THE SIMULATION PLATFORM FOR POWER ELECTRONIC SYSTEMS

TI C2000 Target Support User Manual June 2019

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
	`http://www.plexim.com`	Web

TI C2000 Target Support User Manual

Contents

1

Quick Start

Requirements

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

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

- a host computer (with Microsoft Windows, Mac OS X or Linux),
- PLECS Blockset or Standalone 4.3 or newer and
- PLECS Coder.

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

Install the Target Support Package

Download the appropriate ZIP archive, extract it and move the `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 are 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 are also included in the `demo` folder of the TI C2000 Target Support Package. All of the 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`. 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, but it requires using an external application to build and flash the target device.

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

Download and install C2000 Code Gen Tools and UniFlash. The latest releases are available online:

UniFlash: `http://www.ti.com/tool/UNIFLASH`
C2000 CGT: `http://www.ti.com/tool/C2000-CGT`

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.

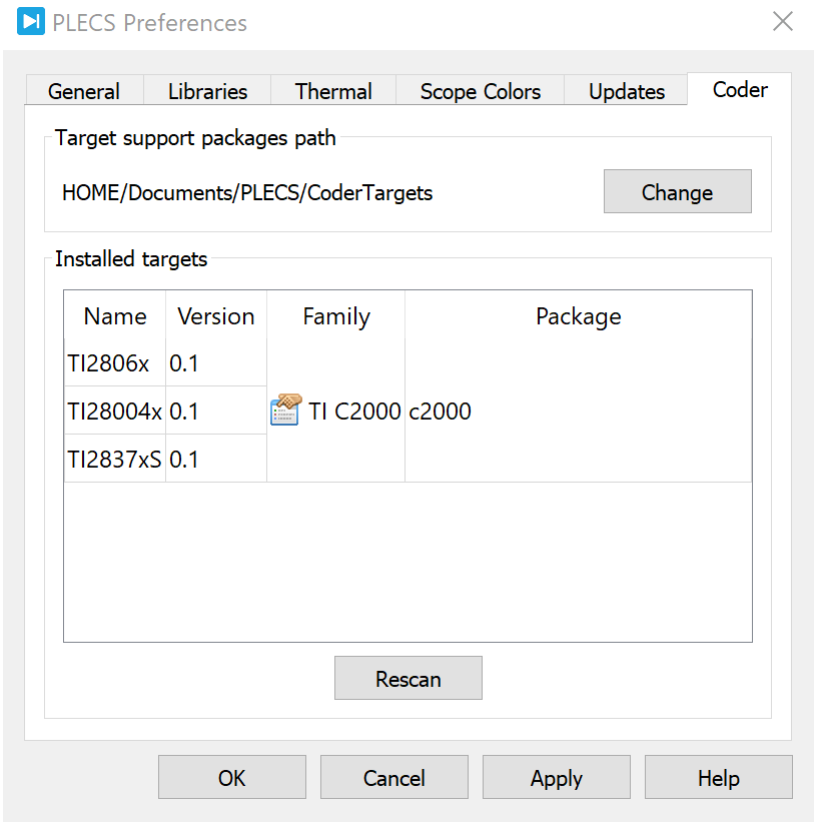


Figure 1.1: Configuring the target support package and external tool paths

After downloading the target configuration file navigate back to the PLECS **Coder Options + Target** window, set the **Board** field to `Custom`. The **Uni-Flash 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 diagnos-

tics window will contain additional debugging information. The diagnostics 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 from the TI website. The most recent CCS version is available at the following location:

```
http://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, extract the contents of the relevant `projects/28xx.zip` archives. Open CCS and click the **Project** drop-down menu and then select **Import CCS Projects...**. Choose the uncompressed `projects/28xx` folder that corresponds to the desired target. You will notice a new project created in your CCS workspace.

Next, in the **Project Explorer** tab of CCS select the `ccxml` file for your target configuration. Modify any required settings and test the connection with your MCU. Navigate to the CCS project build settings in the **Project + Properties** drop-down menu. Select the **CCS Build** option in the left hand menu and then open the **Steps** tab. Modify the post-build steps configuration depending on your operating system as shown below.

Windows X Post-Build Step:

```
${workspace_loc:${ProjName}/cg/buildsteps.bat} <âĂę>
```

Mac OS X Post-Build Step:

```
source ${workspace_loc:/${ProjName}/cg/buildsteps.sh} <âĂę>
```

In the post-build steps above, the `<...>` string represents the rest of the default build steps in the provided project. 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.

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

Jumper	Position	Purpose
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

Switch	Position	Purpose
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

Jumper	Position	Purpose
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

Switch	Position	Purpose
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

Switch	Position	Purpose
S1-SW1	On/Off	GPIO84 logic level for boot mode configuration
S1-SW2	On/Off	GPIO72 logic level for boot mode configuration

LAUNCHXL-F2837x Boot Mode DIP Switch Settings (contd.)

Switch	Position	Purpose
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.

2

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

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.

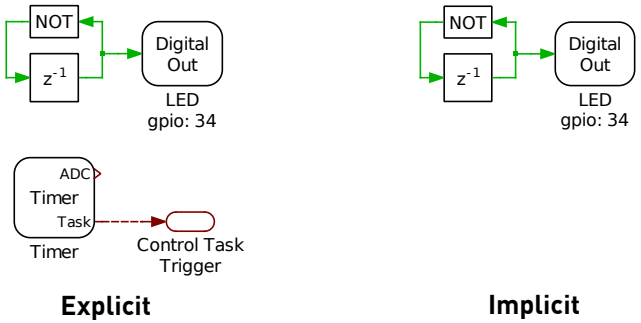


Figure 2.1: Basic model with control task triggered by CPU Timer

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.

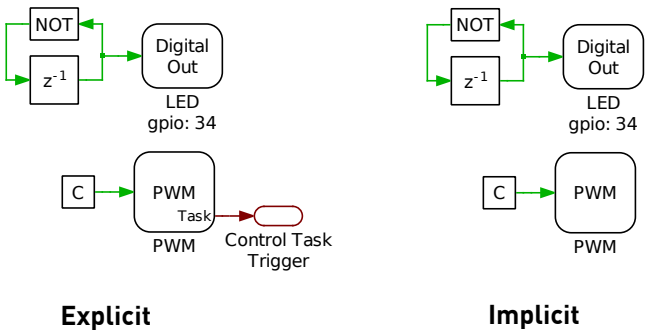


Figure 2.2: Basic model with control task triggered by PWM

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.

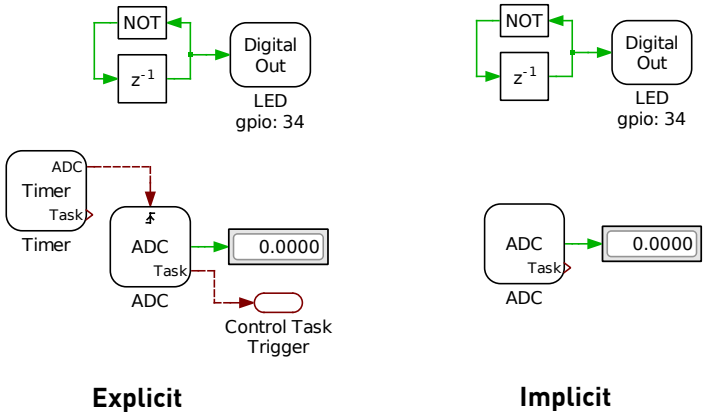


Figure 2.3: Basic model with control task triggered by ADC

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.

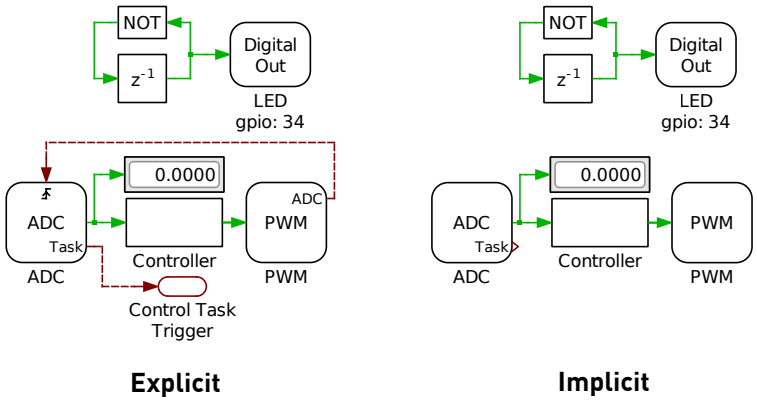


Figure 2.4: Basic model with control task triggered by PWM via ADC

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.

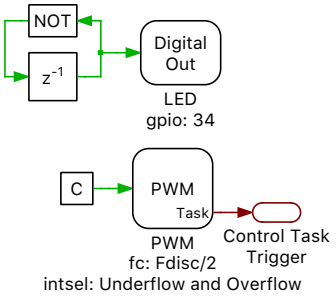


Figure 2.5: PWM frequency set to half the control task frequency

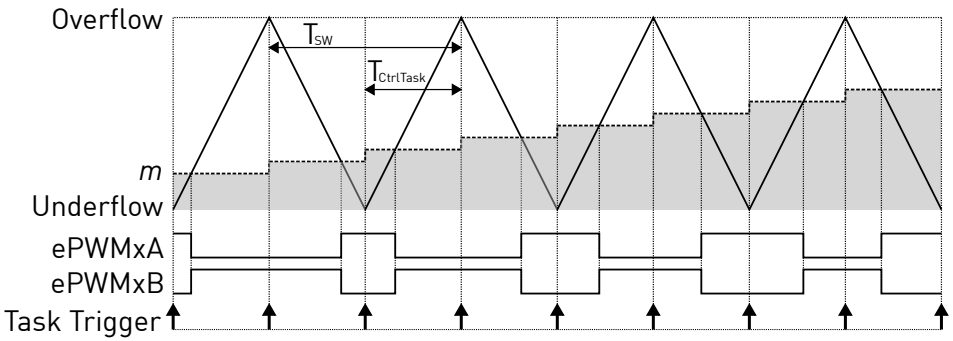


Figure 2.6: PWM carrier and task interrupts for PWM frequency set to half the control task frequency

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.

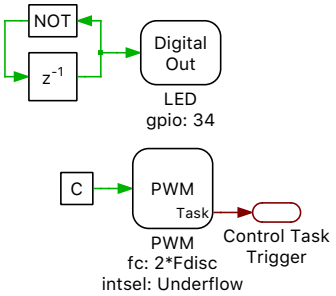


Figure 2.7: Schematic of PWM frequency set to twice the control task frequency

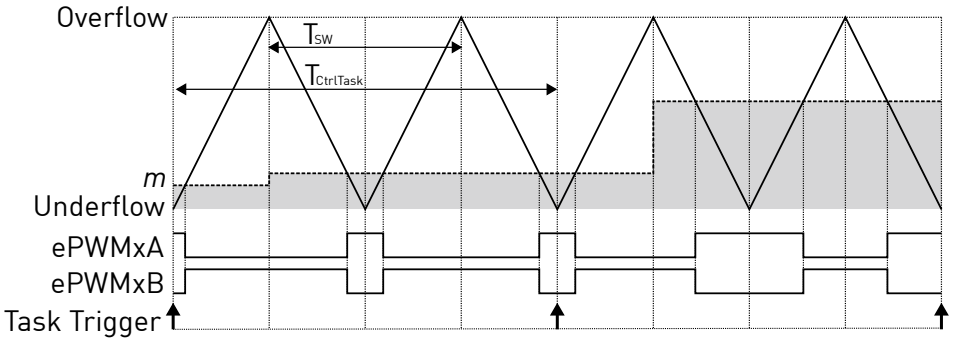


Figure 2.8: PWM carrier and task interrupts for PWM frequency set to twice the control task frequency

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.

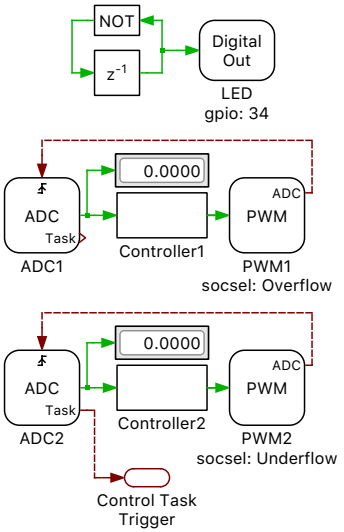


Figure 2.9: Explicit phase-shifted ADC sampling

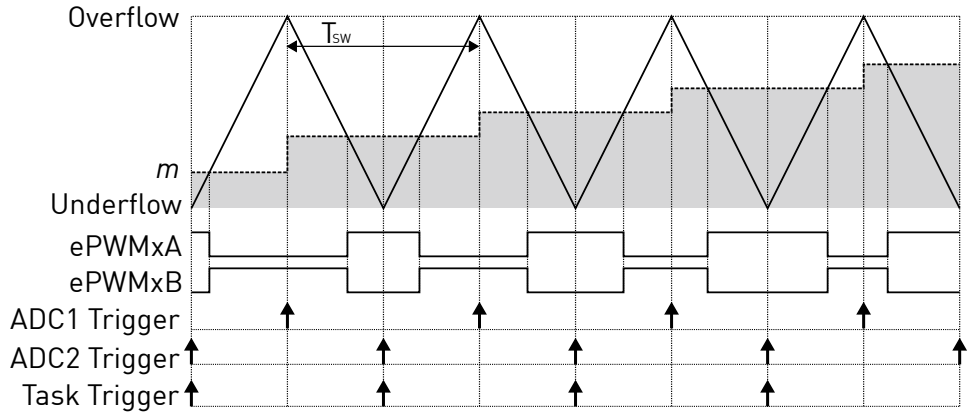


Figure 2.10: PWM carrier and interrupts for phase-shifted ADC sampling

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_loc}/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 provided application utilizes a nested task structure to ensure predictable and reliable execution of the digital control loop. There is one fast high-priority task that calls the control code generated from the PLECS model, referred to as the control task, and a slower lower-priority task, referred to as the infrastructure task. In the provided code generation project the infrastructure task implements a finite state machine (FSM) for power stage control. A low priority background task is also used to handle non-time critical tasks. Figure 2.11 shows the configuration with control task, infrastructure task, and a background task executing in real-time on the MCU.

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

the highest priority. In addition, the lower priority task is periodically triggered using a software interrupt, for example on first and fourth hardware interrupts in the figure below. In practice the frequency of the infrastructure task software interrupt is defined in the **Coder + Coder Options + Infrastructure task frequency** setting. Once the control and infrastructure tasks have completed, the system continues with the background task where lowest priority operations are processed.

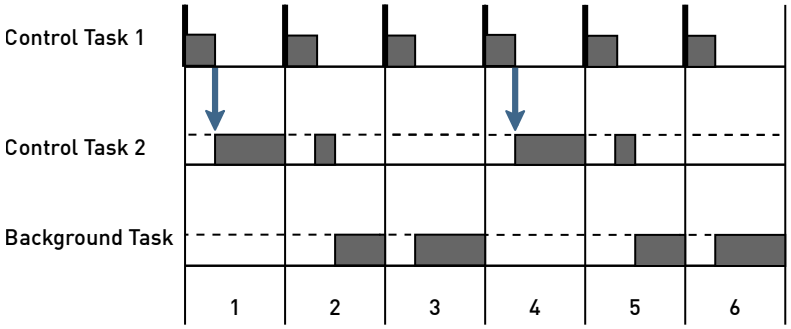


Figure 2.11: Nested control tasks

If the control task is executing and a second control task interrupt is received then the processor will halt and an assertion will be generated. This will happen when the control task execution time exceeds the nominal control task interrupt period. Similar behavior occurs if a second infrastructure task software interrupt is received prior to completion of the previous infrastructure task execution. Assertions can be monitored using CCS debug tools. The optimization settings of the CCS project can be tuned to improve the controller execution time in some instances.

Embedded project architecture

Figure 2.12 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 infrastructure and control tasks. Next, there is a minimal real-time operating system that consists of a dispatch routine that handles 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.

Note that in this above architecture, only the `CONTROL_INIT` and `CONTROL_STEP` functions are dynamically generated from the PLECS Coder; the bulk of the project is static.

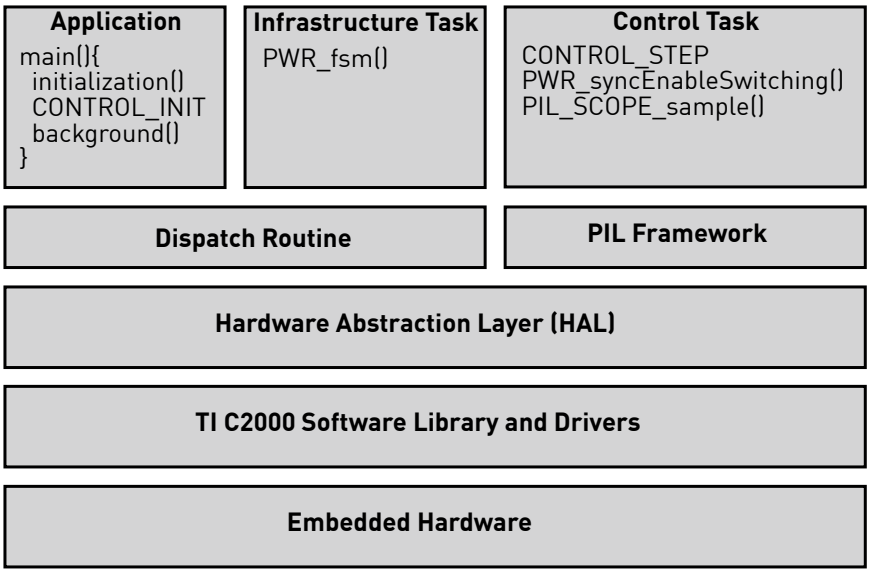


Figure 2.12: Embedded project architecture

3

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.

Infrastructure task frequency Specifies the frequency of the infrastructure task in hertz (Hz).

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

Build type This setting specifies the action of the **Build** button. `Generate code into CCS project` will generate code into the specified Code Composer Studio (CCS) project. CCS must 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 an option to choose the appropriate target MCU board. If using either a TI LaunchPad or a TI controlCARD, `LaunchPad` or `ControlCard` should be selected. If using a `Custom` board instead, a UniFlash target configuration file must be generated.

UniFlash target configuration Defines the `.ccxml` target configuration file for the target device. 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.

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.

4

TI C2000 Target Support Library Component Reference

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

ADC

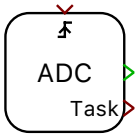
Purpose

Output the measured voltage of the ADC peripheral.

Library

TI C2000

Description



This component configures the ADC peripheral as a single-ended input with an internal bandgap reference voltage of 3.3 V. The ADC component output signal represents the measured voltage of the ADC 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 input*Scale+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.

Control Task Trigger

Purpose

Specifies the nominal 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.

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 controls in seconds. The nominal base sample time is synchronized with the model **Discretization step size** of the PLECS Coder settings. This parameter must align with the period of the input signal.

CPU Load

Purpose

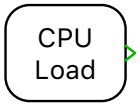
Provide the CPU load in percent.

Library

TI C2000

Description

This block outputs the percentage of time that is used by the control task with one interrupt period.



DAC

Purpose

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

Library

TI C2000

Description

This component 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 input*Scale+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)

GPIO resource of the digital input channel. For vectorized input signals a vector of input channel indices must be specified.

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)

GPIO resource of the digital output channel. For vectorized output signals a vector of output channel indices must be specified.

External Sync

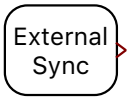
Purpose

External synchronization port for PWM output.

Library

TI C2000

Description



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

Parameters

External GPIO

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

Powerstage Protection

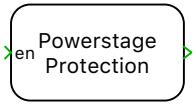
Purpose

Provides powerstage safety features.

Library

TI C2000

Description



The Powerstage Protection block implements a finite state machine (FSM) to enable or disable all PWM outputs on the target device. The PWM outputs are disabled unless there is a logical low to high transition on the input signal. A logical high to low transition will disable the PWM outputs. When the powerstage is enabled the output of the Powerstage Protection component will be set to logical high and an active low GPIO pin, defined in the **Powerstage enable GPIO number** parameter, will also be set.

Each PWM output can be independently assigned to different trip zones. A trip event is detected when the GPIO pin assigned to the trip zone input is driven to logical low. The action following a trip event is assigned in the PWM component settings.

When the PWM outputs are disabled via the Powerstage Protection input all output signals are set to the PWM safe state. When a trip zone is triggered only the PWM outputs associated with the trip zone are set to the PWM safe state. The advantage of the using the trip zone logic is the PWM output will be set to the PWM safe state prior to the execution of the next control task interrupt.

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

Parameters

Powerstage enable GPIO number (active low)
Specifies a GPIO pin set logical low when the powerstage is enabled.

PWM safe state
Specifies the forced PWM output state when a trip zone GPIO is triggered. Selecting the forced inactive option drives all associated PWM outputs to logical low. Selecting the floating option sets the associated PWM outputs to a high impedance state.

TZ1 GPIO number
Set a GPIO corresponding to trip zone 1. The GPIO is configured as an active low input; a logical low input will activate the trip zone logic.

TZ2 GPIO number
Set a GPIO corresponding to trip zone 2. The GPIO is configured as an active low input; a logical low input will activate the trip zone logic.

TZ3 GPIO number

Set a GPIO corresponding to trip zone 3. The GPIO is configured as an active low input; a logical low input will activate the trip zone logic.

PWM

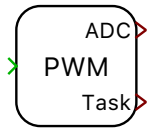
Purpose

Generate a complementary PWM signal pair.

Library

TI C2000

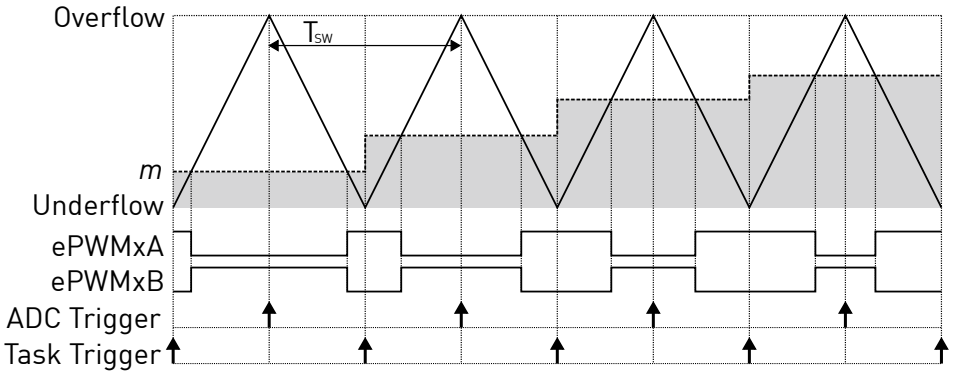
Description



The PWM block generates a 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

The trip zone submodule can be used to disable the powerstage following a trip event. Trip events are detected when there is an active low condition on the trip zone GPIO inputs assigned in the Powerstage Protection component. The PWM block protection parameters configure the trip settings for all

PWM resources associated with the component. When a trip event is detected the PWM 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 when a trip zone is activated. 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 configured in the Powerstage Protection block.

Parameters

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
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.

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.

Protection

TZ1 Mode
Select the action following trip event detection.

TZ2 Mode
Select the action following trip event detection.

TZ3 Mode
Select the action following trip event detection.

PWM (Variable)

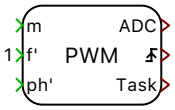
Purpose

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

Library

TI C2000

Description

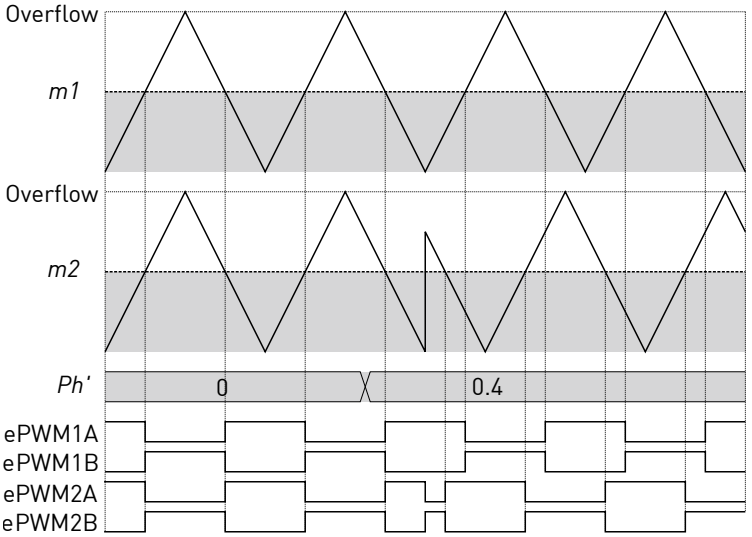


The PWM (Variable) block generates a complementary PWM pair on a grouping of one to three PWM channels that share a common synchronization impulse. The modulation index for each channel must be provided via the input signal m, which is a vectorized signal if the block uses more than one PWM channel. 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 component, 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 component is required. Each PWM (Variable) block also has a synchronization output that can be connected to other PWM (Variable) components.

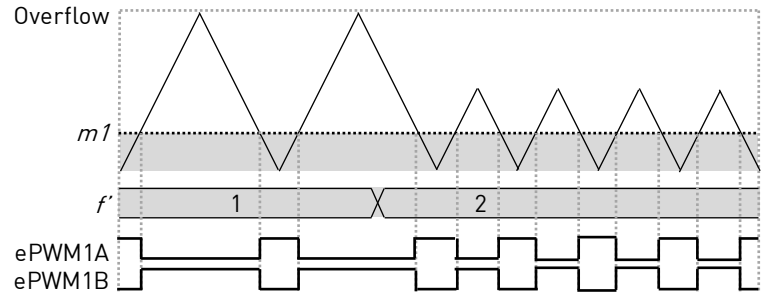
Each group of PWM generators has the first channel configured as the master and the other channels configured as slaves. When a synchronization impulse is received, the master transmits a synchronization signal to the slaves of the same block and other PWM (Variable) blocks connected to the blocks synchronization 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 figure below shows an example of the symmetric PWM carrier with two self-synchronized PWMs and a the polarity set to an active state logic of '1'.



Two channel symmetrical PWM with self-synchronized phase shift

The first element of the input signal ph corresponds to the phase delay of the master channel and is accurate only when the synchronization source is another PWM (Variable) component 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.

The PWM (Variable) component 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.

The trip zone submodule can be used to disable the powerstage following a trip event. Trip events are detected when there is an active low condition on the trip zone GPIO inputs assigned in the Powerstage Protection component. The PWM block protection parameters configure the trip settings for all PWM resources associated with the component. When a trip event is detected the PWM 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 when a trip zone is activated. 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 configured in the Powerstage Protection block.

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.

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
Enable or disable 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.

Protection

TZ1 Mode
Select the action following trip event detection.

TZ2 Mode
Select the action following trip event detection.

TZ3 Mode
Select the action following trip event detection.

Quadrature Encoder Counter (QEP)

Purpose

Counts 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.

Timer

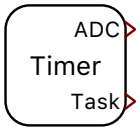
Purpose

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

Library

TI C2000

Description



The Timer component 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
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

Plexim GmbH info@plexim.com www.plexim.com