Freescale Semiconductor
Application Note

Document Number: AN4764
Rev. 0, 8/2013

USB Human Interface Device Boot Loader for ColdFire Plus, Kinetis K, and Kinetis L MCUs

1 Introduction

Boot loader is a small program put into a device that allows user application codes to be programmed to the device. USB boot loaders using the human interface device (HID) class were built for Freescale 32-bit ColdFire Plus and the Kinetis K and L series MCU families. Using the USB HID class provides the advantages of small boot loader code size and the use of standard USB HID drivers provided by all common operating systems.

2 Boot loader overview

The USB HID boot loader provides an easy and reliable way to load user application codes to devices. Boot loader firmware, user application demo firmware and PC software were built to demonstrate how USB HID boot loader systems can be implemented using ColdFire Plus and Kinetis MCUs.

Contents

The 4 KB boot loader, complying with the USB HID class, receives commands and data from the PC to program and erase the flash memory of the MCUs. The application demos show how user programs can be programmed and re-programmed into the MCUs by the boot loader through the PC software running on Windows XP or Windows 7 operating system. The boot loader code and application demos were tested under the Development Kits of the following platforms:

- TWR-MCF51JF ColdFire Plus Tower CPU board
- TWR-K40X256-KIT Kinetis K Tower Kit
- TWR-K60N512-KIT Kinetis K Tower Kit
- TWR-KL25Z48M Kinetis L Tower CPU board
- FRDM-KL25Z Freedom development platform

The memory maps of the boot loader system are shown in the following table.

Table 1. Boot loader memory map

0x0000_0000 to 0x0000_0008	Initial Stack Pointer and Program Counter
0x0000_0009 to 0x0000_03FF	Boot loader code
0x0000_0400 to 0x0000_040F	Flash configuration
0x0000_0410 to 0x0000_0FFF	Boot loader Code
0x0000_1000 to 0x0000_1FFF (256 KB flash device) 0x0000_1000 to 0x0000_3FFF (512 KB flash device)	Reserved
0x0000_1000 to 0x0001_FFFF (128 KB flash device) 0x0000_2000 to 0x0003_FFFF (256 KB flash device) 0x0000_4000 to 0x0007_FFFF (512 KB flash device)	User application interrupt and exception vectors and user application code

The default interrupt and exception vector table is put into the starting address of the flash area and is used by the boot loader, which should remain unaltered. The application interrupt and exception vector table is stored in the flash areas beginning at 0x1000 or at the first unprotected flash area. The interrupt and exception vector table can be redirected to the RAM area by storing the user application interrupt and exception vector table into the application flash area and copying it to the RAM memory in the application startup routines.

The boot loader erases the application flash, parses the user application data, and programs it to the flash memory of the user application area, which is the free flash memory after the boot loader is loaded into the flash. The boot loader flash area has to be protected and may occupy more memory than its actual size.

The code size of the boot loader is 4 KB. If the flash protection block size of a device is larger than 4 KB, the boot loader flash area occupies the same size of the flash protecting block. For MCF51JF128, PKL25Z128/MKL25Z128, MK40X256 and MK60N512, the boot loader area occupies 4 KB, 8 KB and 16 KB of flash when it is protected.

The user application can use the whole RAM memory regardless the size of RAM the boot loader uses.

3 Boot loader architecture

The following figure (Figure 3.1) presents the architecture of the boot loader system.

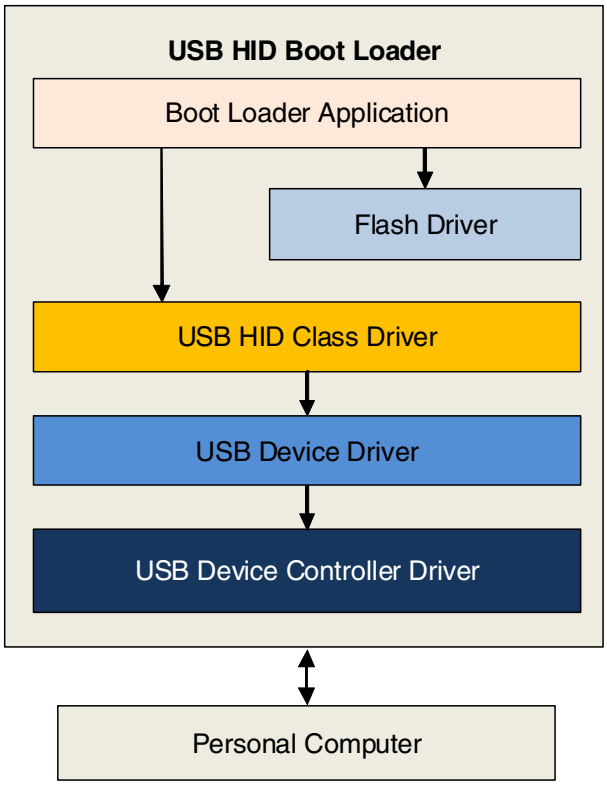

Figure 1. USB HID boot loader architecture

Included in the boot loader is a boot loader application, a flash driver, a USB HID class driver, a USB device driver and a USB device controller driver.

- The boot loader application receives commands and data from the PC. After processing a program or erase command, it sends an acknowledgement to the PC through the HID class driver.
- The flash driver supports the writing, verifying and erasing functions of flash memory.
- The USB HID class driver contributes to the application program interface specified in the USB HID class.
- The USB device driver and the USB device controller driver communicate with the PC though the USB protocols.

3.1 Boot loader software flow

The boot loader system is integrated with a boot loader firmware for user program upgrade and user application performing the product's main function. After reset and initialization, the system determines if the user application program or the boot loader mode should start. If there is no valid user application

program, the device will automatically start in boot loader mode. If there is a valid application, the device will run the boot loader program if a specified key is pressed. Otherwise, it will run the user application.

Once the system has entered the boot loader mode, it continues to check on whether any commands and data are received. The following figure is the flow chart of the boot loader:

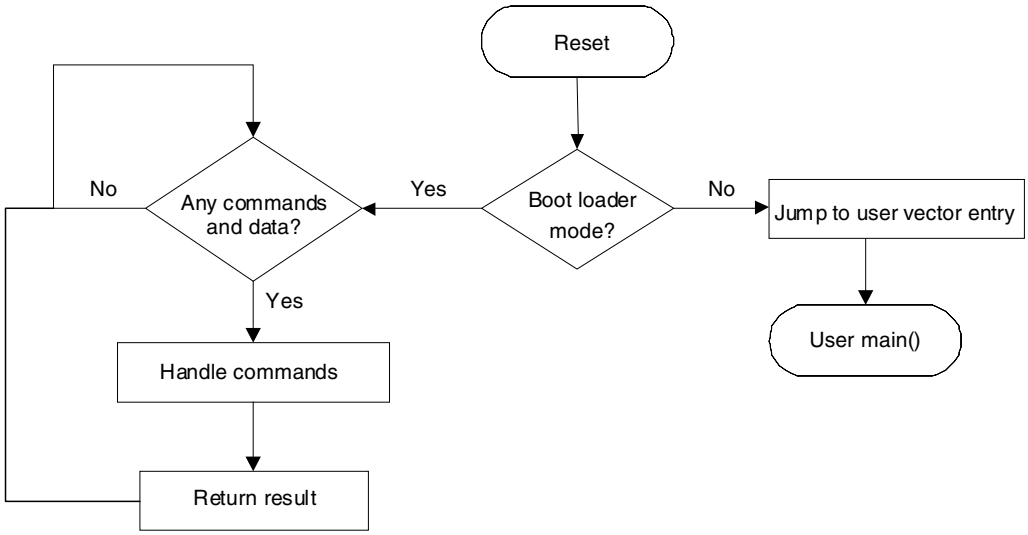

Figure 2. Boot loader software flow

4 Developing new user applications

This section describes how normal application programs can be modified for the bootloader system.

4.1 Modifying linker files

For normal applications using ColdFire Plus or Kinetis MCUs, the interrupt vector table is located at the beginning of the flash area, and the application code can be put into any other program flash areas. In a boot loader system, certain areas of the beginning flash are occupied by the boot loader code and the application code must be put into other flash areas. The linker file must be modified to direct the linker to put the application interrupt vector table and the application code into specified memory regions.

4.1.1 Modifying CFV1 or CFV1 plus linker file

The following code of a normal application linker file, "project.lcf" for MCF51JF128, tells the linker that the application code can be put into the flash area of 0x410 to 0x1FFFF.

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

To work with the boot loader system for MCF51JF128, the user application should be located at the flash memory area starting from the address 0x1410. The linker file can be modified as the following:

```
MEMORY {
```

```
   code        (RX)  : ORIGIN = 0x00001410,LENGTH = 0x0001EBF0
   userram    (RWX) : ORIGIN = 0x00800400,LENGTH = 0x00007C00
}
```

4.1.2 Modifying Kinetis K linker file

The following code of a normal application linker file, "MK60N512VMD100_flash.icf" for MK60N512VMD100, tells the linker that the application code can be put in the flash area of 0x410 to 0x7FFFF.

```
MEMORY
{
  interrupts  (RX): ORIGIN = 0x00000000, LENGTH = 0x00000400
  cfmprotrom  (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
  code        (RX) : ORIGIN = 0x00000410, LENGTH = 0x0007FBF0
}
```

To work with our boot loader system for MK60N512VMD100, the user application should be located at the flash memory area starting from the address 0x4410. The linker file can be modified as the following:

```
MEMORY
{
  Interrupts(RX)  : ORIGIN = 0x00004000, LENGTH = 0x00000400
  cfmprotrom (RX) : ORIGIN = 0x00004400, LENGTH = 0x0000010
  code        (RX) : ORIGIN = 0x00004410,LENGTH = 0x0007BBF0
}
```

4.1.3 Modifying Kinetis L linker file

The following code of a normal application linker file, "128KB_Pflash.icf" for PKL25N128, tells the linker that the code can be put in the flash area of 0x410 to 0x1FFFF.

```
define symbol __ICFEDIT_region_ROM_start__ = 0x00000000;
 define symbol __ICFEDIT region_ROM_end__ = ICFEDIT_region_ROM_start__ + (128*1024);
 define symbol __code_start__ = 0x00000410;
```

To work with our boot loader system for PKL2560N128, the user application should be located at the flash memory area starting from the address 0x10C0. The linker file can be modified as below:

```
define symbol __ICFEDIT_region_ROM_start__ = 0x00001000;
 define symbol __ICFEDIT region_ROM_end__ = ICFEDIT_region_ROM_start__ + (124*1024);
 define symbol __code_start__ = 0x000010C0;
```

4.2 Redirecting interrupts and exception vectors

The default interrupt and exception vector table is fixed at the starting address of the flash area and used by the boot loader, which should remain unaltered. If the user application uses interrupts, the user application interrupt and exception vector table should be put into the flash or RAM memory. The way to redirect and use interrupt vectors in RAM may be different for different MCUs. This section describes how vector redirection can be done when using the Freescale MQX stack and other stacks.

4.2.1 MQX stack

The Freescale MQX stack uses the identifier MQX_ROM_VECTORS in the configuration file, "userconfig.h", to configure applications to put interrupt vectors in RAM or ROM (flash). Customers can define MQX_ROM_VECTORS to 0 to put the interrupt vectors into RAM. Remember to rebuild the library if the configuration file is changed.

```
/* userconfig.h */
#define MQX_ROM_VECTORS   0 // 1=ROM, 0=RAM
```

4.2.2 Other software stack

The way to redirect the vector table may be different for different platforms. The following sections describe how to redirect the vectors table to RAM for ColdFire V1 (CFV1) Plus MCUs.

4.2.3 CFV1 Plus ColdFire

Because the vector table starting from the address 0x0000 is used by the boot loader, the redirected vector table should be put into RAM. The vector base register (VBR) of CFV1 Plus ColdFire (eg: MCF51JF128) contains the 1-MB-alligned base address of the exception vector table, which can be used to relocate the vector table from its default position in the flash memory (address 0x0000) to the base address of the RAM (eg: 0x0080_0000).

The following code is the original code for the linker to put the interrupt routines address into the original flash vector table.

```
/* exceptions.c */
__declspec(weak) vectorTableEntryType vector_0 @INITSP = (vectorTableEntryType)&_SP_INIT;
__declspec(weak) vectorTableEntryType vector_1 @INITPC = (vectorTableEntryType)&_startup;
__declspec(weak) vectorTableEntryType vector_2 @Vaccerr = asm_exception_handler;
```

The application vector table is put into the application flash area and then copied into RAM. The new vector table in the boot loader framework can be declared by the following code.

```
/* user_configure.h */
#define  APPLICATION_FLASH_START0x00001000;

/* exceptions.c */
#define GetIntVectAddr_BL(Vnum) ((Vnum * 4)+APPLICATION_START_ADDRESS)
__declspec(weak) vectorTableEntryType vector_0 @(GetIntVectAddr_BL(INITSP))=
(vectorTableEntryType)&_SP_INIT;
 __declspec(weak) vectorTableEntryType vector_1 @( GetIntVectAddr_BL (INITPC))=
__declspec(weak) __declspec(weak) vectorTableEntryType vector_2 @(GetIntVectAddr_BL(Vaccerr))
= asm_exception_handler;;}
```

The following code copies the above vector table from the application flash area to the vector table region in RAM.

```
/* main.c */
#define  APP_RAM_START0x00800000;
         uint32 *pdst;
         uint32 *psrc;
         pdst=(uint32 *) APP_RAM_START
```

```
        psrc=(uint32 *) APPLICATION_START_ADDRESS;
        for (i=0;i<110;i++){
                *pdst++=*psrc++;}
```

The following code is used to configure the MCU to use vector table at the RAM address 0x0080_0000.

```
/* main.c */
        asm (move.l  #0x00800000,d0);
        asm (movec  d0,vbr);
```

4.2.4 Kinetis K and L microcontrollers

In Kinetis K and L MCUs, the SCB_VTOR register contains the base address of the exception vector table. To redirect the vector table, copy the vector table to RAM and set the SCB_VTOR to the RAM address. Most of the released example codes have implemented vector redirection.

5 MCF51JF128 demo

The demo shows the procedures of programming the boot loader code into TWR-MCF51JF board and programming and re-programming the application demo code though the PC software "HIDBootloader.exe". The following table shows how the MCF51JF128 pins are used.

Table 2. MCF51JF128 pins usage

Pins	Function
J3 pin 2 (PTA4)	Short to Ground to enter boot loader mode
SW1 (PTB0)	Press it to toggle LED 2 during demo application

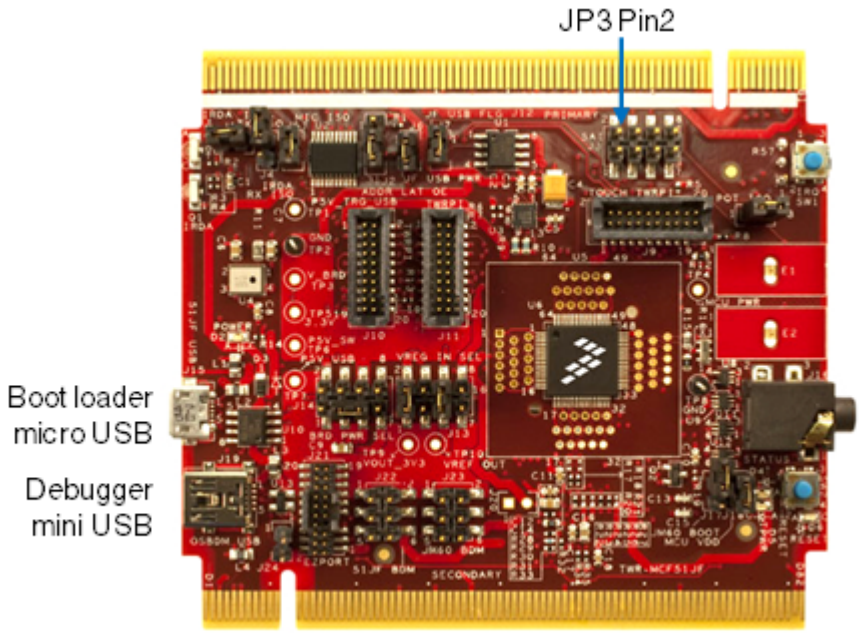

Figure 3. TWR-MCF51JF board

5.1 Programming the boot loader

The boot loader code was built and tested under CodeWarrior 10.4. The code can be programmed into the TWR-MCF51JF board using IAR, CodeWarrior, Keil or any other programming tools. The following steps show how the boot loader code can be compiled and programmed under CodeWarrior.

1. Connect a USB cable from the PC to the mini USB debug port of the TWR-MCF51JF board.
2. Launch CodeWarror 10.4.
3. Select a workspace.
   - For example: Workspace: usb_hid_bootloader.
4. Import project.
   - Click file -> Import to import a project.
   - Expand the General Directory.
   - Choose Existing Projects into workspace.
   - Click the Next button.
   - Click the Browse button to browse the project.
   - Choose the directory jf128_hid_bootloader.
   - Click OK to confirm.
   - Select the project jf128_hid_bootloader.
   - Click the Finish button.
   - Click the Yes button if the Remote System Missing window pops up.
   - Close the Target Tasks and the Welcome windows if they appear.
5. Build project.
   - Select CV1Plus_USB_HID_Bootloader\ jf128_hid_bootloader in the CodeWarrior Projects window.
   - Select Project -> Build All to rebuild the project.
6. Program boot loader.
   - Select run -> Debug Configurations.
   - Expand CodeWarrior Download.
   - Choose jf128_hid_bootloader_MCF51JF128_Internal_Flash_PnE U-Multillink.
   - Click the Debug button to program the boot loader to the device. The device has now been programmed with the boot loader code.
7. Close CodeWarrior.

5.2 Programming application demo into device

1. Press the reset button to reset the board to run the boot loader. Because there is no user application program, the system will enter boot loader mode.
2. Connect a USB cable from the PC to the micro USB boot loader port of the TWR-MCF51JF board.
3. Run the file PC_Software\HIDBootloader.exe.

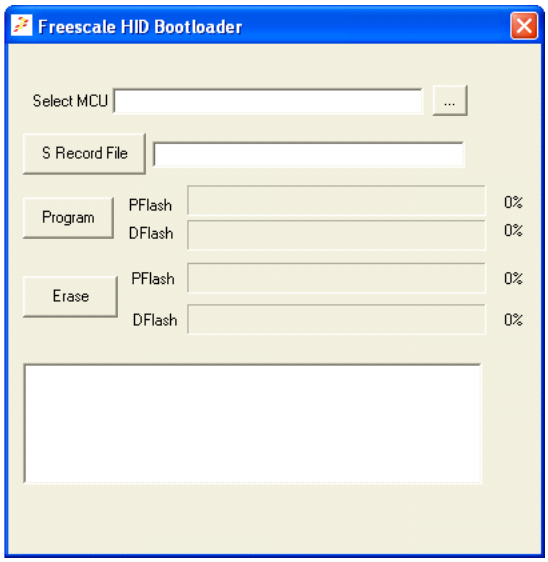

Figure 4. HID boot loader

4. Select MCF51_128KB_PFLASH.imp in Select MCU.
5. Select Image_Files\TWR-MCF51JF_Bootloader_Application_Example.s19 in S Record File. Customers can choose their own s-record to program.
6. Click Program to program the application code to the device.

5.3 Running the application example

1. Reset the device by clicking the Reset button. Now the application example is running and LED is ON.
2. Press SW1 to toggle LED 2.

5.4 Re-entering the boot loader mode

1. Connect JP3 pin 2 to ground (remember to disconnect it before running a user application). Reset the device by clicking the Reset button.
2. The boot loader should now be running. The new program can now be programmed into the device.

6 Kinetis L demo

The demo shows the procedures of programming the boot loader code into the FRDM-KL25Z and the TWR-KL25Z48M boards, and programming and re-programming of the application demo code through the PC software "HIDbootloader.exe". The following table (Table 6.1) shows how the pins are used.

Table 3. PKL25N128 pin to enter boot loader

Pins	Function
FRDM-KL25Z – PTE31	Connect it to ground and then reset the device to enter boot loader mode
TWR-KL25Z48M – SW4 (PTC3)	Pressing it and then reset the device to enter boot loader mode

The following figures present the FRDM-KL25Z and TWR-KL25Z48M boards.

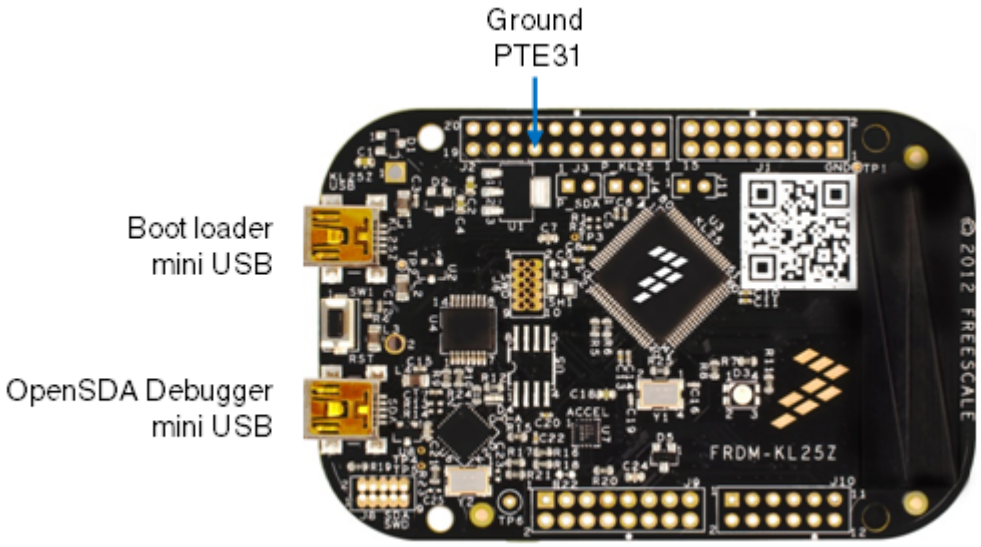

Figure 5. FRDM-KL25Z board

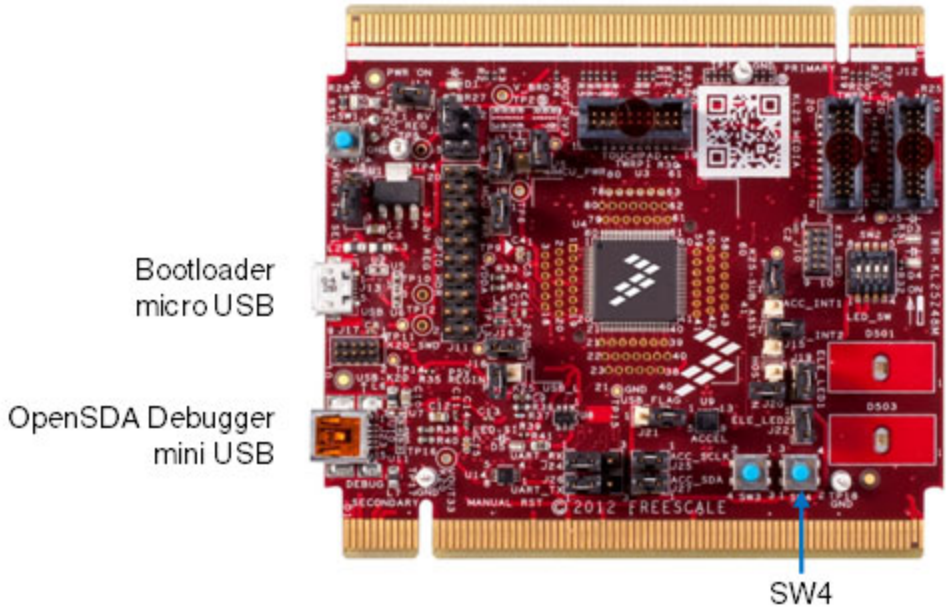

Figure 6. TWR-MCF51 board

6.1 Programming the boot loader

The boot loader code was built and tested under IAR Embedded Workbench IDE version 6.60.1. The code can be programmed into the FRDM-KL25Z or the TWR-KL25Z48M board using IAR, CodeWarrior, Keil or any other programming tools. Please note that patch files may be needed for some development and programming tools. The following steps explain how the boot loader code can be compiled and programmed under IAR.

1. Launch the IAR embedded Workbench IDE.
2. Open the project file "kinetis_L_USB_HID_Bootloader\build\iar\kl25z_HID_Booltoader\kl25z_Lite_Bootloader.eww".
3. Connect a USB cable from the PC to the mini USB debugger port of the FRDM-KL25Z or the TWR-KL25Z48M board.
4. Choose the platform kl25z_Lite_Bootloader_freedom for the FRDM-KL25Z board. Choose the platform kl25z_Lite_Bootloader_tower for the TWR-KL25Z48M board.
5. Click Project -> Rebuild All to rebuild the project.
6. Click Project -> Options -> Debugger -> Setup, in Driver, select PE micro.
7. Click Project -> Download -> Erase memory -> Erase all to erase the flash memory.
8. Click Project -> Download -> Download active application to download the bootloader code into the KL25 MCU.
9. PEMICRO Connection Manager will be popped up. In Interface, choose OpenSDA Embedded Tower Debug - USB Port. The device has been programmed with the bootloader code.
10. Close IAR.

6.2 Programming the application example

1. Press the reset button to reset the board to run the boot loader. Since there is no user application program, the system will enter boot loader mode.
2. Connect a USB cable from the PC to the mini USB boot loader port for the FRDM-KL25Z board. Connect a USB cable from the PC to the micro USB boot loader port for the TWR-KL25Z48M board.
3. Run the file PC_Software\HIDBootloader.exe.

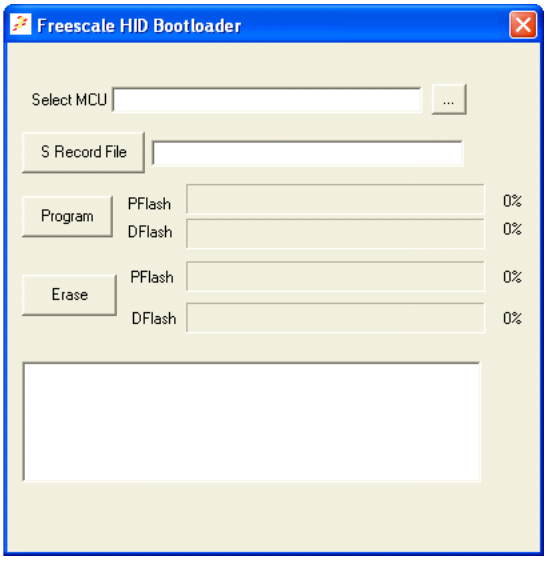

Figure 7. HID boot loader program

4. Select Kinetis_L_128KB_PFlash.imp in Select MCU, .
5. Select Image_Files\ KL25_blinky_freedom.srec in S Record File for the FRDM-KL25Z board. Select Image_Files\ KL25_blinky_tower.srec in S Record File for the TWR-KL25Z48M board.
6. Click Program to program the application code to the device.

6.3 Running the application example

Reset the device by clicking the Reset button. Now the application example is running and the LEDs are flashing.

6.4 Re-entering the boot loader mode

1. Connect PTE31 to ground for the FRDM-KL25Z board (remember to disconnect it before running user application). For the TWR-KL25Z48 board, pressing SW4.
2. Press and release the reset button. Now the boot loader is running.
3. Release SW4 for the TWR-KL25Z48 board. The new program can now be programmed into the device.

7 Kinetis K demo

The demo shows the procedures of programming the boot loader code into TWR-K40X256 and TWR-K60N512 kits, and programming and re-programming of the application demo code thought the PC software "HIDbootloader.exe". The following table shows how the MCU pins are used.

Table 4. MCU pins usage

Pins	Function
TWR-K40 – SW1 (PTC5) TWR-K60 – SW1 (PTA19)	Pressing it and then reset the device to enter boot loader mode
TWR-K40 – PTC13 (SW2) TWR-K60 – PTE26 (SW2)	Press it in the GPIO demo to toggle the LED

The figure below shows the development board of the K60 Tower development kit TWR-K60N256-KIT.

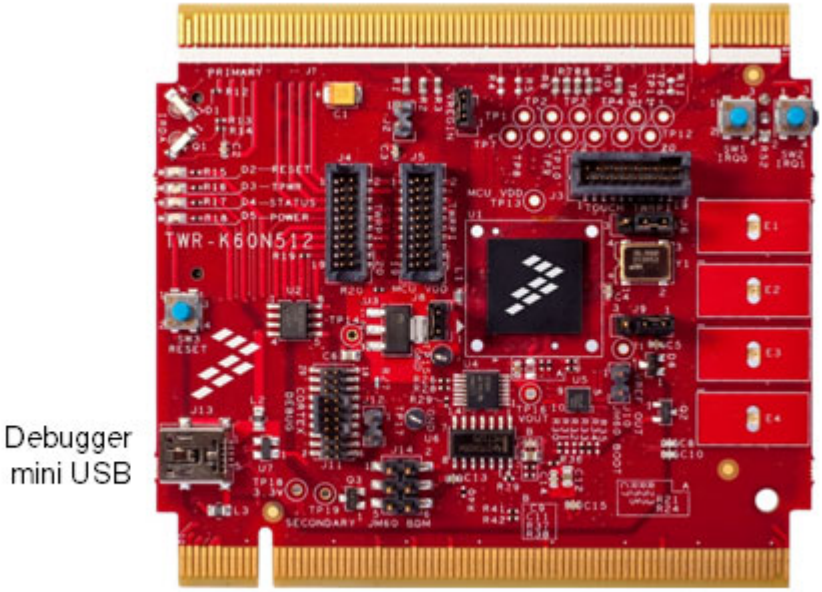

Figure 8. TWR-K60N512 CPU board

7.1 Programming the boot loader

The boot loader code was built and tested under IAR Embedded Workbench IDE version 6.60.1. The code can be programmed into the TWR-K40X256 or the TWR-K60N512 kit using IAR, CodeWarrior, Keil or any other programming tools. The following steps show how the boot loader code can be compiled and programmed under IAR.

1. Construct the TWR-K40X256 or TWR-K60N512 Tower system kit including the Tower serial module (TWR-SER).
2. Launch IAR embedded Workbench IDE.
3. Open the project file "Kinetis_K_USB_HID_ICP\build\iar\USB_hid_icp"
4. Connect a USB cable from the PC to the USB port of the Kinetis board.
5. Choose the platform of USB_hid_icp_k40_tower for the TWR-K40X256 kit.
6. Choose the platform of USB_hid_icp_k60_tower for the TWR-K60N512 kit.
7. Click Project > Rebuild All to rebuild the project.

8. Click Project > Options > Debugger > Setup, in Driver, select PE micro
9. Click Project -> Download -> Erase memory -> Erase all to erase the flash memory.
10. Click Project -> Download -> Download active application to download the bootloader into the Kinetis MCU. The device has been programmed with the bootloader code.
11. Close IAR.

7.2 Programming the application example

1. Press the reset button to reset the board to run the boot loader. Since there is no user application program, the system will enter boot loader mode.
2. Connect a USB cable from the PC to the mini USB port of the TWR-SER board.
3. Run the file PC_Software\HIDBootloader.exe.

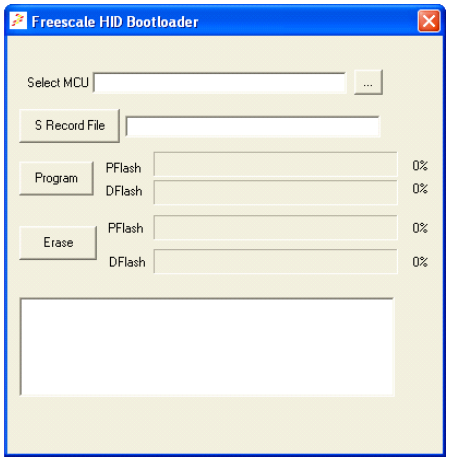

Figure 9. HID boot loader program

4. Choose Kinetis_256KB_PFlash_256KB_DFlash in Select MCU for the TWR-K40X256 kit. Choose Kinetis_K_512KB_PFlash.imp in Select MCU for the TWR-K60N512 kit.
5. For the TWR-K40X256 kit, in S Record File, select Image_Files\USB_device_k40_tower_gpio.srec. For the TWR-K60N512 kit, in S Record File, select Image_Files\USB_device_k60_tower_gpio.srec.Customers can choose their own s-record to program.
6. Click Program to program the application code to the device.

7.3 Running the application example

1. Reset the device by pressing the Reset button. The application example should now be running.
2. Press SW1 or SW2 will toggle LEDs.

7.4 Re-entering the boot loader mode

1. Press SW1.
2. Click the reset button.

3. Release SW1. The boot loader should now be running, and the new program can now be programmed into the device.

8 Customization

When plotting the examples to other platforms, the following factors must be considered:

- "The BDM or programming interface
- "Method of re-entering boot loader mode

8.1 The BDM or programming interface

Freescale provides the following embedded BDM interfaces on the ColdFire and Kinetis MCU developments boards. No external BDM hardware is required.

- "P&E Multilink/Cyclone Pro (for example: M51JM128EVB)
- "CFV1 Open Source BDM (for example: TWR-MCF51JF)
- "PEMICRO_USB (for example: M52259EVB)
- "ColdFire v2-v4 JM60 OSBDM (for example: TWR-MCF5225X)
- "USB Multilink, Embedded OSJTAG - USB Port (for example: TWR-K60N512)
- "OpenSDA (for example: FRDM-KL25Z board)

Customers using a different BDM interface must set the corresponding BDM interface correctly.

8.2 Method of re-entering boot loader mode

In the demos, connecting a special pin to ground during power-up (or reset) forces the system to enter boot loader mode. Customers can choose other input pins or other methods to enter boot loader mode.

9 Conclusion

Sample codes for the USB HID class boot loader have been built for Freescale 32-bit ColdFire Plus and Kinetis K and L series MCU families. User application codes can be programmed to the MCUs through the HIDBootloader.exe PC program. Customers can plot the example codes to other Freescale MCUs and customize the codes for their own applications.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4764
Rev. 0
08/2013