



Arm Development Studio Tutorial

Revision: NA

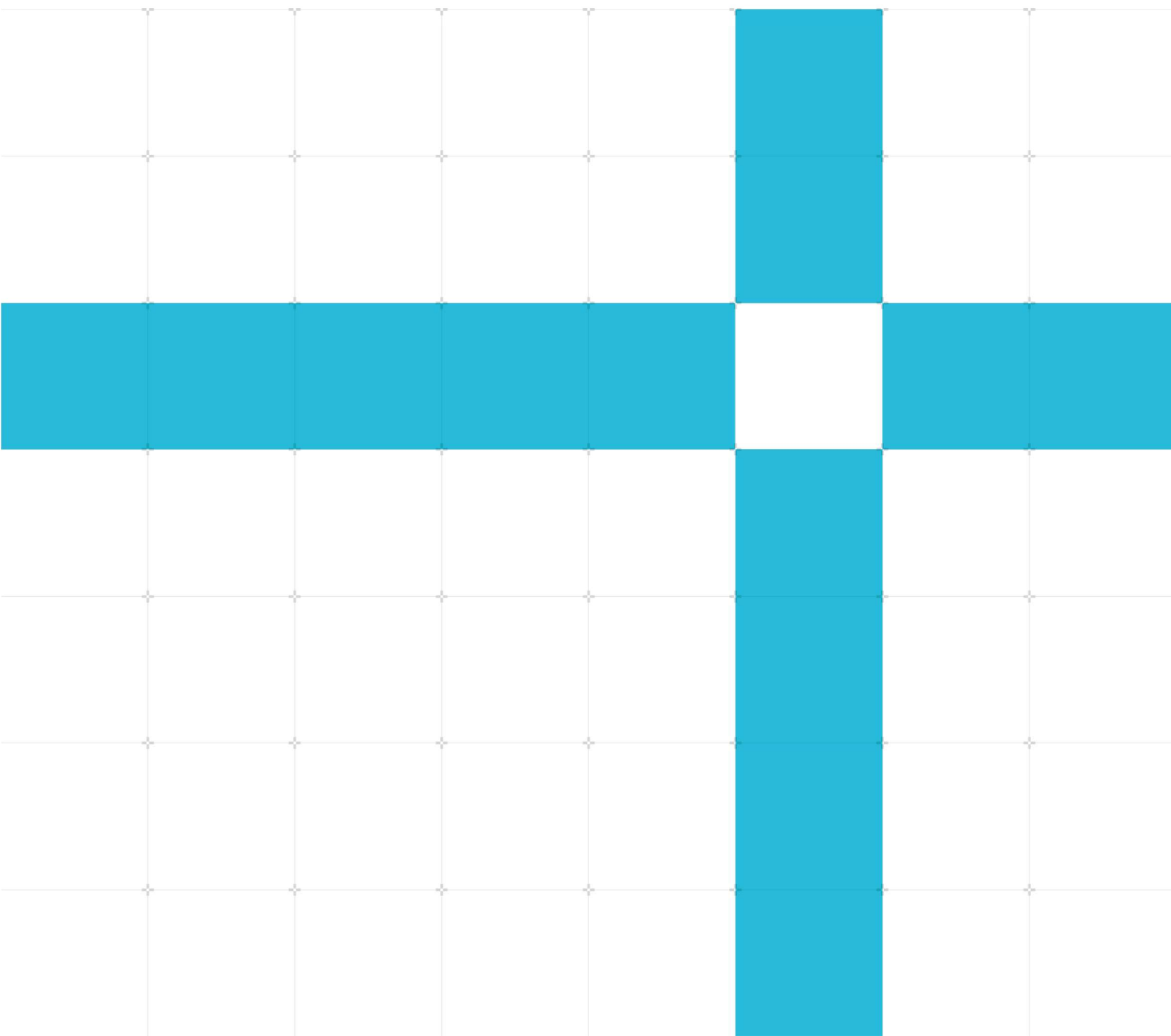
Targeting processors, floating-point units, and NEON with Arm DS

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 0.0

NA



Arm Development Studio Tutorial

Targeting processors, floating-point units, and NEON with Arm DS

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0.0	5 th March 2021	Non-Confidential	First version

Confidential Proprietary Notice

This document is **CONFIDENTIAL** and any use by you is subject to the terms of the agreement between you and Arm or the terms of the agreement between you and the party authorized by Arm to disclose this document to you.

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information: **(i)** for the purposes of determining whether implementations infringe any third party patents; **(ii)** for developing technology or products which avoid any of Arm's intellectual property; or **(iii)** as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or **(iv)** for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arm technology described in this document with any other products created by you or a third party, without obtaining Arm's prior written consent.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Web Address

<http://www.arm.com>

Contents

1 Tutorial summary	5
2 Introduction	6
3 Selecting the target processor	7
4 Selecting the target FPU.....	11
5 Enabling NEON automatic vectorization	13
6 Further reading	16

1 Tutorial summary

Arm Development Studio (Arm DS) tutorial for selecting specific processors with Arm Compiler 6 to maximize performance, selecting Floating Point Unit (FPU) and enabling NEON.

2 Introduction

This tutorial assumes you have installed and licensed [Arm Development Studio](#). For more information, see [Arm® Development Studio Getting Started Guide](#). The content in this tutorial applies to non-Makefile projects.

3 Selecting the target processor

The [Arm Compiler 6](#) lets you target either an architecture or a specific processor when generating code:

- Specifying an architecture provides the greatest code compatibility. The generated code can run on any processor supporting that architecture.
- Specifying a particular processor provides optimum performance. The compiler can use processor-specific features such as instruction scheduling to generate optimized code for that specific processor.

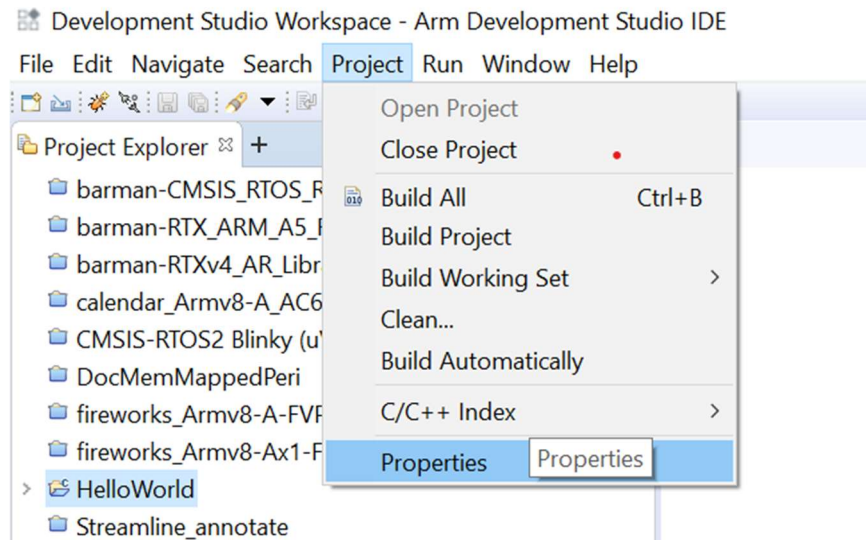
We consider the following [armclang](#) command-line options in this section:

- The [--target](#) command-line option is mandatory and allows you to specify the target triple. The target triple has the form *architecture-vendor-OS-abi*, for example `aarch64-arm-none-eabi`. The available triples are limited by the compiler version (see [supported triples](#)).
- Use the [--march](#) option to generate code for a specific architecture (for example `armv8-a`). The supported architectures vary according to the selected target. To see a list of all the supported architectures for the selected target, use `-march=list`.
- Use the [--mcpu](#) option to generate code for a specific processor (for example `cortex-a53`). The supported processors vary according to the selected target. To see a list of all the supported processors for the selected target, use `-mcpu=list`.

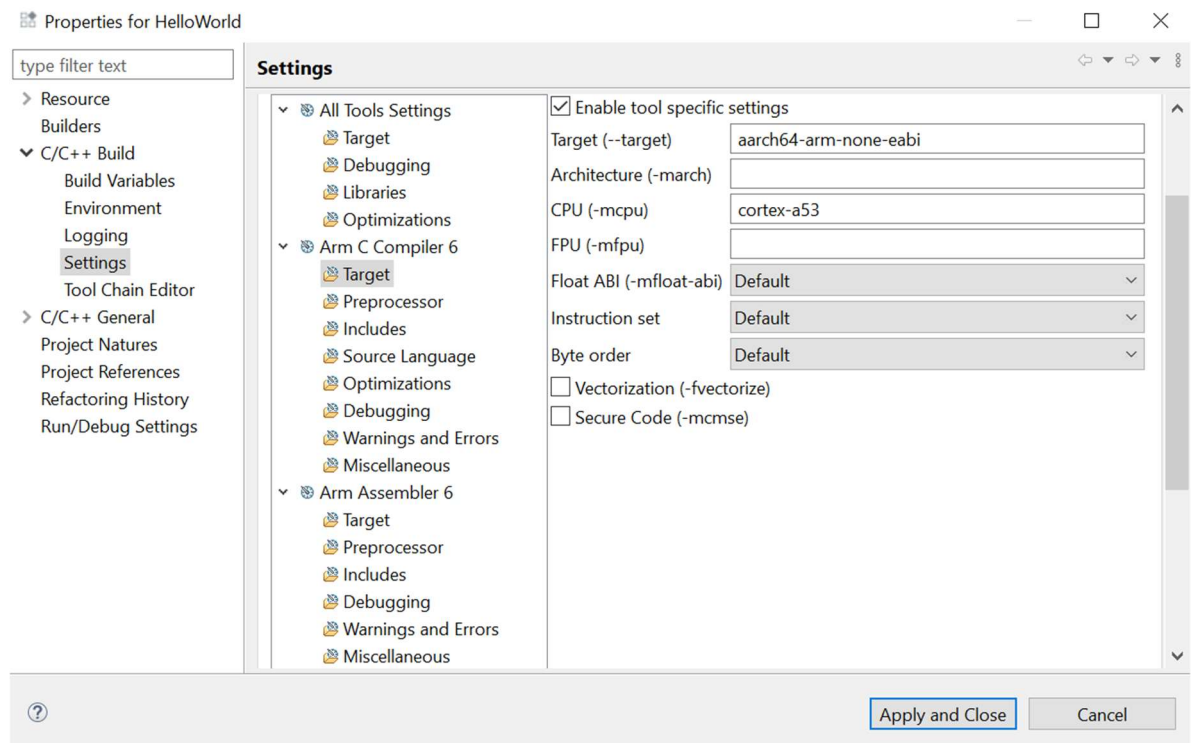
You must avoid specifying both the architecture (`-march`) and the processor (`-mcpu`) because it can cause a conflict. The compiler infers the correct architecture from the processor. For example, `-mcpu=cortex-a53` infers `-march=armv8-a`. We recommend you understand the [mandatory armclang options](#) to set these options correctly.

To configure the `--target`, `-march` and `-mcpu` options in Arm DS:

1. Select your project in the **Project Explorer** view.
2. To display the **Properties** dialog box, select **Project > Properties** from the main menu. You can also right-click on your project in the **Project Explorer** view to select **Properties**.



3. Expand **C/C++ Build**, then **Settings** in the **Properties** dialog box.
4. On the **Tool Settings** tab, select **Arm C Compiler 6 > Target** to display the code generation settings.
5. Select **Enable tool specific settings**.
6. Enter a value for **Target (--target)**. In this tutorial, we specify `--target=aarch64-arm-none-eabi` to generate A64 instructions for AArch64 state.
7. Enter a value for **Architecture (-march)** or **CPU (-mcpu)**. Remember to only include a value for one of these two fields. In the following example, we set **CPU (-mcpu)** to `cortex-a53` to build for a Cortex-A53 processor.



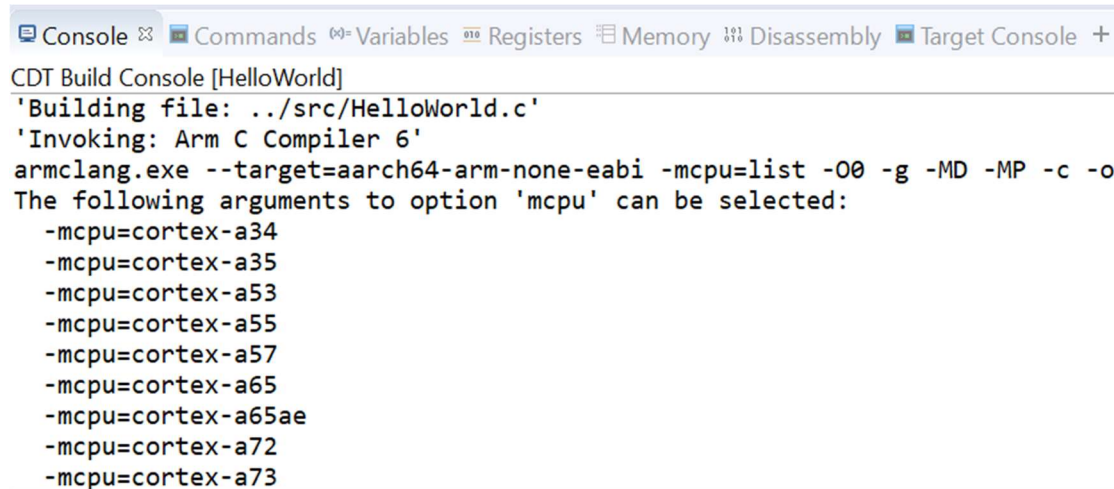
8. Click **Apply and Close** to save the settings.

You can see a list of all supported architectures by specifying `list` for the **Architecture (-march)** setting, then building your project. The console (**Window > Show View > Console**) shows the list of architecture names.

```

CDT Build Console [HelloWorld]
'Building file: ../src/HelloWorld.c'
'Invoking: Arm C Compiler 6'
armclang.exe --target=aarch64-arm-none-eabi -march=list -O0 -g -MD -MP -c -o
The following arguments to option 'march' can be selected:
-march=armv8-a
-march=armv8-r
-march=armv8.1-a
-march=armv8.2-a
-march=armv8.3-a
-march=armv8.4-a
-march=armv8.5-a
-march=armv8.6-a
-march=armv8.7-a
    
```

You can also see a list of all supported processors by specifying `list` for the **CPU (-mcpu)** setting, then building your project.



```
CDT Build Console [HelloWorld]
'Building file: ../src/HelloWorld.c'
'Invoking: Arm C Compiler 6'
armclang.exe --target=aarch64-arm-none-eabi -mcpu=list -O0 -g -MD -MP -c -o
The following arguments to option 'mcpu' can be selected:
-mcpu=cortex-a34
-mcpu=cortex-a35
-mcpu=cortex-a53
-mcpu=cortex-a55
-mcpu=cortex-a57
-mcpu=cortex-a65
-mcpu=cortex-a65ae
-mcpu=cortex-a72
-mcpu=cortex-a73
```

If the compiled program is to run on a specific Arm architecture-based processor, select the target processor. For example, to compile code to run on a Cortex-A53 processor use the **CPU** (**-mcpu**) setting `cortex-a53`.

Alternatively, if the compiled program is to run on different Arm processors, choose the lowest common denominator architecture appropriate for the application. Use the **Architecture** (**-march**) setting to set the architecture. For example, if you want your program to compile for Cortex-A53 and Cortex-A57 processors, you must use the `-march=armv8-a` option.

4 Selecting the target FPU

Each target architecture has a default floating-point unit (FPU) option. However, you can use the `--mfpu` option to specify a target FPU architecture and override the default option.

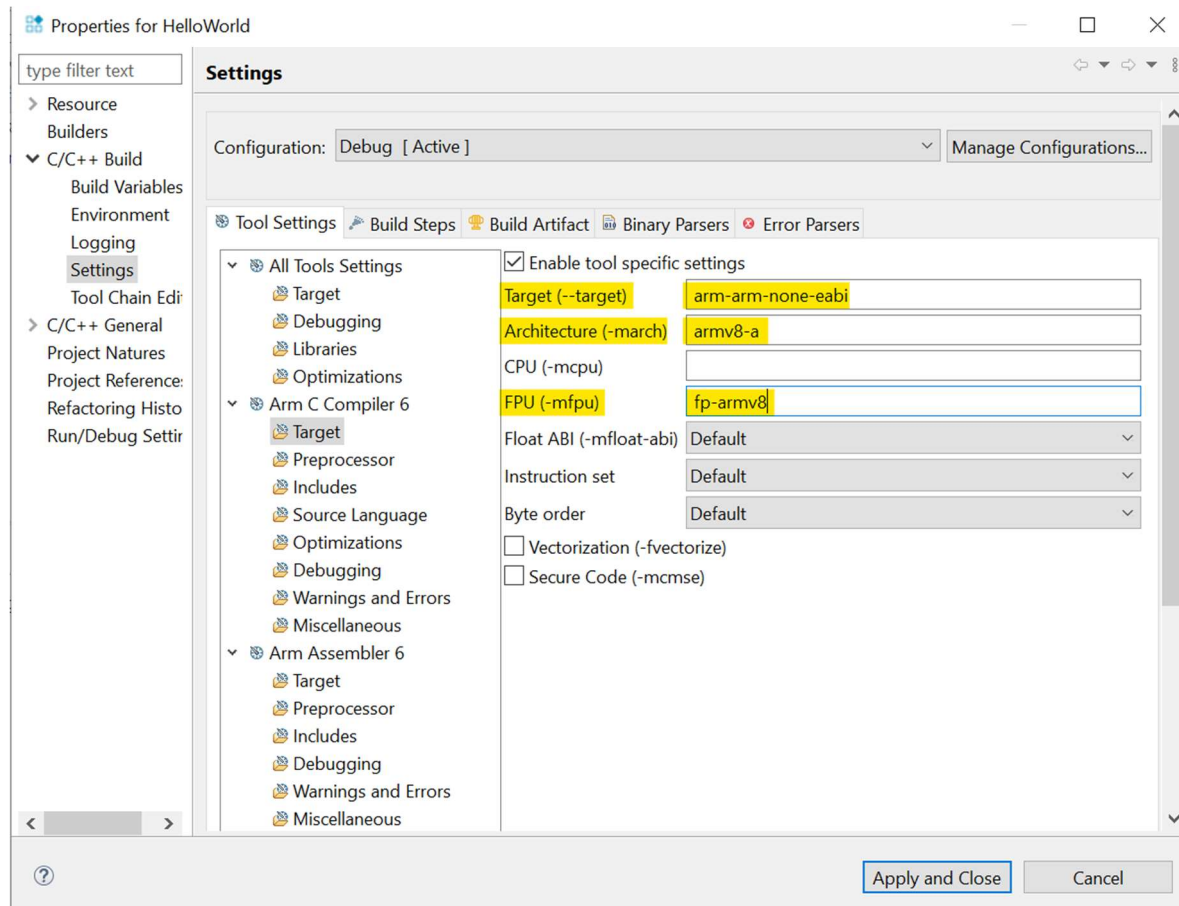
Note: For AArch64 targets, the `-mfpu` option is ignored with AArch64 targets. In this case, you must use the `-mcpu` option to override the default FPU for `aarch64-arm-none-eabi` targets. For example, to prevent the use of floating-point instructions or floating-point registers for the `aarch64-arm-none-eabi` target use the `-mcpu=name+nofp` option.

There are no software floating-point libraries for targets in AArch64 state. When linking for targets in AArch64 state, `arm1ink` uses AArch64 libraries that contain Advanced SIMD and floating-point instructions and registers. The use of the AArch64 libraries applies even if you compile the source with `-mcpu=name+nofp+nosimd` to prevent the compiler from using Advanced SIMD and floating-point instructions and registers. Therefore, there is no guarantee that the linked image for targets in AArch64 state is entirely free of Advanced SIMD and floating-point instructions and registers.

You can prevent the use of Advanced SIMD and floating-point instructions and registers in images that are linked for targets in AArch64 state. Either re-implement the library functions or create your own library that does not use Advanced SIMD and floating-point instructions and registers.

To configure the `-mfpu` option in Arm DS, use the **FPU (-mfpu)** setting. This setting is in the same location on the **Properties** dialog box as the **Target (-target)** setting. You can find the location of this setting in the *Selecting the target processor* section in this tutorial.

For example, to generate A32 and T32 instructions for AArch32 state, we specify the option `--target=arm-arm-none-eabi`. To select the Armv8 application architecture profile we set the option `-march=armv8-a`. Then, to enable the Armv8 Floating-point Extension and disable the Cryptographic Extension and the Advanced SIMD extension, we set `-mfpu=fp-armv8`.



Set the value `list` to the `-mfpu` option to view a list of all the supported FPU architectures. Then build your project. The console shows the list of FPU architectures.

5 Enabling NEON automatic vectorization

Arm NEON technology is the implementation of the Advanced SIMD architecture extension. It is a 64-bit and 128-bit hybrid SIMD technology targeted at advanced media and signal processing applications and embedded processors.

Specific NEON instructions let you use the NEON unit to perform operations in parallel on multiple lanes of data.

There are various methods of creating code that use NEON instructions:

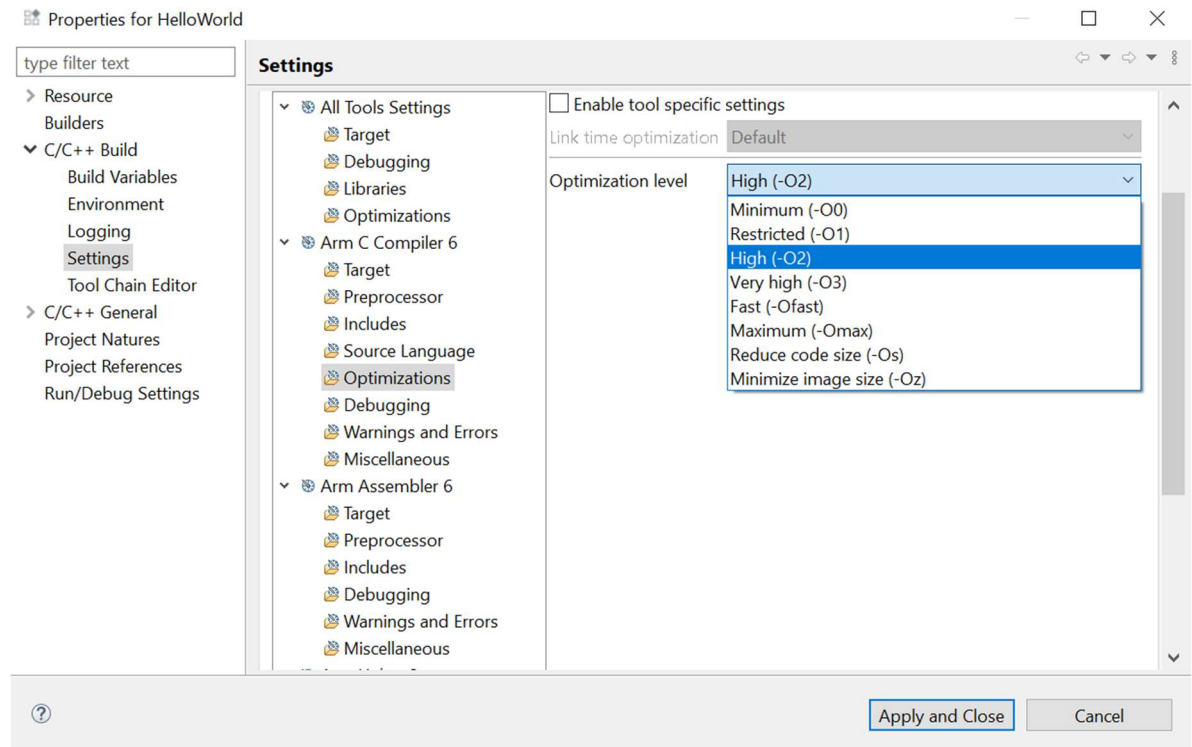
- Write assembly language, or use embedded assembly language in C, and use the NEON instructions directly.
- Write in C or C++ using the NEON intrinsics.
- Call a library routine that has been optimized to use NEON instructions.
- Have the compiler use automatic vectorization to optimize loops for NEON.

For additional information, you can visit the [Introducing Neon for Armv8-A](#) guide.

To enable automatic vectorization, you must target a processor that has a NEON unit.

You must set an optimization level `-O1` or higher to enable the generation of Advanced SIMD instructions directly from C or C++ code. To configure the optimization level in Arm DS:

1. Select **Project > Properties** and expand **C/C++ Build**, then **Settings** in the **Properties** dialog box.
2. On the **Tool Settings** tab, select **Arm C Compiler 6 > Optimizations** to display the code generation settings.
3. Select the **Optimization level** you require.



4. Click **Apply and Close** to save the settings.

The **--fvectorize** option allows you to enable the generation of Advanced SIMD instructions directly from C or C++ code at optimization levels **-O1** and higher. Depending on the optimization level, the steps to enable automatic vectorization are different:

Level **-O2** or higher:

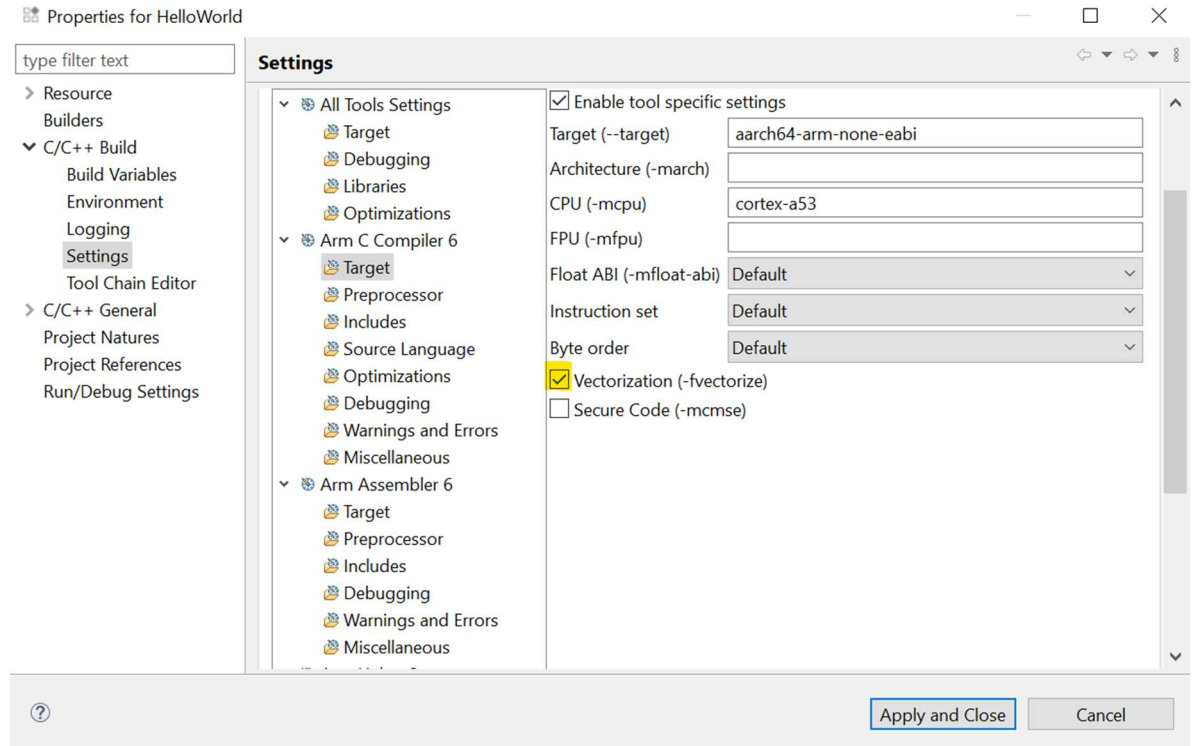
-fvectorize option is set by default when building for a NEON-capable processor.

Level **-O1**:

To set **-fvectorize**:

1. Select your project in the **Project Explorer** view.
2. To display the **Properties** dialog box, select **Project > Properties** from the main menu. You can also right-click on your project in the **Project Explorer** view to select **Properties**.
3. Expand **C/C++ Build**, then **Settings** in the **Properties** dialog box.
4. On the **Tool Settings** tab, select **Arm C Compiler 6 > Target** to display the code generation settings.
5. Select **Enable tool specific settings**.

6. Click the **Vectorization (-fvectorize)** box to enable.



7. Click **Apply and Close** to save the settings.

6 Further reading

- [Arm Compiler Reference Guide](#)
- [Compiling for NEON with Auto-Vectorization](#)