RL78 Family

RL78 One Image Bootloader Example

One Image Bootloader
Rev.1.00
September 17, 2020

Introduction

This example project was created to simplify the understanding and creation of bootloader programs on the RL78. The RL78 has a unique flash interface called the Flash Self-programming Library (FSP). This example project shows the use of the FSP along with hardware features such as boot swap and the general hardware CRC.

Note that a bootloaders design is completely custom. Depending upon your projects requirements and limitations your bootloader will be different. This module shows only one possible example, modifications can and should be made to adapt it to your specific requirements.

Target Device

The following is a list of devices that are currently supported by this example program:

RL78/G14: R5F104ML

Contents

1. Terminology

There is a lot of similar terminology in this document that references to specific locations in memory. These are described in the sections as they are introduced, but to alleviate confusion please reference the below table.

Boot A	Denotes the physical memory location from address 0x0 to 0xFFF.	Hardware Terminology
Boot B	Denotes the physical memory location from address 0x1000 to 0x1FFF.	
App	Denotes the physical memory location from address 0x2000 to the end of memory.	
Boot L	Denotes the application built from the bootloader project. Since this is referencing the software it can be located in either physical memory boot locations (Boot A or Boot B)	Software Terminology
App X/Boot X	Denotes the Boot-App pair build from the bootApplication project. The “X” is a variable for the version number (1, 2, etc…). The Boot section of this is just what code has been allocated into the boot section.	
App 0/Boot 0	Denotes the Boot-App pair built for version 0.	
App 1/Boot 1	Denotes the Boot-App pair built for version 1.	

2. Overview

2.1 Hardware

This module is designed to run with the RL78/G14 Fast Prototyping Board. The bootloader interfaces with the Flash Self-Programming Library to program new applications to the device. The techniques used in this module can be adapted to any RL78 device that uses boot swap. Please review the hardware manual of the specific MCU you are using for any differences.

These applications are downloaded in real time via a UART connection going to UART0. The PC system connects to this UART via PMOD1. A USB to UART converter is needed for communication, this module was developed using a TTL-232R-3V3 converter. The converters TXD line is placed into PMOD1 pin 3, the RXD line is placed into PMOD1 pin 2, and GND to PMOD1 pin 5.

This module also makes use of shorting EJ1. When this jumper is shorted the on-board debugger is held in reset and the MCU is set to RUN mode.

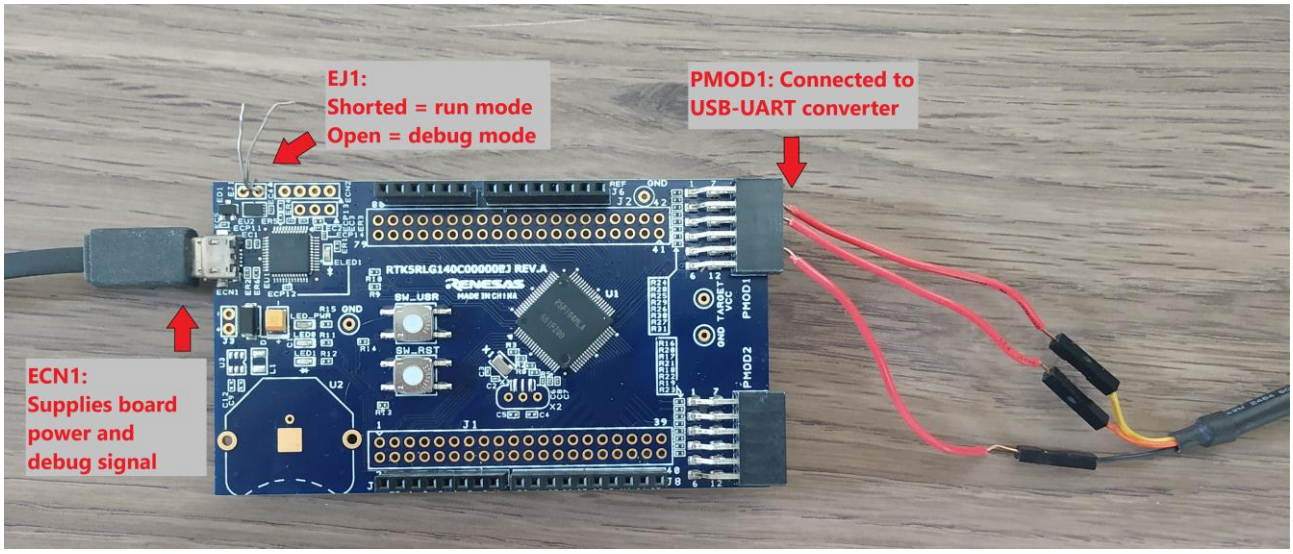

Image 1

2.2 Software

This modules software can be viewed as 3 separate components: the RL78 Bootloader, the RL78 application, and the PC program. The batch files used in conjunction with the srec_cat.exe program are located within the RL78 Application.

2.2.1 RL78 Bootloader

The bootloader program, developed in CCRL, is used to receive a new application from the UART communications and program it into Flash memory.

2.2.2 RL78 Application

The Application program is meant to act as a stand in for a custom application. Its only functions are to turn on an LED and wait for a new application program to become available.

(1) **Srec batch files**

The srec files are located inside the RL78 Application project, in the srec directory. The application project executes these as a post build process.

The srec batch files are used to generate files used to program to the RL78 device. The CRC for the application is calculated here and placed in the last two bytes of memory. Hex files are generated of only the application and its CRC, these files are used by our PC program to download a new version of the application to the RL78 device. Mot files are generated of the bootloader, application, and the applications CRC. The mot files are used as an initial image to be flashed to the RL78 device with Renesas Flash Programmer.

The batch files are executed with the tool srec_cat.exe. This tool can be found online and is not affiliated with Renesas.

2.2.3 PC Program

The PC program is used to parse the files generated from the srec output and download them to the RL78 device.

2.3 CRC

All CRCs used in this application are CRC-CCITT-Kermit functions, with a polynomial of x^16 + x^12 + x^5 + 1. This is to keep uniform with the RL78s hardware CRC that uses this function.

2.4 Tools

2.4.1 E2 Studio

This example program uses the E2 Studio IDE, version 7.8 or later, for development.

2.4.2 CCRL

This example program uses the CCRL version 1.09 toolchain/compiler

2.4.3 SREC CAT

This example program uses the third party srec_cat.exe. At the time of writing this document, srec_cat and supporting documentation can be found at http://srecord.sourceforge.net/

3. System Architecture and Design

3.1 Embedded Memory Allocation

Figure 1 below shows the memory segments for this bootloader solution. It is broken down into sections we will call Boot A, Boot B and App. This terminology will reference the memory locations themselves.

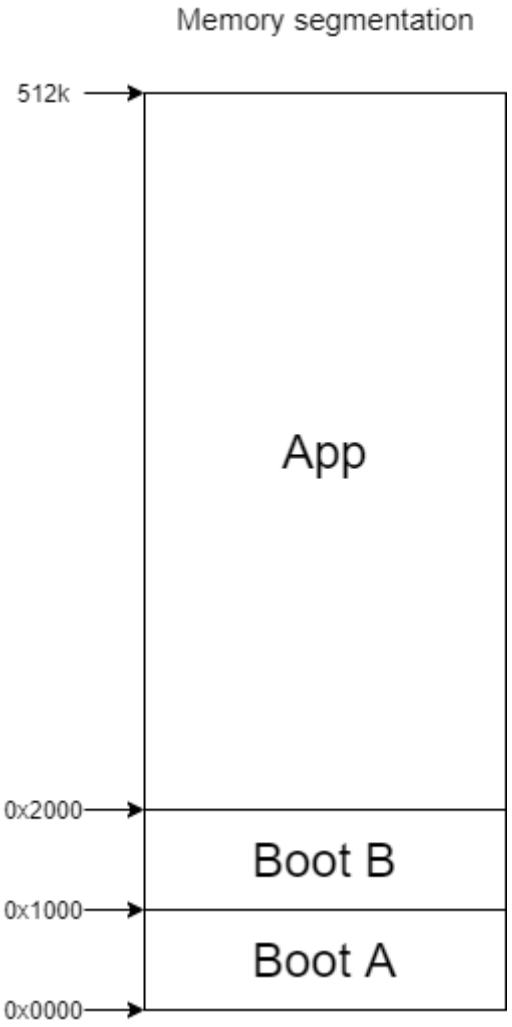

Figure 1

Boot A refers to the first boot block. In the case of the G14 this exists from memory addresses 0x0000 to 0xFFF.

Boot B refers to the second boot block. This segment exists from addresses 0x1000 to 0x1FFF.

App refers to your application area. It begins at address 0x2000 and goes to the end of memory.

3.1.1 Boot Swap

An important thing to keep in mind is that with the RL78s boot swap ability, Boot A and Boot B are interchangeable. When the boot swap bit is set, these two segments will physically swap memory locations. For example: if the boot swap bit is set, and you attempt to access the memory at address 0x80. You will read what you originally programmed to address 0x1080.

This module takes advantage of this feature and relies on the code knowing what state the boot swap bit is in. For more information on the boot swap ability please review the hardware manual.

3.1.2 Monitor Areas

This example project uses the ON-Chip Debug (OCD) option. This is not necessary at all, in fact it makes things more difficult. It is left in to showcase an example on how to work with the OCD used. When building the application generally the OCD will be set, and when you deploy your application it is expected you will remove the OCD use.

The monitor area will be last 512 or 256 bytes of flash memory. This bars you from using these areas for program space. This will exist at the end of flash memory. If you attempt to overlap your code with this section, the linker will generate an error. For more information on these sections please review the hardware manuals chapter on On-Chip Debug Function.

Since the example project uses the OCD setting, even though memory extends to 0x7FFFF, it's program image can only go to address 0x7FDFF. In this document you will see the use of the terms "End of memory" and "512k". Keep in mind that this will refer to the address 0x7FDFF as the OCD will be taking up the final 0x200 bytes of memory.

Also noteworthy, when an image is downloaded we do not include the monitor area in the download. This is because once the image is ready to download, debugging is complete and the monitor area is no longer necessary.

3.2 Program Images

This section describes each of the images created in the example project and their contents.

3.2.1 Application Image

Figure 2 shows a physical representation of the memory layout of the application project. Notice that the memory segments Boot A and App are filled but Boot B is empty.

- The Empty section will be used to hold the bootloader application.
- Boot X will hold the first 0x1000 of the application. The X in this name is a stand in for whatever is the current version of the application. What code is placed in this section is not important, what is important is to not allow any code to flow into the Empty segment.
- App X will hold the rest of your application. The X in this name also is a stand in for the current version of your application.

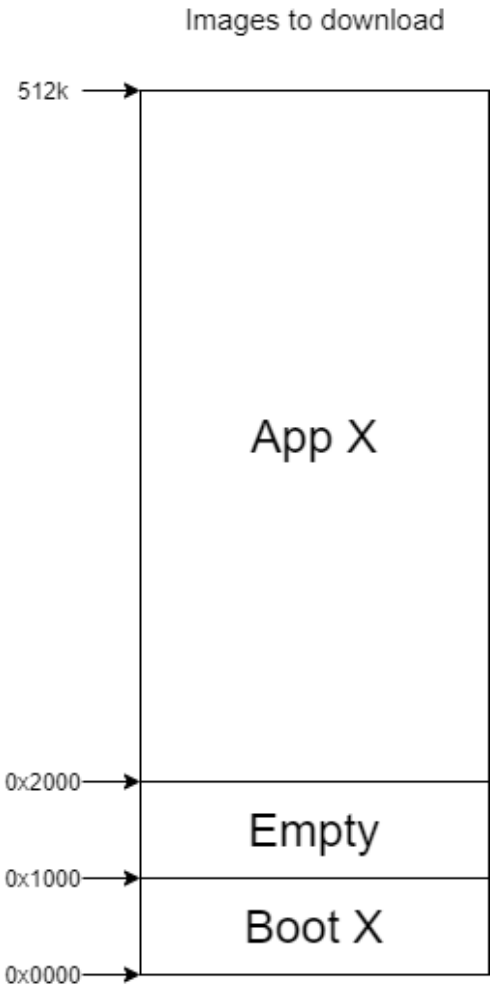

Figure 2

This design is used so that the memory segment Boot B can be used to hold the bootloader. When the application receives notice from the UART that there is a new image to download, the boot swap bit is flipped, and the device is reset.

The bootApplication image of the example program is to be used as your custom application.

3.2.2 Bootloader Image

Figure 3 shows a physical representation of the memory layout of the bootloader project. It is important to note, when the bootloader application is running it will be located at the physical addresses of 0x0000 to 0xFFF. This is due to the RL78s boot swap ability.

- The Empty section from address 0x1000 to 0x1FFF will hold the boot section of the application project (Boot X from ***Figure 2***)
- The Empty section from address 0x2000 to the top of memory is used to hold the app section of the application project (App X from ***Figure 2***)
- The Boot L section will hold all code necessary to download a new application image and flash it into memory.

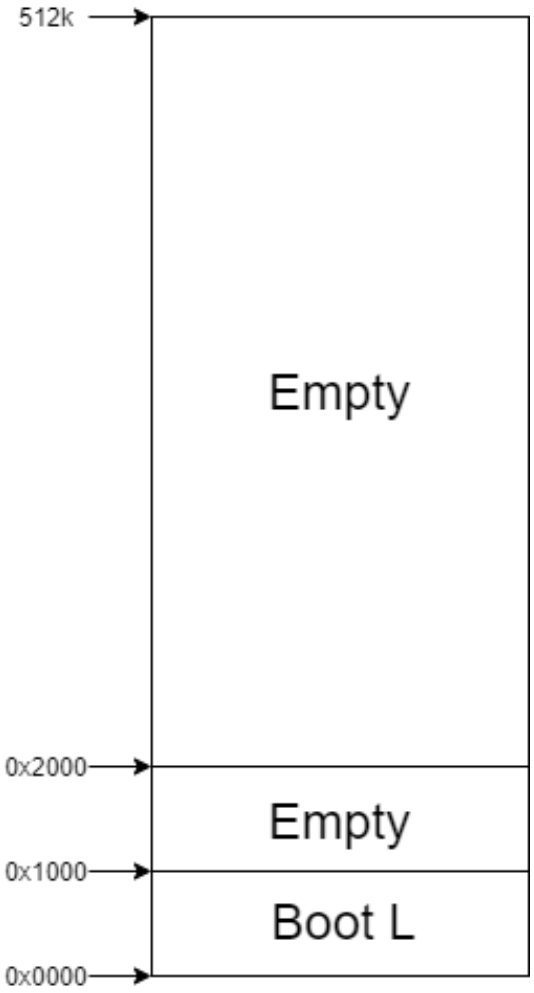

Figure 3

3.2.3 Combined Image

Figure 4 shows both the application and bootloader combined into one image. This image is how the devices memory will look at any given time. The only exception to this is when the boot swap bit is set, causing the Boot L and Boot X sections to be swapped.

While the bootloader (Boot L) was built and linked to the memory addresses starting at 0x0000, it can be stored in the 0x1000 addressed boot sector until it is needed.

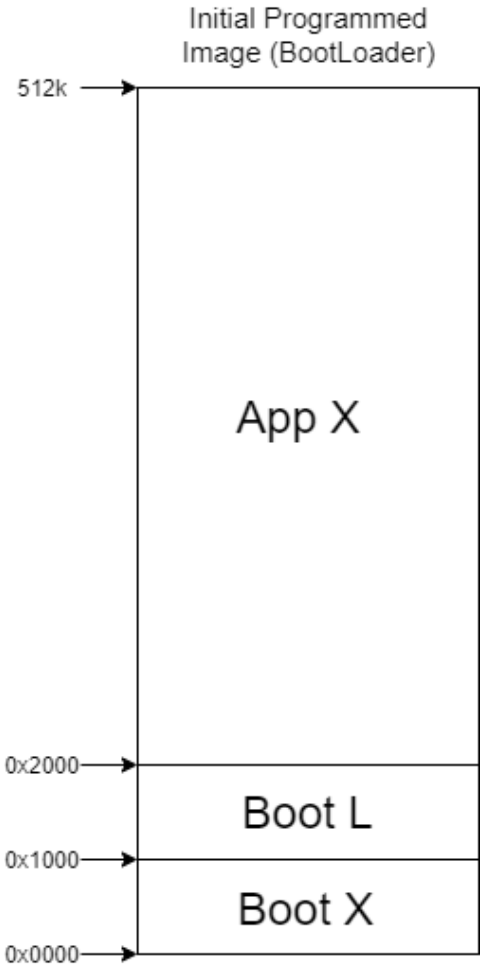

Figure 4

3.3 Embedded Code Flow

3.3.1 Downloading the Image

The download works on a Command-ACK structure. The PC program will send commands and the RL78 will complete a task as per the command. When the task has been completed an ACK or NAK will be sent to the server, indicating that the MCU is ready to receive another command.

Every command and response are compiled into a packet. Command packets consist of 69 bytes. ***Figure 5*** shows the structure of the command packets, while ***Table 1*** shows all the possible types of command packets that can be sent.

The packet CRC is used as a check for packet integrity. Both the packet counter and the data are CRCd and that CRC is appended to the end of the message. To verify the CRC the example project runs the same CRC algorithm (in this case the general hardware CRC) on the entire packet, including the appended CRC, and verifes that the CRC operation resolves to 0.

Command Packet (69 bytes)		
Packet Counter	Data	Packet CRC
Bytes 0 - 1	Bytes 2 - 66	Bytes 67 - 68

Figure 5

Command Packets		
Command	Command ID	Description
Data	0x00	This packet contains the next 64 bytes ready to be written to the MCU as well as a counter for the packets. When received the RL78 will take the data and program it into the next available 64 bytes in flash.
Version	0x01	This packet requests the version number of the application from the RL78 device.
Reset	0x02	This packet instructs the RL78 to reset without initiating a boot swap.
Transfer Complete	0x03	This packet lets the RL78 know that the last packet has been transmitted and the download has finished.
Swap and Reset	0x04	This packet instructs the RL78 to reset after initiating a boot swap.

Table 1

When a message is received by the RL78 from the PC program it will send a response, an ACK if you will. These responses will follow every message from the PC application. ***Figure 6*** shows the structure of the response packets, while ***Table 2*** shows all the possible types of response packets that can be sent.

Response Packet (5 bytes)		
Response type	Data A	Data B
Byte 0	Bytes 1 - 2	Bytes 3 - 4

Figure 6

Response Packets				
Response	Response ID	Data A Usage	Data B Usage	Description
Begin Transmission	0x01	Fill - 0xAA	Fill - 0xAA	The device has been set into bootloader mode and is ready for programming
ACK Packet	0x02	Packet Count	Current running CRC	The packet was flashed successfully
NAK Packet	0x03	Packet Count	Fill - 0xAA	The packet failed to flash properly
Send Version	0x04	Version High Bytes	Version Low Bytes	Contains the version of the application currently flashed to the device
ACK Program	0x05	Packet Count	Current running CRC	The program was flashed correctly and verified
NAK Program	0x06	Packet Count	Current running CRC	The program failed the CRC check
NAK CRC	0x07	Packet Count	Fill - 0xAA	The CRC for the last packet sent did not validate

Table 2

3.3.2 Flashing the Image

The example program uses the Flash Self-Programming Library (FSL) to program the new application. For more information about the specifics of the FSL please review the FSL documentation.

Since the example project only initializes the bootloader after it receives the command from the server, it automatically deletes the application area when it is initialized. Once the deletion is complete, it notifies the server that it is ready to begin flashing.

As each chunk of data (64 bytes at a time) comes in it is written to the flash area. A counter keeps track of the location of the writes and is incremented with each chunk. The counter begins at address 0x1000 and increments by 64 until we reach the end of the image.

3.3.3 System Execution

The images in ***Figure 7*** show a high-level overview of the code flow and the entire download process. In it, the application sections are denoted with a number. This number is representative of the version of the application currently in memory. For example: the original image flashed to the device would be named “Version 0”, the next image to be downloaded would be called “Version 1” and so on.

Below is a brief description of the steps completed in ***Figure 7***:

- Initial State
  - The part has been flashed with Boot 0, App 0, and Boot L (bootloader or Boot B described in **Error! Reference source not found.**)
- **Error! Reference source not found.**
  - App 0 verifies via the UART that a new image is ready to download
  - App 0 then swaps the boot areas and initiates a reset
- **Error! Reference source not found.**
  - Boot L, now existing at address 0x0000, downloads the newest image from the UART and flashes this into the other boot section and app section.
- **Error! Reference source not found.**
  - Boot L verifies the CRC of the new application before swapping the boot sectors and initiating a reset.

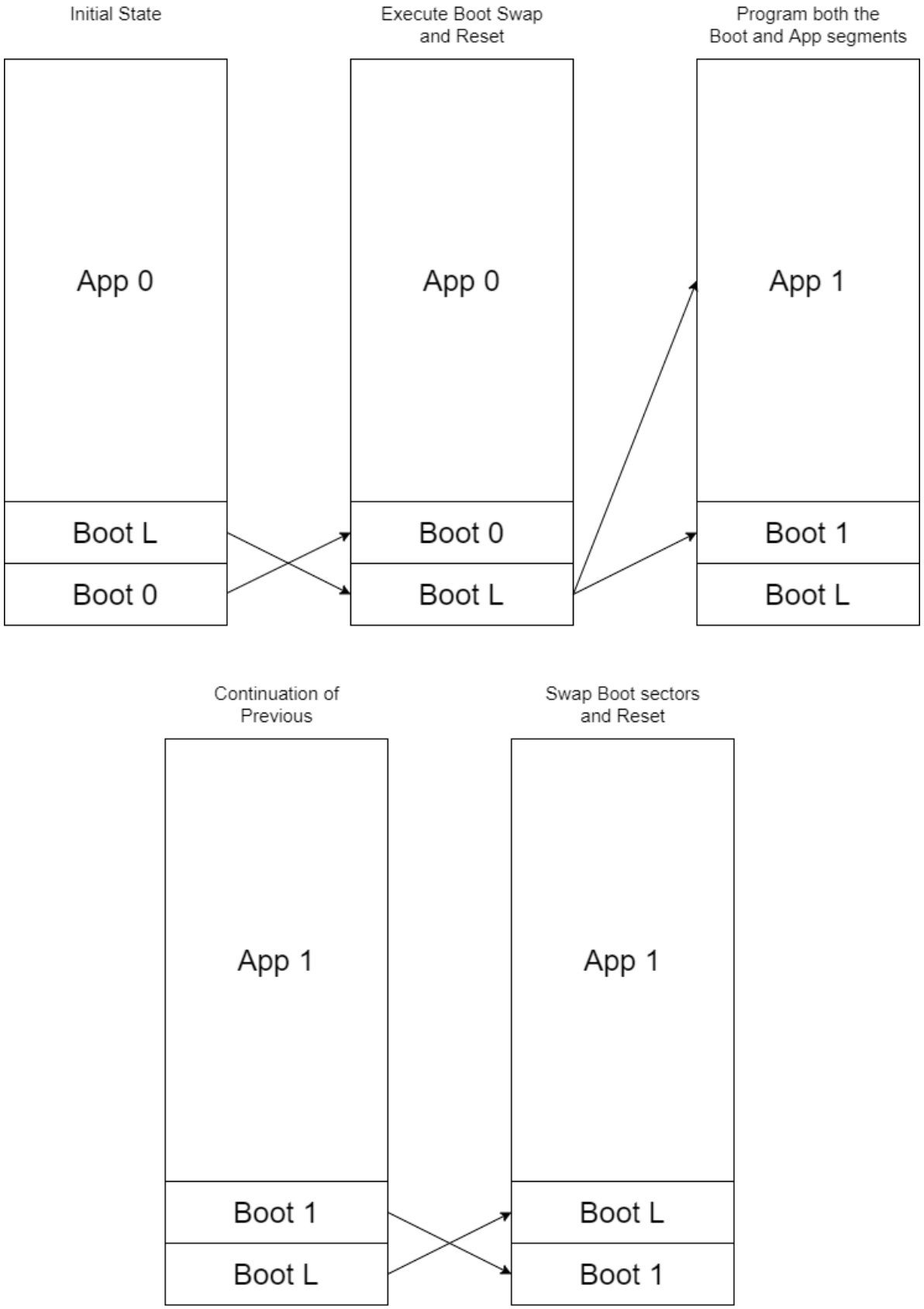
Initial State
Execute Boot Swap and Reset
Program both the Boot and App segments
App 0
App 0
App 1
Boot L
Boot 0
Boot 0
Boot L
Boot 1
Boot L
Continuation of Previous
Swap Boot sectors and Reset
App 1
App 1
Boot 1
Boot L
Boot L
Boot 1

Figure 7

4. Source Code Architecture

This example project is meant to guide for your own application. It is expected that you will have to make some changes, in this section we will discuss the source code for each project individually.

4.1 RL78 Bootloader

4.1.1 Sections

Apart from the modifications required from the flash libraries, the bootloader project itself doesn't require any modifications to the sections. Though there are some important quirks you will have to account for. ***Figure 8*** shows the section output the bootloader project was built with.

It is highly recommended to review the map file to ensure that your code fits to the following criteria.

(1) **Memory limitation**

Ensure that all your data fits within the memory addresses 0xD8 to 0xFFF. Before 0xD8 is reserved for intrinsic values such as the vector table, option bytes, etc. After this memory range is beyond the boot swap sector.

(2) **Constant data**

There can be no constant data used. Due to mirror memory limitations, constant data cannot exist before address 0x3000, and since the entirety of the bootloader project must exist in the first 0x1000 addresses it cannot accommodate any data that is designated for const.

(3) **Flash sections**

The flash sections must exist in the project. It is unimportant where exactly as long as the location is in our first 0x1000 allocation.

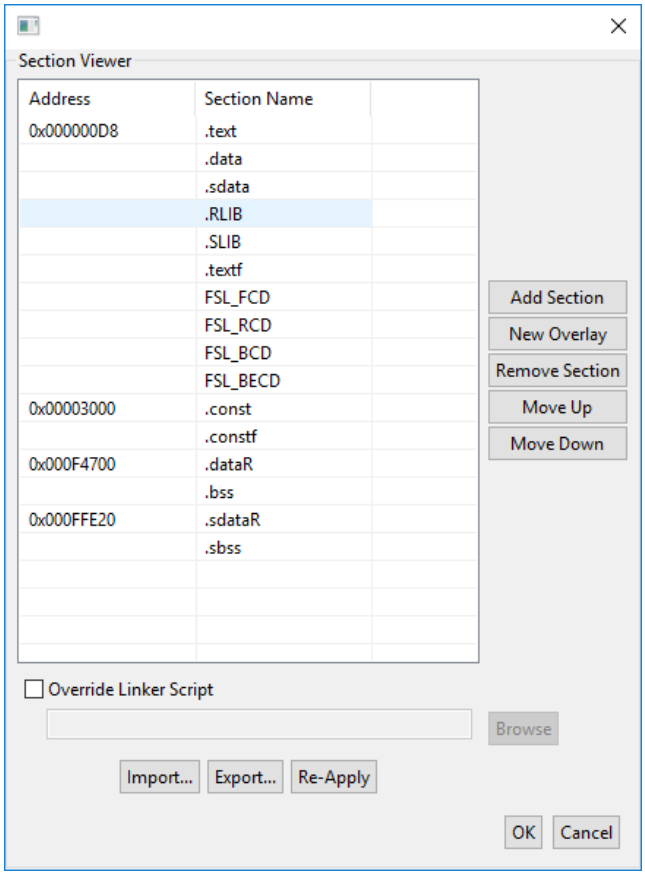

Figure 8

4.1.2 Code Generator Output

The code generator is used to output most of the driver source code. All code generator source files are in the src/CodeGenerator directory. When implementing the bootloader concepts from this example project into your custom application you will need to reconcile the differences from your applications requirements and the code generator output.

(1) **Clock Generation Settings**

The example project uses a HOCO setting of 64 (fHOCO = 64MHz; fIH = 32MHz).

(2) **Serial Settings**

The UART is set to a baud of 38,400, this was arbitrarily chosen. This is also the highest speed available in both the RL78 code generator and a standard baud setting in QTs QSerialPort using standard selections.

4.1.3 Flash Libraries

This example project uses the T02 Flash Self-Programming Library. This can be substituted for a different type of the FSL, just verify with the FSL documentation that the type being used can support the functionality required by your bootloader application.

4.1.4 Relevant Defined Values

A quick description of each of the defines may need special attention when integrating this example program with your application.

utility.h	UART_PACKET_LENGTH	Size of packets moving from the PC to the MCU

	UART_RESPONSE_LENGTH	Size of packets moving from the MCU to the PC
	MAX_PACKETS	Number of packets in the image
	CRC_ZERO_SEED	Initial seed for the running CRC of the entire image
	CRC_OK	When a CRC verifies against any data segment with that segments CRC appended at the end the CRC will always evaluate to zero
	VERSION_ADDRESS	Physical address where the version information is located
fsl_interface.h	BLOCK_XXX_YYY	These defines show the start and end blocks of each memory segment. The blocks being the 1KB memory blocks. For example: The define BLOCK_BOOT0_BEGIN gives the first memory block for the boot 0 memory segment, which is 0. Since the boot 0 memory segment is 0x1000 bytes long (or 4KB) the last block inside boot 0 would be the 3rd block or BLOCK_BOOT0_END
	BYTES_PER_WRITE	The size of each write to flash memory via the FSL
bsp.h	PORT4_OUTPUT	Used for LED initialization settings for the Fast Prototyping Board, your custom board may differ
	PM_43_44_MODE_OUTPUT	Used for LED initialization settings for the Fast Prototyping Board, your custom board may differ
	LED0_PIN	Used for Toggling the pin for LED0
	LED1_PIN	Used for Toggling the pin for LED1

4.2 RL78 Application

Much of the code from example project BootApplication is the same as in BootLoader. Some of it is not necessary for execution of the code, though it is left in to keep the code structures as similar as possible.

4.2.1 Sections

The application project requires a few more changes. ***Figure 9*** shows the section output the bootloader project was built with.

It is highly recommended to review the map file to ensure that your code fits to the following criteria.

(1) **Application Version**

The version information is held in the .constfAPP_VERSION_f section. This is only a 4 byte section used to hold the version variable. The version variable is located in the main.c file, to make a new version to download to the MCU ensure that this value has been updated.

(2) **Memory Segmentation**

As discussed in section 3.2.1 we need to ensure that no code exists from 0x1000 to 0x1FFF. This blank area will hold the bootloader project. There are multiple ways to do this so use the method that you are most comfortable with, this example program creates a separate boot section for code to be placed in 0x0000 to 0xFFF. The Boot section begins immediately after the Application Version section, at address 0x00DC.

Text code can now be inserted into the boot section by the #pragma section compiler command. An example of this can be found in the bootApplication project as the main function is placed in the boot section. Fill this section as you see fit but ensure it does not extend past address 0xFFF, you can do this via the map file.

All the other code is placed after address 0x2000. This is accomplished by moving all remaining sections to address 0x2000 and beyond.

The final segment that is placed is the placeholder for the CRC. This will be generated post-build in out srec batch files for the entirety of the application. For now, we will place the section and an uninitialized data container in the required position. The section .constfPROGRAM_CRC_f at address 0x7FDFE is used to hold the data.

(3) **Flash sections**

The flash sections must exist in the project. It is unimportant where exactly as long as the location is not within addresses 0x1000 and 0x1FFF.

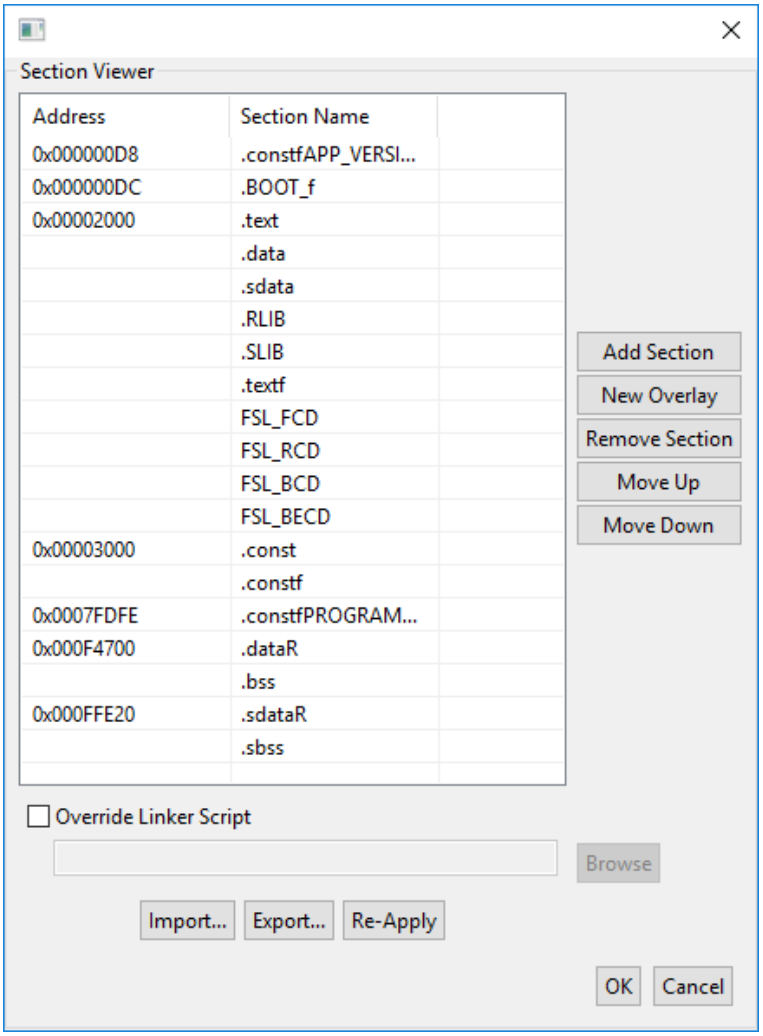

Figure 9

4.2.2 Code Generator Output

The code generator is used to output most of the driver source code. All code generator source files are in the src/CodeGenerator directory. When implementing the bootloader concepts from this example project into your custom application you will need to reconcile the differences from your applications requirements and the code generator output.

(1) **Clock Generation Settings**

The example project uses a HOCO setting of 64 (fHOCO = 64MHz; fIH = 32MHz).

(2) **Serial Settings**

The UART is set to a baud of 38,400, this was arbitrarily chosen. This is also the highest speed available in both the RL78 code generator and a standard baud setting in QTs QSerialPort using standard selections.

4.2.3 Flash Libraries

This example project uses the T02 Flash Self-Programming Library. This can be substituted for a different type of the FSL, just verify with the FSL documentation that the type being used can support the functionality required by your application.

4.2.4 Relevant Defined and Global Values

A quick description of each of the defines may need special attention when integrating this example program with your application.

main.c	USE_LED_0	Determines if LED0 or LED1 will be used when notifying the user that the device is ready to be flashed
	version	Stores the current version
	program_crc	Stores the CRC value
utility.h	UART_PACKET_LENGTH	Size of packets moving from the PC to the MCU
	UART_RESPONSE_LENGTH	Size of packets moving from the MCU to the PC
	MAX_PACKETS	Number of packets in the image
	CRC_ZERO_SEED	Initial seed for the running CRC of the entire image
	CRC_OK	When a CRC verifies against any data segment with that segments CRC appended at the end the CRC will always evaluate to zero
	VERSION_ADDRESS	Physical address where the version information is located
fsl_interface.h	BLOCK_XXX_YYY	These defines show the start and end blocks of each memory segment. The blocks being the 1KB memory blocks. For example: The define BLOCK_BOOT0_BEGIN gives the first memory block for the boot 0 memory segment, which is 0. Since the boot 0 memory segment is 0x1000 bytes long (or 4KB) the last block inside boot 0 would be the 3rd block or BLOCK_BOOT0_END
	BYTES_PER_WRITE	The size of each write to flash memory via the FSL
bsp.h	PORT4_OUTPUT	Used for LED initialization settings for the Fast Prototyping Board, your custom board may differ
	PM_43_44_MODE_OUTPUT	Used for LED initialization settings for the Fast Prototyping Board, your custom board may differ
	LED0_PIN	Used for Toggling the pin for LED0
	LED1_PIN	Used for Toggling the pin for LED1

4.2.5 Post-Build Processes

This project uses post build processes to execute the files for use with srec_cat.exe. While the executable can be ran from a command prompt, we can wrap the execution into our build process. To do this you will either need the executable srec_cat.exe added to your PCs path or placed directly in the project directory.

The files exist in the srec directory and are executed with post-build steps. To set the post build steps go to Project > Porperties > Settings and select the Build Steps tab. You should now see a screen similar to ***Figure 10***. The commands are as follows:

- srec_cat @..\srec\crc_gen.txt
- srec_cat @..\srec\compile_app.txt
- srec_cat @..\srec\compile_full.txt

Each command is separated by " & " (including the whitespace). Note that if you are using a linux system the ampersand (&) should be replaced with a semicolon (;).

When executing these files, it is recommended to have the BootLoader application in your workspace as well, if not you will have to modify the compile_full.txt file to point to the correct location. More on how to do this is in section 4.4.22 Full File Generation.

It is also very important that the crc_gen.txt file be run before the other two, so ensure that it is placed first. This is due to the fact that the other two files use the crc.mot file it generates.

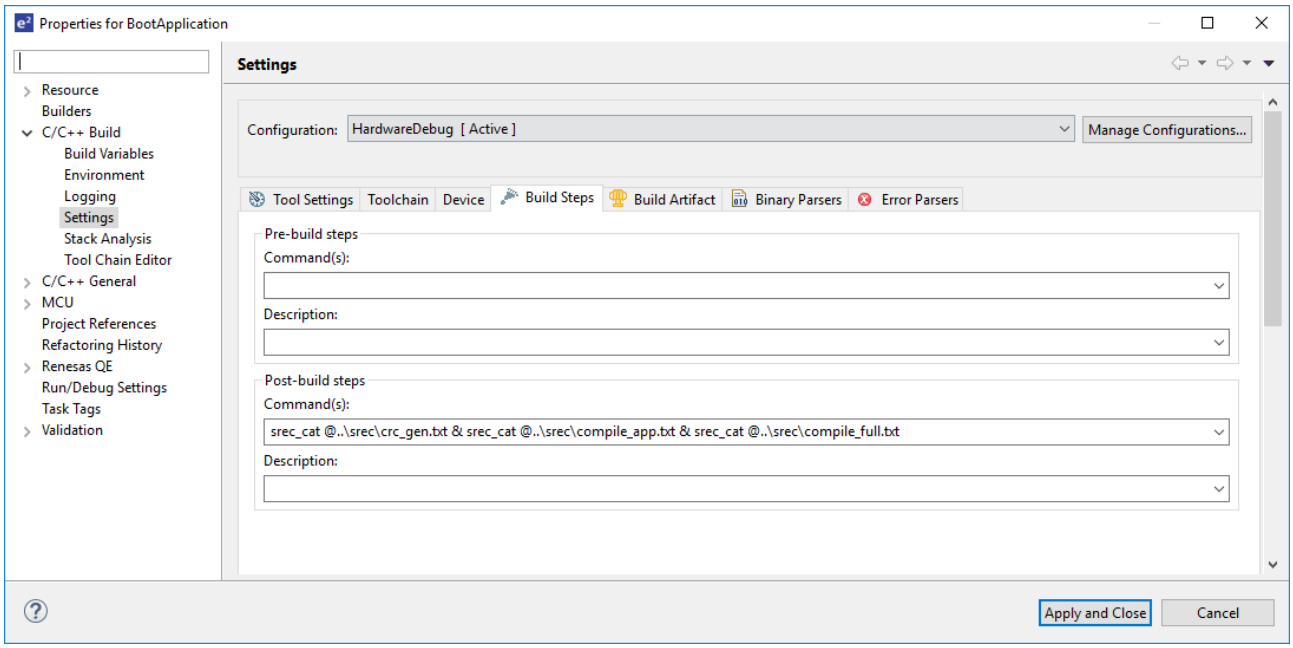

Figure 10

4.3 PC Application

The PC program is meant as a stand in for your method of transmitting the data. This example project was built with QT Creator and supplied open source under the GPL open source license.

4.3.1 Overview

This application drives the entire download process. Every message sent from the PC will receive a response message, please review ***Table 1*** and ***Table 2*** for specifics on what responses are sent. Once you click the "Begin Download" button this begins the process.

Every time a response has been received from the MCU the PC application knows that any process from the last message have been completed and it is ready for a new message. The logic for this can be followed in the function command_action() in the file mainwindow.cpp.

4.3.2 COM Port

This application uses the QSerialPort API to communicate to the MCU. Our USB to UART converter will translate it to UART for the MCU.

4.3.3 File Reader

This application uses the QFile API to read the new MCUs application file.

4.4 Srec Batch Files

The srec batch files are used to complete multiple post-build processes. Note that it is set that each one executes automatically, described in section 4.2.5 Post-Build Processes, when you build the bootApplication project. No extra steps are required to run these tasks.

Each one is described individually below:

4.4.1 CRC Generation

Of all the txt files this one should run first; the others require the crc.mot file generated here.

What this execution does:

1. After a build occurs this file accesses the file BootApplication.mot file in the HardwareDebug directory. It fills any unused locations with 0xFF, this is the same as what happens inside the RL78 as unused memory locations are read as 0xFF when read for an internal CRC check.
2. Once all the relevant areas have been filled (addresses 0x0-0xFFF, 0x2000-0x7FDFD), a CRC is performed on those areas, and the CRC is placed in a new file (crc.mot). Note that the CRC matches the internal General CRC of the RL78 and is placed in memory in a byte reversed order. This is so that when the General CRC is calculated of memory, it will evaluate to 0 once it reaches the last byte of the CRC.
3. When srec_cat execution has completed the file crc.mot should exist in the HardwareDebug folder. Inside it will be the CRC in reverse byte order at address 0x7FDFE.

Execution will give the warning that a CRC was calculated with a hole inside it, this is exactly the behavior needed so the warning can be ignored.

4.4.2 Full File Generation

The compile_full.txt file, when executed with srec_cat, generates an image to directly flash to your device.

Ensure that you have built the BootLoader project as the mot file from that is also accessed. In this example project it is assumed that both the BootLoader and BootApplication projects exist in the same directory. If they do not it will be necessary to modify the file to point to the BootLoader.mot file. Simply modify line 14 to point to the mot files location.

What this execution does:

1. After a build, and the execution of crc_gen.txt, this file will access the BootApplication.mot file and fill any unused locations with 0xFF exactly like the CRC generation file. Then it will ensure that no other code exists than other than what the application area (addresses 0x0-0xFFF, 0x2000-0x7FDFD).
2. It will also access the BootLoader.mot file generated from the Bootloader project. Shift it by 0x1000 bytes to fit into the ‘hole’ left in the application. And combine it with the application to make one large file to flash to the device.
3. After it has isolated the application image, and merged it with the shifted bootloader project, it will access the crc.mot file generated earlier and place the CRC contained in it at the end of the application.
4. When srec_cat execution has been completed the file JoinedBootAndApp.mot should exist in the HardwareDebug folder. It should now contain your entire application (0x0000-0x0FFF and 0x2000-0x7FDFD), the BootLoader project (0x1000-0x1FFF), and the CRC (0x7FDFD-7FDFF)

You should now be able to flash your device using the JoinedBootAndApp.mot file. This can be done either with E2 Studio itself or with Renesas Flash Programmer.

4.4.3 Application File Generation

The compile_app.txt file, when executed with srec_cat, generates the file used to download a new application version to your MCU. This will be used in conjunction with the PC application.

Since we do not require the use of the BootLoader in this image it is not necessary for execution of this file.

What this execution does:

1. After a build, and the execution of crc_gen.txt and compile_full.txt, this file will access the BootApplication.mot file and fill any unused locations with 0xFF exactly like the CRC generation file. Then it will ensure that no other code exists than other than what the application area (addresses 0x0-0xFFF, 0x2000-0x7FDFD).
2. After it has isolated the application image, it will access the crc.mot file generated earlier and place the CRC contained in it at the end of the application.
3. When srec_cat execution has been completed the file AppWithCRC.hex should exist in the HardwareDebug folder. It should now contain your entire application (0x0000-0x0FFF and 0x2000-0x7FDFD), and the CRC (0x7FDFD-7FDFF)

This file is a hex file instead of an srec/mot. This is so it is easier for the PC application to parse and download.

5. Running the Sample Program

Before running the sample program ensure you have the program srec_cat.exe added to your path.

5.1 Building the RL78 Projects

5.1.1 Importing the Projects

Open E2 Studio to a workspace of your choosing and click File > Import. Then select General > Existing Projects into Workspace as shown in ***Figure 11***.

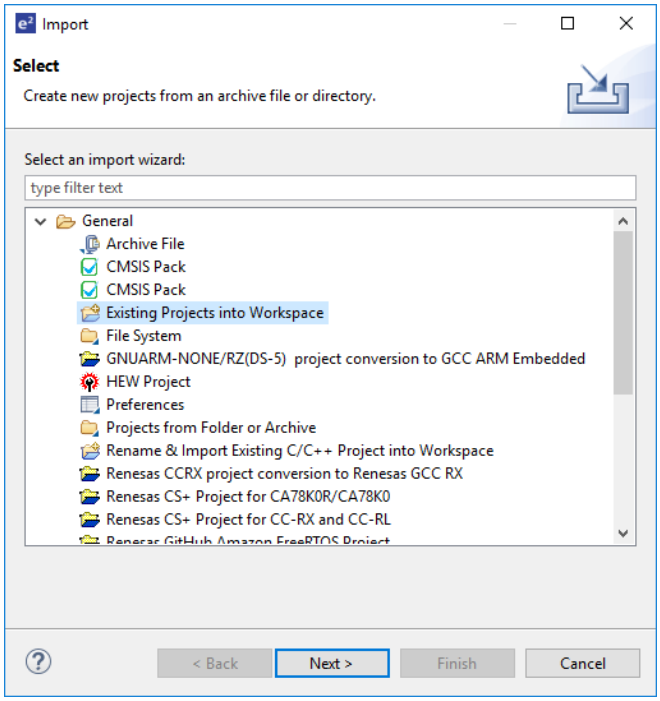

Figure 11

Click “Select archive file” and navigate to the RL78 Projects directory in this example project download. Select the BootApplication.zip file, ensure your screen looks like ***Figure 12*** before clicking finish. This will import the BootApplication project into your workspace.

Repeat this process for the BootLoader project and you should have both projects ready in your workspace.

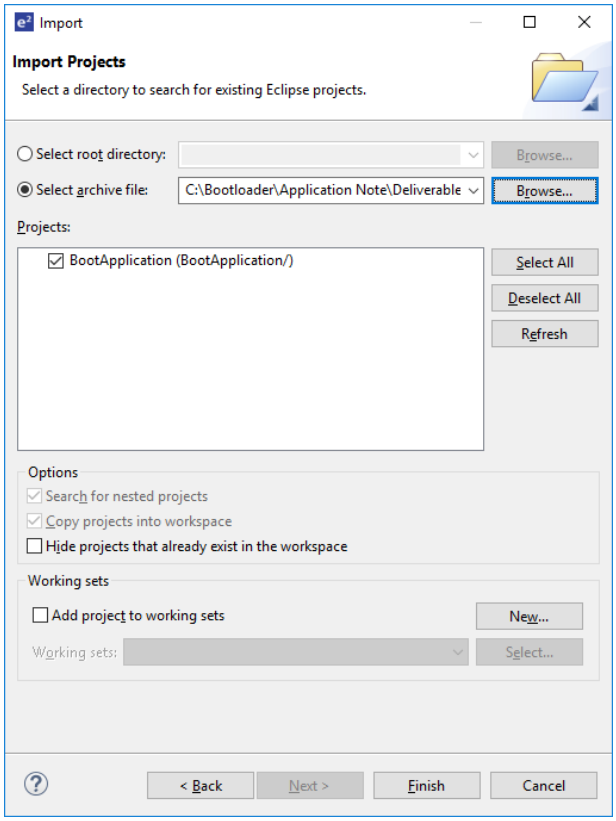

Figure 12

5.1.2 Building the BootLoader Project

Once you have both projects imported build the BootLoader project first. This is built first so that the application project can use its output file to combine the two images into one.

Once the bootloader project is built open the file HardwareDebug\BootLoader.map. Here it is important to verify two things:

1. All your text sections, from .text to FSL_BECD, exist in between addresses 0x00D8 and 0x0FFF.
2. There is nothing in the .const and .constf sections.

5.1.3 Building the BootApplication Project

Here we will be building two separate images of this project. One that turns on LED0 and one that turns on LED1.

(1) **Initial Image**

Our first image will be version 0x01010101 (the older version) and it will use LED0.

1. Go to the bootApplication projects main.c file, located in the src directory.
2. Line 44 should contain a #define for LED0, ensure this is uncommented. If it is, uncomment it now.
3. Lines 48-50 should have options to select a version number. Uncomment the version 0x01010101 and ensure all other options are commented out.

Once this is completed save the changes to the file and build the application. In your HardwareDebug folder you should see multiple files created. The file we care about here is JoinedBootAndApp.mot. We will use this file to flash the part with the initial image, this image will contain the application we just built as well as the bootloader project.

Connect your Fast Prototyping Board to your PC via a USB micro cable. Using Renesas Flash Programmer (RFP) download the JoinedBootAndApp.mot file generated to the RL78. Once this is completed ensure you disconnect the USB cable.

(2) **Downloadable Image**

The second image we will create will be version 0x01020304 (newer version) and will use LED1.

1. Go to the bootApplication projects main.c file, located in the src directory.
2. Comment out the #define on line 44, ensuring that this image will turn on LED1.
3. Lines 48-50 should have options to select a version number. Uncomment the version 0x01020304 and ensure all other options are commented out.

Again, once this is completed save the changes and build the project. This time we only care about the AppWithCRC.hex file. We will use this file in section 5.3 Running the PC Application.

That's it! All that is left to do now is set up the hardware to download the new application and run the PC program.

5.2 Setting up the hardware

Lets now connect the hardware so that we can download the new image. To prevent shorts, ensure power is disconnected before making any physical changes to the board.

5.2.1 Run mode

Short the pins of jumper EJ1 to place the device into run mode. EJ1 is located in the corner of the board, next to the USB port. When in run mode the device will not interface with an emulator at all and normal programming of the device is impossible.

5.2.2 UART Communication

Connect your USB to UART converter to the PMOD pins. Ensure the configuration of the wires is as below.

- The Converters RXD line is connected to PMOD1 pin 2
- The Converters TXD line is connected to PMOD1 pin 3
- The Converters Ground line is connected to PMOD1 pin 5

5.2.3 Power

Once the above hardware is set up you can re-connect your device to the micro-USB cable. This is where the RL78 will draw power from, and since EJ1 is shorted the on-board debug chip is held in reset.

When power is connected verify that ELED1 is not flashing, this means that the device is indeed in run mode. Also verify that LED0 has turned on, this notifies you that our code is running and ready for a download to occur.

At this point, ensure your USB-UART converter is connected to a COM port on your PC.

5.3 Running the PC Application

Run the BootServer.exe executable included with this example project. It can be found in the PC Application\Executable directory. You should see a screen like in ***Figure 13***.

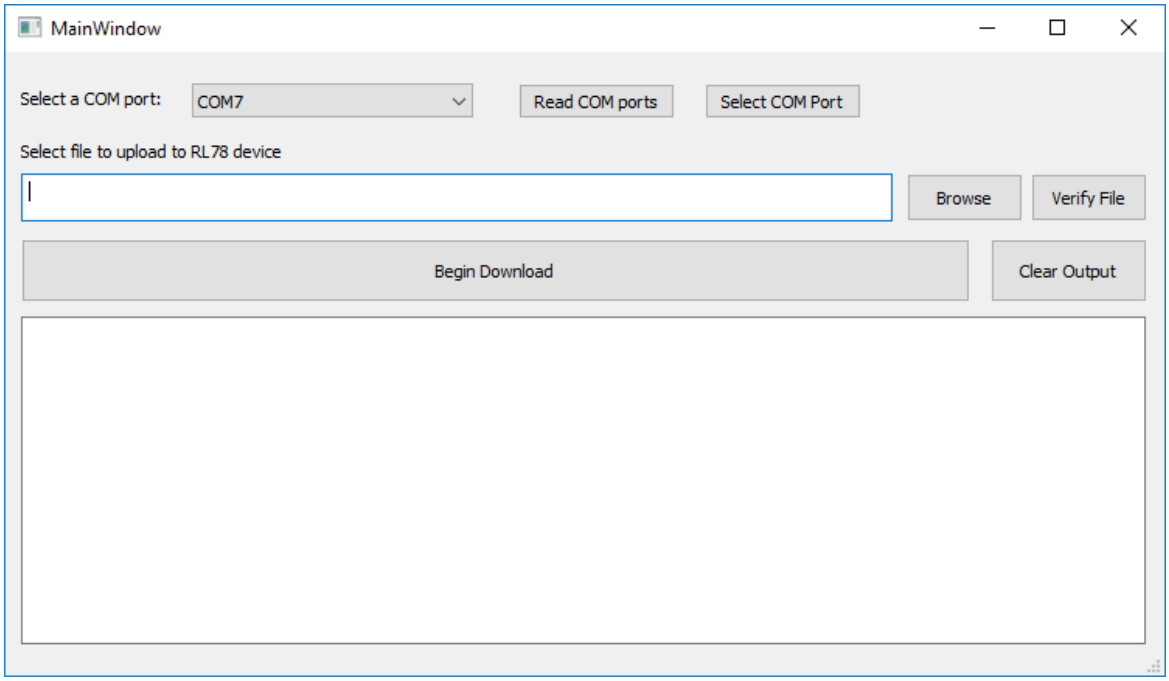

Figure 13

5.3.1 Selecting a COM port

Use the drop-down box to select the COM port connected to your USB to UART converter. If you are not sure which one to select please review your PCs device manager.

Once a COM port has been chosen you must hit the “Select COM Port” button. Once selected, a message will appear stating “Selected COM port: COMx”. If this button is not clicked communication with the board is impossible.

5.3.2 Selecting the file to download

Using the “Browse” button navigate to the AppWithCRC.hex file we created in section 5.1.3(2) Downloadable Image. You can now use the “Verify File” button to verify the PC application can calculate and match the CRC we generated with the srec tools during the build.

Verify that your screen now looks like in ***Figure 14***.

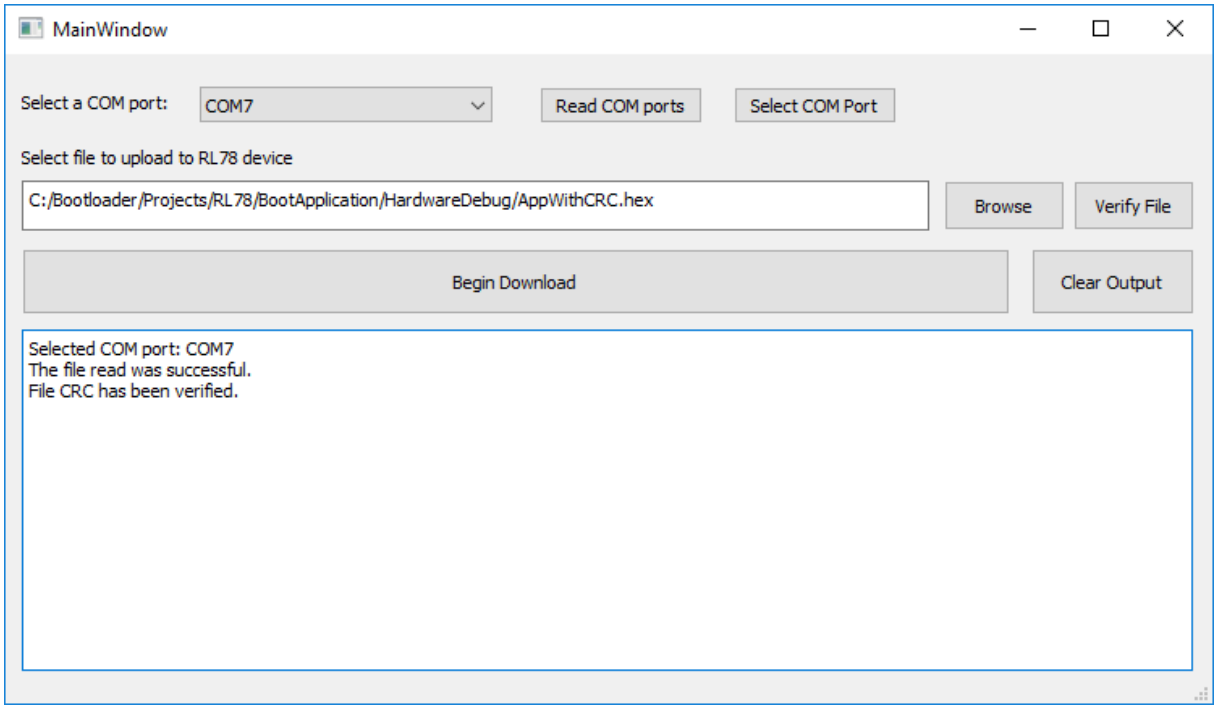

Figure 14

5.3.3 Download

Now click the “Begin Download” button. This will kickstart the download process and, assuming everything has been set up correctly, the download will begin. LED0 will turn off at this point, verifying that the BootLoader portion of the code is active.

You can view the download progress with the messages output to the message output box, including the packets being transmitted and the ACKs received. The download process may take a couple minutes to complete but should end on packet 8119.

When the download is complete you should see a message stating “Download Success!”. To finally verify that the download worked the device will automatically reset into the new application, turning on LED1. If you see LED1 the new image has been downloaded, verified, and taken action to be the active application.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	09/16/2020		Initial Release

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks