R01AN1869EU0110
Rev 1.10
Nov 25, 2014

RX111

CDC Flashloader - A Complete Solution for Programming a Device over USB

Introduction

The USB CDC Flashloader solution lets a user easily reimage the firmware of a device in-field, using a PC and a USB connection.

The solution consists of three parts. 1) A USB *host downloader GUI* for downloading the image file from a PC, 2) P*CDC Flashloader*, an on-chip program that receives and writes an image to user memory, and 3) the user application program *UserApp*, the target board application that is to be downloaded using 1 and 2.

Choosing a USB class for Flashloader

One may easily be led to believe that CDC can pack much more data per USB frame since it uses Bulk transfers. With 12 Mbit/s for Full Speed USB this means a theoretical speed of 12 Mbit/s (~1 MB/s) if all bandwidth is available for Bulk transfers. When studied closer however, these numbers are for *total* Bulk throughput for *all* endpoints. For just *one* endpoint the data is limited to 64 bytes per frame and per endpoint. (A PC driver will most likely not pack multiple packets into one frame for the same endpoint.) This means that for a typical application using one endpoint to channel data the speed is only max 64000 bits/sec. One way to bypass this is obviously to add multiple endpoints and spread the download image among them to achieve a speed of *nr endpoints x packet size*. This is not done in this implementation. The PC GUI downloader *CDC_USB_Downloader_Renesas.exe* sends one MOT-line at a time, approximately 20 bytes, and *CDC_USB_Downloader_Renesas_double_line.exe* sends two lines at a time if possible.

Here are reasons for using the CDC USB class, in particular in comparison with the commonly used HID class.

- HID reports are more complex compared with “straightforward” streamed data.
- Most host PC programming platforms include an API for USB CDC class, but not for HID.
- Using http://www.signal11.us/oss/hidapi may be an option for an alternative HID-based design.

Target Device

RX100 based MCUs, RSK111 in particular is used in this demo with respect to LEDs and board switches. The code should be very easy to port to any RX100 USB device by downloading the latest r_bsp FIT package from the www.renesas.com SW Library.

Content

1. e² studio

The source code is available at the Renesas SW Library download site. It is on the same search result line as the application note, under the *C source* link.

Here is how to import the accompanying project source code to e² studio. Add the archive to e2 studio by following New or Existing Workspace below.

New workspace

- Create an empty folder, where you want the workspace.
- Start e2 studio, and point to above folder when e2 studio asks what workspace to open.
- Click Workbench icon (bottom right in blue intro-screen).
- Continue with “Existing workspace” below.

Existing workspace

- Select Import.
- Select General => Existing Projects into workspace. ("Create new projects from an archive file or directory.")
  - **Alt 1.** Select the root directory of the project, that is, the folder containing the “.project” file.

    Make sure checkbox "Copy project to workspace" is checked.
  - **Alt. 2**

    Select the archive zip-file.
- You have now imported this project into the workspace. You can go ahead and import other projects into the same workspace... Follow below for importing to existing workspace.

Run the code

- Build with Cntrl+B.
- Create a debug session, download and run the code.

2. Description and Usage

The CDC Flashloader solution contains three different parts which are described separately.

1. **PCDC Flashloader**. The MCU firmware that communicates with the PC host and rewrites the RX111 with the new user application UserApp.
2. **USB Host PC downloader**. A QT based host download GUI for Windows.
3. **UserApp**. The to-be-downloaded program for the RX111. This is by default a sample application that blinks the LEDs and reads the three board switches to alter the blink pattern. It can also boot into PCDC Flashloader as discussed in 2.3.

2.1 PCDC Flashloader

The PCDC Flashloader project for the RX111 is the on-chip program that receives and writes the image to user memory. CDC Flashloader will flash UserApp area when the host GUI downloads the image. After flashing, PCDC Flashloader will reboot into itself.

CDC Flashloader itself resides in the highest address section of memory.

Downloading PCDC Flashloader

1. To enable USB Function mode on the RSK111, Jumper J12 must be set to the rightmost position, towards the USB mini B connector.
2. In e² studio, select Debug Configuration from the menu bar to download the PCDC Flashloader binary to the target board.

Please consult the RSK Quick Start Guide or equivalent that comes with the board for how to connect and download a program.

- There may be debug configurations ready to use. Note that any existing Debug Configurations assume that 5V is supplied to the board separately.
- If there are none, or you are not using an E1 or E20, create a configuration according to the guide for your board. The frequency for the RSK111 must be set to 16 MHz.

3. With PCDC Flashloader running on the board, plug a *USB A-Mini B* cable into the PC, and the board's lower right USB port (there are two USB ports on the board).
4. Windows will now try to enumerate the board, but will be unable to do so the very first time the board is plugged in. An INF file for PCDC Flashloader is needed for Windows to access the new USB COM port that PCDC Flashloader makes available. The INF file is archived and named *PCDC_FlashLoader file cdc_inf* and comes with the source code. *Windows' Found New HW Wizard should pop up when PCDC Flashloader is connected and running. If not, open Windows' Device Manager and locate the new USB device. You may have to right-click on an 'Unknown Device' to help Windows couple the device with the correct Windows driver (typically USBSER.SYS). Select "Update Driver Software", then browse manually to the supplied INF-file.*
5. After a correct installation with Windows' Found New HW Wizard, the device can be seen as shown in Figure 1. The COM port's description string is "Renesas CDC USB Flashloader".

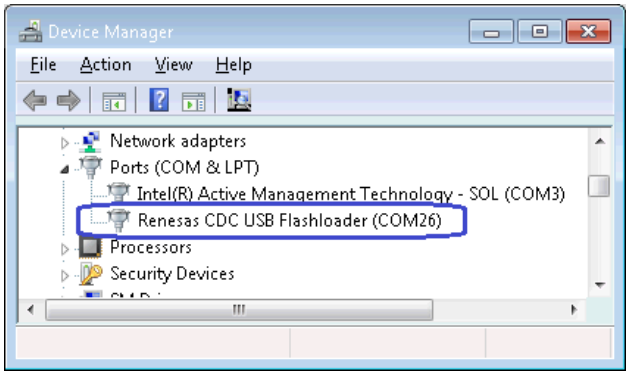


Figure 1. PCDC Flashloader is available for use by Windows after the user directs Window's Found New HW Wizard to the supplied INF-file*. If the device does not appear after PCDC Flashloader has been downloaded and is running, press the Reset button on the target board.

*Note that the VID of PCDC Flashloader must later be changed by the product designer.

2.2 PC Host Flashloader GUI

1. To run the host PC code, open

   ```
   ..\PCDC_FlashLoader_PC_build-Qt_5_1_1_MinGW_32b\release\CDC_USB_Downloader_Renesas.exe.
   ```

   The GUI should start. See Figure 2.
2. Select MCU **R5F51115**.
3. With the target running PCDC Flashloader (check Windows Device Mgr.), press **Search**.
4. Select the **COM port** that produces the text **Renesas CDC USB Flashloader** in the output text field.
5. Select a UserApp MOT-file to download to the board. (It must first be built with e^2 studio). Press **Select** in the GUI. And browse to e.g. *../CDC_Flashloader/UserApp/HardwareDebug/UserApp.mot.*
6. Press **Program** to flash.
7. If the GUI output window says "**send_image()=OK**." Flashing was successful.

Figure 2. When the UserApp image is flashed successfully, "*send_image() = OK*" is printed in the GUI text output.

8. At the conclusion of programming, PCDC Flashloader reboots and will jump to the new UserApp image **if the UserApp header's security code matches the expected value**.
9. To verify that UserApp is running properly, the LEDs should be blinking and pressing SW1-3 should alter the blink pattern.

2.3 UserApp

UserApp is the user program that is to be downloaded to the RX111. Downloading is accomplished by using the other two parts of the solution – the PC host downloader while PCDC Flashloader is executing.

The default UserApp blinks LEDs in a pattern. The pattern is altered by pressing the switches on the RSK.

Executing UserApp

If the UserApp header contains the expected security code, PCDC Flashloader will start execution at *UserAppStart* which resides at the lowest address of flash memory. For a 128 kB device this is at 0xFFFE0000. See 6.2 for how UserApp is validated by PCDC Flashloader and entered into.

Debugging UserApp Standalone

UserApp can be executed on its own on the target, without PCDC Flashloader resident in memory. See 6.5.

Switching Execution from UserApp to PCDC Flashloader

- If the default UserApp is NOT downloaded, just press the Reset button on the RSK to execute PCDC Flashloader. (Flashloader must first be downloaded. See 2.1).
- If the default UserApp is downloaded and is executing, pressing SW1+3 makes PCDC Flashloader run without booting to UserApp.

3. Memory Layout

PCDC Flashloader fits within 16 kB of flash with optimization for size level 2 using the RXC compiler. The exact memory sizes for PCDC Flashloader and UserApp are found in the MAP-file in the project build directories. Here we will discuss memory architectural usage aspects.

3.1 RAM

All RAM is available for both PCDC Flashloader and UserApp. RAM is always re-initialized and never shared between the two projects. They execute at different times and never simultaneously. Execution is always restarted when going from one to the other, and memory always reinitialized.

3.2 ROM (Flash)

UserApp will occupy the lowest memory region and PCDC Flashloader the highest.

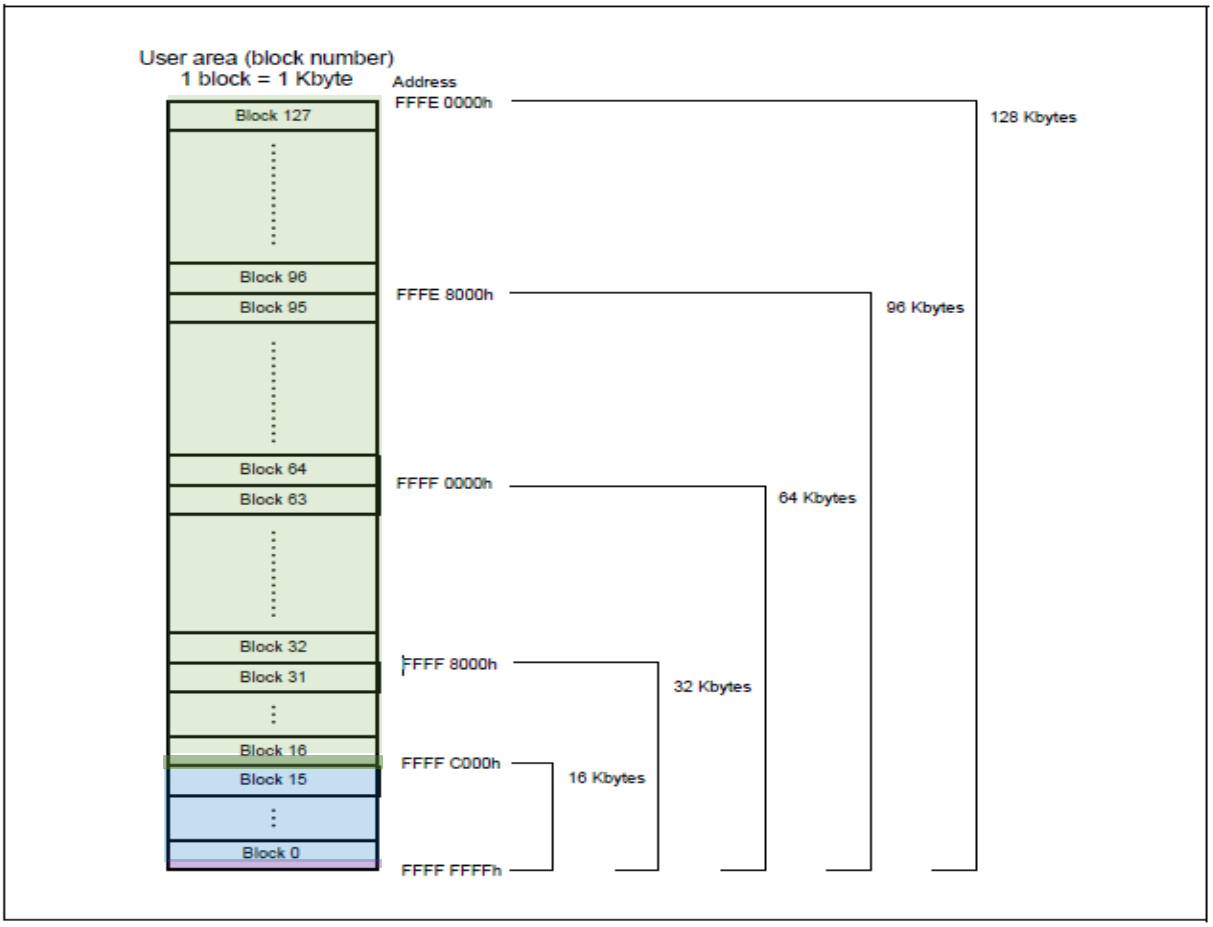


Figure 35.1 ROM Area and Block Configuration

Table 35.2 Correspondence Between User Area Capacity and Addresses

User Area Capacity	Address
128 Kbytes	FFFE 0000h to FFFF FFFFh
96 Kbytes	FFFE 8000h to FFFF FFFFh
64 Kbytes	FFFF 0000h to FFFF FFFFh
32 Kbytes	FFFF 8000h to FFFF FFFFh
16 Kbytes	FFFF C000h to FFFF FFFFh

Figure 3. Flash memory (ROM) disposition. UserApp is in green, at lower memory. Flashloader is shown in blue, and resides at the highest memory addresses. Block numbers and corresponding addresses can be seen in section 12.

UserApp's starting execution point *UserAppStart()* is at 0xFFFE0000, which is also the lowest possible address for an RX100 device. UserApp occupies *all* memory up to PCDC Flashloader. The *UserAppStart* address is currently fixed. It will not move even if the UserApp project's code is changed and recompiled.

PCDC Flashloader's start execution point, and also its lowest address, is set at 0xFFFFC000. It occupies *all* higher memory (blue in Figure 3). This is with RXC compiler optimization level 2 for size.

3.3 Moving the Boundary Flashloader / UserApp

The boundary address between PCDC Flashloader vs. UserApp can be changed but is unlikely to be necessary, nor is it encouraged. Should this be warranted use these guidelines. One reason to move the boundary could be that one wishes to remove compiler optimization of PCDC Flashloader to be able to single step through the code. But to follow the code's actions, there already exists a means to view the action of PCDC Flashloader; namely by using the Debug Console. See 9.

Moving UserApp

- In the Linker -> Section settings of the e^2 studio project, change the address of the *UserAppStartSect* section. This is where UserApp starts executing, and is also the lowest memory address available.
- The block numbers for UserApp need to be changed also in PCDC Flashloader. See below.

Moving PCDC Flashloader

- In the Linker -> Section settings of the e^2 studio project, change the address of the *P_Reset* section. This is where PCDC Flashloader starts.
- The block numbers for UserApp are needed by PCDC Flashloader to erase the memory that will later be written with UserApp. They are defined in file *r_usb_pcdc_apl_flashloader.c. Change them* as shown below to be able to use the DebugConsole.

```
#define     BOUNDARY_ADDRESS_FLASHLOADER    0xFFFF6000UL
#define     LOW_BLOCK_USERAPP               40
```

Flashloader will not allow anything above BOUNDARY_ADDRESS_FLASHLOADER to be flashed. If the UserApp image contains such addresses an error will be seen in the GUI, though any existing UserApp Fixed Vector area in the download image is simply ignored.

Observe: The MOT file may not start on an address boundary smaller than MIN_APP_FLASH_SZ. There will be a flashing error otherwise. This is because the code assumes an even boundary start address of the application.

Block numbers and corresponding addresses for the RX100 are shown in section 12.

4. Fixed Vector Table

The Fixed Vector Table resides in the PCDC Flashloader area. It is logical to have the fixed vectors owned by PCDC Flashloader since it manages reset and boot.

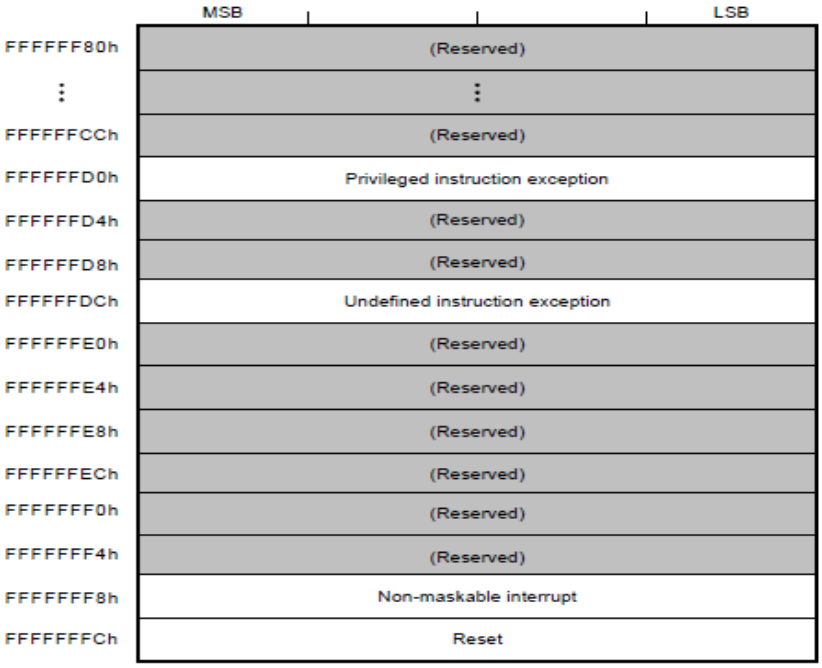


Figure 4. The fixed vectors are located at the very highest memory.

The Fixed Vector Table cannot be moved in memory, thereof the name *Fixed*.. The table therefore belongs toped Flashoader, which is always resident in memory to service these interrupts. The fixed vectors should therefore always vector to within Flashloader.

If for some reason this needs to change, here are a few ways this could be approached. Method 1below is likely the most straightforward solution.

For both cases below, the Reset vector must always point to PCDC Flashoader's reset function. Flashloader's reset must therefore be place at a known address so the user can enter this into the UserApp reset vector position. Also, when UserApp is flashed (after the UserApp area is erased) there will be no code in place to service the fixed interrupts. This could be remedied by before erasing UserApp, flashing temporary handlers pointing to functions within Flashloader.

1. Change PCDC Flashloader so that it programs the fixed vector area fr.om UserApp. See *BOUNDARY_ADDRESS_FLASHLOADER* in *r_usb_pcdc_apl_flashloader.c*.
2. The user could add an additional UserApp header field containing an address to a fixed vector array to flash into the fixed vector area. PCDC Flashloader would at the very end of flashing UserApp read this field and overwrite the existing fixed vector pointer array (pointing to handlers in the Flashloader area).

5. Reset (Boot) Procedure

After an image has been reflashed, here is how execution proceed starting at the time of running UserApp. Execution while in PCDC Flashloader is shown in yellow, and while in UserApp is shown in green

UserApp

- Execute from UserAppStart; setup clock, HW-setup etc.
- User presses SW3
  - Device resets

    Reset vector is in PCDC Flashloader

PCDC Flashloader

- Reset vector is in PCDC Flashloader
- Boot; setup clock, HW-setup etc.
- Detect if SW1 pressed. If so, stay in PCDC Flashloader.
  - SW1 not pressed:
    - Is there a valid UserApp header?

      Yes:

      Jump to UserApp (execute UserAppStart)

      No:

      Stay in CDC bootloader

6. UserApp

A sample UserApp is provided so that the user code developer will have some key items already set up, such as the entry function *UserAppStart()*. In this chapter we will study what is needed for UserApp to work together with PCDC Flashloader.

The sample UserApp user application toggles the RSK LEDs. The toggle pattern changes when the user presses the board switches.

6.1 Entering PCDC Flashloader from UserApp

By default, if the user presses SW3, a software reset is issued. Since the fixed vector (and thus Reset) belongs to PCDC Flashloader, the flashloader will start from reset. This can easily be changed to suit the specific needs of the product.

6.2 The UserApp Header

There is a UserApp "header" located at a predefined location in memory. This header contains the *only* information that PCDC Flashloader can know about UserApp. This is to avoid problems with the linker moving code around for UserApp. The header location remains fixed by a specific section setting for *UserAppStartSect*, so that Flashloader can always find the start address etc header information provided for UserApp. *If* the header were moved, PCDC Flashloader must be made aware of the new location. In the first versions of PCDC Flashloader, instead of reading the header, there is just a call to *jump_to_userapp*, located in *r_usb_pcdc_apl_flashloader.c*.

The UserApp header should be located at the end of UserApp flash as shown with the dark green field in Figure 3. There is a prototype for the header in *resetprog.c*.

The UserApp header should contain at a minimum items 1-3 below:

1. UserApp start address ("soft" reset)
2. A security code, e.g. 55AA55AAh.
3. Version id

..more items can be added.

6.3 Moving the UserApp header

The UserApp header element *userapp_entry_addr* is automatically updated by the linker if the function *UserAppStart* is moved. So moving the header is not necessary. (This is part of the reason for having the header!) To move the UserApp header in memory should this be warranted for some reason, follow these steps.

1. Make a note, for your own future reference, where the code is mapped in the text file: *..\CDC_Flashloader\Doc\RX111_blocks_flashloader_userapp.txt*.
2. In the Linker Section settings, change the address for *UserApp_Head_Sect* to where you want UserApp to reside. It is recommended to have the header at the highest address in the UserApp Area since it is flashed last. If flashing fails or is interrupted, the header will most likely not be correct.

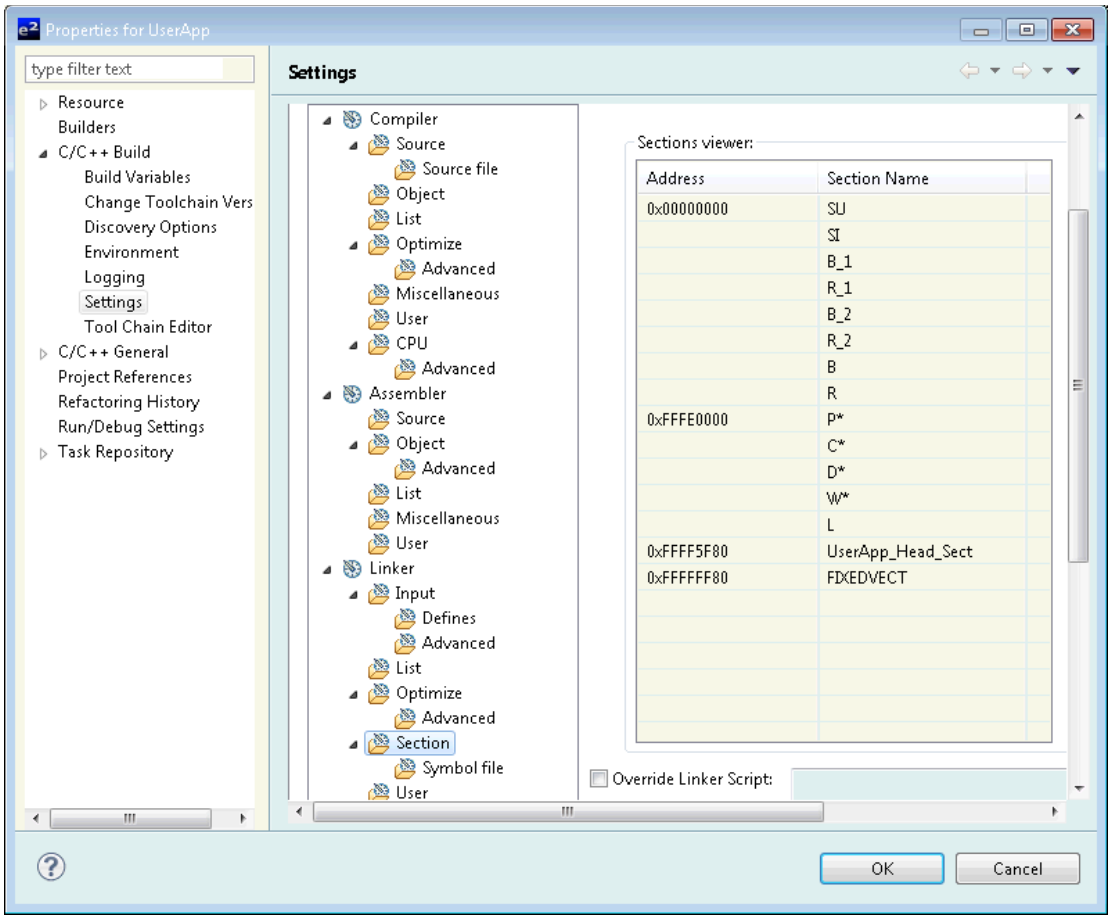


Figure 5. Linker Section settings. The UserApp header section is here at 0xFFFF5F80, towards the high end of UserApp's memory area.

3. In the function *jump_to_userapp()* in r_usb_pcdc_apl_flashloader.c, set this same address as:
   MOV.L **#0XXXXXXXXh**, R9 ;This hex value is the start address of the UserAppStart function.
4. In *r_usb_pcdc_apl_flashloader.c*
   1. Set LOW_BLOCK_USERAPP and HIGH_BLOCK_USERAPP according to how you mapped it out in 1.
   2. Change USERAPP_HEADER_ADDR to same value as in 1 and 3 above.

6.4 UserApp's Fixed Vector Area

Under normal usage, with PCDC Flashloader resident, the fixed vector will not be flashed, even if the CDC downloader GUI sends it to the target. However the sample UserApp does have a fixed vector so that it can be developed and debugged on its own. See 6.5.

6.5 Debugging UserApp Standalone

When UserApp is being developed, it is not necessary to have PCDC Flashloader resident to take care of reset. Therefore to make debugging simpler, set DEBUG_USERAPP=1in e^2 studio's compiler settings. (Project Properties => C//C++ build settings.).

7. PC Host Code & Qt

The host GUI used to select and download the target UserApp was created using the Qt IDE and libraries. You do not need Qt to run the executable, as explained in 2.2.

Should you wish to modify the host code, download the IDE for QT from http://qt-project.org. Follow these steps.

1. Click on the download link. Download and run Qt Online Installer for Windows.
2. When selecting toolkits, note that the supplied executable was built with the *2012 MinGW 32-bit* version.

   Installing the *2012 MSVC 64-bit* is also an option, but the compiled executable will not run on a 32-bit OS PC in case backwards compatibility with older systems is desired.
3. After the IDE is installed, run the host source code by double-clicking on the file *Qt_CDC_USB_Downloader.pro* in folder *..\USBFW_mini\PCDC_FlashLoader_PC_host*.
4. Qt Creator should start.
5. When opening the project for the first time you may have to specify build output folders. Note that these may not be subfolders of the source code or you will later run into errors. Pick a place parallel to the source code or above it.
6. Run Qmake from the menu bar.
7. Build and debug with F5.

Tip: The Help button opens up a window with lots of valuable and searchable information. Here you can reference Qt classes and widgets, and in particular for this project the *QtSerialPort* class information.

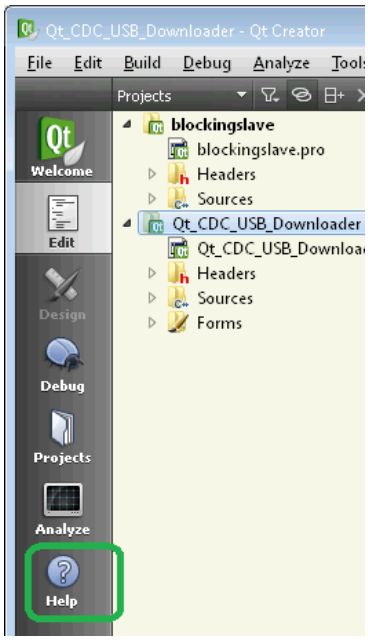


Figure 6. Qt Creator is the tool to edit and compile code.
Qt Help marked in green is an effective aid in learning about Qt classes and widgets.

8. PCDC Flashloader

8.1 VID & PID

The Vendor ID must be changed by the user before releasing any product to market. Vendor and Product ID are found in *r_usb_phid_usrcfg.h* and in multiple places in the INF-file. The numbers must match exactly or enumeration with the PC host will not happen.

8.2 Entering UserApp from PCDC Flashloader

After a device reset, PCDC Flashloader checks whether the UserApp header is present in flash memory and contains the expected security code. If so, UserApp is executed.

8.3 Reprogramming

If SW1is held down at reset, UserApp will not be entered, and the device can be reflashed.

8.4 S-record Processing

The incoming S-records are already processed by the host to be in hex format, and not ASCII characters.

All data records are deemed to arrive from the host in increasing address order. If this is not the case, the host code must be changed to send the data records in order.

There is a minimum flash size *MIN_FLASH_SZ. F*or the RX this is four bytes. Data records must be processed to start at an even multiple of the *MIN_FLASH_SZ* boundary and have a size being a multiple of *MIN_FLASH_SZ*. To "sew" together a flash buffer that satisfies this, two *srec_flash_data* buffers exist and are referred to with indexes *prev* and *new*. Besides *rec_flash_data[]*, these buffer structures also contain the structure elements *address, nr_flashdata_bytes,* and *checksum*. The use of these buffers is toggled. Before *prev* is flashed, it is padded with any bytes from *new* that are within the same minimum boundary. Also, each *new* data record's start address is adjusted to a MIN_FLASH_SZ boundary.

Or, more precisely:

- For each incoming data record (*new*) that does not intrude in the previous (*prev*), the start address must only be adjusted to the next lower MIN_FLASH_SZ boundary address, and the data within *new* "right shifted" to compensate. The new start bytes at the beginning of new must be set to FFh. The data count is unaffected.
- A data record may have a start address that intrudes into the end of the previous data record's *MIN_FLASH_SZ* area (end of *prev*). The previous data record can therefore not be flashed until the new data record has been processed so intruding bytes are moved to the correct space within *prev. (new*'s count and start address must also be adjusted.
- Before flashing *prev*, its end may need to be padded with FFh. If *new* intruded in *prev* as per above, this should already have been taken care of.

8.5 Changes Made to Original PCDC

The core additions made to the original PCDC package to create PCDC Flashloader are in file *r_usb_pcdc_apl_flashloader.c*, in particular in function *usb_psmpl_MainTask*(). The following subroutine functions have been added to the file.

```
r_fcl_status_t   analyze_new_dr_move_intruding_bytes_and_flash(prev, last_record);
r_fcl_status_t   flash_prev_and_check(prev);
void             prepare_and_send_response_record(response_type, response_field);
```

9. Using the Renesas Debug Console

To enable trace strings and data via the E1/E20 to e^2 studio, use the Renesas Debug Console. This means you have the ability to use printf() statements in C to send trace strings to the standard output. Standard output will in this case be the E1/E20 debug register.

To use this set BSP_CFG_IO_LIB_ENABLE to 1 in *../r_config/r_bsp_config.h.*

This macro will enable the following in order to use the Debug Console.

1. *INIT_IOLIB()* must be called. This is sometimes commented out in *resetprog.c* to save object code space.
2. The code in *lowlvl_debug.c* contains the *putchar* and *getchar* functions so that the E1/E20 debug registers are used for Debug Console I/O processing.
3. Include <*stdio.h*> in any files where you wish to use printf-statements.
4. In e^2 studio, add the Debug Console window by switching on both icons

   "1/0" and

   "Pin Console" as shown below.

   Both must be on so that the print buffer in E1/E20 can be emptied, and not block.

Figure 7. Buttons to control the Debug Console. Press the I/O button for the console in e^2 studio again if the console seems unresponsive.

5. If nothing is printed, press the Clear icon a few times. (The icon partially concealed by the red border of Figure 7.)

9.1 Adding traces to code

To any file where *printf()* is called, add

```
#if BSP_CFG_IO_LIB_ENABLE
    #include <stdio.h>
#endif
```

9.2 UserApp

Set DEBUG_USERAPP = 1 in e2 studio as explained in 6.5.

9.3 PCDC Flashloader

In PCDC Flashloader you can follow how the image is flashed. Since by default the space for PCDC Flashloader is constrained in order to maximize space for UserApp, PCDC Flashloader's Debug Console code is turned off (and compiler optimization is turned on for size). Follow section 6.3 to move PCDC Flashloader to make space for the stdio code.

10. Error Messages

Here are error messages displayed in the host GUI's text output when the RX target does not respond with a regular ACK.

(1) **Target sent RR_ERR_NAK_RESEND**

"Response Type error code: RR_ERR_NAK_RESEND."

"Target requested resend of the record..."

(2) **Target sent RR_ERR_NAK_END sent from target**

"Response Type error code: RR_ERR_NAK_END." plus, if there is an error code in the Response Field, you will get a ***"Response Field error:"*** plus one of the Flash API errors (flash_err_t):

- "FLASH_ERR_BUSY"	***1***	***Peripheral Busy***
- "FLASH_ERR_ACCESSW	***2***	***Access window error***
- "FLASH_ERR_FAILURE	***3***	***Operation failure, Programming or erasing error due to something other than lock bit***
- "FLASH_ERR_CMD_LOCKED	***4***	***RX64M - Peripheral in command locked state***
- "FLASH_ERR_LOCKBIT_SET	***5***	***RX64M - Pgm/Erase error due to lock bit.***
- "FLASH_ERR_FREQUENCY	***6***	***RX64M - Illegal Frequency value attempted (4-60Mhz)***
- "FLASH_ERR_ALIGNED	***7***	***RX600/RX200 - The address that was supplied was not on aligned correctly for ROM or DF***
- "FLASH_ERR_BOUNDARY	***8***	***RX600/RX200 - Writes cannot cross the 1MB boundary on some parts***
- "FLASH_ERR_OVERFLOW	***9***	***RX600/RX200 - Address + number of bytes' for this operation went past the end of this memory area.***
- "FLASH_ERR_BYTES	***10***	***Invalid number of bytes passed***
- "FLASH_ERR_ADDRESS	***11***	***Invalid address or address not on a programming boundary***
- "FLASH_ERR_BLOCKS	***12***	***The "number of blocks" argument is invalid***
- "FLASH_ERR_PARAM	***13***	***Illegal parameter***
- "FLASH_ERR_NULL_PTR	***14***	***Received null ptr; missing required argument***
- "FLASH_ERR_TIMEOUT	***15***	***Timeout***

You will also get

"**ERROR in send_image(). END**."

(3) **The target's Ack-field is neither ACK nor NACK**

"**Response Type error code: Unknown.**"

You will also get

"**ERROR in send_image(). END**."

(4) **Other Faults**

- "**Record Type error. Expected a Response Record. Got something else**."
- "**Response Length error**."
- "**Response Checksum err. It does not match the host's calculation.**"
- "**Response Type error code: Unknown.**"

11. Communication Protocol

11.1 Download Sequence

The USB host and USB target (RX100) are shown in Figure 8. The host PC and PCDC Flashloader communicate with each other using “records”.

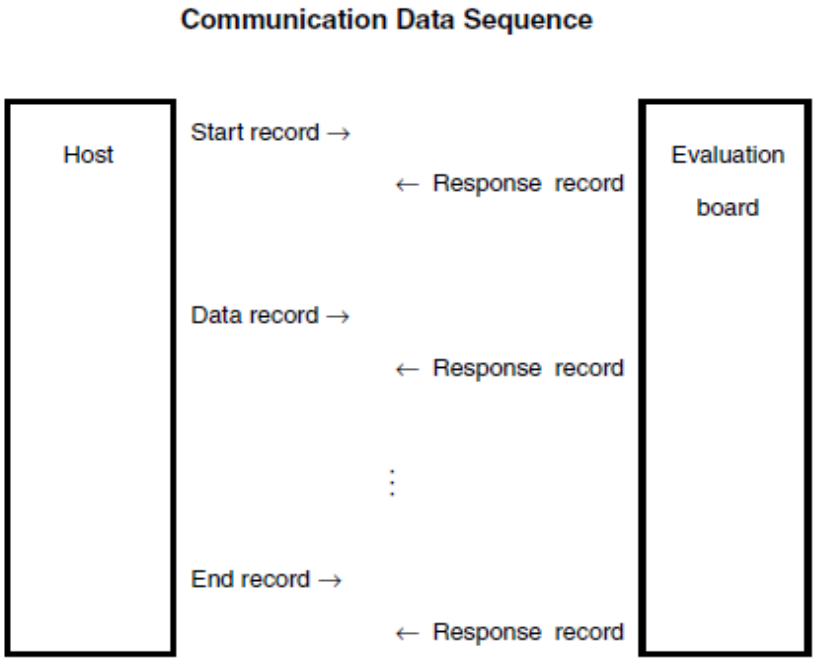


Figure 8. Communication sequence between the USB PC host GUI to the left, and the RX100 target to the right.

The record formats of data sent over USB by the host PC, and the responses from the CDC Flashloader target are shown in detail below in 11.2.

11.2 Transfer Records

Start Record

The host will start the download of the new user image with a ‘Start Record’.

Data transmitted by the host

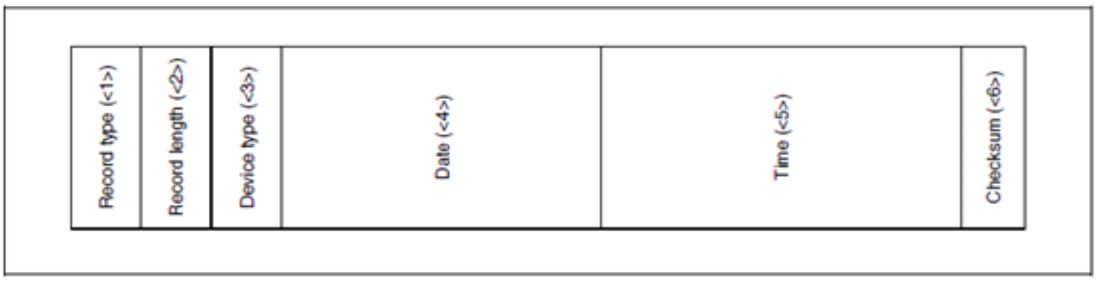
The host transmits a start record, data records, and an end record.

Records are transmitted one by one, and the next record is not transmitted until a response record is received.

(1) Start record

This record is transmitted first when updating the firmware.

Start Record Format



<1> Record type

The type of record

1 byte

The record type of the start record is 0x00.

<2> Record length

The number of bytes for the device type and later

1 byte

<3> Device type

The type of device

1 byte

<4> Date

The current date

The year, month, and day require 1 byte each.

The last two digits of the year are specified (based on the Western calendar).

<5> Time

The current time

The hour, minute, and second require 1 byte each.

<6> Checksum

The record checksum

1 byte

This is the checksum for the record length, device type, date, and time.

The checksum is the lower 8 bits of the one's complement of the sum of each byte value.

Figure 9. Start Record. This is sent from the USB host to start the download.

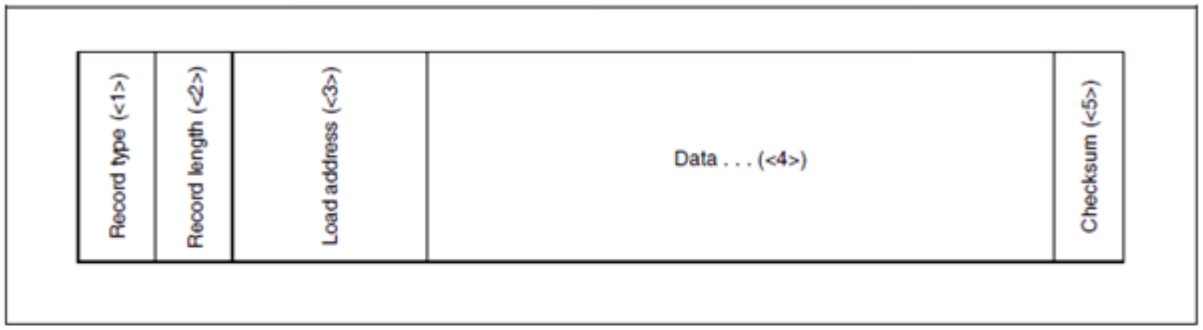
Data Record

If the target determines that the Start Record and checksum contain valid data, the target will make itself available to receive flash data, and to start flashing the user area. The data arrives in the form of Data Records.

Data records

These records contain the data to be loaded into the flash memory.

Data Record Format



<1> Record type

The type of record
1 byte
The record type of a data record is 0x0f.

<2> Record length

The number of bytes for the load address and later
1 byte

<3> Load address

A flash memory address
4 bytes
Data is loaded into the flash memory starting at this address.
The load address is a 32-bit number in little endian format.

<4> Data

The data to load into the flash memory
Each record can contain up to 256 bytes.

<5> Checksum

The record checksum
1 byte
This is the checksum for the record length, load address, and data.
The checksum is the lower 8 bits of the one's complement of the sum of each byte value.

Figure 10. Data Record.

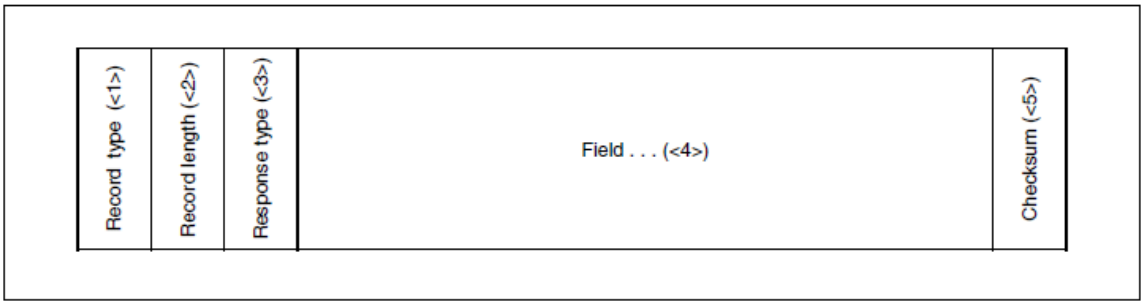
Response Record

Data transmitted by the evaluation board

The evaluation board transmits records in response to records from the host.

(1) Response records

Figure 7-5. Response Record Format



<1> Record type

The type of record

1 byte

This is the type of record for which this response record is returned.

<2> Record length

The number of bytes for the response type and later

1 byte

<3> Response type

The response type

1 byte

The following three types are available:

0x00: ACK

0x0f: NAK (a request to transmit the record again)

0xf0: NAK (an error termination)

<4> Field

If an error occurs, the field is a 1-byte error code.

If no error occurs, the contents vary depending on the record type as follows.

Start record: Device type

Data record: Load address

End record: Device type

<5> Checksum

The record checksum

1 byte

This is the checksum for the record length, response type, and field.

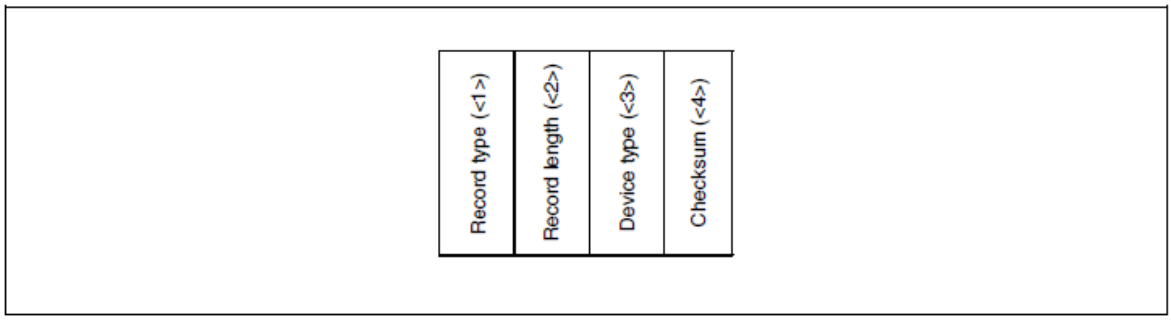
The checksum is the lower 8 bits of the one's complement of the sum of each byte value.

Figure 11. Response Record

End Record

End record
This record is transmitted after all other records.

Figure 7-4. End Record Format



<1> Record type
The type of record
1 byte
The record type of the end record is 0xf0.

<2> Record length
The number of bytes for the device type and later
1 byte

<3> Device type
The type of device
1 byte

<4> Checksum
The record checksum
1 byte
This is the checksum for the record length and device type.
The checksum is the lower 8 bits of the one's complement of the sum of each byte value.

Figure 12. End Record

12. Block Numbers and Addresses of RX100

```
#define BLOCK_0     0     /*   1KB: 0xFFFFFC00 - 0xFFFFFFFF */
#define BLOCK_1     1     /*   1KB: 0xFFFFF800 - 0xFFFFFBFF */
#define BLOCK_2     2     /*   1KB: 0xFFFFF400 - 0xFFFFF7FF */
#define BLOCK_3     3     /*   1KB: 0xFFFFF000 - 0xFFFFF3FF */
#define BLOCK_4     4     /*   1KB: 0xFFFFEC00 - 0xFFFFEFFF */
#define BLOCK_5     5     /*   1KB: 0xFFFFE800 - 0xFFFFEBFF */
#define BLOCK_6     6     /*   1KB: 0xFFFFE400 - 0xFFFFE7FF */
#define BLOCK_7     7     /*   1KB: 0xFFFFE000 - 0xFFFFE3FF */
#define BLOCK_8     8     /*   1KB: 0xFFFFDC00 - 0xFFFFDFFF */
#define BLOCK_9     9     /*   1KB: 0xFFFFD800 - 0xFFFFDBFF */
#define BLOCK_10    10    /*   1KB: 0xFFFFD400 - 0xFFFFD7FF */
#define BLOCK_11    11    /*   1KB: 0xFFFFD000 - 0xFFFFD3FF */
#define BLOCK_12    12    /*   1KB: 0xFFFFCC00 - 0xFFFFCFFF */
#define BLOCK_13    13    /*   1KB: 0xFFFFC800 - 0xFFFFCBFF */
```

```
#define BLOCK_14    14    /*   1KB: 0xFFFFC400 - 0xFFFFC7FF */
#define BLOCK_15    15    /*   1KB: 0xFFFFC000 - 0xFFFFC3FF */
/* Flashloader above here. Move this line if change. ********/
/* UserApp below here. Move this line if UserApp is moved. **/
#define BLOCK_16    16    /*   1KB: 0xFFFFBC00 - 0xFFFFBFFF */
#define BLOCK_17    17    /*   1KB: 0xFFFFB800 - 0xFFFFBBFF */
#define BLOCK_18    18    /*   1KB: 0xFFFFB400 - 0xFFFFB7FF */
#define BLOCK_19    19    /*   1KB: 0xFFFFB000 - 0xFFFFB3FF */
#define BLOCK_20    20    /*   1KB: 0xFFFFAC00 - 0xFFFFAFFF */
#define BLOCK_21    21    /*   1KB: 0xFFFFA800 - 0xFFFFABFF */
#define BLOCK_22    22    /*   1KB: 0xFFFFA400 - 0xFFFFA7FF */
#define BLOCK_23    23    /*   1KB: 0xFFFFA000 - 0xFFFFA3FF */
#define BLOCK_24    24    /*   1KB: 0xFFFF9C00 - 0xFFFF9FFF */
#define BLOCK_25    25    /*   1KB: 0xFFFF9800 - 0xFFFF9BFF */
#define BLOCK_26    26    /*   1KB: 0xFFFF9400 - 0xFFFF97FF */
#define BLOCK_27    27    /*   1KB: 0xFFFF9000 - 0xFFFF93FF */
#define BLOCK_28    28    /*   1KB: 0xFFFF8C00 - 0xFFFF8FFF */
#define BLOCK_29    29    /*   1KB: 0xFFFF8800 - 0xFFFF8BFF */
#define BLOCK_30    30    /*   1KB: 0xFFFF8400 - 0xFFFF87FF */
#define BLOCK_31    31    /*   1KB: 0xFFFF8000 - 0xFFFF83FF */
#define BLOCK_32    32    /*   1KB: 0xFFFF7C00 - 0xFFFF7FFF */
#define BLOCK_33    33    /*   1KB: 0xFFFF7800 - 0xFFFF7BFF */
#define BLOCK_34    34    /*   1KB: 0xFFFF7400 - 0xFFFF77FF */
#define BLOCK_35    35    /*   1KB: 0xFFFF7000 - 0xFFFF73FF */
#define BLOCK_36    36    /*   1KB: 0xFFFF6C00 - 0xFFFF6FFF */
#define BLOCK_37    37    /*   1KB: 0xFFFF6800 - 0xFFFF6BFF */
#define BLOCK_38    38    /*   1KB: 0xFFFF6400 - 0xFFFF67FF */
#define BLOCK_39    39    /*   1KB: 0xFFFF6000 - 0xFFFF63FF */

#define BLOCK_40    40    /*   1KB: 0xFFFF5C00 - 0xFFFF5FFF */
…
```

See file
..Doc\RX111_blocks_flashloader_userapp.txt
for full list of block numbers and addresses.

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	Jan 2, 2014		First version.
1.10	Nov 15, 2014	None (code only)	- Changed to newer RX Flash API r01an2184eu0105_rx. - Macro MIN_FLASH_APP_SZ added. Can be changed to a multiple of MIN_FLASH_SZ to increase buffering of data from USB before flashing. - Added RPB111 for flashloader.

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 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 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.

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, 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: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

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: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, 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.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0