Technical Document

CAN Boot Loader Design on the Kinetis EA128

Contents

1 Introduction

This technical document is used to introduce the technical implementation for CAN boot loader designed on the KEAZ128.

The boot loader is a built-in firmware implemented to program the application code to on-chip nonvolatile memory (flash on KEA) via the communication interface. This document introduces how to implement the CAN boot loader on the KEAZ128 board.

2 Introduction of KEA

The Kinetis Auto family is a highly scalable portfolio of 32-bits ARM Cortex-M0+MCUs aimed for the automotive markets. The family is optimized for cost-sensitive applications offering low pin-count option with very low power consumption. With 2.7-5.5 V supply and focus on exceptional EMC/ESD robustness, Kinetis Auto series devices are well suited to a wide range of applications ranging from body applications, safety companion chip or generic sensor nodes. In automotive body application, the Kinetis Auto family is a great option for entry level body controller or gateway module, window/roof/sun-roof controller, immobilizer or seat/mirror controller just to mention a few.

Many modules are available on KEAZ128 including ARM Cortex-M0+ core, internal reference for 48 MHz system clock. system module such as SIM/PWC/WDOG/AIPS/BME/MCM, memories such as Up to 128 KB flash memory/ Up to 16KB RAM, Clocks, ADC, Timers such as RTC/PIT/FTM/Systick, communication modules such as CAN/LIN/UART/SPI/I2C and HMI modules.

2.1 Boot Loader Memory Map

This device contains various memories and memory-mapped peripherals which are located in a 4 GB memory space.

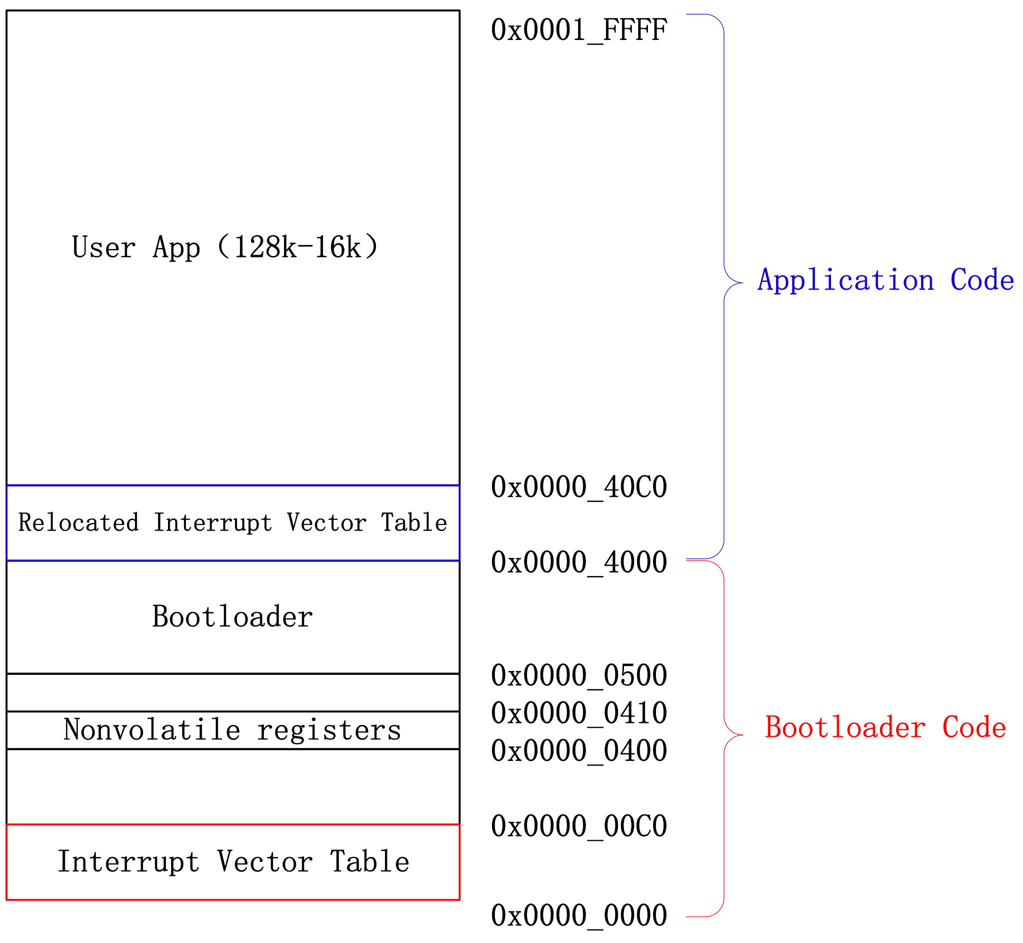

Figure1. Memory allocation

2.2 flash module

The flash memory is ideal for single-supply applications allowing for field reprogramming without requiring external high voltage sources for program or erase operations. The flash module includes a memory controller that executes commands to modify flash memory contents. The user interface to the memory controller consists of the indexed Flash Common Command Object (FCCOB) register which is written to with the command, global address, data, and any required command parameters. The memory controller must complete the execution of a command before the FCCOB register can be written to with a new command. (KEA128RM Chapter18)

2.2 CAN module

Freescale's scalable controller area network (MSCAN) is a communication controller implementing the CAN 2.0A/B protocol as defined in the Bosch specification dated September 1991. For users to fully understand the MSCAN specification, it is recommended that the Bosch specification be read first to familiarize the reader with the terms and concepts contained within this document.

Though not exclusively intended for automotive applications, CAN protocol is designed to meet the specific requirements of a vehicle serial data bus: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth.

MSCAN uses an advanced buffer arrangement resulting in predictable real-time behavior and simplified application software.

3 Software architecture

Software tools on the PC, DemoCAN.exe, can be downloaded from the freescale compass and will decode the S19 file and communicate with the target device via CAN transceiver. It is compatible with boot loader communication (BLC) protocol.

3.1 BLC protocol

Using the DemoCAN.exe to open the application s19 file, the PC will send FF (*DOWN_LINK*) data to KEAZ128 until the KEAZ128 demo board feedback. The KEAZ128 sends C3 (*UP_READY*) to the PC via a high speed CAN transceiver (33901). Then, the PC sends one standard frame of application s19 file after receiving a C3 (*UP_READY*) feedback. After sending the last frame of a line of s19 file, PC sends an FE (*DOWN_LINE_END*) to KEAZ128. The KEAZ128 feedback C2 (*UP_BUSY*) to ask PC not to send CAN frames and begin to program the one line of s19 file to flash. After that, the KEAZ128 change to send C3 (*UP_READY*) to be ready. After sending the whole s19 file, PC sends FD (*DOWN_FILE_END*) to

KEAZ128 with the feedback C1 (*UP_PRGEND*) and the update is successful. The flow chart of BLC protocol based on the KEA is shown in figure 2.

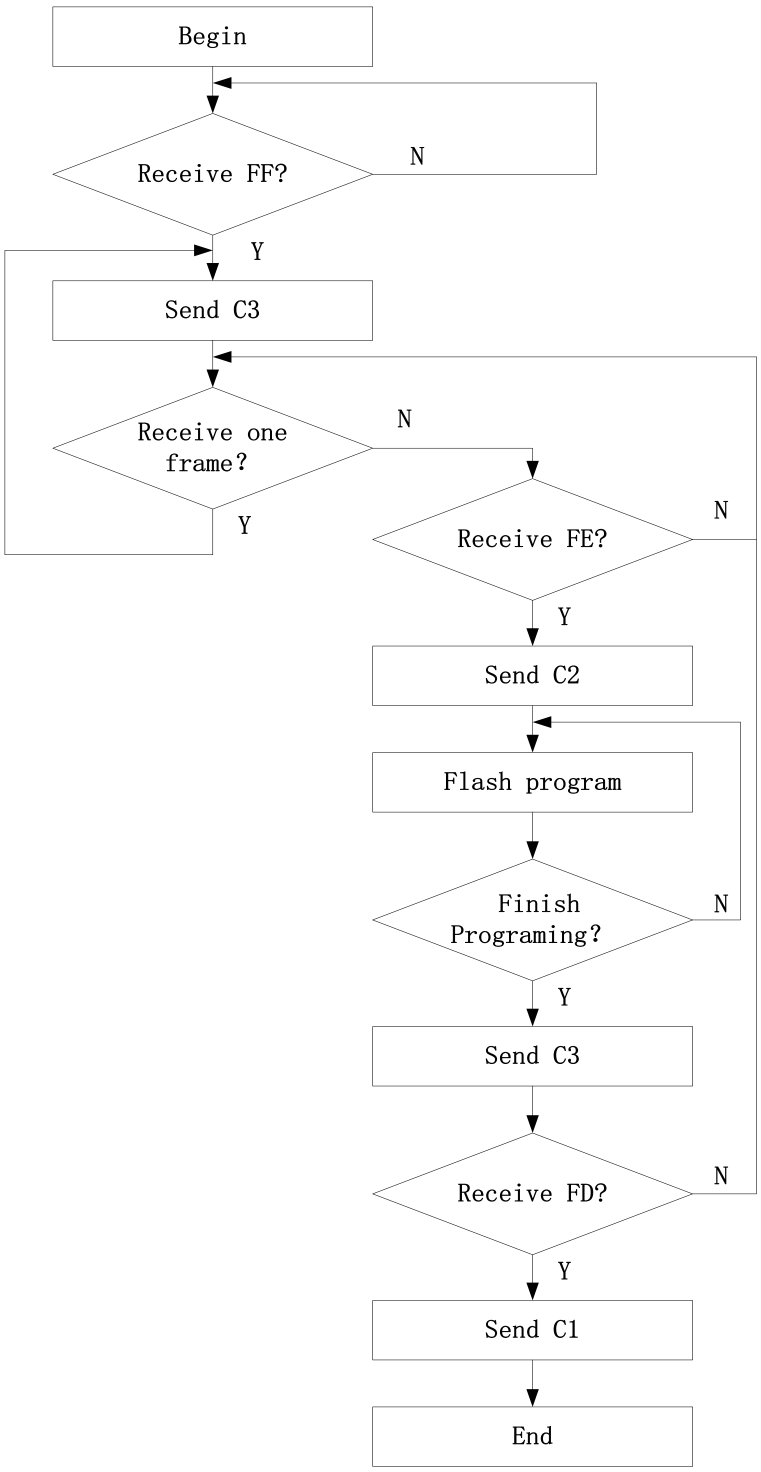

Figure 2. The flow chart of BLC protocol

FF: DOWN_LINK; FD: DOWN_LINE_END; FD: DOWN_FILE_END;

C1: UP_PRGEND; C2: UP_BUSY; C3: UP_READY

3.2 Flash operation Code running in flash

KEAZ128 contains a piece of 128 KB flash memory. This flash block is divided into 256 sectors of 512 bytes. The small size of sectors is benefit to erase and program in the flash without running the flash operation code in RAM.

KEAZ128 Includes Flash Memory Controller (FMC) feature. FMC is a memory acceleration unit that provides an interface between Cortex M0+ core and the 32-bit program flash memory. The FMC contains one 32-bit speculation buffer and one 32-byte cache that can accelerate instruction fetch and flash access. It includes the stalling flash controller, the prefetch buffer, the single entry buffer, and cache memory, so that code is able to program or erase the flash while running in flash. If the stalling flash controller feature is enabled when flash is busy, it can hold the access to flash until the flash is idle. In this case, it is not necessary to disable any interrupt. This makes code more efficient.

3.3 Keys for flash operation

KEAZ128 contains a piece of 128 KB flash memory. This flash block is divided into 256 sectors of 512 bytes. In the flash operation of boot loader base on sectors. The flow of flash operation is shown in figure 3.

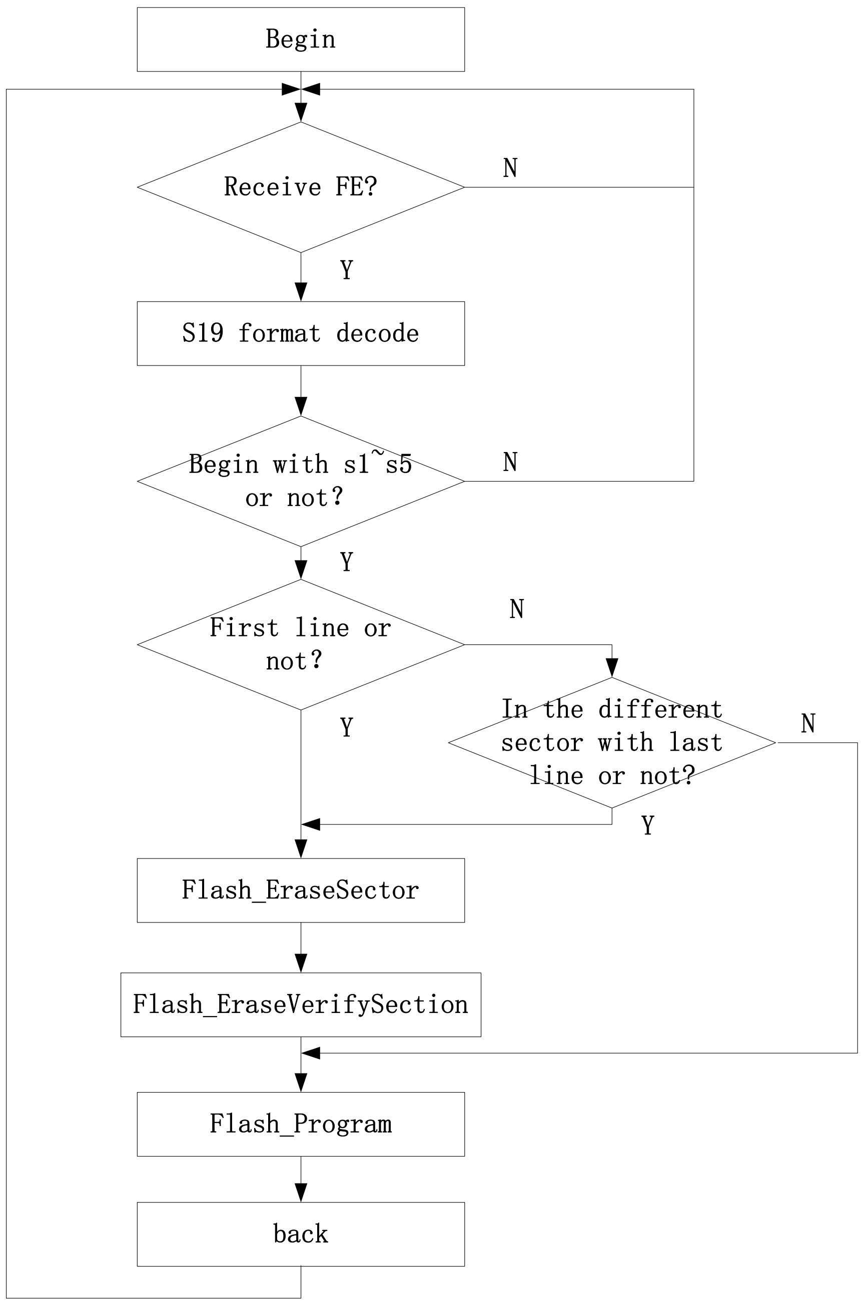

Figure 3. The flow of flash operation

3.4 Code protection in flash

The FPROT register can be set to protect regions in the flash memory from accidental programing or erasing. Three separate memory regions, one growing upward from global address 0x0000 in the flash memory, called the lower region; one growing downward from global address 0x7FFF in the flash memory, called the higher region; and the remaining addresses in the flash memory, can be activated for protection. The flash memory addresses covered by these protectable regions are shown in the flash memory map.

During the reset sequence, the FPROT register is loaded with the contents of the flash protection byte in the flash configuration field at global address 0x040D in flash memory. The protection functions depend on the configuration of bit settings in FPROT register.

Table1. The configuration of FPROT register

FPOPEN	FPHDIS	FPLDIS	Function
1	1	1	No flash protection
1	1	0	Protected low range
1	0	1	Protected high range
1	0	0	Protected high and low ranges
0	1	1	Full flash memory protected
0	1	0	Unprotected low range
0	0	1	Unprotected high range
0	0	0	Unprotected high and low ranges

For our boot loader is in the lower address range, we can refer to the table below.

Table2. The configuration of lower address

FPLS[1:0]	Global address range	Protected size
00	0x0000 – 0x07FF	2KB
01	0x0000 – 0x0FFF	4KB
10	0x0000 – 0x1FFF	8KB
11	0x0000 – 0x3FFF	16KB

The FPLS of boot loader set 11 to protect the flash from 0x0000 – 0x3FFF, before update the application code.

3.5 Interrupt vector table relocation

Interrupt vector table relocation for the boot loader design is how to handle the interrupt vector table. Kinetis EA series support interrupt vector table relocation. By default, the vector table is in 0x00 to 0xBF. The user can change the vector table to any other available address, such as another flash address or RAM. The boot loader application assigns the boot loader code and application code to different flash spaces. See the figure 1, we learn the memory allocation for the boot loader and application code. 0x00 to 0xBF are assigned the Interrupt vector of boot loader and interrupt vector of application relocate to 0x4000 to 0x40BF.

Before jumping to application, we need to relocate the interrupt vector table by simply writing the SCB_VTOR register with the following relocation address:

SCB_VTOR = RELOCATION_ADDRESS

When downloading user code with the boot loader, the software of PC will decode the S19 file (application code) and write the contents of the address space (0x00–0x00BF) to the relocation address (0x4000 to 0x40BF). This ensures that after reset, the boot loader will first start to run and jump to boot vector of the application whenever possible.

3.6 Software flow chart

After the power is on, the software will first run the boot loader to check whether a hook up will be successful. If overtime occurs, jump to the boot vector of application to initialize SP and write SCB_VTOR register with the user interrupt vector address. Then, run the application code. The software flow is shown in Figure 4.

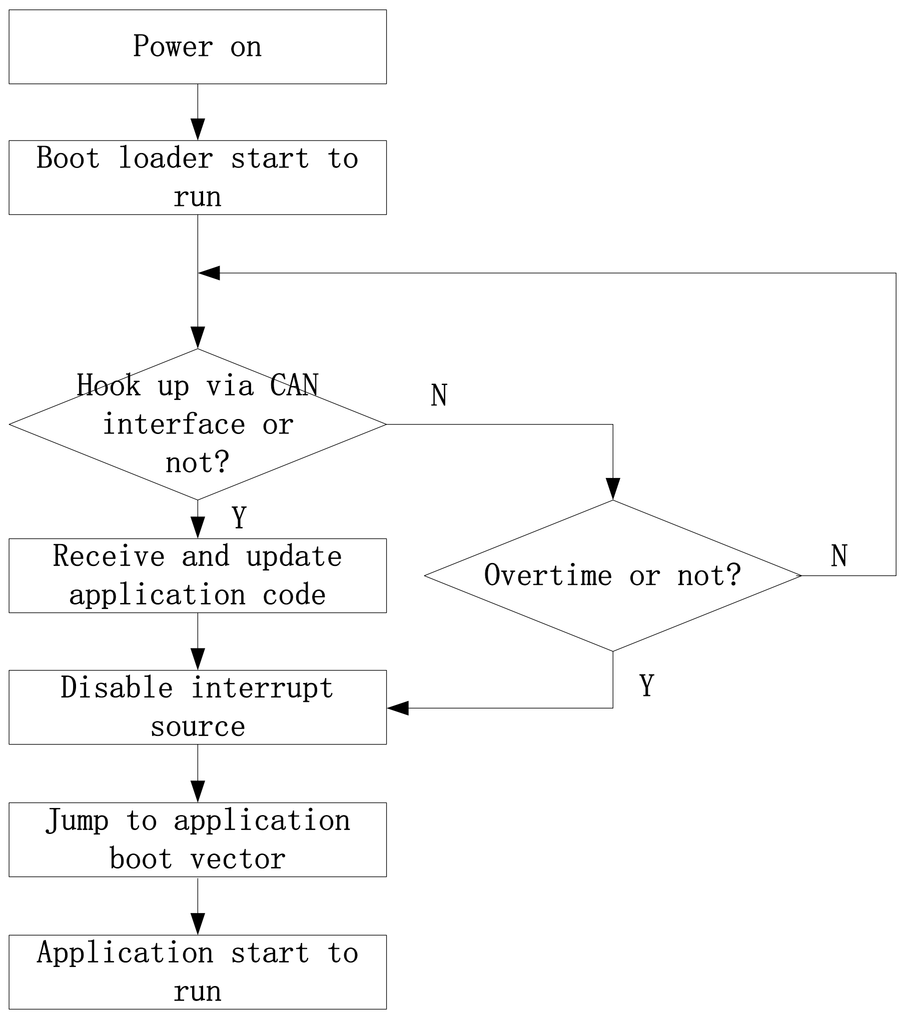

Figure 4. Software flow chart

4 User Application

For the relocation of vector table of application, we need to modify the application template to adapt to the boot loader. There are three user applications based on the KEAZ128. Two is with driver library and another built base on the default template. The first application is a lighting demo with RTC interrupt, the second one is a flash LED demo with cycle(without interrupt), and the last one is a FTM demo with RTC interrupt base on the default template of Codewarrior 10.6.

4.1 Link file

The normal project is available to define the code start address to any acceptable address. The memory is divided as below.

```
MEMORY
{
  m_interrupts     (rx) : ORIGIN = 0x00004000, LENGTH = 0xC0
  m_cfmprotrom     (rx) : ORIGIN = 0x00000400, LENGTH = 0x10
  m_text           (rx) : ORIGIN = 0x000040C0, LENGTH = 128K - 0x40C0
  m_data          (rwx) : ORIGIN = 0x1FFFF000, LENGTH = 16K
}
```

The application code will begin with 0x4000 which is the relocation vector address (RELOCATION_VERTOR_ADDR), so the boot vector should be RELOCATION_VERTOR_ADDR plus 0x0004.

4.2 Flash configuration region

The flash configuration region is located in 0x400 to 0x40F, which is in the boot loader region. If this section is protected, it cannot be modified by application code. Otherwise, there are errors of flash programming. The application code will first read out all of contents of the sector that contain a flash configuration field at the address 0x400 to 0x40F. Then, erase this sector, modify the buffer, and then write back to this sector.

In our app_1 and app_2 demo, the flash configuration can be modified in the vector.h file as below.

```
#ifdef USE_BOOTLOADER
#else
#define CONFIG_1          0xffffffff
#define CONFIG_2          0xffffffff
#define CONFIG_3          0xffffffff
#define CONFIG_4          0xfffeffff
#endif
```

Because of boot loader protected, we need to define USE_BOOTLOADER in our application code.

```
#define USE_BOOTLOADER
```

Our app_3 demo is based on the default template of Codewarrior 10.6. If building a new MCU project by Codewarrior 10.6 by default like app_3, the flash configuration is default, so USE_BOOTLOADER is not needed to be defined as mentioned above. Only need to cover the default SKEAZ128_flash.ld file of one of the three applications demos.

5 Conclusion

This document introduces a way of implementing the CAN boot loader of KEA by using PC side host software, DemoCAN. It introduces the key points of boot loader in KEA series which is M0+ core include BLC protocol, flash operation, Interrupt vector table relocation, Software flow, and so on. It is convenient for the user to update the application code in some special applications without other programming tools.