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.

RENESAS

H8 Family, H8S Family, SuperH RISC Engine Family

Example of Using User Boot Mode of Renesas 0.18µm Flash Devices with Xmodem Data Transfer

Introduction

All Renesas SH and H8 Flash based microcontrollers have the ability to self-program their Flash memory. These devices have a built-in bootloader called ‘boot mode’ which enables a blank device to be programmed via a specified serial channel. The code of ‘boot mode’ is hard coded into the micro and cannot be changed by the user.

Boot mode can sometimes impose limitations on the system being designed as data must be transferred by a particular serial channel which in turn can limit the speed of transferring data into the device. In situations where a different communications medium, such as a parallel interface or CAN bus, is to be used then a bootloader must be implemented by the user. Such a bootloader for Renesas H8S microcontrollers is described in apps note REG05B0023-0100.

When a custom bootloader integrates with a target application the performance of this application can be impacted as often the interrupt vector table cannot be modified and a jump table must be used. Also, some sort of protection must be implemented to prevent corruption of the bootloader itself. Renesas 0.18µm SH and H8 microcontrollers feature an additional ‘user boot mode’ which simplifies the implementation of a custom Flash bootloader. In user boot mode the microcontroller is booted from a section of Flash memory, typically 8kB in size, which is separate from the main Flash memory. In this mode this smaller Flash block is mapped into the address map from address zero.

This application note presents the implementation of a serial Flash bootloader with data transfer via the Xmodem protocol using user boot mode for H8/3069F and SH7058F microcontrollers. The projects, including C source code, are available for download with this apps note. It is not the intention of this document to detail the Flash programming algorithms, for this it is recommended that apps notes REG05B0021-0100 and REG05B0022-0100 are studied in conjunction with this apps note.

The H8/3069F project has been produced using Renesas’ High-performance Embedded Workshop (HEW) v1.3 and the Renesas H8 compiler v4.0a and tested using an EDK3069F. The SH7058F code has been developed again using HEW v1.3 with the Renesas SH compiler v6.0a and tested using a CDK7058F.

Contents

THE APPLICATION 3

FIGURE 1: BOOTLOADER TERMINAL WINDOW 3

USER BOOT MODE.. 4

FIGURE 2: H8/3069F USER BOOT MODE APPLICATION MEMORY MAP 5

FIGURE 3: SH7058F USER BOOT MODE APPLICATION MEMORY MAP 6

BUILDING APPLICATION CODE FOR EXECUTION FROM INTERNAL RAM 7

FIGURE 4: BOOTLOADER PROJECT HIERARCHY 9

CALLING THE RAM BASED PROGRAMMING & ERASING ROUTINES 11

PUTTING IT ALL TOGETHER.. 12

FIGURE 5: ERASING ALL USER FLASH BLOCKS.. 12

FIGURE 6: DOWNLOADING THE TARGET APPLICATION USING XMODEM 13

SUMMARY 14

The Application

It is intended that the application is run from user boot mode of the device and SCI channel 1 of the microcontroller is connected, via an RS-232 converter, to a host computer running terminal software, e.g. Hyperterminal. When the application starts it configures the SCI channel to 57600 baud 8-N-1 and sends the menu shown in figure 1 to the host.

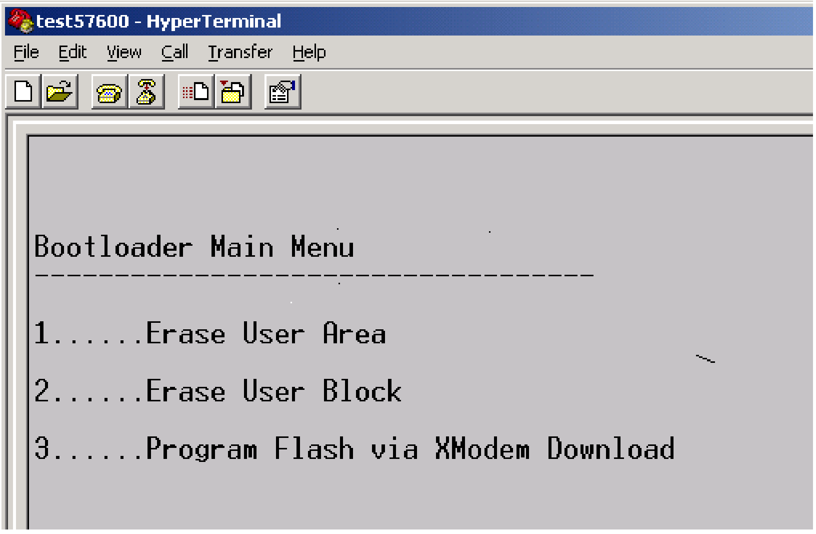

Figure 1: Bootloader Terminal Window

The bootloader menu options should not need any explanation. However, how the code behind these options functions will be covered by the rest of this apps note.

User Boot Mode

As previously mentioned, all Renesas H8 and SH microcontrollers with Flash memory based on 0.18µm technology have two built in boot modes. 'Boot mode' is the normal mode found on all Renesas H8 and SH Flash microcontrollers which enables a micro with blank Flash to be programmed. It functions by executing a program from 'hidden' memory which auto-bauds with a host computer via a specific SCI channel and then downloads a programming kernel into internal RAM which is then executed. It is this kernel which performs the programming operation on the Flash. The operation described above and the SCI channel used for the kernel download is set at the factory and cannot be changed by the user.

However, a second boot mode is available on 0.18µm Flash based Renesas H8 and SH microcontrollers called 'user boot mode'. In this mode the device boots from an 8kB area of Flash memory which is separate to the main, larger area of Flash. Using 'normal' boot mode and a utility such as Flash Development Toolkit (FDT) available from the Renesas website, it is possible to load a user written bootloader program which executes whenever user boot mode is entered. Therefore, the way the main, larger area of Flash is programmed i.e. gets it data, can be completely user defined. The advantage of using this user boot mode is that the program that resides in the main area of Flash does not have to have any limitations imposed on it as it does not have to live with a bootloader. Such limitations are described in apps note REG05B0023-0100. Also, the limitations of normal boot mode such as using a specific SCI channel are not imposed on the bootloader. It is as easy to produce a bootloader that receives its data across a CAN channel as one that uses standard serial comms. In addition, problems of protecting the bootloader are reduced as its Flash can only be erased and programmed via 'normal' boot mode. So, as long as 'normal' boot mode is not entered and intialised the user boot mode Flash will be protected.

For clarity it is worth stating that the user boot mode 8kB area of Flash must be programmed from 'normal' boot mode before any code can be executed from it. In other words this area of Flash is blank at the time of manufacture. Also, both the boot mode and user boot mode areas of Flash are erased during 'normal' boot mode as a security precaution.

The memory map for this xmodem user boot mode application for the H8/3069F is shown in figure 2.

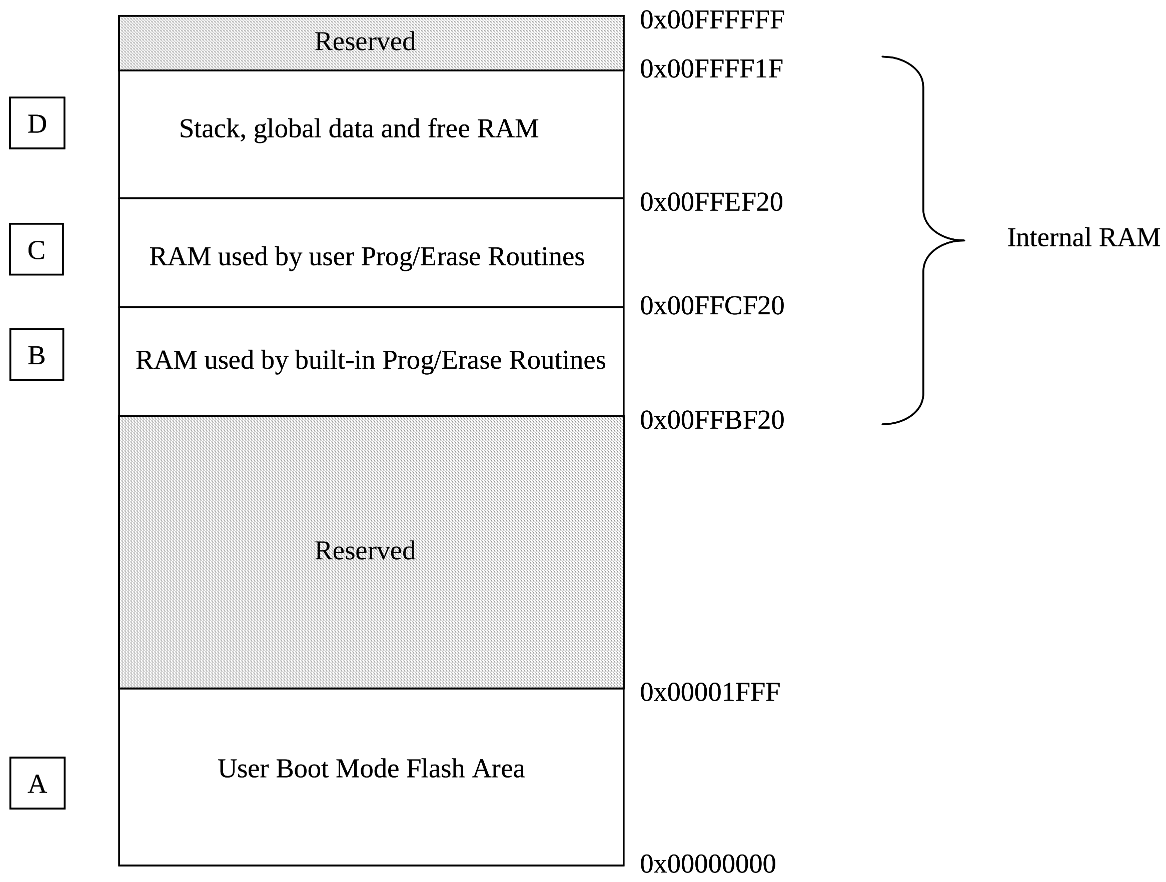

Figure 2: H8/3069F User Boot Mode Application Memory Map

Figure 3 shows the memory map for the xmodem user boot mode application for the SH7058F microcontroller.

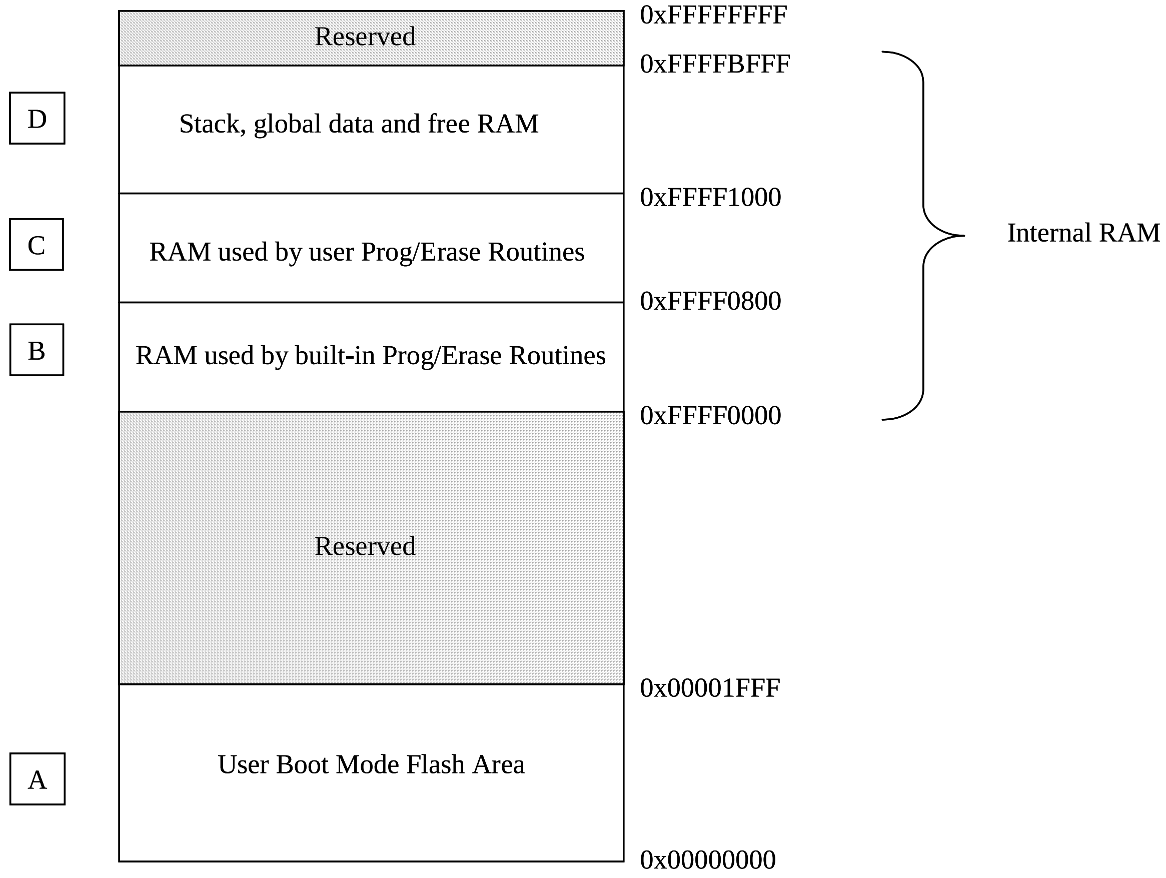

Figure 3: SH7058F User Boot Mode Application Memory Map

Section ‘A’ shown in the memory map is the user boot mode 8kB of Flash memory where the bootloader program, described in this apps note, is stored and executed from. This area of Flash can be programmed using FDT.

Section ‘B’ is a 4kB area of internal RAM which is used by the 0.18µm built-in erasing and programming routines. See the ROM section in the relevant device’s hardware manual and apps note REG05B0022-0100 for further details. The specific area of internal RAM used by the built-in routines is configurable.

Section ‘C’ is an area of internal RAM that stores user written code that calls the built-in routines in section ‘B’. The code that erases and programs the Flash memory must execute from an area other than Flash memory, i.e. internal RAM.

Section ‘D’ is the remaining area of internal RAM available to the bootloader program for global data and its stack etc.

Building Application Code for Execution from Internal RAM

When programming or erasing the internal Flash memory of Renesas H8 and SH 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 H8 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 an 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:

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.

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.

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 'function pointer' 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.

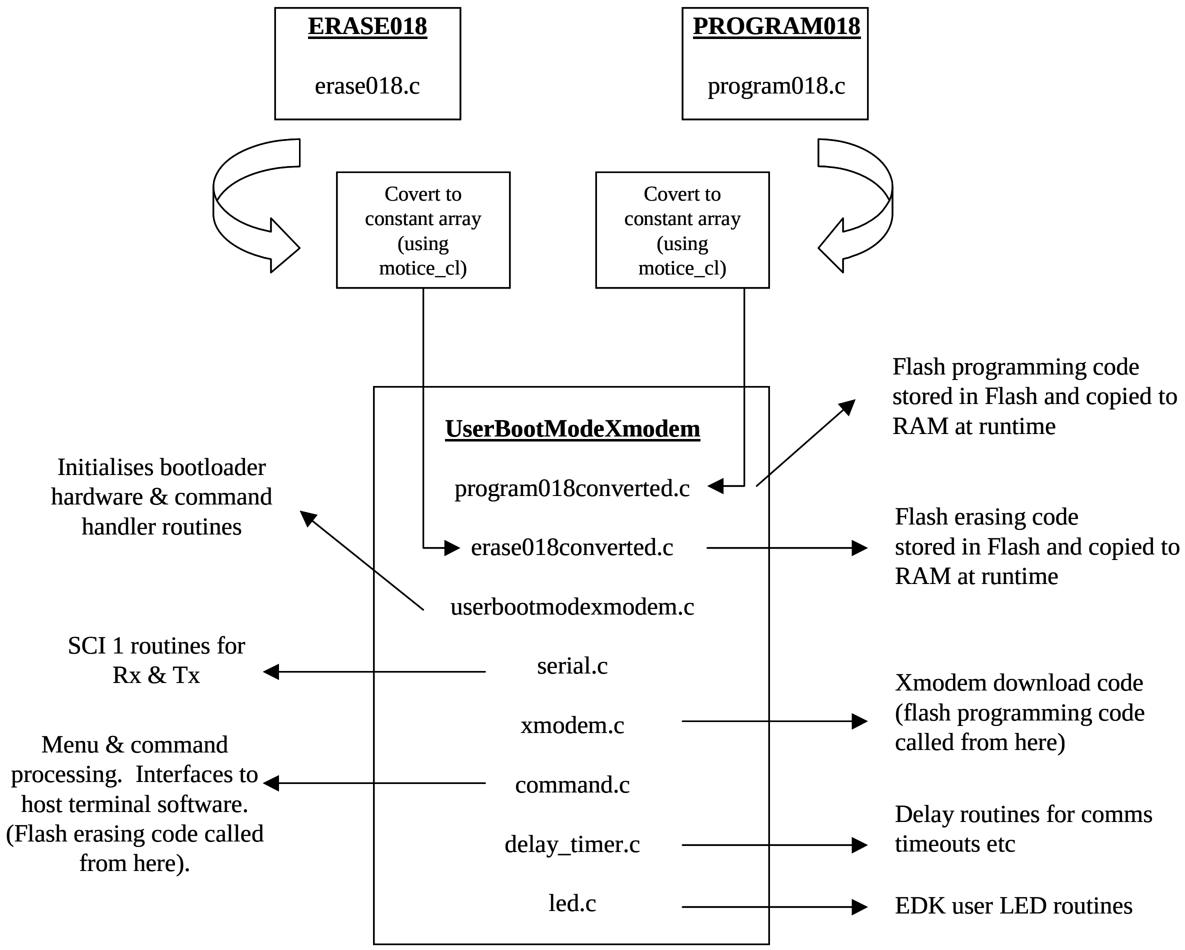

Figure 4: Bootloader Project Hierarchy

Space must be reserved in the bootloader RAM space for the section 'C' 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 available RAM. This is possible as both routines are not dependant on each other and so they don't need to be ready for execution simultaneously.

The RAM is reserved by locating a suitably sized array and locating it at the correct addresses using the Renesas H8 and SH compilers' '#pragma section' directive. A section called 'BPROGERASE' is created in the linker section map and the array, called 'ProgEraseArray' of type unsigned char, located there using the code shown below which is taken from 'userbootmodexmodem.c'.

```
#pragma section PROGERASE
unsigned char ProgEraseArray[ 512 ];
#pragma section
```

Although 512 bytes are reserved the actual code space used is less than this but this allows a margin of safety.

As previously mentioned, a function pointer table is located at the start of the RAM based code (section 'C'). These function pointers allow the bootloader to call the RAM based functions. The code segment below from 'program018.c' shows the creation of this pointer table; there is a similar code segment in 'erase018.c'.

```
#pragma section PTRTABLE
const pt2Function ptrtable[] = {
     Program018FlashLine
};
#pragma section
```

The section 'CPTRTABLE' is created in the linker section map for both the 'program018' and 'erase018' projects.

Calling the RAM Based Programming & Erasing Routines

Taking the call to the code which programs a Flash line found in ‘command.c’ as an example. Running the RAM based code in section ‘C’ is a case of copying the constant data from the Flash based array in the ‘program’ structure located in ‘program018converted.c’ into the correct locations in RAM. A function pointer to the ‘Program018FlashLine’ function is initialised and used to call the function. The code segment below, from ‘command.c’ shows this process in practice.

```
// load the program function into RAM
memcpy( (void *)program.start_address, (void *)program.data,
program.data_length );


…


// load the function pointer
ptr = ( (unsigned long *) &ProgEraseArray);
fprog = (pt2FunctionProg) *ptr;

Status = fprog( Address, (unsigned char *) &RxByteBuffer[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 procedure for intialising and running the erase function is similar and can be seen by examining the code in the ‘command.c’ file.

Putting It All Together

The release build of the bootloader application as supplied in the file downloadable with this apps note is the only valid build configuration. It is expected that the resulting s-record file from this project is loaded directly into the user boot mode area of Flash.

The H8/3069F device on the EDK3069F can be placed in user boot mode by setting the 'Boot_NMI' jumper to position 2-3 and resetting the board.

The SH7058F on the CDK7058F can be placed into user boot mode by setting all J5 jumpers to the 1-2 position and resetting the micro.

When the bootloader is first started with SCI1 connected to a host terminal running at 57600 8-N-1 the terminal should display the menu shown in figure 1. Option 1 erases the whole of the 'normal' Flash memory area as shown in figure 5.

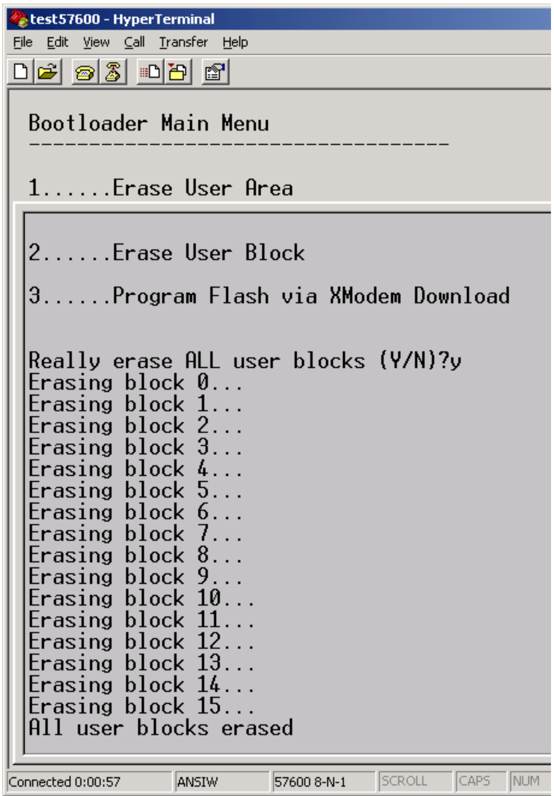

Figure 5: Erasing All User Flash Blocks

Option 2 allows individual Flash blocks to be erased. Remember that the Flash must be erased prior to programming.

With the Flash memory erased a target application can be downloaded into the 'normal' Flash memory using the Xmodem transfer protocol. Figure 6 shows that when option 3 is selected a 32-bit address must be entered which is the address where Flash programming will begin. In figure 6 the address is H'00000000. The transfer is set-up in the terminal program and the Xmodem download starts in response to the bootloader sending a start signal to the host. The file downloaded should be in binary format.

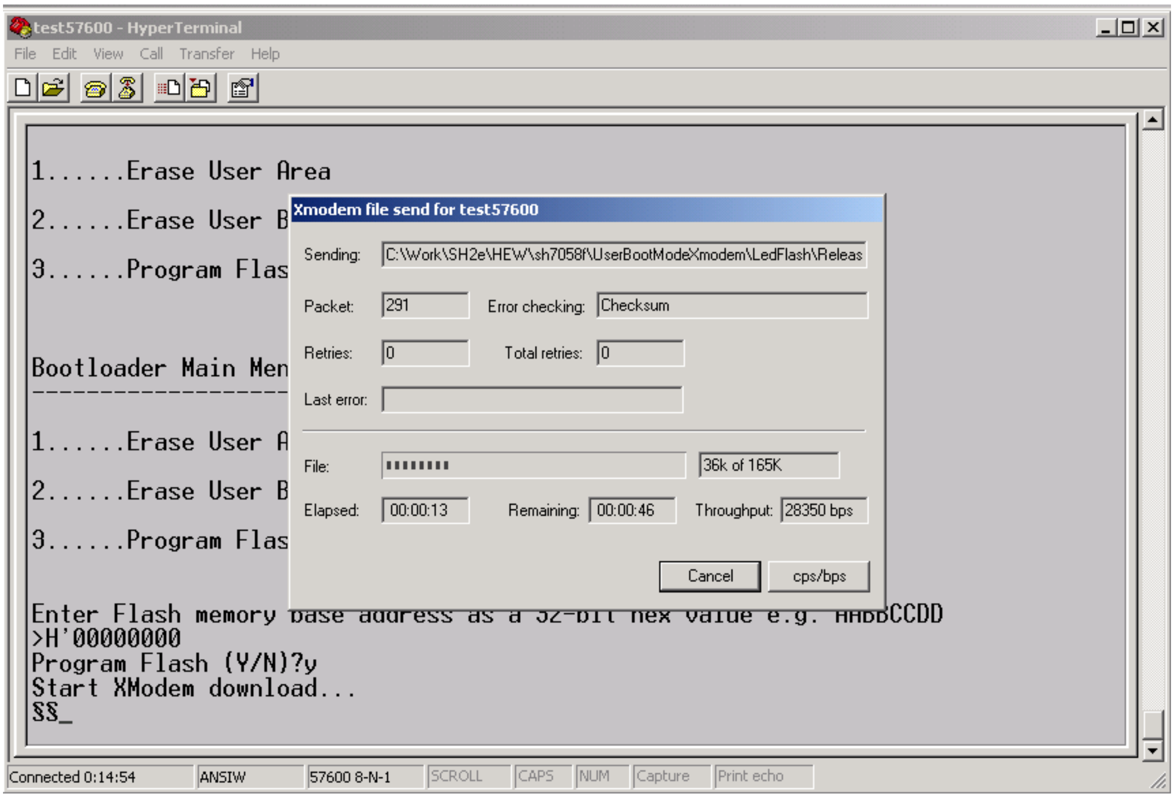

Figure 6: Downloading the Target Application Using Xmodem

The HEW workspace downloadable with this apps note contains a project named 'LedFlash' with the release build configured to produce a binary file output called 'ledflash.bin'. This test program can be downloaded into the Flash memory; if successful when the device is booted into a 'normal' operating mode with on-chip Flash enabled the LEDs on the EDK/CDK should Flash.

Summary

Although Renesas H8 and SH Flash microcontrollers have a built-in boot mode there are situations where a custom bootloader is advantageous. The 0.18μm Flash technology makes user boot mode available which greatly simplifies the development of such a bootloader. It is hoped that this application note has gone some way to discuss the implementation of a custom bootloader using the xmodem protocol for its data transfer. 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

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.