# Modicon M241 Logic Controller

## High Speed Counting HSC Library Guide

04/2017


EIO0000001444.03

www.schneider-electric.com

Schneider Electric

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.



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.

# Table of Contents

# 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 will acquaint you with the High Speed Counter (HSC) functions and variables offered within the M241 logic controller.

This documentation describes the functions and variables of the M241 HSC library.

In order to use this manual, you must:

- Have a thorough understanding of the M241, 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)

SoMachine software can also be used to program these controllers using CFC (Continuous Function Chart) language.

### Validity Note

This document has been updated for the release of SoMachine V4.3.

### Related Documents

| Title of Documentation | Reference Number |
|---|---|
| SoMachine Programming Guide | *EIO0000000067 (ENG)*, *EIO0000000069 (FRE)*, *EIO0000000068 (GER)*, *EIO0000000071 (SPA)*, *EIO0000000070 (ITA)*, *EIO0000000072 (CHS)* |
| Modicon M241 Logic Controller Programming Guide | *EIO0000001432 (ENG)*, *EIO0000001433 (FRE)*, *EIO0000001434 (GER)*, *EIO0000001435 (SPA)*, *EIO0000001436 (ITA)*, *EIO0000001437 (CHS)* |

You can download these technical publications and other technical information from our website at http://www.schneider-electric.com/en/download

## Product Related Information

### ⚠ WARNING

**LOSS OF CONTROL**

- 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.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

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

[1] 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.**

# Part I
# Introduction

## Overview

This part provides an overview description, available modes, functionality and performances of the different functions.

## What Is in This Part?

This part contains the following chapters:

# Chapter 1
# Expert Function Introduction

## Overview

This chapter provides an overview description, functionality, and performances of:

- High Speed Counter (HSC)
- Pulse Train Output (PTO)
- Pulse Width Modulation (PWM)
- Frequency Generator (FreqGen)

## What Is in This Chapter?

This chapter contains the following topics:

# Expert Functions Overview

## Introduction

The M241 logic controller supports the following expert functions:

| Functions | | Description |
|---|---|---|
| Counters | HSC Simple | The HSC function executes fast counts of pulses from sensors, switches, and so on.<br>For more information about the HSC functions, refer to the High Speed Counter types *(see page 21)*. |
| | HSC Main Single Phase | |
| | HSC Main Dual Phase | |
| | Frequency Meter | |
| | Period Meter | |
| Pulse Generators | PTO *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)* | The PTO function generates a pulse train output to control a linear single-axis stepper or servo drive in open loop mode. |
| | PWM *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)* | The PWM function generates a square wave signal with a variable duty cycle. |
| | FreqGen *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)* | The FreqGen (frequency generator) function generates a square wave signal with a fixed duty cycle (50%). |

As of the release of SoMachine 4.3, any regular I/O, not already in use, can be configured for use by any of the expert function types as it is for fast I/O.

The maximum number of expert functions that can be configured depends on:

1. The logic controller reference.
2. The expert function types and number of optional functions *(see page 119)* configured. Refer to Embedded Expert I/O Assignment *(see page 17)*.
3. The number of I/Os that are available.

Maximum number of expert functions by logic controller reference:

| Expert Function Type | | 24 I/O References (TM241•24•) | 40 I/O References (TM241•40•) |
|---|---|---|---|
| Total number of HSC functions | | 14 | 16 |
| HSC | Simple | 14 | 16 |
| | Main Single Phase | 4 | |
| | Main Dual Phase | | |
| | Frequency Meter(1) | | |
| | Period Meter | | |
| PTO | | | |
| PWM | | | |
| FreqGen | | | |
| (1) When the maximum number is configured, only 12 additional HSC Simple functions can be added. | | | |

The maximum number of expert functions possible may be further limited by the number of I/Os used by each expert function.

Example configurations:

- 4 PTO(1) + 14 HSC Simple on 24 I/O controller references
- 4 FreqGen(1) + 16 HSC Simple on 40 I/O controller references
- 4 HSC Main Single Phase + 10 HSC Simple on 24 I/O controller references
- 4 HSC Main Dual Phase + 8 HSC Simple on 40 I/O controller references
- 2 PTO(1) + 2 HSC Main Single Phase + 14 HSC Simple on 40 I/O controller references

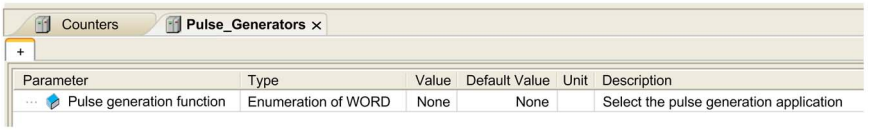
(1) with no optional I/O configured

The performance of the expert function is limited by the I/Os used:

- HSC with fast inputs: 100 kHz/200 kHz
- HSC with regular inputs: 1 kHz

## Configuring an Expert Function

To configure an expert function, proceed as follows:

| Step | Description |
|---|---|
| 1 | Double-click the **Counters** or **Pulse_Generators** node in the **Devices Tree**.<br>**Result:** The **Counters** or **Pulse_Generators** configuration window appears:<br>  |

| Step | Description |
|---|---|
| 2 | Double-click **None** in the **Value** column and choose the expert function type to assign.<br>**Result:** The default configuration of the expert function appears when you click anywhere in the configuration window. |
| 3 | Configure the expert function parameters, as described in the following chapters. |
| 4 | To configure an additional expert function, click the **+** tab.<br>**NOTE:** If the maximum number of expert functions is already configured, a message appears at the bottom of the configuration window informing you that you can now add only HSC Simple functions. |

# Embedded Expert I/O Assignment

## I/O Assignment

The following regular or fast I/Os can be configured for use by expert functions:

| | 24 I/O References | | 40 I/O References | |
|---|---|---|---|---|
| | TM241•24T, TM241•24U | TM241•24R | TM241•40T, TM241•40U | TM241•40R |
| Inputs | 8 fast inputs (I0...I7)<br>6 regular inputs (I8...I13) | | 8 fast inputs (I0...I7)<br>8 regular inputs (I8...I15) | |
| Outputs | 4 fast outputs (Q0...Q3)<br>4 regular outputs (Q4...Q7) | 4 fast outputs (Q0...Q3) | 4 fast outputs (Q0...Q3)<br>4 regular outputs (Q4...Q7) | 4 fast outputs (Q0...Q3) |

When an I/O has been assigned to an expert function, it is no longer available for selection with other expert functions.

**NOTE:** All I/Os are by default disabled in the configuration window.

The following table shows the I/Os that can be configured for expert functions:

| Expert Function | Name | Input (Fast or Regular) | Output (Fast or Regular) |
|---|---|---|---|
| HSC Simple | Input | M | |
| HSC Main | Input A | M | |
| | Input B/EN | C | |
| | SYNC | C | |
| | CAP | C | |
| | Reflex 0 | | C |
| | Reflex 1 | | C |
| Frequency Meter/Period Meter | Input A | M | |
| | EN | C | |
| PWM/FreqGen | Output A | | M |
| | SYNC | C | |
| | EN | C | |
| **M** Mandatory<br>**C** Optionally configurable | | | |

| Expert Function | Name | Input (Fast or Regular) | Output (Fast or Regular) |
|---|---|---|---|
| PTO | Output A/CW/Pulse | | M |
| | Output B/CCW/Dir | | C |
| | REF (Origin) | C | |
| | INDEX (Proximity) | C | |
| | PROBE | C | |
| **M** Mandatory<br>**C** Optionally configurable | | | |

## Using Regular I/O with Expert Functions

Expert function I/O within regular I/O:

- Inputs can be read through standard memory variables even if configured as expert functions.
- All I/Os that are not used by expert functions can be used as regular I/Os.
- An I/O can only be used by one expert function; once configured, the I/O is no longer available for other expert functions.
- If no more fast I/Os are available, a regular I/O can be configured instead. In this case, however, the maximum frequency of the expert function is limited to 1 kHz.
- You cannot configure an input in an expert function and use it as a Run/Stop, Event, or Latch input at a same time.
- An output cannot be configured in an expert function if it has already been configured as an alarm.
- Short-circuit management still applies on all outputs. Status of outputs are available. For more information, refer to Output Management *(see Modicon M241 Logic Controller, Hardware Guide)*.
- When inputs are used in expert functions (PTO, HSC,…), the integrator filter is replaced by an anti-bounce filter *(see page 142)*. The filter value is configured in the configuration window.

For more details, refer to Embedded Functions Configuration *(see Modicon M241 Logic Controller, Programming Guide)*.

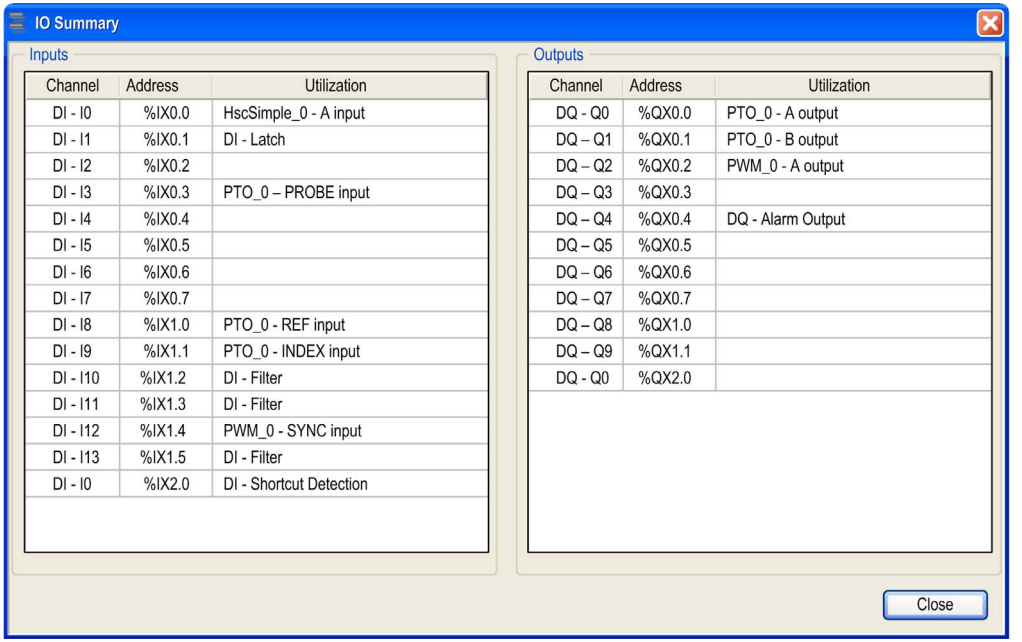
## I/O Summary

The **IO Summary** window displays the I/Os used by the expert functions.

To display the **IO Summary** window:

| Step | Action |
|---|---|
| 1 | In the **Devices tree** tab, right-click the **MyController** node and choose **IO Summary**. |

Example of IO Summary window:



# Chapter 2
# High Speed Counter Types

## Overview

This chapter provides an overview of the different types of HSC.

## What Is in This Chapter?

This chapter contains the following topics:

# Choosing Your Counter

## Overview

Start the HSC configuration by choosing a counter type according to the type of sensor you are using and the application need.

In the **Counters** editor, select a **Counting function** from the list that offers the following 5 types of counters (for more information, refer to the Counter Function *(see Modicon M241 Logic Controller, Programming Guide)*):

- **HSC Simple**
- **HSC Main Single Phase**
- **HSC Main Dual Phase**
- **Frequency Meter**
- **Period Meter**

The **Frequency Meter** type and the **Period Meter** type are based on an **HSC Main** type.

For each counter function block, the **Instance name** is automatically given by SoMachine. The **Instance name** is editable. However, whether the **Instance name** is software-defined or user-defined, use the same instance name as an input to the function blocks dealing with the counter, as defined in the **Counters** editor.

## Type and Mode Matrix

This table presents the different types and modes available:

| Type<br>Mode | HSC Simple | HSC Main Single Phase | HSC Main Dual Phase | Frequency Meter | Period Meter |
|---|---|---|---|---|---|
| One-shot | X | X | – | – | – |
| Modulo-loop | X | X | X | – | – |
| Event Counting | – | X | – | – | – |
| Free-large | – | – | X | – | – |
| Edge to Edge | – | – | – | – | X |
| Edge to Opposite | – | – | – | – | X |

## HSC Simple

This table presents an overview of the specifications available in **HSC Simple** type according to the mode requested:

| Feature | Function | |
|---|---|---|
| | **One-shot Mode** | **Modulo-loop Mode** |
| Counting mode | Count down | Count up |
| Enable with an HSC physical input | No | No |
| Synchronization / preset with an HSC physical input | No | No |
| Comparison function | No | No |
| Capture function | No | No |
| Configuration tuning | Preset | Modulo |

## HSC Main Single Phase

This table presents an overview of the specifications available in **HSC Main Single Phase** type according to the mode requested:

| Feature | Function | | |
|---|---|---|---|
| | **One-shot Mode** | **Modulo-loop Mode** | **Event Counting Mode** |
| Counting mode | Count down | Count up | Pulse counting during given time base |
| Enable with an HSC physical input | Yes | Yes | No |
| Synchronization / preset with an HSC physical input | Yes | Yes | Yes |
| Comparison function | Yes, 4 thresholds, 2 outputs, and 4 events | Yes, 4 thresholds, 2 outputs, and 4 events | No |
| Capture function | Yes, 1 capture register | Yes, 1 capture register | No |
| Configuration tuning | Stop event | – | Time base |

## HSC Main Dual Phase

This table presents an overview of the specifications available in **HSC Main Dual Phase** type according to the mode requested:

| Feature | Function | |
|---|---|---|
| | **Modulo-Loop Mode** | **Free-Large Mode** |
| Counting mode | Count up / down<br>Pulse / direction<br>Quadrature | Count up / down<br>Pulse / direction<br>Quadrature |
| Enable with an HSC physical input | No | No |
| Synchronization / preset with an HSC physical input | Yes | Yes |
| Comparison function | Yes, 4 thresholds, 2 outputs, and 4 events | Yes, 4 thresholds, 2 outputs, and 4 events |
| Capture function | Yes, 1 capture register | Yes, 1 capture register |
| Configuration tuning | – | Limits management |

## Frequency Meter

This table presents an overview of the specifications available in **Frequency Meter** type:

| Feature | Function |
|---|---|
| Counting mode | Pulse frequency in Hz with updated value available every time base value (10, 100, or 1000 ms). |
| Enable with an HSC physical input | Yes |
| Synchronization / preset with an HSC physical input | No |
| Comparison function | No |
| Capture function | No |
| Configuration tuning | Time base |

## Period Meter

This table presents an overview of the specifications available in **Period Meter** type according to the mode requested:

| Feature | Function |
|---|---|
| Counting modes | Edge to edge: Measure the duration of an event.<br>Edge to opposite: Measure the time between two events.<br>Duration counting with configurable resolution (0.1 µs, 1 µs, 100 µs, or 1000 µs). |
| Enable with an HSC physical input | Yes |
| Synchronization / preset with an HSC physical input | No |
| Comparison function | No |
| Capture function | No |
| Configuration tuning | Resolution<br>Time out |

# Simple Type Overview

## Overview

The **Simple** type is a single input counter.

Any operation on the counter (enable, sync) and any action triggered (when count value is reached) is executed in the context of a task.

With the **Simple** type, you cannot trigger an event or a reflex output.

## Simple Type Modes

The **Simple** type supports 2 configurable counting modes, only on single-phase pulses:

**One-shot** *(see page 35)*: In this mode, the counter current value register decrements (from a user-defined value) for each pulse applied to A input, until the counter reaches 0.

**Modulo-loop** *(see page 55)*: In this mode, the counter repeatedly counts from 0 to a user-defined modulo value then returns to 0 and restarts counting.

## Performance

The maximum frequency admissible on a fast input is 100 kHz if the bounce filter value is 0.005 ms (default value for configuration). If the bounce filter value is 0.002 ms, the maximum frequency is 200 kHz.

The maximum frequency admissible on a regular input is 1 kHz if the bounce filter value is 0.5 ms. If the bounce filter value is 1 ms, the maximum frequency is 500 Hz.

For more information about the bounce filter, refer to the Dedicated Features *(see page 142)*.

# Main Type Overview

## Overview

The **Main** type is a counter that uses up to 4 fast or regular inputs and 2 reflex outputs. The M241 Logic Controller can have up to 4 **Main** type High Speed Counters.

## Main Type Modes

The **Main** type supports the following counting modes on single phase (1 input) or dual-phase (2 inputs) pulses:

**One-shot** *(see page 41)*: In this mode, the counter current value register decrements (from a user-defined value) for each pulse applied to A input until the counter reaches a 0.

**Modulo-loop** *(see page 61)*: In this mode, the counter repeatedly counts from 0 to a user-defined modulo value then returns to 0 and restarts counting. In reverse, the counter counts down from the modulo value to 0 then presets to the modulo value and restarts counting.

**Free-large** *(see page 77)*: In this mode, the counter behaves like a high range up and down counter.

**Event Counting** *(see page 89)*: In this mode, the counter accumulates a number of events that are received during a user-configured time base.

## Optional Features

Optional features can be configured depending on the selected mode:

- hardware inputs to operate the counter (enable, preset) or capture the current counting value
- up to 4 thresholds
- up to 4 events (1 for each threshold) can be associated with external tasks
- up to 2 reflex outputs

## Performance

The maximum frequency admissible on an **Expert I/O** interface is 100 kHz if the bounce filter value is 0.005 ms (default value for configuration). If the bounce filter value is 0.002 ms, the maximum frequency is 200 kHz.

If the expert function is configured with a regular I/O, the minimum period admissible is 0.4 ms.

# Frequency Meter Type Overview

## Overview

The **Frequency Meter** type is a counter that uses up to 2 fast or regular inputs. The M241 Logic Controller can have up to 4 **Frequency Meter** type High Speed Counters.

## Frequency Meter Type Mode

The **Frequency meter** *(see page 101)* counter measures the frequency of events. Frequency is the number of events per second (Hz).

## Performance

The maximum frequency admissible on a fast input is 100 kHz if the bounce filter value is 0.005 ms (default value for configuration). If the bounce filter value is 0.002 ms, the maximum frequency is 200 kHz.

The maximum frequency admissible on a regular input is 1 kHz if the bounce filter value is 0.5 ms. If the bounce filter value is 1 ms, the maximum frequency is 500 Hz.

For more information about the bounce filter, refer to the Dedicated Features *(see page 142)*.

# Period Meter Type Overview

## Overview

The **Period Meter** type is a counter that uses up to 2 fast or regular inputs. The M241 Logic Controller can have up to 4 **Period Meter** type High Speed Counters.

## Period Meter Type Mode

Use the **Period meter** counting mode to:

- Determine the duration of an event
- Measure the time between 2 events
- Set and measure the execution time for a process

## Performance

The minimum period admissible on a fast input is 0.005 ms.

If the expert function is configured with a regular I/O, the minimum period admissible is 0.4 ms.

# Part II
# One-shot Mode

## Overview

This part describes the use of a HSC in **One-shot** Mode.

## What Is in This Part?

This part contains the following chapters:

| Chapter | Chapter Name | Page |
|---|---|---|
| 3 | **One-shot** Mode Principle | 33 |
| 4 | **One-shot** with a **Simple** Type | 35 |
| 5 | **One-shot** With a **Main** Type | 41 |

# Chapter 3
# One-shot Mode Principle

## One-shot Mode Principle Description

### Overview

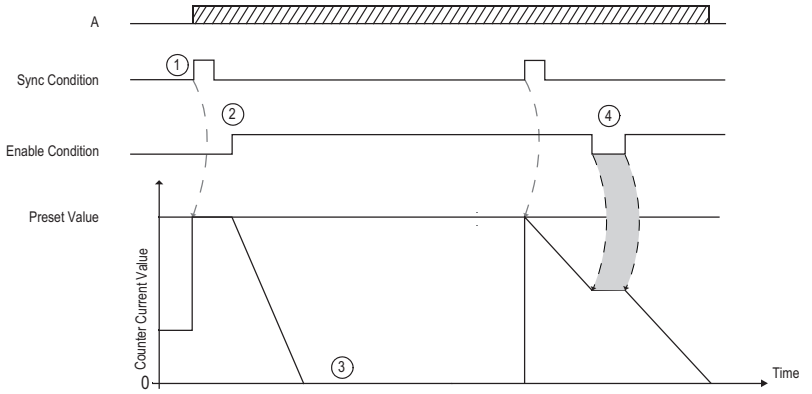
The counter is activated by a synchronization edge, and the preset value is loaded.

When counting is enabled, each pulse applied to the input decrements the current value. The counter stops when its current value reaches 0.

The counter value remains at 0 even if new pulses are applied to the input.

A new synchronization is needed to activate the counter again.

### Principle Diagram




This table explains the stages from the preceding graphic:

| Stage | Action |
|---|---|
| 1 | On the rising edge of the Sync condition, the preset value is loaded in the counter (regardless of the current value) and the counter is activated. |
| 2 | When Enable condition = 1, the current counter value decrements on each pulse on input A until it reaches 0. |
| 3 | The counter waits until the next rising edge of the Sync condition.<br>**Note:** At this point, pulses on input A have no effect on the counter. |

| Stage | Action |
|---|---|
| 4 | When Enable condition = 0, the counter ignores the pulses from input A and retains its current value until the Enable condition again = 1. The counter resumes counting pulses from input A on the rising edge of the Enable input from the held value. |

**NOTE:** Enable and Sync conditions depends on configuration. These are described in the Enable *(see page 137)* and Preset *(see page 134)* function.

# Chapter 4
# One-shot with a Simple Type

## Overview

This chapter describes how to implement a High Speed Counter in **One-shot** mode using a **Simple** type.

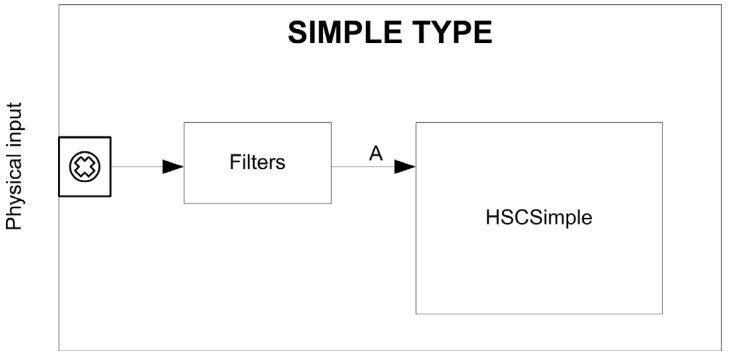
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Simple** type in **One-shot** mode:




A is the counting input of the `High Speed Counter`. **Simple** type counting for **One-shot** mode only counts down.

## Configuration of the Simple Type in One-Shot Mode

### Procedure

Follow this procedure to configure a **Simple** type in **One-shot** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Simple**.<br>**Result**: Depending on the selected counter function, the configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **One-shot**. |
| 5 | In **Counting inputs → A input → Location** select the fast or regular input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Enter the value of the **Range → Preset** parameter to set the counting initial value. |

# Programming the Simple Type

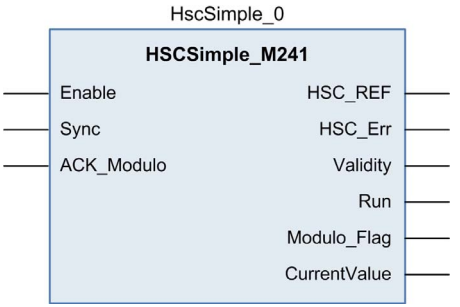
## Overview

A **Simple** type is always managed by an HSCSimple_M241 *(see page 168)* function block.

**NOTE:** At build, a detected error code is given if the HSCSimple_M241 function block is used to manage a different HSC type.

## Adding a HSCSimple Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCSimple_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Simple** type instance name (defined in configuration) or select the function block instance by clicking:<br>  <br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |




## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **One-shot** mode.

This table describes the input variables:

| Input | Type | Comment |
|---|---|---|
| `Enable` | `BOOL` | `TRUE` = authorizes changes to the current counter value. |
| `Sync` | `BOOL` | On rising edge, presets and starts the counter |
| `ACK_Modulo` | `BOOL` | Not used |

This table describes the output variables:

| Output | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used as input of **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that the output values on the function block are valid. |
| Run | BOOL | Set to 1 when the counter is running.<br>Switches to 0 when CurrentValue reaches 0. A synchronization is needed to restart the counter. |
| Modulo_Flag | BOOL | Not relevant |
| CurrentValue | DWORD | Current count value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the `EXPERTGetParam` *(see page 160)* or `EXPERTSetParam` *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the `EXPERT_PARAMETER_TYPE` *(see page 150)* that can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| `EXPERT_PRESET` | to get or set the Preset value of an HSC |

# Chapter 5
# One-shot With a Main Type

## Overview

This chapter describes how to implement a High Speed Counter in **One-shot** mode using a **Main** type.

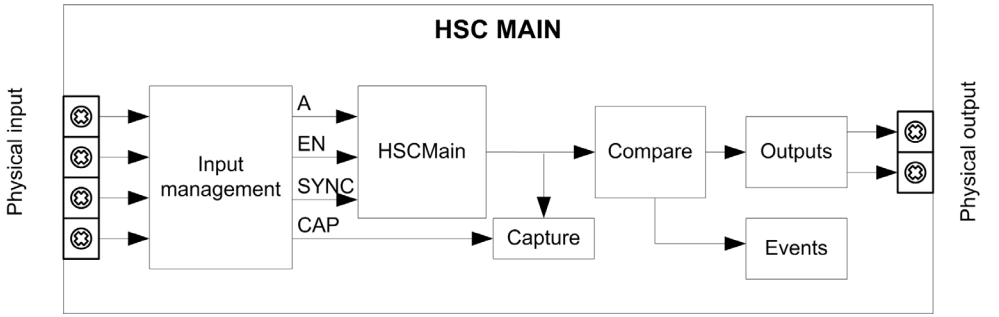
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main** type in **One-shot** mode:




A is the counting input of the counter.

EN is the enable input of the counter.

CAP is the capture input of the counter.

SYNC is the synchronization input of the counter.

## Optional Function

In addition to the **One-shot** mode, the **Main** type can provide the following functions:

- Comparison function *(see page 121)*
- Capture function *(see page 129)*
- Preset function *(see page 134)*
- Enable function *(see page 137)*

## Configuration of the Main Type Single Phase in One-Shot Mode

### Procedure

Follow this procedure to configure a **Main** type single phase in **One-shot** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Main Single Phase**.<br>**Result**: The configuration parameters appear in the **Counters** tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **One-shot**. |
| 5 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Enter the value of the **Range → Preset** parameter to set the initial counting value of the Preset function *(see page 134)*. |
| 8 | Optionally, you can enable these functions:<br>• Enable function *(see page 137)*<br>• Capture function *(see page 129)*<br>• Comparison function *(see page 121)* |
| 9 | Optionally, set the value of the **Events → Stop Event** parameter to **Yes** to enable the External Event function *(see page 127)*. |

# Programming the Main Type

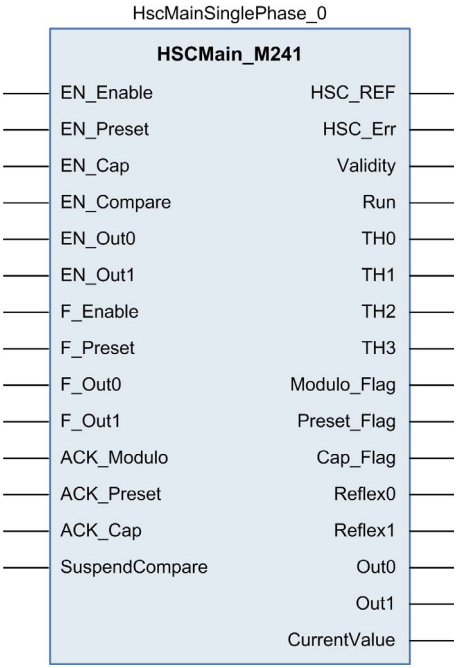
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>??? ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **One-shot** mode.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | When **EN** input is configured: if TRUE, authorizes the counter enable via the Enable input *(see page 137)*. |
| EN_Preset | BOOL | When **SYNC** input is configured: if TRUE, authorizes the counter Preset via the Sync input *(see page 134)*. |
| EN_Cap | BOOL | When **CAP** input is configured: if TRUE, enables the Capture input. |
| EN_Compare | BOOL | TRUE = enables the comparator operation *(see page 121)* (using Thresholds 0, 1, 2, 3):<br>● basic comparison (TH0, TH1, TH2, TH3 output bits)<br>● reflex (Reflex0, Reflex1 output bits)<br>● events (to trigger external tasks on threshold crossing) |
| EN_Out0 | BOOL | TRUE = enables physical output Out_R0 to echo the Reflex0 value (if configured). |
| EN_Out1 | BOOL | TRUE = enables physical output Out_R1 to echo the Reflex1 value (if configured). |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | On rising edge, presets and starts the counter. |
| F_Out0 | BOOL | TRUE = forces Out_R0 to 1 (if Reflex0 is configured in HSC Embedded Function. Takes priority over EN_Out0. |
| F_Out1 | BOOL | TRUE = forces Out_R1 to 1 (if Reflex1 is configured in HSC Embedded Function. Takes priority over EN_Out1. |
| ACK_Preset | BOOL | On rising edge, resets Preset_Flag. |
| ACK_Cap | BOOL | On rising edge, resets Cap_Flag. |
| SuspendCompare | BOOL | TRUE = compare results are suspended:<br>● TH0, TH1, TH2, TH3 , Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value.<br>● Hardware Outputs 0, 1 maintain their last value.<br>● Events are masked.<br>**NOTE:** EN_Compare, EN_ReflexO, EN_Reflex1, F_Out0, F_Out1 remain operational while SuspendCompare is set. |

This table describes the output variables:

| Output | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used as input of **Administrative** function blocks. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid. |
| Run | BOOL | TRUE = counter is running.<br>The Run bit switches to 0 when CurrentValue reaches 0. |
| TH0 | BOOL | Set to 1 when CurrentValue > Threshold 0 *(see page 121)*. |
| TH1 | BOOL | Set to 1 when CurrentValue > Threshold 1 *(see page 121)*. |
| TH2 | BOOL | Set to 1 when CurrentValue > Threshold 2 *(see page 121)*. |
| TH3 | BOOL | Set to 1 when CurrentValue > Threshold 3 *(see page 121)*. |
| Preset_Flag | BOOL | Set to 1 by the preset of the counter *(see page 134)*. |
| Cap_Flag | BOOL | Set to 1 when a new capture value is stored in the Capture register.<br>This flag must be reset before a new capture can occur. |
| Reflex0 | BOOL | State of Reflex0 *(see page 122)*.<br>Only active when EN_Compare is set. |
| Reflex1 | BOOL | State of Reflex1 *(see page 122)*.<br>Only active when EN_Compare is set. |
| Out0 | BOOL | State of physical output Out_R0 (if Reflex0 configured). |
| Out1 | BOOL | State of physical output Out_R1 (if Reflex1 configured). |
| CurrentValue | DINT | Current value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* which can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| EXPERT_PRESET | to get or set the Preset value of an HSC |
| EXPERT_THRESHOLD0 | to get or set the Threshold 0 value of an HSC |
| EXPERT_THRESHOLD1 | to get or set the Threshold 1 value of an HSC |
| EXPERT_THRESHOLD2 | to get or set the Threshold 2 value of an HSC |
| EXPERT_THRESHOLD3 | to get or set the Threshold 3 value of an HSC |
| EXPERT_REFLEX0 | to get or set output 0 reflex mode of an EXPERT function |
| EXPERT_REFLEX1 | to get or set output 0 reflex mode of an EXPERT function |

# Part III
# Modulo-loop Mode

## Overview

This part describes the use of a HSC in **Modulo-loop** mode.

## What Is in This Part?

This part contains the following chapters:

# Chapter 6
# Modulo-loop Principle

## Modulo-loop Mode Principle Description

### Overview

The **Modulo-loop** mode can be used for repeated actions on a series of moving objects, such as packaging and labeling applications.

### Principle

On a rising edge of the Sync condition *(see page 134)*, the counter is activated and the current value is reset to 0.

When counting is enabled *(see page 137)*:

**Incrementing direction:** the counter increments until it reaches the modulo value. At the next pulse, the counter is reset to 0, a modulo flag is set to 1, and the counting continues.

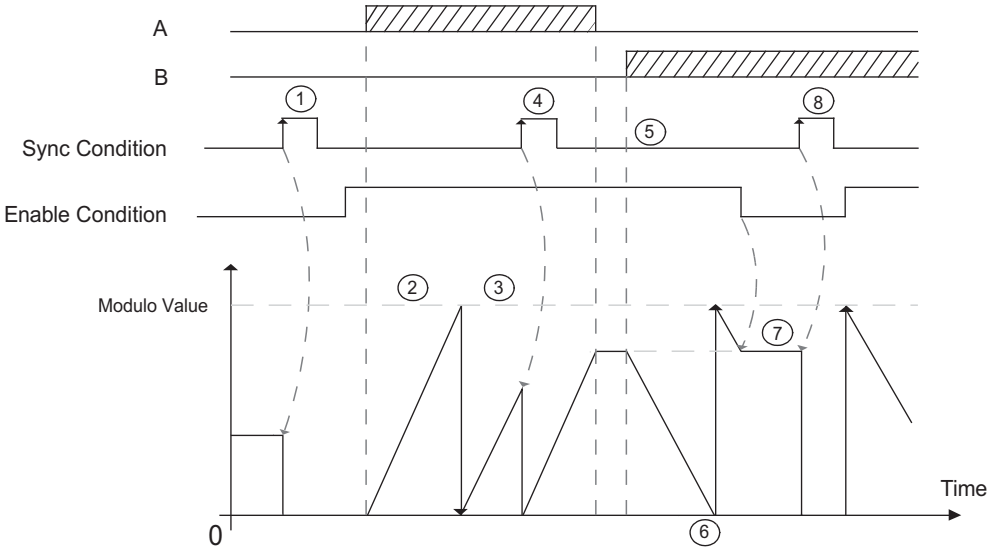
**Decrementing direction:** the counter decrements until it reaches 0. At the next pulse, the counter is set to the modulo value, a modulo flag is set to 1, and the counting continues.

### Input Modes

This table shows the 8 types of input modes available:

| Input Mode | Comment |
|---|---|
| A = Up, B = Down | default mode<br>The counter increments on A and decrements on B. |
| A = Impulse, B = Direction | If there is a rising edge on A and B is true, then the counter decrements.<br>If there is a rising edge on A and B is false, then the counter increments. |
| Normal Quadrature X1 | A physical encoder always provides 2 signals 90° shift that first allows the counter to count pulses and detect direction:<br>● X1: 1 count by Encoder cycle<br>● X2: 2 counts by Encoder cycle<br>● X4: 4 counts by Encoder cycle |
| Normal Quadrature X2 | |
| Normal Quadrature X4 | |
| Reverse Quadrature X1 | |
| Reverse Quadrature X2 | |
| Reverse Quadrature X4 | |

## Up Down Principle Diagram

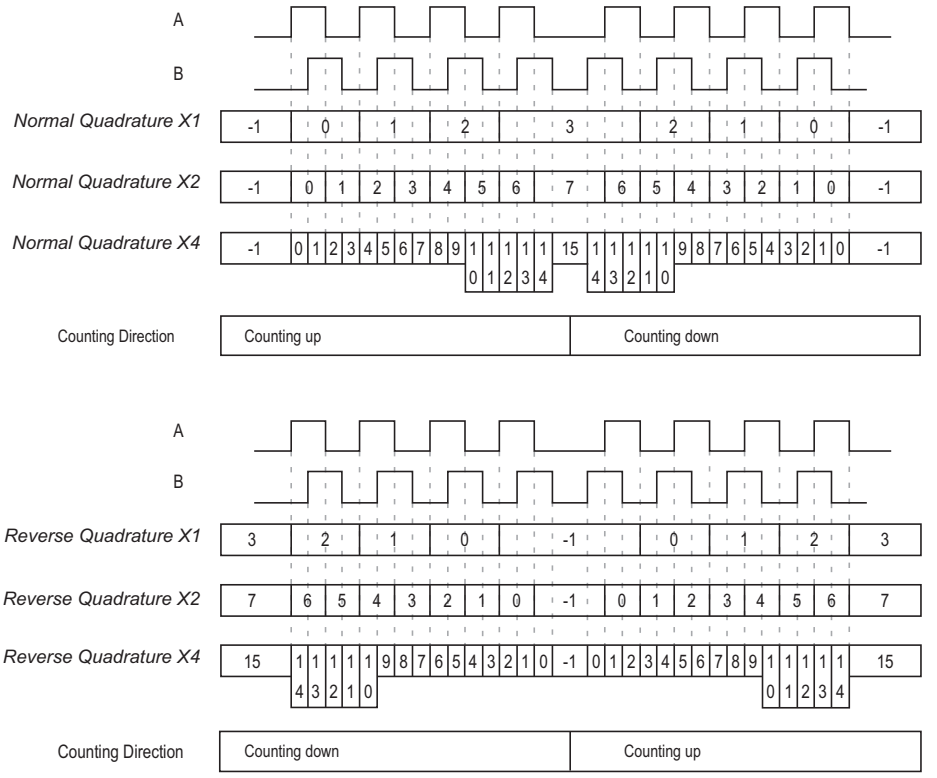



| Stage | Action |
|---|---|
| 1 | On the rising edge of Sync condition, the current value is reset to 0 and the counter is activated. |
| 2 | When Enable condition = 1, each pulses on A increments the counter value. |
| 3 | When the counter reaches the (modulo-1) value, the counter loops to 0 at the next pulse and the counting continues. `Modulo_Flag` is set to 1. |
| 4 | On the rising edge of Sync condition, the current counter value is reset to 0. |
| 5 | When Enable condition = 1, each pulse on B decrements the counter. |
| 6 | When the counter reaches 0, the counter loops to (modulo-1) at the next pulse and the counting continues. |
| 7 | When Enable condition = 0, the pulses on the inputs are ignored. |
| 8 | On the rising edge of Sync condition, the current counter value is reset to 0. |

**NOTE:** Enable and Sync conditions depends on configuration. These are described in the Enable *(see page 137)* and Preset *(see page 134)* function.

## Quadrature Principle Diagram

The encoder signal is counted according to the input mode selected, as shown below:



# Chapter 7
# Modulo-loop with a Simple Type

## Overview

This chapter describes how to implement a High Speed Counter in **Modulo-loop** mode using a **Simple** type.

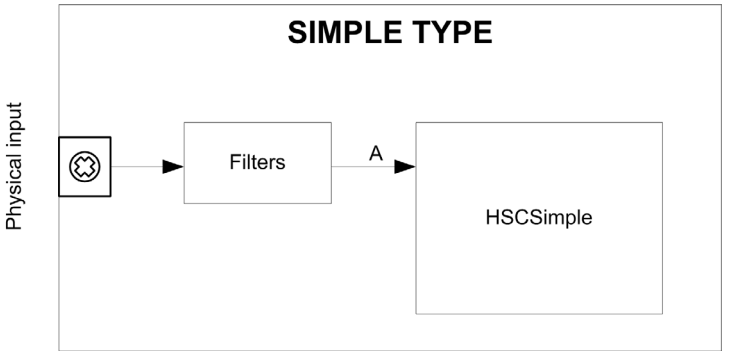
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Simple** type in **Modulo-loop** mode:




A **Simple** type counting for **Modulo-loop** mode only counts up.

## Configuration of the Simple Type in Modulo-Loop Mode

### Procedure

Follow this procedure to configure a **Simple** type in **Modulo-loop** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Simple**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE:** **Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **Modulo-loop**. |
| 5 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Enter the value of the **Range → Modulo** parameter to set the counting modulo value. |

# Programming the Simple Type

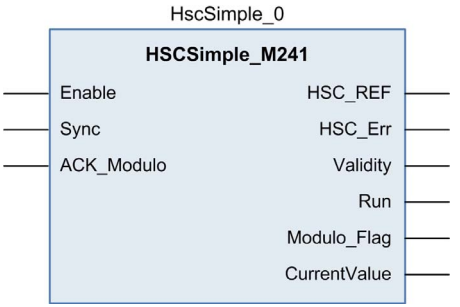
## Overview

A **Simple** type is always managed by an HSCSimple_M241 *(see page 168)* function block.

**NOTE:** At build, a detected error code is given if the HSCSimple_M241 function block is used to manage a different HSC type.

## Adding a HSCSimple Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCSimple_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Simple** type instance name (defined in configuration) or select the function block instance by clicking:<br>  <br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |




## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **Modulo-loop** mode.

This table describes the input variables:

| Input | Type | Comment |
|---|---|---|
| `Enable` | `BOOL` | `TRUE` = authorizes changes to the current counter value. |
| `Sync` | `BOOL` | On rising edge, resets and starts the counter. |
| `ACK_Modulo` | `BOOL` | On rising edge, resets `Modulo_Flag`. |

This table describes the output variables:

| Output | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used as input of the **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that the output values on the function block are valid. |
| Run | BOOL | Not relevant |
| Modulo_Flag | BOOL | Set to 1 when the counter roll overs the modulo. |
| CurrentValue | DWORD | Current value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* that can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| EXPERT_MODULO | to get or set the modulo value of an HSC |

# Chapter 8
# Modulo-loop With a Main Type

## Overview

This chapter describes how to implement a High Speed Counter in **Modulo-loop** mode using a **Main** type.

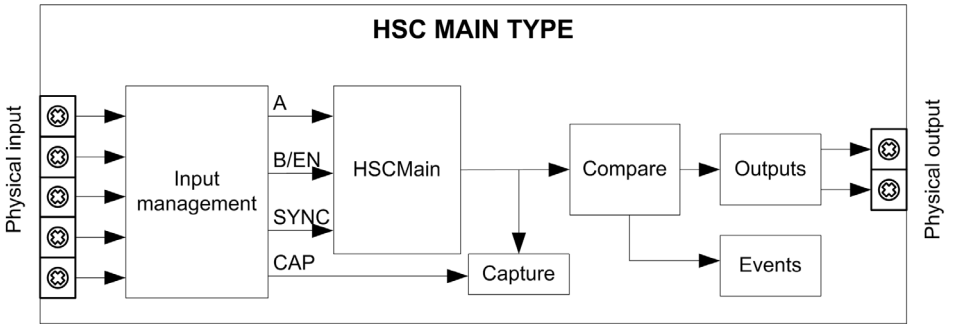
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main** type in **Modulo-loop** mode:




A and B are the counting inputs of the counter.

EN not configurable when B input is used.

CAP is the capture input of the counter.

SYNC is the synchronization input of the counter.

## Optional Function

In addition to the **Modulo-loop** mode, the **Main** type can provide the following functions:

- Comparison function *(see page 121)*
- Capture function *(see page 129)*
- Preset function *(see page 134)*
- Enable function *(see page 137)*

## Configuration of the Main Type Single Phase in Modulo-Loop Mode

### Procedure

Follow this procedure to configure a **Main** type single phase in **Modulo-loop** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Main Single Phase**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **Modulo-loop**. |
| 5 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Enter the value of the **Range → Modulo** parameter to set the counting modulo value. |
| 8 | Optionally, you can enable these functions:<br>● Enable function *(see page 137)*<br>● Capture function *(see page 129)*<br>● Preset function *(see page 134)*<br>● Comparison function *(see page 121)* |

## Configuration of the Main Type Dual Phase in Modulo-Loop Mode

### Procedure

Follow this procedure to configure a **Main** type dual phase in **Modulo-loop** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Main Dual Phase**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **Modulo-loop**. |
| 5 | Set the value of the **General → Input mode** parameter to select the modulo loop input mode *(see page 51)*. |
| 6 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 7 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 8 | In **Counting Inputs → B input → Location** select the regular or fast input to use as the B input. |
| 9 | Set the value of the **Counting inputs → B input → Bounce filter** parameter. |
| 10 | Enter the value of the **Range → Modulo** parameter to set the counting modulo value. |
| 11 | Optionally, you can enable these functions:<br>● Preset function *(see page 134)*<br>● Capture function *(see page 129)*<br>● Comparison function *(see page 121)* |

# Programming the Main Type

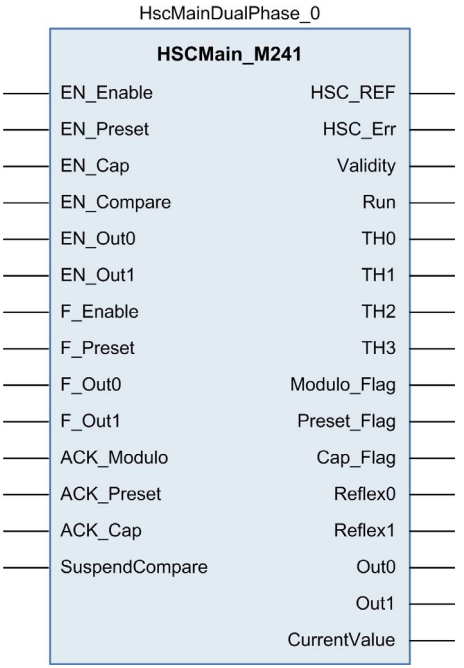
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>??? ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **Modulo-loop** mode.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | When **EN** input is configured: if TRUE, authorizes the counter enable via the Enable input *(see page 137)*. |
| EN_Preset | BOOL | When **SYNC** input is configured: if TRUE, authorizes the counter Preset via the Sync input *(see page 134)*. |
| EN_Cap | BOOL | When **CAP** input is configured: if TRUE, enables the Capture input. |
| EN_Compare | BOOL | TRUE = enables the comparison function *(see page 121)* using Threshold 0, 1, 2, 3:<br>• basic comparison (TH0, TH1, TH2, TH3 output bits)<br>• reflex (Reflex0, Reflex1 output bits)<br>• events (to trigger external tasks on threshold crossing) |
| EN_Out0 | BOOL | TRUE = enables physical output Out_R0 to echo the Reflex0 value (if configured). |
| EN_Out1 | BOOL | TRUE = enables physical output Out_R1 to echo the Reflex1 value (if configured). |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | On rising edge, resets, and starts the counter. |
| F_Out0 | BOOL | TRUE = forces Out_R0 to 1 (if Reflex0 is configured). |
| F_Out1 | BOOL | TRUE = forces Out_R1 to 1 (if Reflex1 is configured). |
| ACK_Modulo | BOOL | On rising edge, resets Modulo_Flag. |
| ACK_Preset | BOOL | On rising edge, resets Preset_Flag. |
| ACK_Cap | BOOL | On rising edge, resets Cap_Flag. |
| SuspendCompare | BOOL | TRUE = compare results are suspended:<br>• TH0, TH1, TH2, TH3 , Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value.<br>• Hardware Outputs 0, 1 maintain their last value.<br>• Events are masked.<br>**NOTE:** EN_Compare, EN_ReflexO, EN_Reflex1, F_Out0, F_Out1 remain operational while SuspendCompare is set. |

This table describes the output variables:

| Output | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used as input of **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid. |
| Run | BOOL | TRUE = counter is running.<br>The Run bit switches to 0 when CurrentValue reaches 0.<br>A synchronization is needed to restart the counter. |
| TH0 | BOOL | Set to 1 when CurrentValue > Threshold 0 *(see page 121)*. |
| TH1 | BOOL | Set to 1 when CurrentValue > Threshold 1 *(see page 121)*. |
| TH2 | BOOL | Set to 1 when CurrentValue > Threshold 2 *(see page 121)*. |
| TH3 | BOOL | Set to 1 when CurrentValue > Threshold 3 *(see page 121)*. |
| Modulo_Flag | BOOL | Set to 1 when the counter roll overs the modulo or 0. |
| Preset_Flag | BOOL | Set to 1 by the preset of the counter *(see page 134)*. |
| Cap_Flag | BOOL | Set to 1 when a new capture value is stored in the Capture register *(see page 130)*.<br>This flag must be reset before a new capture can occur. |
| Reflex0 | BOOL | State of Reflex0 *(see page 123)*.<br>Only active when EN_Compare is set. |
| Reflex1 | BOOL | State of Reflex1 *(see page 123)*.<br>Only active when EN_Compare is set. |
| Out0 | BOOL | State of physical output Out_R0 (if Reflex0 is configured). |
| Out1 | BOOL | State of physical output Out_R1 (if Reflex1 is configured). |
| CurrentValue | DINT | Current value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 160)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* that can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| EXPERT_MODULO | to get or set the Modulo value of an HSC |
| EXPERT_THRESHOLD0 | to get or set the Threshold 0 value of an HSC |
| EXPERT_THRESHOLD1 | to get or set the Threshold 1 value of an HSC |
| EXPERT_THRESHOLD2 | to get or set the Threshold 2 value of an HSC |
| EXPERT_THRESHOLD3 | to get or set the Threshold 3 value of an HSC |
| EXPERT_REFLEX0 | to get or set output 0 reflex mode of an EXPERT function |
| EXPERT_REFLEX1 | to get or set output 1 reflex mode of an EXPERT function |

# Part IV
# Free-large Mode

## Overview

This part describes the use of an HSC in **Free-large** mode.

## What Is in This Part?

This part contains the following chapters:

# Chapter 9
# Free-large Mode Principle

## Overview

This chapter describes the principle of the **Free-large** mode.

## What Is in This Chapter?

This chapter contains the following topics:

# Free-large Mode Principle Description

## Overview

The **Free-large** mode can be used for axis monitoring or labeling in cases where the incoming position of each part has to be known.

## Principle

In the **Free-large** mode, the module behaves like a standard up and down counter.

When counting is enabled *(see page 137)*, the counter counts as follows in:
**Incrementing direction:** the counter increments.
**Decrementing direction:** the counter decrements.

The counter is activated by a preset edge *(see page 136)* which loads the preset value.

The current counter is stored in the capture register by using the Capture *(see page 129)* function.

If the counter reaches the counting limits, the counter will react according to the Limits Management *(see page 75)* configuration.

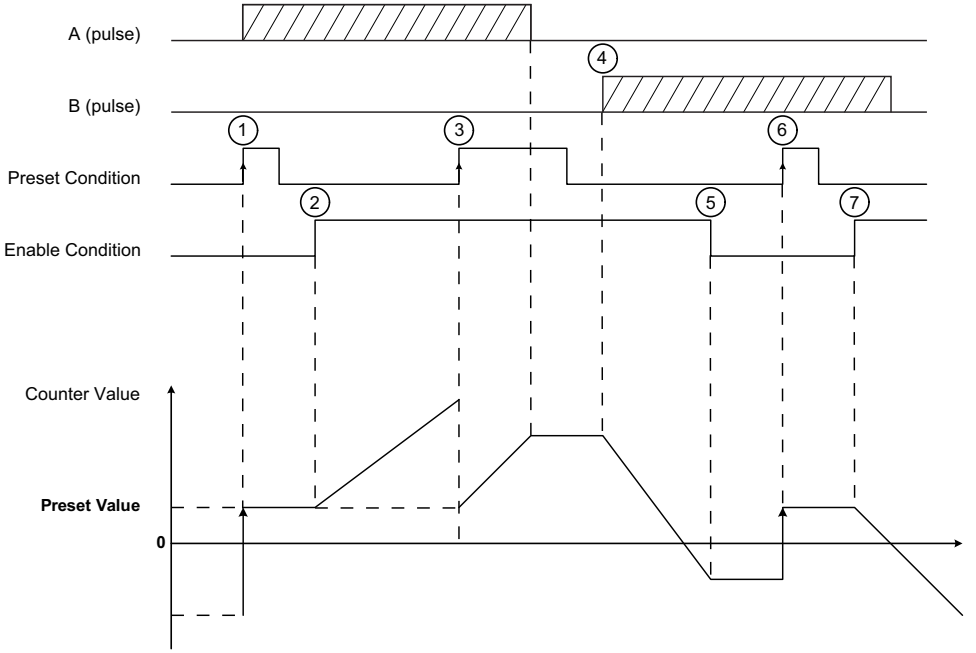
## Input Modes

This table shows the 8 types of input modes available:

| Input Mode | Comment |
|---|---|
| A = Up, B = Down | default mode<br>The counter increments on A and decrements on B. |
| A = Impulse, B = Direction | If there is a rising edge on A and B is true, then the counter decrements.<br>If there is a rising edge on A and B is false, then the counter increments. |
| Normal Quadrature X1 | A physical encoder always provides 2 signals 90° shift that first allows the counter to count pulses and detect direction:<br>● X1: 1 count by Encoder cycle<br>● X2: 2 counts by Encoder cycle<br>● X4: 4 counts by Encoder cycle |
| Normal Quadrature X2 | |
| Normal Quadrature X4 | |
| Reverse Quadrature X1 | |
| Reverse Quadrature X2 | |
| Reverse Quadrature X4 | |

## Up Down Principle Diagram

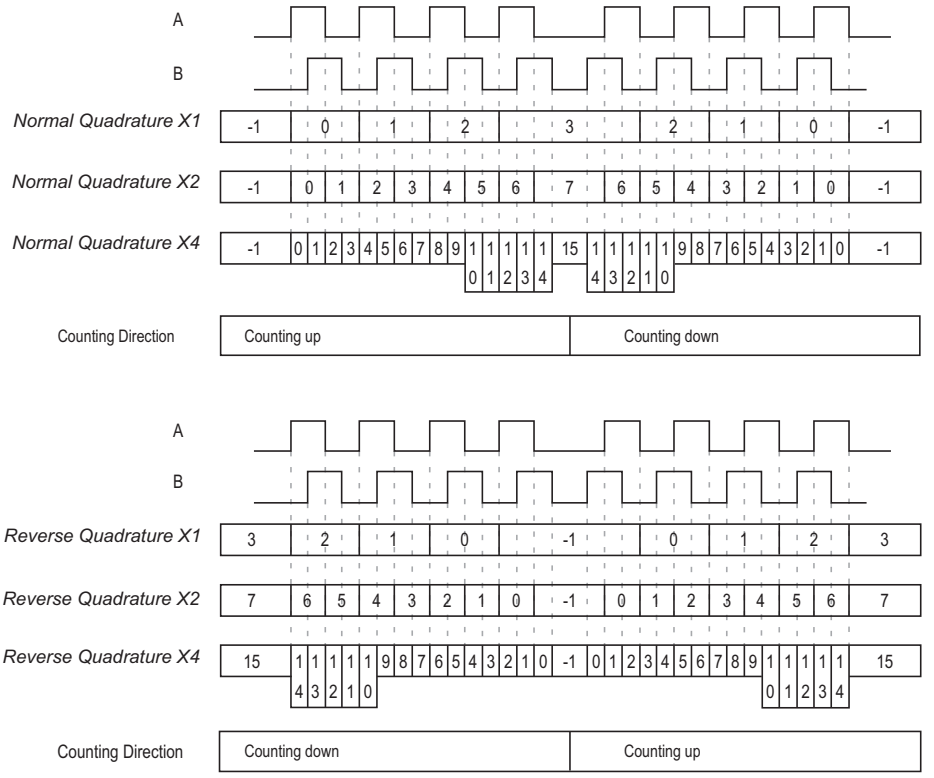
The figures shows the **A = Up, B = Down** mode:




| Stage | Action |
|---|---|
| 1 | On the rising edge of Preset condition, the current value is set to the preset value and the counter is activated. |
| 2 | When Enable condition = 1, each pulse on A increment the counter value. |
| 3 | On the rising edge of Preset condition, the current value is set to the preset value. |
| 4 | When Enable condition = 1, each pulse on B decrements the counter value. |
| 5 | When Enable condition = 0, the pulses on A or B are ignored. |
| 6 | On the rising edge of Preset condition, the current value is set to the preset value. |
| 7 | When Enable condition = 1, the pulses on B decrements the counter value. |

## Quadrature Principle Diagram

The encoder signal is counted according to the input mode selected, as shown below:



# Limits Management

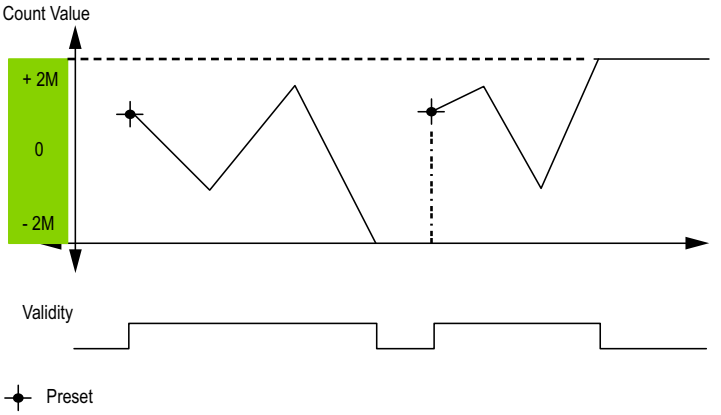
## Overview

When the counter limit is reached, the counter can have 2 behaviors depending on configuration:

- Lock on limits
- Rollover

## Lock on Limits

In the case of overflow or underflow counter: the current counter value is maintained to the limit value, the validity bit goes to 0, and the `Error` bit indicates that this detected error until the counter is preset again.




2M value is given as:

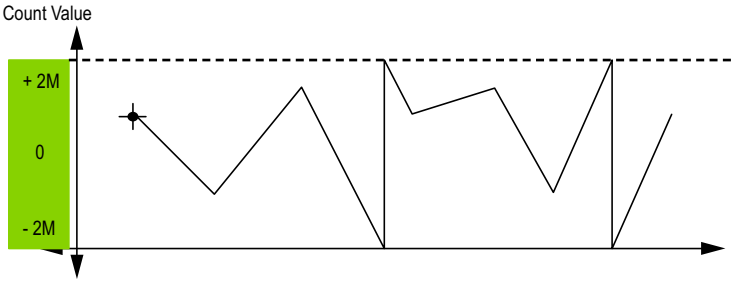
- +2M = 2 (exp 31) -1
- -2M = -2 (exp 31)

## Rollover

In the case of overflow or underflow of the counter, the current counter value goes automatically to the opposite limit value.

`Modulo_Flag` output is set to

1.



# Chapter 10
# Free-large With a Main Type

## Overview

This chapter describes how to implement a High Speed Counter in **Free-large** mode using a **Main** type.

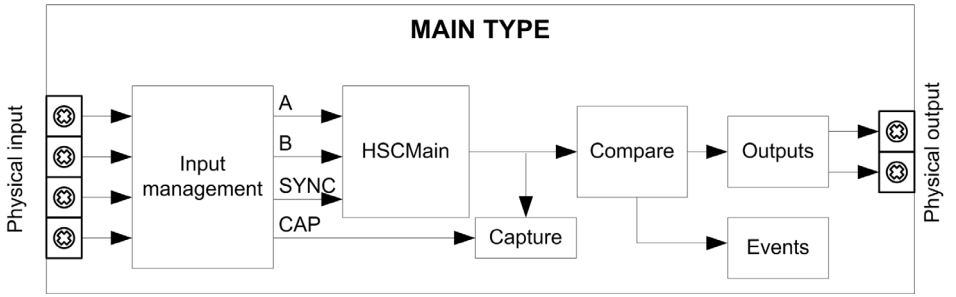
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main** type in **Free-large** mode:




A and B are the counting inputs of the counter.

EN is the enable input of the counter.

CAP is the capture input of the counter.

SYNC is the synchronization input of the counter.

## Optional Function

In addition to the **Free-large** mode, the **Main** type can provide the following functions:

- Comparison function *(see page 121)*
- Capture function *(see page 129)*
- Preset function *(see page 134)*
- Enable function *(see page 137)*

## Configuration of the Main Type Dual Phase in Free-Large Mode

### Procedure

Follow this procedure to configure a **Main** type dual phase in **Free-large** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Main Dual Phase**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **Free-large**. |
| 5 | Set the value of the **General → Input mode** parameter to select the modulo loop input mode *(see page 51)*. |
| 6 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 7 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 8 | Set the value of the **Counting inputs → B input → Bounce filter** parameter. |
| 9 | Enter the value of the **Range → Preset** parameter to set the counting initial value. |
| 10 | Enter the value of the **Range → Limits** for limits management. |
| 11 | Optionally, you can enable these functions:<br>• Preset function *(see page 134)*<br>• Capture function *(see page 129)*<br>• Comparison function *(see page 121)* |

# Programming the Main Type

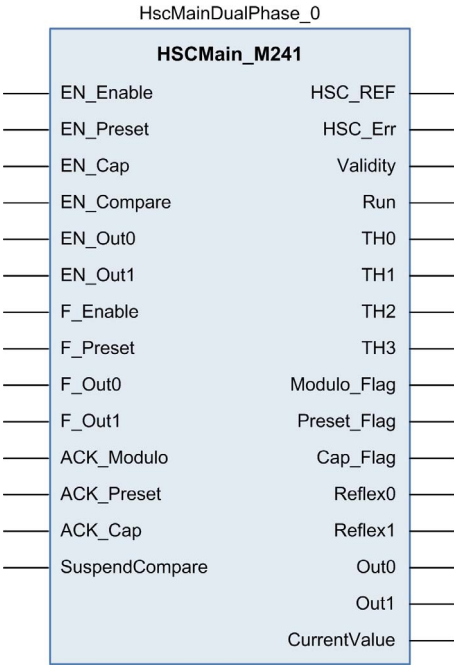
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>???  ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **Free-large** mode.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | When **EN** input is configured: if TRUE, authorizes the counter enable via the Enable input *(see page 137)*. |
| EN_Preset | BOOL | When **SYNC** input is configured: if TRUE, authorizes the counter Preset via the Sync input *(see page 134)*. |
| EN_Cap | BOOL | When **CAP** input is configured: if TRUE, enables the Capture input. |
| EN_Compare | BOOL | TRUE = enables the comparator operation *(see page 121)* (using Thresholds 0, 1, 2, 3):<br>• basic comparison (TH0, TH1, TH2, TH3 output bits)<br>• reflex (Reflex0, Reflex1 output bits)<br>• events (to trigger external tasks on threshold crossing) |
| EN_Out0 | BOOL | TRUE = enables physical output Out_R0 to echo the Reflex0 value (if configured). |
| EN_Out1 | BOOL | TRUE = enables physical output Out_R1 to echo the Reflex1 value (if configured). |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | On rising edge, presets and starts the counter. |
| F_Out0 | BOOL | TRUE = forces Out_R0 to 1 (if Reflex0 is configured in HSC Embedded Function. Takes priority over EN_Out0. |
| F_Out1 | BOOL | TRUE = forces Out_R1 to 1 (if Reflex1 is configured in HSC Embedded Function. Takes priority over EN_Out1. |
| ACK_Modulo | BOOL | On rising edge, resets Modulo_Flag. |
| ACK_Preset | BOOL | On rising edge, resets Preset_Flag. |
| ACK_Cap | BOOL | On rising edge, resets Cap_Flag. |
| SuspendCompare | BOOL | TRUE = compare results are suspended:<br>• TH0, TH1, TH2, TH3 , Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value.<br>• Hardware Outputs 0, 1 maintain their last value.<br>• Events are masked.<br>**NOTE:** EN_Compare, EN_ReflexO, EN_Reflex1, F_Out0, F_Out1 remain operational while SuspendCompare is set. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used as input of **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid. |
| Run | BOOL | Not relevant |
| TH0 | BOOL | Set to 1 when CurrentValue > Threshold 0 *(see page 121)*. |
| TH1 | BOOL | Set to 1 when CurrentValue > Threshold 1 *(see page 121)*. |
| TH2 | BOOL | Set to 1 when CurrentValue > Threshold 2 *(see page 121)*. |
| TH3 | BOOL | Set to 1 when CurrentValue > Threshold 3 *(see page 121)*. |
| Modulo_Flag | BOOL | Set to 1 when the counter rollovers its limits. |
| Preset_Flag | BOOL | Set to 1 by the preset of the counter *(see page 134)* |
| Cap_Flag | BOOL | Set to 1 when a new capture value is stored in the Capture register.<br>This flag must be reset before a new capture can occur. |
| Reflex0 | BOOL | State of Reflex0.<br>Only active when EN_Compare is set. |
| Reflex1 | BOOL | State of Reflex1.<br>Only active when EN_Compare is set. |
| Out0 | BOOL | State of physical outputs Out_R0 (if Reflex0 is configured in HSC Embedded Function, otherwise FALSE if not configured). |
| Out1 | BOOL | State of physical outputs Out_R1 (if Reflex1 is configured in HSC Embedded Function, otherwise FALSE if not configured). |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* which can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| EXPERT_PRESET | to get or set the Preset value of the HSC |
| EXPERT_THRESHOLD0 | to get or set the Threshold 0 value of an HSC |
| EXPERT_THRESHOLD1 | to get or set the Threshold 1 value of an HSC |
| EXPERT_THRESHOLD2 | to get or set the Threshold 2 value of an HSC |
| EXPERT_THRESHOLD3 | to get or set the Threshold 3 value of an HSC |
| EXPERT_REFLEX0 | to get or set output 0 reflex mode of an expert function |
| EXPERT_REFLEX1 | to get or set output 0 reflex mode of an expert function |

# Part V
# Event Counting Mode

## Overview

This part describes the use of an HSC in **Event Counting** mode.

## What Is in This Part?

This part contains the following chapters:

# Chapter 11
# Event Counting Principle

## Event Counting Mode Principle Description

### Overview

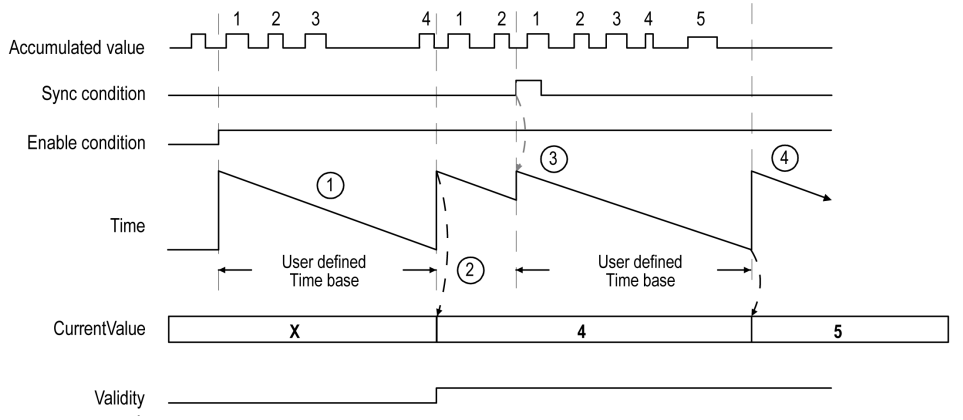
The **Event Counting** mode allows you to count a sequence of events during a given period of time.

### Principle

The counter assesses the number of pulses applied to the input for a predefined period of time. The counting register is updated at the end of each period with the number of events received.

The synchronization can be used over the time period. This restarts the counting event for a new predefined time period. The counting restarts at the edge Sync condition *(see page 134)*.

### Principle Diagram




| Stage | Action |
|---|---|
| 1 | When Enable condition = 1, the counter accumulates the number of events (pulses) on the physical input during a predefined period of time.<br>If Validity = 0, the current value is not relevant. |
| 2 | Once the first period of time has elapsed, the counter value is set to the number of events counted over the period and Validity is set to 1.<br>The counting restarts for a new period of time. |

| Stage | Action |
|---|---|
| 3 | On the rising edge of the Sync condition:<br>● the accumulated value is reset to 0<br>● the current value is not updated<br>● the counting restarts for a new period of time |
| 4 | Once the period of time has elapsed, the counter value is set to the number of events counted over the period.<br>The counting restarts for a new period of time. |

**NOTE:**
On the **Main** type, when the Enable condition is:
- Set to 0: the current counting is aborted and `CurrentValue` is maintained to the previous valid value.
- Set to 1: the accumulated value is reset to 0, the `CurrentValue` remains unchanged, and the counting restarts for a new period of time.

# Chapter 12
# Event Counting With a Main Type

## Overview

This chapter describes how to implement a High Speed Counter in **Event Counting** mode using a **Main** type.

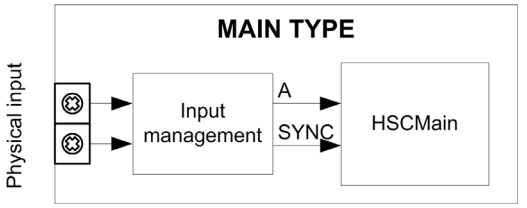
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main** type in **Event Counting** mode.




A is the counting input of the counter.

SYNC is the synchronization input of the counter.

## Optional Function

In addition to the **Event Counting** mode, the **Main** type provides the Preset function *(see page 134)*.

## Configuration of the Main Type Single Phase in Event Counting Mode

### Procedure

Follow this procedure to configure a **Main** type single phase in **Event Counting** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **HSC Main Single Phase**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → Counting Mode** parameter to **Event Counting**. |
| 5 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Set the value of the **Range → Time base** parameter to determine the period during which the number of events is counted.<br>Select the measurement of the update cycle time:<br>● 0.1 s<br>● 1 s (default value)<br>● 10 s<br>● 60 s |
| 8 | Optionally, set the value of the **Control inputs → SYNC input → Location** parameter to enable the Preset Function *(see page 134)*. |

# Programming the Main Type

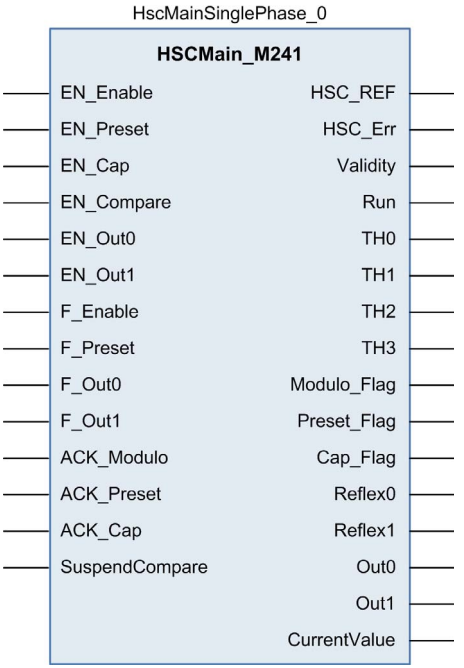
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>???  ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

These tables describe how the different pins of the function block are used in the mode **Event**.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | Not used |
| EN_Preset | BOOL | When **SYNC** input is configured: if TRUE, authorizes the counter Preset via the Sync input *(see page 134)*. |
| EN_Cap | BOOL | Not used |
| EN_Compare | BOOL | Not used |
| EN_Out0 | BOOL | Not used |
| EN_Out1 | BOOL | Not used |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | On rising edge, restarts the internal timer relative to the time base. |
| F_Out0 | BOOL | Not used |
| F_Out1 | BOOL | Not used |
| ACK_Modulo | BOOL | Not used |
| ACK_Preset | BOOL | On rising edge, resets Preset_Flag. |
| ACK_Cap | BOOL | Not used |
| SuspendCompare | BOOL | Not used |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used with the EXPERT_REF_IN input pin of the **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>EXPERTGetDiag *(see page 156)* function block may be used to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid. |
| Run | BOOL | Counter is running |
| TH0 | BOOL | Not relevant |
| TH1 | BOOL | Not relevant |
| TH2 | BOOL | Not relevant |
| TH3 | BOOL | Not relevant |
| Modulo_Flag | BOOL | Not relevant |

| Outputs | Type | Comment |
|---|---|---|
| Preset_Flag | BOOL | Set to 1 by the preset of the counter *(see page 134)* |
| Cap_Flag | BOOL | Not relevant |
| Reflex0 | BOOL | Not relevant |
| Reflex1 | BOOL | Not relevant |
| Out0 | BOOL | Not relevant |
| Out1 | BOOL | Not relevant |
| CurrentValue | DINT | Current value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* which can be read or modified while the program is running:

| Parameter | Type | Description |
|---|---|---|
| EXPERT_TIMEBASE | EXPERT_HSCMAIN_TIMEBASE_TYPE<br>For more information, refer to Type for HSC *(see page 148)*. | To get or set the Timebase value of the HSC. |

# Part VI
# Frequency Meter Type

## Overview

This part describes the use of an HSC in **Frequency meter** type.

## What Is in This Part?

This part contains the following chapters:

# Chapter 13
# Frequency Meter Principle

## Description

### Overview

The **Frequency meter** type measures an event frequency in Hz.

The **Frequency meter** type calculates the number of pulses in time intervals of 1 s. An updated value in Hz is available for each time base value (10, 100, or 1000 ms).

When there is a variation in the frequency, the value restoration time is 1 s with a value precision of 1 Hz.

### Operation Limits

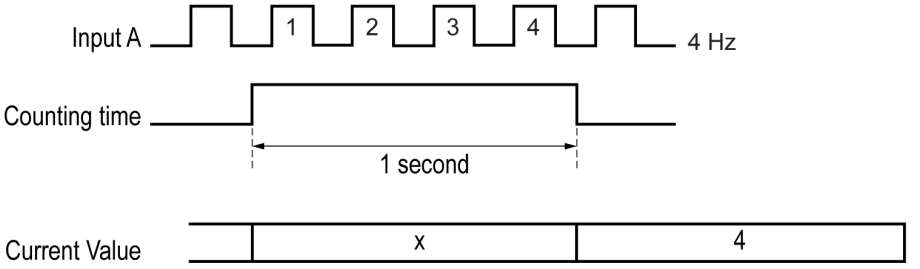
The maximum frequency that the module can measure on the A input is 200 kHz. Beyond 200 kHz, the counting register value may decrease until it reaches 0.

If the expert function is configured with a regular I/O, the minimum period admissible is 0.4 ms.

The maximum duty cycle at 200 kHz is 60%.

### Synopsis Diagram

This diagram provides an overview of the **Frequency meter** principle:



# Chapter 14
# Frequency Meter Type

## Overview

This chapter describes how to implement a High Speed Counter in **Frequency meter** type.

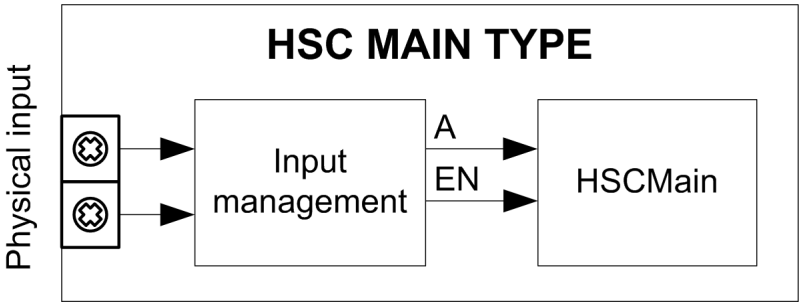
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main** type in **Frequency meter** type:




A is the counting input of the counter.

EN is the enable input of the counter.

## Optional Function

In addition to the **Frequency meter** type, the **Main** type can provide the following function:

- Enable function *(see page 137)*

# Configuration of the Frequency Meter Type

## Procedure

Follow this procedure to configure a **Frequency Meter** type:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **Frequency Meter**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE:** **Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | In **Counting Inputs → A input → Location** select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 5 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 6 | Set the value of the **Range → Time base** parameter to determine the period during which the number of events is counted.<br>Select the measurement of the update cycle time:<br>● 10 ms<br>● 100 ms<br>● 1000 ms (default value) |
| 7 | Optionally, set the value of the **Control inputs → EN input → Location** parameter to enable the Enable Function *(see page 137)*. |

# Programming

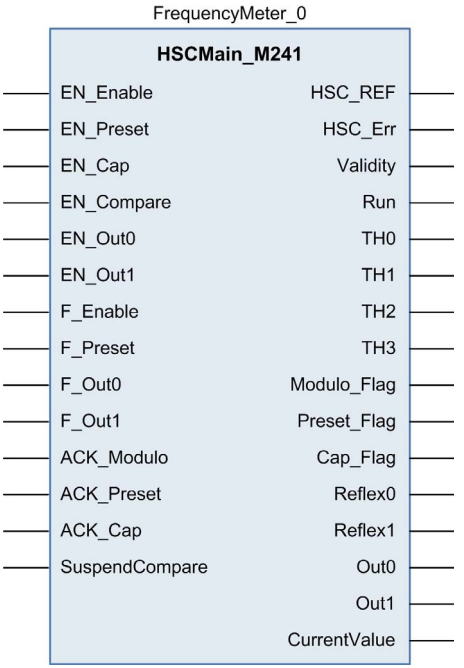
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>???  ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **Frequency meter** type.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | When **EN** input is configured: if TRUE, authorizes the counter enable via the Enable input *(see page 137)*. |
| EN_Preset | BOOL | Not used |
| EN_Cap | BOOL | Not used |
| EN_Compare | BOOL | Not used |
| EN_Out0 | BOOL | Not used |
| EN_Out1 | BOOL | Not used |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | On rising edge, restarts the internal timer relative to the time base. |
| F_Out0 | BOOL | Not used |
| F_Out1 | BOOL | Not used |
| ACK_Modulo | BOOL | Not used |
| ACK_Preset | BOOL | On rising edge, resets . |
| ACK_Cap | BOOL | Not used |
| SuspendCompare | BOOL | Not used |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used with the EXPERT_REF_IN input pin of the **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid. |
| Run | BOOL | Counter is running |
| TH0 | BOOL | Not relevant |
| TH1 | BOOL | Not relevant |
| TH2 | BOOL | Not relevant |
| TH3 | BOOL | Not relevant |

| Outputs | Type | Comment |
|---|---|---|
| Modulo_Flag | BOOL | Not relevant |
| Preset_Flag | BOOL | Set to 1 by the preset of the counter *(see page 134)* |
| Cap_Flag | BOOL | Not relevant |
| Reflex0 | BOOL | Not relevant |
| Reflex1 | BOOL | Not relevant |
| Out0 | BOOL | Not relevant |
| Out1 | BOOL | Not relevant |
| CurrentValue | DINT | Current counter value of the counter. |

# Part VII
# Period Meter Type

## Overview

This part describes the use of an HSC in **Period meter** type.

## What Is in This Part?

This part contains the following chapters:

# Chapter 15
# Period Meter Type Principle

## Description

### Overview

Use the **Period meter** type to:
- Determine the duration of an event
- Determine the time between two events
- Set and measure the execution time for a process

The **Period meter** can be used in two ways:
- Edge to opposite: Allows measurement of the duration of an event.
- Edge to edge: Allows measurement of the time between two events.

A timeout value can be specified in the configuration screen. Measurement is stopped if this timeout value is exceeded. In this case, the counting register is not valid until the next complete measurement.

The measurement is expressed in the units defined by the **Resolution** parameter (0.1 µs, 1 µs, 100 µs, 1000 µs).

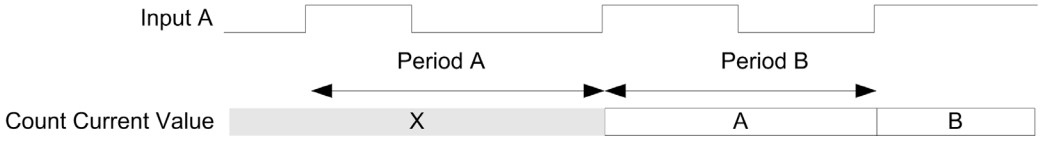
For example, if `CurrentValue` = 100 and the **Resolution** parameter is:
**0.0001 (0.1 µs)** measurement = 0.01 ms
**0.001 (1 µs)** measurement = 0.1 ms
**0.1 (100 µs)** measurement = 10 ms
**1 (1000 µs)** measurement = 100 ms

### Edge to Opposite Mode

When the Enable condition = 1, the measurement is taken between the rising edge and the falling edge of the A input. The counting register is updated as soon as the falling edge is detected.

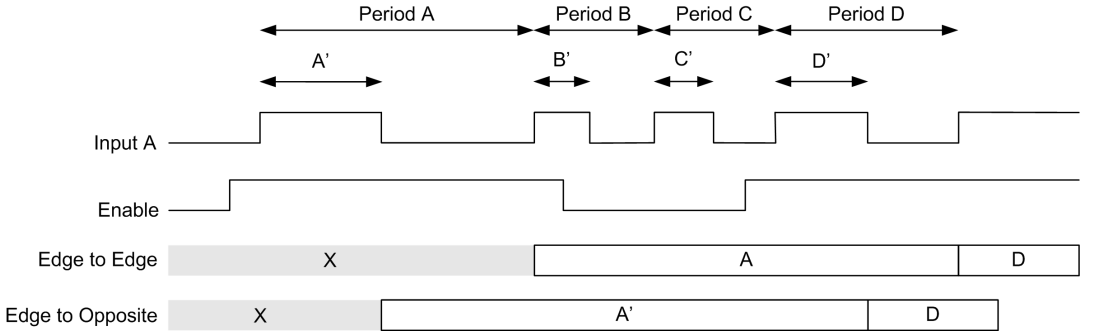
## Edge to Edge Mode

When the Enable condition = 1, the measurement is taken between two rising edges of the A input. The counting register is updated as soon as the second rising edge is detected.




## Enable Condition Interruption Behavior

The trend diagram below describes the behavior of the counting register when the Enable condition is interrupted:




## Operating Limits

The module can perform a maximum of one measurement every 5 ms.

The shortest pulse that can be measured is 100 µs, even if the unit defined in the configuration is 1 µs.

The maximum duration that can be measured is 1,073,741,823 units.

# Chapter 16
# Period Meter Type

## Overview

This chapter describes how to implement a High Speed Counter in **Period meter** type.

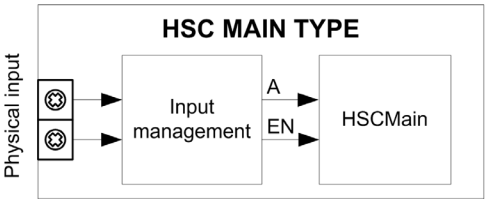
## What Is in This Chapter?

This chapter contains the following topics:

# Synopsis Diagram

## Synopsis Diagram

This diagram provides an overview of the **Main type** in **Period meter** type:




A is the counting input of the counter.

EN is the enable input of the counter.

## Optional Function

In addition to the **Period meter** type, the **Main** type can provide the following function:

- Enable function *(see page 137)*

## Configuration of the Period Meter Type in Edge to Edge Mode

### Procedure

Follow this procedure to configure a **Period Meter** type in **Edge to Edge** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **Period Meter**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE:** **Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → PeriodMeter Mode** parameter to **Edge to Edge**. |
| 5 | In **Counting Inputs → A input → Location**, select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the inputs.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Set the value of the **Range → Resolution** parameter.<br>Select the unit of measurement:<br>● 0.1 µs<br>● 1 µs (default value)<br>● 100 µs<br>● 1000 µs |
| 8 | Enter the value of the **Range → Timeout** parameter to set the time value that a measured period shall not exceed. |
| 9 | Optionally, set the value of the **Control inputs → EN input → Location** parameter to enable the Enable Function *(see page 137)*. |

## Configuration of the Period Meter Type in Edge to Opposite Mode

### Procedure

Follow this procedure to configure a **Period Meter** type in **Edge to Opposite** mode:

| Step | Action |
|---|---|
| 1 | Double-click **MyController → Counters**.<br>**Result**: **Counters** editor tab opens for HSC configuration.<br>**NOTE:** A message appears at the bottom of the configuration screen if the maximum number of HSC Main functions has already been configured. Consider using an HSC Simple function instead. |
| 2 | In the **Counters** editor tab, set the value of the **Counting function** parameter to **Period Meter**.<br>**Result**: The configuration parameters appear in the **Counters** editor tab. |
| 3 | If necessary, enter the value of the **General → Instance name** parameter.<br>**NOTE: Instance name** is automatically given by the software and can be used as it is for the counter function block. |
| 4 | Set the value of the **General → PeriodMeterMode** parameter to **Edge to Opposite**. |
| 5 | In **Counting Inputs → A input → Location**, select the regular or fast input to use as the A input.<br>**NOTE:** A message is displayed at the bottom of the configuration window if no more I/Os are available for configuration. Free up one or more I/Os before continuing configuration of this function. |
| 6 | Set the value of the **Counting inputs → A input → Bounce filter** parameter to reduce the bounce effect on the inputs.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 7 | Set the value of the **Range → Resolution** parameter.<br>Select the unit of measurement:<br>● 0.1 µs<br>● 1 µs (default value)<br>● 100 µs<br>● 1000 µs |
| 8 | Enter the value of the **Range → Timeout** parameter to set the time value that a measured period shall not exceed. |
| 9 | Optionally, set the value of the **Control inputs → EN input → Location** parameter to enable the Enable Function *(see page 137)*. |

# Programming

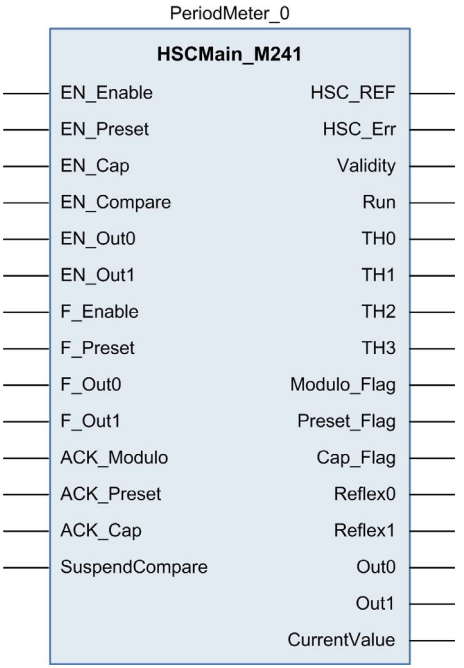
## Overview

**Main** type is always managed by an HSCMain_M241 function block.

**NOTE:** At build, a detected error code is given if the HSCMain_M241 function block is used to manage a different HSC type.

## Adding the HSCMain Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → HSC → HSCMain_M241** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Type the **Main** type instance name (defined in configuration) or select the function block instance by clicking:<br>???  ...<br>Using the input assistant, the HSC instance can be selected at the following path: **<MyController> → Counters**. |



## I/O Variables Usage

The tables below describe how the different pins of the function block are used in **Period meter** type.

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | When **EN** input is configured: if TRUE, authorizes the counter enable via the Enable input *(see page 137)*. |
| EN_Preset | BOOL | Not used |
| EN_Cap | BOOL | Not used |
| EN_Compare | BOOL | Not used |
| EN_Out0 | BOOL | Not used |
| EN_Out1 | BOOL | Not used |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| F_Preset | BOOL | Not used |
| F_Out0 | BOOL | Not used |
| F_Out1 | BOOL | Not used |
| ACK_Modulo | BOOL | Not used |
| ACK_Preset | BOOL | Not used |
| ACK_Cap | BOOL | Not used |
| SuspendCompare | BOOL | Not used |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC.<br>To be used with the EXPERT_REF_IN input pin of the **Administrative** function blocks. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the EXPERTGetDiag *(see page 156)* function block used to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid.<br>If the time-out value is exceeded, Validity = FALSE. |
| Run | BOOL | TRUE = Counter is running. |
| TH0 | BOOL | Not relevant |
| TH1 | BOOL | Not relevant |
| TH2 | BOOL | Not relevant |

| Outputs | Type | Comment |
|---|---|---|
| TH3 | BOOL | Not relevant |
| Modulo_Flag | BOOL | Not relevant |
| Preset_Flag | BOOL | Not relevant |
| Cap_Flag | BOOL | Not relevant |
| Reflex0 | BOOL | Not relevant |
| Reflex1 | BOOL | Not relevant |
| Out0 | BOOL | Not relevant |
| Out1 | BOOL | Not relevant |
| CurrentValue | DINT | Current value of the counter. |

# Adjusting Parameters

## Overview

The list of parameters described in the table below can be read or modified by using the EXPERTGetParam *(see page 160)* or EXPERTSetParam *(see page 162)* function blocks.

**NOTE:** Parameters set via the program override the parameters values configured in the HSC configuration window. Initial configuration parameters are restored on cold or warm start *(see Modicon M241 Logic Controller, Programming Guide)*.

## Adjustable Parameters

This table provides the list of parameters from the EXPERT_PARAMETER_TYPE *(see page 150)* which can be read or modified while the program is running:

| Parameter | Description |
|---|---|
| EXPERT_TIMEBASE | To get or set the Resolution value of the HSC. |
| EXPERT_PERIODMETER_ RESOLUTION_TYPE | To dynamically read or modify the time base.<br>For more information, refer to Type for period meter *(see page 151)*. |

# Part VIII
# Optional Functions

## Overview

This part provides information on optional functions for HSC.

## What Is in This Part?

This part contains the following chapters:

# Chapter 17
# Comparison Function

## Overview

This chapter provides information on the comparison function for the HSC.

## What Is in This Chapter?

This chapter contains the following topics:

# Comparison Principle with a Main type

## Overview

The compare block with the **Main** type manages thresholds, reflex outputs and events in the following modes:

- One-shot *(see page 35)*
- Modulo-loop *(see page 49)*
- Free-Large *(see page 69)*

Comparison is configured in the Configuration screen *(see page 126)* by activating at least one threshold.

Comparison can be used to trigger:

- a programming action on thresholds *(see page 123)*
- an event on a threshold associated with an external task *(see page 123)*
- reflex outputs *(see page 123)*

## Principle of a Comparison

The **Main** type can manage up to four thresholds.

A threshold is a configured value that is compared to the current counting value. Thresholds are used to define up to five zones or to react to a value crossing the threshold value.

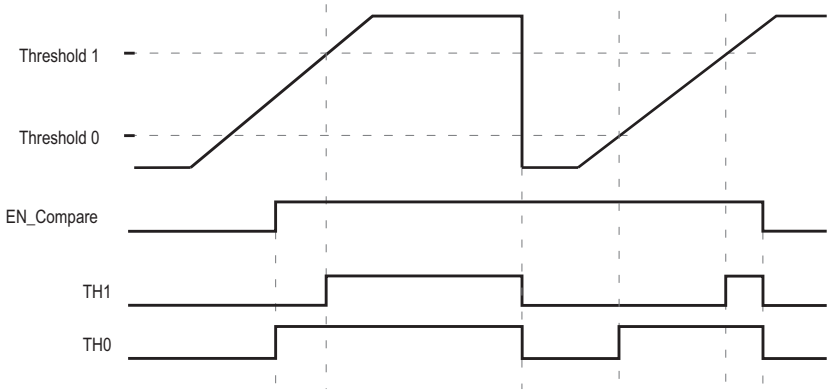
Threshold values are defined in the configuration window and can also be adjusted in the application program by using the EXPERTSetParam *(see page 162)* function block.

If Thresholdx (x= 0, 1, 2, 3) is configured and comparison is enabled (`EN_Compare` = 1), output pin THx of the `HSCMain_M241` function block is:

- set when counter value >= Thresholdx
- reset when counter value < Thresholdx

**NOTE:** When `EN_Compare` is set to 0 on `HSCMain_M241` function block, comparison functions are disabled, including external tasks triggered by a threshold event and Reflex outputs.

The following example for two thresholds shows comparison in the `HSCMain_M241` function block:




## Threshold Behavior

Using thresholds comparison status available in the task context (`TH0` to `TH2` output pins of the function block) is suitable for an application with a low time constant.

It can be used, for example, to monitor the liquid level in a tank.

## Configuring Event Triggering

Configuring an event on threshold crossing allows to trigger an external task *(see page 127)*. You can choose to trigger an event when a configured threshold is crossed as follows:

- **Upward Cross**. The event is triggered when the threshold value goes above the threshold value.
- **Downward Cross**. The event is triggered when the threshold value goes below the threshold value.
- **Both Cross**. The event is triggered when the threshold value goes above the threshold value and when the threshold value goes below the threshold value.

## Reflex Output Behavior

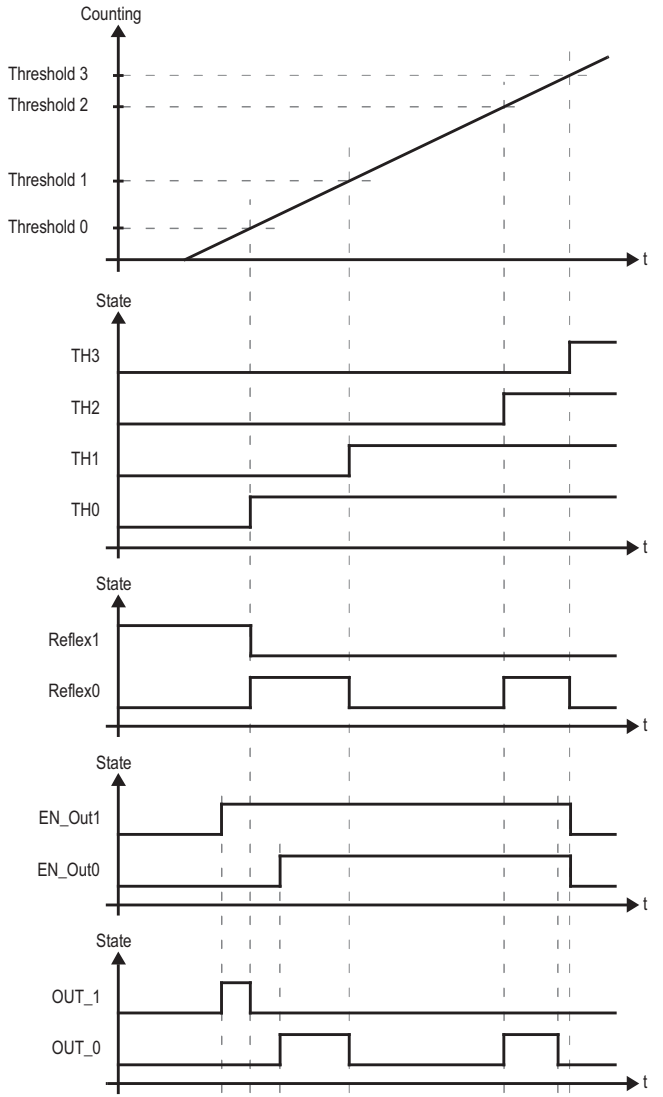
Configuring reflex outputs allows to trigger physical reflex outputs.

These outputs are not controlled in the task context, reducing the reaction time to a minimum. This is convenient for operations that need fast execution.

Outputs used by the High Speed Counter can only be accessed through the function block. They cannot be read or written directly within the application.

The performance is directly linked with the type of output used: fast or regular. For more information, refer to Embedded Expert I/O Assignment *(see page 17)*.

Example of the reflex outputs triggered by threshold:




**NOTE:** The state of the reflex outputs depends on the configuration.

## Changing the Threshold Values

Care must be exercised when threshold compares are active to avoid unintended or unexpected results from the outputs or from sudden Event task execution. If the compare function is disabled, threshold values can be modified freely. However, if the compare function is enabled, suspend at least the threshold compare function while modifying the threshold values.

**⚠ WARNING**

**UNINTENDED EQUIPMENT OPERATION**

- Do not change the Threshold values without using the `SuspendCompare` input if `EN_Compare` is equal to 1.
- Verify that `TH0` is less than `TH1`, that `TH1` is less than `TH2`, and that `TH2` is less than `TH3` before reactivating the threshold compare function.

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

While `EN_Compare` = 1, the comparison is active, and it is necessary to follow this procedure to apply changes to threshold values:

| Step | Action |
|---|---|
| 1 | Set `SuspendCompare` to 1.<br>The comparison is frozen at the current value:<br>• `TH0`, `TH1`, `Reflex0`, `Reflex1`, `Out0`, `Out1` output bits of the block maintain their last value.<br>• Physical Outputs 0, 1 maintain their last value<br>• Events are masked<br>**NOTE:** `EN_Compare`, `EN_Out0`, `EN_Out1`, `F_Out0`, `F_Out1` remain operational while `SuspendCompare` is set. |
| 2 | Modify the Threshold values as needed using the `EXPERTSetParam` *(see page 160)* function block.<br>**NOTE:** Follow this rule to configure the threshold values: TH0 < TH1 < TH2 < TH3. |
| 3 | Set `SuspendCompare` to 0.<br>The new Threshold values are applied and the comparison is resumed. |

# Configuration of the Comparison on a Main Type

## Configuration Procedure

Follow this procedure to configure the comparison function on a **Main** type:

| Step | Action |
|---|---|
| 1 | In the **Devices tree**, double-click **MyController → Counters**. |
| 2 | Set the value of the **Counting function** parameter to **HSC Main Single Phase** or **HSC Main Dual Phase**. |
| 3 | In the **Number of thresholds** parameter, select the number of thresholds to use. |
| 4 | Set the value of each threshold.<br>**NOTE:** Follow this rule to configure the threshold values: TH0 < TH1 < TH2 < TH3 |
| 5 | Optionally, define event conditions for the thresholds:<br>1. Configure external events *(see page 127)* associated with tasks.<br>2. In **Events → Threshold x**, set a trigger type (**Upward Cross**, **Downward Cross**, **Both Cross**)<br>3. In **HSC Main Id**, select the group of external events (HSC0...HSC3) containing the external event.<br>**Result:** External events in the selected group (HSCx_TH0, HSCx_TH1, HSCx_TH2, HSCx_TH3, HSCx_STOP) appear below **Threshold x External Event**. |

# External Event Configuration

## Procedure

The following procedure describes how to configure an external event *(see Modicon M241 Logic Controller, Programming Guide)* to activate a task:

| Step | Action |
|---|---|
| 1 | In the **Applications tree** tab, add a task. |
| 2 | Double-click the task node to associate it with to an external event. |
| 3 | In the **Type** dropdown menu, select **External**. |
| 4 | In the **External event** dropdown menu, select the event to associate to the task (see the list below). |

## External Events

This table provides a description of the possible external events to associate to a task:

| Event Name | Description |
|---|---|
| I0 | Task is activated when the input I0 is set to 1. |
| I1 | Task is activated when the input I1 is set to 1. |
| I2 | Task is activated when the input I2 is set to 1. |
| I3 | Task is activated when the input I3 is set to 1. |
| I4 | Task is activated when the input I4 is set to 1. |
| I5 | Task is activated when the input I5 is set to 1. |
| I6 | Task is activated when the input I6 is set to 1. |
| I7 | Task is activated when the input I7 is set to 1. |
| HSC0_TH0 | Task is activated when the threshold TH0 of the HSC0 is set to 1. |
| HSC0_TH1 | Task is activated when the threshold TH1 of the HSC0 is set to 1. |
| HSC0_TH2 | Task is activated when the threshold TH2 of the HSC0 is set to 1. |
| HSC0_TH3 | Task is activated when the threshold TH3 of the HSC0 is set to 1. |
| HSC0_STOP | Task is activated when the HSC0.Value is set to 0. |
| HSC1_TH0 | Task is activated when the threshold TH0 of the HSC1 is set to 1. |
| HSC1_TH1 | Task is activated when the threshold TH1 of the HSC1 is set to 1. |
| HSC1_TH2 | Task is activated when the threshold TH2 of the HSC1 is set to 1. |
| HSC1_TH3 | Task is activated when the threshold TH3 of the HSC1 is set to 1. |
| HSC1_STOP | Task is activated when the HSC1.Value is set to 0. |
| HSC2_TH0 | Task is activated when the threshold TH0 of the HSC2 is set to 1. |
| HSC2_TH1 | Task is activated when the threshold TH1 of the HSC2 is set to 1. |

| Event Name | Description |
|---|---|
| HSC2_TH2 | Task is activated when the threshold TH2 of the HSC2 is set to 1. |
| HSC2_TH3 | Task is activated when the threshold TH3 of the HSC2 is set to 1. |
| HSC2_STOP | Task is activated when the HSC2.Value is set to 0. |
| HSC3_TH0 | Task is activated when the threshold TH0 of the HSC3 is set to 1. |
| HSC3_TH1 | Task is activated when the threshold TH1 of the HSC3 is set to 1. |
| HSC3_TH2 | Task is activated when the threshold TH2 of the HSC3 is set to 1. |
| HSC3_TH3 | Task is activated when the threshold TH3 of the HSC3 is set to 1. |
| HSC3_STOP | Task is activated when the HSC3.Value is set to 0. |

**NOTE:** The Stop event is only available on HSC Main Single Phase, One-shot mode.

# Chapter 18
# Capture Function

## Overview

This chapter provides information on capture function for HSC.

## What Is in This Chapter?

This chapter contains the following topics:

# Capture Principle with a Main Type

## Overview

The capture function stores the current counter value upon an external input signal.

The capture function is available in **Main** type with the following modes:

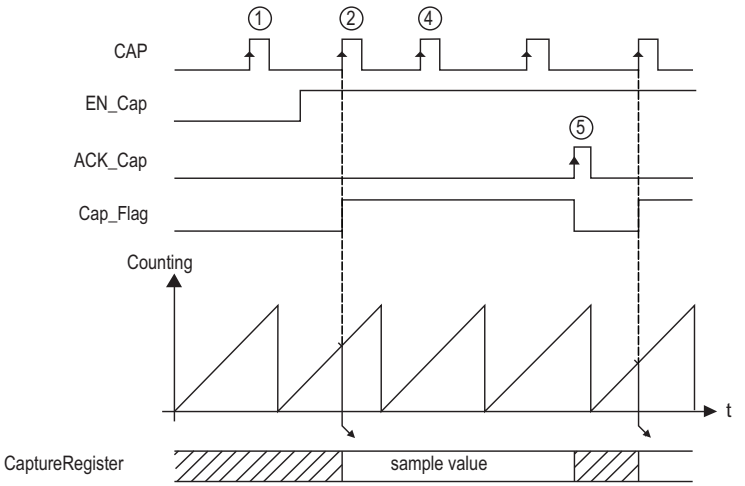
- One-shot *(see page 41)*
- Modulo-loop *(see page 61)*
- Free-large *(see page 77)*

Using this function requires to:

- configure the optional Capture input: **CAP**
- use `EXPERTGetCapturedValue` *(see page 154)* function block to retrieve the captured value in your application.

## Principle of a Capture

This graphic illustrates how the capture works in **Modulo-loop** mode:




| Stage | Action |
|---|---|
| 1 | When `EN_Cap` = 0, the function is not operational. |
| 2 | When `EN_Cap` = 1, the edge on **CAP** captures the current counter value and puts it into the Capture register, and triggers the rising edge of `Cap_Flag`. |
| 3 | Get the stored value using `EXPERTGetCapturedValue` *(see page 154)*. |
| 4 | While `Cap_Flag` = 1, any new edge on the physical input CAP is ignored. |

| Stage | Action |
|---|---|
| 5 | The rising edge of `HSCMain_M241` *(see page 164)* function block input `ACK_Cap` triggers the falling edge `Cap_Flag` output.<br>A new capture is authorized. |

# Configuration of the Capture on a Main Type

## Configuration Procedure

Follow this procedure to configure the capture function on a **Main** type:

| Step | Action |
|---|---|
| 1 | In the **Devices tree**, double-click **MyController → Counters**. |
| 2 | Set the value of the **Counting function** parameter to **HSC Main Single Phase** or **HSC Main Dual Phase**. |
| 3 | Select a value for the **Capture → CAP input → Location**. |
| 4 | Select a value for the **Capture → CAP input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |
| 5 | Select a triggering mode for the **Capture → Mode** parameter:<br>• Preset *(see page 134)* (default value)<br>• **CAP Rising**<br>• **CAP Falling**<br>• **CAP Both** |

# Chapter 19
# Preset and Enable Functions

## Overview

This chapter provides information on preset and enable functions for a HSC.

## What Is in This Chapter?

This chapter contains the following topics:

# Preset Function

## Overview

The preset function is used to set/reset the counter operation.

The preset function authorizes counting function, synchronization, and start in the following counting modes:

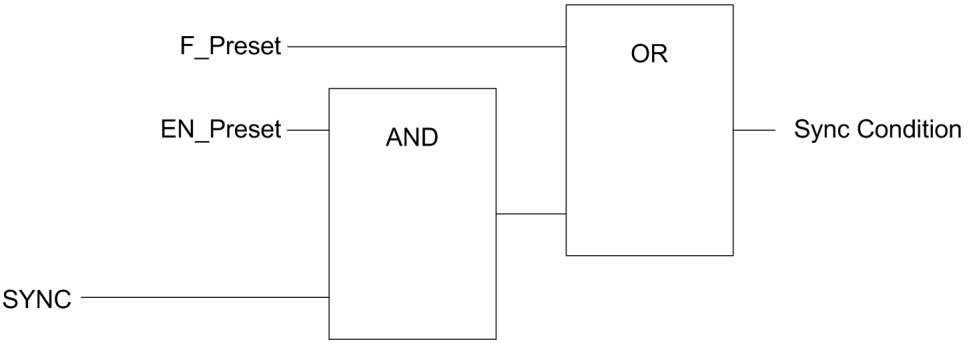
- **One shot** counter: preset and start the counter
- **Modulo-loop** counter: reset and start the counter
- **Event counting**: restart the internal time base at the beginning

**NOTE:** Sync condition for a **Simple** HSC type corresponds to the function block input `Sync`.

## Description

This function is used to synchronize the counter depending on the status and the configuration of the optional SYNC physical input and the function block inputs `F_Preset` and `EN_Preset`.

This diagram illustrates the Sync conditions of the HSC:




**EN_Preset** input of the HSC function block
**F_Preset** input of the HSC function block
**SYNC** physical input SYNC

The function block output `Preset_Flag` is set 1 when the Sync Condition is reached.

Either of the following events trigger the capturing of the Sync Condition:

- Rising edge of the `F_Preset` input
- Rising edge, falling edge, or rising and falling edge, of the SYNC physical input (if the SYNC input is configured, and the `EN_Preset` input is TRUE).

## Configuration

This procedure describes how to configure a preset function:

| Step | Action |
|---|---|
| 1 | In the **Devices tree**, double-click **MyController → Counters**. |
| 2 | Set the value of the **Counting function** parameter to **HSC Main Single Phase** or **HSC Main Dual Phase**. |
| 3 | Select the value of the **Control inputs → SYNC input → Location** parameter. |
| 4 | Select the value of the **Control inputs → SYNC input → Bounce filter** parameter. |
| 5 | Select the value of the **Control inputs → SYNC input → Preset condition** parameter to specify the transition type of the SYNC physical input:<br>• **SYNC Rising**. Rising edge of the SYNC input<br>• **SYNC Falling**. Falling edge of the SYNC input<br>• **SYNC Both**. Both edges of the SYNC input |

## Free-large or Period Meter Preset Conditions

### Overview

In **Free-large** mode, the Preset condition is created by using one physical input:

- SYNC

Preset condition available:

- At the edge of the input SYNC (rising)

### At the Edge of the Input SYNC (Rising)

The counter synchronizes upon the encoder reference point.

# Enable Function

## Overview

The enable function is used to authorize the counting operation.

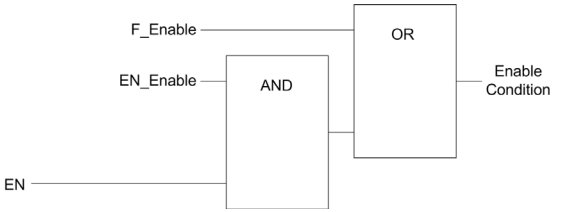
The enable function is available in the following HSC modes:

- HSC Main Single Phase (One-shot)
- HSC Main Single Phase (Modulo Loop)
- Frequency Meter
- Period Meter

## Description

This function is used to authorize changes to the current counter value depending on the status of the optional `EN` physical input and the function block inputs `F_Enable` and `EN_Enable`.

The following diagram illustrates the enable conditions:




**EN_Enable** input of the HSC function block
**F_Enable** input of the HSC function block
**EN** physical input Enable

As long as the function is not enabled, the counting pulses are ignored.

**NOTE:** Enable condition for a **Simple** type corresponds to the function block input `Enable`.

## Configuration

This procedure describes how to configure an Enable function:

| Step | Action |
|---|---|
| 1 | In the **Devices tree**, double-click **MyController → Counters**. |
| 2 | Select the **Counters** tab. |
| 3 | Select a **Counting function** that supports the Enable function:<br>• **HSC Main Single Phase** (**One-shot** or **Modulo-loop**)<br>• **Frequency Meter**<br>• **Period Meter** |
| 4 | Set the value of the **Control inputs → EN input → Location** parameter. |

| Step | Action |
|---|---|
| 5 | Select the value of the **Control inputs → EN input → Bounce filter** parameter to reduce the bounce effect on the input.<br>The filtering value determines the counter maximum frequency as shown in the Bounce Filter table *(see page 142)*. |

# 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:

# Appendix A
# General Information

## What Is in This Chapter?

This chapter contains the following topics:

# 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<br>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 |
| | 0.010 | 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.<br>B is the counting input of the dual phase counter. | | | |

## Dedicated Outputs

Outputs used by the Frequency Generator, Pulse Train Output, Pulse Width Modulation, and High Speed Counters can only be accessed through the function block. They can not be read or written directly within the application.

**⚠ WARNING**

**UNINTENDED EQUIPMENT OPERATION**

- Do not use the same function block instance in different program tasks.
- Do not modify or otherwise change the function block reference (AXIS) while the function block is executing.

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

# General Information on Administrative and Motion Function Block Management

## Management of Input Variables

At the `Execute` input rising edge, the function block starts.

Any further modifications of the input variables are not taken into account.

Following the IEC 61131-3 standards, if any variable input to a function block is missing, that is, left open or unconnected, then the value from the previous invocation of the instance of the function block will be used. In the first invocation, the initial, configured value is applied in this case. Therefore, it is best that a function block always has known values attributed to its inputs to help avoid difficulties in debugging your program. For HSC and PTO function blocks, it is best to use the instance only once, and preferably the instance be in the main task.

## Management of Output Variables

The `Done`, `InVelocity`, or `InFrequency` output is mutually exclusive with `Busy`, `CommandAborted`, and `Error` outputs: only one of them can be TRUE on one function block. If the `Execute` input is TRUE, one of these outputs is TRUE.

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

The `Done`, `InVelocity`, or `InFrequency` output is set when the function block execution has been completed successfully.

When a function block execution is interrupted by another one, the `CommandAborted` output is set instead.

When a function block execution ends due to a detected error, the `Error` output is set and the detected error number is given through the `ErrId` output.

The `Done`, `InVelocity`, `InFrequency`, `Error`, `ErrID`, and `CommandAborted` outputs are reset with the falling edge of `Execute`. If `Execute` input is reset before the execution is finished, then the outputs are set for one task cycle at the execution ending.

When an instance of a function block receives a new `Execute` before it is finished, the function block does not return any feedback, such as `Done`, for the previous action.

## Handling a Detected Error

All blocks have 2 outputs that can report a detected error during the execution of the function block:

- `Error` = `TRUE` when an error is detected.
- `ErrID` When `Error` = `TRUE`, returns the detected error ID.

# Appendix B
# Data Types

## Overview

This chapter describes the data types of the HSC Library.

## What Is in This Chapter?

This chapter contains the following topics:

# EXPERT_ERR_TYPE: Type for Error Variable of EXPERT Function Block

## Enumerated Type Description

The enumeration data type ENUM contains the different types of detected error with the following values:

| 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.<br>See PARAMETER_TYPE section for valid parameters *(see page 150)*. |
| EXPERT_INVALID_PARAMETER | 03 hex | The value of the parameter is incorrect.<br>For example, Preset Value is <TH1 or <TH0. |
| EXPERT_COM_ERROR | 04 hex | Communication error was detected with the EXPERT module. |
| EXPERT_CAPTURE_NOT_CONFIGURED | 05 hex | Capture is not configured.<br>It is impossible to get a captured value. |

# EXPERT_FREQMETER_TIMEBASE_TYPE: Type for Frequency Meter Time Base Variable

## Enumerated Type Description

The enumeration data type ENUM contains the different time base values allowed for use with an EXPERT function block:

| Name | Value |
|---|---|
| EXPERT_FREQMETER_10ms | 10 |
| EXPERT_FREQMETER_100ms | 100 |
| EXPERT_FREQMETER_1000ms | 1000 |

# EXPERT_HSCMAIN_TIMEBASE_TYPE: Type for HSC Main Time Base Variable

## Enumerated Type Description

The enumeration data type ENUM contains the different time base values allowed for use with an EXPERT Main function block:

| Name | Value |
|---|---|
| EXPERT_HSCMAIN_100ms | 00 hex |
| EXPERT_HSCMAIN_1s | 01 hex |
| EXPERT_HSCMAIN_10s | 02 hex |
| EXPERT_HSCMAIN_60s | 03 hex |

# EXPERT_IMMEDIATE_ERR_TYPE: Type for Error Variable of the GetImmediateValue Function Block

## Enumerated Type Description

The enumeration data type ENUM contains the different types of detected error with the following values:

| Enumerator | Value | Description |
|---|---|---|
| EXPERT_IMMEDIATE_FUNC_NO_ERROR | 00 hex | No error detected |
| EXPERT_IMMEDIATE_FUNC_UNKNOWN | 01 hex | The reference of IMMEDIATE function is incorrect or not configured |
| EXPERT_IMMEDIATE_FUNC_UNKNOWN_PARAMETER | 02 hex | A parameter reference is incorrect |

# EXPERT_PARAMETER_TYPE: Type for Parameters to Get or to Set on EXPERT

## Enumerated Type Description

The enumeration data type ENUM contains the following values:

| Enumerator | Value | Description |
|---|---|---|
| EXPERT_PRESET | 00 hex | To get or set the Preset value of an EXPERT function. |
| EXPERT_MODULO | 01 hex | To get or set the Modulo value of an EXPERT function. |
| EXPERT_TIMEBASE | 03 hex | To get or set the Timebase value *(see page 148)* of an EXPERT function. |
| EXPERT_THRESHOLD0 | 06 hex | To get or set the Threshold 0 value of an EXPERT function. |
| EXPERT_THRESHOLD1 | 07 hex | To get or set the Threshold 1 value of an EXPERT function. |
| EXPERT_THRESHOLD2 | 08 hex | To get or set the Threshold 2 value of an EXPERT function. |
| EXPERT_THRESHOLD3 | 09 hex | To get or set the Threshold 3 value of an EXPERT function. |
| EXPERT_REFLEX0 | 0A hex | To get or set output 0 reflex mode of an EXPERT function |
| EXPERT_REFLEX1 | 0B hex | To get or set output 1 reflex mode of an EXPERT function |

# EXPERT_PERIODMETER_RESOLUTION_TYPE: Type for Period Meter Time Base Variable

## Enumerated Type Description

The enumeration data type ENUM contains the different time base values allowed for use with an EXPERT function block:

| Name | Value |
|---|---|
| EXPERT_PERIODMETER_100ns | FFFFFFFF hex (-1 decimal) |
| EXPERT_PERIODMETER_1µs | 00 hex (0 decimal) |
| EXPERT_PERIODMETER_100µs | 01 hex (1 decimal) |
| EXPERT_PERIODMETER_1000µs | 02 hex (2 decimal) |

# EXPERT_REF: EXPERT Reference Value

## Data Type Description

The EXPERT_REF is a byte used to identify the EXPERT function associated with the administrative block.

# Appendix C
# Function Blocks

## Overview

This chapter describes the functions and the function blocks of the HSC Library.

## What Is in This Chapter?

This chapter contains the following topics:

# EXPERTGetCapturedValue: Read Value of Capture Registers

## Function Block Description

This administrative function block returns the content of a capture register.

## Graphical Representation


## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_IN` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. Must not be changed during block execution. |
| `Execute` | `BOOL` | On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates. |
| `CaptureNumber` | `BYTE` | Index of the capture register: 0 |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_OUT` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. |
| `Done` | `BOOL` | `TRUE` = indicates that `CaptureValue` is valid. Function block execution is finished. |
| `Busy` | `BOOL` | `TRUE` = indicates that the function block execution is in progress. |
| `Error` | `BOOL` | `TRUE` = indicates that an error was detected. Function block execution is finished. |

| Outputs | Type | Comment |
|---|---|---|
| ErrID | EXPERT_ERR_TYPE *(see page 146)* | When Error is TRUE: type of the detected error. |
| CaptureValue | DINT | When Done is TRUE: Capture register value is valid. |

**NOTE:** In case of detected error, variables take the last value captured.

**NOTE:** For more information about Done, Busy and Execution pins, refer to General Information on Function Block Management *(see page 143)*.

## Adding the EXPERTGetCapturedValue Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → Administrative → EXPERTGetCapturedValue** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Link the **EXPERT_REF_IN** input to the **HSC_REF** output of the HSC. |

# EXPERTGetDiag: Return Detail of a Detected HSC Error

## Function Block Description

This administrative function block returns the details of a detected HSC error.

## Graphical Representation


## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_IN` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. Must not be changed during block execution. |
| `Execute` | `BOOL` | On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_OUT` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. |
| `Done` | `BOOL` | `TRUE` = indicates that `HSCDiag` is valid. Function block execution is finished. |
| `Busy` | `BOOL` | `TRUE` = indicates that the function block execution is in progress. |
| `Error` | `BOOL` | `TRUE` = indicates that an error was detected. Function block execution is finished. |

| Outputs | Type | Comment |
|---|---|---|
| ErrID | EXPERT_ERR_TYPE *(see page 146)* | When Error is TRUE: type of the detected error. |
| EXPERTDiag | DWORD | When Done is TRUE: diagnostic value is valid, refer to the table below. |

**NOTE:** For more information about Done, Busy and Execution pins, refer to General Information on Function Block Management *(see page 143)*.

This table indicates the diagnostic values:

| Bit | BASE (HSCMain or HSCSimple) | Description |
|---|---|---|
| 0 | – | No error detected |
| 1 | – | Timeout reached on period meter |
| 2 | – | Shortcut detected on HSC Main expert output |
| 7 | – | Error detected in the configuration of the counter |

## Adding the EXPERTGetDiag Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → Administrative → EXPERTGetDiag** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Link the **EXPERT_REF_IN** input to the **HSC_REF** output of the HSC. |

# EXPERTGetImmediateValue: Read Counter Value of HSC

## Function Block Description

This administrative function block permits to read the counter value of an HSC bypassing the controller cycle.

## Graphical Representation


## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| EXPERT_REF_IN | EXPERT_REF *(see page 152)* | Reference to the EXPERT function block. |
| Execute | BOOL | On rising edge, starts the function block execution. On falling edge, resets the outputs of the function block when its execution terminates. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| EXPERT_REF_OUT | EXPERT_REF *(see page 152)* | Reference to the EXPERT function block. |
| Done | BOOL | TRUE = indicates that ExpertDiag is valid. Function block execution is finished. |
| Error | BOOL | TRUE = indicates that an error was detected. |
| ErrID | IMMEDIATE_FUNC_ERR_TYPE *(see page 149)* | When Error is TRUE: type of the detected error. |
| ImmediateValue | DINT | Contains the counter value. |

## Adding the EXPERTGetImmediateValue Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → Administrative → EXPERTGetImmediateValue** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Link the **EXPERT_REF_IN** input to the **HSC_REF** output of the HSC. |

# EXPERTGetParam: Return Parameters Value of an HSC

## Function Block Description

This administrative function block returns a parameter value of an HSC.

## Graphical Representation


## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_IN` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. Must not be changed during block execution. |
| `Execute` | `BOOL` | On rising edge, starts the function block execution.<br>On falling edge, resets the outputs of the function block when its execution terminates. |
| `Param` | `EXPERT_PARAMETER_TYPE` *(see page 150)* | Parameter to read. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_OUT` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. |
| `Done` | `BOOL` | `TRUE` = indicates that `ParamValue` is valid.<br>Function block execution is finished. |

| Outputs | Type | Comment |
|---|---|---|
| Busy | BOOL | TRUE = indicates that the function block execution is in progress. |
| Error | BOOL | TRUE = indicates that an error was detected.<br>Function block execution is finished. |
| ErrID | EXPERT_ERR_TYPE *(see page 146)* | When Error is TRUE: type of the detected error. |
| ParamValue | DINT | Value of the parameter that has been read. |

**NOTE:** For more information about Done, Busy and Execution pins, refer to General Information on Function Block Management *(see page 143)*.

## Adding the EXPERTGetParam Function Block

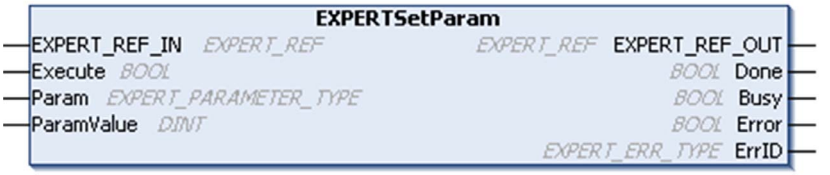
| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → Administrative → EXPERTGetParam** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Link the **EXPERT_REF_IN** input to the **HSC_REF** output of the HSC. |

# EXPERTSetParam: Adjust Parameters of a HSC

## Function Block Description

This administrative function block modifies the value of a parameter of an HSC.

## Graphical Representation




## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_IN` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block.<br>Must not be changed during block execution. |
| `Execute` | `BOOL` | On rising edge, starts the function block execution.<br>On falling edge, resets the outputs of the function block when its execution terminates. |
| `Param` | `EXPERT_PARAMETER_TYPE` *(see page 150)* | Parameter to read. |
| `ParamValue` | `DINT` | Parameter value to write. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| `EXPERT_REF_OUT` | `EXPERT_REF` *(see page 152)* | Reference to the EXPERT function block. |
| `Done` | `BOOL` | `TRUE` = indicates that the parameter was successfully written.<br>Function block execution is finished. |

| Outputs | Type | Comment |
|---|---|---|
| Busy | BOOL | TRUE = indicates that the function block execution is in progress. |
| Error | BOOL | TRUE = indicates that an error was detected. Function block execution is finished. |
| ErrID | EXPERT_ERR_TYPE *(see page 146)* | When Error is TRUE: type of the detected error. |

**NOTE:** For more information about Done, Busy, and Execution pins, refer to General Information on Function Block Management *(see page 143)*.

## Adding the EXPERTSetParam Function Block

| Step | Description |
|---|---|
| 1 | Select the **Libraries** tab in the **Software Catalog** and click **Libraries**.<br>Select **Controller → M241 → M241 HSC → Administrative → EXPERTSetParam** in the list, drag-and-drop the item onto the **POU** window. |
| 2 | Link the **EXPERT_REF_IN** input to the **HSC_REF** output of the HSC. |

# HSCMain_M241: Control a Main Type Counter for M241

## Function Block Description

This function block controls a **Main** type counter with the following functions:

- up/down counting
- frequency meter
- thresholds
- events
- period meter
- dual phase

The HSC Main function block is mandatory when using **Main** counter.

The function block instance name must match the name defined by configuration. Hardware related information managed by this function block is synchronized with the MAST task cycle.

**⚠ WARNING**

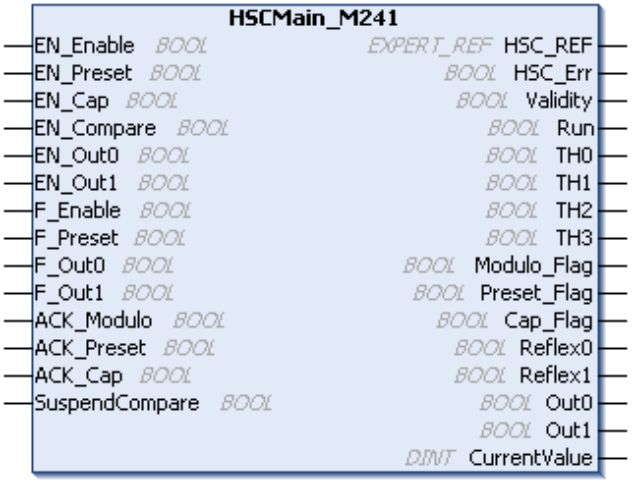
**UNINTENDED OUTPUT VALUES**

- Only use the Function Block instance in the MAST task.
- Do not use the same Function Block instance in a different task.

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

**NOTE:** Forcing the logical output values of the FB is allowed by SoMachine but it will have no impact on hardware related outputs if the function is active (executing).

## Graphical Representation




## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Input | Type | Description |
|---|---|---|
| EN_Enable | BOOL | TRUE = enables the counter. |
| EN_Preset | BOOL | TRUE = authorizes the counter synchronization and start via the Sync input. |
| EN_Cap | BOOL | TRUE = enables the Capture input (if configured in **One shot**, **Modulo loop**, **Free large** modes). |
| EN_Compare | BOOL | TRUE = enables the comparator operation (using Thresholds 0, 1, 2, 3):<br>• basic comparison (TH0, TH1, TH2, TH3 output bits)<br>• reflex (Reflex0, Reflex1 output bits)<br>• events (to trigger external tasks on threshold crossing) |
| EN_Out0 | BOOL | TRUE = enables Output0 to echo the Reflex0 value (if configured in **One shot**, **Modulo loop**, **Free large** modes). |
| EN_Out1 | BOOL | TRUE = enables Output1 to echo the Reflex1 value (if configured in **One shot**, **Modulo loop**, **Free large** modes). |
| F_Enable | BOOL | TRUE = authorizes changes to the current counter value. |

| Input | Type | Description |
|---|---|---|
| F_Preset | BOOL | On rising edge, authorizes counting function synchronization and start in the following counting modes:<br>**One-shot counter:** to preset and start the counter<br>**Modulo loop counter:** to reset and start the counter<br>**Free large counter:** to preset and start the counter<br>**Event counter:** to restart the internal time base at the beginning<br>**Frequency meter:** to restart the internal time base at the beginning |
| F_Out0 | BOOL | TRUE = forces Output0 to 1 (if configured in **One-shot**, **Modulo loop**, **Free large** modes). |
| F_Out1 | BOOL | TRUE = forces Output1 to TRUE (if configured in **One-shot**, **Modulo loop**, **Free large** modes). |
| ACK_Modulo | BOOL | On rising edge, resets Modulo_Flag (**Modulo loop** and **Free large** modes). |
| ACK_Preset | BOOL | On rising edge, resets Preset_Flag. |
| ACK_Cap | BOOL | On rising edge, resets the Cap_Flag (**One-shot**, **Modulo loop**, **Free large** modes). |
| SuspendCompare | BOOL | TRUE = compare results are suspended:<br>• TH0, TH1, TH2, TH3, Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value.<br>• Physical outputs Output0 and Output1 maintain their last value.<br>• Events are masked.<br>**NOTE:** EN_Compare, EN_Out0, EN_Out1, F_Out0, F_Out1 remain operational while SuspendCompare is set. |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC. |
| Validity | BOOL | TRUE = indicates that output values on the function block are valid.<br>In the **Period Meter** Type, if the time-out value is exceeded, Validity = FALSE.<br>In **One-Shot** mode, Validity is set to TRUE when a rising edge of **Preset** is detected. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected.<br>Use the HSCGetDiag *(see page 156)* function block to get more information about this detected error. |
| Run | BOOL | TRUE = counter is running.<br>In One-shot mode, the Run bit switches to 0 when CurrentValue reaches 0. |

| Outputs | Type | Comment |
|---|---|---|
| TH0 | BOOL | TRUE = current counter value > Threshold 0 (if configured in **One shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| TH1 | BOOL | TRUE = current counter value > Threshold 1 (if configured in **One shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| TH2 | BOOL | TRUE = current counter value > Threshold 2 (if configured in **One-shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| TH3 | BOOL | TRUE = current counter value > Threshold 3 (if configured in **One-shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| Modulo_Flag | BOOL | Set to TRUE by the rolls over of:<br>• **Modulo loop counter**: when the counter rolls over to the modulo or 0<br>• **Free large** counter: when the counter roll overs its limits |
| Preset_Flag | BOOL | Set to TRUE by the synchronization of:<br>• **One-shot** counter: when the counter presets and starts<br>• **Modulo loop** counter: when the counter resets<br>• **Free large** counter: when the counter presets<br>• **Event** counter: when the internal timer relative to the time base restarts<br>• **Frequency meter**: when the internal timer relative to the time base restarts |
| Cap_Flag | BOOL | TRUE = indicates that a value has been latched in the capture register.<br>This flag must be reset before a new capture can occur. |
| Reflex0 | BOOL | State of Reflex0 (if configured in **One shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| Reflex1 | BOOL | State of Reflex1 (if configured in **One shot**, **Modulo loop**, **Free large** modes).<br>Only active when EN_Compare is set. |
| Out0 | BOOL | Indicates the state of Output0. |
| Out1 | BOOL | Indicates the state of Output1. |
| CurrentValue | DINT | Current counter value of the counter. |

# HSCSimple_M241: Control a Simple Type Counter for M241

## Function Block Description

This function block controls a **Simple** type counter with the following reduced functions:

- one-channel counting
- no threshold
- no event
- no capture
- no reflex

The `HSCSimple` function block is mandatory when using a **Simple** counter type.

The function block instance name must match the name defined by configuration. Hardware related information managed by this function block is synchronized with the MAST task cycle.

**⚠ WARNING**

**UNINTENDED OUTPUT VALUES**

- Only use the Function Block instance in the MAST task.
- Do not use the same Function Block instance in a different task.

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

**NOTE:** Forcing the logical output values of the FB is allowed by SoMachine but it will have no impact on hardware related outputs if the function is active (executing).

## Graphical Representation


## IL and ST Representation

To see the general representation in IL or ST language, refer to *Function and Function Block Representation* *(see page 171)*.

## I/O Variables Description

This table describes the input variables:

| Inputs | Type | Comment |
|---|---|---|
| Enable | BOOL | TRUE = authorizes changes to the current counter value. |
| Sync | BOOL | On rising edge, presets and starts the counter. |
| ACK_Modulo | BOOL | On rising edge, resets the modulo flag Modulo_Flag (in Modulo loop mode). |

This table describes the output variables:

| Outputs | Type | Comment |
|---|---|---|
| HSC_REF | EXPERT_REF *(see page 152)* | Reference to the HSC. |
| HSC_Err | BOOL | TRUE = indicates that an error was detected. Use the EXPERTGetDiag *(see page 156)* function block used to get more information about this detected error. |
| Validity | BOOL | TRUE = indicates that the output values on the function block are valid. |
| Run | BOOL | TRUE = counter is running. In One-shot mode, switches to 0 when CurrentValue reaches 0. A rising edge on Sync is needed to restart the counter. |
| Modulo_Flag | BOOL | Set to 1 by the rolls over of a Modulo loop counter when the counter rolls overs the modulo. |
| CurrentValue | DWORD | Current count value of the counter. |

# Appendix D
# 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:

## 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`:

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    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.

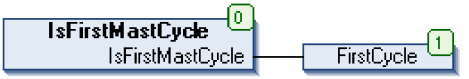
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

## Using a Function in IL Language

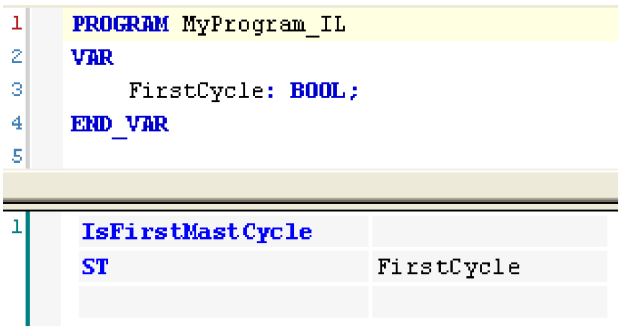
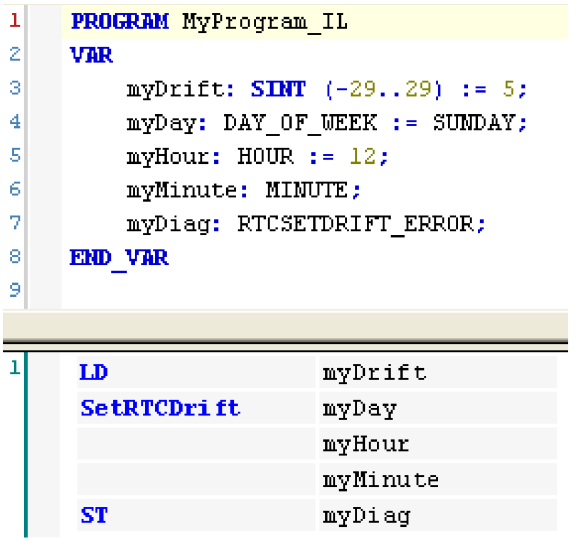
This procedure describes how to insert a function in IL language:

| Step | Action |
|---|---|
| 1 | Open or create a new POU in Instruction List language.<br>**NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*. |
| 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:<br>● type the name of the function in the operator column (left field), or<br>● use the **Input Assistant** to select the function (select **Insert Box** in the context menu). |
| 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:<br>`IsFirstMastCycle` |  |
| with input parameters:<br>`SetRTCDrift` |  |

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` |  |
| IL example of a function with input parameters: `SetRTCDrift` |  |

## Using a Function Block in IL Language

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

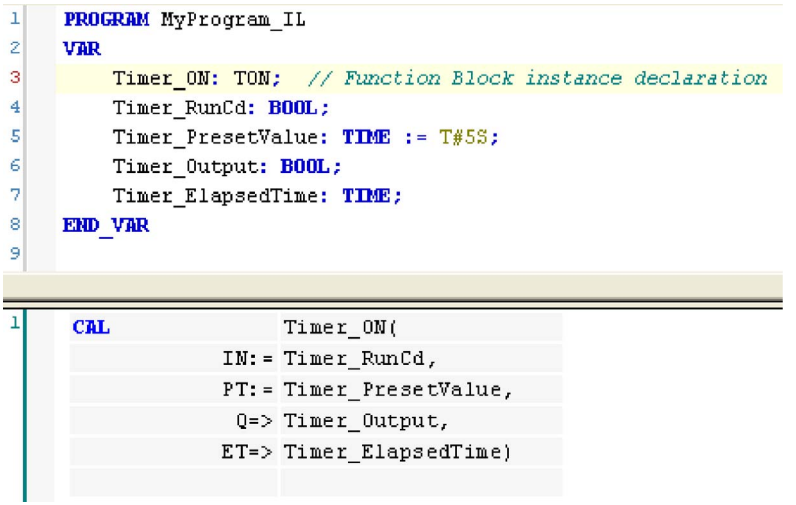
| Step | Action |
|---|---|
| 1 | Open or create a new POU in Instruction List language.<br>**NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*. |
| 2 | Create the variables that the function block requires, including the instance name. |

| Step | Action |
|---|---|
| 3 | Function Blocks are called using a CAL instruction:<br>● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu).<br>● Automatically, the CAL instruction and the necessary I/O are created.<br><br>Each parameter (I/O) is an instruction:<br>● Values to inputs are set by ":=".<br>● Values to outputs are set by "=>". |
| 4 | In the CAL 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:

| Function Block | Graphical Representation |
|---|---|
| TON |  |

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

| Function Block | Representation in POU IL Editor |
|---|---|
| TON |  |

# 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.<br>**NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*. |
| 2 | Create the variables that the function requires. |
| 3 | Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is:<br>`FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);` |

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

| Function | Graphical Representation |
|---|---|
| `SetRTCDrift` |  |

The ST language of this function is the following:

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

## Using a Function Block in ST Language

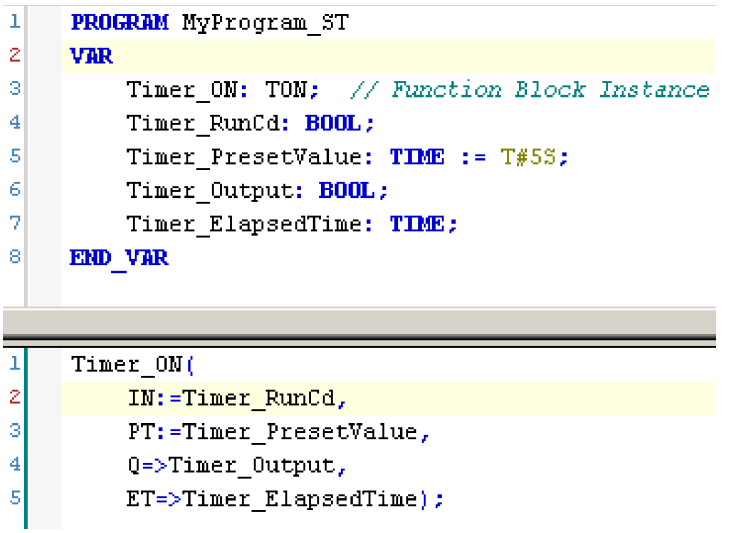
This procedure describes how to insert a function block in ST language:

| Step | Action |
|---|---|
| 1 | Open or create a new POU in Structured Text language.<br>**NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see SoMachine, Programming Guide)*. |
| 2 | Create the input and output variables and the instance required for the function block:<br>● Input variables are the input parameters required by the function block<br>● Output variables receive the value returned by the function block |
| 3 | Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is:<br>`FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,... Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);` |

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

| Function Block | Graphical Representation |
|---|---|
| `TON` |  |

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

| Function Block | Representation in POU ST Editor |
|---|---|
| TON |  |

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

# Glossary

## 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.

**configuration**

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

**controller**

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

## E

**expansion bus**

An electronic communication bus between expansion I/O modules and a controller.

## 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 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.

## I

**I/O**

(*input/output*)

**ID**

(*identifier/identification*)

**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

**node**

An addressable device on a communication network.

## 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.

**program**

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

**PTO**

(*pulse train outputs*) A fast output that oscillates between off and on in a fixed 50-50 duty cycle, producing a square wave form. PTO is especially well suited for applications such as stepper motors, frequency converters, and servo motor control, among others.

## 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.

## V

**variable**

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

# Index

## H



## M



## P