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.

H8S Family

Example Bootloader with Xmodem Data Transfer

Introduction

All Renesas Flash microcontrollers have the ability to self-program their Flash memory. In order to perform this feat code must run on the microcontroller to receive the data from an external source, load it into the Flash and manipulate the Flash control registers. The program that performs this task is typically called a bootloader.

Although Renesas Flash microcontrollers have a built in bootloader called 'boot mode' it can sometimes impose contraints on the system being designed. For example, the data tansfer must be via a specific serial channel on the device and the mode must be entered via setting a combination of pins to predetermined logic levels and resetting the micro. Where a different communications medium must be used such as a parallel interface or a CAN bus or where the boot code must be entered via software, a user provided bootloader is required. It is the purpose of this application note to describe the concepts behind such a bootloader and its implementation using a Renesas H8S/2636F microcontroller.

This implementation uses an SCI channel as the method to communicate with the device for simplicity. The bootloader is intended to be menu driven from a PC based terminal program with the data to be programmed transferred via the Xmodem comms protocol. This provides an easy to understand and implement demonstration although it is expected that the user may need to change this for inclusion in a final application.

The project, including all source code, is available for download with this apps note. Two implementations of the bootloader project have been produced. The first using HEW (High_Performance Embedded Workshop) and the Renesas C/C++ compiler v4.0a. The second utilises the free GNUH8 compiler (v202 HMS build) produced by KPIT. Both projects were debugged using an EDK2636F evaluation board from with the serial monitor version of the Debugging Interface (HDI) v5.01.

It is recommended that application notes REG05B0021-0100 and REG05B0022-0100 are read in conjunction with this document.

Contents

INTRODUCTION 1

THE APPLICATION 3

BUILDING APPLICATION CODE FOR EXECUTION FROM INTERNAL RAM 6

CALLING THE RAM BASED PROGRAMMING & ERASING ROUTINES 9

PROTECTION OF THE BOOTLOADER 10

THE TARGET APPLICATION 10

PUTTING IT ALL TOGETHER 12

SUMMARY 15

The Application

This application note by means of a demo application covers many of the concepts and considerations required when developing a bootloader. The hardware used is an EDK2636F with an additional RS232 driver and connector added in order to connect SCI0 to the host machine running terminal emulation software. This allows HDI-M debugging via HDI-M to be possible. Figure 1 shows a block diagram representation of the hardware.

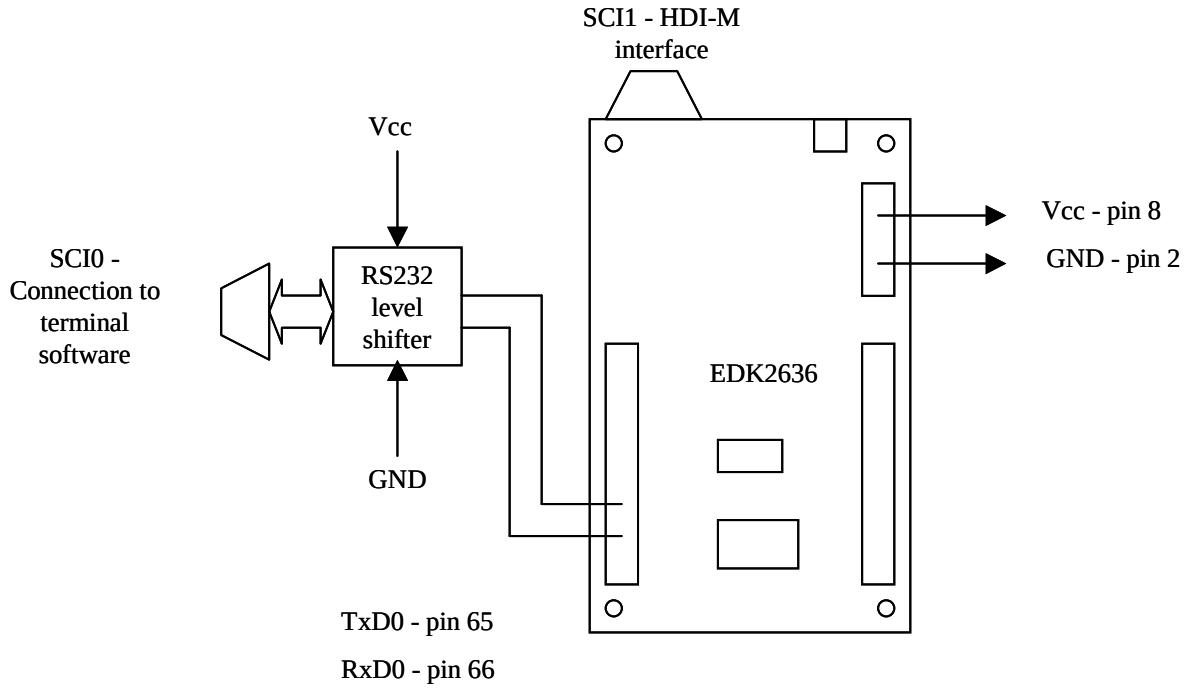

Figure 1: EDK2636 Based System Block Diagram

When the application starts it configures SCI0 to 115200 baud 8-N-1 and listens for any activity for 3 seconds. If no activity is detected then the first address in the user code area (0x8000) is read and if it is valid, i.e. not the erased state (0xFFFFFFFF), it is used as a reset vector and program execution continues at the address pointed to by this reset vector. Otherwise, if serial activity is detected it is assumed that a terminal emulator has generated the traffic, by someone hitting the spacebar for example, and a menu is displayed as shown in figure 2.

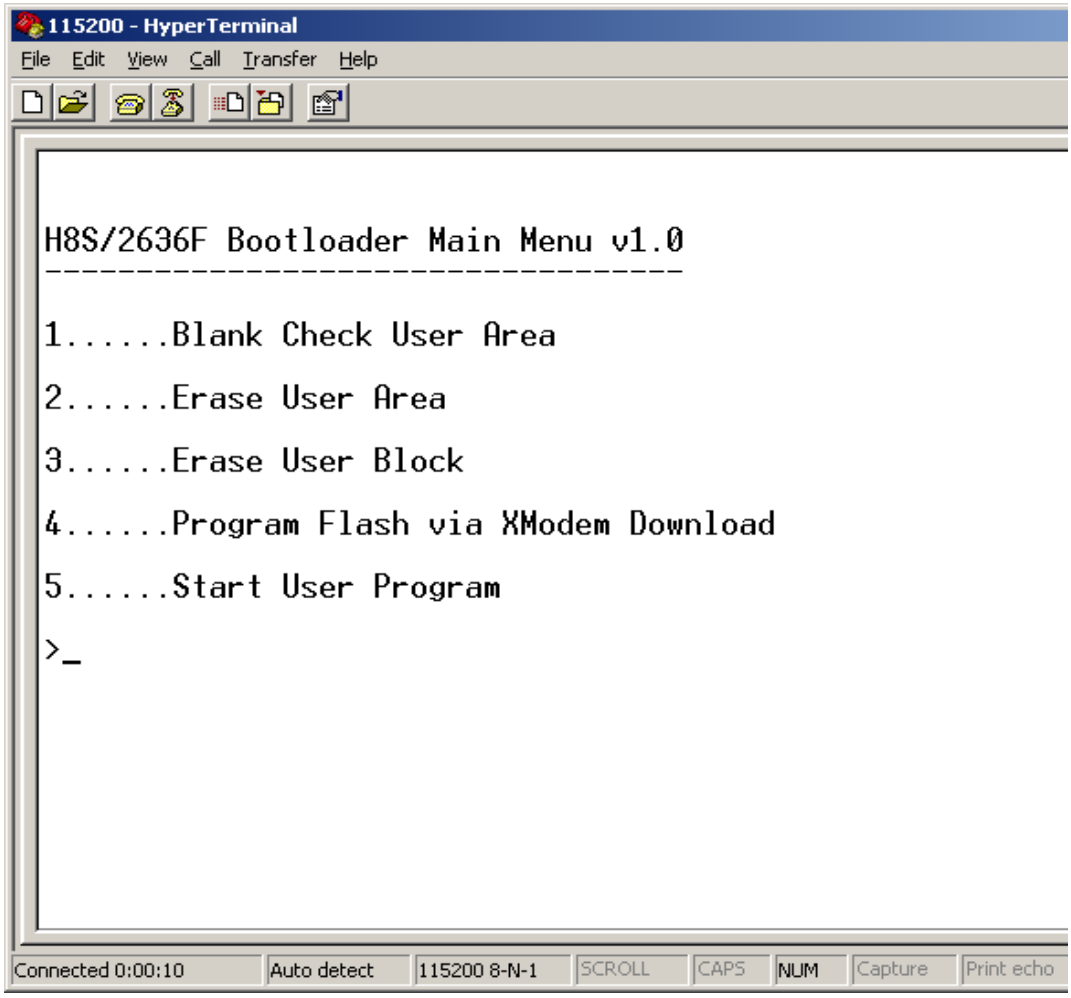

Figure 2: Bootloader Terminal Window

The menu options should certainly need no explanation to what they do but rather in how they do it, which will be covered by the rest of this application note.

Figure 3 shows the memory map of the bootloader and target memory areas.

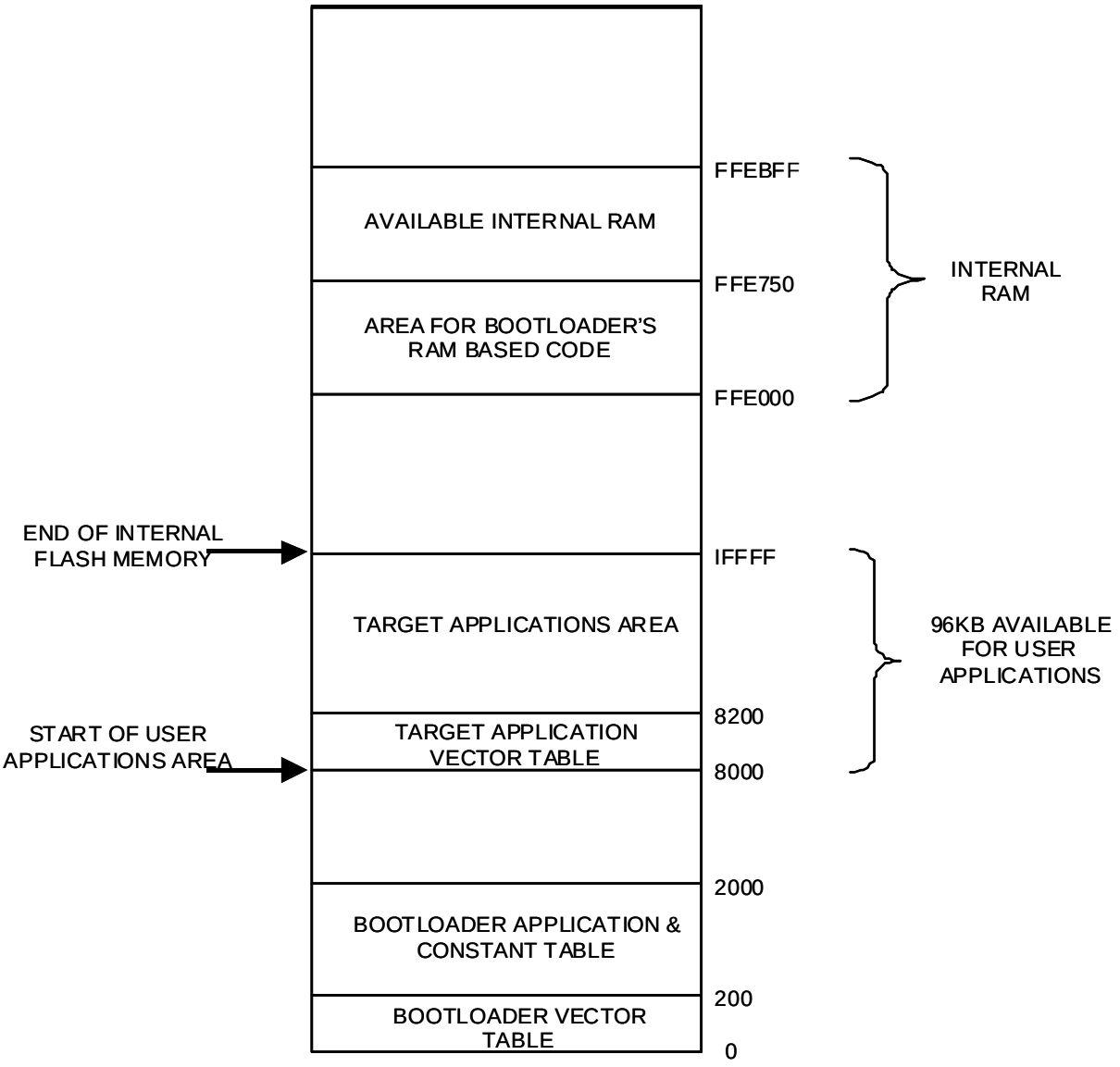

Figure 3: Bootloader and Target Application Memory Map

As can be seen from figure 3, the main bootloader application code sits at the bottom of Flash memory between addresses H'0 and H'2300. Due to the configuration of the H8S/2636F's Flash memory blocks the next available Flash block starts at H'8000. Therefore, this is the first address made available to the user's target application. Including the target application's vector table there is 96kB of internal Flash memory available to the user. When the bootloader is executing it requires up to 700 bytes (1800 bytes with GNUH8) of internal RAM in order to load the program or erase routines into as they must execute somewhere other than internal Flash. When the bootloader is not executing then all of the internal RAM is available to the user's target application.

Building Application Code for Execution from Internal RAM

When programming or erasing the internal Flash memory of Renesas microcontrollers code must be executed from outside of the Flash memory. Typically this means from internal RAM. At first the solution to this 'problem' seems straight forward enough. At runtime copy the program or erase routine from Flash into RAM and call it via a function pointer. In many cases this method will work but cannot be guaranteed. The reason being that any jumps within the code or to subroutines may refer to absolute addresses. So, the code may be executing ok in RAM and then jump back into the Flash unexpectedly. This can be avoided by using only branch statements that use offsets relative to the program counter but unfortunately with the current H8S tools there is no way to force the output of position independent code exclusively utilising branches.

The solution to this problem is to link the code that must run from RAM to the actual RAM addresses at build time. This can introduce further problems. The first is that of library routines. If a RAM based function is part of a larger project, such as a bootloader, then it may happily run from RAM but may feature calls to library routines that are linked to Flash addresses causing accesses to Flash memory at undesirable moments during execution. Even something as innocuous as the C statement below can result in a library call.

```
i = 1 << some_variable;
```

So, simply looking through the C source and avoiding calls to functions such as 'printf' is not enough to guarantee that there are no library calls to Flash based routines.

The second issue concerning copying functions from Flash to RAM is that of constant data. If the RAM routine makes reference to constant data, including things like string literals, this can cause the Flash memory to be accessed.

A third consideration is how to get code that is linked to RAM into Flash for storage at build time and then back into RAM at runtime for execution.

A solution to these problems is to place the RAM based routines into completely separate projects or builds with all code, variable and constant data linked to the RAM addresses. This eliminates the problems of jumps back into Flash for code, libraries and constant data. Getting this code from the RAM addresses into the Flash for storage at build time can be achieved by using the 'motice_cl' utility and method described in application note REG05B0021-0100.

This utility converts an s-record file into a constant C array. For example, a Flash erasing function is built as a separate project and linked to RAM. The linker is configured so that it outputs a s-record file for this project. This file is processed by 'motice_cl' which turns it into a constant C array which can then be included in the bootloader project. As the array is constant data it resides in the Flash. When the erase routine is to be called by the bootloader the constant array data is copied to the correct place in RAM and called by a function pointer. While the erase routine is executing only RAM is accessed for program code and data as this is all the routine knows about as it has been linked to RAM addresses in a separate project.

The above method relies on 3 things being known at runtime. These are:

1. The start address that the RAM code should be copied to from Flash. This is achieved by storing the constant data as part of a structure which contains the start address (put there by 'motice_cl' from the s-record) and the length of the data.
2. The size of the data to be copied to RAM so the copying routine knows how much data to move. See the explanation above for how this is known.
3. If the RAM based code contains multiple functions, e.g. erase and delay routines, the start addresses for these functions must be known so they can be correctly called via function pointers. This can be achieved by loading these addresses into a 'vector' table starting at the beginning of the RAM code area. Although the addresses of the functions may change, the

location of where the value and order of these are stored does not and is known by the bootloader. So, all the bootloader must do is read the correct address and call the function via a pointer.

Figure 4 shows the hierarchy of the bootloader project with its dependencies on the erasing and programming projects.

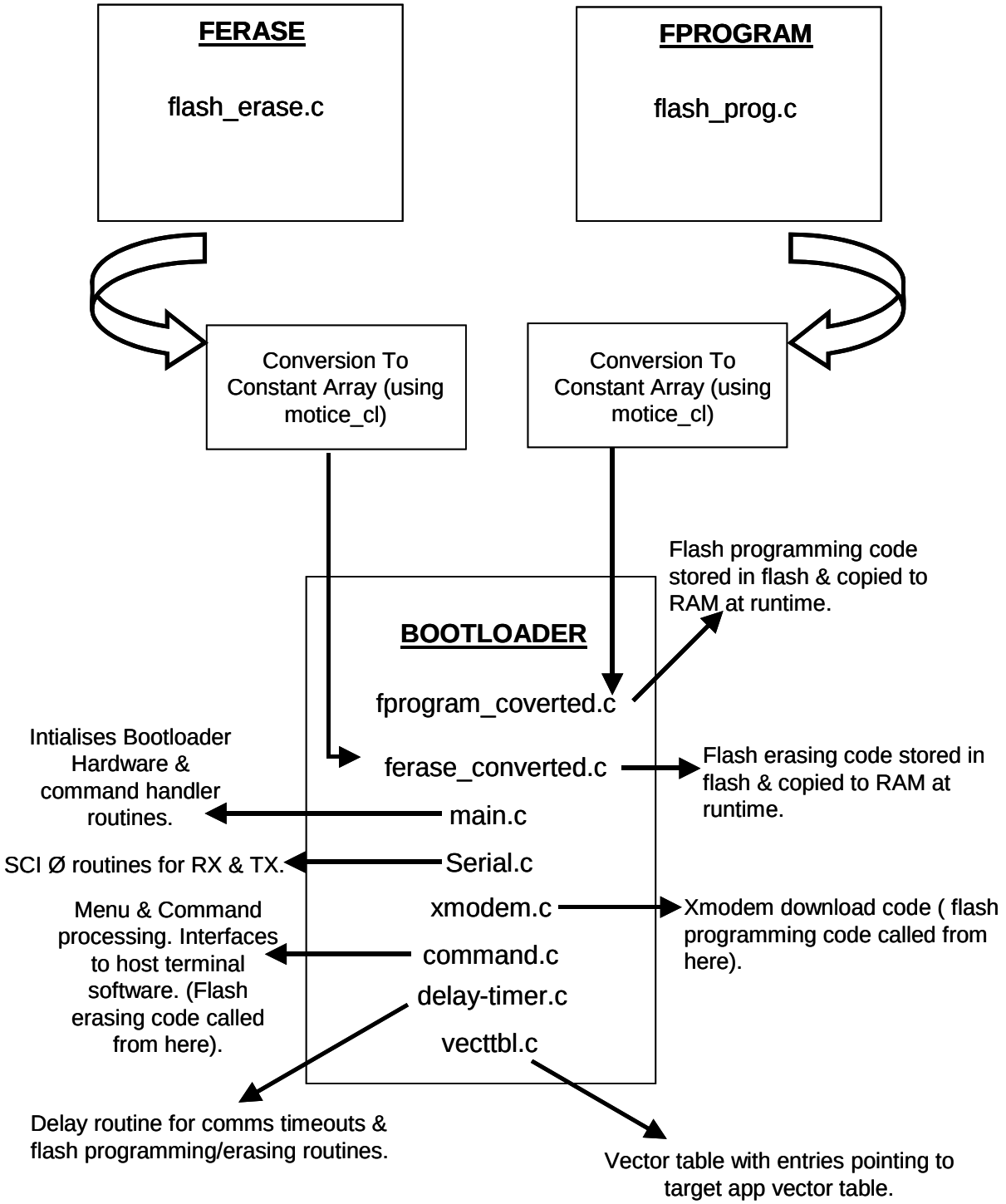

Figure 4: Bootloader Project Hierarchy

Space must be reserved in the bootloader RAM space for the RAM based programming and erasing routines to be copied to and run from. Both the programming and erasing routines are linked to occupy the same area of RAM in order to make the most efficient use of the RAM available. This is possible as both routines are not dependent on each other so they don't need to be ready for execution simultaneously.

This RAM area is reserved differently depending on the toolchain used. Using HEW and the Renesas compiler an array of the correct number of bytes is created at buildtime and linked into the correct RAM addresses. This guarantees that no other variables or code are located at the same place in the memory map. The code snippet below from 'command.c' shows the definition of the array and its location in the linker section 'PROGERASE'.

```
// storage for the program and erase routines
#pragma section PROGERASE
unsigned char ProgEraseArray[ 700 ];
#pragma section
```

Figure 5 shows this linker section configured as an uninitialised data segment in the memory map.

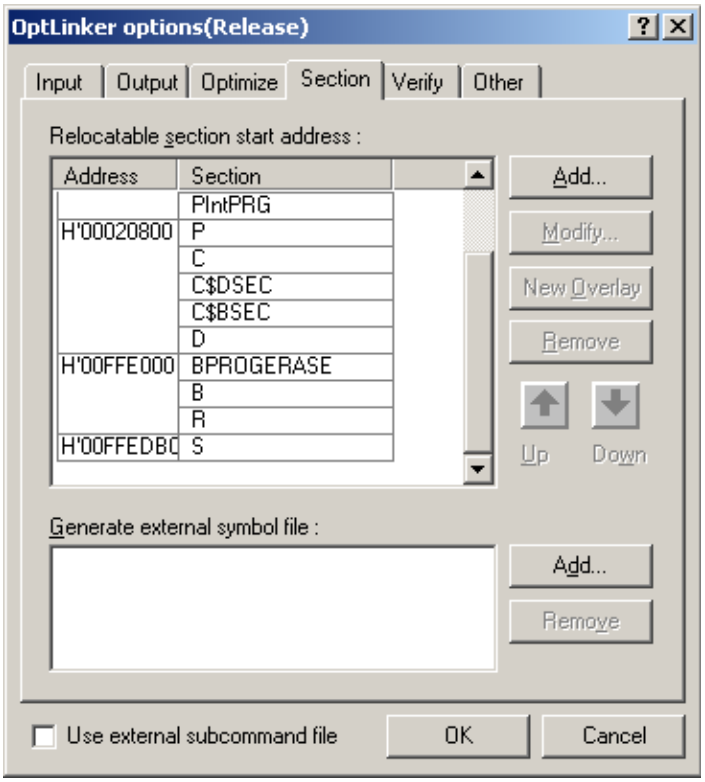

Figure 5: HEW linker sections

With the GNUH8 toolchain version of the bootloader project, space is not actually allocated by use of an array etc. In this instance the bootloader link map has its RAM starting at 0xFFE750. This leaves internal RAM addresses 0xFFE000 to 0xFFE74F free for the program and erase routines.

This method is not as elegant as the HEW solution as it depends on the user remembering not to allocate any variables to this section of memory. Also the addresses for the routines to be loaded to in RAM are hard coded into the software (see the memcpy calls in 'command.c' and 'xmodem.c').

As previously described, in point 3, a 'vector' table of function pointers is located at the start of the RAM based code for both the program and erase routines. These function pointers enable the bootloader to call the RAM based functions. The code segment below from 'flash_prog.c' shows the instantiation of this function table.

```
typedef unsigned char (*pt2Function)(unsigned long, union char_rd_datum_union
*);


#pragma section PROGFTBL
const pt2Function fp[] = {
      init_prog_delay_timer,
      prog_flash_line_128
};
#pragma section
```

This results in the address of the function 'init_prog_delay_timer' being located at 0xFFE000 and the address of 'prog_flash_line_128' at address 0xFF004. As can be seen from above the same function pointer type is used for both functions. This means that the initialisation function for the delay timer must be defined and called with the same interface as the programming function. This is to simplify the code by only having a single function pointer type and table per programming and erasing code.

A similar approach is taken with the erase project.

Calling the RAM Based Programming & Erasing Routines

Taking the programming code as an example. Running the RAM based code is a case of copying the constant data from the Flash based array to the correct locations in RAM, calling the timer initialisation function via a pointer and then calling the actual programming routine via a pointer. The code example below shows this in practise taken from the file 'xmodem.c'.

```
// copy RAM based code from Flash to RAM
memcpy( &ProgEraseArray[0], &fprog.data[0], fprog.data_length );
// call delay timer intialiser
ptr = (unsigned long *) 0x00ffe000;
fp = (pt2Function) *ptr;
// the call to the delay timer init function in RAM does not need any
//parameters but dummy parameters are passed to it in order to make the
//function pointer the same as that for calling the programming the flash //line
which does need parameters passing to it
fp( Address, (union char_rd_datum_union *) &RxByteBuffer.uc[3 + 1] );
```

```
// load the address of the program routine function pointer
ptr = (unsigned long *) 0x00ffe004;
fp = (pt2Function ) *ptr;
Status = fp( Address, (union char_rd_datum_union *) &RxByteBuffer.uc[3 + 1] );
```

The call to the programming function is passed the Flash address that is to be the first address to be programmed. The second parameter is a pointer to the data to be programmed at this address. The address of the data to be programmed is passed as a pointer to a union data type as the union allows the data to be referenced with both byte and word wide accesses. This is useful as the data must be programmed as bytes but verified as words.

The procedure for initialising and running the erase function is similar and can be seen by examining the code in the 'command.c' file.

Protection of the Bootloader

As the bootloader is responsible for erasing and reprogramming the Flash memory it is important that the bootloader code itself is not corrupted. In this bootloader example this is achieved by software. When a request is made to erase a block of Flash or to program a Flash line the block number or address is checked to ensure that it is not in the bootloader code space. Any attempt to corrupt the bootloader is rejected. In the current implementation the valid address check is performed before the function to program the Flash memory is called. A more secure method may be to check the address is valid prior to the programming pulse being applied to the Flash memory. This would help to reduce the likelihood of program runaway corrupting the Flash contents.

In some situations it may be advantageous to be able to reload the bootloader. There are a number of different methods of achieving this but one may be to load the new bootloader into another area of Flash first and then copy it over the original after it is fully downloaded. This eliminates the possibility of the Flash memory being corrupted due to the comms failing during the download. The ability to reprogram the bootloader is not covered any further by this apps note nor by the example applications.

The Target Application

The target application that is downloaded into Flash by the bootloader must be modified slightly from a standard application to take into account that it must live with the bootloader. The issues requiring consideration include those below.

1. Base address. In a normal application the Flash memory starts with the reset vector at address zero and works up from there. When using the described H8S/2636F bootloader the start address, and so reset vector address of the target app, is H'8000. This is reflected in the linker section information.
2. Interrupts. In a standard H8S application an interrupt service routine (ISR) has its address placed at the correct place in the vector table and when the interrupt is triggered this address is read and the ISR starts executing. When using the bootloader the interrupt vector table is in the bootloader's Flash area, starting at address zero, and so cannot be modified. So, when using the bootloader a jump table is necessary. The jump table takes the form shown in figure 6.

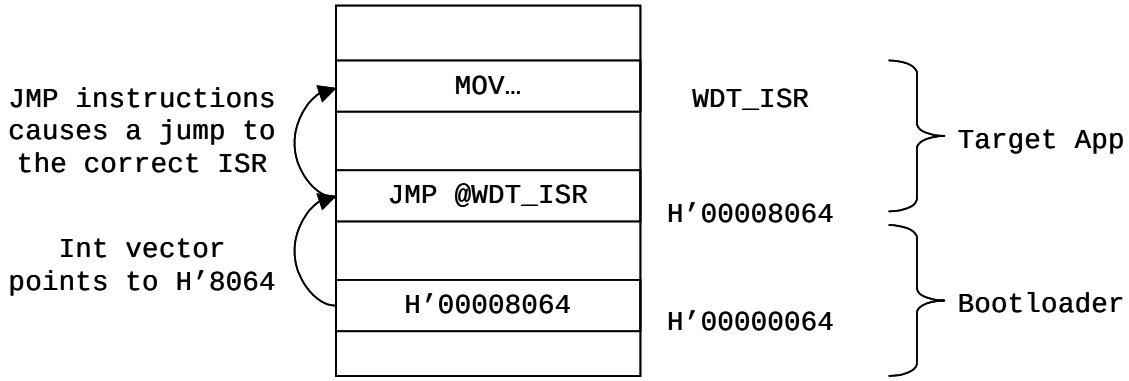

Figure 6: Use of Jump Table to Handle Target Interrupts

Each entry in the bootloader's vector table is populated with the corresponding address in the target application's vector table. For example, the watchdog timer overflow interrupt, INT_WOVI0, vector address in the bootloader H'00000064 is populated with the value H'00008064. When a watchdog overflow interrupt occurs execution will continue from H'00008064. This will cause a problem unless the contents of this address are modified. The reason being that, as expected, the target application will have the address of its watchdog timer overflow ISR in address H'00008064 but the CPU is expecting this address to contain a valid instruction as it has just vectored to it from H'00000064. So, the vector contents must be changed into a jump instruction to the target watchdog ISR. The first byte of the 'JMP @xxxxxx:24 instruction is H'5A. So by changing the most significant byte in the target vector table entry from H'00 to H'5A the entry will become a valid jump instruction and the ISR will execute as expected. Hopefully figure 6 helps to illustrate this further.

The files 'vecttble.c' (HEW / Renesas compiler) and 'vects.c' (GNUH8) show how the bootloader vector table is generated with entries to corresponding addresses in the target application vector table space.

The use of a jump table like this adds more latency to the ISR call as the jump must be performed. However, as the H8S vector table cannot be relocated there is little alternative. The SH series of microcontrollers have a vector base register (VBR) which allows the vector table to be moved and so this problem is eliminated.

3. Output format. The format of the target application's output file must be in such a form that the bootloader can accept it and extract the contents and load it into the correct Flash memory addresses. In the case of this bootloader the output file is in binary form so it can be downloaded via xmodem by a terminal program into the H8S.

The target application contained within the bootloader project files is a simple one designed to run on the EDK2636F. The program flashes one of the red user LEDs using a software delay and flashes the other red user LED using the watchdog timer overflow interrupt to generate the delay. This demonstrates the implementation of the interrupt jump table previously described.

The target project's main code is contained in the file 'wdt_main.c' in the 'user_prog' directory.

Putting It All Together

The debug build of the bootloader application is designed for use with HDI-M on the EDK2636F. The release build can be loaded directly into the Flash of the micro and operated standalone.

When the bootloader is first started something similar to the screenshot in figure 7 should be seen if the keyboard is pressed within 3 seconds of it starting.

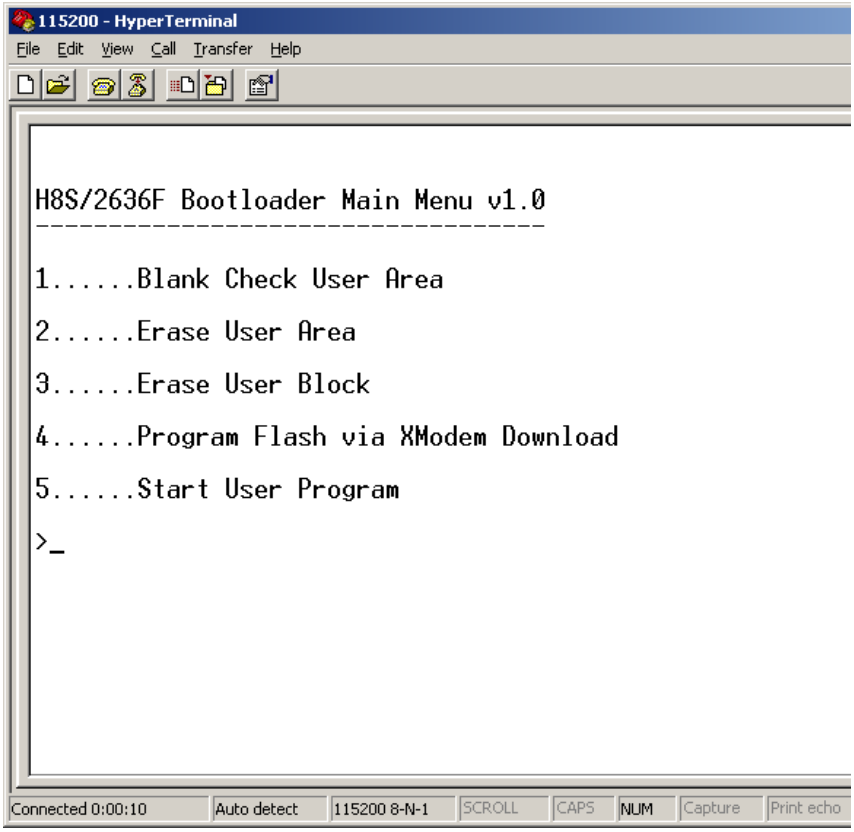

Figure 7: Main Bootloader Menu

Option 1 performs a blank check on the user Flash area as shown in figure 8.

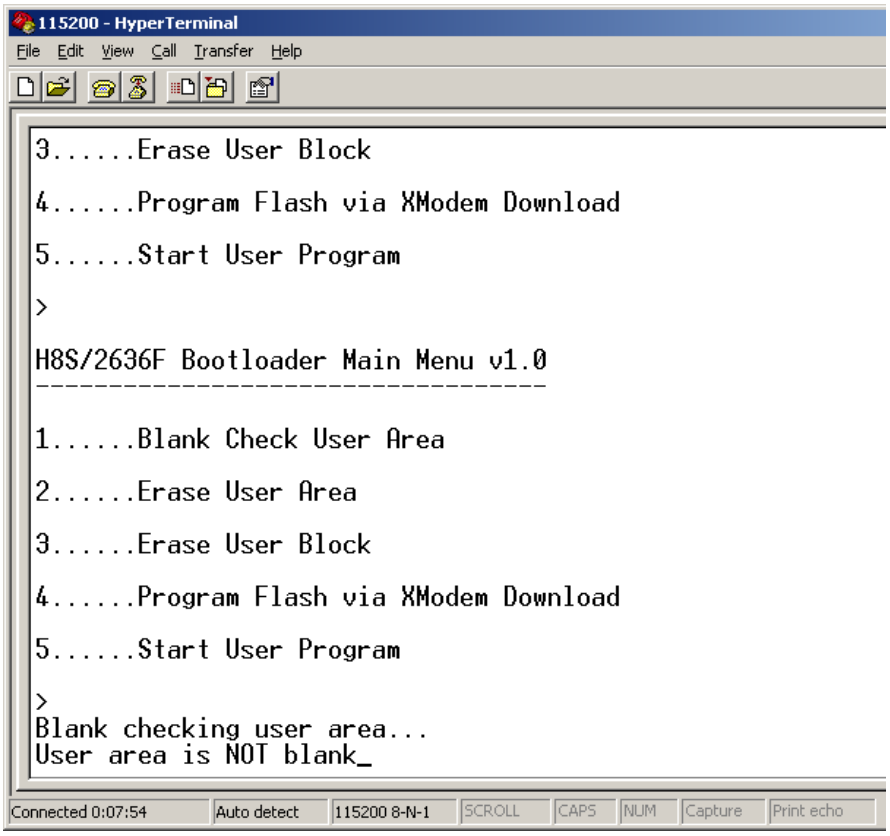

Figure 8: Blank Checking the User Area

All of the user (target) area of Flash can be erased via option 2 as shown in figure 9. Remember that the contents of the Flash must be erased before attempting programming.

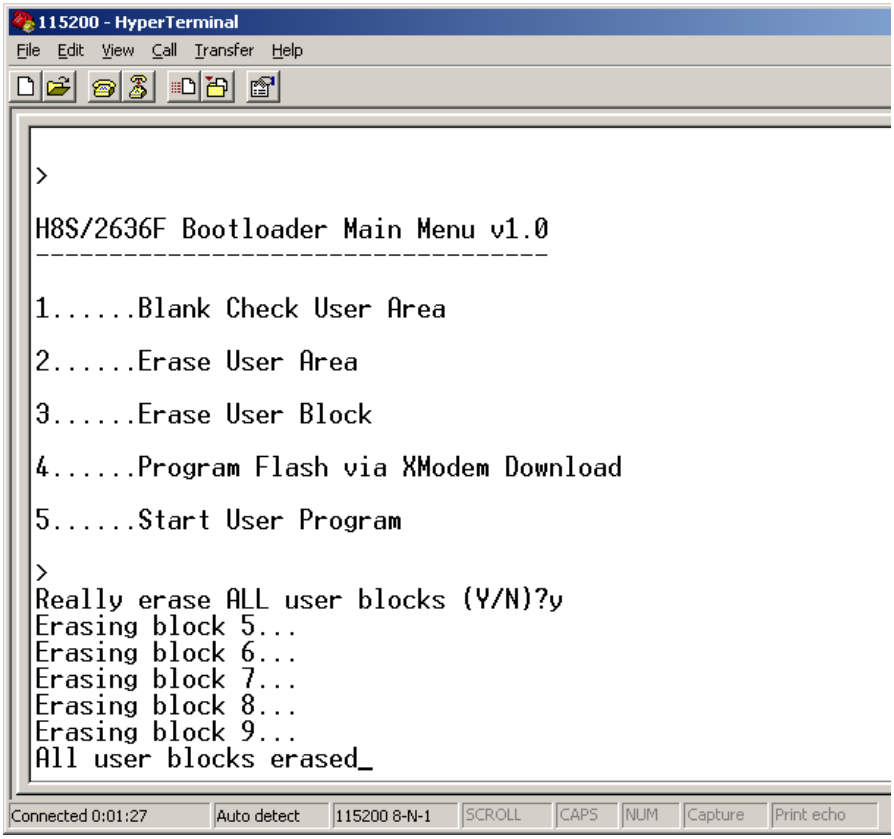

Figure 9: Erasing All User Flash Blocks

With all the user blocks erased the target application can be downloaded into the device. The screenshot in figure 10 shows that when option 4 is selected a 32-bit address must be entered which is the address at which Flash programming will begin. In the example this address is H'00008000. Here a transfer is set-up in the terminal program using xmodem which starts in response to the bootloader sending a start signal to the host. A 3kB binary file containing the target program is downloaded.

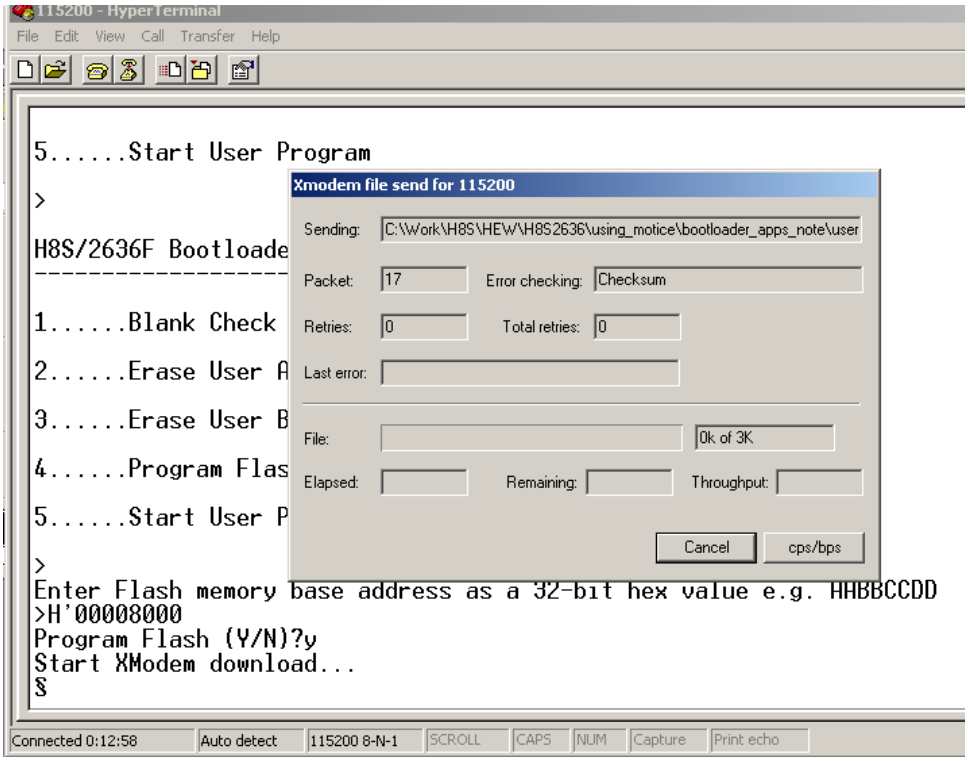

Figure 10: Downloading the Target Application Using Xmodem

With the target program successfully in the Flash memory it can be executed either by selecting option 5 or by resetting the bootloader and waiting 3 seconds without any activity on the serial channel.

Summary

Although Renesas Flash microcontroller provide a built-in boot mode there are situations where a custom bootloader is advantageous. It is hoped that this application note has gone some way in discussing the areas that need consideration when developing such a bootloader. The sample application should hopefully provide a basis for further development.

Website and Support

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

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

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. Renesas Technology Corp., All rights reserved.