Application Note

Using the Edde Flex CAN Controller on the EK-TM4C1294XL LaunchPad

TEXAS INSTRUMENTS

Abstract

This application report demonstrates how to use the TM4C CAN module for simple transmit and receive operations. It also includes the code and utilities needed to implement a CAN-based bootloader.

Project collateral and source code discussed in this application can be downloaded from the following URL: http://www.ti.com/lit/zip/spna245.

Table of Contents

Trademarks

1 Introduction

TM4C MCUs offer a broad array of wired connectivity peripheral including an integrated CAN bus which only requires external CAN transceivers. Texas Instruments low-cost evaluation platforms such as the EK-TM4C1294XL LaunchPad™ Evaluation Kit for Arm® Cortex®-M4F based microcontrollers allow for quick and affordable evaluation for many device features. However, the LaunchPad lacks the required on board CAN transceivers. The Edde Flex CAN BoosterPack provides the transceivers needed for the EK-TM4C1294XL to communicate over CAN.

2 Installation of Edde Flex CAN Booster Pack for TI TIVA C

Installation of the Edde Flex CAN booster pack on the EK-TM4C1294XL or EK-TM4C129EXL Launchpad is straight forward. The first step is to put the jumpers for JP4 and JP5 in the vertical position as shown in Figure 2-1.

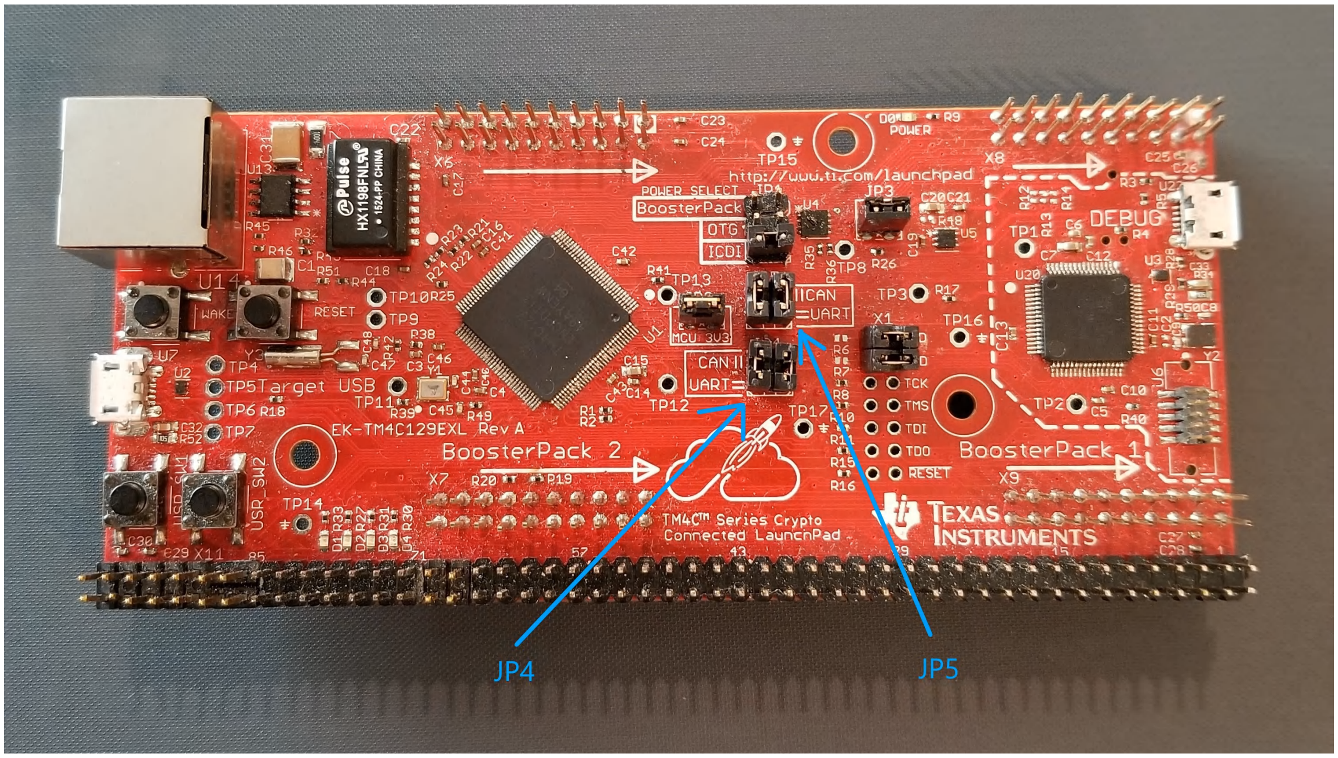


Figure 2-1. EK-TM4C1294XL Jumper Position for CAN

In this position, the CAN0RX (PA0) and CAN0TX (PA1) are routed to the BosterPack. UART2RX (PD4) and UART2TX (PD5) are then connected to the USB virtual serial port. All of the examples in this application report use CAN0 and UART2.

The Edde Flex BoosterPack is then placed on the BoosterPack 2 headers. Be careful to orient the BoosterPack as shown in Figure 2-2.

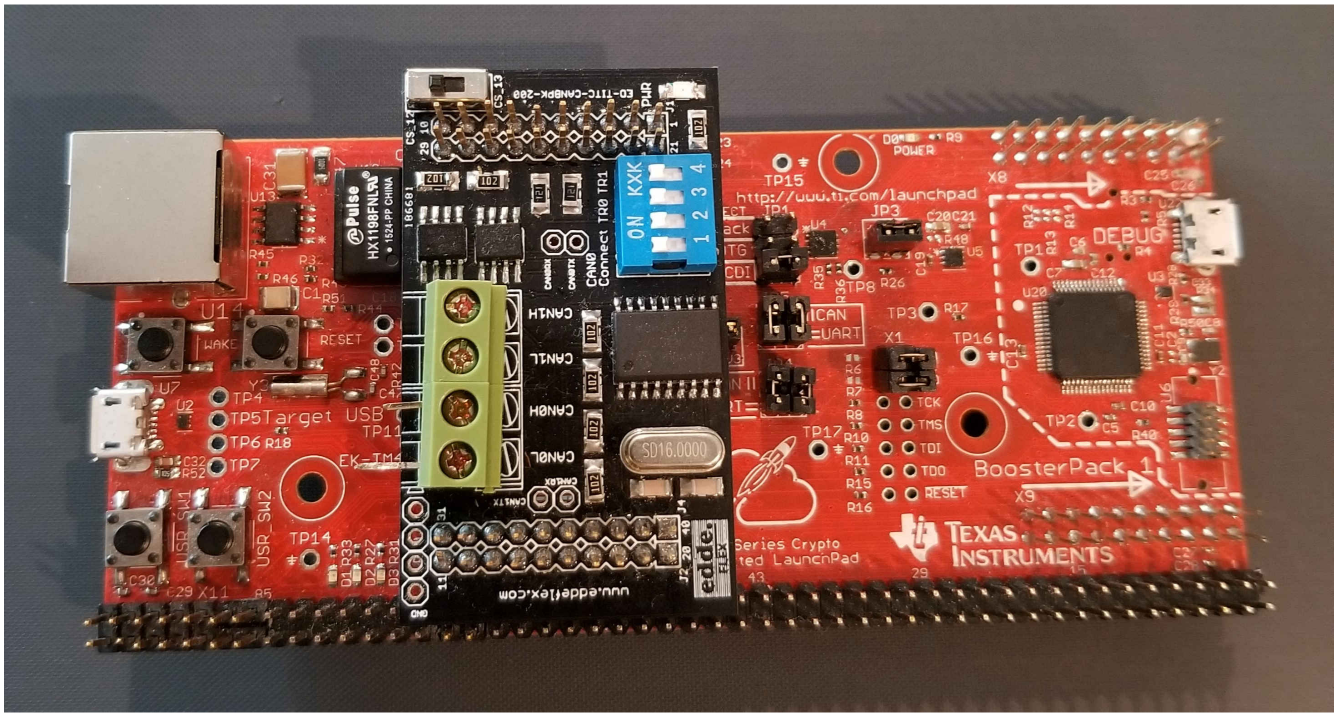


Figure 2-2. EK-TM4C1294XL With Edde Flex CAN BoosterPack

With switches 1 and 2 in the “on” position, PA0 and PA1 are connected to the transmit and receive pins of the booster pack CAN transceiver. Switch 3 in the "on" position connects a 120 Ω termination resistor between CAN0H and CAN0L.

With exception to the internal loopback example, all examples require a working CAN network. A simple CAN network can be constructed using two EK-TM4C1294XL each with an Edde Flex booster pack as shown in Figure 2-3. Note that there are three connections between the two systems: CAN0H, CAN0L and GND.

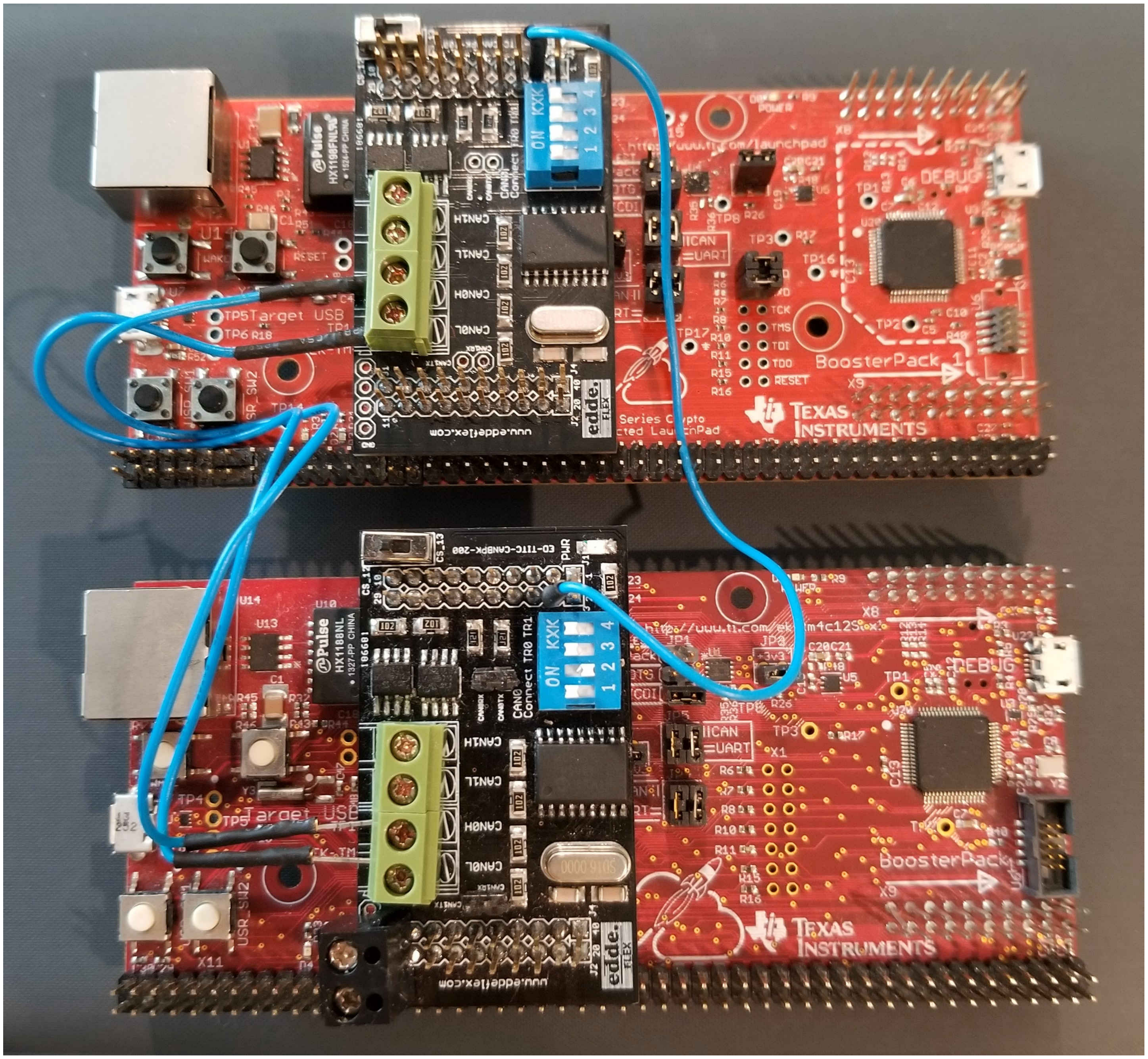


Figure 2-3. Dual EK-TM4C1294XL CAN Network

3 Download and Import the CAN Examples

There are seven Code Composer Studio™ (CCS) project examples attached as collateral to this application report. The examples can be downloaded from the following URL: http://www.ti.com/lit/zip/spna244. You can unzip the project or keep it in zip format. Both formats can be imported into CCS.

1. To import the project into CCS, first select "File" → "Import".

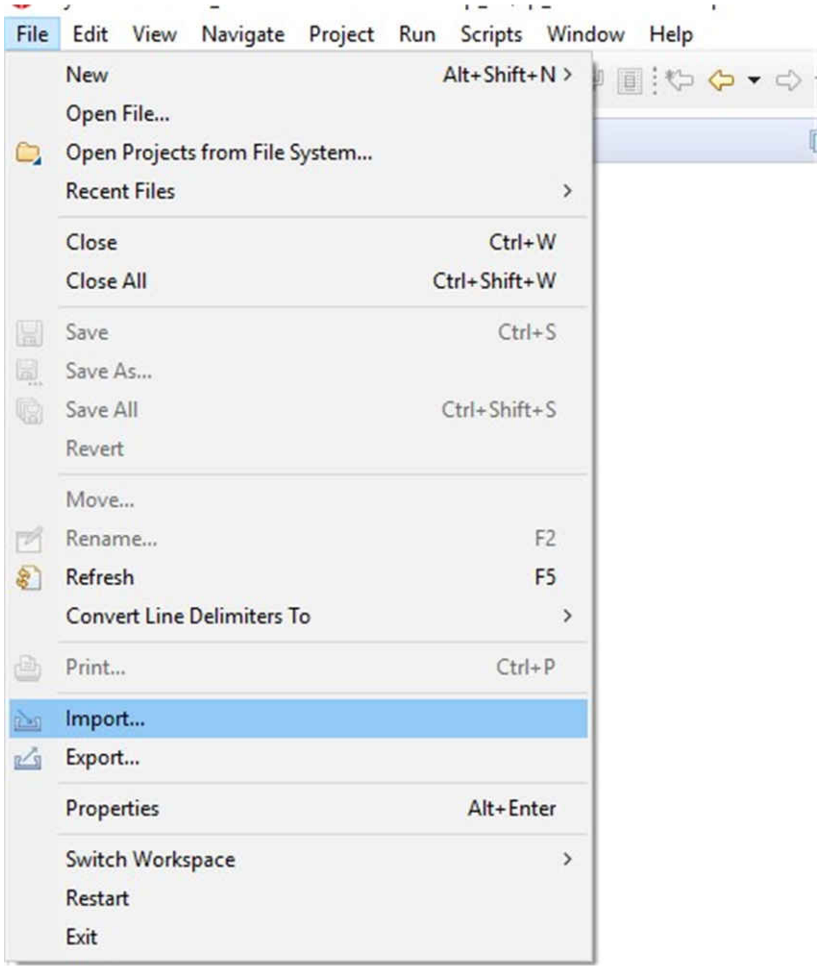
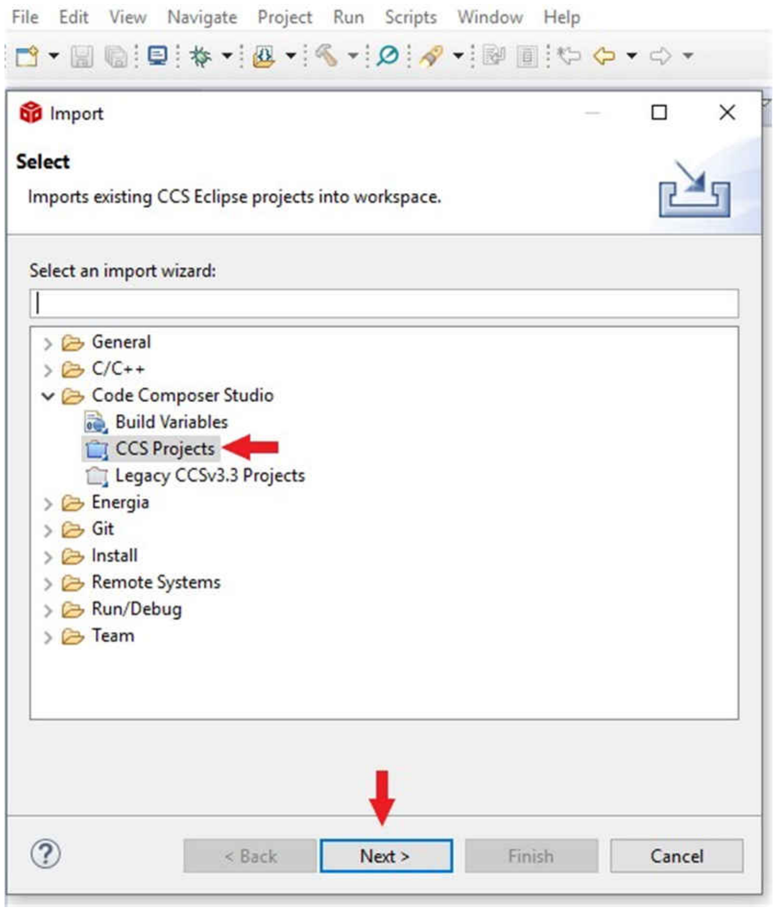


Figure 3-1. Import CCS Projects Step 1

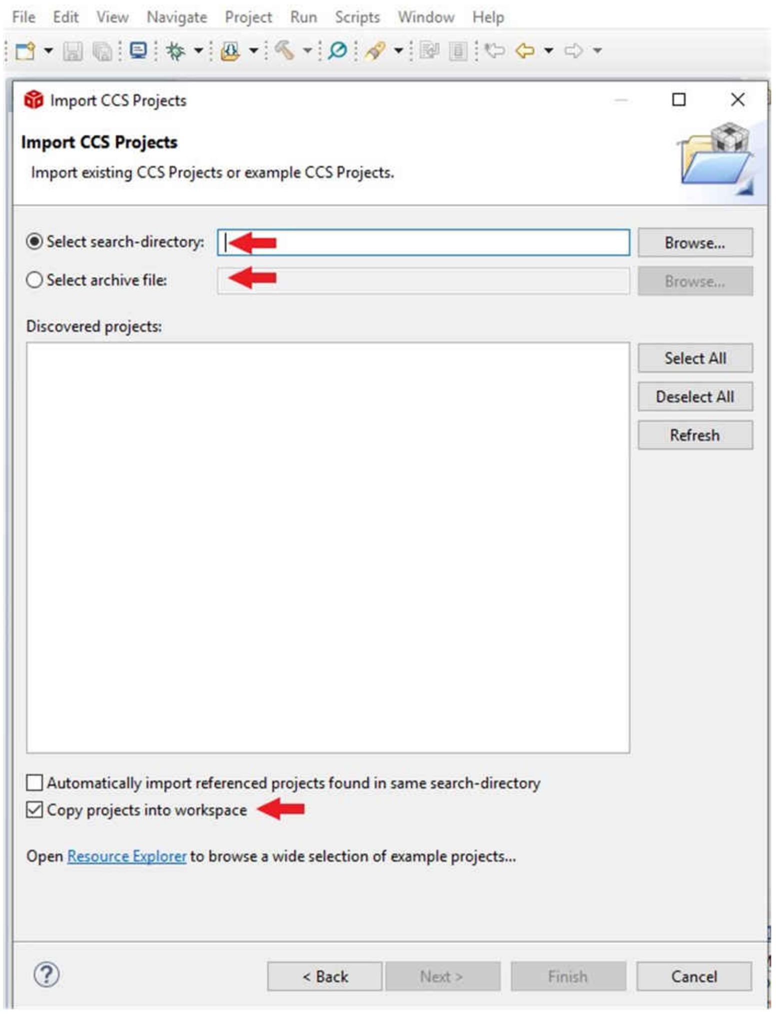
2. Select "CCS Projects" to import the example and then click "Next".



Imports existing CCS Eclipse projects into workspace.

Figure 3-2. Import CCS Projects Step 2

3. Provide the path to either the unzipped project by selecting the first radio button or import from the zip file directly by selecting the second radio button. Click the "Copy projects into workspace".



Import CCS Projects

Import existing CCS Projects or example CCS Projects.

(● Select search-directory:) ○ Select archive file:

Discovered projects:

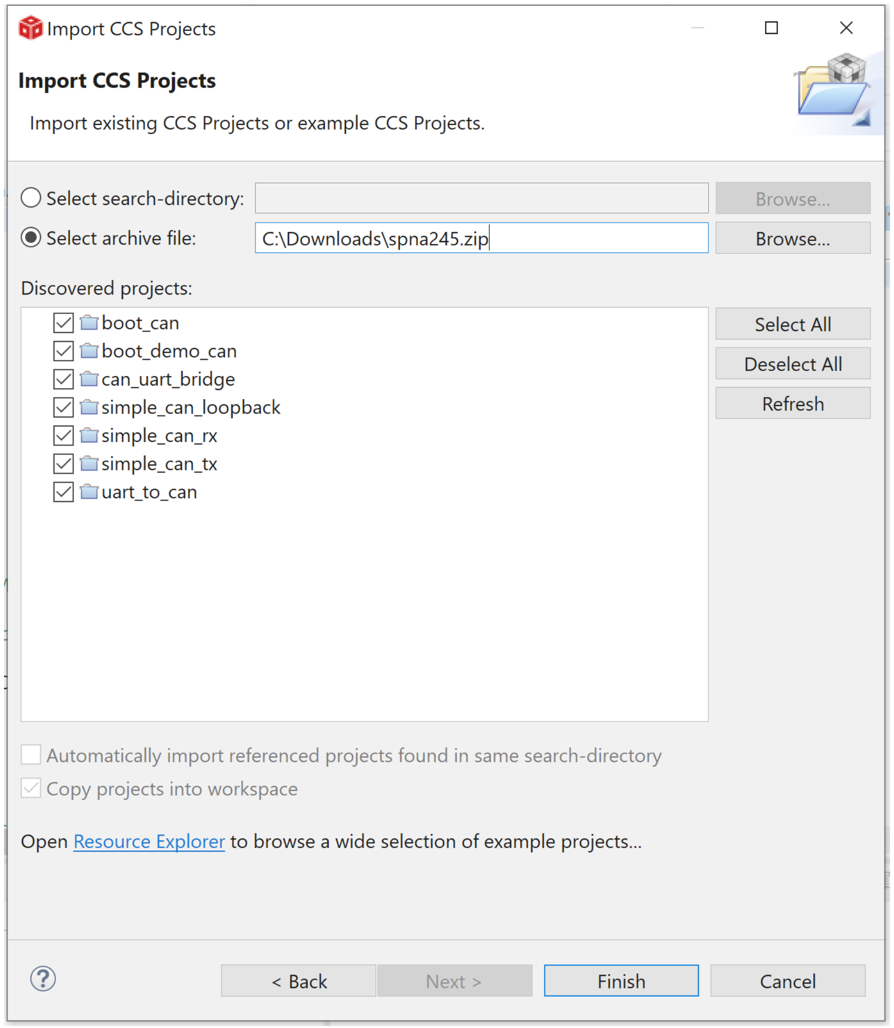
☐ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

Figure 3-3. Import CCS Projects Step 3

4. After the project path is provided, seven discovered projects will show up. First, click the "Select All" button and then click the "Finish" button to complete the import.


Import CCS Projects

Import CCS Projects

Import existing CCS Projects or example CCS Projects.

○ Select search-directory:

● Select archive file: C:\Downloads\spna245.zip

Discovered projects:

- ☑ boot_can
- ☑ boot_demo_can
- ☑ can_uart_bridge
- ☑ simple_can_loopback
- ☑ simple_can_rx
- ☑ simple_can_tx
- ☑ uart_to_can

☐ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

Figure 3-4. Import CCS Projects Step 4

These projects were built using CCS version 9.3 and the Texas Instruments Arm compiler tools version 18.12.6.

4 Modified CAN.C for Interrupts

The version of the driver library file can.c in TivaWare version 2.2.0.295 and earlier, and the ROM based version, use interface one (IF1) registers for most functions and interface two (IF2) registers for receiving messages (CANMessageGet()). This can create an issue when the CAN functions are used in the main application code and in interrupt routines or in multiple threads of an RTOS. The consistency of the interface registers is not maintained during the interrupt or task switch. The updated version of can.c included with this application note uses IF1 registers for all functions when the CPU is not in an exception and uses IF2 registers when the CPU is in an exception state. This simplifies writing code for CAN using interrupts. When using an RTOS, all CAN functions should be in one thread or in one interrupt routine.

The can.c file located in the driverlib folder of the latest TivaWare installation can be replaced with the provided can.c file and the driver library rebuilt, or the provided can.c file can be added directly to your project.

5 Example Projects

These examples use the following peripherals and I/O signals. You must review these and change as needed for your own board:

- CAN0 peripheral
- GPIO Port A peripheral (for CAN0 pins)
- CAN0RX - PA0
- CAN0TX - PA1

The following UART signals are used. You must review these and change as needed for your own board.

- GPIO port D peripheral (for UART2 pins)
- UART2RX - PD4
- UART2TX - PD5

5.1 Internal Loopback With Interrupts (simple_can_loopback)

This example is the only one of these examples that can be run on the Launchpad without a CAN transceiver or a CAN network. When in loopback mode, the CAN module is allowed to receive the message it transmits if an additional mailbox is configured to receive that message. Also in loopback mode the module ignores the absence of an acknowledge bit. The CAN TX pin is active and can be viewed on an oscilloscope or logic analyzer. A CAN frame from this example is shown below with the missing acknowledge bit circled in red.

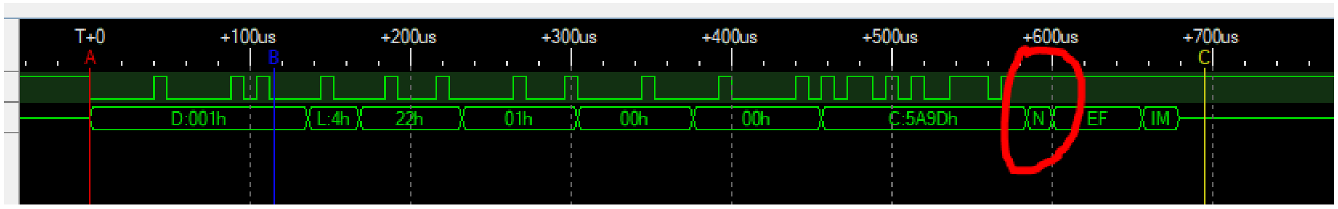


Figure 5-1. Logic Analyzer Waveform

The message that is sent is a 4 byte message that contains an incrementing pattern. The value sent and the message received are echoed out the UART. In this example the CAN0 baud rate is 125000 and the UART2 baud rate is 115200.

This example uses the following interrupt handlers. To use this example in your own application you must add this interrupt handler to your vector table.

- CANIntHandler - CAN 0 interrupt handler

5.2 Simple CAN Transmit (simple_can_tx)

This example shows the basic setup of CAN in order to transmit messages on the CAN bus. The CAN peripheral is configured to transmit messages with a specific CAN ID. A message is then transmitted once per second by using a simple delay loop for timing. The message that is sent is a 4 byte message that contains an incrementing pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent. The message sent and the message count are printed out the UART.

This example uses the following interrupt handler. To use this example in your own application you must add this interrupt handler to your vector table.

- CANIntHandler - CAN 0 interrupt handler

5.3 Simple CAN Receive (simple_can_rx)

This example shows the basic setup of CAN in order to receive messages from the CAN bus. The CAN peripheral is configured to receive messages with any CAN ID and then print the message contents to the console via the UART.

This example uses the following interrupt handler. To use this example in your own application you must add this interrupt handler to your vector table.

- CANIntHandler - CAN 0 interrupt handler

5.4 CAN UART Bridge (can_uart_bridge)

This example interfaces a UART to a CAN bus. It receives data on UART2 then sends it out on CAN2 and receives data on CAN0 then sends it out on UART2. For this example UART2 is configured for 115200 baud (defined in *UartFunctions.h*) and CAN0 is configured for 1M baud (defined in *CanFunctions.h*). At these rates continuous data received on the UART will consume 17% of the CAN bus bandwidth.

Data received by either UART or CAN is handled by interrupt routines. The received data is put into its respective circular buffer. Both buffers are defined with a size of 256 bytes (defined in *main.h*). When continuous data is received on the UART, the UART interrupt routine is triggered after receiving four bytes, a half full FIFO. The data is then collected until eight bytes are received before being put in the circular buffer. This allows the CAN transmission to be the most efficient by supporting up to eight bytes in a CAN packet. When the UART reception halts or pauses, a receive timeout (RT) interrupt is generated. In this case, all the bytes in the UART receive FIFO are transferred to the circular buffer and a CAN frame of less than eight bytes is sent.

The arbitration ID for CAN transmitted messages and CAN received messages is the same, 0x101 (defined in *CanFunctions.h*). This allows two devices running the same software to transmit data from one serial port to another over a CAN network. With more than two devices on the network running the same software, data received on one serial port is transmitted through the CAN network and then output on the serial ports of all the other devices.

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- CAN0IntHandler - CAN 0 interrupt handler
- UART2IntHandler - UART 2 interrupt handler

6 CAN Bootloader

6.1 Bootloader Configuration

The project “boot_can” is a copy of the TivaWare 2.2.0.295 “boot_loader” directory and a modified copy of “bl_config.h”. The modifications to “bl_config.h” set the crystal frequency to 16MHz and enable the CAN bootloader. The CAN baud rate is set to 125K and pins PA0 and PA1 are configured as the CAN pins. The application code must be offset to start at address 0x4000.

6.2 Boot Demo Program (boot_demo_can)

This project contains an example application code that is linked to start at address 0x4000 and is compatible with the CAN bootloader. This application blinks one, two, three or four of the LEDs and looks for a specific CAN message that will cause it to jump back to the CAN bootloader. Instead of a “debug” configuration, there are four configurations named “ONE”, “TWO”, "THREE" and “FOUR”. There is a “boot_demo_can.bin” file in each of the four configuration directories. The version in the “ONE” directory blinks one LED and jumps back to the bootloader when it sees a CAN frame with ID of 0x1F028000 and the first data byte is 0x01. The version in the “TWO” directory blinks two LEDs and jumps back to the bootloader when it sees a CAN frame with ID of 0x1F028000 and the first data byte is 0x02. Likewise with configurations "THREE" and "FOUR". See Section 6.4 for a description on how to use the CAN bootloader and these demo programs.

6.3 UART-to-CAN Bridge for LM Flash Programmer Support

6.3.1 SendCANID PC Program

This is a command line utility that can be run on a PC. It is used to send a serial stream to a device running the uart_to_can software. This causes the device to issue a frame on the CAN bus. This frame can be used by a device on the CAN bus to put that device in bootloader mode.

The format to execute this program from a PC command line prompt or batch file is:

C:\>SendCANID COMnn 0xXXXXXXXX 0xXX

- nn = COM port number (1 – 99)
- 0xXXXXXXXX = The CAN arbitration ID of the command to switch to the bootloader
- 0xXX = 0 to 8 hex bytes that are sent to identify which device is selected

6.3.2 CCS Demo Program (uart_to_can)

This is a Code Composer Studio project for code in a TM4C device that allows it to bridge serial port commands from LM Flash programmer to CAN frames that can be used by the CAN bootloader. LM Flash Programmer is a utility from Texas Instruments that can be used to program the flash on TM4C microcontrollers. It can communicate with the serial bootloader, Ethernet bootloader or USB DFU bootloader on a TM4C device. There is no equivalent tool for talking to the CAN bootloader on a TM4C device.

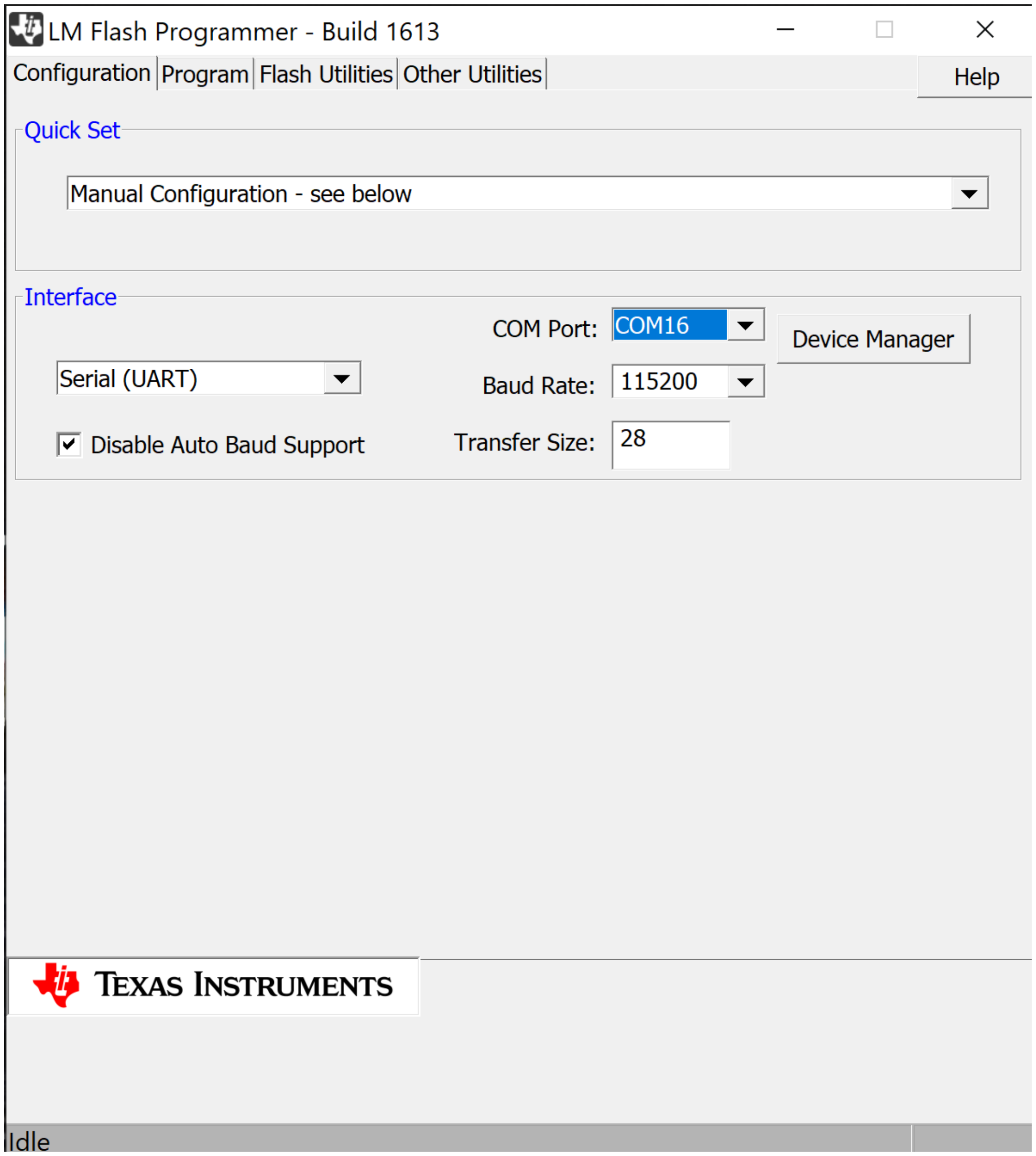
Using the uart_to_can program will offer a method to use LM Flash Programmer to flash program a TM4C microcontrollers that is running a CAN bootloader. This application code can be programmed into a device using JTAG or a serial bootloader then installed on a CAN network. It will then translate the serial bootloader commands that it receives on UART2 into the CAN bootloader protocol using CAN0. See Section 6.4 for an example of using this application code.

6.4 Using the CAN Bootloader

Program the object file “uart_to_can\debug\uart_to_can.out” into one EK-TM4C1294XL board on your CAN network. Identify which COM port is assigned to this board. Program the object file “boot_can\debug\boot_can.out” into one or more other EK-TM4C1294XL Launchpads on your CAN network. Code Composer Studio only recognizes one Launchpad at a time, so it may be necessary to connect to each Launchpad individually. Once all of the Launchpads are programmed, the demo can be run without the use of Code Composer Studio.

The Launchpad with “uart_to_can.out” should be connected to the PC so you have access to its serial port. The other Launchpads can be connected to a PC or a wall USB charger for power only.

Open LM Flash Programmer and configure it for using the UART interface at 115200 baud. It should use the same COM port as the one attached to the Launchpad with the “uart_to_can.out” program.



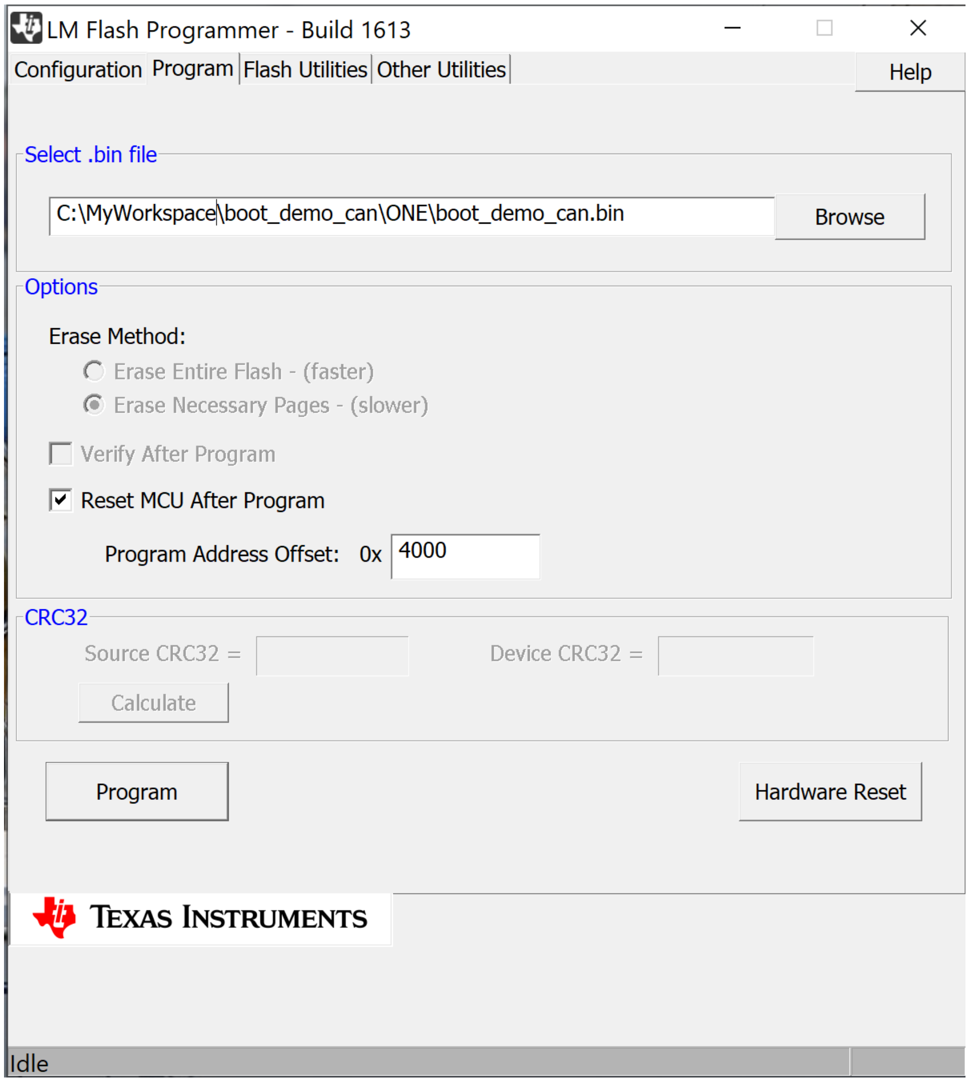
Manual Configuration - see below

COM Port: COM16
Baud Rate: 115200
Transfer Size: 28
Serial (UART)
[✓] Disable Auto Baud Support

Idle

Figure 6-1. Configuring LM Flash Programmer

On LM Flash Programmer select the “Program” tab and browse to the “boot_demo_can\ONE\boot_demo_can.bin” file. Set the “Program Address Offset:” to 0x4000. Press the “Program” button.



Select .bin file

C:\MyWorkspace\boot_demo_can\ONE\boot_demo_can.bin

Options

Erase Method:

( ) Erase Entire Flash - (faster)

(●) Erase Necessary Pages - (slower)

☐ Verify After Program

☑ Reset MCU After Program

Program Address Offset: 0x 4000

CRC32

Source CRC32 =

Device CRC32 =

Idle

Figure 6-2. Programming With LM Flash Programmer

When programming is complete, you will see one blinking LED on the board with the boot loader. This device is now running the application code that was programmed through the CAN bootloader. This application code blinks a single LED, but it also is looking for a specific CAN frame that will cause the CPU to jump back to the CAN bootloader. A CAN frame with arbitration ID of 0x1F028000 and the first data byte equal to 0x01 will cause this unit to return to the CAN bootloader.

Open a command window on the PC and change to the directory with the example projects. In that directory is a PC executable program “SendCANID.exe” and four .bat files. From the command window enter “StopONE COMnn” substituting the number of your COM port for “nn’. You should see a result similar to those below and the LED should stop blinking.

```
C:\MyWorkspace>StopONE COM16
C:\MyWorkspace>SendCANID COM16 0x1F028000 0x01
Program to select CAN ID for download, Version 1.00
Opened serial port COM16 successfully
Sent ID: 0x1F028000 Data: 0x01
Received: 0xCC
C:\MyWorkspace>
```

Now that the LaunchPad has been reverted to executing the CAN bootloader, you can use LM FlashProgrammer to change the application code. Try programming “boot_demo_can\TWO\boot_demo_can.bin”.

Using this example application code and bootloader code, a single unit on a CAN bus can be put in CAN bootloader mode and updated. Each device on the CAN bus can be updated one at a time. The can_to_uart code does not support updating multiple devices at the same time. To do that the can_to_uart code and the CAN bootloader would need to be modified to support multiple status responses to each CAN command.

As an additional feature, if the “uart_to_can” program receives from the “SendCANID” program the ID 0xFFFFFFFF and data 0x00, then the “uart_to_can” program will jump to its ROM serial bootloader allowing the “uart_to_can” program to be updated or overwritten.