To our customers,

# Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

# Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "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.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. 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.
9. 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 system manufactured by you.
10. 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.
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.

# Families R8C, M16C, R32C, SH2 & SH2-A (RCAN-ET)

Flash Over CAN

## Table of Contents

## 1. Introduction to Flash over CAN

Flash over CAN, or FoCAN, is the ability to reprogram a Renesas CAN MCU using the CAN API over an existing CAN network bus. This removes the need for standard debug, UART or serial interfaces to update the device firmware.

FoCAN uses a Systec 'CANmodul' CAN bus interface which is part of the RCDK, or 'CAN-D-Kit'. This HW interface is used to send the bus CAN frames to the device. See chapter 13 下の for information on this interface. A Windows based Application, FoCAN Download, provides a graphical interface to program the MCU via the CAN network. Each device can be flashed individually in-network using the unique FoCAN Device Unlock Code.

At this writing, FoCAN firmware with a demo application is available for the R32C/118, M16C/6NK, M16C/29, R8C/23. Also available are the SH RCAN-ET MCUs SH2A-7286 and SH2-SH7137. The SH2A-7216 is soon to be released. FoCAN can be adapted to any other CAN equipped MCU within these families. The following MCUs are also available on Renesas Starter Kit boards: R32C/118, M32C/87, M16C/6NK, M16C/29, R8C/23, SH7286, SH7137. For these, FoCAN workspaces are available to download.

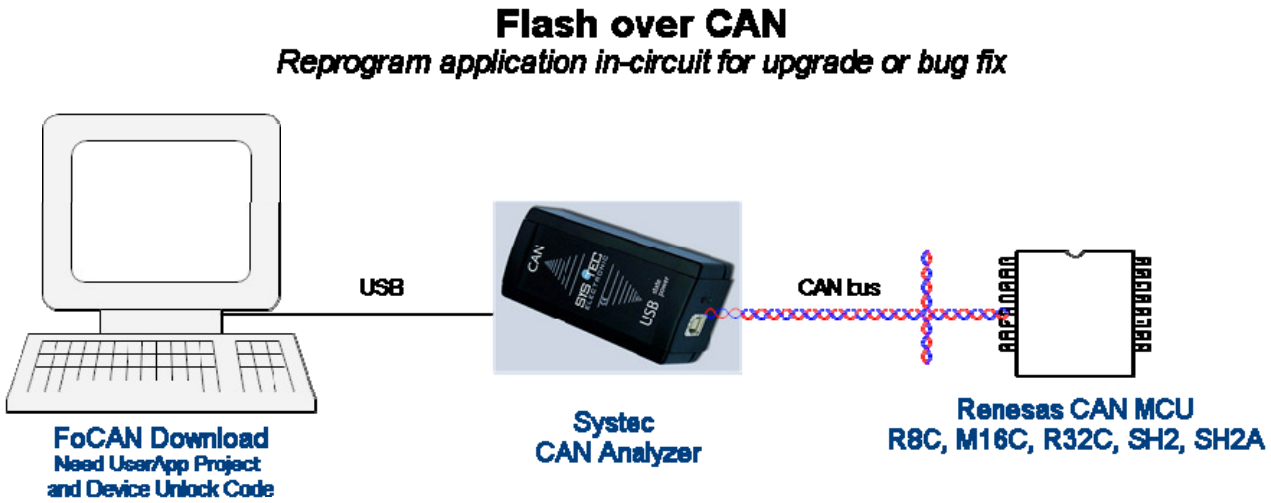


**Figure 1. Flash over CAN may be used to reflash an application after a device is installed by giving the device a unique programming ID.**

Flash over CAN consists of a High-performance Embedded Workshop (HEW) workspace containing two projects, **CANloader** and **UserApp**. These two projects are independent of each other in that they are not sensitive to any code or data remapping by the linker. Both projects have a header so that the other project can read key data; entry access points into the other program, device and version IDs, data consistency checksum value, and the security FoCAN Device Unlock Code.

CANloader, the flashing project, is located in the first flash block of the MCU. The user application is free to occupy the other flash block(s).

To reflash, the user PC program PC_Download will send reprogramming commands and data frames down through the sniffer over CAN to the MCU. CANloader will first erase the user blocks and then reflash the new user application. After completion, the device reboots, checks for data consistency and enters the user application.

The application may be reflashed any time a user connects the PC to the bus and invokes the device's proper FoCAN Unlock Code and reprogramming CAN ID. When this happens the application exits, reads the CANloader header for entry address into CANloader, tells the PC it's ready and UserApp is reflashed again.

## 2. General description of the Flash over CAN concept

F-o-CAN will work on any M16C family device with at least one CAN peripheral and internal flash memory. The PC application, FoCAN Download, sends programming command frames and application content data frames via USB to the Systec CANmodul sniffer which sends them over CAN to the MCU.

The FoCAN firmware is written in one HEW workspace containing two projects; CANloader, and UserApp. CANloader resides in the first flash memory block, or first two blocks if block 0 is too small for that specific device. CANloader erases the user application block(s) and then reflashes the UserApp firmware. The UserApp firmware includes reflash exit code to transfer execution over to CANloader whenever reflashing is invoked. This feature allows the device to enter reflash mode during normal execution of the application. UserApp is also where user application code is inserted by the linker. After reflash completes, the device reboots and checks for data consistency before entering the application.

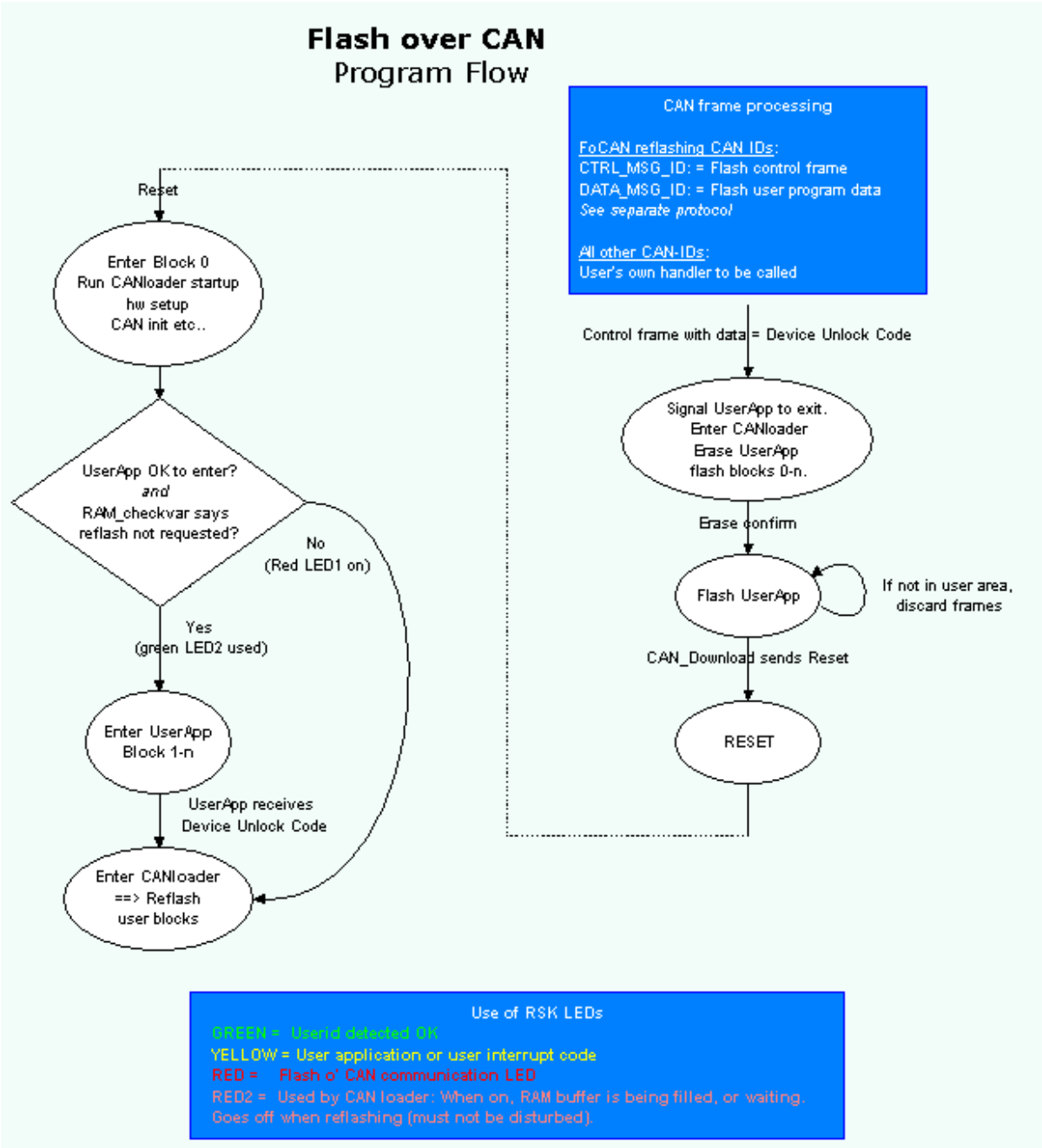


**Figure 2. Program flow. To the left: Reset, CANloader's startup code, and entering the user application (UserApp). To the right is shown processing of CAN frames in UserApp and in turn CANloader if a valid download is asserted.**

## 3. The Device Unlock Code

The default FoCAN **Device Unlock Code** to unlock the device and reflash is

1. For the **RSK-R8C23**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**1**.
2. For the **RSK-M16C29**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**2**.
3. For the **RSK-M32C87**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**3**.
4. For the **RSK-R32C118**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**4**.
5. For the **RSK-SH2A7286**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**5**.
6. For the **RSK-SH27137**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**6**.
7. For the **RSK-SH27216**: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1**7**.

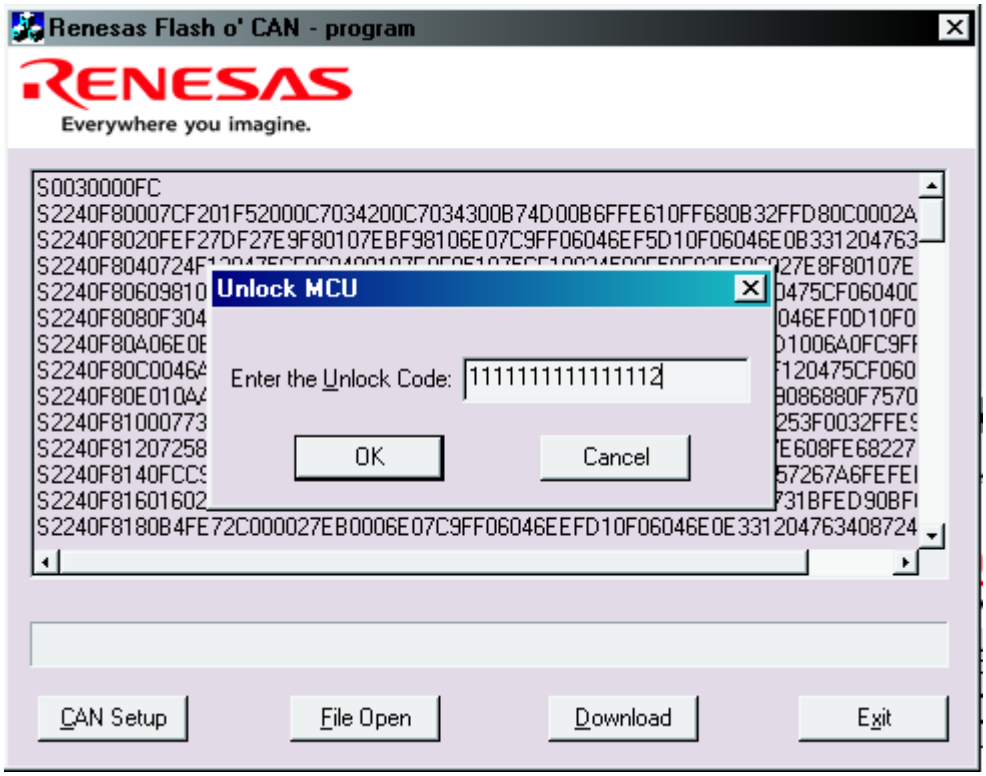


**Figure 3. The PC_Download application. The Device Unlock Code is entered, after pressing 'Download'. This example shows using the default Device Unlock Code for the RSK29.**

In the source code, this code is set in the file canload_head.c. Note that this is not the same as the code used to unlock the device when accessing it via a serial flash programmer e.g. the E8 debugger.

If FoCAN Download sends the correct control frame CAN-ID, the UserApp application reads from canload_head.c in the CANloader section for an entry address to enter CANloader. CANloader then confirms to the sniffer that it is ready and the application is reflashed.

Warning: No other interrupts should occur during reflash, as the MCU goes into and out of MCU erase / rewrites (E/W) mode.

## 4. Reflashing Steps

When the 'Program' button in the Windows PC application FoCAN Download is pressed, commands will be sent to the Systec sniffer to start downloading the user application code to the MCU flash. The first flash command frame that is sent is a FoCAN Unlock Code that the user enters right after the 'Program' button was pressed on the PC. If the code matches the value in the device's FoCAN Device Unlock Code in project

CANloader, programming is accepted, the flash_request_flag is set and the target device firmware enters into 'Flash over CAN' mode. CANloader then confirms to PC_Download to proceed with the flash update. However, only CAN frames with CAN ID value CTRL_MSG_ID are accepted and processed for reprogramming.

The CANloader firmware will idle when not processing CAN interrupts.

UserApp firmware is reflashed into the user flash memory blocks using EW1 mode. After the user blocks are reflashed the device reboots.

After reboot, CANloader checks that the UserApp space was successfully flashed by verifying UserApp's memory using a checksum algorithm, and comparing the result with a checksum reference value. If CANloader does not calculate a matching checksum to what is stored in the App-header, the device stays in CANloader and will wait until a new attempt is made to flash UserApp.

To add a checksum to the application header you can use the default checksum algorithm or your own. This is to be done when all application development activities are finished. See 'Checksum protection against failed application reflash'.

Note that the relocatable user mode (UserApp) interrupt vector will automatically be used when UserApp is entered, as everything such as interrupt vectors, stack pointer, clock frequency etc is initialized by UserApp's own startup code.

### 4.1 Download procedure

Program the RSK board with the CANloader project binary using HEW or FDT. See your RSK's Quick Start Guide for details on which MCU device to use, and how to connect and program the device.

**Do not download the Debug folder and session binary to the target** with FoCAN Download, as it contains the reset vector, which will then overwrite CANloader's fixed vector. See Debugging below.

**a.** Program the RSK board with the CANloader project binary using HEW or FDT. See your RSK's Quick Start Guide for details on which MCU device to use, and how to connect and program the device.

**b.** After programming CANloader, unplug your debugger and press reset or power cycle the device. The bottom red LED should light up indicating the device is in CAN bootload mode waiting to be programmed.

**c.** In HEW switch to the UserApp project.

**d.** Add your application under user_main(). To debug your application see 'Debugging your application' below.

**e.** Compile and link UserApp.

**f.** Start FoCAN Download.exe
The PC application FoCAN Download.exe can be found in the "PC_Download" folder in the "Flash over CAN" Hew workspace directory. If your PC says it is missing a DLL file there are spare DLL files in the folder of your FoCAN distribution, next to the FoCAN executable.

**g.** When CANloader is running on target RSK board, the FoCAN Download.exe will send CAN control messages with CAN-ID CTRL_MSG_ID (0x201 in the example code) and subsequent program data frames with the id DATA_MSG_ID (0x200).

**h.** Connect a Systec USB CANmodul sniffer to your PC and to CAN ch 0. (For the 6NK that is the CAN connectors closest to the E8 connector.)

**i.** Inside FoCAN Download, Press 'CAN Setup' and set to 500 kbaud. CAN Message ID should default to 200 in Standard.

**j.** Press 'File Open' and select the new application binary created from compiling UserApp.

**k.** Press Download. The FoCAN Device Unlock Code is typically default 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11 in the source code, but can be changed. See the variable unlockcode[8] located in the source code file canload_head.c.

**l.** Press OK. It takes a moment to erase the user blocks, then the LED will flash each time a control frame arrives and the flash process continues.

**m.** When the flash process is complete and CANloader is sent the ‘file complete code’ from PC_Download, the device is fully programmed.

**n.** After rebooting, the UserApp header is checked for a correct checksum or code before entering. This header code would preferably contain a valid checksum generated via the process mentioned below in ‘*Checksum protection against failed application reflash*’.

## 5. Modifying the default FoCAN project

To adjust the sample project to your particular MCU variant there are a few code mapping issues to be looked over. Most of the items you need to consider modifying for your particular project are commented in the file focan.h.

### 5.1 Memory Mapping of CANloader and UserApp

Flash memory Block 0 of the MCU contains CANloader, the flashing protocol handling project.

UserApp - the user application - may be mapped into any remaining block(s). All program code and constant data are mapped either before or after each project's header. The header locations are defined by the CANLOAD_HEAD and APP_HEAD section start addresses.

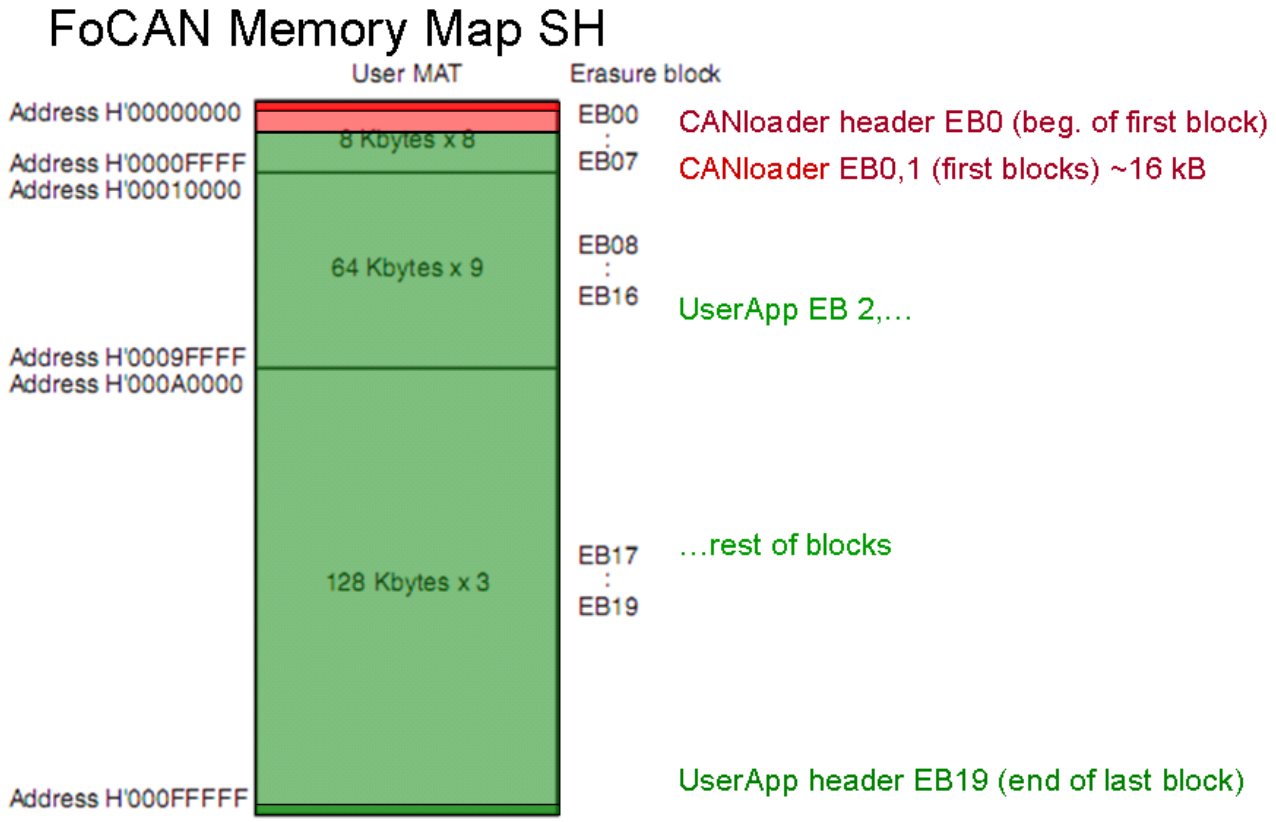


**Figure 4 Flash o' CAN memory layout for an SH7216 MCU.**

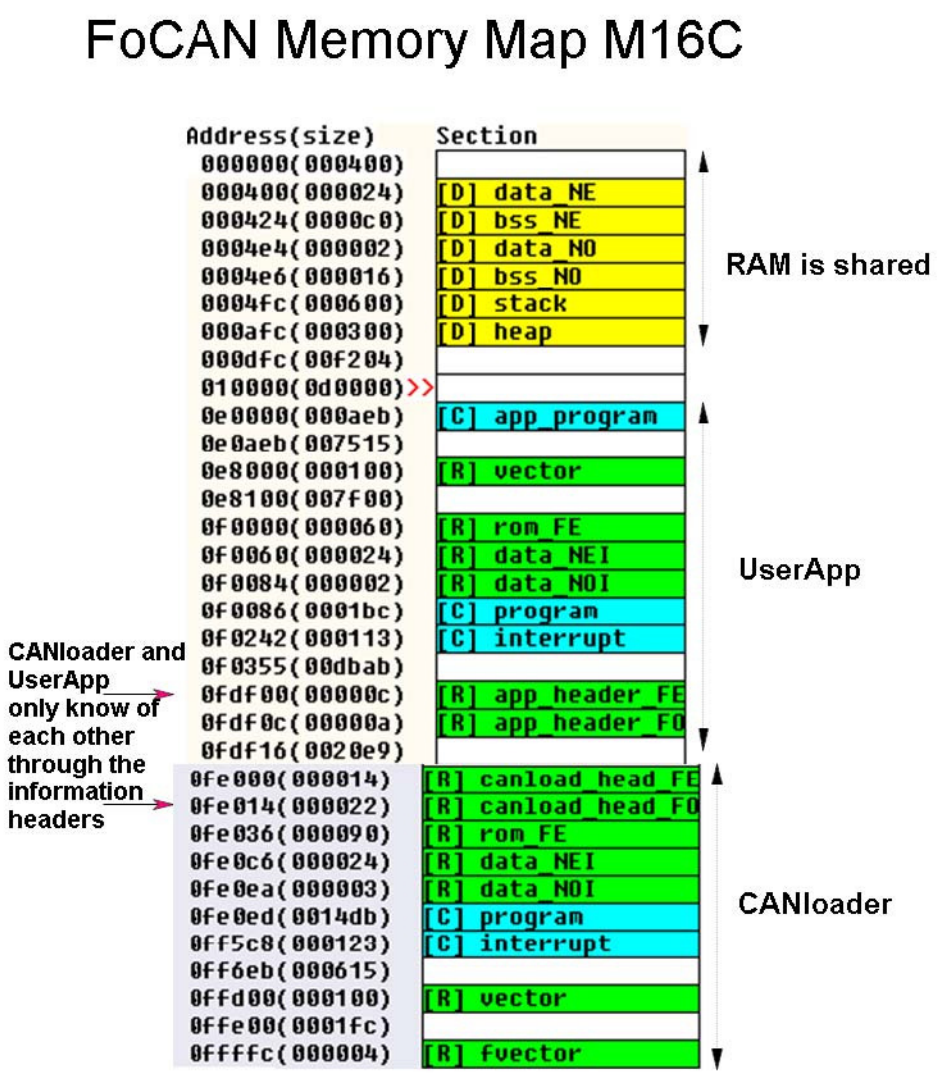


**Figure 5 For R8C, M16C and R32C block zero is at the high end of the address spectrum. Note the red arrows indicating headers. This example shows an M16C/6Nx MCU.**

These header addresses are the only thing the projects know about each other. The header member data are certain key information pieces that the projects must be able to read from each other. The header information members are at fixed relative offsets from the header tip. Besides execution entry point there is information such as Firmware Version nr, Device Unlock Code, Security checksum. More can very easily be appended to the structure.

For project CANloader, note the location of section ‘canload_head’ in Figure 5. All project sections are subsequently placed within flash Block 0.

For project UserApp, all program code is mapped to section ‘app_program’. Note the section’s location and the subsequent placement of the other sections for this project. Last is the header. For M16C type devices, block numbers are reversed with respect to memory address and so ‘app_header’ is at the *end* of Block 1 right up against Block 0. This is done in order to keep the headers conveniently close together and the code as ‘tight’ as possible. For SH devices, ‘app_header’ is by default located at the *beginning* of the first UserApp block, e.g. Block 1.

The mapping can be rearranged. As long as a header is not in the middle of code. The checksum calculation would then be more difficult as the checksum reference value can not be included in the checksum calculation of the application code.

To pick header and code section addresses for your memory footprint, see your device's HW manual under section 'Memory Map' in the 'Flash Memory' chapter.

## 5.2 The CANloader and UserApp Headers

The headers are used to convey certain information between the CANloader and UserApp projects. The reason for the split into two separate programs is so that they are independent of each other; so the linker doesn't remap code or data location by surprise. Each project has a header at a fixed address. This is the only information that the projects have of each other.

### 5.2.1 CANloader

**a. CANloader Entry Point**

UserApp reads this to enter CANloader upon a successful reflash request.

**b. CANloader Firmware Version ID**

16 bytes.

**c. Device Unlock Code**

8 bytes; > 18 * $10^{18}$ combinations.

**d. HW Unit Device Nr**

4 bytes; > 4 million combinations.

### 5.2.2 UserApp

**a. UserApp Entry Point**

CANloader reads this to enter UserApp upon a successful checksum evaluation of UserApp.

**b. UserApp Firmware Version ID**

16 bytes.

**c. UserApp Checksum Reference**

Default testing value 0x55AA55AA.

**d. Low Address for UserApp**

Devices highest block nr start address.

**e. UserApp Code Length**

Top of App header minus the above UserApp low address.

# 6. CANloader and UserApp boot procedure

## 6.1 R32C, M16C, R8C using C-startup

The boot procedure, or transfer of control between CANloader and UserApp for R32C, M16C, R8C using C-startup files is shown below. For assembly startup versions, the procedure is roughly the same. Assembly 'labels' are used to set up code references in the headers instead of using C-references for example. Also the *focan_state_var* is setup as a separate section using HEW instead of the sect32.inc file, etc.

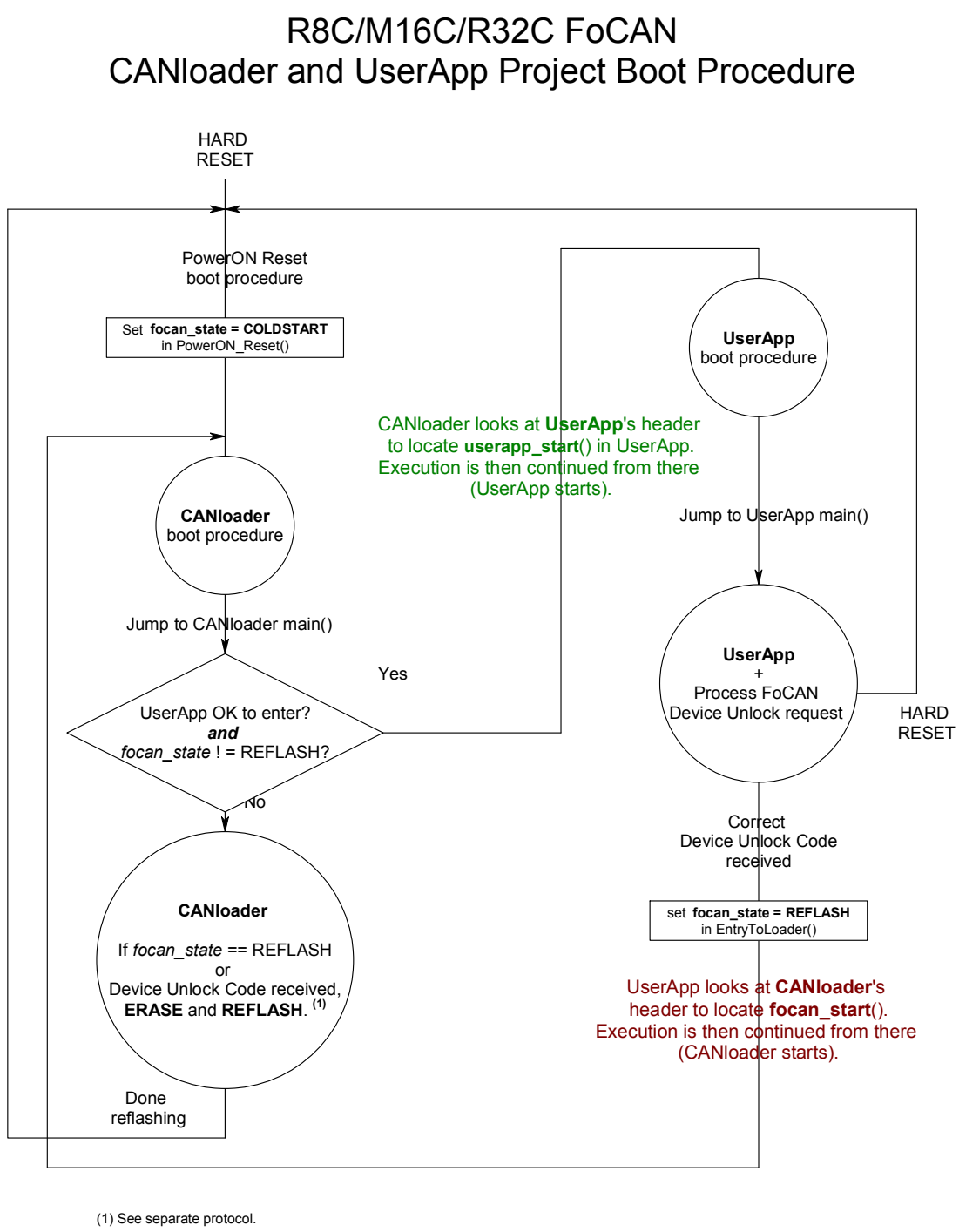


**Figure 6. Boot procedure, or transfer of control between CANloader and UserApp for R32C, M16C, R8C using C-startup files.**

## 6.2 SH RCAN-ET

The SH RCAN ET FoCAN boot procedure uses C-startup files. The Manual Reset vectors are used for transfer of control between the projects. This is so the stack pointer can be set differently in CANloader and UserApp if desired. The stack pointer in SH is set with either by dong a hard reset or manually as we shall see.

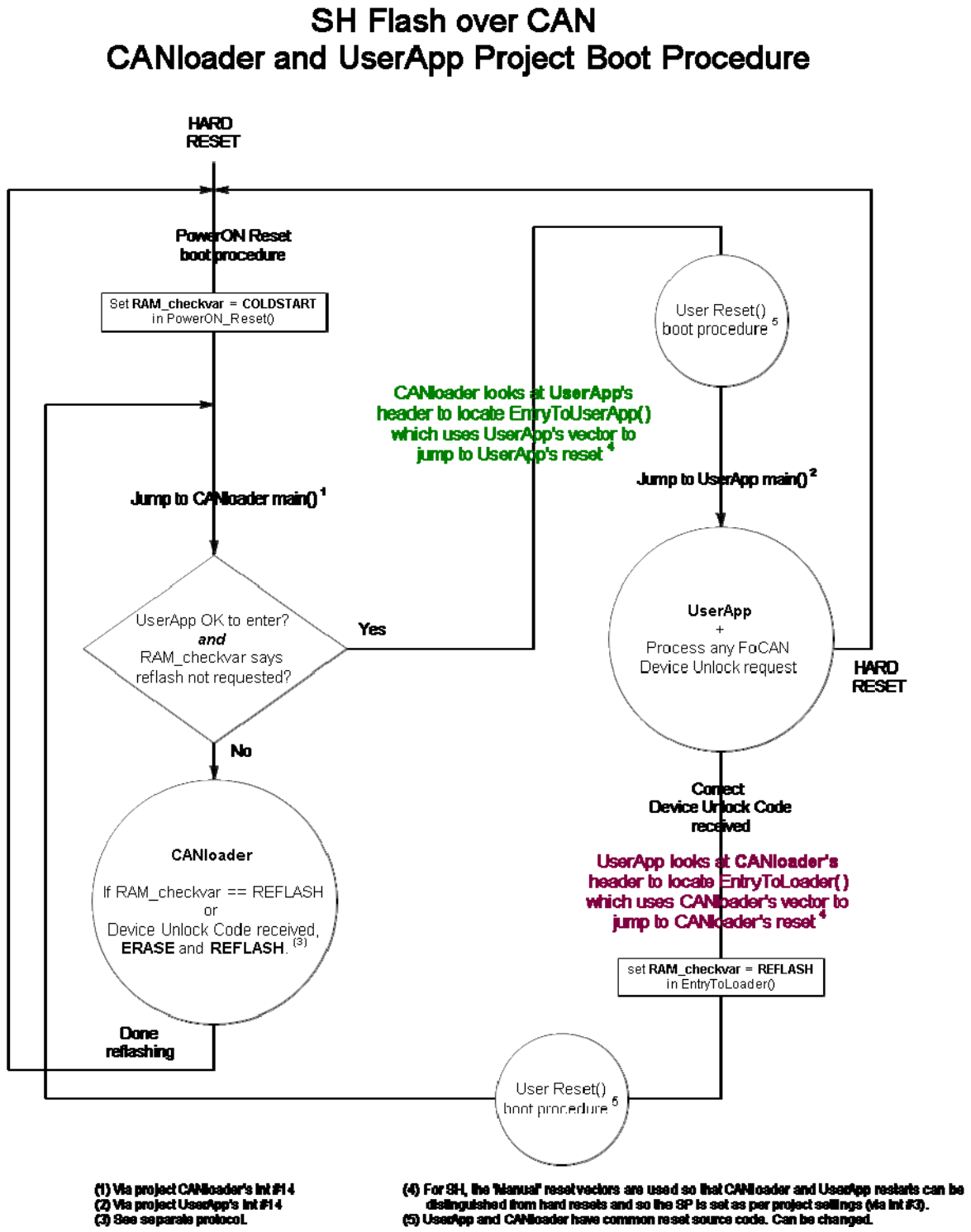


**Figure 7. For SH devices the FoCAN boot and handover procedure between CANloader and UserApp is slightly different from the R8C/M16C/R32C family C-startup in that the Manual reset vector is used when the projects transfer control to each other.**

The CANloader and UserApp Manual reset vectors are used together with the '*set_jmp*' function to transfer control between the two projects. This yields a convenient way to set the stack pointer; it will be set with the value selected by the user in the "EntryTo…" functions. The 'set_jmp' assembly function was added. This function takes as the first argument the new stack pointer (value in R4) and the address to jump to as the second argument (in R5).

Hard resets use the 'normal' Reset vector. For hard resets, the SP is set in HEW using the 'S' section address setting. This value is written to the interrupt vector address adjacent to the reset vector. This value for the SP can be used by referring to it with *RESET_Vectors[1]*. The default manual reset vector setting is in *RESET_Vectors[3]*. Any 'custom' value can instead be passed for the SP using the first argument to '*set_jmp()*'.

### 6.3 The headers and control transfer details

*This is not necessary reading -- only for the interested!*

Here follows a detailed explanation on how the source code is set up to enable the code execution to transfer control between CANloader and UserApp. The example is from R8C but is applicable to all FoCAN devices.

Each project has a header so that the other one can read key data; entry access points into the other program space, device and version IDs etc, (see header source files.) The headers' locations in flash memory space are specified by the flash_o_can.h file defines.

```
#define CANLOAD_HEAD_ADR      0xC000
...
#define APP_HEAD_ADR          0xBF00
```

Here is how we jump from CANloader to UserApp using an address specified inside the UserApp header: The actual jump consists of two assembly lines of code. The first one clears register A0:

```
asm("mov.w #0,A0");
```

The next line does an indirect jump. It jumps to the address contained in [an offset+]A0, that is, to the address *contained in* BF00+0.

```
asm("jmpi.a 0BF00h[A0]");
```

Notice the "contained in" above. The program will start executing from the address that is stored in BF00+0. (This is not the address to jump to, but the location of the address in App-header). The next question is what value is actually stored in BF00 (in the App-header)? This is determined at link time. The location of the application start address is determined by the following lines in sect30_canapp.inc.

Notice the comments after ';'

```
;-----------------------------------------------------------------
; Application ROM data and program section (M16C)
;-----------------------------------------------------------------
APP_HEAD_ADR .equ 0BF00h
.section program,CODE    ;Tell the linker that we are now defining a new flash memory
                        ;section named 'program' and it will contain actual program code.
.org  APP_ADR             ;This is the starting address where the code should reside.
.glb _app_mem_start       ;This is just telling the linker to first of all place a label
_app_mem_start:           ;'_app_mem_start' at the beginning of the section.
```

The next piece of the puzzle is filling the App-header at link time with the values we need. Go to app_head.c: (The lines are numbered for reference)

```
0. #pragma SECTION rom app_header
1. const uint32  app_entry_addr = (uint32) &app_entry;
2. const uint8   app_id[0x10] = "APPLIC_ID_123\0";
3. const uint16  app_check = 0x55AA;//0x73B8;        //0x55AA=testing value
4. const uint8 * const app_mem_start_addr = &app_mem_start;
5. const uint16  app_length = (const uint16) ((uint8 const *)&app_entry_addr - APP_ADR);
```

Line 0: The header itself is located at the end of the UserApp area. Its place is specified in sect30_canapp.inc (search for 'app_header').

Line 1: We want the address of where to execute when jumping into UserApp. This would be right at the reset address. See ncrt0_canapp.a30. Note that this is not at APP_ADR because the first code to execute in time is not necessarily the code right at the beginning of user memory.

Line 4: This 4th item of the header contains the actual start address in flash of all User Application code. Its value will be APP_ADR. See explanation for sect30_canapp.inc above. (".org APP_ADR" etc above.)

cent to reset. This is used by hardware to set the SP.

## 7. Checksum protection

To prevent a failed application reflashing image from running, there is an application checksum protection function added by default. All you have to do is add the UserApp checksum to the UserApp header.

When finished developing UserApp's application source code, a checksum for the UserApp firmware can be calculated as follows:

1. Press SW3 at startup to read the calculated checksum from the LCD display if you are using the RSK, or
2. Stop the debugger in CANloader where the checksum is calculated.
3. Make a note of the calculated checksum and write it to the application source code header for variable app_check_ref. Also make the calc-checksum routine of CANloader return a calculated checksum instead of the default constant test value 0x55AA (0x55AA55AA for the R32C).

F-o-CAN has a data checksum byte in the control message frames (CTRL_MSG_ID frames), but this is currently not used on the embedded side. The reason is because there already is a CRC field in all CAN data frames. Thus, it is already implemented by the CAN standard.

## 8. Debugging

### 8.1 R8C, M16C, R32C

#### 8.1.1 Debugging with an E8(A)

When debugging with an E8: If your application uses all available memory blocks (it cannot use Block 0 where CANloader is) you must in the dialog box for E8 place the target debug kernel code in block 0 above canloader at a higher address. See the map file for free space in block 0. By default most FoCAN UserApp sample code uses all blocks except Block 0. Here is an example using the M16C/29.

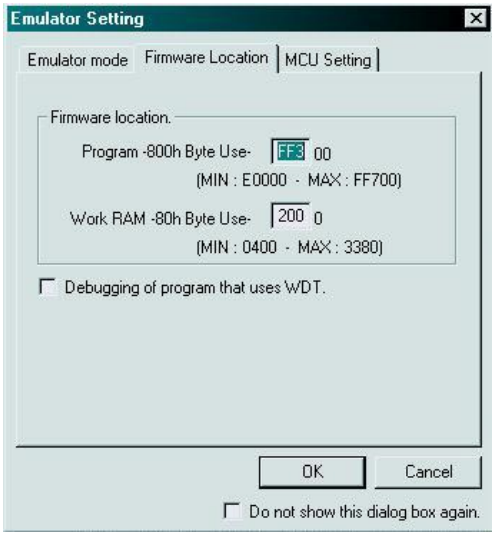


**Figure 8. You can place the target debug kernel code in block 0 above canloader at a higher address and thereby leave all of UserApp free for your code. See the map file of your CANloader project to find free space in block 0 that the debugger can use.**

### 8.1.2 Connecting with E8 and the ID Code Check Function

The default serial ID Code Check Function code for connecting to the MCU with the E8(a), not to be confused with the reflashing Device Unlock Code, is 00000000000000h, or 0xFFFFFFFFFFFFFFh. Again; this is not the same as the FoCAN Device Unlock Code.

*Always take note of the programming ID code is shown in the confirmation window when you flash the device. It should be* 00000000000000h, or 0xFFFFFFFFFFFFFFh *unless you have set it to something else.*

If the default value was not used, and you cannot remember the programming code, read the CANloader.mot file that you used to program the device with, at the ID code addresses. These addresses are found in your MCU HW manual. It will specify at what addresses the ID code values lie. Again, this is not the FoCAN Device Unlock Code.

Example; In a certain project using the R8C/23 one could read the following ID code values by reading the mot file at the addresses in the second column:

| ID byte | Address | Value read |
|---|---|---|
| ID1 | FFDF | 00 |
| ID2 | FFE3 | 00 |
| ID3 | FFE8 | 00 |
| ID4 | FFEF | 00 |
| ID5 | FFF3 | 00 |
| ID6 | FFF7 | 00 |
| ID7 | FFFB | FF |

In this case, use the ID-code 000000000000FF to unlock the device.

Tip: Use the power supply from E8 if there are problems unlocking.

### 8.1.3 Debugging the Application Standalone

You can debug UserApp with the E8(A) debugger standalone without CANloader. That way, CANloader does not have to be downloaded when debugging. Use the 'Debug' session in UserApp to do this. Using this session the Reset vector of UserApp (otherwise not used) is downloaded so CANloader is not needed.

To do this sith the FoCAN assembly startup versions, set

```
DEBUG_APP    .equ  1
```

in sect30_canapp.inc. The reset vector will now point to the Application memory start, and not CANloader's.

In addition, for both UserApp and CANloader projects, use the macro

```
#define      DEBUG    1
```

in focan.h if you want to use or add debug code.

## 8.2 SH

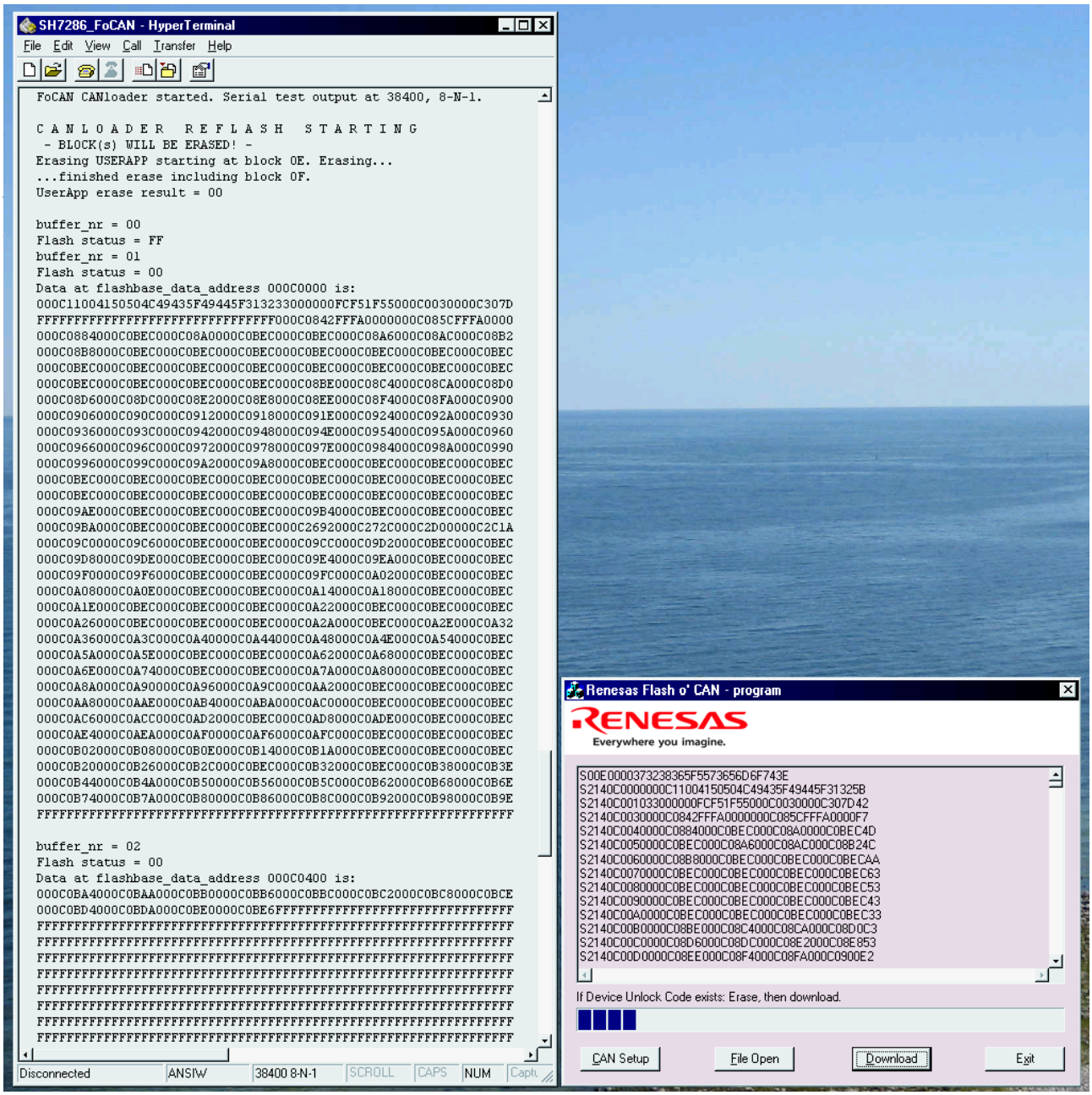


**Figure 9. SH reflashing session with a serial terminal attached. To receive data via the serial port the macro DEBUG must be defined. Flash status is default 0xFF (Firmware did not download) for each buffer before the Flash API is called. If length to flash is 0, the Flash API is not called and neither is anything output to the trace port. Status then remains 0xFF. This happens when non-continuous flash data is filling up a reflash RAM buffer.**

The intent of the serial output feature is to be able to debug flash erase / rewrites (E/W), as there are severe restrictions on debugging E/W code, and the flashing results cannot be seen. Trace functions have been added for serial trace output. These can convert constants and variables to strings so they can be output. This will be valuable for the end user to output trace messages and data, without a debugger. See source code file sci.h in the SH FoCAN project.

There is an initial flash test that can optionally be run at the beginning of a reflash session. This ‘feature’ is activated with the macro REFLASH_TEST. It enables one to see if the serial SCI port and the Flash API are functioning as expected. It reads, erases and flashes a test block with dummy data and traces the data via the serial port. Again, this test is not necessary, but a convenient debug mechanism.

For SH, CANloader needs to be taken into and out of a ‘FoCAN E/W state’. The functions *SlowClockHaltCanMaskInterrupts* and *ResumeClockCANandInterrupts* are added to do this. The clock must be slowed to max 40 MHz before erasing and flashing since flash E/W cannot run at the default 100MHz. The CAN peripheral must then be halted/unhalted by these functions so that no Error Frames are sent during flash E/W. This is because when the system frequency changes the CAN bitrate changes as well with the system clock. Also, all interrupts must completely be masked to eliminate ANY interrupt during E/W. Slowing the clock is done by the Flash API anyway, but the frequency must nonetheless be restored by the latter function. This function also restores interrupts and the CAN peripheral from halt.

Note that the CPU clock change during flash E/W disrupts the serial output from clocking data out with the correct speed. A short delay after trace data is output was therefore added in some places so the SCI peripheral has time to clock out trace data before clock speed changes.

### 8.2.1 Debugging the Application

To debug the Application using E10A, CANloader must be present.

For the SH workspaces, the E10A debugger cannot be used while reflashing with FoCAN.

## 9. The Download Protocol

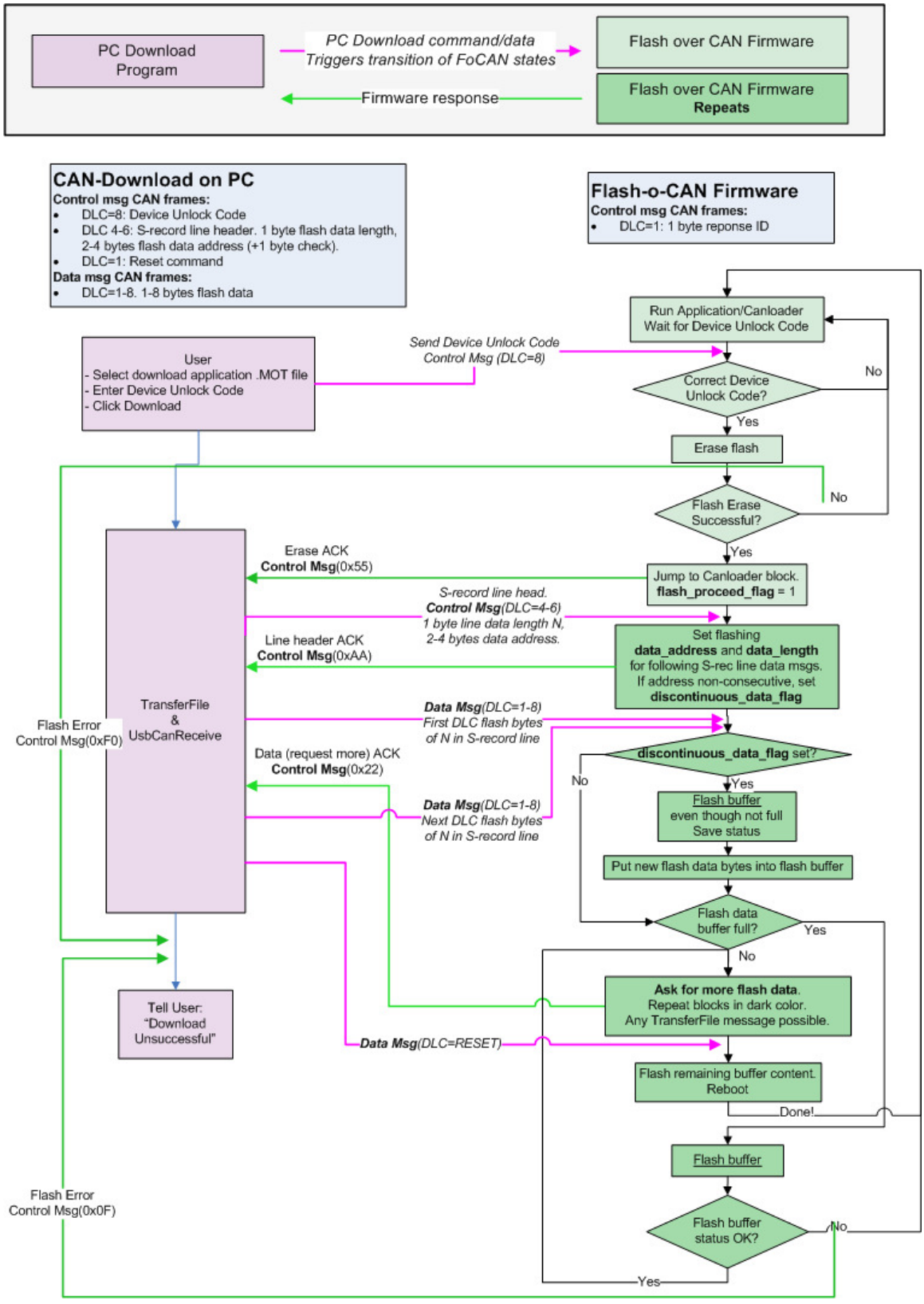


**Figure 10. The Flash-over-CAN firmware download interaction process (protocol).**

# 10. The SREC Format

An SREC format file consists of a series of ASCII records. All hexadecimal (hex) numbers are Big Endian. The records have the following structure:

**Start code**: One character, an *S*.

**Record type**: One digit, *0* to *9*, defining the type of the data field.

**Byte count**: Two hex digits, indicating the number of bytes (hex digit pairs) that follow in the rest of the record (in the address, data and checksum fields).

**Address**: Four, six, or eight hex digits as determined by the record type for the memory location of the first data byte.

**Data**: A sequence of 2*n* hex digits, for *n* bytes of the data.

**Checksum**, two hex digits - the one's complement of the least significant byte sum of the values represented by the two hex digit pairs for the byte count, address and data fields.

## 10.1 Record Types

| Record Type | Description | Address Bytes | Data Sequence | Notes |
|---|---|---|---|---|
| **S0** | Block header | 2 | Yes | Vendor specific data |
| **S1** | Data sequence | 2 | Yes | |
| **S2** | Data sequence | 3 | Yes | |
| **S3** | Data sequence | 4 | Yes | |
| **S5** | Record count | 2 | No | Record count stored in the 2-byte address |
| **S7** | End of block | 4 | No | Address field may contain start address of program |
| **S8** | End of block | 3 | No | “ - |
| **S9** | End of block | 2 | No | “ - |

# 11. Example

S00F000068656C6C6F202020202000003C
S11F00007C0802A6900100049421FFF07C6C1B787C8C23783C600000386300026
S11F001C4BFFFFE5398000007D836378800100143821001 07C0803A64E800020E9
S1110038 48656C6C6F20776F726C642E0A0042
S5030003F9
S9030000FC

Start code Record type Byte count Address Data Checksum

# 12. Suggested Improvements to FoCAN

Today each node is individually chosen for reprogramming in-network using a unique FoCAN Device Unlock Code and CAN ID. The number of standard IDs is only 2048, but there are $18x10^{18}$ possible combinations of Device Unlock Codes. The PC application FoCAN Download asks the user for which FoCAN Unlock Code to

use, where after the user enters one and the programming proceeds with the unit in network that has the corresponding FoCAN Unlock Code written to its Canload Head firmware structure.

- There is no simple mechanism to program each product with an individual FoCAN Device Unlock Code (or programming CAN ID). Adding a mechanism to increment the Device Unlock Code for each device's firmware image without having to rebuild the code has been suggested. The Device Unlock Code could then be put on a sticker onto the product.
- Similarly, entering the checksum to the image in a post build phase is desirable – that is, add the checksum after compile/build instead of as today where it is entered before compile time. See section 7 above for how to get the application checksum.

## 13. Appendix

## 14. More Information

a. **CAN MCUs**
**Devices which can use this concept are primarily SH RCAN-ET MCUs, R32C/11x, M32C/8x, M16C/6Nx, M16C/1N, M16C/29, R8C/23. FoCAN software already exists for some of these devices.**

b. **CAN Specification Version 2.0**. 1991, Robert Bosch GmbH

c. **IEC standards 118981-5.**

d. **Systec CAN sniffer**
**GW002, or USB-CANmodul1 or 2, or all types 3204001-4.**

## Website and Support

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

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

## Revision Record

| | | Description | |
|---|---|---|---|
| **Rev.** | **Date** | **Page** | **Summary** |
| 1.09 | Jun 16 '08 | — | First edition issued |
| 1.30 | Jan 14 '09 | All pages | Major Worldwide Release. All text reedited and updated for R32C. |
| 1.31 | Jun 1 '09 | Sections 1, 2, 3, 4, 8.2, 9, and 10.2. | Updated for SH RCAN-ET. |
| 1.32 | Aug 17 '09 | Section 6, 9. | Section 6 rearranged. "R32C, M16C, R8C using C-startup" sub-section added. Figure section 9 modified. |
| 1.33 | Nov 4, 09 | Figure 1,4, 5. | Figures modified + SH map added. |
| 1.34 | Jan 1, '10 | Section "Download procedure" | Changed DLL file replace instructions. |

All trademarks and registered trademarks are the property of their respective owners.

## Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

© 2008 (2010) Renesas Technology Corp., All rights reserved.