NXP Semiconductors
Application Note

Document Number: AN5319
Rev. 0, 08/2016

Serial Bootloader for MPC5748G

by: Lukas Zadrapa

Contents

1 Introduction

This application note covers the operation and use of a flash resident bootloader for the MPC5748G microcontroller.

The bootloader can be a convenient way to support programming during production or “in-system”, where support for the dedicated JTAG interface may not be available. Users must pre-program the MPC5748G with the bootloader during pre-production or at a programming vendor. The bootloader resides in the MCU for further use.

This bootloader implementation allows user software to be downloaded into the MCU flash memory using the serial (UART) interface.

The bootloader described in this document is only an example and comes with no guarantees and no support.

2 Used SW and HW tools

The software and hardware tools used are:

- SW IDE: Green Hills MULTI IDE v6.1.4
- Debugger: Lauterbach Trace32 In-Circuit debugger
- EVB: Main board MPC574XG-MB Rev.C and expansion board MPC574XG-324DS Rev.A
- MCU: PPC5748GMMN6A, mask set 1N81M
- Terminal emulator: Tera Term, version 4.76 (SVN#5085)

3 Creating of projects

It is recommended to use two independent projects for bootloader and user application in order to avoid possible cross references (typically when using libraries). The projects must be configured to use only selected flash address range without overlapping. Both projects will take an advantage of several possible locations of boot header. Both projects will have own boot header and reset vector placed at different locations, so that projects can be developed, debugged, and executed independently without any modifications. Once the bootloader is ready, it can be linked to user application in a binary form, so only one image is downloaded to flash in production.

3.1 Boot header

The BAF searches the flash memory for the boot header, checking the first word location at the addresses mentioned in the table below. The blocks are searched in the order shown in the following table. Once a Boot Header is found, no further blocks are searched.

The first header found that contains the value 005Ah in the first halfword is valid for booting.

Table 1. Locations of boot headers

Search order	Block	Address
1	16KB Code Flash block	00F8_C000
2	32KB Code Flash block	00FC_0000
3	32KB Code Flash block	00FD_8000
4	64KB Code Flash block	00FE_0000
5	16KB Code Flash block	00F9_0000
6	16KB Code Flash block	00F9_4000
7	16KB Code Flash block	00F9_8000
8	16KB Code Flash block	00F9_C000
9	16KB Code Flash block	00FA_0000
10	16KB Code Flash block	00FA_4000
11	16KB Code Flash block	00FA_8000

3.2 Project for bootloader

Create a project for bootloader and open its linker file (standalone_romrun.ld). This is default definition of flash memory segments:

Example 1. Default memory segments

```
flash_rsvd1  : ORIGIN = 0x00400000, LENGTH = 0x00b8c000
flash_memory : ORIGIN = 0x00f8c000, LENGTH = 0x005f4000
flash_rsvd2  : ORIGIN = .,          LENGTH = 0
```

Now it is necessary to allocate flash memory that will be used only for bootloader. Let’s say that we will use first two 16 KB blocks: 0x00F8_C000 – 0x00F8_FFFF and 0x00F9_0000 – 0x00F9_3FFF. The bootloader will use boot header at address 0x00F8_C000. Change the flash_memory segment as follows:

Example 2. Modified memory segments for bootloader

```
 Example 2.    Modified memory segments for bootloader
 flash_memory : ORIGIN = 0x00f8c000, LENGTH = 0x00008000
flash_rsvd2   : ORIGIN = .,          LENGTH = 0
```

The rest of flash memory will be used for user application.

3.3 Project for user application

Create a project for user application and open its linker file. User application will use flash memory range 0x00F9_4000 – 0x0157_FFFF. Boot header at address 0x00F9_4000 will be used. Change the flash_memory segment as follows:

Example 3. Modified memory segments for user application

```
flash_rsvd1   : ORIGIN = 0x00400000, LENGTH = 0x00b8c000
flash_memory  : ORIGIN = 0x00f94000, LENGTH = 0x005ec000
flash_rsvd2   : ORIGIN = .,          LENGTH = 0
```

Now we have two projects which works independently. In default linker file, the boot header is placed at the beginning of flash_memory segment:

Example 4. ROM sections

```
//
// ROM SECTIONS
//
    .boot_header:                      > flash_memory
    .text:                                  > flash_memory
    .vletext:                              > .
```

That means no other setting are needed because the boot header will be moved to base address of bootloader or user application automatically.

4 Bootloader

As mentioned in Boot header, if more valid boot headers is available in flash memory, the first one is used. If both bootloader and user application are programmed in flash memory, the bootloader will be always executed because the boot header at address 0x00F8_C000 is first in order. The startup code of bootloader must make a decision if bootloader or user application will be executed. The decision can be made on the basis of state of GPIO pin, content of flash memory, command received via a communication interface or other condition which is required by application. This example uses GPIO pin for decision.

If fast start up of user application is required, the startup condition should be checked immediately after reset by assembler code before executing startup code of bootloader project. Jumping at reset vector of user application can take only a few tens of cycles.

Project created in Green Hills MULTI IDE uses file crt0.ppc as a default startup file. This code is added at the beginning of _start() function:

Example 5. ROM sections

```
;******************************************************************************
; Check if Bootloader or User Application is supposed to be executed
; Pin PA[1] is used. PA[1] signal is routed to SW3 on EVB board.
; If PA[1] == 0  -  execute User Application
; If PA[1] == 1  -  execute Bootloader

        ;configure pin PA[1] as input - write SIUL2.MSCR[1].R = 0x00080000
              ;load address of SIUL2.MSCR[1] to r12
              e_lis    r12,0xFFFC
              e_or2i    r12,0x0244

              ;load immediate value to r11 which will enable the buffer
              e_lis    r11,0x0008

              ;write the value to SIUL2.MSCR[1]
              e_stw    r11,0(r12)

       ;check if SW3 is pushed down - if SIUL.GPDI[1] == 0x01
              ;load address of SIUL.GPDI[1] to r12
              e_lis    r12,0xFFFC
              e_or2i    r12,0x1501

              ;read the SIUL.GPDI[1] register
              e_lbz    r0,0(r12)

              ;compare the values - is the SW1 pushed?
              se_cmpli    r0,0x01

      ;execute bootloader or user application
              ;if pushed,executed bootloader
              e_beq    bootloader

              ;otherwise jump to user application

      ;but first check, if there's valid boot header at address 0xF9_4000
              ;load address of boot header to r12
              e_lis    r12,0x00F9
            e_or2i    r12,0x4000

              ;load upper half of boot header word to r0
              ;(lower half is application dependent)
              e_lhz    r0,0(r12)

              ;boot header should be 0x005A
              e_cmpl16i    r0,0x005A

      ;if the boot header is not valid, execute bootloader
              e_bne     bootloader

     ;if the boot header is valid, execute application
             ;load address of reset vector to r12
            e_lis    r12,0x00F9
            e_or2i    r12,0x4010

            ;load reset vector to r0
             e_lwz    r0,0(r12)
            ;move reset vector to link register
            mtlr    r0
            ;branch to address in link register
            se_blrl

;******************************************************************************
bootloader:
;******************************************************************************
      ; When bootloader is going to be executed, disable Software watchdog 0
      ; The watchdog is kept enabled if user application is executed.

        ;SWT_0.SR.R = 0xc520;
                e_lis    r12,0xFC05
                e_li    r0,0xC520
                e_stw    r0,0x10(r12)
        ;SWT_0.SR.R = 0xd928;
                e_li    r0,0xd928
                e_stw    r0,0x10(r12)
       ;SWT_0.CR.R = 0xFF00010A;
                e_lis    r0,0xFF00
                e_or2i    r0,0x010A
                e_stw    r0,0x0(r12)

               ;below is default startup code - continue the execution...
;*******************************************************************************
```

4.1 Initialization, serial communication interface, and user interface

The bootloader initializes system frequency to maximum, it initializes interrupts and LINFlexD_2 module which is used for serial communication.

The EVB board for MPC5748G incorporates a USB RS232 serial interface providing RS232 connectivity via a direct USB connection between the PC and the EVB. Read the user guide for the EVB for more details about drivers and connection.

If custom design is used, RS232 level shifter is necessary to communicate with PC.

Serial communication is set to format:

- 8-data bits
- One start bit
- One stop bit
- No parity
- Baud rate 115200 bit/s
- XOn / XOff flow control

As a user interface, we can use any terminal emulation program like Tera Term or Microsoft HyperTerminal. The program must support:

- Format described above
- Xon / XOff flow control
- Text file transfers

User interface guide provides more details about user interface.

4.2 Flash programming

It is recommended to use Standard Software Drivers (SSD) for flash that are available on website www.nxp.com. MPC5xxx devices implement several versions of flash module. SDD drivers for selected device can be found on its summary page under “Sofware & Tools” tab.

This example uses MPC5700 C55FG Flash Standard Software Driver (REV 1.1.0): http://www.nxp.com/files/product/software/C55_JDP_SSD.exe.

Flash memory on MPC5748G is divided to partitions. Read-while-write operation is supported only between partitions. It is not possible to access partition which is currently being programmed or erased. The code for flash programming must be executed from RAM memory or from another partition.

Bootloader occupy one block in partition 0 and one block in partition 2. In order to be able to program whole flash memory, the code must be executed from RAM memory. Following functions and resources are moved to RAM in bootloader project:

- FlashProgram_C
- FlashErase_C
- FlashCheckStatus_C
- LINFlexD_2_RX_ISR()
- LINFlexD_2_TX_ISR()
- IVORnTable
- IntcIsrVectorTable

Used C-array drivers are defined as constants by default. C-array drivers are compiled as position independent and they can be copied to any location. We can either copy the drivers to RAM manually or we can simply re-define them as variables (just delete “const”) and they will be automatically copied to RAM by startup code.

See the source files and linker file for more details about the relocation of code to RAM memory.

5 User application

There are only two requirements that must be met by user application:

- The linker file must be changed as mentioned in Project for user application.
- The S-record file which will be loaded by bootloader must have specific format as described in S-record format.

5.1 S-record format

All MPC5xxx devices implement Error Correction Code (ECC) in flash memory. ECC is handled on a 64-bit boundary. Thus, if only 1 word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word results in an operation failure (most likely).Due to ECC, flash memory must be programmed by double-words (64 bits) in a single step.

S-record files generated by Green Hills MULTI IDE or other IDEs may not be always aligned to 64-bit boundary. It is worth considering to convert the s-record file of user application to format where all s-records line are aligned to 64-bit boundary and where the length of each line is constant. This will make the flash programming much easier and safer.

Once the user application is ready to be downloaded by bootloadder, user can use attached tool SRECCONV_v2.exe to convert the file. It will align all lines to 64-bit boundary and the length of all data fields will be constant 16 bytes. The tool accepts two parameters:

SRECCONV_v2.exe <input file> <output file>

Example:

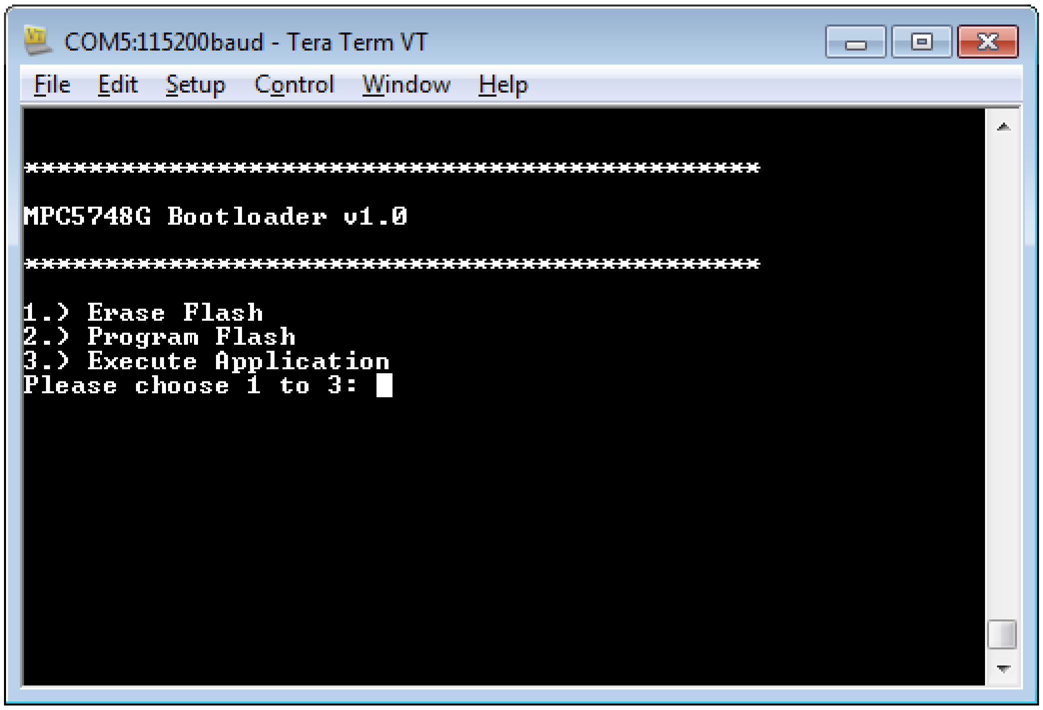
SRECCONV_v2.exe core0.run converted.s19

converted.s19 file then can be downloaded by bootloader.

Green Hills MULTI IDE does not generate s-record file by default. It is necessary to turn this feature on in Build Options.

6 User interface guide

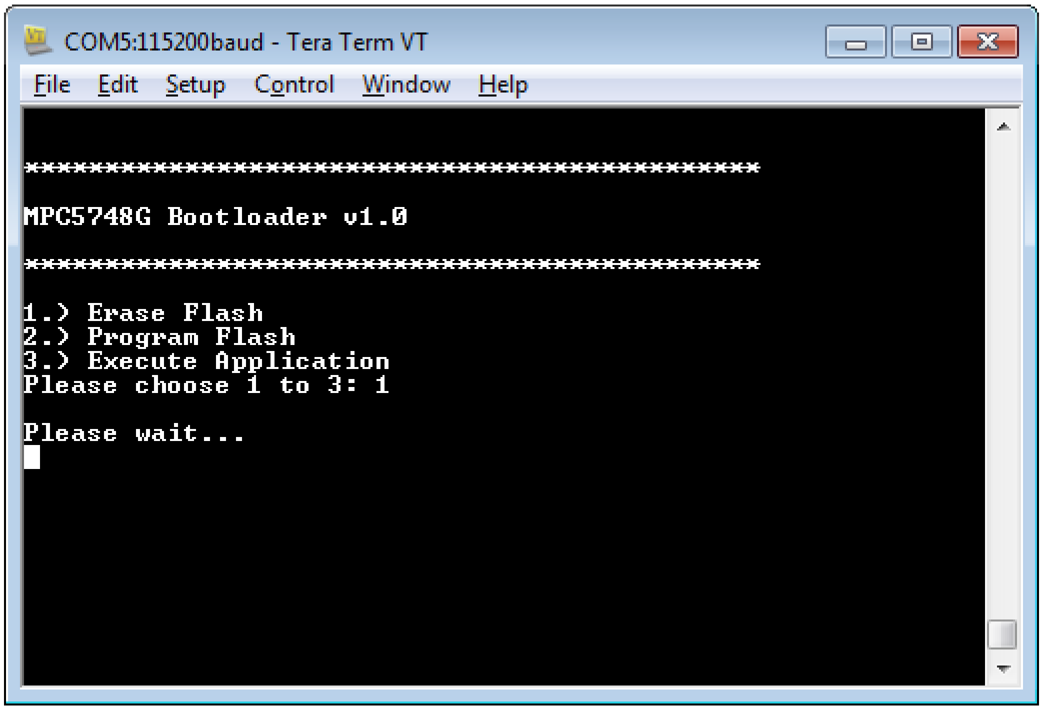
Once the bootloader is started, following welcome screen is sent via serial interface to terminal emulation program:


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
**********************************************

MPC5748G Bootloader v1.0

**********************************************

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3:
```

Figure 1. Welcome screen

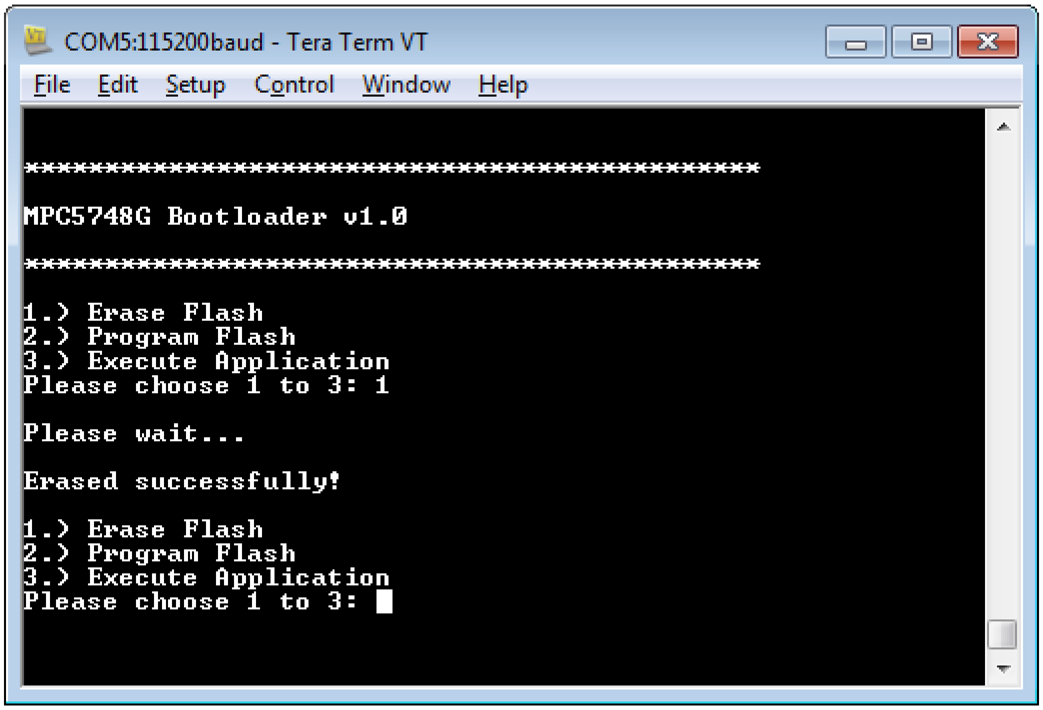
If we press ‘1’ on the keyboard, bootloader will erase all the flash memory except area occupied by bootloader. “Please wait…” message is displayed. There are 6 MB of flash in MPC5748G, so it may take a couple of seconds.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
**********************************************

MPC5748G Bootloader v1.0

**********************************************

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3: 1

Please wait...
```

Figure 2. Erase operation in progress

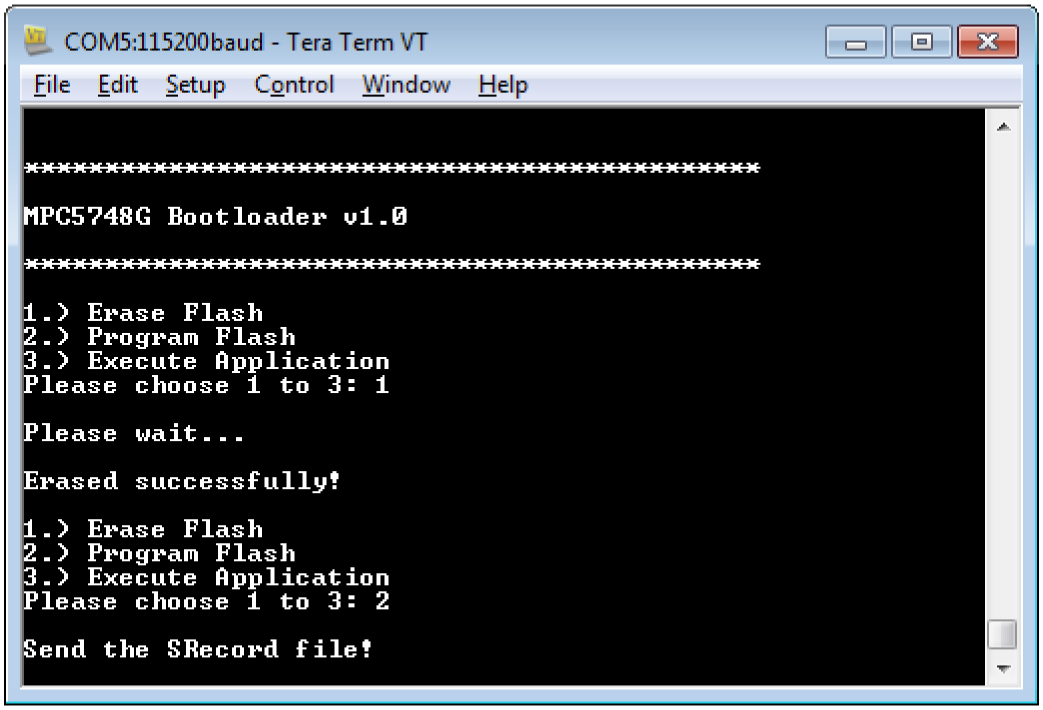
Once the erase operation is finished, bootloader will inform us if the operation was successful and displays the menu again.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
**********************************************

MPC5748G Bootloader v1.0

**********************************************

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3: 1

Please wait...

Erased successfully!

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3:
```

Figure 3. Erase successful

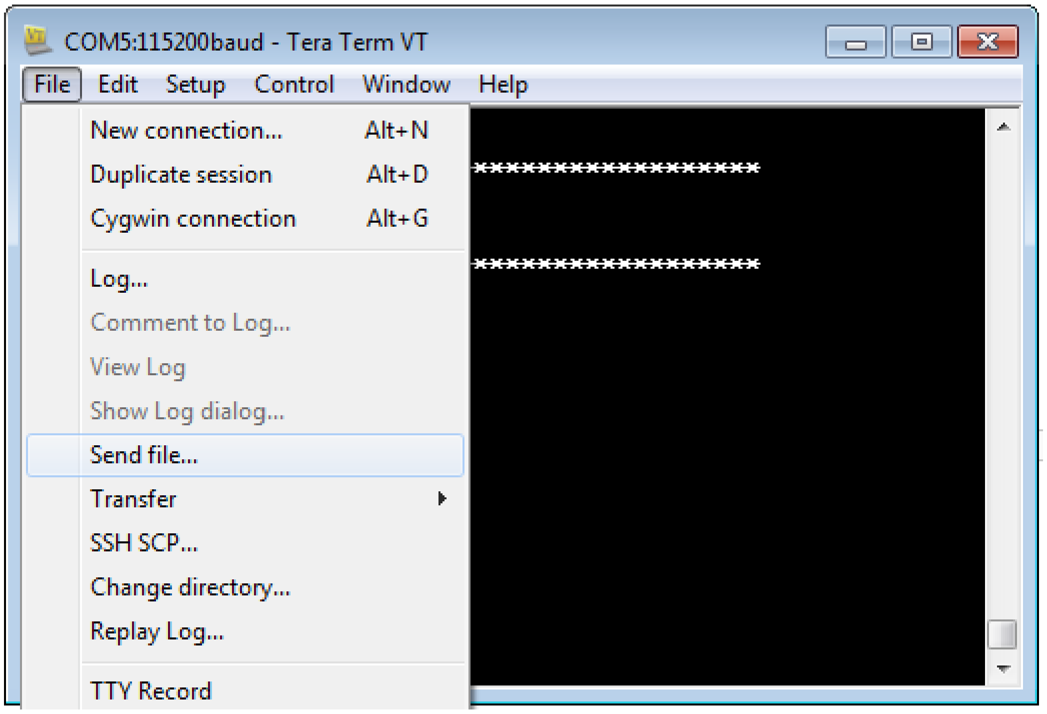
Press key ‘2’ to program the flash.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
**********************************************

MPC5748G Bootloader v1.0

**********************************************

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3: 1

Please wait...

Erased successfully!

1.) Erase Flash
2.) Program Flash
3.) Execute Application
Please choose 1 to 3: 2

Send the SRecord file!
```

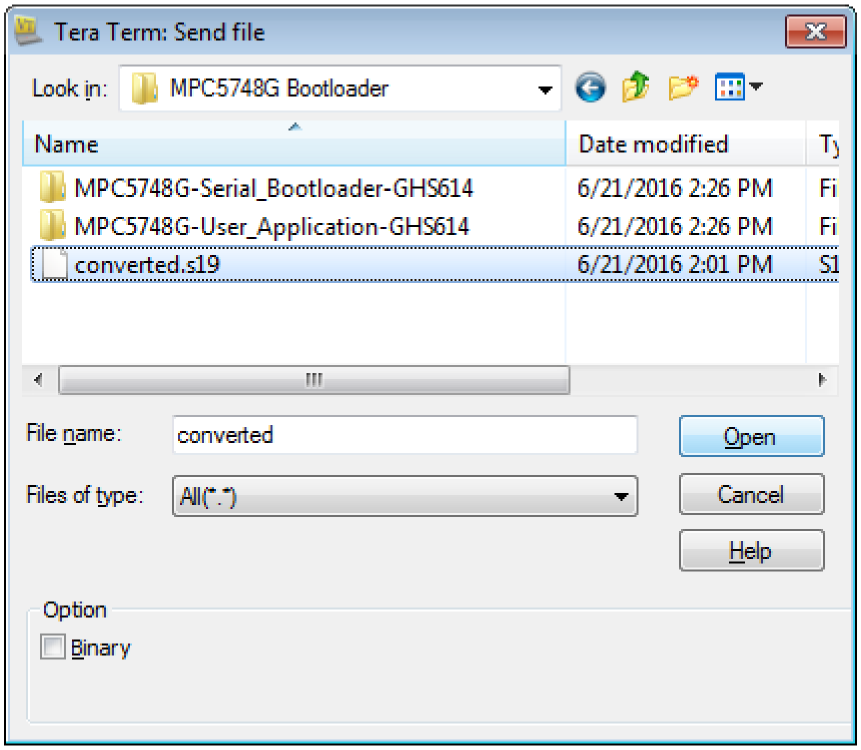
Figure 4. Program flash memory

Bootloader asks us to send the S-record file. Use the menu of terminal emulation program to select the converted S-record file.



```
********************

********************
```

Figure 5. Send S-record file



Look in: MPC5748G Bootloader

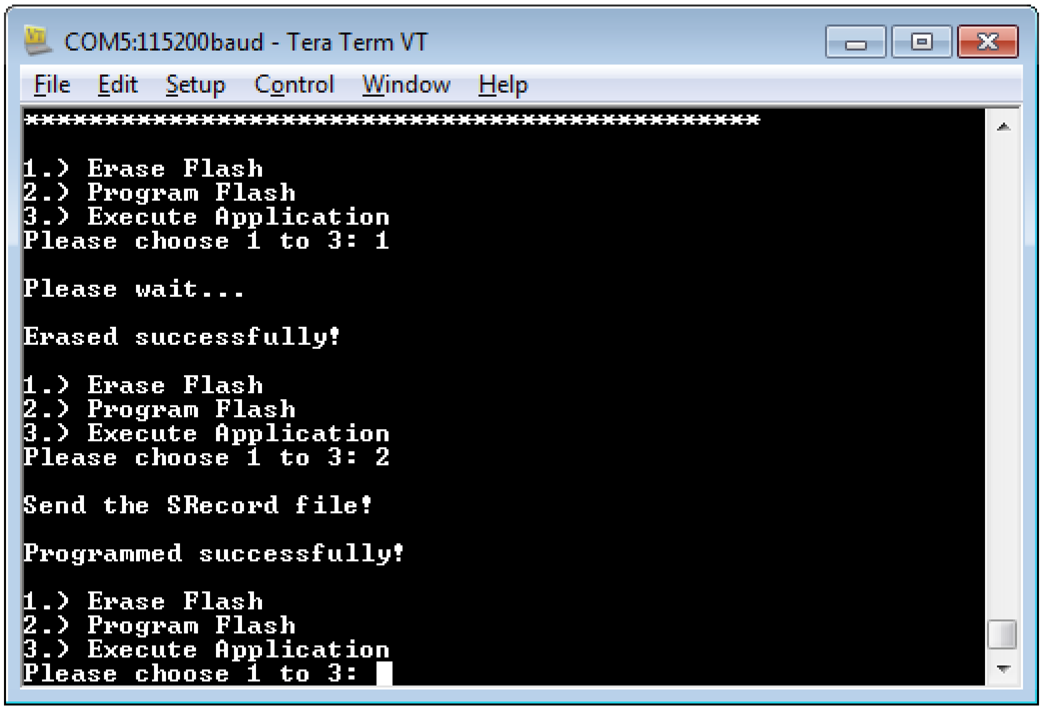
Name	Date modified	Ty
MPC5748G-Serial_Bootloader-GHS614	6/21/2016 2:26 PM	Fi
MPC5748G-User_Application-GHS614	6/21/2016 2:26 PM	Fi
converted.s19	6/21/2016 2:01 PM	S1

File name: converted

Files of type: All(*.*)

Figure 6. Browse for the S-record file

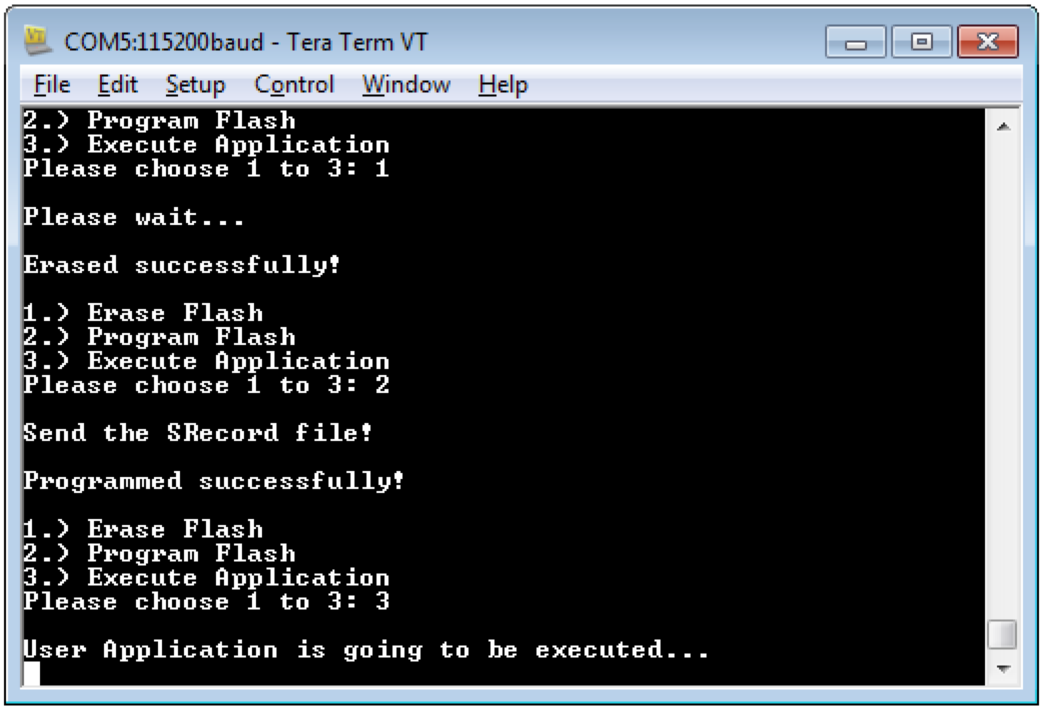
Once the program operation is finished, bootloader will inform us if the operation was successful and displays the menu again.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
*********************************************

1.> Erase Flash
2.> Program Flash
3.> Execute Application
Please choose 1 to 3: 1

Please wait...

Erased successfully!

1.> Erase Flash
2.> Program Flash
3.> Execute Application
Please choose 1 to 3: 2

Send the SRecord file!

Programmed successfully!

1.> Erase Flash
2.> Program Flash
3.> Execute Application
Please choose 1 to 3:
```

Figure 7. Program successful

Last step is to start the user application. It can be done either by pressing key ‘3’ in the terminal or we can reset the device directly. We have to ensure that the SW3 button on EVB is not pressed.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
2.> Program Flash
3.> Execute Application
Please choose 1 to 3: 1

Please wait...

Erased successfully!

1.> Erase Flash
2.> Program Flash
3.> Execute Application
Please choose 1 to 3: 2

Send the SRecord file!

Programmed successfully!

1.> Erase Flash
2.> Program Flash
3.> Execute Application
Please choose 1 to 3: 3

User Application is going to be executed...
```

Figure 8. Start user application

7 Content of zip file

All mentioned projects and utilities can be found in a zip file associated with this application note:

- MPC5748G-Serial_Bootloader-GHS614 - Bootloader project written in Green Hills MULTI IDE v6.1.4
- MPC5748G-User_Application-GHS614 – User Application written in Green Hills MULTI IDE v6.1.4
- SRECCONV – Utility for S-record converting. The folder also contains core0.run file of User Application which was generated by Green Hills MULTI IDE v6.1.4 and converted.s19 file which was converted using the SRECCONV utility. Converted file can be used to test the bootloader.

8 References

- MPC5748G Reference Manual (document MPC5748GRM, available at nxp.com)
- Serial Bootloader for S12(X) Microcontrollers Based on 180 nm Technology (document AN4258, available at nxp.com)

Document Number AN5319
Revision 0, 08/2016