Application Note 171

Migrating from ADS to RVDS 3.0

Released on: September, 2006

ARM®

DAI0171A

Application Note 171
Migrating from ADS to RVDS 3.0

Release Information

The following changes have been made to this Application Note.

Table 1 Change history

Date	Issue	Change
September 2006	A	First release

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

1 Introduction

1.1 Scope

The purpose of this document is to assist users of ARM Developer Suite (ADS) v1.x to migrate their development platforms to the latest RealView Development Suite (RVDS) 3.x. While this document has not been written with consideration for the RVDS 2.x toolchain, several of the changes here in may also apply to 2.x.

This document assumes familiarity with ARM tool syntax and other characteristics. If explanation of specific syntax is required, please consult the documentation provided with the appropriate toolchain.

Note

This document does not discuss usage of new features available in RVDS (such as linker feedback) except where such functionality would prevent an equivalent ADS project from building.

1.2 Changes to the tool structure

In RVDS, there is a single compiler executable, armcc, rather than separate components for C and C++ compilers, in ARM and Thumb configurations. To keep compatibility with earlier makefiles, the previous names are still recognized, but users are urged to change their build environments to the new naming structure. This is discussed in detail in section 3.2.1 of this document.

RealView Debugger (RVD) is the supported debugger, rather than AXD or `armsd`. RealView ICE (RVI) is the preferred JTAG debug control unit, rather than Multi-ICE. RealView Trace (RVT) replaces Multi-Trace as the preferred trace capture unit. Note that to use RVT you must also have RVI available.

1.3 Key features in RVDS 3.x

All users can benefit from the improved code size and performance that RVDS will offer over ADS.

Users developing for processors implementing ARM Architecture 6 or later will have to migrate to the newer toolchain, as these architecture versions are not supported by ADS.

Code generated by RVDS supports the ABI for the ARM Architecture (ABI), which allows users to share RVDS object code with users of other ABI-conforming tool-chains. This could allow users, for example, to develop Linux application code with RVDS and use these with their traditional GNU-based development environment. This topic is discussed in a separate Application Note (#150) available from the ARM website, which you can download from:

`http://www.arm.com/documentation/Application_Notes/index.html`

You can find documentation on the ABI from the ARM website:

`http://www.arm.com/products/DevTools/ABI.html`

Please be aware that the compiler/assembler option `--apcs/adsabi` is being removed - please see section 3.3 "Changes to linker usage" for further details.

1.4 Installation of multiple tools versions on a single machine

ADS allowed only a single installation on a single machine, regardless of version differences. RVDS does allow multiple versions to be installed, and these can reside on a machine with a single ADS installation (any version).

ARM provides a utility to switch between different versions of the development tools. You can find this "SuiteSwitcher" utility in the technical support downloads section of the ARM website:

`http://www.arm.com/support/downloads/`

2 Changes to source code

This section discusses what changes, if any, may be required to your source code in order to build it using RVDS 3.0 instead of ADS.

2.1 Changes to C and C++ source code

As a general rule, any source code conforming to ANSI C or ISO C++ standards will not need to be changed. ADS did not offer full ISO C++ support, and so it would be sensible to re-visit your ADS C++ sources to make use of features not previously supported. However this is not a requirement.

Some ARM tool specific features have changed or are obsolete, and these will need to be changed.

Inline Assembler (__asm)

The C compiler has a built-in inline assembler. However this is not being maintained for ARM Architecture 6 and later instruction sets. It is recommended that users migrate all inline assembler code to either embedded assembler, or to make use of compiler intrinsic functions where possible.

Note

Thumb inline assembly is no longer supported. Any such code from ADS will need to be modified.

Note

Direct reference to machine registers will result in a compilation error.

Table 2 Migrating inline assembly code to compiler intrinsic function

```
const int Q_Flag=0x08000000;
/* Q flag is bit 27 of PSR */
```

```
/* Inline Assembly Example */
int Clear_Q_flag (void) {
   int old_psr, new_psr, result;
    __asm {
        MRS old_psr, CPSR
        BIC new_psr, old_psr, #Q_Flag
        AND result,  old_psr, #Q_Flag
        MSR CPSR_f, new_psr
        }
}
        return result;
}
```

```
/* Embedded Assembly Example */
__asm int Clear_Q_flag (void){
MRS r0, CPSR
BIC r1, r0, #__cpp( Q_Flag )
AND r0, r0, #__cpp( Q_Flag )
MSR CPSR_f, r1
BX  lr
```

```
/* Inline Assembly Example */
__inline int current_sp(void)
{
int temp;
__asm {mov temp, sp}
return temp;
}
…
current_sp();
```

```
/* Compiler Intrinsic Example */
__current_sp();
```

Changes to library calls

ADS 1.x used optimized versions of the printf and scanf family of functions provided no floating point arguments were used. In RVDS 3.x, there are several variants of each function, optimized for specific argument types. If you have retargeted these functions, you may need to rewrite this retargeting code to support each variant. See the FAQ entry "*How do the ARM Compilers handle `printf`, `sprintf`, `fprintf`, and `scanf`*", in the technical support section of the ARM website.

Note

You can disable this optimization in RVCT if you prefer with the compiler option:
`--library_interface=aeabi_clib`

Some library functions which were named `__rt_*` have now been renamed `__aeabi_*` in accordance with the ABI. If you have retargeted these functions, you may need to change your function names. You will need to change source that re-implements such functions to support the ABI naming convention. Some examples (not an exhaustive list) of such functions are listed in Table 3.

Table 3 Examples of ABI renaming of library functions

ADS naming	RVDS Naming
`__rt_memcpy_w`	`__aeabi_memcpy4`
`__rt_div0`	`__aeabi_idiv0`
`__rt_sdiv`	`__aeabi_idiv`
`__rt_udiv`	`__aeabi_uidiv`
`__rt_errno_addr`	`__aeabi_errno_addr`
`__rt_fp_status_addr`	`__aeabi_fp_status_addr`

Changes to placement of data

In ADS, all initialized ("RW") data was placed in a RW section in the ELF output file. In RVDS, by default, such data will be placed in an RW section only if it is of size larger than 8 bytes, else it will be located in a `.bss` (ZI) section in the ELF output. This reduces the time taken at performing scatter-loading at start-up. But it may affect operation of your system if you have made assumptions about the placement of such data by the compiler.

Note

You can revert to the ADS behavior by using the compiler option:
`--bss_threshold=0`

Please see the 'Placement of small global ZI data (<= 8 bytes) in memory' FAQ in the technical support section of the ARM website.

The compiler has also changed its handling of data of type `volatile const`, which is typically used to define a read-only memory-mapped peripheral. RVCT places such data in RW or ZI sections (in accordance to the size of the data as above), as mandated by the ISO C standard. ADS placed such data in RO sections. Again this may affect your system operation if assumptions of placement have been made.

Changes to C++ code

There is no compatibility between ADS and RVDS C++ objects, and so all ADS C++ objects must be recompiled with the RVDS C++ compiler in order to link with other RVDS C++ object code.

RVCT provides full ISO C++ support, which ADS did not. As such, it is recommended to revisit all C++ code to investigate if it is possible to re-implement the functionality more efficiently with features that were not previously supported.

The C++ name mangling scheme has changed, and so any code that attempts to refer directly to a mangled C++ symbol name directly will need to be updated.

If you have initialization code that calls C++ specific library code, it may need to be changed to support the ABI-compliant naming. For example calls to `__cpp_initialize` should be changed to `__cpp_initialize_aeabi`.

2.2 Changes to Assembler source code

Well written ADS assembly code should still continue to assemble with RVDS. Note that one of the requirements of the ABI is that the stack must be 8-byte aligned at all public interfaces.

The RVDS assembler will properly set the build attributes of the generated object file if 8-byte alignment is preserved by this object. If it is not, such incompatibilities will result in an error being raised during linking. In this instance your source code must be changed. The solution is usually to push/pop an additional register upon function entry/exit.

Note

Pre-UAL (legacy) assembly code will continue to be supported. However the disassembly tools will output in Unified Assembler Language (UAL). For full details on the UAL changes, please see section 2.11 of the RVDS 3.0 Assembler Guide.

Table 4 Assembler source changes to support ABI and UAL

```
AREA ADSexample, CODE
    CODE32
    IMPORT foo
start
    MOV r0, #0x10
    MOV r1, #3
    BL  subroutine
stop
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x123456
subroutine
    STMFD r13!, {r4, lr}

    BL  foo
    LDMFD r13!, {r4, pc}
```

```
AREA RVDSexample, CODE
    ARM
    IMPORT foo
Start
    MOV r0, #0x10
    MOV r1, #3
    BL  subroutine
stop
    MOV r0, #0x18
    LDR r1, =0x20026
    SVC 0x123456
subroutine
    PUSH {r4, lr}  ; 8-byte stack
                   ; alignment
    BL  foo
    POP {r4, pc}
```

2.3 Changes to linker input files

As discussed in *Changes to C++ code*, there is no compatibility between ADS and RVDS generated C++ objects, and so the user must re-compile all ADS generated C++ objects if you wish to use them with RVDS built objects.

Scatter description files for ADS should still continue to work with RVDS, but note that if you refer explicitly to the C-library code that must be located in a root region, then the definitions of these regions have changed, as per the example in Table 5.

Table 5 Scatter description file changes

; ADS Example
LOAD <base_address> {
  ROOT <base_address> {
  __main.o (+RO)
  * (Region$$Table)
  * (ZISection$$Table)
  }
}
	; RVDS Example
	LOAD <base_address> {
	  ROOT <base_address> {
	  * (InRoot$$Sections)
	  }
	}

Note

Please see the 'Placing root region library objects in a scatter file' FAQ in the technical support section of the ARM website for further details.

2.4 Support for ARM Architecture 6 and later

If you wish to generate ARM Architecture 6 or later instructions from the compiler, you simply need to use the appropriate `--cpu` option to your build command. If using the assembler, you must re-write your assembly source code to make use of such instructions AND use the appropriate `--cpu` option.

When linking, if any object being linked in is built for support a later architecture, even if it does not make use of any instructions unique to that architecture, the linker will use library code supporting that architecture. This may result in undefined instruction exceptions on your target if it does not support the given architecture.

If this situation occurs you will need to inspect each object (including those from user defined libraries) to see if they are built for a later architecture. Note that the only way to set this attribute is by the `--cpu` option when the object is initially built.

3 Changes to makefiles

This section will discuss what command line options you will need to change when migrating to RVDS.

3.1 Changes that affect all build tools

Use of POSIX style switches

RVDS now uses a POSIX format, where all multi-character compiler options are specified with a double hyphen. The tools will warn when this is not adhered to. For example the ADS option `-cpu` should be changed to `--cpu`. Note that single character switches that are followed by a qualifying tag are still specified with a single hyphen. For example the optimization setting `-O2` (`-O<optimization_level>`) requires only a single hyphen with RVDS.

3.2 Changes to C and C++ compiler command options

Compiler instantiation

While not strictly necessary for the 3.0 release, which provides executable wrappers to maintain backwards compatibility, it is recommended that you no longer call the different compilers separately. You should change your commands as per Table 6.

Table 6 Compiler instantiation changes

ADS convention	RVDS convention	Implied RVDS defaults
`armcc`	`armcc`	`--c90 --arm`
`tcc`	`armcc --thumb`	`--c90`
`armcpp`	`armcc --cpp`	`--arm`
`tcpp`	`armcc --thumb --cpp`	

Changes to default behavior

The default optimization level for ADS was `-O0` if debug table generation (`-g`) was enabled, `-O2` if not. With RVDS the default optimization level is always `-O2`, though the user is warned about this change.

```
Warning: C3010W: -g defaults to -O2 if no optimisation level is specified
```

Obsolete and Deprecated Compiler Options

Please see Table E-2 in the RVDS 3.0 Compiler and Libraries Guide for a full description of earlier compiler options that are no longer supported, along with their RVDS equivalents, where possible.

3.3 Changes to linker usage

Note that the linker more strictly checks ABI compatibility of objects. including 8-byte stack alignment at function interfaces. Linking objects that do not maintain 8 byte stack alignment will result in an linker error of the form:

```
Error: L6238E: foo.o(.text) contains invalid call from '~PRES8' function to 'REQ8'
function
```

There is a compiler/assembler option `--apcs/adsabi` to force the RVDS compiler to generate objects compatible with the ADS procedure call standard, but this will not be supported after RVDS 3.0. ARM strongly recommends that all ADS code is rebuilt with RVDS. For a complete discussion on this topic, please see the FAQ entry "*Are legacy ADS objects/libraries compatible with RVCT ?*" from the ARM website.

Note

There is no compatibility between C++ objects built with ADS and RVDS.

3.4 Changes to IDE project files

Both ADS and RVDS are supplied with versions of Metrowerks' CodeWarrior IDE, and it is possible to import ADS projects into RVDS, but not vice versa. The user may need to manually fix up compiler options are per the above sections.

It is also possible to use RVDS with the Eclipse framework utility for project management and other functionality. See `http://www.eclipse.org` for information on Eclipse. See `http://www.arm.com/eclipse` for information on the integration ARM has done with this framework.

ARM provides a `mcp2make` utility to assist in porting CodeWarrior (`.mcp`) project files to makefiles to allow further importing to Eclipse and other build envionments. This is available from the downloads section of the ARM website:

`http://www.arm.com/support/downloads/`

3.5 Debug changes

AXD was the debugger provided with ADS, and for JTAG connectivity Multi-ICE was the recommended solution. If trace was required, Multi-Trace was the recommended solution. For RVDS, the recommended debugger is RVD, in association with RVI and RVT.

Both RVI and Multi-ICE (and RVT and Multi-Trace) require the same connection signals to the board, and so no changes should be required to the target hardware.

If you have AXD scripts that you used during your AXD debug sessions, a utility is provided within RVDS named `axd2rvd` to assist in the conversion of these scripts to the RVD format.

```
axd2rvd -I axd.txt -O rvd.txt
```

Alternatively you can refer to appendix C.3 'Comparison of RealView Debugger and AXD commands' or appendix D.1 'Comparison of RealView Debugger and `armsd` commands' of the RVD 3.0 User Guide.

Note

AXD and the command line debugger `armsd` are obsolete and will not be supported in future tools versions

4 References and further information

Additional information on the ARM tools can be found in the relevant RVCT 3.0 documentation:

RVCT 3.0 Compiler and Libraries Guide (ARM DUI 0205G)

RVCT 3.0 Linker and Utilities Guide (ARM DUI 0206G)

RVCT 3.0 Developer Guide (ARM DUI 0203G)

RVD 3.0 User Guide (ARM DUI 0153H)

RVD 3.0 Essentials Guide (ARM DUI 0181H)

The full documentation of the ABI for the ARM Architecture can be found at:

`http://www.arm.com/products/DevTools/ABI.html`

ARM Software Tools FAQs

`http://www.arm.com/support/devfaqsindex.html`

