

Building Linux applications using RVCT v4.0 and the GNU Tools and Libraries

Application Note 212



Building Linux applications using RVCT v4.0 and the GNU Tools and Libraries

Application Note 212

Copyright © 2008 ARM Limited. All rights reserved.

Release Information

Table 1 Change history

Date	Issue	Change
August 2008	A	First release

Proprietary Notice

Words and logos marked with or are registered trademarks or trademarks of ARM Limited 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 Limited 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.

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

<http://www.arm.com>

1 Introduction

This section provides important legal notices and support status information.

1.1 Legal notices and support status

Your attention is drawn to the following.

IMPORTANT LEGAL NOTE

ALL USE OF THE RVDS SOFTWARE IS GOVERNED BY THE TERMS OF THE RVDS END USER LICENSE AGREEMENT. NOTHING IN THIS DOCUMENT VARIES THIS LICENSE OR GRANTS YOU ANY ADDITIONAL RIGHTS. NOTHING IN THIS DOCUMENT CONSTITUTES LEGAL ADVICE. IF YOU HAVE ANY QUESTIONS PLEASE CONSULT YOUR LAWYER.

Disclaimer

Please note that should you choose to use the example code referred to elsewhere in this document in conjunction with your own or third party proprietary software and the GNU C library or any other open source code, you do so entirely at your own risk. ARM makes no representation or warranty as to the legal or business implications of such use. You should consult your own legal advisors if you have any concerns about this. Also, for the avoidance of any doubt, the example code is not open source software. Nothing in your license for this example code or your license (if any) for the ARM RealView Development Suite or, if applicable, RealView Compilation Tools or other ARM development tools products, allows you to distribute this example code, or the libraries or example code supplied by ARM with RVDS or RVCT, or any derivative or collective work created using such software, under a GNU Public License, or other open source license.

Support status

Please note that ARM does not provide support on the use of GNU tools or Linux. The information provided here is given for your reference only. Your supplier may be able to provide limited support for information in this document if you have a valid tools support contract with them. However, we suggest that in the first instance you discuss your issue in one of the various public forums, such as the comp.sys.arm newsgroup and the ARM website forums.

Alternatively you may prefer to contact CodeSourcery, who can provide paid support and assistance on the GNU tools or accept defect reports. Further information is available from CodeSourcery's GNU toolchain page at: http://www.codesourcery.com/gnu_toolchains/arm/

CodeSourcery also provide a mailing list for queries on the ARM GNU toolchain. Details of how to subscribe to this list can be found on the CodeSourcery website.

2 Scope of this document

This Application Note introduces you to building a Linux application or library, linked with the GNU C and C++ libraries, using the *RealView Compilation Tools* (RVCT) provided as part of *RealView Development Suite* (RVDS) v4.0. You can create dynamic images with RVCT v4.0 that can run under Linux using the GNU C library (glibc).

This Application Note covers the command-line options used to build a Linux-compatible executable, and describes how to use header files and libraries from glibc.

2.1 Expected use

This Application Note is intended to cover most expected use cases. It is specifically aimed at developing Linux applications and libraries in these situations:

- Building a standalone Linux application with RVCT.
- Building static and shared libraries with RVCT, and linking these to an application built with RVCT.
- Building a static or shared library with RVCT, and linking this to an application built with the GNU toolchain.
- Migrating an existing Linux application build using RVCT v3.1 to RVCT v4.0, retaining explicit search paths on the command line.
- Migrating an existing Linux application build using RVCT v3.1 to RVCT v4.0, using a standard configuration of system search paths and libraries.
- Using armcc and armlink as drop-in replacements for GCC and GNU ld using command-line translation.

————— Note —————

The Eclipse IDE supplied with RVDS v4.0 includes support for creating new ARM Linux applications (but not libraries) with `--arm_linux` and `--arm_linux_paths`.

Examples to demonstrate the interoperation between the GNU toolchain/libraries and RVCT for building applications and shared libraries to run on Linux are found in `install_directory\RVDS\Examples\4.0\...\platform\linux_apps`

2.2 Limitations

There are several limitations on interoperation between the GNU tools and libraries and RVCT:

The GNU binutils (including ld) from CodeSourcery's 2005-q3-2 release cannot consume RVCT v4.0 objects

For linking with the GNU C library, you should use the 2005-q3-2 release of the CodeSourcery tools (or a later release). Because of updates in the ARM *Application Binary Interface* (ABI) ELF specification, the binary utilities (binutils) from this CodeSourcery release cannot consume object files built by RVCT v4.0. Support for the new ELF ABI revision is in the 2006-q1 and later releases.

RVCT cannot be used for building the Linux kernel or kernel-based code, such as device drivers or other kernel modules

This is because a significant portion of the kernel code is written in assembly language using the *GNU assembler* (GAS) syntax. This is incompatible with *armasm*, and there is no performance gain to be made from rebuilding such code with a different assembler.

In addition, the function interfaces for the kernel code prior to version 2.6.16 have not been written to comply with the ABI. This means that drivers and other kernel modules cannot be compiled using RVCT because there are no guarantees that calls would be made correctly between the kernel and the driver code. You must use the GNU toolchain when building the kernel and kernel modules.

ARM architecture v4T is not fully supported

See *Target requirements* on page 6.

C++ exceptions are only supported with CodeSourcery's 2007-q1-10 or later releases

Due to slight implementation differences in the way C++ exceptions are handled between RVCT and GCC, the GNU C/C++ library prior to the CodeSourcery 2007-q1-10 release did not support code generated by RVCT that used C++ exceptions. Therefore to use C++ exceptions you must use the CodeSourcery 2007-q1-10 release or later. This includes using these libraries on your target's filesystem.

If you choose to link with the RVCT exception handling code, this only supports exceptions within a statically linked image. To throw exceptions between applications and shared libraries you must use the unwinding code provided in the GCC support libraries.

You must take care when using `alloca()`

The `alloca()` function is implemented in a compiler-specific way. In RVCT, this is implemented as calls to library functions that allocate storage on the heap using `malloc()`. For use with ARM Linux, a special version of these functions is provided with the RVCT libraries in `arm_linux/armlinux_*`, that makes use of thread-local storage (TLS) to ensure that `alloca()` is safe to use in the presence of multiple threads. This library is automatically added to the system configuration file, and is therefore passed to the linker in GNU emulation mode or when `--arm_linux_paths` is used.

————— Note —————

This implementation of `alloca()` does not interact with the `glibc` `setjmp()` and `longjmp()` functions. Using the RVCT version of `alloca()` in the presence of `setjmp()` and `longjmp()` from `glibc` might lead to memory leaks.

GCC inline assembly code is not compatible with RVCT and vice versa

RVCT and GCC use different syntax for inline assembly. RVCT cannot compile GCC syntax and vice versa. The recommended solution is to conditionally use alternative copies of your inline assembly code with the appropriate syntax for each toolchain.

GAS assembly files are not compatible with the RVCT assembler and vice versa

Assembly language files cannot be built by both *armasm* and the GNU assembler because they use different syntax.

Some GCC language extensions are not supported

Some of the language extensions supported by GCC are not provided by armcc. In particular, using nested functions is not supported.

2.3 Requirements

This Application Note assumes that you are familiar with RVDS, GNU tools and Linux.

Target requirements

Please note that the instructions in this document relate to building Linux applications for ARM architecture v5TE (ARMv5TE) or later targets, such as the ARM926EJ-S or ARM1176JZ-S. This is because the ARM ABI uses ARMv5TE as its base architecture, and earlier architecture versions are not fully covered by the ABI.

You might be able to use these instructions to build Linux applications for ARM architecture v4T (ARMv4T) cores (such as the ARM720T and ARM920T) with RVCT. This is, however, entirely at your own risk and is not supported. In particular, you are not able to use Thumb code built for ARMv4T in shared libraries. It is recommended that you only use the GNU toolchain when building Linux applications for ARMv4T targets.

Your target's filesystem must contain the ABI-compliant library binaries. These are included in the CodeSourcery GNU toolchain releases described in *Build requirements*. Finally, the target must be running a Linux kernel with support for the *Native POSIX Threading Library* (NPTL), which is the more recent mechanism for supporting multithreaded code under Linux with the GNU C library, and TLS. See *About the ARM Application Binary Interface* on page 7 for more details on the ABI requirements on your target system.

For the mainstream kernel source, this means that your target must be running version 2.6.12 (or later) of the Linux kernel. Your Linux distribution, however, may have applied the appropriate patches to its release of an earlier kernel. For more details, you must contact your Linux distributor.

Prebuilt binary images of the Linux kernel configured for the ARM development boards can be found on the ARM website at <http://www.arm.com/products/os/linux.html>

Build requirements

All information in this document relates to the use of RVCT v4.0.

Note that CodeSourcery's 2005-q1 release was the first to allow *Embedded Application Binary Interface* (EABI) compliant interoperability between RVCT and the GNU toolchain. Several enhancements and fixes, however, have been made since then, and the instructions in this document relate only to the CodeSourcery 2006-q1-6 and later releases, because it is now simpler and safer to link with a newer release.

At the time of writing, the CodeSourcery binary and source packages for the GNU toolchains are found at http://www.codesourcery.com/gnu_toolchains/arm/

Your ARM Linux distribution might already use the CodeSourcery toolchain or have the appropriate patches applied. For more details, you must contact your ARM Linux distributor.

2.4 About the ARM Application Binary Interface

The ABI for the ARM Architecture is a collection of standards, some open and some specific to the ARM architecture. The standards regulate the interoperation of binary code, development tools, and a spectrum of ARM core-based execution environments from bare metal to platform operating systems such as ARM Linux.

A third-party toolchain such as the GNU tools must comply with the standards given in the ABI for its objects to link and interoperate correctly with those produced by RVCT. The CodeSourcery release of the GNU tools is specifically tailored to fully support the ARM ABI and allow objects produced using both RVCT and the GNU tools to work together successfully.

For more details of the ARM ABI, including the full ABI documents, see the ARM website at <http://www.arm.com/products/DevTools/ABI.html>

Interactions between mixed-ABI components

If you are not using an ABI-compliant kernel, you might need to build a mixed ABI system. Kernels before version 2.6.16 can only be built using the legacy GNU ABI (use GCC option `-mabi=apcs-gnu` when using the CodeSourcery toolchain). This includes all kernel modules and device drivers.

This can cause problems when your applications or libraries must interface directly with kernel structures or functions (system calls), including through the use of a shared header file describing kernel structures. In this case, you must use assembly code or modified descriptions of the structures to translate between the two ABIs when calling kernel functions or manipulating kernel data structures in your applications or libraries.

From kernel 2.6.16 onwards, the Linux kernel can be built using the new ARM EABI. This allows for much simpler integration of applications and libraries to form a completely EABI-compliant system.

3 Using RVCT to build a Linux application or library

There are several possible routes to producing a Linux application or library with RVCT, depending on the requirements of your build.

In most cases, you have to configure the tools based on an existing GNU toolchain or by providing an alternative location for system header files and libraries. *Configuring RVCT for Linux applications* describes how to configure the tools. Once the tools are configured, you can use RVCT in one of these ways:

- Use the tools directly to produce an application using the standard configuration, but with normal RVCT command-line options. See *Building for ARM Linux using normal RVCT options* on page 9.
- Use RVCT as a drop-in replacement for GCC and the GNU linker. See *Using RVCT as a drop-in replacement for GCC and GNU ld* on page 10.
- Migrate a build from an earlier version of RVCT to use the new features in RVCT v4.0. See *Migrating a build from an earlier version of RVCT* on page 11.

3.1 Configuring RVCT for Linux applications

When building for ARM Linux, it is necessary to use paths to the appropriate system header files and libraries (for example, glibc and libstdc++) together with any appropriate standard options and object files (such as those containing the application entry point function and C library initialization code) for your library configuration. In RVCT v4.0, this information can be obtained automatically from an existing GNU toolchain, or specified manually. This can be a complicated procedure, and the paths and options used remain the same unless you modify your libraries. Consequently, this configuration information is obtained once and stored in a configuration file that you specify on the command line. The information in this configuration file is then re-used when compiling or linking for ARM Linux.

RVCT v4.0 can be configured manually, by specifying particular paths that are used for finding header files and libraries. Alternatively, if you have an existing GNU toolchain, RVCT can configure itself automatically by querying the GNU tools for the paths that they use.

The configuration produced by this process is written to a configuration file used later when building for ARM Linux. You must specify this location by using the `--arm_linux_config_file=`*path* option, where *path* gives the filename of the configuration file. You must specify this both when producing the configuration file and when using the configuration during compilation or linking.

Configuring RVCT automatically

If GCC is in a directory listed in the PATH environment variable, you can configure the tools using the command:

```
armcc --arm_linux_configure --arm_linux_config_file=
```

path

If GCC is not on your system path, you can specify this explicitly:

```
armcc --arm_linux_configure --arm_linux_config_file=
```

config_file_path

```
--configure_gcc=
```

path_to_gcc

where *path_to_gcc* is the path and filename of the GCC driver binary, that is, the actual gcc executable (with .exe suffix on Windows). For a cross-compiler the filename is, for example, arm-none-linux-gnueabi-gcc (with .exe suffix on Windows).

During configuration, the compiler also determines the location of the GNU linker used by GCC and queries that for additional information. If this cannot be determined, or you wish to override the normal path to the GNU linker, you can specify this using the `--configure_gld=path_to_gld` option, where *path_to_gld* is the complete path and filename of the GNU ld binary.

You can also override the sysroot path or the location of the C++ header files, and specify additional search paths for header files and libraries. See *Configuring RVCT manually* for details of these options.

Configuring RVCT manually

To configure the tools manually, you must specify:

- the sysroot path
- the path to the C++ header files.

The sysroot path is the root of the tree into which header files and libraries are normally installed. If you are configuring against a CodeSourcery distribution or another, self-contained cross-compilation GNU toolchain this is typically the root of the directory tree into which glibc was installed. For recent CodeSourcery releases, this is the `arm-none-linux-gnueabi/libc` subdirectory. If you are configuring against the target's own filesystem (for example, to pick up new libraries as they are built and installed into the target filesystem tree) the sysroot is the root of this filesystem.

The C++ header file path is the path of the directory containing the header files from libstdc++. In a CodeSourcery distribution, this is typically the `arm-none-linux-gnueabi/include/c++/version` subdirectory, where *version* is the GCC version.

To configure the tools manually, use:

```
armcc --arm_linux_configure --configure_sysroot=sysroot_path
--configure_cpp_headers=headers_path --arm_linux_config_file=filename
```

You can also specify additional header search paths and library search paths as a comma-separated list using the `--configure_extra_includes=list` and `--configure_extra_libraries=list` options.

To manually configure against a CodeSourcery distribution, you must provide extra library paths for the GCC support libraries, since these are not packaged in the glibc sysroot. For example, you can use a command similar to the following:

```
armcc --arm_linux_configure --arm_linux_config_file=filename
--configure_sysroot=<codesourcery_root>/arm-none-linux-gnueabi/libc
--configure_cpp_headers=<codesourcery_root>/arm-none-linux-gnueabi/include/c++/gcc_version
--configure_extra_libraries=<codesourcery_root>/lib/gcc/arm-none-linux-gnueabi/gcc_version,
<codesourcery_root>/arm-none-linux-gnueabi/lib
```

3.2 Building for ARM Linux using normal RVCT options

Once the tools are configured, you can use this configuration to build code for ARM Linux using the `--arm_linux_paths` option. This is a compiler option only; this follows the typical GCC usage model where the compiler driver is used to direct linking and selection of standard system object files and libraries. You must also specify the location of the configuration file with `--arm_linux_config_file=filename`. Using these options, you can build application code directly, for example to build “hello world”:

```
armcc --arm_linux_paths --arm_linux_config_file=filename -o hello hello.c
```

See the example code in *install_directory*\RVDS\Examples\4.0\...\windows\linux_apps\hello for more details.

To create a shared library, compile and link your code using `--apcs=/fpic --shared`. The compiler provides the `--shared` option to select variants of the system object files and libraries from the configuration that are suitable for linking into a shared library.

For example, to compile a source file `source.c` suitable for use in a shared library:

```
armcc --arm_linux_paths --arm_linux_config_file=filename --apcs=/fpic -c source.c
```

To link two object files `obj1.o` and `obj2.o` into a shared library `libexample.so`:

```
armcc --arm_linux_paths --arm_linux_config_file=filename --shared -o libexample.so  
obj1.o obj2.o source.o
```

Note

When linking a C++ application with `--arm_linux_paths`, you must specify the `--cpp` option to the compiler driver so that it passes the appropriate C++ libraries to the linker.

3.3 Using RVCT as a drop-in replacement for GCC and GNU ld

This section describes the use of RVCT as a replacement for GCC and GNU ld.

Overview of the GCC emulation mode

RVCT v4.0 supports a GCC emulation mode, where `armcc` accepts command lines intended for GCC and GNU ld and translates these internally into standard `armcc` and `armlink` command lines. This follows the typical GCC usage model of using the compiler driver to direct linking, rather than invoking the linker directly. However, `armcc` does provide support for being invoked as if emulating GNU ld directly, and reports itself as the linker if invoked in GCC emulation mode with `--print-prog-name=ld`. This is primarily intended to support a limited number of cases where the linker is invoked directly by existing build scripts targeting the GNU tools, for example in a partial link step.

Using GCC emulation mode

To enable emulation of GCC, invoke `armcc` with one of the following options:

- `--translate_gcc` to emulate gcc
- `--translate_g++` to emulate g++
- `--translate_gld` to emulate GNU ld

You must also provide `--arm_linux_config_file=filename` to give a location for the configuration file.

Note

If you do not provide a configuration file with the `--arm_linux_config_file` option when in translation mode, the compiler performs translation of options but does not set any defaults for ARM Linux, including ABI defaults such as enum size. This mode of operation is provided for convenience and is not intended for building Linux applications.

Passing normal armcc options in GNU emulation mode

In order to take advantage of features specific to RVCT, you can pass normal RVCT options to the compiler when in GCC emulation mode. To do this, use `-Wrvct,option,...`. This is a fake GCC-like option that accepts a comma-separated list of armcc options. These options are passed verbatim to the compiler, and are appended to the translated command line so that they can override any translation options.

Differences in behavior and limitations

There are some differences in behavior between GCC and the emulation mode supported by RVCT v4.0.

- If no optimization level is specified, the armcc default (`-O2 -Ospace`) is used rather than the GCC default (`-O0`). If a GCC numeric optimization level (`-O0` through `-O3`) is used, this is translated into `-On -Otime` for armcc. The GCC `-Os` option translates as `-O3 -Ospace`.
- Support for diagnostic control is limited. In particular, warnings are suppressed by default (similar to GCC) and are re-enabled with `-W` or `-Wall`. The `-w` option (lower case `-w`) is supported to suppress warnings, for example to override a `-Wall` earlier on the command line. Other GCC `-W...` options are ignored; if you require control of individual messages then you can use the normal RVCT options (`-Wrvct,--diag_suppress,`
`-Wrvct,--diag_error,` and so on.)
- Many GCC options do not have an equivalent in armcc. These include, for example, many of the `-f...` GCC options that control optimization phases that are specific to the GCC code generator, and are not applicable to RVCT. Any GCC options that do not have an equivalent in armcc are silently ignored.

3.4 Migrating a build from an earlier version of RVCT

Existing code for ARM Linux that builds successfully using RVCT v3.0 or RVCT v3.1 can work without changes. You can, however, take advantage of the new features in RVCT v4.0 to simplify your makefiles or other build scripts.

Minimal migration path without using a configuration file

The compiler and linker both provide a `--arm_linux` option. This does not require a configuration file, and enables a set of default configuration options, for example ABI-variant options such as `--enum_is_int`. This permits you to simplify the compiler options used in existing makefiles while retaining full and explicit control over the header and library search paths used. When migrating a build from an earlier version of RVCT, you can remove these standard switches from the list of those supplied to the compiler and linker with the single `--arm_linux` switch.

The `--arm_linux` option in the compiler enables the following switches:

```
--gnu --enum_is_int --wchar32 --library_interface=aeabi_glibc
--no_hide_all --apcs=/interwork --preinclude=linux_rvct.h
```

The `--arm_linux` option in the linker enables the following switches:

```
--sysv --no_startup --no_ref_cpp_init --no_scanlib --keep=*(.init) --keep=*(.fini)
--keep=*(.init_array) --keep=*(.fini_array) --linux_abitag=2.6.12
--diag_suppress=6332,6318,6319,6765,6747,6420
```

For more information on the above options, see Application Note 178, *Building Linux Applications using RVDS 3.1 and the GNU Tools and Libraries*, which is available from the ARM website at <http://infocenter.arm.com/>

Migration using a configuration

If you wish to take advantage of the configuration capabilities in RVCT v4.0, you can create a configuration file as described in *Configuring RVCT for Linux applications* on page 8. Once this configuration file is created, you can modify an existing build by replacing the list of standard options and search paths with the `--arm_linux_paths` option. See *Building for ARM Linux using normal RVCT options* on page 9 for more details on how to use the `--arm_linux_paths` option.

General notes for migrating builds

In RVCT v4.0, you no longer need to link with the helper libraries, for example `h_5.1`. If you are recompiling an entire project from source, the required functions are generated in the object files by the compiler. If, however, you are linking with legacy object files compiled using a previous version of RVCT, you must still link with an appropriate helper library.

3.5 Assembler command-line options

When using assembly code in your application or library, two switches must typically be given to the assembler:

`--apcs/interwork`

This instructs the assembler to set the build attributes in the object file to indicate that the code is ARM/Thumb interworking-safe.

`--no_hide_all`

This indicates that the assembler must use dynamic import and export for all global symbols.

3.6 Additional headers from RVCT

Some of the standard RVCT headers must be used when building for ARM Linux. These headers define some implementation-specific macros that are dependent on the compiler rather than the C library used. The files are provided in the `arm_linux` subdirectory of the RVCT v4.0 header files, and this directory should be given before the GNU C library include directories in the path list. If the `RVCT40INC` environment variable is set, then this path is automatically used by the `--arm_linux` and `--arm_linux_paths` options and by GCC emulation mode.

An additional header file, `linux_rvct.h`, is also provided. This defines a number of macros for compatibility with GCC and the Linux environment. This is automatically included (equivalent to using `--preinclude=linux_rvct.h`) when using `--arm_linux` or `--arm_linux_paths`. When using RVCT to emulate GCC, these macros are defined internally in the compiler to permit preprocessing of files other than C or C++ source without automatically including the file.

If you would like to use the DSP or NEON intrinsics available in RVCT v4.0, these are also provided in the `arm_linux` subdirectory for convenience, for example `#include <arm_neon.h>`. Note that both `dspfn.h` and `math.h` include a definition `round()`, therefore you must rename one definition if you want to use both versions of these functions. For example:

```
#define round dsp_round
#include <dspfn.h>
#undef round
```

3.7 Creating and using shared libraries

In a Linux system, you might often want to create a dynamic shared library that can be linked with a variety of applications. This section describes methods of building and using shared libraries.

Building a shared library with RVCT

This section provides details on compiler and linker options for building a shared library.

Compiler options

When building dynamic shared libraries, all of the library code must be compiled and linked to be position-independent. To do this, use the `--apcs/fpic` compiler switch. In GCC emulation mode, use `-shared -fpic`.

Linker options

`armlink` supports the creation of dynamic shared libraries; however this requires some additional options.

`--shared`

This instructs the linker to create a dynamic shared library and not a static library.

`--soname <name>`

This specifies the shared object name (SONAME) for the library.

`--fpic`

This enables you to link position-independent code (compiled with `--apcs/fpic`).

For example, to link `libfunc.o` and `asmfunc.o` into a dynamic shared library `libdynamic.so`, you can use the following linker command line:

```
armlink --arm_linux --fpic --shared --soname libdynamic.so -o libdynamic.so libfunc.o
asmfunc.o libc.so.6
```

When using GCC emulation mode, if `-shared` is passed to the compiler driver this automatically passes `--shared --fpic` to `armlink`. You can still specify the shared object name or other options, for example using `-Wl, -soname, libexample.so`.

Using shared libraries in your application

Shared libraries can be used with `armlink` in the same way as normal libraries by specifying them on the linker command line. References to the shared library are added to the image and resolved to the library by the dynamic loader at runtime.

Library search order

The order in which references are resolved to libraries is the order in which libraries are specified on the command line. This is also the order in which the dependencies are resolved by the dynamic linker. You can specify the runtime location of libraries using the `--rpath` linker option.

Unlike GNU `ld`, `armlink` repeatedly searches libraries in command-line order until either all references are resolved or no further references can be resolved by the given libraries. That is, `armlink` behaves similarly to:

```
ld --start-group lib1.a lib2.a lib3.a ... --end-group
```

Selection of static and dynamic libraries

In RVCT v4.0, armlink supports a `--library=name` option similar to the `-l` option in GNU ld. This can search for libraries named as `libname.so` or `libname.a` depending on whether dynamic library searching is enabled at that point on the command line. The searching of dynamic libraries is controlled by the `--[no_]search_dynamic_libraries` option, as shown in the last two lines of the example given below. These two command lines would perform a link searching for `libfoo.so` before `libfoo.a`, but only searching for `libbar.a`:

```
gcc -shared -fPIC -Wl,-Bdynamic -lfoo -Wl,-Bstatic -lbar
```

```
armcc --arm_linux -L--shared -L--fpic \  
-L--search_dynamic_libraries -L--library=foo \  
-L--no_search_dynamic_libraries -L--library=bar
```

4 Frequently-asked questions and troubleshooting

This section provides answers to common questions as well as additional information.

4.1 Frequently-asked questions

Here is a list of potential questions that you might want to ask.

Where can I find further information?

The recommended starting point for further information is the CodeSourcery toolchain FAQ at http://www.codesourcery.com/gnu_toolchains/arm/faq.html

You may also wish to look at ARM and Linux forums and newsgroups, or at mailing list archives. <http://www.arm.linux.org.uk/> and the ARM Linux wiki <http://www.linux-arm.org/> provide resources relating specifically to ARM Embedded Linux.

Note

ARM does not provide support on the use of the GNU tools. For more information, see <http://gcc.gnu.org>.

How do I build an EABI-compliant Linux kernel?

Prior to kernel version 2.6.16 an EABI-compliant kernel could not be built. This is only, however, an issue for applications and libraries which directly access kernel structures or functions because the EABI-compliant GNU C library translates calls appropriately from EABI-compliant applications to the non-EABI compliant kernel system calls.

From kernel version 2.6.16, it is possible to build an EABI kernel, however you must still use the GNU toolchain.

Can I build the Linux kernel using RVCT?

The Linux kernel has a large amount of assembly code that is written in GNU assembler syntax. The RVCT assembler does not support the GAS syntax and therefore cannot be used to build the Linux kernel.

Also, because the most critical parts of the kernel are written in assembly and not C, you are unlikely to see a significant improvement if RVCT was used to build the kernel.

Which kernel version should I use?

The CodeSourcery toolchain as provided in binary form is built to use NPTL and it expects to have TLS support in the kernel. Recent CodeSourcery binary releases have a dependency on kernel version 2.6.16 or later, so you might have to use kernel version 2.6.16 or later. Alternatively, your Linux distributor may have already applied the appropriate patches to their kernel build. You should contact your Linux distributor for more information.

Can I use EABI-compliant and non-EABI-compliant applications together?

Yes. You should place the libraries and the dynamic linker in a different directory to the normal libraries. We recommend that you use `/libeabi` for the EABI-compliant libraries, and leave the original, non-EABI compliant libraries in `/lib`.

You must then set the library search path for EABI applications using the environment variable `LD_LIBRARY_PATH=/libeabi` or by using the `--rpath` linker option. You are recommended to rebuild all applications to use the EABI in your final system because the extra libraries take up a significant amount of space in the filesystem.

The GNU tools report “ERROR: Source object ... has EABI version 5, but target ... has EABI version 4” when used on objects generated by RVCT v4.0

RVCT v4.0 generates ELF files conforming to revision 5 of the ARM ABI ELF (AAELF) specification. However, CodeSourcery’s 2005-q3-2 release only supports revision 4 of the AAELF specification, and does not consume objects produced by RVCT v4.0 tools. Support for the new ABI revision is included in the 2006-q1-3 and later releases of the CodeSourcery toolchain.

The GNU linker reports “ld: ERROR: ... : Conflicting definitions of wchar_t”, or armlink reports: “Error: L6242E: Cannot link object dummy.o as its attributes are incompatible with the image attributes ... wchar_t-16 clashes with wchar_t-32”

This is because the linker has detected a mismatch between the `wchar_t` types used. The CodeSourcery document *ARM GNU/Linux Application Binary Interface Supplement* for building Linux applications specifies that `wchar_t` must be 32 bits.

A similar error exists for incompatible sizes of enumeration types. For ARM Linux, an enum must be 32 bits wide.

To resolve these errors, ensure that all of your code is compiled for 32-bit `wchar_t` and 32-bit enums, for example using the `--wchar32` and `--enum_is_int` armcc options. This is done automatically if `--arm_linux` is used.

Alternatively, armlink supports the options `--no_strict_wchar_size` and `--no_strict_enum_size` that avoid these errors. Be aware, however, that binary compatibility might be broken between the objects with differing attributes if they pass data of enum or `wchar_t` types between each other and this might lead to runtime failures.

armlink reports “Fatal error: L6033U: Symbol in crt1.o is defined relative to an invalid section”

In the 2006-q1-3 release of the CodeSourcery toolchain the `crt1.o` object file has not been correctly stripped. This has been fixed in the 2006-q1-6 CodeSourcery release. Alternatively you can strip the `crt1.o` object yourself.

armlink reports “Error: L6449E: While processing ../libgcc.a: Symbol #7 in symbol table section #10 is defined in section #17 but the last section is #11”

The `libgcc.a` contained in some CodeSourcery distributions contains object files with invalid symbols in their symbol table. These symbols are never used or needed, but armlink generates an error when trying to read these object files. There are three alternative solutions:

- Under GCC emulation mode, invoke the link step using `-shared-libgcc`. This forces the use of the shared library version of `libgcc`.
- Strip your copy of the `libgcc.a` member object files to remove the corrupt symbols from the symbol table.
- Upgrade to a newer CodeSourcery release. This issue is fixed in the 2008-q1 release.

Using hardware VFP instructions

ARM Linux uses software floating-point linkage, where floating-point arguments are passed in integer registers even if functions themselves perform operations in hardware VFP registers. To use hardware VFP instructions within functions, compile your code with, for example, `--fpu=SoftVFP+VFP` to select software floating-point linkage.

Can I use the RVCT libraries in a Linux application?

In general, you are not recommended to use the RVCT libraries when building a Linux application. The libraries provided with RVCT are targeted at standalone applications running directly on the target hardware, that is, without an OS. They contain semihosting calls and memory handling that is not suitable for use under an operating system like Linux. It is sometimes possible to use small, self-contained portions of the RVCT library code. However, you must take care to retarget any semihosted I/O functions and signal handling. Also, the RVCT libraries can only be statically linked into an application or shared library.

How can I see which libraries are being used?

The linker provides an option `--info=libraries` that lists the libraries it uses. For information on which library functions are being used, you can request verbose output from the linker with `--verbose` and redirect this to a file with `--list=filename.txt`.

When using `--arm_linux_paths` or the GCC emulation mode, the configuration file provides the list of system paths and standard libraries with which to link. This file is in XML format, and you can examine this file in a text editor to check the libraries that are used by the tools.

How can I have greater control over which libraries are linked into my application?

If you require explicit control over the libraries that are linked with your application, this can be done with a manual link step by passing the `--arm_linux` linker option. The `--arm_linux` option sets the `--no_scanlib` option, which disables searching of system library paths. You are then free to provide your own list of search paths with `--userlibpath`, and a list of libraries to use.

4.2 Common problems with running your application

Some common problems with running your application are described in Table 2.

Table 2 Common problems with running applications

Problem	Solution
Cannot find the application	<ul style="list-style-type: none"> Check that the application is on the path, or you are running it with <code>./program</code> in the current directory. The dynamic loader may not be the same as specified at link time. In this case, use <code>/path-to-linker/dynamic-loader program-path/program</code>. For example: <code>/libeabi/ld-linux.so.3 /opt/bin/eabi/hello</code> <p>———— Note ————</p> <p>You can specify an alternative dynamic loader for an application by passing, for example, <code>--dynamiclinker=/libeabi/ld-linux.so.3</code></p>
Permission denied	Check that you have set the executable flag for the program (use <code>chmod +x program</code>).

Table 2 Common problems with running applications (continued)

Problem	Solution
“GLIBC_2.4 not found” error “unable to find library XXX.so.X”	This is the dynamic linker reporting that it cannot use the libraries found on its default path. You can use the LD_LIBRARY_PATH environment variable to access the correct libraries. For example: LD_LIBRARY_PATH=/libeabi ./helloworld Alternatively, you can use the --rpath linker option.
“Illegal instruction” error before main()	This indicates that the image has been built for the incorrect architecture (for example, ARMv6 code running on an ARMv5TE core), or the kernel has been built without NPTL support. Check that you have built the image for the correct ARM architecture and check that you are using either a 2.6.12 (or later) Linux kernel or one with the appropriate patches applied as part of your distribution. Also ensure that the system call interface matches between the Linux kernel and the EABI C library you are using. That is, an old-ABI kernel uses the old system call interface and the C library might have been built to use the new system call interface. Note that the binary libraries from recent CodeSourcery releases are built for the new system call interface. Once at main() this is likely to be an actual undefined instruction in the application.
Various dynamic linker errors	The pre-built libraries supplied in the 2006-q1-3 and later releases of the CodeSourcery toolchain use the new system call interface and have been built with an ABI tag that require Linux kernel 2.6.16 or later. The dynamic linker generates various error messages when run on older kernel revisions. To avoid this you must update to the 2.6.16 or later kernel which supports the new system call interface or rebuild the libraries to support an older kernel revision. The 2.6.16 or later kernel can be configured to support both the new and old system call interface.

Segmentation faults

There are a variety of possible causes of segmentation faults. They might be caused by problems with your application. You must also ensure that:

- When you are using a manual link step, --no_scanlib or --arm_linux are passed to the linker. This ensures that the linker does not search the RVCT libraries and accidentally link in semihosted I/O functions. If you are explicitly linking with any portions of the RVCT libraries, ensure that any semihosted I/O and signal handling functions are retargeted appropriately.
- When creating a dynamic library, you have compiled and linked as position-independent code (use --apcs/fpic for the compiler and --fpic for the linker).
- When creating an application using a manual link, you have used either the two linker switches --no_startup and --entry _start, or the linker switch --arm_linux.
- When you are using C++ exceptions you must be linking with libraries from an appropriate CodeSourcery release (2007-q1-10 or later) and using these libraries on your target filesystem.

4.3 Image sizes and stripping debug data

Both the GNU and ARM toolchains add a significant amount of information to an image that is generally only of use for debugging.

For production systems, it is likely that you want to strip the debugging data from your applications and shared libraries. With RVCT, this can be removed using the --no_debug switch at the link stage or by running fromelf on the linked image. In addition, you can use fromelf to remove the .comment sections and symbols from the file. For example:

```
fromelf --strip debug,comment,symbols --elf -o stripped.axf image.axf
```

In addition, the data sizes in RVCT images can be slightly larger than those in GNU images. This is typically because some ZI data (BSS) is moved into the RW data area for performance reasons on bare-metal systems. You can move this data to ZI sections using the compiler switch `--bss_threshold=0`. For more details, see `--bss_threshold=num` in the RVCT v4.0 *Compiler Reference Guide*.

5 Further reading

This section lists publications by ARM and by third parties.

5.1 ARM publications

The full documentation of the ABI for the ARM Architecture can be found at <http://www.arm.com/products/DevTools/ABI.html>

The ARM GNU/Linux ABI Supplement can also be found at http://www.codesourcery.com/gnu_toolchains/arm/arm_gnu_linux_abi.pdf

Prebuilt kernel, binaries, and an example filesystem image for the ARM development boards are available from the ARM website at <http://www.arm.com/linux/>

Additional information on the ARM tools can be found in the relevant RVCT v4.0 documentation:

- *RVCT v4.0 Compiler Reference Guide*
- *RVCT v4.0 Compiler User Guide*
- *RVCT v4.0 Developer Guide*
- *RVCT v4.0 Libraries and Floating Point Support Guide*
- *RVCT v4.0 Linker Reference Guide*
- *RVCT v4.0 Linker User Guide*
- *RVCT v4.0 Utilities Guide.*

See also Application Note 201, *Building and Debugging ARM Linux Using ARM Embedded Linux, ARM RealView Development Suite 3.1 and RealView ICE 3.2*, which is available from the ARM website at <http://infocenter.arm.com/>

General information on ARM Linux can be found from the open-source community. Useful starting points are:

- The comp.sys.arm newsgroup
- The ARM Linux project website: <http://www.arm.linux.org.uk/>
- The ARM Linux wiki: <http://www.linux-arm.org/>

5.2 Other information

The following items might also be useful to you:

- For further information on the GNU toolchain supplied by CodeSourcery, see http://www.codesourcery.com/gnu_toolchains/arm/
- In particular, the FAQ for the ARM GNU toolchain can be found at http://www.codesourcery.com/gnu_toolchains/arm/faq.html