



Migrating Projects from Arm Compiler 5 to Arm Compiler 6

Version 1.0

Non-Confidential

Copyright © 2019 Arm Limited (or its affiliates).
All rights reserved.

Issue 02

102682_0100_02_en



Migrating Projects from Arm Compiler 5 to Arm Compiler 6

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	11 April 2019	Non-Confidential	First release

Proprietary Notice

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 for the purposes of determining whether implementations infringe any third party patents.

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, 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.

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 affiliates) 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 <https://www.arm.com/company/policies/trademarks>.

Copyright © 2019 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview.....	6
2. The Fireworks project.....	7
3. Importing the Fireworks project example into the DS-5 Eclipse workspace.....	8
4. Building the Fireworks example.....	11
5. Running your application.....	12
6. Clean and rebuild the project.....	15
7. Modifying the makefile.....	16
8. Specifying which compiler to use.....	17
9. Specifying which compiler options to use.....	18
10. Modifying your code.....	19
11. Running the modified application.....	21
12. Converting assembly code.....	22

1. Overview

Follow the steps in this tutorial to migrate an existing Arm Compiler 5 bare-metal project to an Arm Compiler 6 bare-metal project. Once you have completed the migration, you can rebuild your executable and run it on a Fixed Virtual Platform (FVP) model provided with DS-5.

2. The Fireworks project

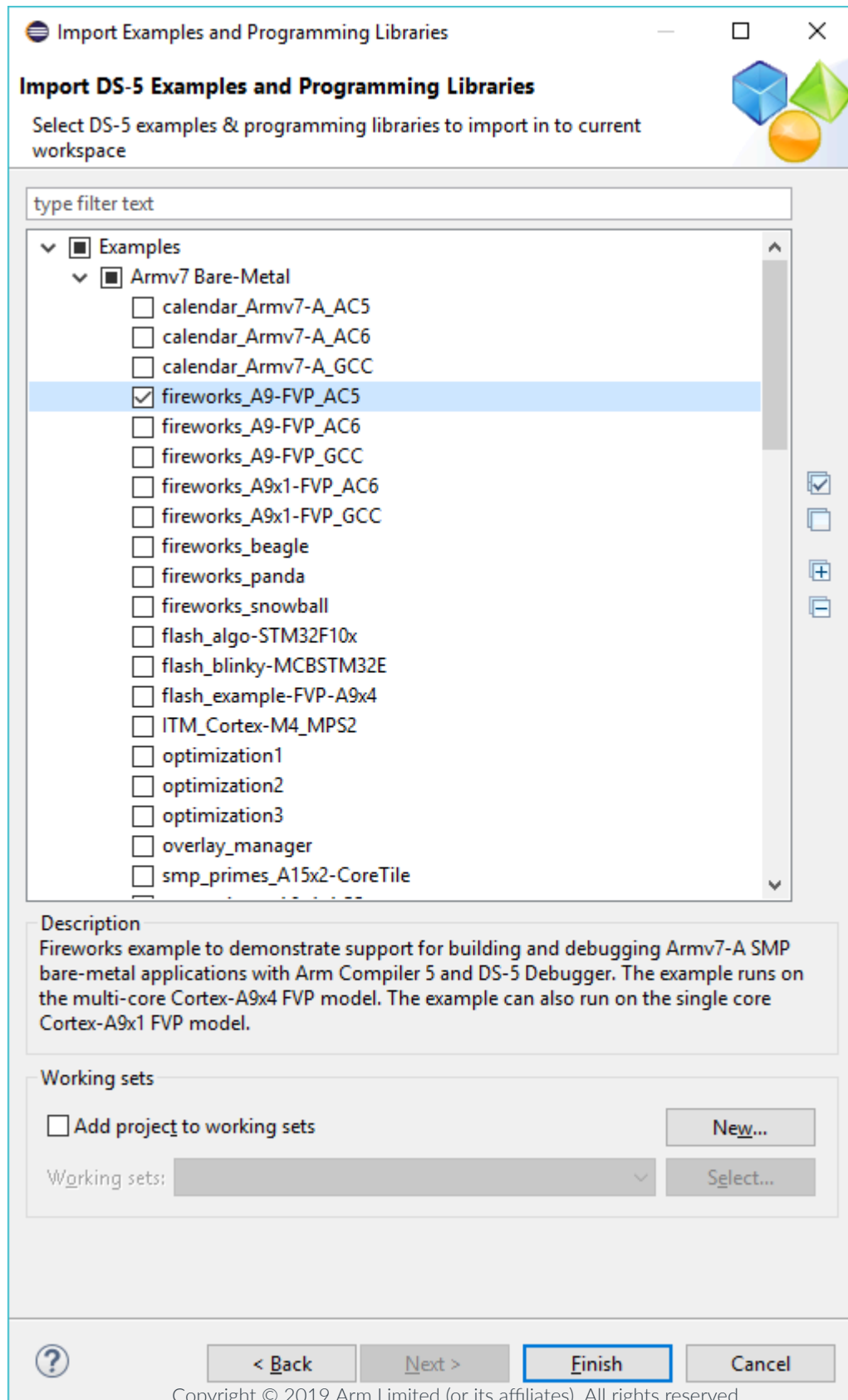
To show you the steps to migrate your project, this tutorial uses the Fireworks example project built with Arm Compiler 5. This example is included in DS-5. The steps in this tutorial can be easily adapted to work on your individual projects.

We also provide the Fireworks example built with Arm Compiler 6, but for the purposes of this tutorial, we are ignoring it.

3. Importing the Fireworks project example into the DS-5 Eclipse workspace

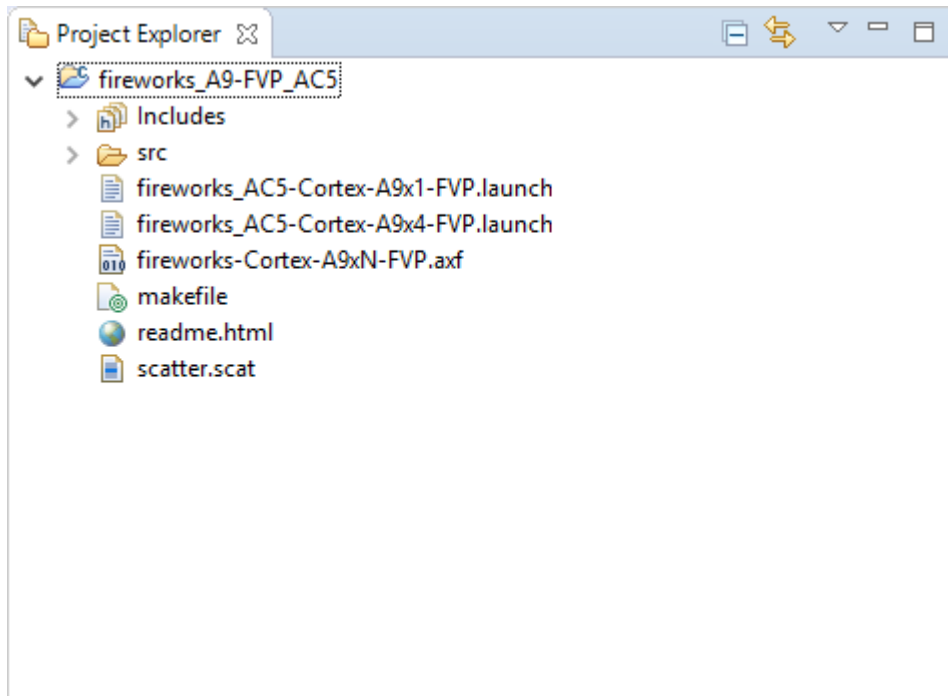
To import the Fireworks project example into the DS-5 Eclipse workspace, do the following:

1. Launch Eclipse for DS-5 from the Start Menu.
2. Select a workspace for your DS-5 projects. The default workspace is fine.
3. Close the Welcome screen, if it appears.
4. Select File > Import.... to open the Import Selectio dialog.
5. Expand the DS-5 group and select Examples and Programming Libraries. Click Next.
6. Expand the Examples group, then expand the Armv7 Bare-Metal group.
7. Several Armv7-based bare-metal examples are provided, including startup code and simple applications for a variety of Cortex-A, -R and M-family platforms. For this tutorial, select fireworks_A9-FVP_AC5.

Figure 3-1: Importing the Fireworks project example into the DS-5 Eclipse workspace

8. Click Finish. The Project Explorer view populates with the project. You can view the imported example in the Project Explorer view.

Figure 3-2: Fireworks project example in Project Explorer



4. Building the Fireworks example

To import the Fireworks project example into the DS-5 Eclipse workspace, do the following:

The Fireworks project we just imported is pre-built, that is, the executable is already available in the project. But it is worthwhile rebuilding the project to understand how to start the project build process and to ensure DS-5 is setup correctly.


Depending on your version of DS-5, the imported project may attempt to auto-generate makefiles instead of using the provided one which we will be modifying in this tutorial. To fix this, make sure that generate makefiles automatically is unticked and then remove /Debug from the end of the Build directory path.

You can find these settings by right clicking on the imported example project, then Properties > C/C++ Build.

This will allow DS-5 to use the provided makefile and the makefile to write into the obj directory in the project root directory.

To clean the project, right-click the fireworks project and select Clean Project to remove the contents of the obj directory, if it already exists.

Then, to build the project, you can either:

- Right-click the project and select Build Project, or
- On the main menu, click  button to start the build process. That is it, you have now built your project!

5. Running your application

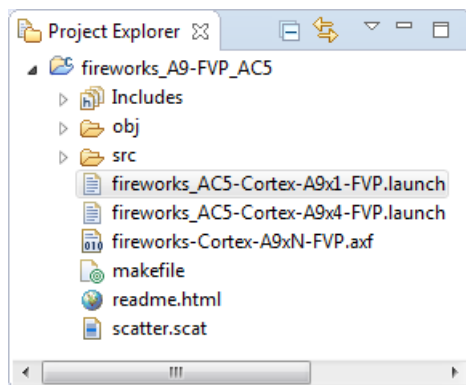
To run the application you just built, you can use the Cortex-A9 FVP models provided with DS-5.



You can use FVP models to run your image when you do not have access to hardware, for example when the design is not yet completed, or where hardware prototype boards are scarce. Your software development can start early with FVPs which reduces the time to market for your product.

The Fireworks project contains `.launch` files:

Figure 5-1: .launch files



- fireworks-AC5-Cortex-A9x1-FVP.launch - Simulates a single-core Cortex-A9 system.
- fireworks_AC5-Cortex-A9x4-FVP.launch - Simulates a quad-core Cortex-A9 system.

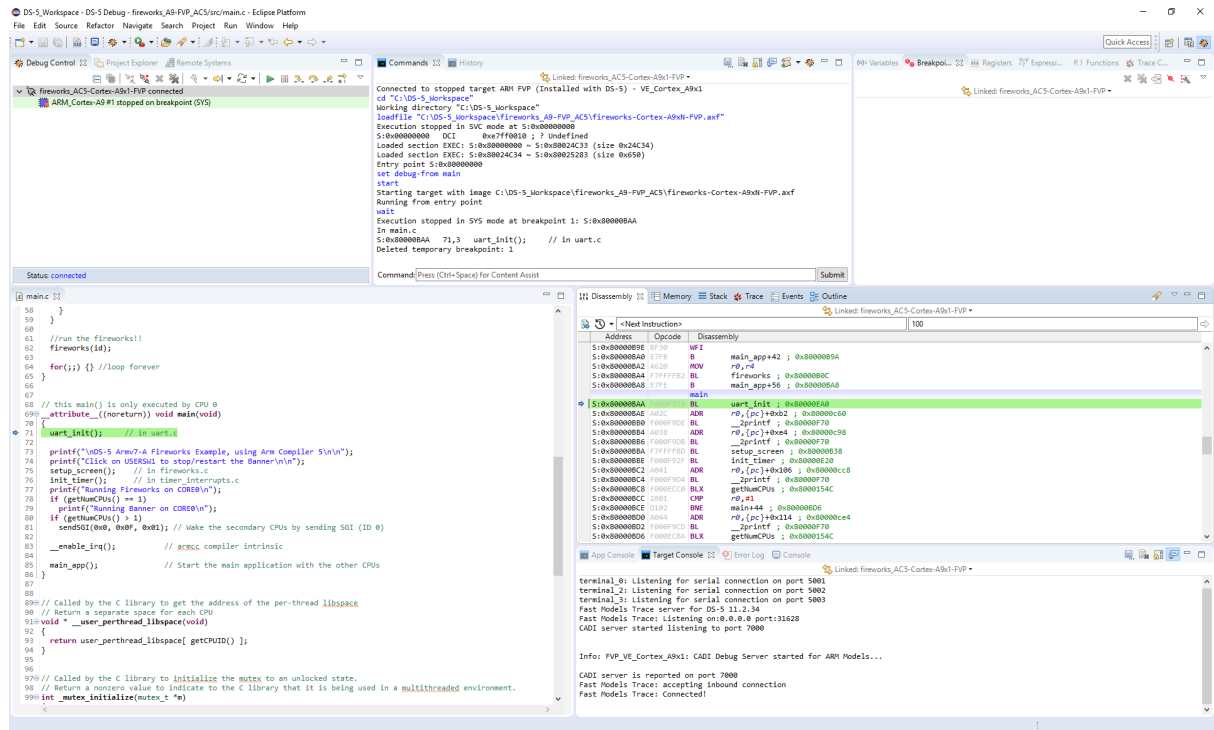


A `.launch` file instructs DS-5 Debugger what target to connect to, what files to download, which breakpoints to set, and so forth. Launch files are created from the Debug Configurations dialog. We provide them with our examples as an easy method to get you up and running out of the box.

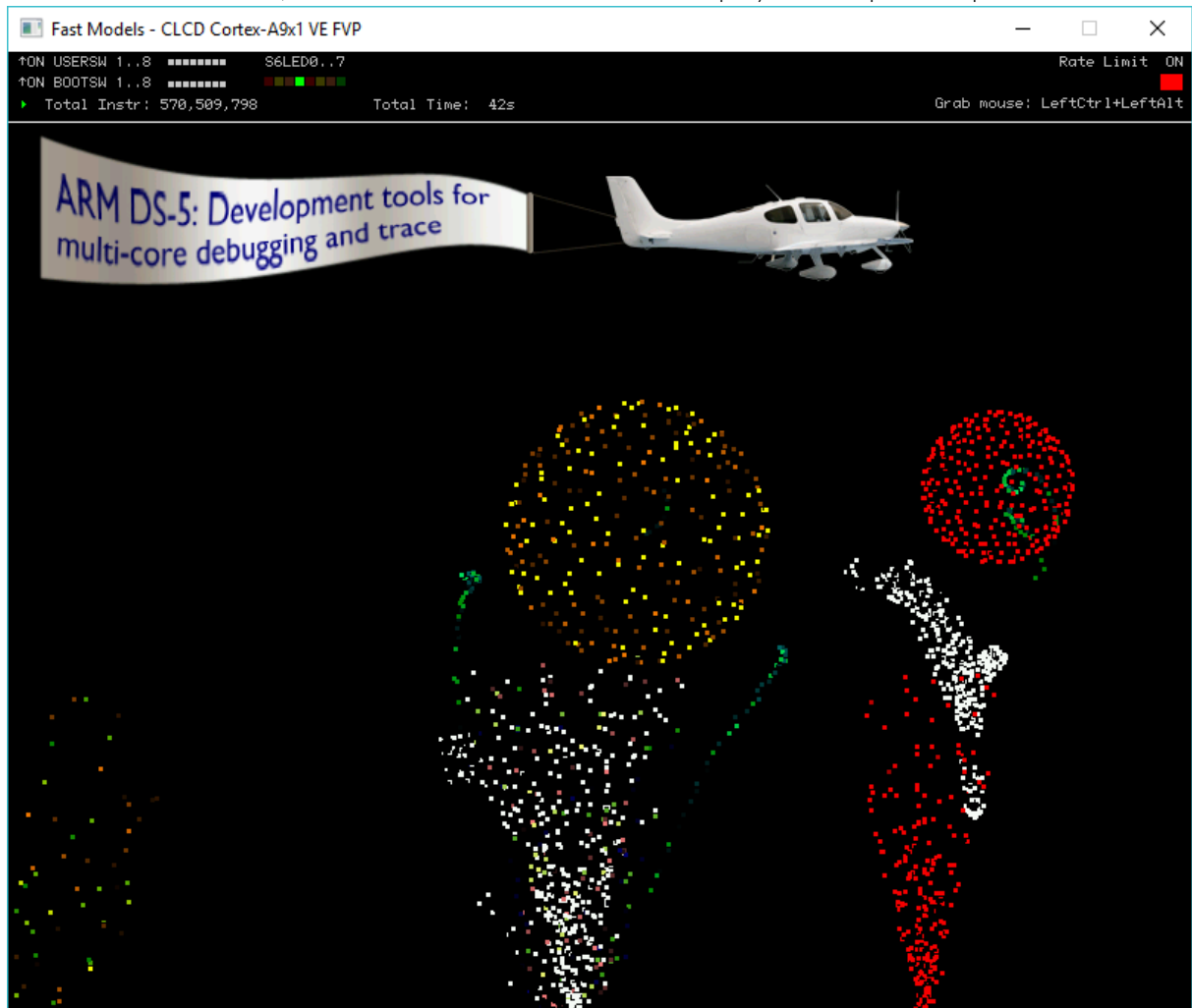
For this tutorial, we are using the single-core configuration:

1. Right-click `fireworks_AC5-Cortex-A9x1-FVP.launch` and select **Debug As > fireworks_AC5-Cortex-A9x1-FVP**.
2. If DS-5 asks if you want to change perspective, click **Yes** to switch to the Debug perspective. DS-5 automatically launches the FVP model, downloads the application, sets a breakpoint on the `main()` function, loads debug information into DS-5 Debugger,

and runs to main(). You should get a view similar to the following screenshot:



3. To continue execution, click  Button. The fireworks display starts up in a separate window.



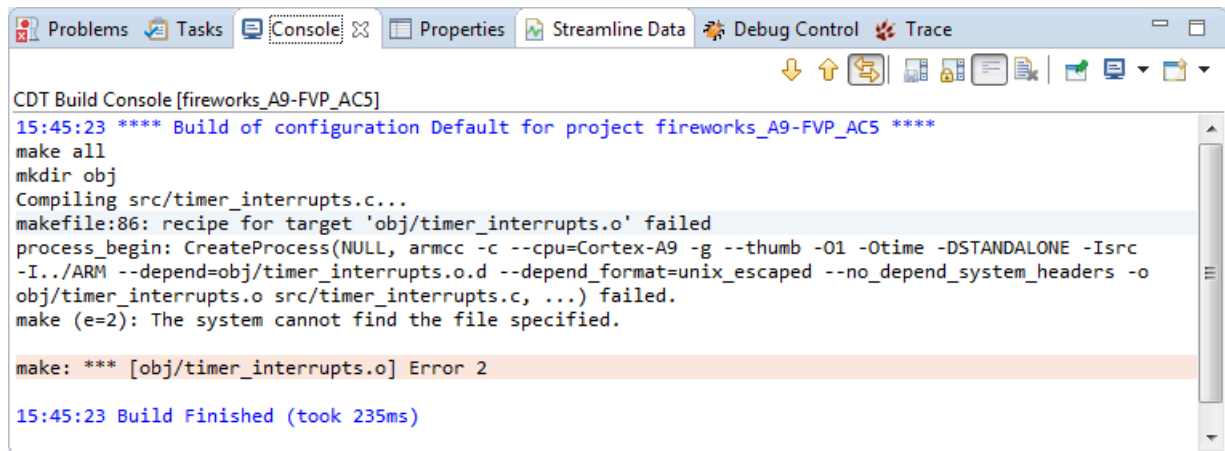
You have now imported the project, rebuilt it, and then run the application on a FVP model. Now, click Disconnect button to stop the simulation.

6. Clean and rebuild the project

Changing the compiler means we have to rebuild all the object files from scratch. Right-click the fireworks project and select Clean Project to remove the contents of the `obj` directory.

The changes are not complete yet, but just to see what happens, we are going to build the project. Right-click the fireworks project and select Build Project. You can view the results of the build in the Console view. The build fails with the error message:

Figure 6-1: Error message



```
CDT Build Console [fireworks_A9-FVP_AC5]
15:45:23 **** Build of configuration Default for project fireworks_A9-FVP_AC5 ****
make all
mkdir obj
Compiling src/timer_interrupts.c...
makefile:86: recipe for target 'obj/timer_interrupts.o' failed
process_begin: CreateProcess(NULL, armcc -c --cpu=Cortex-A9 -g --thumb -O1 -Otime -DSTANDALONE -Isrc
-I../ARM --depend=obj/timer_interrupts.o.d --depend_format=unix_escaped --no_depend_system_headers -o
obj/timer_interrupts.o src/timer_interrupts.c, ...) failed.
make (e=2): The system cannot find the file specified.

make: *** [obj/timer_interrupts.o] Error 2

15:45:23 Build Finished (took 235ms)
```

This is because even though we have changed the default compiler for the project to be Arm Compiler 6, the Eclipse project for Fireworks is a makefile project as opposed to a managed project), and the makefile still references Arm Compiler 5 (`armcc`). As part of switching the compiler, you need to make some additional changes to the project for it to build correctly with Arm Compiler 6 (`armclang`).

7. Modifying the makefile

This project uses a `makefile` with the actual compiler calls embedded in it. We need to make changes in the file.

We are going to make changes in two stages:

- In the first stage, we are going to specify which compiler to use.
- In the second stage, we are going to specify which compiler options to use.

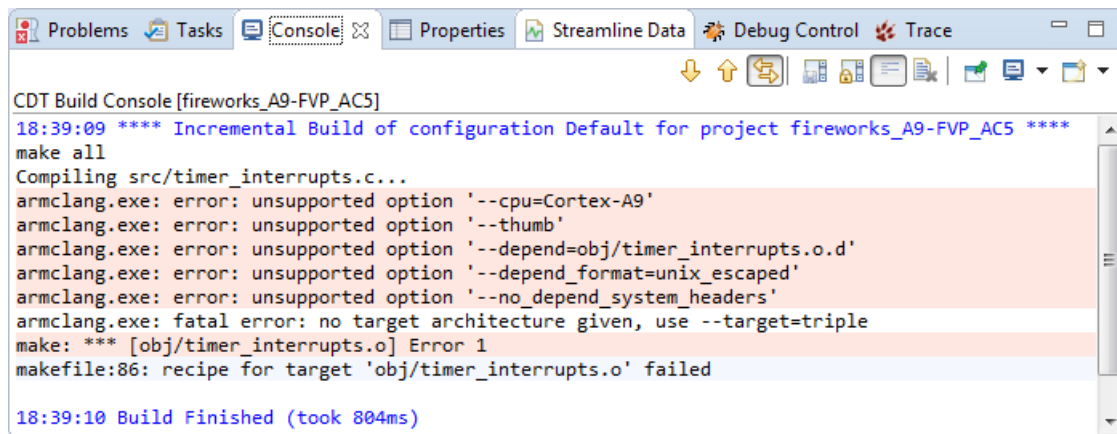
8. Specifying which compiler to use

We are going to specify which compiler to use with the following steps:

1. In the Project Explorer view, browse the `Fireworks` project and locate the `makefile` file.
2. Double-click the file to open it in the editor.
3. Change the entry `cc = armcc` to `cc = armclang`. `armcc` is the name of the Arm Compiler 5 binary. We are replacing this with the name of the Arm Compiler 6 binary, which is `armclang`.
4. Save the `makefile`.

We have not finished making changes, but to see what happens, right-click and rebuild the project. The build fails with the following errors:

Figure 8-1: error message



The screenshot shows the CDT Build Console for the project `fireworks_A9-FVP_AC5`. The console output indicates an incremental build failure. The errors are as follows:

```
18:39:09 **** Incremental Build of configuration Default for project fireworks_A9-FVP_AC5 ****
make all
Compiling src/timer_interrupts.c...
armclang.exe: error: unsupported option '--cpu=Cortex-A9'
armclang.exe: error: unsupported option '--thumb'
armclang.exe: error: unsupported option '--depend=obj/timer_interrupts.o.d'
armclang.exe: error: unsupported option '--depend_format=unix_escaped'
armclang.exe: error: unsupported option '--no_depend_system_headers'
armclang.exe: fatal error: no target architecture given, use --target=triple
make: *** [obj/timer_interrupts.o] Error 1
makefile:86: recipe for target 'obj/timer_interrupts.o' failed
18:39:10 Build Finished (took 804ms)
```

This is because the `makefile` is attempting to use `armcc` command-line options with `armclang`. `armclang` and `armcc` use different command-line options. The second set of changes translate the `armcc` options into `armclang` options.

9. Specifying which compiler options to use

With the following steps, we are going to specify which compiler options to use:

1. Locate the below text in the `makefile`:

```
DEPEND_FLAGS = --depend=$@.d --depend_format=unix_escaped
CFLAGS = --cpu=$(CPU) $(DEBUG_FLAGS) --$(CODE_TYPE) -O$(OPT_LEVEL) -O$(OPT_FOR)
$(DEFINES) $(INCLUDES) $(DEPEND_FLAGS) --no_depend_system_headers $(SUPPRESS)
AFLAGS = --cpu=$(CPU) $(DEBUG_FLAGS) --apcs=/interwork $(DEPEND_FLAGS)
$(SUPPRESS)
```

2. Replace it with:

```
DEPEND_CFLAGS = -MF $@.d
DEPEND_AFLAGS = --depend=$@.d --depend_format=unix_escaped
CFLAGS = --target=arm-arm-none-eabi -m$(CODE_TYPE) $(DEBUG_FLAGS) -O$(OPT_LEVEL)
-mcpu=$(CPU) $(DEFINES) $(INCLUDES) $(DEPEND_CFLAGS) -MMD $(SUPPRESS)
AFLAGS = --cpu=$(CPU) $(DEBUG_FLAGS) --apcs=/interwork $(DEPEND_AFLAGS)
$(SUPPRESS)
```

None of the assembler options (`AFLAGS` and `DEPEND_AFLAGS`) or linker options (`LFLAGS`) need changing since these tools are included in Arm Compiler 6 and are backwards compatible with Arm Compiler 5.

To summarize the changes required in the `makefile`:

Description	Arm Compiler 5	Arm Compiler 6
Change compiler binary name	armcc	armclang
Specify the target architecture	None (only AArch32)	-target=arm-arm-none-eabi
Select CPU	-cpu	-mcpu
Generate Thumb code	-thumb	-mthumb
Optimize for time	-Otime	None (default)
Writes makefile dependency lines to a file during compilation	-depend=\$@.d	-MF \$@.d
Specifies the format of output dependency files	-depend_format=unix_escaped	None (default)
Disables the output of system include dependency lines	-no_depend_system_headers	-MMD

See the [Arm Compiler Migration and Compatibility Guide](#) for more information about migrating from older versions of Arm Compiler to Arm Compiler 6.

10. Modifying your code

Now that you have made the necessary modifications to your `makefile`, you have to make some changes to your code.

1. Rebuild the project and view the errors and warnings generated in the Console view:

Figure 10-1: Console view

```

CDT Build Console [fireworks_A9-FVP_AC5]
09:52:08 **** Build of configuration Debug for project fireworks_A9-FVP_AC5 ****
make all
mkdir obj
Compiling src/timer_interrupts.c...
Compiling src/banner_data.c...
Compiling src/main.c...
src/main.c:69:27: warning: return type of 'main' is not 'int' [-Wmain-return-type]
__attribute__((noreturn)) void main(void)
                           ^
src/main.c:69:27: note: change return type to 'int'
__attribute__((noreturn)) void main(void)
                           ^
                           int
src/main.c:83:3: warning: implicit declaration of function '__enable_irq' is invalid in C99 [-Wimplicit-function-declaration]
__enable_irq();           // armcc compiler intrinsic
                           ^
2 warnings generated.
Compiling src/Fireworks.c...
Compiling src/screen.c...
Compiling src/uart.c...
Compiling src/retarget.c...
src/retarget.c:35:9: error: '#pragma import' is an ARM Compiler 5 extension, and is not supported by ARM Compiler 6 [-Warmcc-pragma-import]
#pragma import(__use_no_semihosting)
           ^
1 error generated.
make: *** [makefile:88: obj/retarget.o] Error 1

09:52:12 Build Finished (took 3s.670ms)

```

2. Diagnose and fix the errors and warnings:
 - a. The return type of the main function: void is not allowed in standard C. You can fix this by replacing the following line in `main.c`:

```
__attribute__((noreturn)) void main(void)
```

With

```
__attribute__((noreturn)) int main(void)
```

- b. The function `__enable_irq()` is supported by Arm Compiler 6 in a compatibility header file. Add `#include <arm_compat.h>` to the top of `main.c`.
- c. `#pragma import` is not supported by Arm Compiler 6 and must be replaced by the assembler directive equivalent. Replace the following line in `retarget.c`:

```
#pragma import(__use_no_semihosting)
```

With

```
asm(".global __use_no_semihosting");
```

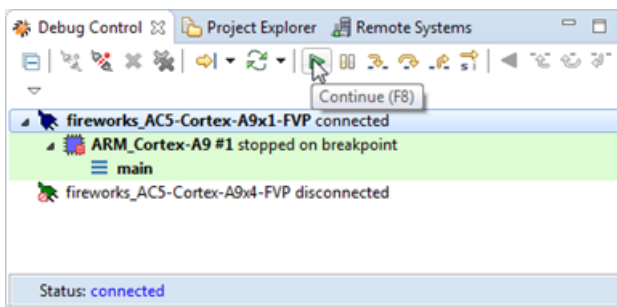
3. Rebuild the project again.

11. Running the modified application

You have completed all required changes to the project and the executable is ready to be run on the FVP.

1. Switch to the DS-5 Debug perspective.
2. In the Debug Control view, either double-click fireworks_AC5-Cortex-A9x1-FVP or right-click and select Connect to target.
3. Click Continue.

Figure 11-1: Running the modified application



You can view the fireworks output running successfully.

12. Converting assembly code

A further step would be to convert the assembler format used from `armasm` format to `gas` format that can be consumed by `armclang`'s built-in assembler. The instructions are the same in both formats, but the assembler directives, predefined macros and comment characters are different. If you are writing new assembler in `gas` format, it can be assembled with `armclang -x assembler-with-cpp`.