RX600 Series

Flash over CAN

R01AN0235EU0121
Rev. 1.21
Mar 1, 2012

Introduction

Flash over CAN (FoCAN) is a method to re-flash a Renesas CAN-enabled MCU over the CAN network bus. This provides an alternative to the need for debug or serial interfaces to update the device firmware.

FoCAN uses the Systec "CANmodul" CAN bus interface (Systec order# 3204000) which is part of the RCDK, the Renesas CAN-D-Kit (Renesas order# RCDK32C). This bus interface unit is used to communicate with the MCU to be reprogrammed. A Windows based application, “FoCAN Download”, provides a graphical interface to program the MCU via the CAN network. Each Renesas CAN-enabled MCU device in a CAN network can be flashed individually, in-network using its unique FoCAN Device Unlock Code.

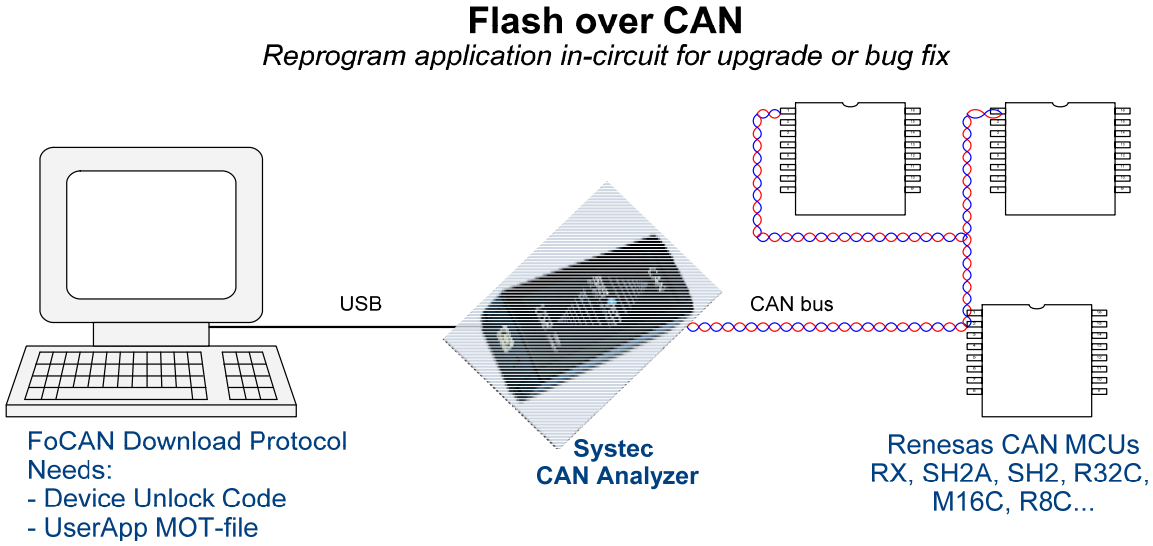

Figure 1. Flash over CAN may be used to re-flash an application after a device is installed in network

This Flash over CAN demonstration application, created with the High-performance Embedded Workshop (HEW), consists of two projects: “CANloader” and “UserApp”, both contained in a single HEW workspace. Both projects use the Renesas CAN API to communicate over the bus. They 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 region in firmware from which the other project can read key data such as FoCAN Device Unlock Code, firmware entry addresses, version IDs, and UserApp code checksum.

Within the MCU, the CANloader project manages the task of re-flashing the user application code contained in the UserApp project. CANloader is located near the end of ROM in flash blocks EB00-04 of the MCU. The user application is free to occupy the remaining flash blocks. CANloader must be flashed initially using a programmer (e.g. JLink or E1-E20).

With CANloader running on the MCU, the user application area can then be re-flashed through use of the Systec CAN Analyzer and the “FoCAN Download” GUI tool. “FoCAN Download” will send reprogramming commands and data frames through the Systec device onto the CAN bus to the MCU. CANloader will first erase the ROM blocks occupied by UserApp, then re-flash the new user application into that area. On completion, the MCU reboots, checks for data consistency of the newly flashed UserApp code, and then begins execution of the user application if successful.

When the “UserApp” application is already running, it may be re-flashed at any time by connecting the PC to the CAN bus and running “FoCAN Download”. UserApp monitors the CAN bus and knows when FoCAN Download wants to re-flash it. The re-flashing process is protected against undesired access through the use of the FoCAN Device Unlock Code. If the correct unlock code and CAN ID have been entered in the “FoCAN Download” tool, then the UserApp exits, transferring control to the CANloader, which tells the PC it’s ready and the new user application code selected in “FoCAN Download” is flashed.

This Application Note covers the FoCAN demo application as implemented for the Renesas RX600 Series of MCUs, including the RX62N, RX62T, RX630, and RX63N. Other RX600 Series devices may be added in the future.

Target Device

RX600 Series MCUs with CAN.

Other Renesas devices for which FoCAN is available include the R32C/118, M16C/6NK, M16C/29, R8C/23, and also the SH RCAN-ET MCUs; SH2A-7286, SH2-SH7137, and SH2A-7216.

Contents

General description

The Flash over CAN (FoCAN) concept reprograms Flash ROM memory using the CAN peripheral. The PC application, FoCAN Download, sends programming command frames and application content data frames via USB to the SYSTEC CANmodul bus analyzer which passes them over CAN to the MCU.

The FoCAN firmware is written in a single HEW workspace that contains two projects; CANloader, and UserApp. The CANloader project manages the task of re-flashing the user application code contained in the UserApp project. CANloader is located near the end of ROM in flash blocks EB00-04 of the MCU. The user application is free to occupy the remaining flash blocks. CANloader must be flashed initially using a programmer (e.g. JLink or E1-E20).

The FoCAN demo UserApp project consists mainly of a loop that executes continually while waiting to receive a CAN mailbox receive interrupt. The loop contains routines to blink the board LEDs and occasionally transmit CAN test data frames onto the CAN bus. It also checks the CAN bus state for error conditions. The mailbox receive interrupt callback routine is also contained in the UserApp project. This callback routine is responsible for checking the received CAN frame to see if it has the FoCAN Device Unlock Code which is used to set the application into the re-flash state. The UserApp firmware includes re-flash exit code to transfer execution over to CANloader whenever re-flashing is invoked. This feature allows the device to enter re-flash mode during normal execution of the application.

Startup operation of the FoCAN application depends on whether the UserApp portion is present already. It is possible for the CANloader to run on its own without UserApp being present. This feature permits a user application to be flashed at a later time so long as the CANloader code is resident. From a cold start, the CANloader program is always executed first, whether or not UserApp is present. among the first things that CANloader then does is to check for the presence of a valid UserApp. If UserApp is present and its checksum verifies, CANloader will then turn over execution to UserApp. If UserApp is not present or its checksum fails, then CANloader will prepare to re-flash the UserApp area.

When the “UserApp” application is already running, it may be re-flashed at any time by connecting the PC to the CAN bus and running “FoCAN Download”. UserApp monitors the CAN bus and knows when FoCAN Download wants to re-flash it. The re-flashing process is protected against undesired access through the use of the unlock code. If the correct unlock code and CAN ID have been entered in the “FoCAN Download” tool, then the UserApp exits, transferring control to the CANloader, which tells the PC it’s ready and the new user application code selected in “FoCAN Download” is flashed.

1. Reflashing Sequence

When the ‘Download’ button in the Windows PC application “FoCAN Download” is pressed, commands will be sent to the Systec CAN bus analyzer to start downloading UserApp to the MCU flash ROM. The first flash command frame sent is the FoCAN Device Unlock Code entered by the user after the ‘Download’ button was pressed on the PC. If the entered code matches the value stored in the device’s 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 responds to “FoCAN Download” confirming that the flash update may proceed.

As the re-flashing sequence commences, CANloader first erases the flash ROM blocks occupied by the user application code, and then it reprograms that area with the new specified user application code. During the download and re-flash process, only CAN frames with a CAN ID value matching the device’s CAN ID (defined in `focan.h` as: CTRL_MSG_ID) are accepted and processed for reprogramming. While not processing CAN interrupts, the CANloader routine will idle in a loop.

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[1] reference value. If the checksum calculated by CANloader does not match the checksum stored in the App-header, the CANloader does not transfer control to the UserApp, but instead will wait until a new attempt is made to flash UserApp.

Table 1. LED Activity on RSK Board During Re-flash

Color	Designator	Function
Green	LED0	User ID detected as OK
Yellow	LED1	User application or user interrupt
Red	LED2	Flash over CAN communication in progress
Red	LED3	Used by CANloader: Turns on/off when erasing or writing to flash.

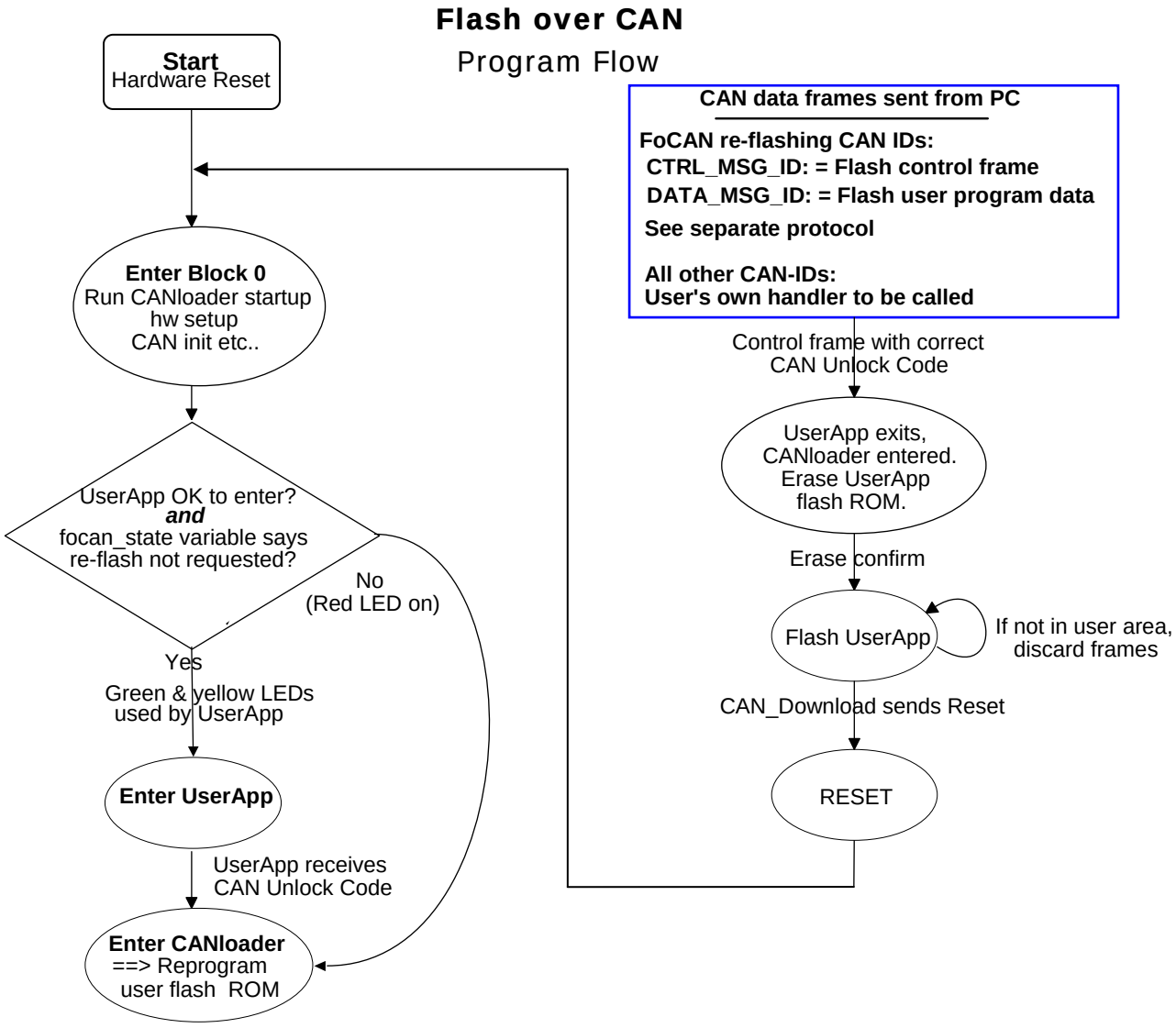

Figure 2. Program flow. To the left: Reset, CANloader's startup code, and entering the user application (UserApp). To the right is shown processing of CAN frames in UserApp and in turn CANloader if a valid download is asserted.

Notes: 1. To add a checksum to the application header you can use the default checksum algorithm or your own. This should be done when all application development activities are finished. See the section 6.

2. 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.

2. Download Procedure Using HEW

When downloading code to the MCU in the HEW environment, the CANloader project must be built and downloaded first before the UserApp is downloaded. Do not download the UserApp first, because the CANloader download will overwrite it. The HEW debug session is configured to preserve the CANloader section when UserApp is downloaded.

Both projects should be built using the DEBUG mode, done by using the macro: "`#define DEBUG 1`" in focan.h (without quotes). This will cause the CRC_AUTOGEN mode to be disabled, and, instead of a live CRC checksum, a fixed dummy checksum will be used. Later, when building release versions, the CRC_AUTOGEN mode should be used as described in the UserApp Checksum section.

1. After the CANloader project has been built select the UserApp project and set it as the current project. Build the UserApp project.
2. Connect to the board and download both CANloader and UserApp in that order. See your board's Quick Start Guide for details on which MCU device to use, and how to connect to the device.
3. Do a Reset-Go after download is done. CANloader should now enter UserApp if UserApp is programmed correctly and the checksum matches. The checksum is forced to 55AA55AAh by default in the DEBUG mode.
4. When UserApp is running it should output CAN frames as shown below.

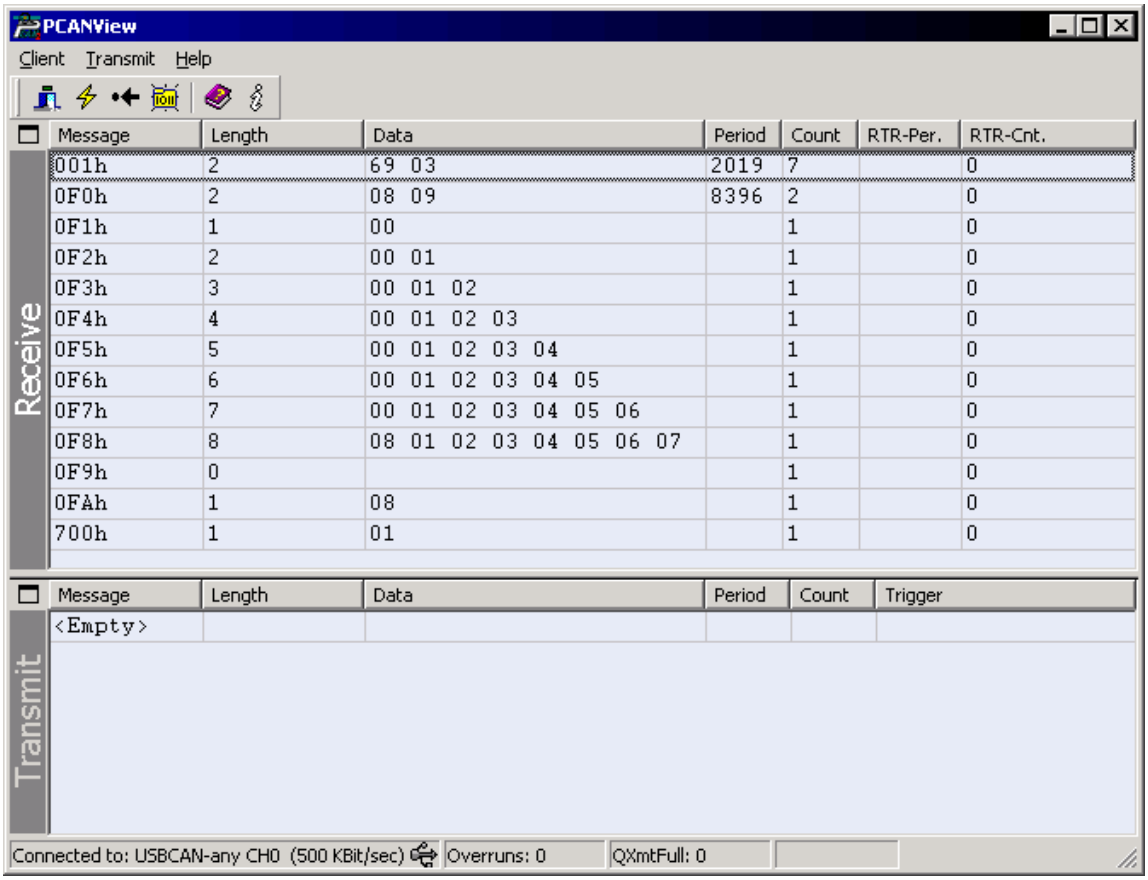

Figure 3. Default UserApp output as seen from the Systec CAN Analyzer.

Notes: 1. CANloader can be downloaded and run by itself without downloading UserApp. When CANloader is running, either because no UserApp was downloaded, or because the UserApp image failed CRC check, a red LED should light up indicating the device is in CAN 'bootload' mode waiting to be programmed.

2. When CANloader is running idle, it periodically sends a test CAN frame with ID 123h to help verify the CAN connection. This feature must be disabled in release code for devices in a network because multiple CAN devices must not transmit on the same ID.

3. Using FoCAN Download

1. This section describes the steps for re-flashing UserApp with the PC-based CAN_Download.exe utility and the Systec CAN Bus Analyzer, instead of using the debugger. This process requires that the CANloader code is already present in the MCU flash ROM. CAN_Download.exe is found in the Flash over CAN Renesas workspace directory.
2. Start the CANloader program running on the MCU, either by connecting in the HEW environment and performing a Reset-Go, or in a board stand-alone situation by applying power and pressing the reset button. If no valid UserApp program has already been flashed, or if the UserApp image failed CRC check, a red LED should light up indicating the device is in CAN 'bootload' mode waiting to be programmed.
3. Connect a Systec USB CANmodul CAN bus analyzer to your PC and to CAN channel 0 of the board
4. Start “CAN_Download.exe” located inside the workspace folder. If the "PcanView" application is still running from the previous section, close it. The FoCAN Download application can not proceed if PcanView is running.
5. Inside FoCAN Download, Press ‘CAN Setup’ and set Bit Rate to 500 Kbits/s (should be the default). Leave the CAN Message ID section with the default values of 200, Standard.
6. With CANloader running on the target MCU, the FoCAN Download program 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).
7. Press ‘File Open’, and in the browser window select the new application binary created from compiling UserApp. This should be found either in the Debug or the Release subfolder of the project directory in which UserApp was built, depending on the build mode that was used. The UserApp image file will be of the SREC type and will have the file extension: ".mot".
8. Press Download, then enter the FoCAN Device Unlock Code. This code is determined by the variable unlockcode[8] located in the file canload_head.c.

   The default Unlock Code is entered from the PC as <11111111111111x> where x depends on board type and is listed in 3.1.

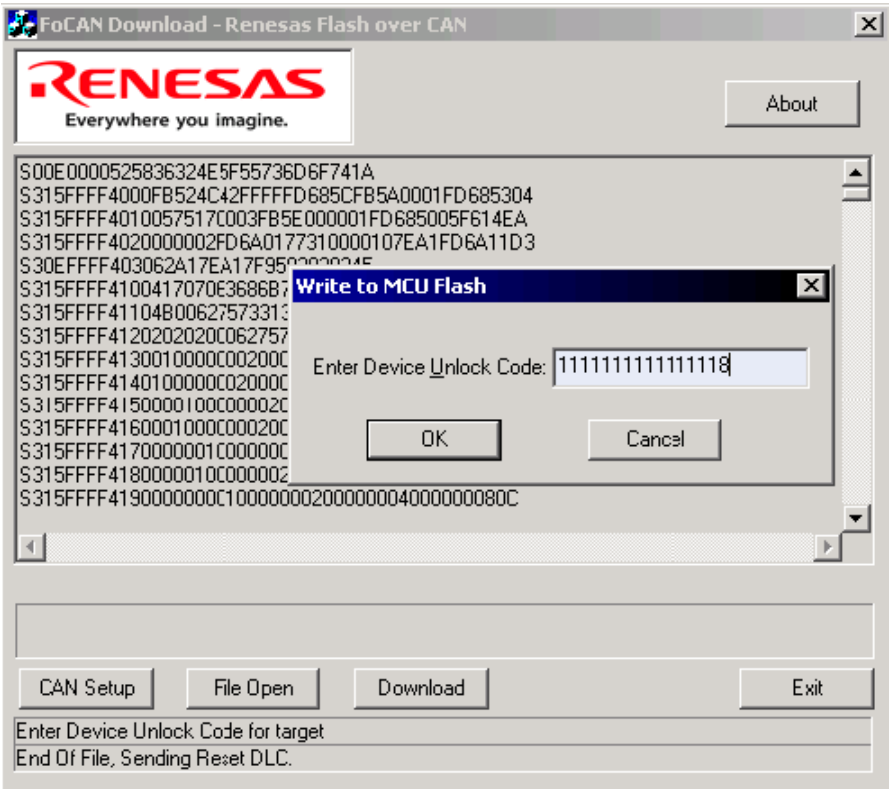

Figure 4. The PC Download application.

The FoCAN Device Unlock Code must be entered after pressing ‘Download’.

9. Press OK. After a taking a moment to erase the user area ROM blocks, the flash process will continue to download the UserApp code with a progress bar displayed in FoCAN Download showing the progress. LED2 will illuminate during the flashing process, and LED1 will flash each time a control frame arrives as the flash process continues (the LED may appear to be illuminated continuously, depending on speed of the process).
10. When the flash process is complete FoCAN Download sends the 'file complete code' to CANloader, indicating that the device is fully programmed. CANloader will then restart and proceed to calculate a checksum over the newly flashed data. The calculated checksum is compared against the value stored in the UserApp header before entering UserApp. This header code would preferably contain a valid checksum generated via the process mentioned below in the 'UserApp Checksum' section as protection against failed application re-flash'.
11. Press Exit to close the FoCAN Download tool.
12. Start the PcanView application back up. If everything is working, your CAN analyzer should show CAN bus output as in Figure 3.

3.1 FoCAN Device Unlock Code

In the source code, the FoCAN Device Unlock 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 flash programmer such as the JLink, or E1, E20 debuggers.

If FoCAN Download sends the correct control frame message CAN ID (200 by default) and the correct unlock code for that particular node, UserApp reads from CANloader's header the for an entry address to enter CANloader. CANloader then confirms to the CAN bus analyzer that it is ready for the application to be re-flashed.

Default FoCAN Device Unlock Codes:

For the RX62N: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x**18**.

For the RX62T: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x**19**.

For the RX630: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x**1A**.

For the RX63N: 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x**1B**.

4. The FoCAN Workspace

As many files as is sensible are common to both projects, such as `common.c` for functions such as setting up FoCAN mailboxes and checksum calculation, the CAN API, and `lcd.c`. However, this means that if changes are made to one of the common files, both projects will need to be rebuilt.

Each project has its own `resetprog.c` file instead of using `#ifdef`, so that debug is easier. Otherwise it could be confusing for the user to have both projects stepping in the same `resetprog.c`.

4.1 The Flash over CAN Header fields

The CANloader and UserApp projects contain header fields used to convey certain information between them. 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. Also, CANloader may be optionally run by itself with no UserApp program yet downloaded. Each project has a header information field located at a fixed memory address. This is the only information that the projects have of each other. In the CANloader project its header data is defined in the file: "canload_head.c". The UserApp project's header is defined in "app_head.c".

In addition to the header fields there is one global variable shared by CANloader and UserApp: `focan_state` . `focan_state` is used so the re-flash state is known by both projects at run time.

4.1.1 CANloader Header Members

(a) **CANloader Entry Point**

UserApp reads this to enter CANloader upon a successful reflash request. Assembly inline routines are used for jumps between CANloader and UserApp.

```
const uint32_t start_addr_cloader = (uint32_t) FoCanStart;
```

(b) **HW Unit Device Nr**

4 bytes gives > 4 million combinations!

```
const uint32_t  mac_id = 0x1234;
```

(c) **Reserved space for free use**

```
const uint32_t future_use_1 = 0x12345678;
```

(d) **CANloader Firmware Version ID**

```
const uint8_t canload_id[16] = "CANLOAD_ID_123";
```

(e) **FoCAN Device Unlock Code**

The unlock code is 8 bytes. This makes 2^{64} different combinations possible--every CAN device may have its own unique code! Each node in a CAN network is individually chosen for reprogramming based on using a unique unlock code. While the number of standard CAN IDs is only 2048, there are 2^{64} possible combinations of unlock codes. The PC application "FoCAN Download" prompts the user to enter the unlock code that will be used to select the targeted node in the network to be reprogrammed.

```
const uint8_t unlockcode[8] = {0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11,
0x18};
```

4.1.2 UserApp Header Members

(a) **UserApp Entry Point**

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

```
const uint32_t app_entry_addr = (uint32_t) UserAppReset;
```

(b) **UserApp Checksum Reference**

It is recommended not to use a real checksum while developing since it will change too frequently. Instead, a fixed value can be used to make things easier. The default value used while debugging: 0x55AA55AA..When completely done, change this to the actual checksum value. For RSK boards, the calculated checksum value can be displayed by pressing SW1 during startup. See the chapter on UserApp Checksum.

```
#if CRC_AUTOGEN
 /* Actual value is entered by the linker. */
 const uint32_t  app_checksum_ref = 0xFFFFFFFF;
#else
 const uint32_t app_checksum_ref = 0x55AA55AA;
#endif
```

(c) **UserApp High Address**

The start address of the app_head section

```
const uint32_t * const app_hi_addr = __sectop("app_head");
```

(d) **UserApp Low Address**

The start address of the UserApp section (at ResetPRG)

```
const uint32_t * const app_low_addr =__sectop("PResetPRG");
```

(e) **UserApp Firmware Version ID**

16 bytes.

```
const uint8_t   app_id[0x10] = "APPLIC_ID_123\0";
```

4.2 Memory Mapping of CANloader and UserApp

Flash memory Blocks 0-4 of the MCU contain 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.

These header addresses are the only thing the projects know about each other. The header members contain certain key information pieces that the projects must be able to read from each other. The members are at fixed relative offsets from the header start address. Besides execution entry point there is information such as firmware version number, FoCAN Device Unlock Code, and application checksum. More can easily be appended to the structure.

For project CANloader, note the location of section CANloader header in Figure 5. All CANloader project sections are placed in the region from flash Block 0 to Block 4 (for the RX600 Series).

In HEW, look at the placement of sections by opening the Build window, clicking on the Link tab and Category 'Section'. Note how canload_head and app_head in their respective projects are set apart from other code sections. Also note that UserApp has no Fixed Vector section. The file vecttbl.c is excluded from UserApp so that only CANloader takes care of the fixed interrupts such as reset for safety reasons.

When referencing flash ROM Block numbers as designated in the RX Hardware Manual, note that the block numbering order is reversed with respect to memory address order. app_head is located in the high address range of Block 5 right up against CANloader's header which is at the start of Block 4. This is to keep the headers conveniently close together and the code as 'tight' as possible.

The mapping can be rearranged as long as a header is not placed somewhere within other code sections. If located elsewhere however, the checksum calculation would be more difficult as the stored checksum reference value can not be included in the checksum calculation of UserApp.

For more on memory footprint, see your device's HW manual under section 'Memory Map' in the 'Flash Memory' chapter.

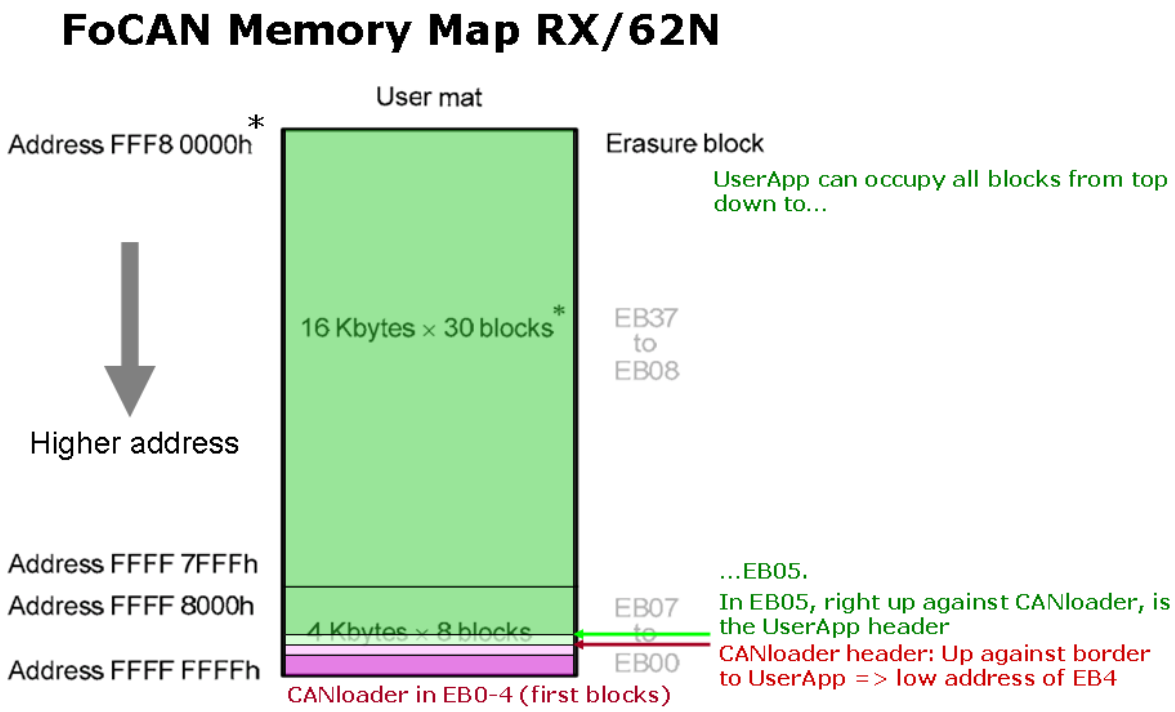

Figure 5. Flash o' CAN memory map layout for the RX62N CANloader and UserApp projects. CANloader is placed at Block 0 and up so it takes up the first 5 blocks (Block 0 - 4). CANloader is about 18 kB in size. UserApp can occupy any of the rest. Notice the location of the headers.

Product Code	Address	Number of 16-Kbyte Erasure Blocks	Erasure Blocks
R5F562x8	FFF8 0000h	30	EB08 to EB37
R5F562x7	FFFA 0000h	22	EB08 to EB29
R5F562x6	FFFC 0000h	14	EB08 to EB21

*** Variants of UserApp's low address, which depends on the RX MCU type.**

4.3 The focan.h file

Any changes to memory layout, for example used block numbers for CANloader and the UserApp projects, involves changes to focan.h. Normally you should only need to change the high APP_BLOCK_HI number. If you need to change the low block number:

- CANLOAD_HEAD_ADR and APP_HEAD_ADR must align with both the low block number chosen.
- The section addresses for app_head and the canload_head must be revised according to this low block number under menu Build->RX-Standard-Toolchain-Link tab-> Category: 'Section'.
- The absolute jump addresses in the functions JumpToCanLoader and JumpToUserApp must be changed.

For more details on focan.h, see the file's comments.

5. Debugging the Application

When downloading code to the MCU in the HEW environment, the CANloader project must be built and downloaded first before the UserApp is downloaded. Do not download the UserApp first, because the CANloader download will overwrite it. The Emulator Memory settings are configured so that UserApp does not overwrite CANloader You must change the addresses in the Emulator Memory settings dialog box (when connecting to the board) if CANloader is remapped.

If new code is being developed, or for debug purposes, Both projects should be built using the DEBUG configuration. This is set by using the macro: "#define DEBUG 1" in focan.h (without quotes). This will cause the CRC_AUTOGEN mode to be disabled, and, instead of a live CRC checksum, a fixed dummy checksum will be used. Later, when building release versions, the CRC_AUTOGEN mode should be used as described in the UserApp Checksum section.

CANloader and UserApp are set to run in Supervisor mode so that the interrupts can be globally disabled or enabled from within the code.

6. The UserApp Checksum

To insure the integrity of the re-flashed user application code, a checksum of the code is calculated and the value is stored in the UserApp header checksum field. When the application starts up via the CANloader reset vector, CANloader will validate the UserApp by doing its own checksum calculation on the UserApp code section, and then comparing the result to the stored value. If the values match, execution of UserApp can commence.

When debugging, if the checksum in the UserApp header is not inserted each time automatically, e.g. your tool does not support this, debugging can become tedious as a faulty checksum stops the MCU from executing UserApp. To use a fake, constant checksum, you can set **DEBUG** to **1** in focan.h. With this, CANloader will not calculate a checksum, and instead use the debug value 0x55AA. Likewise, the UserApp checksum "app_checksum_ref" in the app-header will assume the same value 0x55AA when it is compiled. This way, using a calculated checksum can be omitted while developing and debugging should this be desired. *Make sure to recompile both CANloader and UserApp with DEBUG set to 1.*

When finished with development, set DEBUG to 0. When downloading with the PC downloader, the Release session compile and output for UserApp must be used. CANloader must then also be compiled (with DEBUG set to 0). This will make CANloader do a CRC checksum calculation, which should then match the checksum "app_checksum_ref" in UserApp's header . *Make sure to recompile both CANloader and UserApp with DEBUG set to 0.*

This can be summarized as follows:

When **debugging**, set DEBUG to 1 in file focan.h and use the Debug session output of UserApp. *The checkum in UserApp header will then not be inserted automatically.* Canloader must also be used with DEBUG set to 1.

When **finished** with UserApp, set DEBUG to 0, and use only the Release session in HEW for UserApp (that is, use the –crc=… linker option) and CANloader. (That is, make sure DEBUG is set to 0 when both CANloader and UserApp are compiled and used.)

6.1 CRC checksum

The preferred method for creating the checksum for Release level code is to use the CRC checksum that can be automatically written to the binary image at build time. This is possible using the HEW linker option to add a CRC to the code image.

1. To activate this feature open the Link/Library tab in the RX Standard Toolchain menu: Build=>RX Standard Toolchain=>Link/Library
2. Select Category: Output
3. Select Type of output file: Stype via absolute
4. Select Show entries for: Generate CRC code
5. Select CRC code: CRC-CCITT
6. Set output address: 0xFFFFAFD4 (this is the address of header member app_checksum_ref defined in app_head.c)
7. Set CRC calculation range: Add=>Start Address: 0xFFF80000 End Address: 0xFFFFAFCF. The range of memory to run the checksum over is specified under Build options - Linker output. This needs to match what is set as the address of the beginning of the UserApp header address (APP_HEAD_ADR) in file focan.h , and the start of UserApp application flash memory (APP_LOWEST_POSSIBLE_ADR by default) also specified in focan.h. Change these depending on the location of your UserApp
8. Leave all the other entries with their default settings. Click OK to finish.
9. To enable this CRC method in the application code, set CRC_AUTOGEN = 1 in file focan.h (automatically accomplished by commenting out the DEBUG define).

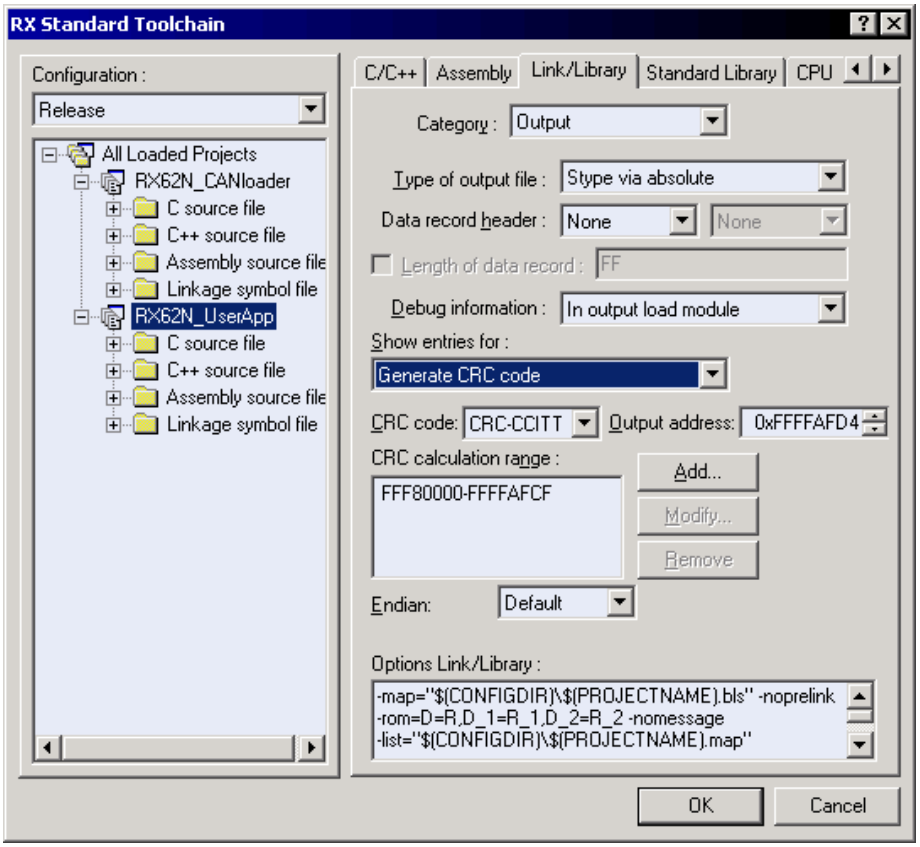

Figure 6. The range of memory to run the CRC auto generated checksum over is specified under Build options - Linker output. The result is placed in “Output address” which needs to match the UserApp reader member address for the checksum reference value.

6.2 Simple determination of checksum

As an alternative to using HEW’s CRC code calculation to add a CRC checksum to the binary automatically, there is a simple pplication checksum protection function. All you have to do is add the UserApp checksum to the UserApp header.

When development of UserApp’s application source code is completed, a checksum for the UserApp firmware can be calculated as follows:

1. Press SW1 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.
4. Make the calc-checksum routine of CANloader return a calculated checksum instead of the default constant test value 0x55AA55AA.

FoCAN 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.

7. Boot Procedure & the FoCAN State Variable

The boot procedure, or transfer of control between CANloader and UserApp is shown below. The focan_state variable is a shared variable in a common RAM section between CANloader and UserApp.

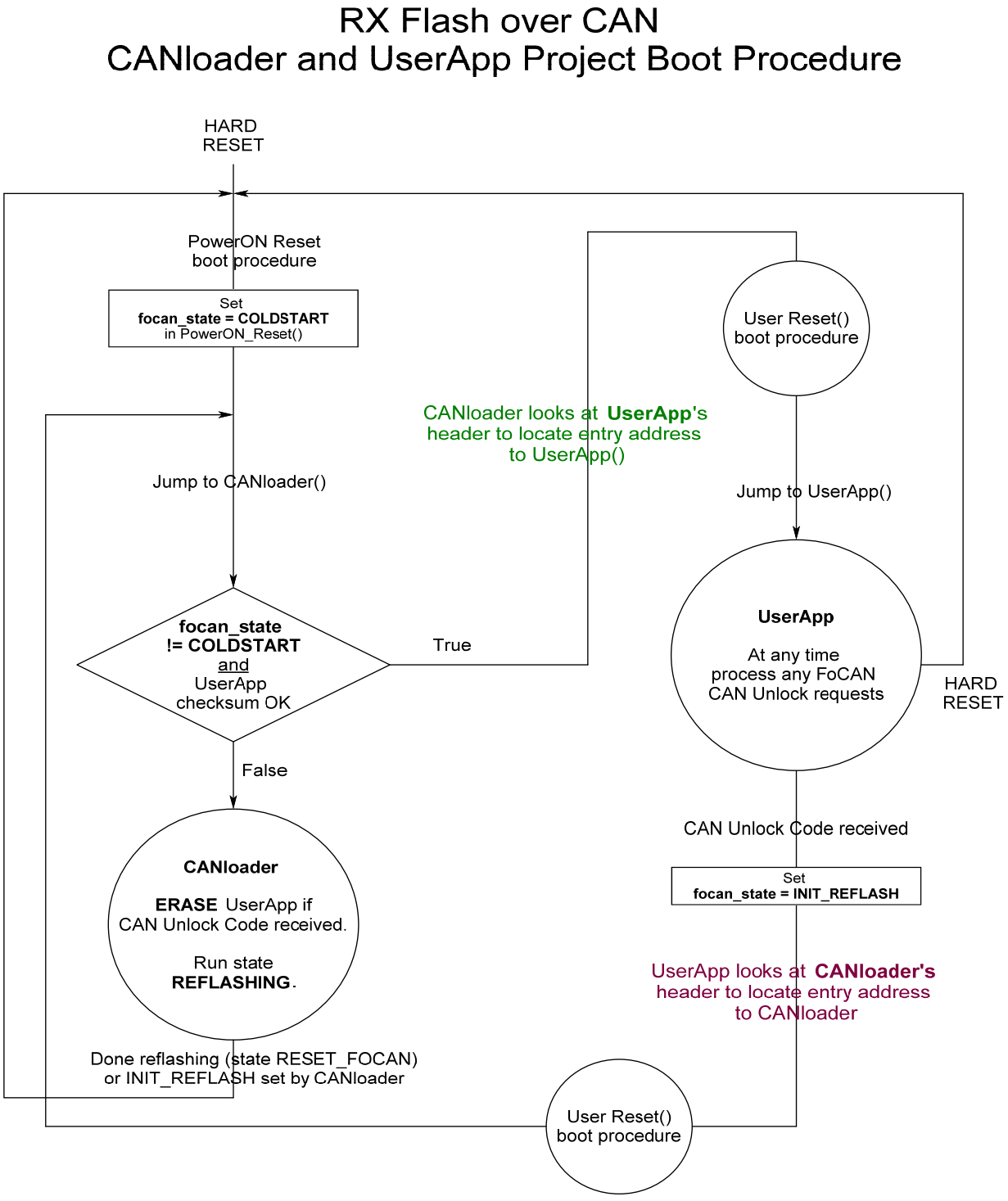

Figure 7. Flash over CAN boot procedure; how the common variable 'focan_state' is used.

8. The Download Protocol

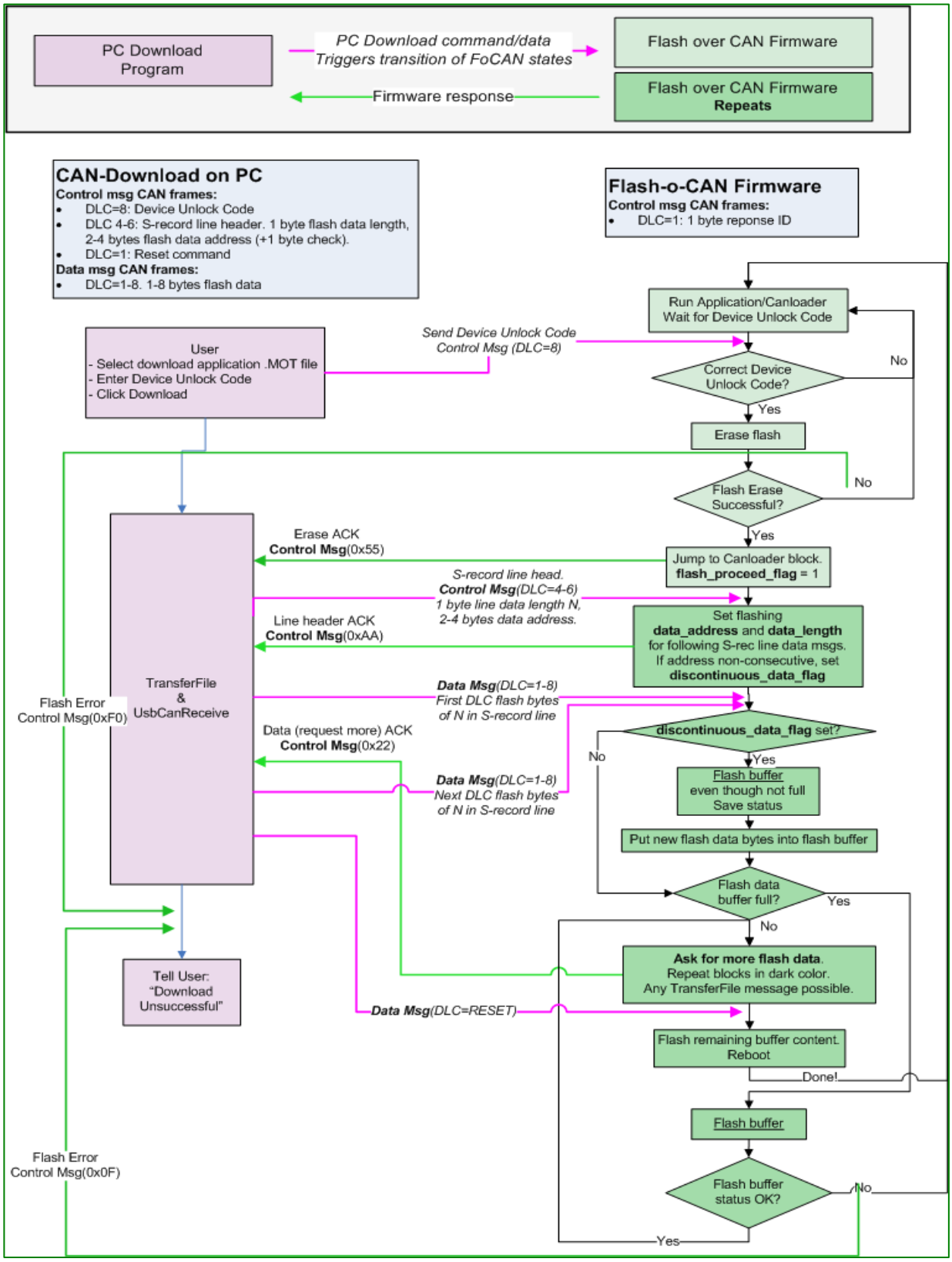

Figure 8. The Flash-over-CAN firmware download interaction process (the protocol).

9. Improvements to FoCAN

There is no automatic mechanism to program each product with an individual FoCAN Device Unlock Code. Adding a mechanism to increment the unlock code for each device's firmware image without having to rebuild the code has been suggested. The unlock code could then be put on a sticker onto the product.

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 2n 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.

Table 2. 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	" -

Example

S00F000068656C6C6F202020202000003C

S11F00007C0802A6900100049421FFF07C6C1B787C8C23783C600000386300002 6

S11F001C4BFFFFE5398000007D836378800100143821001 07C0803A64E800020E9

S1110038 48656C6C6F20776F726C642E0A0042

S5030003F9

S9030000FC

Start code Record type Byte count Address Data Checksum

11. More Information

Other CAN MCUs

Devices which use this concept and for which FoCAN software already exists are the SH RCAN-ET MCUs, R32C/11x, M32C/8x, M16C/6Nx, M16C/1N, M16C/29, R8C/23.

CAN Specification Version 2.0. 1991, Robert Bosch GmbH

IEC standards 118981-5.

Systec CAN bus analyzer

GW002, or USB-CANmodul1 or 2, or all types 3204001-4.

Website and Support

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

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

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep 11, 2010.	—	First edition issued, for RX62N.
1.10	June 06, 2011		Revised while adding code for RX62T. This edition to be valid for all devices within RX600 Series.
1.20	Nov 15, 2011		Source code for RX630 has been added (RSK) to this version.
1.21	Mar 1, 2012	Section 6	Explained DEBUG preprocessor option which is set in focan.h. Added source code for RX63N.
		All	Changed to “FoCAN Device Unlock Code” consistenly throughout.

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

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

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. 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.

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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

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

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

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

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

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

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

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

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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

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

Colophon 1.1