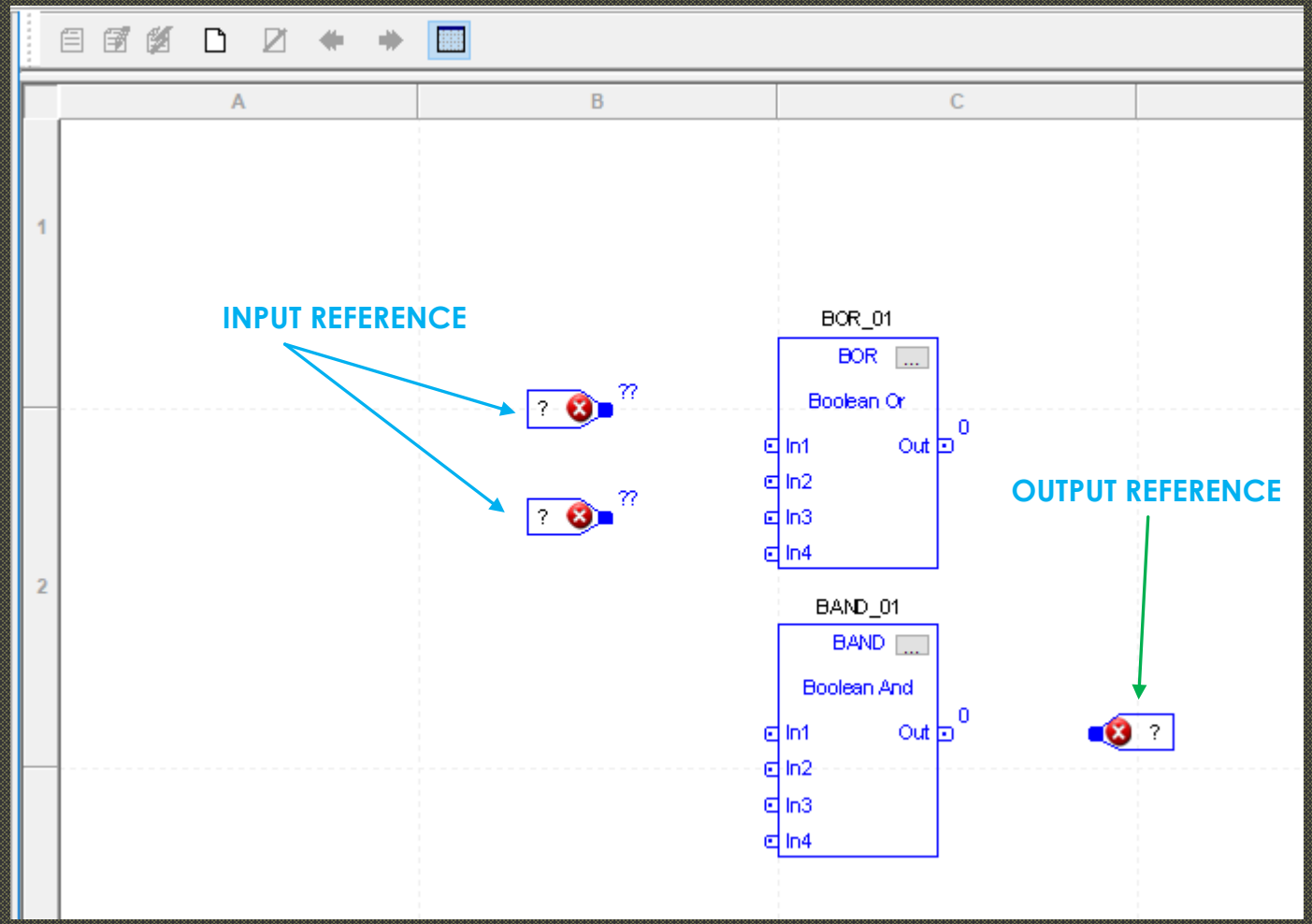


Figure 15-105 Setup procedure for FDB programming.

Source: Image Courtesy of Rockwell Automation, Inc.

FUNCTION BLOCK DIAGRAM

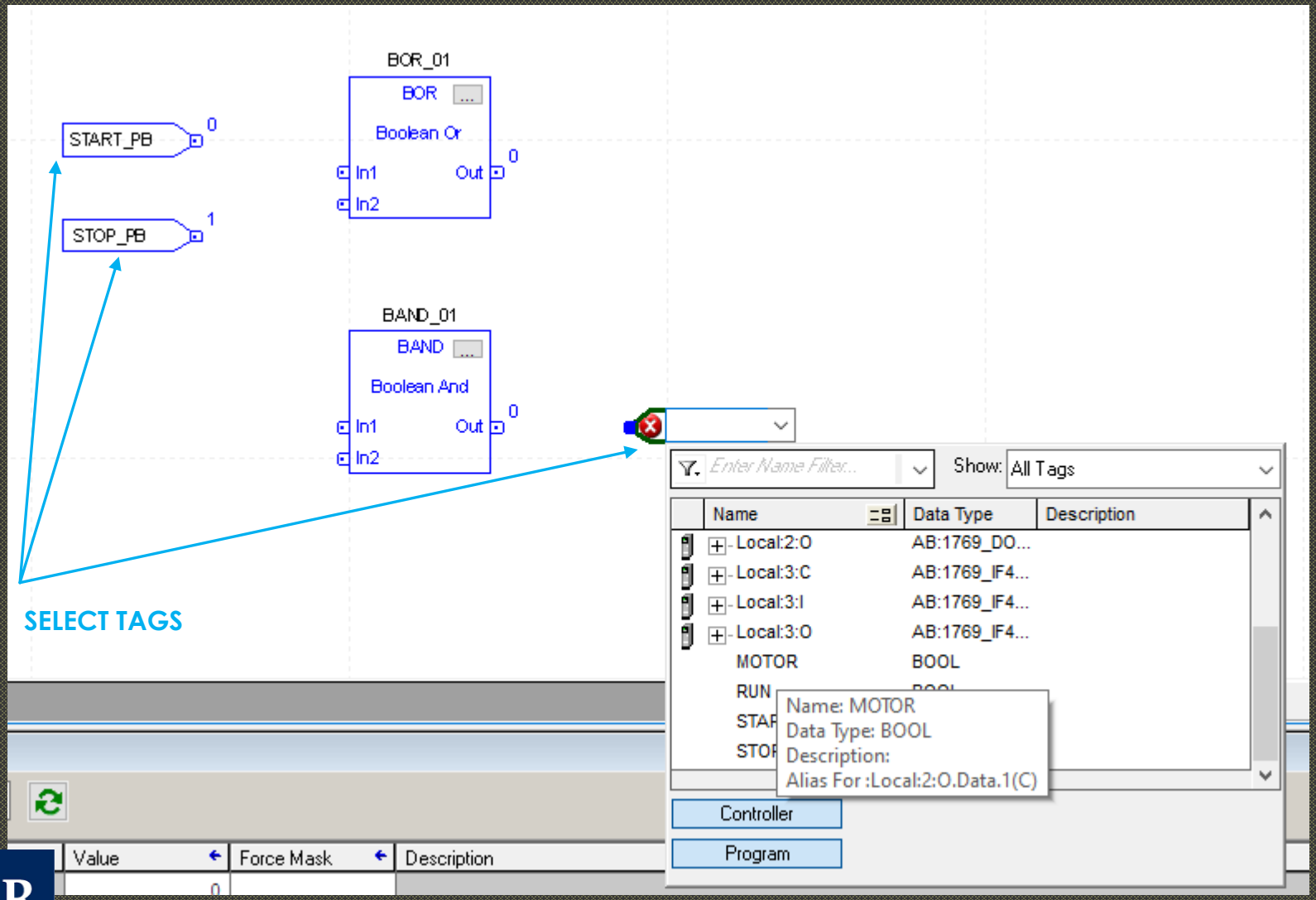
START_PB (N.O.) – STOP_PB (N.C.) - MOTOR CONTROL



HUMBER

FUNCTION BLOCK DIAGRAM

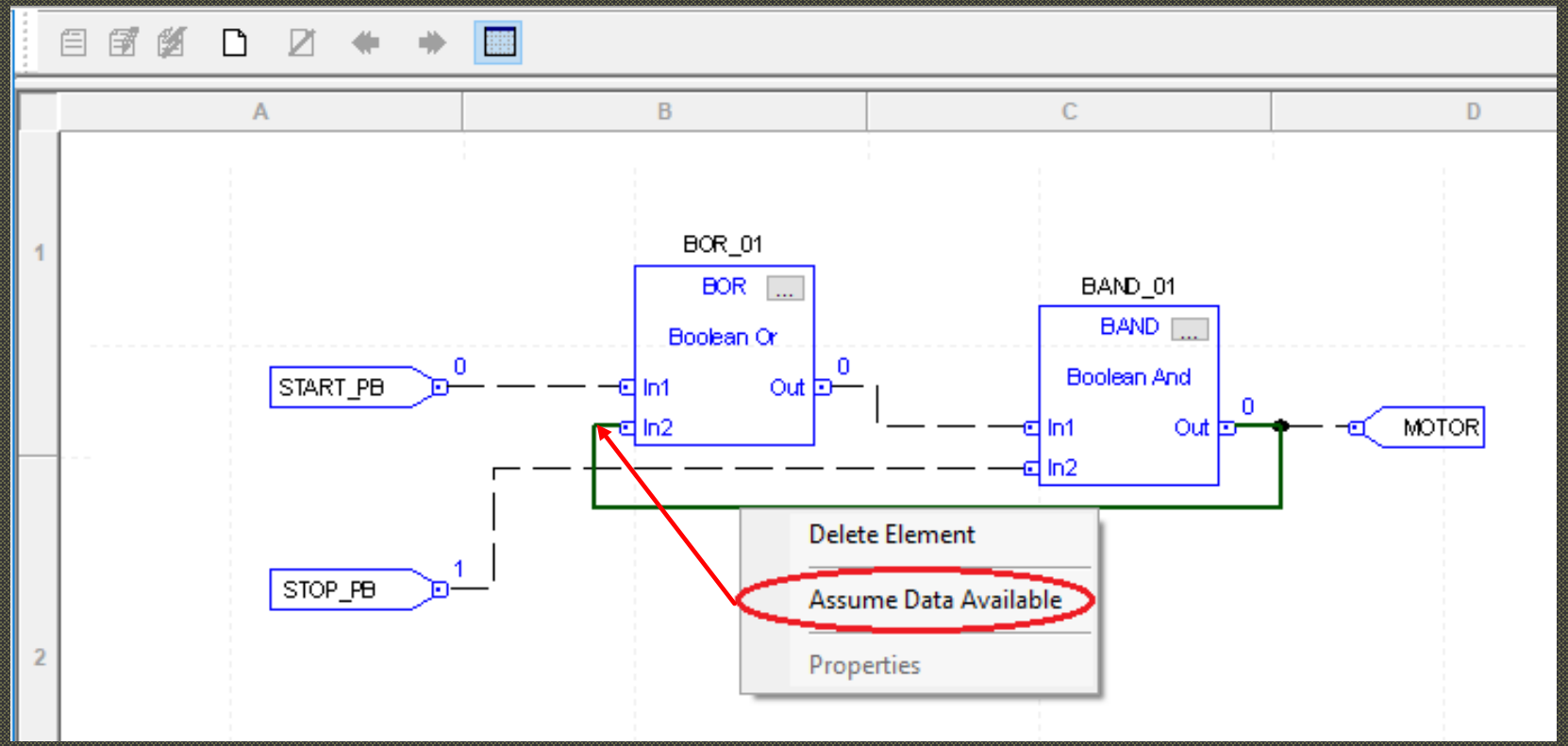
START_PB (N.O.) – STOP_PB (N.C.) - MOTOR CONTROL



HUMBER

FUNCTION BLOCK DIAGRAM

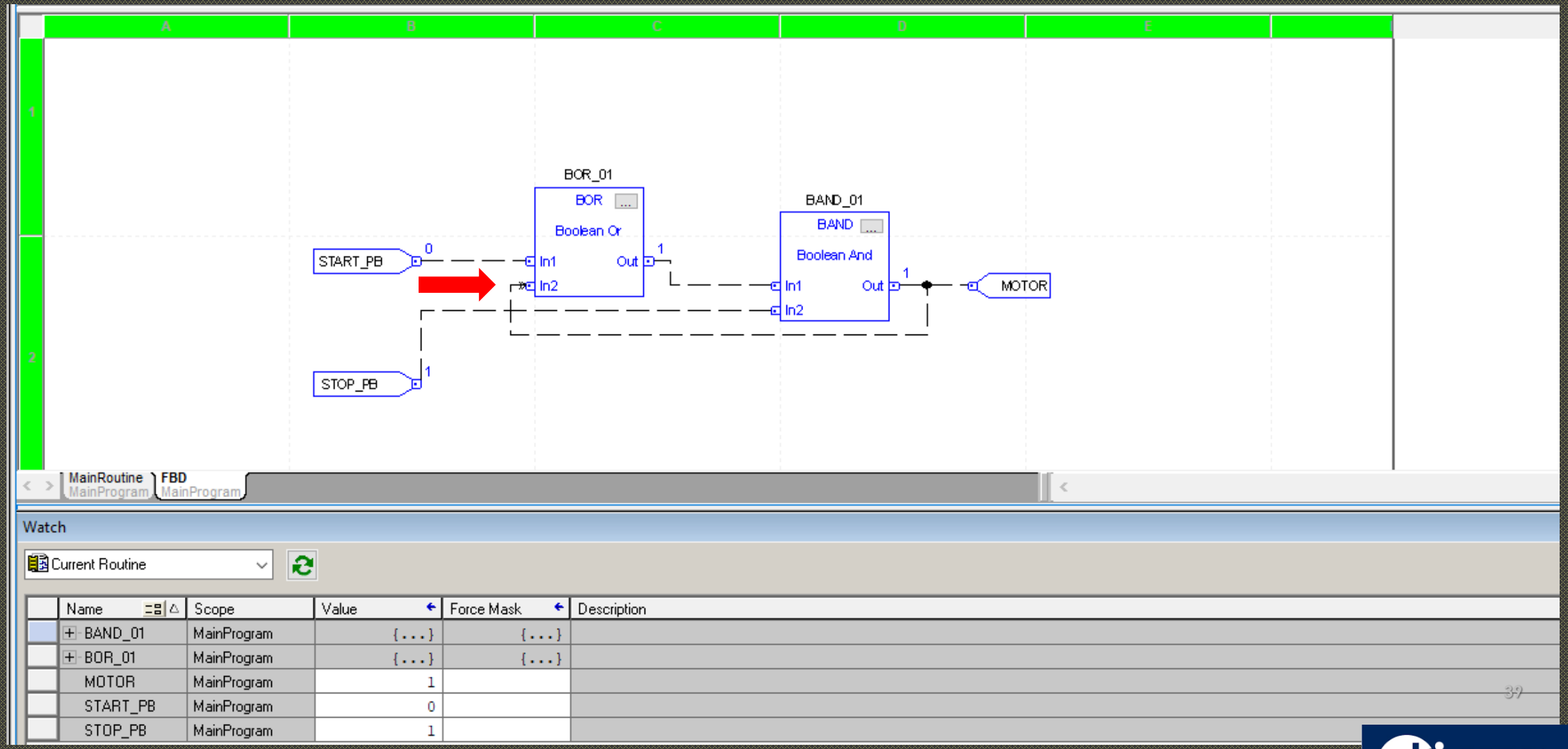
START_PB (N.O.) – STOP_PB (N.C.) - MOTOR CONTROL



HUMBER

FUNCTION BLOCK DIAGRAM

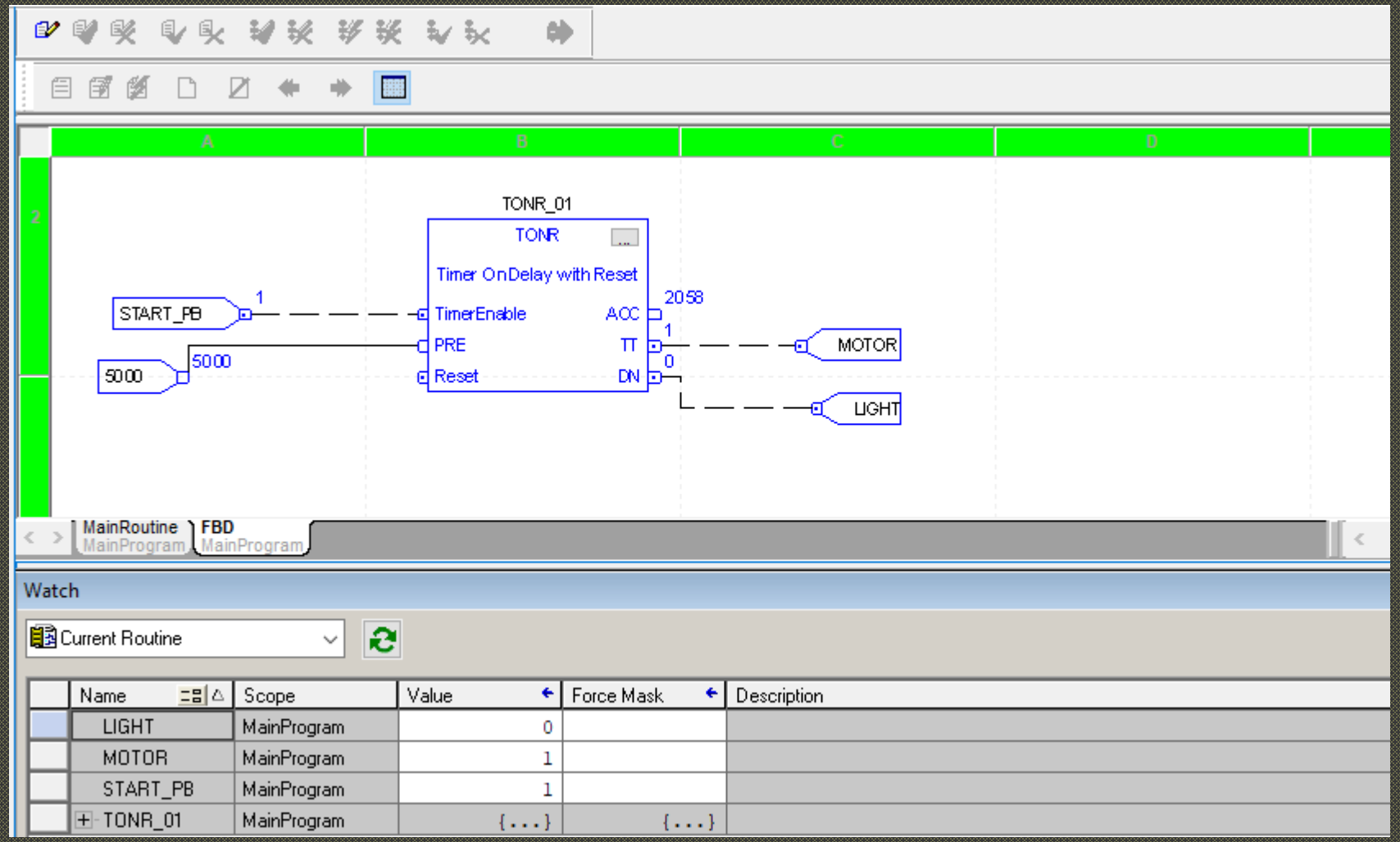
START_PB (N.O.) – STOP_PB (N.C.) - MOTOR CONTROL



HUMBER

FUNCTION BLOCK DIAGRAM

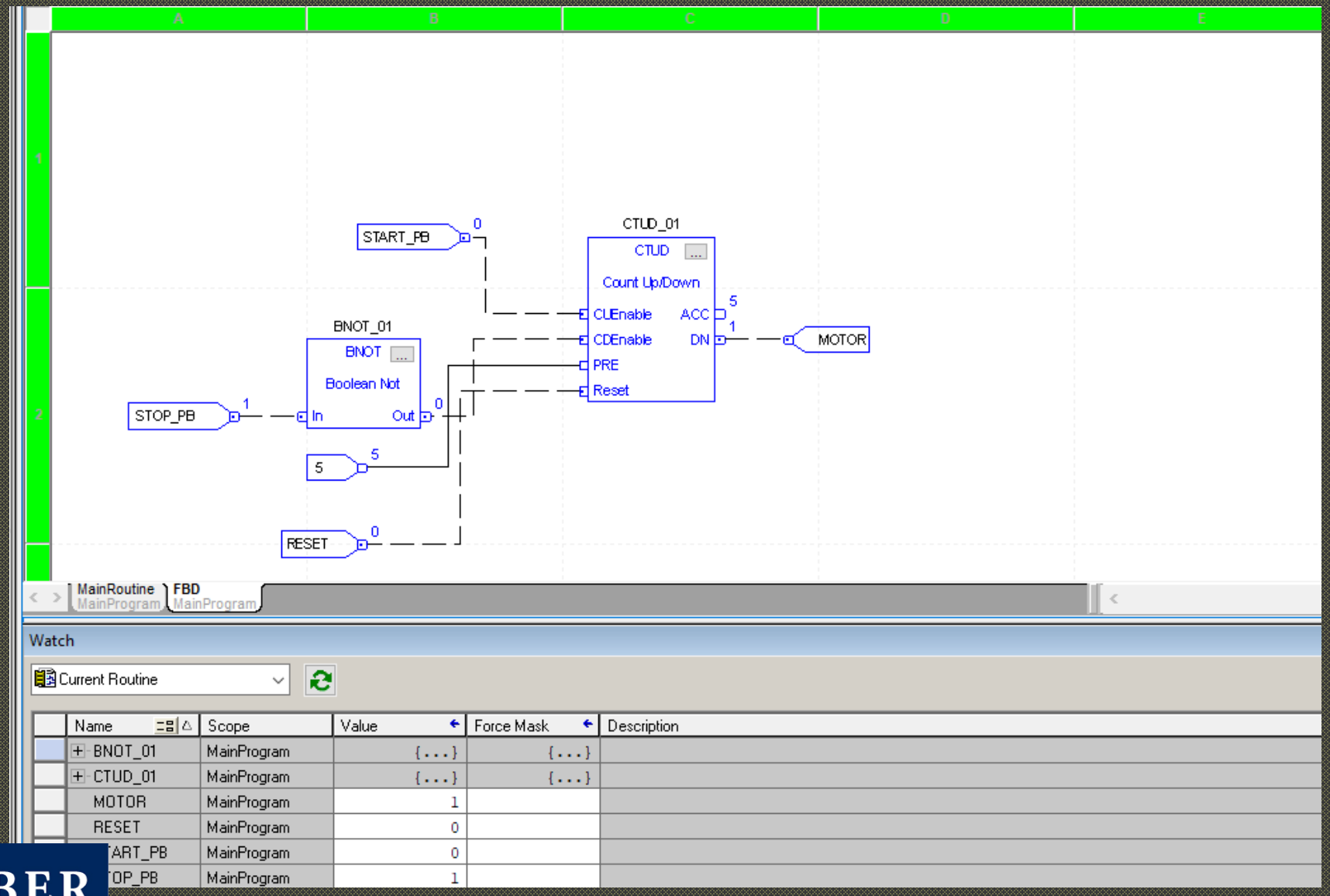
TIMER BASED LIGHT - MOTOR CONTROL



HUMBER

FUNCTION BLOCK DIAGRAM

COUNTER



HUMBER

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Example 1

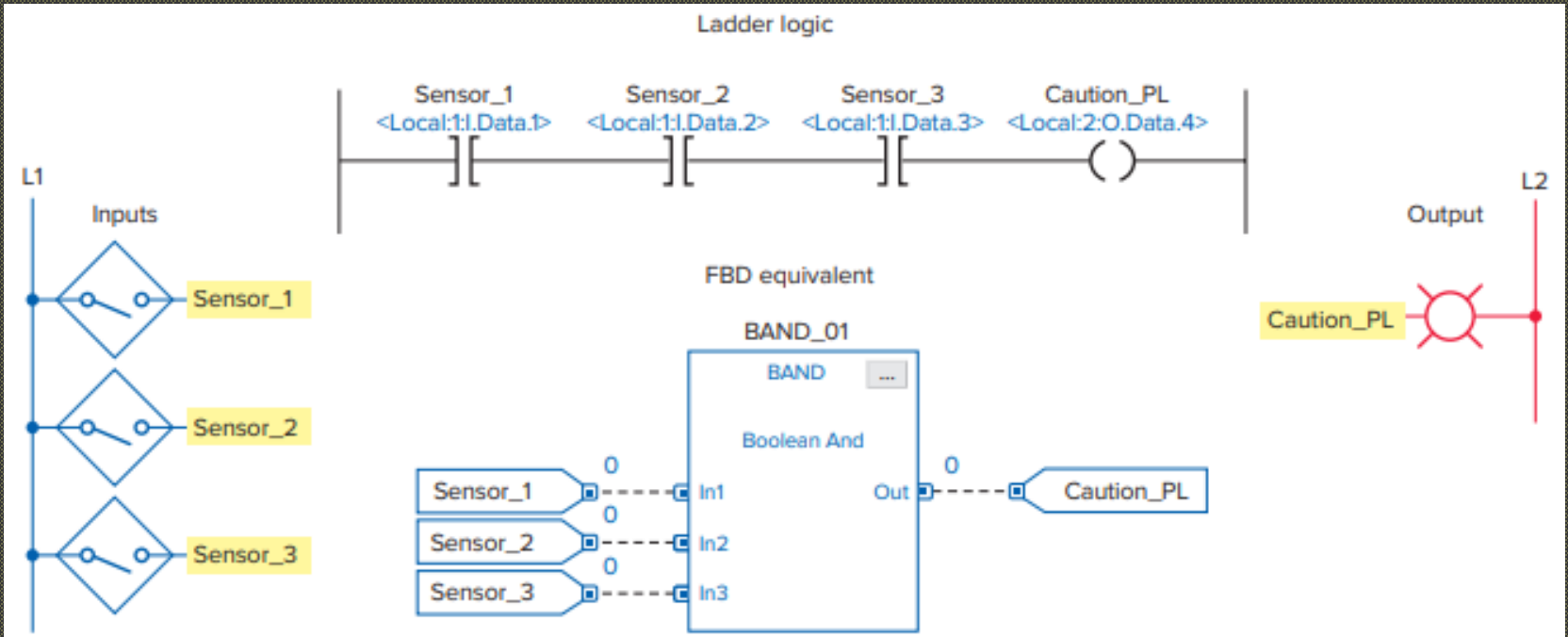


Figure 15-106 Comparison between ladder logic and the FBD equivalent for a three-input AND ladder logic rung.

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Example 2

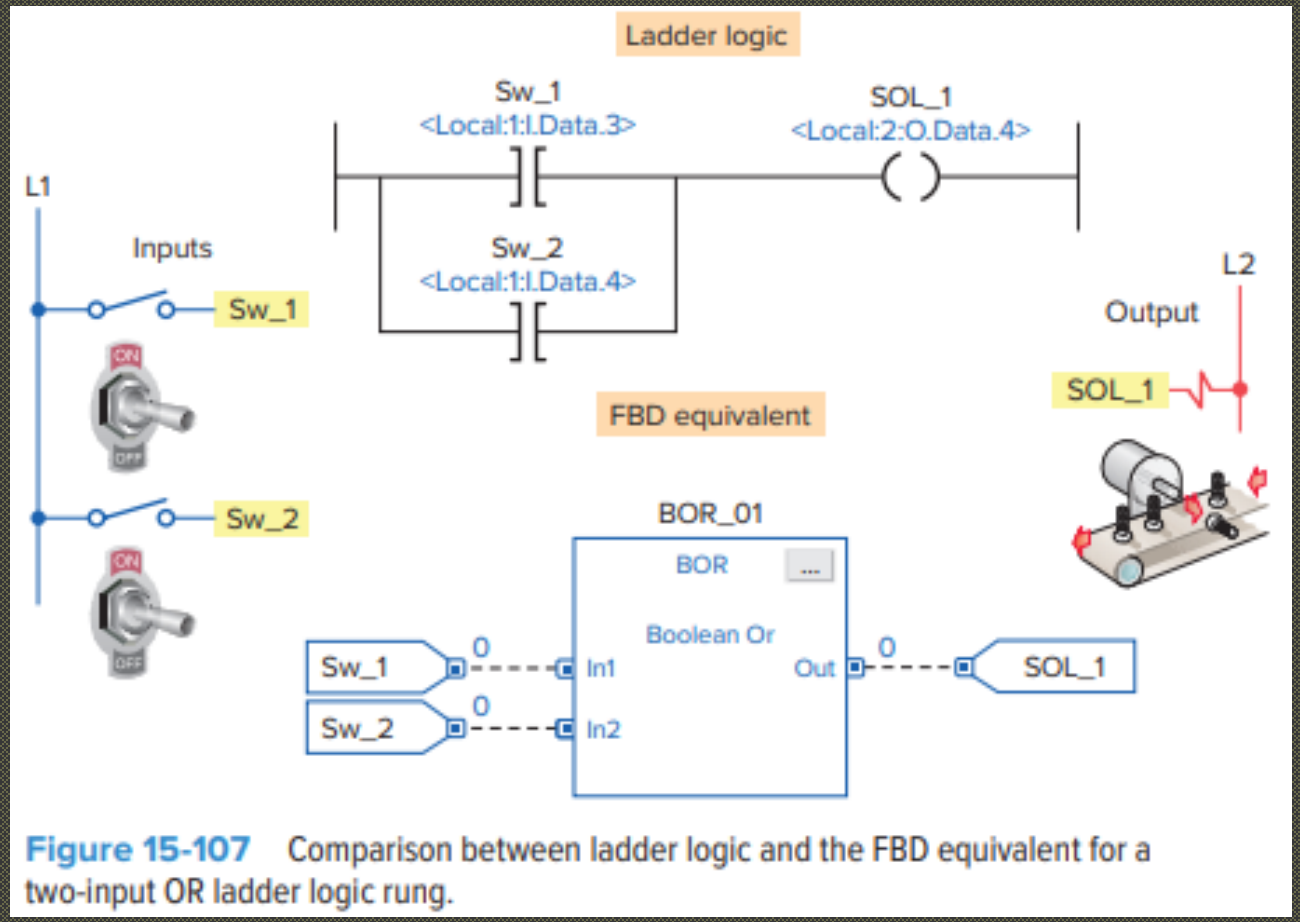


Figure 15-107 Comparison between ladder logic and the FBD equivalent for a two-input OR ladder logic rung.

FUNCTOIN BLOCK DIAGRAM PROGRAMMING

Example 3

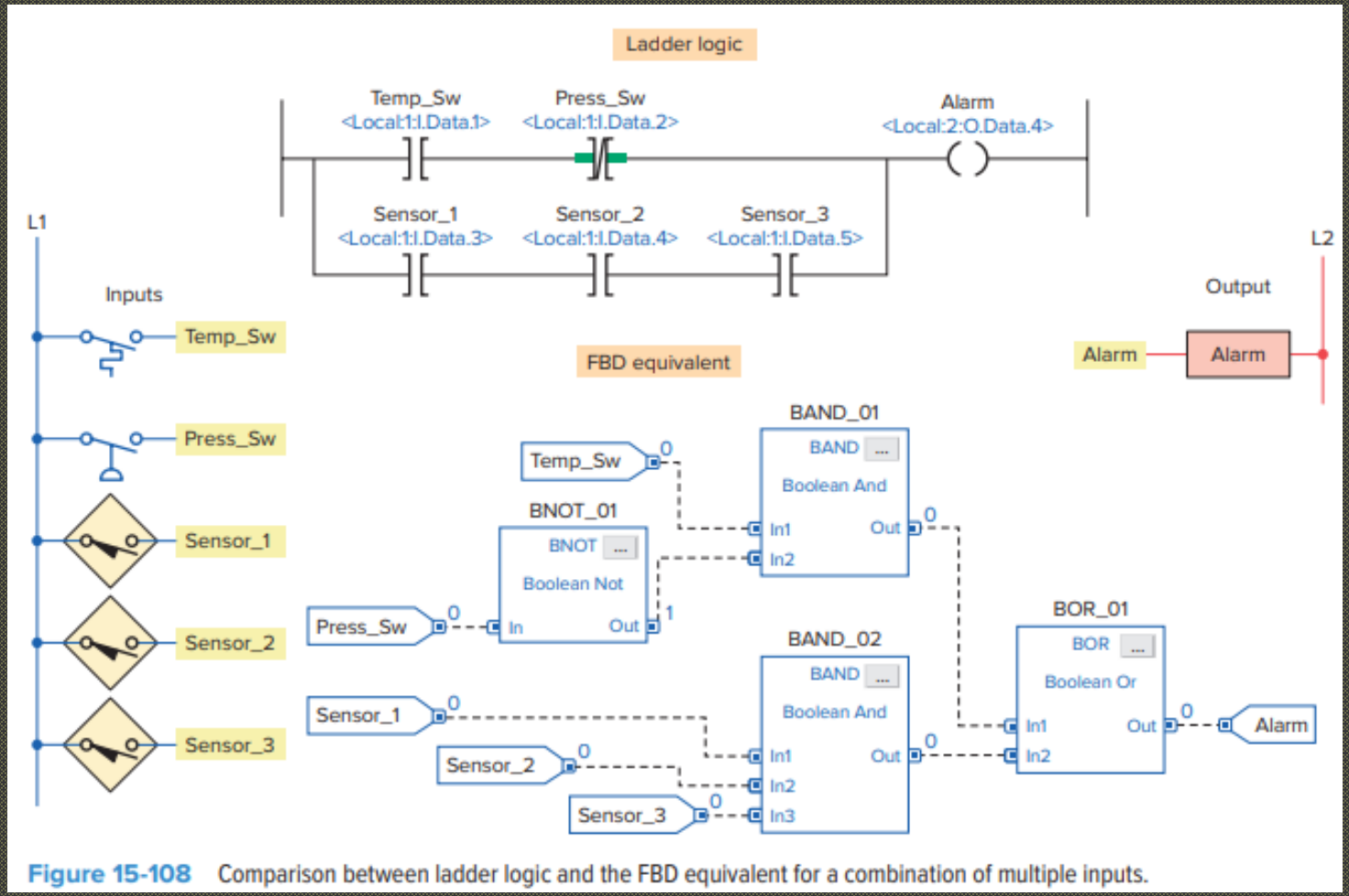
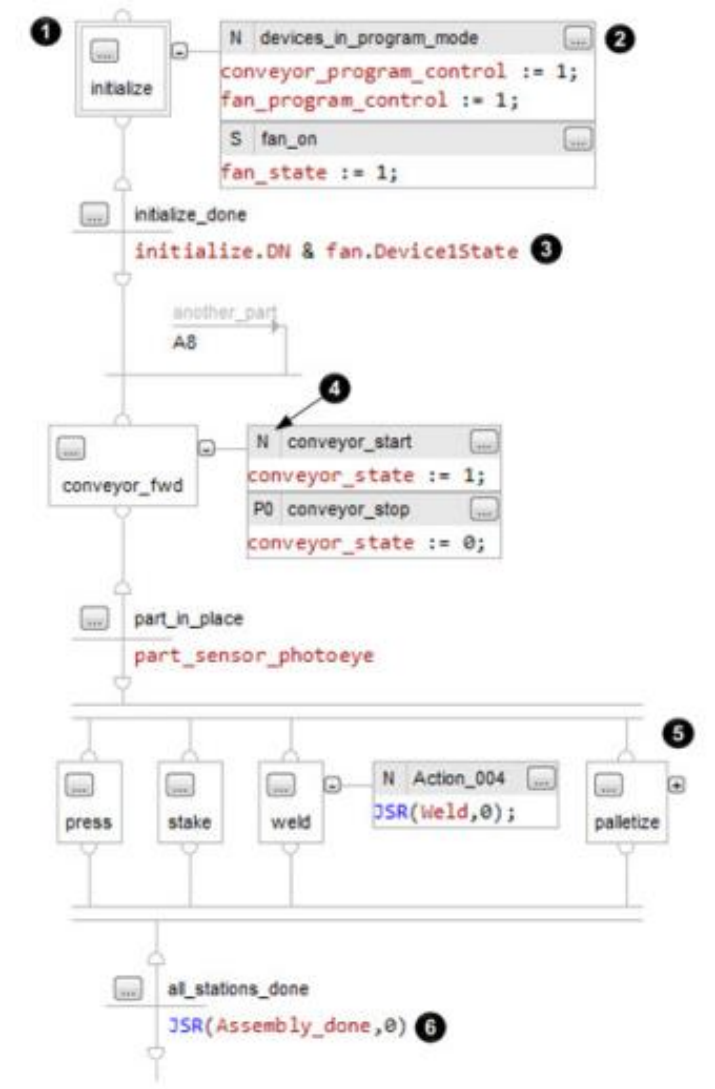


Figure 15-108 Comparison between ladder logic and the FBD equivalent for a combination of multiple inputs.

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING



- A sequential function chart (SFC) is like a flowchart of a process.
- It uses steps and transitions to perform specific operations or actions.

1	A step represents a major function of your process. It contains the actions that occur at a particular time, phase, or station.
2	An action is one of the functions that a step performs.
3	A transition is the TRUE or FALSE condition that tells the SFC when to go to the next step.
4	A qualifier determines when an action starts and stops.
5	A simultaneous branch executes more than 1 step at the same time.
6	JSR instruction calls a subroutine.

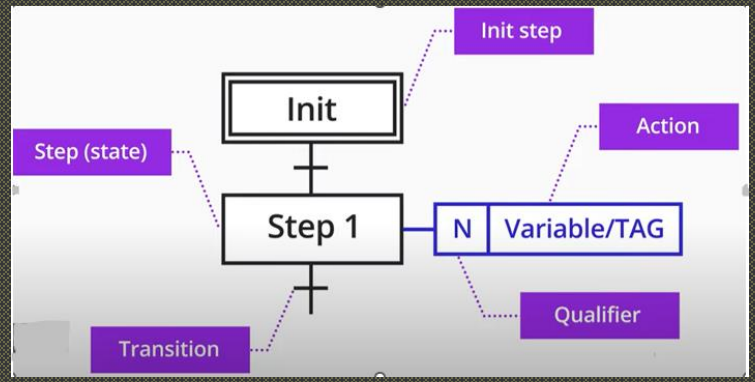
Rockwell Automation Publication 1756-PM006K-EN-P

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

- Each action (non-Boolean and Boolean) uses a qualifier to determine when it starts and stops.
- The default qualifier is N, Non-Stored.
- The action starts when the step is activated and stops when the step is deactivated.

If you want the action to	And	Then assign this qualifier	Which means
Start when the step is activated	Stop when the step is deactivated	N	Non-Stored
	Execute only once	P1	Pulse (Rising Edge)
	Stop before the step is deactivated or when the step is deactivated	L	Time Limited
	Stay active until a Reset action turns off this action	S	Stored
	Stay active until a Reset action turns off this action Or a specific time expires, even if the step is deactivated	SL	Stored and Time Limited
Start a specific time after the step is activated and the step is still active	Stop when the step is deactivated	D	Time Delayed
	Stay active until a Reset action turns off this action	DS	Delayed and Stored
Start a specific time after the step is activated, even if the step is deactivated before this time	Stay active until a Reset action turns off this action	SD	Stored and Time Delayed
Execute once when the step is activated	Execute once when the step is deactivated	P	Pulse
Start when the step is deactivated	Execute only once	P0	Pulse (Falling Edge)
Turn off (reset) a stored action • S Stored • SL Stored and Time Limited • DS Delayed and Stored • SD Stored and Time Delayed	---------->	R	Reset

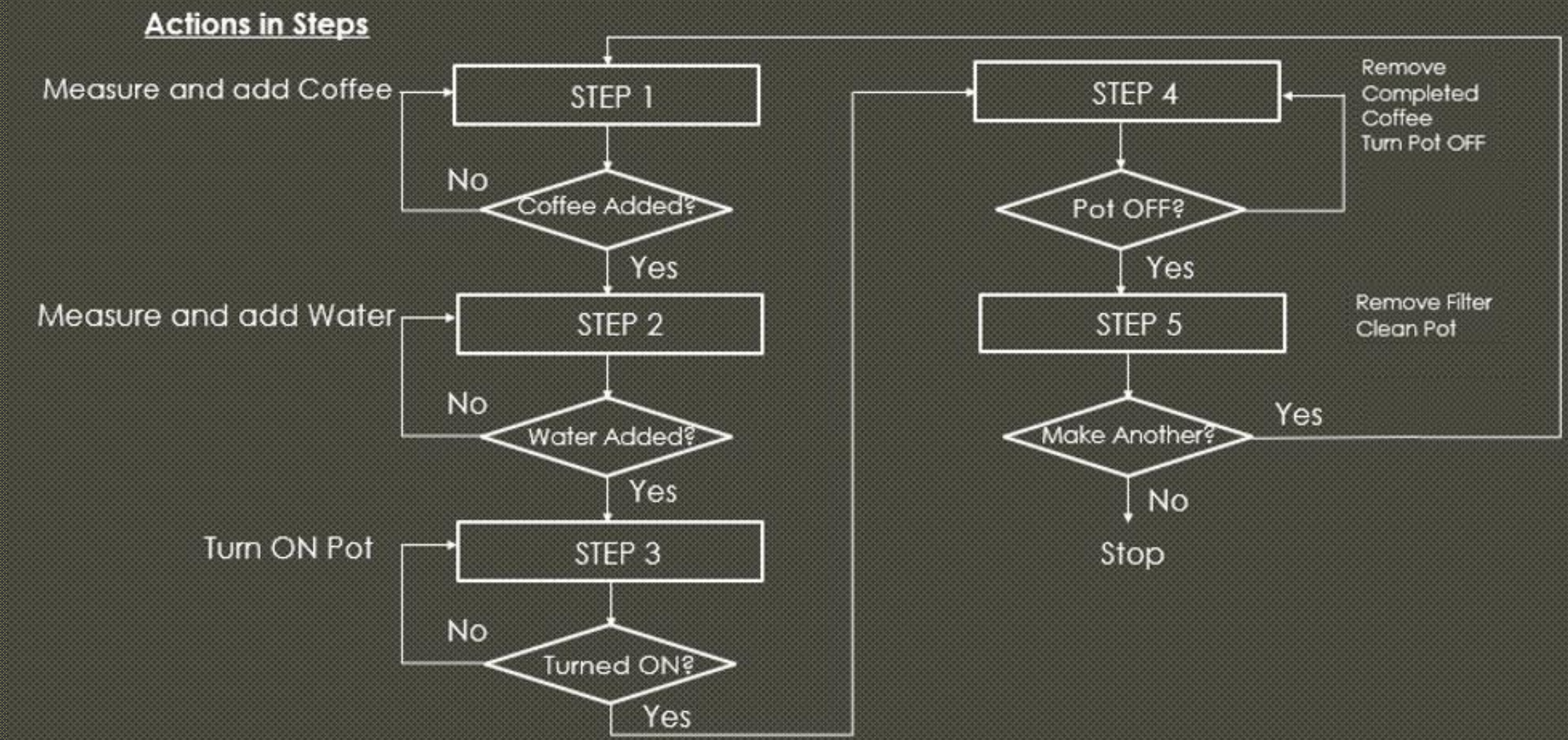
Rockwell Automation Publication 1756-PM006K-EN-P



https://www.realpars.com/blog/sequential-function-chart

HUMBER

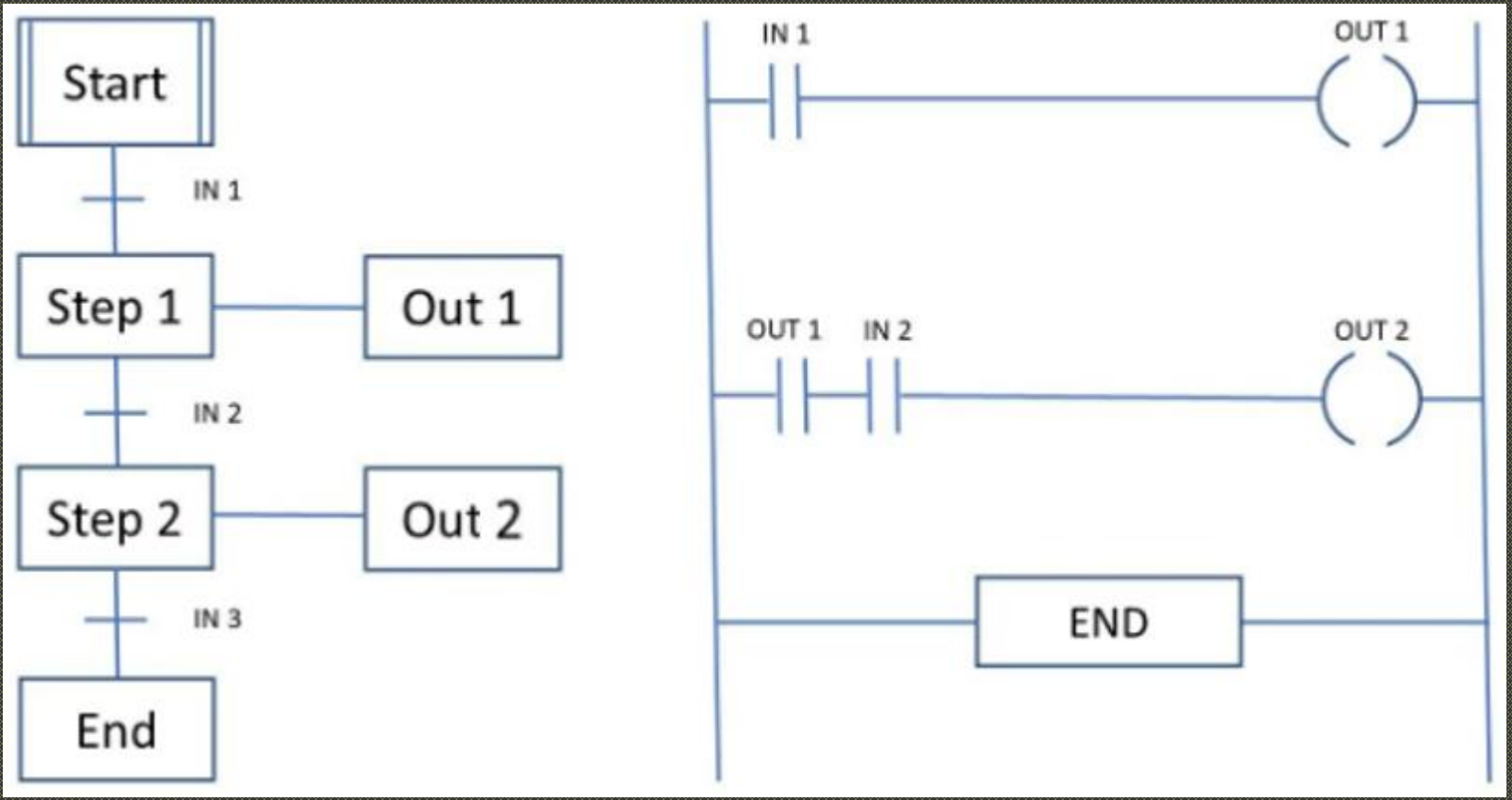
SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING



SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

SFCs can be summarized as a combination of:

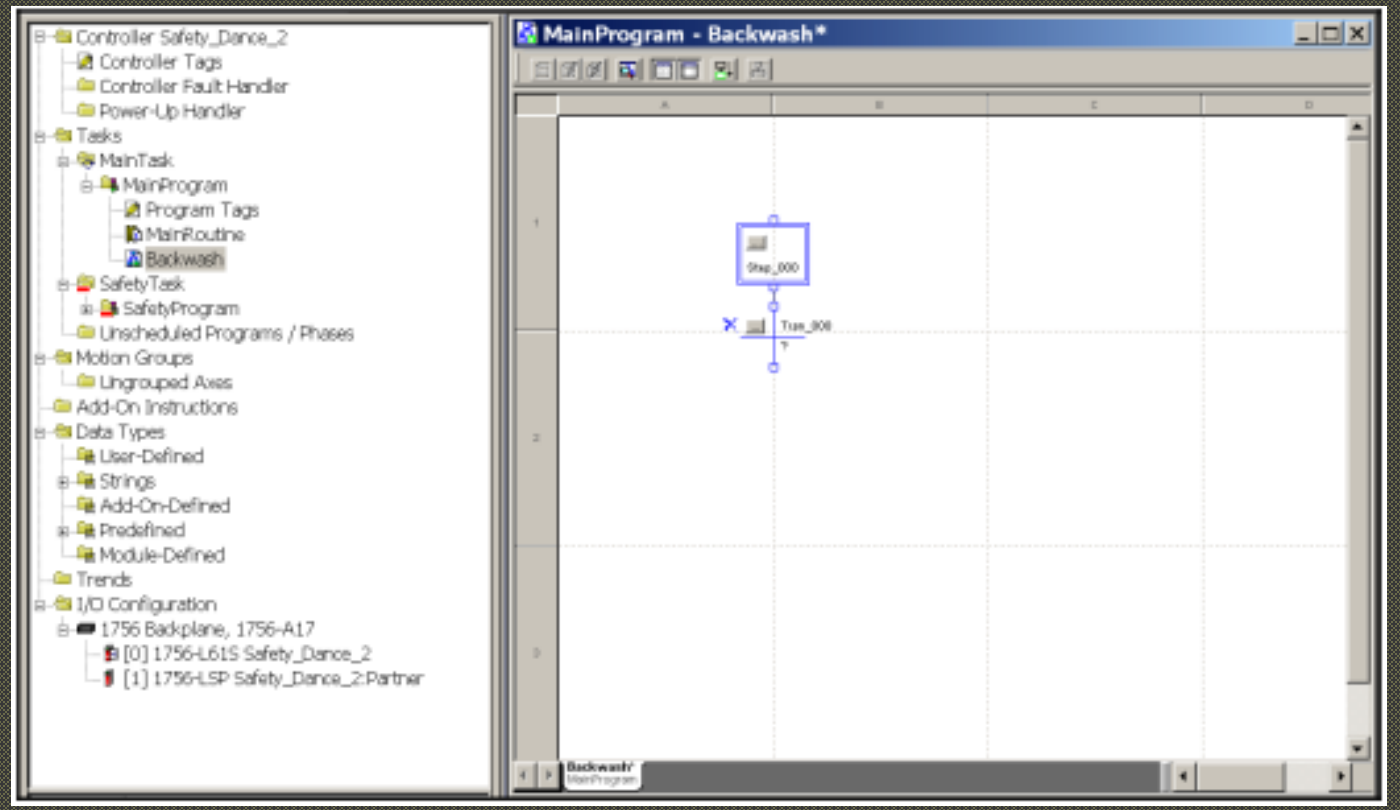
- Steps with corresponding actions
- Transitions with associated logic conditions
- Links between steps and transitions which are direct



https://control.com/technical-articles/an-overview-of-sequential-function-chart-sfc-programming/

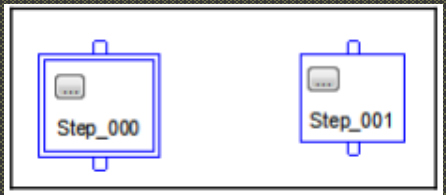
SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

- The SFC editor window allows you to add the SFC elements from the SFC element group, drag them around the sheet, and connect the flow of the SFC elements using wires.
- All SFC routines must have an initial step defined to run. The initial step is indicated with a double line around its rectangle shape.
- To specify which step is the initial step, right-click on it and check Initial Step.



Austin Scott, Learning RSLogix 5000 Programming

HUMBER

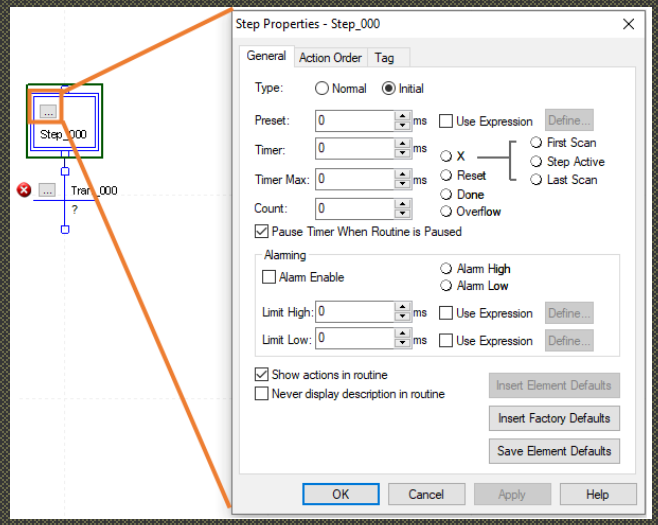


SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Step Properties

A step is associated with the logic to be executed at a specific point in the process.
The Step Properties window serves the dual purpose of programming the step and displaying its status when monitoring the SFC online.
Steps can be referenced just like Function Block elements and contain properties, as follows:

- **Type** – If the "Normal" is selected, the step is a normal step. If the "Initial" is selected, the step is designated as the initial SFC step.
- **Preset** – the step timer preset value in milliseconds. When the step .T value is at least this value, the step .DN bit is set.
- **Timer** – the current value of the step timer accumulator (.T value) in milliseconds, indicates how long the step has been active during this execution. Monitoring only.
- **Count** – the number of times the step has been active since this count was last reset. Monitoring only.
- **Timer Max** – monitoring only.
- **X** – indicator is blue when .X is on, indicating the step is active.
- **Step Active** – indicator is blue when step is active.
- **Done** – Indicator is blue when the step .DN bit is set.

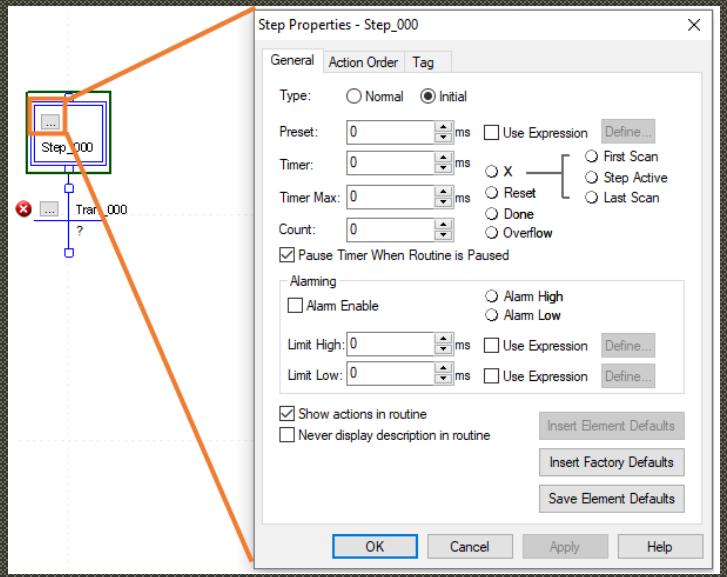


SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Step Status

The step status is stored as part of the step tag which is SFC_STEP structured data type.

Field Name		Description
.X	Step flag	ON when step is active
.DN	Timer done	ON when step timer accumulator exceeds preset
.PRE	Preset value	Timer preset value (ms)
.T	Step accumulation	Timer accumulator value (ms). Continues to run after reaching the preset value.



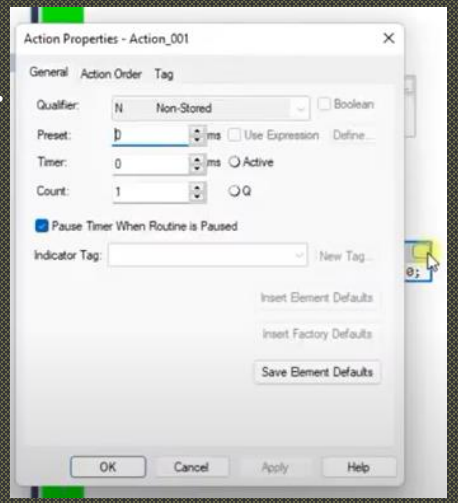
HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Step Actions

The step actions are added to the step and the step action properties window is used for specifying the general properties of the step action and showing the status of the action if monitoring the SFC online.

- **Qualifier** – one of the action qualifiers.
- **Boolean** – if checked, the action .Q bit is set when the step is active. Structured text cannot appear in the SFC action.
- **Preset** – specifies the time (in ms) for one of the action time-based qualifiers (L, D, SL, SD).
- Timer – the current value of the action timer accumulator (.T value) in milliseconds. Monitoring only.
- Count the number of times the action has been active since the count was last reset. Monitoring only.
- Active – indicator is blue when the action is active.



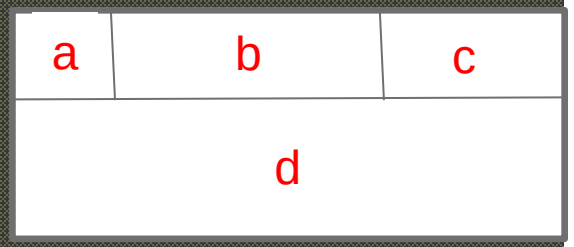
HUMBER

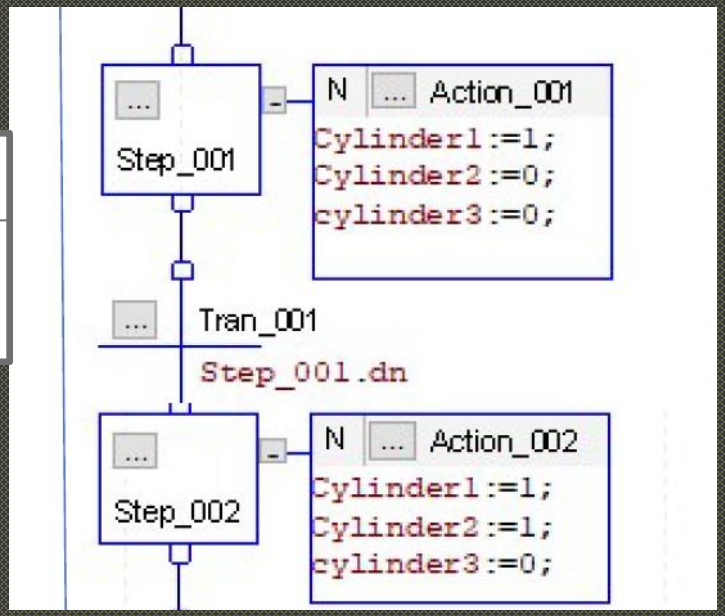
SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

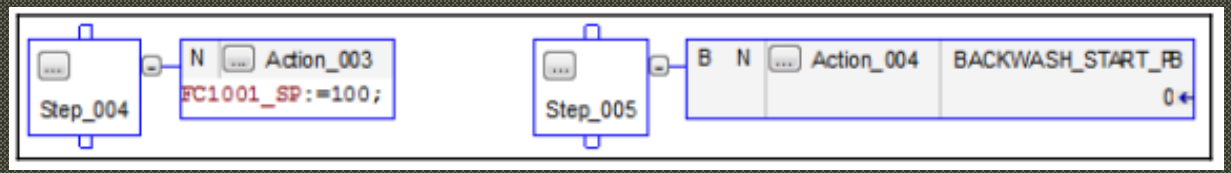
Action Blocks

Action blocks are associated with a step. Each step can have zero or more action blocks.

- An action can be a Boolean variable, a ladder logic diagram (LD), a function block diagram (FBD) or a collection of structured text (ST) statements.
- The action box is used to perform a process action such as opening a valve, starting a motor, calculating formula for the transition condition or executing a time delay.
- Each step action block may have up to four parts:
  a. action qualifier
  b. action name
  c. Boolean indicator variable
  d. action description using ST, LD, FBD, SFC.





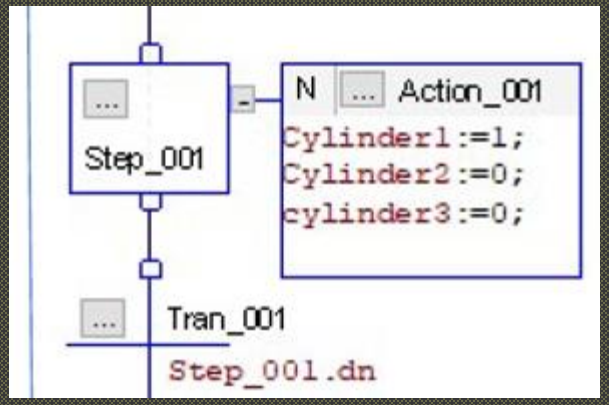


Austin Scott, Learning RSLogix 5000 Programming

HUMBER

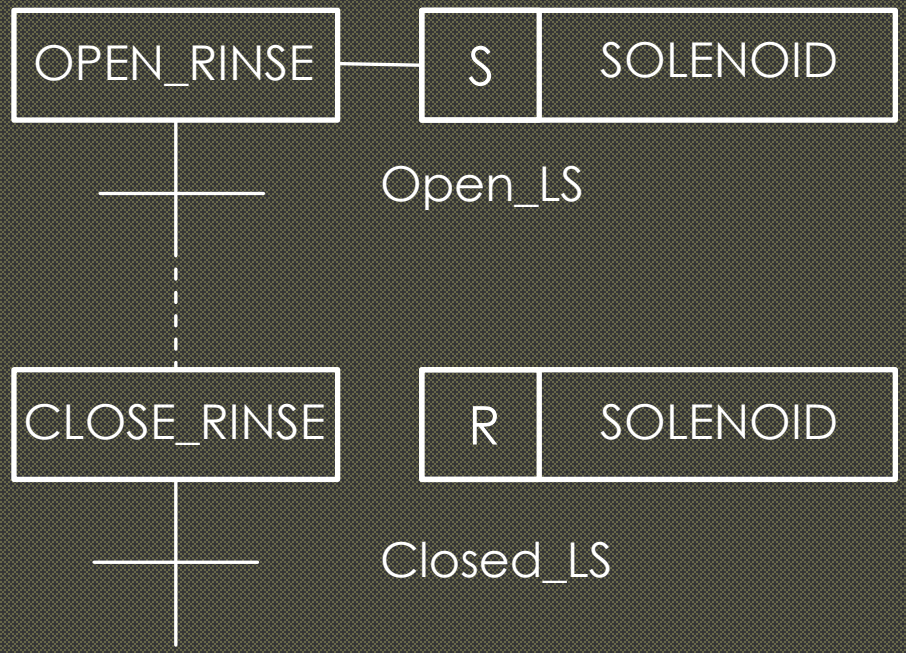
N-Non-stored action qualifier

- A non-stored action qualifier is active only when the step is active.
- The Action_001 executes continuously while the Step_001 step is active.
- When the transition Trans_001 turns ON, the Step_01 step becomes inactive and the Action_001 is turned OFF if it is Boolean variable. If Action_001 is an action described by one of the languages, as in the example below, deactivation of the step causes the action to execute one last time (postscan) to deactivate outputs and reset action logic.



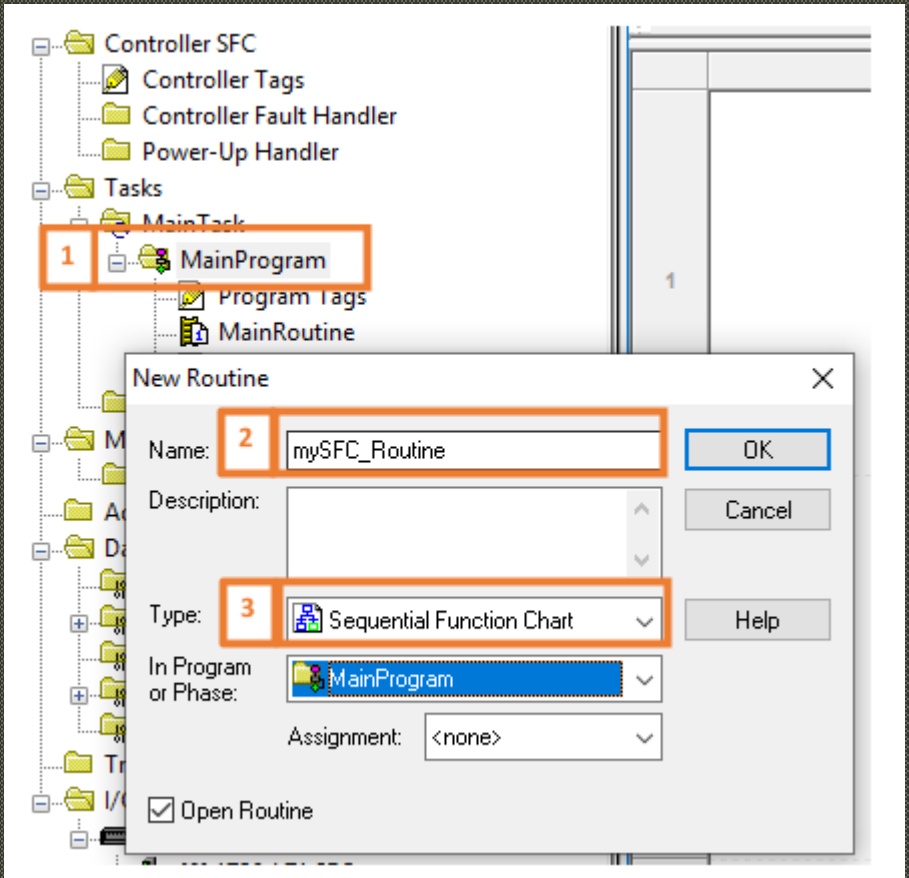
S and R Stored (Set) and Reset action qualifiers

- A stored action becomes active when the step becomes active.
- The action continues to be executed even after the step is inactive. To stop the action, another step must have an R qualifier that references the same action.



SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Creating a Sequential Function Chart routine.

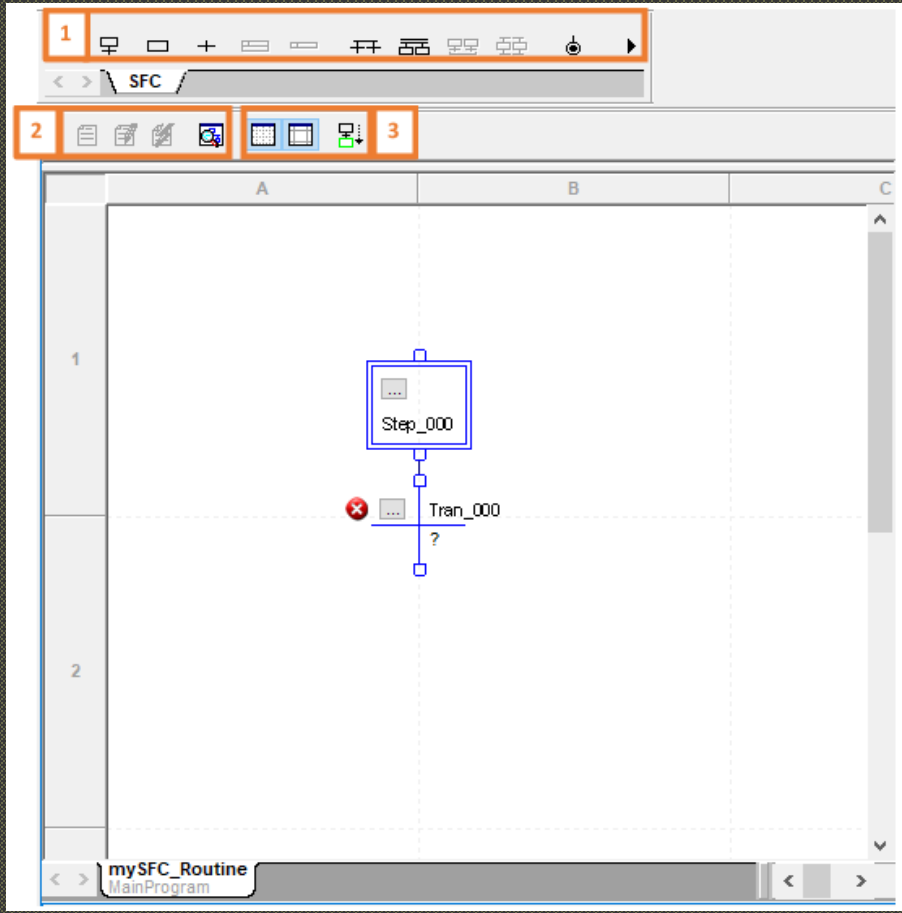


1. Right Click on the Program in which the routine needs to be created.
2. Type in a routine name.
3. Select from the drop-down and select Sequential Function Chart.

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Familiarizing with development environment.

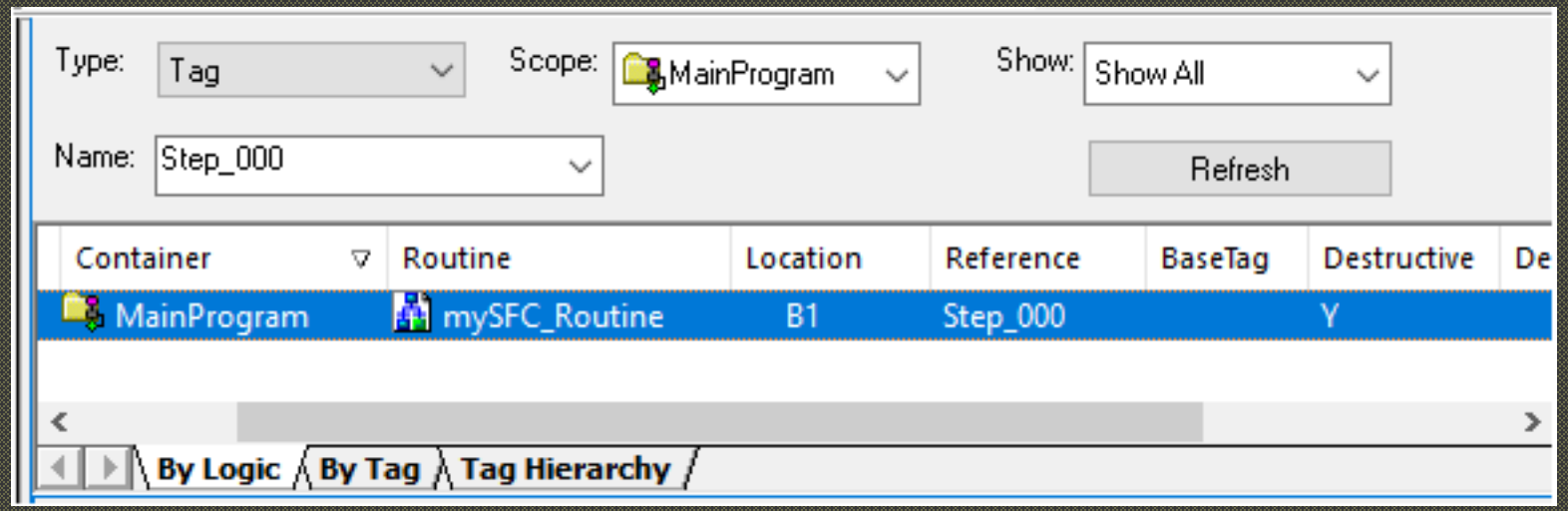


1. The SFC Element Toolbar is displayed.
2. The views available can be changed here between Original View, Pending Edits View, Test Edits View and Routine Overview.
3. The Grid and Sheet boundaries can be toggled between Show and Hide respectively. The Auto-Scroll feature, which when enabled allows the SFC routine to automatically scroll to the active step, can either be enabled or disabled.

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Cross-referencing a tag used in a Sequential Function Chart.

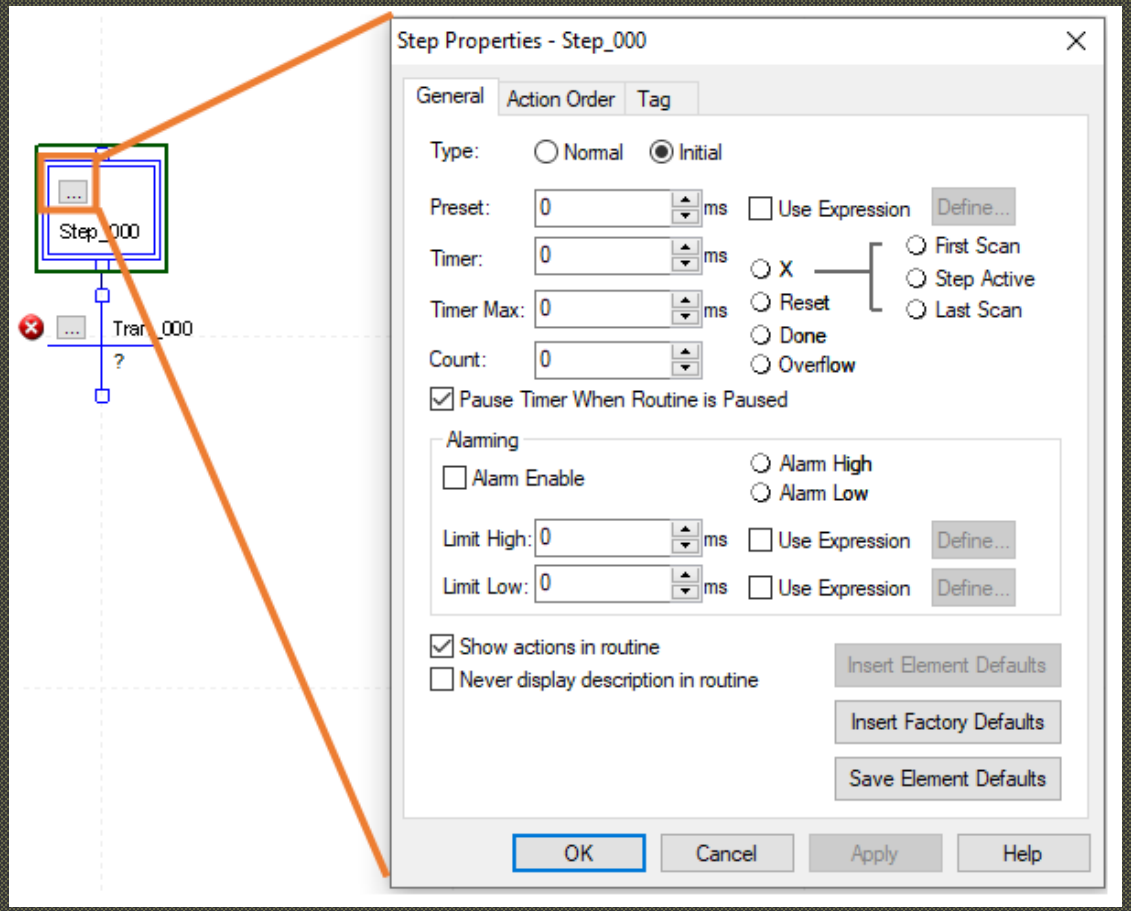


https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Step properties

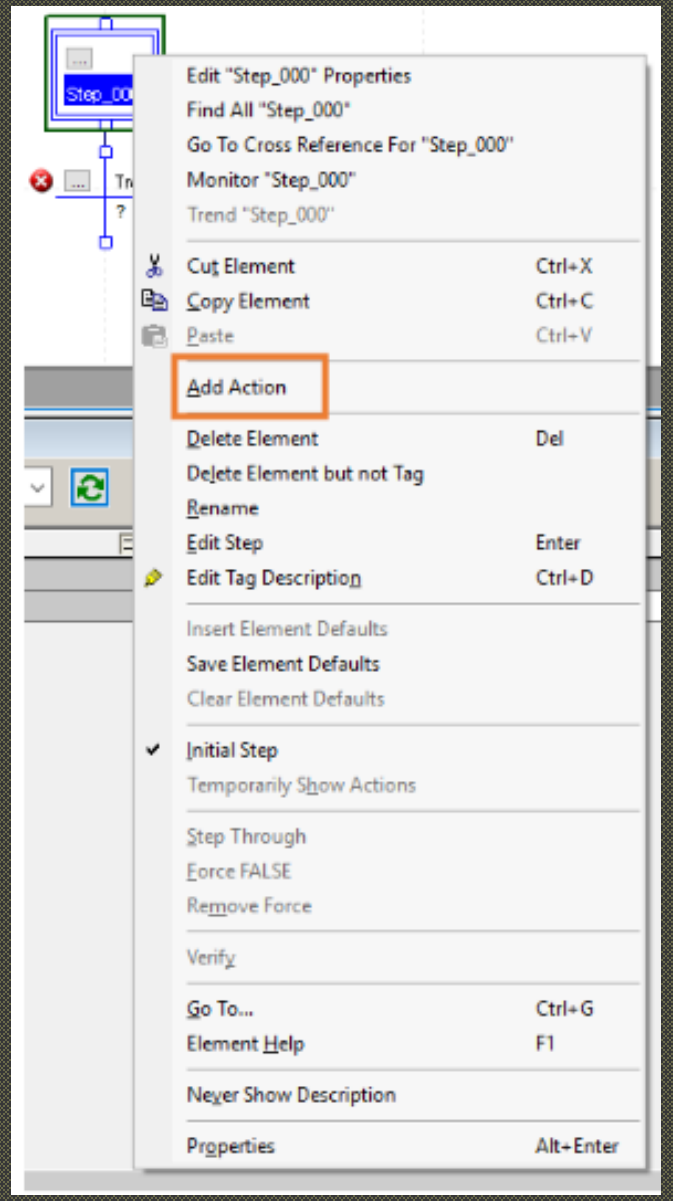


https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

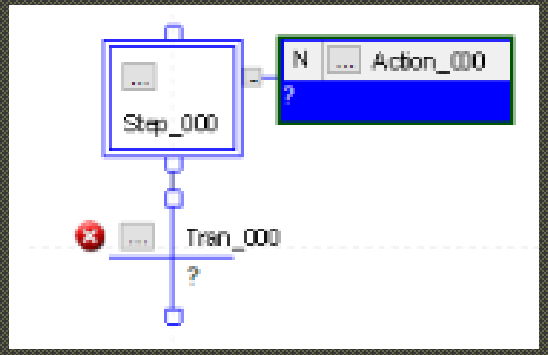
Right click on the step to add an action.



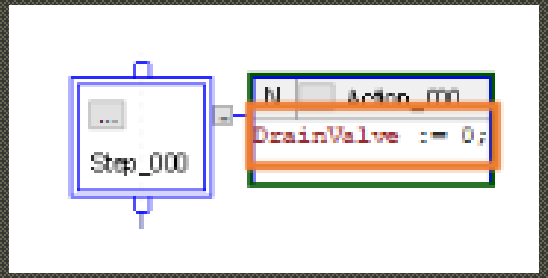
https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

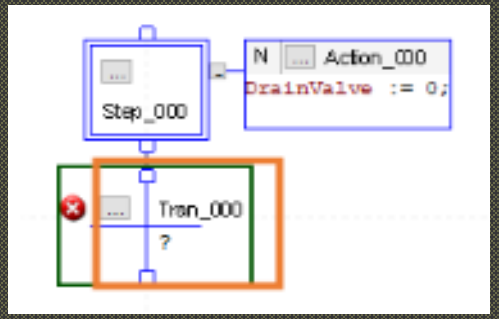
Action Element added
to Step element



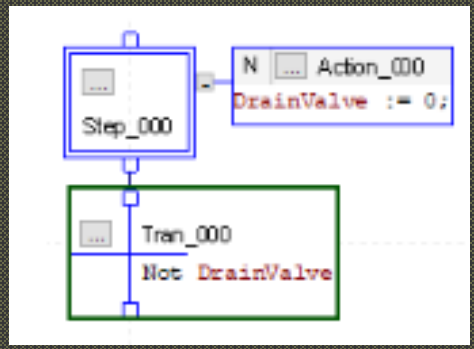
Structured Text in
Action Element added



Transition element added



Structured Text in
Transition Element added

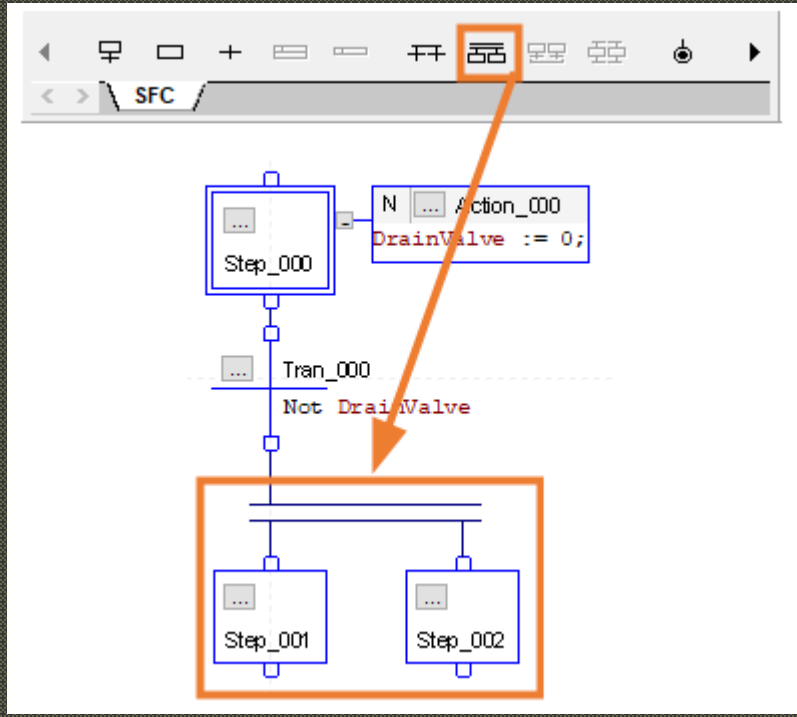


HUMBER

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Simultaneous Branch Diverge selection on the SFC Toolbar element added when two or more steps need to occur after a transition.

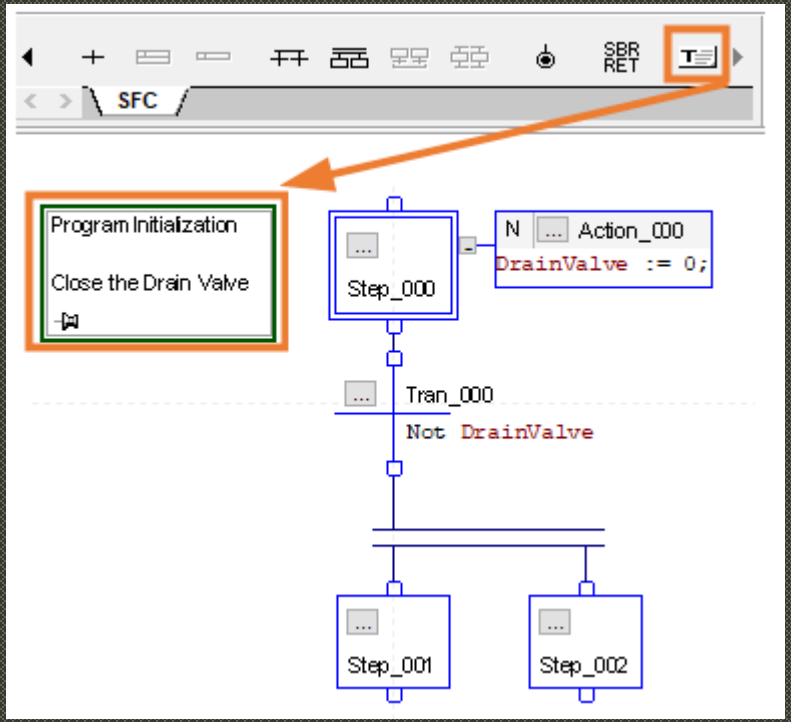


HUMBER

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Adding text box for comments

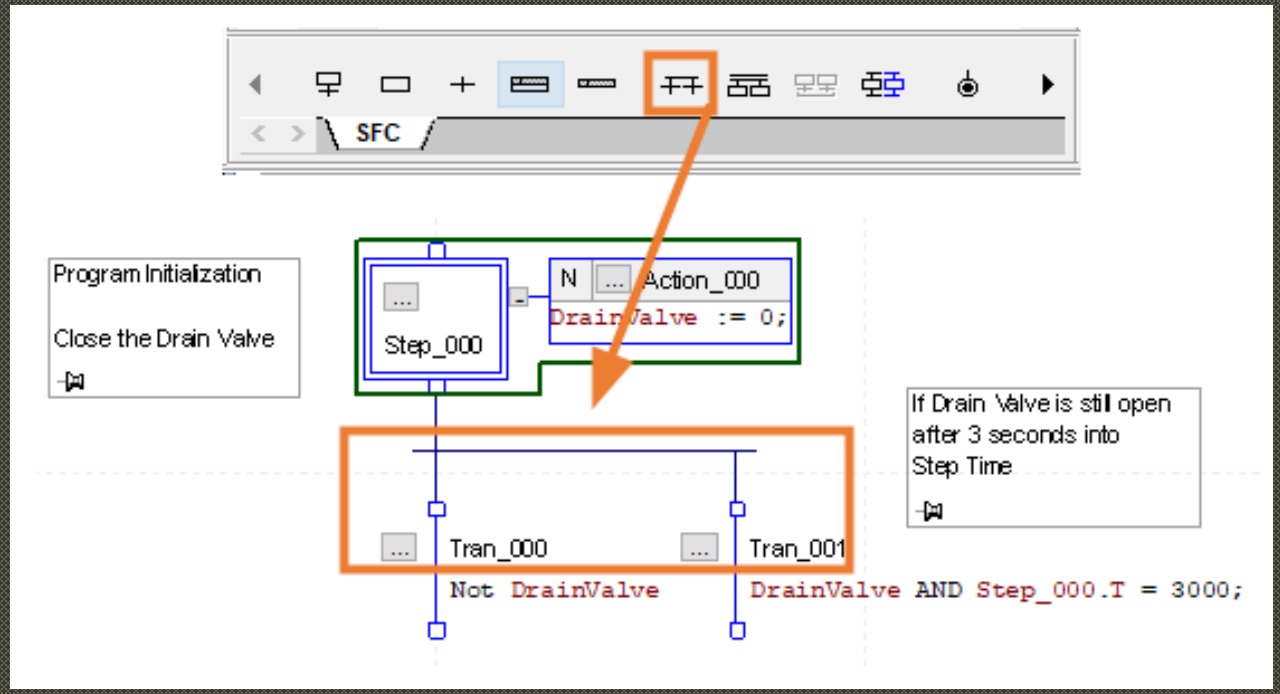


HUMBER

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

A single step may have multiple transitions.

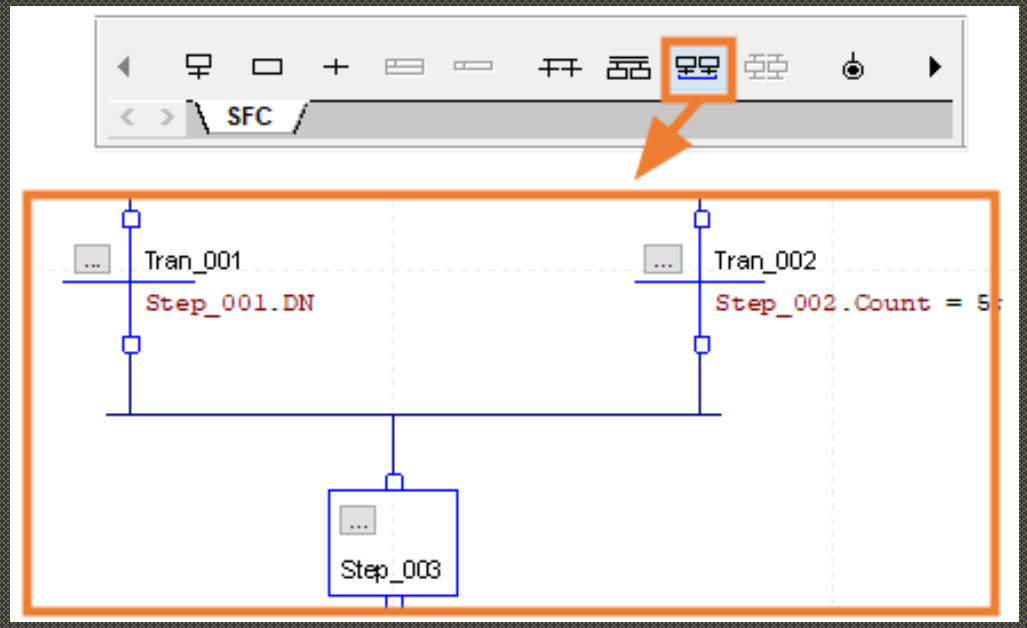


HUMBER

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Two different transition conditions result in the same next step.

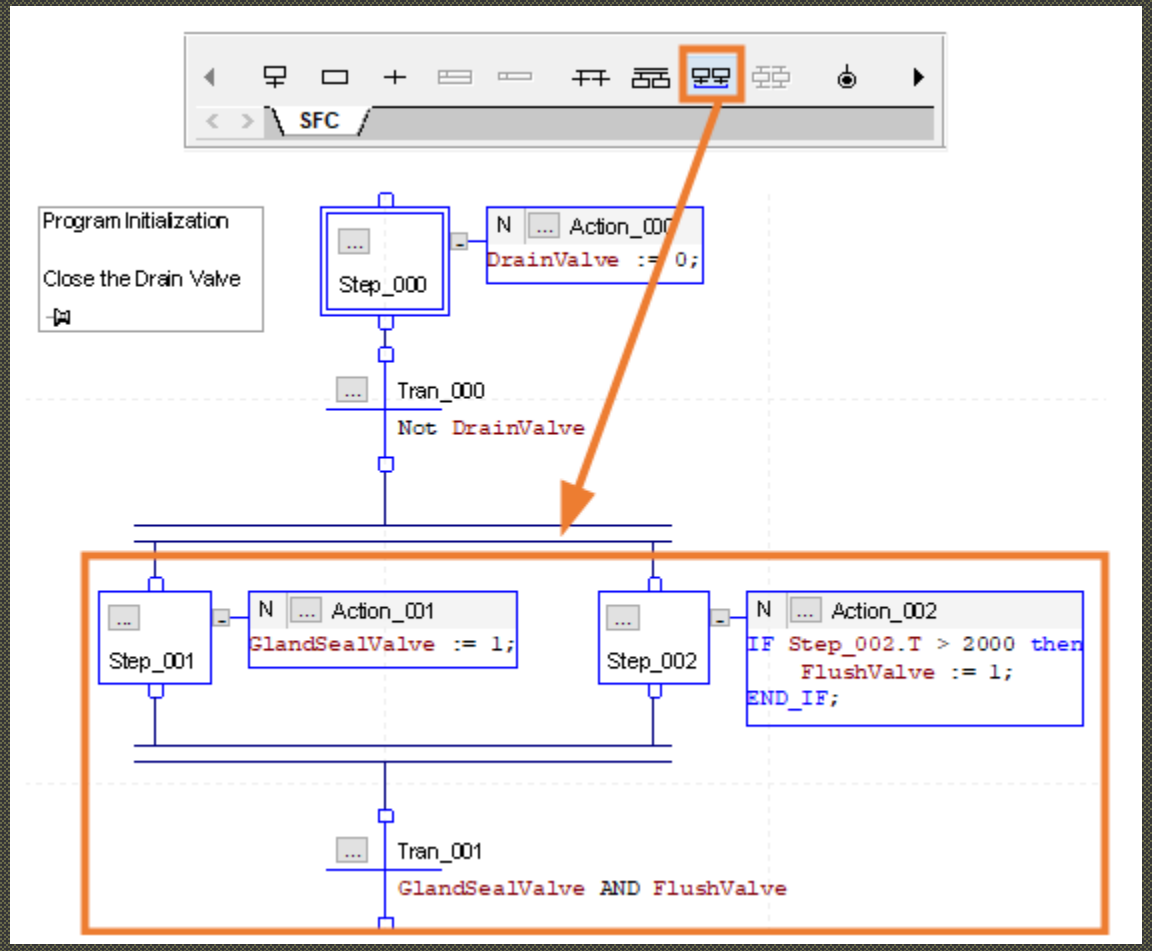


The conditions of the two transitions are different, but if either of these conditions should be TRUE, the respective transition will trigger the step irrespective of the other transition's state.

https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Multiple steps use the same transition.

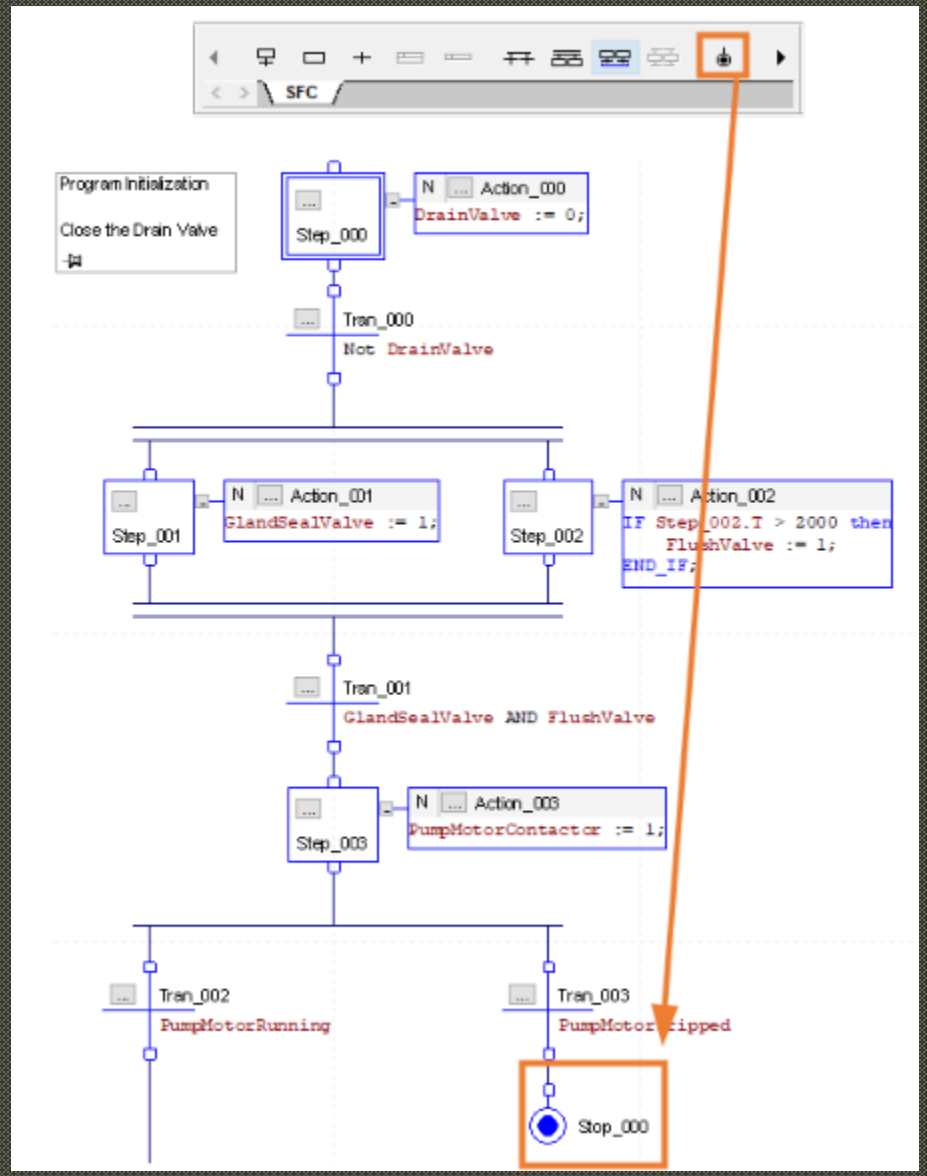


https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

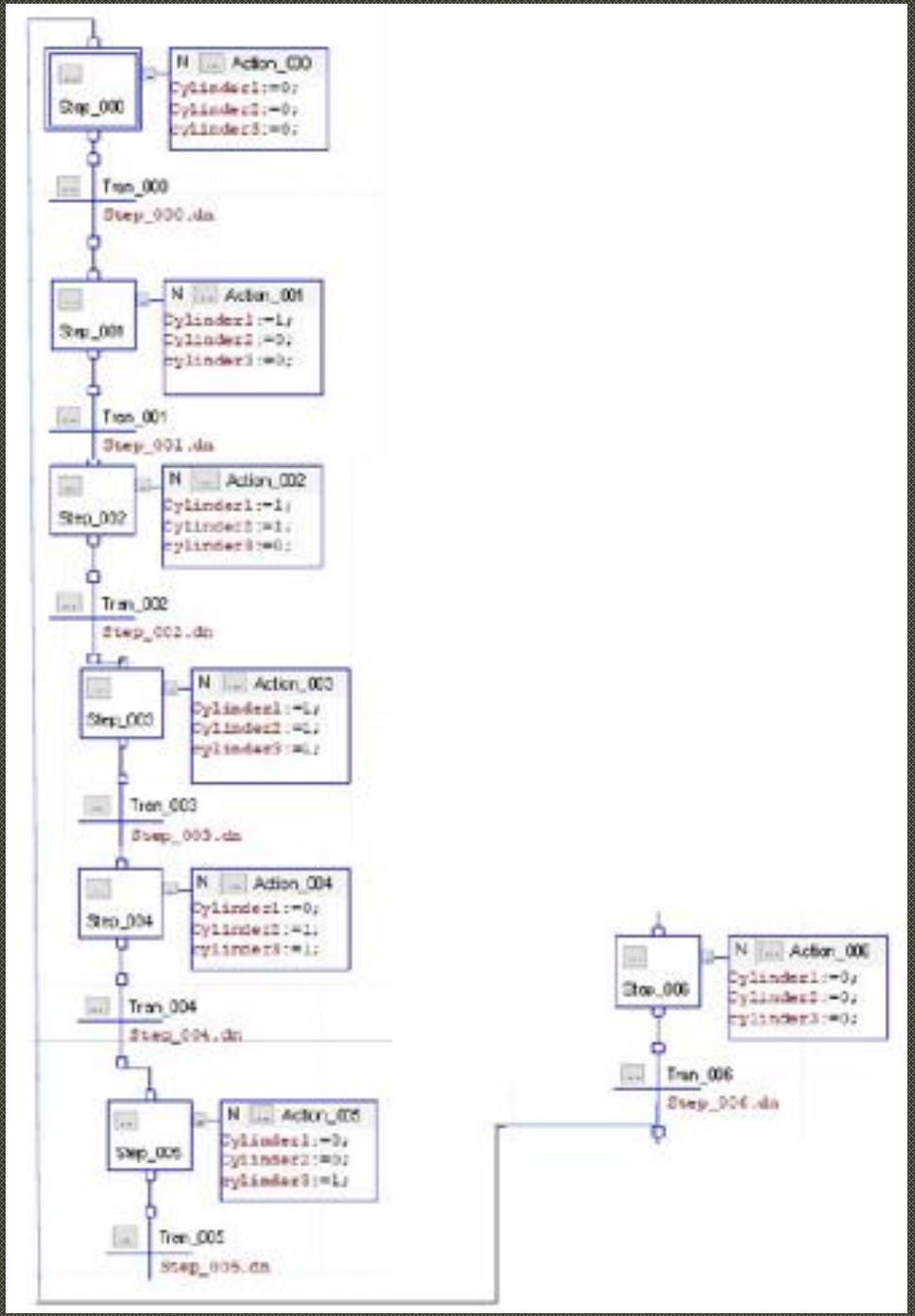
To achieve a continuous Sequential Function Chart, it is necessary connecting the last transition in the diagram to the initial step, thus creating a loop.

In some instances, this may not be the desired flow and stops may be used when and where necessary and may even be used multiple times throughout the routine.



https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

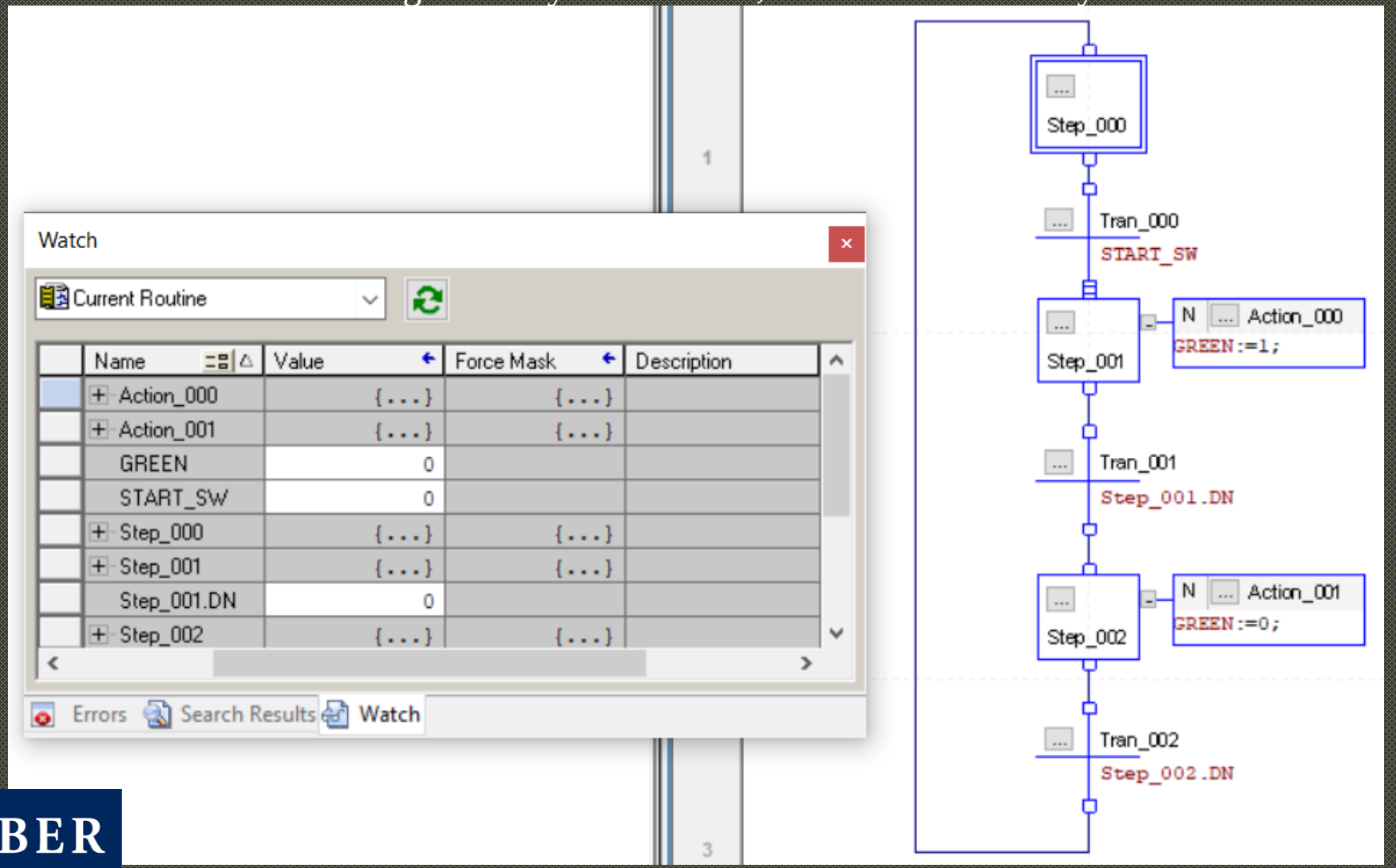


HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Programming a simple SFC

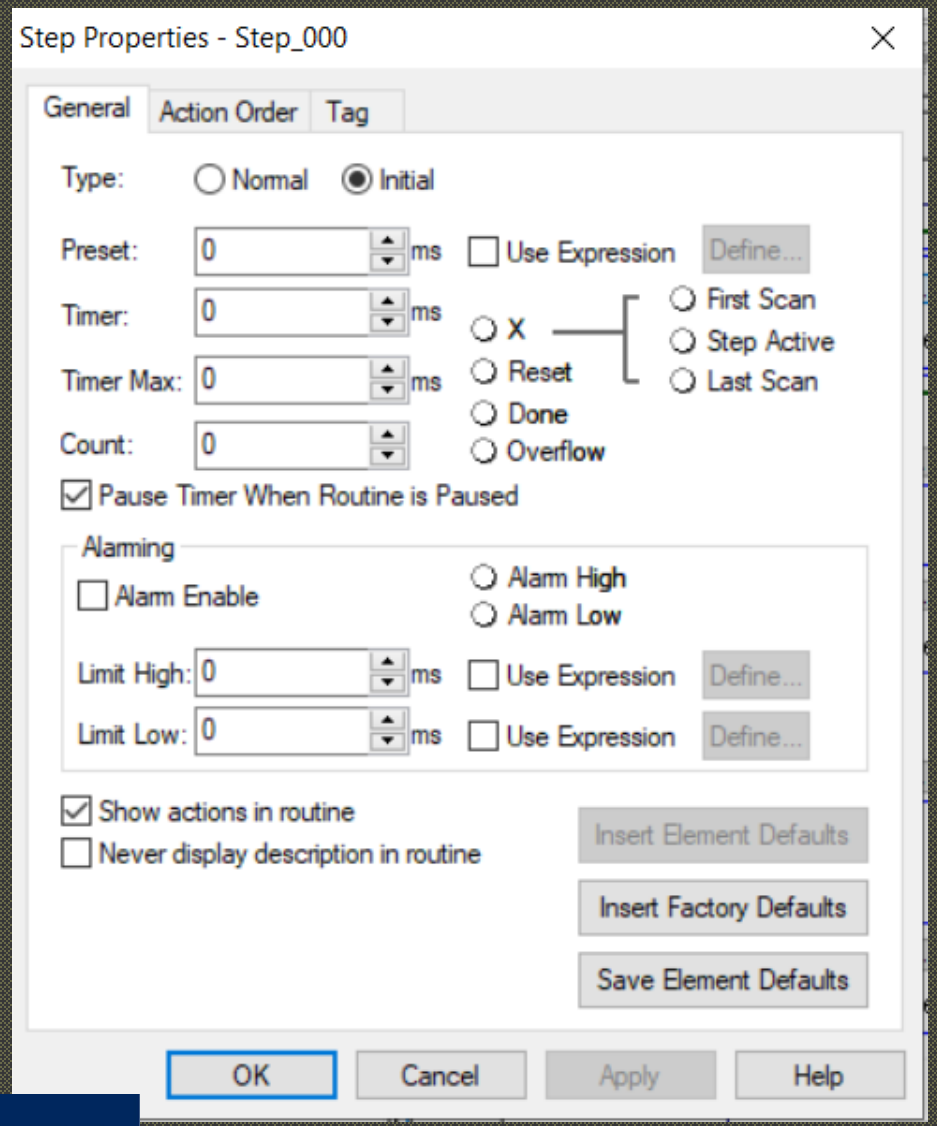
Create a SFC program to turn ON the GREEN light for 6 seconds and turn it OFF for 3 seconds using a START switch. While the switch is On the light will cycle ON-OFF, otherwise it will stay OFF.

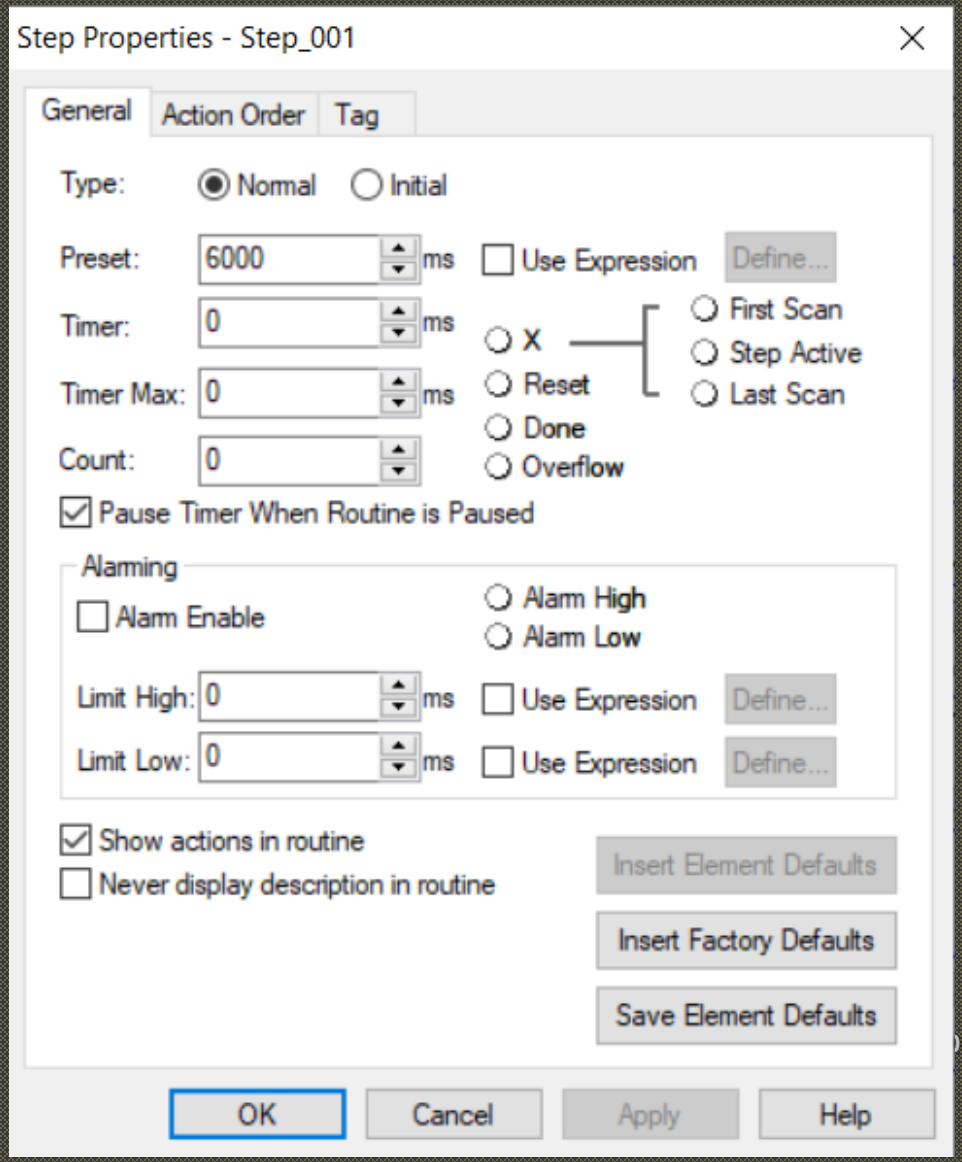


HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Programming a simple SFC

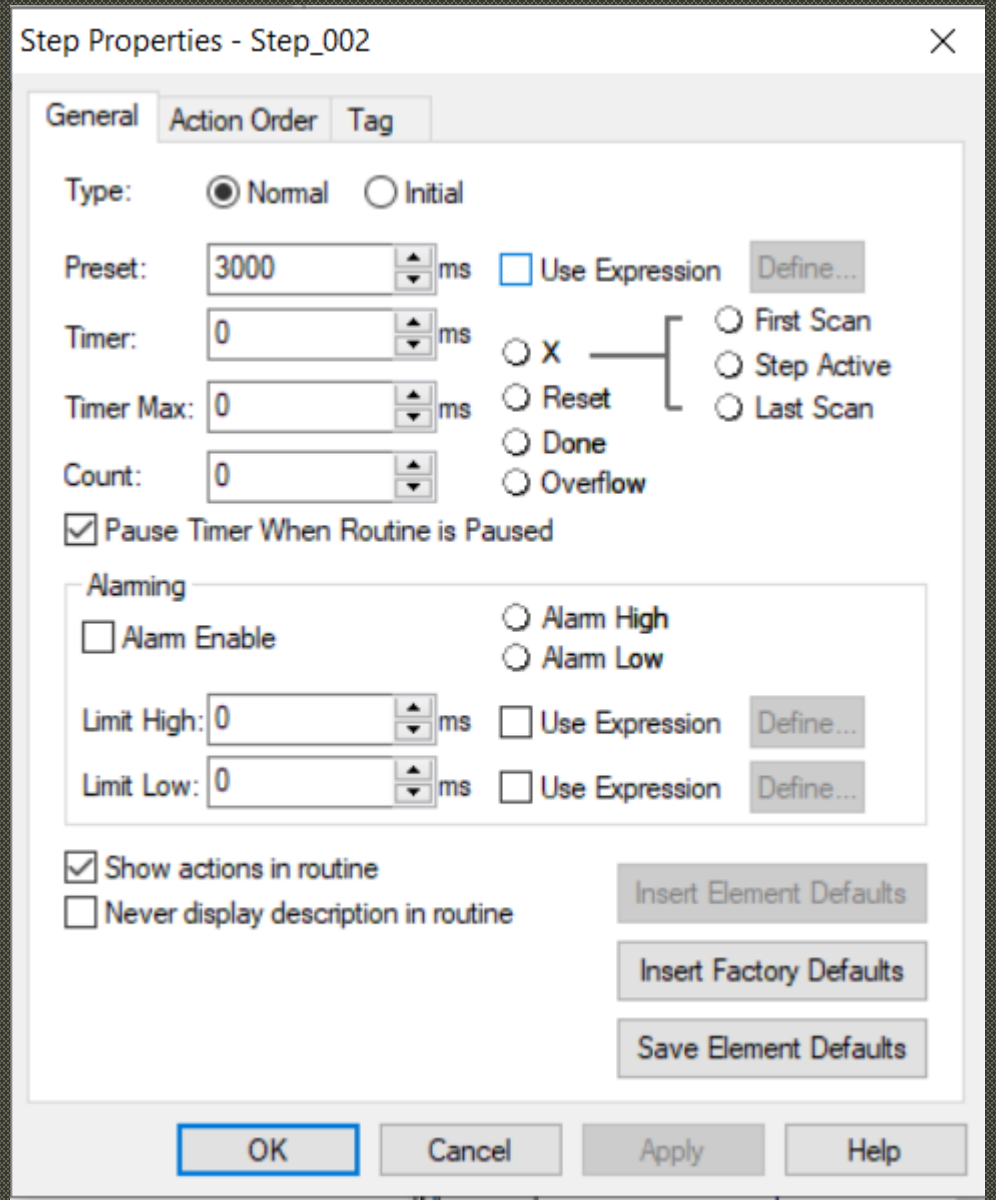




HUMBER

SEQUENTIAL FUNCTION CHART (SFC) PROGRAMMING

Programming a simple SFC



HUMBER

STRUCTURED TEXT, FUNCTION BLOCK DIAGRAM, SEQUENTIAL FUNCTION CHART PROGRAMMING

Erickson, K. (2016) Programmable logic controllers: An emphasis on design and application (3rd edition). Rolla MO: Dogwood Valley Press.

Chapter 11, 12, 14

Rehg, J.A, Sartori, G.J. (2007) Programmable Logic Controllers. Upper Saddle River, New Jersey, Pearson Prentice Hall.

Chapter 13, 15, 16

Thank you!

Discussions?