RX600 Series

The Flash Loader Project - SD Card Implementation

R01AN1535EU0100
Rev.1.00
March 5, 2013

Introduction

This application note is an extension of the default Flash Loader Project. As such, this document will only cover the modifications that were made to the default implementation. For a thorough explanation of the Flash Loader Project the default implementation's application note should be reviewed.

The Flash Loader Project is a flexible bootloader framework that is meant to be customized for each user's application. The default implementation uses asynchronous serial for communications and stores firmware updates into an external SPI flash. This application note will focus on an implementation where firmware updates are received from a SD card.

Target Device

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

- **RX621, RX62N Group**
- **RX631, RX63N Group**

Related Documents

- The Flash Loader Project (R01AN0287EU0300)
- Simple Flash API for RX (R01AN0544EU0240)

Contents

1. Overview

The default Flash Loader Project implementation downloads firmware updates over asynchronous serial and stores them into an external serial flash. After the image has been successfully downloaded and validated the MCU is reset at which point the Flash Loader Bootloader runs. The bootloader will detect the new firmware image in the external SPI flash and will program it into the MCU's User Application ROM space.

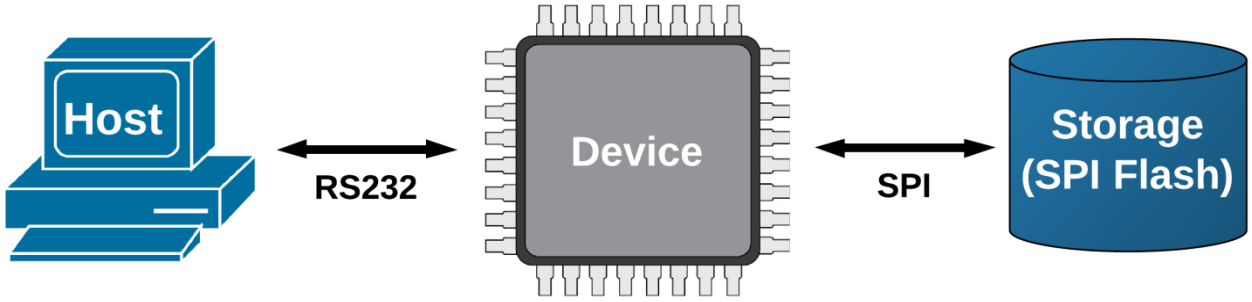


Figure 1-1 : Default Flash Loader Project

This application note focuses on a modified version of the Flash Loader Project where firmware images are obtained from a SD card. With this configuration the Device does not communicate with the Host directly. Instead the Host stores the firmware image directly into Flash Loader Storage (the SD card). The Device then reads the firmware image from Storage.

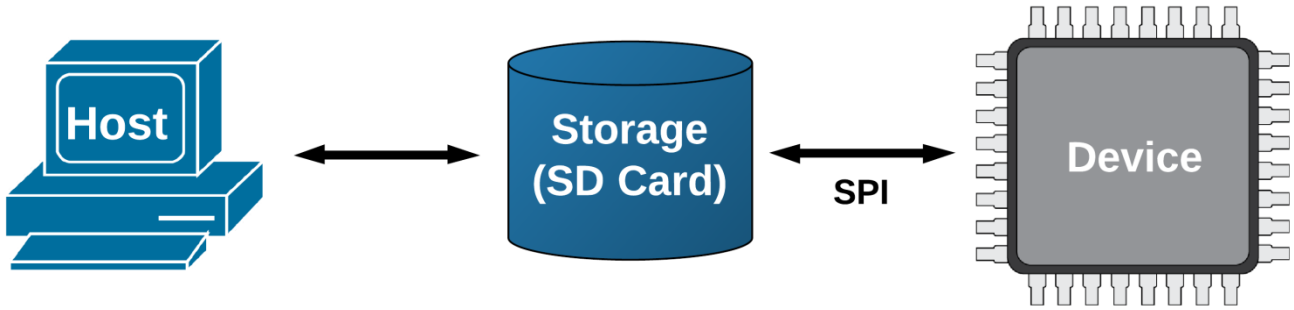


Figure 1-2 : Flash Loader Project – SD Card Implementation

The SD Card implementation is much simpler in regards to the Flash Loader code as two large parts are removed. The first is external communications with the host. Communicating with a host requires protocols to be followed with retries, error detection, and timeouts. Since the firmware image is being presented whole to the MCU on the SD card, these parts can be removed. The other part that can be (and is) removed is the ability for the Device to write to the Storage. With the default implementation, the MCU was required to write data to the external SPI flash as it was received from the Host. Since the firmware image is stored directly to the SD card by the Host, there is no reason for the Device to have the capability to write to the SD card. While the device drivers supplied with this project have the capabilities to write to the SD card, they are not used. This was done to save code space and for simplicity. Since the device drivers do support writes to the SD card, if the user wishes to add these features, they may do so easily.

The largest modification required for this implementation was to move from using the simple default Flash Loader file system (Store Manager) to using the FAT file system. By moving to the FAT file system users can simply copy their Load Image files directly to their SD card.

2. Modifications Made

This section will detail the modifications that were made to the default Flash Loader Project.

2.1 Host Side

No changes were required on the host side. The *r_fl_mot_converter.py* tool is still used to generate Flash Loader firmware files which can be directly copied to the user's SD card. Since Host to Device communications are not used in this implementation, the *r_fl_serial_flash_loader.py* tool has been removed.

2.2 Device Side

Several files were removed for this implementation since they were not needed. These files include:

File Removed	Reason
r_fl_comm.h	No Device to Host communications with this implementation.
r_fl_comm_uart.c	No Device to Host communications with this implementation.
r_fl_memory_spi_flash.c	SD card is now used for Storage.

2-1 : Files removed

Files that were modified or added are discussed below.

2.2.1 r_flash_loader_rx_config.h

The configuration file was stripped of all unneeded macros; which left one. The one macro that was left was FL_CFG_DATA_BLOCK_MAX_BYTES. The Flash Loader Project still needs a RAM buffer large enough to store a full data block from the Load Image file. If the user wishes to save RAM in their project they can make this macro much smaller. If this is done the user will also need to make sure that they change the maximum data block size when creating the Load Image with the *r_fl_mot_converter.py* tool. If the maximum data block size of the Load Image file is larger than the RAM buffer on the MCU (as defined by this macro) then the MCU cannot use the file.

2.2.2 r_fl_bootloader.c

There are two parts of the bootloader. The first part is the code that checks for new Load Images, checks for an application already in the User Application space, and decides what to do based on the current state of the system. All of this code is located in the main() function. The logic behind this code has remained unchanged. The only thing that was added was testing to see if a SD card is present. The default Flash Loader Project used an external SPI flash for Storage and it was always assumed to be present. If a SD card is not detected then the bootloader will check for an image already in the User Application space. If a valid image is found in ROM then it is executed; otherwise the bootloader starts the Flash Loader state machine and waits for a SD card to be inserted.

The other part of the bootloader is the code that programs a Load File into the MCU's ROM. This code changed slightly because this implementation uses the FAT file system. This means that data is now being read from a file using the file system code instead of being read directly from a SPI flash. This is highlighted by the change from using the fl_mem_read() function to using the new fl_read_file_data() function. Once the data has been read from Storage, the processing and writing of the data is unchanged.

2.2.3 r_fl_downloader.c

This file implements the Flash Loader state machine. The state machine code in the default Flash Loader project is devoted to implementing the communication protocols. Since Host to Device communications have been removed in this implementation, there is only one state the MCU can be in. The processing done in this state is shown in Appendix A.

2.2.4 r_fl_store_manager.c

This file has been heavily modified since the FAT file system is used with this implementation. This file is simpler for three reasons:

1) The file system is now handled by r_fatfs module which uses the ELM ChaN FatFs code. This means that all of the file system code is handled externally to the Flash Loader code which removes complexity from this file.
2) The ability to write and erase Storage has been removed.
3) Host to Device communications have been removed. In the default implementation this file has several functions for handling communication retries.

Below is a table of functions that were added and removed.

Functions Removed	Functions Added
fl_store_init	fl_file_system_init
fl_store_image_header	fl_read_file_data
fl_start_erase_load_block	fl_move_to_data
fl_continue_erase_load_block	
fl_store_retry_init	
fl_store_retry_continue	
fl_store_retry_get_block	
fl_store_block_init	
fl_store_block_continue	
fl_store_finish	

Table 2-2 : Modifications to r_fl_store_manager.c

2.2.5 r_fl_types.h

The macro FL_CFG_MEM_NUM_LOAD_IMAGES which is located in the configuration file for the default Flash Loader Project has been moved to this file for this implementation. It was moved out of the configuration file because it no longer needs to be modified by the user; it should always be set to ‘1’. This macro could have been removed all together and replaced with a hard coded ‘1’ but that would have required more changes to the code.

2.2.6 r_fl_memory_sdcard.c

This file was added to implement the Flash Loader memory layer. Since separate file system code is now used, this file is really only used for initializing SD card communications and returning the status. All memory reads, writes, and erases are handled in the r_fatfs module.

3. API Information

This Middleware API follows the Renesas API naming standards.

3.1 Hardware Requirements

This middleware requires your MCU support the following features:

- Timer able to generate timer tick
- Ability to rewrite memory area where application code is stored
- Ability to communicate with a SD card (RSPI used for RX devices)

3.2 Hardware Resource Requirements

This section details the hardware peripherals that this middleware requires. Unless explicitly stated, these resources must be reserved for the middleware and the user cannot use them.

3.2.1 CRC

CRCs are used for checking data blocks and firmware images.

3.2.2 Flash Control Unit (FCU)

The FCU takes care of programming and erasing internal memory. The bootloader uses the FCU but once the user's application has started they can use it as well.

3.2.3 RSPI

One RSPI channel is used for communicating with a SD card.

3.2.4 WDT

The watchdog timer is used to reset the MCU when a new firmware image has been downloaded.

3.2.5 CMT (compare match timer)

This timer is used to generate a timer tick that drives the state machine. The user is responsible for calling the state machine so they can use any timer they wish. The CMT is listed here because it is used in the demo and in the bootloader.

3.3 Software Requirements

This middleware depends on the following packages.

3.3.1 r_crc_rx

This package is used for generating CRC codes.

3.3.2 r_flash_api_rx

This package is used for rewriting the MCU's internal ROM and data flash. Only the Flash Loader Bootloader uses this package.

3.3.3 r_glyph

This package is used for controlling the LCD on RDK boards (i.e. RDKRX62N, RDKRX63N).

3.3.4 r_rspi_rx

This package is used for communicating with a SD card.

3.3.5 r_cmt_rx

This package is used to control a CMT channel which generates a timer tick that drives the state machine.

3.3.6 r_delay

This package is used to implement delays. This is currently used by the r_glyph and r_mmc modules.

3.3.7 r_fatfs

This package is used for FAT file system support. The FAT file system code is 3rd party and is written by ELM ChaN. The code and documentation can be found here:

http://elm-chan.org/fsw/ff/00index_e.html

The r_fatfs package contains the unmodified FAT file system code. The only thing the r_fatfs package adds is code that implements the necessary Disk I/O functions as specified by the FAT file system code. These low-level Disk I/O functions use functions from the r_mmc package.

3.3.8 r_mmc

This package contains a basic MMC driver implementation. It uses the r_rspi_rx module for communicating with MMC/SD cards.

3.4 Limitations

The r_fatfs package currently only supports implementing a FAT file system on a SD card. This could be modified to add support for other storage mediums like an external SPI flash.

3.5 Supported Toolchains

This middleware is tested and working with the following toolchains:

- Renesas RX Toolchain v1.02

3.6 Header Files

All API calls are accessed by including a single file: *r_flash_loader_rx_if.h*. This header file is supplied with this application note's project code.

3.7 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

3.8 Configuration Overview

The core Flash Loader code is configured through the *r_flash_loader_rx_config.h* header file. The configuration options available in this header file are shown in the table below.

Configuration Options in *r_flash_loader_rx_config.h*	
FL_CFG_DATA_BLOCK_MAX_BYTES	Maximum block data size supported. sizeof(fl_block_header_t) is added to this #define when declaring the receive buffer. This is done because the r_fl_mot_converter.py program accepts a parameter to set the data block size. That parameter does not take into account the block header size so neither does this one. The value of this macro directly sets the size of a RAM buffer that is used. Since Device to Host communications are not used in this implementation, setting the block size in regards to how dependable your communications medium is not needed. The user can set this value to whatever value they want; just make sure to also set the maximum block size of the Load Image accordingly.

Table 3-1 : Flash Loader Configuration Options

3.9 API Data Structures

None.

3.10 Return Values

Not applicable.

3.11 Adding Middleware to Your Project

This section details how to add the Flash Loader code to your own project. The Flash Loader project is made up of two projects: a bootloader and a user application.

3.11.1 Flash Loader Bootloader

1. Copy the ‘r_flash_loader_rx’ directory (packaged with this application note) to your project directory.
2. Add the following source files to your project.
    a. src\r_fl_bootloader.c
    b. src\r_fl_downloader.c
    c. src\r_fl_store_manager.c
    d. src\r_fl_utilities.c
    e. src\memory\r_fl_memory_sdcard.c
3. Add an include path to the 'r_flash_loader' directory.
4. Add an include path to the 'r_flash_loader\src' directory.
5. Copy r_flash_loader_config_reference.h from 'ref' directory to your desired location and rename to r_flash_loader_config.h.
6. Configure middleware through r_flash_loader_config.h.
7. If you are placing the bootloader in the User Boot area then make sure to:
    a. Configure your linker to place the code in the correct area.
    b. Configure your BSP to choose User Boot Mode. This is done by configuring *r_bsp_config.h* if you are using the r_bsp package.
    c. Read the section titled “Calling Flash API from User Boot Area” from the default Flash Loader Project’s application note for information about using the Flash API from the User Boot Area.

3.11.2 Flash Loader User Application

1. Copy the ‘r_flash_loader_rx’ directory (packaged with this application note) to your project directory.
2. Add the following source files to your project.
    a. src\r_fl_app_header.c
    b. src\r_fl_downloader.c
    c. src\r_fl_store_manager.c
    d. src\r_fl_utilities.c
    e. src\memory\r_fl_memory_sdcard.c
3. Add an include path to the 'r_flash_loader' directory.
4. Add an include path to the 'r_flash_loader\src' directory.
5. Copy r_flash_loader_config_reference.h from 'ref' directory to your desired location and rename to r_flash_loader_config.h.
6. Configure middleware through r_flash_loader_config.h.
7. Add a #include for r_flash_loader_rx_if.h to files that need to use this package.

4. API Functions

The same API functions as in the default Flash Loader Project are provided. The API functions are discussed here to provide information on what has changed inside of the functions.

4.1 Summary

The following functions are included in this API:

Function	Description
R_FL_DownloaderInit()	Initializes Flash Loader Downloader
R_FL_StateMachine()	Calls the Flash Loader State Machine
R_FL_GetVersion()	Returns the current version of this API

4.2 R_FL_DownloaderInit

Initializes everything needed to run the Flash Loader Downloader.

Format

void R_FL_DownloaderInit(void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_flash_loader_if.h”
Implemented in file “r_fl_downloader.c”

Description

Initializes pointer that points to current application’s load image header.

Reentrant

Yes

Example

```
/* Initialize the Flash Loader code. */
R_FL_DownloaderInit();

/* Now start timer tick that will trigger Flash Loader state machine. */
...
```

4.3 R_FL_StateMachine

Calls the state machine that runs the Flash Loader Downloader.

Format

```
void R_FL_StateMachine(void);
```

Parameters

None

Return Values

None

Properties

Prototyped in file “r_flash_loader_if.h”
Implemented in file “r_fl_downloader.c”

Description

This function implements the Flash Loader state machine. Since Device to Host communications have been removed in this implementation, there is only one state. This state is shown in Appendix A.

Reentrant

No, but the state machine does protect against multiple calls (i.e. only one process is allowed in at any given time)

Example

```
bool g_sm_process;

void main(void)
{
    uint32_t cmt_channel;

    /* Initialize state machine process flag. */
    g_sm_process = false;

    /* Initialize the Flash Loader code. */
    R_FL_DownloaderInit();

    /* Create periodic timer to call Flash Loader state machine. */
    R_CMT_CreatePeriodic(USER_APP_CMT_FREQUENCY,
                         fl_trigger_sm,
                         &cmt_channel);

    while (1)
    {
        /* Call state machine after flag has been set. */
        if (true == g_sm_process)
        {
            /* Trigger state machine. */
            R_FL_StateMachine();

            g_sm_process = false;
        }

        /* Do other work. */
    }
}

/* CONTINUED ON NEXT PAGE */
```

```
/* CMT Interrupt Service Routine that will set a flag which will alert the
   main() loop that the Flash Loader state machine should be called. */
static void fl_trigger_sm (void * pdata)
{
    /* Create periodic timer to call Flash Loader state machine. */
    g_sm_process = true;
}
```

Special Notes:

There is only 1 state which means this function will perform the same routine every time it is called. This function will first test to see if a SD card is present. If a SD card is found then it will then look for a Load Image. If the user has a SD card inserted that has many files on it, but no Load Images, then this function will 'waste' time each time it is called. Therefore the user could optimize their system by only calling this function each time a SD card is inserted. If the function is called and it finds out that no Load Images are present on the SD card then further attempts are futile until a new SD card is inserted.

4.4 R_FL_GetVersion

Returns the current version of the module.

Format

```
uint32_t R_FL_GetVersion(void);
```

Parameters

None.

Return Values

Version of the Flash Loader project.

Properties

Prototyped in file “r_flash_loader_rx_if.h”
Implemented in file “r_fl_utilities.c”

Description

This function will return the version of the currently installed Flash Loader code. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number. For example, Version 4.25 would be returned as 0x00040019.

Reentrant

Yes.

Example

```
uint32_t cur_version;

/* Get version of installed Flash Loader. */
cur_version = R_FL_GetVersion();

/* Check to make sure version is new enough for this application’s use. */
if (MIN_VERSION > cur_version)
{
    /* This Flash Loader version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ....
}
```

Special Notes:

- This function is specified to be an inline function in *r_fl_utilities.c.*

5. Demo Projects

This application note contains demo projects for both HEW and E2Studio. For HEW the demo is packaged as an entire HEW workspace which contains projects for each Renesas development board. For E2Studio, each Renesas development board has its own zipped project that can be imported into an existing E2Studio workspace. This version includes projects for the following boards:

- RDKRX63N
- RDKRX62N

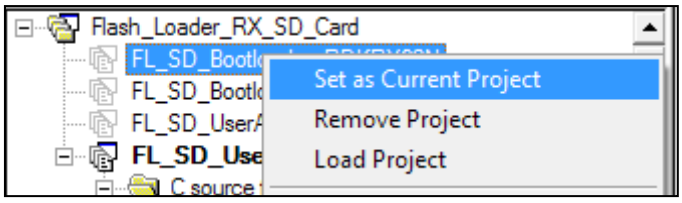
There are two projects per supported board (e.g. 2 for RDKRX62N, 2 for RDKRX63N). These projects are:

- **FL_SD_Bootloader_*board* –** This is the Flash Loader Bootloader project prebuilt for the board referenced by *board*. If the user wants to modify the bootloader then they can edit this project. When the user is ready to program in the bootloader to their MCU then they can get the S-Record file from this project's build directory.
- **FL_SD_UserApp_*board* –** This is a pre-setup Flash Loader user application for the board referenced by *board*. It does nothing but run the Flash Loader state machine and show a message on the LCD. The purpose of this project is to give the user a shell project that they can use if they wish. Everything is preconfigured for Flash Loader use so the user can also use it as a reference for their own Flash Loader project.

5.1 HEW Workspace

The HEW workspace that comes packaged with this application note has a project for each supported Renesas development board. The only code that changes between these projects is the board support code that is used along with the demo and Flash Loader code. To choose a project follow these steps:

1. Open the HEW workspace
2. Right-click on the project you wish to load in the navigation pane (by default on left) and click 'Set as Current Project'.



3. The Flash Loader code and demo workspace use the r_bsp package for startup code, board support code, and for getting MCU information. The r_bsp package is easily configured through the *platform.h* header file which is located in the r_bsp folder. To configure the r_bsp package, open up *platform.h* and uncomment the #include for the board you are using. For example, to run the demo on a RDKRX63N board, the user would uncomment the #include for './board/rdkrx63n/r_bsp.h' and make sure all other board #includes are commented out.

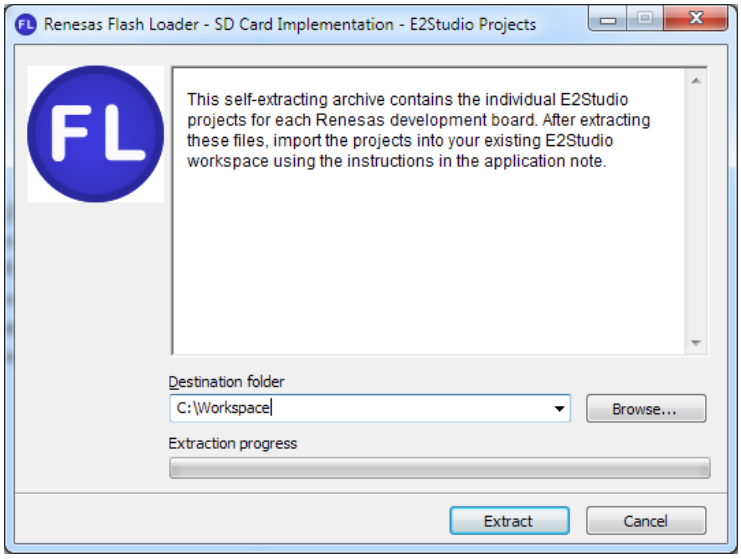
```
/* RSKRX630 */
//#include "./board/rskrx630/r_bsp.h"

/* RSKRX63N */
//#include "./board/rskrx63n/r_bsp.h"

/* RSKRX63T_64PIN */
//#include "./board/rskrx63t_64pin/r_bsp.h"

/* RSKRX63T_144PIN */
//#include "./board/rskrx63t_144pin/r_bsp.h"

/* RDKRX63N */
#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"
```

4. You can now build and execute the demo.

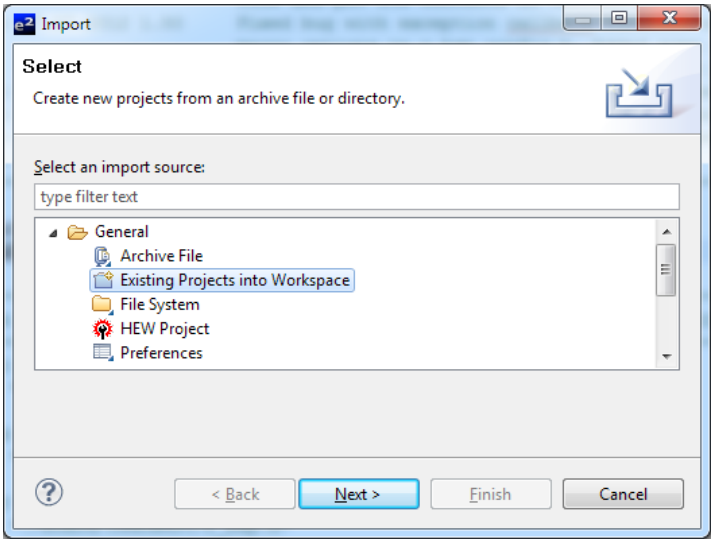
5.2 E2Studio Projects

E2Studio handles workspaces differently than HEW and therefore projects must be imported into your existing E2Studio workspace. In order to use the demo for your development board follow these steps:

1. The E2Studio projects are distributed as a self-extracting archive with this application note. The first thing that will need to be done is to extract this archive. Double click on the self-extracting archive file (should be *.exe under Workspace\e2studio directory).
2. Choose where to extract the projects and click Extract.

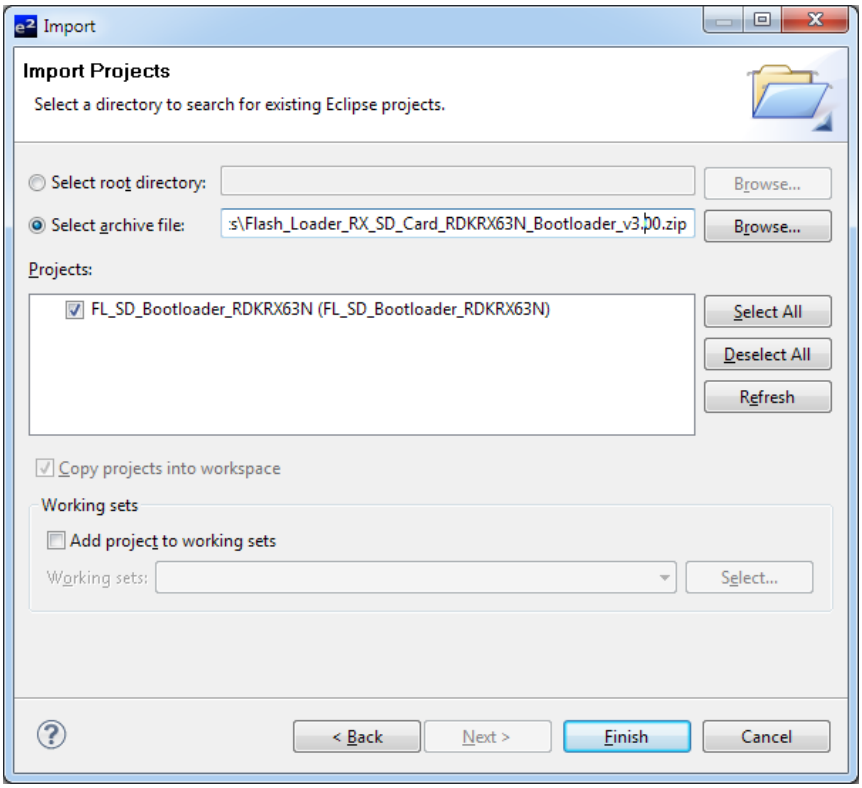


3. Open your E2Studio workspace
4. Click File >> Import
5. Choose General >> Existing Projects into Workspace and click Next.



6. Click ‘Select archive file’ and click browse.
7. Browse to the directory where you extracted the E2Studio projects and choose the zip file for your development board.

8. Check the box next to the project you wish to import and click Finish. In this screenshot the RDKRX63N project is being imported.



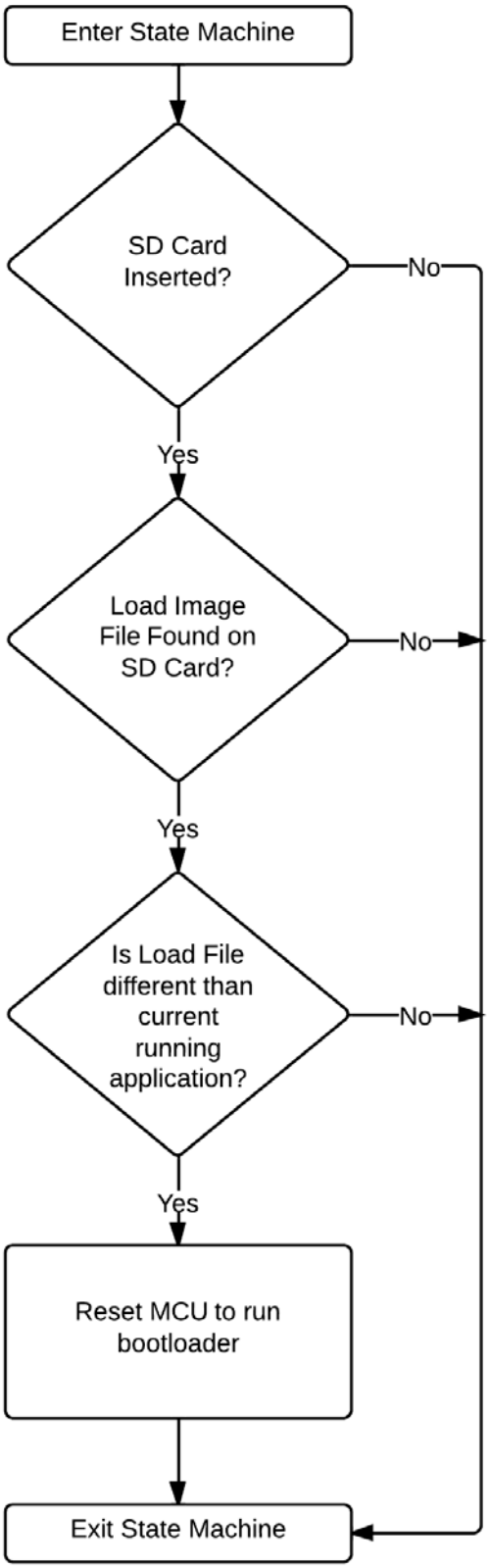
5. The Flash Loader code and demo workspace use the r_bsp package for startup code, board support code, and for getting MCU information. The r_bsp package is easily configured through the *platform.h* header file which is located in the r_bsp folder. To configure the r_bsp package, open up *platform.h* and uncomment the #include for the board you are using. For example, to run the demo on a RDKRX63N board, the user would uncomment the #include for ‘./board/rdkrx63n/r_bsp.h’ and make sure all other board #includes are commented out.

```
/* RSKRX630 */
//#include "./board/rskrx630/r_bsp.h"

/* RSKRX63N */
//#include "./board/rskrx63n/r_bsp.h"

/* RSKRX63T_64PIN */
//#include "./board/rskrx63t_64pin/r_bsp.h"

/* RSKRX63T_144PIN */
//#include "./board/rskrx63t_144pin/r_bsp.h"

/* RDKRX63N */
#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"
```

6. You can now build and execute the demo.

Appendix A : Flash Loader State Machine Diagram

This flowchart shows the execution flow of the Flash Loader state machine.



Website and Support

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

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

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Mar.07.13	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. Handling of Unused Pins

   Handle unused pins in accord 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2