AN12689

Porting USB Bootloader from LPC54018 to LPC51U68

Rev. 0 — 02/2020

Application Note

by: NXP Semiconductors

Contents

1 Introduction

There are several means for firmware upgrade, including the bootloader, hardware debugger (such as CMSIS-DAP, J-Link, and ULink), and third-party programmer (such as Flash Magic). Efficient and fast flash programming in the production flow is an important part of cost-effective manufacturing and field servicing of MCUs. From the firmware upgrade tools mentioned above, the hardware debugger is suitable for product development. The third-party programmer and bootloader are suitable for mass production. For NXP MCUs, Flash Magic only supports SWD, Ethernet, and UART (except for the USB). The secondary bootloader in the NXP MCU bootloader provides multiple serial protocols (such as USB HID/MSC, UART, SPI, I2C, and CAN) for quick and easy programming through the entire product lifecycle. The USB interface is widely used due to high data transmission rate and driver support by various mainstream operating systems.

2 Why to port USB bootloader

As described on the official NXP website, a secondary bootloader is supported in the SDK for i.MXRT1050, LPC54018, LPC54016, and LPC54005. Therefore, the LPC51U68 USB bootloader can be ported from the SDK of the mentioned MCUs. This application note selects LPC54018 as a reference and describes how to port the USB bootloader from LPC54018 to LPC51U68.

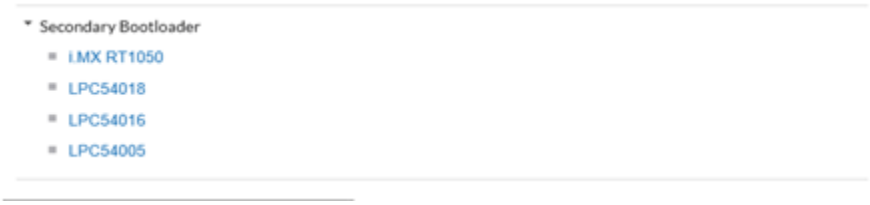

Figure 1. MCUs supported by secondary bootloader

3 System hardware setup and connection

3.1 System hardware setup

LPCXpresso51U68 version Rev A is used as the evaluation board for USB bootloader porting. Make sure that jumper JP10 is fitted, so that JP10 connects the VBUS from the target USB connector J5 to the LPC51U68. The other jumpers (configured by default) are described in Table 1 and Figure 2.

Table 1. Board jumper setting

Jumper number	Description	Status
J1, J2, J7, J8	Expansion connectors	Open
J3	External processor control header	Open
J4	PMod™ (SPI / I2C) bridge connector	Open
J5	Target MCU power / USB device connector	Connect to PC via USB cable for firmware update
J6	Link2 micro USB B-type connector	Connect to PC via USB cable for bootloader preprogramming
JP1	LPC51U68 target SWD disable	Open
JP2	Buffer power selection	Fit to Loc
JP3	Isolate the Link2 debug probe (SPI bridge function) from the LPC51U68 target to prevent current leakage.	Open
JP4	Disconnect all reset sources from the LPC51U68 device, except for the reset from the AP control port (J3).	Open
JP5	Reduce the voltage sense resistance.	Open
JP6	Measure the LPC51U68 current consumption.	Open
JP7	Link2 (LPC43xx) force DFU boot	Open
JP8	Tri-color LED anode voltage enable	Open
JP9	Power supply voltage selection	Fit pin 2–3 to select 3.3 V power supply
JP10	Target connector USB (J5) VBUS to LPC51U68 connection	Fit
P1	10-pin SWD connector	Open
P2	LPC51U68 VDD current monitor Vsense measurement	Open
P3	FTDI serial header	Open
P4	External ADC reference input	Open

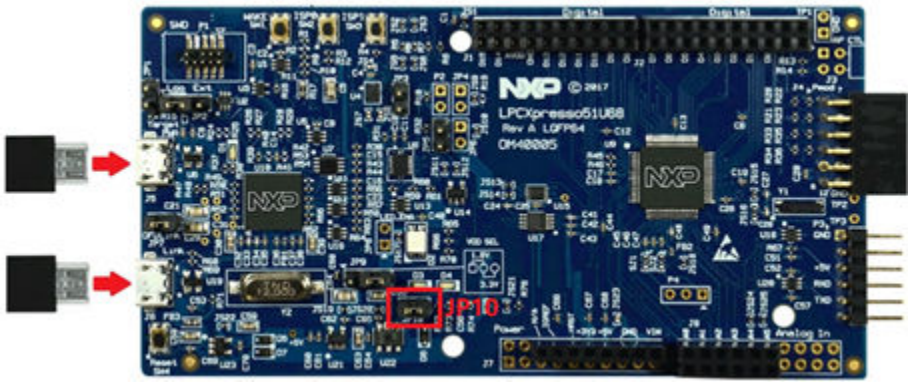

Figure 2. Board jumper setting

3.2 Connection

The connection shown in Figure 3 is a must for normal communication between the LPCXpresso51U68 board and PC. The USB cable connected to connector J6 is used to download the bootloader and the cable connected to connector J5 is used to update the user application firmware.

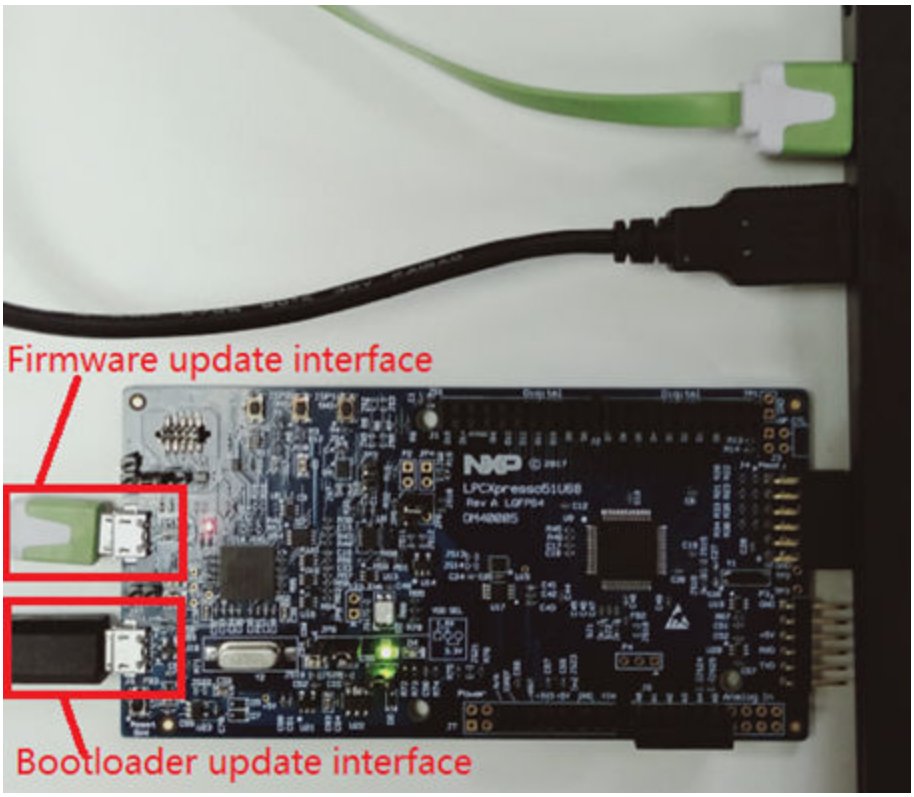

Figure 3. USB connection between LPC51U68 and PC

4 USB bootloader porting from LPC54018 to LPC51U68

4.1 Flash-resident USB bootloader porting

4.1.1 Creating new project and adding necessary files

- Download SDK_2.6.0_LPCXpresso51U68 from the official NXP website. This application note uses the Keil IDE to perform USB bootloader porting. Therefore, create a new empty project (as shown in Figure 4) by clicking "Project -> New uVision Project" and selecting the project directory (as shown in Figure 5), assuming that the project directory is *\boards*.

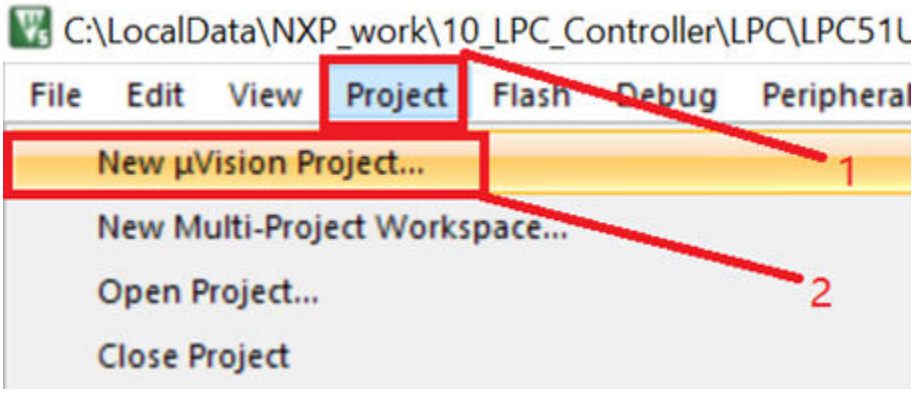

Figure 4. Creating new Keil project

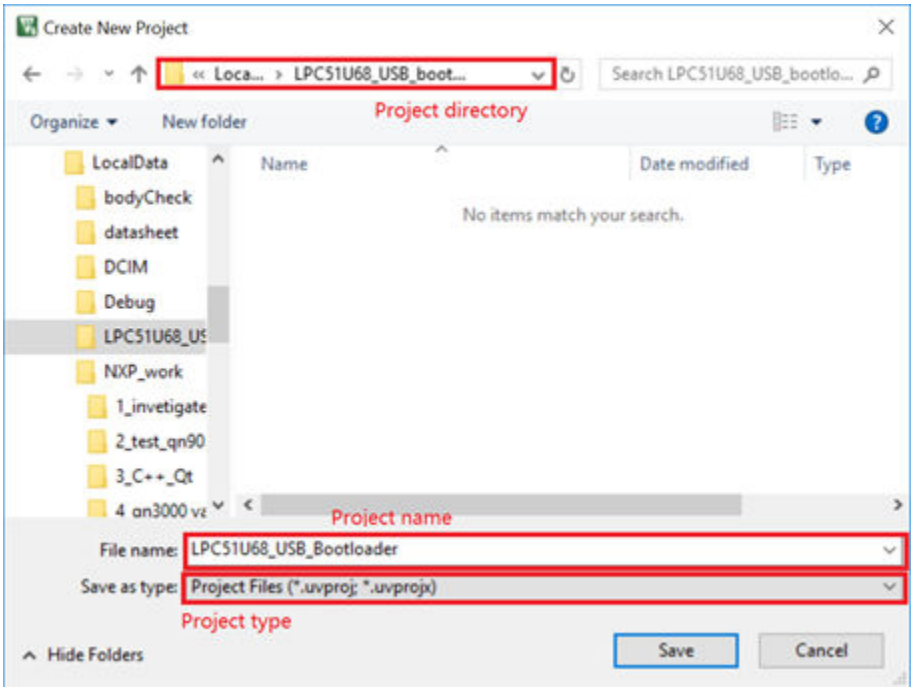

Figure 5. Selecting project directory and names

- Download SDK_2.6.0_LPCXpresso54018 from the official NXP website.
- Copy the *mcu-boot* folder from the *SDK_2.6.0_LPCXpresso54018\middleware* folder to the *SDK_2.6.0_LPCXpresso51U68\middleware* folder.
- Modify the "LPC54018" string in the *external_memory_property_map_LPC54018.c*, *hardware_init_LPC54018.c*, *memory_map_LPC54018.c*, and *peripherals_LPC54018.c* files in the *\middleware\mcu-boot\targets\LPC51U68\src* folder to "LPC51U68".
- Create a source file named *internalFlashAPI.c* (other names are also OK) and a header file named *internalFlashAPI.h* (other names are also OK) for the LPC51U68 on-chip flash to provide operations such as erase, read, write, and execute in the *\middleware\mcu-boot\src\memory\src* folder.

- Copy the *pin_mux.c*, *pin_mux.h*, *clock_config.c*, and *clock_config.h* files from the *\boards\lpcxpresso51u68\usb_examples\usb_device_cdc_vcom\freertos* folder to the *\boards* folder.
- Copy the files shown in Table 2 from the *\boards\lpcxpresso51u68\usb_examples\usb_device_msc_ramdisk\freertos* folder to the *\boards* folder.

Table 2. USB device files

Number	File
1	usb_device_ch9.c
2	usb_device_ch9.h
3	usb_device_class.c
4	usb_device_class.h
5	usb_device_config.h
6	usb_device_descriptor.c
7	usb_device_descriptor.h
8	usb_device_msc.c
9	usb_device_msc.h
10	usb_device_msc_ufi.c
11	usb_device_msc_ufi.h

- Copy the *usb_device_hid.c* and *usb_device_hid.h* files from the *\boards\lpcxpresso51u68\usb_examples\usb_device_hid_generic\freertos* folder to the *\boards* folder.
- Add the groups and files shown in Table 3 to the project created in the first step.

Table 3. Groups and files

Group	File
device	\devices\LPC51U68\fsl_device_registers.h \devices\LPC51U68\LPC51U68.h \devices\LPC51U68\LPC51U68_features.h \devices\LPC51U68\system_LPC51U68.c \devices\LPC51U68\system_LPC51U68.h
startup	\devices\LPC51U68\arm\startup_LPC51U68.s
source-usb-bm_composite	\middleware\mcuboot\src\bm_usb \bootloader_hid_report_ids.h \middleware\mcu-boot\src\bm_usb\composite.c \middleware\mcu-boot\src\bm_usb\composite.h \middleware\mcu-boot\src\bm_usb\fat_directory_entry.h \middleware\mcu-boot\src\bm_usb\hid_bootloader.c \middleware\mcu-boot\src\bm_usb\hid_bootloader.h \middleware\mcu-boot\src\bm_usb\msc_disk.c

Table continues on the next page...

Table 3. Groups and files (continued)

Group	File
	\middleware\mcu-boot\src\bm_usb\msc_disk.h \middleware\mcu-boot\src\bm_usb\usb_descriptor.c \middleware\mcu-boot\src\bm_usb\usb_descriptor.h \middleware\mcu-boot\targets\LPC51U68\src \usb_device_config.h
source-autobaud	\middleware\mcu-boot\src\autobaud\autobaud.h \middleware\mcu-boot\src\autobaud\src\autobaud_irq.c
source-bootloader	\middleware\mcu-boot\src\bootloader\bl_app_crc_check.h \middleware\mcu-boot\src\bootloader\bl_command.h \middleware\mcu-boot\src\bootloader\bl_context.h \middleware\mcu-boot\src\bootloader\bl_irq_common.h \middleware\mcu-boot\src\bootloader\bl_peripheral.h \middleware\mcu-boot\src\bootloader \bl_peripheral_interface.h \middleware\mcu-boot\src\bootloader\bl_shutdown_cleanup.h \middleware\mcu-boot\src\bootloader\bl_user_entry.h \middleware\mcu-boot\src\bootloader\bl_version.h \middleware\mcu-boot\src\bootloader\bootloader.h
source-bootloader-src	\middleware\mcu-boot\src\bootloader\src\bl_app_crc_check.c \middleware\mcu-boot\src\bootloader\src\bl_command.c \middleware\mcu-boot\src\bootloader\src\bl_context.c \middleware\mcu-boot\src\bootloader\src \bl_exception_handler.c \middleware\mcu-boot\src\bootloader\src\bl_main.c \middleware\mcu-boot\src\bootloader\src\bl_misc.c \middleware\mcu-boot\src\bootloader\src \bl_shutdown_cleanup.c \middleware\mcu-boot\src\bootloader\src\bl_tree_root.c \middleware\mcu-boot\src\bootloader\src\bl_user_entry.c \middleware\mcu-boot\src\bootloader\src \usb_hid_msc_peripheral_interface.c
source-crc	\middleware\mcu-boot\src\crc\crc16.h \middleware\mcu-boot\src\crc\crc32.h \middleware\mcu-boot\src\crc\src\crc16.c

Table continues on the next page...

Table 3. Groups and files (continued)

Group	File
	\middleware\mcu-boot\src\crc\src\crc32.c
source-include	\middleware\mcu-boot\src\include\bootloader_common.h \middleware\mcu-boot\src\include\bootloader_core.h
source-memory	\middleware\mcu-boot\src\memory\memory.h
source-memory-src	\middleware\mcu-boot\src\memory\src\device_memory.c \middleware\mcu-boot\src\memory\src\device_memory.h \middleware\mcu-boot\src\memory\src\memory.c \middleware\mcu-boot\src\memory\src\normal_memory.c \middleware\mcu-boot\src\memory\src\normal_memory.h \middleware\mcu-boot\src\memory\src\pattern_fill.h \middleware\mcu-boot\src\memory\src\pattern_fill.s \middleware\mcu-boot\src\memory\src\sram_init.h \middleware\mcu-boot\src\memory\src\sram_init_lpc.c \middleware\mcu-boot\src\memory\src\internalFlashAPI.c \middleware\mcu-boot\src\memory\src\internalFlashAPI.h
source-packet	\middleware\mcu-boot\src\packet\command_packet.h \middleware\mcu-boot\src\packet\serial_packet.h
source-packet-src	\middleware\mcu-boot\src\packet\src\serial_packet.c
source-property	\middleware\mcu-boot\src\property\property.h
source-sbloader	\middleware\mcu-boot\src\sbloader\sb_file_format.h \middleware\mcu-boot\src\sbloader\sbloader.h
source-sbloader-src	\middleware\mcu-boot\src\sbloader\src\sbloader.c
source-utilities	\middleware\mcu-boot\src\utilities\fsl_assert.h \middleware\mcu-boot\src\utilities\fsl_rtos_abstraction.h \middleware\mcu-boot\src\utilities\vector_table_info.h
source-utilities-src	\middleware\mcu-boot\src\utilities\src\fsl_assert.c \middleware\mcu-boot\src\utilities\src\fsl_rtos_abstraction.c
source-drivers	\middleware\mcu-boot\src\drivers\smc\smc.h
source-property-src	\middleware\mcu-boot\src\property\src\property_lpc.c
LPC51U68	\middleware\mcu-boot\targets\LPC51U68\src \bootloader_config.h \boards\clock_config.c

Table continues on the next page...

Table 3. Groups and files (continued)

Group	File
	\middleware\mcu-boot\targets\LPC51U68\src \external_memory_property_map_LPC51U68.c \middleware\mcu-boot\targets\LPC51U68\src \hardware_init_LPC51U68.c \middleware\mcu-boot\targets\LPC51U68\src \memory_map_LPC51U68.c \middleware\mcu-boot\targets\LPC51U68\src \peripherals_LPC51U68.c \middleware\mcu-boot\targets\LPC51U68\src \peripherals_pinmux.h \middleware\mcu-boot\targets\LPC51U68\src\target_config.h \middleware\mcu-boot\targets\common\src \pinmux_utility_lpc.c \boards\pin_mux.c \boards\pin_mux.h
usb-device-class-hid	\boards\usb_device_hid.c \boards\usb_device_hid.h
usb-device-class-msc	\boards\usb_device_msc.c \boards\usb_device_msc.h \boards\usb_device_msc_ufi.c \boards\usb_device_msc_ufi.h
usb-device-source	\boards\usb_device_ch9.c \boards\usb_device_ch9.h \middleware\usb\device\usb_device_dci.c \middleware\usb\device\usb_device_dci.h
usb-device-class	\boards\usb_device_class.c \boards\usb_device_class.h
usb-device-source-lpcip3511	\middleware\usb\device\usb_device_lpcip3511.c \middleware\usb\device\usb_device_lpcip3511.h
usb-include	\middleware\usb\include\usb.h \middleware\usb\include\usb_misc.h \middleware\usb\include\usb_spec.h
osa	\middleware\usb\osa\usb_osa.h \middleware\usb\osa\usb_osa_bm.c

Table continues on the next page...

Table 3. Groups and files (continued)

Group	File
	\middleware\usb\osa\usb_osa_bm.h
usb-device-include	\middleware\usb\device\usb_device.h
drivers	devices\LPC51U68\drivers\fsl_clock.c devices\LPC51U68\drivers\fsl_clock.h devices\LPC51U68\drivers\fsl_common.c devices\LPC51U68\drivers\fsl_common.h devices\LPC51U68\drivers\fsl_crc.c devices\LPC51U68\drivers\fsl_crc.h devices\LPC51U68\drivers\fsl_inputmux.c devices\LPC51U68\drivers\fsl_inputmux.h devices\LPC51U68\drivers\fsl_inputmux_connections.h devices\LPC51U68\drivers\fsl_iocon.h devices\LPC51U68\drivers\fsl_power.c devices\LPC51U68\drivers\fsl_power.h devices\LPC51U68\drivers\fsl_reset.c devices\LPC51U68\drivers\fsl_reset.h devices\LPC51U68\drivers\fsl_iap.c devices\LPC51U68\drivers\fsl_iap.h devices\LPC51U68\drivers\fsl_gpio.c devices\LPC51U68\drivers\fsl_gpio.h
source-drivers-microseconds	\middleware\mcu-boot\src\drivers\microseconds \microseconds.h \middleware\mcu-boot\src\drivers\microseconds\src \microseconds_sysclk.c

4.1.2 Configuring Keil IDE

4.1.2.1 Selecting MCU part number

Click the "Device" tab and select the MCU part number, as shown in Figure 6.

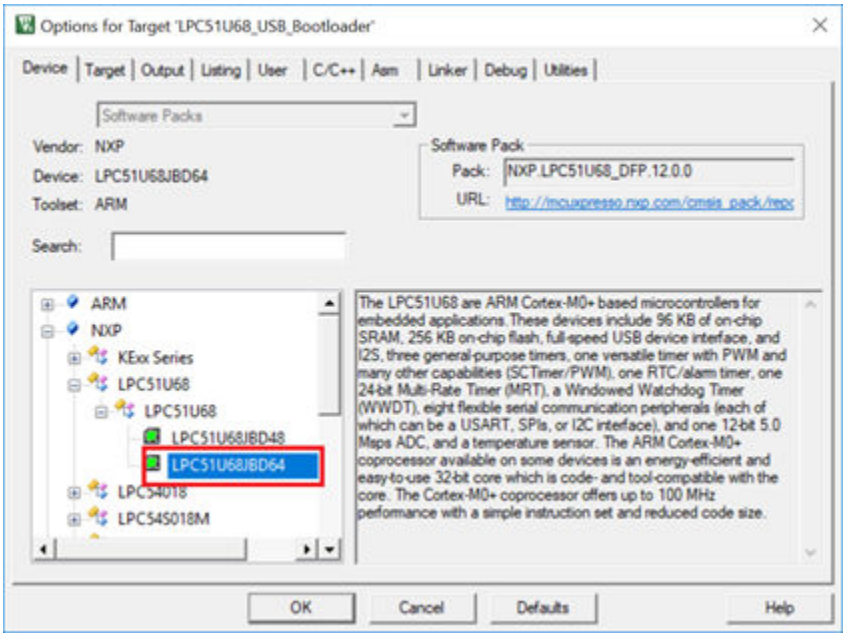

Figure 6. Selecting MCU part number

4.1.2.2 Generating binary executable file

Click the "User" tab and carry out the configuration (as shown in Figure 7) to generate a binary executable file when the project is compiled.

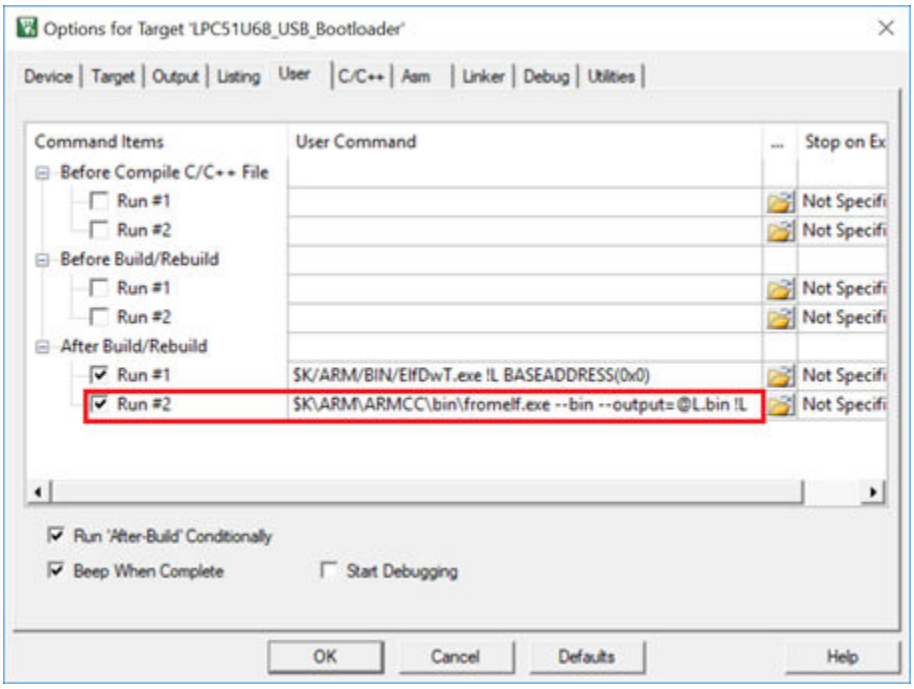

Figure 7. Binary executable file generation setting

4.1.2.3 Seting preprocessor symbols and header file path

Click the "C/C++" tab and use the "_DEBUG=1,DEBUG, CPU_LPC51U68JBD64, USB_STACK_BM, USB_STACK_USE_DEDICATED_RAM=1, BL_TARGET_FLASH" string as the preprocessor symbols (as shown in Figure 8).

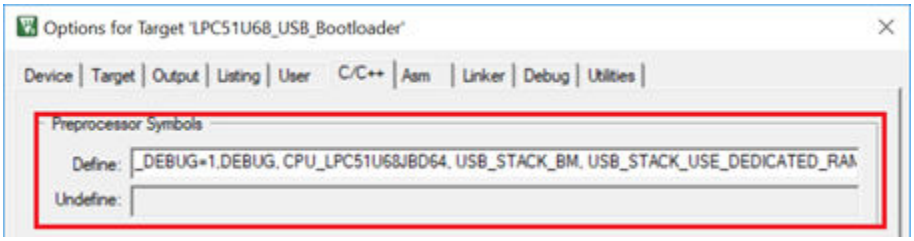

Figure 8. Preprocessor symbols setting

Click the "C/C++" tab and add the header file search path. Click the "Include Paths" button (as shown in Figure 9) and enter the "Folder Setup" interface to add the header file search path (as shown in Figure 10). Add the header file directories according to Table 4.

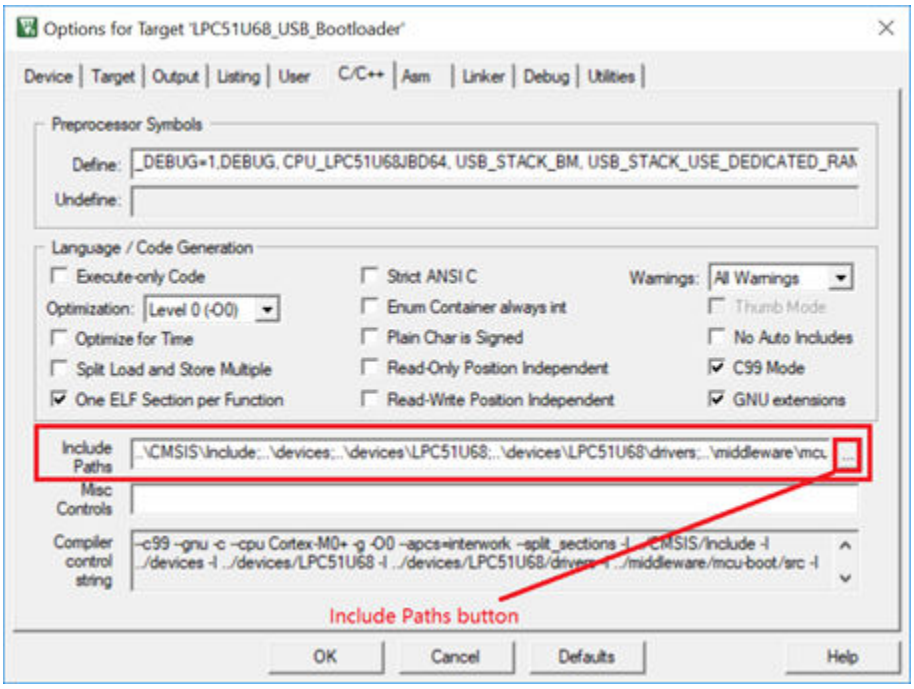

Figure 9. Enter folder setup interface

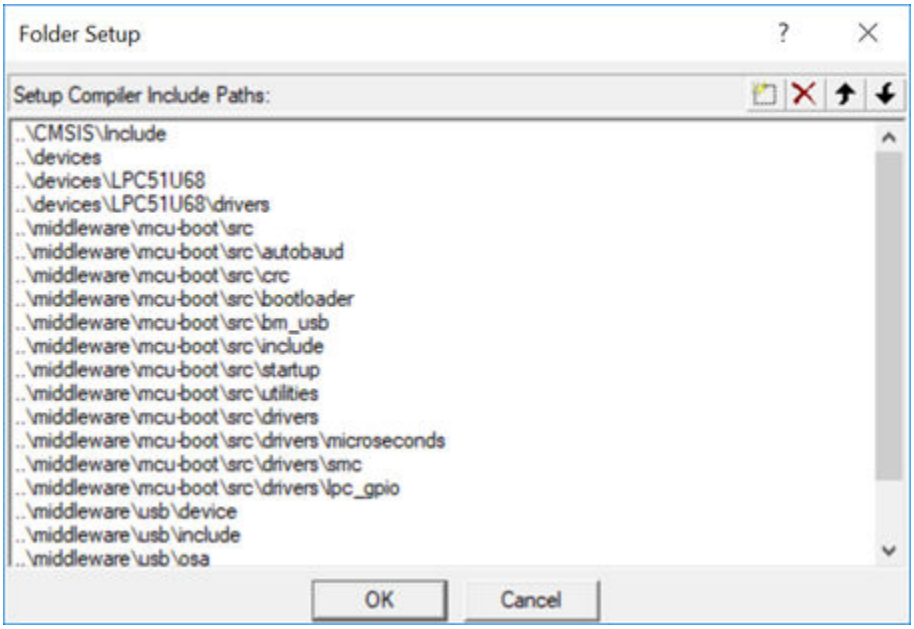

Figure 10. Folder setup interface

Table 4. Header file directories

Number	Header file directory
1	..\CMSIS\Include
2	..\boards
3	..\devices
4	..\devices\LPC51U68
5	..\devices\LPC51U68\drivers
6	..\middleware\mcu-boot\src
7	..\middleware\mcu-boot\src\autobaud
8	..\middleware\mcu-boot\src\crc
9	..\middleware\mcu-boot\src\bootloader
10	..\middleware\mcu-boot\src\bm_usb
11	..\middleware\mcu-boot\src\include
12	..\middleware\mcu-boot\src\startup
13	..\middleware\mcu-boot\src\utilities
14	..\middleware\mcu-boot\src\drivers
15	..\middleware\mcu-boot\src\drivers\microseconds
16	..\middleware\mcu-boot\src\drivers\smc
17	..\middleware\mcu-boot\src\drivers\lpc_gpio
18	..\middleware\usb\device

Table continues on the next page...

Table 4. Header file directories (continued)

Number	Header file directory
19	..\middleware\usb\include
20	..\middleware\usb\osa
21	..\middleware\mcu-boot\targets\LPC51U68\src

4.1.2.4 Assembly setting

Click the "Asm" tab and perform the assembly language settings (as shown in Figure 11).

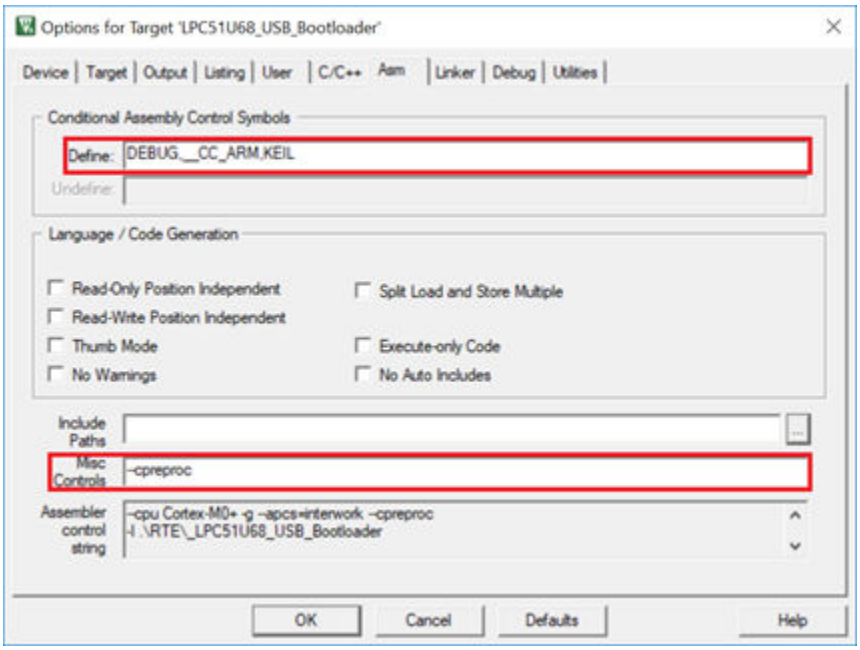

Figure 11. Assembly language setting

4.1.2.5 Linker setting

Uncheck the "Use Memory Layout from Target Dialog" checkbox to use the scatter file to assign the memory area for the binary executable file and carry out the miscellaneous control settings shown in Figure 12.

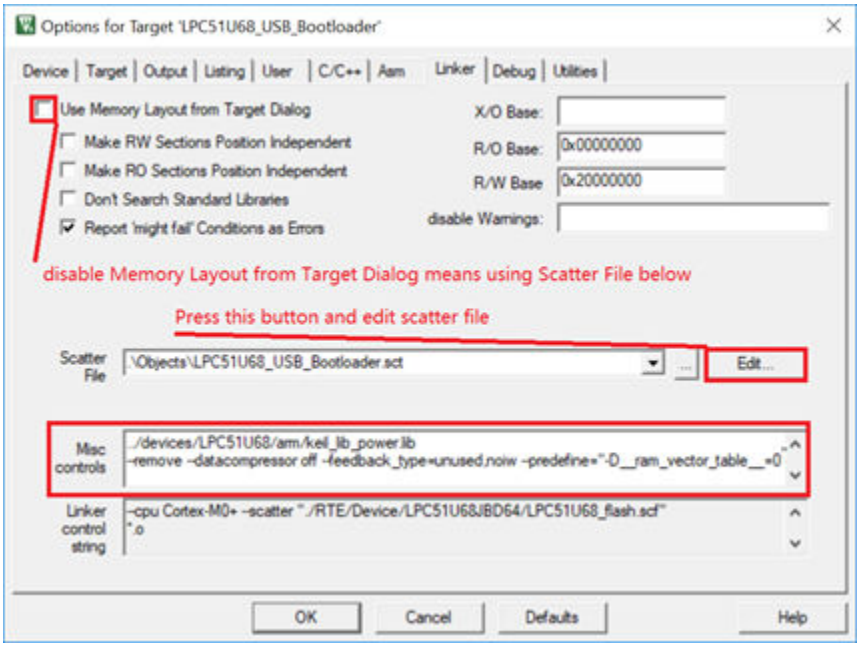

Figure 12. Linker setting

Click the "Edit" button and edit the scatter file shown in Figure 14. The memory setting in the scatter file is derived from the memory map shown in Figure 13.

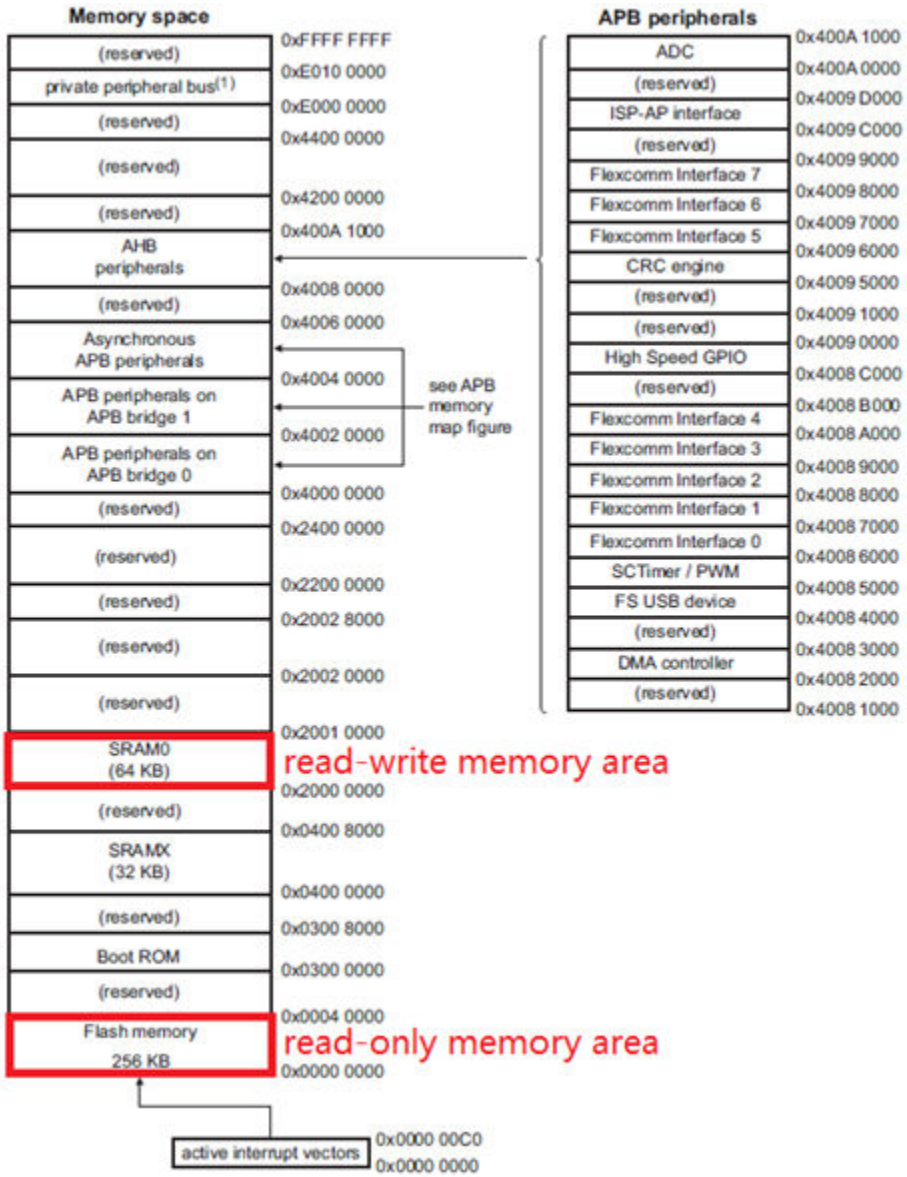

Figure 13. LPC51U68 memory map

```
#! armcc -E

#if (defined(__ram_vector_table__))
  #define __ram_vector_table_size__     0x00000400
#else
  #define __ram_vector_table_size__     0x00000000
#endif

#define m_interrupts_start              0x00000000
#define m_interrupts_size               0x00000400

#define m_text_start                    0x00000400
#define m_text_size                     0x0003FC00

#define m_interrupts_ram_start          0x20000000
#define m_interrupts_ram_size           __ram_vector_table_size__

#define m_data_start                    (m_interrupts_ram_start + m_interrupts_ram_size)
#define m_data_size                     (0x00010000 - m_interrupts_ram_size)

/* Sizes */
#if (defined(__stack_size__))
  #define Stack_Size                    __stack_size__
#else
  #define Stack_Size                    0x0400
#endif

#if (defined(__heap_size__))
  #define Heap_Size                     __heap_size__
#else
  #define Heap_Size                     0
LR_m_text m_interrupts_start m_text_start+m_text_size-m_interrupts_start
{;load region size_region
  VECTOR_ROM m_interrupts_start m_interrupts_size
  {;load address = execution address
    * (RESET,+FIRST)
  }

  ER_m_text m_text_start FIXED m_text_size
  {;load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
  }
 #if (defined(__ram_vector_table__))
   VECTOR_RAM m_interrupts_ram_start EMPTY m_interrupts_ram_size
   {
   }
 #else
   VECTOR_RAM m_interrupts_start EMPTY 0
   {
   }
 #endif

   RW_m_data m_data_start m_data_size-Stack_Size-Heap_Size
   {;RW data
     .ANY (+RW +ZI)
   }

   ARM_LIB_HEAP +0 EMPTY Heap_Size
   {;Heap region growing up
   }

   ARM_LIB_STACK m_data_start+m_data_size EMPTY -Stack_Size
   {;Stack region growing down
   }
 }
```

Figure 14. Scatter file

4.1.2.6 Debug setting

Click the "Debug" tab and select the simulator according to the actual situation. This application note uses the on-board J-Link debugger of the LPCXpresso51U68 board. Check the "Load Application at Startup" and "Run to main()" checkboxes (as shown in Figure 15).

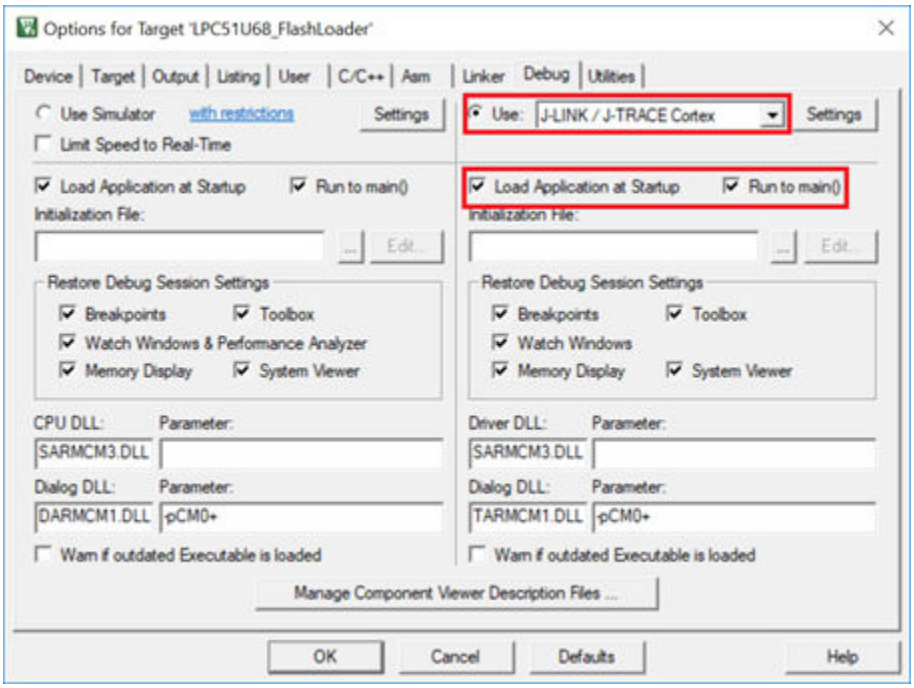

Figure 15. Debug setting

4.1.3 Compiling and fixing errors

Errors are generated when compiling the project. To fix these errors, make these modifications:

- Open *\middleware\mcu-boot\targets\LPC51U68\src\bootloader_config.h* and remove this macro:

```
#define FSL_FEATURE_SYSCON_FLASH_PAGE_SIZE_BYTES (0)
```

- Open *\middleware\mcu-boot\src\crc\src\crc16.c* and modify the header file include command from `#include "lpc_crc/fsl_crc.h"` to `#include "fsl_crc.h"`.
- Open *\middleware\mcu-boot\src\crc\src\crc32.c* and modify the header file include command from `#include "lpc_crc/fsl_crc.h"` to `#include "fsl_crc.h"`.
- Open *\middleware\mcu-boot\src\crc\src\crc16.c* and modify the "crc16_onfi_update" function. For the specific function definition, see the project code that matches the documentation.
- Open *\middleware\mcu-boot\targets\LPC51U68\src\hardware_init_LPC51U68.c* and modify the "init_hardware" function. For the specific function definition, see the project code that matches the documentation.
- Open *\middleware\mcu-boot\targets\LPC51U68\src\hardware_init_LPC51U68.c* and include the *pin_mux.h* header file:

```
#include “pin_mux.h”
```

- Open *\middleware\mcu-boot\targets\LPC51U68\src\hardware_init_LPC51U68.c* and remove the "spifi_clock_gate", "spifi_source_clock", and "spifi_iomux_config" functions.
- Open *\middleware\mcu-boot\targets\LPC51U68\src\hardware_init_LPC51U68.c* and modify the "usb_clock_init" function. For the specific function definition, see the project code that matches the documentation.

- Open *\middleware\mcu-boot\targets\LPC51U68\src\bootloader_config.h* and modify the "BL_CONFIG_USB_HID", "BL_CONFIG_HS_USB_HID", BL_CONFIG_FLEXCOMM_USART_0", "BL_CONFIG_FLEXCOMM_I2C_2", "BL_CONFIG_FLEXCOMM_SPI_9", and "BL_FEATURE_SPIFI_NOR_MODULE" macros:

```
#define BL_CONFIG_USB_HID (1)
#define BL_CONFIG_HS_USB_HID (0)
#define BL_FEATURE_SPIFI_NOR_MODULE (0)
#define BL_CONFIG_FLEXCOMM_USART_0 (0)
#define BL_CONFIG_FLEXCOMM_I2C_2 (0)
#define BL_CONFIG_FLEXCOMM_SPI_9 (0)
```

- Open *\boards\clock_config.c* and add the definition of the "configure_clocks" function. For the specific function definition, see the project code that matches the documentation.
- Open *\boards\clock_config.c* and include three header files and one macro definition:

```
#include "bootloader_common.h"
#include "property/property.h"
#include "bootloader/bl_context.h"
#define BOOTLOADER_CLOCK_FREQ 48000000U
```

- Open *\middleware\mcu-boot\src\bootloader\src\bl_shutdown_cleanup.c* and add the definition of the "init_interrupts" function. For the specific function definition, see the project code that matches the documentation.

4.1.4 Testing USB communication between LPCXpresso51U68 and PC

To test the USB communication between the LPCXpresso51U68 board and PC, follow these steps:

- Compile the project.
- Fit the JP10 jumper on the LPCXpresso51U68 board to connect the USB (J5) VBUS to the LPC51U68.
- Use a USB cable to connect the J6 header to the PC and download the project executable binary file to the LPC51U68 flash area.
- Use a USB cable to connect the J5 header to the PC.
- Power up and reset the LPCXpresso51U68 board.
- Run *\middleware\mcu-boot\bin\Tools\blhost\win\blhost.exe* and enter the blhost command (as shown in Figure 16). If the following command feedback appears, it indicates that the USB bootloader porting is implemented successfully. The *MCU Bootloader host (blhost) User's Guide* (document MCUBLHOSTUG) describes the detailed usage of the blhost commands mentioned below.

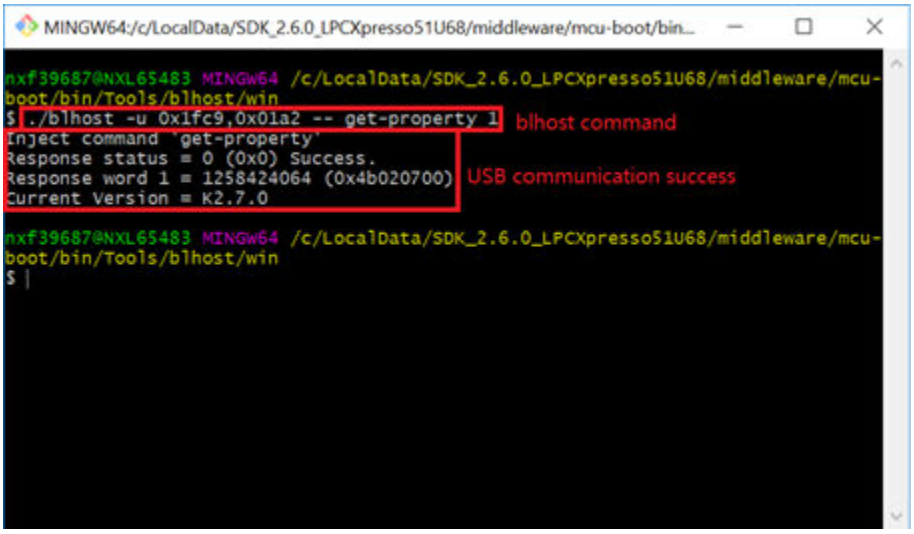

Figure 16. USB communication test

4.1.5 Flash API porting

The USB bootloader for LPC54018 updates the firmware to the on-chip Quad SPI Flash through the SPIFI interface. Unlike LPC54018, LPC51U68 contains a 256-KB on-chip flash, which can be accessed through the Flash In-Application Programming (IAP). Therefore, the LPC51U68 USB bootloader requires the Flash API which implements the read, write, and erase operations to update the firmware to the on-chip flash. The Flash API for LPC51U68 is implemented in the *internalFlashAPI.c* and *internalFlashAPI.h* files created in step 8 in Creating new project and adding necessary files.

4.1.6 Memory map porting

The memory maps for LPC54018 and LPC51U68 are different. Open *\middleware\mcu-boot\targets\LPC51U68\src\memory_map_LPC51U68.c* and redefine the "g_memoryMap" structure, as shown in Figure 17.

```
memory_map_entry_t g_memoryMap[] =
{
    #if BL_FEATURE_USING_INTERNAL_FLASH
    { 0x00000000, 0x0003ffff, kMemoryNotExecutable | kMemoryType_FLASH, &g_internalFlashInterface },
    #endif
    { 0x04000000, 0x04007fff, kMemoryIsExecutable  | kMemoryType_RAM, &g_normalMemoryInterface    },
    { 0x20000000, 0x2000ffff, kMemoryIsExecutable  | kMemoryType_RAM, &g_normalMemoryInterface    },
    { 0x40000000, 0x4001ffff, kMemoryNotExecutable | kMemoryType_Device, &g_deviceMemoryInterface },
    { 0x40020000, 0x4003ffff, kMemoryNotExecutable | kMemoryType_Device, &g_deviceMemoryInterface },
    { 0x40040000, 0x4005ffff, kMemoryNotExecutable | kMemoryType_Device, &g_deviceMemoryInterface },
    { 0x40080000, 0x400a0fff, kMemoryNotExecutable | kMemoryType_Device, &g_deviceMemoryInterface },
    { 0 }
};
```

Figure 17. LPC51U68 memory map structure

Open *\middleware\mcu-boot\targets\LPC51U68\src\bootloader_config.h* and define the "BL_FEATURE_USING_INTERNAL_FLASH" macro:

```
#define BL_FEATURE_USING_INTERNAL_FLASH (1)
```

Open *\middleware\mcu-boot\src\memory\memory.h* and export "g_internalFlashInterface":

```
#if BL_FEATURE_USING_INTERNAL_FLASH
extern const memory_region_interface_t g_internalFlashInterface;
#endif
```

4.1.7 Testing flash operations for flash-resident version

To test flash operations such as erase, read, write, and reset for a flash-resident version, follow these steps:

- Compile the project.
- Fit the JP10 jumper on the LPCXpresso51U68 board to connect the USB (J5) VBUS to the LPC51U68.
- Use a USB cable to connect the J6 header and PC and download the project executable binary file to the LPC51U68 flash area.
- Use a USB cable to connect the J5 header and PC.
- Power up and reset the LPCXpresso51U68 board.
- Run *\middleware\mcu-boot\bin\Tools\blhost\win\blhost.exe* and enter the blhost commands shown in Figure 18, Figure 19, and Figure 20. If the following command feedbacks appear, it indicates that the bootloader porting of flash operations is implemented successfully. Note that *gpio_led_output.bin* is an example of a user application binary file (other user binary file names are also OK). Note that "0x80d5" in the execute command is a routine entry address which is a reset handler address in a vector table and it can be read from a binary file (as shown in Figure 21).

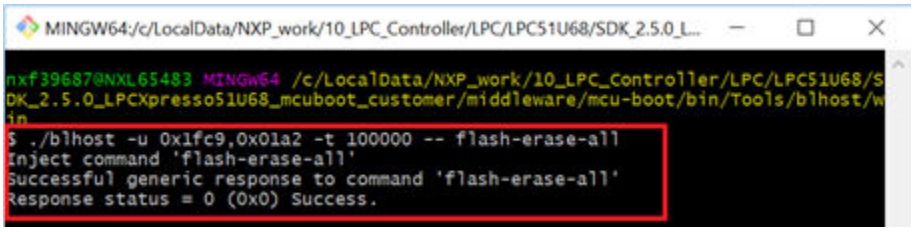

Figure 18. Flash erase all

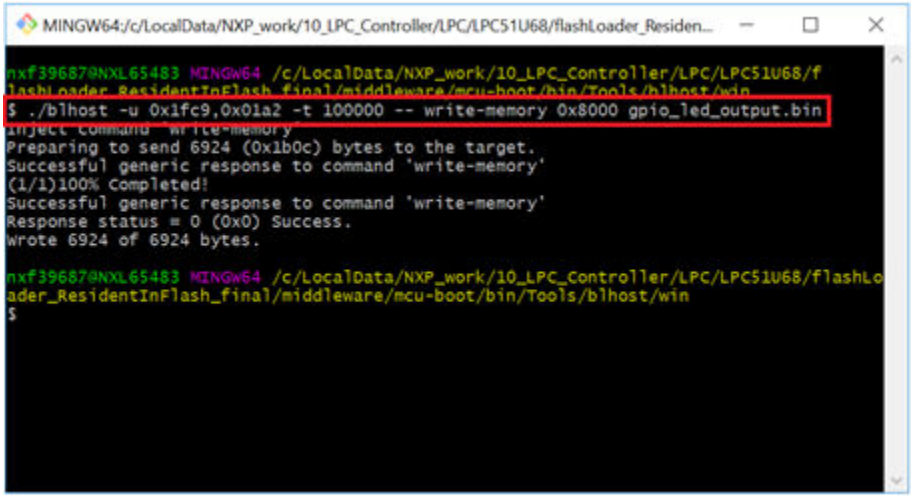

Figure 19. Flash write

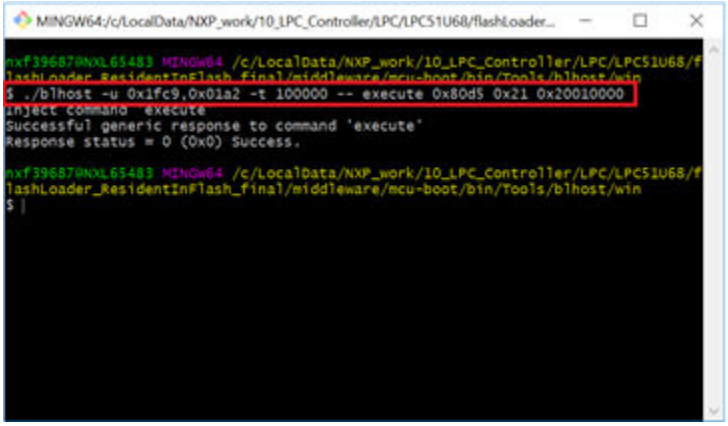

Figure 20. Execute program

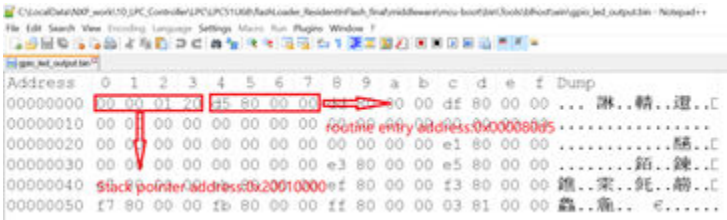

Figure 21. Entry address and stack pointer address

4.2 RAM-resident USB bootloader porting

4.2.1 Configuring Keil IDE

The project configurations for the RAM-resident version are basically consistent with the flash-resident version. The following are the differences between the RAM-resident version and the flash-resident version.

4.2.1.1 Setting preprocessor symbols and header file path

Click the "C/C++" tab and use character string "_DEBUG=1,DEBUG, CPU_LPC51U68JBD64, USB_STACK_BM, USB_STACK_USE_DEDICATED_RAM=1, BL_TARGET_RAM" as the preprocessor symbols (see Figure 22).

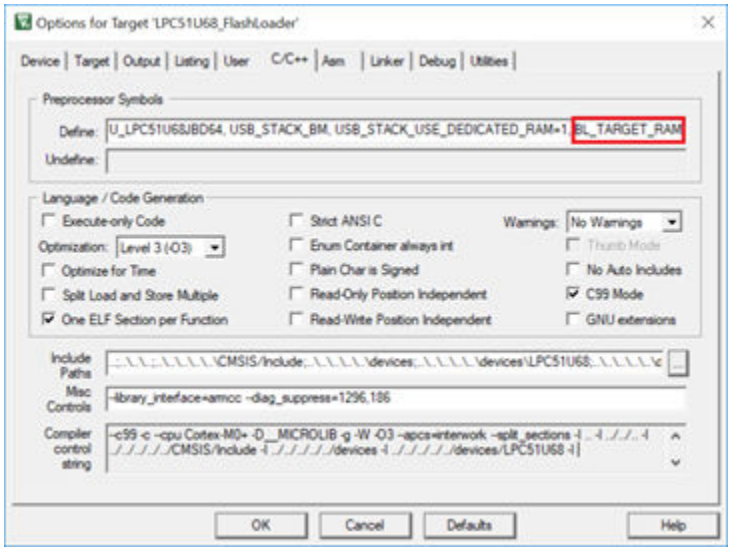

Figure 22. Setting bootloader target location

4.2.1.2 Linker setting

The scatter file for the RAM-resident version is shown in Figure 23. It is different from that for the flash-resident version.

```
#! armcc -E

/* Sizes */
#if (defined(__stack_size__))
  #define Stack_Size                    __stack_size__
#else
  #define Stack_Size                    0x2000
#endif

#if (defined(__heap_size__))
  #define Heap_Size                     __heap_size__
#else
  #define Heap_Size                     0x1000
#endif

#define  m_interrupts_start             0x20000000
#define  m_interrupts_size              0x00000400

#define  m_text_start                   0x20000400
#define  m_text_size                    0x0000FC00
LR_m_text m_text_start m_text_size {   ; load region size_region
  ER_m_text m_text_start m_text_size - Stack_Size-Heap_Size{ ; load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
    .ANY (+RW +ZI)
  }
  ARM_LIB_HEAP (ImageLimit(ER_m_text)) EMPTY Heap_Size {    ; Heap region growing up
  }
  ARM_LIB_STACK (ImageLimit(ARM_LIB_HEAP) + Stack_Size) EMPTY -Stack_Size { ; Stack region growing down
  }
}

LR_m_interrupts m_interrupts_start m_interrupts_size {
  VECTOR_ROM m_interrupts_start m_interrupts_size { ; load address = execution address
    * (RESET,+FIRST)
  }
}

LR_m_interrupts_ram m_interrupts_start m_interrupts_size {
  VECTOR_RAM m_interrupts_start m_interrupts_size { ; load address = execution address
    .ANY (.m_interrupts_ram)
  }
}
```

Figure 23. Scatter file

4.2.1.3 Debug setting

Click the "Debug" tab and select the simulator according to the actual situation. This application note uses the on-board J-Link debugger of the LPCXpresso51U68 board. Uncheck the "Load Application at Startup" checkbox (as shown in Figure 24). Click the "Edit" button in the "Initialization File" area and edit the *JLink Settings.ini* file (as shown in Figure 25).

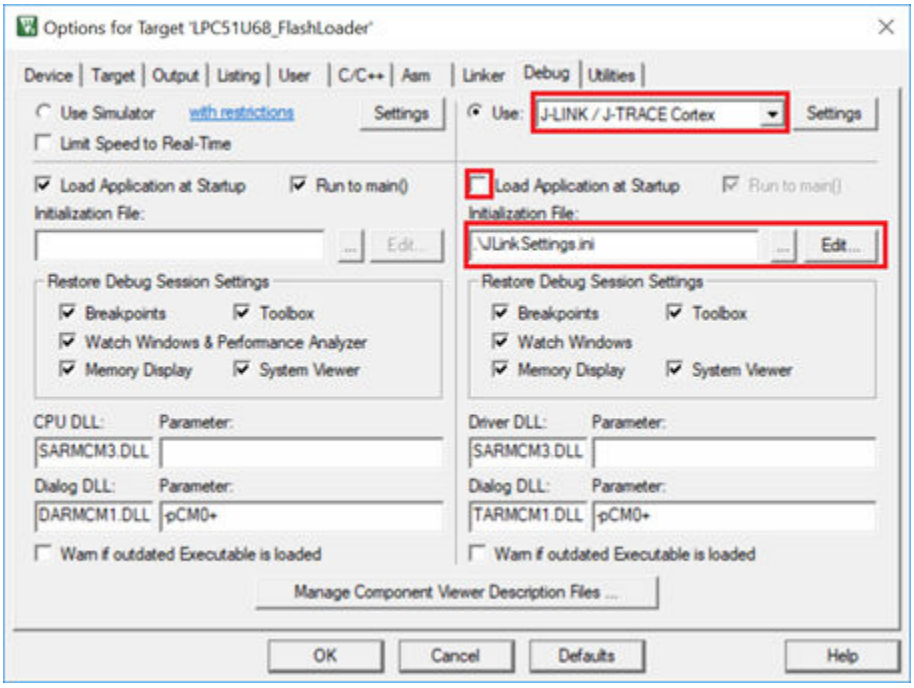

Figure 24. Debug setting

```
/*
 * Copyright (c) 2015 Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

FUNC void Setup (void) {
  SP = _RDWORD(0x20000000);            // Setup Stack Pointer
  PC = _RDWORD(0x20000004);            // Setup Program Counter
  _WDWORD(0xE000ED08, 0x20000000);     // Setup Vector Table Offset Register
}

LOAD %L INCREMENTAL                    // Download to RAM
Setup();

g, main
```

Figure 25. Initialization file

4.2.1.4 Utilities setting

Click the "Utilities" tab and uncheck the "Update Target before Debugging" checkbox (as shown in Figure 26). Click the "Settings" button and enter the "Cortex JLink/JTrace Target Driver Setup" interface. The download and programming algorithm settings are shown in Figure 27.

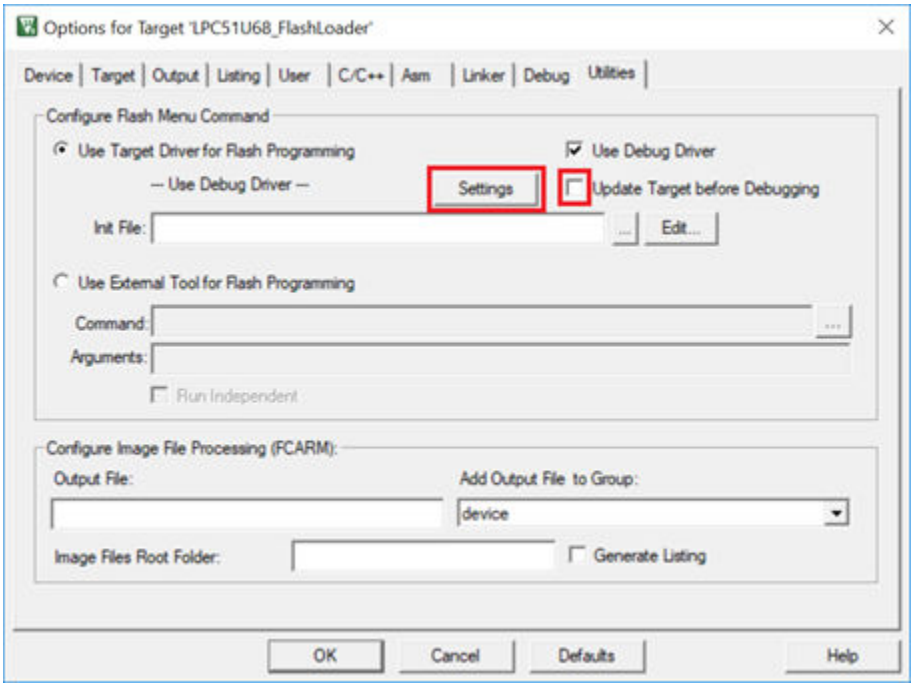

Figure 26. Utilities setting

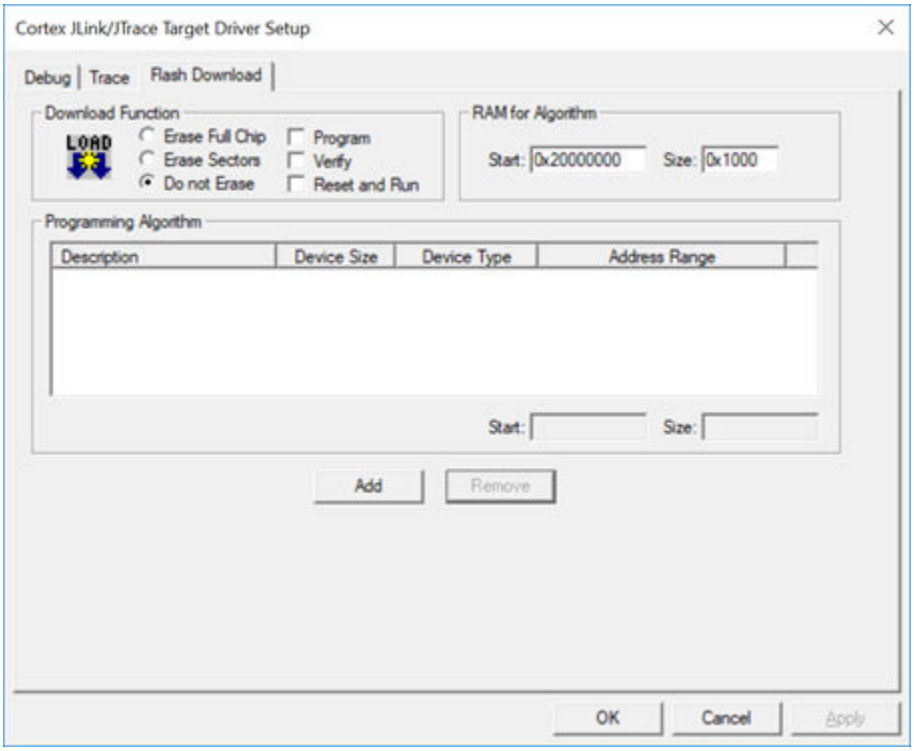

Figure 27. Download and programming algorithm

4.2.2 Testing flash operations for RAM-resident version

To test the flash operations such as erase, read, write, and reset for the RAM-resident version, follow these steps:

- Compile the project.
- Fit the JP10 jumper on the LPCXpresso51U68 board to connect the USB (J5) VBUS to the LPC51U68.
- Use a USB cable to connect the J6 header and PC and download the project executable binary file to the LPC51U68 RAM area.
- Use a USB cable to connect the J5 header and PC.
- Power up and reset the LPCXpresso51U68 board.
- Run *\middleware\mcu-boot\bin\Tools\blhost\win\blhost.exe* and enter the "blhost" command (as shown in Figure 28, Figure 29, and Figure 30). If the following command feedbacks appear, the bootloader-porting flash operations are implemented successfully.

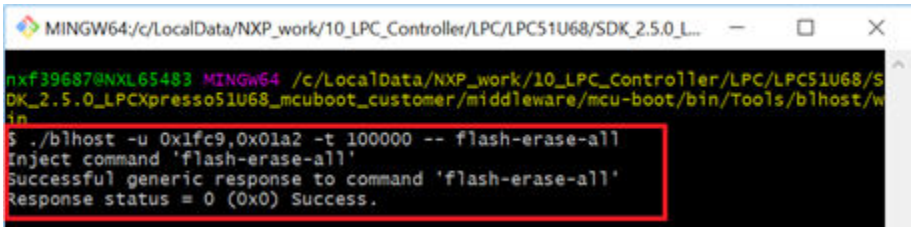

Figure 28. Flash erase all

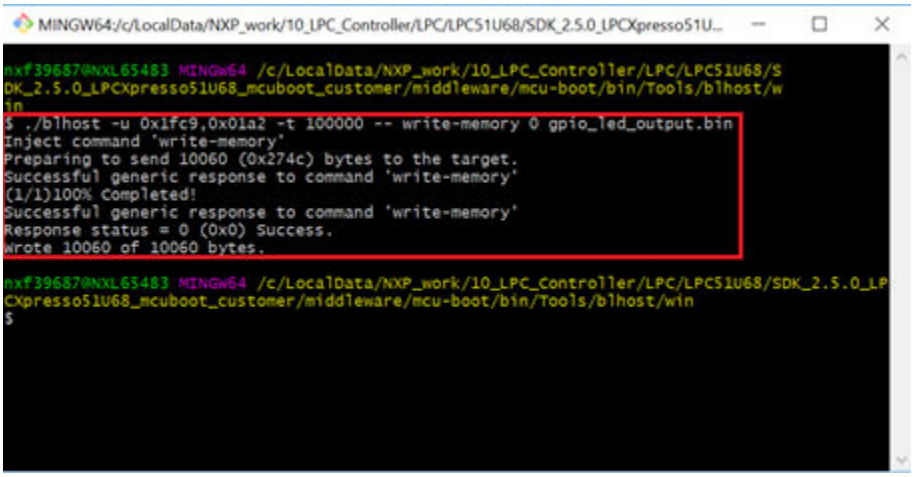

Figure 29. Flash write

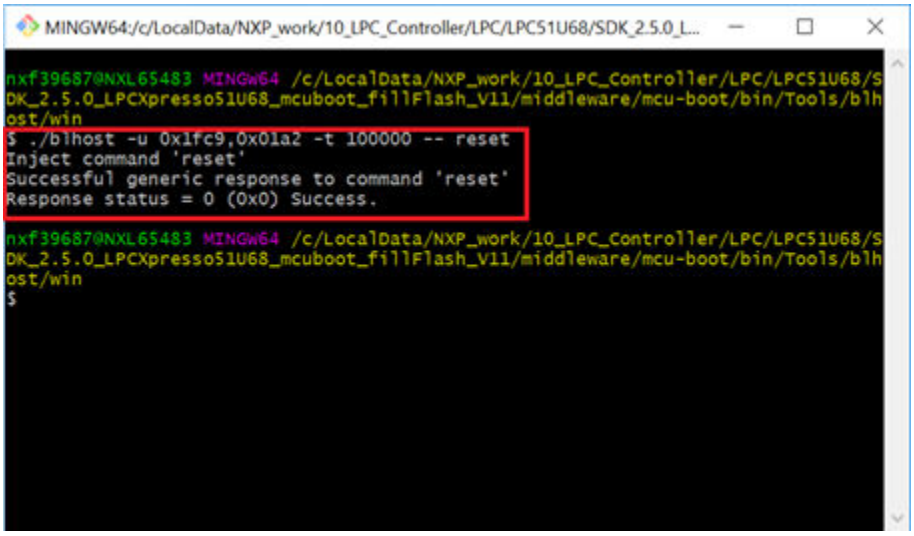

Figure 30. Execute program

5 References

- SDK_2.6.0_LPCXpresso51U68 URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpcxpresso51u68-for-the-lpc51u68-mcus:OM40005
- SDK_2.6.0_LPCXpresso54018 URL: https://www.nxp.com.cn/design/microcontrollers-developer-resources/lpcxpresso-boards/lpcxpresso54018-development-board:OM40003#buy

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 02/2020

Document identifier: AN12689