Freescale Semiconductor
Application Note

Document Number: AN4511
Rev. 0, 4/2012

CAN Bootloader in the MCF51JM

by: **Xu Weiping and Alejandro Lozano**
Automotive and Industrial Solutions Group

1 Introduction

A traditional method of providing in-circuit programming of an Electronic Control Module (ECM) is based on the Universal Asynchronous Receiver / Transmitter (UART), or an MCU-specific interface. Using this approach requires dedicated hardware on each ECM.

The benefits of utilizing Controller Area Network (CAN) as an in-circuit programming method can be easily understood. For systems already using CAN connections, no additional hardware is required, and connection to any node allows communication to all other nodes via CAN. This concept offers benefits throughout the product life span, from the development phase through to in-field upgrades, servicing, and diagnostic capabilities. During development and testing, any module connected to the network can be reprogrammed in-circuit, saving time and effort as well as minimizing the dependencies between product assembly and software development. In-field system upgrades, servicing, and diagnostic reports can be easily achieved using CAN.

For a system that has nodes with identical firmware, it can be more efficient to upgrade these nodes in a single pass like in gang-programmer fashion. But when the system has unique nodes with different firmware, just a single connection to program the device is considered. In an active network, it is recommended to shut down all other nodes in order to avoid priority issues with these nodes or low-programming performance.

Contents

This application note explains a simple bootloader implementation using CAN. It also provides a low-level driver (MSCAN driver) that can be used in CAN applications.

2 Functional description

This section describes the architecture of the CAN bootloader and the software flow of each implemented device. Figure 1 shows the architecture of the entire system.

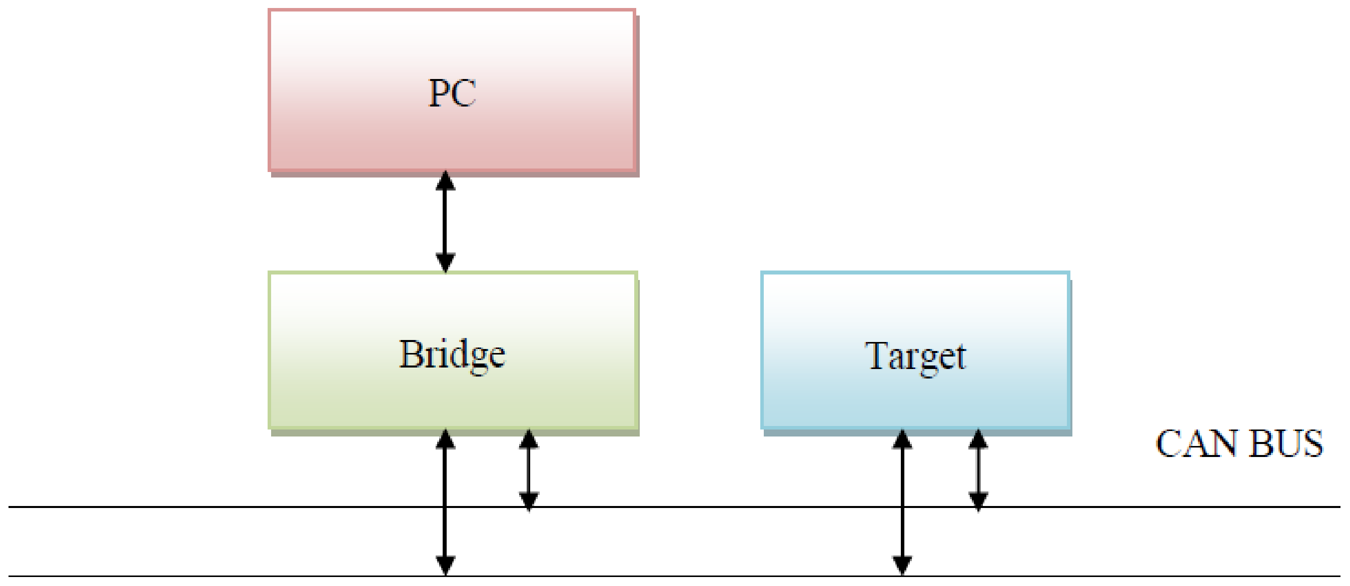
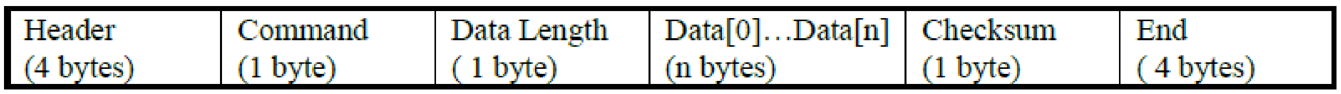


Figure 1. System architecture

This application note includes a graphic user interface (GUI) supported in Windows XP that can used to choose the S19 file. This file contains the object code that will be downloaded in the internal flash memory. The GUI parses the S19 and sends commands, data and address to the USB-to-CAN bridge. The bridge checks whether all the data sent by the GUI is valid and sends the data to be loaded into the target.

The PC communicates with the bridge via virtual COM port (USB CDC class). The baud rate is 115,200 bit/s, 1 stop bit and no parity bit using big-endian for data containing multiple bytes. Every time the PC sends one packet to the bridge, it sends an ACK back indicating the receive status. The data that the PC sends looks like the packet below:



Header (4 bytes)	Command (1 byte)	Data Length (1 byte)	Data[0]…Data[n] (n bytes)	Checksum (1 byte)	End (4 bytes)

Figure 2. Packet format

The following two tables depict the description of several data packet fields and the PC commands.

Table 1. Packet format description

Packet Field	Description
Header	0x55,0x55,0x55,0x55
Command	The command code defines the action to be executed by the bridge.
Data length	The number of bytes in the Data field.

Table continues on the next page...

Table 1. Packet format description (continued)

Packet Field	Description
Data	This can be either the command parameters or the formatted data sent in response to a command. The size of the data varies depending on the command.
Checksum	Checksum of data in the Data Length Field and Data Field
End	0xAA,0xAA,0xAA,0xAA

Table 2. PC command summary

Command	Description	Data
0x01	Mass Erase Target	None
0x02	Sector Erase Target	Data[0–3]: Starting address of the sector
0x03	Blank Check Target	None
0x04	Program Target	Data[0–3]: Starting address; Data[4]–Data[n < 36]: code
0x05	Verify Target	None
0x06	Read Target	Data[0–3]: Starting address; Data[4–7]: Byte length (Byte length <= 64)
0x07	Unsecure Target	Data[0–7]: Backdoor key

2.1 Functional description of target

The general architecture of the target is shown in the figure below.

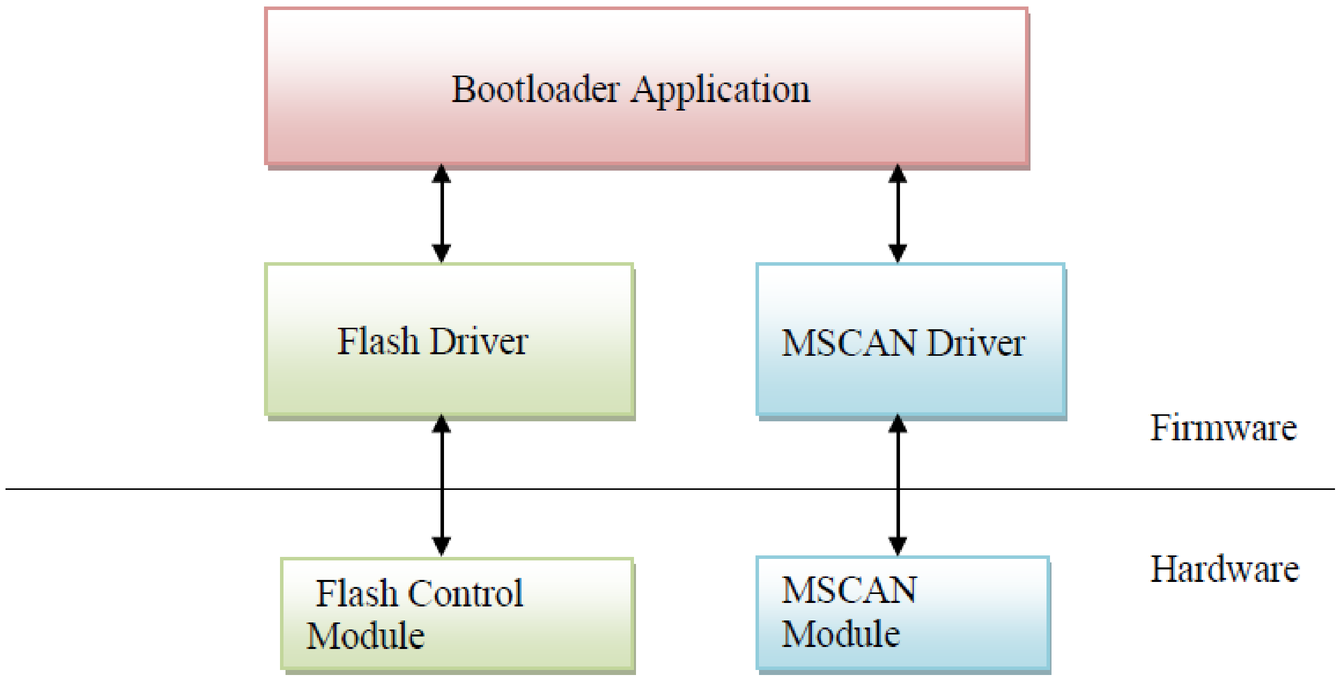


Figure 3. Target architecture

1. After reset, the bootloader executes and checks whether the application should start or the bootloader should keep on running. In Bootloader mode, the target executes and sends response back to the bridge.
2. Depending on the command sent by the bridge, the target executes the needed procedure like unsecure target, download the object code, or mass erase. If the bridge does not send any command, the target just executes an idle function. The target communicates with the bridge at 250 kbit/s.

There are 9 supported commands. The following table lists the description of bootloader commands and the specific actions for each command.

Table 3. Target command summary

Command	Host sent message	Target response message	Action
UNSEC	CAN ID 0x011 + 64-bit backdoor key	CAN ID 0x001 + 64-bit backdoor key	The target should unsecure by using the security code, if the flash is secured and backdoor key is enabled. When unsecured successfully, sends the response message.
SET_ADDR	CAN ID 0x012 + 32-bit start address + 32-bit number of bytes	CAN ID 0x002 + 32-bit start address + 32-bit number of bytes	Starting address is stored for programming flash. Data size is stored for programming flash.
LOAD_DATA	CAN ID 0x013 + 8–64 bits of raw binary data	CAN ID 0x003 + 8–64 bits of raw binary data	The data containing 1–8 bytes will be programmed to flash from the start address received from the SET_ADDR command.

Table continues on the next page...

Table 3. Target command summary (continued)

Command	Host sent message	Target response message	Action
MASS_ERASE	CAN ID 0x014 without data	CAN ID 0x004 without data	The target mass erases its flash. When finished erasing, sends the response message.
SECTOR_ERASE	CAN ID 0x015 + 32-bit address	CAN ID 0x005 + 32-bit address	The target erases the flash sector including the address. When finished erasing, sends the response message.
BURST_PROG	CAN ID 0x016 without data	CAN ID 0x006 without data	The target programs the flash using the address and data from the SET_ADDR and LOAD_DATA commands. When finished programming, sends the response message.
BYTE_PROG	CAN ID 0x017 + 32-bit address + 8-bit data	CAN ID 0x007 + 32-bit address + 8 bit data	The target programs one byte of the flash at the address and the data. When finished programming, send the response message.
BLANK_CHECK	CAN ID 0x018 without data.	CAN ID 0x008 with blank check result	The target verifies all flash memory bytes mass erased and returns the result: 0xF0-Flash is not blank, 0xFF-Flash is blank.
READ_TARGET	CAN ID 0x019+ 32-bit address + 32-bit byte length	CAN ID 0x009 + 8-byte data	The target should return its flash data from the specific address.

The flowchart shown in Figure 4 shows the flow of the target program, with just a few commands.

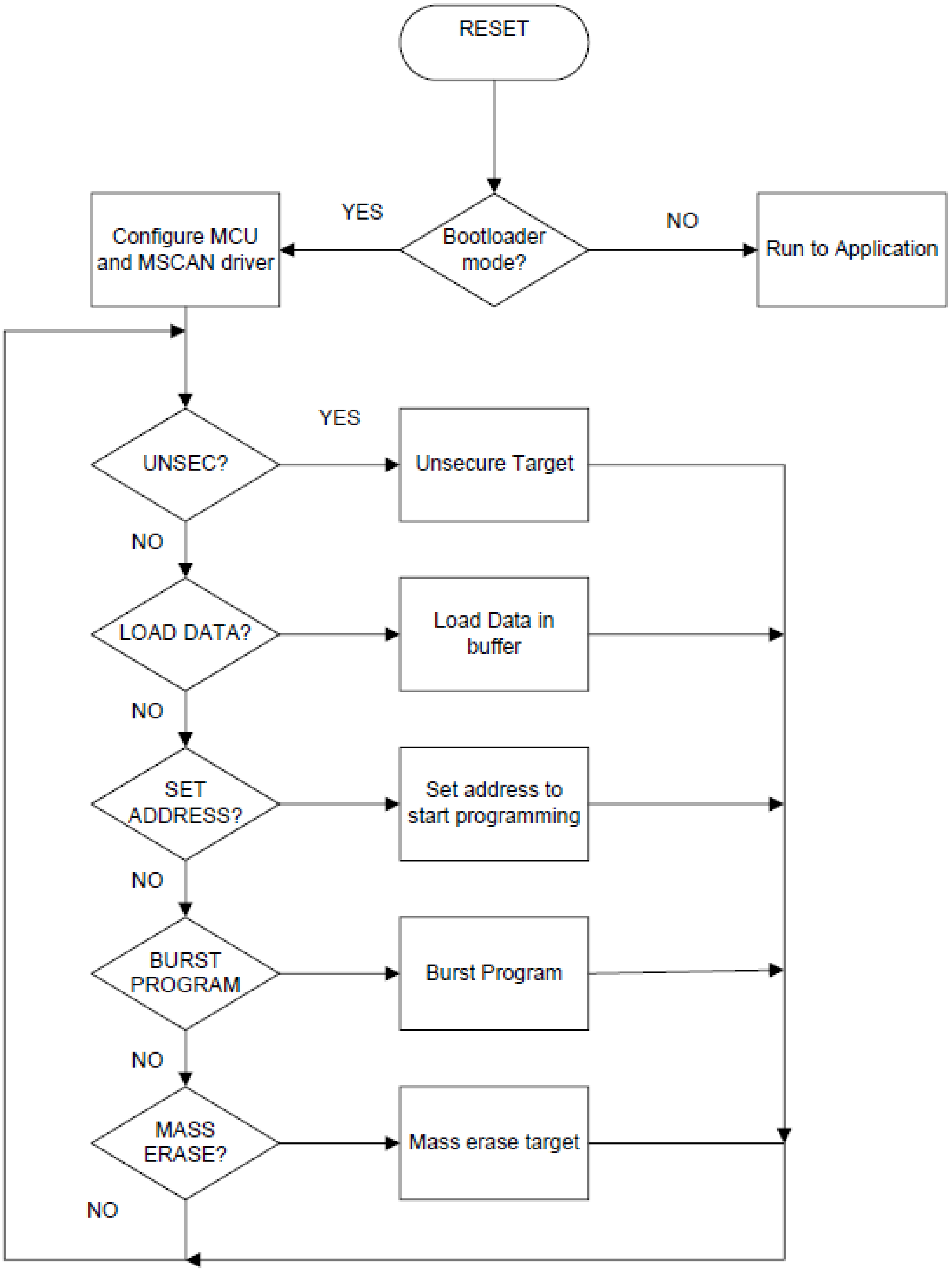


Figure 4. Target flow chart

2.2 Functional description of USB-to-CAN bridge

This section describes the USB-to-CAN bridge architecture and firmware design.

The USB-to-CAN bridge is implemented by the MCF51JM MCU, which includes a USB module and an MSCAN module. A DEMOJM board or EVB51JM board can be implemented as the bridge.

Figure 5 shows the architecture of the USB-to-CAN bridge. The bridge communicates with the PC GUI via virtual COM port based on the USB interface. It receives commands from the USB host by calling USB CDC driver, and then, interprets and manages the commands. It also transmits the CAN messages received from the CAN network by calling the MSCAN driver, and then bridges the message to the host via USB by calling the USB CDC driver.

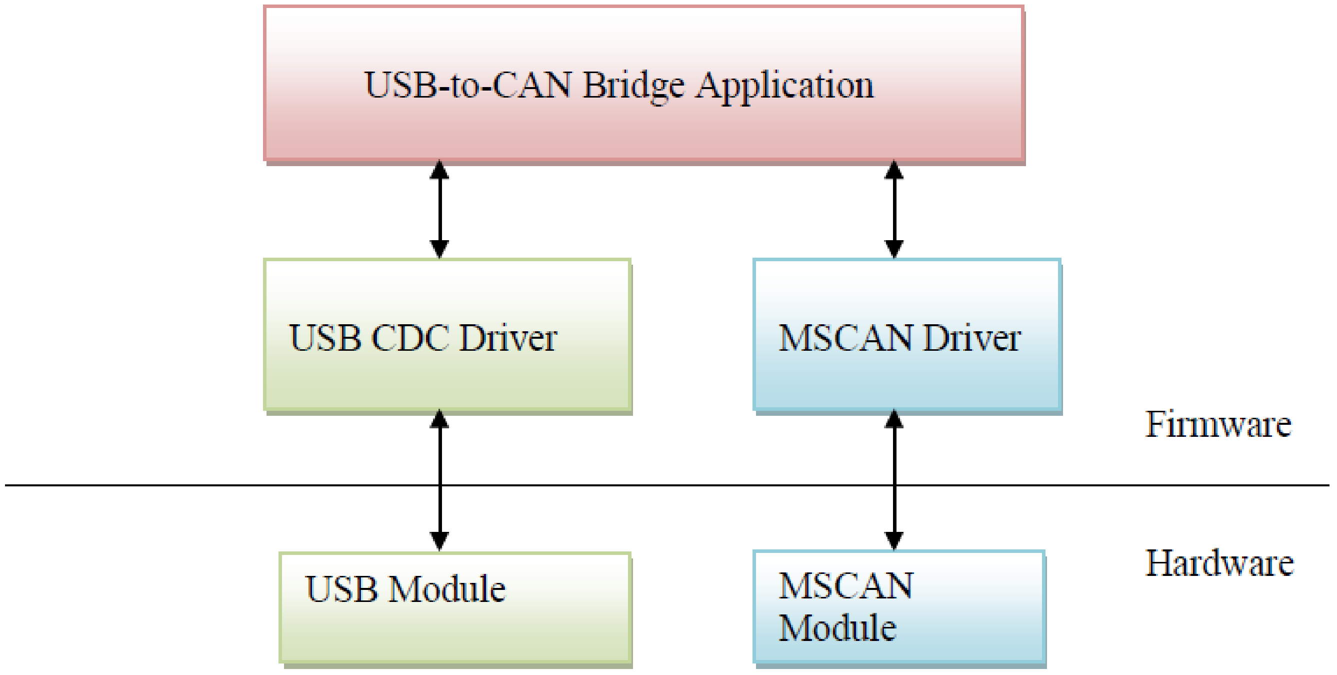


Figure 5. USB-to-CAN architecture

2.2.1 USB CDC driver

Freescale USB stack is used in the USB-to-CAN Bridge. The Freescale USB stack contains the low-level driver code, commonly used class drivers, and some basic applications. The USB-to-CAN bridge uses CDC class for data communication. When the bridge is plugged into PC, a driver needs to be installed. After successfully installing the driver, one virtual COM port must be detected in hardware manager of the Windows operating system.

2.2.2 MSCAN driver

Freescale MSCAN driver for ColdFire V1 is used in the USB-to-CAN Bridge. The driver is optimized for a single on-chip MSCAN. The driver emulates the 'full-CAN' model, with a message buffer per CAN identifier. The number of message buffers is configurable, allowing full control over RAM requirements.

The main driver features are:

- Optimized for a single MSCAN module
- Up to 255 message objects (Standard and Extended Identifiers)
- Up to 32 message buffers
- Advanced priority-based management of queued message objects

- Auto-reply to received Remote Transmission Request Frames
- Auto-receive of data frames in response to Remote Transmission Request Frames
- Sleep and Wakeup functions
- Abort transmission function

The MSCAN driver routines perform the following functions:

- Initialization of the MSCAN module
- Transmission of CAN messages, with a local priority-based queue
- Store received CAN messages in appropriate message buffers
- Functions to configure, load, read, and obtain the status of message buffers
- MSCAN control functions (CAN_Reset, CAN_Sleep, CAN_Wakeup, CAN_CheckStatus)

To be applied in various cases, the predefined bit rate is 250 KB/s, so that such MCUs with long distance bus are possible to be programmed as well. Higher speed can be achieved by modifying the MSCAN configuration file, msCANcfg.h. The related macros are listed below.

```
/******************************************************************************
* Define S08 msCAN module clock source: BUSCLK or MCGERCLK
* Permitted values: BUSCLK: Bus clock is used as clock source
* MCGERCLK: Oscillator clock is used as clock source
******************************************************************************/
#define CLKSRC_CAN MCGERCLK
/******************************************************************************
* Define clock prescaler for msCAN module: permitted values 1 to 64
* msCAN module clock = CLKSRC_CAN / PRESCALER_CAN
******************************************************************************/
#define PRESCALER_CAN 6
/******************************************************************************
* Define msCAN module bit timing
*
* Permitted values: PROP_SEG_CAN: 1 to 8 time quanta
* PHASE_SEG1_CAN: 1 to 8 time quanta
* PHASE_SEG2_CAN: 2, or PHASE_SEG1_CAN if greater
* Bit time = (1 + PROP_SEG_CAN + PHASE_SEG1_CAN + PHASE_SEG2_CAN) * time quanta
******************************************************************************/
#define PROP_SEG_CAN 1
#define PHASE_SEG1_CAN 4
#define PHASE_SEG2_CAN 2
```

NOTE

The configuration for the MSCAN module is the same for both target and bridge.

2.2.3 Bridge application

As shown in Figure 6, the application implements the communication management between USB CDC driver and MSCAN driver, which includes two main tasks:

- Virtual COM task, which manages the virtual COM serial communication.
- USB2CAN task, which includes a USB command parser

The parser parses the USB command and then calls MSCAN driver to send the corresponding CAN frame to the target. Also, it receives the response of the target and calls back to the virtual COM task.

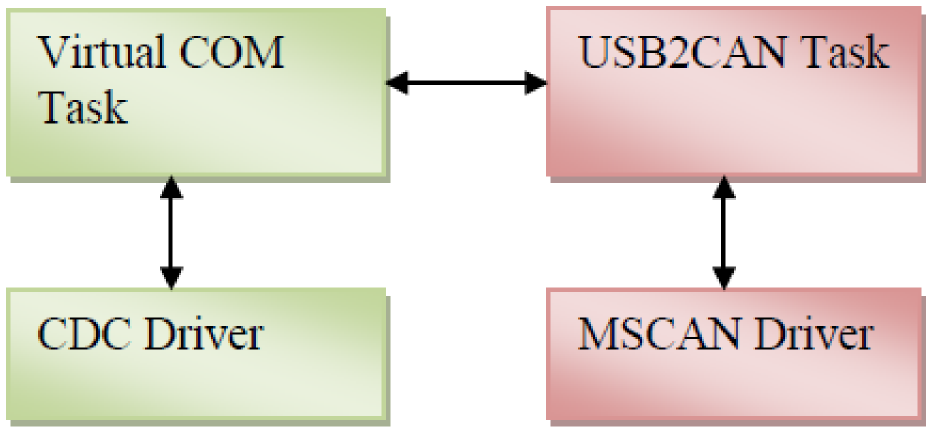


Figure 6. USB-to-CAN application

2.3 File image parser

The GUI includes a firmware image parser to allow firmware image files to be sent unaltered to the target. Because the parser is in the GUI this decreases the size of bootloader code in the target, leaving more free memory space for the application.

After a file has been transferred to the GUI, it assumes the file is a firmware image file and starts parsing it with the first line of the S19 file that stores a part of the firmware image. If the storage is successful, it continues parsing the next part of the image until the whole image is parsed. At the end, the GUI sends the command to program and load the data to the target.

2.4 Flash protection and vector redirection

The bootloader must always be protected from erasing itself or getting corrupted. With flash protection enabled, the worst-case scenario during a botched firmware update is that the application is erased or corrupted, but the bootloader is still intact. The bootloader can then run again and reload the new application.

For this specific case, only 8 KB of memory will be protected. The NVPROT register which is a nonvolatile space of memory must be written with:

```
const unsigned char NVPROT_INIT @0x0000040D = 0xFD;
```

Sometimes it is necessary to secure the flash and enable backdoor key, which be achieved with the following code line:

```
const unsigned char NVOPT_F @0x0000040F = 0x83;
```

This secures the device and the below code line contains the key to unsecure the target.

```
const unsigned char BackdoorKey[8] @0x00000400 = "12345678";
```

Because the first locations are protected, the vectors are also protected. Therefore, vector redirection has to be performed for the application to use its own vectors. The VBR can be used to relocate the exception vector table from its default position in the flash memory (address 0x(00)00_0000) to the base of the RAM (address 0x(00)80_0000).

2.5 Memory map

The memory map of the target will be like the below one.

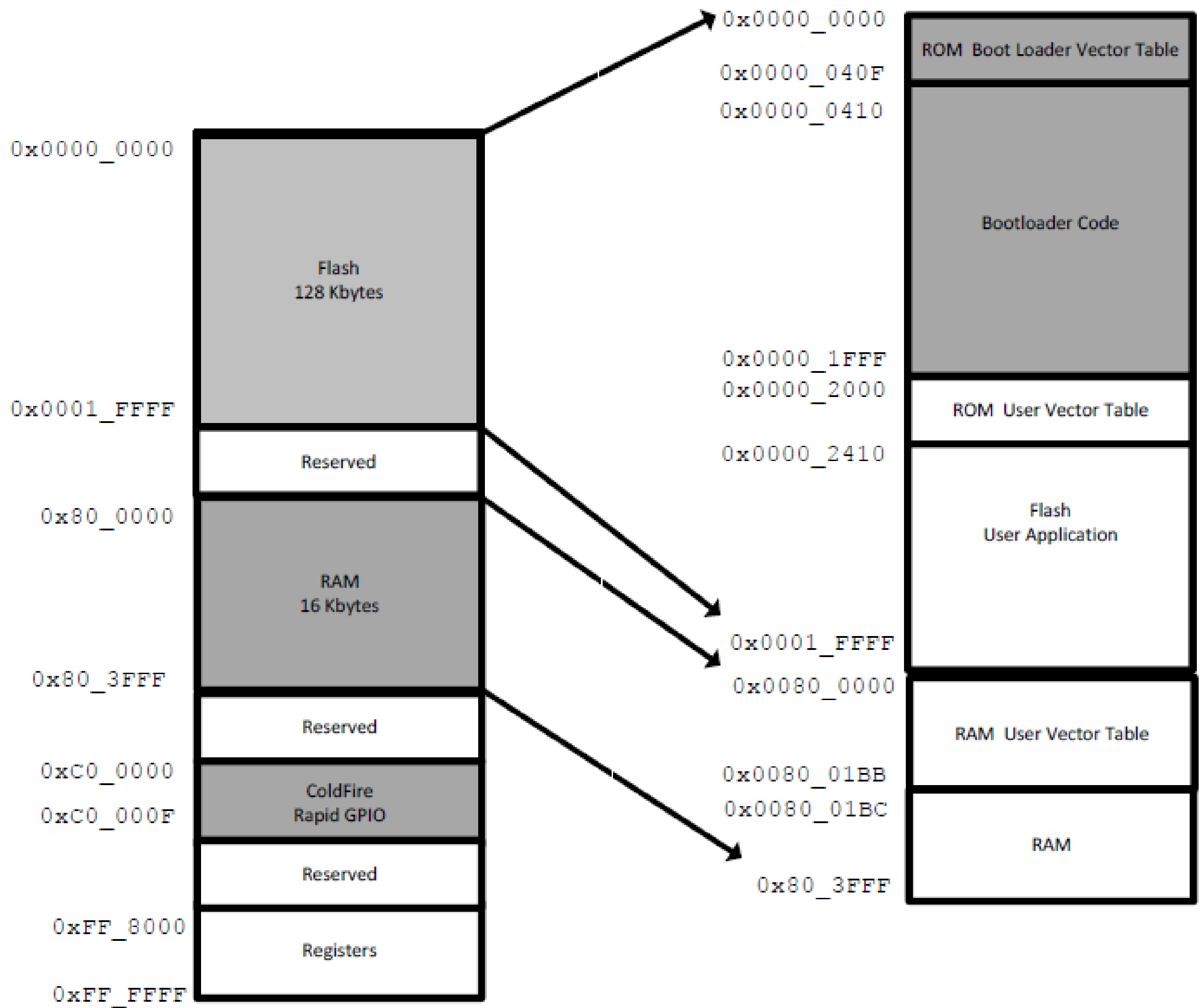


Figure 7. Memory map

3 Developing new applications with CAN bootloader support

To create new applications that can be used with the CAN bootloader, two things must be performed:

- Modify the linker command file .lcf
- Redirect the interrupt vectors

3.1 Modifying the LCF

Normally an application is placed at the beginning of the flash but when the bootloader is used, an application is placed at the beginning of the flash after the bootloader. Therefore, .lcf of the application must be changed.

The following code snippet demonstrates an example of an original lcf file for the MCF51JM.

```
MEMORY {
code  (RX)  : ORIGIN = 0x00000410, LENGTH = 0x0001FBF0
userram (RWX)  : ORIGIN = 0x00800000, LENGTH = 0x00004000
}
```

To make the application compatible with the CAN bootloader the .lcf must be modified like the code given below:

```
MEMORY {
code  (RX)  : ORIGIN = 0x00002410, LENGTH = 0x0001BBF0
userram (RWX)  : ORIGIN = 0x008001BC, LENGTH = 0x00003E44
}
```

It needs to save space for the 8 KB of memory for the bootloader and space for the 110 interrupt vectors in RAM.

For vector reallocation, a new vector table must be created and hardcoded to the first flash address of the application. After this, in the application code, the VBR must be changed to 0x80_0000 and the vectors in flash must be copied to RAM.

Create the new vector table.

```
#define IMAGE_ADDR ((uint_32_ptr)0x02000)
void (* const RAM_vector[])()@(int)&IMAGE_ADDR[0]= {
(pFun)&__SP_INIT, // vector_0 INITSP
(pFun)&_startup, // vector_1 INITPC
(pFun)&dummy_ISR, // vector_2 Vaccerr
(pFun)&dummy_ISR, // vector_3 Vadderr
(pFun)&dummy_ISR, // vector_4 Viinstr
......
(pFun)&dummy_ISR, // vector_64 Virq
(pFun)&dummy_ISR, // vector_65 Vlvd
(pFun)&dummy_ISR, // vector_66 Vlol
(pFun)&dummy_ISR, // vector_67 Vspi1
(pFun)&dummy_ISR, // vector_68 Vspi2
(pFun)&dummy_ISR, // vector_69 Vusb
......
(pFun)&dummy_ISR, // vector_104 VL7swi
(pFun)&dummy_ISR, // vector_105 VL6swi
(pFun)&dummy_ISR, // vector_106 VL5swi
(pFun)&dummy_ISR, // vector_107 VL4swi
(pFun)&dummy_ISR, // vector_108 VL3swi
(pFun)&dummy_ISR, // vector_109 VL2swi
(pFun)&dummy_ISR, // vector_110 VL1swi
};
```

Move the vector base address.

```
// Move vector table to start of RAM, 0x800000
asm (move.l #0x00800000,d0);
asm (movec d0,vbr);
```

Copy the interrupt vectors to RAM.

```
// Copy stored application interrupt vectors
// to re-directed vector table in RAM
pdst=(uint_32_ptr)0x00800000;
psrc=(uint_32_ptr)&RAM_vector;
for (i=0;i<111;i++,pdst++,psrc++)
{
*pdst=*psrc;
}
```

4 Using the bootloader

This section explains how to use the bootloader. The included CodeWarrior projects are the USB-to-CAN bridge and the target. Two DEMOJM boards and a PC are required.

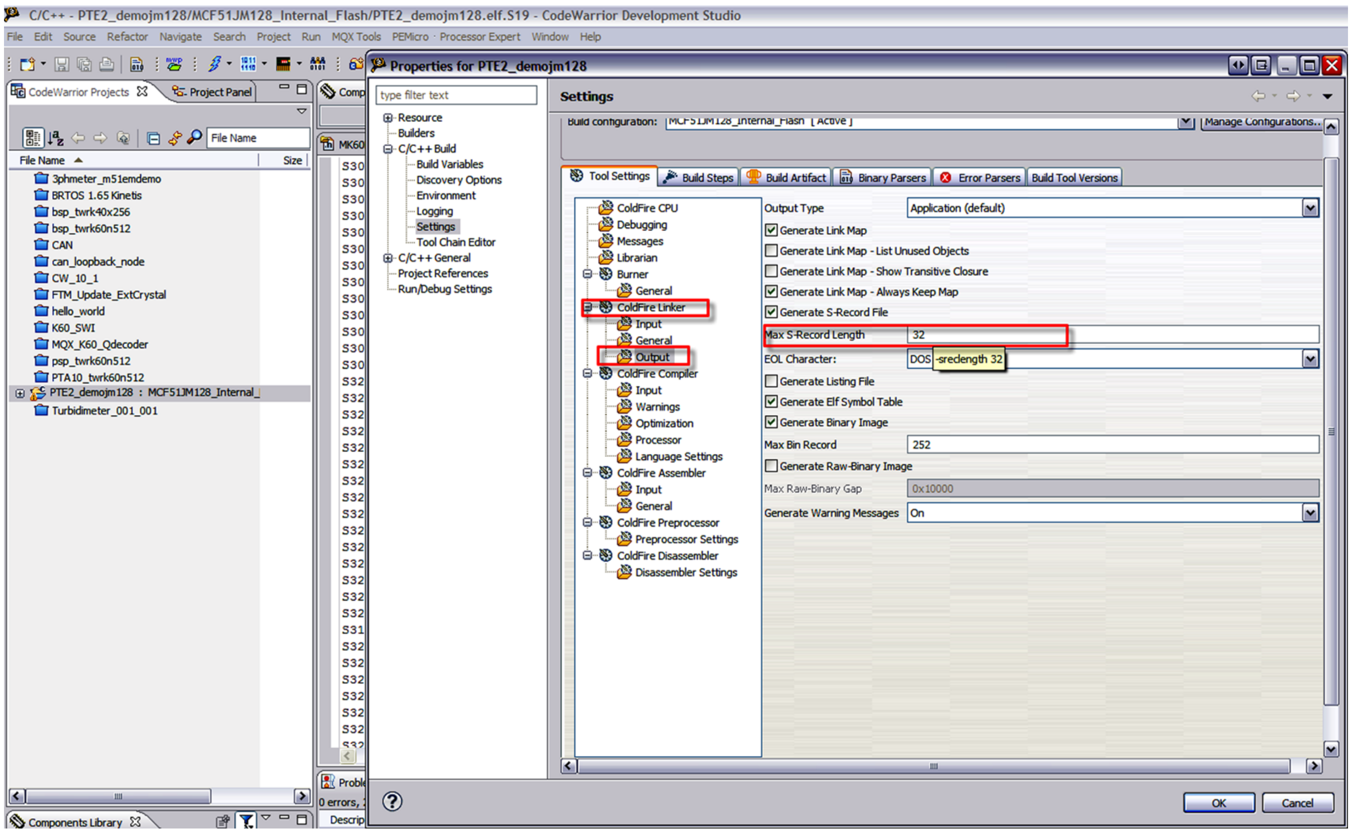
NOTE

Please see DEMOJMUM: DEMOJM User Manual, on http://www.freescale.com for more details on board documentation to get started and learn how to program the boards with CodeWarrior.

1. The USB-to-CAN bridge must be set up for the USB device operation.
2. Load the USB-to-CAN firmware to one of the MCF51JM boards.
3. Load the target or bootloader firmware to the other MCF51JM board.
4. Reset both the boards. The USB-to-CAN bridge must be recognized as a COM virtual. Choose the driver .inf file that is included in the example software attached with this application note.
5. Open the GUI that is included with the example.
6. After modifying the .lcf files and redirecting the interrupts (See Developing new applications with CAN bootloader support), choose the generated S19 file with the Open S-rec button.

NOTE

The maximum S-rec length must be less than 32. This is due to the size of the USB endpoint of the USB-to-CAN bridge. Change the S-rec length in the linker properties under the Output folder of the ColdFire Linker.


C/C++ - PTE2_demojm128/MCF51JM128_Internal_Flash/PTE2_demojm128.elf.S19 - CodeWarrior Development Studio
Properties for PTE2_demojm128
Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire Linker
Input
General
Output
OK
Cancel

Build configuration: MCF51JM128_Internal_Flash [Active]

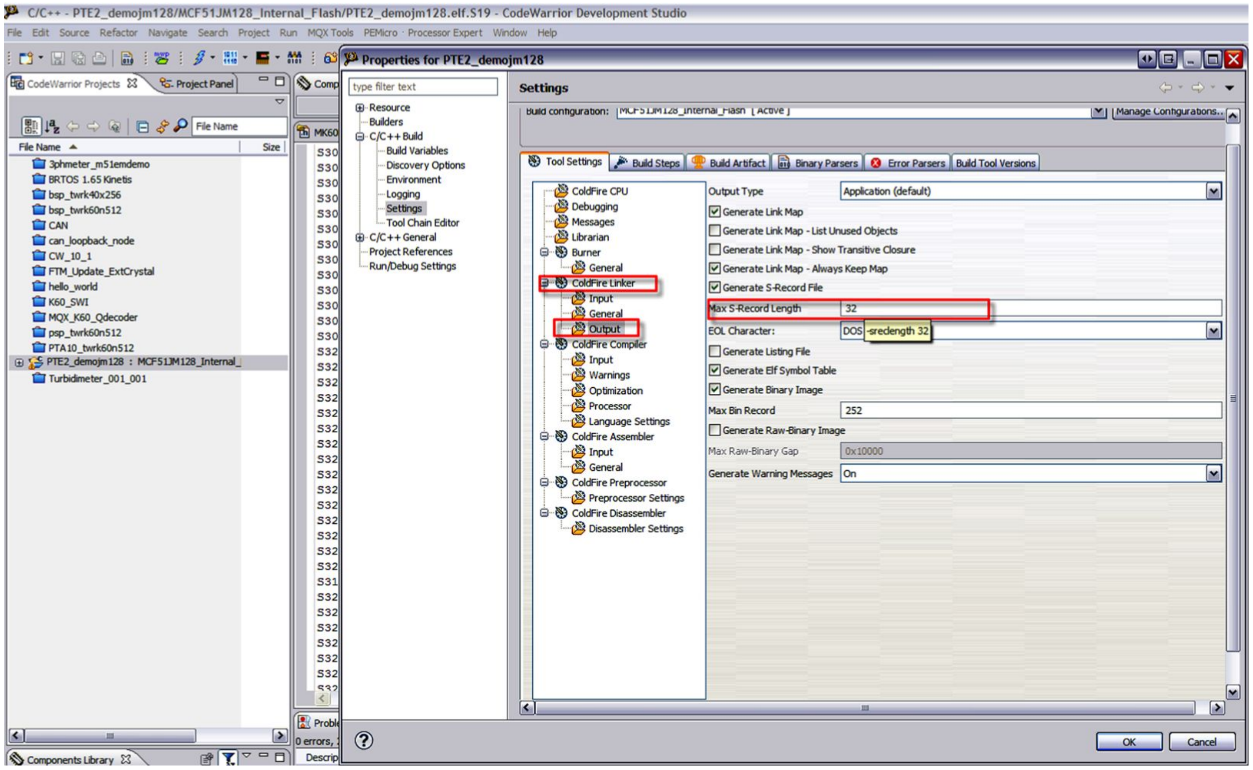
Output Type: Application (default)
- [✓] Generate Link Map
- [ ] Generate Link Map - List Unused Objects
- [ ] Generate Link Map - Show Transitive Closure
- [✓] Generate Link Map - Always Keep Map
- [✓] Generate S-Record File

Max S-Record Length: 32
EOL Character: DOS -sreclength 32
- [ ] Generate Listing File
- [✓] Generate Elf Symbol Table
- [✓] Generate Binary Image

Max Bin Record: 252
- [ ] Generate Raw-Binary Image

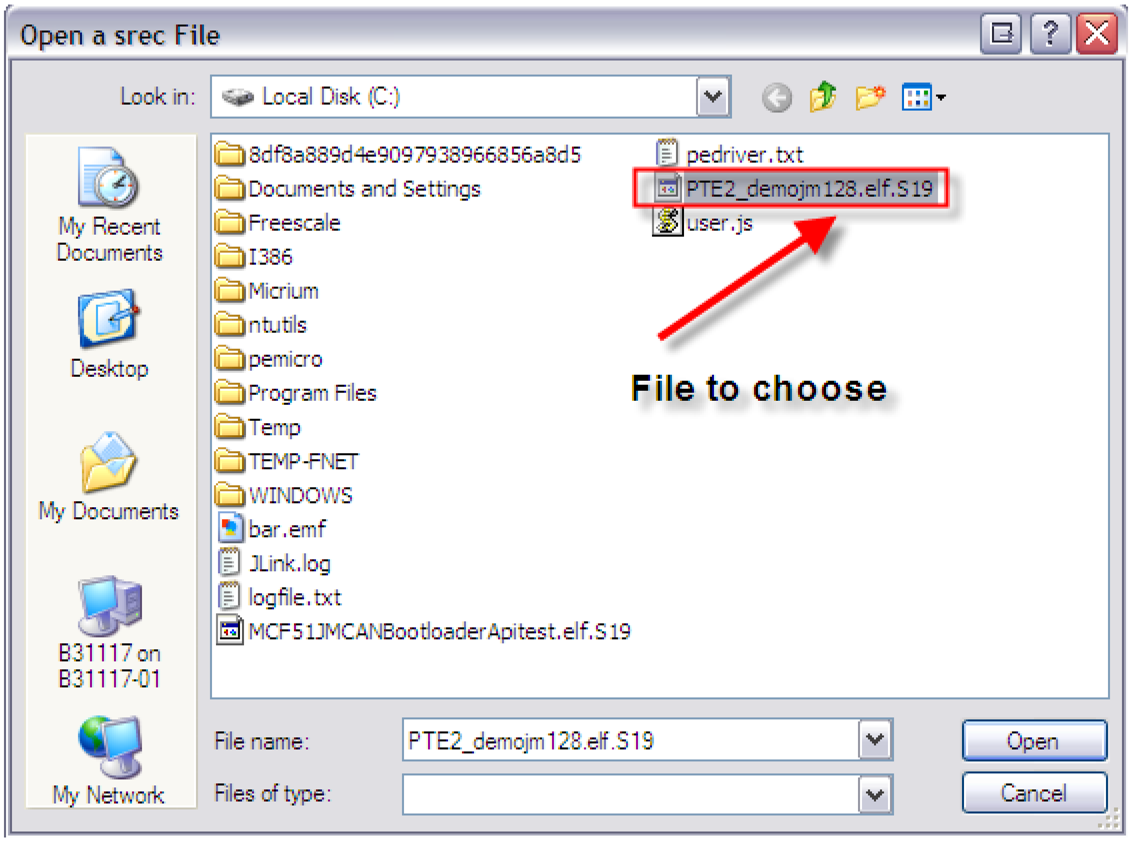
Max Raw-Binary Gap: 0x10000
Generate Warning Messages: On

Figure 8. Modifying the S-record length


C/C++ - PTE2_demojm128/MCF51JM128_Internal_Flash/PTE2_demojm128.elf.S19 - CodeWarrior Development Studio
Properties for PTE2_demojm128
Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire Linker
Output
OK
Cancel

Output Type: Application (default)
☑ Generate Link Map
☐ Generate Link Map - List Unused Objects
☐ Generate Link Map - Show Transitive Closure
☑ Generate Link Map - Always Keep Map
☑ Generate S-Record File
Max S-Record Length: 32
EOL Character: DOS
☐ Generate Listing File
☑ Generate Elf Symbol Table
☑ Generate Binary Image
Max Bin Record: 252
☐ Generate Raw-Binary Image
Max Raw-Binary Gap: 0x10000
Generate Warning Messages: On

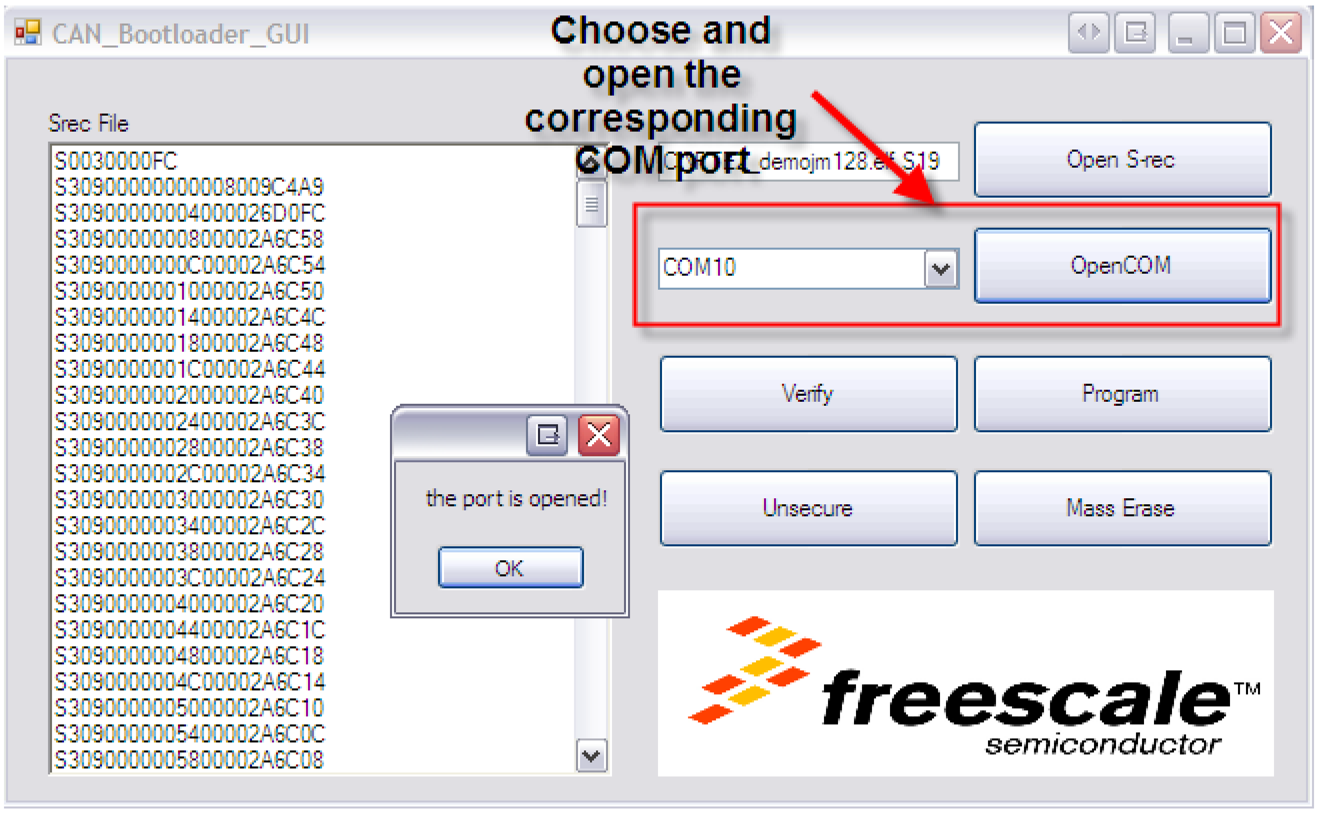
Figure 9. Choose the S-record file



8df8a889d4e9097938966856a8d5
Documents and Settings
Freescale
I386
Micrium
ntutils
pemicro
Program Files
Temp
TEMP-FNET
WINDOWS
bar.emf
JLink.log
logfile.txt
MCF51JMCANBootloaderApitest.elf.S19
pedriver.txt
PTE2_demojm128.elf.S19
user.js

File name: PTE2_demojm128.elf.S19
Files of type:

Figure 10. File to download



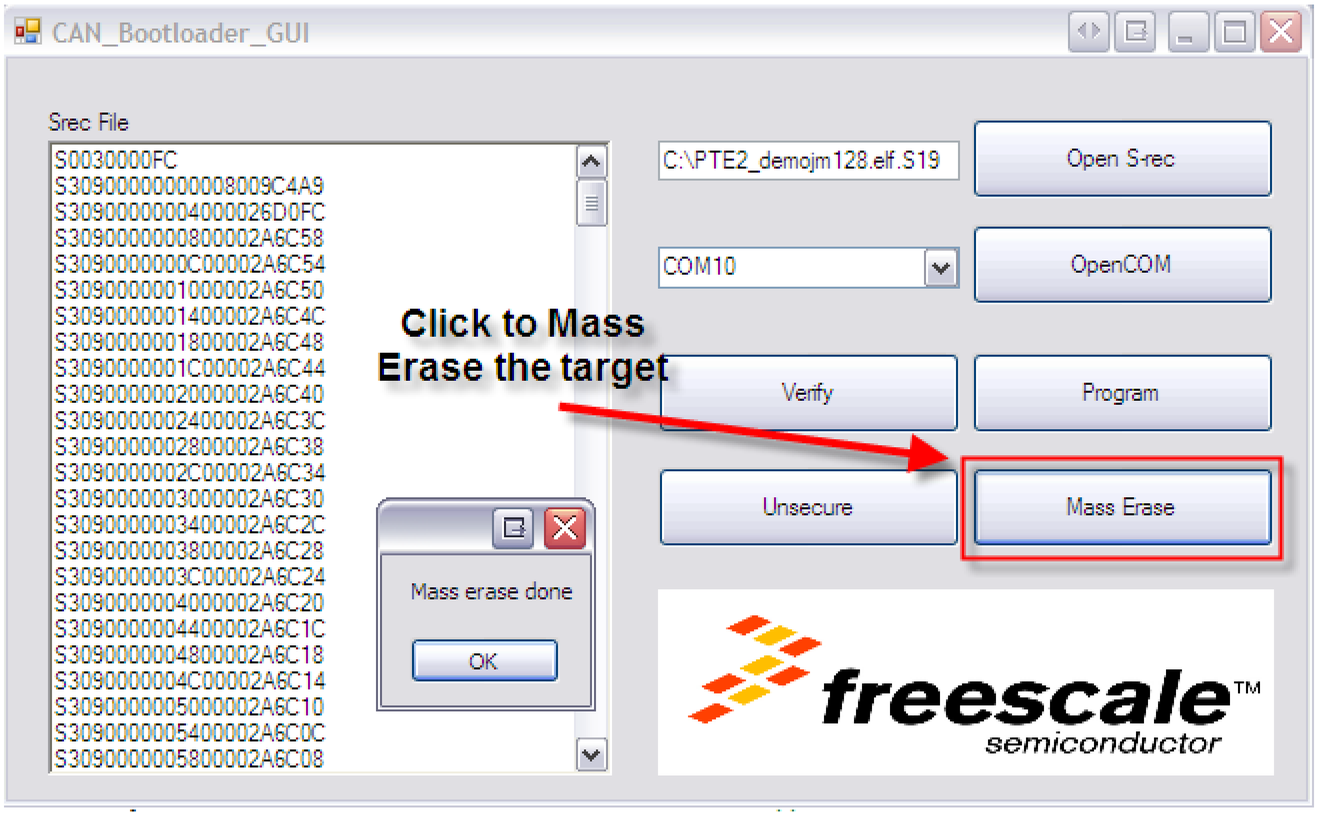
```
S0030000FC
S3090000000000008009C4A9
S30900000004000026D0FC
S30900000008000002A6C58
S3090000000C00002A6C54
S3090000001000002A6C50
S3090000001400002A6C4C
S3090000001800002A6C48
S3090000001C00002A6C44
S3090000002000002A6C40
S3090000002400002A6C3C
S3090000002800002A6C38
S3090000002C00002A6C34
S3090000003000002A6C30
S3090000003400002A6C2C
S3090000003800002A6C28
S3090000003C00002A6C24
S3090000004000002A6C20
S3090000004400002A6C1C
S3090000004800002A6C18
S3090000004C00002A6C14
S3090000005000002A6C10
S3090000005400002A6C0C
S3090000005800002A6C08
```

C...demojm128.elf.S19

the port is opened!

Figure 11. Open the COM port

7. Connect both the boards via CAN.
8. Click the Mass Erase button to erase the target and wait till the message "Mass erase done" appears on the screen (see Figure 12)



```
S0030000FC
S30900000000008009C4A9
S30900000004000026D0FC
S30900000008000002A6C58
S3090000000C00002A6C54
S30900000010000002A6C50
S30900000014000002A6C4C
S30900000018000002A6C48
S3090000001C00002A6C44
S30900000020000002A6C40
S30900000024000002A6C3C
S30900000028000002A6C38
S3090000002C00002A6C34
S30900000030000002A6C30
S30900000034000002A6C2C
S30900000038000002A6C28
S3090000003C00002A6C24
S30900000040000002A6C20
S30900000044000002A6C1C
S30900000048000002A6C18
S3090000004C00002A6C14
S30900000050000002A6C10
S30900000054000002A6C0C
S30900000058000002A6C08
```

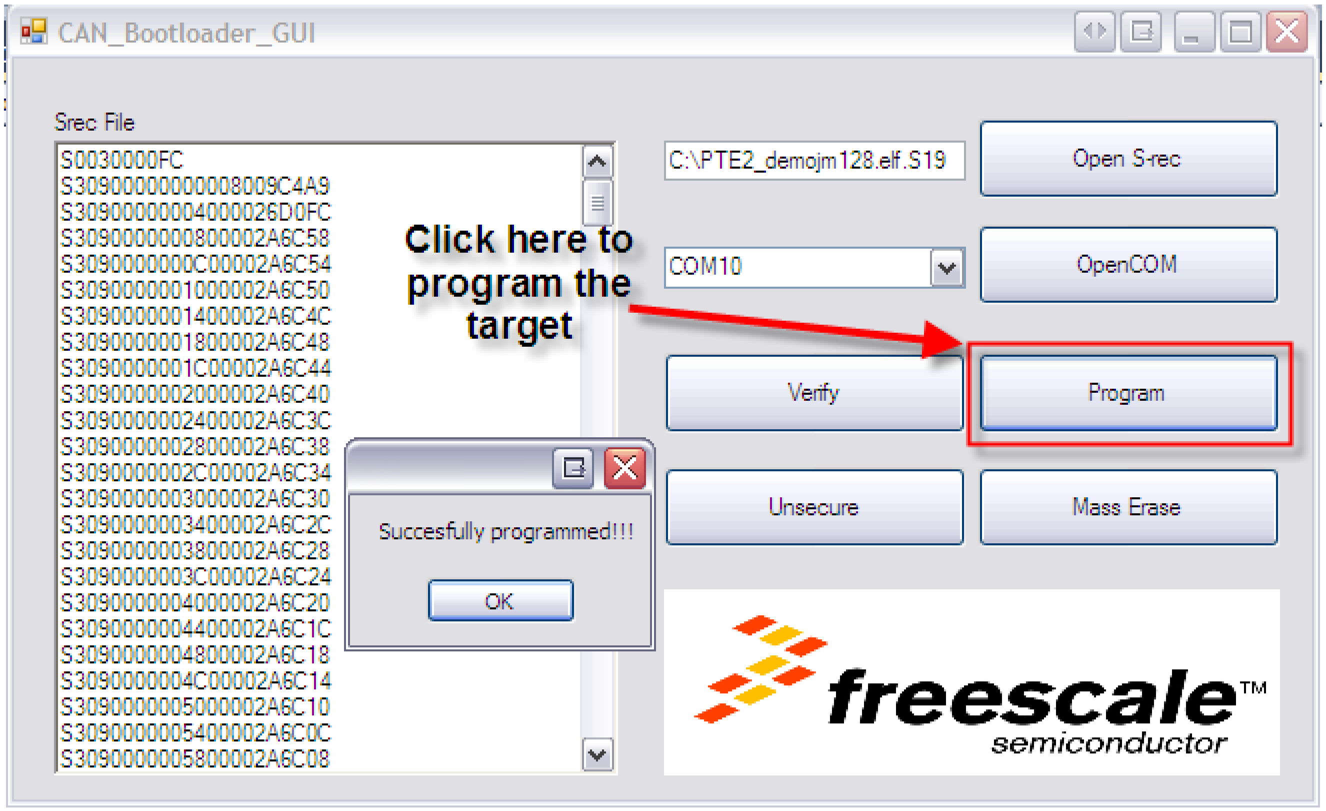
C:\PTE2_demojm128.elf.S19

COM10

Mass erase done

Figure 12. Mass erase

9. Click the Program button to program the target and wait for the message "Successfully programmed" to appear. (Figure 13)



```
S0030000FC
S3090000000000008009C4A9
S309000000004000026D0FC
S30900000000800002A6C58
S309000000000C00002A6C54
S30900000001000002A6C50
S30900000001400002A6C4C
S30900000001800002A6C48
S309000000001C00002A6C44
S30900000002000002A6C40
S30900000002400002A6C3C
S30900000002800002A6C38
S309000000002C00002A6C34
S30900000003000002A6C30
S30900000003400002A6C2C
S30900000003800002A6C28
S309000000003C00002A6C24
S30900000004000002A6C20
S30900000004400002A6C1C
S30900000004800002A6C18
S309000000004C00002A6C14
S30900000005000002A6C10
S30900000005400002A6C0C
S30900000005800002A6C08
```

Succesfully programmed!!!

Figure 13. Flash target

10. To ensure that the project was downloaded correctly, reset the target board, but press the PTG0 switch before reset. For a quick test of the bootloading procees, the application .S19 file attached with the example software can be used. The LED will turn on and off after the firmware update.

5 Conclusions

This bootloader provides a firmware update for devices that feature CAN interfaces. It can be used for automotive applications where the ECM needs a firmware update via the CAN network. This can be used instead of using other methods that need extra hardware.