Magelis SCU
HMI Controller
HSC Library Guide

02/2014

EIO0000001512.04

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, 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 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 an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

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

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can 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 HMI SCU controller.

This documentation describes the functions and variables of the HMI SCU HSC library.

In order to use this manual, you must:

- Have a thorough understanding of the HMI SCU, 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)

NOTE: Read and understand this document and all related documents before installing, operating, or maintaining your HMI SCU.

The HMI SCU users should read through the entire document to understand all features.

Validity Note

This document has been updated with the release of SoMachine V4.1.

Related Documents

Title of Documentation	Reference Number
Magelis SCU HMI Controller Programming Guide	EIO0000001240 (eng), EIO0000001241 (fre), EIO0000001242 (ger), EIO0000001243 (spa), EIO0000001244 (ita), EIO0000001245 (chs)
Magelis SCU HMI Controller PLCSystem Library Guide	EIO0000001246 (eng), EIO0000001247 (fre), EIO0000001248 (ger), EIO0000001249 (spa), EIO0000001250 (ita), EIO0000001251 (chs)
Magelis SCU HMI Controller PTO/PWM Library Guide	EIO0000001518 (eng), EIO0000001519 (fre), EIO0000001520 (ger), EIO0000001521 (spa), EIO0000001522 (ita), EIO0000001523 (chs)
PLCCommunication Library Guide	EIO0000000361 (eng), EIO0000000742 (fre), EIO0000000743 (ger), EIO0000000744 (spa), EIO0000000745 (ita), EIO0000000746 (chs)
Magelis SCU HMI Controller Hardware Guide	EIO0000001232 (eng), EIO0000001233 (fre), EIO0000001234 (ger), EIO0000001235 (spa), EIO0000001236 (ita), EIO0000001237 (chs), EIO0000001238 (por)

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

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
High Speed Counter Overview

Chapter 1
Embedded Functions

Overview

This chapter describes how to configure the embedded functions of the Magelis SCU HMI Controller.

The number of inputs and outputs dedicated to the embedded function depends on the HMI controller reference *(see Magelis SCU, HMI Controller, Programming Guide)*.

What Is in This Chapter?

This chapter contains the following topics:

HSC Embedded Function

Overview

The HSC function can execute fast counts of pulses from sensors, encoders, switches, and so on, that are connected to the dedicated fast inputs.

There are 2 types of HSC:

- **Simple** type: a single input counter *(see page 19)*.
- **Main** type: a counter that uses up to 4 inputs (2 fast inputs and 2 standard inputs) and 2 reflex outputs *(see page 20)*.

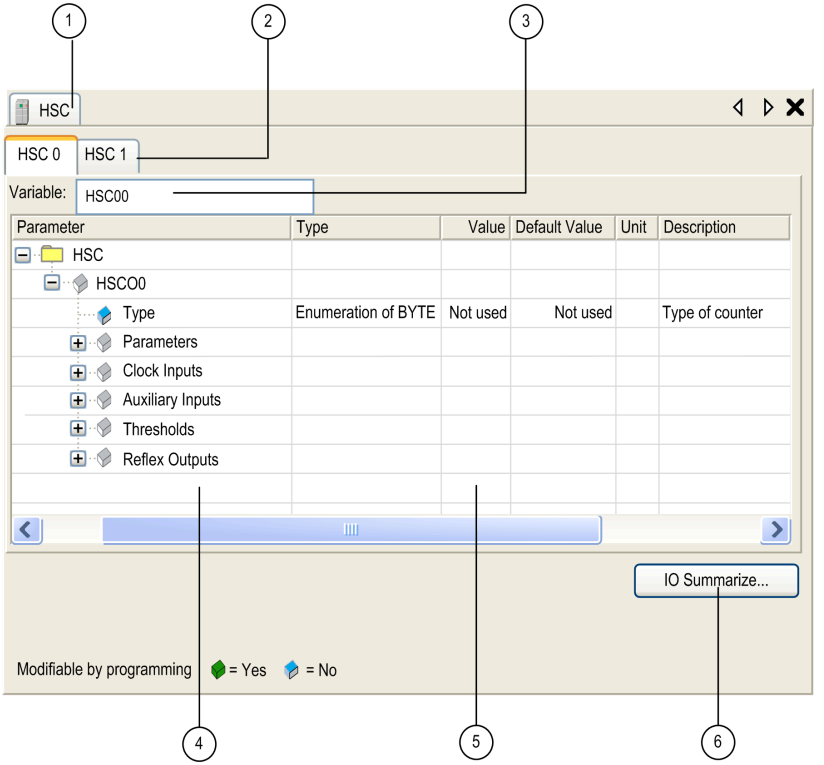
Accessing the HSC Configuration Window

Follow these steps to access the embedded HSC configuration window:

Step	Description
1	In the **Devices tree**, double-click **HMISCUxx5 →Embedded Functions →HSC**. **Result**: The **HSC Configuration** window is displayed.

HSC Configuration Window

The figure shows a sample HSC configuration window used to configure the HSC:



The table describes the areas of the HSC configuration window:

Number	Action
1	If necessary, select the **HSC** tab to access the HSC configuration Windows.
2	Select a specific **HSC** • tab to access the HSC channel you need to configure.
3	Choose the type of HSC (**Simple** or **Main**) you want. The global variable name representing the channel instance can be defined here. Default for HSC 0 is **HSC00**, and for HSC 1 is **HSC01**.
4	Expand each parameter by clicking the plus sign next to it to access its settings.
5	Configuration window where the HSC parameters are set depending on the mode used.
6	When you click the **IO Summarize** button, the **IO Summary** window appears. It allows you to check your configured physical I/O mapping.

For detailed information on configuration parameters, refer to HMI SCU HSC choice matrix *(see page 21)*.

HSC I/O Mapping

HSC I/O Mapping

The table shows the availability of the simple type HSC functions according to the inputs:

Function		HSC	
Type		Simple	
Channel		0	1
Fast Input	FI0	A	–
	FI1	–	A
A Input counting signal			

NOTE: You cannot configure both a **Simple** and **Main** HSC.

The table shows the availability of the main type HSC functions according to the inputs and outputs:

Function		HSC
Type		Main
Channel		0
Fast Input	FI0	A
	FI1	B(1)(2)
Regular Input	DI0	Sync(2)
	DI1	Cap(2)
Fast Output	FQ0	HSC0 reflex Output0(2)
	FQ1	HSC0 reflex Output1(2)
A Input counting signal **B** Input counting signal (optionally used depending on configuration of HSC Mode) **Sync** Reset and start counting **Cap** Capture current counter value		

(1) A and B input signal usage depends on the configuration of Main HSC mode *(see page 20)*.

(2) Optional according to the configuration of Main HSC mode.

Simple Type Overview

Overview

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

A Simple type HSC can count-up/count-down to/from a predefined value.

You can program the actions when the count is reached. These actions are done in the context of the programmed task.

Simple Type Modes

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

One-shot *(see page 27)*: 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 45)*: 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 for the **Simple** type is 100 kHz.

NOTE:

The maximum counting frequency depends on the filter setting:

- 4 µs filter: 100 kHz
- 40 µs filter: 14.5 kHz

Main Type Overview

Overview

The **Main** type is a counter that uses up to 2 fast inputs, 2 regular inputs, and 2 fast outputs.

Main Type Modes

The **Main** type supports the following counting modes on single (1 input) or dual-phase (2 inputs) 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 a 0.

Modulo-loop *(see page 53)*: 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 and then presets to the modulo value and restarts counting. You can also use **Modulo-loop** mode with an encoder.

Free-large *(see page 61)*: In this mode, the counter behaves like a high range up and down counter and can be used with an encoder.

Event Counting *(see page 73)*: In this mode, the counter accumulates a number of events that are received during a user-configured time base.

Frequency meter *(see page 83)*: In this mode, the counter measures the frequency of events during a user-configured time base. Frequency is the number of events per second (Hz).

Optional Features

Optional features can be configured depending on the selected mode:

- hardware inputs to operate the counter (enable, sync) or capture the current counting value
- up to 2 thresholds
- up to 2 reflex outputs

Performance

The maximum frequency with a **Main** type is 50 kHz.

NOTE:

The maximum counting frequency depends on the filter setting:

- 4 µs filter: 50 kHz
- 40 µs filter: 14.5 kHz

Choosing your HSC

Overview

This section provides an overview of all the HSC and their functions to help you choose the appropriate HSC for your system.

HSC Choice Matrix

The table provides an overview of all the HSC available with their specifications according to the mode requested:

Mode	Feature	Simple Type	Main Type
One-shot	Counting mode	Count down	Count down
	Maximum rated counting frequency	100 kHz	50 kHz
	Enable with an HSC physical input	No	Yes
	Synchronization / Preset with an HSC physical input	No	Yes
	Compare function	No	Yes, 2 thresholds, 2 reflex outputs, and 2 external event triggers
	Capture function	No	Yes, 1 capture register
Modulo-loop	Counting mode	Count down	Count up Normal Quadrature (X2 and X4) Reverse Quadrature (X2 and X4)
	Maximum counting frequency	100 kHz	50 kHz
	Enable with an HSC physical input	No	Yes, exclusive with second counting input
	Synchronization / Modulo with an HSC physical input	No	Yes
	Compare function	No	Yes, 2 thresholds, 2 reflex outputs, and 2 external event triggers
	Capture function	No	Yes, 1 capture register

Mode	Feature	Simple Type	Main Type
Free-large	Counting mode	–	Normal Quadrature (X2 and X4) Reverse Quadrature (X2 and X4)
	Maximum counting frequency	–	50 kHz
	Enable with an HSC physical input	–	No
	Synchronization / Preset with an HSC physical input	–	Yes
	Compare function	–	Yes, 2 thresholds, 2 reflex outputs, and 2 external event triggers
	Capture function	–	Yes, 1 capture register
Event Counting	Counting mode	–	Single phase pulse counting during user-defined time base.
	Maximum counting frequency	–	50 kHz
	Enable with an HSC physical input	–	No
	Synchronization / Preset with an HSC physical input	–	Yes
	Compare function	–	No
	Capture function	–	No
Frequency Meter	Counting mode	–	Single phase pulse counting during user-defined time base
	Maximum counting frequency	–	50 kHz
	Enable with an HSC physical input	–	No

Part II
HSC Modes

Overview

This part describes the use the modes of a HSC.

What Is in This Part?

This part contains the following chapters:

Chapter 2
One-shot Mode Principle

One-shot Mode Principle Description

Overview

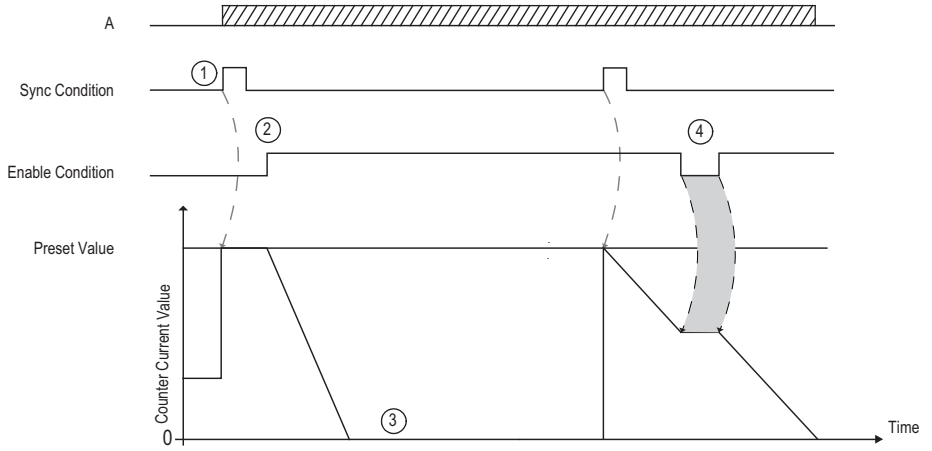
The counter value is set by a synchronization edge, which loads the configured preset value.

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 value is set.
2	When Enable condition = `TRUE`, 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. **Note:** At this point, pulses on input A have no effect on the counter.
4	When Enable condition = `FALSE`, the counter ignores the pulses from input A and retains its current value until the Enable condition again = `TRUE`. 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 107)* and Synchronization *(see page 106)* function.

Chapter 3
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.

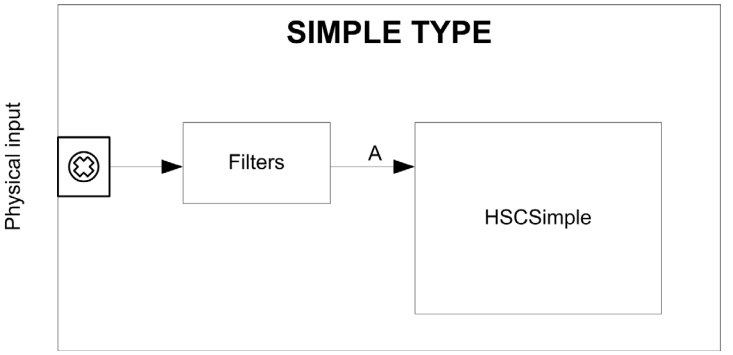
What Is in This Chapter?

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 up.

Configuration of the Simple Type in One-shot Mode

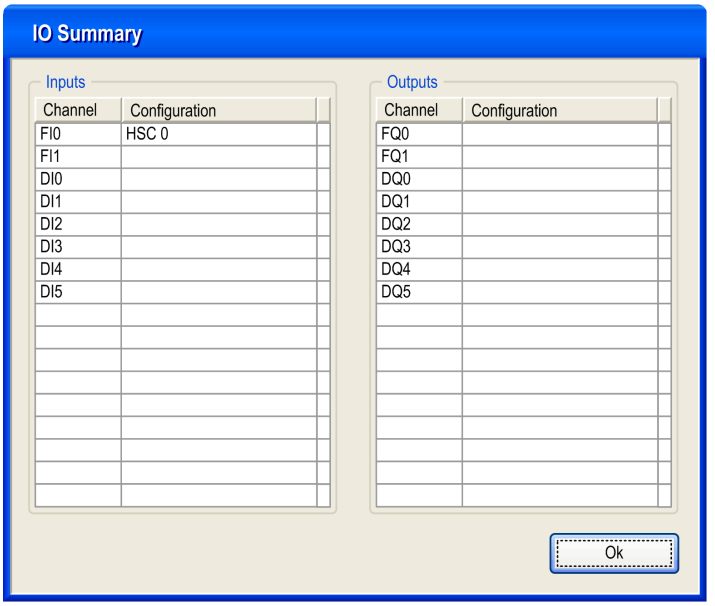
Configuration Procedure

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Select a **HSC** • tab.
3	Set the value of **HSC** • → **Type** to **Simple**.
4	The instance of the **Simple** type is created, you can rename it from the **Variable** field.
5	If necessary, set the **HSC** • → **Parameters** → **Mode** to **One-shot**.
6	Set the preset value for **Parameters** → **Preset/Modulo**. In **One-shot** mode, this field represents the initial **Modulo Value** parameter.
7	Set the anti-bounce filter value of the **HSC** • → **Clock Inputs** → **A Filter** parameter.

IO Summary

Click the **IO Summarize...** button to display the input and output assignments.



NOTE: Any physical I/O conflicts (for example, the same input or output pin shared by two different functions) will be highlighted in red in the IO Summary.

Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Simple** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Simple Type

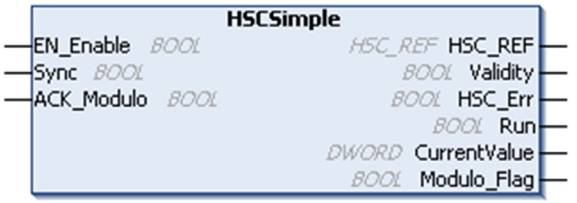
Overview

A **Simple** type is always managed by an HSCSimple *(see page 19)* function block.

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

Adding a HSCSimple Function Block

Step	Description
1	Drag the **Libraries** → **Controller** → **HMISCU** → **HMISCU_HSC** → **HSCSimple** FB to the **Application tree** → **HMISCUxx5** →**POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** → **Embedded Functions** → **HSC** → **HSC0•** with the **HSC0•** → **Type** that is set to **Simple**.
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The table describes the input variables:

Input	Type	Comment
`EN_Enable`	`BOOL`	`TRUE` = authorizes changes to the current counter value.
`Sync`	`BOOL`	On rising edge, sets the counter value with the configured preset
`ACK_Modulo`	`BOOL`	Not used

The table describes the output variables:

Output	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that the output values on the function block are valid.
HSC_Err	BOOL	TRUE = indicates that an error was detected. Use the HSCGetDiag *(see page 124)* function block to get more information about this detected error.
Run	BOOL	TRUE = counter is running. Switches to FALSE when CurrentValue reaches 0. A rising edge on Sync is needed to restart the counter.
CurrentValue	DWORD	Current count value of the counter.
Modulo_Flag	BOOL	Not used

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the HSCGetParam *(see page 126)* or HSCSetParam *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the HSC_PARAMETER_TYPE *(see page 117)* that can be read or modified while the program is running:

Parameter	Description
HSC_PRESET	To get or set the Preset value of the HSC.

Chapter 4
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.

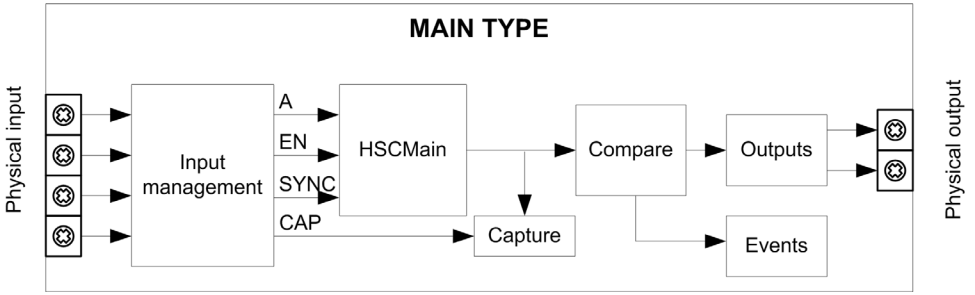
What Is in This Chapter?

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 95)*
- Capture function *(see page 101)*
- Synchronization function *(see page 106)*
- Enable function *(see page 107)*

Configuration of the Main Type in One-shot Mode

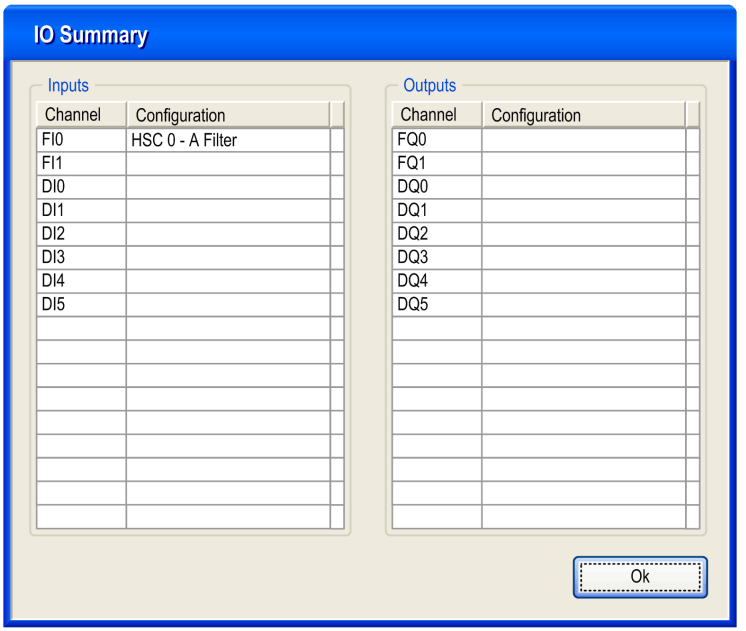
Configuration Procedure

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Set the type to **Main** from the **HSC0•** → **Type** drop down menu.
3	The instance of the **Main** type is created, you can rename it from the **Variable** field.
4	If necessary, set the mode to **One-shot** from the **HSC0•** → **Parameters** → **Mode** drop down menu.
5	Set the preset value from **Parameters** → **Preset/Modulo** In **One-shot** mode, this field represents the initial Modulo Value.
6	Set the anti-bounce filtering value from the **HSC0•** → **Clock Inputs** → **A Filter** drop down menu.
7	Optionally, enable the **SYNC**, **EN** and **CAP** auxiliary inputs from the **HSC0•** →**Auxiliary Inputs** drop down menu to enable the Synchronization function *(see page 106)*, Enable function *(see page 107)* and Capture function *(see page 101)* on a physical input.
8	Optionally, enable the thresholds from the drop down menu, by selecting **HSC0•** → **Thresholds** → **Threshold 0** → **Enable/Disabled** to authorize the Compare function and to configure the Reflex Outputs *(see page 95)*. Threshold 1 can also be Enabled after Threshold 0 is Enabled. **NOTE:** For the **One-shot** mode, configured values must follow this rule: 0 < Threshold 0 Value < Threshold 1 Value < (Preset - 1)

IO Summary

Click the **IO Summarize...** button to display the input and output assignments.



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Main** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Main Type

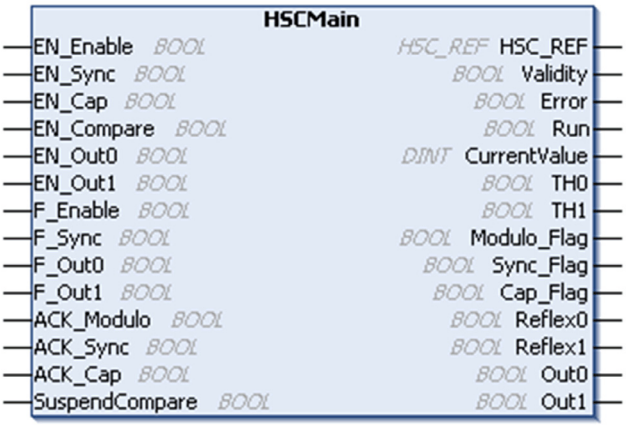
Overview

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

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

Adding the HSCMain Function Block

Step	Description
1	Drag the **Libraries** →**Controller** →**HMISCU** →**HMISCU_HSC** →**HSCMain** FB to the **Application tree** →**HMISCUxx5** →**POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** →**Embedded Functions** →**HSC** →**HSC0•** with the **HSC0•** →**Type** that is set to **Main**. Using the input assistant, the HSC instance can be selected at the following path: **Embedded Functions** → **HSC**
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The 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 107)*.
EN_Sync	BOOL	When **SYNC** input is configured: if TRUE, authorizes the counter synchronization and start via the Sync input *(see page 105)*.
EN_Cap	BOOL	When **CAP** input is configured: if TRUE, enables the Capture input *(see page 101)*.
EN_Compare	BOOL	TRUE = enables the comparator operation *(see page 95)* (using Thresholds 0, 1): • basic comparison (TH0, TH1 output bits) • reflex (Reflex0, Reflex1 output bits) • events (to trigger external tasks on threshold crossing)
EN_Out0	BOOL	TRUE = enables physical output Output0 to echo the Reflex0 value (if configured).
EN_Out1	BOOL	TRUE = enables physical output Output1 to echo the Reflex1 value (if configured).
F_Enable	BOOL	Forces the Enable condition *(see page 107)*.
F_Sync	BOOL	Forces the Sync condition *(see page 106)*
F_Out0	BOOL	TRUE = forces Output0 to TRUE (if Reflex0 is configured).
F_Out1	BOOL	TRUE = forces Output1 to TRUE (if Reflex1 is configured).
ACK_Modulo	BOOL	On rising edge, resets Modulo-Flag.
ACK_Sync	BOOL	On rising edge, resets Sync_Flag.
ACK_Cap	BOOL	On rising edge, resets Cap_Flag.
SuspendCompare	BOOL	TRUE = compare results are suspended: • Physical Outputs FQ0 and FQ1 maintain their last value. • Events are masked. **NOTE:** EN_Compare, EN_Out0, EN_Out1, F_Out0, F_Out1 remain operational while SuspendCompare is set.

The table describes the output variables:

Output	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that output values on the function block are valid.
Error	BOOL	TRUE = indicates that an error was detected. Use the HSCGetDiag *(see page 124)* function block to get more information about this detected error.
CurrentValue	DINT	Current count value of the counter.
Run	BOOL	TRUE = counter is running. Switches to 0 when CurrentValue reaches 0. A rising edge on Sync is needed to restart the counter.
TH0	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 0. [Counting Down] FALSE when CurrentValue ≤ Threshold 0.
TH1	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 1. [Counting Down] FALSE when CurrentValue ≤ Threshold 1.
Modulo_Flag	BOOL	Set to TRUE when counter reaches 0.
Sync_Flag	BOOL	Set to TRUE by the synchronization of the counter *(see page 106)*.
Cap_Flag	BOOL	Set to TRUE when a new capture value is stored in the Capture register *(see page 101)*. This flag must be reset before a new capture can occur.
Reflex0	BOOL	State of Reflex0. *(see page 95)*
Reflex1	BOOL	State of Reflex1. *(see page 95)*
Out0	BOOL	State of physical output Output0 (if Reflex0 configured).
Out1	BOOL	State of physical output Output1 (if Reflex1 configured).

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the `HSCGetParam` *(see page 126)* or `HSCSetParam` *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the `HSC_PARAMETER_TYPE` *(see page 117)* which can be read or modified while the program is running:

Parameter	Description
`HSC_PRESET`	To get or set the Preset value of an HSC.
`HSC_THRESHOLD0`	To get or set the Threshold 0 value of an HSC.
`HSC_THRESHOLD1`	To get or set the Threshold 1 value of an HSC.

NOTE: For the **One-shot** mode, configured values must follow the rule:

0 < Threshold 0 Value < Threshold 1 Value < (Preset - 1)

For example:

If the current configured values are:
- **Threshold 0 Value** = 100
- **Threshold 1 Value** = 200
- **Preset** = 300

And if the desired configuration values are:
- **Threshold 0 Value** = 50
- **Threshold 1 Value** = 120
- **Preset** = 150

Set the value of **Threshold 1** before **Preset**.

If the **Preset** is set to 150 first, **HSCSetParam** returns a parameter error because the desired **Preset** (150) is less than the current **Threshold 1 Value** (200).

Chapter 5
Modulo-loop Principle

Modulo-loop Mode Principle Description

Overview

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

Principle

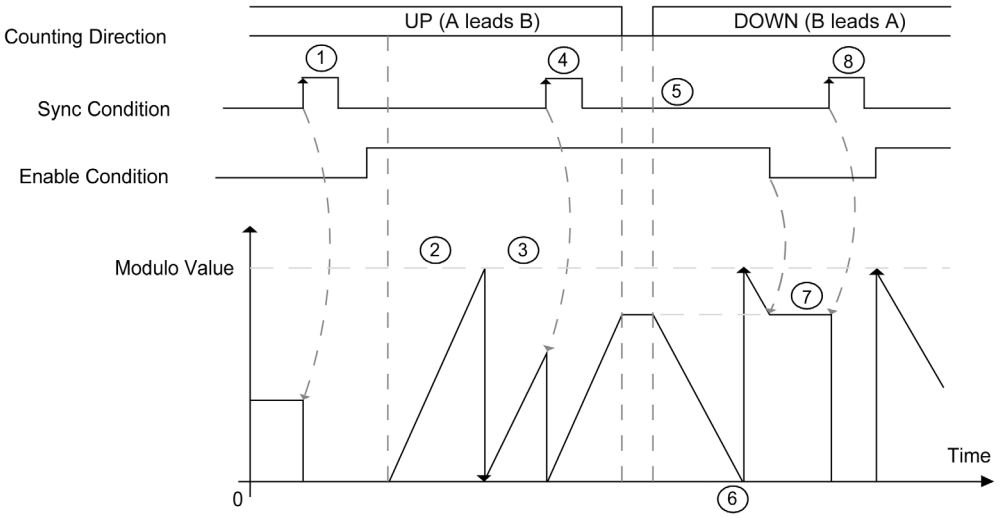
On a rising edge of the Sync condition *(see page 106)*, the counter is activated and the current value is reset to 0.

When counting is enabled *(see page 107)*:

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 `TRUE`, 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 `TRUE`, and the counting continues.

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	As long as Enable condition = `TRUE`, each pulse on A (for single phase) or each pulse pair with leading edge on signal A (for normal quadrature) 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 `TRUE`.
4	On the rising edge of Sync condition, the current counter value is reset to 0.
5	As long as Enable condition = `TRUE`, each pulse pair with a leading edge from signal B (for normal quadrature) decrements the counter.
6	When the counter reaches 0, the counter loops to (modulo-1) at the next pulse pair and the counting continues.
7	When Enable condition = `FALSE`, 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 107)* and Synchronization *(see page 106)* function.

Chapter 6
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.

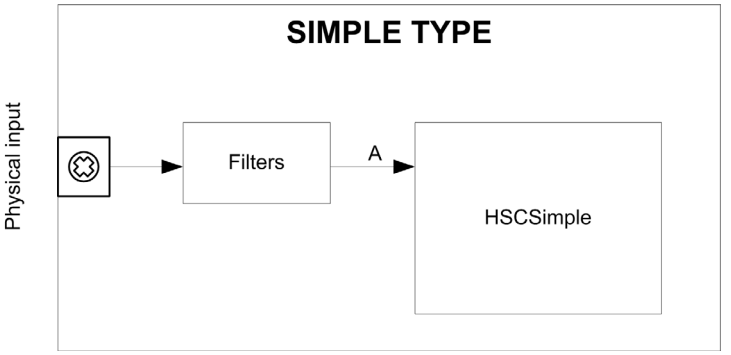
What Is in This Chapter?

This chapter contains the following topics:

Synopsis Diagram

Synopsis Diagram

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



A is the counting input of the `High Speed Counter`.

A **Simple** type can only count up. **Simple** type counting for **Modulo-loop** mode only counts down. **Simple** type counting for **One-shot** mode only counts up.

Configuration of the Simple Type in Modulo-loop Mode

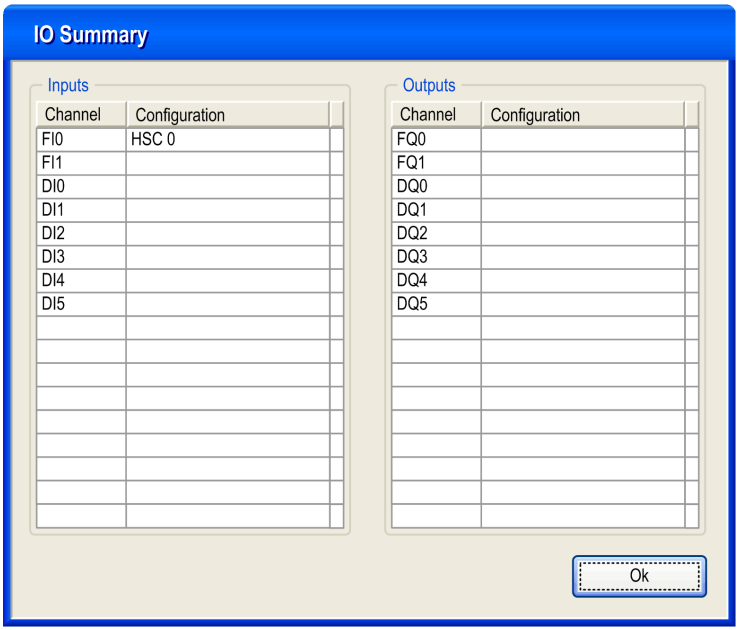
Configuration Procedure

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Set the type to **Simple** from the **HSC0•** → **Type** drop down menu.
3	The instance of the **Simple** type is created, you can rename it from the **Variable** field.
4	Set the mode to **Modulo-loop** from the **HSC0•** → **Parameters** → **Mode** drop down menu.
5	Set the modulo value from **Parameters** → **Preset/Modulo**.
6	Set the anti-bounce filtering value from the **HSC0•** → **Clock Inputs** → **A Filter** drop down menu.

IO Summary

Click the **IO Summary...** button to display the input and output assignments:



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Simple** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Simple Type

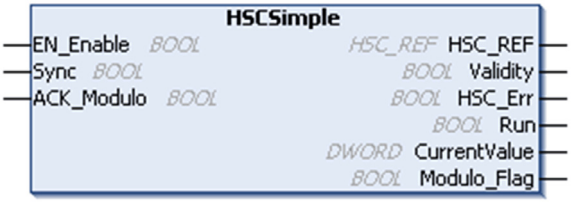
Overview

A **Simple** type is always managed by an HSCSimple *(see page 19)* function block.

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

Adding a HSCSimple Function Block

Step	Description
1	Drag the **Libraries** → **Controller** → **HMISCU** → **HMISCU_HSC** → **HSCSimple** FB to the **Application tree** → **HMISCUxx5** → **POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** → **Embedded Functions** → **HSC** → **HSC0•** with the **HSC0•** → **Type** that is set to **Simple**.
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The table describes the input variables:

Input	Type	Comment
EN_Enable	BOOL	TRUE = authorizes changes to the current counter value.
Sync	BOOL	On rising edge, sets the counter value to 0.
ACK_Modulo	BOOL	On rising edge, resets Modulo_Flag.

The table describes the output variables:

Output	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that the output values on the function block are valid.
HSC_Err	BOOL	TRUE = indicates that an error was detected. HSCGetDiag *(see page 124)* function block may be used to get more information about this detected error.
Run	BOOL	TRUE = indicates counter is running.
CurrentValue	DWORD	Current count value of the counter.
Modulo_Flag	BOOL	Set to TRUE when the counter value rolls over the Modulo Value when counting up, or rolls over 0 when counting down.

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the HSCGetParam *(see page 126)* or HSCSetParam *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the HSC_PARAMETER_TYPE *(see page 117)* that can be read or modified while the program is running:

Parameter	Description
HSC_MODULO	To get or set the modulo value of an HSC.

Chapter 7
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.

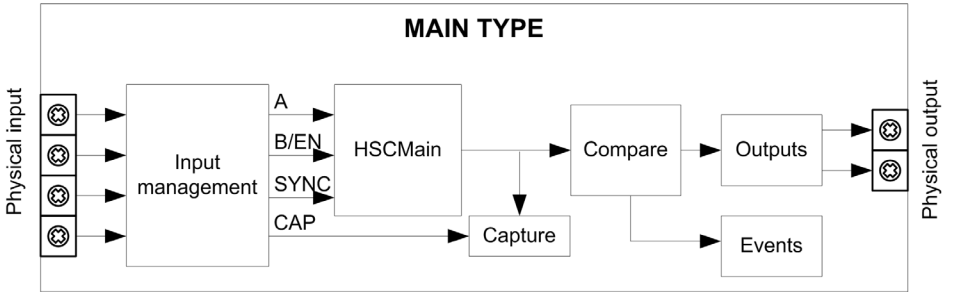
What Is in This Chapter?

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

- Comparison function *(see page 95)*
- Capture function *(see page 101)*
- Synchronization function *(see page 106)*
- Enable function *(see page 107)*

Configuration of the Main Type in Modulo-loop Mode

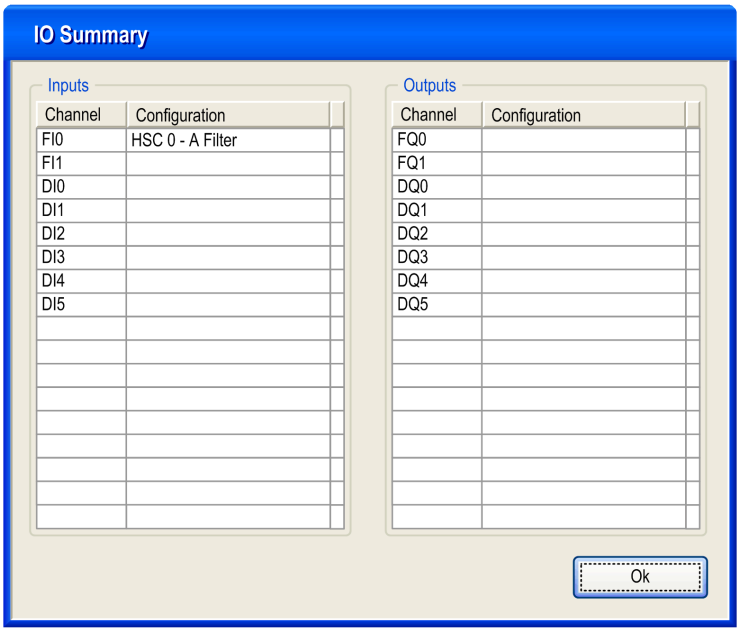
Configuration Procedure

Follow this procedure to configure a **Main** type:

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Set the type to **Main** from the **HSC0•** → **Type** drop down menu.
3	The instance of the **Main** type is created, you can rename it from the **Variable** field.
4	Set the mode to **Modulo-loop** from the **HSC0•** → **Parameters** → **Mode** drop down menu.
5	Set the modulo value for **Parameters** → **Preset/Modulo**
6	Select an input mode value from the **HSC0•** →**Clock Inputs** →**Input mode** drop down menu. This enables the **A Filter** (and **B Filter**, depending on the **Input mode** used).
7	Set the anti-bounce filtering value from the **Clock Inputs** → **A Filter** (and **B Filter**, when applicable) drop down menu.
8	Optionally, enable the **SYNC**, **EN** (only if input mode = **Single Phase**) and **CAP** auxiliary inputs from the **HSC0•** → **Auxiliary Inputs** → **SYNC or EN or CAP** drop down menus, to enable the Synchronization function *(see page 106)*, Enable function *(see page 107)* and Capture function *(see page 102)* on a physical input.
9	Optionally, enable the thresholds from the drop down menu, by selecting **HSC0•** → **Thresholds** → **Threshold 0** → **Enable/Disabled** to authorize the Compare function and to configure the Reflex Outputs *(see page 95)*. **NOTE:** For the **Modulo-Loop** mode, configured values must follow the rule: 0 < Threshold 0 Value < Threshold 1 Value < (Modulo - 1)

IO Summary

Click the **IO Summarize...** button to display the input and output assignments.



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Main** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A, B	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Main Type

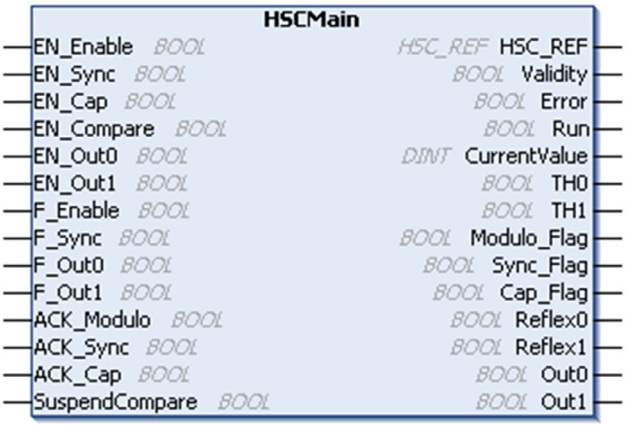
Overview

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

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

Adding the HSCMain Function Block

Step	Description
1	Drag the **Libraries** → **Controller** → **HMISCU** → **HMISCU_HSC** → **HSCMain** FB to the **Application tree** → **HMISCUxx5** → **POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** →**Embedded Functions** → **HSC** → **HSC0•** with the **HSC0•** → **Type** that is set to **Main**. Using the input assistant, the HSC instance can be selected at the following path: **Embedded Functions** → **HSC**
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The 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 107)*.
EN_Sync	BOOL	When SYNC input is configured: if TRUE, authorizes the counter synchronization and start via the Sync input *(see page 106)*.
EN_Cap	BOOL	When CAP input is configured: if TRUE, enables the Capture input *(see page 102)*.
EN_Compare	BOOL	TRUE = enables the comparator operation *(see page 95)* (using Thresholds 0, 1): • basic comparison (TH0, TH1 output bits) • reflex (Reflex0, Reflex1 output bits) • events (to trigger external tasks on threshold crossing)
EN_Out0	BOOL	TRUE = enables physical output Output0 to echo the Reflex0 value (if configured).
EN_Out1	BOOL	TRUE = enables physical output Output1 to echo the Reflex1 value (if configured).
F_Enable	BOOL	Forces the Enable condition *(see page 107)*. Takes priority over EN_Enable input.
F_Sync	BOOL	Forces the Sync condition *(see page 106)*. Takes priority over the EN_Sync input.
F_Out0	BOOL	TRUE = forces Output0 to TRUE (if Reflex0 is configured). Takes priority over EN_Out0.
F_Out1	BOOL	TRUE = forces Output1 to TRUE (if Reflex1 is configured). Takes priority over EN_Out1.
ACK_Modulo	BOOL	On rising edge, resets Modulo_Flag.
ACK_Sync	BOOL	On rising edge, resets Sync_Flag.
ACK_Cap	BOOL	On rising edge, resets Cap_Flag.
SuspendCompare	BOOL	TRUE = compare results are suspended: • Physical Outputs FQ0 and FQ1 maintain their last value. • Events are masked. **NOTE:** EN_Compare, EN_Out0, EN_Out1, F_Out0, F_Out1 remain operational while SuspendCompare is set.

The table describes the output variables:

Output	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that output values on the function block are valid.
Error	BOOL	TRUE = indicates that an error was detected. Use the HSCGetDiag *(see page 124)* function block used to get more information about this detected error.
Run	BOOL	TRUE = counter is running.
CurrentValue	DINT	Current count value of the counter.
TH0	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 0. [Counting Down] FALSE when CurrentValue ≤ Threshold 0.
TH1	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 1. [Counting Down] FALSE when CurrentValue ≤ Threshold 1.
Modulo_Flag	BOOL	Set to TRUE when the counter value rolls over the Modulo Value when counting up, or rolls over 0 when counting down.
Sync_Flag	BOOL	Set to TRUE by the synchronization of the counter *(see page 106)*.
Cap_Flag	BOOL	Set to TRUE when a new capture value is stored in the Capture register *(see page 102)*. This flag must be reset before a new capture can occur.
Reflex0	BOOL	State of Reflex0 *(see page 95)*.
Reflex1	BOOL	State of Reflex1 *(see page 95)*.
Out0	BOOL	State of physical output Output0 (if Reflex0 configured in SoMachine HSC Embedded Functions, otherwise FALSE if not configured).
Out1	BOOL	State of physical output Output1 (if Reflex1 configured in SoMachine HSC Embedded Functions, otherwise FALSE if not configured).

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the `HSCGetParam` *(see page 126)* or `HSCSetParam` *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the `HSC_PARAMETER_TYPE` *(see page 117)* that can be read or modified while the program is running:

Parameter	Description
`HSC_MODULO`	To get or set the Modulo value of an HSC.
`HSC_THRESHOLD0`	To get or set the Threshold 0 value of an HSC.
`HSC_THRESHOLD1`	To get or set the Threshold 1 value of an HSC.

Chapter 8
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:

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, quadrature is supported.

When counting is enabled *(see page 107)*, the counter counts as follows in:
Incrementing direction: the counter increments.
Decrementing direction: the counter decrements.

With a **Main** type, on the rising edge of the Sync condition *(see page 106)*, the counter is activated and the current value is set to the preset value.

The current counter is stored in the capture register by using the Capture *(see page 101)* function.

If the counter reaches the counting limits, the counter will react according to the Limits Management *(see page 65)* configuration.

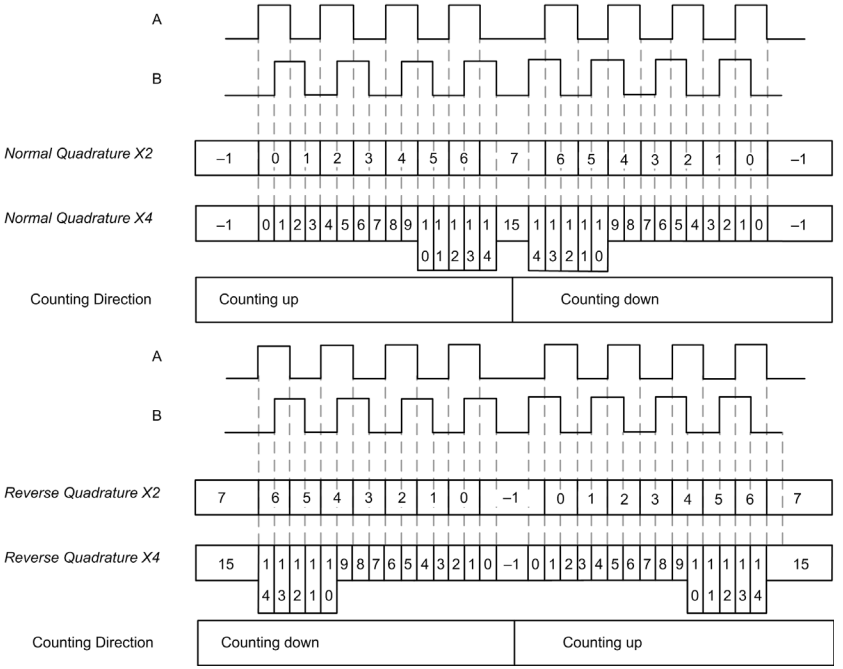
Input Modes

The table shows the 4 types of input modes available:

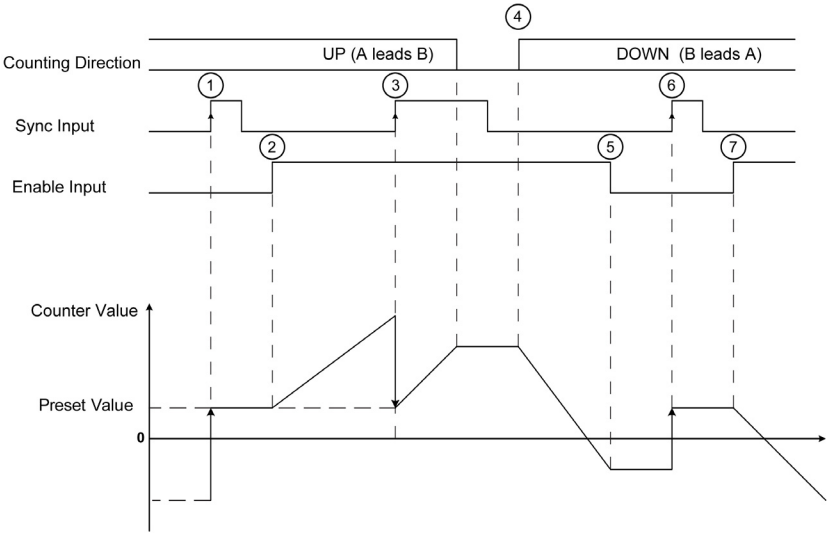
Input Mode	Comment
Normal Quadrature X2	A physical encoder always provides 2 signals 90° shift that first allows the counter to count pulses and detect direction: ● X2: 2 counts by Encoder cycle ● X4: 4 counts by Encoder cycle
Normal Quadrature X4	
Reverse Quadrature X2	
Reverse Quadrature X4	

Quadrature Principle Diagram

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



The figures shows the affect of the inputs on the counter value for Normal Quadrature:



Stage	Action
1	On the rising edge of `Sync` input, the current value is set to the configured preset value.
2	When Enable condition = `TRUE`, each pulse pair with leading edge on A increments the counter value.
3	On the rising edge of Preset condition, the current value is set to the configured preset value.
4	When Enable condition = `TRUE`, each pulse pair with leading edge on B decrements the counter value.
5	When Enable condition = `FALSE`, the all further pulses are ignored.
6	On the rising edge of `Sync` input, the current value is set to the configured preset value.
7	When Enable condition = `TRUE`, the pulse pair with leading edge on B decrements the counter value.

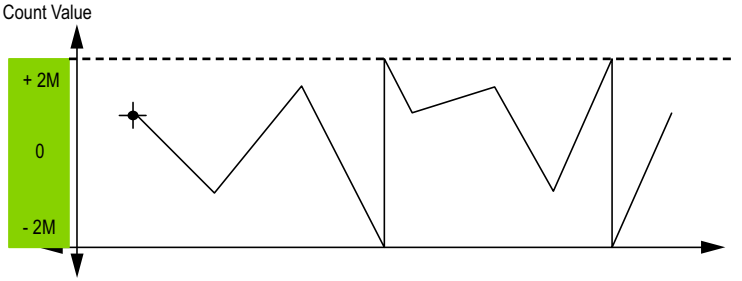
Limits Management

Overview

When the counter limit is reached, the counter behaves as **Rollover**.

Rollover

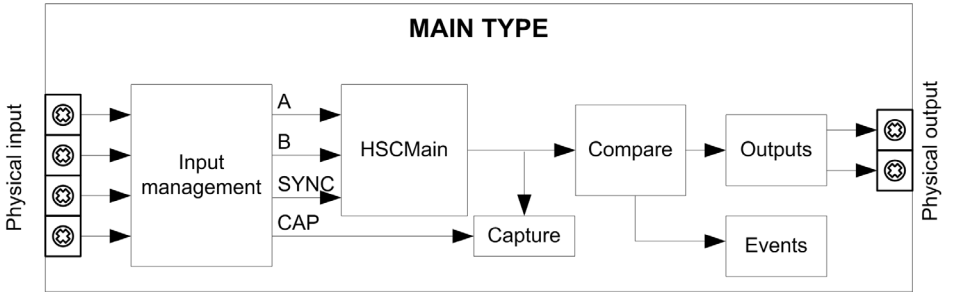
In the case of overflow or underflow of the counter, the current counter value goes automatically to the opposite limit value. `Modulo_Flag` output is set to `TRUE`.



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.

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 function:

- Compare *(see page 95)*
- Capture *(see page 101)*
- Synchronize by a physical input *(see page 106)*

Configuration of the Main Type in Free-Large Mode

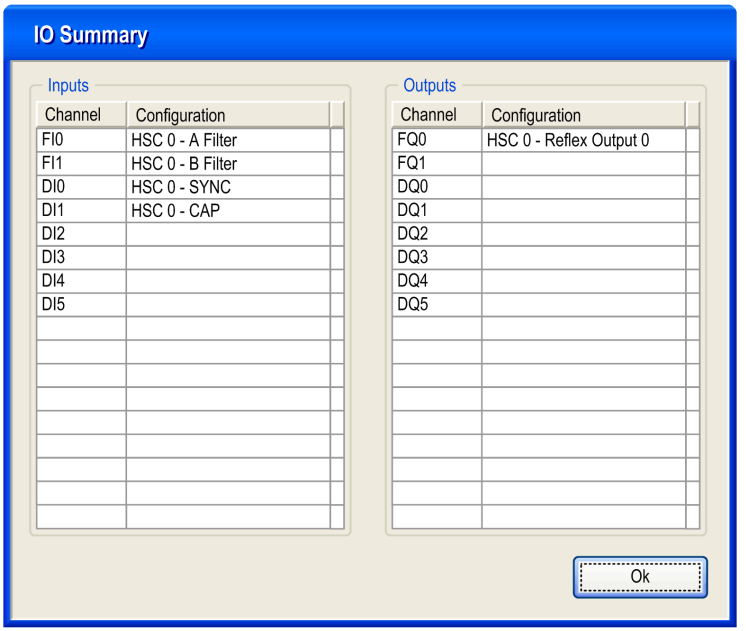
Configuration Procedure

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC** node.
2	Set the type to **Main** from the **HSC0•** → **Type** drop down menu.
3	The instance of the **Main** type is created, you can rename it from the **Variable** field.
4	Set the mode to **Free-large** from the **HSC0•** → **Parameters** → **Mode** drop down menu.
5	Set the preset value from **Parameters** → **Preset/Modulo** For the **Free-Large**, this parameter is the Preset Value.
6	Set the anti-bounce filtering value from the **HSC0•** →**Clock Inputs** →**A Filter** and **B Filter** → drop down menus.
7	Optionally, enable the **SYNC** and **CAP** auxiliary inputs from the **HSC0•** → **Auxiliary Inputs** → **SYNC or CAP** drop down menus, to enable the Synchronization function *(see page 106)*, and Capture function *(see page 101)* on a physical input.
8	Optionally, enable the thresholds from the drop down menu, by selecting **HSC0•** →**Thresholds** → **Threshold 0** to authorize the Compare function and to configure the Reflex Outputs *(see page 95)*. **NOTE:** For the **Free-large** mode, configured values must follow the rule: 0 < Threshold 0 Value < Threshold 1 Value Threshold values are not restricted by the **Preset** value for the **Free-large** mode.

IO Summary

Click the **IO Summarize...** button to display the input and output assignments.



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Main** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A, B	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Main Type

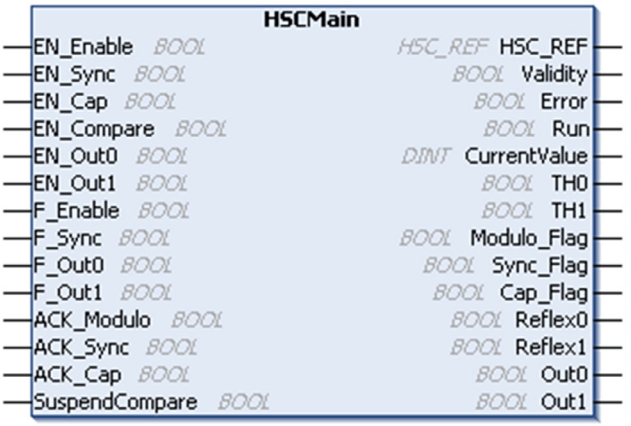
Overview

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

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

Adding the HSCMain Function Block

Step	Description
1	Drag the **Libraries** →**Controller** →**HMISCU** →**HMISCU_HSC** →**HSCMain** FB to the **Application tree** →**HMISCUxx5** →**POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** →**Embedded Functions** →**HSC** →**HSC0•** with the **HSC0•** →**Type** that is set to **Main**. Using the input assistant, the HSC instance can be selected at the following path: **Embedded Functions** → **HSC**
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The table describes the input variables:

Input	Type	Description
EN_Enable	BOOL	Not used
EN_Sync	BOOL	When **SYNC** input is configured: if TRUE, authorizes the counter synchronization and start via the Sync input *(see page 106)*.
EN_Cap	BOOL	When **CAP** input is configured: if TRUE, enables the Capture input *(see page 101)*.
EN_Compare	BOOL	TRUE = enables the comparator operation *(see page 95)* (using Thresholds 0, 1): • basic comparison (TH0, TH1 output bits) • reflex (Reflex0, Reflex1 output bits) • events (to trigger external tasks on threshold crossing)
EN_Out0	BOOL	TRUE = enables physical output Output0 to echo the Reflex0 value (if configured).
EN_Out1	BOOL	TRUE = enables physical output Output1 to echo the Reflex1 value (if configured).
F_Enable	BOOL	Forces the Enable condition *(see page 107)*.
F_Sync	BOOL	Forces the Sync condition *(see page 106)*
F_Out0	BOOL	TRUE = forces Output0 to TRUE (if Reflex0 is configured).
F_Out1	BOOL	TRUE = forces Output1 to TRUE (if Reflex1 is configured).
ACK_Modulo	BOOL	On rising edge, resets Modulo_Flag.
ACK_Sync	BOOL	On rising edge, resets Sync_Flag.
ACK_Cap	BOOL	On rising edge, resets Cap_Flag.
SuspendCompare	BOOL	TRUE = compare results are suspended: • TH0, TH1, Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value. • Physical outputs FQ0 and FQ1 maintain their last value • Events are masked **NOTE:** EN_Compare, EN_Out0, EN_Out1, F_Out0, F_Out1 remain operational while SuspendCompare is set.

The table describes the output variables:

Outputs	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that output values on the function block are valid.
Error	BOOL	TRUE = indicates that an error was detected. Use the HSCGetDiag *(see page 124)* function block used to get more information about this detected error.
CurrentValue	DINT	Current count value of the counter.
Run	BOOL	TRUE = counter is running.
TH0	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 0. [Counting Down] FALSE when CurrentValue ≤ Threshold 0.
TH1	BOOL	[Counting Up] TRUE when CurrentValue ≥ Threshold 1. [Counting Down] FALSE when CurrentValue ≤ Threshold 1.
Modulo_Flag	BOOL	Set to TRUE when the counter rollovers its limits
Sync_Flag	BOOL	Set to TRUE by the synchronization of the counter *(see page 106)*
Cap_Flag	BOOL	Set to TRUE when a new capture value is stored in the Capture register *(see page 101)*. This flag must be reset before a new capture can occur.
Reflex0	BOOL	State of Reflex0. *(see page 95)*
Reflex1	BOOL	State of Reflex1. *(see page 95)*
Out0	BOOL	State of physical outputs Output0 (if Reflex0 configured).
Out1	BOOL	State of physical outputs Output1 (if Reflex1 configured).

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the `HSCGetParam` *(see page 126)* or `HSCSetParam` *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the `HSC_PARAMETER_TYPE` *(see page 117)* which can be read or modified while the program is running:

Parameter	Description
`HSC_PRESET`	To get or set the Preset value of an HSC.
`HSC_THRESHOLD0`	To get or set the Threshold 0 value of an HSC.
`HSC_THRESHOLD1`	To get or set the Threshold 1 value of an HSC.

NOTE: For the **Free-large** mode, configured values must follow the rule:

0 < Threshold 0 Value < Threshold 1 Value

Threshold values are not restricted by the **Preset** value

For example:

If the current configured values are:
- **Threshold 0 Value** = 100
- **Threshold 1 Value** = 200
- **Preset** = 300

And if the desired configuration values are:
- **Threshold 0 Value** = 50
- **Threshold 1 Value** = 120
- **Preset** = 150

Unlike, the **One-shot** and **Modulo-loop** modes, the value of **Threshold 1** does not need to be set before the **Preset**. Even if the **Preset** is set to 150 first, HSCSetParam does NOT return a parameter error because the desired **Preset** (150) is less than the current **Threshold 1** (200).

Chapter 9
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:

Event Counting Mode Principle Description

Overview

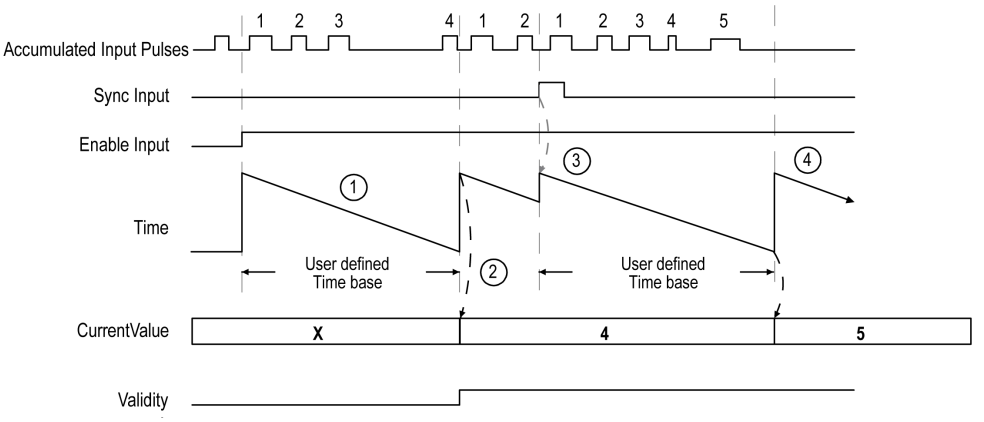
The **Event Counting** mode allows you to count a sequence of events during a given period of time.

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 106)*

Principle Diagram



Stage	Action
1	When Enable condition = `TRUE`, the counter accumulates the number of events (pulses) on the physical input during a predefined period of time. If Validity = 0, the current value is not used.
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 `TRUE`. The counting restarts for a new period of time.
3	On the rising edge of the Sync condition: ● the accumulated input pulse value is reset to 0 ● the current value is not updated ● 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. The counting restarts for a new period of time.

NOTE:

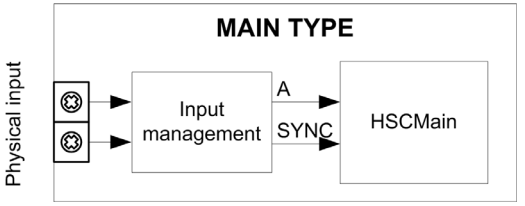
On the **Main** type, when the Enable condition is:

- Set to `FALSE`: the current counting is aborted and `CurrentValue` is maintained to the previous valid value.
- Set to `TRUE`: the accumulated value is reset to 0, the `CurrentValue` remains unchanged, and the counting restarts for a new period of time.

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 Synchronization function *(see page 106)*.

Configuration of the Main Type in Event Counting Mode

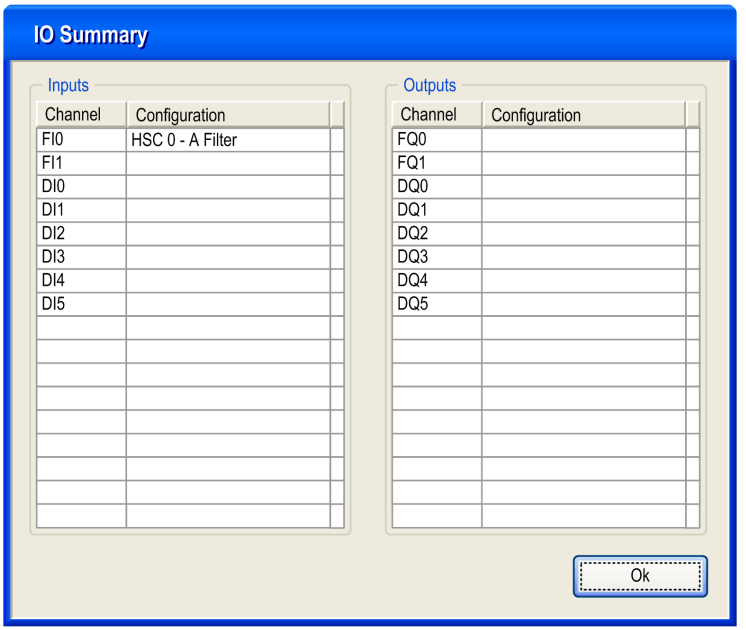
Configuration Procedure

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Select value **Main** in the **HSC0•** → **Type** field.
3	The instance of the **Main** type is created, you can rename it from the **Variable** field.
4	Set the preset condition value in **HSC0•** → **Preset /Modulo**.
5	Set the mode to **Event** from the **HSC0•** →**Parameters** →**Mode** drop down menu for the counting mode.
6	Define the time base from the **Parameters** → **Time** drop down menu, by selecting
7	Set the anti-bounce filtering value from the **HSC0•** → **Clock Inputs** → **A Filter** drop down menu.
8	Optionally, enable the **SYNC** auxiliary inputs from the **HSC0•** →**Auxiliary Inputs** →**SYNC** drop down menu to enable the Synchronization function *(see page 106)* on a physical input. If **SYNC** is enabled, set the **SYNC Filter** and **SYNC Edge** values.

IO Summary

Click the **I/O Summarize...** button to display the input and output assignments.



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Main** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Main Type

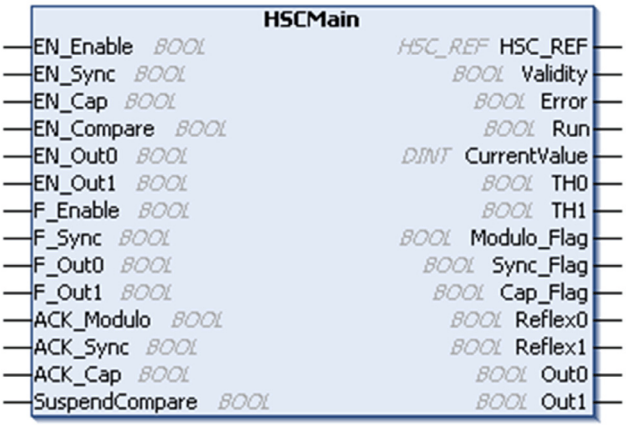
Overview

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

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

Adding the HSCMain Function Block

Step	Description
1	Drag the **Libraries** →**Controller** →**HMISCU** →**HMISCU_HSC** →**HSCMain** FB to the **Application tree** →**HMISCUxx5** →**POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** →**Embedded Functions** →**HSC** →**HSC0•** with the **HSC0•** →**Type** that is set to **Main**. Using the input assistant, the HSC instance can be selected at the following path: **Embedded Functions** → **HSC**
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

These table describes how the different pins of the function block are used in the mode **Event**.

The table describes the input variables:

Input	Type	Description
EN_Enable	BOOL	Not used
EN_Sync	BOOL	When **SYNC** input is configured: if TRUE, allows the setting of the counter value to 0.
EN_Cap	BOOL	Not used
EN_Compare	BOOL	Not used
EN_Out0	BOOL	Not used
EN_Out1	BOOL	Not used
F_Enable	BOOL	Forces the Enable condition *(see page 107)*.
F_Sync	BOOL	Forces the Sync condition *(see page 106)*
F_Out0	BOOL	Not used
F_Out1	BOOL	Not used
ACK_Modulo	BOOL	Not used
ACK_Sync	BOOL	On rising edge, resets Sync_Flag.
ACK_Cap	BOOL	Not used
SuspendCompare	BOOL	Not used

The table describes the output variables:

Outputs	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that output values on the function block are valid.
Error	BOOL	TRUE = indicates that an error was detected. HSCGetDiag *(see page 124)* function block may be used to get more information about this detected error.
CurrentValue	DINT	Current count value of the counter.
Run	BOOL	TRUE = Counter is running.
TH0	BOOL	Not used
TH1	BOOL	Not used
Modulo_Flag	BOOL	Not used
Sync_Flag	BOOL	Set to TRUE by the synchronization of the counter *(see page 106)*

Outputs	Type	Comment
Cap_Flag	BOOL	Not used
Reflex0	BOOL	Not used
Reflex1	BOOL	Not used
Out0	BOOL	Not used
Out1	BOOL	Not used

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the `HSCGetParam` *(see page 126)* or `HSCSetParam` *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the `HSC_PARAMETER_TYPE` *(see page 117)* which can be read or modified while the program is running:

Parameter	Description
`HSC_TIMEBASE`	To get or set the Timebase value of the HSC.

Chapter 10
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:

Description

Overview

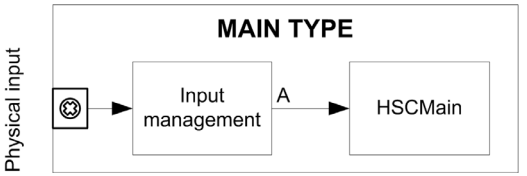
The **Frequency meter** type measures an event frequency in Hz.

The measured frequency is a mean frequency: number of events in a user configured time interval which is then converted to the mean number of events per second (Hz).

Synopsis Diagram

Synopsis Diagram

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



The **Frequency meter** counts the number of pulses on the physical input A over a predefined period of time. The value is stored in the counting register in Hz.

Configuration of the Main Type in Frequency Meter Mode

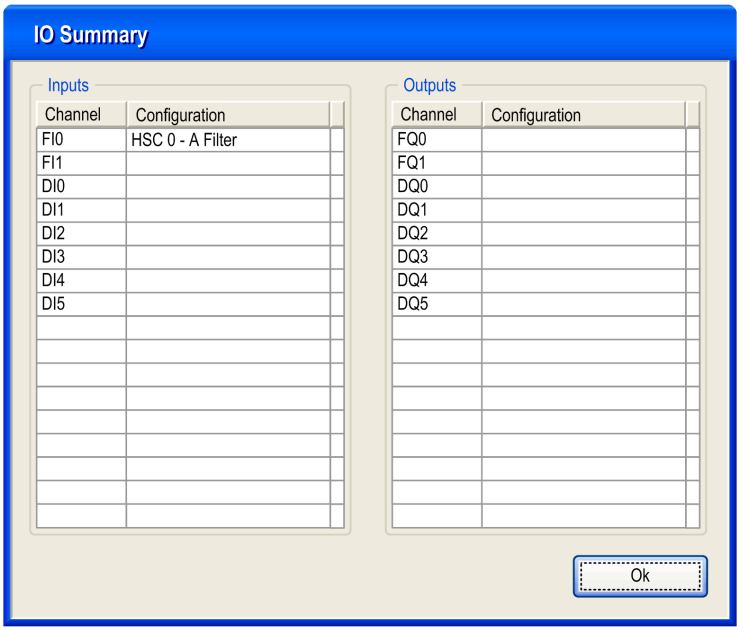
Configuration Procedure

Follow this procedure to configure a **Main** type in **Frequency meter** mode:

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Select value **Main** in the **HSC0•** → **Type** field.
3	The instance of the **Main** type is created, you can rename it from the **Variable** field.
4	Set the mode to **Frequency meter** from the **HSC0•** → **Parameters** → **Mode** drop down menu.
5	Set the time base value from **Parameters** → **Time** drop down menu.
6	Set the anti-bounce filter value from the **HSC0•** → **Clock Inputs** → **A Filter** drop down menu.

IO Summary

Click the **IO Summarize...** button to display the input and output assignments.



Refer to the hardware guide for wiring details *(see Magelis SCU, HMI Controller, Hardware Guide)*.

Programmable Filter

The filtering value on the **Main** type input determines the counter maximum frequency as shown in the table:

Input	Filter value	Maximum counter frequency
A	4 µs	50 kHz
	40 µs	14.5 kHz

Programming the Main Type

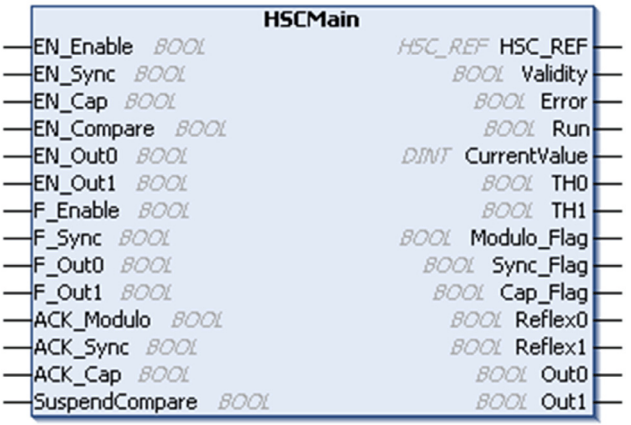
Overview

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

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

Adding a HSCMain Function Block

Step	Description
1	Drag the **Libraries** →**Controller** →**HMISCU** →**HMISCU_HSC** →**HSCMain** FB to the **Application tree** →**HMISCUxx5** →**POU** and drop it on the `Start Here` box in the lower window.
2	The instance name is located in the **Variable** field at the **Device tree** → **HMISCU••5** →**Embedded Functions** →**HSC** →**HSC0•** with the **HSC0•** →**Type** that is set to **Main**. Using the input assistant, the HSC instance can be selected at the following path: **Embedded Functions** → **HSC**
NOTE: This method is for ST, LD, or FBD languages.	



I/O Variables Usage

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

The table describes the input variables:

Input	Type	Description
EN_Enable	BOOL	Not used
EN_Sync	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	Forces the Enable condition *(see page 107)*.
F_Sync	BOOL	Forces the Sync condition *(see page 106)*
F_Out0	BOOL	Not used
F_Out1	BOOL	Not used
ACK_Modulo	BOOL	Not used
ACK_Sync	BOOL	On rising edge, resets Sync_Flag.
ACK_Cap	BOOL	Not used
SuspendCompare	BOOL	Not used

The table describes the output variables:

Outputs	Type	Comment
HSC_REF	HSC_REF *(see page 118)*	Reference to the HSC. To be used with the HSC_REF_IN input pin of the function blocks.
Validity	BOOL	TRUE = indicates that output values on the function block are valid.
Error	BOOL	TRUE = indicates that an error was detected. Use the HSCGetDiag *(see page 124)* function block to get more information about this detected error.
CurrentValue	DINT	Current count value of the counter.
Run	BOOL	TRUE = counter is running.
TH0	BOOL	Not used
TH1	BOOL	Not used
Modulo_Flag	BOOL	Not used
Sync_Flag	BOOL	Set to TRUE by the synchronization of the counter
Cap_Flag	BOOL	Not used

Outputs	Type	Comment
Reflex0	BOOL	Not used
Reflex1	BOOL	Not used
Out0	BOOL	Not used
Out1	BOOL	Not used

Adjusting Parameters

Overview

The list of parameters described in the table can be read or modified by using the HSCGetParam *(see page 126)* or HSCSetParam *(see page 128)* function blocks.

NOTE: Parameters set via the program override the parameters values configured in the HSC configuration. Initial configuration parameters are restored on cold or warm start.

Adjustable Parameters

This table provides the list of parameters from the `HSC_PARAMETER_TYPE` *(see page 117)* which can be modified while the program is running:

Parameter	Description
`HSC_TIMEBASE`	To get or set the Time value of the HSC.

Part III
Optional Functions

Overview

This part provides information on optional functions for HSC.

What Is in This Part?

This part contains the following chapters:

Chapter 11
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 53)*
- Free-Large *(see page 61)*

Comparison is configured in the Configuration screen *(see page 99)* by activating at least one threshold.

Comparison can be used to trigger:

- programming action on thresholds *(see page 97)*
- an event on threshold associated with an external task *(see page 97)*
- reflex outputs *(see page 97)*

Principle of a Comparison

The **Main** type can manage up to 2 thresholds.

A threshold is a configured value that is compared to the current counting value. Thresholds are used to define up to 3 zones or to react to a value crossing.

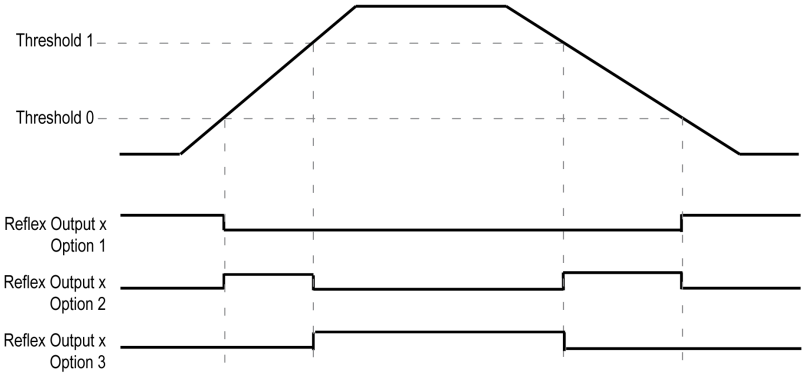
They are defined by configuration and can also be adjusted in the application program by using the HSCSetParam *(see page 128)* function block.

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

- Option 1:
  Counting Up – Reflex Output x is `TRUE` when value < TH0 (reset when value = TH0).
  Counting Down – Reflex Output x is `TRUE` when value ≤TH0 (set when value = TH0).
- Option 2:
  Counting Up – Reflex Output x is `TRUE` when TH0 ≤ value < TH1 (set when value= TH0 and reset when value = TH1).
  Counting Down – Reflex Output x is `TRUE` when TH0 < value ≤ TH1 (set when value = TH1 and reset when value = TH0).
- Option 3:
  Counting Up – Reflex Output x is `TRUE` when value ≥TH1 (set when value = TH1).
  Counting Down – Reflex Output x is `TRUE` when value > TH1 (reset when value =TH1).

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

This diagram shows the state of the Reflex Output (fast digital output) for each individual option:



Threshold Behavior

`TH0` and `TH1` are managed by the task and are updated at the rate of the task cycle time.

Configuring Event

Configuring an event on threshold crossing allows to trigger an external task *(see page 100)*. You can choose to trigger an event when a configured threshold is crossed downward, upward, or both ways.

When the HSC is counting:

- up, the configured **External Event Task** is triggered when the counting value = Threshold value + 1
- down, the configured **External Event Task** is triggered when the counting value = Threshold value - 1

If overflow or underflow, no **External Event Task** is triggered.

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.

NOTE: The state of the reflex outputs depends on the configuration *(see page 99)*.

Changing the Threshold Values

The TH0, TH1, Reflex0, Reflex1, Out0 and Out1 as well as physical outputs will operate with respect to the threshold values, even when the threshold values are dynamically changed as long as SuspendCompare= TRUE.

Therefore, care must be exercised when threshold compares are active to avoid unintended or unexpected results from the physical reflex outputs and HSCMain function block outputs. If the compare function is disabled, threshold values can be modified without worry of unintended outputs. However, if the compare function is enabled, you must, at least, suspend 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 = 1.
- Ensure that TH0 is less than TH1 before reactivating the threshold compare function.

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

While EN_Compare = TRUE, the comparison is active, and it is necessary to follow this procedure:

Step	Action
1	Set SuspendCompare to TRUE. The comparison is frozen at the current value: ● TH0, TH1, Reflex0, Reflex1, Out0, Out1 output bits of the block maintain their last value. ● Physical Outputs 0, 1 maintain their last value **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 HSCSetParam *(see page 128)* function block. **NOTE:** Follow these rules to configure the threshold values: ● For the **One-shot** mode: 0 < Threshold 0 Value < Threshold 1 Value < (Preset - 1) ● For the **Modulo-Loop** mode: 0 < Threshold 0 Value < Threshold 1 Value < (Modulo - 1) ● For the **Free-large** mode: 0 < Threshold 0 Value < Threshold 1 Value The threshold values are not restricted by the **Preset** value for **Free-large** mode.
3	Set SuspendCompare to FALSE. The new Threshold values are applied and the comparison is resumed.

Configuration of the Comparison on a Main Type

Configuration Window

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

Step	Action
1	In the **Devices tree**, double-click **Embedded Functions** → **HSC**.
2	Set the mode to **Main** in the **HSC0•** → **Type** → **Value** drop down menu.
3	Enable the **Thresholds** by selecting the **One-shot**, **Modulo-loop**, or **Free-large** value in the **Parameters** → **Mode** → **Value** drop down menu.
4	Enable **Threshhold 0** and **Threshhold 1** in their **Value** drop down menus.
5	Provide the threshold values. Follow these rules to configure the threshold values: • For the **One-shot** mode: 0 < Threshold 0 Value < Threshold 1 Value < (Preset - 1) • For the **Modulo-Loop** mode: 0 < Threshold 0 Value < Threshold 1 Value < (Modulo - 1) • For the **Free-large** mode: 0 < Threshold 0 Value < Threshold 1 Value The threshold values are not restricted by the **Preset** value for **Free-large** mode.
6	Optionally, provide an event condition *(see page 96)*.
7	Optionally, configure the **Reflex Outputs** behavior *(see page 96)*: • `Reflex Output 0`: Counting Up – Reflex Output x is `TRUE` when value < TH0 (reset when value = TH0). Counting Down – Reflex Output x is `TRUE` when value ≤ TH0 (set when value = TH0). • `Reflex Output 0`: Counting Up – Reflex Output x is `TRUE` when TH0 ≤ value < TH1 (set when value= TH0 and reset when value = TH1). Counting Down – Reflex Output x is `TRUE` when TH0 < value ≤ TH1 (set when value = TH1 and reset when value = TH0). • `Reflex Output 0`: Counting Up – Reflex Output x is `TRUE` when value ≥ TH1 (set when value = TH1). Counting Down – Reflex Output x is `TRUE` when value > TH1 (reset when value =TH1). • `Reflex Output 1`: Counting Up – Reflex Output x is `TRUE` when value < TH0 (reset when value = TH0). Counting Down – Reflex Output x is `TRUE` when value ≤ TH0 (set when value = TH0). • `Reflex Output 1`: Counting Up – Reflex Output x is `TRUE` when TH0 ≤ value < TH1 (set when value= TH0 and reset when value = TH1). Counting Down – Reflex Output x is `TRUE` when TH0 < value ≤ TH1 (set when value = TH1 and reset when value = TH0). • `Reflex Output 1`: Counting Up – Reflex Output x is `TRUE` when value ≥TH1 (set when value = TH1). Counting Down – Reflex Output x is `TRUE` when value > TH1 (reset when value =TH1).

External Event Configuration

Procedure

The following procedure describes how to configure an external event (see the *Magelis SCU SoMachine Programming Guide*) to activate a task:

Step	Action
1	Add a task by left clicking the **Task Configuration** node.
2	In the **Program** window, double click the task to associate it to an External Event.
3	In the **Type** drop-down menu, select **External**.
4	Select in the **External Event** drop-down menu the event to associate it to the task.

External Events

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

Event Name	Description
FI0	This task is activated when the input **FI0** signal detected a rising edge, falling edge, or both edges. The type of signal detection can be configured in the **I/O Configuration** tab.
FI1	This task is activated when the input **FI1** signal detected a rising edge, falling edge, or both edges. The type of signal detection can be configured in the **I/O Configuration** tab.
HSC0_TH0	This task is activated when the **Threshold 0 Value** of the HSC0 is crossed. Task activation can be triggered when counting up, counting down, or both. This depends on the user's configuration of HSC.
HSC0_TH1	This task is activated when the **Threshold 1 Value** of the HSC0 is crossed. Task activation can be triggered when counting up, counting down, or both. This depends on the user's configuration of HSC.

Chapter 12
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:

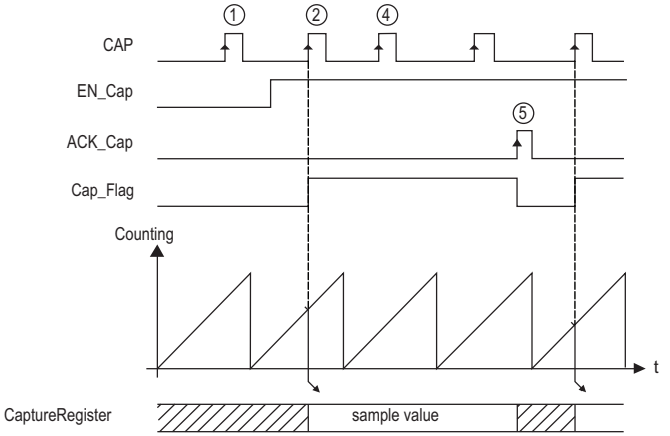
- One-shot *(see page 35)*
- Modulo-loop *(see page 53)*
- Free-large *(see page 61)*

Using this function requires to:

- configure the optional Capture input: **CAP**
- use `HSCGetCapturedValue` *(see page 122)* 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 `HSCGetCapturedValue` *(see page 122)*.
4	While `Cap_Flag` = 1, any new edge on the physical input CAP is ignored.
5	The rising edge of `HSCMain` function block input `ACK_Cap` triggers the falling edge `Cap_Flag` output. 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 **Embedded Functions** → **HSC**.
2	Enable the Capture input in the **HSC0•** → **Auxiliary Inputs** → **CAP** drop down menu.
3	Select a filtering value in the **Auxiliary Inputs** → **CAP Filter** drop down menu.
4	Define the triggering edge in the **Auxiliary Inputs** → **CAP Filter** drop down menu.

Chapter 13
Synchronization and Enable Functions

Overview

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

What Is in This Chapter?

This chapter contains the following topics:

Synchronization Function

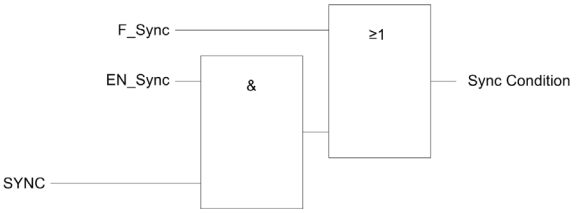
Overview

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

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_Sync` and `EN_Sync`.

This diagram illustrates the synchronization conditions:



EN_Sync input of the HSC function block
F_Sync input of the HSC function block
SYNC physical input SYNC

The function block output `Sync_Flag` is set to 1 when the Sync condition is reached.

The Sync condition operates on a rising edge.

Simple Type Specifications

Sync condition for a **Simple** type corresponds to the function block inputs `Sync`.

The synchronization function can be used in the following counting modes:
- One shot counter: to preset and start the counter
- Modulo loop counter: to reset and start the counter

Main Type Specifications

The SYNC input can be enabled in the configuration.

The synchronization function can be used in the following counting modes:
- One shot counter: to preset and start the counter
- Modulo loop counter: to reset and start the counter
- Free large counter: to preset and start the counter
- Event counting: to restart the internal timer relative to the time base
- Frequency meter: to restart the internal timer relative to the time base

NOTE: In the **Frequency meter** mode, the synchronization function can only be activated with the function block pin `F_Sync`.

Enable Function

Overview

The enable function is used to authorize the counting operation.

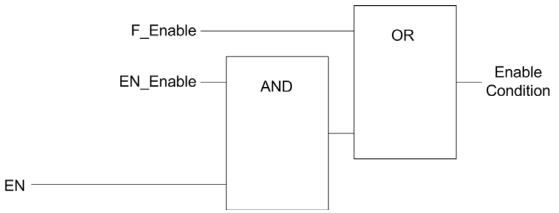
This function is used in the following counting modes:

- One shot counter
- Modulo loop counter

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 diagrams 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 inputs `Enable`.

Configuration

This procedure describes how to configure an enable function:

Step	Action
1	In the **Devices tree**, double-click **HMISCUxx5** → **Embedded Functions** → **HSC**.
2	Select the **HSC•** tab.
3	Set the value of the **HSC** → **HSC0•** → **Auxiliary Inputs** → **EN** parameter to **Enabled**.
4	Select the value of the **HSC** → **HSC0•** → **Auxiliary Inputs** → **EN Filter** parameter.

Appendices

Overview

The appendices provides an overview of data types, function blocks and general information about the function blocks used.

What Is in This Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	General Information	111
B	Data Types	115
C	Function Blocks	121
D	Function and Function Block Representation	131

Appendix A
General Information

Overview

The information described in this chapter is common for PTO and HSC features.

What Is in This Chapter?

This chapter contains the following topics:

Dedicated Features

Dedicated Outputs

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not use the same instance of a function block in more than 1 task.
- Do not modify function block references (••_REF_IN) while the function block is active (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:

HSC_ERR_TYPE: HSC Variable Detected Error Type

Enumerated Type Description

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

Enumerator	Value	Description
HSC_NO_ERROR	00 hex	No error detected.
HSC_UNKNOWN	01 hex	The value assigned to the HSC_REF input pin is incorrect or not configured.
HSC_UNKNOWN_PARAMETER	02 hex	The parameter reference is incorrect. See PARAMETER_TYPE section for valid parameters *(see page 117)*.
HSC_INVALID_PARAMETER	03 hex	The value of the parameter is incorrect. For example, Preset Value is <TH1 or <TH0.
HSC_COM_ERROR	04 hex	Communication error was detected with the HSC module.
HSC_CAPTURE_NOT_CONFIGURED	05 hex	Capture is not configured. It is impossible to get a captured value.

HSC_PARAMETER_TYPE: Type for Parameters to Get or to Set on HSC Variable

Enumerated Type Description

The enumeration data type ENUM contains the following values:

Enumerator	Value	Description
HSC_PRESET	00 hex	To get or set the Preset value of an HSC embedded used for One-Shot and Free-Large mode.
HSC_MODULO	01 hex	To get or set the Modulo value of an HSC embedded used for Modulo-Loop mode.
HSC_TIMEBASE	02 hex	To get or set the Timebase value *(see page 119)* of an HSC embedded used for Event Counting and Frequency mode.
HSC_THRESHOLD0	04 hex	To get or set the Threshold 0 value of an HSC embedded mode.
HSC_THRESHOLD1	05 hex	To get or set the Threshold 0 value of an HSC embedded mode.

HSC_REF: HSC Reference Value

Data Type Description

The HSC_REF is a byte used to identify the HSC function associated with the administrative block.

HSC_TIMEBASE_TYPE: Type for HSC Time Base Variable

Enumerated Type Description

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

Name	Value
HSC_100ms	00 hex
HSC_1s	01 hex
HSC_10s	02 hex
HSC_60s	03 hex

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:

HSCGetCapturedValue: Returns Content of Capture Registers

Function 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 the *Function and Function Block Representation* *(see page 131)* chapter.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
`HSC_REF_IN`	`HSC_REF` *(see page 118)*	Reference to the HSC. 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: ● for Main type counter: always 0

This table describes the output variables:

Outputs	Type	Comment
HSC_REF_OUT	HSC_REF *(see page 118)*	Reference to the HSC.
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.
ErrID	HSC_ERR_TYPE *(see page 116)*	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 113)*.

Adding the HSCGetCapturedValue Function Block

Step	Description
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU HSC** → **HSCGetCapturedValue** in the list, drag-and-drop the item onto the **POU** window.
2	Link the **HSC_REF_IN** input to the **HSC_REF** output of the HSC.

HSCGetDiag: Provides Detail of Detected Error on HSC

Function 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 the *Function and Function Block Representation* *(see page 131)* chapter.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
HSC_REF_IN	HSC_REF *(see page 118)*	Reference to the HSC. 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
HSC_REF_OUT	HSC_REF *(see page 118)*	Reference to the HSC.
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.
ErrID	HSC_ERR_TYPE *(see page 116)*	When Error is TRUE: type of the detected error.
HSCDiag	DWORD	When Done is TRUE, the diagnostic value is output to this pin in the function block. When Bit 7 of the DWORD = TRUE, a configuration error is detected. The Bits 0...6 and 8...15 are not used.

NOTE: For more information about Done, Busy and Execution pins, refer to General Information on Function Block Management *(see page 113)*.

Adding the HSCGetdiag Function Block

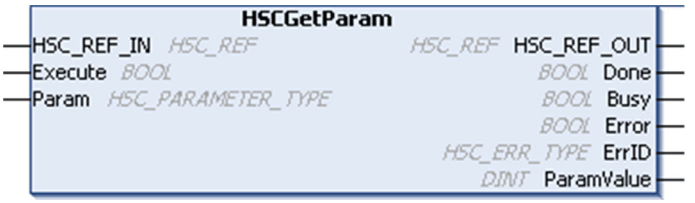
Step	Description
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** →**HMISCU** →**HMISCU HSC** →**HSCGetDiag** in the list, drag-and-drop the item onto the **POU** window.
2	Link the **HSC_REF_IN** input to the **HSC_REF** output of the HSC.

HSCGetParam: Returns Parameters of HSC

Function 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 the *Function and Function Block Representation* *(see page 131)* chapter.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
`HSC_REF_IN`	`HSC_REF` *(see page 118)*	Reference to the HSC. 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.
`Param`	`HSC_PARAMETER_TYPE` *(see page 117)*	Parameter to read.

This table describes the output variables:

Outputs	Type	Comment
HSC_REF_OUT	HSC_REF *(see page 118)*	Reference to the HSC.
Done	BOOL	TRUE = indicates that ParamValue 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.
ErrID	HSC_ERR_TYPE *(see page 116)*	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 113)*.

Adding the HSCGetParam Function Block

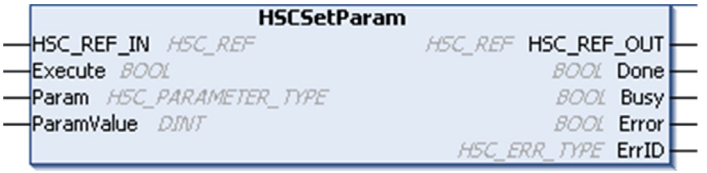
Step	Description
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU HSC** → **HSCGetParam** in the list, drag-and-drop the item onto the **POU** window.
2	Link the **HSC_REF_IN** input to the **HSC_REF** output of the HSC.

HSCSetParam: Adjust Parameters of a HSC

Function 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 the *Function and Function Block Representation* *(see page 131)* chapter.

I/O Variables Description

This table describes the input variables:

Inputs	Type	Comment
`HSC_REF_IN`	`HSC_REF` *(see page 118)*	Reference to the HSC. 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.
`Param`	`HSC_PARAMETER_TYPE` *(see page 117)*	Parameter to read.
`ParamValue`	`DINT`	Parameter value to write.

This table describes the output variables:

Outputs	Type	Comment
HSC_REF_OUT	HSC_REF *(see page 118)*	Reference to the HSC.
Done	BOOL	TRUE = indicates that the parameter was successfully written. 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.
ErrID	HSC_ERR_TYPE *(see page 116)*	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 113)*.

Adding the HSCSetParam Function Block

Step	Description
1	Select the **Libraries** tab in the **Software Catalog** and click **Libraries**. Select **Controller** → **HMISCU** → **HMISCU HSC** → **HSCSetParam** in the list, drag-and-drop the item onto the **POU** window.
2	Link the **HSC_REF_IN** input to the **HSC_REF** output of the HSC.

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.

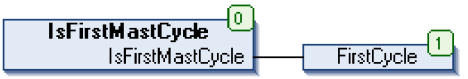
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

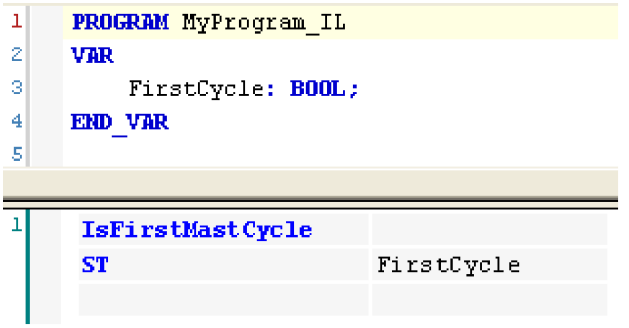
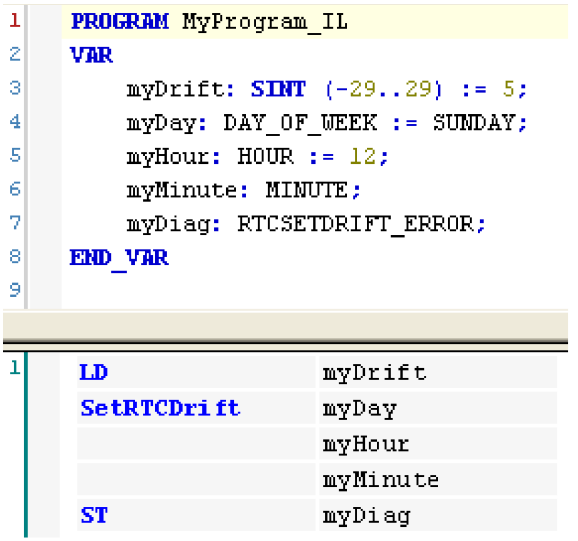
This procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **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: ● type the name of the function in the operator column (left field), or ● 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: `IsFirstMastCycle`	
with input parameters: `SetRTCDrift`	

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

Function	Representation in SoMachine 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

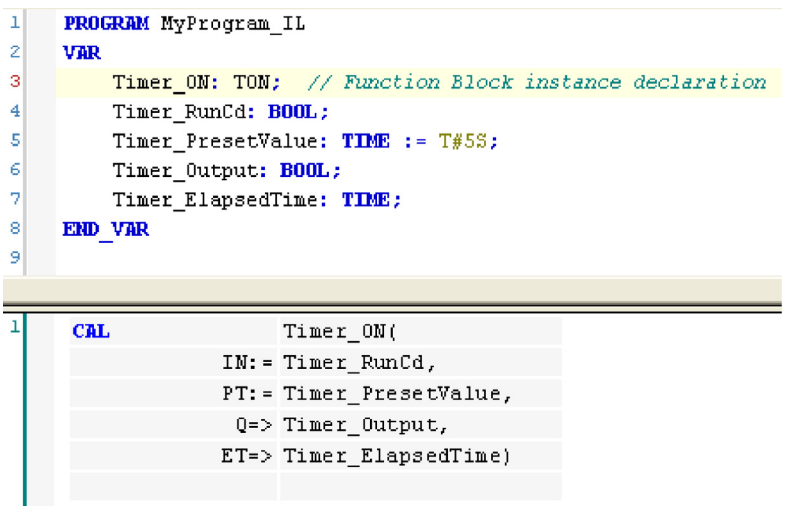
This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **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.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by ":=". ● 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 SoMachine POU IL Editor
TON	

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

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

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. **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: `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 SoMachine POU ST Editor

`SetRTCDrift`	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

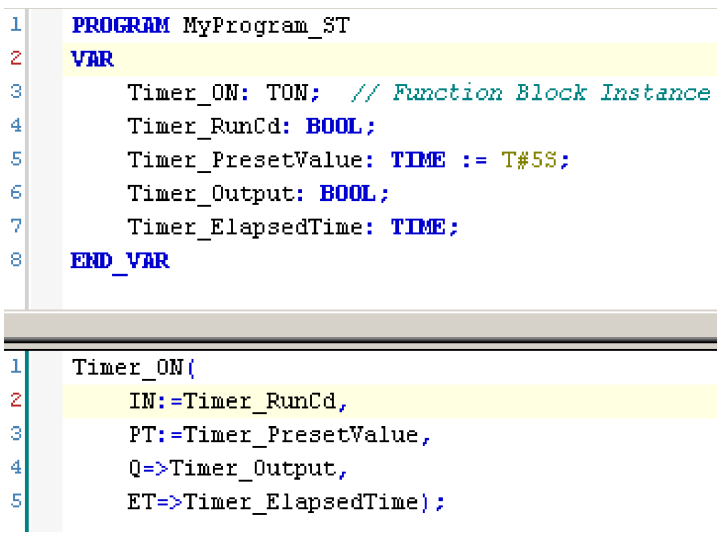
This procedure describes how to insert a function block in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **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: ● Input variables are the input parameters required by the function block ● 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: `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 SoMachine 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

BOOL

(*boolean*) A basic data type in computing. A `BOOL` variable can have one of these values: 0 (`FALSE`), 1 (`TRUE`). A bit that is extracted from a word is of type `BOOL`; for example, `%MW10.4` is a fifth bit of memory word number 10.

byte

A type that is encoded in an 8-bit format, ranging from `16#00` to `16#FF` in hexadecimal representation.

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.

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.

H

HSC

(*high-speed counter*)

I

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).

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. The 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

I

M

S

T