

# Modicon LMC058 Motion Controller

Pulse Width Modulation

LMC058 Expert I/O Library Guide

09/2020

---

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

You agree not to reproduce, other than for your own personal, noncommercial use, all or part of this document on any medium whatsoever without permission of Schneider Electric, given in writing. You also agree not to establish any hypertext links to this document or its content. Schneider Electric does not grant any right or license for the personal and noncommercial use of the document or its content, except for a non-exclusive license to consult it on an "as is" basis, at your own risk. All other rights are reserved.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2020 Schneider Electric. All rights reserved.

---

# Table of Contents

---



	<b>Safety Information</b> .....	5
	<b>About the Book</b> .....	7
<b>Chapter 1</b>	<b>Introduction</b> .....	11
	Expert I/O Overview .....	12
	Add an Expert function .....	15
	Embedded Expert I/O Mapping .....	18
<b>Chapter 2</b>	<b>Generalities</b> .....	21
	PWM Naming Convention .....	23
	Synchronization and Enable Functions .....	24
<b>Chapter 3</b>	<b>Pulse Width Modulation (PWM)</b> .....	25
	Description .....	26
	Configuration .....	28
	PWM_LMC058: Command a Pulse Width Modulation Signal .....	30
	Programming the PWM Function Block .....	32
<b>Chapter 4</b>	<b>Frequency Generator (FreqGen)</b> .....	35
	Description .....	36
	Configuration .....	37
	Frequency_Generator_LMC058: Commanding a Square Wave Signal .....	39
	Programming .....	41
<b>Appendices</b> .....		43
<b>Appendix A</b>	<b>General Information</b> .....	45
	Dedicated Features .....	46
	General Information on Function Block Management .....	47
<b>Appendix B</b>	<b>Function and Function Block Representation</b> .....	49
	Differences Between a Function and a Function Block .....	50
	How to Use a Function or a Function Block in IL Language .....	51
	How to Use a Function or a Function Block in ST Language .....	55
<b>Appendix C</b>	<b>Data Unit Types</b> .....	59
	EXPERT_ERR_TYPE: Type for Error Variable on an Expert Function Block .....	59
<b>Glossary</b> .....		61
<b>Index</b> .....		65



---

# Safety Information

---



## Important Information

### NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.



The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.



This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

## **DANGER**

**DANGER** indicates a hazardous situation which, if not avoided, **will result in death** or serious injury.

## **WARNING**

**WARNING** indicates a hazardous situation which, if not avoided, **could result in death** or serious injury.

## **CAUTION**

**CAUTION** indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

## **NOTICE**

**NOTICE** is used to address practices not related to physical injury.

---

## PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

---

# About the Book

---



## At a Glance

### Document Scope

This documentation shows you with the pulse width modulated output and frequency generator functions offered within the Modicon LMC058 Motion Controller.

This documentation also describes the data types and functions of the LMC058 PWM/FG library.

To use this manual, you must:

- Have a thorough understanding of the LMC058, including its design, functionality, and implementation within control systems.
- Be proficient in the use of the following IEC 61131-3 PLC programming languages:
  - Function Block Diagram (FBD)
  - Ladder Diagram (LD)
  - Structured Text (ST)
  - Instruction List (IL)
  - Sequential Function Chart (SFC)

Only DM72F• expert module and Encoder module can be used with the LMC058 PWM/FG library.

### Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V1.2.5.

The technical characteristics of the devices described in the present document also appear online.

To access the information online, go to the Schneider Electric home page

<https://www.se.com/ww/en/download/>.


The characteristics that are described in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

## Related Documents

Title of Documentation	Reference Number
Modicon LMC058 Logic Controller Programming Guide	<a href="#">EIO0000004165 (ENG)</a> <a href="#">EIO0000004166 (FRE)</a> <a href="#">EIO0000004167 (GER)</a> <a href="#">EIO0000004168 (SPA)</a> <a href="#">EIO0000004169 (ITA)</a> <a href="#">EIO0000004170 (CHS)</a>
Modicon LMC058 Logic Controller Hardware Guide	<a href="#">EIO0000004189 (ENG)</a> <a href="#">EIO0000004190 (FRE)</a> <a href="#">EIO0000004191 (Ger)</a> <a href="#">EIO0000004192 (SPA)</a> <a href="#">EIO0000004193 (ITA)</a> <a href="#">EIO0000004194 (CHS)</a>

You can download these technical publications and other technical information from our website at <https://www.se.com/ww/en/download/> .

## Product Related Information

 <b>WARNING</b>
<b>LOSS OF CONTROL</b> <ul style="list-style-type: none"><li>• The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.</li><li>• Separate or redundant control paths must be provided for critical control functions.</li><li>• System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.</li><li>• Observe all accident prevention regulations and local safety guidelines.<sup>1</sup></li><li>• Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.</li></ul> <b>Failure to follow these instructions can result in death, serious injury, or equipment damage.</b>

<sup>1</sup> For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.



## WARNING

### UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

### Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2015	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2015	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2016	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.

---

Standard	Description
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive (2006/42/EC)* and *ISO 12100:2010*.

**NOTE:** The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

---

# Chapter 1

## Introduction

---

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Expert I/O Overview	12
Add an Expert function	15
Embedded Expert I/O Mapping	18

## Expert I/O Overview

### Introduction

The controller base provides:

- 2 embedded expert I/O modules (DM72F0 and DM72F1) with:
  - 5 fast inputs
  - 2 regular inputs
  - 2 fast outputs
- 1 Hardware Encoder port that can support:
  - Incremental encoder
  - SSI absolute encoder
- 1 Controller Power Distribution Module (CPDM)

Each embedded expert I/O module (DM72F\*) can support expert functions (*see page 15*).

### Embedded Expert I/O Configuration

To configure the expert I/Os, double-click the **Expert** node in the **Devices tree**.

This figure presents the configuration tab screenshot:

Parameter	Type	Value	Default Value	Unit	Description
Run/Stop Input	Enumeration of BYTE	BLOCK0_I0	None		
Alarm Output	Enumeration of BYTE	BLOCK0_Q1	None		
Rearming Output Mode	Enumeration of BYTE	Auto	Auto		

This table presents the function of the different parameters:

Parameter	Function
Run/Stop Input	Define one input to be used as Run/Stop input ( <i>see page 13</i> ).
Alarm Output	Define one output to be used as alarm output ( <i>see page 13</i> ).
Rearming Output Mode	Define the rearming output mode ( <i>see page 14</i> ).

## Run/Stop Input

This table presents the different states:

Input states	Result
State 0	Stops the controller and ignores external Run commands.
A rising edge	From the STOPPED state, initiates a start-up of an application in RUNNING state.
State 1	The application can be controlled by: <ul style="list-style-type: none"> <li>● EcoStruxure Machine Expert (Run/Stop)</li> <li>● the application (Controller command)</li> <li>● a network command</li> </ul>

**NOTE:** Run/Stop input is managed even if the option **Update IO while in stop** is not selected in the PLC settings tab (see *Modicon LMC058 Motion Controller, Programming Guide*).

Inputs assigned to configured expert functions can not be configured as Run/Stop.

For further details about controller states and states transitions, refer to Controller State Diagram (see *Modicon LMC058 Motion Controller, Programming Guide*).

### WARNING

#### UNINTENDED MACHINE OR PROCESS START-UP

- Verify the state of security of your machine or process environment before applying power to the Run/Stop input.
- Use the Run/Stop input to help prevent the unintentional start-up from a remote location.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

## Alarm Output

This output is set logical 1 when the controller is in the RUNNING state and the application program is not stopped at a breakpoint.

Outputs assigned to configured expert functions can not be configured as the Alarm output.

#### NOTE:

The alarm output is set to 0 when:

- a task is stopped at a breakpoint, the alarm output signals that the controller has stopped executing the application.
- an error is detected on the expert I/O (power interruption, short-circuit detection).

## Rearming Output Mode

Fast outputs of DM72F• modules are push/pull technology. In the case of an error detected (short-circuit or over temperature), the output is put in tri-state and the condition is signaled by status bit (DM72F• channel IB1.0) and PLC\_R.i\_wLocalIOStatus (refer to the Modicon LMC058 Motion Controller System Functions and Variables PLCSystem Library Guide).

Two behaviors are possible:

- **Automatic rearming:** as soon as the detected error is corrected, the output is set again according to the current value assigned to it and the diagnostic value is reset.
- **Manual rearming:** when an error is detected, the status is memorized and the output is forced to tri-state until user manually clears the status (see I/O mapping channel).

In the case of a short-circuit or current overload, the common group of outputs automatically enters into thermal protection mode (all outputs in the group are set to 0), and are then periodically rearmed (each second) to test the connection state. However, you must be aware of the effect of this rearming on the machine or process being controlled.

### WARNING

#### UNINTENDED MACHINE START-UP

Inhibit the automatic rearming of outputs if this feature is an undesirable behavior for your machine or process.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

---

## Add an Expert function

### Introduction

Each DM72F• expert module can support expert functions. Expert functions are defined as either simple or complex. Only one type can be configured per module:

- simple functions:
  - High Speed Counter Simple
  - Event\_Latch I/O
- complex functions:
  - High Speed Counter Main
  - Encoder
  - Frequency Generator (FreqGen)
  - Pulse Width Modulation (PWM)

When an I/O is not used by an expert function, it can be used as a regular I/O.

#### NOTE:

- When a regular input is used as Run/Stop, it can not be used by an expert function.
- When a regular output is used as Alarm, it can not be used by an expert function.

For more details, refer to Embedded expert I/O Configuration (*see page 12*).

### Adding an Expert Function

To add an Expert function (Event\_Latch, HSC, PWM or Frequency Generator) to your controller, select the Expert function you want in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

For more information on adding a device to your project, refer to:

- Using the Drag-and-drop Method
- Using the Contextual Menu or Plus Button

To add an Encoder function, select the **Standard Encoder** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

The following expert functions can be added:

Function	Description	Refer to...
Event_Latch	With the Event_Latch function, the Embedded Expert inputs can be configured as event or latch.	Event_Latch configuration (see <i>Modicon M258 Logic Controller, Programming Guide</i> )
HSC	The HSC functions can execute fast counts of pulses from sensors, encoders, switches, etc. that are connected to dedicated fast inputs.	LMC058 HSC Library (see <i>Modicon LMC058 Motion Controller, High Speed Counting, LMC058 Expert I/O Library Guide</i> )
PWM Frequency Generator	The PWM function generates a square wave signal on dedicated output channels with a variable duty cycle. The Frequency Generator function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).	LMC058 PWM library
Encoder	The goal of this function is to connect an encoder to acquire a position. This function can be implemented on an Embedded Expert I/O interface and an Hardware Encoder interface. The Encoder can be Incremental or absolute SSI on an Hardware Encoder interface. The Embedded Expert I/O interface supports only an Incremental Encoder. You can configure a linear or rotary axis for incremental encoder.	LMC058 HSC Library (see <i>Modicon LMC058 Motion Controller, High Speed Counting, LMC058 Expert I/O Library Guide</i> )

### Expert Function Assignment

Expert functions assignment according to the interface (columns exclude each other):

I/F Interface	Expert Functions					
	Simple functions: ● Fast I/O: Event or latched ● HSC Simple	HSC_Main	SM_Encoder	Encoder	PWM	Frequency Generator
DM72F0	Up to 4	1	1	1	1	1
DM72F1	Up to 4	1	1	1	1	1
Encoder	Not allowed	Not allowed	1	1	Not allowed	Not allowed

For more details, refer to Expert I/O Mapping (*see page 18*).



### Expert Function I/O within Regular I/O

Expert Function I/O within Regular I/O:

- Inputs can be read through memory variable standard even if configured in expert function.
- An Input can not be configured in an expert function if it has already been configured as a Run/Stop.
- An Output can not be configured in an expert function if it has already been configured as an Alarm.
- %Q will not have any impact on reflex output.
- Short-Circuit management still applies on all outputs. Status of outputs are available.
- All I/O that are not used by expert functions are available as fast or regular I/O.

When inputs are used in expert functions (Latch, HSC,...), integrator filter is replaced by anti-bounce filter (*see Modicon LMC058, Motion Controller, Hardware Guide*). Filter value will be configured in expert function screen.

## Embedded Expert I/O Mapping

### I/O Mapping for Expert Functions on DM72F-

Embedded Expert I/O mapping by expert function:

		I0	I1	I2	I3	I4	I5	Q0	Q1
Event_Latch 0/4	Input	M							
Event_Latch 1/5	Input		M						
Event_Latch 2/6	Input			M					
Event_Latch 3/7	Input				M				
HSC Simple 0/4	Input A	M							
HSC Simple 1/5	Input A		M						
HSC Simple 2/6	Input A			M					
HSC Simple 3/7	Input A				M				
HSC Main 0/1	Input A	M							
	Input B		C						
	SYNC			C					
	CAP				C				
	EN					C			
	REF						C		
	Outputs							C	C
PWM 0/1	Outputs							M	
	SYNC			C					
	EN					C			
Frequency Generator 0/1	Outputs							M	
	SYNC			C					
	EN					C			
Standard Encoder	Input A	M							
	Input B		M						
	SYNC			C					
	CAP				C				
	EN					C			
	REF						C		
	Outputs							C	C
<b>M</b> Mandatory <b>C</b> Depends on Configuration									

		I0	I1	I2	I3	I4	I5	Q0	Q1
Motion Encoder	Input A	M							
	Input B		M						
	Input Z			M					
	CAP				C				
<b>M</b> Mandatory <b>C</b> Depends on Configuration									

**NOTE:** The DM72F• I6 inputs can only be configured by encoder on ENC.

## IO Summary

The **IO Summary** window displays the DM72F• I/O and the I/O used by expert functions.

The **IO Summary** window is accessible from **DM72F•** nodes:

Step	Action
1	In the <b>Devices tree</b> tab, expand the <b>Expert</b> node.
2	Right-click <b>DM72F•</b> and select <b>IO Summary</b> in context menu.

Example of IO Summary:

The screenshot shows the 'IO Summary' window with two panes: 'Inputs' and 'Outputs'. Each pane contains a table with columns for Channel, Address, and Utilization.

Inputs			Outputs		
Channel	Address	Utilization	Channel	Address	Utilization
DM72F0 - I0	%IX1.0	HSCMain_1 - A input, DM72F0 - Filter	DM72F0 - Q0	%QX0.0	HSCMain_1 - Reflex 0 Output
DM72F0 - I1	%IX1.1	DM72F0 - Filter	DM72F0 - Q1	%QX0.1	HSCMain_1 - Reflex 1 Output
DM72F0 - I2	%IX1.2	HSCMain_1 - SYNC, DM72F0 - Filter	DM72F1 - Q0	%QX1.0	HSCMain - Reflex 0 Output
DM72F0 - I3	%IX1.3	HSCMain_1 - CAP, DM72F0 - Filter	DM72F1 - Q1	%QX1.1	HSCMain - Reflex 1 Output
DM72F0 - I4	%IX1.4	HSCMain_1 - EN, DM72F0 - Filter			
DM72F0 - I5	%IX1.5	DM72F0 - Filter			
DM72F0 - I6	%IX1.6	DM72F0 - Filter			
DM72F0 - I0	%IX2.0	DM72F0 - Shortcut Detection			
DM72F1 - I0	%IX3.0	HSCMain_1 - A input, DM72F1 - Filter			
DM72F1 - I1	%IX3.1	DM72F1 - Filter			
DM72F1 - I2	%IX3.2	HSCMain_1 - SYNC, DM72F1 - Filter			
DM72F1 - I3	%IX3.3	HSCMain_1 - CAP, DM72F1 - Filter			
DM72F1 - I4	%IX3.4	HSCMain_1 - EN, DM72F1 - Filter			
DM72F1 - I5	%IX3.5	DM72F1 - Filter			
DM72F1 - I6	%IX3.6	DM72F1 - Filter			
DM72F1 - I0	%IX4.0	DM72F1 - Shortcut Detection			

A 'Close' button is located at the bottom right of the window.



---

# Chapter 2

## Generalities

---

### Overview

This chapter provides general information of the Pulse Width Modulation (PWM) functions.

The functions provide simple, yet powerful solutions for your application. In particular, they are useful for controlling movement. However, the use and application of the information contained herein require expertise in the design and programming of automated control systems. Only you, the user, machine builder or integrator, can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or related processes, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, you must also consider any applicable local, regional, or national standards and/or regulations.

### WARNING

#### REGULATORY INCOMPATIBILITY

Ensure that all equipment applied and systems designed comply with all applicable local, regional, and national regulations and standards.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

The functions provided by the expert functions libraries were conceived and designed assuming that you incorporate the necessary safety hardware into your application architecture, such as, but not limited to, appropriate limit switches and emergency stop hardware and controlling circuitry. It is implicitly assumed that functional safety measures are present in your machine design to prevent undesirable machine behavior such as over-travel or other forms of uncontrolled movement. Further, it is assumed that you have performed a functional safety analysis and risk assessment appropriate to your machine or process.

### WARNING

#### UNINTENDED EQUIPMENT OPERATION

Ensure that a risk assessment is conducted and respected according to EN/ISO 12100 during the design of your machine.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
PWM Naming Convention	23
Synchronization and Enable Functions	24

## PWM Naming Convention

### Definition

In this document, use the following naming convention:

Name	Description
SYNC	Synchronization function ( <i>see page 24</i> ).
EN	Enable function ( <i>see page 24</i> ).
IN_SYNC	Physical input dedicated to the SYNC function.
IN_EN	Physical input dedicated to the EN function.
OUT_PWM	Physical output dedicated to the PWM.

## Synchronization and Enable Functions

### Introduction

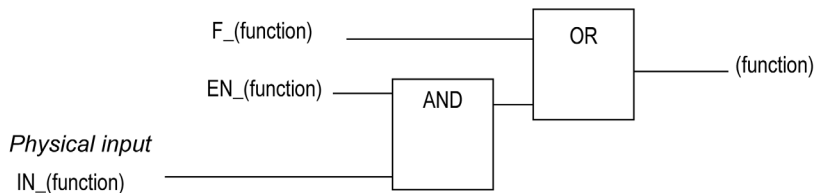
This section presents the functions used by the PWM:

- **Synchronization** function
- **Enable** function

Each function uses the 2 following function block bits:

- **EN\_(function) bit:** Setting this bit to 1 allows the (function) to operate on an external physical input if configured.
- **F\_(function) bit:** Setting this bit to 1 forces the (function).

The following diagram explains how the function is managed:



**NOTE:** (function) stands either for **Enable** (for Enable function) or **Sync** (for Synchronization function).

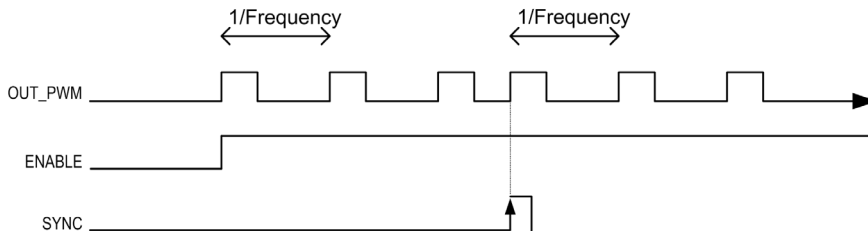
If the physical input is required, enable it in the configuration screen ([see page 29](#)).

### Synchronization Function

The **Synchronization** function is used to interrupt the current PWM cycle and then restart a new cycle.

### Enable Function

The **Enable** function is used to activate the PWM:





---

# Chapter 3

## Pulse Width Modulation (PWM)

---

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	26
Configuration	28
PWM_LMC058: Command a Pulse Width Modulation Signal	30
Programming the PWM Function Block	32

## Description

### Overview

The pulse width modulation function generates a programmable pulse wave signal on a dedicated output with adjustable duty cycle and frequency.

**NOTE:** Enable the functionality either by setting `F_Enable` to 1, or by an external event with the `IN_EN` input and `EN_Enable=1`, otherwise the output (`OUT_PWM`) stays to 0.

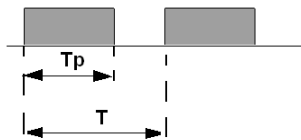
When a Pulse Width Modulation (PWM) is configured on an Expert I/O module, no other functions can be added (*see page 12*).

### Signal Form

The signal form depends on the following input parameters:

- **Frequency** configurable from 0.1 Hz to 20 kHz with a 0.1 Hz step
- **Duty Cycle** of the output signal from 0% to 100%

Duty Cycle =  $T_p/T$



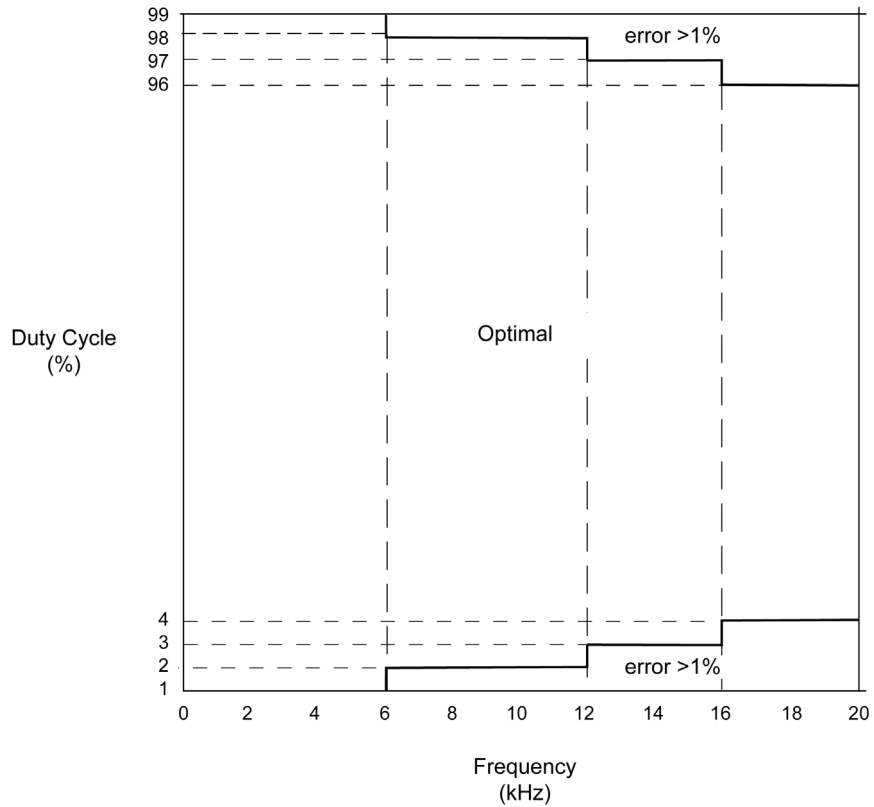
**T<sub>p</sub>** pulse width

**T** pulse period (1/Frequency)

Modifying the duty cycle in the program modulates the width of the signal. Below is an illustration of an output signal with varying duty cycles.



The duty cycle is shown for a precision of 1% for each step of 80 Hz. When duty cycle is below 4% or above 96%, depending on the frequency, the deviation is above 1% as illustrated in the graphic below:



## Configuration

### Overview

Two pulse width modulation functions can be configured on the controller.

### Adding a Pulse Width Modulation Function

To add a pulse width modulation function to your controller, select **PWM** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

For more information on adding a device to your project, refer to:

- Using the Drag-and-drop Method
- Using the Contextual Menu or Plus Button

### Accessing the Parameters

To access the PWM function parameters, proceed as follows:

Step	Action
1	In the <b>Devices tree</b> , double-click <b>MyController</b> → <b>Expert</b> → <b>DM72Fx</b> → <b>PWM</b> .
2	Select the <b>PWM Configuration</b> tab.

## Parameters

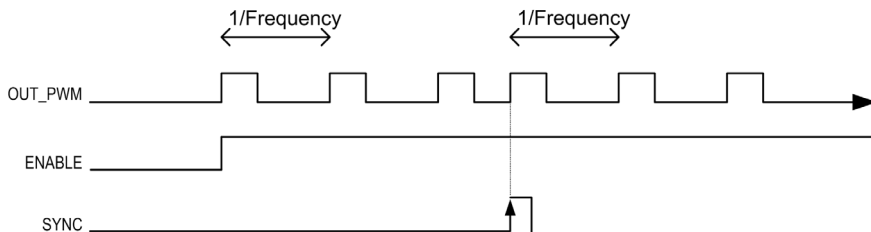
The pulse width modulation function has the following parameters:

Parameter		Value	Default Value	Description
SYNC input	Location	Disabled I1.2	Disabled	Select the controller input used for presetting the pulse generator function.
	Bounce filter	0.002 0.004 0.012 0.04 0.12 0.4 1.2 4	0.002	Set the filtering value to reduce the bounce effect on the SYNC input.
	SYNC Edge	Rising Falling Both	Rising	Select the condition to preset the pulse generator function with the SYNC input.
EN input	Location	Disabled I1.4	Disabled	Select the controller input used for enabling the pulse generator function.
	Bounce filter	0.002 0.004 0.012 0.04 0.12 0.4 1.2 4	0.002	Set the filtering value to reduce the bounce effect on the EN input.

## Synchronizing with an External Event

On a rising edge on the IN\_SYNC physical input (with EN\_Sync = 1), the current cycle is interrupted and the PWM restarts a new cycle.

This illustration provides a pulse diagram for the Pulse Width Modulation function block with use of IN\_SYNC input:



## PWM\_LMC058: Command a Pulse Width Modulation Signal

### Overview

The Pulse Width Modulation function block commands a pulse width modulated signal output at the specified frequency and duty cycle.

### Graphical Representation



### IL and ST Representation

To see the general representation in IL or ST language, refer to the *Differences Between a Function and a Function Block* (see page 50) chapter.

### Input Variables

This table describes the input variables:

Inputs	Type	Comment
EN_Enable	BOOL	TRUE = authorizes the PWM enable via the IN_EN input (if configured).
F_Enable	BOOL	TRUE = forces the Enable function.
EN_SYNC	BOOL	TRUE = authorizes the restart via the IN_Sync input of the internal timer relative to the time base (if configured).
F_SYNC	BOOL	On a rising edge, forces a restart of the internal timer relative to the time base.
Frequency	DWORD	Frequency of the Pulse Width Modulation output signal in tenths of Hz (range: 1 (0.1 Hz)...200,000 (20 kHz)).
Duty	BYTE	Duty cycle of the Pulse Width Modulation output signal in % (range: 0...100).

## Output Variables



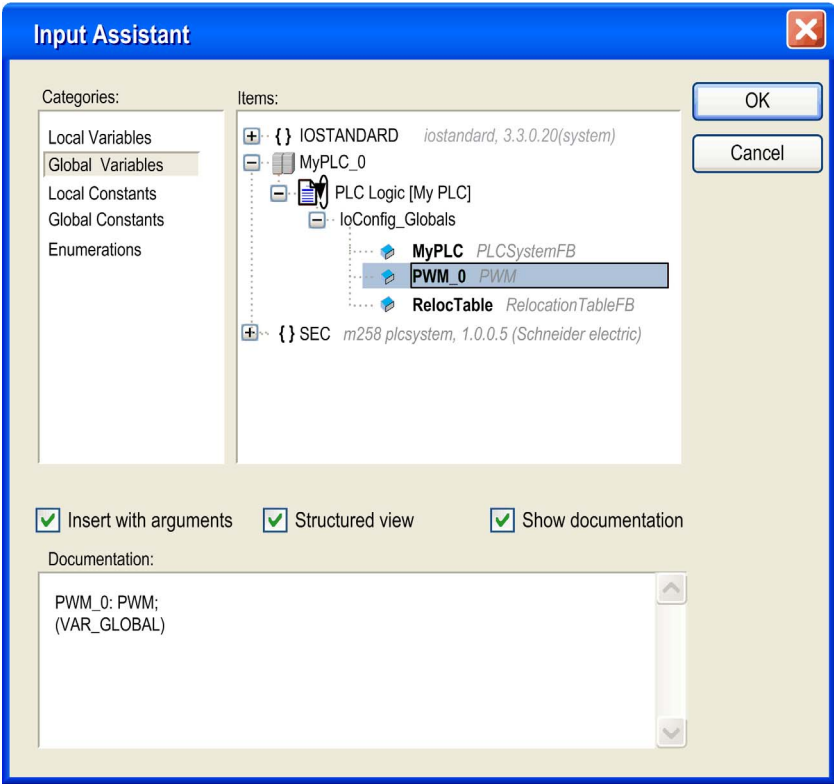
This table describes the output variables:

Outputs	Type	Comment
InFrequency	BOOL	TRUE = the Pulse Width Modulation signal is currently being output at the specified frequency and duty cycle.
Busy	BOOL	Busy is used to indicate that a command change is in progress: the frequency is changed. Set to TRUE when the Enable command is set and the Frequency Generator signal is not output at the specified Frequency. Reset to FALSE when InFrequency or Error is set, or when the Enable command is reset. When a command change execution is immediate, Busy remains FALSE.
Error	BOOL	TRUE = indicates that an error was detected.
ErrID	EXPERT_ERR_TYPE (see page 59)	When Error is set: type of the detected error.

## Programming the PWM Function Block

### Procedure

Follow these steps to program a **PWM** function block:

Step	Action
1	Select the <b>Libraries</b> tab in the <b>Software Catalog</b> and click <b>Libraries</b> . Select <b>Controller</b> → <b>LMC058</b> → <b>LMC058 Expert IO</b> → <b>OutputGenerator</b> → <b>PWM_LMC058</b> in the list, drag-and-drop the item onto the <b>POU</b> window.
2	Select the function block instance by clicking   . The <b>Input Assistant</b> dialog is displayed. Select the global variable which references to the added PWM ( <i>see page 28</i> ) during the configuration and confirm. <div data-bbox="292 557 1126 1336" style="border: 1px solid black; padding: 10px; margin: 10px 0;">  </div>
3	The inputs/outputs are detailed in the function block ( <i>see page 30</i> ). The interaction between the inputs/outputs are detailed in the General Information ( <i>see page 45</i> ).

**NOTE:** If the function block instance is not visible, verify if the PWM is configured.



**Program Illustration**

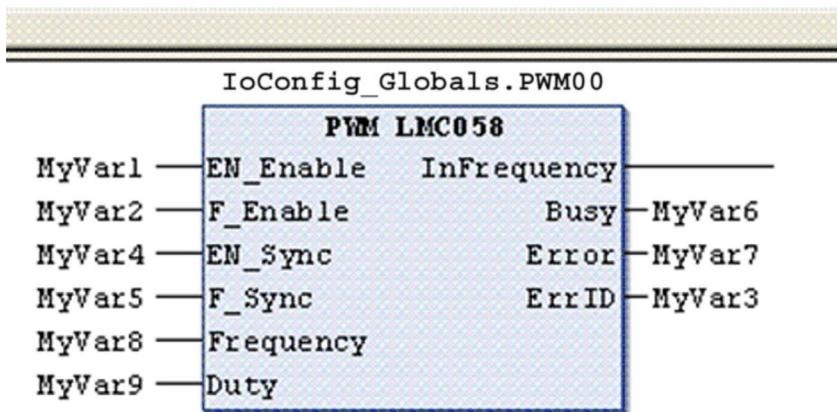
This illustration shows an example of a `PWM_LMC058` function block programmed.

```
PROGRAM POU_1
```

```
VAR
```

```
MyVar1: BOOL;  
MyVar2: BOOL;  
MyVar3: PTOPWM_ERR_TYPE;  
MyVar4: BOOL;  
MyVar5: BOOL;  
MyVar6: BOOL;  
MyVar7: BOOL;  
MyVar8: DWORD;  
MyVar9: BYTE;
```

```
END_VAR
```





---

# Chapter 4

## Frequency Generator (FreqGen)

---

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	36
Configuration	37
Frequency_Generator_LMC058: Commanding a Square Wave Signal	39
Programming	41

## Description

### Overview

The frequency generator function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

**Frequency** is configurable from 0.1 Hz to 100 kHz with a 0.1 Hz step.

When a frequency generator is configured on an Expert I/O module, no other functions can be added (*see page 12*).

## Configuration

### Overview

Two frequency generator functions can be configured on the controller.

### Adding a Frequency Generator Function

To add a frequency generator function, proceed as follows:

Step	Action
1	Select the <b>FreqGen</b> in the <b>Hardware Catalog</b> , drag it to the <b>Devices tree</b> , and drop it on one of the highlighted nodes. For more information on adding a device to your project, refer to: <ul style="list-style-type: none"><li>• Using the Drag-and-drop Method</li><li>• Using the Contextual Menu or Plus Button</li></ul>
2	Double-click <b>FreqGen</b> and open the <b>Frequency Generator Configuration</b> tab.
3	Set the parameters as shown in the table below.
4	Double-click the <b>Pulse Generators</b> node of your controller in the <b>Devices Tree</b> .
5	Double-click the <b>Pulse generation function</b> value and select <b>FreqGen</b> . <b>Result:</b> The frequency generator configuration parameters are displayed.

### Parameters

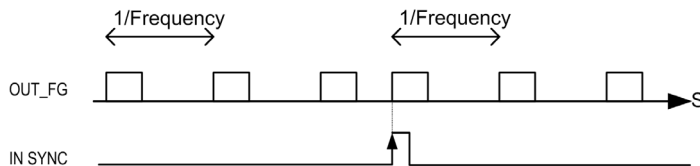
The frequency generator function has the following parameters:

Parameter		Value	Type	Description
SYNC input	Location	Disabled (default) I2	BOOL	Enables the IN_SYNC physical input to be used for synchronization.
	Bounce filter	0.002 (default) 0.004 0.012 0.04 0.12 0.4 1.2 4	ENUM	Defines the value of the IN_SYNC filter value.
	SYNC EDGE	Rising Falling Both	ENUM	Defines the IN_SYNC edge on which synchronization occurs.
EN input	Location	Disabled (default) I4	BOOL	Enables the IN_EN physical input to be used for enabling the functionality.
	Bounce filter	0.002 (default) 0.004 0.012 0.04 0.12 0.4 1.2 4	ENUM	Defines the value of the IN_EN filter value.

### Synchronizing with an External Event

On a rising edge on the IN\_SYNC physical input (with EN\_Sync = 1), the current cycle is interrupted and the FreqGen restarts a new cycle.

This illustration provides a pulse diagram for the frequency generator function block with use of IN\_SYNC input:



## Frequency\_Generator\_LMC058: Commanding a Square Wave Signal

### Overview

The `Frequency_Generator` function block commands a square wave signal output at the specified frequency.

### Graphical Representation (LD/FBD)



### IL and ST Representation

To see the general representation in IL or ST language, refer to the *Differences Between a Function and a Function Block* (see page 50) chapter.

### Input Variables

This table describes the input variables:

Inputs	Type	Comment
EN_Enable	BOOL	TRUE = authorizes the <code>Frequency_Generator</code> enable via the IN_EN input (if configured).
F_Enable	BOOL	TRUE = forces the Enable function.
EN_SYNC	BOOL	TRUE = authorizes the restart via the IN_SYNC input of the internal timer relative to the time base (if configured).
F_SYNC	BOOL	On rising edge, forces a restart of the internal timer relative to the time base.
Frequency	DWORD	Frequency of the <code>Frequency_Generator</code> output signal in tenths of Hz. (Range: min 1 (0.1Hz)...max 1,000,000 (100kHz))

### Output Variables

This table describes the output variables:



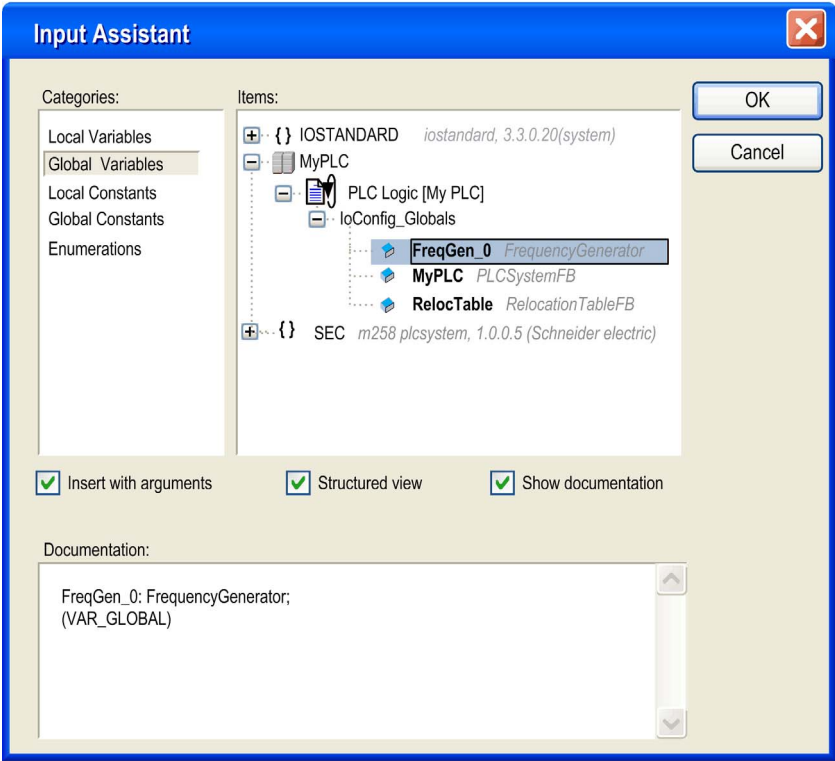
Outputs	Type	Comment
InFrequency	BOOL	TRUE = the Frequency Generator signal is output at the specified Frequency.
Busy	BOOL	<p>Busy is used to indicate that a command change is in progress: the frequency is changed.</p> <p>Set to TRUE when the Enable command is set and the Frequency Generator signal is not output at the specified Frequency.</p> <p>Reset to FALSE when InFrequency or Error is set, or when the Enable command is reset.</p> <p>When a command change execution is immediate, Busy remains FALSE.</p>
Error	BOOL	TRUE = indicates that an error was detected.
ErrID	EXPERT_ERR_TYPE <i>(see page 59)</i>	When Error is set: type of the detected error.



## Programming

### Procedure

Follow these steps to program a Frequency Generator function block:

Step	Action
1	Select the <b>Libraries</b> tab in the <b>Software Catalog</b> and click <b>Libraries</b> . Select <b>Controller</b> → <b>LMC058</b> → <b>LMC058 Expert IO</b> → <b>OutputGenerator</b> → <b>FrequencyGenerator_LMC058</b> in the list; drag-and-drop the item onto the <b>POU</b> window.
2	Select the function block instance by clicking   . The Input Assistant screen is displayed. Select the global variable which references to the added FreqGen ( <i>see page 37</i> ) during the configuration and confirm.
	
	<b>NOTE:</b> If the function block instance is not visible, verify if the frequency generator is configured.
3	The inputs/outputs are detailed in the function block ( <i>see page 39</i> ). The interaction between the inputs/outputs are detailed in the General Information ( <i>see page 45</i> ).

### Program Illustration

The following illustration shows an example of a **Frequency Generator\_LMC058** function block programmed.

```
PROGRAM POU_1
```

```
VAR
```

```
MyVar1: BOOL;
```

```
MyVar2: BOOL;
```

```
MyVar3: EXPERT_ERR_TYPE;
```

```
MyVar4: BOOL;
```

```
MyVar5: BOOL;
```

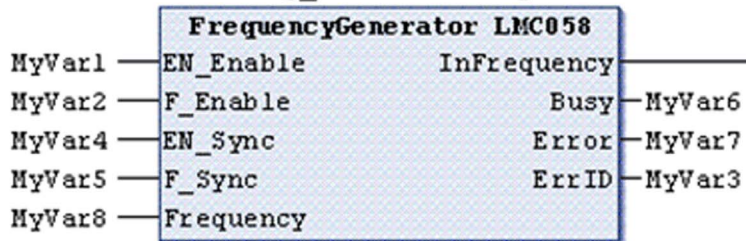
```
MyVar6: BOOL;
```

```
MyVar7: BOOL;
```

```
MyVar8: DWORD;
```

```
END_VAR
```

IoConfig\_Globals.FreqGen00



---

# Appendices

---



## Overview

This appendix extracts parts of the programming guide for technical understanding of the library documentation.

## What Is in This Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	General Information	45
B	Function and Function Block Representation	49
C	Data Unit Types	59



---

# Appendix A

## General Information

---

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Dedicated Features	46
General Information on Function Block Management	47

## Dedicated Features


### Bounce Filter

This table shows the maximum counter frequencies determined by the filtering values used to reduce the bounce effect on the input:

Input	Bounce Filter Value (ms)	Maximum Counter Frequency Expert	Maximum Counter Frequency Regular
A B	0.000	200 kHz	1 kHz
	0.001	200 kHz	1 kHz
	0.002	200 kHz	1 kHz
	0.005	100 kHz	1 kHz
	0.01	50 kHz	1 kHz
	0.05	25 kHz	1 kHz
	0.1	5 kHz	1 kHz
	0.5	1 kHz	1 kHz
	1	500 Hz	500 Hz
	5	100 Hz	100 Hz
A is the counting input of the counter. B is the counting input of the dual phase counter.			

### Dedicated Outputs

Outputs used by the high speed expert functions can only be accessed through the function block. They cannot be read or written directly within the application.

<b> WARNING</b>
<b>UNINTENDED EQUIPMENT OPERATION</b>
<ul style="list-style-type: none"> <li>Do not use the same function block instance in different program tasks.</li> <li>Do not modify or otherwise change the function block reference (AXIS) while the function block is executing.</li> </ul>
<b>Failure to follow these instructions can result in death, serious injury, or equipment damage.</b>

---

## General Information on Function Block Management

### Management of Input Variables

The variables are used with the rising edge of the `Execute` input. To modify any variable, it is necessary to change the input variables and to trigger the function block again.

According to IEC 61131-3, if any variable of a function block input is missing (= open), then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.

### Management of Output Variables

The `Done`, `Error`, `Busy`, and `CommandAborted` outputs are mutually exclusive; only one of them can be TRUE on one function block. When the `Execute` input is TRUE, one of these outputs is TRUE.

At the rising edge of the `Execute` input, the `Busy` output is set. It remains set during the execution of the function block and is reset at the rising edge of one of the other outputs (`Done`, `Error`).

The `Done` output is set when the execution of the function block is successfully completed.

If an error is detected, the function block terminates by setting the `Error` output, and the error code is contained within the `ErrID` output.

The `Done`, `Error`, `ErrID`, and `CommandAborted` outputs are set or reset with the falling edge of `Execute` input:

- reset if the function block execution is finished.
- set for at least one task cycle if the function block execution is not finished.

When an instance of a function block receives a new `Execute` before it is finished (as a series of commands on the same instance), the function block does not return any feedback, like `Done`, for the previous action.

### Error Handling

All blocks have two outputs that can report error detection during the execution of the function block:

- `Error`= The rising edge of this bit informs that an error was detected.
- `ErrID`= The error code of the error detected.

When an `Error` occurs, the other output signals, such as `Done` are reset.





---

# Appendix B

## Function and Function Block Representation

---

### Overview

Each function can be represented in the following languages:

- IL: Instruction List
- ST: Structured Text
- LD: Ladder Diagram
- FBD: Function Block Diagram
- CFC: Continuous Function Chart

This chapter provides functions and function blocks representation examples and explains how to use them for IL and ST languages.

### What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Differences Between a Function and a Function Block	50
How to Use a Function or a Function Block in IL Language	51
How to Use a Function or a Function Block in ST Language	55

## Differences Between a Function and a Function Block

### Function

A function:

- is a POU (Program Organization Unit) that returns one immediate result.
- is directly called with its name (not through an instance).
- has no persistent state from one call to the other.
- can be used as an operand in other expressions.

**Examples:** boolean operators (AND), calculations, conversion (BYTE\_TO\_INT)

### Function Block

A function block:

- is a POU (Program Organization Unit) that returns one or more outputs.
- needs to be called by an instance (function block copy with dedicated name and variables).
- each instance has a persistent state (outputs and internal variables) from one call to the other from a function block or a program.

**Examples:** timers, counters

In the example, `Timer_ON` is an instance of the function block `TON`:

```
1  PROGRAM MyProgram_ST
2  VAR
3      Timer_ON: TON; // Function Block Instance
4      Timer_RunCd: BOOL;
5      Timer_PresetValue: TIME := T#5S;
6      Timer_Output: BOOL;
7      Timer_ElapsedTime: TIME;
8  END_VAR
```

```
1  Timer_ON(
2      IN:=Timer_RunCd,
3      PT:=Timer_PresetValue,
4      Q=>Timer_Output,
5      ET=>Timer_ElapsedTime);
```

## How to Use a Function or a Function Block in IL Language

### General Information

This part explains how to implement a function and a function block in IL language.

Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

### Using a Function in IL Language

This procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to <i>Adding and Calling POU's (see EcoStruxure Machine Expert, Programming Guide)</i> .
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: <ul style="list-style-type: none"> <li>type the name of the function in the operator column (left field), or</li> <li>use the <b>Input Assistant</b> to select the function (select <b>Insert Box</b> in the context menu).</li> </ul>
5	If the function has more than 1 input and when Input Assistant is used, the necessary number of lines is automatically created with ??? in the fields on the right. Replace the ??? with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: <code>IsFirstMastCycle</code>	
with input parameters: <code>SetRTCDrift</code>	

In IL language, the function name is used directly in the operator column:

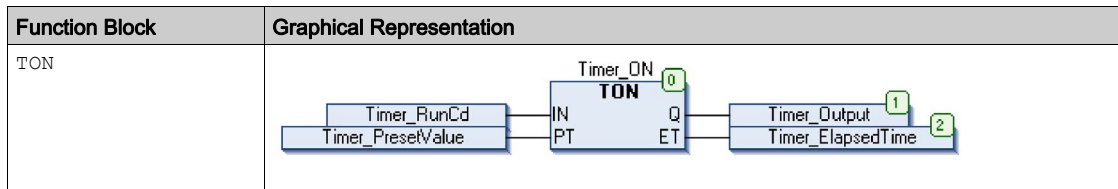
Function	Representation in POU IL Editor															
IL example of a function without input parameter: IsFirstMastCycle	<pre> 1  PROGRAM MyProgram_IL 2  VAR 3      FirstCycle: BOOL; 4  END_VAR                     </pre> <hr/> <table border="1" data-bbox="381 459 979 570"> <tr> <td data-bbox="381 459 444 488">1</td> <td data-bbox="444 459 742 488"><b>IsFirstMast Cycle</b></td> <td data-bbox="742 459 979 488"></td> </tr> <tr> <td data-bbox="381 488 444 518"></td> <td data-bbox="444 488 742 518"><b>ST</b></td> <td data-bbox="742 488 979 518">FirstCycle</td> </tr> <tr> <td data-bbox="381 518 444 547"></td> <td data-bbox="444 518 742 547"></td> <td data-bbox="742 518 979 547"></td> </tr> </table>	1	<b>IsFirstMast Cycle</b>			<b>ST</b>	FirstCycle									
1	<b>IsFirstMast Cycle</b>															
	<b>ST</b>	FirstCycle														
IL example of a function with input parameters: SetRTCDrift	<pre> 1  PROGRAM MyProgram_IL 2  VAR 3      myDrift: SINT (-29..29) := 5; 4      myDay: DAY_OF_WEEK := SUNDAY; 5      myHour: HOUR := 12; 6      myMinute: MINUTE; 7      myDiag: RTCSETDRIFT_ERROR; 8  END_VAR                     </pre> <hr/> <table border="1" data-bbox="381 971 930 1146"> <tr> <td data-bbox="381 971 444 1000">1</td> <td data-bbox="444 971 683 1000"><b>LD</b></td> <td data-bbox="683 971 930 1000">myDrift</td> </tr> <tr> <td data-bbox="381 1000 444 1029"></td> <td data-bbox="444 1000 683 1029"><b>SetRTCDrift</b></td> <td data-bbox="683 1000 930 1029">myDay</td> </tr> <tr> <td data-bbox="381 1029 444 1058"></td> <td data-bbox="444 1029 683 1058"></td> <td data-bbox="683 1029 930 1058">myHour</td> </tr> <tr> <td data-bbox="381 1058 444 1088"></td> <td data-bbox="444 1058 683 1088"></td> <td data-bbox="683 1058 930 1088">myMinute</td> </tr> <tr> <td data-bbox="381 1088 444 1117"></td> <td data-bbox="444 1088 683 1117"><b>ST</b></td> <td data-bbox="683 1088 930 1117">myDiag</td> </tr> </table>	1	<b>LD</b>	myDrift		<b>SetRTCDrift</b>	myDay			myHour			myMinute		<b>ST</b>	myDiag
1	<b>LD</b>	myDrift														
	<b>SetRTCDrift</b>	myDay														
		myHour														
		myMinute														
	<b>ST</b>	myDiag														

## Using a Function Block in IL Language

This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs ( <i>see EcoStruxure Machine Expert, Programming Guide</i> ).
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a <b>CAL</b> instruction: <ul style="list-style-type: none"> <li>● Use the <b>Input Assistant</b> to select the FB (right-click and select <b>Insert Box</b> in the context menu).</li> <li>● Automatically, the <b>CAL</b> instruction and the necessary I/O are created.</li> </ul> Each parameter (I/O) is an instruction: <ul style="list-style-type: none"> <li>● Values to inputs are set by " := ".</li> <li>● Values to outputs are set by " =&gt; ".</li> </ul>
4	In the <b>CAL</b> right-side field, replace ??? with the instance name.
5	Replace other ??? with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the **TON** Function Block graphically presented below:



In IL language, the function block name is used directly in the operator column:

Function Block	Representation in POU IL Editor
TON	<pre>1 PROGRAM MyProgram_IL 2 VAR 3     Timer_ON: TON; // Function Block instance declaration 4     Timer_RunCd: BOOL; 5     Timer_PresetValue: TIME := T#5S; 6     Timer_Output: BOOL; 7     Timer_ElapsedTime: TIME; 8 END_VAR 9</pre> <hr/> <pre>1 CAL          Timer_ON(                 IN:= Timer_RunCd,                 PT:= Timer_PresetValue,                 Q=&gt; Timer_Output,                 ET=&gt; Timer_ElapsedTime)</pre>

## How to Use a Function or a Function Block in ST Language

### General Information

This part explains how to implement a Function and a Function Block in ST language.

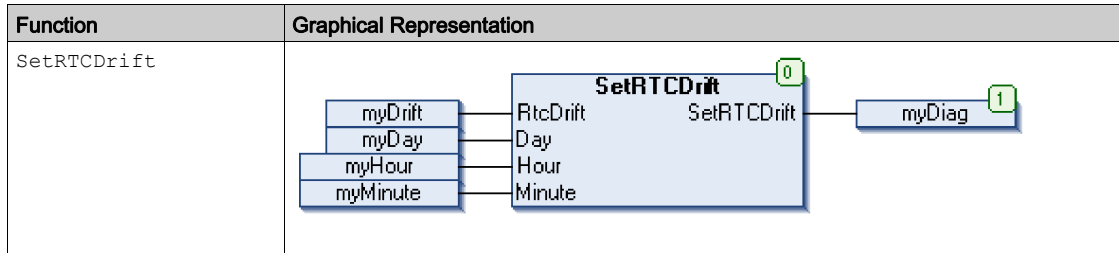
Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

### Using a Function in ST Language

This procedure describes how to insert a function in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POU's (see <i>EcoStruxure Machine Expert, Programming Guide</i> ).
2	Create the variables that the function requires.
3	Use the general syntax in the <b>POU ST Editor</b> for the ST language of a function. The general syntax is: FunctionResult:= FunctionName (VarInput1, VarInput2,.. VarInputx);

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:



The ST language of this function is the following:

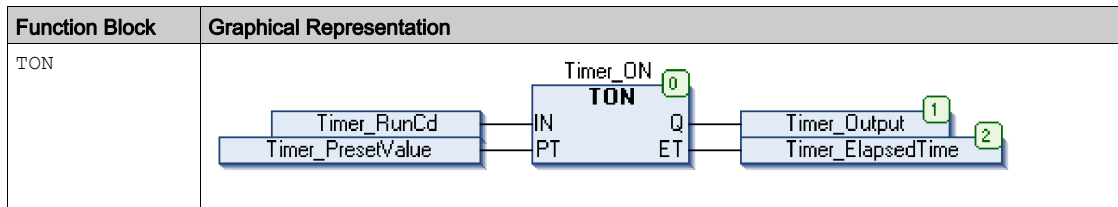
Function	Representation in POU ST Editor
SetRTCDrift	<pre>PROGRAM MyProgram_ST VAR myDrift: SINT(-29..29) := 5; myDay: DAY_OF_WEEK := SUNDAY; myHour: HOUR := 12; myMinute: MINUTE; myRTCAdjust: RTCDRIFT_ERROR; END_VAR myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);</pre>

### Using a Function Block in ST Language

This procedure describes how to insert a function block in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. <b>NOTE:</b> The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POU's (see <i>EcoStruxure Machine Expert, Programming Guide</i> ).
2	Create the input and output variables and the instance required for the function block: <ul style="list-style-type: none"> <li>• Input variables are the input parameters required by the function block</li> <li>• Output variables receive the value returned by the function block</li> </ul>
3	Use the general syntax in the <b>POU ST Editor</b> for the ST language of a Function Block. The general syntax is: <pre>FunctionBlock_InstanceName (Input1:=VarInput1, Input2:=VarInput2, ... Ouput1=&gt;VarOutput1, Ouput2=&gt;VarOutput2, ...);</pre>

To illustrate the procedure, consider this example with the TON function block graphically presented below:





This table shows examples of a function block call in ST language:

Function Block	Representation in POU ST Editor
TON	<pre>1  PROGRAM MyProgram_ST 2  VAR 3      Timer_ON: TON; // Function Block Instance 4      Timer_RunCd: BOOL; 5      Timer_PresetValue: TIME := T#5S; 6      Timer_Output: BOOL; 7      Timer_ElapsedTime: TIME; 8  END_VAR  1  Timer_ON( 2      IN:=Timer_RunCd, 3      PT:=Timer_PresetValue, 4      Q=&gt;Timer_Output, 5      ET=&gt;Timer_ElapsedTime);</pre>



---

# Appendix C

## Data Unit Types

---

### EXPERT\_ERR\_TYPE: Type for Error Variable on an Expert Function Block

#### Enumerated Type Description

The table below lists the values for the enumerated data type ENUM:

Enumerator	Value	Description
EXPERT_NO_ERROR	00 hex	No error detected.
EXPERT_UNKNOWN	01 hex	The reference EXPERT is incorrect or not configured.
EXPERT_UNKNOWN_PARAMETER	02 hex	The parameter reference is incorrect.
EXPERT_INVALID_PARAMETER	03 hex	The value of the parameter is incorrect.
EXPERT_COM_ERROR	04 hex	A communication problem was detected with the EXPERT module.
EXPERT_CAPTURE_NOT_CONFIGURED	05 hex	Capture is not configured.





## A

### **application**

A program including configuration data, symbols, and documentation.

## B

### **byte**

A type that is encoded in an 8-bit format, ranging from 00 hex to FF hex.

## C

### **CFC**

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

### **control network**

A network containing logic controllers, SCADA systems, PCs, HMI, switches, ...

Two kinds of topologies are supported:

- flat: all modules and devices in this network belong to same subnet.
- 2 levels: the network is split into an operation network and an inter-controller network.

These two networks can be physically independent, but are generally linked by a routing device.

### **controller**

Automates industrial processes (also known as programmable logic controller or programmable controller).

### **CPDM**

(*controller power distribution module*) The connection of the controller to the external 24 Vdc power supplies and the beginning of the power distribution for the local configuration.

## E

### **encoder**

A device for length or angular measurement (linear or rotary encoders).

## F

### FB

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

### function

A programming unit that has 1 input and returns 1 immediate result. However, unlike FBs, it is directly called with its name (as opposed to through an instance), has no persistent state from one call to the next and can be used as an operand in other programming expressions.

Examples: boolean (AND) operators, calculations, conversions (BYTE\_TO\_INT)

### function block diagram

One of the 5 languages for logic or control supported by the standard IEC 61131-3 for control systems. Function block diagram is a graphically oriented programming language. It works with a list of networks where each network contains a graphical structure of boxes and connection lines representing either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

## H

### HSC

(*high-speed counter*) A function that counts pulses on the controller or on expansion module inputs.

## I

### I/O

(*input/output*)

### IEC 61131-3

Part 3 of a 3-part IEC standard for industrial automation equipment. IEC 61131-3 is concerned with controller programming languages and defines 2 graphical and 2 textual programming language standards. The graphical programming languages are ladder diagram and function block diagram. The textual programming languages include structured text and instruction list.

### IL

(*instruction list*) A program written in the language that is composed of a series of text-based instructions executed sequentially by the controller. Each instruction includes a line number, an instruction code, and an operand (refer to IEC 61131-3).

### INT

(*integer*) A whole number encoded in 16 bits.

**L****LD**

*(ladder diagram)* A graphical representation of the instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller (refer to IEC 61131-3).

**N****network**

A system of interconnected devices that share a common data path and protocol for communications.

**P****POU**

*(program organization unit)* A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

**PWM**

*(pulse width modulation)* A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave).

**R****reflex output**

Among the outputs of HSC are the reflex outputs associated to a threshold value that is compared to the counter value depending on the configuration of the HSC. The reflex outputs switch to either on or off depending on the configured relationship with the threshold.

**run**

A command that causes the controller to scan the application program, read the physical inputs, and write to the physical outputs according to solution of the logic of the program.

**S****ST**

*(structured text)* A language that includes complex statements and nested instructions (such as iteration loops, conditional executions, or functions). ST is compliant with IEC 61131-3.

**STOP**

A command that causes the controller to stop running an application program.

## T

### **task**

A group of sections and subroutines, executed cyclically or periodically for the MAST task or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in relation to the task.

A controller can have several tasks.

## V

### **variable**

A memory unit that is addressed and modified by a program.





## D

data unit types  
    EXPERT\_ERR\_TYPE, *59*  
dedicated features, *46*

## E

error handling  
    ErrID, *47*  
    Error, *47*  
EXPERT\_ERR\_TYPE  
    data unit types, *59*

## F

frequency generator  
    configuration, *37*  
    description, *36*  
    programming, *41*  
FrequencyGenerator\_LMC058  
    commanding a square wave signal , *39*  
function blocks  
    FrequencyGenerator\_LMC058, *39*  
    PWM\_LMC058, *30*  
functions  
    differences between a function and a  
    function block, *50*  
    enable, *24*  
    how to use a function or a function block  
    in IL language, *51*  
    how to use a function or a function block  
    in ST language, *55*  
    synchronization, *24*

## M

management of status variables  
    Busy, *47*  
    CommandAborted, *47*  
    Done, *47*  
    ErrID, *47*  
    Error, *47*  
    Execute, *47*

## P

Programming  
    PWM, *32*  
pulse width modulation  
    configuration, *28*  
    description, *26*  
PWM  
    programming, *32*  
    PWM\_M241, *30*  
PWM\_LMC258  
    commanding a pulse width modulation  
    signal, *30*

