Migration guide

Migrating from Renesas HEW toolchain for SH to IAR Embedded Workbench® for ARM

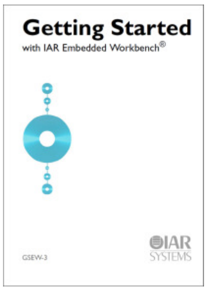
Use this guide as a guideline when converting source code written for the Renesas HEW toolchain for SH to IAR Embedded Workbench® for ARM.

	Product	Version number
Migrating from	Renesas HEW toolchain for SH (HEW)	V.9.x
Migrating to	IAR Embedded Workbench® for ARM (EWARM)	V 6.x

Migration overview

Migrating an existing project from Renesas HEW for SH requires that you collect information about your current Renesas HEW project and then apply this information to the new IAR Embedded Workbench for ARM project. In addition, you need to make some changes in the actual source code. The information in this document is intended to simplify this process. For a complete list of user guides, see IAR Information Center in the IDE.

Note: Basic introduction to IAR Embedded Workbench and how to work in the IDE can be found in the guide *Getting Started with IAR Embedded Workbench* available in the Information Center. A detailed step-by-step introduction is available in the tutorials, also available in the Information Center



Getting Started
with IAR Embedded Workbench®

GSEW-3

IAR SYSTEMS

Converting a Renesas HEW project to an IAR Embedded Workbench project

Both Renesas HEW and IAR Embedded Workbench use *workspaces* for organizing multiple projects. This is useful when you are simultaneously managing several related projects. Workspace files have the filename extension `.hws` and project files `.hwp` in Renesas HEW and for IAR Embedded Workbench, the corresponding file name extensions are `.eww` and `.ewp`.

The filename extensions of C source, header files and assembler files are `.c`, `.h`, and `.s` respectively, in both Renesas HEW and IAR Embedded Workbench. The object files produced by the compiler or assembler have the filename extension `.obj` in Renesas HEW and .o in IAR Embedded Workbench.

Create a new project and workspace

Start the IAR Embedded Workbench IDE and create a new workspace by choosing **File>New>Workspace**. Thereafter, create a new project by choosing **Project>Create New Project...**.

Add source files

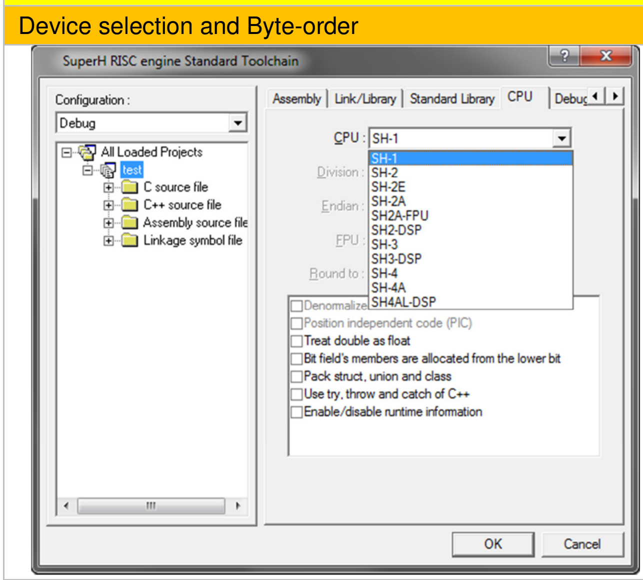
Add the C source and assembler files from the Renesas HEW project into the new IAR Embedded Workbench project. To add project files, choose **Project>Add Files...**.

Tool settings

To change project settings, choose **Project>Options...**. Below is an overview of the most important tool settings where Renesas HEW dialog boxes appear in the left-hand column and the IAR Embedded Workbench counterpart in the right column. Make sure that these settings match.

Renesas HEW	IAR Embedded Workbench

Device selection and Byte-order



SuperH RISC engine Standard Toolchain

Configuration : Debug

All Loaded Projects
- test
  - C source file
  - C++ source file
  - Assembly source file
  - Linkage symbol file

Assembly | Link/Library | Standard Library | CPU | Debug

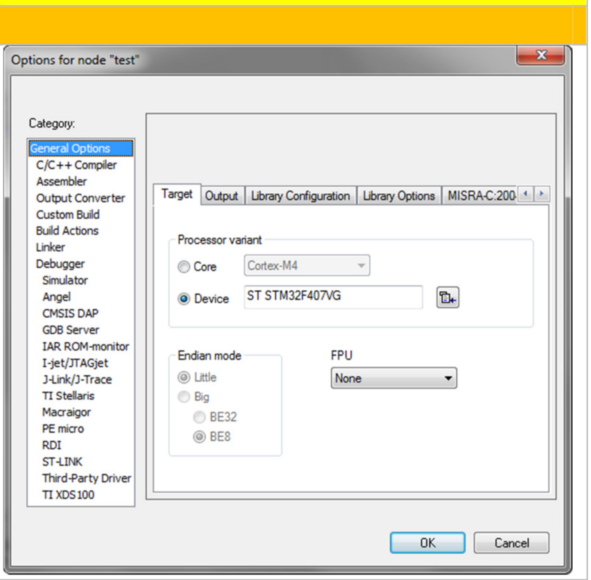
CPU : SH-1

SH-1
SH-2
SH-2E
SH-2A
SH2A-FPU
SH2-DSP
SH-3
SH3-DSP
SH-4
SH-4A
SH4AL-DSP

Division :
Endian :
FPU :
Round to :

□ Denormalize
□ Position independent code (PIC)
□ Treat double as float
□ Bit field's members are allocated from the lower bit
□ Pack struct, union and class
□ Use try, throw and catch of C++
□ Enable/disable runtime information

OK | Cancel



Options for node "test"

Category:
General Options
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
Angel
CMSIS DAP
GDB Server
IAR ROM-monitor
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
RDI
ST-LINK
Third-Party Driver
TI XDS100

Target | Output | Library Configuration | Library Options | MISRA-C:200

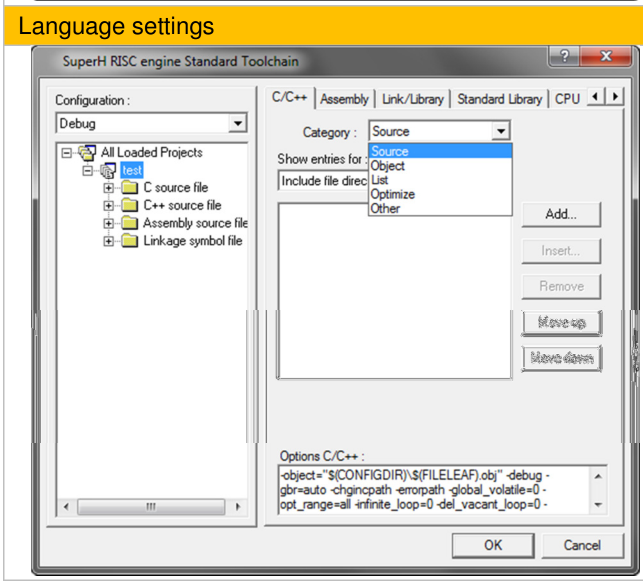
Processor variant
○ Core Cortex-M4
● Device ST STM32F407VG

Endian mode
● Little
○ Big
○ BE32
● BE8

FPU: None

OK | Cancel

Language settings



SuperH RISC engine Standard Toolchain

Configuration : Debug

C/C++ | Assembly | Link/Library | Standard Library | CPU

Category : Source

Source
Object
List
Optimize
Other

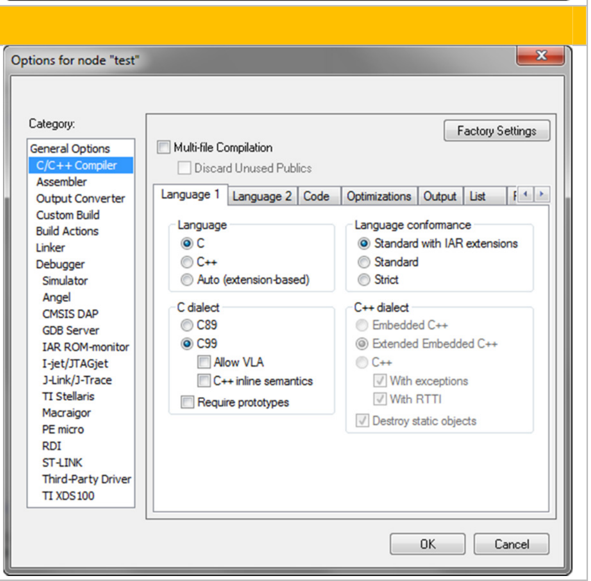
Show entries for : Include file direc

Add... | Insert... | Remove | Move up | Move down

Options C/C++ :

```
-object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -
gbr=auto -chgincpath -errorpath -global_volatile=0 -
opt_range=all -infinite_loop=0 -del_vacant_loop=0 -
```

OK | Cancel



Options for node "test"

Category: C/C++ Compiler

□ Multi-file Compilation
□ Discard Unused Publics

Factory Settings

Language 1 | Language 2 | Code | Optimizations | Output | List

Language
● C
○ C++
○ Auto (extension-based)

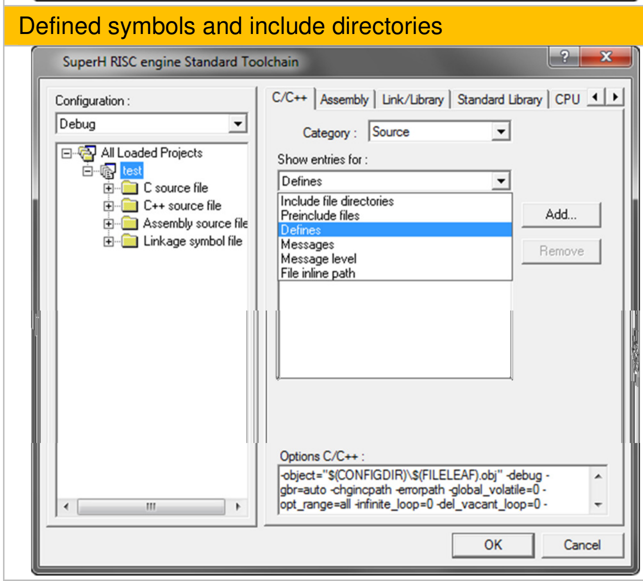
Language conformance
● Standard with IAR extensions
○ Standard
○ Strict

C dialect
○ C89
● C99
□ Allow VLA
□ C++ inline semantics
□ Require prototypes

C++ dialect
○ Embedded C++
● Extended Embedded C++
○ C++
☑ With exceptions
☑ With RTTI
☑ Destroy static objects

OK | Cancel

Defined symbols and include directories



SuperH RISC engine Standard Toolchain

Configuration : Debug

C/C++ | Assembly | Link/Library | Standard Library | CPU

Category : Source

Show entries for : Defines

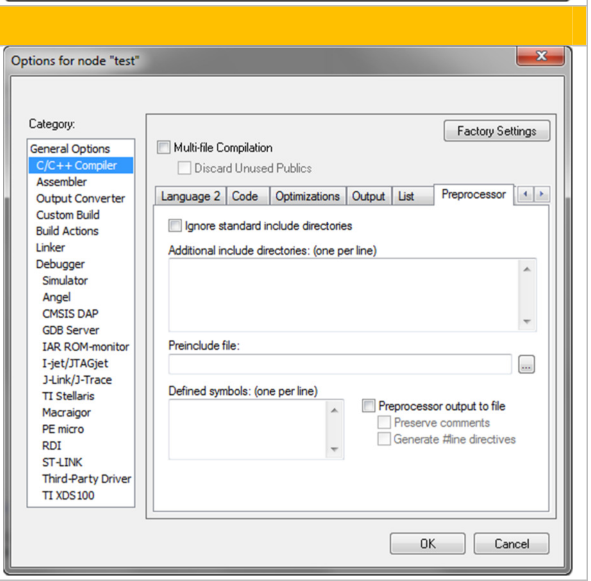
Include file directories
Preinclude files
Defines
Messages
Message level
File inline path

Add... | Remove

Options C/C++ :

```
-object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -
gbr=auto -chgincpath -errorpath -global_volatile=0 -
opt_range=all -infinite_loop=0 -del_vacant_loop=0 -
```

OK | Cancel



Options for node "test"

Category: C/C++ Compiler

□ Multi-file Compilation
□ Discard Unused Publics

Factory Settings

Language 2 | Code | Optimizations | Output | List | Preprocessor

□ Ignore standard include directories

Additional include directories: (one per line)

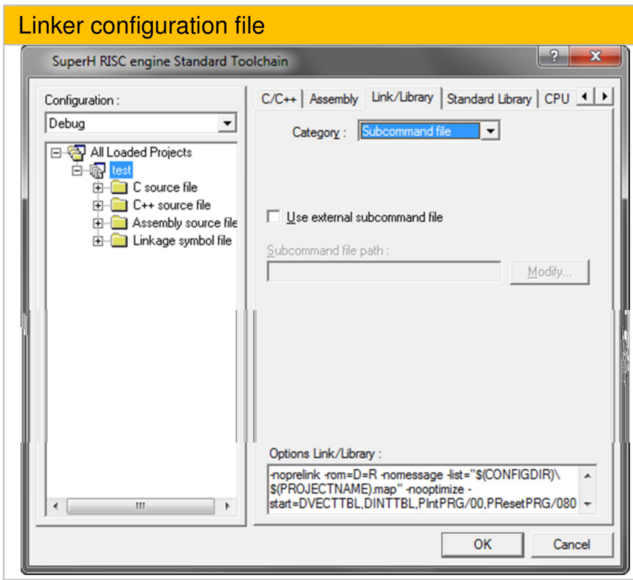
Preinclude file:

Defined symbols: (one per line)

□ Preprocessor output to file
□ Preserve comments
□ Generate #line directives

OK | Cancel

Linker configuration file



SuperH RISC engine Standard Toolchain

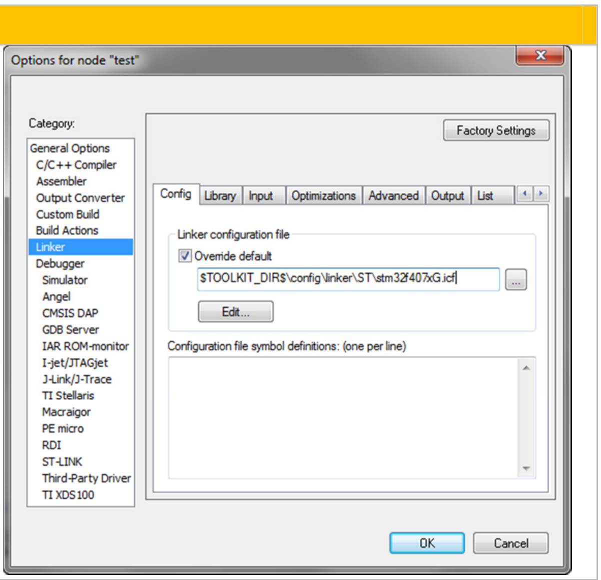
Configuration : Debug

Category : Subcommand file

□ Use external subcommand file

Subcommand file path :

Options Link/Library :
-noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize -start=DVECTTBL,DINTTBL,PIntPRG/00,PResetPRG/080



Options for node "test"

Category: Linker

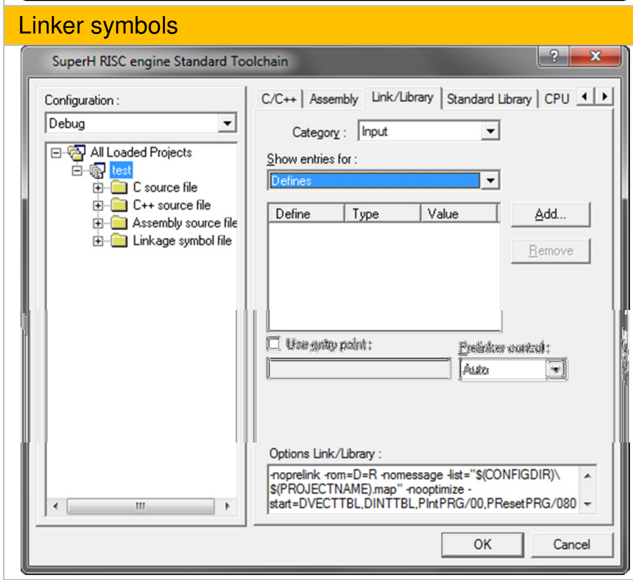
Linker configuration file

☑ Override default

$TOOLKIT_DIR$\config\linker\ST\stm32f407xG.icf

Configuration file symbol definitions: (one per line)

Linker symbols



SuperH RISC engine Standard Toolchain

Configuration : Debug

Category : Input

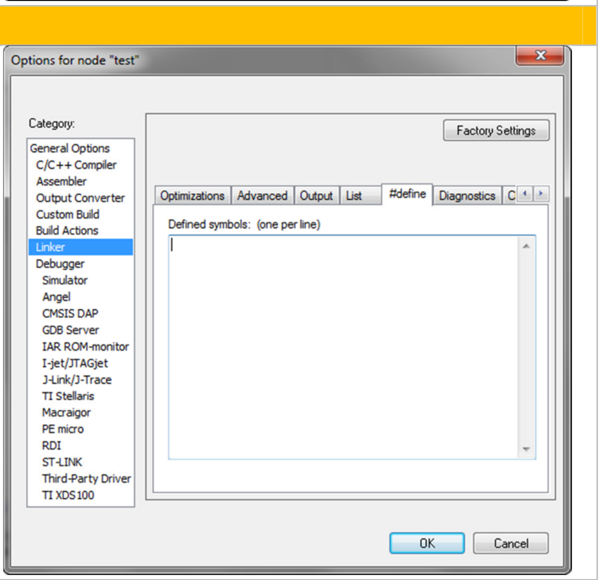
Show entries for : Defines

Define	Type	Value

□ Use entry point :

Prelinker control : Auto

Options Link/Library :
-noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize -start=DVECTTBL,DINTTBL,PIntPRG/00,PResetPRG/080



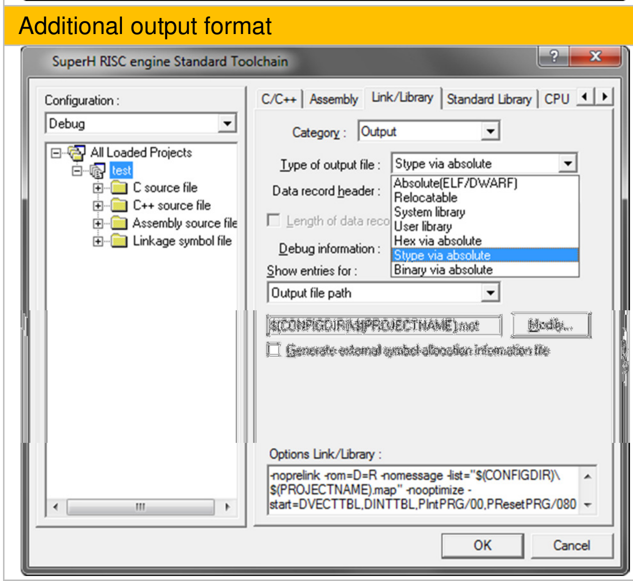
Options for node "test"

Category: Linker

#define

Defined symbols: (one per line)

Additional output format



SuperH RISC engine Standard Toolchain

Configuration : Debug

Category : Output

Type of output file : Stype via absolute

Absolute(ELF/DWARF)
Relocatable
System library
User library
Hex via absolute
Stype via absolute
Binary via absolute

Data record header :

□ Length of data reco

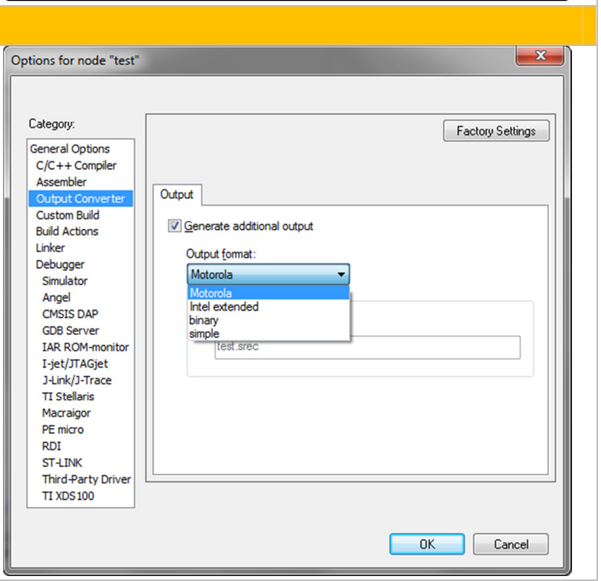
Debug information :

Show entries for : Output file path

$(CONFIGDIR)\$(PROJECTNAME).mot

□ Generate external symbol-allocation information file

Options Link/Library :
-noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize -start=DVECTTBL,DINTTBL,PIntPRG/00,PResetPRG/080



Options for node "test"

Category: Output Converter

Output

☑ Generate additional output

Output format: Motorola

Motorola
Intel extended
binary
simple

test.srec

Note: We recommend that you verify all settings to make sure they match your project needs.

Basic code differences

This table shows some of the basic differences between code written for Renesas HEW and IAR Embedded Workbench® for ARM that you must handle before building your converted project.

Renesas HEW	IAR Embedded Workbench
Initialization code	

The following files contain startup code that you normally <u>do not need to migrate</u> as the functionality is covered by the IAR Embedded Workbench® for ARM `cstartup.s` or in the linker configuration files:
• `resetprg.c`
Startup code
• `dbsct.c`
Data initialization
• `sbrk.c`
Configures the MCU heap memory
• `intprg.c`
Empty interrupt handler functions
• `vecttbl.c`
Vector table initialization
• `vect.h`
Vector function definitions.
• `sbrk.h`
Heapsize
• `stacksct.h`
Stacksize
 	
Startup code that you normally <u>need to migrate</u>:
• `HardwareSetup()`
Customized HW initialization
	`cstartup.s`
	System startup executed after reset performing data and segment initialization. Part of the runtime library but can be overridden by including this assembler file in your project. You find the file in `arm\src\lib\arm`.
	
	`int __low_level_init(void);`
	Called from `cstartup.s` before initializing segments and calling `main()`. You may include your own version of this routine in you project. Suitable for hardware initialization. This function shall return `1` for the data sections to be initialized, otherwise, `0`.

SFR I/O files	
The SFR header file is created by the HEW project generator: Name: `iodefine.h`	One file per device family located in `arm/inc/Renesas` Name: `io<device family>.h` Example: `ior7s721000.h`
Interrupt declarations	

`#pragma interrupt func_name (interrupt specification)`
 	
Example:
`#pragma interrupt _timer_a0(vect=12)`
`void _timer_a0(void)`
`{`
`}`
	For non-Cortex-M:
	
	`{ __irq | __fiq } [__nested] __arm`
	`void IRQ_Handler(void)`
	`{`
	`    /* Do something */`
	`}`
	
	Example:
	
	`__irq __arm void IRQ_Handler(void)`
	`{`
	
	`}`
	
	For Cortex-M it’s just a regular function:
	`void IRQ_Handler(void)`
	`{`
	
	`}`

Building your project

After successfully converting the Renesas HEW project and considered the basic code differences described above, you will still most likely need to fine-tune parts of the source code so that it follows the IAR Embedded Workbench for ARM syntax.

1. Verify that your specific device is selected under **Project>Options>General Options**.
2. Choose **Project>Make**.
3. To find the different errors/warnings, press **F4** (Next Error/Tag).
   This will shift focus to the location in the source code that generated this error/warning.
4. For each error/warning, modify the source code to match the IAR Embedded Workbench for ARM syntax.
   Note: See the **Reference information** section below for this step.
5. After correcting one or more errors/warnings, repeat the procedure.

Note: It is always a good idea to start by correcting the first couple of errors/warnings and then rebuild. This is because errors and warnings later in the source code might just be effects of faulty syntax at the beginning of the source code.

Reference information

Locate a feature in the left-hand column; then you can find the IAR Systems counterpart in the right column. For detailed information about this feature specific to IAR Embedded Workbench, see the relevant documentation. For a complete list of guides, see IAR Information Center in the IDE.

Compiler-specific details

Renesas SH	IAR Systems
Programming languages	
C, C++, EC++	Supported programming languages: assembler, C, Embedded C++, Extended Embedded C++, and C++. For C, the C99 standard is default, but C89 can optionally be used. C99 is supported by the library.
Processor configuration	
- `CPU` type： SH-1, SH-2, SH-2E, SH-2A, SH2A-FPU, SH2-DSP, SH-3, SH3-DSP, SH-4, SH-4A, SH4AL-DSP	Supported cores: ARM7TDMI, ARM10E, Cortex-M0+, ARM7TDMI-S, ARM1020E, Cortex-M1, ARM710T, ARM1022E, Cortex-Ms1, ARM720T, ARM1026EJ-S, Cortex-M3, ARM740T, ARM1136J, Cortex-M4, ARM7EJ-S, ARM1136J-S, Cortex-M4F, ARM9TDMI, ARM1136JF, Cortex-R4, ARM920T, ARM1136JF-S, Cortex-R4F, ARM922T, ARM1176J, Cortex-R5, ARM940T, ARM1176J-S, Cortex-R5F, ARM9E, ARM1176JF, Cortex-R7, ARM9E-S, ARM1176JF-S, Cortex-R7F, ARM926EJ-S, Cortex-A5, XScale, ARM966E-S, Cortex-A7, XScale-IR7, ARM968E-S, Cortex-A15, ARM946E-S, Cortex-M0
- Endianess (big or little endian)	- Big or little endian
- FPU precision	- FPU
- Change bit field order (left or right)	- Bit order (in bit fields) left or right
Memory models/Data models/Code models	
None	None
Overriding default placement of given code/data model	
Segment names for both code and data segments can be modified using the "`#pragma section`" command.	To place a variable or function in a named section, use: `#pragma location="FLASH"`
Absolute placement of variables	

`#pragma ADDRESS variable_name =` *`absolute_address`*	`__no_init char a @0x80;`
	or
	`#pragma location=0x80`
	`__no_init const int a;`

Absolute placement of functions	

#pragma section P MyFunction
void foo(void);
	void foo(void) @ 0x2000;
	or
	void foo(void) @ "MyFunctions"
	or
	#pragma location="MyFunctions"
	void foo(void);

The section MyFunction must be defined using the linker options.	The section MyFunction must be placed by customizing the linker configuration file. See *Customizing the linker configuration file* in the development guide. To place a function at a specific location, the section has to be created first in the linker configuration file (.icf). This can be achieved with: place at address Mem:[0] {readonly section MyFunction}; Where the MyFunction section will be placed at address 0 in Mem.
Constants in ROM	
Const unsigned char c_char[] = [0x1234, 0x5678};	const unsigned short constants[] = {0x1234, 0x5678}
Interrupt functions	

#pragma interrupt function_name (interrupt specification)
Interrupt Specifications
1. Stack switching
sp= variable|constant
Defines the new address for the stack pointer.
2. Trap instruction return
tn=constant
The interrupt exits using a TRAPA instruction.
3. Register bank
resbank
Output of code for saving following registers is suppressed:
R0-R14, GBR, MACH, MACL, PR
4. Register bank switching and RTS instruction return
sr_rts
The interrupt function exits with the RTS instruction. The code for saving only the registers used in the function is output.
5. Interrupt handling function
bank
When a sr_jsr() intrinsic function is used, code for saving SSR and SPC is generated and output of code that saves R0 to R7 is suppressed.
6. RTS instruction return
rts
Interrupt function exits with RTS instruction. Output of code for saving the SSR, SPC, or R0 to R7 is suppressed. Code for saving other registers used in the function is generated.
	When compiling source code for Cortex-M, refer to the files cstartup_M.c for function names. To implement an interrupt function just name the new function the same as in cstartup_M.c.
	For non-Cortex-M devices the interrupt function must be executed in ARM mode. This can be achieved with #pragma type_attribute=__arm or with the __arm extended keyword.
	The __nested keyword modifies the enter and exit code of an interrupt function to allow for nested interrupts.
	{ __irq | __fiq } [__nested] __arm
	void IRQ_Handler(void)
	{
	    /* Do something */
	}

Inline assembler	

#pragma inline_asm
 	
#pragma inline_asm(rot1)
static int rotl (int a)
{
  ROTL R4
  MOV R4,R0
}
	asm [volatile](string [assembler-interface])
	string can contain one or more valid assembler instructions or data definition assembler directives, separated by \n.
	Example:
	asm("movw ax, sp");
	asm("mov a, 0xff");
	Example:
	int Add(int term1, int term2)
	{
	int sum;
	asm("add %2,%1,%0 \n"
	: "=r"(sum)
	: "r"(term1), "r"(term2));
	return sum;

	}

Renesas SH		IAR Systems
Sizes of integers and floating-point		
8 bits	`char`	8 bits
32 bits	`int`	32 bits
16 bits	`short`	16 bits
32 bits	`float`	32 bits
32 bits	`long`	32 bits
64 bits	`long long`	64 bits
32 bits or 64 bits (depends on FPU precision selection)	`double`	64 bits
Pragma directives		
`#pragma section [<section type>] [ <new section name>]`	Switches sections.	`#pragma section = "<section name>"`

`#pragma abs16 <identifier>`
`#pragma abs20 <identifier>`
`#pragma abs28 <identifier>`
`#pragma abs32 <identifier>`
	Specifies address range.	–

`#pragma stacksize <constant>`	Creates a stack section.	Defined in the linker file.

`#pragma interrupt [(]<function name> [(<interrupt specification> [,...])][,...][)]`	Declares an interrupt function.	`#pragma type_attribute={__fiq|__irq |__swi}`
		`void <function> (void)`
		
		Not for cortex-m.

`#pragma inline [(]<function name>[,...][)]`
`#pragma noinline [(]<function name>[,...][)]`
	Performs inline expansion of a function or disables inlining of a function.	`#pragma inline[=forced|=never]`

`#pragma inline_asm [(]<function name> [,...][)]`	Performs inline expansion of an assembly-language function.	–

`#pragma regsave [(]<function name>[,...][)]`
`#pragma noregsave [(]<function name>[,...][)]`
`#pragma noregalloc [(]<function name>[,...][)]`
	Generates or does not generate save and restore code at the start and end of functions.	`#pragma object_attribute=__task`

`#pragma entry[(]<function name>[(sp=<constant>)][)]`	Creates an entry function.	Done in linker: `--entry symbol`
`#pragma ifunc <function name>`	Suppresses saving and restoring of the floating-point registers.	–
`#pragma tbr[()<function name> [({sn=<section name> | ov=<offset>})][,...][)]`	Calls functions by using `TBR` relative addresses.	–
`#pragma align4[(]<function name>=<type>[,...][)]`	Branch destination addresses in the specified function are placed on 4-byte boundaries.	–
`#pragma global_register [(]<variable name>=<register name>[,...][)]`	Allocates global variables to registers.	–

`#pragma gbr_base [(]variable name[,...][)]`
`#pragma gbr_base1 [(]variable name[,...][)]`
	Specifies `GBR` base variables.	–

`#pragma bit_order [{left | right}]`	Switches the order of bit assignment.	`#pragma bitfield={reversed|default}`

`#pragma pack {1|4}`
`#pragma unpack`
	Specifies the boundary alignment value for structure members and class members.	`#pragma pack(n)`
		`#pragma pack()`
		`#pragma pack({push|pop}[,name][,n])`

`#pragma address [(]<variable name>=<absolute address>[,...][)]`	Specifies an absolute address for a variable.	`#pragma location = {address|register|NAME}`
Intrinsic functions		

void set_cr(int cr)	Writes to SR.	void __set_PSP(unsigned long);
int get_cr(void)	Reads SR.	unsigned long __get_PSR(void);
void set_imask(int mask)	Write interrupt mask bits	Cortex-M: __set_PRIMASK
int get_imask (void)	Reads interrupt mask bits	Cortex-M: __get_PRIMASK
void set_vbr(void* base)	Writes to VBR.	–
void* get_vbr(void)	Reads VBR.	–
void set_gbr(void* base)	Writes to GBR.	–
void* get_gbr(void)	Reads GBR.	–
unsigned char gbr_read_byte(int offset)	Reads a GBR-based byte.	–
unsigned short gbr_read_word(int offset)	Reads a GBR-based word.	–
unsigned char gbr_read_long(int offset)	Reads a GBR -based longword.	–
void gbr_write_byte(int offset, unsigned char data)	Writes a GBR -based byte.	–
void gbr_write_word(int offset, unsigned char data)	Writes a GBR -based word.	–
void gbr_write_long(int offset, unsigned char data)	Writes a GBR -based longword.	–
void gbr_and_byte(int offset, unsigned char mask)	ANDs a GBR -based byte.	–
void gbr_or_byte(int offset, unsigned char mask)	ORs a GBR -based byte.	–
void gbr_xor_byte(int offset, unsigned char mask)	XORs a GBR -based byte.	–
int gbr_tst_byte(int offset, unsigned char mask)	Tests a GBR -based byte.	–
Void sleep(void)	Sleep instruction.	void __WFE(void) void __WFI(void)
Int tas(char* addr)	TAS instruction.	–
Int trapa(char* addr)	TRAPA instruction.	–
int trapa_svc (int trap_no, int code, type1 para1, type2 para2, type3 para3, type4 para4)	OS system call.	–
void prefetch (void *p)	PREF instruction.	–
void trace(long v)	TRACE instruction.	–
void ldtlb(void)	LDTLB instruction.	–
void nop(void)	NOP instruction.	void __no_operation(void)
long dmuls_h(long data1, long data2)	Upper 32 bits of the numbers for a signed 64-bit multiplication.	–
unsigned long dmuls_l(long data1, long data2)	Lower 32 bits of the numbers for a signed 64-bit multiplication.	–
unsigned long dmulu_h(unsigned long data1,unsigned long data2)	Upper 32 bits of the numbers for an unsigned 64-bit multiplication.	–
unsigned long dmulu_l(unsigned long data1,unsigned long data2)	Lower 32 bits of the numbers for an unsigned 64-bit multiplication.	–
unsigned short swapb(unsigned short data)	SWAP.B instruction.	–
unsigned long swapw(unsigned long data)	SWAP.W instruction.	–
unsigned long end_cnvl(unsigned long data)	Reverses the byte order inside 4-byte data.	unsigned long __REV(unsigned long);
int macw(short *ptr1, short *ptr2, unsigned int count)	MAC.W instruction.	–
int macwl(short *ptr1, short *ptr2, unsigned int count, unsigned int mask)	MAC.W instruction.	–
int macl(int *ptr1, int *ptr2, unsigned int count)	MAC.L instruction.	–
int macll(int *ptr1, int *ptr2, unsigned int count, unsigned int mask)	MAC.L instruction.	–
void set_fpscr(int cr)	Sets FPSCR.	__set_FPSCR

`int get_fpscr(void)`	Gets `FPSCR`.	`__get_FPSCR`
`float fipr(float vect1[4], float vect2[4])`	`FIPR` instruction.	Can be reproduced with SIMD intrinsics. Example: Load the two vectors with `vld1q_f32`. Multiply with: `vmulq_f32`. Store it back to memory with `vst1q_f32.` Add the floats together.

`void ftrv(float vec1[4],float vec2[4])`	`FTRV` instruction.	Can be reproduced with SIMD intrinsics.
		Load the 4x4 matrix with `vld1` instructions. Load column by column.
		Use `vmla` to multiply.
		Example using floats:
		`vect3 = vmlaq_f32(vect1, mat_col_1)`
		`vect3 = vmlaq_f32(vect1, mat_col_2)`
		`vect3 = vmlaq_f32(vect1, mat_col_3)`
		`vect3 = vmlaq_f32(vect1, mat_col_4)`

`void ftrvadd(float vec1[4],float vec2[4], float vec3[4])`	Transforms a 4-dimensional vector by 4×4 matrix, and adds the result to a 4-dimensional vector.	Can be reproduced with SIMD intrinsics. Same as for FTRV but add: `vect3 = vaddq_f32(vect3, vect2)`
`void ftrvsub(float vec1[4],float vec2[4], float vec3[4])`	Transforms a 4-dimensional vector by 4×4 matrix, and subtracts a 4-dimensional vector from the result.	Can be reproduced with SIMD intrinsics. Same as FTRV but add: `vect3 = vsubq_f32(vect3, vect2)`
`void add4(float vec1[4],float vec2[4], float vec3[4])`	Performs addition of 4-dimension vectors.	Can be reproduced with SIMD intrinsics. `float32x4_t vaddq_f32(float32x4_t, float32x4)`
`void sub4(float vec1[4],float vec2[4], float vec3[4])`	Performs subtraction of 4-dimension vectors.	Can be reproduced with SIMD intrinsics. `float32x4_t vaddq_f32(float32x4_t, float32x4)`

`void mtrx4mul(float mat1[4][4], float mat2[4][4])`	Performs multiplication of 4×4 matrices.	Can be reproduced with SIMD intrinsics.
		Example of one row:
		Use `vld1` for loading into the SIMD registers and `vst1` to save back to RAM.
		`row1 = vmulq_n_f32(mat_2_row1, vgetq_lane_f32(mat_1_row1, 0));`
		`row1 = vmlaq_n_f32(row1, mat_2_row2, vgetq_lane_f32(mat_1_row1, 1));`
		`row1 = vmlaq_n_f32(row1, mat_2_row3, vgetq_lane_f32(mat_1_row1, 2));`
		`row1 = vmlaq_n_f32(row1, mat_2_row4, vgetq_lane_f32(mat_1_row1, 3));`
		For more SIMD intrinsic functions, see `arm_neon.h` found in the `\arm\inc\c` folder.

`void mtrx4muladd(float mat1[4][4], float mat2[4][4],float mat3[4][4])`	Performs multiplication and addition of 4×4 matrices.	Can be reproduced with SIMD intrinsics.
`void mtrx4mulsub(float mat1[4][4], float mat2[4][4],float mat3[4][4])`	Performs multiplication and subtraction of 4×4 matrices.	Can be reproduced with SIMD intrinsics.
`void ld_ext(float mat[4][4])`	Loads mat (4×4 matrix) to the	`float32x4_t vld1q_f32(float32_t`

	extension register.	`*)` load 4 floats into SIMD registers.
`void st_ext(float mat[4][4]`	Stores contents of the extension register to mat (4×4 matrix).	`void vst1q_f32(float32_t *, float32x4)` store content of 4 floats32_t back to memory.
`long __fixed pabs_lf(long __fixed data)`	Computes the absolute value.	`int32x2_t vabs_s32(int32x2_t data)`
`long __accum pabs_la (long __accum data)`	Computes the absolute value.	–
`__fixed pdmsb_lf(long __fixed data)`	Detects the MSB (most significant bit).	_
`__fixed pdmsb_la(long __accum data)`	Detects the MSB.	–
`long __fixed psha_lf(long __fixed data, int count)`	Shifts data arithmetically.	Shift operators << and >> will be translated to appropriate shifting instruction.
`long __accum psha_la (long __accum data,int count)`	Shifts data arithmetically.	Shift operators << and >> will be translated to appropriate shifting instruction.
`__accum rndtoa(long __accum data)`	Rounds data.	–
`__fixed rndtof(long __fixed data)`	Rounds data.	–
`long __fixed long_as_lfixed(long data)`	Copies a bit pattern.	–
`long lfixed_as_long (long __fixed data)`	Copies a bit pattern.	–
`void set_circ_x (__X __circ __fixed array[ ], size_t size)`	Specifies modulo addressing.	–
`void set_circ_y (__Y __circ __fixed array[ ], size_t size)`	Specifies modulo addressing.	–
`void clr_circ(void)`	Cancels modulo addressing.	–
`void set_cs(unsigned int mode)`	Specifies the `CS` bit value (`DSR` register).	–
`void fsca(long angle, float *sinv, float *cosv)`	Computes the sine and cosine values.	–
`float fsrra(float data)`	Computes the inverse of the square root.	–
`void icbi(void *p)`	Invalidates the instruction cache block.	–
`void ocbi(void *p)`	Invalidates the cache block.	–
`void ocbp(void *p)`	Purges the cache block.	–
`void ocbwb(void *p)`	Writes back the cache block.	–
`void prefi(void *p)`	Prefetches instructions into the instruction cache.	–
`void synco(void)`	Synchronize data operation.	–

`int movt(void)`	Refers to the `T` bit.	Depending on cpu mode and core the corresponding bit can be read with:
		`unsigned long __get_FPSCR(void);`
		`unsigned long __get_IPSR(void);`
		`unsigned long __get_CPSR(void);`
		`unsigned long __get_PSR(void);`
		`unsigned long __get_CONTROL(void);`

`void clrt(void)`	Clears the `T` bit.	–

`void sett(void)`	Sets the `T` bit.	Depending on cpu mode and core the corresponding bit can be read with:
		`void __set_FPSCR(unsigned long);`
		`void __set_CPSR(unsigned long);`
		`void __set_CONTROL(unsigned long);`

`unsigned long xtrct(unsigned long data1, unsigned long data2)`	Extracts middle 32 bits from contiguous 64 bits.	–
`long addc(long data1, long data2)`	Adds two values and the `T` bit, and sets the carry to the `T` bit.	–
`int ovf_addc(long data1, long`	Adds two values and the `T` bit, and	–

data2)	refers to the carry.	
long addv(long data1, long data2)	Adds two values, and sets the carry to the T bit.	–
int ovf_addv(long data1, long data2)	Adds two values, and refers to the carry.	–
long subc(long data1, long data2)	Subtracts data2 and the T bit from data1, and sets the borrow to T bit.	–
int unf_subc(long data1, long data2)	Subtracts data2 and the T bit from data1, and refers to the borrow.	–
long subv(long data1, long data2)	Subtracts data2 from data1, and sets the borrow to the T bit.	–
int unf_subv(long data1, long data2)	Subtracts data2 from data1, and refers to the borrow.	–
long negc(long data)	Subtracts data and the T bit from 0, and sets the borrow to the T bit.	–
unsigned long div1(unsigned long data1, unsigned long data2)	Performs division data1/data2 for one step, and sets the result to the T bit.	–
int div0s(long data1, long data2)	Performs initial settings for signed division data1/data2, and refers to the T bit.	–
void div0u(void)	Performs initial settings for unsigned division.	–
unsigned long rotl(unsigned long data)	Rotates data to left by one bit, and sets the bit pushed out of the operand to the T bit.	–
unsigned long rotr(unsigned long data)	Rotates data to right by one bit, and sets the bit pushed out of the operand to the T bit.	–
unsigned long rotcl(unsigned long data)	Rotates data including the T bit to left by one bit, and sets the bit pushed out of the operand to the T bit.	–
unsigned long rotcr(unsigned long data)	Rotates data including the T bit to right by one bit, and sets the bit pushed out of the operand to the T bit.	–
unsigned long shll(unsigned long data)	Shifts data to left by one bit, and sets the bit pushed out of the operand to the T bit.	–
unsigned long shlr(unsigned long data)	Shifts data logically to right by one bit, and sets the bit pushed out of the operand to the T bit.	–
long shar(long data)	Shifts data arithmetically to right by one bit, and sets the bit pushed out of the operand to the T bit.	–
long clipsb(long data)	Performs signed saturation operation for 1-byte data.	
long clipsw(long data)	Performs signed saturation operation for 2-byte data.	–
unsigned long clipub(unsigned long data)	Performs unsigned saturation operation for 1-byte data.	–
unsigned long clipuw(unsigned long data)	Performs unsigned saturation operation for 2-byte data.	unsigned long __USAT16(unsigned long, unsigned long);
void set_tbr(void *data)	Sets data to TBR.	–
void *get_tbr(void)	Refers to TBR value.	–
void sr_jsr(void *func, int imask);	Clears the RB and BL bits of SR to 0, sets the imask value in the I0 to I3	–

	bits of SR, and calls the func function.	
void bset(unsigned char *addr, unsigned char bit_num);	Sets 1 to the specified bit (bit_num) of the specified address (addr).	–
void bclr(unsigned char *addr, unsigned char bit_num);	Sets 0 to the specified bit (bit_num) of the specified address (addr).	–
void bcopy(unsigned char *from_addr, unsigned char from_bit_num, unsigned char *to_addr, unsigned char to_bit_num);	Sets the value of bit [1] (from_bit_num) of address [1] (from_bit_num) to bit T and bit [2] (to_bit_num) of address [2] (to_addr).	–
void bnotcopy(unsigned char *from_addr, unsigned char from_bit_num, unsigned char *to_addr, unsigned char to_bit_num);	Sets the inverted value of bit [1] (from_bit_num) of address [1] (from_bit_num) to bit T and bit [2] (to_bit_num) of address [2] (to_addr).	–
__sectop("<section name>")	Refers to the start address of the specified <section name>.	__section_begin("<section name>")
__secend("<section name>")	Refers to the end address of the specified <section name>.	__section_end("<section name>")
__secsize("<section name>")	Refers to the size of the specified <section name>.	__section_size("<section name>")
Preprocessor symbols		
_SH1 / _SH2 / _SH2E / _SH2A / _SH2AFPU/ _SH2DSP / _SH3 / _SH3DSP / _SH4 / _SH4A / _SH4ALDSP	Processor type.	__ARM4TM__ /__ARM5__ /__ARM5E__ /__ARM6__ / __ARM6M__ __ARM6SM__ /__ARM7M__ /__ARM7EM__ /__ARM7A__ /__ARM7R__
_PIC	Position independent code.	__ROPI__
_BIG / _LIT	Little/big endian.	__BIG_ENDIAN__ / __LITTLE_ENDIAN__
_FLT / __FLT__	double = float.	–
_FPS	FPU = single.	–
_FPD	FPU = double.	–
_DON	Denormalize = on.	–
_RON	Round to nearest.	–
_DPSC	DSPC.	–
_FXD	Fixed const.	–
__HITACHI__	Hitachi compiler.	__IAR_SYSTEMS_ICC__
__HITACHI_VERION__	Compiler version.	__IAR_SYSTEMS_ICC__
__RENESAS__	Renesas compiler.	__IAR_SYSTEMS_ICC__
__RENESAS_VERION__	Compiler version.	__VER__
_SH	SH compiler.	__ICCARM__
Compiler options		
include=<path name>[,...]	Include file directory.	-I <path>
preinclude=<file name>[,...]	Default include file.	--preinclude <file name>
DEFine = <macro name> [=<string literal>] [,...]	Macro name definition.	-D
MEssage NOMEssage [= <error number> [- <error number>][,...]]	Information message.	--remark --diag_suppress=tag[,tag,...]
FILE_INLINE_PATH = <path name>[,...]	Inter-file inline expansion directory specification.	–
CHAnge_message ={Information | Warning | Error } [=<n>[-m],...] [,...]	Message level.	--diag_error=tag[,tag,...] --diag_remark=tag[,tag,...] --diag_suppress=tag[,tag,...] --diag_warning=tag[,tag,...]
PREProcessor [= <file name>]	Pre-processor expansion.	--preprocess[=[c][n][l]] {filename|directory}
Code = {Machinecode | Asmcode }	Object type.	–

<table>
<tr><td>DEBug
NODEBug</td><td>Debugging information.</td><td>--debug</td></tr>
<tr><td>SEction = {
Program= <section name>
| Const=<section name>
| Data=<section name>
| Bss=<section name>
}[,...]</td><td>Section name.
Program section (P).
Const section (C).
Data section (D).
Uninitialized data section (B).</td><td>-</td></tr>
<tr><td>STring = { Const | Data }</td><td>Area of string literal to be output.</td><td>-</td></tr>
<tr><td>OBjectfile = <file name></td><td>Object file name specification.</td><td>--output {filename|directory}
-o {filename|directory}</td></tr>
<tr><td>Template={None|Static|Used|
ALl|AUto}</td><td>Template instance generation.</td><td>-</td></tr>
<tr><td>{ABS16|ABS20|ABS28|ABS32}=
{Program|Const|Data|Bss|Run
|All}[,...]</td><td>ABS16/20/28/32 declaration.</td><td>All in linker file.</td></tr>
<tr><td>DIvision=Cpu={Inline|
Runtime}</td><td>Method of division.</td><td>aapcs=std</td></tr>
<tr><td>IFUnc</td><td>Disables save and restore of floating-point registers.</td><td>-</td></tr>
<tr><td>ALIGN16
ALIGN32
NOALign</td><td>16-byte or 32-byte alignment of labels</td><td>-</td></tr>
<tr><td>TBR [= <section name>]</td><td>Calls functions using TBR relative addresses.</td><td>-</td></tr>
<tr><td>BSs_order={DEClaration
| DEFinition }</td><td>Order of uninitialized variables.</td><td>-</td></tr>
<tr><td>STUff[={Bss|Data|Const}
[,...]]
NOSTuff</td><td>Assigns variables according to the size of variables.</td><td>-</td></tr>
<tr><td>STUFF_GBR</td><td>Assigns variables according to the size of variables in $G0/$G1.</td><td>-</td></tr>
<tr><td>ALIGN4={ALL|LOOP|INMOSTLOOP }</td><td>Alignment of branch destination:
- All branch destination addresses
- Start addresses of all loops
- Start addresses of the innermost loops.</td><td>-</td></tr>
<tr><td>CONST_VOLATILE={DATA|CONST}</td><td>Allocate const volatile variables to the initialized data area or to the constant area.</td><td>-</td></tr>
<tr><td>Listfile [= <file name>]
NOListfile</td><td>Generates a list file.</td><td>-l[a|A|b|B|c|C|D][N][H]
{filename|directory}</td></tr>
<tr><td>SHow={Source | NOSOurce
| Object | NOObject
| STatistics | NOSTatistics
| Include | NOInclude
| Expansion | NOExpansion
| Width = <numeric value>
| Length = <numeric value>
| Tab = {4 | 8} }[,...]</td><td>Listing contents and format.</td><td>-l[a|A|b|B|c|C|D][N][H]
{filename|directory}</td></tr>
<tr><td>OPtimize ={0|1|Debug_only}</td><td>Optimization.</td><td>-O[n|l|m|h|hs|hz]</td></tr>
<tr><td>SPeed
SIze
NOSPeed</td><td>Selects the optimization item.</td><td>-O[n|l|m|h|hs|hz]</td></tr>
<tr><td>Goptimize</td><td>Outputs information for inter-module optimization.</td><td>-</td></tr>
<tr><td>MAP=<file name></td><td>Output information to optimize access to external variables.</td><td>Linker option:
--map {filename|directory}</td></tr>
<tr><td>SMap</td><td>Optimizes access to external variables defined in the file to be compiled.</td><td>-</td></tr>
<tr><td>GBr={Auto|User}</td><td>Automatic or manual creation of GBR relative access code.</td><td>-</td></tr>
<tr><td>CAse={Ifthen|Table}</td><td>switch statement expansion method.</td><td>-</td></tr>
<tr><td>SHIft={Inline|Runtime}</td><td>Shift-operation expansion.</td><td>-</td></tr>
<tr><td>BLOckcopy={Inline|Runtime}</td><td>Transfer-code expansion.</td><td>-</td></tr>
</table>

Unaligned={Inline|Runtime}	Unaligned data transfer.	-
INLine[=<numeric value>] NOINLine	Automatic inline expansion.	Only happens when optimization is set to high. To turn of inlining when optimization is set to high, use --no_inline
FILe_inline=<file name> [,...]	Inter-file inline expansion.	-
GLOBAL_Volatile={0|1}	External variables handled as volatile.	-
OPT_Range={All|NOLoop |NOBlock}	External variable optimizing range	-
DEL_vacant_loop={0|1}	Vacant loop elimination.	-
LOop NOLOop	Loop unroll.	-O[h|hs|hz] --no_unroll
MAX_unroll = <numeric value> <numeric value>: 1 to 32	Maximum number of loop expansions.	-
INFinite_loop={0|1}	Elimination of expression preceding infinite loop.	-
GLOBAL_Alloc={0|1}	External variable register allocation	-
STRUCT_Alloc={0|1}	Structure/ union member register allocation.	-
CONST_Var_propagate={0|1}	const constant propagation	-
CONST_Load={Inline|Literal}	Expansion of constant loading instructions.	-
SChedule={0|1}	Instruction scheduling.	-
SOftpipe	Software pipelining.	-
SCOpe NOSCope	Division of optimizing ranges.	
LOGIc_gbr	GBR relative logic operation generation.	-
CPP_NOINLINE	Preventing expansion of C++ Inline functions.	-
ALIAS={ANSI|NOANSI}	Optimization considering type of object indicated by pointer.	-
ECpp	Embedded C++ language.	--ec++
DSpc	DSP-C language [SH2-DSP, SH3-DSP and SH4AL-DSP].	-
COMment=Nest|NONest}	Comment nesting.	-
Macsave={0|1}	Keep MAC register contents before and after a function is called.	-
RTnext NORTnext	Extension of return value.	-
APproxdiv	Converting the floating-point constant divisions to multiplications.	-
PAtch=7055	Avoiding SH7055 illegal operation [SH-2E].	-
FPScr={Safe|Aggressive}	FPSCR register switching.	-
Volatile_loop	Suppresses optimization of loop iteration condition.	-
AUto_enum	Automatically selects the enumeration data size.	-
ENAble_register	Allocates preferentially the variables with register storage class specification to registers.	-
STRIct_ansi	ANSI conformance.	For usage of extended language use -e. Otherwise, the compiler uses ANSI conformance
FDIv	Converts integer division to floating-point division.	-
FIXED_Const	Handles floating-point values as fixed-point values.	-

FIXED_Max	Handles 1.0r (1.0R) as the maximum value of __fixed (long __fixed) type.	–
FIXED_Noround	Omits type conversion for the operation result of __fixed type multiplication.	–
SIMple_float_conv	Omitting range check for conversion between floating-point nmber and integer.	–
NOUSE_DIV_INST	Suppress DIVS and DIVU instruction generation.	–
FLOAT_ORDER	Change operation order for floating-point expression.	–
CPu = { SH1 | SH2 | SH2E | SH2A | SH2AFPU | SH2DSP | SH3 | SH3DSP | SH4 | SH4A |SH4ALDSP }	CPU/operating mode.	--cpu=core
FPu = { Single | Double }	Floating-point operating mode.	--fpu={VFPv2 | VFPv3 | VFPv3_d16 | VFPv4 | VFPv4_sp | VFP9-S | none}
Round = { Zero | Nearest }	Rounding mode.	–
Pic= { 0 | 1 }	Program section position independent .	--ropi
DOuble=Float	double to float conversion.	–
BIt_order={Left | Right }	Bit field order specification.	–
PACK = { 1 | 4 }	Boundary alignment of structure, union, and class members.	–
EXception NOEXception	Exception handling.	–
RTTI= {ON | OFF }	Runtime type information.	–
DIvision = { Cpu | Peripheral | Nomask }	Method of division (SH-2).	–
lang = c/cpp LOGO NOLOGO	Defines C variant: C89 / C ++ Disable of copyright output.	--c89/--c++/--ec++/--eec++, if none is specified, c99 is used.
Euc SJis LATin1	Character code select in string literals.	–
OUtcode = { EUc | SJis }	Japanese character code specified within object.	–
SUbcommand = <file name>	Subcommand file specified.	–

Assembler-specific details

Renesas SH	IAR Systems
Limitations in source code structure	
All segments are defined using ".SECTION" command.	Code segments are defined using the assembler directives SECTION or RSEG, which means segments. A CSTACK segment can also be defined.
.SECTION <name>,<attribute>,<ALIGN=[2|4|8]>	RSEG name:CODE[:flags] [(ALIGN=0-8)] RSEG name:DATA[:flags] [(ALIGN=0-8)] RSEG name:CONST[:flags] [(ALIGN=0-8)] or SECTION name:CODE[:flags] [(ALIGN=0-8)] SECTION name:DATA[:flags] [(ALIGN=0-8)] SECTION name:CONST[:flags] [(ALIGN=0-8)]
	Bit segments cannot be defined explicitly, but can easily be defined using bit operators in code or data segments. As a byte is the smallest allocatable memory segment, no memory is lost or gained using either tool.
Binary representation	
	Not supported, should be replaced by 0x0f.

Renesas SH		IAR Systems
Integer constants		
`B'1000`	Binary	`1010b, b'1010`
`Q'210`	Octal	`1234q, q'1234, 01234`
`D'136`	Decimal	`1234, -1, d'1234, 1234d`
`H'88`	Hexadecimal	`0FFFFh, 0xFFFF, h'FFFF`
Operand modifiers in assembler		
`STARTOF`	Section start address.	`SFB`
`SIZEOF`	Section size in bytes.	`SIZEOF`
`HIGH`	Extracts the high-order byte.	`HIGH`
`LOW`	Extracts the low-order byte.	`LOW`
`HWORD`	Extracts the high-order word.	`HWRD`
`LWORD`	Extracts the low-order word.	`LWRD`
Assembler directives		
`.CPU<target CPU>`	Specifies the target CPU.	–
`.SECTION<section name> [,<section attribute> [,<section type>]]`	Declares a section.	`SECTION <segment> :type [:flag] [(align)]`
`.ORG<location-counter value>`	Sets the value of the location counter.	–
`.ALIGN<boundary alignment value>`	Corrects the value of the location counter to a multiple of boundary alignment value.	`ALIGNRAM <align>`
`<symbol>[:].EQU<symbol value>`	Sets a symbol value.	`<label> EQU <expr>`
`<symbol>[:].ASSIGN<symbol value>`	Sets or resets a symbol value.	`<label> ASSIGN <expr>`
`<symbol>[:].REG<register name>`	Defines the alias of a register name.	–
`<symbol>[:].FREG<floating-point register name>`	Defines a floating-point register name.	–
`[<symbol>[:]].[{ B | W | L }]<integer data>[..]`	Reserves integer data.	`DC{8|16|32} <value>`
`[<symbol>[:]].DATAB[{ B | W | L }]<block count>,<integer data>`	Reserves an integer data block.	`DS{8|16|32} <count>`
`[<symbol>[:]].SDATA"<string literal>"[,...]`	Reserves string literal data.	`<symbol> DC8 ‘<string>’`
`[<symbol>[:]].SDATAB<block count>,"<string literal>"`	Reserves a string literal data block.	–
`[<symbol>[:]].SDATAC"<string literal>"[,...]`	Reserves string literal data (with length).	–
`[<symbol>[:]].SDATAZ"<string literal>"[,...]`	Reserves string literal data (with zero terminator).	`<symbol> DC8 “<string>”`
`[<symbol>[:]].FDATA[{ S | D }]<floating-point data>[,...]`	Reserves floating-point data.	`DF{32|64}`
`[<symbol>[:]].FDATAB[{ S | D }]<block count>, <floating-point data>`	Reserves a floating-point data block.	–
`[<symbol>[:]].XDATA[{ W | L }]<fixed-point data>[,...]`	Reserves fixed-point data.	–
`[<symbol>[:]].RES[{ B | W | L }]<area count>`	Reserves data area.	–
`[<symbol>[:]].SRES<string literal area size>[,...]`	Reserves string literal data area.	–
`[<symbol>[:]].SRESC<string literal area size>[,...]`	Reserves string literal data area (with length).	–
`[<symbol>[:]].SRESZ<string literal area size>[,...]`	Reserves string literal data area (with zero terminator).	–
`[<symbol>[:]].FRES[{ S | D }]<area count>`	Reserves floating-point data area.	–
`.EXPORT<symbol>[,...]`	Declares externally defined symbols.	`EXTERN <symbol> [,<symbol>]`
`.IMPORT<symbol>[,<symbol>..]`	Declares externally referenced symbols.	`IMPORT <symbol> [,<symbol>]`
`.GLOBAL<symbol>[,<symbol>..]`	Declares externally defined and externally referenced symbols.	`PUBLIC <symbol> [,<symbol>]`

`.OUTPUT { OBJ | NOOBJ | DBG | NODBG } [,...]`	Controls object module and debugging information output.	–
`.DEBUG{ ON | OFF }`	Controls the output of symbolic debugging information.	–
`.ENDIAN{ BIG | LITTLE}`	Selects big endian or little endian.	–
`.LINE ["<file name>",]<line number>`	Changes line number.	–
`.PRINT{ LIST | NOLIST | SRC | NOSRC | CREF | NOCREF | SCT | NOSCT } [,...]`	Controls assemble listing output.	–
`.LIST{ ON | OFF | COND | NOCOND | DEF | NODEF | CALL | NOCALL | EXP | NOEXP | CODE | NOCODE }[,...]`	Controls the output of the source program listing.	–
`.FORM{ LIN = <line count> | COL = <column count> | TAB = {4 | 8} }[,...]`	Sets the number of lines and columns in the assemble listing.	–
`.HEADING"<string literal>"`	Sets the header for the source program listing.	–
`.PAGE`	Inserts a new page in the source program listing.	–
`.SPACE[<line count>]`	Outputs blank lines to the source program listing.	–
`.PROGRAM<object module name>`	Sets the name of the object module.	`PROGRAM <symbol>`
`.RADIX{ B | Q | D | H }`	Sets the radix in which integer constants with no radix specifier are interpreted.	–
`.END[<symbol>]`	Specifies an entry point and the end of the source program.	`END`
`.STACK<symbol> = <stack value>`	Defines the stack value for the specified symbol.	–
Assembler options		
`Include = <path name>[, ...]`	Include file directory.	`-I<path>`
`DEFine = <replacement symbol> = "<string literal>" [, ...]`	Replacement symbol definition.	`-D<symbol>[=value]`
`ASsignA = <variable name> = <integer constant> [, ...]`	Integer preprocessor variable definition.	–
`ASsignC = <variable name> = "<string literal>" [, ...]`	Character preprocessor variable definition.	–
`Debug` `NODebug`	Debugging information.	`-r`
`EXPand [= <output file name>]`	Pre-processor expansion result.	–
`LITERAL = {Pool | Branch | Jump | Return} [, ...]`	Literal pool output point.	–

`Object [= <output file name>]`
`NOObject`
	Object module output.	`-o {filename|directory}`

`DIspsize = {4 | 12}`	Unresolved symbol size.	–

`LISt [= <output file name>]`
`NOLISt`
	Assemble listing output control.	`-l`

`SOurce` `NOSOurce`	Source program listing output control.	`-cA`

`SHow [={CONditionals | Definitions | CAlls | Expansions |  CODe | TAB={ 4 | 8 } } [, ...]]`
`NOSHow [={CONditionals | Definitions | CAlls | Expansions |  CODe | TAB={ 4 | 8 } } [, ...]]`
	Part of source program listing output control and tab size setting.	`-t<n>`

`CRoss_reference` `NOCRoss_reference`	Cross-reference listing output control.	`-x{D|I|2}`
`SEction` `NOSEction`	Section information listing output control.	–
`AUTO_literal`	Size mode specification for automatic literal pool generation.	–
`Exclude`	Preventing output of information on	–

NOExclude	unreferenced external symbols.	
CHKMd	Specification to check privileged-mode instructions.	–
CHKTlb	Specification to check LDTLB instructions.	–
CHKCache	Specification to check cache-related instructions.	–
CHKDsp	Specification to check DSP-related instructions.	–
CHKFpu	Specification to check FPU-related instructions.	–
CHKAlign8	Specification to check 8-byte boundary alignment of .FDATA.	–
CPU = <target CPU>	Target CPU specification.	--cpu <target_core>
ENdian = {Big | Little}	Endian type specification.	--endian={little|l|big|b}
Round = {Nearest | Zero}	Rounding direction of floating-point data.	–
DENormalize = {ON | OFF}	Handling denormalized numbers in floating-point data.	–
ABort = {Warning | Error}	Change of error level at which the assembler is abnormally terminated.	–
LATIN1	Western code character enabled.	–
SJIS	Interpretation of Japanese character as Shift JIS code.	–
EUC	Interpretation of Japanese character as EUC code.	–
OUtcode = {SJIS | EUC}	Specification of Japanese character.	–
LINes = <number of lines>	Setting of the number of lines in the assemble listing.	-p<lines>
COlumns = <number of digits>	Setting of the number of digits in the assemble listing.	–
LOGO NOLOGO	Copyright.	–
SUBcommand = <file name>	Specification of subcommand.	-f <filename>

Linker and library details

Renesas SH	IAR Systems
Device-specific header files	
All SFR are defined in header files named <device_number>.h . When a new project is created the corresponding header file is copied as iodefine.h to the project directory.	All SFRs are defined in ioxxx.h files.

Renesas SH		IAR Systems
Linker options		
Input = <file name> [(<module name>[,...])] [{,|}...]	Input file.	No specific option. Just list the files.
LIBrary = <file name>[,...]	Library file.	No specific option. Just list the files.
Binary = <file name>(<section name> [:<boundary alignment>] [/<section attribute>] [,<symbol name>]) [,...]	Binary file.	–
DEFine = <symbol name> = {<symbol name> |<numerical value>} [,...]	Symbol definition.	--define_symbol *symbol=constant_value*
ENTry = { <symbol name>| <address>}	Execution start address.	--entry <symbol>
NOPRElink	Pre-linker.	–
FOrm ={ Absolute | Relocate | Object | Library [= {S|U}] | Hexadecimal | Stype | Binary }	Output format.	Produces the ELF/DWARF format. To convert, use ielftool.exe.
DEBug	Debugging information.	Compiler option:

SDebug NODEBug		--debug
REcord={ H16 | H20 | H32 | S1 | S2 | S3 }	Record size unification.	–
ROm = <ROM section name> = <RAM section name> [,...]	ROM support function.	–
OUtput = <file name>[={ <start address>-<end address>| <section name>[:...]}][,...]	Output file.	-o <file name> --output <file name>
MAp [= <file name>]	External symbol-allocation information file.	--map {filename|directory}
SPace [= {<numerical value> | Random}]	Output to unused area.	–
Message NOMessage [=<error code> [-<error code>][,...]]	Information message.	--remarks
MSg_unused	Notification of unreferenced defined symbol.	–
DAta_stuff	Reduce empty areas of boundary alignment.	--no_fragments
BYte_count=<numerical value>	Specification of data record byte count.	–
CRc =<address>=<start address> - <end address>[,...] [/{ CCITT | 16 }] [:{BIG | LITTLE}]	Uses the checksum algorithm (CRC).	Is performed by ielftool.exe (--checksum) but space can be reserved with --place_holder symbol [,size[,section[,alignment]]]
PADDING	Filling padding data at section end.	–
VECTN = <vector number> = {<symbol> | <address>} [,...]	Address setting for specified vector number.	By default, the vector table is populated with a *default interrupt handler* which calls the abort function. For each interrupt source that has no explicit interrupt service routine, the default interrupt handler will be called. If you write your own service routine for a specific vector, that routine will override the default interrupt handler.
VECT={<symbol>|<address>}	Address setting for unused variable vector area.	See above.
LISt [= <file name>]	List file.	--map {file|directory}
SHow [= {SYmbol | Reference | SEction | Xreference | Total_size | VECTOR | ALL } [,...]]	List contents.	–

OPtimize = {STring_unify | SYmbol_delete | Variable_access | Register | SAMe_code | SHort_format | Function_call | Branch | Speed | SAFe } [...]
NOOPtimize}
	Optimization.	--inline
		--vfe=[forced]

SAMESize = <size> (default: sames=1e)	Minimum size to unify same codes.	–
PROfile = <file name>	Profile information file.	–
CAchesize=Size=<size> | Align=<line size> (default: ca=s=8,a=20)	Cache size.	–

SYmbol_forbid= <symbol name>[,...]
SAMECode_forbid= <function name>[,...]
Variable_forbid= <symbol name>[,...]
FUnction_forbid= <function name>[,...]
	Optimization partially disabled.	#pragma optimize=[goal] [level][no_optimization...]

SEction_forbid = [<file name>|<module name>] (<section name>[,...]) [,...]
 	
Absolute_forbid=<address>[+<size>][,...]		

STARt = [(]<section name> [{ : | , }<section name>[,...]] [)][,...] [/<address>] [,...]	Section address.	Done in linker configuration file with the place in directive.
FSymbol = <section name>[,...]	Symbol address file.	–
ALIGNED_SECTION = <section name>[,...]	Section alignment specification.	–
CPu = { <cpu information file name> | { ROm | RAm | XROm | XRAm | YROm | YRAm } = <start address> -<end address> [,...] | STRIDE}	Address check.	–
PS_check=<start address> -<end address>,<start address> -<end address> [,...] [:<start address> -<end address>,<start address> -<end address> [,...] ...]	Physical space overlap check.	–
CONTIGUOUS_SECTION = <section name>[,...]	Specifies section not to be divided.	–
S9	Always output S9 recode as end code.	–
STACk	Output stack information file.	–
Compress NOCOmpress	Debug information compression.	–
MEMory = [High | Low]	Memory = [High | Low].	–
REName = {<file name> (<name> = <name> [,...]) | <module name> (<name> <name>[,...])}[,...]	Symbol name modification.	--redirect <from_symbol>=<to_symbol>
DELete = {<module name> | [<file name>] (<name>[,...])}[,...]	Symbol name deletion.	–
REPlace = <file> [(<module> [,...])] [,...]	Module replacement.	–
EXTract = <module>[,...]	Module extraction.	–
STRip	Debugging information deletion.	--strip
CHange_message={Information | Warning | Error } [=<error number> [-<error number>] [,...]] [,...]	Message level.	--diag_error=tag [,tag,...] --diag_remark=tag [,tag,...] --diag_suppress=tag [,tag,...] --diag_warning=tag [,tag,...]
Hide	Local name hide.	–
Total_size	Showing total size of sections.	–
RTs_file	Information file for the emulator.	–
SUbcommand = <file name>	Subcommand file.	-f <filename>
LOgo NOLOgo	Copyright message.	–
END	Executes option strings already input, inputs continuing option strings and continues processing.	–
EXIt	Specifies the termination of option input.	–
Segments/Sections		
B / B$1 / B$2 / B$4	BSS section: uninitialized data, variable size 1/2/4 byte.	.bss
D / D$1 / D$2 / D$4	Data section: initialized data, variable size 1/2/4 byte.	.data
P	Program section.	.text
R	ROM section: initialization data for “D” , alignment 4/2/1 byte.	.data_init

C / C$1 / C$2 / C$4	Constant section, variable size 1/2/4 byte.	.rodata
C$INIT	C++ initial processing/ postprocessing data area.	.init_array
C$VTBL	C++ virtual function table area	–
S	Stack area.	CSTACK
$G0 / $G1	GBR section.	–
$TBR	TBR table section.	–
$ADDRESS $<section> <address>	Stores variables defines using #pragma address.	–

Runtime environment

Renesas SH	IAR Systems
Calling convention	
Parameters passed on the stack	
(1) Parameters whose types are other than target types for register passing	When there are more parameters then registers in a function, all parameters that do not fit in the registers are passed on the stack.
(2) Parameters of a function which has been declared by a prototype declaration to have variable-number parameters	
(3) When other parameters are already allocated to R4 to R7.	
(4) When other parameters are already allocated to FR4 (DR4) to FR11 (DR10).	
(5) long long type and unsigned long long type parameters	
(6) __fixed type, long __fixed type, __accum type, and long __accum type parameters	
Parameters passed in registers	
8-bit values in: R4–R7	8-bit values in: R0–R3
16-bit values in: R4–R7	16-bit values in: R0–R3
	24-bit values in: R0–R3
32-bit values in: R4–R7	32-bit values in: R0–R3
Floating-point values (32 bit) in: R4–R7 (not SH-2E, SH2A-FPU)	Floating-point values (32 bit) in: R0–R3
Floating-point values (32 bit) in: FR4–FR11 (SH-2E, SH2A-FPU)	-
Floating-point values (64 bit) in: DR4–DR10 (SH2A-FPU)	Floating-point values (64 bit) in: R0:R1–R2:R3
Return values	
8-bit values in: R0	8-bit values in: R0
16-bit values in: R0	16-bit values in: R0
	24-bit values in: R0
32-bit values in: R0	32-bit values in: R0
Floating-point values (32 bit) in: R0 (not SH-2E, SH2A-FPU)	Floating-point values (32 bit) in: R0
Floating-point values (32 bit) in: FR0 (SH-2E, SH2A-FPU)	-
Floating-point values (64 bit) in: DR0 (SH2A-FPU)	Floating-point values (64 bit) in: R0:R1
Preserved registers	
R8–R15, MACH, MACL, PR, FR12–FR15, DR12–DR14	R4–R11
Scratch registers	
R0–R7, FR0–FR11, DR0–DR11, FPUL, FPSCR, A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1, DSR, MOD, RS, RE	R0–R3 and R12
System startup and exit code	
The system startup code is located in resetprg.c and uses dbsct.c. Customized hardware initialization may be placed in function HardwareSetup() in file hwsetup.c. Interrupt vectors and interrupt functions are predefined for all possible interrupt sources. These can be found in intprg.c and vecttbl.c.	The system startup code is located in the ready-made cstartup.s file. In addition, you specify additional settings, for example for the stack and heap size in the linker configuration file. It is likely that you need to customize the code for system initialization. For example, your application need to initialize memory-mapped special function registers, or omit the default initialization of data segments performed by

	cstartup. You can do this by providing a customized version of the routine __low_level_init, which is called from cstartup before the data segments are initialized. Modifying cstartup directly should be avoided.
Global variable initialization	
Static and global variables are initialized: zero-initialized variables are cleared and the values of other initialized variables are copied from ROM to RAM memory. The file dbsct.c holds some arrays which define which sections should be initialized, and whether the section contents has to be copied or has to be cleared.	Static and global variables are initialized: zero-initialized variables are cleared and the values of other initialized variables are copied from ROM to RAM memory. This initialization can be overrided by returning 0 from the __low_level_init function. Variables declared __no_init are not initialized at all: __no_init int i;
Reentrancy and recursive functions	
The SH compiler does not have precompiled libraries, but always builds application specific library files based on the selected header files. The library generator has an option for creating reentrant library files.	The compiler is always reentrant when using the DLIB library.
Other operations	

All information is subject to change without notice. IAR Systems assumes no responsibility for errors and shall not be liable for any damage or expenses.