ARM® Compiler toolchain v4.1 for µVision

Migration and Compatibility
ARM Compiler toolchain v4.1 for µVision
Migration and Compatibility

Copyright © 2011 ARM. All rights reserved.

Release Information

The following changes have been made to this book.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2011</td>
<td>A</td>
<td>Non-Confidential</td>
<td>Release for ARM Compiler toolchain v4.1 for µVision</td>
</tr>
</tbody>
</table>

Proprietary Notice

Words and logos marked with or are registered trademarks or trademarks of ARM in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com
Contents
ARM Compiler toolchain v4.1 for μVision Migration and Compatibility

Chapter 1 Conventions and feedback

Chapter 2 Migrating from RVCT v4.0 for μVision to ARM Compiler v4.1 for μVision
2.1 General changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-2
2.2 Compiler changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-3
2.3 Linker changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-4
2.4 Assembler changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-7
2.5 C and C++ library changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-9
2.6 fromelf changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision .................................................. 2-10

Chapter 3 Migrating from RVCT v3.1 for μVision to RVCT v4.0 for μVision
3.1 General changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-2
3.2 Changes to symbol visibility between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-3
3.3 Compiler changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-5
3.4 Linker changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-6
3.5 Assembler changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-11
3.6 fromelf changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-12
3.7 C and C++ library changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision .................................................. 3-13
Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions
The following typographical conventions are used:

*monospace* Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

*monospace* Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

*monospace* _italic_ Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

*monospace* _bold_ Denotes language keywords when used outside example code.

_italic_ Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

_bold_ Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product
If you have any comments and suggestions about this product, contact your supplier and give:

* your name and company
• the serial number of the product
• details of the release you are using
• details of the platform you are using, such as the hardware platform, operating system type and version
• a small standalone sample of code that reproduces the problem
• a clear explanation of what you expected to happen, and what actually happened
• the commands you used, including any command-line options
• sample output illustrating the problem
• the version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:
• the title
• the number, ARM DUI 0593A
• if viewing online, the topic names to which your comments apply
• if viewing a PDF version of a document, the page numbers to which your comments apply
• a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and Frequently Asked Questions (FAQs).

Other information

Chapter 2

Migrating from RVCT v4.0 for µVision to ARM Compiler v4.1 for µVision

The following topics describe the changes that affect migration and compatibility between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision:

- General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-2
- Compiler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-3
- Linker changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-4
- Assembler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-7
- C and C++ library changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-9.
2.1 General changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision

The convention for naming environment variables, such as those for setting default header and library directories, has changed. These are now prefixed with ARMCC rather than RVCT. For example, ARMCC41INC rather than RVCT40INC.

2.1.1 See also

Reference

Introducing the ARM® Compiler toolchain:

- Toolchain environment variables on page 2-12.
2.2 Compiler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The following changes to the compiler have been made:

Sign rules on enumerators have changed in line with convention. The enumerator container is now unsigned unless a negative constant is defined.

-O3 no longer implies --multifile. The --multifile option has always been available as a separate option and it is recommended you put this into your builds.

The compiler faults use of the at attribute when it is used on declarations with incomplete non-array types. For example, if foo is not declared, the following causes an error:

```c
struct foo a __attribute__((at(0x16000)));
```

2.2.1 See also

Concepts

• General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-2.

Reference

Compiler Reference:

• --multifile, --no_multifile on page 3-67
• -Onum on page 3-71.
• __attribute__((at(address))) variable attribute on page 5-40.
2.3 Linker changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision

The following changes to the linker have been made:

ARM/Thumb synonyms removed

Support for the deprecated feature ARM/Thumb Synonyms has been removed in 4.1. The ARM/Thumb synonyms feature permits an ARM Global Symbol Definition of symbol \( S \) and a Thumb Global Symbol Definition of \( S \) to coexist. All branches from ARM state are directed at the ARM Definition, all branches from Thumb state are directed at the Thumb Definition.

In 4.0 armlink gives a deprecated feature warning L6455E when it detects ARM/Thumb Synonyms.

In 4.1 armlink gives an error message L6822E when it detects ARM/Thumb Synonyms.

To recreate the behavior of synonyms rename both the ARM and Thumb definitions and relink. For each undefined symbol you have to point it at the ARM or the Thumb synonym.

String merging enhancements

The linker can merge strings where the address is generated by \texttt{MOVT} and \texttt{MOVW} instructions.

Placing and sorting input sections

Additional command-line options are available to control the placement and the sort order of input sections when using the .ANY module selector:

- \texttt{--any_contingency}
- \texttt{--any_placement=algorithm}
- \texttt{--any_sort_order=order}.

The \texttt{--info=any} option is provided to show details of the sorting and placement of .ANY sections.

Using a tiebreaker when sorting input sections with equal properties

In RVCT v4.0 for μVision and earlier, the sorting algorithm used by the linker is based on the order that the linker creates sections in its section data structure.

In ARM Compiler v4.1 for μVision and later, an additional sorting algorithm is available based on the order that sections appear on the linker command-line.

You select the algorithm using the \texttt{--tiebreaker} command-line option. For backwards compatibility, the section create order is the default.

The \texttt{--section_index_display=cmdline} option is provided to show the sections in the memory map output in the same order they appear on the command-line.

Controlling veneer sharing across execution regions

The \texttt{--[no]crosser_veneershare} command-line option is available to enable and disable veneer sharing across execution regions.

Linker-defined symbols

The linker now supports the following linker-defined symbols:

\texttt{Load$$region_name$$ZI$$BaseLoad$$region_name$$ZI$$LimitLoad$$region_name$$ZI$$Length}
Additional command-line options supported

The following command-line options have been added:

- `--emit_non_debug_relocs`
- `--[no_]load_addr_map_info`
- `--[no_]strict_flags`
- `--[no_]strict_symbols`
- `--[no_]strict_visibility`
- `--veneer_inject_type`
- `--veneer_pool_size`.

Changes to command-line options

The following changes have been made to existing command-line options:

- the `any` and `veneer pools` topics have been added to `--info`
- the `cmdline` type has been added to `--section_index_display`
- the `Alignment`, `BreadthFirstCallTree` and `LexicalState` algorithms have been added to `--sort`

Changes to scatter files

The following changes have been made to scatter-loading:

- `+offset` can be used in a conditional operator
- execution address built-in functions can now be used for the `max_size` of a region.

2.3.1 See also

Concepts

Introducing the ARM Compiler toolchain:

- About the ARM Compiler toolchain on page 2-3.
- General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-2.

Using the Linker:

- Load$$ execution region symbols on page 7-7
- Placing unassigned sections with the .ANY module selector on page 8-23.

Linker Reference:

- Load region attributes on page 4-7
- Execution region attributes on page 4-11
- Considerations when using a relative address +offset for load regions on page 4-16
- Considerations when using a relative address +offset for execution regions on page 4-17.

Reference

Linker Reference:

- `--any_contingency` on page 2-5
- `--any_placement=algorithm` on page 2-6
- `--any_sort_order=order` on page 2-8
- `--crosser_veneershare`, `--no_crosser_veneershare` on page 2-28
- `--emit_non_debug_relocs` on page 2-40
- `--info=topic[,topic,...]` on page 2-59
- `--load_addr_map_info`, `--no_load_addr_map_info` on page 2-79
• --section_index_display=type on page 2-111
• --sort=algorithm on page 2-116
• --strict_flags, --no_strict_flags on page 2-122
• --strict_symbols, --no_strict_symbols on page 2-125
• --strict_visibility, --no_strict_visibility on page 2-126
• --tiebreaker=option on page 2-133
• --veneer_inject_type=type on page 2-139
• --veneer_pool_size=size on page 2-140.
2.4 Assembler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The following changes to the assembler have been made:

Change to the way the assembler reads and processes files

Older assemblers sometimes permitted the source file being assembled to vary between the two passes of the assembler. In the following example, the symbol `num` is defined in the second pass because the symbol `foo` is not defined in the first pass.

```
AREA x,CODE
[ :DEF: foo
num EQU 42
]
foo DCD num
END
```

The way the assembler reads and processes the file has now changed, and is stricter. You must rewrite code such as this to ensure that the path through the file is the same in both passes.

The following code shows another example where the assembler faults:

```
AREA FOO, CODE
IF :DEF: VAR
ELSE
VAR EQU 0
EDNIF
END
```

The resulting error is:

```
Error: A1903E : Line not seen in first pass; cannot be assembled.
```

To avoid this error, you must rewrite this code as:

```
AREA FOO, CODE
IF :LNOT: :DEF: VAR
VAR EQU 0
EDNIF
END
```

Change to messages output by the assembler

Generally, any messages referring to a position on the source line now has a caret character pointing to the offending part of the source line, for example:

```
"foo.s", line 3 (column 19): Warning: A1865W: '#' not seen before constant expression
```

Changes to diagnostic messages

Various instructions in ARM (using SP) were deprecated when 32-bit Thumb instructions were introduced. These instructions are no longer diagnosed as deprecated unless assembling for a CPU that has 32-bit Thumb instructions. To enable the warnings on earlier CPUs, you can use the option

```
--diag_warning=1745,1786,1788,1789,1892
```

Obsolete command-line option

The `-O` command-line option is obsolete. Use `-o` instead.
2.4.1 See also

Concepts
• General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-2.

Reference
Assembler Reference:
• --diag_warning=tag{, tag} on page 2-11
• -o filename on page 2-19.

Using the Assembler:
• How the assembler works on page 2-4
• 2 pass assembler diagnostics on page 7-20.
2.5 C and C++ library changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The libraries now use more Thumb2 code on targets that support Thumb2. This is expected to result in reduced code size without affecting performance. The linker option --no_thumb2_library falls back to the old-style libraries if necessary.

Math function returns in some corner cases now conform to POSIX/C99 requirements. You can enable older behavior with:

#pragma import __use_rvct_matherr

You can enable the newer behavior with:

#pragma import __use_c99_matherr.

The symbol __use_accurate_range_reduction is retained for backward compatibility, but no longer has any effect.

The C99 complex number functions in the previous hardware floating point version of the library only had the hardfp linkage functions and not the softfp linkage functions. The new library has both the hardfp linkage and softfp linkage functions. This means that existing object code that was built to use hardware floating point might not function correctly when calling complex functions from the library. The linker issues a warning in this case. You must recompile all the code that might use the affected functions and that was built to use hardware floating point. You must relink them with the new library.

The new implementation of alloca() allocates memory on the stack and not on the heap. This does not cause compatibility problems with software that assumes the old heap-based implementation of alloca(). However, such software might contain operations that are no longer required, such as implementing __user_perthread_libspace for the alloca state that is no longer used.

2.5.1 See also

Concepts
- General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision on page 2-2.

Using ARM® C and C++ Libraries and Floating-Point Support:
- Chapter 2 The ARM C and C++ libraries.
- How the ARM C library fulfills ISO C specification requirements on page 2-107.
- Library heap usage requirements of the ARM C and C++ libraries on page 2-8
- Use of the __user_libspace static data area by the C libraries on page 2-21
- Building an application without the C library on page 2-41.

Reference
ARM® C and C++ Libraries and Floating-Point Support Reference:
- Chapter 2 The C and C++ libraries.

Compiler Reference:
- --apcs=qualifier...qualifier on page 3-7
- --fpv=name on page 3-44.
2.6 fromelf changes between RVCT v4.0 for μVision and ARM Compiler v4.1 for μVision

fromelf can now process all files, or a subset of files, in an archive.

2.6.1 See also

Reference

Using the fromelf Image Converter:

• input_file on page 4-37.
Chapter 3
Migrating from RVCT v3.1 for µVision to RVCT v4.0 for µVision

The following topics describe the changes that affect migration and compatibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision:

- General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2
- Changes to symbol visibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-3
- Compiler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-5
- Linker changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-6
- Assembler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-11
- fromelf changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-12
- C and C++ library changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-13.
3.1 General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes affect multiple tools:

3.1.1 Restrictions on --fpu

--fpu=VFPv2 or --fpu=VFPv3 are only accepted if CPU architecture is greater than or equal to ARMv5TE. This affects all tools that accept --fpu.

Note

The assembler assembles VFP instructions when you use the --unsafe option, so do not use --fpu when using --unsafe. If you use --fpu with --unsafe, the assembler downgrades the reported architecture error to a warning.

3.1.2 Remove support for v5TExP and derivatives, and all ARMv5 architectures without T

The following --cpu choices are obsolete and have been removed:

• 5
• 5E
• 5ExP
• 5EJ
• 5EWMMX2
• 5EWMMX
• 5TEx
• ARM9E-S-rev0
• ARM946E-S-rev0
• ARM966E-S-rev0.

3.1.3 See also

Reference

Compiler Reference:
• --cpu=name on page 3-20
• --fpu=name on page 3-44.

Linker Reference:
• --cpu=name on page 2-27
• --fpu=name on page 2-57.

Assembler Reference:
• --cpu=name on page 2-8
• --fpu=name on page 2-13
• --unsafe on page 2-23.
3.2 Changes to symbol visibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes to symbol visibility have been made:

3.2.1 Change to ELF visibility used to represent __declspec(dllexport)

When using the --hide_all compiler command-line option, which is the default, the ELF visibility used to represent __declspec(dllexport) in RVCT v3.1 for µVision and earlier was STV_DEFAULT. In RVCT v4.0 for µVision it is STV_PROTECTED. Symbols that are STV_PROTECTED can be referred to by other DLLs but cannot be preempted at load-time.

When using the --no_hide_all command-line option, the visibility of imported and exported symbols is still STV_DEFAULT as it was in RVCT v3.1 for µVision.

3.2.2 __attribute(visibility(...))

The GNU-style __attribute(visibility(...)) has been added and is available even without specifying the --gnu compiler command-line option. Using it overrides any implicit visibility. For example, the following results in STV_DEFAULT visibility instead of STV_HIDDEN:

__declspec(visibilty("default")) int x = 42;

3.2.3 RVCT v3.1 for µVision symbol visibility summary

The following tables summarize the visibility rules in RVCT v3.1 for µVision:

<table>
<thead>
<tr>
<th>Code</th>
<th>--hide_all (default)</th>
<th>--no_hide_all</th>
<th>--dlllexport_all</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern int x;</td>
<td>STV_HIDDEN</td>
<td>STV_DEFAULT</td>
<td>STV_HIDDEN</td>
</tr>
<tr>
<td>extern int g(void);</td>
<td>STV_HIDDEN</td>
<td>STV_DEFAULT</td>
<td>STV_HIDDEN</td>
</tr>
<tr>
<td>extern int y = 42;</td>
<td>STV_HIDDEN</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
</tr>
<tr>
<td>extern int f() { return g() + x; }</td>
<td>STV_HIDDEN</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
</tr>
<tr>
<td>__declspec(dllexport) extern int imx;</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
</tr>
<tr>
<td>__declspec(dllexport) extern int img(void);</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
</tr>
<tr>
<td>__declspec(dllexport) extern int exy = 42;</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
<td>STV_DEFAULT</td>
</tr>
<tr>
<td>__declspec(dllexport) extern int exf() { return img() + imx; }</td>
<td>STV_HIDDEN</td>
<td>STV_HIDDEN</td>
<td>STV_HIDDEN</td>
</tr>
</tbody>
</table>

/* exporting undefs (unusual?) */
| __declspec(dllexport) extern int exz; | STV_HIDDEN | STV_HIDDEN | STV_HIDDEN |
| __declspec(dllexport) extern int exh(void); | STV_HIDDEN | STV_HIDDEN | STV_HIDDEN |

Table 3-2 RVCT v3.1 for µVision symbol visibility summary for references to run-time functions

<table>
<thead>
<tr>
<th>Code</th>
<th>--no_dllimport_runtime</th>
<th>--hide_all (default)</th>
<th>--no_hide_all</th>
<th>--dlllexport_all</th>
</tr>
</thead>
</table>
| /* references to runtime functions, for example __aeabi_fmul */
| float fn(float a, float b) { return a*b; } | STV_HIDDEN | STV_DEFAULT | STV_DEFAULT |
3.2.4 RVCT v4.0 for µVision symbol visibility summary

The following tables summarize the visibility rules in RVCT v4.0 for µVision:

<table>
<thead>
<tr>
<th>Code</th>
<th>--hide_all (default)</th>
<th>--no_hide_all</th>
<th>--dllexport_all</th>
</tr>
</thead>
</table>
| extern int x;  
extern int g(void); | STV_HIDDEN | STV_DEFAULT | STV_HIDDEN |
| extern int y = 42;  
extern int f() { return g() + x; } | STV_HIDDEN | STV_DEFAULT | STV_PROTECTED |
| __declspec(dllimport) extern int imx;  
__declspec(dllimport) extern int img(void); | STV_DEFAULT | STV_DEFAULT | STV_DEFAULT |
| __declspec(dllexport) extern int exy = 42;  
__declspec(dllexport) extern int exf() { return img() + imx; } | STV_PROTECTED | STV_PROTECTED | STV_PROTECTED |
| /* exporting undefs (unusual?) */  
__declspec(dllexport) extern int exz;  
__declspec(dllexport) extern int exh(void); | STV_PROTECTED | STV_PROTECTED | STV_PROTECTED |

<table>
<thead>
<tr>
<th>Code</th>
<th>--no_dllimport-runtime --hide_all (default)</th>
<th>--no_hide_all</th>
<th>--dllexport_all</th>
</tr>
</thead>
</table>
| /* references to runtime functions, for example __aeabi_fmul */  
float fn(float a, float b) { return a*b; } | STV_HIDDEN | STV_DEFAULT | STV_DEFAULT |

3.2.5 See also

Concepts

* General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2.

Reference

Compiler Reference:

* --default_definition_visibility=visibility on page 3-24
* --hide_all, --no_hide_all on page 3-49.
3.3 Compilation changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision

The following compiler changes have been made:

3.3.1 Single compiler executable

The executables tcc, armcpp and tcpp are no longer delivered.

To compile for Thumb, use the --thumb command-line option.

To compile for C++, use the --cpp command-line option.

--- Note ---

The compiler automatically selects C++ for files with the .cpp extension, as before.

3.3.2 VAST Changes

VAST has been upgraded through two versions (VAST 11 for 4.0 Alpha and 4.0 Alpha2 and later). Apart from the following issue, you do not have to make any changes to your v3.1 builds to use the new VAST.

RVCT v3.1 for μVision reassociated saturating ALU operations. This meant programs like the following could produce different results with --vectorize and --no_vectorize:

```c
int g_448464(short *a, short *b, int n)
{
    int i; short s = 0;
    for (i = 0; i < n; i++)  s = L_mac(s, a[i], b[i]);
    return s;
}
```

In RVCT v4.0 for μVision, you might see a performance degradation because of this issue.

The --reassociate_saturation and --no_reassociate_saturation command-line options have been added to permit reassociation to occur.

3.3.3 See also

Concepts

- General changes between RVCT v3.1 for μVision and RVCT v4.0 for μVision on page 3-2.

Reference

Compiler Reference:

- --cpp on page 3-20
- --reassociate_saturation, --no_reassociate_saturation on page 3-80
- --thumb on page 3-90
3.4  Linker changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following linker changes have been made:

3.4.1  Placing ARM library helper functions with scatter files

In RVCT v3.1 for µVision and earlier, the helper functions reside in libraries provided with the
ARM compiler. Therefore, it was possible to use armlib and cpplib in a scatter file to inform the
linker where to place these helper functions in memory.

In RVCT v4.0 for µVision and later, the helper functions are generated by the compiler in the
resulting object files. They are no longer in the standard C libraries, so it is no longer possible
to use armlib and cpplib in a scatter file. Instead, place the helper functions using *.*
(i._ARM_*) in your scatter file.

3.4.2  Linker steering files and symbol visibility

In RVCT v3.1 for µVision the visibility of a symbol was overridden by the steering file or
.directive commands IMPORT and EXPORT. When this occurred the linker issued a warning
message, for example:

Warning: L6780W: STV_HIDDEN visibility removed from symbol hidden_symbol through EXPORT.

In RVCT v4.0 for µVision the steering file mechanism respects the visibility of the symbol, so
an IMPORT or EXPORT of a STV_HIDDEN symbol is ignored. You can restore the v3.1 behavior
with the --override_visibility command-line option.

3.4.3  Linker-defined symbols

In the majority of cases region related symbols behave identically to v3.1.

Section-relative symbols

The execution region Base and Limit symbols are now section-relative. There is
no sensible section for a $Length symbol so this remains absolute.

This means that the linker-defined symbol is assigned to the most appropriate
section in the execution region. The following example shows this:

ExecRegion ER

RO Section 1 ; Image$ERS$Base and Image$ERS$RO$Base, val 0
RO Section 2 ; Image$ERS$RO$Limit, val Limit(RO Section 2)

RW Section 1 ; Image$ERS$RW$Base, val 0
RW Section 2 ; Image$ERS$Limit and Image$ERS$RW$Limit,
val Limit(RW Section 2)

ZI Section 1 ; Image$ERS$ZI$Base, val 0
ZI Section 2 ; Image$ERS$ZI$Limit, val Limit(ZI Section 2)

In each case the value of the ...$Length symbol is the value of the ...$Limit
symbol minus the ...$Base symbol.

If there is no appropriate section that exists in the execution region then the linker
defines a zero-sized section of the appropriate type to hold the symbols.

Impact of the change

The change to section-relative symbols removes several special cases
from the linker implementation, that might improve reliability.
Alignment

The \$Limit symbols are no longer guaranteed to be four-byte aligned because the limit of the section it is defined in might not be aligned to a four-byte boundary.

This might affect you if you have code that accidentally relies on the symbol values being aligned. If you require an aligned \$Limit or \$Length then you must align the symbol value yourself.

For example, the following legacy initialization code might fail if Image\$<Region_Name>$Length is not word aligned:

```
LDR R1, |Load\$region_name$Base|
LDR R0, |Image\$region_name$Base|
LDR R4, |Image\$region_name$Length|
ADD R4, R4, R0
copy_rw_data
LDRNE R3, [R1], #4
STRNE R3, [R0], #4
CMP R0, R4
BNE copy_rw_data
```

Writing your own initialization code is not recommended, because system initialization is more complex than in earlier toolchain releases. ARM recommends that you use the \$__main code provided with the ARM Compiler toolchain.

Delayed Relocations

The linker has introduced an extra address assignment and relocation pass after RW compression. This permits more information about load addresses to be used in linker-defined symbols.

Be aware that:

- Load\$region_name$Base is the address of \$region_name prior to C-library initialization
- Load\$region_name$Limit is the limit of \$region_name prior to C-library initialization
- Image\$region_name$Base is the address of \$region_name after C-library initialization
- Image\$region_name$Limit is the limit of \$region_name after C-library initialization.

Load Region Symbols have the following properties:

- They are ABSOLUTE because section-relative symbols can only have Execution addresses.
- They take into account RW compression
- They do not include ZI because it does not exist prior to C-library initialization.

In addition to Load\$Base, the linker now supports the following linker-defined symbols:

```
Load\$region_name$BaseLoad\$region_name$LimitLoad\$region_name$LengthLoad\$region_name$RO$BaseLoad\$region_name$RO$LimitLoad\$region_name$RO$LengthLoad\$region_name$RW$BaseLoad\$region_name$RW$LimitLoad\$region_name$RW$Length
```

Limits of Delayed Relocation

All relocations from RW compressed execution regions must be performed prior to compression because the linker cannot resolve a delayed relocation on compressed data.
If the linker detects a relocation from a RW-compressed region \texttt{REGION} to a linker-defined symbol that depends on RW compression then the linker disables compression for \texttt{REGION}.

**Load Region Symbols**

RVCT v4.0 for \uVision now permits linker-defined symbols for load regions. They follow the same principle as the \texttt{Load$$} symbols for execution regions. Because a load region might contain many execution regions it is not always possible to define the \$RO and \$RW components. Therefore, load region symbols only describe the region as a whole.

\begin{verbatim}
Load$$LR$$Load_Region_Name$$Base  ; Base address of <Load Region Name>
Load$$LR$$Load_Region_Name$$Limit ; Load Address of last byte of content
\end{verbatim}

**Image-related symbols**

The RVCT v4.0 for \uVision linker implements these in the same way as the execution region-related symbols. They are defined only when scatter files are not used.

\begin{verbatim}
Image$$RO$$Base  ; Equivalent to Image$$ER_RO$$Base
Image$$RO$$Limit ; Equivalent to Image$$ER_RO$$Limit
Image$$RW$$Base  ; Equivalent to Image$$ER_RW$$Base
Image$$RW$$Limit ; Equivalent to Image$$ER_RW$$Limit
Image$$ZI$$Base  ; Equivalent to Image$$ER_ZI$$Base
Image$$ZI$$Limit ; Equivalent to Image$$ER_ZI$$Limit
\end{verbatim}

**Interaction with ZEROPAD**

An execution region with the ZEROPAD keyword writes all ZI data into the file:

- \texttt{Image$$} symbols define execution addresses post initialization. In this case, it does not matter that the zero bytes are in the file or generated. So for \texttt{Image$$} symbols, ZEROPAD does not affect the values of the linker-defined symbols.
- \texttt{Load$$} symbols define load addresses pre initialization. In this case, any zero bytes written to the file are visible. Therefore, the \texttt{Limit} and \texttt{Length} take into account the zero bytes written into the file.

### 3.4.4 Build attributes

The RVCT v4.0 for \uVision linker fully supports reading and writing of the ABI Build Attributes section. The linker can now check more properties such as \texttt{wchar_t} and \texttt{enum} size. This might result in the linker diagnosing errors in old objects that might have inconsistencies in the Build Attributes. Most of the Build Attributes messages can be downgraded to permit \texttt{armlink} to continue.

The \texttt{--cpu} option now checks the FPU attributes if the CPU chosen has a built-in FPU. For example, \texttt{--cpu=Cortex-R4F} implies \texttt{--fpu=vfpv3_d16}. In RVCT v3.1 for \uVision the \texttt{--cpu} option only checked the build attributes of the chosen CPU.

The error message \texttt{L6463E: Input Objects contain archtype instructions but could not find valid target for archtype architecture based on object attributes. Suggest using --cpu option to select a specific cpu.} is given in one of two situations:

- the ELF file contains instructions from architecture archtype yet the Build Attributes claim that archtype is not supported
- the Build Attributes are inconsistent enough that the linker cannot map them to an existing CPU.
If setting the --cpu option still fails, the option --force_explicit_attr causes the linker to retry the CPU mapping using Build Attributes constructed from --cpu=archtype. This might help if the Error is being given solely because of inconsistent Build Attributes.

3.4.5 C library initialization

A change to the linker when dealing with C library initialization code causes specially named sections in the linker map file created with the --map command-line option. You can ignore these specially named sections.

3.4.6 RW compression

Some error handling code is run later so that information from RW compression can be used. In almost all cases, this means more customer programs are able to link. There is one case where RVCT v4.0 for µVision has removed a special case so that it could diagnose more RW compression errors.

Multiple in-place execution regions with RW compression are no longer a special case. It used to be possible to write:

```c
LR1 0x0
    { 
        ER1 +0 { file1.o(+RW) }
        ER2 +0 { file2.o(+RW) }
    }
```

This is no longer possible under v4.0 and the linker gives an error message that ER1 decompresses over ER2. This change has been made to permit the linker to diagnose:

```c
LR1 0x0
    { 
        ER1 +0 { file1.o(+RW) }
        ER2 +0 { file2.o(+RO) } ; NOTE RO not RW
    }
```

This fails at runtime on RVCT v3.1 for µVision.

3.4.7 See also

Concepts
- General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2.

Using the Linker:
- Optimization with RW data compression on page 5-12
- Accessing linker-defined symbols on page 7-4
- Region-related symbols on page 7-5
- Image$$ execution region symbols on page 7-6
- Load$$ execution region symbols on page 7-7
- Importing linker-defined symbols in C and C++ on page 7-12
- Section-related symbols on page 7-14
- Example of placing ARM library helper functions on page 8-49.

Using ARM C and C++ Libraries and Floating-Point Support:
- Initialization of the execution environment and execution of the application on page 2-55.
Reference

Linker Reference:

- `--cpu=name` on page 2-27
- `--force_explicit_attr` on page 2-54
- `--fpu=name` on page 2-57
- `--map, --no_map` on page 2-83
- `--override_visibility` on page 2-90
- `EXPORT` on page 3-2
- `IMPORT` on page 3-4
- `Execution region attributes` on page 4-11.
3.5 Assembler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes to the assembler have been made:

- The -O command-line option is deprecated. -O is a synonym for -o to output to a named file. This has been deprecated to avoid user confusion with the armcc option with the same name.
- The -D command-line option is obsolete. Use --depend instead.
- LDM r0!, {r0-r4} no longer ignores writeback. Previously in Thumb, LDM r0!, {r0-r4} assembled with a warning to the 16-bit LDM instruction and no writeback was performed. Because the syntax requests writeback, and this encoding is only available in Thumb-2, it produces an error. To get the 16-bit Thumb instruction you must remove the writeback.
- The logical operator | is deprecated because it can cause problems with substitution of variables in source. The assembler warns on the first use of | as a logical operator. Use the :OR: operator instead.

3.5.1 See also

Concepts
- General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2.

Reference
Using the Assembler:
- Addition, subtraction, and logical operators on page 8-26.

Assembler Reference:
- --depend=dependfile on page 2-8
- -o filename on page 2-19.
3.6 fromelf changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

Use of single letters as parameters to the --text option, either with a / or = as a separator is obsolete. The syntax --text/cd or --text=cd are no longer accepted. You have to specify -cd.

3.6.1 See also

Concepts

• General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2.

Reference

• --text on page 4-49.
3.7 C and C++ library changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes to the libraries have been made:

3.7.1 Support for non-standard C library math functions

Non-standard C library math functions are no longer supplied in math.h. They are still provided in the library itself. You can still request the header file from ARM if needed. Contact your supplier.

3.7.2 Remove __ENABLE_LEGACY_MATHLIB

In RVCT v2.2 changes were made to the behavior of some mathlib functions to bring them in-line with C99. If you relied on the old non-C99 behavior, you could revert the behavior by defining the following at compile time:

#define __ENABLE_LEGACY_MATHLIB

This has been removed in RVCT v4.0 for µVision.

3.7.3 See also

Concepts

- General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision on page 3-2.