



## THE SIMULATION PLATFORM FOR POWER ELECTRONIC SYSTEMS

**STM32 Target Support User Manual** Version 1.5.1

### How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zürich Switzerland	Mail
@	info@plexim.com	Email
	<a href="http://www.plexim.com">http://www.plexim.com</a>	Web

### *STM32 Target Support User Manual*

© 2023 - 2025 by Plexim GmbH

The product described in this manual is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from Plexim GmbH.

PLECS is a registered trademark of Plexim GmbH. MATLAB, Simulink and Simulink Coder are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

# Contents

<b>Contents</b>	<b>iii</b>
<b>1 Quick Start</b>	<b>3</b>
Requirements for STM32 TSP 1.5.1 . . . . .	3
Installing the Target Support Package . . . . .	3
Build and Deploy Generated Code . . . . .	4
Program the MCU from PLECS . . . . .	4
Program the MCU from STM32CubeIDE . . . . .	5
Start the External Mode . . . . .	7
<b>2 Target Support Architecture</b>	<b>11</b>
Overview . . . . .	11
The Embedded Code Generation Workflow . . . . .	11
Control Task Execution . . . . .	12
Control Task Accuracy and PWM Frequency Tolerance . . . . .	13
Explicit and Implicit Trigger Definitions . . . . .	14
The Code Generation Project . . . . .	22
<b>3 STM32 Coder Options</b>	<b>25</b>
General . . . . .	25
Advanced . . . . .	27
Protection . . . . .	27
External Mode . . . . .	28

<b>4 STM32 Target Support Library Component Reference</b>	<b>31</b>
Analog In . . . . .	32
Analog In (Triggered) . . . . .	34
Base Task Load . . . . .	36
CAN Port . . . . .	37
CAN Receive . . . . .	40
CAN Status . . . . .	42
CAN Transmit . . . . .	43
Control Task Trigger . . . . .	45
DAC . . . . .	46
Digital In . . . . .	47
Digital Out . . . . .	48
Edge Counter . . . . .	49
External Sync . . . . .	52
HRTIM Master . . . . .	53
HRTIM Timing Unit . . . . .	56
Override Probe . . . . .	64
Peak Current Controller . . . . .	65
Powerstage Protection . . . . .	69
Pulse Capture . . . . .	73
PWM . . . . .	76
Quadrature Encoder Counter (QEP) . . . . .	81
Read Probe . . . . .	84
SinCos . . . . .	85
SPI Master . . . . .	86
SPI Slave . . . . .	91
Timer . . . . .	94



# Quick Start

## Requirements for STM32 TSP 1.5.1

The PLECS STM32 Target Support Package currently supports the STM32G4xx and STM32F3xx single-CPU families.

In order to use the PLECS STM32 Target Support Package you will need:

- a host computer (with Microsoft Windows or Mac OS X)
- PLECS Blockset or Standalone 4.9.5 or newer
- PLECS Coder

If you have not done so yet, please download and install the latest PLECS release on your host computer.

## Installing the Target Support Package

Download the appropriate ZIP archive or disk image from the web page [https://www.plexim.com/download/tsp\\_stm32](https://www.plexim.com/download/tsp_stm32), extract it and move the stm32 folder to the PLECS Coder target support packages path e.g. to HOME/Documents/PLECS/CoderTargets. In PLECS, choose **Preferences...** from the **File** drop-down menu (**PLECS** menu on Mac OS X) to open the PLECS Preferences dialog.

Navigate to the **Coder** tab and click on the **Change** button to select the HOME/Documents/PLECS/CoderTargets folder. The targets included as part of the STM32 Target Support Package should now be listed under **Installed targets**. You will also see these targets available in the **Coder + Coder options...** window in the drop-down menu on the **Target** tab.

Another folder labeled projects is included in the ZIP archive. The contents of this folder is required only when the PLECS Coder is configured to generate

code into STM32CubeIDE projects. The projects/stm32xx.zip files contain Cube IDE projects that are used in conjunction with the embedded code generated from PLECS.

A set of basic demos is also included with the STM32 Target Support Package.

## Build and Deploy Generated Code

There are two primary methods for building and deploying generated embedded code onto a STM32 MCU.

**1 Build and program the MCU from PLECS** You can directly program the target device from the PLECS application. This approach does not require any external tools, although, optionally, the Segger J-Link package can be used in this workflow. Clicking **Build** in the **Coder Options** dialog generates model and supporting hardware configuration code, builds the application using the ARM GCC tools, and then flashes the target via Open OCD (or J-Link).

**2 Build and program the MCU from the STM32CubeIDE** In this approach the PLECS Coder generates code for the specified target into a template Cube IDE project. The Cube IDE is then used to build the project and flash the target device. The advantage of this method is that the generated code can easily be inspected. Further, the developer has access to debugging tools.

If the required software is installed on your PC you can easily switch between the two methods by checking or unchecking the **Generate code only** parameter in the **Coder options... + Target + General** menu.

## Program the MCU from PLECS

PLECS can automatically program the target MCU after it finishes generating and building code. Programming occurs over the GNU debug protocol and requires a GDB server.

Two options exist:


**1 Open OCD** The STM32 Target Support Package includes the Open On-Chip Debugger package and GDB server, supporting the ST-LINK debug link out of the box (no external tools needed).

**2 Segger J-Link** Segger debug probes offer superior throughput and are also supported by the STM32 Target Support Package – see below on how to install the Segger tools.

Note that the debug link can also be used in PLECS External Mode to communicate with the target while it is executing the generated code. It is in External Mode that the high throughput of J-Link is most noticeable.

## Configuring Segger J-Link Tools

The Segger J-Link tools must be downloaded and installed separately.

To configure the PLECS Coder to use the Segger tools, select **Preferences...** from the **File** drop-down menu (**PLECS** menu on Mac OS X) to open the PLECS Preferences dialog. Click the **Coder** tab to see the installed targets. Click the  icon in the **Family** column next to the **STM32** entry and enter the path to the J-Link installation (e.g. /Applications/SEGGER/JLink).

## Deploy code to STM32 target from PLECS

To deploy code to a STM32 target from PLECS, navigate to the PLECS **Coder Options + Target** window, select the target MCU. Uncheck the **Generate code only** parameter, then choose the desired **Programming interface** from the dropdown menu. The default programming interface is **Open OCD**.

## Program the MCU from STM32CubeIDE

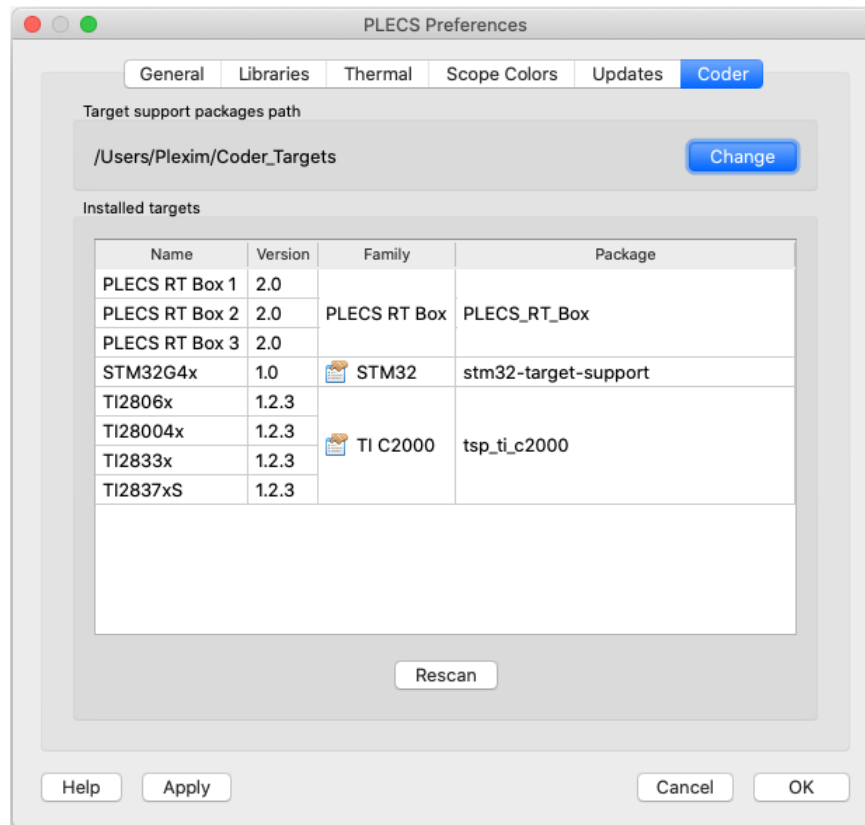
### Configure STM32CubeIDE

Download and install the latest version of STM32CubeIDE from the STM website. This is available at the following location:

<https://www.st.com/en/development-tools/stm32cubeide.html>

After installing STM32CubeIDE, import the appropriate template project from the STM32 Target Support Package. Open STM32CubeIDE and click the **File** drop-down menu and then select **Import...** From **General + Existing Projects into Workspace**, choose the zip archive in the projects folder that corresponds to the desired target. You will notice a new project created in your workspace.





**Figure 1.1: Configuring the target support package and external tool paths**

## Deploy code from STM32CubeIDE

Return to the PLECS application, navigate to the **Coder + Coder Options...** window and select the **Target** tab. Check the **Generate code only** parameter checkbox. Enter the location of the `${workspace_loc}/cube_g4xx/cg`, `${workspace_loc}/cube_f3xx/cg` or `${workspace_loc}/cube_h7xx_single_cpu_targets/cg` folder from the STM32CubeIDE project into the **STM32CubeIDE project directory** field and click **Build**. Note that `{workspace_loc}` refers to the location of the imported project in the STM32CubeIDE workspace. You will notice several new files created in the

`${workspace_loc}/cube_g4xx/cg`, `${workspace_loc}/cube_f3xx/cg` or `${workspace_loc}/cube_h7xx_single_cpu_targets/cg` directory. Then, proceed to build and debug your project as you would a normal STM32CubeIDE project. The project will not compile without first generating code from PLECS.


Note that it is necessary to manually delete the contents of the `${workspace_loc}/cube_g4xx/cg`, `${workspace_loc}/cube_f3xx/cg` or `${workspace_loc}/cube_h7xx_single_cpu_targets/cg` folder when generating code for a new subsystem of a different name, as the STM32CubeIDE builder will build all files in this folder, including old files.

## Start the External Mode

Once the generated code is running on the embedded target, the user can enter the External Mode to update Scopes in the PLECS application with real-time waveforms and change certain simulation parameters.

External mode can be configured to run over JTAG or Serial. This choice must be configured from the **Coder + Coder options... + Target + External Mode** window prior to building the project.

To establish a communication link over JTAG with your target, follow the instructions provided below:

- Open the **Coder options... + External Mode** tab and then select the  icon next to the **Target device** field.
- Select **Serial over GDB**, configure the device name to **127.0.0.1** and then click the **OK** button to proceed.
- Click the **Connect** button and if the connection is successful you will see the trigger controls activate.
- Set the **Number of samples** parameter to e.g. 1000 and click on the **Activate autotriggering** button.

To establish a communication link over Serial with your target, first configure the proper USART channels from the **Coder + Coder options...+Target + External Mode** window. Then, **Scan** for the appropriate **Target device** and proceed to **Connect** to the target as described in the instructions above.

You will now see real-time data from the MCU in the PLECS Scopes. You can synchronize the data capture to a specific trigger event. To do so, change the **Trigger channel** selection from **Off** to the desired signal. The Scope will now show a small square indicating the trigger level and delay. If the level or delay are outside the current axes limits, a small triangle will be shown instead. Drag

the trigger icon to change the trigger level; drag it with the left mouse-button pressed to change the trigger delay. Both parameters can also be set in the External Mode dialog.

---

**Note** While a trigger channel is active, the Scope signals are only updated when a trigger event is detected.

---


While the PLECS model is connected via the External Mode, the model is locked against modifications. To disconnect from the MCU and other External Mode connections, click on the **Disconnect** button or close the Coder Options dialog.

## Parameter Inlining

Certain values on the target device can be changed in real-time, when connected to the target device via the External Mode, if the component is added to the "Exceptions" list found in the **Parameter Inlining** tab of the **Coder options...** window, prior to building the model. Changes in the parameters will be reflected in the Scope traces once they take effect.

## Connecting the external mode to several targets

External mode can be connected to several targets at the same time. When using external mode over Serial each target can be identified by the **Device names**. When the External mode is established via JTAG a unique GDB port has to be opened for each connection. This can be achieved by following the steps below:

- Open the **Coder options... + External Mode** tab and then select the  icon next to the **Target device** field.
- Select **Serial over GDB**, configure the device name to **127.0.0.1:port\_number** and then click the **OK** button to proceed. The port\_number must be an integer number between 1024 and 65535. The port number is separated to the IP address with a colon character ( : ). For each connection a different port number has to be configured.
- Apply the changes by clicking on **Accept** on the lower left side of the **Coder Options** window.

- Click the **Connect** button and if the connection is successful you will see the trigger controls activate.

When working with multiple targets at the same time it is recommended to specify serial numbers to identify the target devices.



# Target Support Architecture

## Overview

As a separately licensed feature, the PLECS Coder can generate C code from a simulation model to facilitate embedded code generation. Plexim provides and maintains target support packages (TSPs) for specific processor families. A TSP enables the PLECS Coder to generate code that is specific to a particular hardware target such as the STM32 family of MCUs or the PLECS RT Box. With the PLECS Coder and a TSP, embedded control code can be generated, compiled, and uploaded to the target device directly from the PLECS environment with minimal effort. Furthermore, the embedded control logic can be tested extensively inside the PLECS simulation environment prior to real-time deployment.

## The Embedded Code Generation Workflow

The embedded workflow is designed for you to easily transition from a PLECS model to an embedded code generation project without having to build and maintain separate models. A typical embedded code generation workflow consists of the following steps:

- 1** Design and simulate a controller and plant in PLECS. The controller represents the application that will run on the embedded target. The plant represents the hardware connected to the embedded target including the power stage and other physical systems.
- 2** Add components from the target support library to configure the embedded peripheral devices. Place the controller and peripheral models into a subsystem representing the embedded target.

- 3** Run an offline simulation. All peripheral components in the target support library have behavioral offline models to facilitate the transition from simulation to real-time deployment.
- 4** Select a discretization step size and nominal control task execution frequency. When generating C code, the PLECS Coder will use the discretization step size to automatically transform all continuous states in the controller to the discrete state-space domain using the Forward Euler method. The control task execution frequency is based on the discretization step size and specifies the nominal execution rate of the digital control loop.
- 5** Build the embedded project and flash the MCU using PLECS or the STM32CubeIDE.
- 6** Connect to the MCU using the External Mode to test the embedded control code executing on the embedded target.

## Control Task Execution

Embedded applications for power electronics typically sense signals from the power converter, process the inputs using digital control laws, and output signals to actuation devices. The STM32 TSP library includes components to model and program the MCU peripherals for sensing and actuation. The control laws are implemented using standard PLECS library components.

Time synchronization of signal measurement via the analog-to-digital converter (ADC), control logic execution, and actuation via PWM outputs is critical in the digital power electronic control loop. The STM32 TSP provides the flexibility to configure the ADC and control loop interrupts through the ADC trigger and task trigger signals.

Note that there are two types of Analog In (ADC) blocks in the STM32 target library: continuous Analog In and triggered Analog In. As the name suggests, continuous Analog In block converts the selected inputs continuously. Whereas the triggered block converts the selected inputs once per trigger. The following sections use the triggered Analog In block.

ADC triggers start injected ADC conversions. The ADC start-of-conversion is driven by an event generated from a timer based component. All injected conversions associated with an ADC unit are converted sequentially when the ADC trigger is activated. The order of conversion is based on the order of the analog input channel vector. Note that regular non-triggered ADC conversions are also available.

Task triggers are generated at the end of injected conversions, PWM counter underflow and overflow events, or the Timer block events. The task trigger that connects to the Control Task Trigger component will trigger one execution of the digital control loop at the nominal base sample rate.

Additionally, the PLECS Coder and the STM32 TSP allow the user to generate multi-tasking code for the STM32 family of MCUs. For further information, refer to the "Code Generation" section in the PLECS User Manual. Multi-tasking code unlocks processing power for controls regulating multiple system outputs with dynamics on a range of time-scales. Using the Task library component, 15 additional tasks that execute at different rates (not including the base task) can be specified, preserving processor time for the fastest, highest priority control task (base task) in the application.

Multi-tasking code generation is configured in the **Scheduling** tab of the **Coder + Coder options...** dialog. By changing the **Tasking mode** to multi-tasking, the sample time for each task can be configured. The base sample time is always equal to the **Discretization step size**. The **Sample time** setting for lower priority tasks must be an integer multiple of the base sample time.

In a multi-tasking mode, the Control Task Trigger component triggers the base task associated with the nominal base sample time.

---

**Note** In the following sections, unless specified otherwise, *control task* and *base task* can be considered synonymous.

---

## Control Task Accuracy and PWM Frequency Tolerance

The MCU system clock frequency, SYSCLK, fundamentally limits the time accuracy of the embedded target. SYSCLK is defined in the **Target + General** tab of the **Coder + Coder Options** window. The Timer and PWM carrier generation clocks are derived from an integer number of counts of SYSCLK. Therefore the time accuracy of task triggers and PWM carriers are also limited.

Consider the case where there is a desired PWM carrier frequency of 150 kHz and the SYSCLK is set to 100 MHz. The closest achievable PWM carrier frequency is 150.15 kHz. Note that if the SYSCLK setting was changed to 90 MHz, then the target PWM frequency of 150 kHz could be achieved exactly.



In cases where the PWM carrier frequency or ADC and task trigger periods cannot be achieved exactly, the default behavior is to generate an error message displaying the desired frequency or step size and the closest achievable value. Adjusting the **Frequency tolerance** parameter overrides this behavior and configures the PLECS Coder to automatically select the closest achievable frequency. The **Frequency tolerance** can be configured in the mask parameters of the Timer, PWM, and Peak Current Controller (PCC) target support library blocks.

The discretization step size configured in the **Scheduling** tab of the **Coder + Coder Options** will also generate an error if the exact step size cannot be achieved. This impacts the nominal period of the task trigger and introduces a numerical inaccuracy since C code derived from the model executes at a different rate than was assumed during model discretization. The **Frequency tolerance** parameter relating to model and control task discretization can be adjusted in the **General** tab of the **Coder + Coder Options + Target** window.

### Explicit and Implicit Trigger Definitions

The interrupt sequence of the embedded application can be defined explicitly by connecting trigger signals, or implicitly where the interrupt sequence is automatically determined based on the components included in the schematic. Implicitly defined control loops will not have a Control Task Trigger component included in the schematic and all ADC trigger sources must be automatically determined. Several possible explicit and implicit trigger sequences are discussed below.

---

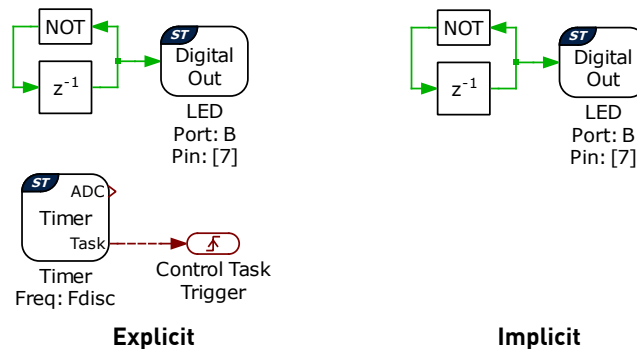
**Note** Explicitly defined trigger systems require that the Control Task Trigger's nominal base sample time parameter agrees with period of the task trigger input signal.

---

## Control task triggered by Timer

In a basic project without an ADC or PWM component from the target support library, the task trigger must be generated by a generic Timer. The schematic below shows a simple application where a digital output is toggled at a fixed rate.

The explicit representation of the control task execution includes a Timer component that generates the input signal for the Control Task Trigger. The nominal base sample time of the Control Task Trigger must agree with the Timer task frequency. In the implicit representation the PLECS Coder will configure the Timer and Control Task Trigger automatically based on the **Discretization step size** parameter set in the **Coder + Coder Options + General** menu.



**Figure 2.1: Basic model with control task triggered by Timer**

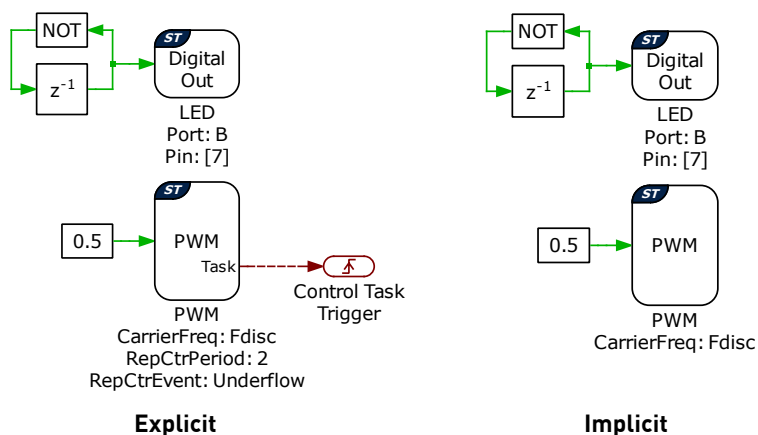
## Control task triggered by PWM

Control task execution can be synchronized with the PWM carrier, and the repetition counter period determines how many events need to occur before a trigger is generated. When the carrier type is symmetrical, for even values of repetition counter period, the interrupts will occur at the carrier underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the respective carrier minimum or carrier maximum values. The task trigger is configured in the **Trigger** tab of the PWM component.

In the explicit representation, the PWM task trigger output is connected to the Control Task Trigger component, such that execution of the digital control loop is synchronized with the PWM carrier. If the schematic does not include a Control Task Trigger or an ADC component, then the PLECS Coder will implicitly

select the most appropriate source for the task trigger. First, the PWM generator that can achieve the control task frequency with the highest precision is chosen, starting from the lowest PWM number. If the control task frequency cannot be achieved exactly using a PWM carrier, then the implicit trigger logic will determine if more accurate task execution can be achieved with the Timer. The most accurate source for the control task interrupt is then selected.

The task trigger will default to triggering on underflow and overflow when the task trigger is set to disabled in the PWM **Trigger** tab and the trigger is implicitly defined.

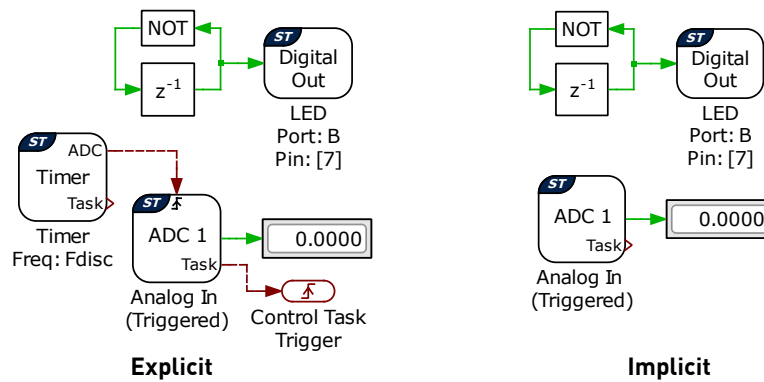


**Figure 2.2: Basic model with control task triggered by PWM**

## Control task triggered by Timer via ADC

If the schematic includes an ADC but no PWM generators, then the ADC start-of-conversion must be triggered by the Timer. In this case, the control task can be triggered by the ADC end-of-conversion or the Timer. When the ADC end-of-conversion is the source of the Control Task Trigger input, as shown in Figure 2.3, then the control loop interrupt will occur after all ADC results registers are updated with the latest measurement values.

The implicit implementation automatically configures the Timer to periodically trigger the ADC start-of-conversion. The ADC trigger period is set by the **Discretization step size** parameter found in the **Coder + Coder Options + Scheduling** menu. The ADC unit with the greatest number of channels will trigger the control task.

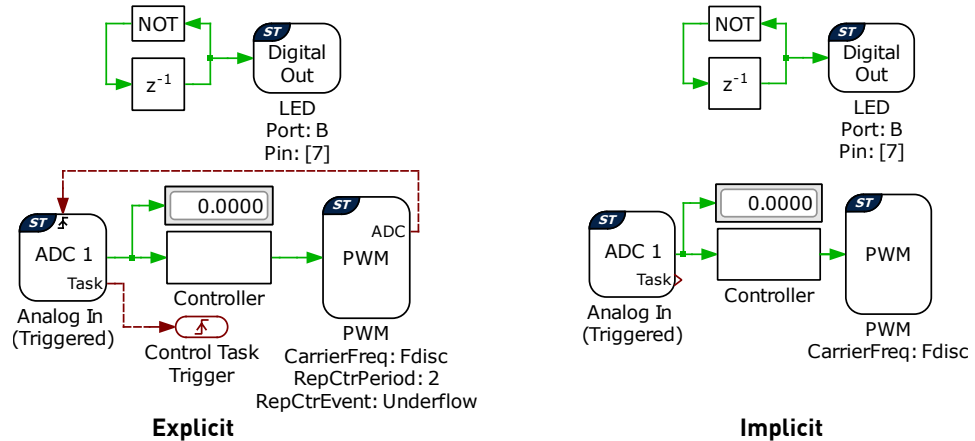


**Figure 2.3: Basic model with control task triggered by ADC**

## Control task triggered by PWM via ADC

Figure 2.4 shows the explicit and implicit implementations of the control task being triggered by the ADC via the PWM. The sequence of events begins when the PWM carrier reaches an underflow or overflow triggering the start-of-conversion signal for the first ADC channel. The ADC channels are sampled and updated sequentially until the result register of the final ADC channel is updated. Once all ADC results are available, the ADC end-of-conversion interrupt triggers the control task. This arrangement synchronizes the ADC start-of-conversion with the PWM actuation and ensures the ADC results registers are updated prior to executing the control loop.

When both ADC and PWM components are included in any schematic, the PLECS Coder will implicitly select the the PWM generator with the highest control task accuracy as the ADC trigger. If the PWM generators cannot trigger the ADC at the exact target frequency, then the Timer will be used if it is more accurate. The control task will always be triggered by the ADC end-of-conversion signal.

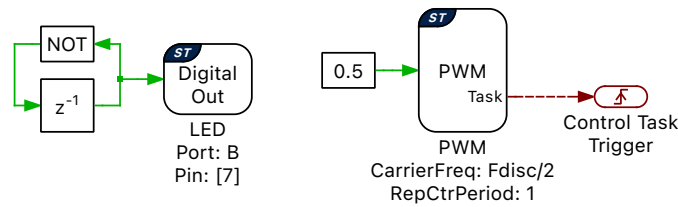


**Figure 2.4: Basic model with control task triggered by PWM via ADC**

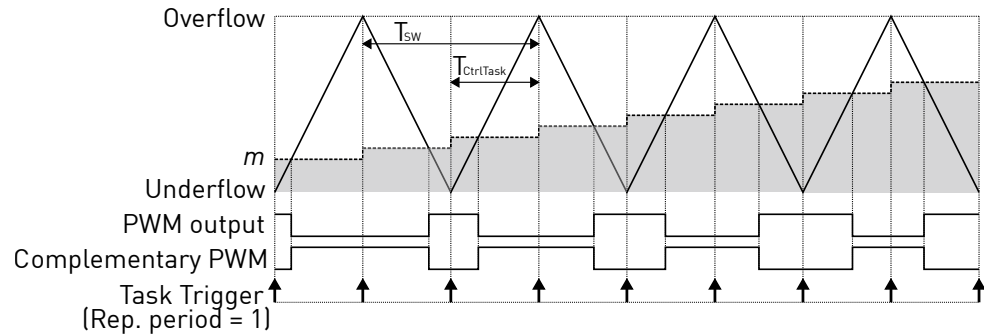
## Advanced explicit configurations

The control task interrupt can execute at integer multiples of the PWM carrier frequency, and for a symmetric carrier the control task can be triggered at twice the PWM carrier frequency.

Figure 2.5 shows a case where the discretization frequency is  $F_{disc}$ , the symmetric PWM carrier period is  $T_{sw} = 2/F_{disc}$  Hz, and the Control Task Trigger interrupt period is  $T_{CtrlTask} = 1/F_{disc}$ . The control task is triggered twice per PWM period. Figure 2.6 shows the corresponding PWM carrier, task trigger, and PWM outputs.



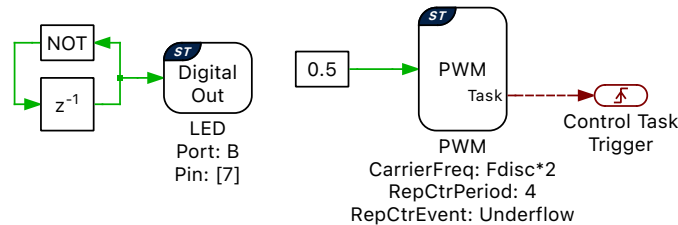
**Figure 2.5: PWM frequency set to half the control task frequency**



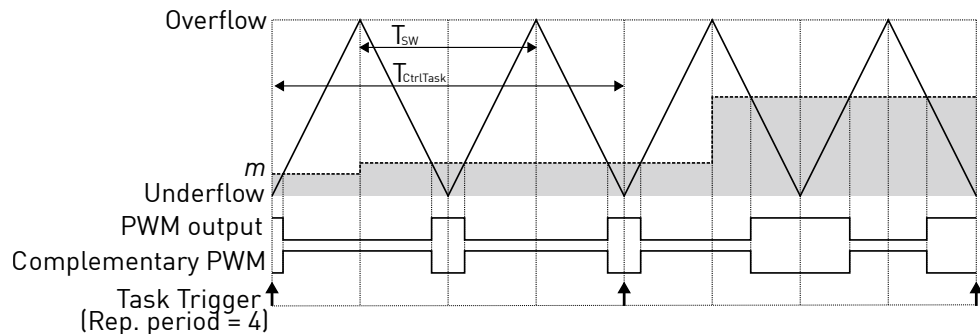
**Figure 2.6: PWM carrier and task interrupts for PWM frequency set to half the control task frequency**

Figure 2.7 shows a case where the discretization frequency is  $F_{disc}$ , the symmetric PWM carrier period is  $T_{sw} = 1/(2 \cdot F_{disc})$  Hz, and the Control Task Trigger interrupt is generated at  $T_{CtrlTask} = 1/F_{disc}$ . Figure 2.8 shows the corresponding PWM carrier, task trigger, and PWM outputs.

The STM32 TSP by default will only update the PWM duty cycle register on PWM underflow and overflow events to prevent data corruption. In Figure 2.8 note the delay between the task trigger and the instant when the duty cycle,  $m$ , is updated in the PWM module. The task trigger initiates the control task computation, but the modulation index is updated on the next overflow or underflow event after the entire control task has been completed. When the control task is triggered by the ADC end-of-conversion, then the modulation index will update on the next overflow or underflow event after all ADC channels are converted and the control task is completed.



**Figure 2.7: Schematic of PWM frequency set to twice the control task frequency**

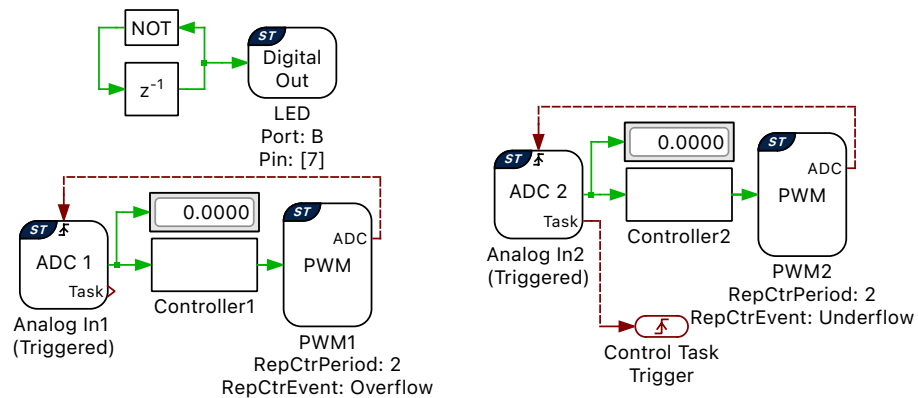


**Figure 2.8: PWM carrier and task interrupts for PWM frequency set to twice the control task frequency**

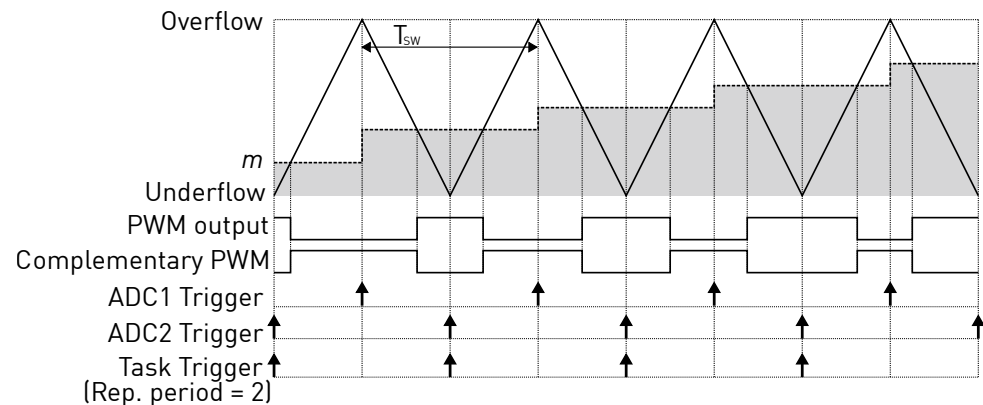
Each ADC can receive independent start-of-conversion triggers from different PWM generators for phase-shifted sampling. Figure 2.9 shows the case where the ADC1 component is triggered on the carrier overflow and ADC2 is triggered

on carrier underflow from two different PWM modules with a common carrier frequency.

After all channels associated with ADC2 are converted the control task is executed with updated measurements from ADC1 and ADC2. On the next carrier overflow the PWM duty cycle register is updated.



**Figure 2.9: Explicit phase-shifted ADC sampling**



**Figure 2.10: PWM carrier and interrupts for phase-shifted ADC sampling**



## The Code Generation Project

This section provides additional technical background on the software architecture of the embedded code generation project included with the STM32 TSP. A STM32CubeIDE project is included for each supported target chip in the projects/ folder of the TSP. When building the project from directly from the PLECS application, the files in gcc/g4 and gcc/f3 folder of the TSP are used.

### Static and dynamic code

The embedded code generation project consists of dynamic and static code. Dynamic code is generated by the PLECS Coder and is overwritten each time the **Build** button is clicked in the **Coder + Coder options...** window. Static code is provided with the TSP and should not be modified. The PLECS Coder also generates additional dynamic configuration files that are used by the embedded application.

When the `Generate code only` parameter checkbox is checked, then all generated dynamic code must be placed into the `${workspace_loc}/cube_g4xx/cg` or `${workspace_loc}/cube_f3xx/cg` folder of the imported STM32CubeIDE project depending on the configured target. Otherwise, by default all generated code is included in a new output directory in the same folder as the saved PLECS model.

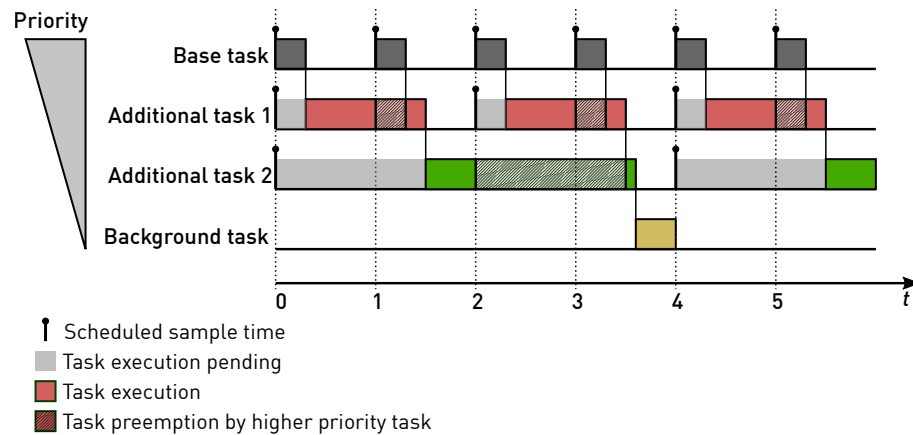
### Control and background task dispatching

The application framework includes a rate monotonic scheduler to allow precise and efficient execution of the digital control loops. The base task is executed at the highest priority. Additionally, up to 15 slower lower-priority tasks, executed at different rates, can be specified. For further information on task scheduling, refer to the "Code Generation" section in the PLECS User Manual. A lowest-priority background task also exists to handle non-time critical tasks. Figure 2.11 shows a configuration with a base task, one additional task, and a background task executing in real-time on the MCU.

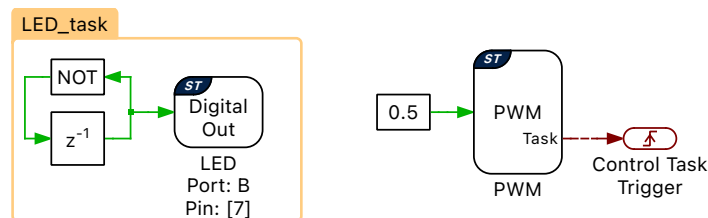
With every control task trigger interrupt issued by the Timer, PWM, or ADC end-of-conversion (bold vertical bar), any lower priority tasks are interrupted and the base task is executed. This ensures that the control task has the highest priority. In addition, the lower priority tasks are periodically triggered and executed when no higher priority tasks are active or pending.

Multi-tasking code generation is configured in the **Scheduling** tab of the **Coder + Coder options...** dialog. By changing the **Tasking mode** to multi-tasking the sample time for each task can be configured. The **Sample time** setting for lower priority tasks must be an integer multiple of the base sample time. The non-default tasks can be defined in the model window using the Task library component. An example of an additional LED task, along with a base PWM task is shown in figure 2.12.

Once the base and additional tasks have completed, the system continues with the background task where lowest priority operations are processed.



**Figure 2.11: Nested control tasks**



**Figure 2.12: Example of an additional LED task along with a base PWM task**

If the base task is still executing when a second control task interrupt is received, then the processor will halt and an assertion will be generated. Similar behavior occurs if a low priority task does not complete by the time it is scheduled to execute again. If the task scheduler detects a CPU overflow and

the model contains a Powerstage Protection (see page 69) target block, all PWM/HRTIM outputs are switched to the passive state. Assertions can be monitored using STM32CubeIDE debug tools.

Embedded project architecture

Figure 2.13 shows the architecture of the embedded project included with the STM32 TSP. At the top of the software stack is an application layer consisting of the main application and the base and additional tasks. Next, there is a dispatch routine that provides a rate monotonic scheduler for the nested control tasks, as previously described. The user can choose between a custom bare-metal task scheduler or FreeRTOS™, a real-time operating system, to be used as the task scheduler. The processor-in-the-loop (PIL) framework acts as middleware for External Mode communication with the PLECS application on the user PC. The hardware abstraction layer (HAL) provides a hardware agnostic interface between the application and chip specific configuration settings. This ensures code portability between different processor platforms. The hardware specific function calls utilize the STM32 drivers to configure the MCU and key peripherals. At the bottom of the stack is the embedded hardware which includes the MCU, peripheral devices, and other onboard accessories.

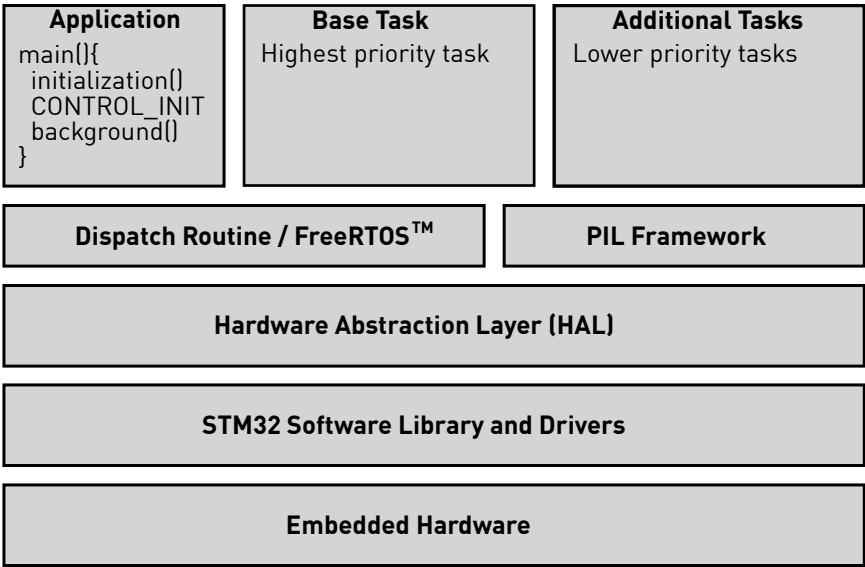


Figure 2.13: Embedded project architecture

# STM32 Coder Options

The **Target** page contains code generation options which are specific to the STM32 Target Support Package.

## General

**Chip** Selects the target device chip series.

**Package size** Selects the number of pins of the chip.

**Memory size** Select the Flash memory size of the chip.

**Full chip name** Read only field that displays the full chip name according to the selected chip, pin count and memory size.

**System clock frequency (SYSCLK)** Specifies the system clock frequency in megahertz (MHz).

**Use internal oscillator** Selects the on-chip oscillator as the clock source. The clock frequency is automatically specified based on the target device.

**External clock frequency** Specifies the frequency in megahertz (MHz) of the external clock source when the internal oscillator is not used.

**External ADC reference** Specifies the external reference for VREF+ in volts (V). VREF+ is the positive reference voltage for an analog input (ADC) or output (DAC) signal. VREF+ range is  $1.62\text{ V} \leq \text{VREF+} \leq \text{VDDA}$ , where VDDA is analog power supply ( $1.62\text{ V} \leq \text{VDDA} \leq 3.6\text{ V}$ ).

**Step size tolerance** The desired control task frequency may not be achievable based on the system clock frequency and the nominal discretization time step. This setting configures the Coder to either Enforce exact value by generating an error when the exact control task frequency is unachievable or to automatically Round to closest achievable value.

**Build type** This setting specifies the action of the **Build** button. Generate code into STM32CubeIDE project will generate code into the specified STM32CubeIDE project. STM32CubeIDE must then be used to build the project and flash the MCU. The Build only option will generate the code and build it. The resulting binary file must then be flashed on the MCU by using a third-party tool, e.g. the STM32CubeProgrammer. The Build and program option will automatically build and flash the target device from within PLECS.

**STM32CubeIDE project directory** Specifies the target folder for code generation. The code must be generated into a pre-configured STM32CubeIDE project. When using the STM32CubeIDE project templates provided with the STM32 target support package, code must be generated into the {workspace\_loc}/cube\_g4xx/cg or {workspace\_loc}/cube\_f3xx/cg folder where {workspace\_loc} refers to the location of the imported project in the STM32CubeIDE workspace.

**Programming interface** Provides an option to automatically program the target MCU from PLECS after it finishes generating and building code. Programming occurs over the GNU debug protocol and requires a GDB server. There are two options available:

- **Open OCD** Supports the Open On-Chip Debugger package and GDB server, supporting the ST-LINK debug link out of the box (no external tools needed).
- **Segger J-Link** Supports Segger debug probes, which offer superior throughput. The Segger J-Link tools must be downloaded and installed separately.

**OpenOCD configuration** Allows to set a Custom OpenOCD configuration script to establish the connection to the target device.

**Autoprobe connected device** Enables the option to automatically check if the device that is connected to the host computer matches the configured Chip. If a different chip is detected the build process aborts with an error message.

**Specify serial number** Enables the option to specify a serial number to identify a target MCU. This is of interest when several targets are connected at the same time to the same host computer.

**Serial number** Allows to specify the serial number of the target. The serial number of a STM32 chip can be identified with third-party tools, e.g. STM32CubeProgrammer. The entry is interpreted as a string and variables are therefore not evaluated.

**Generate pinmap file** Specifies if a pinmap should be generated during the build process. If set to Generic board a generic table independent from any hardware board is generated. The table shows a summary of all input and output pins used in the model for which code is generated for. If set to Nucleo-64

---

board the input and output pins required by the simulation model are mapped to the respective ST morpho connector pins (CN7 and CN10) of a Nucleo 64 pin based board. The pinmap is generated as a .html file in the Output directory as specified in the **General** tab of the **Coder options**.

## Advanced

**Task scheduler** Specifies if the task scheduling is based on FreeRTOS™ or a light-weight bare-metal dispatch routine (Bare-metal). The bare-metal dispatch routine is more efficient compared to FreeRTOS™ and allows the user to reach higher execution rates of the base task. However, only up to 6 different tasks are allowed.

**Freeze timer when pausing the debugger** If set to Enabled, all timers are paused when a breakpoint is hit during debugging. This parameter is only visible if the **Build type** is set to Generate code into STM32CubeIDE project.

## Protection

Analog inputs can be configured on this tab to generate trip events for the Powerstage Protection block. Note that in order for a protection input to have an effect it has to be explicitly activated in the **Protection** tab of the Powerstage Protection block (see page 69).

### Analog protection signals

Up to seven analog comparators can be enabled and configured for generating trip signals for the Powerstage Protection block.

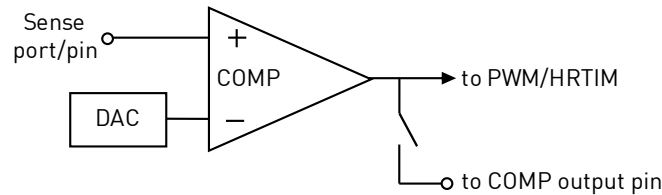
The configured sense port/pin must be connected to a positive (+) comparator GPIO. The comparator is automatically determined by PLECS and an error message is issued in case no suitable comparator is found.

The user can configure a fault threshold voltage. Note that the fault threshold voltage has to be set to a value between 0.0 V and 3.3 V. The fault threshold is internally generated with a digital-to-analog converter (DAC), connected to the negative (–) comparator pin, as shown in the figure below.

**Enable analog protection signal 1-7** Enables an analog protection signal.

**Sense port** Specifies the port used for the analog signal connection.

**Sense pin** Specifies the pin used for the analog signal connection.



#### Setup for an analog protection signal.

**Trip behaviour** Selects the polarity of the trip signal. If set to Trip if signal is below threshold the trip signal will be emitted if the connected analog signal value is below the configured **Fault threshold value**. If set to Trip if signal is above threshold the trip signal will be emitted if the connected analog signal value is above the configured **Fault threshold value**.

**Fault threshold voltage** Configures the fault threshold in volts (V). The value has to be between 0.0 V and 3.3 V.

**Comparator output** Selects if the output signal of the comparator should be fully internal or also connected to a specified pin. If set to Internal the output signal of the comparator is internal only and not accessible. If set to Internal and external the output signal of the comparator is also available for further usage at a specified pin of the MCU, see graphic above.

**Comparator output port** Specifies the port used to output the comparator output signal.

**Comparator output pin** Specifies the pin used to output the comparator output signal.

## External Mode

These options are used to configure the External Mode communication with the target device. This choice must be configured prior to building the project.

**External Mode** This setting adds code to the target device that enables the External Mode. Code size and memory consumption are increased when the External Mode is enabled. There are two communication options available, Serial or JTAG.

---

**Target buffer size** Specifies how much target memory (16-bit words of RAM) should be allocated to buffering signals for the external mode. The number of words  $N_w$  required by the external mode can be calculated as follows:  $N_w = N_{signals} \cdot 2 \cdot (N_{samples} + 1)$ . If more samples are requested than what is supported by the memory allocation, PLECS will automatically truncate the scope traces to the maximal possible  $N_{samples}$  value. Note, however, that requesting more memory than what is available on the target will result in a build error. Recommended values for this setting are in the range of [500 . . . 2000] for STM32F3xxRE or STM32G4xxRx chips. For chips that can be identified with STM32F3xxR8 the target buffer size should be in the range of [100 . . . 500].

**USART Rx Port** Specifies the Rx port used for the External Mode USART connection.

**USART Rx Pin** Specifies the Rx pin used for the External Mode USART connection. This pin cannot be used by other peripherals.

**USART Tx Port** Specifies the Tx port used for the External Mode USART connection.

**USART Tx Pin** Specifies the Tx pin used for the External Mode USART connection. This pin cannot be used by other peripherals.

**Debug interface** Provides an option to communicate with the target MCU from PLECS after building code from STM32CubeIDE. Communication occurs over the GNU debug protocol and requires a GDB server. There are two options available:

- **Open OCD** Supports the Open On-Chip Debugger package and GDB server, supporting the ST-LINK debug link out of the box (no external tools needed).
- **Segger J-Link** Supports Segger debug probes, which offer superior throughput. The Segger J-Link tools must be downloaded and installed separately.





# STM32 Target Support Library Component Reference

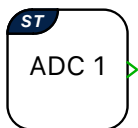
This chapter lists the contents of the STM32 Target Support Library in alphabetical order.

## Analog In

**Purpose** Asynchronous low-priority voltage measurements on an analog input channel

**Library** STM32

### Description



Analog In

The Analog In block configures channels on an ADC submodule specified by the user. The selected input channels are converted continuously without any CPU intervention. Furthermore, the channels are configured to operate in the low-priority regular conversion mode. As a result, the ADC measurement conversions are not synchronous with events created by Timer, PWM or HRTIM blocks and are interrupted by high-priority injected conversions associated with Analog In (Triggered) blocks.

The Analog In block output signal represents the voltage measured on the input pin. The output can be scaled and an offset can be applied after scaling; it is calculated as  $\text{input} \times \text{Scale} + \text{Offset}$ . If the **Analog input channel(s)** parameter is vectorized, each input channel is measured sequentially in the order that it appears in the input channel vector.

The Analog In block is well-suited to background measurements.

### Parameters

#### Main

##### ADC unit

Selects the peripheral index of the ADC submodule that is used.

##### Analog input channel(s)

Index of the analog input channel that is used. If this parameter is vectorized, each input channel is measured sequentially in the order that it appears in the input channel vector.

##### Scale(s)

A scale factor for the input signal.

##### Offset(s)

An offset for the scaled input signal.

##### Acquisition time

Selects between a minimal or user specified ADC acquisition time.

##### Acquisition time value(s)

Sets the ADC acquisition time window in seconds.

**Offline only****Resolution**

The resolution of the offline ADC model in bits. The resolution is applied over the voltage reference range. If the parameter is left blank, ADC quantization is not modeled.

**Voltage reference**

The voltage range of the offline ADC model used to determine the ADC resolution.

## Analog In (Triggered)

**Purpose** Measure the voltage on an analog input channel, synchronous to a trigger signal generated by a Timer, PWM or HRTIM

**Library** STM32

### Description



The Analog In (Triggered) block configures channels on an ADC submodule specified by the user. The single conversion mode is used in order to synchronize the ADC measurement conversion with counter overflow or underflow events created by Timer, PWM or HRTIM blocks. In order to give the channels priority over ADC channels configured using Analog In blocks, the Analog In (Triggered) block uses the injected conversion mode.

The Analog In (Triggered) block output signal represents the voltage measured on the input pin. The output can be scaled and an offset can be applied after scaling; it is calculated as  $\text{input} \times \text{Scale} + \text{Offset}$ . If the **Analog input channel(s)** parameter is vectorized, each input channel is measured sequentially in the order that it appears in the input channel vector.

By setting the **Trigger source** parameter to Show trigger port, users can define the trigger source by connecting a **Timer**, **PWM**, or **HRTIM** block to the trigger input. If **Trigger source** is set to Automatic, the Analog In (Triggered) block is triggered with the base task frequency. If the **ADC task** trigger output is connected to the **Control Task Trigger**, the control task will execute once the last ADC channel is converted.

Although up to four injected measurements are supported by the STM32 ADC, only up to three channels are allowed in PLECS. This is due to an ADC silicon bug (revision Y) that can result in invalid measurements for the first channel. As a consequence, the first measurement is always discarded and the block is limited to a maximum of three conversions.

### Parameters

#### Main

##### Trigger source

Selects an automatic or user-defined start-of-conversion trigger. If this parameter is set to automatic, the ADC will be triggered with the base task frequency.

##### ADC unit

Selects the peripheral index of the ADC submodule that is used.

**Analog input channel(s)**

Index of the analog input channel that is used. If this parameter is vectorized, each input channel is measured sequentially in the order that it appears in the input channel vector.

**Scale(s)**

A scale factor for the input signal.

**Offset(s)**

An offset for the scaled input signal.

**Acquisition time**

Selects between a minimal or user specified ADC acquisition time.

**Acquisition time value(s)**

Sets the ADC acquisition time window in seconds.

**Offline only****Resolution**

The resolution of the offline ADC model in bits. The resolution is applied over the voltage reference range. If the parameter is left blank, ADC quantization is not modeled.

**Voltage reference**

The voltage range of the offline ADC model used to determine the ADC resolution.

## Base Task Load

**Purpose** Provide the CPU load generated by the base task as a percentage of the base task period

**Library** STM32

**Description** This block outputs the percentage of time that is used by the base control task with one interrupt period. In case of multi-tasking, the output corresponds to the Base task load only, and does not include the load created by additional lower-priority tasks.



# CAN Port

**Purpose** Set up a CAN communication port

**Library** STM32

**Description** The block sets up a CAN (Controller Area Network) communication port.



The input **en** determines the CAN port state. Setting **en** to zero will force the CAN port to the *bus-off* state, while setting the port to 1 allows the CAN port to transition to *bus-on*. A *bus-off* condition has to be cleared by setting the enable signal to 0, and then back to 1.

The output **on** is 1 to signal *bus-on* status, 0 otherwise. The output **ea** is 1 to signal *error active* status, 0 otherwise.

## Parameters

### Main

#### CAN interface

Selects the CAN interface to use.

#### CAN protocol

Selects the CAN protocol to use. CAN FD supports bit rates higher than 1 Mbit/s and payloads larger than 8 bytes.

### GPIO

#### Tx port

Selects the port used to transmit data.

#### Tx pin

Selects the pin used to transmit data.

#### Rx port

Selects the port used to receive data.

#### Rx pin

Selects the pin used to receive data.



## Bit rate configuration

### Bit rate [bit/s]

Defines the bit rate that is used on the connected CAN bus. All devices on a CAN bus must be configured to use the same bit rate. This parameter is only available if the parameter **CAN protocol** is set to CAN 2.0.

### Bit sample point [%]

Defines the point in time where the bus level is read and interpreted as the value. This parameter is only available if the parameter **CAN protocol** is set to CAN 2.0.

### Bit rate switching

Enables or disables bit rate switching by setting this parameter to Enabled or Disabled respectively. This option is only available if **CAN protocol** is set to CAN FD.

### Nominal bit rate [bit/s]

Defines the bit rate during the nominal phase (also known as arbitration phase) that is used on the connected CAN bus.

### Nominal bit sample point [%]

Defines the point in time where the bus level is read and interpreted in the nominal phase.

### Data bit rate [bit/s]

Defines the bit rate during the data phase that is used on the connected CAN bus. This parameter is only accessible when **Bit rate switching** is Enabled.

### Data bit sample point [%]

Defines the point in time where the bus level is read and interpreted in the data phase. This parameter is only accessible when **Bit rate switching** is Enabled.

## Advanced

### Advanced configuration

If set to Enabled, advanced bit timing settings can be configured. When set to Disable the bit timing will be automatically deduced from the configured bit rate and sampling point. The following default configuration will be used in this case:

- **Nominal/Data rate bit length** is maximized to have as much as possible time quanta to construct a bit time. This will result in a low bit rate prescaler (BRP).

- **Nominal/Data SJW** is chosen to be as large as possible.
- **SSP offset** is set to be in the middle of the data bit:  $(1 + T_{seg1} + T_{seg2})/2$ .
- **SSP filter** is set to 0.

All following parameters are only available if the advanced configuration is set to Enable and CAN FD protocol is used.

**Nominal rate bit length [1+Tseg1+Tseg2]**

Defines the sum of all bit time segments during the nominal phase expressed in time quanta as  $1 + T_{seg1} + T_{seg2}$ .

**Nominal rate SJW [Tq]**

Synchronization jump width during nominal phase expressed in time quanta.

**Data rate bit length [1+Tseg1+Tseg2]**

Defines the sum of all bit time segments during the data phase expressed in time quanta as  $1 + T_{seg1} + T_{seg2}$ .

**Data rate SJW [Tq]**

Synchronization jump width during data phase expressed in time quanta.

**Secondary sampling point (SSP)**

If set to Enabled advanced secondary sampling point settings can be configured and the automatic transceiver delay compensation is enabled.

**SSP offset [Tq]**

Defines the secondary sampling point offset expressed in time quanta.

**SSP filter [Tq]**

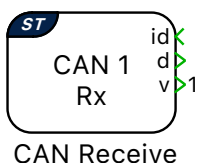
Defines the secondary sampling point filter expressed in time quanta.

## CAN Receive

**Purpose** Receive CAN messages

**Library** STM32

### Description



The block initiates the reception of CAN messages with a given identifier (ID) on the given CAN interface. On reception of a CAN message the data is made available on the block output **d** as a vectorized signal of the provided frame length. The output **v** is 1 for one simulation step when new data is received, 0 otherwise.

### Parameters

#### CAN interface

Selects the CAN interface to use. The selected CAN interface must be configured using a CAN Port (see page 37) block.

#### CAN ID source

Selects whether the CAN ID is specified as a parameter or is supplied as an input signal.

#### CAN ID

The ID for which the block receives CAN messages. The CAN ID can be supplied as either 11-bit value(s) (for CAN 2.0A) or a 29-bit value(s) (for CAN 2.0B).

#### Frame format

Specifies the frame format that is used when filtering for matching CAN messages. Possible values are:

- **Standard CAN** for CAN 2.0A messages with an 11-bit ID. The standard 11-bit ID provides for  $2^{11}$ , or 2048 different message identifiers.
- **Extended CAN** for CAN 2.0B messages with an 29-bit ID. The extended 29-bit ID provides for  $2^{29}$ , or 537 million identifiers.
- **Auto** uses the **Standard** format if the specified **CAN ID** is smaller than 2047. Otherwise, the **Extended** format is used.

#### Frame length

Specifies the frame length of the CAN message in bytes.

**Offline simulation**

Enables or disables data inspection in an offline simulation. If set to **enable**, terminals are added to the subsystem in the top-level schematic. If set to **disable**, no such terminals are added to the subsystem.

## CAN Status

**Purpose** Provide access to the CAN bus status

**Library** STM32

### Description



The block provides access to the error counter registers of a can interface. All nodes on a CAN bus detect errors and maintain two error counters: *Transmit Error Counter* and a *Receive Error Counter*. Each node can be in one of the following 3 error modes:

- **error active (1)** This is the start mode of all the nodes, when both error counters are less than 128. In this mode, a node fully participates in bus communication and transmits an active error flag when it detects errors.
- **error passive (2)** When one of the two error counters is greater than 127, a node goes into error passive mode. In this mode, a node still participates in bus activities, but transmits a passive error flag when it detects errors.
- **bus-off (3)** When the *Transmit Error Counter* is greater than 255, a node goes into a bus-off mode. When in this mode, the node is disconnected from the bus and can no longer participate in bus activities. If Auto bus-on is not enabled, a bus-off condition has to be cleared by setting the enable signal to 0, and then back to 1. After recovering from bus-off condition, both the error counters are reset to 0 and the node goes into error active mode.

The output **REC** provides the *Receive Error Counter*, the output **TEC** the *Transmit Error Counter*. The output **st** is 1 to signal *error active* status, is 2 to signal *error passive* status and 3 to signal *bus-off* status.

### Parameters

#### CAN interface

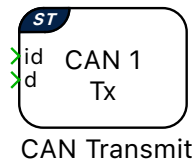
Selects the CAN interface to use. The selected CAN interface must be configured using a CAN Port (see page 37) block.

# CAN Transmit

**Purpose** Transmit CAN messages

**Library** STM32

## Description



The CAN Transmit block sends out data on a CAN bus. The data to send must be provided on the block input **d<sub>x</sub>** as a vectorized signal with data type **uint8**. The length of the transmitted CAN message is determined by the width of the input signal (1 to 8 bytes for CAN2.0 and 1 to 64 bytes for CAN FD).

Messages are either sent regularly with a fixed sample time or on demand when the trigger input changes. When configured for triggered execution, messages are sent when the trigger signal changes in the manner specified by the **Trigger type** parameter:

### rising

Data is sent when the trigger signal changes from 0 to a non-zero value.

### falling

Data is sent when the trigger signal changes from a non-zero value to 0.

### either

Data is sent when the trigger signal changes from 0 to a non-zero value or vice versa.

## Parameters

### CAN interface

Selects the CAN interface to use. The selected CAN interface must be configured using a CAN Port (see page 37) block.

### CAN ID source

Selects whether the CAN identifier (ID) is specified as a parameter or is supplied as an input signal.

### CAN ID

The ID for which the block transmits a CAN message. The CAN ID can be supplied as either 11-bit value(s) (for CAN 2.0A) or a 29-bit value(s) (for CAN 2.0B).

### Frame format

Specifies the frame format of the CAN messages to be transmitted. Possible values are:

- **Standard CAN** for CAN 2.0A messages with an 11-bit ID. The standard 11-bit ID provides for  $2^{11}$ , or 2048 different message identifiers.

- **Extended CAN** for CAN 2.0B messages with an 29-bit ID. The extended 29-bit ID provides for  $2^{29}$ , or 537 million identifiers.
- **Auto** uses the **Standard** format if the specified **CAN ID** is smaller than 2047. Otherwise, the **Extended** format is used.

### **Execution**

Selects between **regular** and **triggered** execution.

### **Trigger type**

The direction of the edges of the trigger signal upon which the data is sent, as described above (for triggered execution only).

### **Offline simulation**

Enables or disables data inspection in an offline simulation. If set to **enable**, terminals are added to the subsystem in the top-level schematic. If set to **disable**, no such terminals are added to the subsystem.

# Control Task Trigger

**Purpose** Specify the base sample time and trigger for the main control tasks

**Library** STM32

## Description



The digital control loop executes at a nominal base sample time. The input to the Control Task Trigger specifies the interrupt that triggers a control loop execution. The source of the interrupt can be from the ADC end-of-conversion signal, PWM counter underflow and overflow events, or the Timer block. When a Control Task Trigger is not included in the subsystem an appropriate trigger source is automatically determined.

In a multi-tasking mode (defined in the Scheduling tab of the Coder Options dialog), the Control Task Trigger block triggers the Base task associated with the base sample time.

The offline simulation will model the impact of controller discretization when the Control Task Trigger is included. For offline simulations the Forward Euler method with the nominal base sample time is used to integrate continuous states within the subsystem containing the Control Task Trigger. Offline simulations will use the default subsystem execution settings when the Control Task Trigger block is not included in the subsystem.

## Parameters

### Nominal base sample time

Specifies the nominal sample time of the discretized model in seconds. The nominal base sample time value is synchronized with the model **Discretization step size** of the PLECS Coder settings.



## DAC

### Purpose

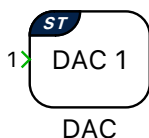
Generate an output voltage from the input signal; the output voltage is calculated as  $\text{input} \times \text{Scale} + \text{Offset}$

### Library

STM32

### Description

This block generates a voltage on the DAC pin in the range of 0 V to 3.3 V. The output is scalable and can be used with an offset, where the output signal is calculated as  $\text{input} \times \text{Scale} + \text{Offset}$ . Output voltage limitations can also be set.



### Parameters

#### DAC selection

Configure the DAC pin either as by `DAC unit` and `channel`, or by `Port` and `Pin number`.

#### DAC unit

Selects the desired DAC interface.

#### DAC channel

Index of the DAC output channel for a specific DAC interface.

#### Port

Selects the port name of the DAC channel.

#### Pin number

Defines the pin number of the DAC channel.

#### Scale

A scale factor for the output signal.

#### Offset

An offset for the scaled output signal.

#### Minimum output voltage

The lowest value that the output voltage can reach.

#### Maximum output voltage

The highest value that the output voltage can reach.

# Digital In

**Purpose** Read a digital input

**Library** STM32

## Description



Digital In

The output signal is 1 if the input voltage is higher than the high level input voltage threshold,  $V_{IH}$ , and 0 if it is lower than the low-level input voltage,  $V_{IL}$ . For other input voltages the output signal is undefined. Refer to the device data sheet for the electrical characteristics of a specific target. During an offline simulation the block behaves like a simple feedthrough.

## Parameters

### Port

Selects the port name of the digital input channel.

### Pin number(s)

Defines the pin number of the digital input channel. For vectorized input signals a vector of input channel indices must be specified.

### Input characteristic

Specifies whether an internal pull-up or pull-down resistor is connected to the digital input.

## Digital Out

**Purpose** Set a digital output

**Library** STM32

**Description** The output is set low if the input signal is zero and is set high for all other values. During an offline simulation the block behaves like a simple feedthrough.



Digital Out

### Parameters

#### Port

Selects the port name of the digital output channel.

#### Pin number(s)

Defines the pin number of the digital output channel. For vectorized output signals a vector of output channel indices must be specified.

#### Output characteristic

Specifies whether an internal pull-pull or open drain resistor is connected to the digital output.

# Edge Counter

**Purpose** Count edges of a pulse train

**Library** STM32

## Description

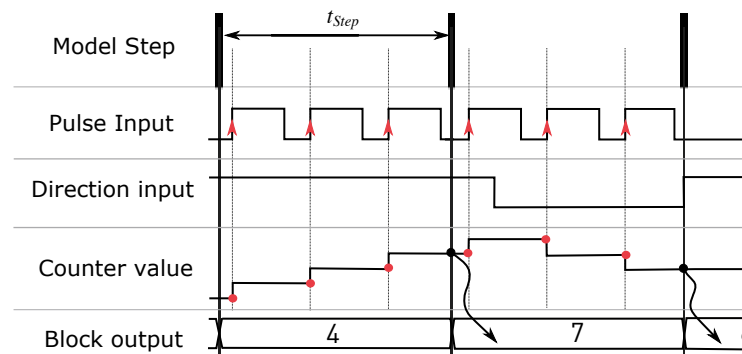


Edge Counter

The Edge Counter block counts edges of an external signal. It can be configured to count rising or falling edges only or both. The output of this block then shows the counter value at the time it was read. The counter value can be reset by an optional block input to the configured **Initial condition** value.

In the Single channel + Direction counting **Mode** the counter counts up or down depending on the signal level on the direction input pin. The counter is incremented if the signal level is logic 1, and decremented if the level is logic 0 at the direction pin.

An example is shown in the figure below. The counter is configured as Single channel + Direction, and reacts to the rising edge only. Every time a rising edge is detected, the counter value is incremented or decremented according to the direction pin signal level. Each task period the actual counter value is read and output at the block output terminal.



**Edge counter value in function of direction signal**

## Parameters

### Main

#### TIM unit

Selects the timer unit used. All timer units are 16-bit counters, except TIM unit 5 is a 32-bit counter.

### **Maximum counter value**

The counter is reset to zero when it has reached the Maximum counter value and detects an input edge in the positive direction. The counter is set to the Maximum counter value when it is zero and detects an input edge in the negative direction.

### **Mode**

Selects whether the counter should count in positive direction only (Single channel) or should change the counting direction based on an additional **Direction** signal (Single channel + Direction). The counter increments its value when the Direction signal is logic 1, and decrements when the direction signal is logic 0.

### **External reset**

The external reset selects the behaviour of the external reset input. The values rising, falling and either cause a reset of the counter to its **Initial condition** on the rising, falling or both edges of the reset signal. A rising edge is detected when the signal changes from 0 to a positive value, a falling edge is detected when the signal changes from a positive value to 0. If set to none the counter value cannot be reset by software.

### **Initial condition**

The initial condition allows to choose the start value of the counter. The initial condition is loaded at the beginning of a simulation or after a reset.

## **Channel**

### **Edge**

The edge detection trigger can be set on Rising, Falling or Either.

### **Port**

Selects the port name.

### **Pin number(s)**

Defines the pin number.

### **Input characteristic**

Specifies whether an internal pull-up or pull-down resistor is connected to the input pin. If high impedance is selected no internal pull resistor is connected.

**Direction****Port**

Selects the port name of the direction signal.

**Pin**

Defines the pin number of the direction signal.

**Input characteristic**

Specifies whether an internal pull-up or pull-down resistor is connected to the input pin. If high impedance is selected no internal pull resistor is connected.

**Offline only****System clock frequency (SYSCLK)[MHz]**

Defines the system clock frequency in MHz for offline simulations. For real-time simulations, the edge counter block uses the system clock frequency parameter specified in the Target tab of the Coder Options dialog.

## External Sync

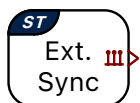
### Purpose

External synchronization port for HRTIM and PWM blocks

### Library

STM32

### Description



External Sync

The External Sync block allows to synchronize the start of connected HRTIM and PWM blocks based on an external signal. In addition, the External Sync block allows to reset the internal counter of the connected HRTIM and PWM block to zero when a rising or falling edge was detected on the external signal connect to the specified pin. The External Sync block can also be used to generate an external synchronization signal.

### Parameters

#### Synchronization

Specifies if the External Sync block should act as master or slave. If set to react to external SYNC signal (slave) the External Sync block reads in an external synchronization signal and allows synchronizing HRTIM and PWM blocks to an external signal. If set to generate external SYNC signal (master) the External Sync block generates a synchronization signal at the specified pin. This generated signal can be used on a different MCU board in order to synchronize the PWM signals together.

#### Synchronization behaviour (slave)

Specifies if the External Sync block should only start the connected HRTIM or PWM counter (Start on sync) or should also reset them (Start and reset on sync).

#### Synchronization behaviour (master)

If set to Output positive pulse the External Sync block will generate a synchronization pulse changing from 0V towards the positive voltage reference. If set to Output negative pulse the External Sync block will generate a falling edge synchronization pulse changing from the positive voltage reference towards 0V.

#### Port

Selects the port name.

#### Pin

Defines the pin number.

#### Input characteristic

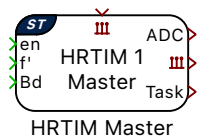
Specifies whether an internal pull-up or pull-down resistor is connected to the input. If high impedance is selected no internal pull resistor is connected.

# HRTIM Master

**Purpose** Synchronize multiple high resolution timer (HRTIM) timing units to generate frequency and/or phase variable PWM signals

**Library** STM32

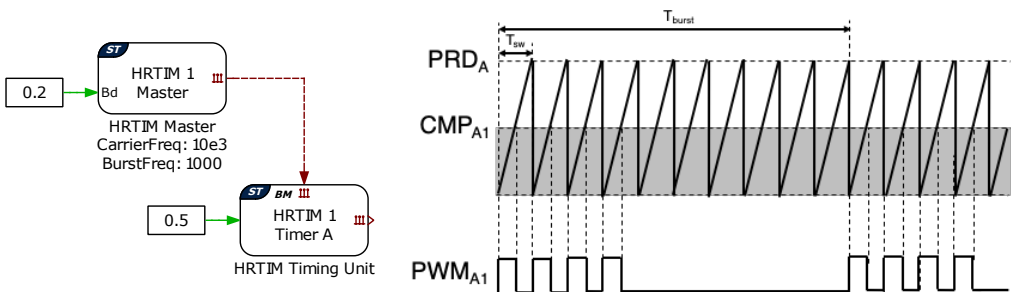
**Description** The HRTIM Master block allows the synchronization of multiple HRTIM timing units (see page 56). This synchronization enables the generation of frequency variable and/or phase shifted PWM output signals.



All synchronized HRTIM timing units will inherit the configured carrier frequency. The carrier frequency can be controlled by using the scalar input signal  $f'$ . The resulting carrier frequency  $f_c$  is calculated as the product of the nominal carrier frequency specified in the block parameter Carrier frequency [Hz] and the input signal  $f'$ .

An additional enable input terminal *en* can be activated. Applying a signal = 0 to this port will set all synchronized HRTIM timing units to their idle state.

The HRTIM Master block can also be used as burst mode controller. Burst mode operation is commonly used during light load conditions to increase the converter efficiency. It allows to have the output channels of synchronized HRTIM timing units alternatively in idle and run state, by hardware. Due to this, some switching periods can be skipped with configurable periodicity and variable duty cycle. The burst mode duty cycle can be set by the *Bd* input terminal and has to be a value between 0 and 1, as shown in the figure below.



**Burst mode controller with a burst mode duty cycle of 0.4.**



---

**Note** Keep in mind that the HRTIM prescaler is set during the initialization process of the MCU, and cannot be changed during operation. Therefore, in case of variable frequency operation, the `Carrier frequency [Hz]` parameter should be selected as the lowest frequency required during operation.

---

The HRTIM master block can configure interrupts to trigger the ADC start-of-conversion and the Control Task Trigger. Interrupts are synchronized with the PWM carrier, and the repetition counter period determines how many events need to occur before a trigger is generated. Trigger events will always occur at the period event (overflow) of the sawtooth carrier.

### Parameters

#### Main

##### HRTIM unit

Selects the HRTIM timer unit to use.

##### Carrier frequency [Hz]

Defines the frequency of the carrier in Hertz (Hz). The HRTIM master carrier is always of type sawtooth.

##### Frequency tolerance

Specifies the behavior when the desired carrier frequency is not achievable based on the system clock frequency.

##### Frequency variation

Enables or disables the frequency input port  $f'$ . The resulting output frequency of all synchronized HRTIM timing units is given by  $f'$  multiplied by the **Carrier frequency** parameter.

##### Enable port

An additional component port is created if **Enable port** is set to Show. Applying a signal = 0 to this port sets all synchronized HRTIM timing unit output channels to their idle state.

#### Trigger

##### ADC trigger

Enables or disables the ADC start-of-conversion trigger. The trigger event always occurs at the period event of the carrier.

**Task trigger**

Enables or disables the Control Task Trigger. The trigger event always occurs at the period event of the carrier.

**Repetition counter period**

Determines how many period events need to occur before the actual trigger is propagated.

**Burst Mode****Burst mode**

Enables or disabled the burst mode controller.

**Burst mode frequency [Hz]**

Defines the burst mode periodicity. This parameter is only visible if **Burst mode** is Enabled.

## HRTIM Timing Unit

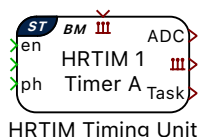
### Purpose

Generate up to two independent PWM signals or a complementary PWM signal pair

### Library

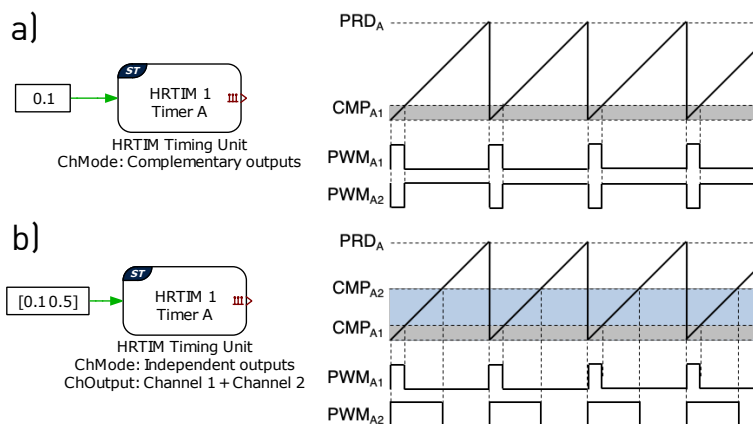
STM32

### Description



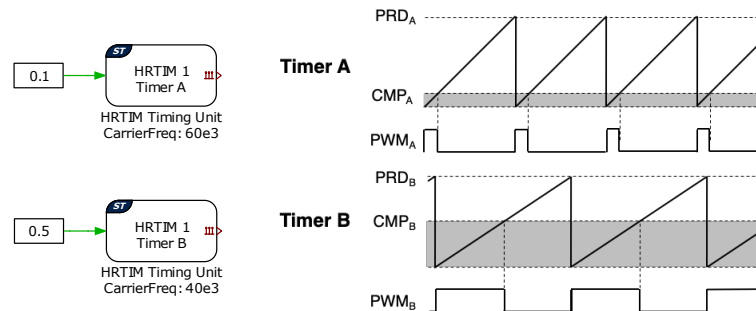
The high resolution timer (HRTIM) timing unit block generates up to two independent PWM signals or a complementary PWM pair with a configurable blanking time. The modulation index must be provided via the input signal, which is a vectorized signal if the block uses two independent output channels. The carrier starts at 0 and varies between 0 and 1. The PWM output is active when the input is greater than the carrier.

The following figure illustrates the difference between the two available output modes: complementary outputs (a) and independent outputs (b). Independent outputs share the same PWM carrier, but have different compare values. In complementary mode the second output channel is always complementary to the first channel and a blanking time for the rising and falling edge can be defined.



**HRTIM timing unit output modes: a) complementary PWM pair, b) two independent PWM outputs**

The figure below shows two HRTIM timing units (A and B) that are Self synchronized. Each HRTIM timing unit has its own carrier frequency and compare



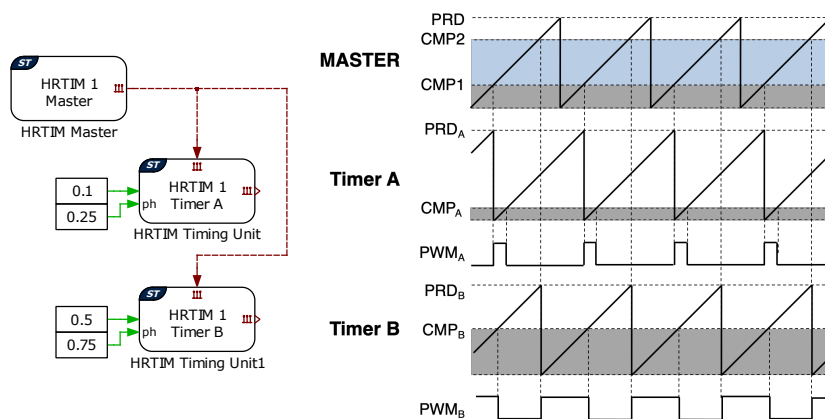
### Two independent HRTIM timing units (Self synchronized)

value. There is no synchronization signal between the two timing units and the relative position between the carriers is not deterministic.

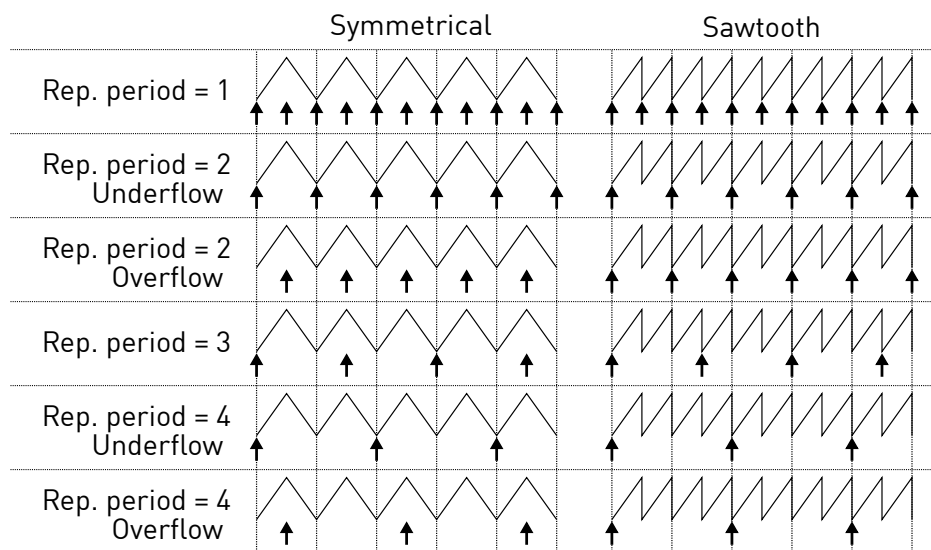
The following figure shows two HRTIM timing units (A and B) that are synchronized to a common HRTIM Master (see page 53) block. The HRTIM timing units A and B inherit the carrier frequency from the master timer. The master timer generates a synchronization signal for each of the two timing units and the relative position between the carriers of the timing units and the master timer is given by the *ph* input terminal. At each synchronization edge from the master the synchronized timing units are reset and the counter restarts at zero. Up to 4 timing units can be arbitrarily phase-shifted in respect to the master carrier. It is possible to synchronize additional timing units with zero phase shift to the same master timer.

The HRTIM timing unit block can configure interrupts to trigger the ADC start-of-conversion and the Control Task Trigger. Interrupts are synchronized with the PWM carrier, and the repetition counter period determines how many events need to occur before a trigger is generated, as shown in the figure below. When the carrier type is symmetrical, for even values of repetition counter period, the interrupts will occur at the carrier underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the respective carrier minimum or carrier maximum values. When the carrier type is sawtooth, the ADC trigger can occur at the period event or in the middle of the relative on-time of the PWM output. This allows to have ripple free ADC conversions. The Control Task trigger for a sawtooth carrier will always occur at the period event of the carrier.

The figure below shows an example of a symmetric PWM carrier with the repetition counter period set to 2, the trigger event set to overflow, and the polarity



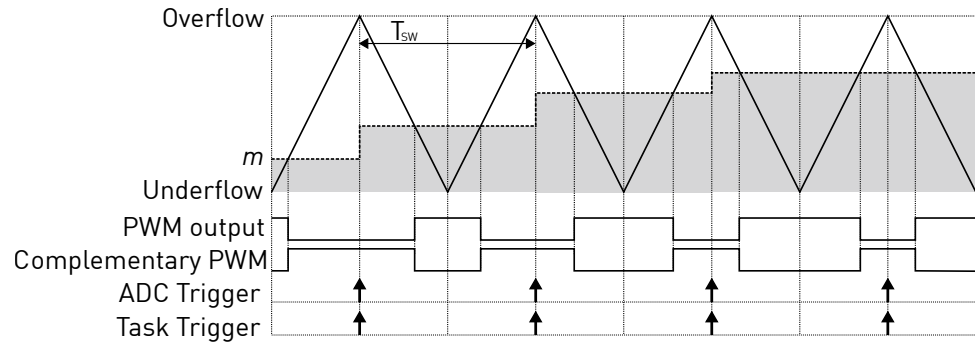
**Two HRTIM timing units synchronized to a HRTIM Master block**



**Trigger events based on the repetition counter period**

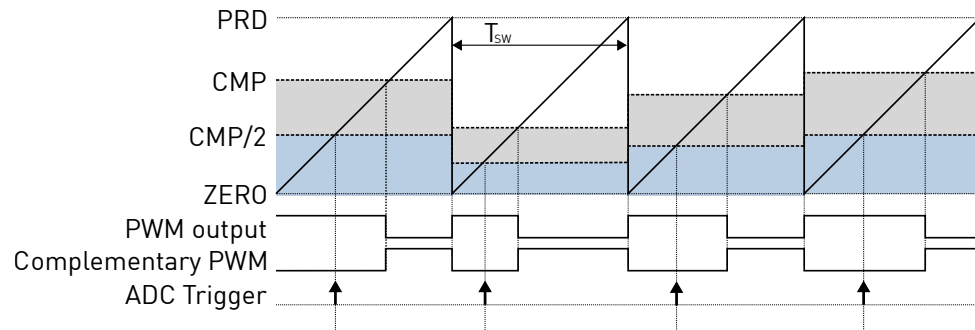
configured with an active state logic of '1'.

The following figure illustrates an example of a sawtooth PWM carrier with the repetition counter period set to 1. The ADC trigger event is set to Midpoint,



### PWM and trigger schemes for symmetric carrier

and the polarity configured with an active state logic of '1'. The sampling point is adjusted in such a way, that it will always occur in the middle of the relative on-time. This can lead to a varying base task period if the timing unit generates the Control Task trigger, especially during transient operation.



### Midpoint sampling for sawtooth carrier, the sampling point is continuously adjusted to be in the middle of the compare value

Configure the **Frequency variation** parameter to enable the frequency input port  $f'$ . The resulting carrier frequency  $f_c$  is calculated as the product of the nominal carrier frequency specified in the block parameter Carrier frequency [Hz] and the input signal  $f'$ .

---

**Note** Keep in mind that the HRTIM prescaler is set during the initialization process of the MCU, and cannot be changed during operation. Therefore, in case of variable frequency operation, the Carrier frequency [Hz] parameter should be selected as the lowest frequency required during operation.

---



---

**Note** If the task scheduler detects a CPU overflow and the model contains a Powerstage Protection (see page 69) target block, all HRTIM Timing unit outputs are switched to the passive state.

---

## Parameters

### Main

#### HRTIM unit

Selects the HRTIM unit to use. There are two HRTIM units available.

#### Timing unit

Selects the HRTIM timing unit of the output channel. Each timing unit can independently generate up to two independent PWM outputs or a complementary PWM pair. HRTIM timing unit F is not available on chips of the F3 target family.

#### Synchronization

If set to Self the HRTIM timing unit acts independent of all other HRTIM blocks. If set to Master a HRTIM Master (see page 53) block must provide a synchronization signal for the HRTIM timing unit. A synchronization to Master enables different features, e.g. phase shifting the HRTIM timing unit carrier or burst mode control. If set to External the HRTIM timing unit can be synchronized to an external event through a External sync (see page 52) block. The HRTIM timing unit will start and eventually reset when a rising edge on the external event signal is detected.

#### Carrier type

Selects the carrier waveform, either sawtooth or symmetrical.

#### Carrier frequency [Hz]

Defines the frequency of the carrier in Hertz (Hz). This option is only visible if the HRTIM timing unit is Self synchronized. If the timing unit is synchronized to Master the timing unit will inherit the carrier frequency of the HRTIM master.

**Frequency tolerance**

Specifies the behavior when the desired carrier frequency is not achievable based on the system clock frequency.

**Frequency variation**

Enables or disables the frequency input port  $f'$ . This option is only selectable if the **Synchronization** parameter is set to Self.

**Phase shift**

Enables or disables the phase shift input port  $ph$ . This option is only visible if the HRTIM timing unit is synchronized to Master.

**Enable port**

An additional component port is created if **Enable port** is set to Show. Applying a signal = 0 to this port sets the HRTIM timing unit outputs to the idle state.

**Channel****Mode**

Configures the output PWM channels to operate independently from each other or act as complementary signals. Each HRTIM unit can generate up to two independent PWM outputs or one complementary PWM pair.

**Output signals**

Defines which output channel should generate PWM signals. If Channel 1 + Channel 2 is selected, two duty cycle values and enable signals (if the **Enable port** is set to Show) are expected at the respective input terminals. The two independent PWM signals share the same carrier frequency and phase shift. This parameter is only visible if **Mode** is set to Independent outputs.

**Polarity**

Defines the logical output of the PWM output when an active state is detected.

**Blanking time [s]**

Delay between the rising and falling edges of a complementary PWM output pair in seconds (s). This parameter is only visible if **Mode** is set to Complementary outputs.

**Idle state**

Defines the level of the output signal when the output is in its idle state. An output channel enters the idle state if it is disabled due to an enable signal = 0, the **en** input of the Powerstage protection block or if it reacts to the



burst mode controller. Active and inactive levels change in function of **Polarity**.

### Trigger

#### ADC trigger

Enables or disables the ADC start-of-conversion trigger.

#### ADC trigger event

Selects the ADC trigger event. This parameter is only visible for a carrier of type sawtooth. When Midpoint is selected the ADC start-of-conversion is started in the middle of the relative on-time of the PWM signal. If set to Period the ADC trigger event occurs at the overflow event of the sawtooth carrier. Please note the the **Timing unit E** cannot generate a period trigger event.

#### Task trigger

Enables or disables the Control Task Trigger. For a sawtooth carrier the task trigger is generated at the period event of the counter.

#### Repetition counter period

Determines how many events need to occur before a trigger is generated.

#### Trigger event

Selects the trigger event. This parameter is only visible for a carrier of type symmetrical. When the carrier type is symmetrical, for even values of repetition counter period, the interrupts will occur at the carrier underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the respective carrier minimum or carrier maximum values.

### Burst Mode

#### Burst mode

If set to Enabled the HRTIM timing unit output channels will react to the burst mode control signal emitted by the HRTIM master block. If set to Disabled the timing unit will ignore the burst mode controller. This option is only selectable if **Synchronization** is set to Master.

### Protection

#### Channel fault state

Defines the level of the output signal when the output is in its fault state. An output channel enters the fault state if an **Analog protection signal**

trips. Active and inactive levels change in function of **Polarity**. If set to high impedance the outputs will be floating in a high impedance state.

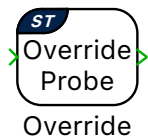
# Override Probe

**Purpose** Allow modifying input value during a PIL simulation

**Library** STM32

**Description** During a PLECS processor-in-the-loop (PIL) simulation, an Override Probe allows PLECS to overwrite variables in the embedded code.

For further details on the PIL simulation, refer to the PIL User Manual.

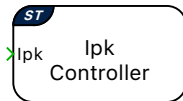


# Peak Current Controller

**Purpose** Implement peak current control with ramp compensation

**Library** STM32

**Description** The Peak Current Controller (PCC) block implements peak current control with slope compensation.



Peak Current Controller

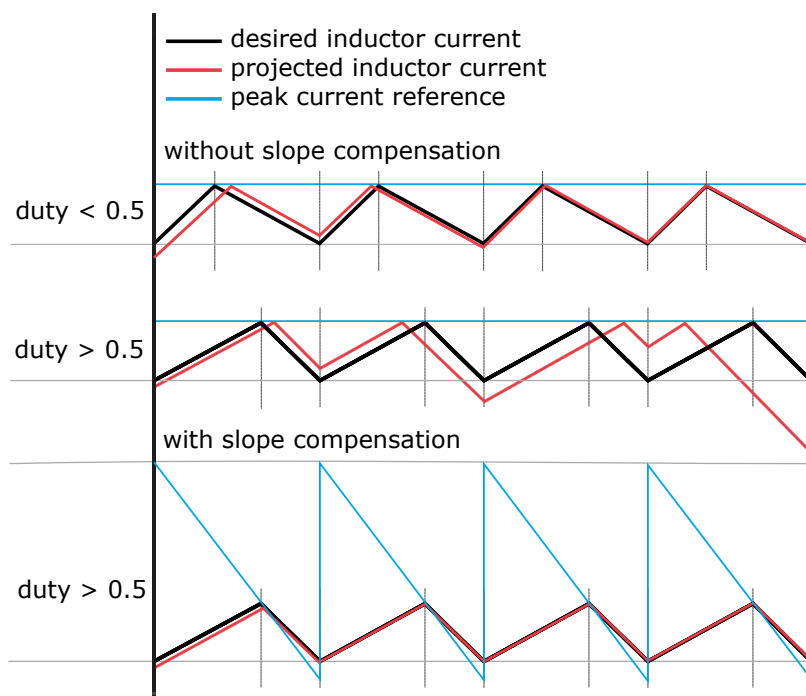
In a peak current-mode controller, at the beginning of each switching cycle the output is set (gate signal is turned ON) without a pre-determined duty cycle. Then, when the sensed inductor current exceeds the peak current reference value, the output is reset (gate signal is turned OFF). The duty cycle is therefore determined by the rise of the inductor current during the on-time.

One of the drawbacks of the peak current-mode controller is that it suffers from an inherent instability if the applied PWM duty cycle is greater than 50%. This is explained in the figure titled “Slope compensation”. If a small disturbance is introduced into the system and if the applied duty cycle is less than 50%, the disturbance eventually decays to zero. However, if the applied duty cycle is greater than 50%, the inductor current will start to diverge and will no longer be stable. The resulting duty cycle values will vary from small to large, on an alternating cycle basis, called sub-harmonic oscillations. To limit these sub-harmonic oscillations, instead of providing a constant peak current reference, additional slope compensation is applied, which then ensures the stability of the inductor current.

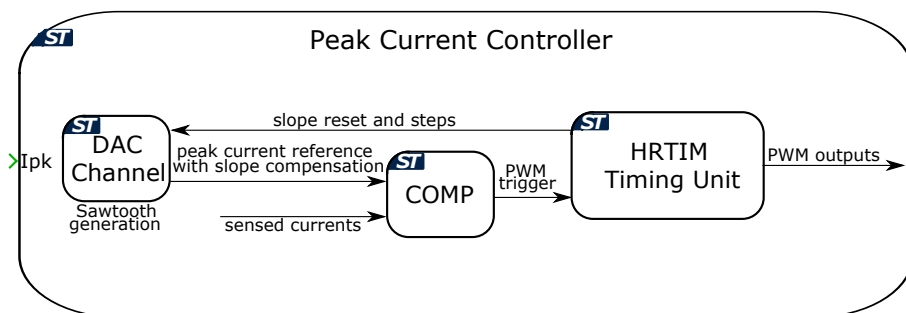
Internally, the PCC block makes use of multiple MCU peripherals. The first component is a DAC that provides a peak current set-point including ramp, for controlling the inductor current. The second is a comparator (COMP); the sensed current is fed to the comparator, which is then compared to the peak current set-point provided by the DAC. The output of the COMP block is fed to the third component, which is the high-resolution timer (HRTIM). The HRTIM generates the PWM waveforms and synchronizes the DAC at the specified frequency.

The peak current reference ( $I_{pk}$ ) is provided as an input to the PCC block. Logic is included in the PCC block such that when  $I_{pk} = 0$ , all associated PWM outputs are driven into the passive state.

The Peak Current Controller block can also be used as burst mode controller. Burst mode operation is commonly used during light load conditions to increase the converter efficiency. It allows to have the output channels alternatively in idle and run state, by hardware. Due to this, some switching periods can be



## Slope compensation



## Peak current controller schematic

skipped with configurable periodicity and variable duty cycle. The burst mode duty cycle can be set by the *Bd* input terminal and has to be a value between 0 and 1.

## Parameters

### Main

#### High resolution timer unit

Selects the high resolution timer (HRTIM) unit to use. There are two HRTIM units available. Each HRTIM unit can independently generate a single PWM output or a complementary PWM pair on up to three PWM channels.

#### Switching frequency

Defines the switching frequency of the output signal in Hertz (Hz).

#### Frequency tolerance

Specifies the behavior when the desired carrier frequency is not achievable based on the system clock frequency.

#### Current sense scale

Scales the peak current reference ( $I_{pk}$ ) into values with physical units to be used for the control algorithm.

#### Ramp slope

Defines ramp slope rate in Amperes per second (A/s). Slope compensation can be applied to ensure stability when the output duty cycle exceeds 50%. Entering a parameter,  $I_{ramp}$ , reduces  $I_{pk}$  during each switching cycle as follows:  $I'_{pk} = I_{pk} - I_{ramp} \cdot t$ , where  $t$  is the time elapsed from the start of the switching cycle. Slope compensation can be omitted by setting  $I_{ramp}$  to 0. On chips of the F3 target family, slope compensation is not supported and has to be omitted by setting the ramp slope to zero.

#### Ramp offset

Defines ramp offset in Amperes (A).

#### Leading edge blanking time

This sets the minimum output on time at the beginning of each switching period in seconds (s). Leading edge blanking time is used to prevent the turn-on transient current from triggering the peak current controller.

### Channel

#### Mode

Configures the output PWM channel. Each HRTIM unit can independently generate a single PWM output or a complementary PWM pair on up to three PWM channels.

**Sense port**

Selects the port of the sensed current.

**Sense pin**

Configures the pin of the sensed current.

**Timing unit**

Selects the HRTIM unit of the output channel. HRTIM timing unit F is not available on chips of the F3 target family.

**Output Polarity**

Defines the logical output of the PWM output when an active state is detected. The active state occurs when the sensed current exceeds the peak reference current set-point.

**Switching blanking time**

Specifies the delay between the rising and falling edges of a complementary PWM output pair in seconds (s).

**Burst mode****Burst mode**

Enables or disabled the burst mode controller.

**Burst mode frequency [Hz]**

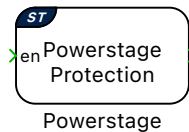
Defines the burst mode periodicity. This parameter is only visible if **Burst mode** is Enabled.

# Powerstage Protection

**Purpose** Provide powerstage safety features

**Library** STM32

## Description



The Powerstage Protection block implements an interlock, which is a safety mechanism, to enable or disable all the PWM outputs on the target device. The PWM outputs are disabled unless there is a logical low to high transition on the input signal, labeled **en**. This prevents the PWM signals from becoming active as soon as the code is executed on the target, thereby ensuring safe operation.

Additionally, there is an option to configure a digital output as a powerstage enable signal. This signal can then be used, for example, to provide an enable signal to external gate driver chips. The **enable polarity** of the digital output, specified in the **Powerstage enable pin** can be defined as:

- **Active low:** a logical low to high transition on the input signal, **en**, sets the digital output pin to logic low (0).
- **Active high:** a logical low to high transition on the input signal, **en**, sets the digital output pin to logic high (1).

To reiterate, the powerstage enable signal (GPIO out) is an output signal of the Powerstage Protection block. This signal does not contribute to enabling or disabling PWM outputs, and can be considered as a status indicator of the Powerstage Protection interlock state. Irrespective of the configuration of this signal (Digital output or None), the Powerstage Protection block, if included in the schematic, disables all the PWM outputs on the target device, unless there is a logical low to high transition on the input signal, labeled **en**.

When the PWM outputs are disabled via the Powerstage Protection input, all associated PWM outputs are driven into the passive state. If the task scheduler detects a CPU overflow and the model contains a Powerstage Protection target block, all PWM/HRTIM outputs are switched to the passive state as well.

If the Powerstage Protection block is omitted from the schematic, then all PWM outputs will be continuously enabled.

## Protection

**Analog protection signals** Each analog protection signal can be configured in the **Protections** tab of the **Coder + Coder Options + Target** window to



emit up to seven specific trip signals (labeled 1-7). When a trip signal is detected the Powerstage Protection module can take no action (Ignore) or activate a one-shot trip event (Disable powerstage (one-shot)). A one-shot trip event will disable the powerstage by setting the outputs of all PWM blocks to the inactive state and HRTIM outputs to the state defined in the channel fault state parameter of the HRTIM timing unit block (see page 56)). All trip events are latched and can only be cleared by cycling the Powerstage Protection block from *disabled* to *enabled*.

## Timing

When the Powerstage Protection **en** input is activated, the digital output (GPIO out) associated with the block (if configured) will immediately become active to e.g. enable a gate driver chip. However, the actual PWM signals will remain disabled for 100 ms to allow the gate driver circuit to stabilize. During this period the signal output of the Powerstage Protection block will remain low. Only at the end of the 100 ms delay will the signal output transition to high, simultaneously with the PWM signals becoming active. Therefore, a regulator reset or anti-windup mechanism must be controlled by the output signal of the Powerstage Protection block (signal out), and not the **en** input.

## Parameters

### Main

#### Powerstage enable signal

Provides an option to configure a digital output as a powerstage enable signal.

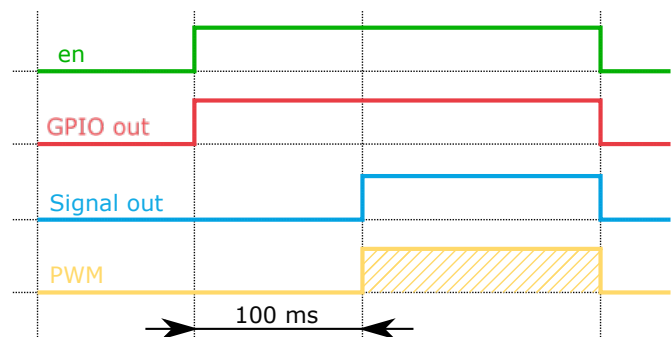
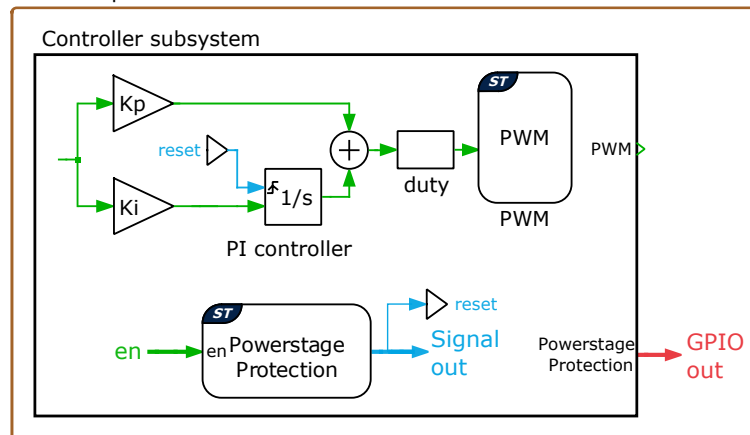
- **Digital output:** Configures a digital output as a powerstage enable signal. This signal can then be used, for example, to provide an enable signal to external gate driver chips. This signal can be considered as a status indicator of the Powerstage Protection interlock state.
- **None:** Powerstage enable signal is not configured.

#### Powerstage enable polarity

Defines the polarity of the powerstage enable signal.

- **Active low:** a logical low to high transition on the input signal, **en**, sets the GPIO pin to logic low (0).
- **Active high:** a logical low to high transition on the input signal, **en**, sets the GPIO pin to logic high (1).

PLECS top-level schematic



### Powerstage Protection timing

#### Powerstage enable port

Selects the digital output port to be configured as the powerstage enable signal.

#### Powerstage enable pin

Defines the digital output pin to be configured as the powerstage enable signal.

### Protection

#### Reaction to analog protection signal 1-7

Selects the action following an analog trip event detection. The Powerstage

Protection can take no action (Ignore) or activate a one-shot trip event (Disable powerstage (one-shot)) as reaction to an analog protection signal.

## Offline only

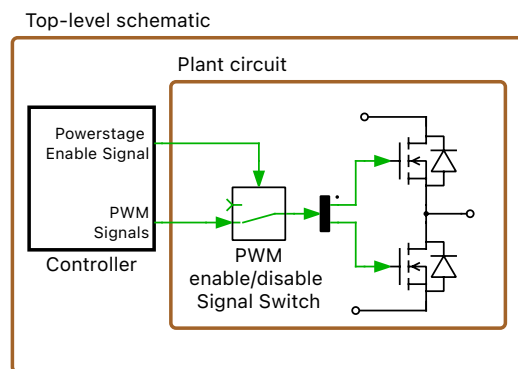
### Interlock

For convenience, the interlock can be enabled or disabled for offline simulations.

- Select **Simulate** to enable the simulation of the interlock safety mechanism. The output of the the Powerstage Protection block in the top-level schematic is disabled unless there is a logical low to high transition on the input **en**.
- Select **Do not Simulate** to disable the simulation of the interlock mechanism. The output of the the Powerstage Protection block can then be enabled at the start of the simulation by tying **en** to 1.

**Note** that the offline model of the Powerstage Protection block only simulates the powerstage interlock mechanism, in conjunction with the block output and the **Powerstage enable signal** digital output.

The effect that the Powerstage Protection block has on the individual PWM signals is not represented in an offline simulation, and needs to be separately implemented by the user. An example is shown in the figure below. The **Powerstage enable signal** digital output is wired to a PWM enable/disable Signal Switch, thereby simulating the enable signal of a gate driver circuit.



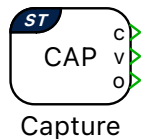
**An offline implementation of the gate driver logic of the Powerstage Protection block**

# Pulse Capture

**Purpose** Time-stamp edges of a pulse train

**Library** STM32

**Description**



The capture blocks allows timestamping signal transitions (events) on input pins, e.g. for period and/or duty cycle measurements. The timestamps are made available on the block output **c**.

The output **v** is 1 for one simulation step after an event has been triggered, 0 otherwise. The output **o** is set to 1 if more than one capture event happened during the last model step. This indicates an overcapture state. It is not possible to deduce how many times the event was capture in the last model step. Overcapture events should therefore be avoided to ensure a reliable pulse detection. The output **o** is automatically cleared after it has been read.

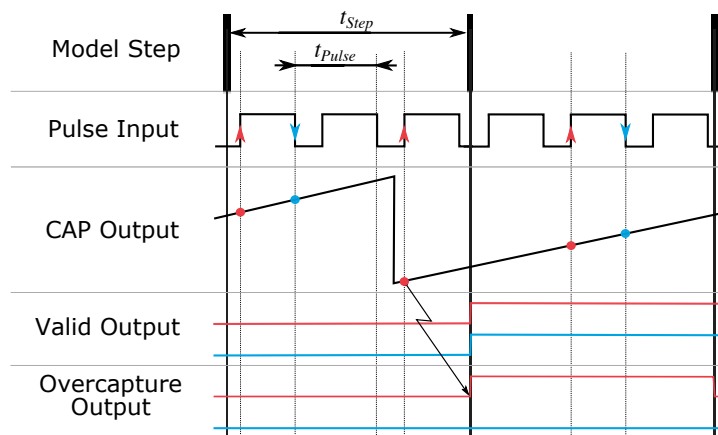
The pulse capture block can be used in either PWM capture **Mode** or as Individual channels.

In PWM capture mode, one channel of the selected **TIM unit** can be used to get information about the period and duty cycle of the captured PWM signal. The first signal of the **c** output reflects the measured period, the second signal of the **c** output can be used to calculate the duty cycle.

When configured as Individual channels, up to three channels can be used to capture edges on individual signals. Also, the outputs **c**, **v** and **o** are vectorized signals if the block uses multiple channels.

The figure below illustrates with an example how the different output signals behave if **Mode** is set to Individual channels. The channels 1 and 2 are configured as described in the following table:

Channel	Edge	Prescale
1	Rising	every 2nd event
2	Falling	every 4th event



**An illustrative example for different capture settings when the Pulse Capture block is used with individual channels (channel 1 in red, channel 2 in blue).**

If multiple capture events happen inside a model step the overcapture output will output a 1 at the end of the respective model step. The counter is operated in free-running mode and therefore not reset by a capture event. To calculate the period of the pulse input signal one has to calculate the difference between two consecutive capture events.

## Parameters

### Main

#### TIM unit

Selects the timer unit to use.

#### Mode

Specifies the desired capture mode. If set to Individual channels the channels of the selected **TIM unit** can be used to capture up to three individual signals. The PWM capture mode facilitates the measurement of frequency and duty cycle. In **PWM capture** mode exactly one Channel has to be enabled.

#### Clock prescale

Specifies the desired prescale value.

## **Channel 1/2/3**

### **Capture**

Provides an option to **disable** or **enable** an input channel.

### **Edge**

The edge detection trigger can be set on **Rising**, **Falling** or **Either**.

### **Prescale**

Specifies the desired prescale value for an individual channel.

### **Port**

Selects the port name of the capture channel.

### **Pin**

Defines the pin number of the capture channel.

## **Offline only**

### **System clock frequency (SYSCLK)[MHz]**

Defines the system clock frequency in MHz for offline simulations. For real-time simulations, the capture block uses the system clock frequency parameter specified in the Target tab of the Coder Options dialog.

## PWM

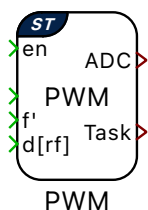
### Purpose

Generate a complementary PWM signal pair

### Library

STM32

### Description



The PWM block generates a single or complementary PWM pair on one or more PWM resources. The modulation index for each channel must be provided via the input signal, which is a vectorized signal if the block uses multiple channels. The carrier starts at 0 and varies between 0 and 1. The PWM output is active when the input is greater than the carrier. The carrier frequency can be controlled by using the optional scalar input signal  $f'$ . The resulting carrier frequency  $f_c$  is calculated as the product of the nominal carrier frequency specified in the block parameter Carrier frequency [Hz] and the input signal  $f'$ . Note that all PWMs channels within one PWM block share the same frequency input.

The start of all PWM blocks is synchronized based on the same hardware trigger signal.

The PWM block can configure interrupts to trigger the ADC start-of-conversion and the Control Task Trigger. Interrupts are synchronized with the PWM carrier, and the repetition counter period determines how many events need to occur before a trigger is generated, as shown in the figure below. When the carrier type is symmetrical, for even values of repetition counter period, the interrupts will occur at the carrier underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the respective carrier minimum or carrier maximum values.

The figure below shows an example of a symmetric PWM carrier with the repetition counter period set to 2, the trigger event set to underflow, and the polarity configured with an active state logic of '1'.

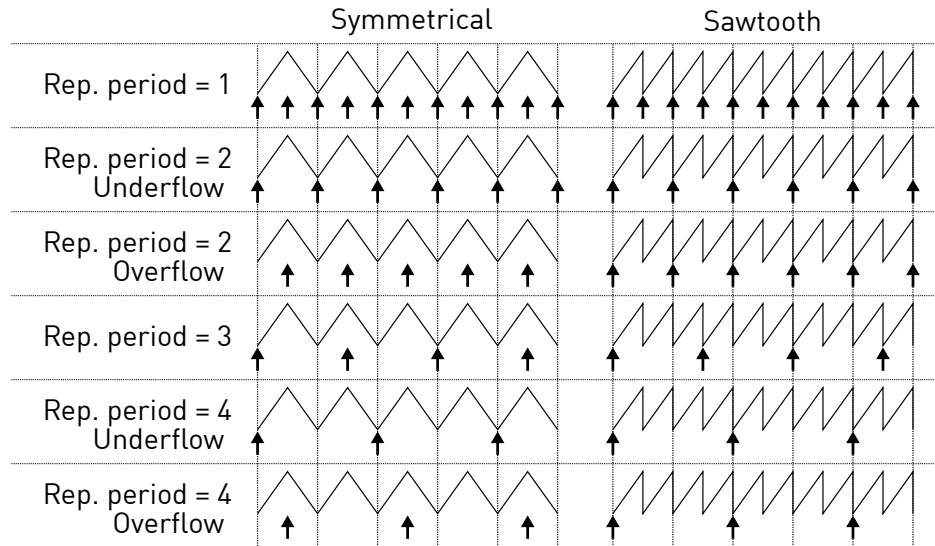
The PWM block protection parameters configure the break input settings for all PWM resources associated with the block. When a break input event is detected, all the PWM outputs are shutdown and are forced into the passive state.

---

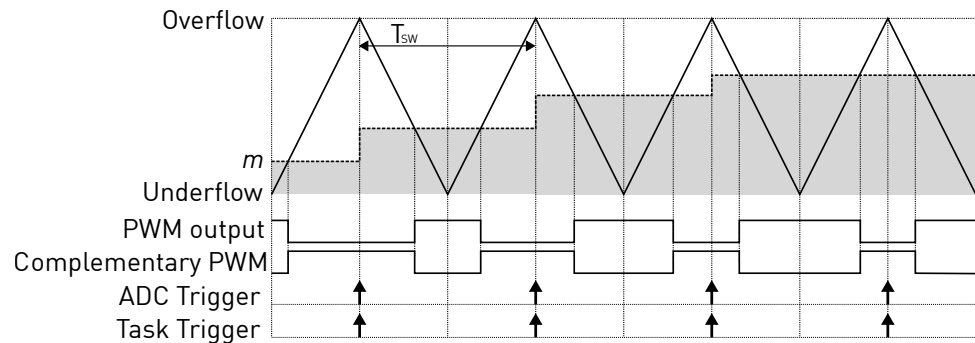
**Note** If the task scheduler detects a CPU overflow and the model contains a Powerstage Protection (see page 69) target block, all PWM outputs are switched to the passive state.

---

### Parameters



### Trigger events based on the repetition counter period



### PWM and trigger schemes for symmetric carrier

#### Main

#### TIM unit

Selects the advanced-control timer unit to use. There are two TIM units available, TIM1, TIM8 or TIM20. Each TIM unit can independently generate a single PWM output or a complementary PWM pair on up to three PWM



channels.

**Carrier type**

Selects the carrier waveform, either sawtooth or symmetrical.

**Carrier frequency**

Defines the frequency of the carrier in Hertz (Hz).

**Frequency tolerance**

Specifies the behavior when the desired carrier frequency is not achievable based on the system clock frequency.

**Blanking time variation**

If enabled, an additional input port **d[r,f]** is displayed to allow variation of the rising-edge and falling-edge delays. This port expects a vector of size 2 to set equal rising and falling delays for all channels. The units of the **d[r,f]** inputs are seconds.

**Blanking time [s]**

Delay between the rising and falling edges of a complementary PWM output pair in seconds (s).

**Minimal blanking time [s]**

The parameter specifies the minimal blanking time when variable blanking time is enabled.

**Enable port**

An additional component port is created if **Enable port** is set to value different than Hide. If set to Use one 'En' signal per Channel a single 'En' signal per channel is used and disables both PWM outputs of a channel. If set to Use one 'En' signal per PWM output an enable signal for each PWM output has to be provided. In consequence, if a PWM channel generates a complementary signal pair the user needs to provide two 'En' signals to disable each PWM output of the same channel independently. Applying a signal = 0 to this port sets the PWM channel(s) or PWM outputs to passive.

**Frequency variation**

Enables or disables the frequency input port  $f'$ . The resulting carrier frequency  $f_c$  is calculated as the product of the Carrier frequency [Hz] and the input signal  $f'$ .

**Channel****Mode**

Configures the output PWM channel. Each TIM unit can independently generate a single PWM output or a complementary PWM pair on up to four

PWM channels.

**Polarity**

Defines the logical output of the PWM output when an active state is detected. The active state occurs when the modulation index exceeds the carrier.

**Port**

Selects the port of the PWM output.

**Pin**

Configures the pin of the PWM output.

**Complementary Port**

Selects the port of the complementary PWM output.

**Complementary Pin**

Configures the pin of the complementary PWM output.

**Sync****Synchronization**

Specifies if the start and reset of the PWM counter should be synchronized to an external event through a External sync (see page 52) block. If set to External a synchronization port will show up and can be connected to a External sync (see page 52) block.

**Trigger type**

Specifies if the PWM block should react to a rising or falling edge of the external event. This parameter is only visible if the **Synchronization** parameter is set to External.

**Trigger****ADC trigger**

Enables or disables the ADC start-of-conversion trigger.

**Task trigger**

Enables or disables the Control Task Trigger.

**Repetition counter period**

Determines how many events need to occur before a trigger is generated.

**Trigger event**

Selects the trigger event. When the carrier type is symmetrical, for even values of repetition counter period, the interrupts will occur at the carrier

underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the respective carrier minimum or carrier maximum values.

### Protection

#### Powerstage protection

Determines whether or not a Powerstage Protection block in the same PLECS model can force the PWM outputs into the passive state.

#### Break input

Provides an option to configure a digital input to force shutdown all the PWM outputs associated with the block and drive them into the passive state.

- **Disabled:** Break input is not configured.
- **Active low:** A low voltage level on the input pin will force the PWM outputs into the passive state. If the input voltage transitions from the low level to the high level, the PWM output signal will resume.
- **Active high:** A high voltage level on the input pin will force the PWM outputs into the passive state. If the input voltage transitions from the high level to the low level, the PWM output signal will resume.

#### Break input port

Selects the digital input port to be configured as the break input.

#### Break input pin

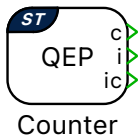
Configures the digital input pin to be configured as the break input.

# Quadrature Encoder Counter (QEP)

**Purpose** Counts edges generated by a quadrature encoder

**Library** STM32

## Description



The Quadrature Encoder Counter counts edges which are generated from a quadrature encoder. The *A*, *B*, and *I* outputs of the encoder are connected to the input GPIOs of the microcontroller.

The block output **c** represents the current counter value, port **i** represents the index pulse, and port **ic** outputs the latched counter value from the previous index pulse.

The counter counts up or down depending on the sequence of input pulses. The counter value will increase when the direction of rotation results in the rising edge of *B* following the rising edge of *A* and will decrease in the opposite direction of rotation. For each rising and falling edge of the *A* and *B* encoder output signals the counter will increment or decrement. Therefore the **Maximum counter value** must match the number of line pairs of the encoder multiplied by the number of counted edges per line pair minus 1. As an example, an encoder with 1024 line pairs would have a maximum count of 4095 since the QEP module counts all edges of *A* and *B* (4 edges per line pair).

Once the counter reaches the value specified in parameter **Maximum counter value** it is reset to zero on the next detected edge in the positive direction. Vice versa, the counter is set to **Maximum counter value** when it is zero and detects an edge in the negative direction. If connected and configured by the **Mode** parameter, the counter is also reset when the rising edge of the index input is detected.

## Parameters

### Main

#### TIM unit

Selects the timer unit used.

#### Maximum counter value

The counter is reset to zero when it has reached the **Maximum counter value** and detects an input edge in the positive direction. The counter is set to the Maximum counter value when it is zero and detects an input edge in the negative direction.

### **Mode**

Selects whether the counter should be reset by a positive pulse on the index input or on overflow only. Three different modes are available:

- **Free running:** The counter is reset on overflow only. Output ports **i** and **ic** are not available, and the output **I** of the encoder is not required.
- **Free running with index capture:** The counter is reset on overflow only. In addition, the index pulse is captured and the latched counter value from the previous index pulse is made available at the output port **ic**.
- **Reset by index pulse:** The counter is reset by a positive pulse on the index input.

### **Channel A**

#### **Port**

Selects the port of the channel A input.

#### **Pin**

Configures the pin of the channel A input.

#### **Polarity**

Selects if the Rising edge or Falling edge defines the start of a pulse.

#### **Input characteristic**

Specifies whether an internal pull-up or pull-down resistor is connected to the digital input. If high impedance is selected no internal pull resistor is connected.

### **Channel B**

#### **Port**

Selects the port of the channel B input.

#### **Pin**

Configures the pin of the channel B input.

#### **Polarity**

Selects if the Rising edge or Falling edge defines the start of a pulse.

#### **Input characteristic**

Specifies whether an internal pull-up or pull-down resistor is connected to the digital input. If high impedance is selected no internal pull resistor is connected.

**Index****Port**

Selects the port of the index pulse input.

**Pin**

Configures the pin of the index pulse input.

**Polarity**

Selects if the Rising edge or Falling edge defines the start of an index pulse.

**Input characteristic**

Specifies whether an internal pull-up or pull-down resistor is connected to the digital input. If high impedance is selected no internal pull resistor is connected.

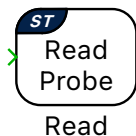
# Read Probe

**Purpose** Provide read access to signal during a PIL simulation

**Library** STM32

**Description** During a PLECS processor-in-the-loop (PIL) simulation, a Read Probe allows PLECS to read variables in the embedded code.

For further details on the PIL simulation, refer to the PIL User Manual.



# SinCos

## Purpose

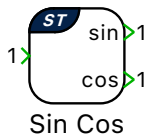
Fast calculation of sine and cosine

## Library

STM32

## Description

Utilizes the CORDIC co-processor for fast calculation of sine and cosine for chips from the G4 target family. For the F3 target family a look-up table based implementation is used to calculate sine and cosine efficiently.



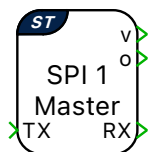


## SPI Master

**Purpose** Implement SPI Master connected to one or multiple slaves

**Library** STM32

### Description



SPI Master

The Serial Peripheral Interface (SPI) is a high-speed synchronous serial input/output device that allows a serial bit stream of programmable length (4 to 16 bits) to be shifted into and out of the device at a configurable bit-transfer rate. The SPI is usually used for communications between the MCU controller and external peripherals, or another controller.

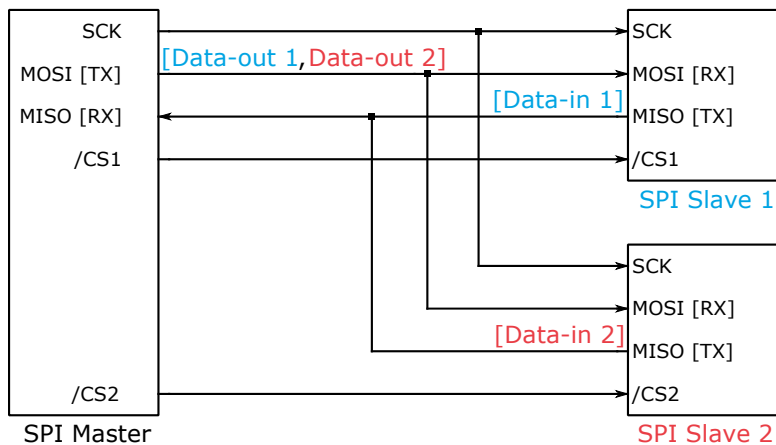
The SPI is a master-slave based interface with a single master and one or more slave devices.

The interface consists of the following signals:

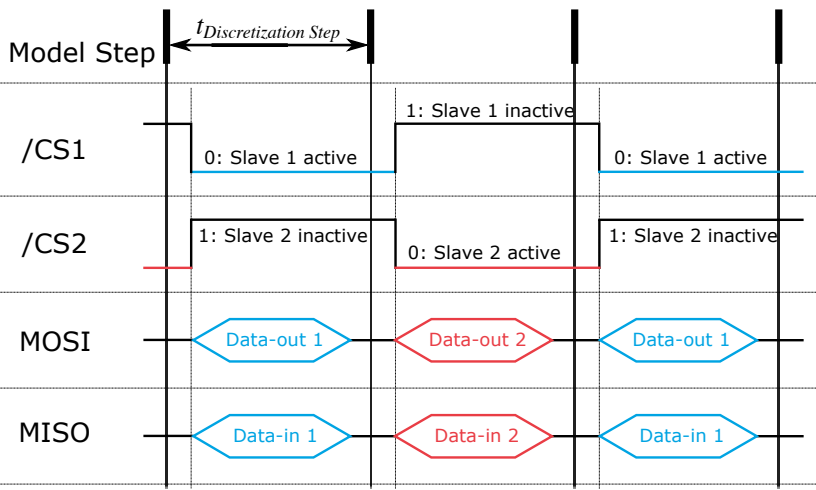
- **MISO** Serial data input (master in/slave out)
- **MOSI** Serial data output (master out/slave in)
- **SCK** Shift-clock, generated by the SPI Master
- **/CS** Chip-select or slave-select signal. The chip-select signal is an active-low signal that enables the MISO and MOSI ports of the SPI Slave.

The SPI Master block provides a clock signal (SCK) which generates a configurable number of clock pulses during each simulation step. For both the slave and the master, data is shifted out of the shift registers on one edge (rising or falling) of the CLK and latched into the shift register on the opposite clock edge. If the clock phase (CPHA) bit is configured to 1, data is transmitted and received a half-cycle before the SCK transition.

Multiple SPI Slaves can be supported by a single master through chip-select (/CS) signals. The figure below shows an example of a single SPI Master with two SPI Slaves.



**SPI Master with two SPI Slaves**



**An example of signal exchange between one SPI Master and two SPI Slaves**

The SPI Master block exchanges data with only one SPI Slave per block execution step. If there are multiple slaves, then data is exchanged over multiple steps. For example, as illustrated in the figure above, during the first step, the

master enables SPI Slave 1 using the chip-select signal and transmits Data-out 1; at the same time the master also receives Data-in 1 from the first slave. During the second step, after enabling SPI Slave 2, the master transmits Data-out 2 and receives Data-in 2 from the second slave simultaneously. This process then repeats.

An output value of 1 at the **v** port indicates that valid data is sent to all the slaves.

The data to be transmitted is provided at the input **TX** and the data received is available at the output **RX**.

- **TX**: For transmitting data to multiple slaves, provide a vector with a length equal to the sum of the number of words per transmission per each slave. For example, in the figures above, if Data-out 1 is a packet of 4 words [1,2,3,4] and Data-out 2 is a packet of 3 words [16,17,18], then the input to the TX block is provided as a vector of 7 words [1,2,3,4,16,17,18].
- **RX**: Similarly, data from multiple slaves is received as a vector with a length equal to the sum of the number of words per transmission per each slave. For example, in the figures above, if Data-in 1 is a packet of 4 words [21,22,23,24] and Data-in 2 is a packet of 3 words [36,37,38], then the output of the RX block is read as a vector of 7 words [21,22,23,24,36,37,38].

If the SPI Master does not have enough time to complete the transmission before the block is executed again, the output **o** turns 1 to indicate an overrun error.

If an overrun error is being signaled at the **o** port of the SPI Master, it is possible that the task with which the SPI Master is associated executes too fast. In this case, either reduce the SPI Master execution task rate or increase the SPI clock rate.

For example, if SCK is set as 180000 Hz, and is expected to transmit a packet of 4 words at 8 bits per word, then the time it would take to transmit one packet is

$$\frac{1}{180000} \cdot 4 \cdot 8 = 1.78 \cdot 10^{-4} \text{ seconds.}$$

In this case, the execution step size of the SPI Master must be set to values greater than 0.178 milliseconds.

## Parameters

### Setup

#### SPI module

Selects the SPI module to use.

**SPI clock [Hz]**

Defines the SPI clock frequency (SCK), also known as the SPI Baud Rate, in Hz. If the SPI clock frequency cannot be hit accurately, a warning is issued during code generation. Please refer to the STM32 technical reference for more information on the range of achievable SPI clock rates.

**Bits per word (4-16)**

Defines the length of a single data word during transmission. The allowed length is 4 to 16 bits per word.

**Mode [CPOL, CPHA]**

Defines four SPI clocking modes, controlled by clock polarity (CPOL) and clock phase (CPHA) bits. CPOL controls whether the clock signal is high (1) or low (0) when idle. CPHA controls whether data is shifted in and out on the rising or falling edge of the clock signal. The following table summarizes the clocking schemes:

**SPI Clocking Modes**

Mode	CPOL	CPHA	SPI Clock Scheme
SPI_MODE0	0	0	Rising edge without delay
SPI_MODE1	0	1	Rising edge with delay
SPI_MODE2	1	0	Falling edge without delay
SPI_MODE3	1	1	Falling edge with delay

**Offline simulation**

Enables or disables data inspection in an offline simulation. If set to **enable**, terminals are added to the subsystem in the top-level schematic. If set to **disable**, no such terminals are added to the subsystem.

**GPIO Settings**

**SCK Port**

Selects the port name of the clock frequency signal.

**SCK Pin**

Defines the pin number of the clock frequency signal.

**MISO Port**

Selects the port name of the MISO signal.

**MISO Pin**

Defines the pin number of the MISO signal.

**MOSI Port**

Selects the port name of the MOSI signal.

**MOSI Pin**

Defines the pin number of the MOSI signal.

**Slave(s)****CS Port**

Selects the port name to use for chip-select. In a multi-salve setup, all individual chip-select signals must be assigned to the same port group.

**CS Pin(s)**

Defines the pin number of the chip-select signal. Any GPIO can be configured to be a chip-select signal. Provide a vector to configure multiple slaves.

**Words per transmission (vector for multiple slaves)**

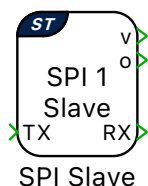
Configures the number of transmitted data words in each simulation step. Provide a vector to configure multiple slaves.

# SPI Slave

**Purpose** Implement SPI Slave

**Library** STM32

## Description



For a detailed description of the SPI protocol and signals, refer to the SPI Master (see page 86) block.

The SPI Slave is synchronized to the clock generated by the SPI Master. The SPI Master uses a dedicated active-low chip-select signal that enables the MISO and MOSI ports of the SPI Slave.

The data to be transmitted is provided at the input **TX** and the data received is available at the output **RX**. An output value of 1 at the **v** port indicates a valid data exchange with the SPI Master.

If the SPI **RX** port receives new data before the previous data has been read, the existing data will be overwritten and lost. If this occurs, the output **o** turns 1 to indicate an overrun error.

There are two considerations to note when overrun errors occur:

- The master is not allowed to start transmitting before the slave is up and running. If the slave is booting up while the master is transmitting, then it may receive an incomplete first message, from which it will not be able to recover.
- In order to avoid overruns, the SPI Slave block must be executed faster than the rate at which the SPI Master is sending data.

## Parameters

### Setup

#### SPI module

Selects the SPI module to use.

#### Bits per word (4-16)

Defines the length of a single data word during transmission. The allowed length is 4 to 16 bits per word.

**Mode [CPOL, CPHA]**

Defines four SPI clocking modes, controlled by clock polarity (CPOL) and clock phase (CPHA) bits. CPOL controls whether the clock signal is high (1) or low (0) when idle. CPHA controls whether data is shifted in and out on the rising or falling edge of the clock signal. The following table summarizes the clocking schemes:

**SPI Clocking Modes**

Mode	CPOL	CPHA	SPI Clock Scheme
SPI_MODE0	0	0	Rising edge without delay
SPI_MODE1	0	1	Rising edge with delay
SPI_MODE2	1	0	Falling edge without delay
SPI_MODE3	1	1	Falling edge with delay

**Words per transmission**

Configures the number of transmitted data words in each simulation step.

**Offline simulation**

Enables or disables data inspection in an offline simulation. If set to **enable**, terminals are added to the subsystem in the top-level schematic. If set to **disable**, no such terminals are added to the subsystem.

**GPIO Settings****SCK Port**

Selects the port name of the clock frequency signal.

**SCK Pin**

Defines the pin number of the clock frequency signal.

**MISO Port**

Selects the port name of the MISO signal.

**MISO Pin**

Defines the pin number of the MISO signal.

**MOSI Port**

Selects the port name of the MOSI signal.

**MOSI Pin**

Defines the pin number of the MOSI signal.

**CS Port**

Selects the port name of the chip-select signal.

**CS Pin**

Defines the pin number of the chip-select signal.



## Timer

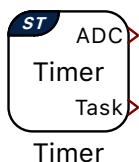
### Purpose

Generate trigger signals for the ADC start-of-conversion and the control task using a general-purpose timer

### Library

STM32

### Description



The Timer block configures the general-purpose timers TIM3 or TIM4, to generate an interrupt at the specified frequency. The timer interrupt can be used to trigger the ADC start-of-conversion or the Control Task Trigger.

The exact timer frequency may not be achievable based on the system clock frequency. The **Frequency tolerance** parameter allows automatically rounding to the closest achievable value when the exact timer frequency is unachievable.

### Parameters

#### Frequency

Defines the frequency of the timer in Hertz (Hz).

#### Frequency tolerance

Specifies the behavior when the desired timer frequency is not achievable.



