



**THE SIMULATION PLATFORM FOR  
POWER ELECTRONIC SYSTEMS**

**TI C2000 Target Support User Manual** Version 1.5

## How to Contact Plexim:

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

### *TI C2000 Target Support User Manual*

© 2022 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. Bonjour is a registered trademark of Apple, 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 . . . . .	3
Install the Target Support Package . . . . .	3
Using the Installer Executable . . . . .	3
Manual Installation . . . . .	4
Build and Flash Configuration Settings . . . . .	5
Program the MCU from PLECS . . . . .	5
Program the MCU from CCS . . . . .	8
Start the External Mode . . . . .	9
Troubleshooting Guide . . . . .	10
Tips for Programming C2000 LaunchPads . . . . .	11
LAUNCHXL-F28069 LaunchPad . . . . .	11
LAUNCHXL-F280049 LaunchPad . . . . .	12
LAUNCHXL-F2837x LaunchPad . . . . .	13
TI C2000 Target Support and the PLECS RT Box . . . . .	14

<b>2</b>	<b>C2000 Target Support Architecture</b>	<b>15</b>
	Overview . . . . .	15
	The Embedded Code Generation Workflow . . . . .	15
	Control Task Execution . . . . .	16
	Control Task Accuracy and PWM Frequency Tolerance . . . . .	17
	Explicit and Implicit Trigger Definitions . . . . .	18
	The Code Generation Project . . . . .	26
<b>3</b>	<b>TI C2000 Coder Options</b>	<b>29</b>
	General . . . . .	29
	PGA . . . . .	30
	Protections . . . . .	30
	External Mode . . . . .	31
<b>4</b>	<b>TI C2000 Target Support Library Component Reference</b>	<b>33</b>
	ADC . . . . .	34
	CAN Port . . . . .	36
	CAN Receive . . . . .	38
	CAN Transmit . . . . .	39
	Control Task Trigger . . . . .	41
	CPU Load . . . . .	42
	DAC . . . . .	43
	Digital In . . . . .	44
	Digital Out . . . . .	45
	External Sync . . . . .	46
	Override Probe . . . . .	47
	Peak Current Controller . . . . .	48
	Powerstage Protection . . . . .	52
	Pulse Capture . . . . .	55
	PWM . . . . .	58
	PWM (Variable) . . . . .	61

Quadrature Encoder Counter (QEP) . . . . . 64  
Read Probe . . . . . 65  
SPI Master . . . . . 66  
SPI Slave . . . . . 71  
Timer . . . . . 73



# Quick Start

## Requirements

The PLECS Texas Instruments (TI) C2000 Target Support Package supports the TI 2806x, TI 2837x, TI 2838x, TI 28004x, and TI 2833x microprocessors.

In order to use the PLECS TI C2000 Target Support Package you will need:

- a host computer (with Microsoft Windows or Mac OS X),
- PLECS Blockset or Standalone 4.6.1 or newer
- PLECS Coder
- UniFlash for TI microcontrollers v6.4.0
- C2000 Code Gen Tools (C2000-CGT-18) v18.12.8.LTS

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

## Install the Target Support Package

For installation on Windows, using the Installer Executable is highly recommended. For other platforms, the manual installation process must be followed.

### Using the Installer Executable

Download the Installer Executable from the web page [https://www.plexim.com/download/tsp\\_c2000](https://www.plexim.com/download/tsp_c2000), and double click on it to start the setup of the TI C2000 Target Support Package. Choose the desired

location to install the files and select the desired installation type to proceed with the setup.

Choosing **Compact Installation** would only install the tools required for core functionality; these include the TI C2000 Target Support Package and the PLECS RT Box Target Support Package. Whereas, choosing **Full Installation** would download and install all the the necessary tools required to build and program the TI MCU from PLECS, including the auxiliary TI tools, described in the section below.

## Manual Installation

Download the appropriate ZIP archive from the web page [https://www.plexim.com/download/tsp\\_c2000](https://www.plexim.com/download/tsp_c2000), extract it and move the `ti_c2000` 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 TI C2000 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 a Code Composer Studio (CCS) project. The `projects/28xx.zip` files contain CCS projects that are used in conjunction with the embedded code generated from PLECS.

A set of basic demos is also included with the TI C2000 Target Support Package. Most of these demos use the PLECS RT Box to perform hardware-in-the-loop testing of the generated code. Therefore, the PLECS RT Box Target Support Package should be installed and configured. The RT Box Target Support Package can be downloaded from [https://www.plexim.com/download/rt\\_box](https://www.plexim.com/download/rt_box). The PLECS RT Box hardware is not required to generate and run microcontroller (MCU) code or to run the demo models offline in PLECS Blockset and Standalone.



## Build and Flash Configuration Settings

There are two primary methods to deploy generated embedded code onto a TI C2000 MCU. Both methods use free tools available from TI. You must download these tools from the TI website as they are not provided with your PLECS installation.

**1 Build and program the MCU from PLECS** You can directly program the target device from the PLECS application. This approach requires two standalone utilities available from TI: C2000 Code Gen Tools (CGT) and UniFlash. The C2000 CGT includes a compiler, assembler, linker, and additional tools to build C/C++ applications for the TI C2000 family of MCUs. UniFlash is a tool to program the on-chip flash memory of TI MCUs. Clicking **Build** in the Coder Options dialog generates model C code, builds the application using C2000 CGT, and then flashes the embedded target using UniFlash.

**2 Build and program the MCU from CCS** In this approach the PLECS Coder generates embedded C code for the specified target into a template CCS project. The CCS application is then used to build the project and flash the target device. The advantage of this method is having access to CCS's debugging tools.

If the required software is installed on your PC you can easily switch between the two methods by changing the **Build type** parameter in the **Coder options... + Target + General** menu.



### Program the MCU from PLECS

#### Configuring TI Code Gen Tools and UniFlash

If “Full Installation” was performed using the Installer Executable the setup process earlier, then these tools should be installed and configured. To manually configure these tools, download and install the following versions of C2000 Code Gen Tools and UniFlash, available online:

UniFlash v6.4.0: <https://www.ti.com/tool/download/UNIFLASH>

C2000-CGT v18.12.8.LTS: <https://www.ti.com/tool/download/C2000-CGT-18>

To configure the PLECS Coder to use the external TI 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. There is a  icon next to the **TI C2000** entry in the **Family** column indicating the external tools are not yet configured. After clicking the icon a dialog will appear where the user can enter the installation directories of the C2000 CGT and UniFlash tools. Once the installation directories are entered, you will see a  icon in the **Family** column, as shown in Figure 1.1.

### Deploy code to C2000 target from PLECS

To deploy code to a C2000 target from PLECS, navigate to the PLECS **Coder Options + Target** window, select the target MCU, then set the **Build type** to Build and program. There is a choice to select either Run from Flash or Run from RAM as the **Build configuration**. If using a Launchpad or a ControlCard as the **Board** type, select the appropriate one from the dropdown menu, and click **Build** to build, program and execute the generated code on the C2000 target.

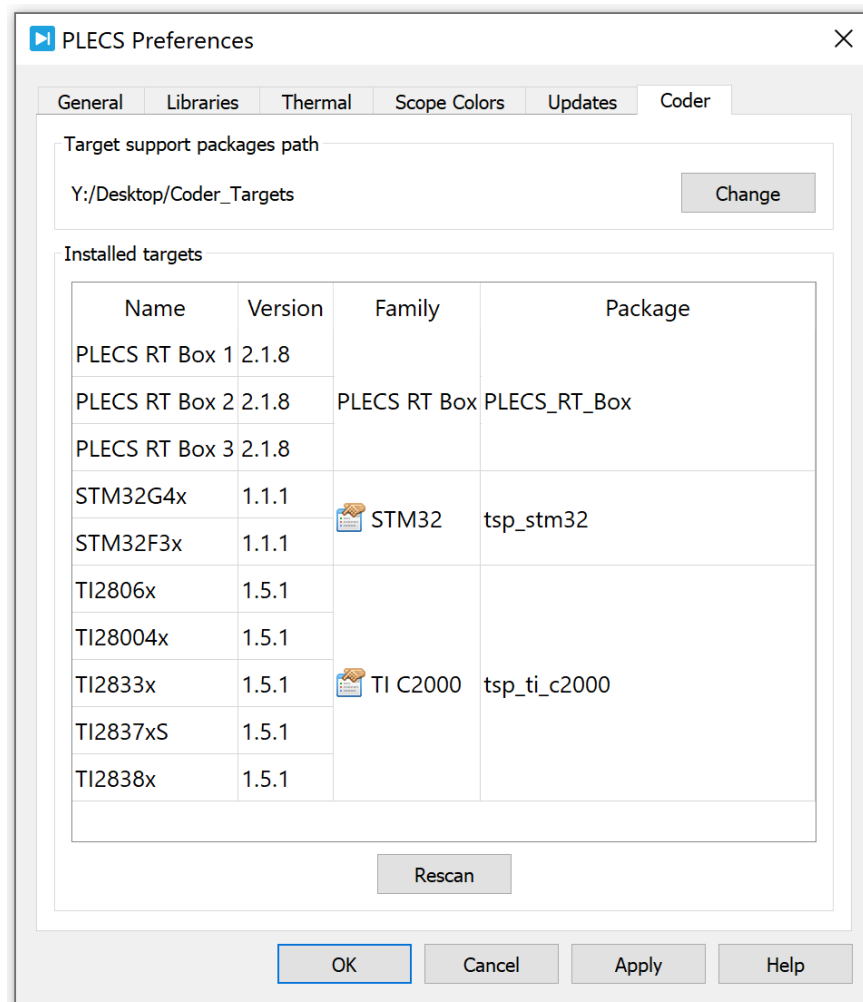
If using a Custom **Board** instead, you need to first generate the target configuration file once for a specific MCU. All information required to program the MCU is contained in the target configuration file. Target configuration files establish the basic communication settings for the MCU. Target configuration files have a ccxml extension and can be generated automatically from the UniFlash tool graphical user interface.

Open the UniFlash application and create a new configuration based on your selected device and connection method. Click the **Start** button after modifying any additional configuration options. After clicking **Start** you will now see a link to download the ccxml file near the top of the window.

After downloading the target configuration file navigate back to the PLECS **Coder Options + Target** window, set the **Board** field to Custom. The **UniFlash target configuration** field will now be visible. Enter the path to the ccxml file downloaded from UniFlash.

Modify any additional settings for your chosen target in the **Coder Options** window, including enabling or disabling the External Mode, and then click **Build**. This will automatically build the code, program the MCU, and start executing the generated code. Note that this programming method requires that only one TI C2000 MCU is connected to your host PC.

In the event there is an error in programming the MCU, the PLECS diagnostics window will contain additional debugging information. The diagnostics



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

window is accessible from the exclamation icon in the lower right hand corner of any PLECS schematic window.

## Program the MCU from CCS

### Configuring CCS

Download and install CCS v9.3 from the TI website. This CCS version is available at the following location:

CCS v9.3: [https://processors.wiki.ti.com/index.php/Download\\_CCS](https://processors.wiki.ti.com/index.php/Download_CCS)

After installing CCS, the next step is to import one of the template projects included as part of the TI C2000 Target Support Package. First, locate the projects/28xx.zip archives. This will be located in the TI C2000 Target Support Package directory that you have downloaded and installed from the web page [https://www.plexim.com/download/tsp\\_c2000](https://www.plexim.com/download/tsp_c2000).

Next, open CCS and click on the **Project** drop-down menu and then select **Import CCS Projects...** Then choose **Select archive file** and **Browse...** the zip archive in the projects folder that corresponds to the desired target. Select the discovered project and click on **Finish**. You will notice a new project created in your CCS workspace.

Then, in the **Project Explorer** tab of CCS, from the context menu of your project, add a new **Target Configuration File** or ccxml file for your target. Modify any required settings and test the connection with your MCU.

Next, re-open the context menu of your project and navigate to the **Properties + General** window, and make sure that the selected **Compiler version** matches the version recommended in the section “Requirements” (on page 3). If not, locate and install the recommended compiler version from the **Help** drop-down menu, under **Install Code Generation Compiler Tools... + TI Compiler Updates**.


Return to the PLECS application, navigate to the **Coder + Coder Options...** window and select the **Target** tab. Ensure the **Generate code into CCS project** option is selected as the **Build type**. Enter the location of the `${workspace_loc}/dev_28xx/cg/` folder from the CCS project into the **CCS project directory** field and click **Build**. Note that `{workspace_loc}` refers to the location of the imported project in the CCS workspace. You will notice several new files created in the `${workspace_loc}/dev_28xx/cg/` directory. Then, proceed to build and debug your project as you would a normal CCS 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}/dev_28xx/cg/` folder when generating code for a new

subsystem of a different name, as the CCS builder will build all files in this folder, including old files.

## Start the External Mode

Once the generated code is running on the C2000 target, the user can enter the External Mode to update Scopes in the PLECS application with real-time waveforms and change certain simulation parameters. The **Enable External Mode** checkbox must be selected when building the project.

To establish a communication link with your target, open the **Coder options... + External Mode** tab and then select the  icon next to the **Target device** field. Select the device type for the MCU connected to your PC. Click the **Scan** button to list of device names of available connections, select the appropriate device name, 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. Note that the XDS interface typically has two serial interface channels. One interface is for debugging and the other is for auxiliary communication including UART. If the External Mode connection to the device is unsuccessful, it is possible the debug channel was selected instead of the auxiliary communication channel.

Set the **Number of samples** parameter to 200 and click on the **Start autotriggering** button. 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.

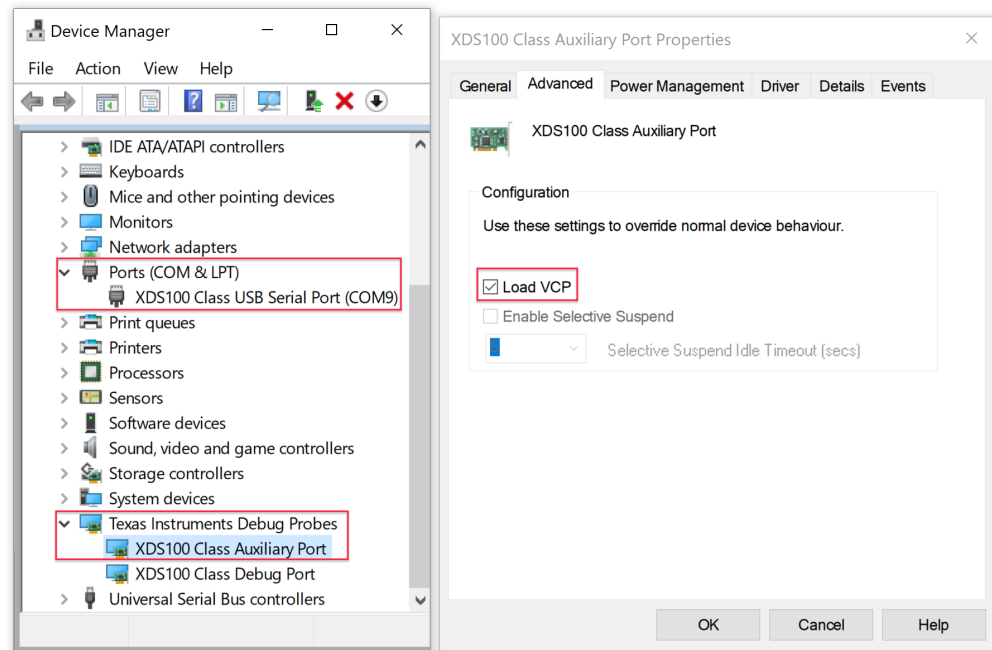
---

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.

## Troubleshooting Guide

If you're unable to connect to the External Mode, see the suggestions below:

- 1 If you're using the Windows operating system, open the **Device Manager** and verify that the “Load VCP” port of the “XDS100 Class Auxiliary Port” under “Texas Instruments Debug Probes” is enabled (unplug the MCU from the computer and replug if necessary). If configured correctly, the appropriate auxiliary port should show up under “Ports”, as shown in Figure 1.2.



**Figure 1.2: Configuring Load VCP**

- 2 Verify that the jumpers are installed in the correct position. For information on the appropriate jumper settings, refer to the section “Tips for Programming C2000 LaunchPads” (on page 11), and to the data sheet of the MCU.
- 3 If there is insufficient time to execute the generated code, then the processor will halt and you won't be able to connect to the External Mode. In such a scenario, first, test that the External Mode works with a very simple model (e.g., toggle a GPIO) to ensure the problem is not due to a hard-

ware or driver configuration. If the connection issue persists with a specific model, then potential solutions are:

- Reduce the complexity of the model, for example by reducing the number of scopes or optimizing computationally intensive calculations.
- Decrease the **Target buffer size** parameter under **Target +External Mode**.
- Increasing the discretization step size.

## Tips for Programming C2000 LaunchPads

Each TI C2000 LaunchPad has several jumpers and DIP switches that configure the target device's power isolation, communication isolation, and boot mode settings. The following settings are recommended to program the MCU and connect via the External Mode for supported LaunchPad devices.

Note the serial communication interface GPIO used for the External Mode will differ for some C2000 targets. The settings configured by the jumper and switch positions below should match the External Mode GPIO configured in the **Target + External Mode** tab of the **Coder Options** window.

### LAUNCHXL-F28069 LaunchPad

JP1 through JP5 configure the board power isolation. These jumper positions should be set based on the required isolation settings. JP6 and JP7 configure the serial communication interface to use GPIO 28 and 29 as the Rx and Tx signals.

The DIP switches configure the boot mode settings. S1-SW3 should be in the position pointing away from the MCU chip.

It is important to note that while the TI28069 MCU has two ADC's, ADCINAx and ADCINBx, the ADC units are structured with a common results register. Therefore, when addressing ADCINBx channels, the **ADC unit** setting should be "ADC A" and the channel offset by a factor of 8. For example, ADCINB1 should be entered with an **ADC unit** value of "ADC A" and an **Analog input channel(s)** value of 9.

**LAUNCHXL-F28069 Key Jumper Settings**

<b>Jumper</b>	<b>Position</b>	<b>Purpose</b>
JP6	Open	Configure USB/UART on GPIO 28 and 29
JP7	Closed	Configure USB/UART on GPIO 28 and 29

**LAUNCHXL-F28069 Key DIP Switch Settings**

<b>Switch</b>	<b>Position</b>	<b>Purpose</b>
S1-SW1	On/Off	GPIO34 logic level for boot mode configuration
S1-SW2	On/Off	GPIO37 / TDO logic level for boot mode configuration
S1-SW3	On	TRSTn tied to XDS100v2 for USB debugger connection

**LAUNCHXL-F280049 LaunchPad**

JP1 through JP9 configure the board power isolation. These jumper positions should be set based on the required isolation settings. The LAUNCHXL-F280049 can be configured with multiple functions set to the same header pins by adjusting the DIP switch positions. The basic recommended switch positions are shown below. Refer to the LaunchPad User's Guide for other possible configurations.

GPIO 28 and 29 or GPIO 35 and 37 [Rx,Tx] can be used for the External Mode interface. The switch settings in the table below configure the device to use GPIO 28 and 29.

Note on the LAUNCHXL-F280049 the XDS110 Debug Probe is only wired to support 2-pin cJTAG mode. This should also be reflected in the ccxml target configuration file.



**LAUNCHXL-F280049 Key Jumper Settings**

<b>Jumper</b>	<b>Position</b>	<b>Purpose</b>
J101-RXD	Closed	Serial receive isolation
J101-TXD	Closed	Serial transmit isolation
J101-TMS	Closed	JTAG test mode select isolation
J101-TCK	Closed	JTAG test clock isolation

Note that S2 is placed upside-down so the off position corresponds to logic 1 and the on position corresponds to logic 0. Both S2 switches should be oriented towards the MCU chip. S6 should be oriented towards the MCU chip and S8 away from the MCU.

**LAUNCHXL-F280049 Boot Mode DIP Switch Settings**

<b>Switch</b>	<b>Position</b>	<b>Purpose</b>
S2-SW1	Off	GPIO 32 boot from flash
S2-SW2	Off	GPIO 24 boot from flash
S6	Off	Route GPIO 28 and 29 to virtual COM port
S8	Off	Select GPIO 28 and 29 as serial pins

**LAUNCHXL-F2837x LaunchPad**

JP1 through JP5 configure the board power isolation. These jumper positions should be set based on the required isolation settings. Note the F2837x processor family has single core and dual core versions. When programming a dual core chip, the PLECS Coder will only generate code for the first core.

The DIP switches configure the boot mode settings. S1-SW3 should be in the position pointing away from the MCU chip. For the DIP switch settings below, GPIO 43 and 42 [Rx,Tx] should be used for External Mode communication

with the TI28379D MCU and GPIO 85 and 84 [Rx,Tx] should be used for the the TI28377S MCU.

#### **LAUNCHXL-F2837x Boot Mode DIP Switch Settings**

<b>Switch</b>	<b>Position</b>	<b>Purpose</b>
S1-SW1	On/Off	GPIO84 logic level for boot mode configuration
S1-SW2	On/Off	GPIO72 logic level for boot mode configuration
S1-SW3	On	TRSTn tied to XDS100v2

---

**Note** As per TI LAUNCHXL-F28379D Overview, in revision 1.1 of the TI 28379D launchpad, ADCINA2 is shorted to VREFHIB. It is recommended that users avoid using the ADCINA2 channel. This is fixed in revision 2.0.

---

### **TI C2000 Target Support and the PLECS RT Box**

Real-time simulation is a powerful tool to validate embedded control code, whether hand written or generated using embedded code generation techniques. The PLECS RT Box is the natural choice for real-time simulation since the offline simulation, real-time model, and embedded control code can all be derived from a common PLECS model.

Plexim offers a set of interface boards to facilitate the connection of LaunchPad and ControlCard development kits from TI. It is important to note that these interface boards route a digital output of the RT Box to the MCU reset pin via the RST jumper. If the jumper is closed then a low-level output from the RT Box will reset the MCU. Do not set this jumper unless you wish to use this feature, as it will interfere with programming the target processor. The RT Box provides board power to the LaunchPad device, and therefore the USB isolation jumpers should be removed when connected to the USB port.

# C2000 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 for specific processor families. A target support package enables the PLECS Coder to generate code that is specific to a particular hardware target such as the TI C2000 family of MCUs or the PLECS RT Box. With the PLECS Coder and a target support package 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 sub-system 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 Code Composer Studio.
- 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 input signals using digital control laws, and output signals to actuation devices. The TI C2000 Target Support Package 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 TI C2000 Target Support Package provides the flexibility to configure the ADC and control loop interrupts through the ADC trigger and task trigger signals.

ADC triggers configure the ADC start-of-conversion. The ADC start-of-conversion is driven by an interrupt from either a PWM carrier or the CPU Timer. All ADC channels associated with the 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.

Task triggers are generated by the ADC end-of-conversion signal, PWM counter underflow and overflow events, or the Timer block. The task trigger

that connects to the Control Task Trigger component will periodically trigger one execution of the digital control loop at the nominal base sample rate.

Additionally, the PLECS Coder and the TI C2000 Target Support Package allow the user to generate multi-tasking code for the TI C2000 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 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, 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 CPU 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 PWM (Variable) target support library blocks.

The discretization step size configured in the **General** 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.

---

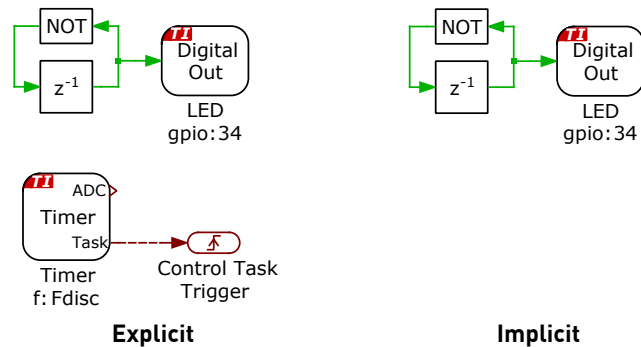
**Note** Explicitly defined trigger systems require that the Control Task Trigger's nominal base sample time parameter agrees with period of the task trigger input signal.

---

## Control task triggered by CPU Timer

In a basic project without an ADC or PWM component from the target support library, the task trigger must be generated by the CPU Timer. The schematic below shows a simple application where a GPIO 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 CPU Timer task frequency. In the implicit representation the PLECS Coder will configure the CPU Timer and Control Task Trigger automatically based on the **Discretization step size** parameter set in the **Coder + Coder Options + General** menu.



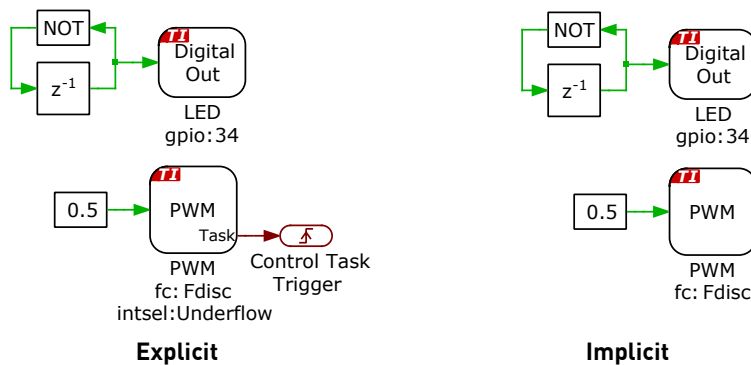
**Figure 2.1: Basic model with control task triggered by CPU Timer**

### Control task triggered by PWM

Control task execution can be synchronized with the PWM carrier underflow and overflow events. The task trigger is configured in the **Events** 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 will begin when the PWM carrier reaches an underflow (minimum value) or overflow (maximum value). 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 CPU 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 **Events** tab and the trigger is implicitly defined.



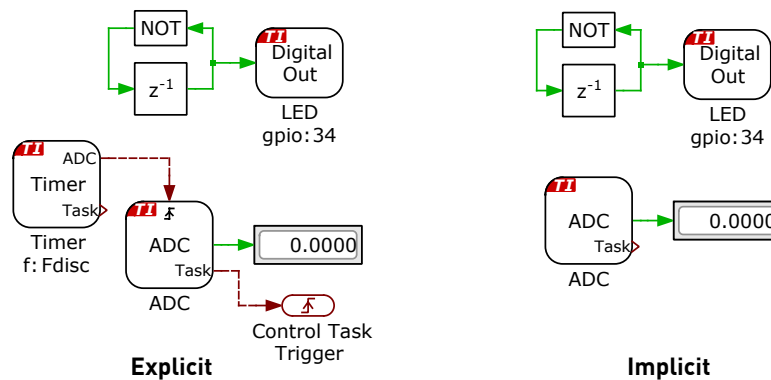
**Figure 2.2: Basic model with control task triggered by PWM**



## Control task triggered by CPU Timer via ADC

If the schematic includes an ADC but no PWM generators, then the ADC start-of-conversion must be triggered by the CPU Timer. In this case, the control task can be triggered by the ADC end-of-conversion or the CPU 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 CPU 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 + General** 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 CPU Timer will be used if it is more accurate. The control task will always be triggered by the ADC end-of-conversion signal.

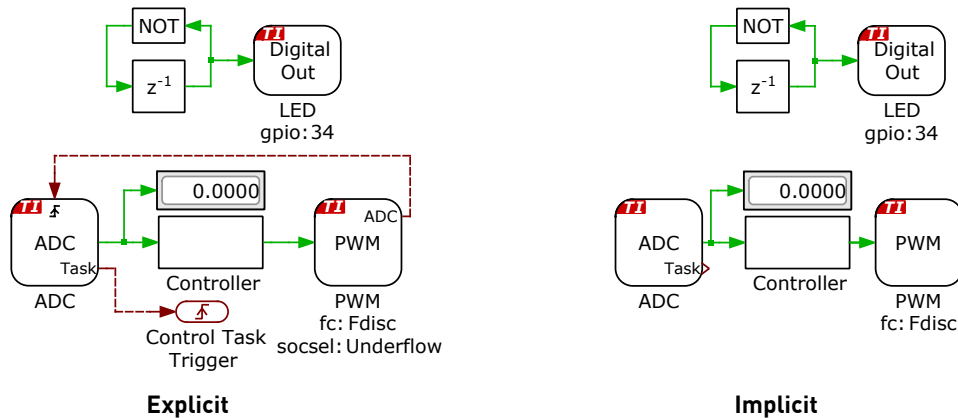
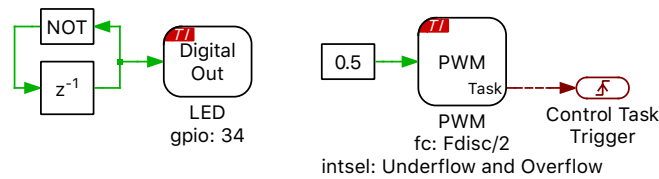


Figure 2.4: Basic model with control task triggered by PWM via ADC

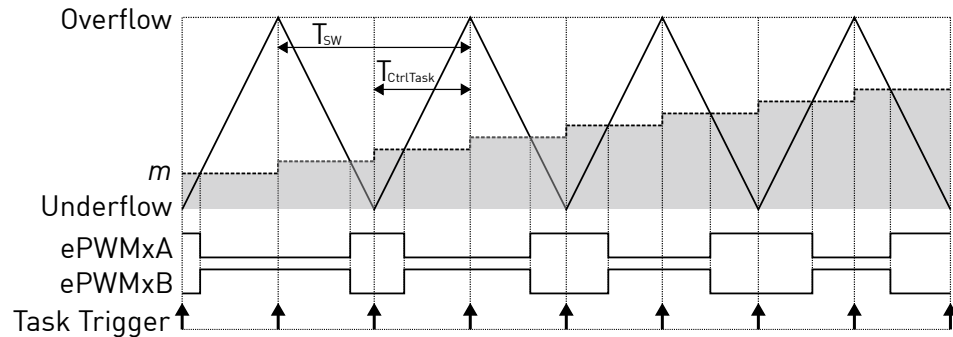
## Advanced explicit configurations

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.



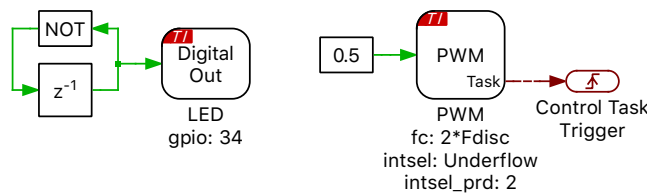
**Figure 2.5: PWM frequency set to half the control task frequency**



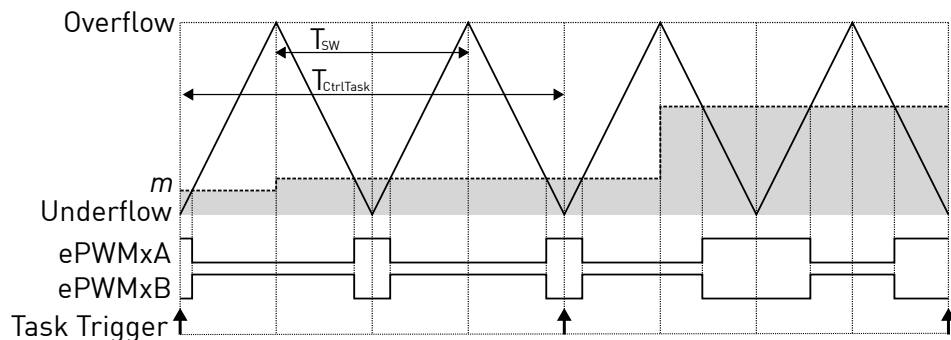
**Figure 2.6: PWM carrier and task interrupts for PWM frequency set to half the control task frequency**

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 C2000 target support package by default will only update the ePWM 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 ePWM 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.



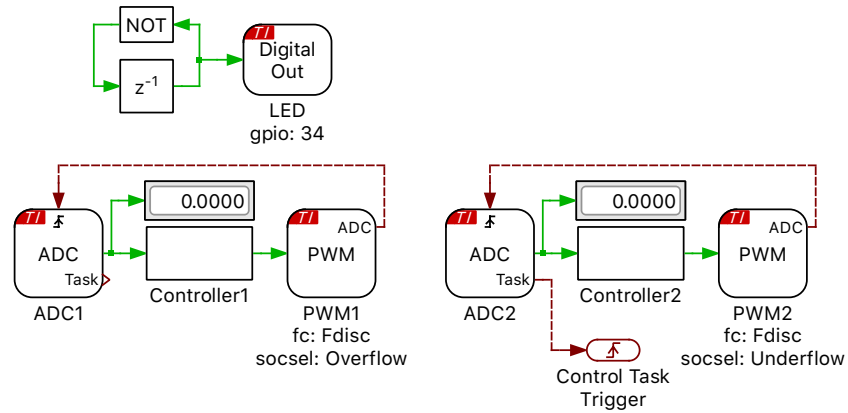
**Figure 2.7: Schematic of PWM frequency set to twice the control task frequency**



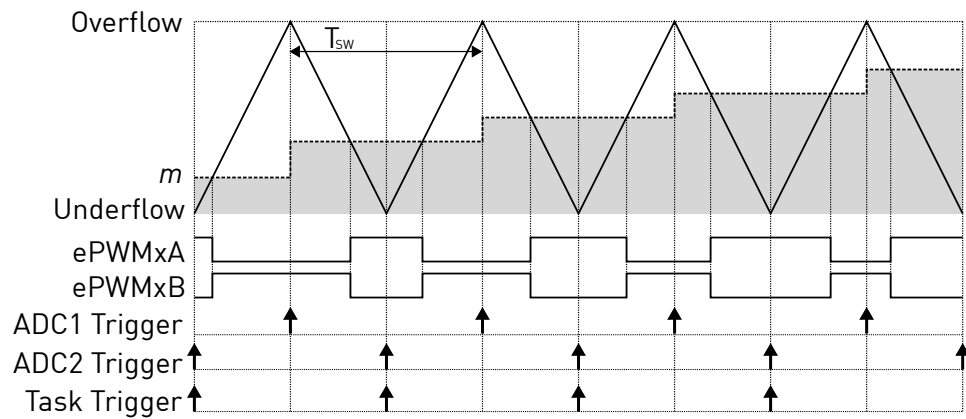
**Figure 2.8: PWM carrier and task interrupts for PWM frequency set to twice the control task frequency**

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 ePWM 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 TI C2000 Target Support Package. A Code Composer Studio (CCS) project is included for each supported target chip in the dev/28xx folder of the target support package. When building the project from directly from the PLECS application, the files in c2000/TI28xx folder of the target support package 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 target support package and should not be modified. The PLECS Coder also generates additional dynamic configuration files that are used by the embedded application.

When the **Build type** option is set to **Generate code into CCS project** then all generated dynamic code must be placed into the {workspace\_1oc}/dev/28xx/cg/ of the imported CCS project. If the **Build type** parameter is set to **Build and program** then by default all generated code is included in a new output directory in the same folder as the saved PLECS model.

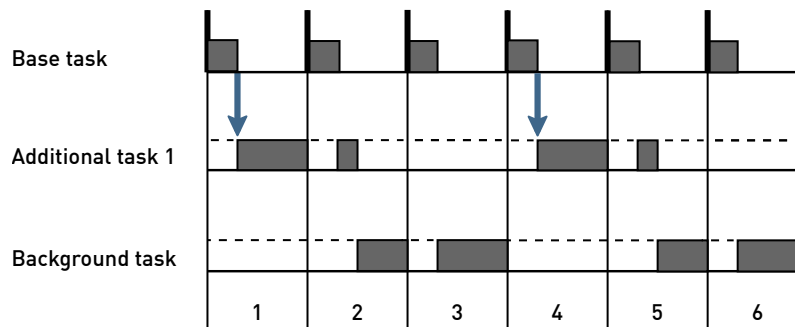
### Control and background task dispatching

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.

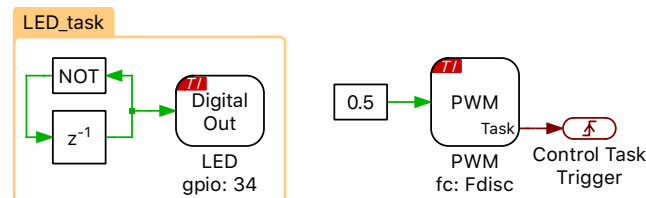
With every control task trigger interrupt issued by the CPU Timer, PWM, or ADC end-of-conversion (bold vertical bar), any lower priority tasks are interrupted and the base task is executed. This ensures that the control task has the highest priority. In addition, the lower priority tasks are periodically 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 2.12.

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**

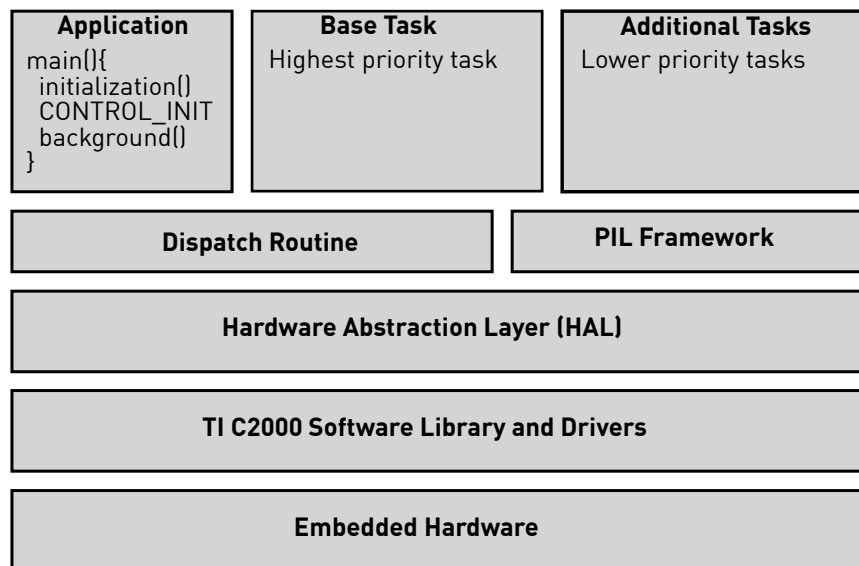


**Figure 2.12: Example of an additional LED task along with a base PWM task**

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. Assertions can be monitored using CCS debug tools.

### Embedded project architecture

Figure 2.13 shows the architecture of the embedded project included with the TI C2000 Target Support Package. 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 minimal real-time operating system that provides a rate monotonic scheduler for the nested control tasks, as previously described, and a processor-in-the-loop (PIL) framework that 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 TI C2000 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**



# TI C2000 Coder Options

The **Target** page contains code generation options which are specific to the TI C2000 Target Support Package.

## General

**Chip** Selects the target device chip.

**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.

**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**.

**Step size tolerance band [%]** Specifies the acceptable percent deviation in frequency from the specified value when the exact control task frequency is unachievable.

**Build type** This setting specifies the action of the **Build** button. **Generate code into CCS project** will generate code into the specified Code Composer Studio (CCS) project. CCS must then be used to build the project and flash the MCU. The **Build and program** option will automatically build and flash the target device from within PLECS using the provided **Build configuration** and **Board** type.

**CCS project directory** Specifies the target folder for code generation. The code must be generated into a pre-configured CCS project. When using the CCS project templates provided with the C2000 target support package, code must be generated into the {workspace\_loc}/dev\_28xx/cg folder where {workspace\_loc} refers to the location of the imported project in the CCS workspace.

**Build configuration** Provides an option to either Run from Flash or Run from RAM.

**Board** This setting allows selecting preconfigured UniFlash target configurations. If using either a TI LaunchPad or a TI controlCARD, LaunchPad or ControlCard should be selected. If using a Custom board instead, a custom UniFlash target configuration file must be provided.

**UniFlash target configuration** Defines the .ccxml target configuration file for custom boards. The target configuration file can be generated from TI UniFlash or from CCS.

## PGA

This tab allows for the configuration of the Programmable Gain Amplifiers that are present on 28004x devices. Each unit can be enabled and configured in terms of its gain and filter resistance.

## Protections

Digital and 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 Powerstage Protection (see page 52) block.

### Digital trips

Up to three trip zone digital input can be enabled and configured. Digital trips are active low inputs, i.e. a logical low input will activate the trip zone logic.

---

## Analog trips

For targets with CMPSS peripherals, up to 4 analog window comparators can be enabled and configured for generating trip signals for the Powerstage Protection block. Both ADC and PGA pins may be selected as protection inputs, as long as they have an internal connection to a CMPSS comparator module. The comparator is automatically determined by PLECS and an error message is issued in case no suitable comparator is found.

The user can configure both an upper and lower threshold. A trip signal is generated if the input falls below the lower threshold or exceeds the upper threshold. Note that it is allowable to set the lower threshold to 0.0 V, or the upper threshold to 3.3 V, thereby eliminate one of the trip regions. Each analog trip can be configured to emit one of three specific trip signals (labeled A, B, or C). A particular trip signal may be emitted by multiple analog trip inputs.

**Input type** Choose between ADC input or PGA input as protection inputs. This parameter is only shown for targets that feature PGA inputs.

**ADC unit** Selects the peripheral index for the ADC input when there are multiple ADC submodules.

**ADC input channel** Index of the analog input channel for a specific ADC submodule to be used as the protection input.

**PGA unit** Selects a Programmable Gain Amplifiers input (not available on all devices).

**Upper threshold voltage** Configures the upper threshold in volts (V). A value of 3.3 V is allowed to eliminate one of the trip regions.

**Lower threshold voltage** Configures the lower threshold in volts (V). A value of 0.0 V is allowed to eliminate one of the trip regions.

**Emit trip signal** Configures to emit one of three specific trip signals: A, B, or C.

## External Mode

These options are used to configure the External Mode communication with the target device.

**Enable 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.

**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].

**GPIO [Rx/Tx]** Specifies the GPIO pins used for the External Mode SCI connection. These GPIO pins cannot be used by other peripherals.

# TI C2000 Target Support Library Component Reference

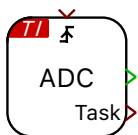
This chapter lists the contents of the TI C2000 Target Support library in alphabetical order.

## ADC

**Purpose** Output the measured voltage of the ADC peripheral

**Library** TI C2000

### Description



This block configures the ADC peripheral as a single-ended input with an internal voltage reference. The ADC block output signal represents the measured voltage at the ADC pin. The output is scalable and can be used with an offset, where the output signal is calculated as  $\text{input} * \text{Scale} + \text{Offset}$ . When the **Analog input channel(s)** parameter is vectorized, each input channel is measured sequentially in the order of the input channel vector.

The **Trigger source** parameter selects between an automatic or external ADC start-of-conversion signal, where the external start-of-conversion signal is connected to the ADC trigger port. If the ADC task output is the source of a Control Task Trigger then the control task will execute once the last ADC channel is converted.

### Parameters

#### Main

##### Trigger source

Selects an automatic or external start-of-conversion trigger.

##### ADC unit

Selects the peripheral index for the ADC input when there are multiple ADC submodules.

##### Analog input channel(s)

Index of the analog input channel for a specific ADC submodule. For vectorized input signals a vector of input channel indices must be specified.

##### 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.

## CAN Port

**Purpose** Set up a CAN communication port

**Library** TI C2000

**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*. 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.

### Error Modes

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

All nodes on a CAN bus detect errors and maintain two error counters: a *Transmit Error Counter* and a *Receive Error Counter*. Each node can be in one of the following 3 error modes:

- **error active** 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** 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** 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.

### Parameters

#### CAN interface

Selects the CAN interface to use.

#### Baud rate

Defines the baud rate that is used on the connected CAN bus. All devices on a CAN bus must be configured to use the same baud rate.

#### GPIO [Rx, Tx]

Specifies the GPIOs to use for CAN communication. Each CAN channel requires one receive (Rx) and one transmit (Tx) pin.



**Auto bus-on**

The Auto bus-on feature, if enabled, will automatically clear a bus-off condition, without the need for setting the enable signal **en** to 0, and then back to 1.

## CAN Receive

**Purpose** Receive CAN messages

**Library** TI C2000

### Description



The block initiates the reception of CAN messages with the 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 in 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 36) 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 a 11-bit value (for CAN 2.0A) or a 29-bit value (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 Transmit

**Purpose** Transmit CAN messages

**Library** TI C2000

### Description



The CAN Transmit block sends out data on a CAN bus. The data to send must be provided on the block input **d** 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).

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 36) 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 that is used for CAN messages sent by this block. The CAN ID can be supplied as either an 11-bit value (for CAN 2.0A) or a 29-bit value (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 task

**Library** TI C2000

### 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.

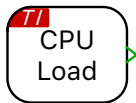
The nominal base sample time value is synchronized with the model **Discretization step size** of the PLECS Coder settings.

### CPU Load

**Purpose** Provide the CPU load in percent

**Library** TI C2000

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



## DAC

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

**Library** TI C2000

**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  $\text{input} * \text{Scale} + \text{Offset}$ . Output voltage limitations can also be set.



### Parameters

#### DAC Unit

Selects the peripheral index for the DAC input when there are multiple DAC submodules.

#### 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** TI C2000

## 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

### Digital input GPIO resource(s)

Defines the GPIO resource 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 resistor is connected to the digital input.



## Digital Out

**Purpose** Set a digital output

**Library** TI C2000

**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** **Digital output GPIO resources(s)**  
Defines the GPIO resource of the digital output channel. For vectorized output signals a vector of output channel indices must be specified.

**Output characteristic**  
Specifies whether an internal Push-pull or Open drain resistor is connected to the digital output.

# External Sync

**Purpose** Set up an external synchronization port for PWM output

**Library** TI C2000

### Description



The PWM (Variable) block can synchronize the PWM carrier phase with an external GPIO signal. This block is used to model the external synchronization input in offline simulations, and to specify the GPIO pin used for PWM synchronization when generating code.

Note that the number of allowable External Sync blocks is limited according to the hardware capabilities of the target MCU.

### Parameters

#### External GPIO

Defines the GPIO used to synchronize the PWM with an external source.

## Override Probe

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

**Library** TI C2000

**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** TI C2000

### Description



The Peak Current Controller (PCC) block implements peak current control with slope compensation. This block is supported on 280049 and 28379D MCUs.

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.

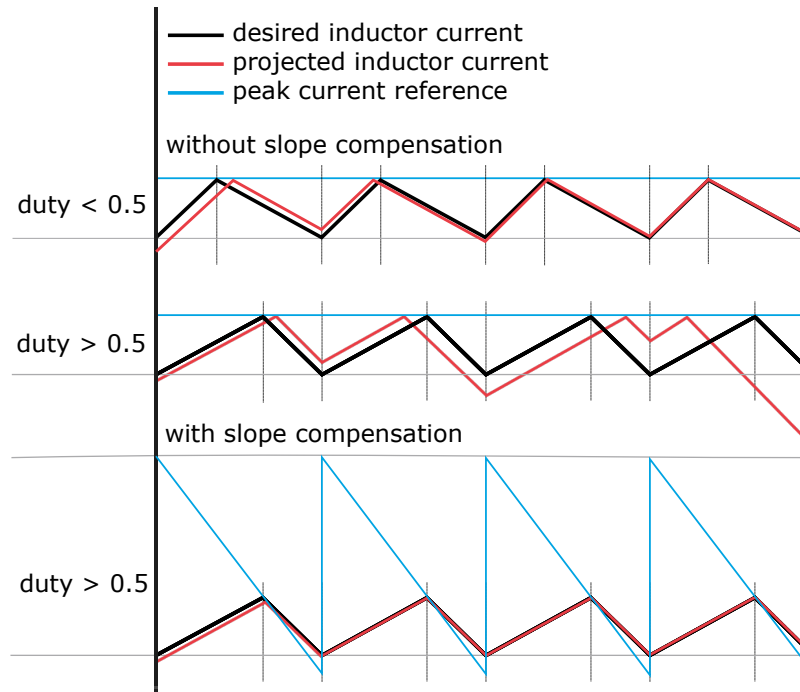
Internally, the PCC block makes use of multiple MCU peripherals. The first component is a DAC that provides a peak current set-point including ramp, for controlling the inductor current. The second is a comparator (COMP); the sensed current is fed to the comparator, which is then compared to the peak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

### Parameters

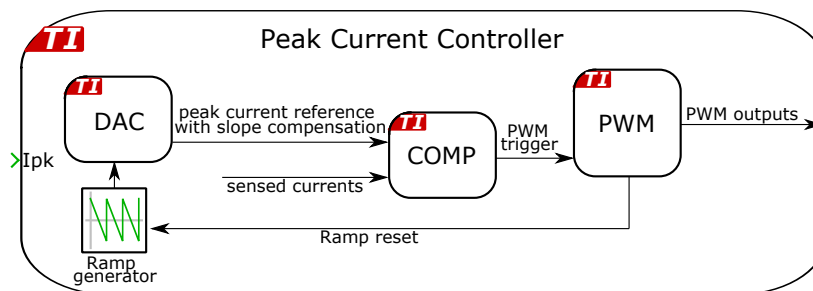
#### Main

##### PWM generator

Selects the index of the PWM resources to use. The PWM generator can independently generate a single PWM output or a complementary PWM pair.



### Slope compensation



### Peak current controller schematic

#### Carrier 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.

### **Sense input**

Selects the desired sense input: ADC input or PGA input.

### **ADC unit**

Selects the peripheral index for the ADC input when there are multiple ADC submodules.

### **ADC input channel**

Index of the analog input channel for a specific ADC submodule. For vectorized input signals a vector of input channel indices must be specified.

### **PGA unit**

Configures the Programmable Gain Amplifiers that are present on 28004x devices.

### **Current sense gain**

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.

### **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.

## **Output**

### **Mode**

- **Complementary outputs** operates channels A & B in complementary fashion with blanking time.
- **Single output (channel A)** only modulates channel A and allows the GPIO of channel B to be used for other purposes.

**Blanking time**

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

**Polarity**

Defines the logical output of the ePWMxA output when an active state is detected. The active state occurs when the modulation index exceeds the carrier. Note that ePWMxB is always complementary to ePWMxA.

**Events****ADC trigger**

Configures the ADC trigger output.

**ADC trigger divider**

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

**Task Trigger**

Configures the control task trigger output.

**Task trigger divider**

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

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

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

## Powerstage Protection

**Purpose** Provide powerstage safety features

**Library** TI C2000

### 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 GPIO (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 output GPIO pin, specified in the **Powerstage enable GPIO number** can be defined as:

- **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).

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**.

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

### Protection

**Digital trips:** The trip zone submodule can be used to disable the powerstage and associated PWM blocks following a trip event. Trip events are detected when there is an active low condition on the trip zone GPIO inputs assigned in the **Protections** tab of the **Coder + Coder Options + Target** window. When a trip event is detected the Powerstage Protection module can take



no action, activate a one-shot trip event, or activate a cycle-by-cycle trip event. A one-shot trip event will latch the PWM output to the PWM safe state and can only be cleared by cycling the Powerstage Protection block from *disabled* to *enabled*. Cycle-by-cycle trip events will set the PWM output to the PWM safe state until the PWM counter reaches an underflow event. At the PWM counter underflow the trip condition will be cleared if the trip zone input is no longer active. The cycle-by-cycle trip event will attempt to clear the trip condition once per PWM cycle. The **PWM safe state** is a configuration of the Powerstage Protection block.

**Analog trips:** Each analog trip can be configured in the **Protections** tab of the **Coder + Coder Options + Target** window to emit one of three specific trip signals (labeled A, B, or C). When a trip signal is detected the Powerstage Protection module can take no action or activate a one-shot trip event. A one-shot trip event will disable the powerstage (setting the PWM outputs to the configured **PWM safe state**). All trip events are latched and can only be cleared by cycling the Powerstage Protection block from *disabled* to *enabled*.

## Parameters

### Main

#### Powerstage enable signal

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

- **Digital output:** Configures a GPIO (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).

#### Powerstage enable GPIO number

Defines the GPIO pin to be configured as the powerstage enable signal.

#### PWM safe state

Specifies the forced PWM output state when a trip zone is triggered. Selecting the forced inactive option drives all associated PWM outputs the

passive state. Selecting the floating option sets the associated PWM outputs to a high impedance state.

### Protection

#### Reaction to TZ1

Selects the action following a digital trip zone event.

#### Reaction to TZ2

Selects the action following a digital trip zone event.

#### Reaction to TZ3

Selects the action following a digital trip zone event.

#### Reaction to trip signal A

Selects the action following an analog trip event detection.

#### Reaction to trip signal B

Selects the action following an analog trip event detection.

#### Reaction to trip signal C

Selects the action following an analog trip event detection.

### 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.

# Pulse Capture

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

**Library** TI C2000

## Description



The capture blocks allows time-stamping 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 all events have been triggered, 0 otherwise. The output **o** is set to 1 should the timestamp counter overflow. After a counter overflow, the counter resets to 0 and continues to count up. The output **o** is automatically cleared after it has been read.

Events can be captured individually for either *absolute time-stamp mode* or *time-difference mode*.

In *absolute time-stamp mode*, the timestamp counter continues incrementing after the capture event occurs.

In *time-difference mode*, the timestamp counter is reset when the event occurs. This feature simplifies determining elapsed time between events.

Depending on the eCAP type, the behavior of the capture block can vary. There are three eCAP types: Type-0, Type-1 and Type-2.

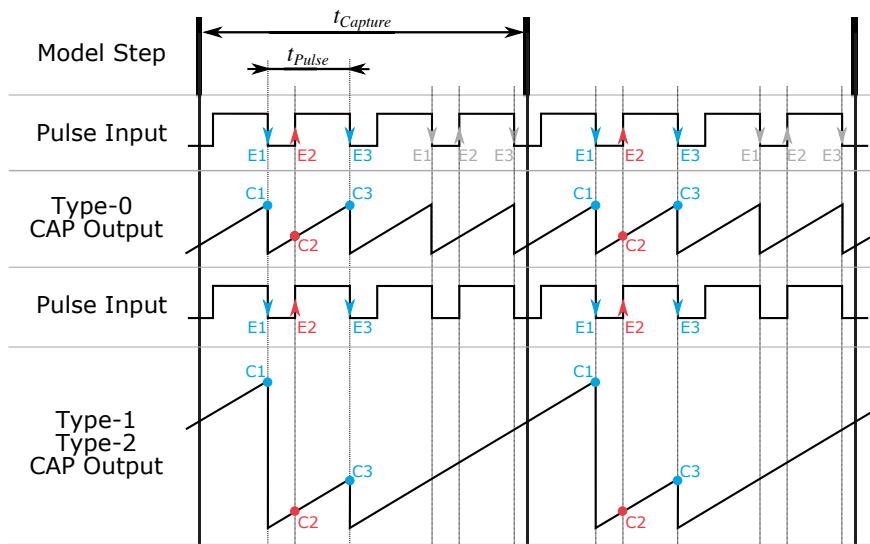
With Type-0 eCAP, as implemented in TI2806x, TI2833x and TI2837x MCUs, the reset actions of events continue even after a full set of events is received.

With Type-1 eCAP, as present on TI28004x MCUs, once a full set of events is received, the reset actions of events do not continue. The counter will therefore count-up without resetting until after the next model step, when the capture results are read and the module is re-armed, resulting in a large count value for the first event.

Type-2 eCAP, as present on TI2838x MCUs, behaves like Type-1 eCAP from a reset event perspective.

The figure below illustrates this with an example for the three events described in the following table:

Event	Trigger	Reset Counter	CAP Output
Event 1 (E1)	Falling	True	C1
Event 2 (E2)	Rising	False	C2
Event 3 (E3)	Falling	True	C3



**An illustrative example of Type-0 and Type-1 CAP outputs**

In the figure above,  $t_{Capture}$  is the execution step size of the eCAP block and  $t_{Pulse}$  is the period of the captured pulse train.

In case of Type-0 eCAP, even after the full set of events E1, E2 and E3 are received, the reset actions for the next set of events within the same model step continue. Therefore, in this case, the outputs C1 and C3 are the same.

In case of Type-1 and Type-2 eCAP modules, once the full set of events E1, E2 and E3 are received, the counter continues to count-up until the next trigger event in the next model step, resulting in a large value for C1.

## Parameters

## Main

### **CAP module**

Selects the eCAP module to use.

### **Input GPIO number**

Defines the GPIO pin number associated with the chosen eCAP module.

### **Prescaling**

Provides an option to disable or enable prescaling an input capture signal (pulse train).

### **Prescale value**

Specifies the desired prescale value. A pulse train can be prescaled by  $N = 2-62$  (in multiples of 2).

## Events

### **Event**

Provides an option to define four capture events. Polarity can be set to trigger on Rising or Falling edge. Unused events can be Disabled.

### **Reset counter on Event**

Events can be captured in absolute time-stamp mode with reset counter set to false or in time-difference mode with reset counter set to true.

## 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.

### **eCAP type**

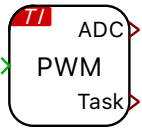
Allows choosing either a Type-0 or a Type-1 and above eCAP behavior for offline simulation.

# PWM

**Purpose** Generate a complementary PWM signal pair

**Library** TI C2000

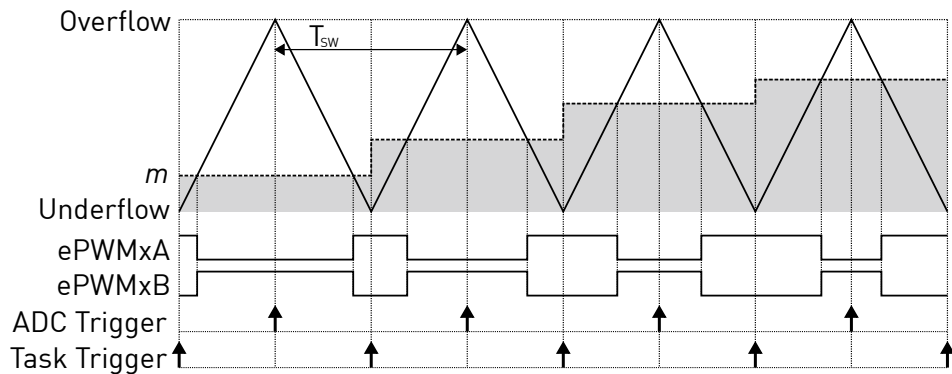
**Description**



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. During an offline simulation it behaves as a normal PWM generation block.

The PWM block can configure independent interrupts to trigger the ADC start-of-conversion and the Control Task Trigger. Interrupts are synchronized with the PWM carrier and will occur at the carrier underflow, overflow, or underflow and 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 task trigger set to underflow, the ADC trigger set to overflow, and the polarity configured with an active state logic of '1'.



**PWM and trigger schemes for symmetric carrier**

## **Main**

### **PWM generator(s)**

Index of the PWM resources. For vectorized output signals a vector of output channel indices must be specified.

### **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**

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

### **Polarity**

Defines the logical output of the ePWMxA output when an active state is detected. The active state occurs when the modulation index exceeds the carrier. Note that ePWMxB is always complementary to ePWMxA.

## **Output**

### **Mode**

- Complementary outputs operates channels A & B in complementary fashion with blanking time.
- Single output (channel A) only modulates channel A and allows the GPIO of channel B to be used for other purposes.
- Outputs disabled does not generated any PWM outputs. The GPIO of both channels remain free to be used for other purposes.

### **Blanking time**

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

### **Polarity**

Defines the logical output of the ePWMxA output when an active state is detected. The active state occurs when the modulation index exceeds the carrier. Note that ePWMxB is always complementary to ePWMxA.

## **Sequence**

- A Positive sequence corresponds to the PWM pattern as before, which starts channel A with the active state.
- The Negative sequence starts channel A with the passive state, resulting in a pattern that is phase shifted by 180 degrees compared to the positive sequence.
- An additional component port is created if Sequence is set to Variable. Applying a signal  $> 0$  to this port sets the sequence to positive.

### **Enable port**

An additional component port is created if **Enable port** is set to Show. Applying a signal  $= 0$  to this port sets the PWM channel(s) to passive.

### **Events**

#### **ADC trigger**

Configures the ADC trigger output.

#### **ADC trigger divider**

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

#### **Task Trigger**

Configures the control task trigger output.

#### **Task trigger divider**

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

### **Deprecated**

This tab contains deprecated settings and is normally disabled.

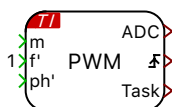


## PWM (Variable)

**Purpose** Generate a complementary PWM signal pair with a variable phase shift, variable frequency and synchronization options

**Library** TI C2000

### Description

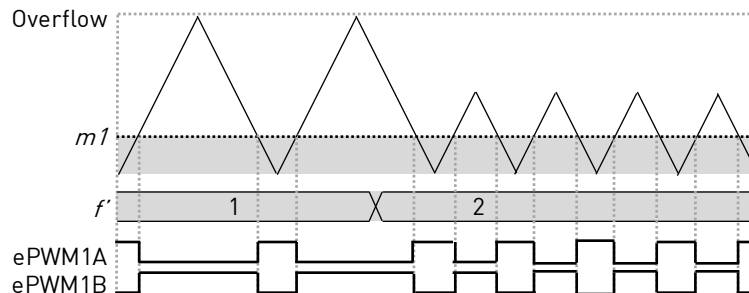


The PWM (Variable) block generates a complementary PWM pair on a grouping of one to three PWM channels that share a common synchronization impulse. The modulation index for each channel must be provided via the input signal  $m$ , which is a vectorized signal if the block uses more than one PWM channel. The carrier starts at 0 and varies between 0 and 1. The phase shift between the carriers of the individual PWM channels can be controlled with the vectorized input signal  $ph'$ . Each element of  $ph'$  specifies the phase delay of the PWM carrier with the same index. The delay is given in p.u. of the carrier period and must lie between 0 and 1.

The carriers of PWM resources are connected to a common synchronization signal, configured in the **Sync + Synchronization impulse from** setting. The synchronization signal can come from carrier zero count of the first PWM resource of the block, from another PWM (Variable) block, or from an external source using a GPIO pin. When an external GPIO synchronization signal is used an External Sync block is required. Each PWM (Variable) block also has a synchronization output that can be connected to other PWM (Variable) blocks.

Each group of PWM generators has the first channel configured as the master and the other channels configured as slaves. When a synchronization impulse is received, the master transmits a synchronization signal to the slaves of the same block and other PWM (Variable) blocks connected to the blocks synchronization output signal. When this happens, the ramp generators of the slaves are set to their initial values computed from the input signal  $ph'$  to achieve the desired phase shift.

The first element of the input signal  $ph$  corresponds to the phase delay of the master channel and is relevant only when the synchronization source is another PWM (Variable) block or an external GPIO. The phase delays between multiple PWM (Variable) blocks are only accurate if the blocks have a common **Carrier frequency**.



## PWM with frequency variation

Configure the **Frequency variation** parameter to enable the frequency input port  $f'$ . The figure above shows an example of the symmetric PWM carrier with frequency variation. The output frequency is given by  $f'$  multiplied by the **Carrier frequency** parameter. Note that all PWMs channels within one PWM (Variable) block share the same frequency input.

Note that the variable frequency operation with synchronized and phase-shifted units can only be achieved on the newer TI processors, but not the TI 28335 and 28069 devices. On the TI 28335 and 28069 devices, either the variable frequency port ( $f'$ ) can be enabled or a non-zero phase-shift ( $ph'$ ) can be configured, but not both at the same time.

The PWM (Variable) block can configure independent interrupts to trigger the ADC start-of-conversion and the Control Task Trigger. Interrupts are synchronized with the master channel PWM carrier and will occur at the carrier underflow, overflow, or underflow and overflow events. Underflow and overflow events correspond to PWM carrier reaching the carrier minimum and carrier maximum values.

## Parameters

### Main

#### Number of synchronized PWMs

The number of PWMs synchronized to a common interrupt.

#### Group of 3 PWM generator(s)

Selects the PWM resources that share a common interrupt.

#### Carrier type

Selects the carrier waveform, either sawtooth or symmetrical.

**Carrier frequency**

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**

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

**Polarity**

Defines the logical output of the ePWMxA output when an active state is detected. The active state occurs when the modulation index exceeds the carrier. Note that ePWMxB is always complementary to ePWMxA.

**Frequency variation**

Enables or disables the frequency input port.

**Sync****Synchronization impulse from**

Selects the source of the synchronization impulse. When self synchronization is chosen, the phase shift of the first allocated PWM resource has no impact.

**Events****ADC trigger**

Configures the ADC trigger output.

**Task Trigger**

Configures the control task trigger output.

**Deprecated**

This tab contains deprecated settings and is normally disabled.

## Quadrature Encoder Counter (QEP)

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

**Library** TI C2000

### 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 QEP inputs of the C2000 target.

The block outputs the current counter value (*c*), the index pulse (*i*), and the latched counter value from the previous index pulse (*ic*).

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*.

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 **Counter reset method** parameter, the counter is also reset when the rising edge of the index input is detected.

### Parameters

#### QEP module

Selects the QEP peripheral module used.

#### GPIO numbers

Defines the A,B, and I GPIO pins assigned to the chosen QEP module.

#### 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.

#### Counter reset method

Selects whether the counter should be reset by a positive pulse on the index input or on overflow only.

## Read Probe

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

**Library** TI C2000

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



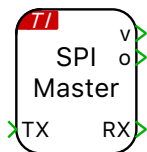
For further details on the PIL simulation, refer to the PIL User Manual.

## SPI Master

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

**Library** TI C2000

### Description



The Serial Peripheral Interface (SPI) is a high-speed synchronous serial input/output device that allows a serial bit stream of programmable length (1 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.

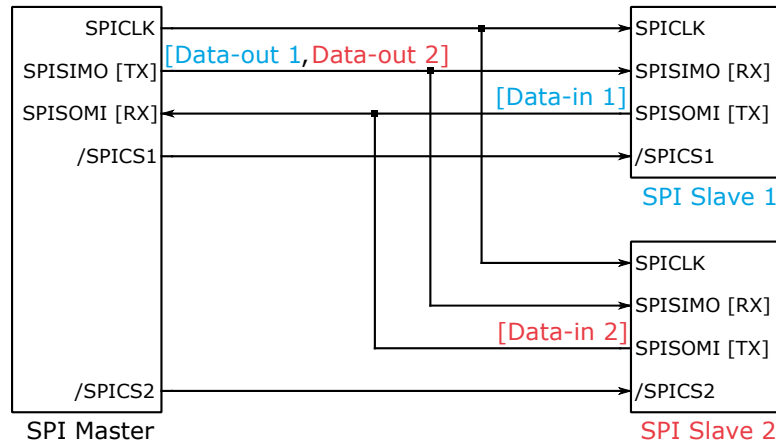
The SPI is a master-slave based interface with a single master and one or more slave devices.

The interface consists of the following signals:

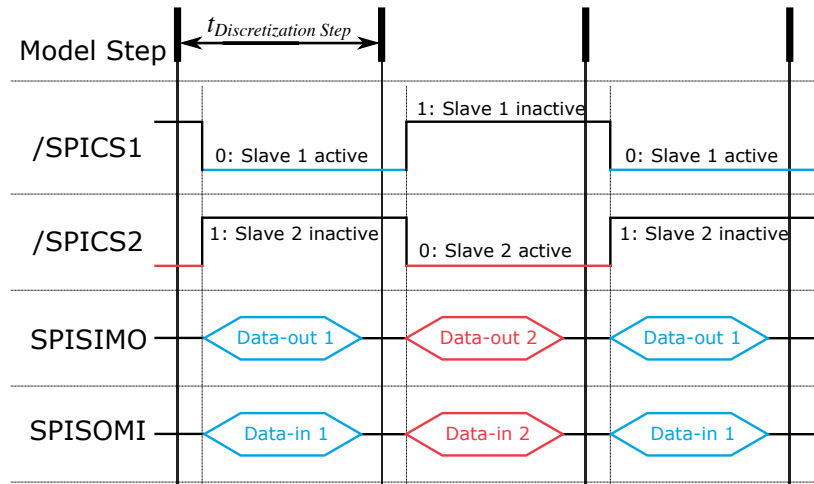
- **SPISOMI** Serial data input (as master in/slave out)
- **SPISIMO** Serial data output (master out/slave in)
- **SPICLK** Shift-clock, generated by the SPI Master
- **/SPICS** Chip-select or slave-enable signal, also referred to as SPISTE. The chip-select signal is an active-low signal that enables the SOMI and SIMO ports of the SPI Slave

The SPI Master block provides a clock signal (SPICLK ) 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 SPICLK 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 SPICLK 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 SPICLK 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

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

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

## Parameters

### Main

#### SPI module

Selects the SPI module to use.



**Clock rate**

Defines the SPI clock frequency (SPICLK), also known as the SPI Baud Rate, in Hz.

Refer to the TI technical reference for more information on the range of achievable SPI clock rates. This range is dependent on LSPCLK, a low-speed peripheral clock frequency, which is device-specific.

LSPCLK is derived by dividing the SYSCLK with a prescaler. SYSCLK is the main high speed clock of the CPU, configured in the Target tab of the Coder Options dialog. The attached table shows the hard-coded values of LSPCLK for all the TI C2000 targets.

TI C2000 Target	LSPCLK prescaler
TI2806x	/4
TI28004x	/4
TI2833x	/6
TI2837x	/4

**Bits per word (1-16)**

Defines the length of a single data word during transmission. The allowed length is 1 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 according to TI's convention:

**SPI Clocking Modes**

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

**Serial GPIO numbers [SIMO, SOMI, CLK]**

Selects the GPIOs to use for SPISIMO, SPISOMI and SPICLK respectively.

**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.

**Slave(s)****/CS GPIO number(s)**

Selects the GPIOs to use for chip-select. Any GPIO, not just SPISTE, can be configured to be a /CS signal. Provide a vector to configure multiple slaves.

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

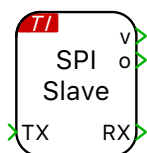
Configures the number of transmitted data words in each simulation step. Provide a vector to configure multiple slaves.

## SPI Slave

**Purpose** Implement SPI Slave

**Library** TI C2000

### Description



For a detailed description of the SPI protocol and signals, refer to the SPI Master (see page 66) 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 SOMI and SIMO 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

#### Main

##### SPI module

Selects the SPI module to use.

##### Bits per word (1-16)

Defines the length of a single data word during transmission. The allowed length is 1 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 according to TI's convention:

**SPI Clocking Modes**

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

**Words per transmission**

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

**Serial GPIO numbers [SIMO, SOMI, CLK, /CS]**

Selects the GPIOs to use for SPISIMO, SPISOMI, SPICLK and /SPICS respectively.

**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.

## Timer

**Purpose** Generate trigger signals for the ADC start-of-conversion and the control task using the CPU Timer

**Library** TI C2000

### Description



The Timer block configures the CPU Timer 0 interrupt to occur 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.

plexim  
electrical engineering software

---