**NXP Easy MCU boot loader**

This product is a simplified version of the NXP official MCUBOOT C implementation, removing most of the features, only retain the serial download function. It can be used with the official command line tool blhost or GUI tool: Kinetis Flash Tool to implement a bootloader based on MCU serial port. It is very easy to port to any MCU，and we recommend that the old-fashioned AN2295 be no longer used and migrated to MCUBOOT.

**Introduction**

* The underlying implementation is completely decoupled from the specific hardware. There are only two .c.h files: kptl.c mcuboot.c. No interrupts are used except for the Systick interrupt, and it will send and receive data using only serial port rotation.
* All hardware-related operations are all displayed by the callback function. Kptl and mcuboot do not have any hardware dependencies themselves. It is very easy to port to any ARM Cortex MCU. Under the premise of C language, kptl and mcuboot will be very easy to understand.
* Currently providing migration demos for LPC802 and FRDM-K64

For the complete MCUBOOT protocol, software and other content, please see our official webpage.

**Content**

[Figure]

**Example**

Project folder:

[Figure]

The following is an example of FRDM-K64:

1. Compile the frdm\_k64\_bl project and use JLINK to download it to the FRDM-K64 board.
2. Open KinetisFlashTool.exe. After the MCU resets, the connection is clicked within 300ms. After the connection is successful, the upper device displays the MCU information.
3. Compile frdm\_k64\_example\_app to get the frdm\_k64\_example\_app.hex file, select the hex file, click "UPDATE" to start the upgrade.

**Migration**

The bootloader itself has only 4 files: *kptl.c kptl.h mcuboot.c, mcuboot.h.*

Kptl is responsible for the basic implementation of the MCUBOOT protocol, unpacking packets etc.. mcuboot implements the basic commands of the bootloader and relying on kptl. Note that mcuboot has removed most of the official full version of the command implementation, leaving only the most basic commands for serial port downloads such as flashing the flash, obtaining chip information, and so on.

**Migrate Instruction**

1. Prepare your MCU: Make sure you are familiar with your MCU. The MCU can run normally, and the serial port transceiver function has been tuned with no problem. The Flash programming and erasing functions have been tuned with no problem. These steps are very important, the serial port transceiver and flash programming functions are the basic operations of the bootloader, and these features have bugs that will directly affect the bootloader project.
2. Add kptl.c and mcuboot.c to the project and add the corresponding include path.
3. Define the global structure in the project: *static mcuboot\_t mcuboot*;
4. After initializing the necessary hardware resources (serial and Flash must be initialized). Fill in the necessary parameters and callback interface for the mcuboot structure

The specific callback operations that need to be implemented are as follows

| **Callback interface function that needs to be implemented** | **Description** |
| --- | --- |
| Op\_send | Serial port to send data |
| Op\_reset | Reset MCU |
| Op\_jump | MCU jumps to the specified address |
| Op\_complete | After the bootloader receives the complete image, it will call back this function, which is generally used to release the hardware resources, and then close the peripheral with the interruption off. |
| Op\_mem\_erase | Flash erase |
| Op\_mem\_write | Flash programming |
| Op\_mem\_read | Flash reading |

Configuration:

| **Configuration to fill in** | **Description** |
| --- | --- |
| Cfg\_flash\_start | APP startup address (usually the default is 0x1000 (4K) or 0x8000 (32K)) |
| Cfg\_flash\_size | Flash size |
| Cfg\_flash\_sector\_size | Flash sector size |
| Cfg\_ram\_start | RAM starting address. This field lets the host computer get the information, and the transplant phase can be filled casually |
| Cfg\_ram\_size | RAM size. This field lets the host computer get the information, and the transplant phase can be filled casually |
| Cfg\_device\_id | The example is usually filled with a 0x12345678. This field lets the host computer get the information, and the transplant phase can be filled casually |
| Cfg\_uuid | MCU unique ID. This field lets the host computer get the information, and the transplant phase can be filled casually |

example:

[Figure]

1. In while1, the wheel training accepts the serial port data, calls *mcuboot\_recv* once receive a byte, and calls *mcuboot\_proc* in turn . At the same time, you can set a timeout function through the Systick interrupt. If you do not receive the serial port data after more than a few seconds, try to jump directly to the app. (example’s normal default is 300MS)

**Other Concern**

1. The implementation of Flash is very important. Generally, flash is a block device with minimal erase and programming units (especially LPC, minimum erase unit and minimum programming unit are different and large). This requires special attention when implementing the function *memory\_write*, make sure it is aligned and complete. the minimum programming unit needs to extract the data before the readout and merging, and the flash is required to be written multiple times beyond the minimum programming unit.
2. To get the realization of an MCU reset, directly call the CMSIS library function *NVIC\_SystemReset*.
3. Before the last jump, it is generally necessary to de-initialize all the peripherals. The interrupts (including SYsTick) must be turned off, and these operations can be completed in the middle of *mcuboot\_complete*.
4. The last jump to the user app. There are three main things to do: Reset the PC, SP, and reset the entry address of the interrupt vector table: SCB->VTOR. Where SP is 0-3 bytes of Image and PC is 4-7 bytes of image.

The ancestor function can be used directly:

[Figure]

1. For APP project, remember to modify the startup address