Application Report
SPNA182–September 2013

CAN Bus Bootloader for RM42 MCU

Quingjun Wang

ABSTRACT

A bootloader enables field updates of application firmware. A controller area network (CAN) bootloader enables firmware updates over the CAN bus. The CAN bootloader described in this application report is based on the Hercules™ ARM® Cortex™-R4 microcontroller. This application report describes the CAN protocol used in the bootloader and details each supported command.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/spna182.

Contents

List of Figures

List of Tables

1 Introduction

The CAN bootloader permanently resides in the first Flash block of target device. It enables programming of the Hercules microcontroller through its CAN interface. The bootloader also helps designers update the user application code for products already deployed in the field.

This document describes how to work with and customize the Hercules CAN bootloader application. The bootloader is provided as source code which allows any part of the bootloader to be completely customized.

The bootloader on the target device configures the CAN module in communication with PC host through the CAN bus. The bootloader polls the CAN port for messages. After a message is received, the bootloader attempts to decode the incoming commands for flash programming. After the internal flash has successfully downloaded the binary image, the bootloader jumps to the starting address of the new application image.

The target side bootloader has been built and validated using Code Composer Studio™ v5 on the RM42 Hercules HDK. The bootloader host application which communicates with the target side bootloader is developed with Visual C++ 2010. Figure 1 and Table 1 show an overview of the source code provided with the bootloader.

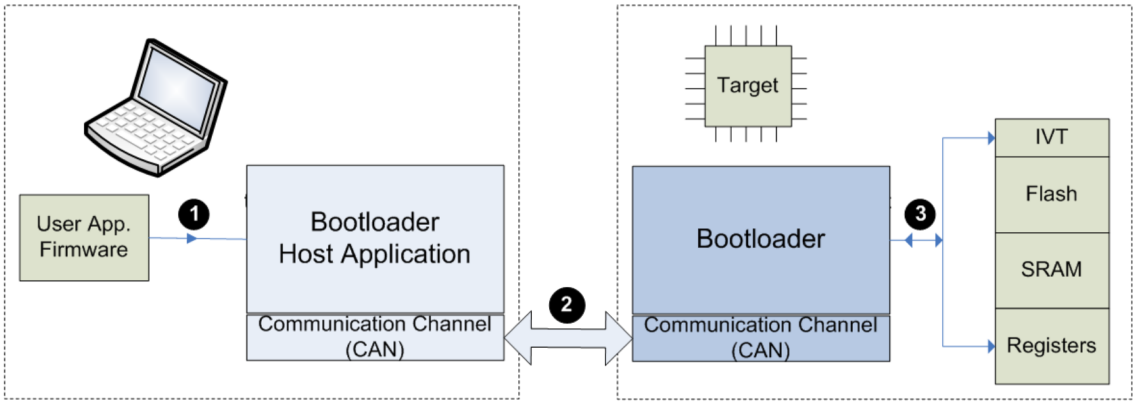

Figure 1. Bootloader Process

Table 1. List of Source Code Files Used in CAN Bootloader

sys_startup.c	The start-up code used when TI's Code Composer Studio compiler is being used to build the bootloader.
sys_intvecs.asm	Interrupt vectors
sys_core.asm	Initialize the core registers, stack pointers , memory, and so forth
system.c	Configure PLL, enable peripherals, and so forth
bl_main.c	The main control loop of the bootloader
bl_can.c	The functions for transferring data via the CAN1 port
bl_check.c	The code to check if a firmware update is required, or if a firmware update is being requested.
hw_pinmux.c	Function that defines the pinmux
sci_common.c	Low-level SCI driver
bl_link.cmd	The linker script used when the Code Composer Studio compiler is being used to build the bootloader.
bl_flash.c	The functions for erasing, programming the Flash, and functions for erase and program check
bl_commands.h	The list of commands and return messages supported by the bootloader.
bl_config.h	Bootloader configuration file. This contains all of the possible configuration values.
bl_flash.h	Prototypes for Flash operations
bl_can.h	Prototypes for the CAN transfer functions.
bw_can.h	Prototypes for the low-level CAN transfer functions.
hw_pinmux.h	Prototypes for pinmux functions

2 Hardware Requirements

The hardware required for configuration includes:

- Power supply: 12 V to HDK
- CAN bus: H, L and GND connecting to CAN1 or CAN2 header on HDK
- Hercules RM42 HDK
- NI USB 8473 high-speed CAN adaptor
- PC with windows XP for running VC++ project
- HyperTerminal for message display via RS232 connected to mini USB connector on HDK
  - Bits/sec: 115200
  - No parity: none
  - Stop bit: 1
  - Flow control: No

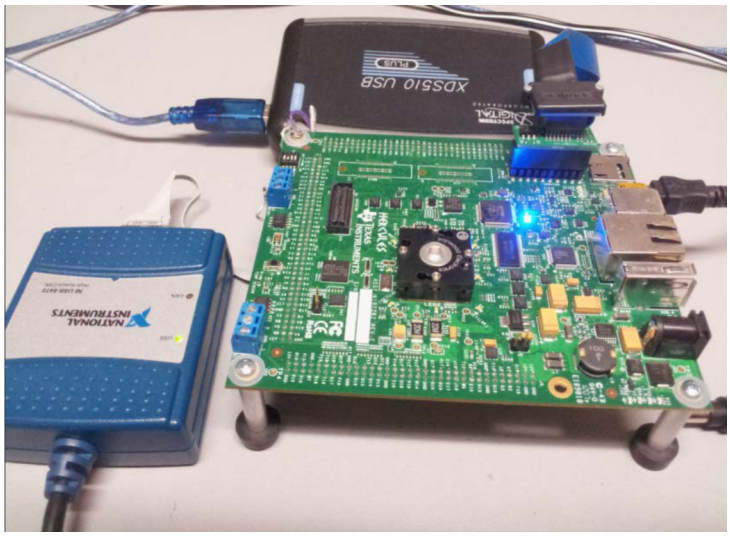

Figure 2. Hardware Setup

3 CAN Settings

The Hercules CAN is compliant with the 2.0A specification with a bitrate up to 1 Mbit/s. It can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers. To change the CAN settings for the bootloader, knowledge of the CAN protocol, revision 2.0 is assumed. For details, see the CAN Protocol Revision 2.0 Specification **(which is located where???)**. Figure 3 shows the essential fields of the standard frame that is used in this CAN bootloader.

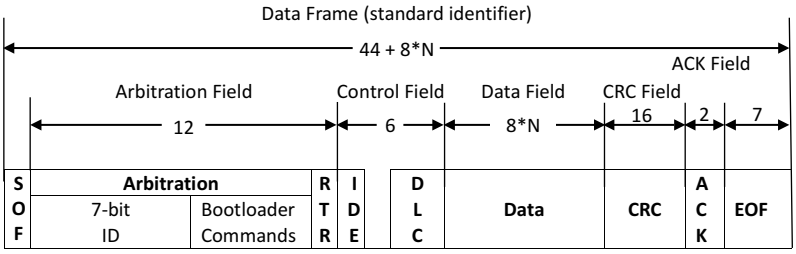

Figure 3. Standard CAN Frame Format

Table 2. Commands Used in Bootloader

Commands	CMD	Description
PING	0x00	See Section 9
DOWNLOAD	0x01	See Section 9
RUN	0x02	See Section 9
GET_STATUS	0x03	See Section 9
SEND_DATA	0x04	See Section 9
RESET	0x05	See Section 9
ACK	0x06	See Section 9

In this application, the CAN settings are:

- Standard identifier (not extended)
- Bitrate: at the default it is 125 kbps
- Functions used: *CANInit()*

The transmit settings (from MCU to the host) are:

- Tx mailbox2: On -- `#define MSG_OBJ_BCAST_TX_ID 1` in *bl_can.c*
- Tx mailbox1: Off -- `#define MSG_OBJ_BCAST_RX_ID 2` in *bl_can.c*
- Tx identifier: 0x5A (device ID) + CMDs (0x00, 0x01, 0x02, v03, 0x04, 0x05, 0x06)
- Functions used: *CANMessageSetTx()*, and *PacketWrite()*

The receive settings (from the host to the MCU) are:

- Synchronization (ACK), 0x06, is in the RX identifier and not in the data field.
- RX identifier depends on the commands (0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06).
- Error checking: Host re-transmits the frames which have lost arbitration or have been disturbed by errors during transmission.
- Incoming messages can contain from 1 to 8 data bytes.
- Functions used: *CANMessageGetRx()*, *CANMessageSetRx()*, and *PacketRead()*

CAN Bit timing setting:

Two clock domains are provided to the CAN module:

- VCLK: general module clock (*system.c*)
- VCLKA1: CAN core clock for generating the CAN Bit Timing (*system.c*)
- Functions used: *CANInit()*

Before configuring the CAN module, evaluate your system specifications such as system propagation delay (wire length and transceiver delay), crystal tolerance, and re-synchronization jump width. To initialize the CAN registers in CAN communication, you must define parameters such as baud rate, propagation segment (Prog_Seg), time segment 1 (Phase_Seg1) and time segment 2 (Phase_Seg2). Using HalCoGen is an easy way to get the correct BTR value. Figure 5 shows what CAN BTR calculations look like in HalCoGen.

$t_{prop} = 2(t_{bus} + tt_{ransmitter} + t_{receiver})$

t_{bus} = Bus Length (meter) * 5 ns/meter

$t_{transmitter}$ and $t_{receiver}$ can be found from the transceiver data sheet **(what is the lit number for this data sheet???)**

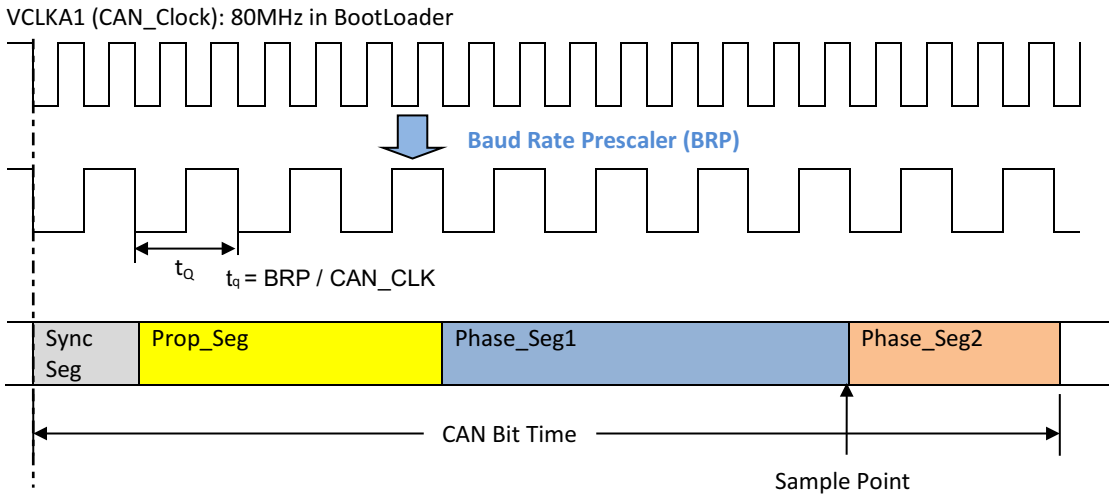

Figure 4. CAN Bit Timing

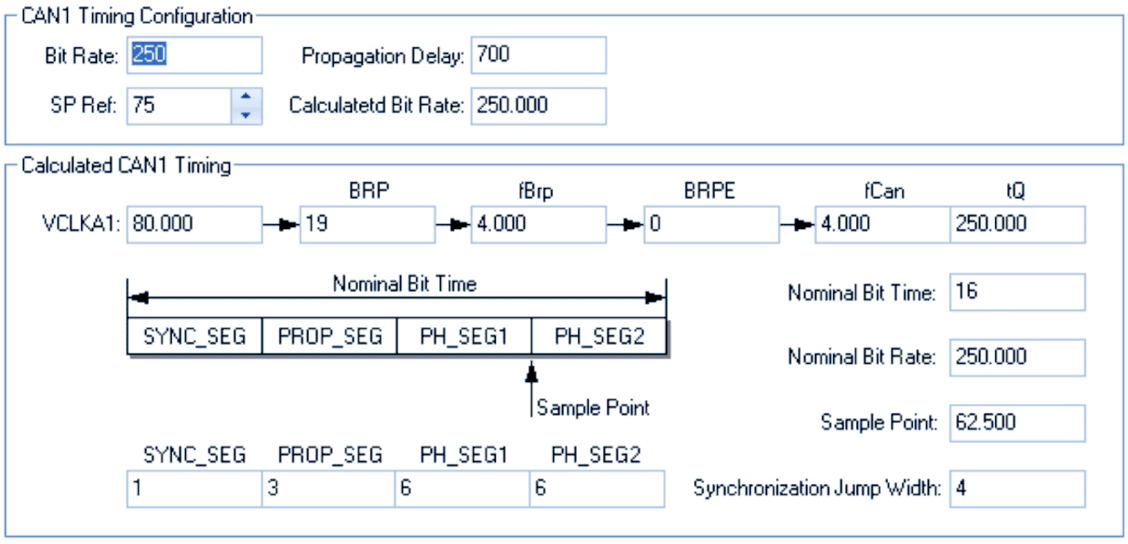

Figure 5. CAN Bit Timing Calculation in HalCoGen

4 Software Coding and Compilation

- The bootloader code is implemented in C, ARM Cortex-R4F, assembly coding is used only when absolutely necessary. The IDE is TI Code Composer Studio 5.4.
- The bootloader is compiled in the 32-bit ARM mode.
- The bootloader is compiled and linked with the TI TMS470 code generation tools V 5.1.

5 On Reset

On reset, the MCU enters in supervisor mode and starts executing the bootloader. The interrupt vectors are setup as shown in Table 3.

Table 3. Vector Table in CAN Bootloader

Offset	Vector	Action
0x00	Reset Vector	Branch to entry point of bootloader (c_int00)
0x04	Undefined Instruction Interrupt	Branch to application vector table
0x08	Software Interrupt	Branch to application vector table
0x0C	Abort (Prefetch) Interrupt	Branch to application vector table
0x10	Abort (Data) Interrupt	Branch to application vector table
0x14	Reserved	Endless loop (branch to itself)
0x18	IRQ Interrupt	Branch to VIM
0x1C	FIQ Interrupt	Branch to VIM

6 During Bootloader Execution

During bootloader execution:

- MCU operates in supervisor mode
- MCU Clock is reconfigured and is maintained throughout the bootloader execution.
  - Clock Source: OSCIN = 16 MHz
  - System clock: HCLK = 80 MHz
  - Peripheral clock: VCLK = 40 MHz
- No interrupts are used
- CAN bit timing: the basic baud rates such as 125000, 250000, 500000, 750000, and 1000000 are supported. The default setting is 1250000. The baud rate is set in *bl_config.h*.
- SCI baudrate: The default setting is: 115200:8:N:1. The basic baud rates such as 9600, 19200, 38400, 57600, and 115200 are supported. The baud rate is set in *bl_config.h*.
- Fix point is used throughout the bootloader execution.
- F021 API V2.00.01 executes in RAM

For device configuration, see the *RM42x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU516) and HalCoGen.

7 Bootloader Flow

Figure 6 shows the execution flow of the CAN Bootloader.

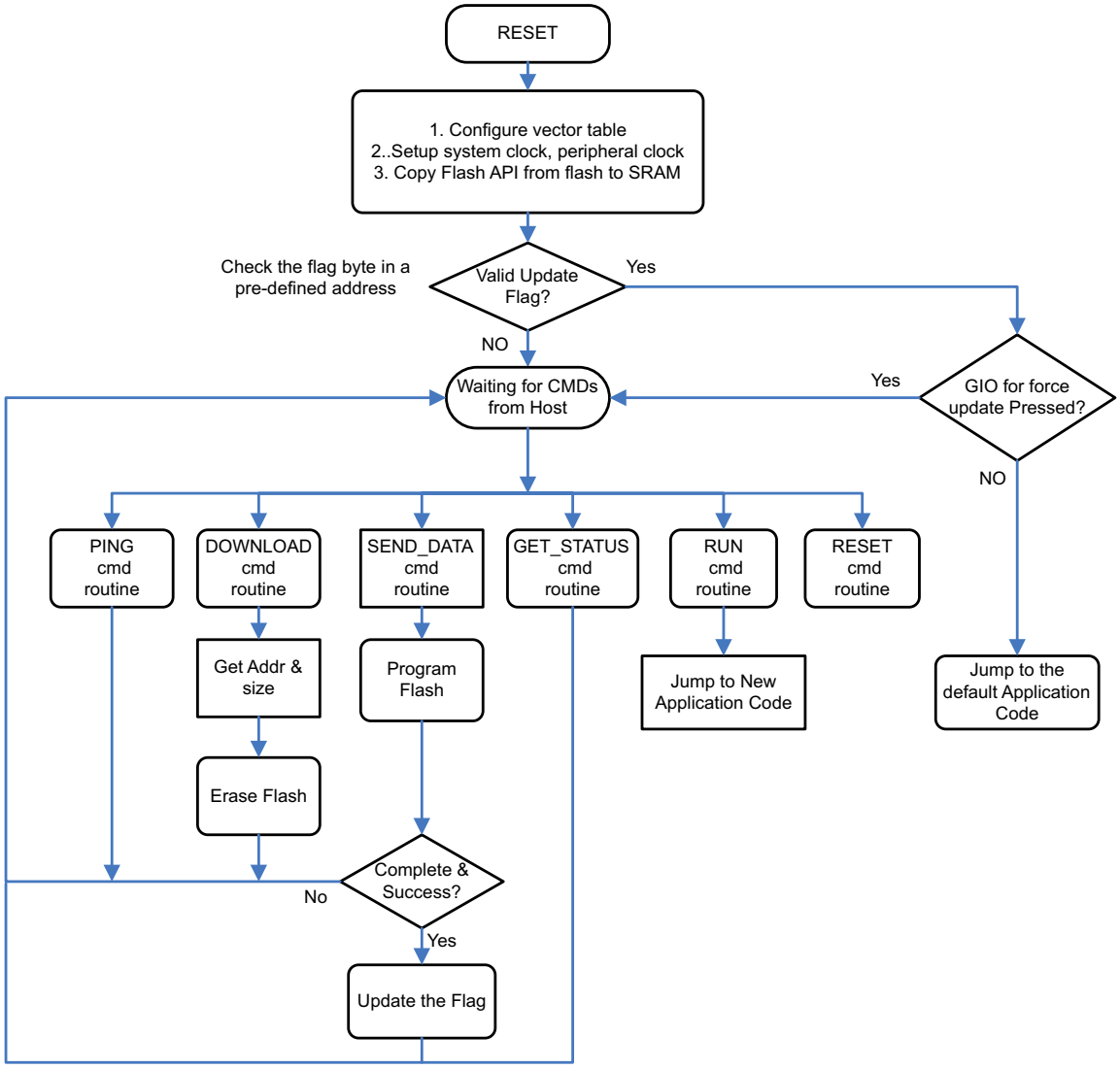

Figure 6. CAN Bootloader Flowchart

8 CAN Bootloader Operation

1. Load the bootloader to Flash.

   The CAN bootloader is built with Code Composer Studio 5.x and loaded through the JTAG port into the lower part of the program memory at 0x0000.

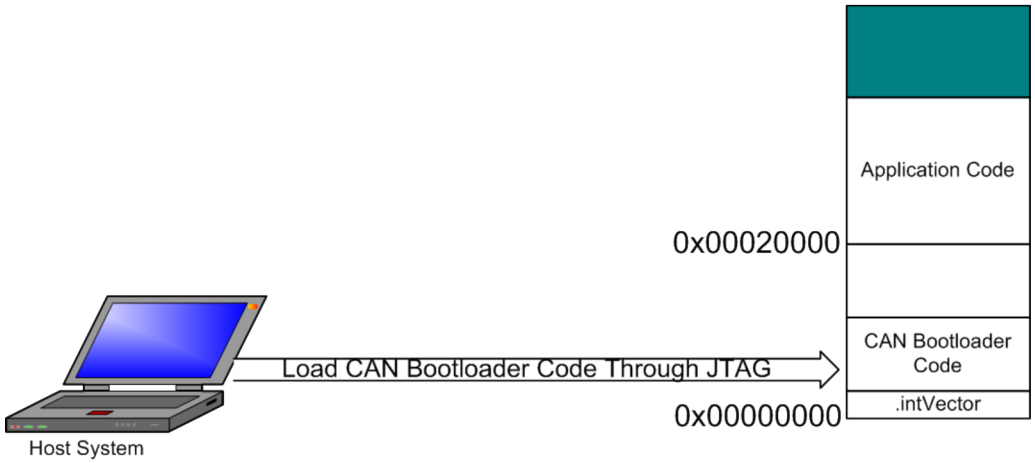

Figure 7. The CAN Bootloader is Loaded Through the JTAG Port

2. Load the user application code.

   After HDK reset, the start-up code copies the Flash API of bootloader from Flash to SRAM, and executes the bootloader in Flash.

   First, it checks to see if the GPIO_A7 pin is pulled low by calling *CheckForceUpdate()*. If GPIO-A7 is pulled LOW, the application code is forced to be updated. The GPIO pin check can be enabled with ENABLE_UPDATE_CHECK in the *bl_config.h* header file, in which case an update can be forced by changing the state of a GPIO pin (with the push button S1 on HDK).

   Then, it checks the magic word or flag at 0x0007FF0. If the flag is a valid number (0x5A5A5A5A), the bootloader jumps to the application code at 0x00020000. If the flag is not the valid number, it configures CAN and SCI, then starts to update the application code by calling *UpdaterCan()*. After all of the application code is programmed successfully, the magic work (flag) is also updated to 0x5A5A5A5A.

   The CAN bootloader uses Message Box 2 to handle incoming messages; Message Box 1 is used for handling the outgoing messages.

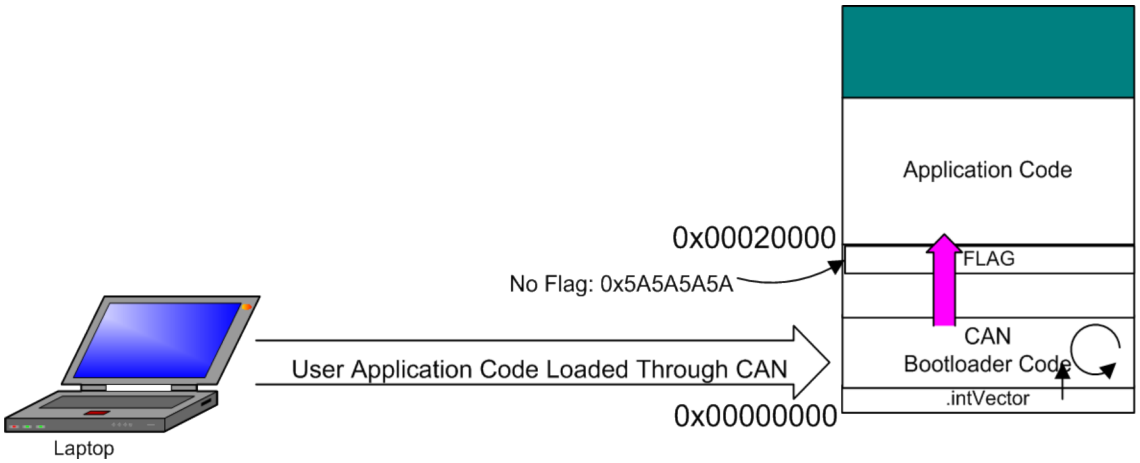

Figure 8. User Application Code is Loaded Through the CAN Bootloader

3. User application is finally loaded and running after sending the reset command to the bootloader.

9 CAN Bootloader Protocol

Messages between a CAN bootloader host and the target use a simple command and acknowledge (ACK) protocol. The host sends a command and within a timeout period the target responds with either an ACK or with a NACK. The command data is combined into message ID. The standard 11 bit message ID is used. Among the 11 bits, the bit 0 to bit 3 is for the bootloader commands, and bit 4 to bit 7 is used for device ID, and the bit 8 to bit 11 is used for manufacturer ID.

The CAN bootloader provides a short list of commands that are used during the firmware update operation. The definitions for these commands are provided in the file *bl_commands.h*. The description of each of these commands is covered in this section.

- CAN_COMMAND_PING (0x00)

  This command is used to receive an acknowledge command from the bootloader indicating that communication has been established. This command has no data. If the device is present, it will respond with a CAN_COMMAND_PING back to the CAN update application.

- CAN_COMMAND_DOWNLOAD (0x01)

  This command sets the base address for the download as well as the size of the data to write to the device. This command should be followed by a series of CAN_COMMAND_SEND_DATA that send the actual image to be programmed to the device. The command consists of two 32-bit values. The first 32-bit value is the address to start programming data into, while the second is the 32-bit size of the data that will be sent.

  This command also triggers an erasure of the full application area in the Flash. This Flash erase operation causes the command to take longer to send the CAN_COMMAND_ACK in response to the command, which should be taken into account by the CAN update application.

  The format of the command is as follows:

```
unsigned char ucData[8];
ucData[0] = Download Address [7:0];
ucData[1] = Download Address [15:8];
ucData[2] = Download Address [23:16];
ucData[3] = Download Address [31:24];
ucData[4] = Download Size [7:0];
ucData[5] = Download Size [15:8];
ucData[6] = Download Size [23:16];
ucData[7] = Download Size [31:24];
```

- CAN_COMMAND_SEND_DATA (0x02)

  This command should only follow a CAN_COMMAND_DOWNLOAD command or another CAN_COMMAND_SEND_DATA command when more data is needed.

  Consecutive send data commands automatically increment the address and continue programming from the previous location. The transfer size is limited to 8 bytes at a time based on the maximum size of an individual CAN transmission. The command terminates programming once the number of bytes indicated by the CAN_COMMAND_DOWNLOAD command have been received.

  The CAN bootloader sends a CAN_COMMAND_ACK in response to each send data command to allow the CAN update application to throttle the data going to the device and not overrun the bootloader with data.

  This command also triggers the programming of the application area into the Flash. This Flash programming operation causes the command to take longer to send the CAN_COMMAND_ACK in response to the command, which should be taken into account by the CAN update application.

  The LED D7 is flashing until the application update is complete.

The format of the command is as follows:

```
unsigned char ucData[8];
ucData[0] = Data[0];
ucData[1] = Data[1];
ucData[2] = Data[2];
ucData[3] = Data[3];
ucData[4] = Data[4];
ucData[5] = Data[5];
ucData[6] = Data[6];
ucData[8] = Data[7];
```

- CAN_COMMAND_RESET (0x03)

  This command is used to tell the CAN bootloader to reset the microcontroller. This is used after downloading a new image to the microcontroller to cause the new application or the new bootloader to start from a reset. The normal boot sequence occurs and the image runs as if from a hardware reset. It can also be used to reset the bootloader if a critical error occurs and the CAN update application needs to restart communication with the bootloader.

- CAN_COMMAND_REQUEST (0x05)

  This command returns the status of the last command that was issued. This command has no data.

10 Create Application for Use With the Bootloader

In order to allow future upgrades using the bootloader, application images must be created with a starting address of 0x20000 (default). The reason for this is that the bootloader itself occupies the Flash area below this address. To achieve this, the default Flash start address defined in the linker command file must be changed as shown in Figure 9.

```
/*----------------------------------------------------------*/
/* Linker Settings                                          */
--retain="*(.intvecs)"
-heap 0x800

/*----------------------------------------------------------*/
/* Memory Map                                               */
MEMORY{
    VECTORS (X)   : origin=0x00020000 length=0x00000020
    FLASH0  (RX)  : origin=0x00020020 length=0x0017FFE0
    STACKS  (RW)  : origin=0x08000000 length=0x00001300
    RAM     (RW)  : origin=0x08001300 length=0x0003ED00
}

/*----------------------------------------------------------*/
/* Section Configuration                                    */
SECTIONS{
    .intvecs : {} > VECTORS
    .text    : {} > FLASH0
    .const   : {} > FLASH0
    .cinit   : {} > FLASH0
    .pinit   : {} > FLASH0
    .bss     : {} > RAM
    .data    : {} > RAM
    .sysmem  : {} > RAM
}
/*----------------------------------------------------------*/
```

Figure 9. The Linker File for Application

To create an application using TI Code Composer Studio 5.x, use the linker files included with this application report for your project. The included linker files set up the starting address of Vector Table and Memory Regions to 0x20000 for the binary. In the project properties window, type the following command in “Post-Built Steps Command”:

```
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin.bat"
"${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin"
"${CG_TOOL_ROOT}/bin/ofd470.exe"
"${CG_TOOL_ROOT}/bin/hex470.exe"
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin.exe"
```

The resulting binary will be placed in your project folder, and binary file name is *projectName.bin* as default.

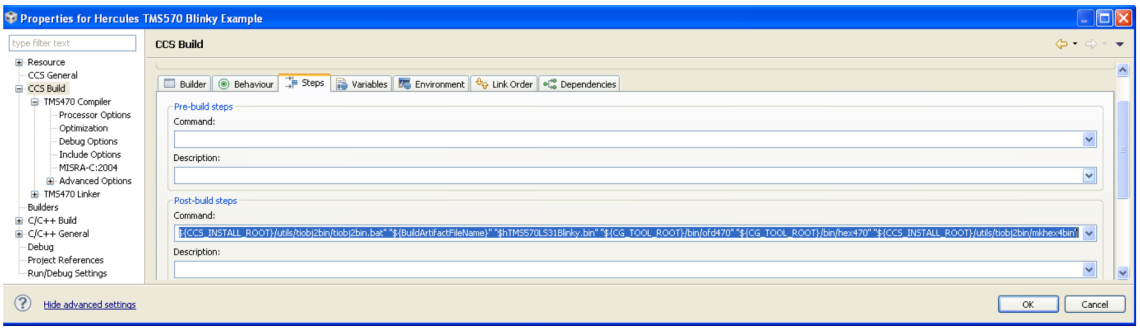

Figure 10. Setup Project Property to Generate Binary File for Bootloader

11 Sample Code for PC-Side Application

The PC-side application is developed using VC++ 2010. The *bl_command.h* defines the commands used for talking with the CAN bootloader on the MCU side. The library and header file for NI-CAN 8473 are included in the project.

The *can_bltest.c* does all the tests for bootlader:

- Opens binary image (user application)
- Sends command to ping MCU bootloader
- Sends starting address and image size to the MCU bootloader
- Sends data of the image to the MCU bootloader
- Sends execution command to run the user application
- Sends Reset command to reset the MCU

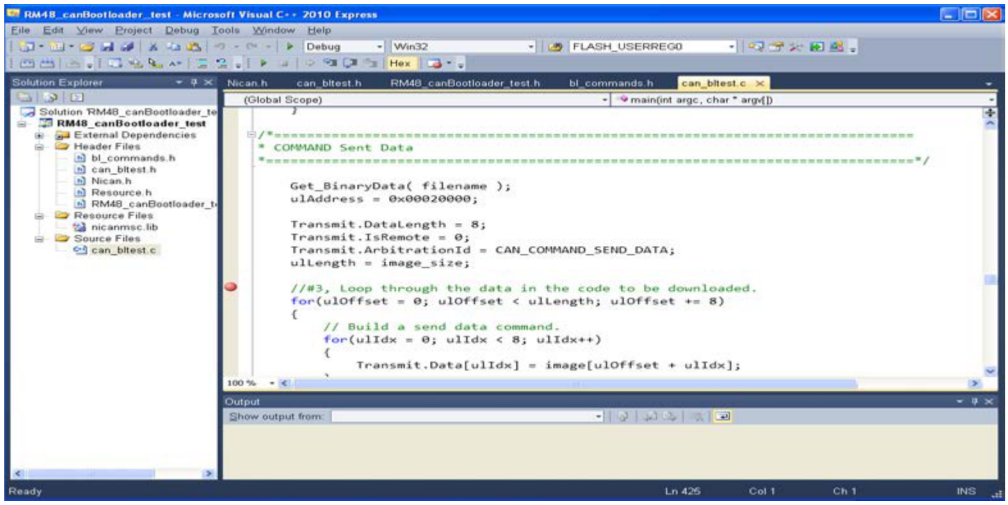

Figure 11. VC++ Project for PC-Side Bootloader

12 References

- *RM42L432 16/32-Bit RISC Flash Microcontroller Data Manual* (SPNS180)
- *RM42x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU516)
- *F021 Flash API Version 2.00.01 Reference Guide* (SPNU501)
- Specification of NI USB-CAN 8473 Adaptor: http://sine.ni.com/nips/cds/view/p/lang/en/nid/203384

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265