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.

User's Manual

Renesas Debugger

HMon Manual

RENESAS SINGLE-CHIP MICROCOMPUTER

Renesas Electronics
www.renesas.com

Rev.3.00 2005.09

Table of Contents

Chapter 1. Preface

Cautions

This document may be, wholly or partially, subject to change without notice.

Trademarks

All brand or product names used in this manual are trademarks or registered trademarks of their respective companies or organisations.

Copyright

Website: http://www.eu.renesas.com/

Glossary

BRR	Baud Rate Register
CCR	Condition Code Register
CPU	Central processing unit of an MCU
DK	Renesas Development Kit (RSK/3DK/EDK etc)
ERR	Error Rate
EXR	EXtended control Register
FDT	Flash Development ToolKit
GUI	Graphical User Interface
HEW	High performance Embedded Workshop
HMon	Embedded Monitor with debugging components from RTE
IRQ	Interrupt ReQuest
ISR	Interrupt Service Routine
Kernel	Flash programming code
LED	Light Emitting Diode
MCU	Microcontroller unit
NMI	Non-Maskable Interrupt
PC	Program Counter
PWM	Pulse Width Modulation
RSK	Renesas Starter Kit
RSO	Renesas Solutions Organisation.
RTC	Real Time Clock
RTE	Renesas Technology Europe Ltd.
RTE	ReTurn from Exception
SR	Status Register
Target	Hardware containing MCU used with HMon

Chapter 2.Introduction

HMon is a debug monitor that integrates into HEW. It will allow you to debug your application code in Flash and/or RAM on your target hardware. It consists of HMon components for HEW communicating with the HMon monitor code, flash programming code and your application code running on the MCU. Most Renesas MCUs include some on-chip debugging functionality, this can comprise of software interrupt instructions (TRAP) and an address break peripheral unit. The monitor makes use of the particular MCUs debug capabilities to extend its capabilities beyond that of a simple monitor. HEW debugging functionality combined with HMon code and Flash programming code enables you to run, step and set breakpoints in your application code as well as using other debugging functionality such as viewing memory and C/C++ source code.

This manual is intended as an advanced extension to the particular DK Quick Start Guide and User Manual. If you are using a Renesas DK then you should be able to run and debug the provided tutorial code without having to consult this manual. If you are using HMON in a non standard way, such as using it to connect to a non Renesas DK, then this manual is for you.

2.1.Software Installation

Please ensure that HEW, FDT and the appropriate Development Kit support for your Target, which includes HMON, are installed on the host PC, as per the supplied Quick Start Guide, before continuing.

2.2.Reference Material

This Manual refers to the HEW User Manual, the FDT User Manual, the MCU Hardware Manuals, the DK Quickstart Guide and the DK User Manual. Please have these manuals to-hand when using this User manual.

2.3.Scope of Toolchain and DK Support

This Manual is written for use with Renesas MCS toolchains for SH or H8 running under HEW3.1 or later. Any references to DKs from Renesas are specific to DKs that are supplied with the HMON monitor software.

2.4.Format of this Manual

The following bullet point shows you when to do something with a specific tool.

- Do something

The following text represents file names, directories and code examples.

```
#include    <machine.h>
```

The following text represents a menu item or dialog.

- ‘Debug | Go Reset’

2.5.Source Extracts

The source extracts contained within this manual are based upon an H8S target platform. Whilst much of the code is generic, and may be directly referenced to other platforms, there is inevitably a small amount of source that is specific to the target and it’s associated toolchain.

Chapter 3.Programming Flash

3.1.Flash Programming Modes

In order for your code to be downloaded and debugged in Flash, HMON needs to be able to Flash program the target MCU. HMON uses FDT for all flash operations. Flash programming can be achieved using two different methods described in detail in the MCU Hardware Manual. These two methods are known as Boot Mode and User Mode.

3.1.1.Boot Mode

Boot Mode is the most basic method of programming Flash on an MCU. It requires the use of a specific pre-defined port to download data to the MCU; this port is defined in the MCU Hardware Manual. For the MCU to enter Boot Mode, specific levels must be applied to some MCU input pins. When Boot Mode is entered all contents of the Flash are erased and the MCU waits for new data and programming algorithms to be downloaded via the Boot Mode port. When the MCU is in Boot Mode, the code to operate the port and erase the Flash is permanently on the MCU and cannot be changed. Once Flash programming is completed the device can be taken out of Boot Mode by removing the levels on the MCU input pins and resetting the MCU system. You can use HMON to program the MCU Flash in this manner if you so wish.

3.1.2.User Mode

User Mode is similar to Boot Mode but you have control over how programming algorithms and data are supplied to the MCU. For the MCU to enter User Mode, specific levels may still need to be applied to some MCU input pins. These levels if needed, are defined in the MCU Hardware Manual. When User Mode is entered the MCU's Flash is not automatically erased. This mode of operation only makes Flash programming achievable; it is up to you as the user, to provide the communications method, programming algorithms and application code to execute this method. Because User Mode does not automatically erase Flash, you can use it to program specific areas or blocks of Flash and thus reduce Flash programming times. It also allows you to decide the communications method. You may also use HMON with this method to program the MCU Flash.

3.2.FDT and HEW

FDT is the Renesas GUI for programming Flash. It can use both Boot and User Mode methods and provides you with pre-built and tested programming algorithms and communication code. HEW is the Renesas GUI for both debugging and building code. If both FDT and HEW are installed on a host machine, HEW will recognise FDT and incorporate a toolbar onto the HEW GUI. This enables you to program Flash on their target hardware, from within the HEW environment, via the host machine's port and the port on the MCU. If you are using HMON to program the Flash, the FDT toolbar (see below) is used to configure what MCU to program but cannot be used to program Flash. HMON is used to program the Flash using the debugger download facility.

For more information on FDT and HEW integration, please refer to the FDT and HEW User Manuals.

3.2.1.FDT Kernels

The pre-built and tested Flash programming algorithms and communication code are known as kernels and are provided with FDT in the form of binary op-code files. FDT allows you to use these kernels usually via the host machine's port to the port on the MCU. For both Boot Mode and User Mode the kernels will communicate through the specific Boot Mode port on the MCU. If you wish to use User Mode through another communications medium, the User Mode kernels must be rebuilt.

3.2.2.Selecting Kernels

When you begin debugging with HMON, the specific kernels for the target MCU must be defined. To choose the kernels to be used by HMON, you must run the FDT Flash Kernel Wizard using the icon shown here:

The method for selecting the kernels is described in more detail in Section 5.2.4 and in the FDT User Manual. Note that the name and file extension are used to identify the correct kernel for the specific target.

3.3.HMON and Flash Programming

The HMON system can use either Boot or User Mode Flash programming methods by automatically interfacing to FDT via HEW. Thus FDT and HEW must be installed so that HMON can program Flash. You can set FDT to use either boot or user mode from the FDT wizard or from the HMON Configuration Dialog on the FDT Tab.

HMON uses a configurable memory map of the target to know which areas require flashing, see section 5.5 Memory Map Configuration.

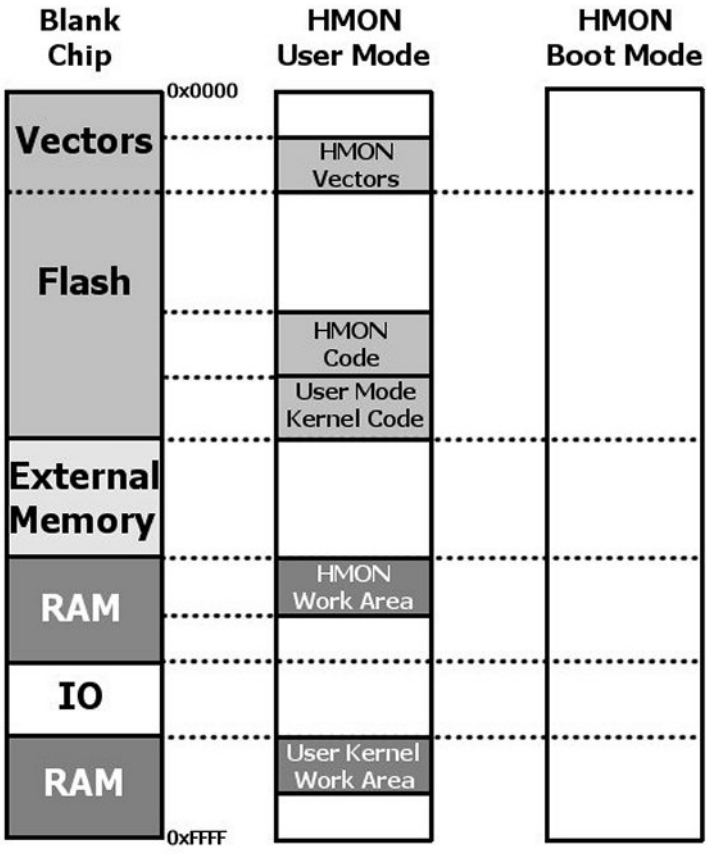
Note: Some interfaces will provide automatic control of the mode pins, for example an RSK with an E8.

3.3.1.Target Hardware Considerations

In order for HMON to be able to program Flash, provision must be made for setting the correct levels on the MCU input pins in order for the target MCU to enter either Boot or User Mode. It is also necessary to make the default Boot Mode port available so that the MCU's Flash may be programmed. On DKs supplied by Renesas there will already be a way to select the mode. The default Boot Mode port is also available on our DKs. Please refer to the DK User Manual for more information.

3.3.2.HMON Memory Map for Boot and User Mode

The diagram below shows both Boot and User Mode memory maps with reference to a blank MCU. This memory map shows the code that needs to be present on the MCU before Boot or User Mode programming is initiated.



3.3.3.HMON in Boot Mode

To program Flash in Boot Mode using HMon, it is necessary to ensure the following: It must be possible to put the target MCU in Boot Mode when prompted, FDT must be configured to use Boot Mode and the port from the host machine must be connected to the MCU Boot Mode port. Once this is achieved you may download code to Flash using the HEW debugger download commands (see Section 5.4). HMon and FDT will automatically program the Flash with the desired code. On completion of programming you will be prompted to put the MCU in user mode and HEW will try and connect up to the HMON monitor code.

3.3.4.HMON in User Mode

Programming Flash in User Mode is a little more complicated. You must have already programmed the Flash with both the working HMon monitor code and the User Mode programming kernels. HMon must be working as it is used to start the User Mode kernel executing. See section 5.4.3 on how to configure HMON so it knows where the kernels are located. The target MCU must be in User Mode and HMon/FDT must be set to use User Mode. The port from the host machine must be connected to the MCU port defined in the User Mode kernels. The default port used by the User Mode kernels is the same as the Boot Mode port. Once this is achieved you may download code to Flash using HEW debugger download commands (see Section 5.4). HMon and FDT will automatically

program the Flash with the desired code. On completion of programming, HMon can be configured to start running code from any location in the memory map. Care should be taken to include both HMon code and the User Mode Kernels in the newly downloaded code, if either are corrupted you must use Boot Mode to program Flash. User Mode programming is more complicated but it does allow you to restrict programming to specific areas or blocks of Flash thus reducing programming time.

Chapter 4.Creating an HMON Project

This section describes how you should create a project that could be debugged using the HMON monitor. The description is based on the Renesas toolchain but similar methods may be used with other toolchains. It is assumed that you have knowledge of HEW projects and the HEW Project Generator Wizard before continuing. For more information on this please refer to the HEW User Manual. If you are using an DK from Renesas, a specific DK Project Generator Wizard will be included with HEW that automates the creation of an HMON project. This section describes how to create an HMON project using the standard Renesas toolchain without the aid of the DK Project Generator Wizard.

4.1.HMON Monitor Components

The HMON monitor comes in the form of a number of library files and some source files. The source files are provided so that you may configure the HMON monitor code (see Section Chapter 6), and are located in your Hew installation directory under System\Pg\Renesas\... The libraries and source files are re-locatable and contain information so that they may be placed at specific addresses in the MCU's memory map. This addressing information is called section information (see Section 4.9).

4.1.1.Library Files

A single re-locatable library file is provided with HMON and is shown in the table below. The code provided in this library cannot be changed.

Name	Description
`Libaries.lib or XXX_hmon.lib`	Monitor base code

The library must be included at the toolchain's link stage (see Section 4.7).

4.1.2.Source Files

The following source files must be included in your project (see Section 4.4).

Name	Description
`Vectors.src`	Assembler file for HMON serial and control interrupts
`HMONConfigUser.c`	C source file for HMON user capability configuration
`HMONConfigUser.h`	C header file for HMON user capability definitions
`HMONConfigUserStruct.h`	C header file for HMON user capability structure
`HMONSerialConfigUser.c`	C source file for HMON serial port configuration
`HMONSerialConfigUser.h`	C header file for HMON serial port definitions
`HMONSerialStruct.h`	C header file for HMON serial port structures

The files are provided so that you may configure HMON for your target hardware (see Section Chapter 6).

4.2.User Mode Kernel Components

The User Mode kernel components of an HMON project are only required if User Mode programming is required. If you only require Boot Mode programming these kernels may be omitted from the project.

These components can be found in your Hew installation directory under System\Pg\Renesas\...

4.2.1.Kernel Binaries

User Mode Kernel code is provided in two binary Op-code files shown below:

Name	Description
`UgenU.cde`	Flash erasing and programming code
`FDTInit.cde`	Communications and initialisation code

These binary files are pre-built and are not re-locatable. This means the kernel's code resides at a fixed address in the MCU memory map. This address cannot be changed unless the binary files are rebuilt. The binaries will also need to be rebuilt if different communication channels are chosen or a different MCU clock is required (see Section 6.7). The binary files must be included at the toolchains link stage if User Mode programming is required (see Section 4.8).

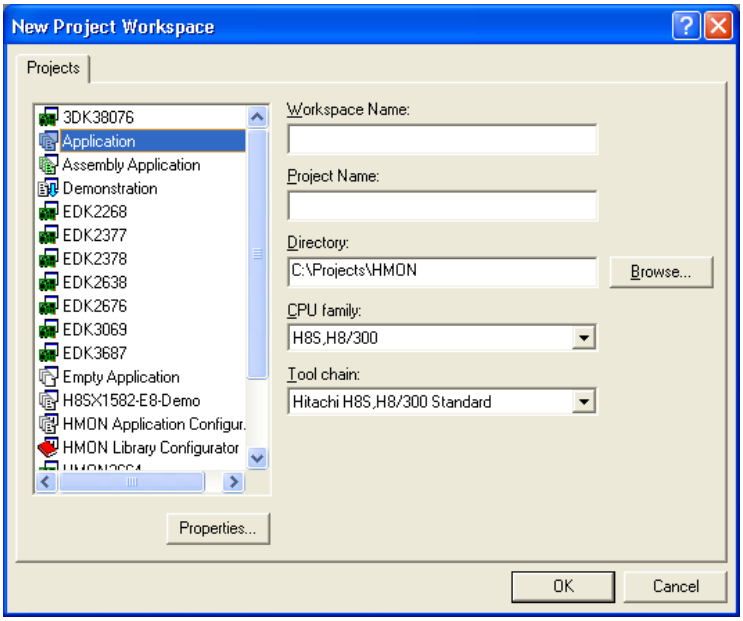
4.3.Using the Renesas MCS Toolchain HEW Project Generator Wizard

HEW groups application code in workspaces. A workspace may contain one or more projects that consist of all the code in your application. HEW provides Project Generator Wizards to assist you in setting up a project for the specific toolchain in use. This section describes how to use the Project Generator Wizard to set up a project to be debugged with HMON.

4.3.1.Starting the Project Generator Wizard

When HEW is started from the Windows Start menu, the 'Welcome!' Screen will be displayed enabling you to create a new workspace.

- *Create a New Workspace from the 'Welcome!' dialog or select the HEW menu item 'File | New Workspace'*



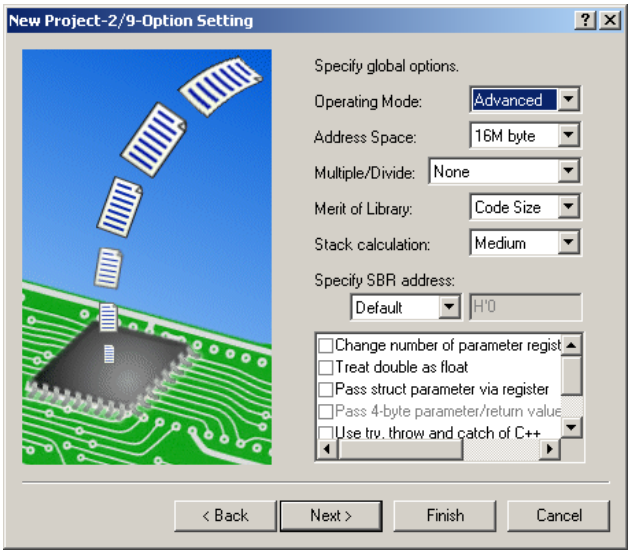
This procedure will display the New Project Workspace dialog as can be seen here:

- *Enter a name for the Workspace and Project*
- *Select a suitable directory for your workspace and project*
- *Select 'Toolchain' for either SH or H8*

Select 'Project Type:' as 'Application'

If HMON software for an DK has been installed, there may be a 'Project Type:' specially for it. This will generate a project that will automatically include the HMON files and can therefore be debugged using HMON. This manual does not describe this type of project but instead outlines how to generate an HMON project from the standard Project Generator Wizard.

- *Click on 'OK'*



This will start the Project Generator Wizard. This wizard takes you through a number of steps to set up a project for use with a specific MCU and target hardware. It enables you to choose the desired MCU and subsequently includes relevant header files and the memory map for the specific MCU chosen.

4.3.2.HMON Specific Selections Within the Project Generator Wizard

When creating a project for use with HMON, some specific selections must be made within the standard Project Generator Wizard. You may choose all the required settings necessary for the needs of your application with the exclusion of the following selection.

- Ensure the Operating Mode is the same as the HMON libraries and User mode kernels

If the target MCU supports the Advanced Operating Mode, then the HMON libraries and User Mode Kernels will be built for this mode. Refer to the MCU Hardware Manual and DK User Manual for more information.

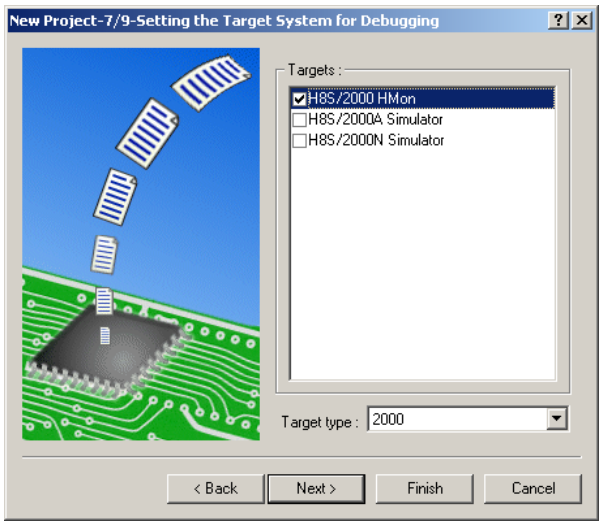
- Select parameter passing with 3 registers

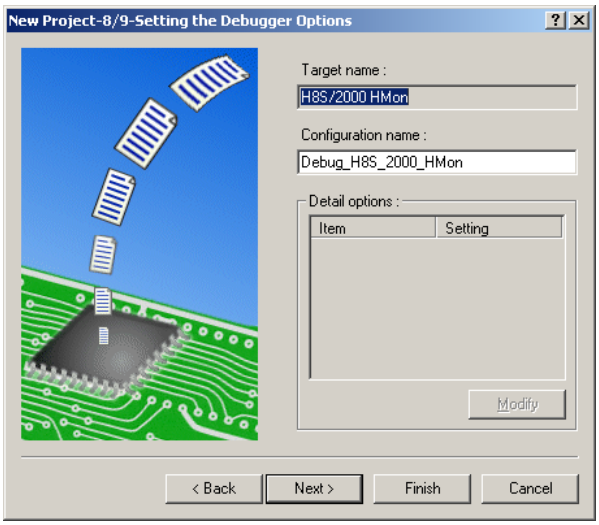
The Renesas Toolchain passes a number of C function parameters using the general registers of the CPU. When using the Renesas toolchain, this may be configured so that either 2 or 3 parameters are passed between functions using the CPU registers. The rest of the parameters are passed on the stack. This option must have project scope so that all functions within the project understand where the parameters will reside. The HMON libraries will normally have been built to pass parameters via 3 CPU registers and so it is therefore necessary for the rest of your project to pass 3 parameters via CPU registers.

4.3.3.Non Essential Selections Within the Project Generator Wizard

Some selections within the Project Generator Wizard are not essential and may be modified at a later stage. The Project Generator Wizard gives you the opportunity to define a target system to debug with. If the HMON software has been installed the following may be achieved

- Select the 'HMON' Target
- Select a suitable name for the HMON Debug Session in the Configuration Name' text box





The selection of a debug configuration is essential for the HEW debugger to be able to communicate with the HMON monitor through the port on the PC. The selection of this debugging target can be defined at a later stage (see Section 5.2.5) but is still included in the Project Generator Wizard.

4.3.4.Stack Pointer Location

You must remember the stack location chosen with the Project Generator Wizard. This will be used by HMON to define where your application code stack is located (see Section 4.4.1).

- Note your application code stack location down
- Click on 'Finish' once all the required selections have been made

4.4.Adding HMON Source Files to a Project

Once you have created a project, the HMON specific files (see Section 4.1.2) may be added to it. Only C and assembler source files should be added to the project. Header files will automatically be recognised as dependent files once the source files have been included in the project. To include the HMON files in the project do the following:

- *Select the HEW menu item 'Project | Add Files'*
- *Select the C and assembler source files described Section 4.1.2*

4.4.1.Setting the Reset Location and Stack

The HEW debugger running with the HMON monitor, provides two commands that will run your code from a pseudo-reset state. HMON will load the stack pointer register and the program counter from predefined values that can be configured in one of the HMON source files. The file `hmonconfiguser.h` contains the code for defining the stack and program counter values. The code extract can be seen below:

```
#define HMON_POWER_ON_RESET_PTR_PTR     0x800     /* Start address */
#define HMON_POWER_ON_RESET_STACK_PTR   0xFFEFC0  /* Stack pointer */
```

The value chosen for the application code start address `HMON_POWER_ON_RESET_PTR_PTR` should be the address of a vector pointer that points to your power-on-reset function. For example in the following, where the function '`PowerON_Reset`' is our power-on-reset function, the address of section (C)User_Vectors must be set as the value of `HMON_POWER_ON_RESET_PTR_PTR.`

```
#pragma section User_Vectors

    void * const UserResetVect = (void *) PowerON_Reset;
```

The value chosen for the stack pointer `HMON_POWER_ON_RESET_STACK_PTR` should be the highest absolute address of your application code stack as this will be used to initialise your Stack Pointer. The stack location is configured in the Project Generator Wizard (see Section 4.3.4). You may need to calculate this address as the stack location + the stack size. Note that it is OK for the Stack Pointer to start just outside RAM as it is pre-decremented when pushing data onto the stack.

4.5.Editing Generated Files

Some of the default files generated by the Project Generator Wizard must be edited in order for HMON to operate correctly.

4.5.1.Editing the Interrupt Vectors File

To enable HMON interrupts, the file `intprg.c` must be edited. A shortened example of this file is shown below for reference purposes:

```
#include    <machine.h>
#pragma section IntPRG
//  vector 1 Reserved
//  vector 2 Reserved
__interrupt(vect=5) void INT_TRAP2(void) {…}      /* ISR for Trap 2 */
__interrupt(vect=27) void INT_PC_Break(void) {…}     /* ISR for PC_Break */
__interrupt(vect=89) void INT_RXI2_SCI2(void) {…}    /* ISR for SCI2 RX*/
```

The code in this file includes all the interrupt service routines (ISR) for the specific MCU chosen in the Project Generator Wizard. Some of these interrupt routines are required by HMON to ensure correct operation. These HMON interrupts must be removed from this generated file in order for HMON to operate. The interrupts used by HMON can be seen in the assembler file `vectors.src`, previously added to the your project (see Section 4.4). A shortened example of `vectors.src` can be seen below for reference purposes:

```
.GLOBAL __HMONCompiledInSWBreakExceptionHandler
.GLOBAL __HMONHWBreakExceptionHandler
.GLOBAL __HMONCommsRxHandler
  .SECTION TRAP_VECTORS,CODE, LOCATE=H'0020
.DATA.L  __HMONCompiledInSWBreakExceptionHandler; TRAPA #2 handler
.SECTION HW_BREAK_VECTORS, CODE, LOCATE=H'6C
.DATA.L __HMONHWBreakExceptionHandler      ; MCU Break Controller
.SECTION SCI_VECTORS, CODE, LOCATE=H'150
.DATA.L __HMONCommsRxHandler            ; SCI RX vector
.END
```

To ensure that HMON has sole access to the interrupts do the following:

- *Remove the ISRs from* `intprg.c` *that are defined in* `vectors.src`

4.5.2.Editing HMON_STARTUP_OPTION

After a power on reset the value of the HMON_STARTUP_OPTION decides whether the HMON monitor will automatically start executing the user code or not.

To get HMON to automatically start running the user code after a power on reset:-

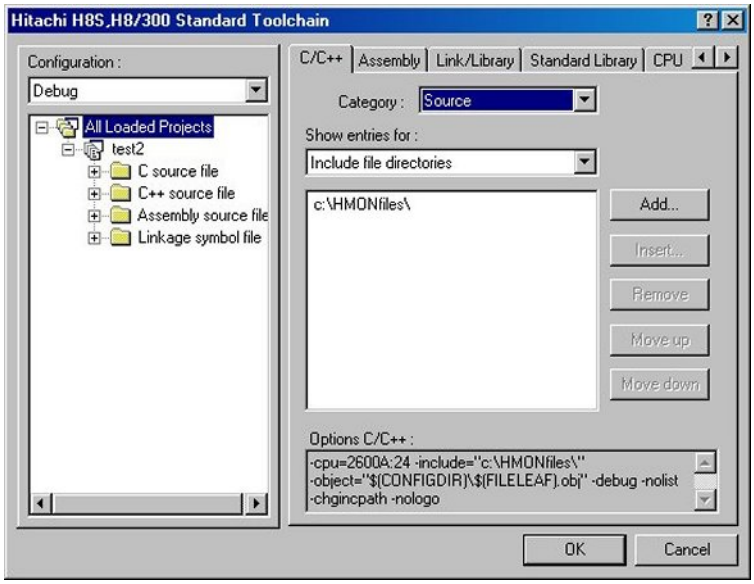
```
#define HMON_STARTUP_OPTION eHMon_StartUser
```

To get HMON to not run the user code automatically after a power on reset:-

```
#define HMON_STARTUP_OPTION eHMon_StartHmon
```

Note, see section 6.1.3 for a method that enables the user code to run directly from the power on reset before initialising HMON.

4.6.Adding Include Paths

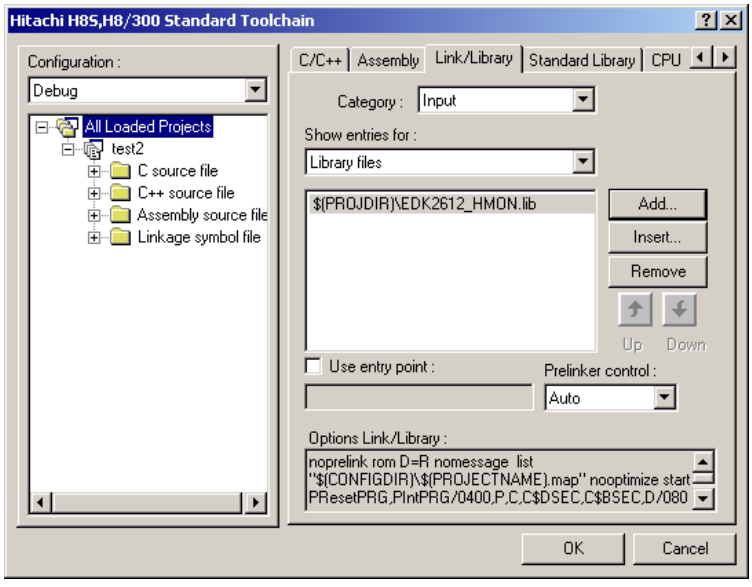


If the added source files are not in the project's directory it may be necessary to tell the toolchain where they are. This will enable the toolchain to find the dependent files, i.e. the header files, associated with the added source files. To add include file directories use the following method:

- *Select the Menu Item 'Build | Toolchain'*
- *Select the 'C/C++' Tab*
- *Select 'Category:' as 'Source'*
- *Select 'Show entries for:' as 'Include Files'*
- *Select 'Add'*
- *Enter the directory where the HMON source files are located*
- *Select 'OK'.*

The compiler will now find the header files.

4.7.Adding the HMON Libraries



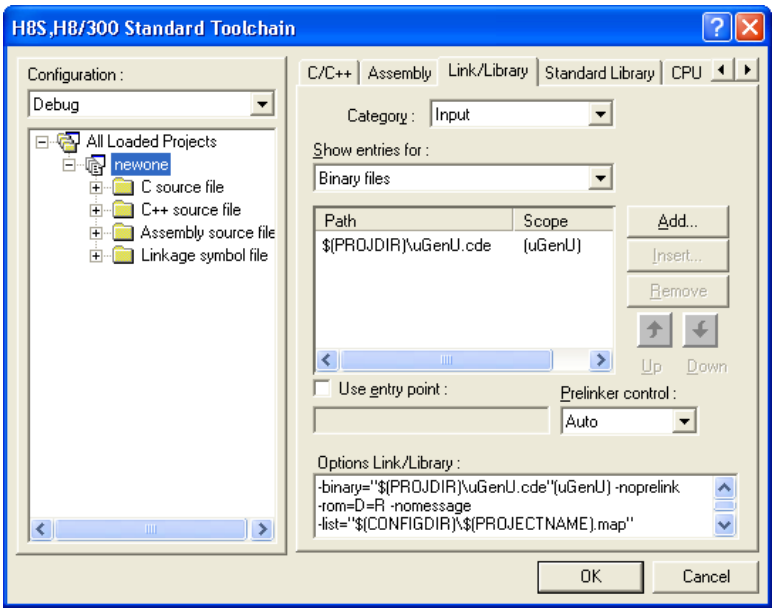
The HMON Libraries described in Section 4.1.1 must be added to the project at the link stage. To add the HMON libraries to the project you should use the following method:

- *Select the Menu Item 'Build | Toolchain'*
- *Select the 'Link/Library' Tab*
- *Select 'Category:' as 'Input'*
- *Select 'Show entries for:' as 'Library Files'*
- *Select 'Add'*
- *Select the HMON library files*
- *Select 'OK'.*

The linker will now include the HMON library files when the project is built.

4.8.Adding User Mode Kernel Binaries

If User Mode Flash programming is required the User Mode kernel binary files described in Section 4.2 must be added to the project at the link stage. To add these binaries to the project you should use the following method:



- *Select the Menu Item 'Build |Toolchain'*
- *Select the 'Link/Library' Tab*
- *Select 'Category:' as 'Input'*
- *Select 'Show entries for:' as 'Binary Files'*
- *Select 'Add'*
- *Select the kernel binary files*
- *Add a 'Section' name for each binary file, FDTInit for FDTInit.cde and FDTUserModeMicroKernel for UgenU.cde. Note :Some targets only have UgenU.cde.*
- *Select 'OK'.*

This procedure ensures that the linker will include your kernel binary files. The inclusion of a section name for each binary will help you when setting up the memory map (see Section 4.9).

4.9.The HMON Memory Map

4.9.1.Default Sections

The Renesas toolchain splits up code into 5 separate entities known as sections. This section information is passed to the linker enabling it to position the different elements at different addresses in the MCU memory map. Code can therefore be assigned to specific addresses or address ranges. The default sections generated by the Project Generator Wizard are shown in the table below.

Section	Attribute	Location	Description
P	Program Code	ROM	The program's Op-code area
C	Constant Data	ROM	Data with constant value
D	Initialised Data	ROM	Data with an initial value, must be copied to RAM at startup
R	Copied Initialised Data	RAM	Initialised data from Section D copied to RAM at startup
B	Un-initialised Data	RAM	Data with no initial value, must be cleared to 0 at startup

On startup the initialised data in section D must be copied to the corresponding variables located in RAM, by default the destination location is a section named R. Un-initialised data (section B) must be initialised to 0 in accordance with the ANSI specification for C programming. This initialisation of the B section and the copying of the D section to the R section is achieved automatically when you use the Project

Generator Wizard. If you wish to add customised section names, these names must be added to the memory map and the toolchain must be made aware of them. Please see the Renesas Compiler Manual for more details on user defined section names.

4.9.2.HMON Library and Source File Section Information

The HMON libraries and source files are built with section names already defined. The sections used are shown below:

Section	Attribute	Location	Description
PHMON	HMON Program Code	ROM	The HMON program Op-code area
CHMON	HMON Constant Data	ROM	The HMON constant values
BHMON	HMON work Area (Un-initialised)	RAM	The HMON data with no initial value

When using HMON, it is not necessary to initialise the un-initialised data section BHMON, the initialisation is performed by the HMON libraries. The HMON source files described in Section 4.1.2 are forced to use these section names by the inclusion of the following code within the files:

```
#pragma section HMON
```

You must include these section names in the memory map of the application. The HMON code is relocatable so you may choose the location of the HMON code and work area. How to include these section names in the memory map is described in Section 4.9.5.

4.9.3.User Mode Kernel Section Information

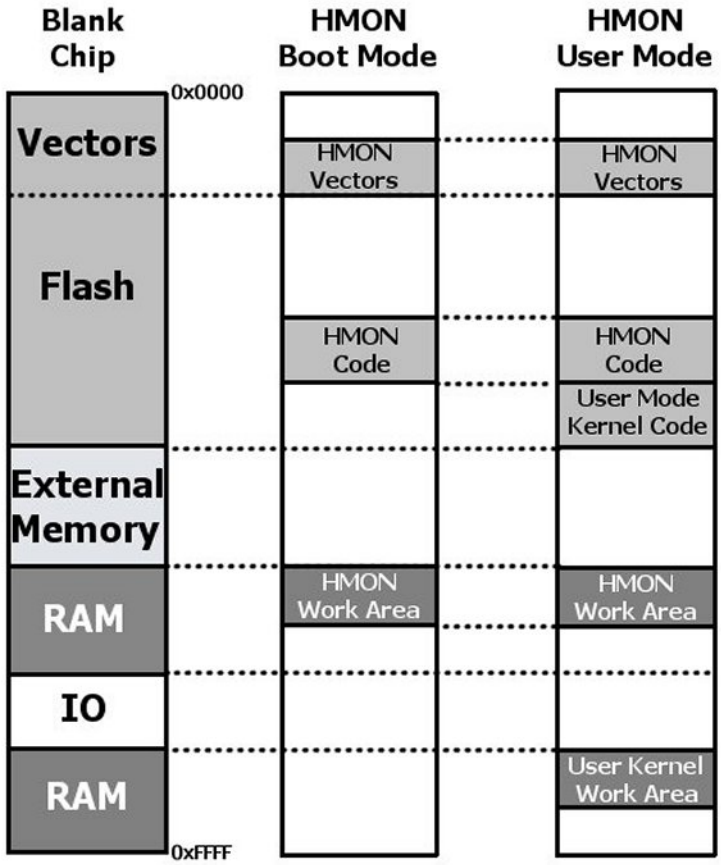
The User Mode kernel binary files have section information defined by yourself when you added the binary files to the build (see Section 4.8). The section information is shown below:

File	Section	Attribute	Location	Description
FDTInit.cde	FDTInit	Kernel Initialisation op-code	ROM	Initialisation functions *Note :Some targets only have UgenU.cde.*
UgenU.cde	FDTUserModeMicroKernel	Kernel program op-code	ROM	Flash programming functions

The section information is used to place the binary file's op-code at the required address in the memory map.. Although you have the facility to place these section names anywhere in the memory map, the kernel code itself is not relocatable because the binaries are pre-built for a specific address. It is therefore necessary to place these sections at the addresses in the memory map to which they were pre-built for. For information on what addresses the kernel binaries were built for and the kernel work area in RAM please refer to the DK User Manual.

4.9.4.Section Positioning

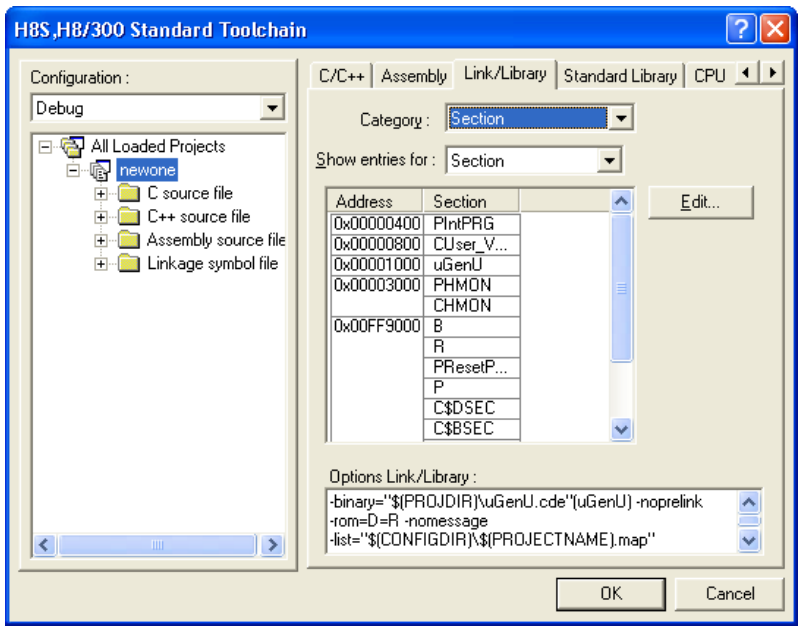
The following diagram shows examples of HMON memory maps for operation in Boot and User Flash programming modes. When you are using Boot Mode it is necessary for the HMON code, interrupt vectors and work area to be positioned in the memory map. When using User Mode it is also necessary for you to include the User Mode kernel code area in the memory map. The rest of the address space is available for your application. For both Flash programming modes the HMON code and work areas may be repositioned anywhere within the address range of the target MCU. This can be achieved because the HMON code is re-locatable. To ensure correct HMON operation, you must ensure that the HMON code, interrupt vectors and data areas are not overwritten or corrupted by your application code. When using User Mode, it is also essential that the User Mode kernel code is not corrupted or overwritten by the application code. If the kernel code is corrupted you will not be able to program the MCU's Flash in User Mode. The User Mode kernel code and work areas are not re-locatable and have a fixed address determined when the kernels binary files were built. The kernel work area is only used during User Mode Flash programming and therefore is only used during the debugger's download operations in User Mode. You do not need to guarantee RAM contents between download operations, this area may also be used by the your application. However, if this area is used by your application or HMON then its contents will be destroyed by a download operation in user mode. So after a download has occurred it is recommended that your application code reinitialises this area of RAM, e.g. your application code is run from reset (see Section 6.1).



4.9.5.Setting Up the Memory Map

The sections created by the inclusion of the HMON libraries, source files and kernel binary files must be included at the toolchains link stage. If the DK Project Generator Wizard is used this is automatic. If the standard Project Generator Wizard is used you must add section and address information to the linker using the following method with reference to the pre-defined section names (See Section 4.9.2 and Section 4.9.3):

- *Select the Menu Item 'Build | Toolchain'*
- *Select the 'Link/Library' Tab*



- *Select 'Category:' as 'Section'*
- *Select 'Show entries for:' as 'Section'*
- *Select 'Add'*
- *Enter the addresses and section names for the HMON libraries and source files*
- *Enter the fixed addresses and section names for the User Mode kernels if used*
- *Select 'OK'.*

The HMON library and source file sections may be place at any address in the memory map because this code is relocatable. You must ensure however, that the User Mode kernels are placed at their pre-built addresses, if this is not achieved Flash programming in User Mode will not be achievable. Refer to the DK User Manual for the kernel's default section location.

Chapter 5.Using The HMON debugger

HMON operations are performed through the HEW debugger. HEW must be informed of how to connect to the HMON code running on the target MCU. This section describes how you should set up HEW so that it can connect and program your target MCU with the HMON code, Flash programming code and your application code

5.1.HMON Monitor Functionality

The HMON monitor code and your application code can be thought of as two separate tasks sharing the MCU resources. Usually the monitor will execute first after reset and set up the MCU system by initialising the HMON communications path and memory work area. It is possible for you to run your application code from reset, but at some stage their code must call the HMON initialisation function (see Section 6.1.3). Once HMON monitor code is running, the HEW debugger may communicate with the monitor code, such communications could include memory dump requests, downloading application code, executing code, setting breakpoints etc. When your application code is executed, the monitor code stops and the application code has full use of the MCU. If a breakpoint occurs or a stop command is sent from the HEW debugger, your application code stops executing and control is handed to the monitor code which records the current state of the application code, including the CPU register values. When the application code is not running, the MCU is still executing HMON monitor code and so the MCU never stops running code of some sort.

5.1.1.Peripheral Operation with HMON

When your application code is stopped, due to a breakpoint or stop command from the HEW debugger, all peripherals continue to operate. For example, if a timer is set up and started by your application code, it will still continue to operate when the application code has stopped running. This is because the monitor code and peripherals are still running on the MCU.

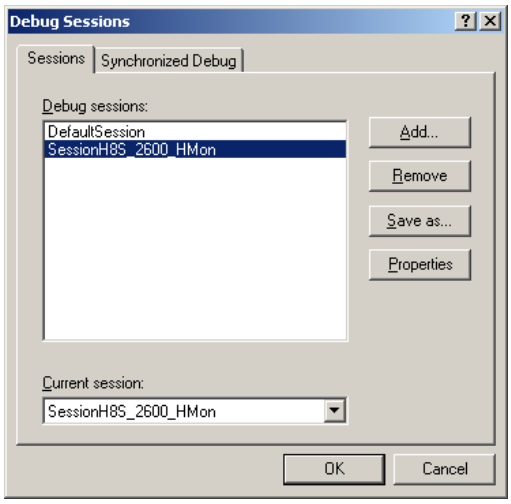
5.1.2.Interrupts and HMON

HMON uses interrupts to interact with the MCU. For example, HMON may use the serial port's receive interrupt to create a stop command that will halt your application code. HMON also uses either the on-chip break controller, or a software trap to trigger an interrupt when a breakpoint is met. Most MCU's have multiple interrupt priority levels, which means that interrupts of higher priority are serviced before those of lower priority. The MCU can mask interrupts by setting the desired level to the interrupt mask register, usually part of the system control registers (EXR or CCR). When two interrupts of different priorities are triggered, the highest priority interrupt is serviced first and the other interrupt remains pending. When an interrupt is serviced the masking level in the system control register is automatically raised to the same level as the triggered interrupt's priority, thus enabling higher priority interrupts to be serviced and lower priority interrupts to be held pending. Any interrupt will remain pending until the interrupt mask level is reduced to the required priority level. For example; If your application code sets up a timer interrupt and then HMON stops the application code running, the timer interrupt will remain pending until HMON re-starts the application code (using menu item 'Debug | Reset Go'), at this time the timer interrupt will be serviced. The interrupt level for all HMON interrupts may be configured to a predefined level. This enables you to create interrupt service routines (ISRs) that will be serviced after the HMON monitor's interrupts. See Section 6.2 for information on how to change the interrupt level for HMON. Please refer to the MCU Hardware Manual for more information on interrupt priority handling.

5.2.Configuring HMON Debugging Session

All debugging information within HEW is stored in session files. A HEW project can have any number of sessions for different target systems such as emulators, monitors and simulators. Each session can contain information on which download modules to use and which target to connect to. Flash interface information is also stored in the session. An empty default debug session, named DefaultSession, is always created when using the standard Project Generator Wizard.

5.2.1.Adding and Removing Sessions



Sessions may be added and removed from projects. This enables you to manage your debugging sessions. If you have not created an HMON debugging session using the Project Generator Wizard (see Section 4.3.3) it must be done using the following method:

- *Select the Menu Item 'Debug | Debug Sessions'*
- *Select 'Add'*
- *Select 'Add new session' and a name*
- *Select 'OK' twice*

This new session may now be selected through the session dropdown dialog on the HEW toolbar shown below:

- *Select the new session from drop down*

If you have already created an HMON debugging session using the Project Generator Wizard (see Section 4.3.3) an attempt will be made to connect to a target MCU (see Section 5.3). Unfortunately at this stage no Flash programming interface has been defined so you must run the Flash Kernel Wizard (see Section 5.2.4) before downloading code to the target MCU.

5.2.2.Saving a Session

Sessions may be saved. If another saved session is selected from the session dropdown dialog, HEW will prompt you to save the current session before loading a new one. Once a debugging session has been configured it must be saved using the following method:

- *Select the Menu Item 'File | Save Session'*

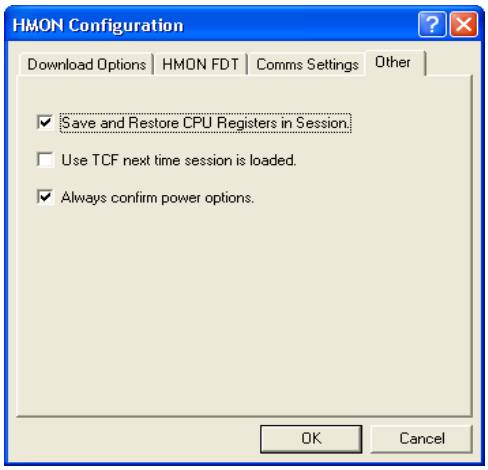
Sessions are never saved automatically therefore the method above must always be used.

5.2.3.The 'Other' Tab of the HMon Configuration

The HMON Configuration consists of 4 Tabs. Three of them are dealt with in the sections that are relevant to them but the last Tab called the 'Other Tab' is dealt with here. It consists of three checkboxes.

- Save and Restore CPU Registers in Session.

If this is checked then the CPU registers will be saved in the session and consequently restored when the session is next opened.



- Use TCF next time session is loaded.

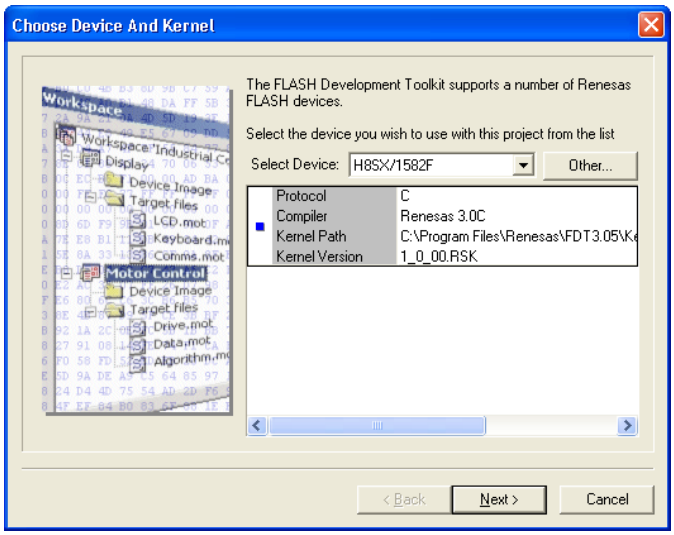
If this is checked and the session saved this will effectively invalidate the HMON part of the session. Hence when the project is next opened HMON will ask the user to select a TCF (Target Configuration File) to get the HMON settings from.

- Always confirm power options.
- If this is checked and an interface type that can supply power to the target is currently in use, then this ensures that no power will be supplied before first checking with the user.

5.2.4.Setting Up the Flash Programming Interface for HMON

The Flash interface must be setup before HMON can program the MCU's Flash. Setting up the Flash interface will inform HMON of device specific information and which programming kernels to use when programming the Flash. To setup the Flash interface you must run the Flash Kernel Wizard using the HEW icon shown here: 

Use the following method to configure the Flash interface:

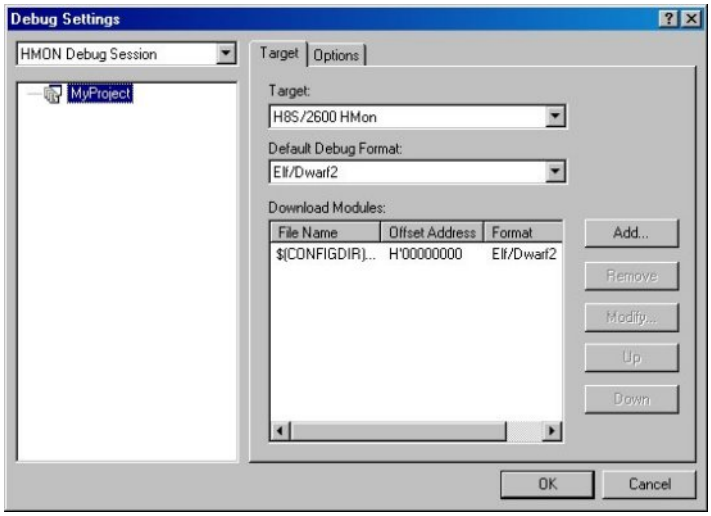


- *Select the desired MCU Device*
- *Select the required Kernel*
- *Select the host machine's download Comms port*
- *Select the clock on the target*
- *Select 'Use default baud rate'*
- *Select 'User Mode' connection method*
- *DO NOT select 'Kernel Already Resident'*
- *Save the workspace on completion*

Once the interface has been defined, HMON may program Flash during the download operation. The host machine's port chosen during this procedure will be the port that is used to program Flash. For more information on this Flash Kernel Wizard Please refer to the FDT User Manual. To change the settings double click on the property you wish to change in the Flash Properties window whilst disconnected.

5.2.5.Configuring an HMON Session

Sessions may be configured for different target platforms. This enables you to specify which target to connect to and what download modules are selectable for downloading to the target. If you have not configured an HMON debugging session using the Project Generator Wizard (see Section 4.3.3) it must be done using the following method:



- *Select the Menu Item 'Debug | Debug Settings'*
- *Select the debugging session to configure*
- *Select the 'Target' as HMON*
- *Select the 'Debug Format' as the toolchain's output format*
- *Select 'Add' to add a download module*
- *Select the project's download module and the toolchain's output format*
- *Select 'OK' twice*

The Renesas toolchain creates Elf/Dwarf2 format files with the extension `*.abs`. Placeholders can be used for the output file name, this will be `$(CONFIGDIR)\$(PROJECTNAME).abs` for the Renesas toolchain. Please refer to the HEW manual for information on build configurations and placeholders. Finally save these settings to the session file (see Section 5.2.2).

- *Save the session on completion*

5.3.Connecting to HMON

The procedure for connecting the HEW debugger to the HMON monitor code running on the target MCU can be different depending on what state the target MCU is in. If the HMON code is already on the target MCU, then a connection can be made. If HMON code is not on the target MCU then a Boot Mode download of HMON must be done. Only when the HMON code is in the MCU's Flash can the HEW debugger communicate with the monitor.

Warning Note. For HEW to be able to stay connected to the target it is important to realise that the HMON monitor code on the target must be able to run. So problems with your user code such as writing over memory that HMON reserves, using an invalid stack pointer etc can all cause the HMON monitor to fail to operate properly and therefore the communication with the target will be lost.

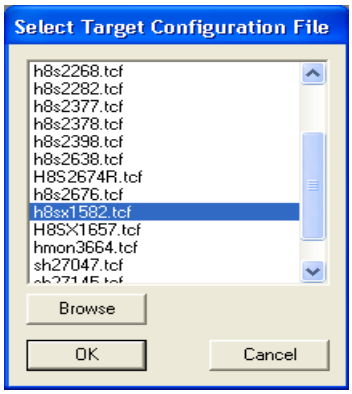
5.3.1.Pre Requisites

Before selecting to connect to the Target the following steps must have been completed.

- On the Menu Item 'Debug | Debug Settings | Target tab':-
- The "Target" Selection must be an HMon Target.
- One of the "Download Modules" must exist and be a module that contains the HMON Monitor.
- FDT must be configured.

5.3.2.First Connection with a newly created project

When using a new session, the first time that a connection is attempted from the HEW debugger to the target MCU, you will be asked to select the 'Target Configuration File' required for the specific MCU as shown in this dialog.



NOTE: If opening an old session file for the first time with HMON 2 that does not contain information relating to RAM emulation support you will be asked if you want to use a Target Configuration File. If you have a Target that supports RAM Emulation and a TCF file with a 'RAMEmulation' entry, then select "Yes" and select the TCF from the Target Configuration Dialog as described below.

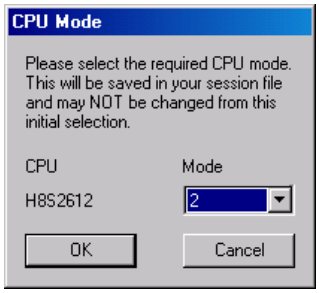
To connect the HEW debugger to the target MCU use the following method:

- *Select the Menu Item 'Debug | Connect'*
- *Select the target configuration file*

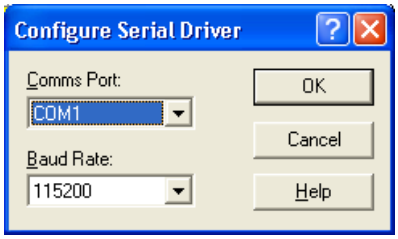
These configurations are stored in `*.tcf` files that can be edited for the specific platform being used (see Section 6.5). If a DK is being used there will be an entry for this in the 'Target Configurations' dialog.

Where the selected target configuration file indicated that there is more than one supported CPU mode:

- *Select the required CPU mode*
- If your session does not have a "CommsPath" entry then if you have more than one communications driver installed then you will be asked to select the driver that you wish to use. For example SerialDrv.dll.



- If your session does not have a “Comms” entry then you will be asked to configure the communications settings relevant to the particular communications driver being used. Here is an example of the serial drivers configuration.
-
- Select ‘Comms Port’ as the host machines debugging serial port
- Select ’Baud Rate‘ as the HMON monitors baud rate.
- Select ‘OK’
- The HMON monitor’s baud rate may be configured by yourself (see Section 6.4.3). If you are using a DK, refer to the particular DK Manual for the default baud rate.



-
- Continue the connection process as explained in the appropriate section below.

5.3.3.Connecting

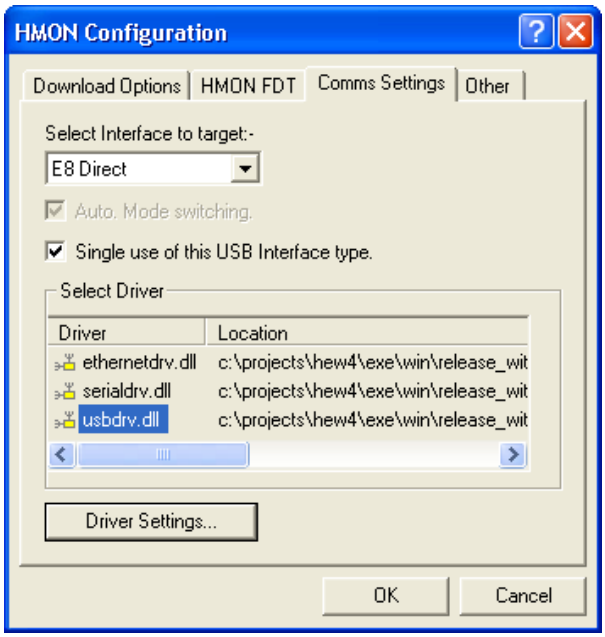
Note: Before trying to connect you can open the HMON Configuration for the current project and check that all the settings are correct by clicking the HMON Configuration button on the HEW toolbar: 

Select the Menu Item ‘Debug | Connect’

Depending upon the type of interface that you are using to connect to the target you may be presented with a dialog particular to that interface. For example if using an E8Direct you will be presented with options to supply power to the target.

If the HEW Debugger fails to connect to the HMON code running on the MCU, the ‘Connection Error’ dialog will be displayed as seen here.

Retry. If the reason for the failure is something obvious like the target not being physically connected then simply make the physical connection, select Retry and then OK. This will retry the process of connecting to the target using the current configuration.



Modify Settings. Selecting this option and OK will bring up the 'HMON Configuration' Dialog with the 'Comms Settings' Tab selected.

From here you can select the Interface type. If you select 'Other' then you can select the communications driver to use. Selecting the 'Settings' button will bring up the configuration dialog for the selected driver. This allows for example for a serial driver the com port and baud rate to be changed. If HMON has control over your targets mode pins then check the 'Auto Mode' checkbox. If you are using driver usbdrv.dll then check the 'Single use of this Interface Type' check box if you will only be plugging in one USB device of this product type at any one time to this PC. This just makes it easier for HMON to know which USB device you are trying to connect to and therefore may save it having to keep asking you to select a particular USB device.

Download HMon using Boot Mode. Select this option if you think that there isn't a valid HMon monitor on the target. This will ensure FDT is in to boot mode and then use it to download the projects download module, or if there is more than one, the download module that you select when asked. Note that for this process to work FDT must have been configured and the Target Type specified correctly. Once FDT has finished HMON will put it back to user mode, if that is what it was before and will then try and connect to the target again.

- *Save the session on completion*

5.4.Downloading Code to the Target MCU

5.4.1.HMON Download Operation

HMON can download code to RAM or Flash. Application code can be downloaded to RAM as long as the MCU has been correctly set up to access the RAM memory area and HMON is running on the target MCU. For a download to external RAM, code may be written in the HMON initialisation function to setup the memory area before a download is attempted (see Section 6.2). Code cannot currently be downloaded to external ROM as HMON has no way of programming this memory. For downloads to on-chip Flash, either Boot or User Mode may be used (see Section 3.3) but you must ensure that a session has been configured with the desired Flash programming kernels (see Section 5.2.4).

5.4.2.Interface Type

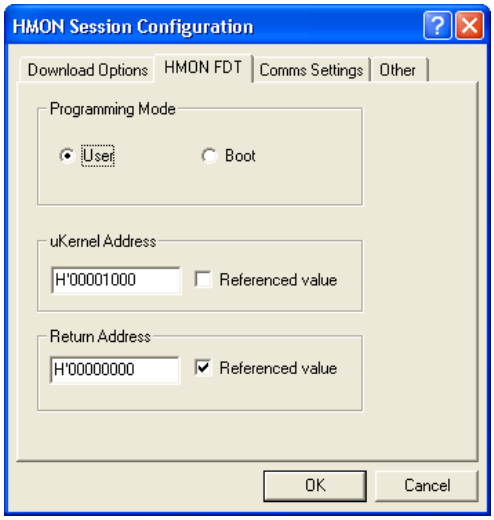
When FDT is performing Boot Mode programming the Target device needs to be put in boot mode and consequently reset after flashing. This is handled differently depending upon the interface being used.

E8Direct	All mode changes are done automatically.
FDM	All mode changes are done automatically.
Serial Cable and 'Auto Mode Switching' check box is selected. (If the target supports this)	RTS line of serial connection used to automatically change mode as required.
Serial Cable and 'Auto Mode Switching' check box is not selected.	User will be prompted to change the mode manually when required

5.4.3.Setting the HMON Flash Download Method

Downloads to Flash may be performed in either Boot or User Mode (see Section 3.3). Once an HMON session and Flash programming interface have been setup (see Section 5.2.4 and Section 5.2.5) you may configure the HMON Flash download method using the following procedure:

- *Select the menu item 'Options | HMON Configuration' then select the HMON FDT Tab.*
- *Select 'Boot' or 'User' Mode download method (Note this is the same as selecting boot or user mode on the FDT configuration)*



If you wish to use User Mode Flash programming, the User Mode kernels must be included in the project and the target MCU must already have this code and the HMON code in Flash (see 3.3.4). If User Mode is required use the following procedure:

- *Enter the pre-built 'uKernel Address'*
- *Uncheck the Referenced value checkbox*
- *Enter the 'Return Address'(Either as a referenced value or an actual address – see below.)*

- *Select 'OK'*

The 'uKernel Address' is the address to which the kernels were pre-built for. Select the address of the FDTInit section as defined in the particular DK manual. The 'Return Address' specifies where to begin running code from once User Mode Flash programming has finished. 'Referenced Value' may be used if a pointer to an address is present at the defined address. For example, selecting a 'Return Address' as 0x0 and for this value to be referenced will force the MCU to run from the reset vector on completion of User Mode Flash programming. To continue debugging with HMON after a User Mode download you must ensure the code at the return address initialises and starts HMON (see Section 6.1.3).

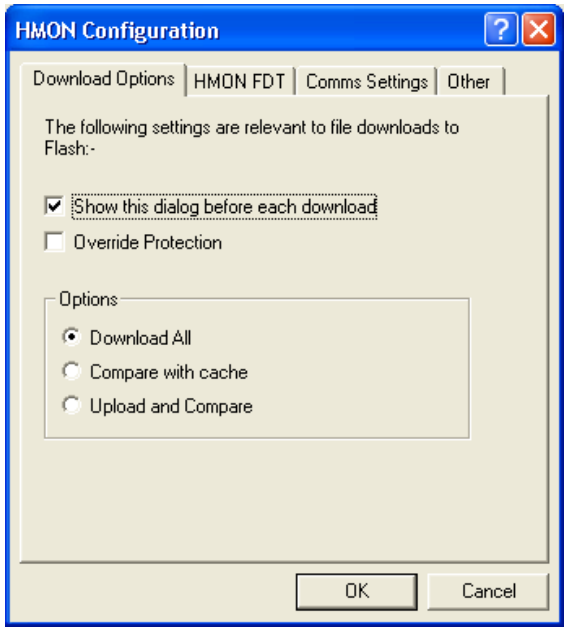
Finally you must save these settings to the session file (see Section 5.2.2).

- *Save the session on completion*

5.4.4.Downloading Code when HMON is Connected

If HMON is running on the MCU and a connection has been made to the MCU and the FDT method has been chosen (see Section 5.4.3), code may be downloaded to the MCU's Flash and RAM. The download modules available are set up using the 'Session Configuration' dialog as described in Section 5.2.5. If User Mode Flash programming is selected (see Section 5.4.3), the User Mode kernel binaries must already be in the MCU's Flash (see Section 4.8).

If FDT is in User mode then the Download Options become relevant. These can be accessed via the menu item 'Options | HMON Configuration' then select the Download Options Tab.



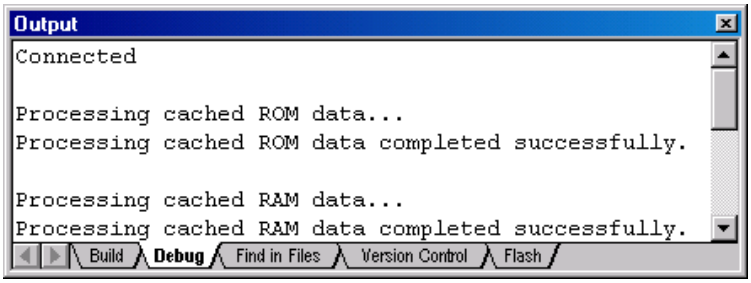
- Show this dialog before each download. If checked this will be shown after selecting to download a module (if FDT is in user mode).
- Override Protection. If checked then all protection setup in the memory map will be overridden.
- Options
    - Download All. If selected then all of the flash blocks that the download module covers will be flashed, unless of course they are protected or overlaid.
    - Compare with cache. If selected then, when a download is being performed, the download modules contents will be compared with the contents of a memory cache built from previous downloads performed during this session. Only flash blocks that require updating will be flashed. Hence this is useful in the situation where you have performed a download of some code which you then slightly edit and therefore need to do a quick flash update of possibly just a single block for example.
    - Upload and Compare. If selected then, when a download is being performed, the contents of all flash blocks covered by the download module will be read and their contents compared with the download modules. Only flash blocks that require updating will be flashed. This is useful to be able to see what blocks have been changed as well as limiting the amount of flashing required.

To download code use the following procedure:

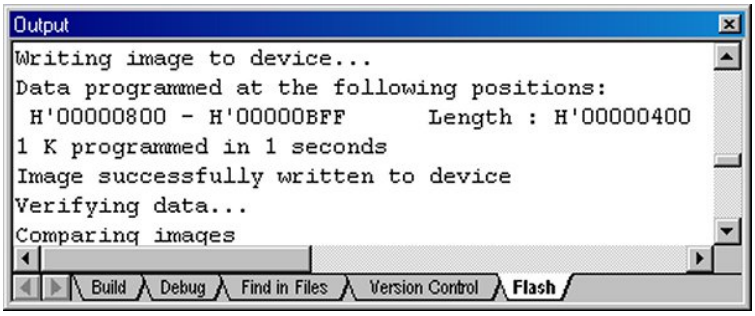
- *Select 'Debug | Download Modules'*
- *Select the desired module to download to the MCU*

If the check box 'Show this dialog before each download' on the Download Options Tab of the HMON Configuration sheet is checked and FDT is in user mode, then the Download Options (See above) will be presented at this point and can be modified.

During downloads to Flash the following a progress bar will be visible in the status bar, and the HEW 'Output dialog's 'Debug' tab will indicate the current stage'.



The HEW 'Output' dialog's 'Flash' tab will also display information about the Flash programming progress.



5.5.Memory Map Configuration

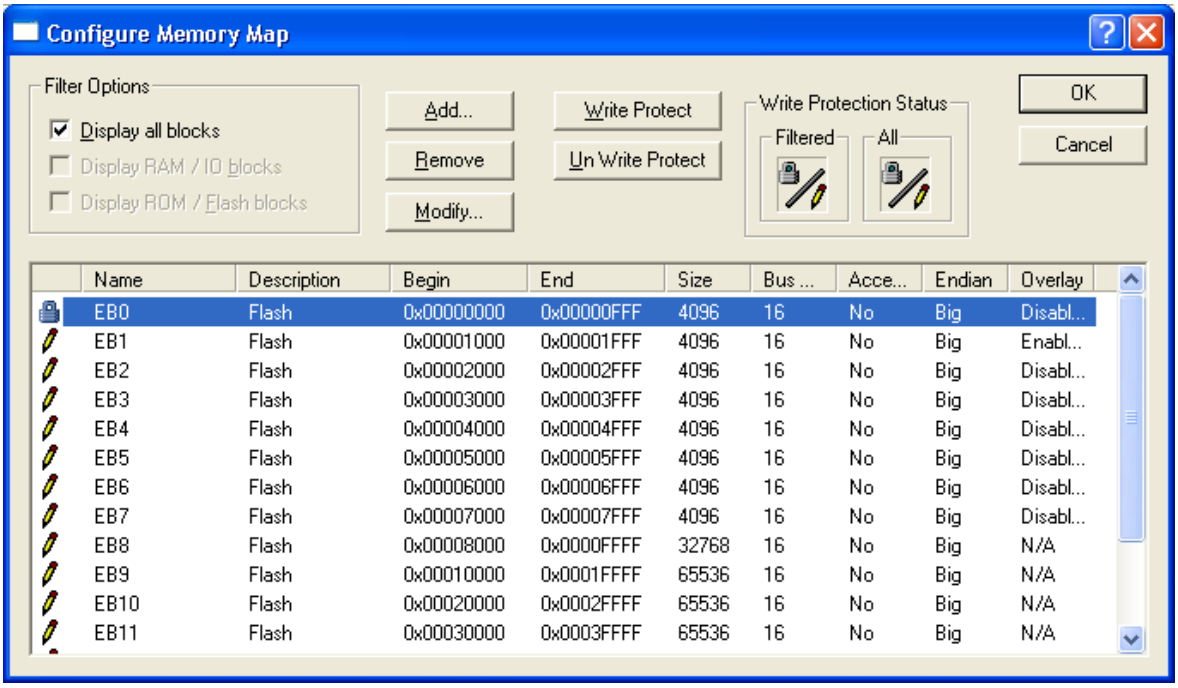
Once HMON has connected to a target MCU running HMON code, the memory map of the target may be edited. This enables you to add new memory resources for external RAM and ROM. It also enables you to protect MCU Flash blocks, please refer to your MCU Hardware Manual for information on MCU Flash Blocks. The default memory resources are obtained from the Flash interface selection in the Flash Kernel Wizard (see Section 5.2.4) and also the configuration file `*.tcf` selected from the 'Target Configuration' dialog (see Section 5.3.2). If the Target supports RAM Emulation you will be able to select a flash block to emulate in RAM (see Section 5.5.5)

5.5.1.Viewing the Memory Map

To access the 'Memory Map' dialog use the following procedure:

- *Select the menu item 'Memory |Configure Map' or the icon shown here:* 

The default memory map will be displayed below. This enables you to display all of the memory attributes for RAM and Flash. The information in this dialog is saved in the current session file (see Section 5.2.2).

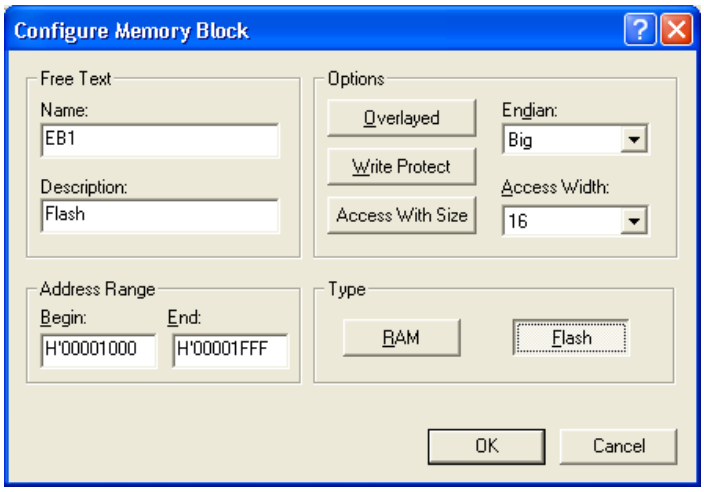


5.5.2.Adding A Memory Block

To add a new memory area, use the following procedure.

- *Select the 'Add…' button*
- *Add the required settings to the 'Configure Memory Block'' dialog shown below*
- The 'Configure Memory Block' dialog enables you to add your target's memory resources so that the HMON can access these areas for downloading and for viewing the contents. Address range and access width may be selected. Care should be taken not to duplicate individual memory areas.

5.5.3.Editing Memory Blocks



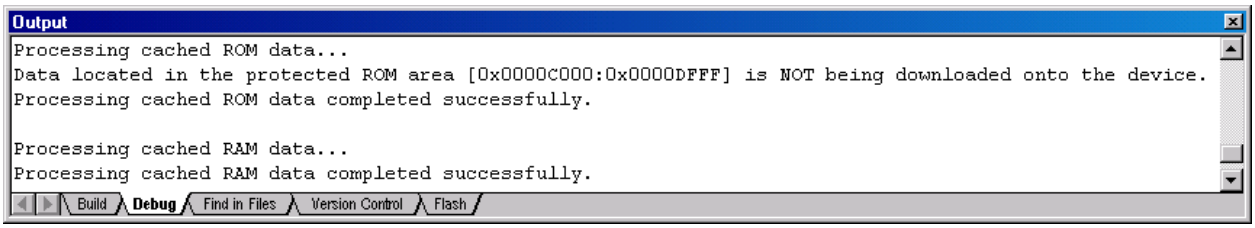
To edit a memory area, use the following procedure.

- *Select the desired memory block*
- *Select the 'Modify…' button*
- *Edit the settings in the 'Configure Memory Block' dialog*

5.5.4.Protecting Memory Blocks

It is also possible for you to protect the memory area selected from the 'Memory Map' dialog by selecting the 'Write Protect' button from the 'Configure Memory Block' dialog. If a block is protected, small padlock will appear beside the block in the 'Memory Map' dialog. Once a Flash block is protected, HMON will not allow this area to be programmed during a User Mode download and a warning will be given during the download, if the module has code in this protected block. This could reduce the time taken to download and program the MCU's memory during development. Protecting a memory block will not

stop it being erased and programmed when downloading in Boot mode. Similarly, if a RAM block is protected any part of the download module that resides in this block will not be sent to the target, and an appropriate warning will be displayed, as shown below:



5.5.5.RAM Emulation

If your Target supports RAM Emulation and HMON has been informed of this from a suitable Target Configuration File then the 'Overlay' column of the memory map will show which flash blocks can be emulated in RAM – i.e. Overlaid. For each memory map entry the overlay column will show the following:-

N/A	Not Available.
Disabled	Target supports overlaying this block but it is currently not overlaid.
Enabled	This block is currently overlaid.

When a block is overlaid you can treat it as if it was RAM. This means you can edit its contents in a memory window and place SW Breakpoints in it. Note however that when the overlay is removed all SW breakpoints in it will be deleted.

Only one block can be overlaid at any one time. The overlapping RAM is part of the normal RAM user space and so when adding an overlay it is vital that that you ensure that the overlapping RAM area is not being used for anything else. When a flash block is overlaid the contents of the flash are copied to the overlapping RAM. When the overlay is removed the contents of the RAM will be compared with the flash and if there is a difference you will be told. If FDT is in user mode then you will also be given the option of updating the Flash with the overlapping RAM contents.

Notes:

1. When disconnecting from a target any overlay will be removed.
2. The existence of an enabled overlay is not stored as part of a session.

5.6.Setting Breakpoints

HMON enables you to place breakpoints on Program Counter values to halt code execution. HMON has a number of different types of breakpoint depending on the peripherals present on the target MCU. Breakpoints come in the form of Hardware and Software breakpoints. Hardware breakpoints use the MCU's address break controller peripheral to stop your application code and return control to HMON code. Software breakpoints come in two forms, a permanent breakpoint compiled with the application, or breakpoint written into memory by HMON. To set a breakpoint on a specific line of code, the cursor should be place on the desired line and the local menu item 'Toggle Breakpoint' selected. If you use the 'run to cursor' command, a temporary breakpoint is placed where the cursor is and code is executed until the breakpoint is met, whereupon the breakpoint is then removed.

5.6.1.Breakpoint Type

As previously stated, breakpoints are implemented by one of two methods: a hardware breakpoint, using the target's own peripheral module, or a software trap. The type of breakpoint used is dependant upon the memory type at the address in question. If the memory type is volatile i.e. RAM a software trap is used. If the memory type is non-volatile i.e. ROM a hardware breakpoint will be used. In the event that the target has no break controller peripheral module, breakpoints or stepping will be not be available in memory areas of type ROM. A warning will be displayed if you try and step in ROM in this situation.

5.6.2.HMON Hardware Breakpoint Operation

Hardware breakpoints use the MCU's address break controller peripheral to stop your application code and return control to HMON. This type of breakpoint is always used when setting breakpoints in Flash or external ROM. These Breakpoint(s) may be disabled during step operations depending upon the target peripheral capabilities (See Section 5.7).

5.6.3.HMON Software Breakpoint Operation

Software breakpoints come in two forms, a permanent breakpoint compiled with the application, or breakpoint written into memory by HMON. To compile a permanent breakpoint into code you must know the breakpoint trap instruction used by HMON. This can be found in the `vectors.src` file as shown below:

```
.SECTION TRAP_VECTORS,CODE, LOCATE=H'0020
.DATA.L   __HMONCompiledInSWBreakExceptionHandler ; TRAPA #0 handler
```

This example file indicates that trap #0 is used for a compiled in software breakpoint. You must call this trap instruction in your application to trigger a compiled in breakpoint. Below is an example of how to write a compiled in breakpoint:

```
#include <machine.h>
void myfunction (void)
{
/* my code */
trapa(0);    /* Permanent compiled in breakpoint */
/* my code */
}
```

This trap instruction is also used when setting a Program Counter breakpoint in RAM. If a breakpoint is set on a line of your code in RAM, this line of code is temporarily replaced by the op-code used to call the trap #2 function. This op-code can be seen in the file `hmonconfiguser.h` as shown below:

```
#define HMON_SOFTWARE_BREAKPOINT_OPCODE  0x5720   /* This is "trapa #2" */
```

If you run code after this type of breakpoint has been met, the trap #2 instruction is temporarily replaced by your original code. This line of code is then executed before the breakpoint op-code is once more placed at this address again.

5.7. HMON Step Operation

The HEW debugger and HMON enable your code to be stepped through, i.e. each line of code may be individually executed. Step operations use the hardware breakpoint resource for stepping code in ROM and a combination of hardware breakpoint and software breakpoint op-code (trap instruction) for stepping code in RAM. Step operations at source level will set a breakpoint on the next appropriate line of code, as a means of halting execution. The behaviour of the step operation in ROM will depend upon the number of break controller channels available, as defined in the Target Configuration File (See Section 6.5.4), and is outlined below in Sections 5.7.1 – 5.7.3.

5.7.1. 1 Break Controller Channel

In this case the single break channel A is shared by a single breakpoint and also the step operation. Attempting to set a second breakpoint will automatically remove the previous one, unless it has been disabled. Where a 'step over' or 'step out' operation encounters a breakpoint in it's execution path, this breakpoint will not trigger, as it will have been temporarily disabled during the step operation.

5.7.2. 2 Break Controller Channels

In this case channel A is given over exclusively to the step operation, leaving Channel B available for the single breakpoint. Attempting to set a second breakpoint will automatically remove the previous one, unless it has been disabled. Where a 'step over' or 'step out' operation encounters a breakpoint in it's execution path, this breakpoint will trigger.

5.7.3. 3+ Break Controller Channels

In this case channel A is given over exclusively to the step operation, leaving Channels B, C etc available for a single breakpoint each. Attempting to set a breakpoint when all Channels B, C etc. are occupied will result in the breakpoint not being created, and a warning displayed in the HEW 'Output' dialog's 'Debug' tab. Under these circumstances it is necessary to remove, or disable one or more breakpoints to free up a currently occupied channel, before a new breakpoint can be set.

5.7.4. Software Traps

When stepping from a breakpoint location, the original opcode is replaced back onto the target, executed and then the software trap opcode restored again. There is no practical limit on the number of breakpoints that can be set in RAM with this method.

5.7.5. Stepping ISRs

Behaviour when stepping interrupts is dependant upon the MCU interrupt controller, and the current MCU interrupt mode (where applicable). For operational details of both, please consult the relevant hardware manual. However the fundamental principle that an ISR can not be debugged unless it's priority is less than that of the HMON interrupts, will always apply.

5.8.HEW Debugger Console

The HEW debugger has a text window that can receive and transmit ASCII text to and from the MCU via HMON. To view this text window use the menu item 'View | Simulated IO'. HMON code contains some functions that you may use to input and output text from the target MCU.

5.8.1.Output

HMON provides a function that can be used in your code to output a text string. The function uses the same port as HMON debugging commands and can be found in file `hmonconfiguser.c`:

```
int _HMONConsoleOutput(UINT8 *pucOutputData, INT16 nSizeofData){ … }
```

This function takes two arguments described below:

Name	Type	Description
`pucOutputData`	`UINT8 *`	Pointer to ASCII text string
`nSizeofData`	`INT16`	Number of characters to output

The following code gives an example of how this function would be used in your application to output text to the 'Simulated IO' dialog.

```
#include <machine.h>
#include "hmonconfig.h"


UINT8 * textout = "Text Output";
INT16 textsize = 0xC;


void myfunction (void)
{
/* my code */
_HMONConsoleOutput(textout, textsize);  /* call to text output function */
/* my code */
}
```

The size of the text string to be output must include the null character at the end of the text string.

5.8.2.Input

HMON also provides a function that can be used in your code to capture text input. The function uses the same port as HMON debugging commands and can be found in file `hmonconfiguser.c`:

```
void _HMONConsoleInputHandler( UINT8 *pucBuffer, INT16 nSizeofInputData,
INT16 *pnSizeofOutputData ) { … )
```

This function is called when HMON receives text characters from the port. To take advantage of this feature you must write code within this function to handle character input. The function takes three arguments described below:

Name	Type	Description
`pucBuffer`	`UINT8 *`	Pointer to received characters
`nSizeofInputData`	`INT16`	Number of received characters
`pnSizeofOutputData`	`INT16 *`	Pointer to number of output characters (echo)

You can assess the input characters in order to control their application. The inclusion of a pointer to the number of output characters `pnSizeofOutputData` enables you to automatically output any received characters. This can be zero if no output is needed.

5.9.Fatal Exception Handling

The supplied user code contains a code stub for a function to be executed when a fatal exception occurs on the target. An example of this is a complete communications failure. This code stub can be found in `hmonconfiguser.c`:

```
void _HMONFatalError(UINT32 userpc)
{
    for(;;)
    {}
}
```

The single parameter passed is the value of the program counter at the point at which the function was called. This may be useful in determining the source of such an error.

5.10.Extending the Command Set

A function stub is included in `hmonconfiguser.c` for the extended command handler. This is called by default, when an unrecognised command is sent to the target. This command should be in the standard GDB packet format to ensure that it is passed down to this function.

```
void _HMONExtendedCommandHandler( UINT8 *pucInputData,
                                  INT16 nSizeofInputData,
                                  UINT8 **ppucReturnData,
                                  UINT16 *punSizeofReturnData )
{
    /* Handle any extended commands here */
```

Chapter 6.Configuring the HMON Monitor

The following section describes the parts of HMON that may be configured for the specific target hardware in use, some parts of this section MUST always be completed for HMON to operate correctly.

6.1.Code Execution from Reset

HMON can be configured to run code from a specific address when the HEW debugger's reset commands are used. HMON can also be configured so that either the monitor or your application code runs from a power on state (see Section 4.5.2).

6.1.1.Setting the Reset Location and Stack

The HEW debugger provides two commands that will run code from a pseudo-reset state. If you use the menu items 'Debug | Reset' or 'Debug | Reset CPU', HMON will load the stack pointer register and the program counter from predefined values that can be configured. The file `hmonconfiguser.h` contains the code for defining the stack and program counter values. The code extract can be seen below

```
#define HMON_POWER_ON_RESET_PTR_PTR      0x800      /* Start address */
#define HMON_POWER_ON_RESET_STACK_PTR    0xFFEFC0   /* Stack pointer */
```

The value chosen for the application code start address should be address of a vector pointer that points to the address of your power-on-reset function, by default the Project Generator Wizard names this function `PowerON_Reset()`defined in the file `Resetprg.c`. You must provide a pointer to this address in their application code. Below is an example of how to do this:

```
#pragma section ResetPointerSection
extern void PowerON_Reset(void);
void * const pointer=((void * const)(&(PowerON_Reset)));
```

This will create a constant section `CResetPointerSection` which you may place at an address using the section control dialog (see Section 4.9.5). The address chosen for the section `CResetPointerSection` must be used for the definition `HMON_POWER_ON_RESET_PTR_PTR`. The stack pointer address should be set to the value chosen in the Project Generator Wizard but may also be calculated by adding the address of the S section in the memory map to the stack size defined in `stacksct.h`. The values chosen for these definitions are also used by HMON to define which code should be executed after HMON initialisation (see Section 6.1.2 and Section 6.1.3)

6.1.2.HMON Code Running First

The easiest way to use HMON is to have it start executing from power-on-reset. To achieve this the power-on-reset vector, located at 0x0 in the MCU memory map must point to HMON initialisation function. The power-on-reset function for HMON is defined in `vectors.src`. The following code must be included in this file:

```
.GLOBAL __HMONCPUPowerOnResetExceptionHandler
.SECTION RESET_VECTOR, CODE, LOCATE=H'0000
.DATA.L __HMONCPUPowerOnResetExceptionHandler  ; Reset handler
```

This code ensures the power-on-reset vector points to the HMON initialisation function. By default the standard Project Generator Wizard will also produce a power-on-reset vector in the file `Resetprg.c` as shown below:

```
__entry(vect=0) void PowerON_Reset(void) { … }  /* Power on reset function */
```

When the code `__entry(vect=0)` is placed in front of a function the toolchain will automatically produce a reset vector at 0x0 that will point to the function. This code will also insert code at the beginning of the function that will initialise the stack pointer. The stack pointer is calculated by adding the address of the S section in the memory map to the stack size defined in file `stacksct.h`. This code `__entry(vect=0)` must therefore be removed from this function definition so that no vector is produced. The lost application reset vector and stack initialisation code is handled by HMON and can be reproduced using the menu items 'Debug | Reset' or 'Debug | Reset CPU' (see Section 6.1.1). Finally HMON must be informed of which code to run after it has been initialised from reset. This is achieved by having the following code present in the file `hmonconfiguser.h`.

```
#define HMON_STARTUP_OPTION          eHMon_StartXXXX
```

If this definition is `eHMon_StartHmon` then HMON will enter an idle state after initialisation and you will have control of the system through the HEW debugger. If the definition is `eHMon_StartUser` then code at the vector address pointed to by `HMON_POWER_ON_RESET_PTR_PTR` will start executing (see Section 6.1.1). The stack pointer will also be set to the value of the definition `HMON_POWER_ON_RESET_STACK_PTR`.

6.1.3.Application Code Running First

If you require the application to run directly from a power on reset this is possible but it is then the applications responsibility to initialise HMON before it can be used for debugging. Firstly the power-on-reset vector, located at 0x0 in the MCU memory map must point to the application code's initialisation function. The application code power-on-reset function is generated automatically by the Project Generator Wizard in the file `Resetprg.c` as shown below:

```
__entry(vect=0) void PowerON_Reset(void) { … }  /* Power on reset function */
```

The code `__entry(vect=0)` ensures the power-on-reset vector points to the `PowerON_Reset()` function. The power-on-reset function for HMON is defined in `vectors.src`. The code shown below must be omitted from this file so that your application code is executed from power-on-reset rather than HMON code:

```
.GLOBAL __HMONCPUPowerOnResetExceptionHandler

.SECTION RESET_VECTOR, CODE, LOCATE=H'0000

.DATA.L __HMONCPUPowerOnResetExceptionHandler  ; Reset handler
```

This code produces a vector that points to the HMON initialisation function and therefore must be removed. If your code has run first, HMON will not been initialised and therefore no communications can take place between HMON and the HEW debugger. To initialise HMON and thus enable communications and HEW's debugging abilities, your application must call the HMON initialisation trap instruction defined in file `vectors.src` and shown below:

```
.DATA.L   __HMONMonitorInitialisationExceptionHandler ; TRAPA #3 handler
```

This code indicates that trap #3 is used to initialise HMON. Below is an example of how your application function can initialise the HMON using this trap instruction:

```
#include <machine.h>

void myfunction (void)

{

/* my code */

trapa(3);    /* Initialise HMON */

/* my code */

}
```

This code must be included in your application in order for HMON debugging to be achievable. Finally HMON must be informed which code to run after it has been initialised from this trap instruction. This is achieved by having the following code present in the file `hmonconfiguser.h`.

```
#define HMON_STARTUP_OPTION          eHMon_StartXXXX
```

If this definition is `eHMon_StartHmon` then HMON will cause the program to break on the next instruction after the HMON initialisation trap instruction. You may then use the HEW debugger to control the target. If this definition is `eHMon_StartUser` then code at the vector address pointed to by `HMON_POWER_ON_RESET_PTR_PTR` will start executing after HMON has initialised (see Section

6.1.1). The stack pointer will also be set to the value of the definition HMON_POWER_ON_RESET_STACK_PTR. The use of the menu items ‘Debug | Reset’ or ‘Debug | Reset CPU’ will also use these defined values.

6.2.Configuring the HMON Interrupt Priority

HMON allows you to configure the interrupt priority levels for the debugging serial port interrupts and the break controller interrupts. Usually an MCU will have 2 or more interrupt priority modes. The simplest mode either masks or unmasks all interrupts, in this mode the HMON interrupt priority cannot be configured. If the interrupt priority mode supports multiple levels then HMON interrupt priority level may be configured. In this mode each peripheral has a register (IPR) dedicated to it's interrupt priority level. Please refer to the MCU Hardware Manual for more information on interrupt priority modes. The system control register address is also defined in the file `hmonconfiguser.h` so that HMON can access the MCU's interrupt priority level. Note: In some cases the INTCR is used to set interrupt priorities rather than the SYSCR. In this case give the address of the INTCR.

```
#define HMON_SYSTEM_CONTROL_REGISTER 0xfffde5  /* Address of SYSCR */
```

Trap interrupt priority cannot be changed. To ensure correct HMON operation the level chosen for HMON interrupts should be higher than the levels chosen for your application interrupts.

6.2.1.Break Controller Interrupt Level

The break controller interrupt priority level is set with the following code in file `hmonconfiguser.h`.

```
#define HMON_DEFAULT_INTERRUPT_LEVEL 3  /* Level set to 3 */
```

6.2.2.HMON Serial Port Interrupt Level

The interrupt priority register for the serial port is defined in `hmonserialconfiguser.h` and shown below:

```
#define P_SCI_IPR (volatile T_SCI_IPR*)0xFFFEEA   /* IPR of serial port */
```

This IPR register is a 16 bit register that contains the interrupt priorities for a number of peripherals. In order for HMON to access the correct bits of this register for the serial port interrupt priority, two operations are included:

```
#define SCI_IPRn_SCI_CLEAR   (T_SCI_IPR)0xBF   /* Clear priority bits */
#define SCI_IPRn_SCI_SHIFT   (T_SCI_IPR)0x06   /* Shift priority bits */
```

The serial port interrupt priority level is set by the following code in file `hmonserialconfiguser.c`.

```
const ST_SCI_CONFIG _HMONgstHmonDebugPortConfig =
{
SCI_CFG_BAUD,       /* Baud rate BRR value */
SCI_CFG_STOP_1,     /* 1 Stop Bit */
SCI_CFG_PARITY_NONE,    /* Parity off */
SCI_CFG_CHAR_8,     /* 8 data bits */
SCI_CFG_BRG_DIV_0,  /* Peripheral clock divider 0 */
SCI_CFG_INT_PRIORITY_5 /* Serial Interrupt priority 5 */
};
```

The HMON serial port interrupt level `SCI_CFG_INT_PRIORITY_x` may be easily configured by using the enumerated type `eIntPriority` in file `hmonserialconfiguser.h`.

6.3.Configuring Software Breakpoints

HMON allows the trap type used for compiled in breakpoints, and the trap type used when setting a PC breakpoint from the debugger to be configured from `hmonconfiguser.h`:

6.3.1.Debugger PC Break Trap

The PC break trap opcode is defined by the following line:

```
#define HMON_SOFTWARE_BREAKPOINT_OPCODE     0x5720
```

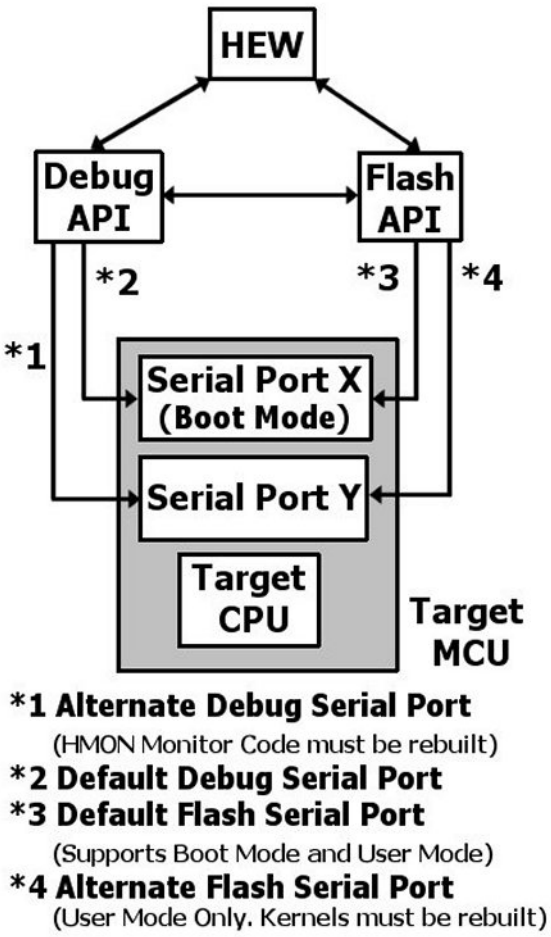
6.3.2.Compiled in Trap

The compiled in trap opcode is defined by the following line:

```
#define HMON_COMPILED_IN_BREAKPOINT_OPCODE  0x5700
```

6.4.Configuring the HMON Serial Port Interface

By default, the HMON communications channel used for debugging commands is the same serial port used for Boot Mode Flash programming. It is recommended that you use the Boot Mode serial port for both programming Flash and debugging your target system. It is possible to re-configure the MCU serial port and the host serial port used for debugging. Changing the host serial port and baud rate is achieved through the 'Comms Settings' Tab of the HMon Configuration Sheet (see Section 5.3.3) or through the Target Configuration file (see Section 6.5.10). To change the MCU serial port used for debugging you must edit a number of files.



6.4.1.Initialising a Serial Port

The serial port must be initialised before communications can take place. The function `_HMONInitHWSerial()` in the file `hmonserialconfiguser.c` contains code that enables the serial port peripheral. Note that this function is also called every time user code is stopped and HMON is started.

```
void _HMONInitHWSerial(void)

{

P_System.MSTPCRB.BIT.SCICKSTP2 = 0;

/* Clear SCI2 Module Stop bits */

}
```

If a different serial port is used you must include code that will enable this specific serial port peripheral. Some MCU's have I/O pins multiplexed with the serial RX and TX pins. This function must also contain the setup for the MCU I/O pins so that they are used exclusively for serial RX and TX. The HMON libraries handle the configuration of serial port registers and enable the serial port's RX and TX functionality. Please refer to the MCU Hardware Manual for details on initialising peripherals and configuring I/O pins.

6.4.2.Selecting a Serial Port

A serial port on the MCU must be used for HMON debugging purposes. The HMON code used to access the desired serial port can be found in the file `hmonserialconfiguser.h`.

```
#define P_SCI_BASE    0xFFFF88  /* SMR register */

#define P_SCI_OFFSET 0x1       /* bytes 1,2,4  */
```

The serial port defined in this file may be changed to another port. This is achieved by changing the address where the serial port registers reside. Each individual serial port register is also configurable in this file.

```
#define P_SCI_SMR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 0))
#define P_SCI_BRR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 1))
#define P_SCI_SCR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 2))
#define P_SCI_TDR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 3))
#define P_SCI_SSR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 4))
#define P_SCI_RDR  (volatile BYTE *)(P_SCI_BASE + (P_SCI_OFFSET * 5))
```

If a different serial port is used you also need to remember to change the interrupt priority register address and bit masking control (see Section 6.2.2).

6.4.3.Configuring the Serial Port Baud Rate

The serial port baud rate is set by the following structure in file `hmonserialconfiguser.c`.

```
const ST_SCI_CONFIG _HMONgstHmonDebugPortConfig =
{
SCI_CFG_BAUD,        /* Baud rate BRR value */
SCI_CFG_STOP_1,      /* 1 Stop Bit */
SCI_CFG_PARITY_NONE,    /* Parity off */
SCI_CFG_CHAR_8,      /* 8 data bits */
SCI_CFG_BRG_DIV_0,   /* Peripheral clock divider 0 */
SCI_CFG_INT_PRIORITY_5 /* Serial Interrupt priority 5 */
};
```

The HMON debugging serial port baud rate must be reconfigured if a different clock is used (see Section 6.6). In this example the values for the definitions `SCI_CFG_BRG_DIV_0` and `SCI_CFG_BAUD` are used to calculate the baud rate. The baud rate pre-scalar value `SCI_CFG_BRG_DIV_0` can be changed to another value described in file `hmonserialconfiguser.h`. The baud rate value `SCI_CFG_BAUD` can be changed in file `hmonconfiguser.h`. In this way you may change the baud rate for the embedded HMON connection. The serial port interrupt priority level is set in this structure but if the serial port has changed, the interrupt priority registers must be reconfigured (see Section 6.2).

6.5.Target Configuration File

When connecting to an MCU for the first time, you may be asked to select a 'Target Configuration' file from a list of the TCF files installed under the HMON installation. A browse button is also available if you wish to use a TCF from a different location, for example to select a TCF that you have generated. This file contains information on how to connect to the HMON monitor. It is only used once when the first connection is made and so should be edited, if required, before a connection is made. The information acquired from this file is subsequently saved in the debug session file when this option is selected from HEW.

6.5.1.Updating a session with a new TCF

You can force HMon to update a session with a new TCF. Connect up with the session you want to update. Check the 'Use TCF next time session is loaded' checkbox on the 'Other' Tab of the HMon Configuration sheet. Save the session and then exit HEW. Next time you connect up with that session you will be given the option of choosing a TCF from a list of all installed TCFs.

6.5.2.Viewing the Target Configuration File

This Target Configuration file is installed with HMON and can be found in the HEW installation directory `…\Tools\Renesas\DebugComp\Platform\HMon\TCFFiles`. Below is an example of this file:

[Version]

HMonVersion=2

[Target]

Target1=[I|VERSION|2] [S|TARGET|RSKH8SX1582] [I|INTERFACE|2] [I|AUTOMODE|1]

Options1=[I|VERSION|2] [S|PRODUCT_NAME|RSKH8SX1582] [S|IO_FILE_NAME|h8sx1582] [XD|BREAKPOINT_SUPPORT|0]

CPUModes1=[I|VERSION|2] [S|CPU_MODES|3] [I|DEFAULT_MODE|3]

FDT1_3=[I|VERSION|2]

Comms1_3=[I|VERSION|1] [S|INTERFACE|USB] [S|PRODUCT|E8DIRECT] [S|DLL|usbdrv.dll]

CopyToSession0=E8DIRECT1_3=[I|VERSION|2] [D|SCI_BAUD|250000] [I|POWER|2] [I|POWER_LEAVE|0]

MemoryAddressable1_3=[I|VERSION|2] [XD|START|00000000] [XD|END|ffffffff] [XD|IO_BASE_ADDRESS|0]

Memory1_3_0=[V|VERSION|2] [S|NAME|On-Chip RAM] [S|TYPE|RAM] [XD|START|00FF9000] [XD|END|00FFBFFF] [S|ENDIAN|BIG] [X|ACCESS_WIDTH|0] [I|ACCESS_WITH_SIZE|0] [XD|ATTRIBUTES|00000007]

Memory1_3_1=[V|VERSION|2] [S|NAME|On-chip Registers] [S|TYPE|IO] [XD|START|00FFEA00] [XD|END|00FFFEFF] [S|ENDIAN|BIG] [X|ACCESS_WIDTH|0] [I|ACCESS_WITH_SIZE|0] [XD|ATTRIBUTES|00000007]

Memory1_3_2=[V|VERSION|2] [S|NAME|On-chip Registers] [S|TYPE|IO] [XD|START|00FFFF20] [XD|END|00FFFFFF] [S|ENDIAN|BIG] [X|ACCESS_WIDTH|0] [I|ACCESS_WITH_SIZE|0] [XD|ATTRIBUTES|00000007]

RAMEmulation1_3=[I|VERSION|2] [I|SUPPORTED|1] [XD|RAMER_ADDR|FFFD9E] [D|NUM_BLOCKS|8] [B|RAMS_BIT_POS|3] [B|RAMn_BIT_POS|0] [XD|OVERLAY_ADDR|FFA000]

It is possible, with care, to edit some of this file, although the basic structure of it must remain the same, or it will be invalid.

6.5.3.Changing the IO File

The HEW debugger can display all the IO register information in an IO dialog launched using the menu item 'View | CPU | IO Area' This provides quick and easy access to the IO registers. To configure HMON to reference a specific IO file `*.io` the following code may be edited in the session file or TCF.

```
Options1= … [S|IO_FILE_NAME|xxxx]
```

The IO file name `xxxx` may be changed to any IO file located in the directory `…\Tools\Renesas\DebugComp\Platform\IOfiles`. Do not include the `*.io` extension. For example, if an IO file `7058.io` resides in this directory then this statement should be:

```
Options1= … [S|IO_FILE_NAME|7058]
```

6.5.4.Breakpoint Support

The types of breakpoint supported by the target MCU are defined by the following code in the session file or TCF:

```
Options1= … [X|BREAKPOINT_SUPPORT|x]
```

The value of `x` defines the number of break controller channels that are available to the debugger for stepping and breakpoints within ROM memory areas. If `x` is `00000000` then no hardware breakpoints are supported, if `x` is `00000001` then 1 channel is available etc. (see Sections 5.6.1 and 5.7). For example if the target is making both Channels A and B available to the debugger then this statement should be:

```
Options1= … [X|BREAKPOINT_SUPPORT|2]
```

6.5.5.Setting the CPU Operating Modes

Most Renesas MCU's support a number of operating modes which support different memory maps and memory resources. You may edit the target configuration file in order to represent their target hardware operating modes.

```
CpuModes1= … [S|CPU_MODES|x,y,z] [I|DEFAULT_MODE|x]
```

All supported MCU operating modes must be entered where `x,y,z` is located. The default mode must also be specified. The CPU modes must be specified as a comma delimited list of numeric values. For example if the MCU supports modes 3 and 4 and mode 3 is then default then this statement will become:

```
CpuModes1= … [S|CPU_MODES|3,4] [I|DEFAULT_MODE|3]
```

6.5.6.Setting the Address Space

The permissible addressable area of memory must be defined in the session or TCF file using the following statement:

```
MemoryAddressable1_x=… [X|START|xxxx][X|END|xxxx][X|IO_BASE_ADDRESS|xxxx]
```

Each MCU operating mode can have a different default memory map. For example, if memory is to be defined in mode 6 then the statement `MemoryAddressable1_x` will be `MemoryAddressable1_6`. The values for `xxxx` must reflect the whole memory map of the device. Below is an example of a memory map for an MCU in mode 3 with a 16 bit address range:

```
MemoryAddressable1_3=… [X|START|00000000][X|END|0000FFFF]
```

Below is an example of a memory map for an MCU in mode 7 with a 25 bit address range:

```
MemoryAddressable1_7=… [X|START|00000000][X|END|00FFFFFF]
```

6.5.7.Setting the IO Base Address

Each MCU operating mode may potentially also have a different IO Base Address. This is the address where the on-chip peripheral registers begin.

```
MemoryAddressable1_x=… [X|IO_BASE_ADDRESS|xxxx]
```

For example, if the address of the first peripheral register listed in the hardware manual is 0xFFF800 the this statement should read:

```
MemoryAddressable1_x=… [X|IO_BASE_ADDRESS|00FFF800]
```

6.5.8.Setting Individual Memory Areas

The Target Configuration file is also used to configure the individual memory areas of the target MCU. The Flash memory area's are obtained from the Flash interface (see Section 5.2.4). This configuration can be edited, after a connection has been made, by using the memory map control dialog (see Section 5.5)

```
Memory1_x_y= … [S|NAME|xxxx] [S|TYPE|xxxx] [X|START|xxxx]
  [X|END|xxxx] [S|ENDIAN|xxxx] [X|ACCESS_WIDTH|x]
  [I|ACCESS_WITH_SIZE|x] [X|ATTRIBUTES|xxxxxx]
```

Each MCU operating mode can have a different default memory map. For example, if memory is to be defined in mode 6 then the statement `Memory1_x_y` will be `Memory1_6_y`. The `y` part of this statement is used to define how many memory areas exist in this mode. If there are two areas to be defined in mode 6 then the first will be `Memory1_6_0=`... and the second `Memory1_6_1=`.... Below is a table defining possible values for this statement.

Field name	Description	Example
`NAME`	Area description	`Boot ROM`
`TYPE`	Generic type of memory	`RAM`, `RAM`, `ROM`, `IO`, `EXT` (external)
`START`	Start of mapped range	`00FFFF00`
`END`	End of mapped range	`00FFFFFF`
`ENDIAN`	Endian of mapped area	`BIG`, `LITTLE`
`ACCESS_WIDTH`	Memory bus access width	`0`,`8`,`10`,`20`,`40` in HEX
`ACCESS_WITH_SIZE`	Flag to denote access specific width memory in those widths	`1` = With size `0` = Without size
`ATTRIBUTES`	Bit field memory attributes	`B0` = Read enabled `B1` = Write enabled `B2` = Volatile (RAM)

6.5.9.Setting the Flash Kernels

In order for you to be able to program Flash using the FDT extension the following statement is necessary in the configuration file:

```
FDT1_x= [V|VERSION|2]
```

6.5.10.Default Baud Rate Configuration

The default baud rate used by the host machine (see Section 5.3.2) can be configured using the following statement:

```
Comms1_x=  … [D|BAUD|xxxx] [D|PORT|x]
```

Below is a table defining possible values for this statement.

Field name	Description	Example
BAUD	Default host baud rate	9600,38400,57600
PORT	Default Comms port	1,2,3,4

6.6.Changing the MCU Clock

The MCU clock may be changed on the target hardware. If the MCU clock is changed a number of files must be edited. If you wish to use User Mode Flash programming, the Flash kernel binary files must be rebuilt to account for a different baud rate for the Flash serial port (see Section 6.7). If only Boot Mode Flash programming is required, these kernels are not included in the project and are therefore not needed. In both cases the new clock value must be included in the Flash Kernel Wizard (see Section 5.2.4). The HMON debugging serial port baud rate must be reconfigured if a different clock is used. For details on changing the baud rate for the HMON debugging serial port see Section 6.4.3.

6.7.User Mode Kernels

User mode kernels are only available with DK's and on the Renesas Website http://www.eu.renesas.com/tools. Please contact your sales office to request target specific User Mode kernels that are not included on the WEB.

Chapter 7.Additional Information

For details on how to use High-performance Embedded Workshop (HEW), refer to the HEW manual available on the CD or from the web site.

For more information on the configuration and use of HMon please refer to the HMon user manual installed from the CD.

Further information available for this product can be found on the Renesas web site at:

http://www.eu.renesas.com/tools

General information on Renesas Microcontrollers can be found at the following URLs.

Global: http://www.renesas.com/

REVISION HISTORY

Rev.	Date		Description
		Page	Summary
3A	14.10.2005		First draft issued
3B	26.10.2005		Update in response to Quality Assurance review.
3C	1.11.2005	28	Added warning note about keeping HMON monitor running.

Renesas Debugger

HMon Manual

Publication Date Rev.3 9/11/05

Published by: Renesas Technology Europe Ltd.

Renesas Debugger
HMon Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REG10J0013-0300