BOOTLOADER DESIGN FOR APPLICATION

Introduction

The purpose of project is to update application firmware (in-application-programming) of a microcontroller. Although various interfaces (USB, UART, SPI, Wireless) can be used to update an application, USB interface is used in this project. The flash memory of MCU is divided two parts one is bootloader and other one is application. The bootloader is designed at once and the firmware which is used by end-user, can be updated any time by manufacturer via bootloader.

The main duties of bootloader (it is just a program in flash memory) are checking flash memory, writing flash memory and erasing. The code which is converted to .hex file will be send to bootloader via USB (Xmodem protocol is used) and bootloader will perform firmware updates. In this project, Nucleo-STM32F070RB (Cortex M0) development card is being used.

Bootloader

To upgrade the application software with a new version, a bootloader should be developed.

A bootloader (it is just a software) makes an MCU a self-programmable device. Bootloader install the new application (if exist) and boots system. Since it is a program, both bootloader and application present in main flash memory. Figure 1 shows parts of flash memory which includes bootloader and application.

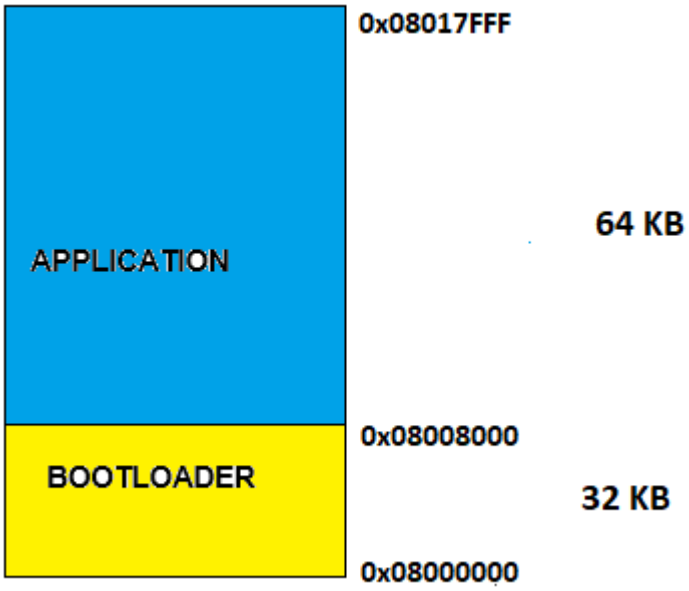


Figure 1: *Main Flash Memory includes two parts*

Bootloader has 32 KB size and application has 64 KB. The whole memory size is 128 KB so you can increase the size of bootloader or application.

HOW TO WORK

Firstly, bootloader offers two options to user to start update process. If user press the button more than 1 second bootloader will start updating application; if the user press button more than five seconds, bootloader launch the existing application and boot it.

After the update process, the bootloader jumps to the main application. To do this process a “jump” function should be written. This function adjusts the “**stack pointer**” and “**reset handler**” which states the address of start address of application. [2]

Another critic point is adjusting the vector table of bootloader and application. Each of them has own vector table. In this stage you should make **vector table relocation**. It can be made just adjusting **VTOR** register for Cortex M0+ processors, but I used Cortex M0 processor. Cortex M0 has no VTOR register and **I remap the vector table of application to SRAM**. The remap code is included in application project (code). The remap function which is present in application code is given below.

```
volatile uint32_t VectorTable[48] __attribute__((section(".RAMVectorTable")));
void remapMemToSRAM( void )
{
   uint32_t vecIndex = 0;
   __disable_irq();

   for(vecIndex = 0; vecIndex < 48; vecIndex++){
      VectorTable[vecIndex] = *(volatile uint32_t*)(APP_ADDRESS + (vecIndex << 2));
   }
                  __HAL_SYSCFG_REMAPMEMORY_SRAM();

   __enable_irq();
}
```

When upgrade process starts, bootloader erase flash memory and starts to write new application to flash memory. All functions are present in bootloader.c source file.

Also, you should write the starting address and size of your application and in KEIL.

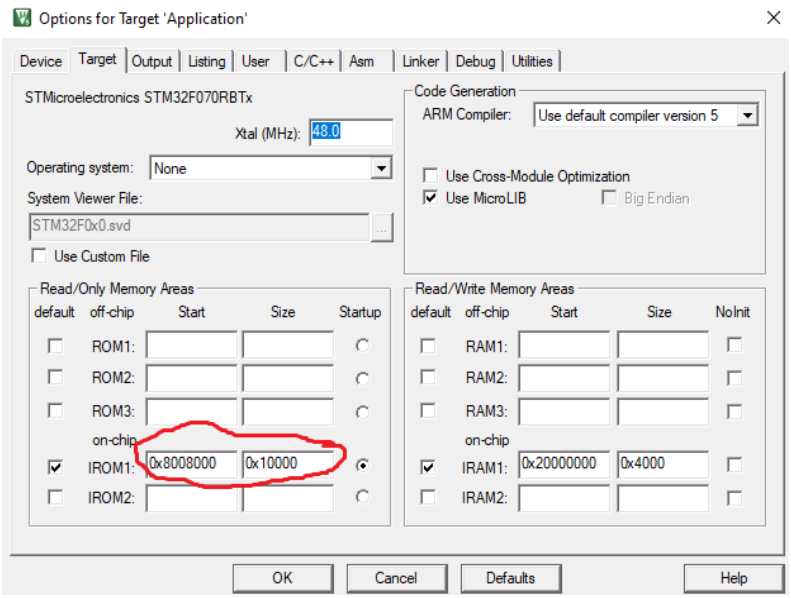


Figure 2: *Adjusting memory size for application project*

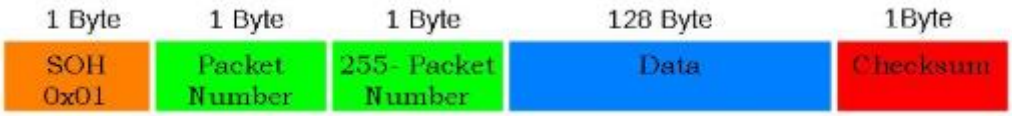
Also, you should do same adjustment in bootloader project. Therefore, the bootloader size is selected as 32 KB and application is 64KB.

XMODEM Protocol

The application is sent from PC over USB (It is worked USB-CDC, its functions are generated from CubeMx and they are present in usbd_cdc_if.c). The PC sends a binary file of application. Firstly, KEIL generates a hex file after that you should convert hex file to binary file.

Xmodem protocol helps to send binary file with 128-byte packets. When a packet is taken by MCU (bootloader program) it is written to flash memory. The detail of protocol is given in reference section.

There are kinds of Xmodem protocols, I use Xmodem checksum method because it is easy to implement as a C code. The figure given below shows a packet according to Xmodem protocol.



To send a file from a PC you should use a terminal program such as Putty or Tera term. The program that you use must support Xmodem protocol.

All code is given on GitHub account.

References

1.http://blog.atollic.com/how-to-develop-and-debug-bootloader-application-systems-on-arm-cortex-m-devices

2.https://www.st.com/content/ccc/resource/technical/document/reference_manual/cf/10/a8/c4/29/fb/4c/42/DM00091010.pdf/files/DM00091010.pdf/jcr:content/translations/en.DM00091010.pdf

3.http://info.atollic.com/hubfs/Whitepapers/Atollic_Develop_Debug_BootloaderApps_ARMCortex.pdf

4. The Definitive Guide to ARM Cortex -M0 and Cortex-M0+ Processors "Joseph Yiu"

5. http://web.mit.edu/6.115/www/amulet/xmodem.htm