plecs

THE SIMULATION PLATFORM FOR POWER ELECTRONIC SYSTEMS

XMC Target Support User Manual Version 1.2

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
	http://www.plexim.com	Web

XMC Target Support User Manual

© 2024 by Plexim GmbH

Contents

Contents

Quick Start

Requirements

The PLECS XMC Target Support Package currently supports the XMC1400 microprocessor from the XMC1xxx family and the XMC4400 microprocessor from the XMC4xxx family.

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

- A host computer (with Microsoft Windows or Mac OS X)
- PLECS Blockset or Standalone 4.8.5 or newer
- PLECS Coder
- Segger J-Link Software Pack V7.82 or newer,
  `https://www.segger.com/downloads/jlink/`

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

Installing the Target Support Package

Start by downloading the Target Support Package from the Plexim website: `https://www.plexim.com/download/tsp_xmc`. Depending on the platform you are running PLECS on, you will receive either a ZIP archive or a disk image.

Next, extract it and move the `xmc` folder to the PLECS Coder Target Support Packages path (e.g. to `HOME/Documents/PLECS/CoderTargets`).

In PLECS, select **Preferences...** from the **File** drop-down menu (or the **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 Target Support Packages path (e.g. `HOME/Documents/PLECS/CoderTargets`).

Click Rescan to make the TSP available without restarting PLECS. The targets included as part of the XMC Target Support Package should now be listed under **Installed targets**. These targets are now also available in the Target dropdown menu which can be found under the **Target** tab of the **Coder + Coder options...** window.

Next, the required Segger J-Link Software Pack must be downloaded from the Segger website (https://www.segger.com/downloads/jlink/) and installed on your host computer. Finally, you must provide PLECS with your J-Link installation path. To do this, return to the **PLECS Preferences** dialog by selecting **Preferences...** from the **File** drop-down menu (or the **PLECS** menu on Mac OS X). Navigate to the **Coder** tab to see the installed targets. Click the 🖼 icon in the **Family** column next to the **XMC** entry and enter the path to the J-Link installation (e.g. /Applications/SEGGER/JLink).

Four more folders labeled /projects/dave_1400, /projects/dave_4400, /projects/mtb_1400, and /projects/mtb_4400 are included in the ZIP archive. The contents of these folders are only required if the PLECS Coder is configured to generate code into a DAVE IDE or ModusToolbox project.

Build and Deploy Generated Code

There are three primary methods for building and deploying generated embedded code onto an XMC MCU.

1 Build and program the MCU from PLECS

You can directly program the target device from the PLECS application. 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 J-Link.

2 Build and program the MCU from ModusToolbox

In this approach the PLECS Coder generates code for the specified target into a template ModusToolbox C Project. ModusToolbox 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.

3 Build and program the MCU from DAVE

Similar to generating code into a template ModusToolbox C project, the PLECS Coder can generate code for the specified target into a template

DAVE project. DAVE 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 different methods by changing the `Build type` parameter in the **Coder options... + Target** 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. The XMC TSP uses Segger J-Link for that purpose.

To deploy code to an XMC target from PLECS, navigate to the PLECS **Coder Options + Target** window, and select the target MCU. Select **Build and program** from the `Build type` dropdown menu.

Program the MCU from ModusToolbox

The target support package includes two archive files containing ModusToolbox C Projects. These projects are pre-configured to build and deploy code for XMC microcontrollers from ModusToolbox 3.2. To proceed, first import the pre-configured ModusToolbox C Projects into ModusToolbox. Depending on the chip you are developing for, that will be either `/projects/mtb_1400.zip` or `/projects/mtb_4400.zip`.

To import, navigate to **File + Import...** and select **Existing Projects into Workspace**. **Browse** to the archive file configured for the MCU you are developing for (i.e. `/projects/mtb_1400.zip` or `/projects/mtb_4400.zip`), and click **Finish**.

Next, the build configuration must be specified according to the size of your microcontroller's flash memory. To do this, right click on your ModusToolbox project in the **Project Explorer** tab and select the appropriate build configuration under **Build Configurations + Set Active**.

Open the PLECS application, navigate to the **Coder + Coder Options...** window and select the **Target** tab. Select **Generate code into ModusToolbox project** from the **Build type** dropdown menu. Enter the location of the `/projects/mtb_1400/cg` or `/projects/mtb_4400/cg` folder into the **ModusToolbox project directory** field and click **Build**. Then, proceed to build and debug your project as you would a normal ModusToolbox C Project.

5

Note that it is necessary to manually delete the contents of the `/project-s/mtb_1400/cg` and `/projects/mtb_4400/cg` folders when generating code for a new subsystem of a different name, as the ModusToolbox builder will build all files in this folder, including old files.

Program the MCU from DAVE

The target support package includes two archive files containing DAVE Projects. These projects are pre-configured to build and deploy code for XMC microcontrollers from DAVE 4.5.0. To proceed, first import the pre-configured DAVE Projects into DAVE. Depending on the chip you are developing for, that will be either `/projects/dave_1400.zip` or `/projects/dave_4400.zip`.

To import, navigate to **File + Import...** and select **DAVE Project**. **Browse** to the archive file configured for the MCU you are developing for (i.e. `/projects/-dave_1400.zip` or `/projects/dave_4400.zip`), and click **Finish**.

Next, the build configuration must be specified according to the size of your microcontroller's flash memory. To do this, right click on your DAVE project in the **C/C++ Projects** tab and select the appropriate build configuration under **Build Configurations + Set Active**.

Open the PLECS application, navigate to the **Coder + Coder Options...** window and select the **Target** tab. Select **Generate code into DAVE project** from the **Build type** dropdown menu. Enter the location of the `/projects/-dave_1400/cg` or `/projects/dave_4400/cg` folder into the **DAVE project directory** field and click **Build**. Then, proceed to build and debug your project as you would a normal DAVE project.

Note that it is necessary to manually delete the contents of the `/projects/-dave_1400/cg` and `/projects/dave_4400/cg` folders when generating code for a new subsystem of a different name, as the DAVE IDE 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 PLECS 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.

2

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 XMC 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, ModusToolbox, or the DAVE IDE.

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 XMC 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 XMC TSP provides the flexibility to configure the ADC and control loop interrupts through the ADC trigger and task trigger signals.

ADC triggers start ADC conversions. The ADC start-of-conversion is driven by an event generated by a timer based component (i.e. a Timer or PWM block).

Task triggers are generated by an ADC block at the end of ADC conversions, or by Timer or PWM counter underflow and overflow 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 XMC TSP allow the user to generate multi-tasking code for the XMC 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

tasks can be defined in addition to the base task. These tasks can be executed at different rates, preserving processor time for the highest priority control task (also referred to as the base task).

Multi-tasking code generation is configured in the **Scheduling** tab of the **Coder + Coder options...** dialog. By changing the **Tasking mode** to `multi-tasking` and the **Task configuration** to `specify`, 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 fundamentally limits the time accuracy of the embedded target. The Timer and PWM carrier generation clocks are derived from the system clock. Therefore the time accuracy of task triggers and PWM carriers are also limited. The system clock frequency is defined in the **Target + General** tab of the **Coder + Coder Options** window.

Explicit and Implicit Trigger Definitions

The interrupt sequence of the embedded application can be defined explicitly by connecting trigger signals. If no Control Task Trigger is placed in the model, the task trigger source will be configured implicitly. Several possible explicit and implicit trigger sequences are discussed below.

Control task triggered by Timer

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

The explicit representation of the trigger chain includes a Timer component that generates the input signal for the Control Task Trigger. The **Discretization step size** parameter set in the **Coder + Coder Options + General** menu must agree with the Timer 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.
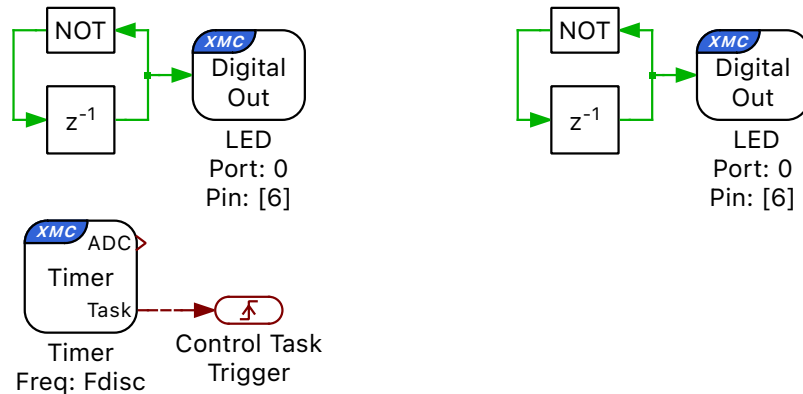
Explicit　　　　　　　　　　**Implicit**

Figure 2.1: Basic model with control task triggered by a Timer block

Control task triggered by PWM

If a PWM block is placed in the model, it is possible to synchronize the control task execution with the PWM carrier. This relationship can be defined explicitly by connecting the PWM task trigger output to the Control Task Trigger component.

If the schematic does not include a Control Task Trigger, the PLECS Coder will implicitly select the most appropriate source for the task trigger. If the model

contains a PWM block with a carrier frequency that matches the control task frequency, that PWM block is used as an implicit task trigger. If no such block exists, a Timer block is created implicitly to trigger the control task.

If the task trigger is set to disabled in the PWM **Trigger** tab and the trigger is defined implicitly, the timer overflow will provide the trigger event.
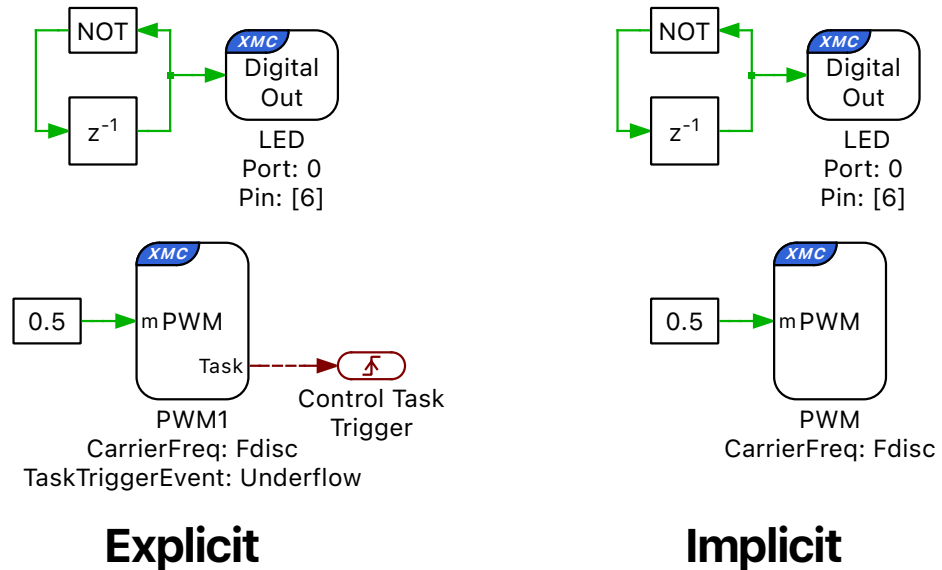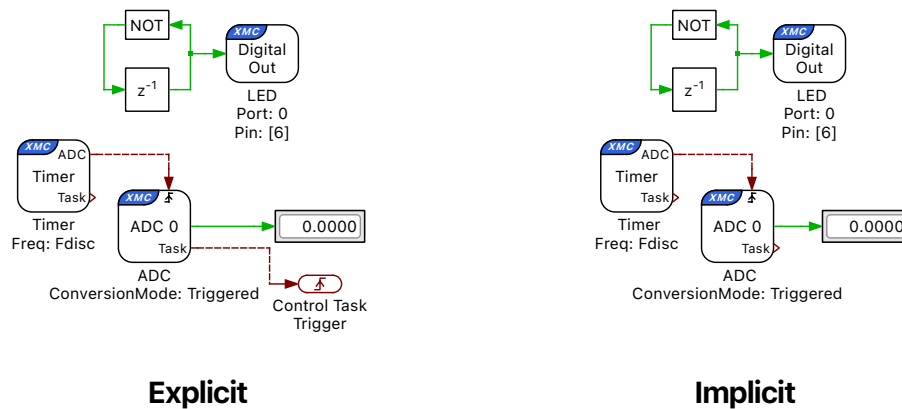

Figure 2.2: Basic model with control task triggered by PWM

Control task triggered by Timer via ADC

It is possible to trigger the start-of-conversion of an ADC by introducing an ADC block to the model, setting its coversion mode to `Triggered`, and connecting a Timer block. The frequency with which the ADC is read is then detemined by the frequency specified for the Timer block. In this configuration, an ADC block can serve as a task trigger, with the ADC end-of-conversion generating the trigger event.

If the ADC end-of-conversion is the source of the Control Task Trigger input, as shown in Figure 2.3, then the control loop execution will occur immediately after all ADC result registers are updated with the latest measurement values. The ADC can also act as an implicit trigger for the control task if the frequency of the Timer that triggers the ADC agrees with the discretization step size specified in the **Coder + Coder Options + General** menu (see fig. 2.3).

Explicit   **Implicit**

Figure 2.3: Basic model with control task triggered by ADC

Control task triggered by PWM via ADC

The ADC start-of-conversion can also be triggered by a PWM block instead of a Timer block. This arrangement synchronizes the ADC start-of-conversion with the PWM actuation. If the triggered ADC also acts as a task trigger, the ADC result registers are updated immediately before executing the control loop.

In this configuration, the ADC can be connected explicitly to a Control Task Trigger, or it can act as an implicit trigger if the frequency of the PWM that

triggers the ADC agrees with the discretization step size specified in the **Coder + Coder Options + General** menu (see fig. 2.4).
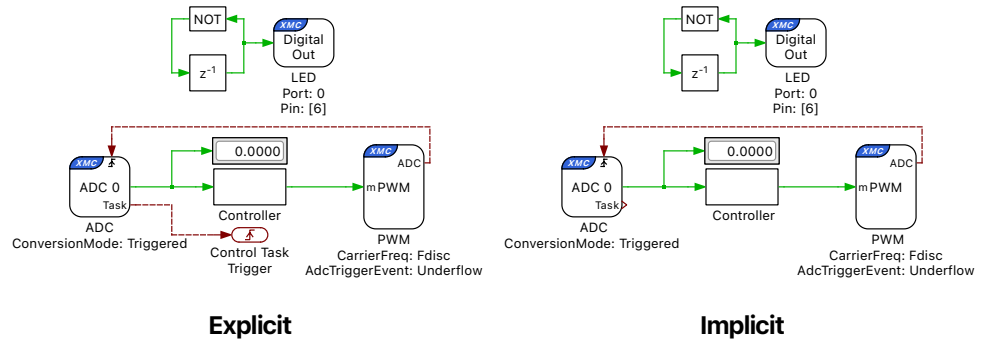

**Explicit**   **Implicit**

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

Advanced explicit configurations

Figure 2.5 displays a first advanced configuration. If a PWM block provides the task trigger, the control task interrupt can execute at integer multiples of the PWM carrier frequency by specifying a trigger divider. If a symmetric carrier is used, the control task can be triggered at twice the PWM carrier frequency by triggering on both the carrier overflow and underflow.

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.
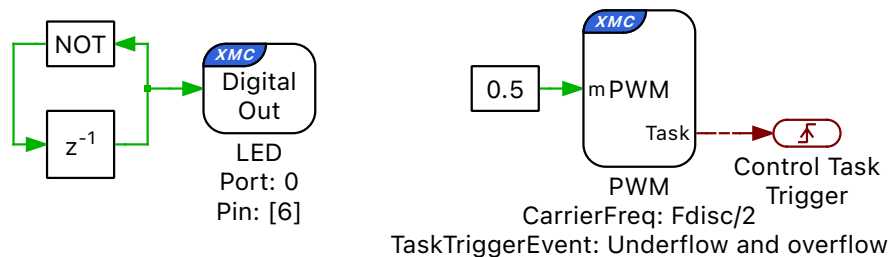
Figure 2.5: PWM frequency set to half the control task frequency
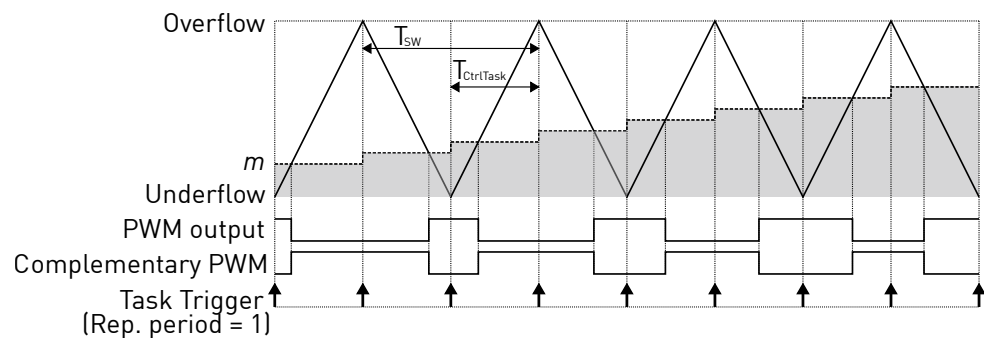
Figure 2.6: PWM carrier and task interrupts for PWM frequency set to half the control task frequency

A second advanced configuration is displayed in fig. 2.7. Each ADC can receive

independent start-of-conversion triggers from different PWM blocks for phase-shifted sampling. Figure 2.7 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 modulation index register is updated to a value calculated in the controller subsystem.
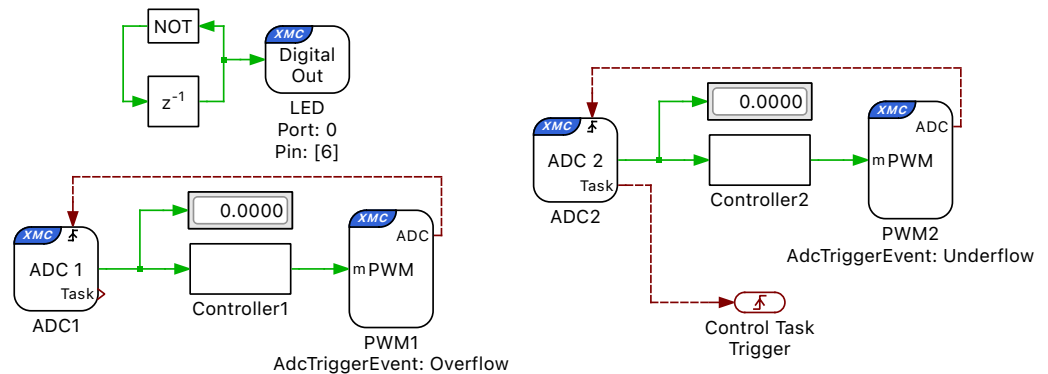
Figure 2.7: Explicit phase-shifted ADC sampling
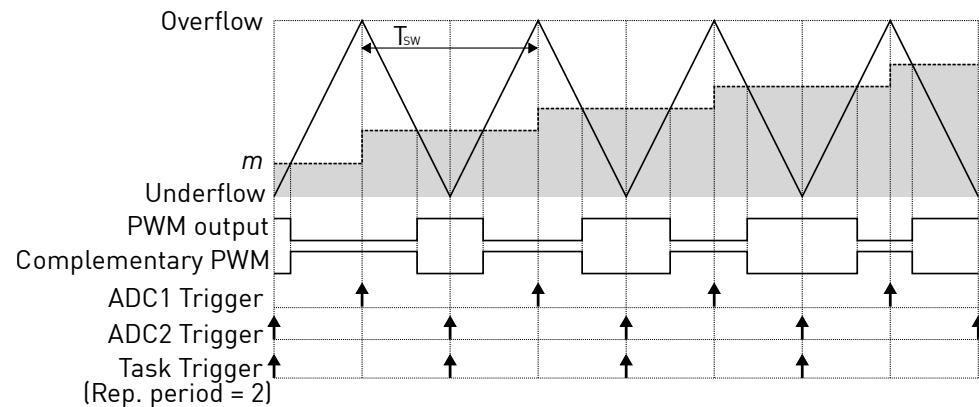
Figure 2.8: PWM carrier and interrupts for phase-shifted ADC sampling

17

The Code Generation Project

This section provides additional technical background on the software architecture of the embedded code generation project included with the XMC TSP. A ModusToolbox project and a DAVE IDE project is included for each supported target chip in the `projects/` folder of the TSP.

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 `Generate code into ModusToolbox project` is selected for the `Build type`, a `ModusToolbox project directory` must be specified. To conveniently edit the code in the ModusToolbox and flash the microcontroller from there, the code should be placed in one of the pre-configured ModusToolbox projects: `/projects/mtb_4400/cg` or `/projects/mtb_4400/cg`. By default all generated code is placed in a newly created folder in the same directory as the saved PLECS model.

When `Generate code into DAVE project` is selected for the `Build type`, a `DAVE project directory` must be specified. To conveniently edit the code in the DAVE IDE and flash the microcontroller from there, the code should be placed in one of the pre-configured DAVE projects: `/projects/dave_4400/cg` or `/projects/dave_4400/cg`. By default all generated code is placed in a newly created folder in the same directory 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 2.9 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 and the **Task configuration** to specify, 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. 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 2.10.

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

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. Assertions can be monitored using ModusToolbox or DAVE IDE debug tools.
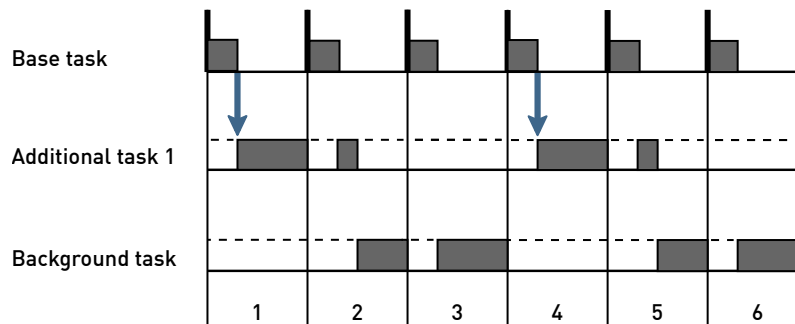
Figure 2.9: Nested control tasks
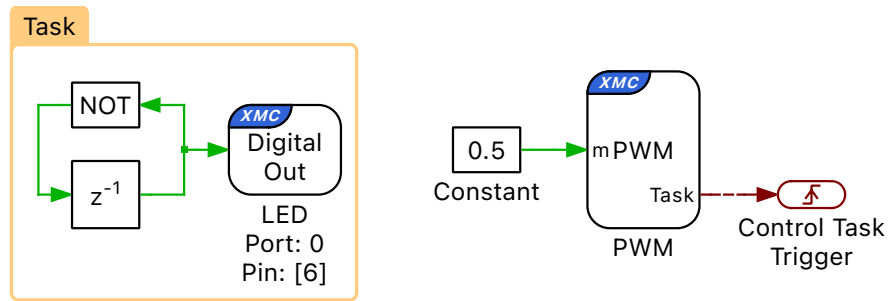
Figure 2.10: Example of an additional LED task along with a base PWM task

Embedded project architecture

Figure 2.11 shows the architecture of the embedded project included with the XMC 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. A custom light-weigth task scheduler was developed by Plexim for the XMC TSP. 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 XMC 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.

Application	Base Task	Additional Tasks

main(){
   initialization()
   CONTROL_INIT
   background()
}
	Highest priority task	

Dispatch Routine	PIL Framework

Hardware Abstraction Layer (HAL)

XMC Lib Software Library and Drivers

Embedded Hardware
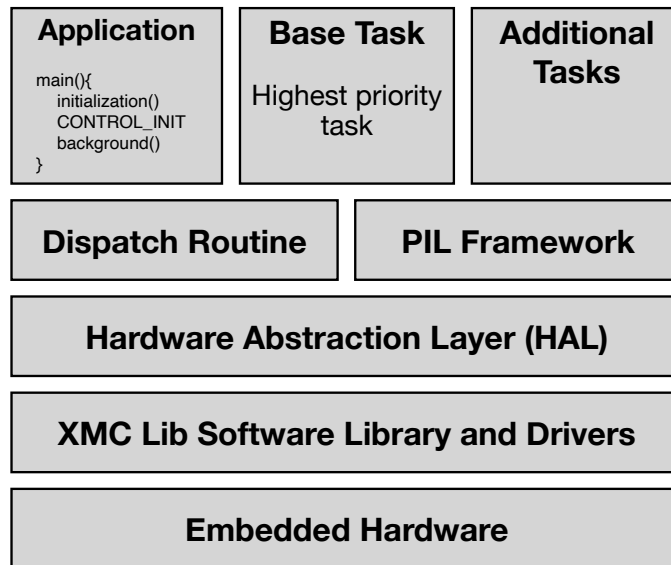
Figure 2.11: Embedded project architecture

3

XMC Coder Options

The **Target** tab contains code generation options which are specific to the XMC Target Support Package.

Target   Selects the target device family.

General

Chip   Selects the target device chip.

Package size   Selects the target device packaging.

Memory size   Selects the target device flash memory size.

System clock frequency (SYSCLK)[MHz]   Specifies the system clock frequency in megahertz (MHz). In the case of XMC1400 this corresponds to the main clock frequency (MCLK) and in the case of XMC4400 it corresponds to the PLL clock frequency (f_{PLL}).

System clock source   Selects the internal or external system clock source. The internal clock source corresponds to DCO1 in the case of XMC1400 and to the "fast internal oscillator" in the case of XMC4400.

External oscillator frequency [MHz]   Specifies the frequency in megahertz (MHz) of the external crystal High Precision Oscillator. The circuit for the external crystal High Precision Oscillator must be connected to pins XTAL1 and XTAL2.

External clock frequency [MHz]   Specifies the frequency in megahertz (MHz) of the external clock source. The external clock must be applied to pin XTAL1/CLKIN and XTAL2 must be left unconnected.

External ADC reference (Vref+)[V]   Specifies the external reference in volts (V). This reference voltage is used for an analog input (ADC) or output (DAC) signal.

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 ModusToolbox project` will generate code into the specified Modus-Toolbox project. ModusToolbox must then be used to build the project and flash the MCU. `Generate code into DAVE project` will generate code into the specified DAVE project. DAVE must then be used to build the project and flash the MCU. The `Build only` option will generate the code and build it. The `Build and program` option will automatically build and flash the target device from within PLECS.

ModusToolbox project directory Specifies the target folder for code generation. The code must be generated into pre-configured ModusToolbox C Projects. When using the ModusToolbox project templates provided with the XMC target support package, code must be generated into the `/projects/mtb_1400/cg` or `/projects/mtb_4400/cg` folder.

DAVE project directory Specifies the target folder for code generation. The code must be generated into a pre-configured DAVE project. When using the DAVE project templates provided with the XMC target support package, code must be generated into the `/projects/dave_1400/cg` or `/projects/dave_4400/cg` folder.

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. This is implemented using Segger J-Link.

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.

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.

# XMC Target Support Library Component Reference

This chapter lists the contents of the XMC Target Support library in alphabetical order.

Analog Comparator

Purpose   Configure an analog comparator.

Library   XMC

Description   This block configures the ACMP peripheral. The output can be routed in two ways: It can be connected to a digital output pin and it can be connected internally to a CCU8 slice.
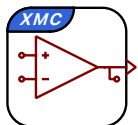
Parameters

Main

ACMP Unit
> Selects the analog comparator unit.

Hysteresis
> Selects the voltage hysteresis level. Adding hysteresis makes the analog comparator more robust to jitter.

Enable port
> The analog comparator can be enabled and disabled in software.

Output

Out
> The output of the analog comparator can be connected to a CCU8 slice through an internal hardware connection. The output can also be routed to an external digital output pin.

Out Port
> Port number of the output pin.

Out Pin
> Pin number of the output pin.

Output inversion

Allows the user to invert the polarity of the analog comparator output. By default, the output of the analog comparator is logic '1' if the voltage level at the positive input is greater than the voltage level at the negative input.

Output filter

Enables the optional anti glitch filter.

ADC

Purpose   Output the voltage measured on an analog input channel.

Library   XMC

Description   This block configures the VADC peripheral. The ADC 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 input*Scale+Offset. The ADC is configured to measure voltages with a 12 bit resolution.
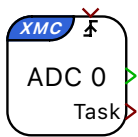
The ADC measurement conversion can either be triggered synchronously by a trigger event generated by a Timer or PWM block, or asynchronously by a conversion request source that is internal to the VADC peripheral. The user can toggle between these two configurations using the Conversion request source parameter.

If the **Task** output is connected to a Control Task Trigger, then the control task will execute every time the ADC is triggered by the Timer or PWM block, but only after all the enabled ADC channels have been converted.

Parameters

Main

Conversion mode
Determines whether the ADC measurement conversion is triggered synchronously by a trigger event generated by a Timer or PWM block, or asynchronously by a conversion request source that is internal to the VADC peripheral.

If the parameter is set to Continuous, the ADC conversion requests are issued by a **channel scan source**. If the Analog input channel(s) parameter is vectorized, each input channel is measured sequentially, starting from the highest enabled channel number and continuing towards lower channel numbers.

If the parameter is set to Triggered, the ADC conversion requests are issued by a **queued source**. This conversion request source is triggered by a CCU4 or CCU8 slice associated with a Timer or PWM block. The trigger block can be defined explicitly in the PLECS schematic by connecting an ADC trigger output to the ADC trigger input, or it can be chosen automatically if the ADC trigger input is left unconnected (see Chapter 2: Target Support Architecture for more information on explicit and implicit trigger

definitions). The ADC block converts the selected inputs once per trigger. If the `Analog input channel(s)` parameter is vectorized, each input channel is measured sequentially in the order that it appears in the vector.

ADC group

Selects the ADC group used by the ADC block. The same group can be used by multiple ADC blocks, but only one of these blocks can have an external trigger.

Analog input channel(s)

Indices of the enabled analog input channel(s) for the specific ADC group. For vectorized input signals a vector of input channel indices must be specified.

Scale(s)

A scale factor for the input signal. Can be a vector of the same length as the analog input channel vector or a scalar if the same scale factor is applied to all analog input channels.

Offset(s)

An offset for the scaled input signal. Can be a vector of the same length as the analog input channel vector or a scalar if the same offset is applied to all analog input channels.

Programmable-gain amplifier(s)

Allows the user to make the amplifier gains configurable if programmable-gain amplifiers are present on the target device.

Amplifier gain(s)

Sets the gains of the programmable-gain amplifiers. The amplifier gains affect the resolution of the ADC but do not scale the input signal. The scaling that results from the amplifier gain is cancelled by inverse scaling in software. The output of the ADC remains input*Scale+Offset. Can be a vector of the same length as the analog input channel vector or a scalar if the same gain is applied to all analog input channels.

Acquisition time

Selects between a user defined or minimum ADC acquisition time.

Acquisition time value [s]

Sets the ADC acquisition time window in seconds.

Base Task Load

Purpose   Provide the CPU load measured as the execution time of the base task as a percentage of the base task period.

Library   XMC

Description   This block provides the execution time of the base task as a percentage of the base task period. If only a single task is executed on the microcontroller, this fully represents the CPU load. In case of multi-tasking, the output corresponds to the CPU load generated by the base task only, and does not include the load created by additional lower-priority tasks.
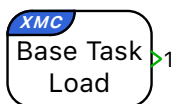
For robust operation, it must be ensured that the base task load never approaches 100 %. The base task load can be reduced by increasing the `Discretization step size` specified in the `Scheduling` tab of the `Coder Options` menu, or by simplifying the PLECS model.

Control Task Trigger

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

Library   XMC

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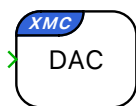
Parameters   **Nominal base sample time**
Specifies the nominal sample time of the discretized model in seconds.
The nominal base sample time value is synchronized with the model `Discretization step size` of the PLECS Coder settings.

DAC

Purpose   Generate an output voltage from the input signal; the output voltage is calculated as input*Scale+Offset.

Library   XMC

Description   The DAC block configures one channel of the Digital to Analog Converter (DAC) peripheral. The voltage at the output pin of the DAC channel is given by the block input signal, scaled by an optional scaling factor, offset by an optional offset voltage, and subsequently clipped by the minimum and maximum voltages. The maximum voltage range available to the DAC is 0.3 [V] to 2.5 [V]. The DAC is configured to generate voltages with a 12 bit resolution.

Parameters   **DAC channel**
   Selects which channel of the DAC peripheral is used by the DAC block. Each DAC channel is connected to a unique output pin. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Scale
   A scale factor for the output signal.

Offset
   An offset for the scaled output signal given in volts [V].

Minimum output voltage
   The output voltage is clipped if it drops below the minimum output voltage.

Maximum output voltage
   The output voltage is clipped if it exceeds the maximum output voltage.

Digital In

Purpose   Read a digital input.

Library   XMC

Description   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 number 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   XMC

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.

Parameters   **Port**
                     Selects the port number 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
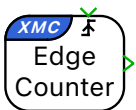   Specifies whether the digital output is configured with an internal pull-pull resistor or as an open drain output.

Edge Counter

**Purpose**   Count edges of a pulse train.

Library   XMC

Description The edge counter block counts edges of an external signal. It can be config-
ured to count rising edges, falling edges, or both. The block outputs the cur-
rent counter value. The counter value can be reset to its initial condition by a
software signal. When operating in `Single channel + Direction` mode, the
counter counts up or down depending on the signal level on the direction input
pin.

In order to provide this functionality, the edge counter block uses the POSIF
peripheral and the CCU4 peripheral. The POSIF peripheral is operated in di-
rection count mode.
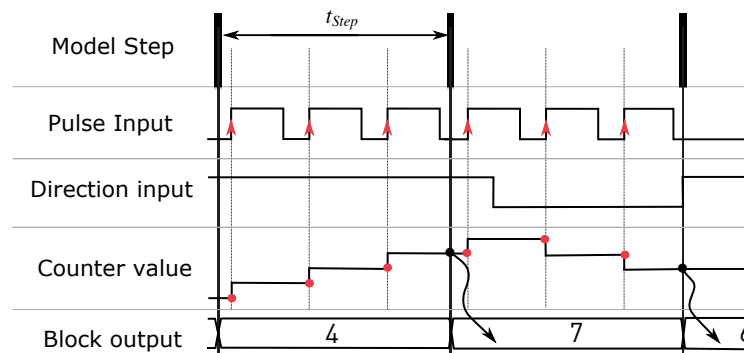
Edge counter value as a function of the direction signal

Parameters

Main

POSIF module
   Selects which POSIF module is used by the edge counter.

Maximum counter value
   Sets the value at which the counter wraps.

Mode
   Allows the user to enable or disable the optional direction input.

37

External reset
Adds an optional reset to the edge counter block. The edge counter is reset to its initial condition in response to a software signal. The reset can be configured to occur on the rising edge, the falling edge, or both. A rising edge is defined as a change from 0 to a non-zero value. A falling edge is defined as a change from a non-zero value to 0.

Initial condition
The initial condition defines the counter value before an edge is detected. The counter reverts to its initial condition after a reset.

Channel

Edge
Defines whether the counter is triggered by a `Rising` edge, a `Falling` edge, or `Either`.

Port
Configures the input port for the pulse train. The selected port/pin combination must be connected to input 0 of the selected POSIF module. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin
Configures the input pin for the pulse train. The selected port/pin combination must be connected to input 0 of the selected POSIF module. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

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

Direction

Counting direction
Specifies the counting direction as a function of the voltage level at the direction input pin.

Port
Configures the input port for the direction signal. The selected port/pin combination must be connected to input 1 of the selected POSIF module.

This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin

Configures the input pin for the direction signal. The selected port/pin combination must be connected to input 1 of the selected POSIF module. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.
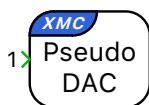
Input characteristic

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

Pseudo-DAC

Purpose   Generate an output voltage from the input signal; the output voltage is calculated as input*Scale+Offset.

Library   XMC

Description   This block generates an output signal consisting of rectangular pulses. The width and spacing of the pulses is modulated in such a way that the desired analog voltage level is generated when the signal is applied to an external low-pass filter with a sufficiently low cut-off frequency. The output signal can be scaled and an offset can be applied after scaling; it is calculated as input*Scale+Offset. Output voltage limitations can also be set.

Two peripherals can be used to generate the pulse train: the CCU4 peripheral and the BCCU peripheral. If the CCU4 peripheral is used, the Pseudo-DAC block will generate a PWM signal. The period of the PWM signal is defined by the specified resolution and the system clock frequency whereas the duty cycle of the PWM signal is defined by the input to the Pseudo-DAC block. The cut-off frequency of the external low pass filter should be set at least one decade below the frequency of the PWM signal. The Pseudo-DAC block will throw an error if that threshold is not met.

The BCCU generates a train of irregularly spaced pulses. Each pulse has a duration of 5 [us]. The density of the pulses is defined by the input to the Pseudo-DAC block. As a consequence of this modulation scheme, the fundamental frequency of the signal is a function of the input to the Pseudo-DAC block and will in some cases be variable because of the irregular spacing of the pulses. This makes it difficult to define a cut-off frequency for the external low pass filter that will be suitable for the full input range. Minimizing the cut-off frequency will result in a stable output voltage but will limit the frequency range of the signal. Infineon published an application note with the title "Pseudo Digital-to-Analog Converter (DAC) with XMC1000" which guided the development of the Pseudo-DAC block included in Plexim's XMC Target Support Package. In their application note, Infineon proposes an external low pass filter with a cut-off frequency of 3.4 [Hz]. If the input to the Pseudo-DAC block is confined to an interval near the center of the voltage range, a low pass filter with a much higher cut-off frequency can be used. At the 25th and 75th percentile of the voltage range, there will be on average one pulse every 20 [us]. This would result in a frequency of 50 [kHz]. However, the irregular spacing of the pulses will introduce lower frequency components. As a rule of thumb, the cut-off frequency can

be placed at one twentieth of the lowest frequency calculated for a regularly-spaced signal. For a Pseudo-DAC generating voltages between the 25th and 75th percentile of the voltage range, that would result in a cut-off frequency of 2.5 [kHz].

Parameters

Main

Peripheral
Lets the user choose whether they want to use the BCCU or a CCU4 to generate the rectangular pulse signal.

BCCU Channel
BCCU channel used to generate the rectangular pulse signal.

CCU4 Module
CCU4 module used to generate the rectangular pulse signal.

CCU4 Slice
CCU4 slice used to generate the rectangular pulse signal.

Scale
A scale factor for the filtered output signal.

Offset
An offset for the scaled filtered output signal.

Minimum output voltage
The lowest value that the filtered output voltage can reach.

Maximum output voltage
The highest value that the filtered output voltage can reach.

External filter cut-off frequency [Hz]
Cut-off frequency of the external low-pass filter. An error is thrown if the maximum achievable PWM frequency for the desired resolution is less than one decade above the cut-off frequency of the external low pass filter.

Resolution
Defines the quantization of the duty cycle of the rectangular pulse signal.

Output

Out Port
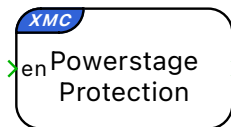Port number of the output pin.

Out Pin
Pin number of the output pin.

Powerstage Protection

Purpose   Provide powerstage safety features.

Library   XMC

Description

The Powerstage Protection block implements an interlock, which is a safety mechanism that enables or disables 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, can be defined as:

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

To reiterate, the powerstage enable signal 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 to the passive state.

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

Timing

When the Powerstage Protection **en** input is activated, the digital output associated with the block will immediately become active to, for example, enable a gate driver chip. However, the 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.

PLECS top-level schematic
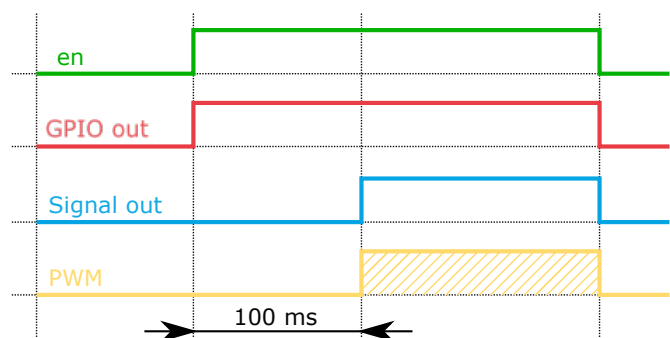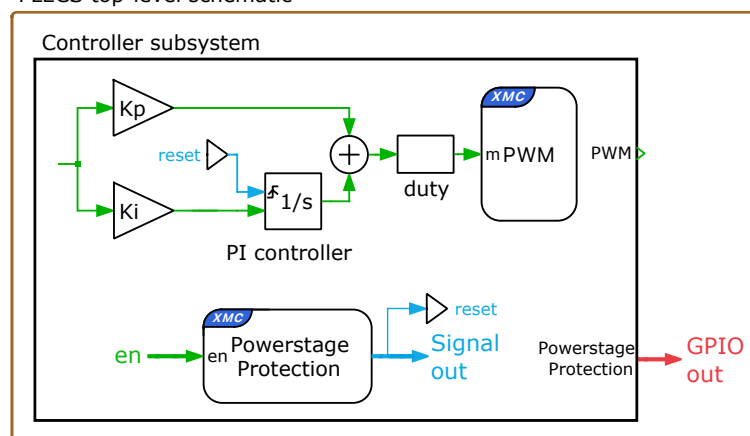
Powerstage Protection timing

Parameters

Main

Powerstage enable signal
Provides the option of configuring a digital as a powerstage enable indica-

tor.

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

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.

Powerstage enable pin output characteristic
   Specifies whether the digital output is configured with an internal pull-pull resistor or as an open drain output.

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 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 Powerstage Protection block can then be enabled athe 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 signal 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.
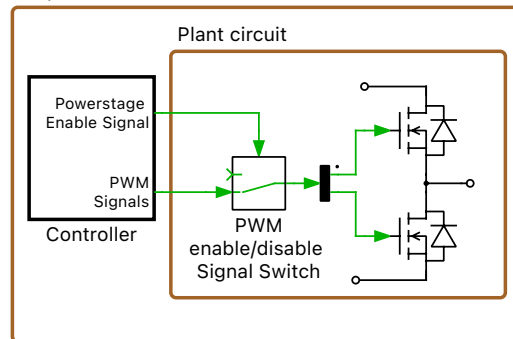
An offline implementation of the gate driver logic of the Powerstage Protection block
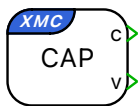
Pulse Capture

Purpose   Time-stamp edges of a pulse train.

Library   XMC

Description   The capture block allows timestamping signal transitions (events) on input pins, for example for period and/or duty cycle measurements. The timestamps are made available on the block output **c** and are given in clock cycles. The output **v** is 1 for one simulation step after an event has been triggered, 0 otherwise.

Note   The frequency of the clock supplied to the CCU4 peripheral used by the Capture block is equal to the **System clock frequency (SYSCLK)[MHz]** defined in the Coder Options on **XMC4xxx** targets, and **double** the **System clock frequency (SYSCLK)[MHz]** on **XMC1xxx** targets.

The pulse capture block uses one slice from the CCU4 peripheral, which is operated in capture mode. Two events of the CCU4 slice are configured by the capture block. They respond to rising or falling edges detected on the input pin and can be configured to capture and optionally reset the counter value of the CCU4 slice. The user can either elect to configure these events manually, or use the pre-configured PWM capture mode. In PWM capture mode, event 1 is used to capture the period and reset the counter and event 2 is used to capture the on-time. Which edge corresponds to which event will depend on the selected polarity.

The block output **c** is vectorized when multiple events are used. Two events are used when the capture block is operating in PWM capture mode. The block output **v** is vectorized when multiple input pins are used.

Parameters

Main

CCU4 Module
Selects which CCU4 module is used. CCU4 modules are further subdivided into CCU4 slices. Each pulse capture block configures one CCU4 slice.

CCU4 Slice
> Selects which CCU4 slice is used. Each pulse capture block configures one CCU4 slice and each CCU4 slice can be used by only one pulse capture block.

Clock prescaler
> The clock prescaler can be increased to increase the period of the signal which can be captured. The clock prescaler must be a power of two between 1 and 32768.

Mode
> Selects whether the capture events are configured manually or the pre-configured `PWM capture` mode is used.

Port
> Configures the input port in `PWM capture` mode. The selected port/pin combination must be a valid input for the selected CCU4 slice. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin
> Configures the input pin in `PWM capture` mode. The selected port/pin combination must be a valid input for the selected CCU4 slice. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Input characteristic
> Specifies whether an internal pull-up or pull-down resistor is connected to the input in `PWM capture` mode.

Polarity
> Specifies the polarity of the captured PWM signal when operating the pulse capture block in `PWM capture` mode.

Event [1, 2]

Capture
> Determines whether or not the event is used by the pulse capture block.

Reset counter on event [1, 2]
> Specifies whether or not the counter should be reset when the event occurs.

Edge
> Selects whether a rising or falling edge triggers the event.

Port

Configures the input port for event [1, 2]. The selected port/pin combination must be a valid input for the selected CCU4 slice. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin

Configures the input pin for event [1, 2]. The selected port/pin combination must be a valid input for the selected CCU4 slice. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.
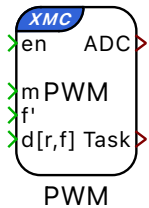
Input characteristic

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

PWM

Purpose   Generate a PWM signal or a complementary PWM signal pair.

Library   XMC

Description   The PWM block configures a CCU8 slice in order to generate PWM signals. Each CCU8 slice has four outputs which are grouped into two compare channels with a different compare value. Each channel can generate either a single PWM signal or a complementary PWM signal pair. The compare value is calculated using the input signal which provides the modulation index for each active channel. 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 is 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 by the block parameter Carrier frequency [Hz] and the input signal f'. Note that all PWM signals generated by the same PWM block share the same frequency input.

The start of all PWM blocks is synchronized using a single trigger event.

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. If the carrier type is symmetrical the interrupts will occur at the carrier underflow or overflow events. Underflow and overflow events correspond to PWM carrier reaching the carrier minimum or carrier maximum values respectively.

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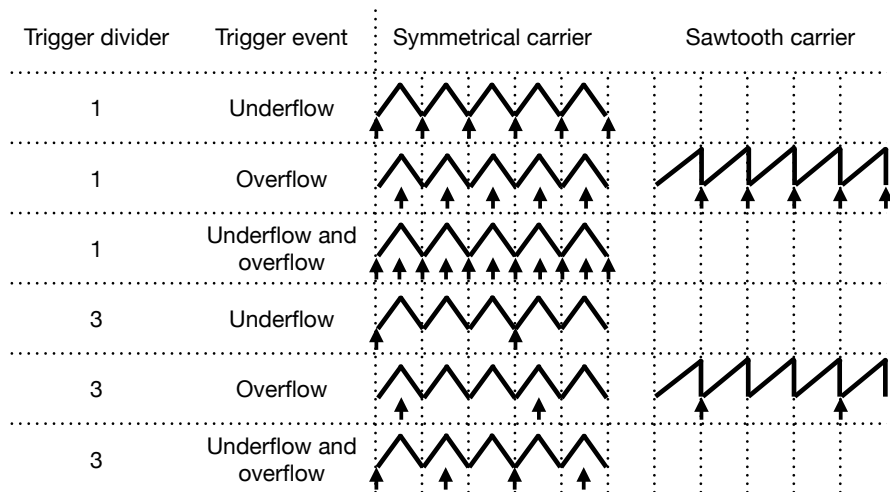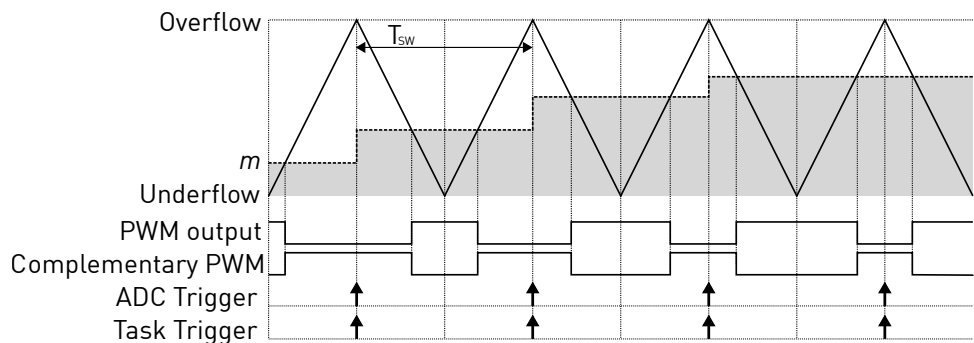
Parameters

Main

Module for PWM generation
Selects which CCU8 module is used. CCU8 modules are further subdivided into CCU8 slices. Each PWM block configures one CCU8 slice.

Slice for PWM generation
Selects which CCU8 slice is used. Each PWM block configures one CCU8 slice and each CCU8 slice can be used by only one PWM block. Each CCU8

Trigger divider	Trigger event	Symmetrical carrier	Sawtooth carrier
1	Underflow		
1	Overflow		
1	Underflow and overflow		
3	Underflow		
3	Overflow		
3	Underflow and overflow		

Trigger events as a function of the carrier type and the trigger divider

PWM and trigger schemes for symmetric carrier

slice has four outputs grouped into two compare channels with two different compare values.

Carrier type
Selects the carrier waveform, either sawtooth or symmetrical.

Carrier frequency [Hz]
Defines the frequency of the carrier in Hertz.

Frequency tolerance
Specifies the behavior if the desired carrier frequency is not achievable with the specified system clock frequency.

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'.

Enable port
Creates an additional component port if set to Show. Applying a signal equal to zero to this port drives the PWM signals to the passive state.

Compare Channel [1, 2]

Mode
Every CCU8 slice has four outputs which are grouped into two compare channels. Outputs zero and one are controlled by compare value one and outputs two and three are controlled by compare value two. Each compare channel can generate either a single PWM signal or a complementary PWM signal pair.

Output
Selects which output channel is used if the compare channel is operated in single output mode.

Port (output [0, 1, 2, 3])
Selects the port of the PWM output. The selected port/pin combination must be connected to the CCU8 slice output that is generating the PWM signal. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin (output [0, 1, 2, 3])
Configures the pin of the PWM output. The selected port/pin combination must be connected to the CCU8 slice output that is generating the PWM signal. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Complementary port (output [1, 3])

Selects the port of the complementary PWM output. The selected port/pin combination must be connected to the CCU8 slice output that is generating the PWM signal. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Complementary pin (output [1, 3])

Configures the pin of the complementary PWM output. The selected port/pin combination must be connected to the CCU8 slice output that is generating the PWM signal. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Polarity

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

Dead time variation

If enabled, an additional input port d[rf] is displayed. The inputs to this port change the rising-edge and falling-edge dead time in real time. The port expects:

- A vector of size two to set equal rising and falling edge dead time for all channels, or
- A vector that specifies an individual dead time for each rising and falling edge.

Dead time [s]

Specifies the delay between the rising and falling edges of a complementary PWM signal pair in seconds. This parameter is only applicable if dead time variation is disabled.

Minimum dead time [s]

Specifies the minimum dead time in seconds when variable dead time is enabled.

Protection

Powerstage protection

Selects whether or not the PWM signals generated by the PWM block can be forced into the passive state by the powerstage protection block. This allows the user to protect the powerstage from undefined behavior during microcontroller power up or when the system enters a failure state.

Trigger

ADC trigger

Allows the user to select which trigger event is used. The interrupts can be configured to occur at the carrier underflow or overflow events or at both the underflow and overflow events. Underflow and overflow events correspond to the PWM carrier reaching the its minimum or maximum values respectively. If a sawtooth carrier is used, the interrupts must occur at the carrier overflow event.

ADC trigger divider

Downsamples the ADC trigger events by specifying how many times the trigger event needs to occur before a trigger signal is generated.

Task trigger

Allows the user to select which trigger event is used. The interrupts can be configured to occur at the carrier underflow or overflow events or at both the underflow and overflow events. Underflow and overflow events correspond to the PWM carrier reaching the its minimum or maximum values respectively. If a sawtooth carrier is used, the interrupts must occur at the carrier overflow event.

Task trigger divider

Downsamples the task trigger events by specifying how many times the trigger event needs to occur before a trigger signal is generated.
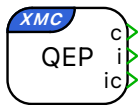
Quadrature Encoder Counter (QEP)

Purpose   Count edges generated by a quadrature encoder.

Library   XMC

Description   The Quadrature Encoder Counter block counts edges which are generated by a quadrature encoder. The *A*, *B*, and *I* outputs of the encoder are connected to the input pins of the POSIF peripheral.

The block features up to three output ports. Each port's availability is a function of the block's operating mode. Output **c** represents the current counter value, **i** the index pulse, and **ic** 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 if the direction of rotation results in rising edge of *B* trailing the rising edge of *A* and will decrease in the opposite direction of rotation. The counter is incremented or decremented at every rising and falling edge of channel *A* and channel *B*. Therefore, the user must set the `Maximum counter value` to the number of line pairs of the encoder multiplied by the number of counted edges per line pair minus one. As an example, an encoder with 1024 line pairs would have a maximum count of 4095 because the QEP module counts all edges detected on channel *A* and channel *B*, which results in four edges per line pair.

Once the counter reaches the value specified for the `Maximum counter value` parameter, it is reset to zero on the next detected edge in the positive direction. Vice versa, the counter is set to the specified `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 additionally reset when the rising edge of the index input is detected.

In order to provide this functionality, the QEP block uses the POSIF peripheral and the CCU4 peripheral. The POSIF peripheral is operated in standard quadrature decoder mode.

Parameters

Main

POSIF module
  Selects which POSIF module is used by the quadrature encoder counter.

Maximum counter value

Sets the value at which the counter wraps.

Mode

Selects whether the counter should be reset by an index pulse or on overflow only. The QEP block provides four different modes:

- `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 the index pulse. Output port **ic** is not available.
- `Reset by first index pulse only`: The counter is reset by the first index pulse only. This index pulse serves to initialize the QEP block. Output port **ic** is not available.

Channel [A, B]

Port

Configures the input port for channel [A, B]. The selected port/pin combination must be connected to input [0, 1] of the selected POSIF module. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin

Configures the input pin for channel [A, B]. The selected port/pin combination must be connected to input [0, 1] of the selected POSIF module. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Input characteristic

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

Polarity

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

Index

Port

Configures the input port for the index signal. If the index pulse is routed to the POSIF peripheral, the selected port/pin combination must be connected to input 2 of the selected POSIF module. If the index pulse is routed directly to the CCU4 peripheral, the selected port/pin combination must be a valid input for the selected CCU4 module/slice combination. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Pin

Configures the input pin for the index signal. If the index pulse is routed to the POSIF peripheral, the selected port/pin combination must be connected to input 2 of the selected POSIF module. If the index pulse is routed directly to the CCU4 peripheral, the selected port/pin combination must be a valid input for the selected CCU4 module/slice combination. This information can be found in the **Port I/O Function Table** in the reference manual provided by Infineon.

Input characteristic

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

Index pulse routing

Selects whether the index pulse is routed to the POSIF peripheral, or directly to the CCU4 peripheral. For most users, routing the index pulse to the POSIF peripheral is recommended. This simplifies the user input and makes the allocation of CCU4 resources more flexible. The option of manually allocating the CCU4 resources and routing the index pulse directly to the CCU4 peripheral is provided as a workaround for known silicon bugs which cause the POSIF peripheral to malfunction for certain QEP signals (see **POSIF_AI.001** in the **XMC4400 Errata Sheet** provided by Infineon for more information).

CCU4 Module

Selects the CCU4 module that is used as a counter by the QEP block. This setting is only available if the index pulse is routed directly to the CCU4 peripheral. Otherwise, a CCU4 resource is assigned automatically.

CCU4 Slice

Selects the CCU4 slice that is used as a counter by the QEP block. This setting is only available if the index pulse is routed directly to the CCU4 peripheral. Otherwise, a CCU4 resource is assigned automatically.

Timer

Purpose   Generate trigger signals for the ADC start-of-conversion and the control task using the CCU4 peripheral.

Library   XMC

Description   The Timer block configures the CCU4 peripheral 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.
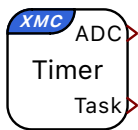
The exact timer frequency may not be achievable for a given system clock frequency. The Frequency tolerance parameter can configure the Timer block to automatically round to the closest achievable value when the exact timer frequency is unachievable.

Parameters   **Frequency [Hz]**
    Defines the frequency of the timer in Hertz (Hz).

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

plexim

electrical engineering software