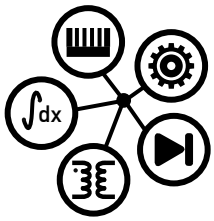

Embedded
Code Generation

*DEMO MODEL*

# Simple CAN Model

**A simple model to explore CAN communication with the TI C2000 TSP**

Last updated in C2000 TSP 1.4.5

**www.plexim.com**

- Request a PLECS and PLECS Coder trial license
- Get the latest TI C2000 and RT Box Target Support Package
- Check the PLECS, RT Box and TI C2000 TSP documentation

# 1 Overview

This demo features a simple model using the Controller Area Network (CAN) blocks on Texas Instruments (TI) C2000 microcontrollers (MCUs) with the PLECS Coder and the TI C2000 Target Support Package.

A CAN bus is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each without a central host computer.

The model is split into eight distinct subsystems, corresponding to four different MCU targets, as shown in Fig. 1. Each subsystem can be independently deployed to the corresponding TI C2000 LaunchPad hardware. The following sections provide a brief description of the model and instructions on how to simulate it.

### Requirements

There is one CAN interface with an integrated transceiver available on each of the LaunchPad targets. Therefore, to explore the CAN communication, this model requires any two LaunchPad targets, as described in Section 2.

# 2 Model

The top level schematic contains eight separate subsystems, as shown in Fig. 1, corresponding to four different C2000 targets. The subsystem with an extension “tx” is configured to transmit CAN messages, whereas the subsystem with an extension “rx” is configured to receive CAN messages. The subsystems labeled “28069” are configured for the TI 28069 LaunchPad [1], the subsystems labeled “280049” are configured for the TI 280049C LaunchPad [2], the subsystems labeled “28377S” are configured for the TI 28377S LaunchPad [3] and lastly the subsystems labeled “28379D” are configured for the TI 28379D LaunchPad [4].

Each subsystem is enabled for code generation, as indicated by the thick outer border of the subsystem blocks. This step is necessary to generate the model code for a subsystem via the PLECS Coder. This setting is configured by selecting the subsystem, opening **Edit + Subsystem + Execution settings...** menu, and then selecting the **Enable code generation** option.

The generated code runs at a base sample time on the MCU or the **Discretization step size**. In this model, the discretization step size of each of the subsystems is set to 100 $\mu$s.

Each of the LaunchPad targets is configured to either transmit or receive CAN messages, as shown in Fig. 2 and Fig. 3 respectively. To explore the CAN communication, choose any two desired targets; one to transmit and the other to receive CAN messages.

### CAN Port

The CAN Port block, shown in Fig. 2 and Fig. 3, sets up a CAN 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. A detailed description of the CAN error modes is given in the “Help” section of this block.

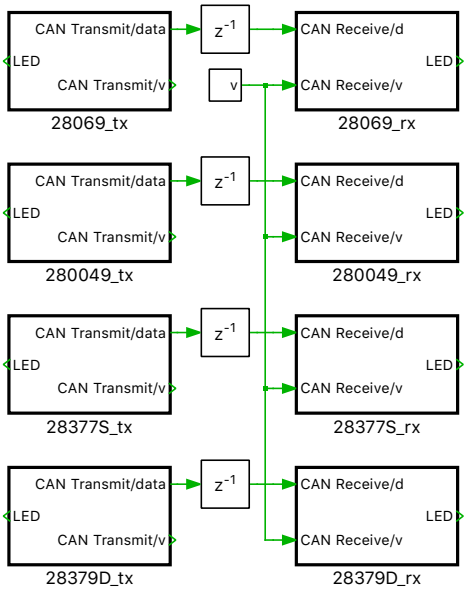


**Figure 1: Top level schematic of the model with four subsystems**

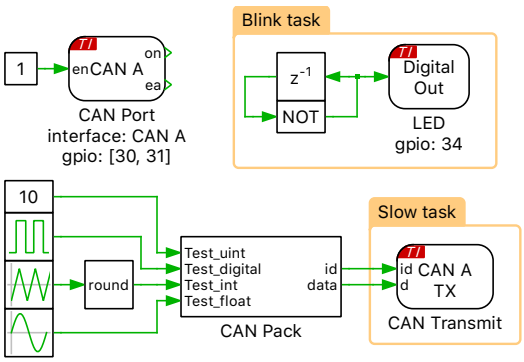


**Figure 2: Schematic of the 28069_tx subsystem to transmit CAN messages**

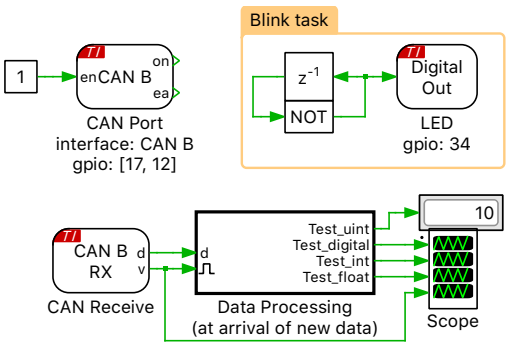


**Figure 3: Schematic of the 28379D_rx subsystem to receive CAN messages**

## CAN Transmitting

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).

The CAN Pack block generates a CAN message by packing the input signals into a byte vector, as

shown in Fig. 4. The **CAN ID** field specifies the ID of the CAN message. In this demo, it is set as `0x1`. The CAN ID can be supplied as either an 11-bit value (for CAN 2.0A compliance) or a 29-bit value (for CAN 2.0B compliance). A signal in a CAN message is defined by its data type, its byte order (Little Endian / Big Endian) and its start bit and length within the 64-bit CAN message. A signal can be scaled and an offset can be applied to efficiently send floating point signals as integers. The bits within the CAN message are numbered from 0 (least significant bit of the first byte) to 63 (most significant bit of the last byte).

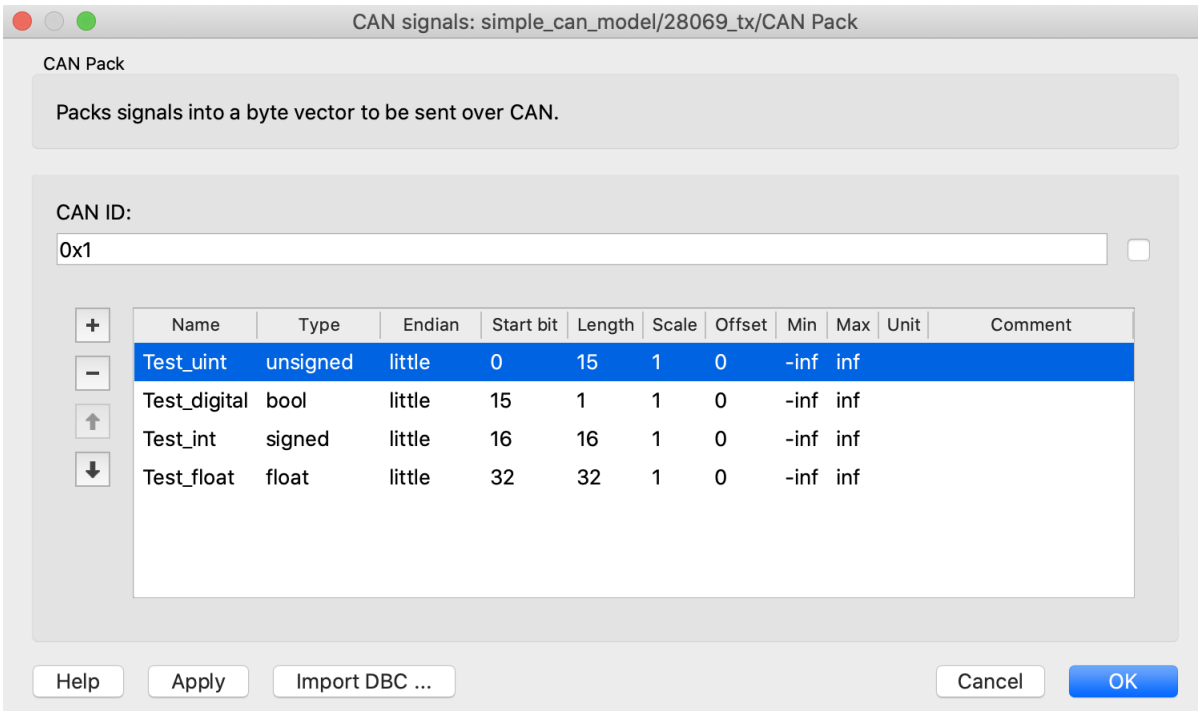


**Figure 4: Parameters of the CAN Pack block**

There is an option to import signal definitions from a DBC (data base CAN) file by clicking the **Import DBC ...** button. Most CAN networks are proprietary in the sense that only the Original Equipment Manufacturer (OEM) has the DBC file required to decode the data. This is the case, for example for raw CAN data in most cars, bikes, EVs, production machinery, etc.

In this demo model, we send out:

1. an `unsigned integer` type from bit 0 to bit 14 (15-bit length) provided by a Constant block,
2. a `bool` type digital signal on bit 15 provided by a Pulse Signal of 1 Hz and 0.5 duty cycle with values alternating between 0 and 1,
3. a 5 Hz Triangular Wave between -5 and +5 as a `signed integer` type from bit 16 to bit 31 (16-bit length),
4. a 5 Hz Sine Wave with amplitude of 1 as a `float` type from bit 32 to bit 63 (32-bit length).

This information is stored in a total of 64 bits which corresponds to the maximum amount of data in one package. Note that the Constant block for the first test signal is added into the **Exceptions** field of the **Parameter Inlining** tab of **Coder options...** window. This means that once the model is running on the embedded target, the value can be tuned on the fly in real-time via the External Mode.

For more details on CAN transmitting, click on the "Help" button of the CAN Transmit and CAN Pack block descriptions.

### CAN Receiving

The CAN Receive 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 for one simulation step when new data is received, 0 otherwise.

The CAN Unpack block decodes signals from a byte vector received over CAN into the original message. Its CAN ID and signal definitions should be set in the exact same way as the CAN Pack block shown in Fig. 4.

For more details on CAN receiving, click on the "Help" button of the CAN Receive and CAN Unpack block descriptions.

### Multi-tasking code

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. Multi-tasking code unlocks processing power for controls regulating multiple system outputs with dynamics on a range of time-scales. In this model multi-tasking code is used since CAN Transmit block can be executed at a slower rate.

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. Up to 15 slower lower priority tasks that execute at different rates can be specified, preserving processor time for the fastest, highest priority task in the application. For further information, refer to the "Code Generation" section in the PLECS User Manual [6].

Blocks in the PLECS schematic are assigned to lower priority tasks using the Task library component. In this model two lower priority tasks are defined in addition to the Base task, as seen in Fig. 2. The Base task is executed at 100 $\mu$s, the Slow task is executed at 0.01 s and the Blink task is executed at 0.5 s.

# 3 Simulation

Each subsystem can be directly converted into target specific code for the corresponding TI LaunchPad hardware.

---

**Note** Before proceeding, ensure the DIP switch position and jumper configuration on the LaunchPad device are correctly configured. Guidance for each LaunchPad device is provided in the “Tips for Programming C2000 LaunchPads” section of the TI C2000 Target Support User Manual [5].

---

### Connect the hardware

There is one CAN interface with an integrated transceiver available on each of the LaunchPad targets. The CAN terminals on the TI 28069, TI 28377S and TI 28379D are available through the connector J12; on TI 280049, they are available through the connector J14.

Choose any two LaunchPad targets as desired. Then, connect the CAN_H, CAN_L and GND pins of the two CAN interfaces together using jumper wires.

### Flash the MCU

Follow the instructions below to upload the subsystems to a TI MCU.

- Connect the desired MCU to the host computer through a USB cable.
- From the **System** tab of the **Coder + Coder options...** window, select the MCU of interest.
- Next, from the **Target** tab, select the appropriate target from the dropdown menu. Then under the **General** sub-tab, select the desired **Build type**.
- Then, to `Build and program` the MCU directly from PLECS, choose either `Run from Flash` or `Run from RAM` as the **Build configuration**, then select `LaunchPad` as the **Board** type, and click **Build**.

---

**Note** If programmed correctly, the LED on the LaunchPad board should blink.

---

For advanced users who are familiar with Code Composer Studio (CCS), there is an option to `Generate code into CCS project`. Included with the TI C2000 Target Support package is a folder titled `projects`. Within the folder there are ZIP archives containing pre-built CCS projects for each MCU. Import the zip archive folder that corresponds to the desired target into CCS. You will notice a new project created in your CCS workspace. Enter the location of the `${workspace_loc}/dev_28xx/cg/` folder from the CCS project into the **CCS project directory** field and click **Build**. Then, proceed to build and debug the project as a normal CCS project. Refer to "Quick Start" section of the TI C2000 Target Support User Manual [5] for detailed step-by-step instructions.

### External Mode

Once the generated code is running on the C2000 target, the user can enter the External Mode to update scopes and displays in the PLECS application with real-time waveforms and change certain simulation parameters. The steps below outline how to connect to the target device, with additional debugging details provided in the "Start the External Mode" section of the user manual [5].

- First, from the **System** menu on the left hand side of the **Coder + Coder options...** window, select the desired MCU.
- Then, from the **External Mode** tab, select the **Target device** by clicking  icon next to the **Target device** field.
- Next, click **Connect** and then **Activate autotriggering** to observe the results in the subsystem display and scope.

The received CAN waveforms on the 28379D target are shown in Fig. 5.

### Parameter Inlining

To configure parameters as tunable, open the **Coder + Coder options...** menu and navigate to the **Parameter Inlining** tab. When a component from the schematic is dragged and dropped into the **Exceptions** list, tunable parameters associated with that component will be tunable during runtime. Note this behavior depends on the **Default behavior** setting, as the **Exceptions** list specifies components which have opposite behavior of the default setting.

In this case, the "Test_unit" input to the CAN Transmit block can be adjusted on the fly when the model executes on the embedded target device, when connected to External Mode. Changes in the parameters will be reflected in the Scope traces and Display once they take effect.

# 4 Conclusion

This model explores the CAN communication with the TI C2000 TSP using a simple model.

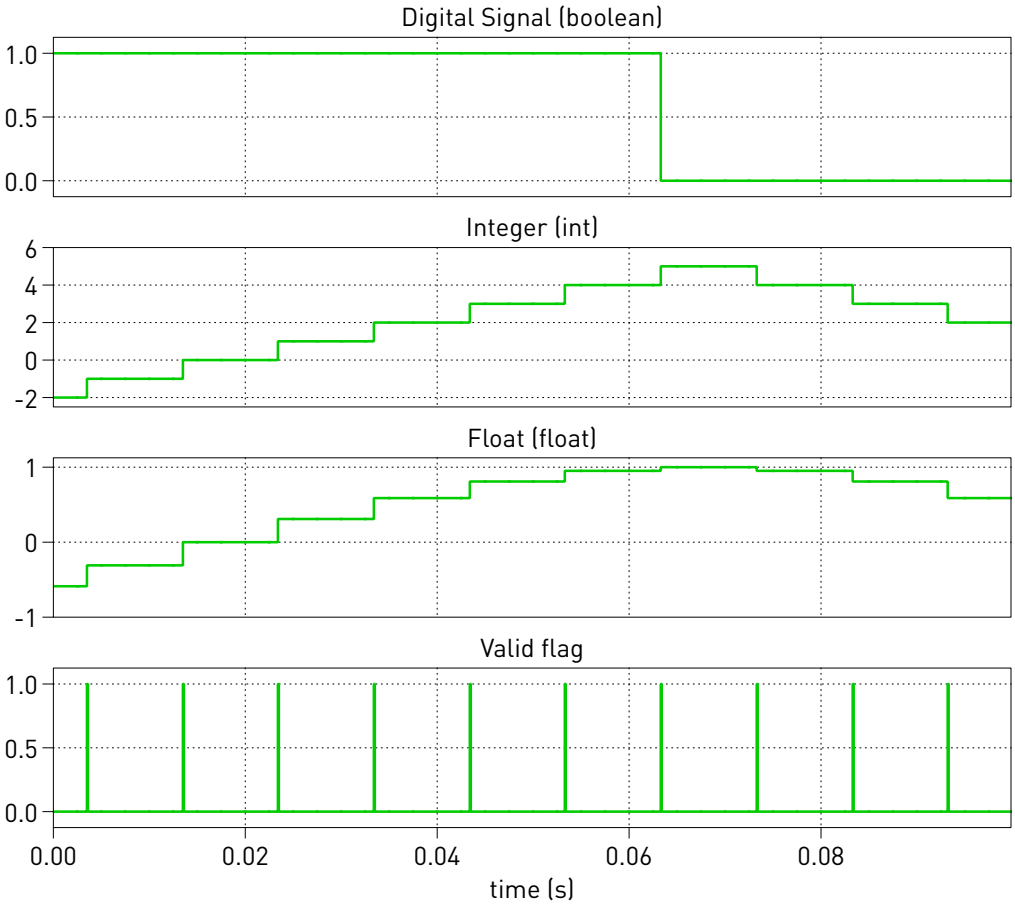


**Figure 5: Received CAN waveforms on the 28379D target**

# References

[1] TI C2000 Piccolo MCU F28069M LaunchPad Development Kit,
URL: http://www.ti.com/tool/LAUNCHXL-F28069M.

[2] TI C2000 Piccolo MCU F280049C LaunchPad Development Kit
URL: http://www.ti.com/tool/LAUNCHXL-F280049C.

[3] TI C2000 Delfino MCUs F28377S LaunchPad Development Kit
URL: https://www.ti.com/lit/pdf/sprui25

[4] TI C2000 Delfino MCUs F28379D LaunchPad Development Kit,
URL: http://www.ti.com/tool/LAUNCHXL-F28379D.

[5] PLECS TI C2000 Target Support User Manual,
URL: https://www.plexim.com/sites/default/files/c2000manual.pdf.

[6] PLECS User Manual,
URL: https://www.plexim.com/sites/default/files/plecsmanual.pdf.

**Revision History:**

| | |
|---|---|
| C2000 TSP 1.3.1 | First release |
| C2000 TSP 1.4.5 | Updated the web links |

**How to Contact Plexim:**


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


*Embedded Code Generation Demo Model*