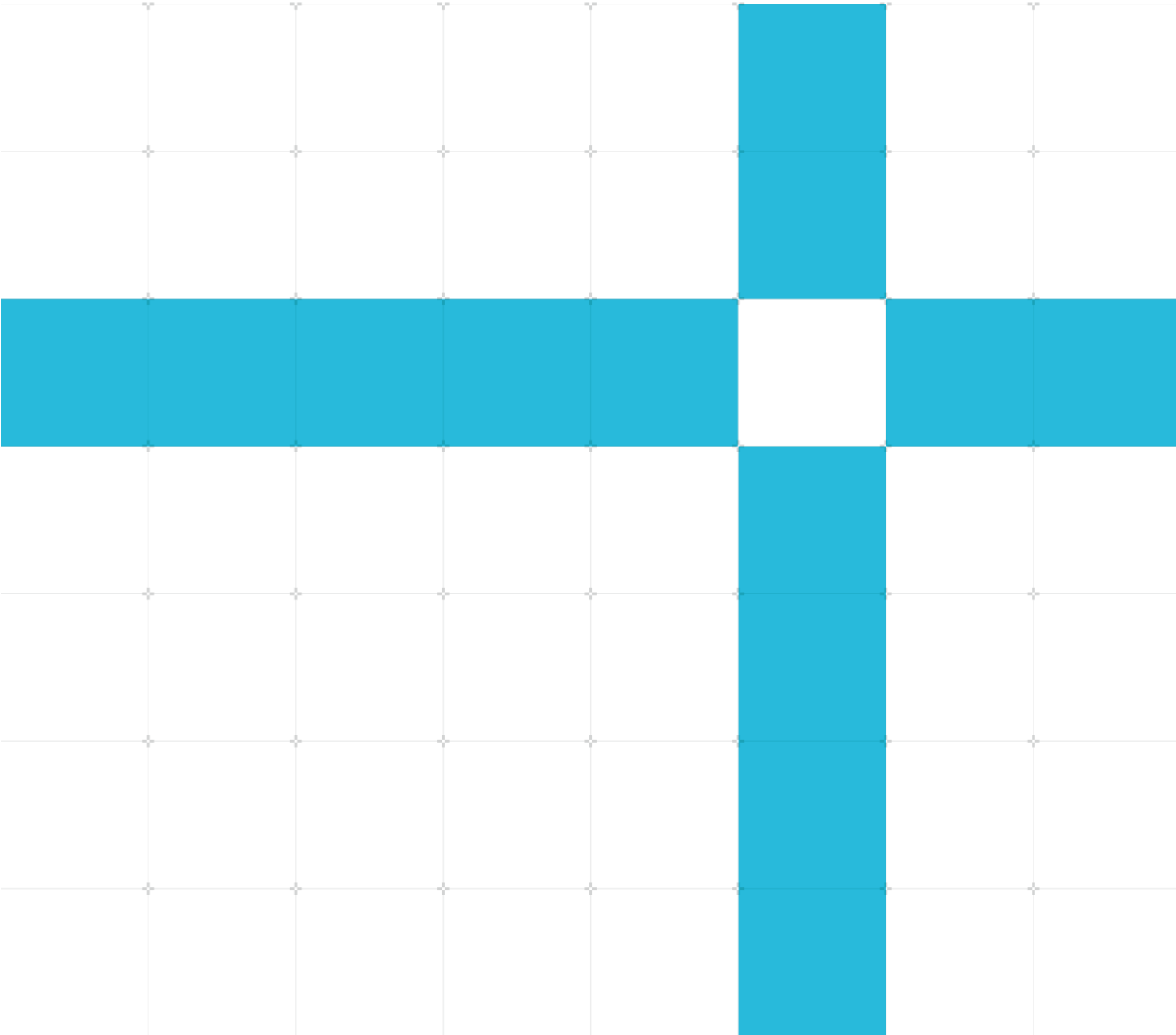
Arm® DesignStart FPGA on Cloud

Revision: r0p0

Arm DS Getting Started Guide

Non Confidential

Issue 1.1



DesignStart FPGA on Cloud

Arm DS Getting Started Guide

Release information

Document history

A	15 November 2019	Non-Confidential	First release for r0p0
B	25 November 2019	Non-Confidential	Additional information related to multi core debug

Non-Confidential Proprietary Notice

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.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

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

Contents

1 Preface

1.1 Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this document, for example, r*1*p*2*, where:

r*m* Identifies the major revision of the product, for example, r*1*.

p*n* Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

This document is written to provide information to hardware and software engineers who are implementing designs on Amazon Web Services (AWS). This document is written for experienced hardware and software engineers who might or might not have experience of Arm products, but who have experience of writing software and of targeting development platforms.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the **Arm Glossary** for more information.

1.3.2 Typographical conventions

Convention	Use
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`Monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.
	Caution
	Warning
	Note

1.4 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Arm publications

- **DesignStart FPGA on Cloud: Cortex-M33 based platform Technical Reference Manual**
- **Arm Development Studio Getting Started Guide**

Other publications

- **Amazon EC2 F1 Instances**

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 Feedback on this product

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

- The product name
- The product revision or version
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.5.2 Feedback on content

If you have comments on content, send an e-mail to **rce-contact@arm.com** and give:

- The title Arm DS Getting Stated Guide
- If applicable, the page numbers to which your comments refer
- A concise explanation of your comments

Arm also welcomes general suggestions for additions and improvements.

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

2 Introduction

Amazon Web Services (AWS) is a cloud provider that enables you to create virtual private server instances, some of which include a Field Programmable Gate Array (FPGA) that can be programmed to run your own software payloads. Arm Development Studio (Arm DS) allows you to connect to this AWS instance from your local machine and remotely debug the software payload.

This Getting Started Guide is intended to provide:

- Information on how to set up an AWS FPGA instance
- Information on how to connect a local Arm DS installation to the AWS instance
- Information on building test applications, running them on AWS, and debugging them from a local Arm DS installation
- Troubleshooting information for configuring AWS and Arm DS

3 Setting up AWS and Arm DS

This document describes the steps that are necessary to use Arm DS with the DesignStart FPGA on Cloud offering on AWS. The solution is based on a server-client architecture, where the AWS instance acts as the server, while a remote computer acts as the client.

3.1 Prerequisites

In order to ensure that you can connect to the instance, the following items are required:

- The AWS EC2 F1 instance is launched by selecting the " DesignStart FPGA on Cloud with ArmDS Software Debug Enabled 1.0" AMI.
- A valid Arm DS license and DS installation version 2019.1 or above on the user's host machine.

3.2 Host setup

The AMI security settings need to be changed to enable the debug server to communicate over TCP/IP.

The following inbound ports need to be opened:

- TCP 3010
- TCP 22

Ports can be opened by selecting the instance and clicking its security group in the bottom panel and editing the Inbound settings. The following example configuration shows a setup which enables connections from anywhere. Arm also recommends restricting incoming addresses to specific IP addresses for additional security.

Figure 1: Example configuration for AWS security settings

3.3 Client setup

As a first step, Arm recommends that you explore the **Arm Development Studio Getting Started Guide**.

Download the configuration database from the path "/home/ec2-user/developmentstudio-workspace/AWS_M33x2_Bronze" to local directory.

Do this by connecting to the instance over SSH, the IP address is supplied under the **Connect** button, an example is shown below in Figure 2 Connect To Your Instance, the name of the .pem file and IP address will change on your usage

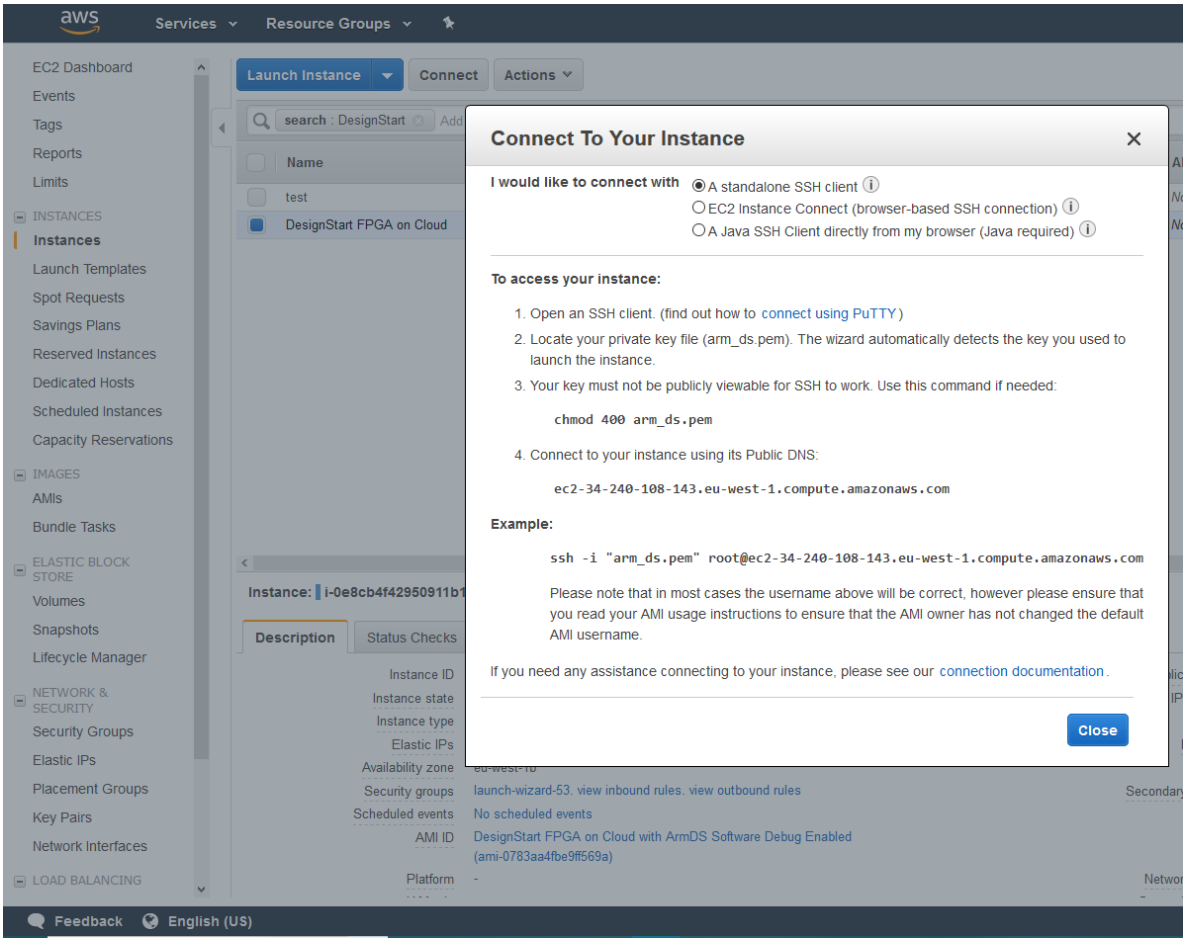


Figure 2 Connect To Your Instance

In your Unix shell you can scp the configuration database to your local machine, an example is given below. Note the IP address will be different every time you start an instance, use the IP address provided by the AWS console.

```
scp -r -i "arm_ds.pem" ec2-user@ec2-34-240-108-143.eu-west-
1.compute.amazonaws.com:/home/ec2-user/developmentstudio-
workspace/AWS_M33x2_Bronze ./
```

Use the following steps to import configuration database provided as part of the AMI into Arm DS.

1. Launch Arm DS
2. **Click File** > **Import…** as shown in Figure 3 Importing an existing configuration databaseError! Reference source not found.

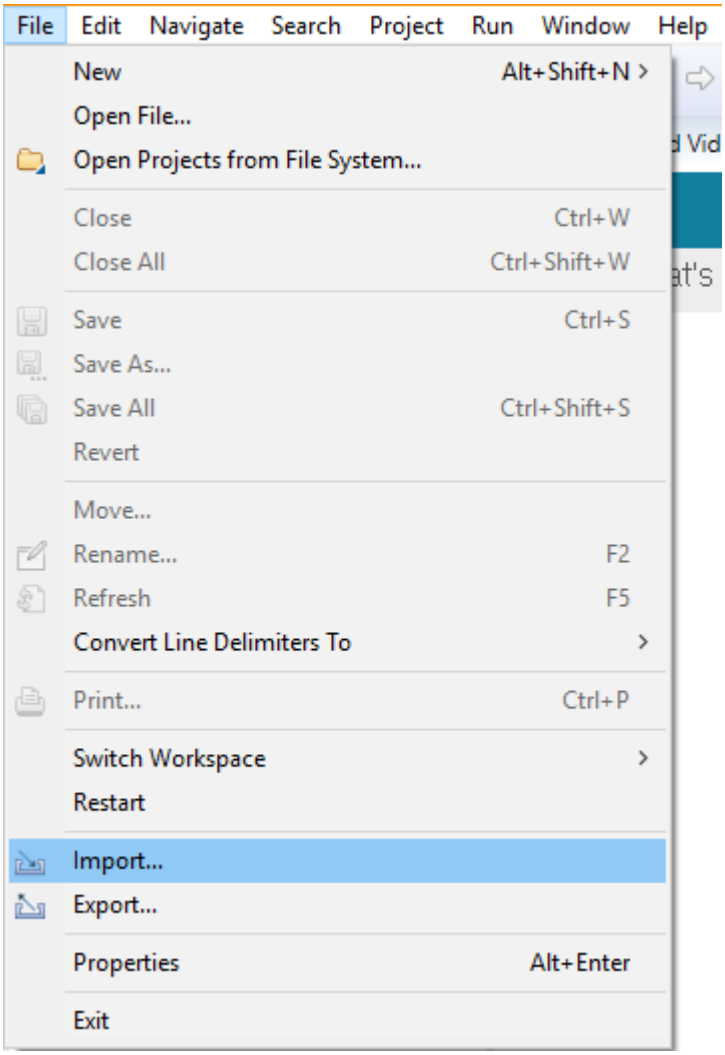


Figure 3 Importing an existing configuration database

2. Select **General** > **Existing Projects into Workspace** in the pop-up window shown in Figure 4 Importing existing Project into Workspace

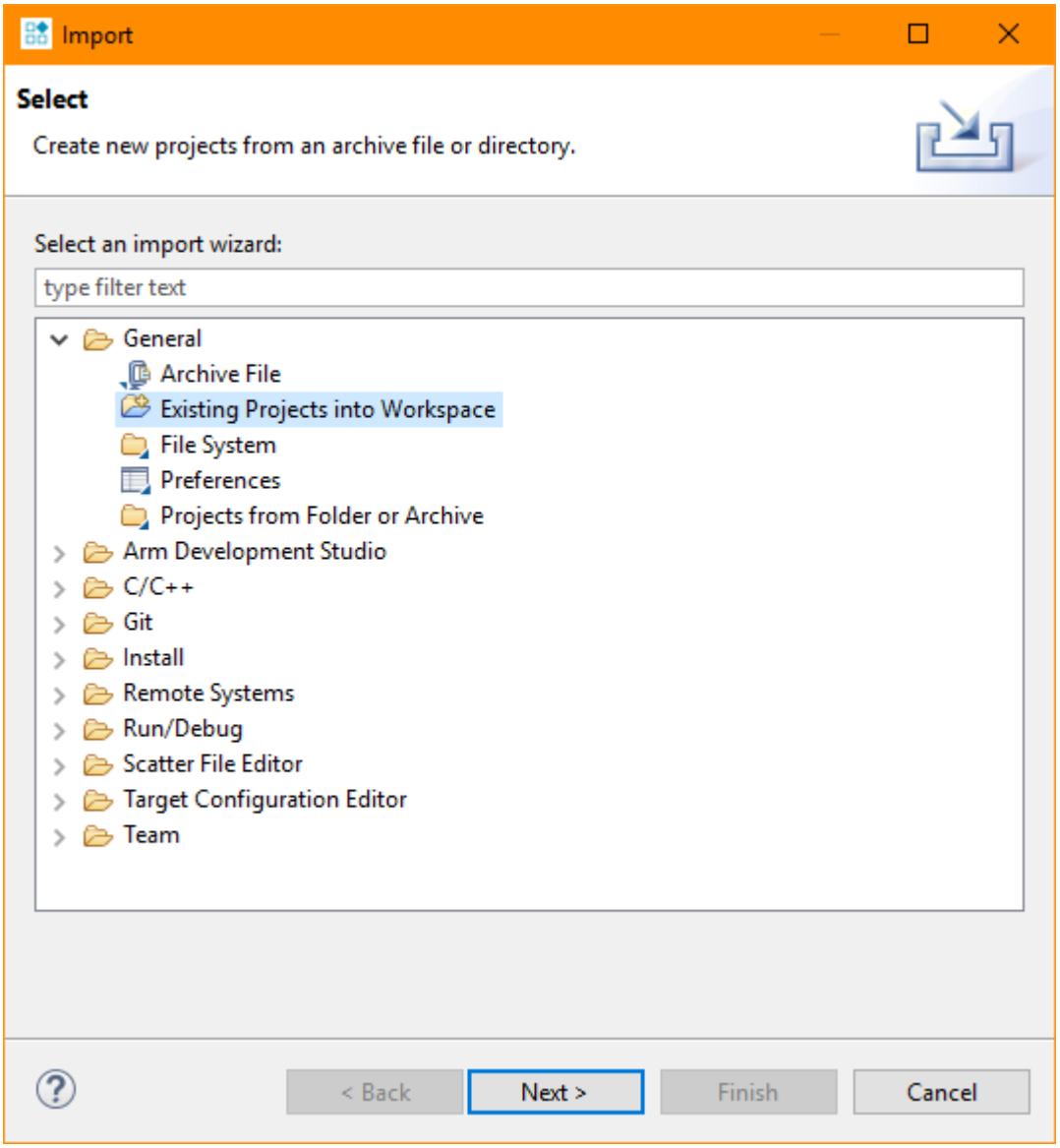


Figure 4 Importing existing Project into Workspace

3. In the **Import Projects** dialog select the path to configuration database from your local directory and select the "Copy projects into workspace" option before clicking **Finish** as shown in Figure 5 Importing an existing configuration database

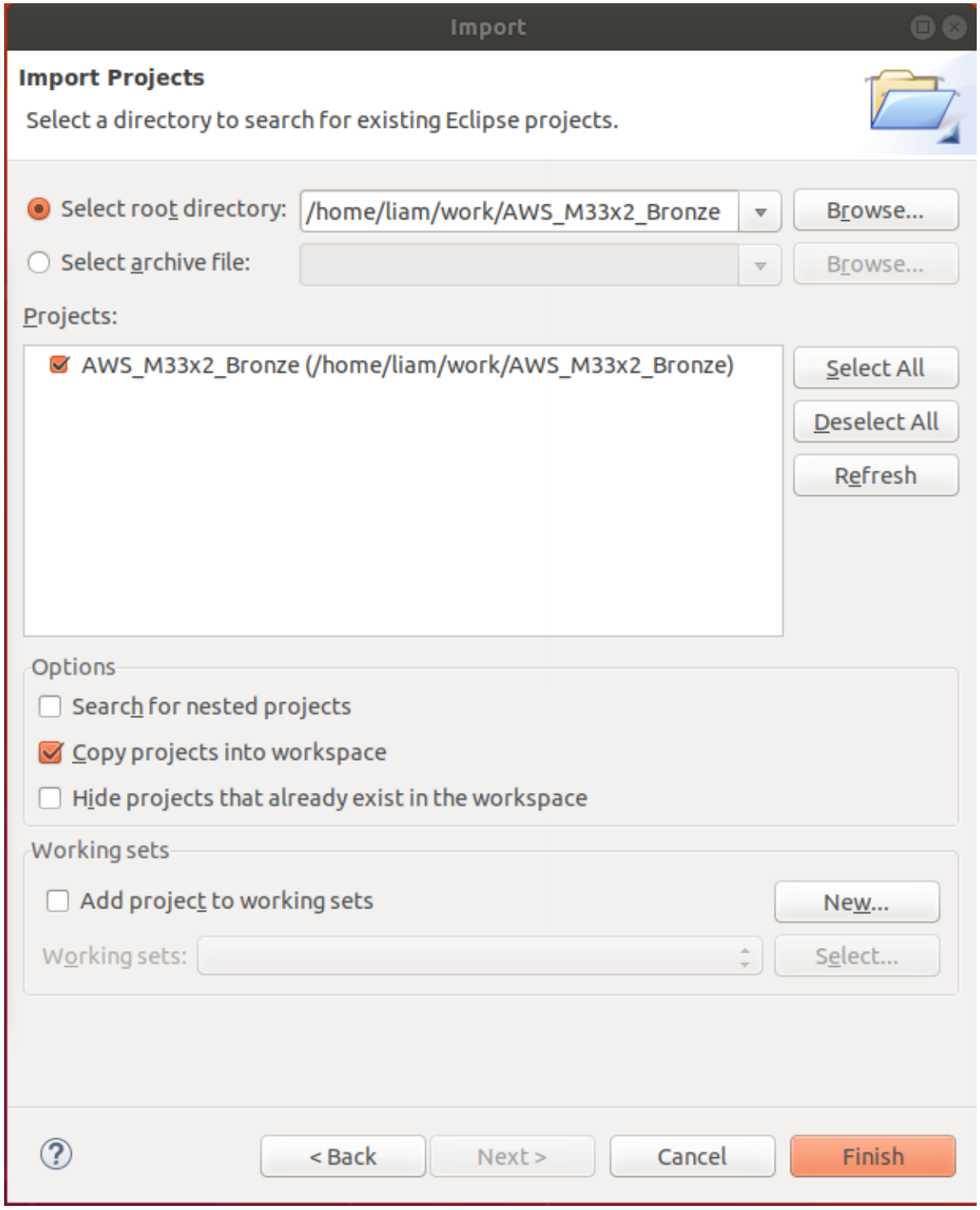


Figure 5 Importing an existing configuration database

4. Select Window -> Preferences as shown in Figure 6 Rebuilding imported configuration database

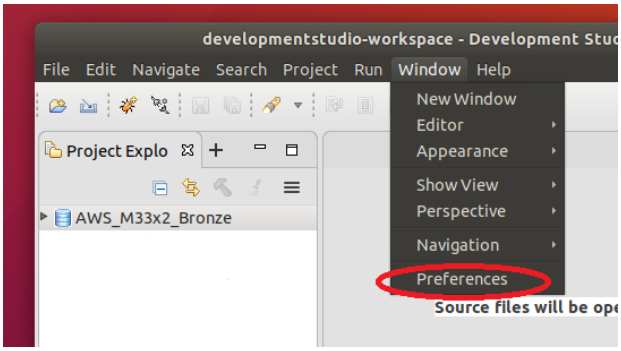


Figure 6 Rebuilding imported configuration database

5. In the **Preferences** dialog choose Arm DS -> Configuration Database option and select the imported configuration database under "User Configuration Databases". Now, click on the "Rebuild database" and "OK" to complete the process as shown in Figure 7 Rebuilding imported configuration database

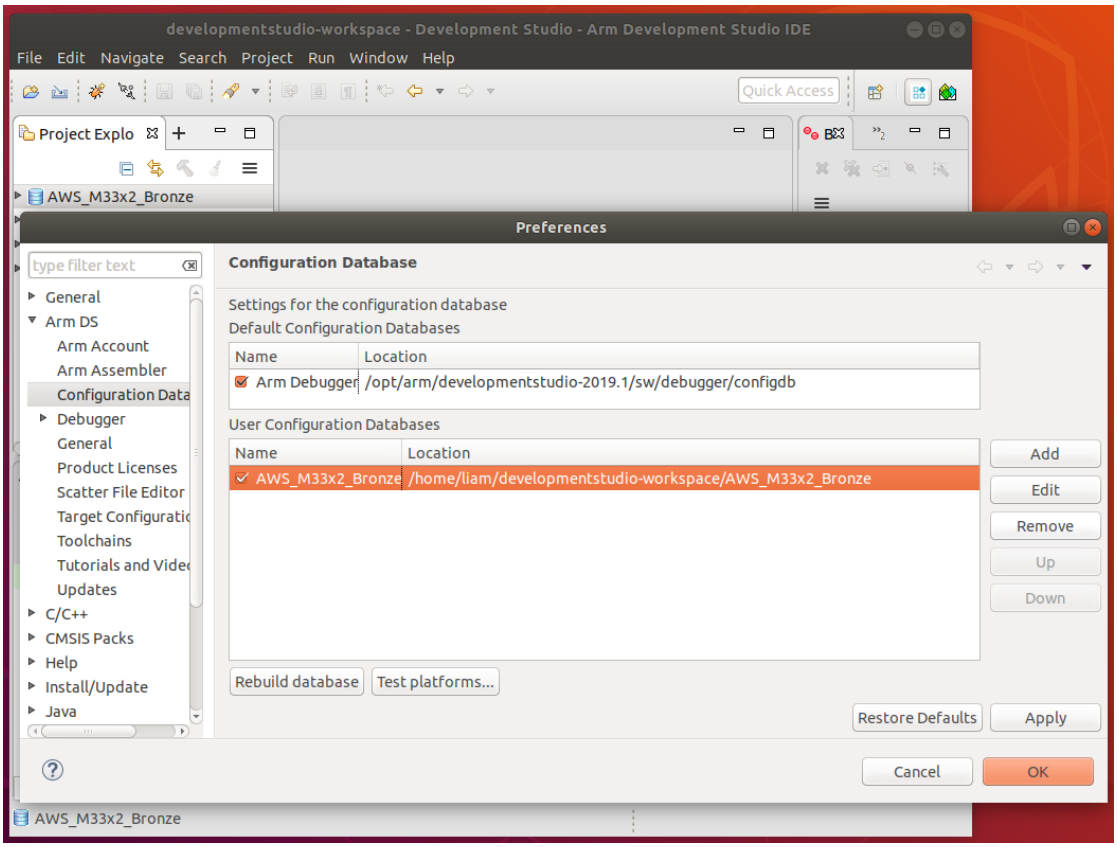


Figure 7 Rebuilding imported configuration database

3.4 Using Arm DS with DesignStart FPGA on cloud

The FPGA module on the AWS instance must be cleared and loaded with an image and payload before we connect Arm DS to the instance. Connect to the instance and run the following commands on the AWS instance to achieve this:

1. Connect to the instance via SSH, as before the name of the .pem file and IP address will change on your usage

```
ssh -i "arm_ds.pem" ec2-user@ec2-34-240-108-143.eu-west-1.compute.amazonaws.com
```

2. The next step is to clear the FPGA and load the AFI

```
#Load FPGA image: Terminal session 1
sudo fpga-clear-local-image -S 0
sudo fpga-load-local-image -S 0 -I agfi-0b63123c24cc6f8df
```

3. Now SSH again and setup the a Virtual UART in a different console

```
#Virtual UART : Terminal session 2
ssh -i "arm_ds.pem" ec2-user@ec2-34-240-108-143.eu-west-1.compute.amazonaws.com
cd /home/ec2-user/vUART
sudo ./uart 0x000 0x004 0x008 ./out.txt 1
```

4. Switching back to the first terminal, load the example software image Load software payload: Terminal session 1

```
#Load FPGA image: Terminal session 1
cd /home/ec2-user/preloader
sudo ./preload_software -S 0 -p ./TestPayloads/single_file_test/Hello_World.axf
```

5. Start the debug server

```
sudo bash
cd /opt/arm/developmentstudio-2019.1/sw/debughw/debug_server
export LD_LIBRARY_PATH=/opt/arm/developmentstudio-2019.1/sw/debughw/debug_server
 ./rddidap_serverd -log -rddi_dap_dll ./librddi_3rd_party_probe_aws.so -debuglog
```

3.5 Creating new Arm DS debug configuration

The following steps will guide you through the process of setting up a new debug configuration on a local Arm DS installation.

1. Click **Run** > **Debug Configurations…** on the Arm DS menu bar as shown in the Figure 8 Debug Configuration

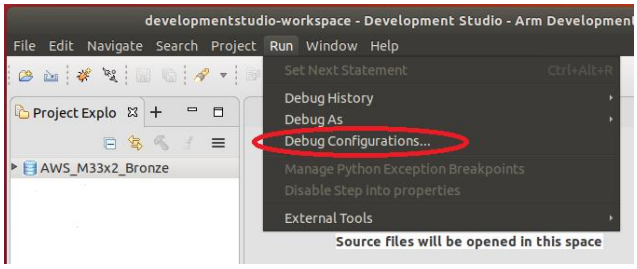


Figure 8 Debug Configuration

2. Set the target to **Arm** > **AWS_M33x2_Bronze** > **Bare Metal Debug** > **Cortex-M33** under the **CONNECTION** tab in the "Debug Configurations" dialog.

   Choose the "AWS Virtual Debug" option from the **Target Connection** drop down list.

Now, set the connection to the IP address provided by AWS, see Figure 9 Public IP address location as an example

```
server=<AWS Instance Public IP Address>;probe=none
```

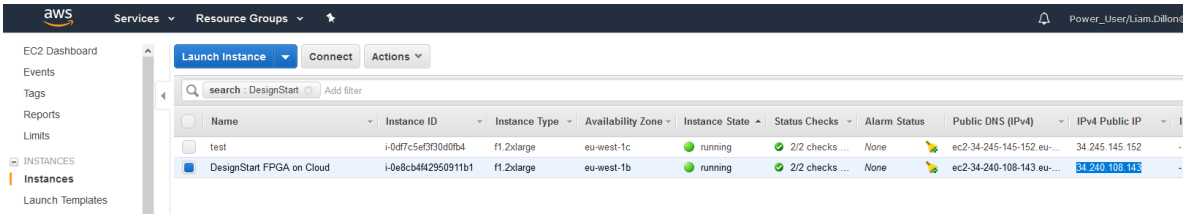


Figure 9 Public IP address location

3. Click "Debug" as shown in the Figure 10 Creating a new debug configuration

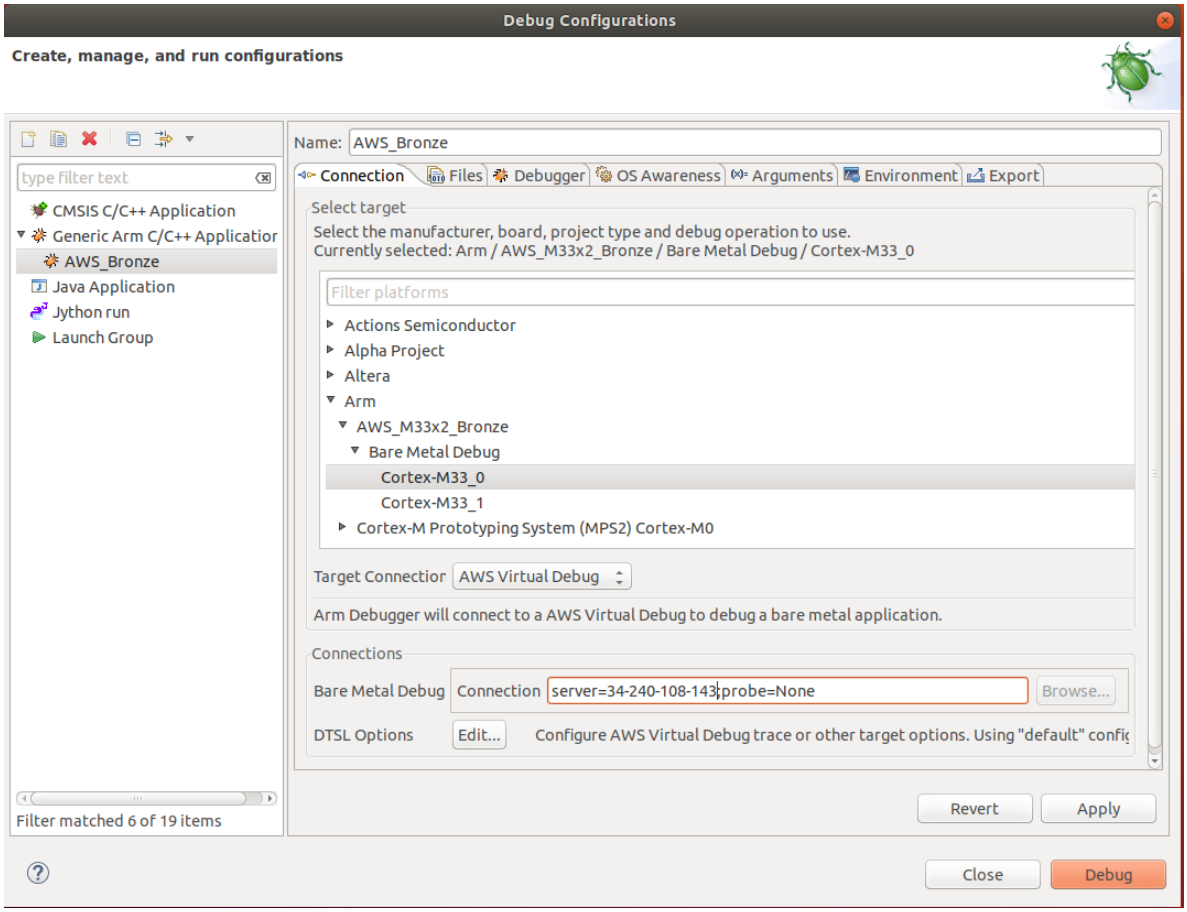


Figure 10 Creating a new debug configuration

4. The debug connection with the target on AWS FPGA cloud is as shown in the Figure 11 Connected to target on AWS FPGA cloud

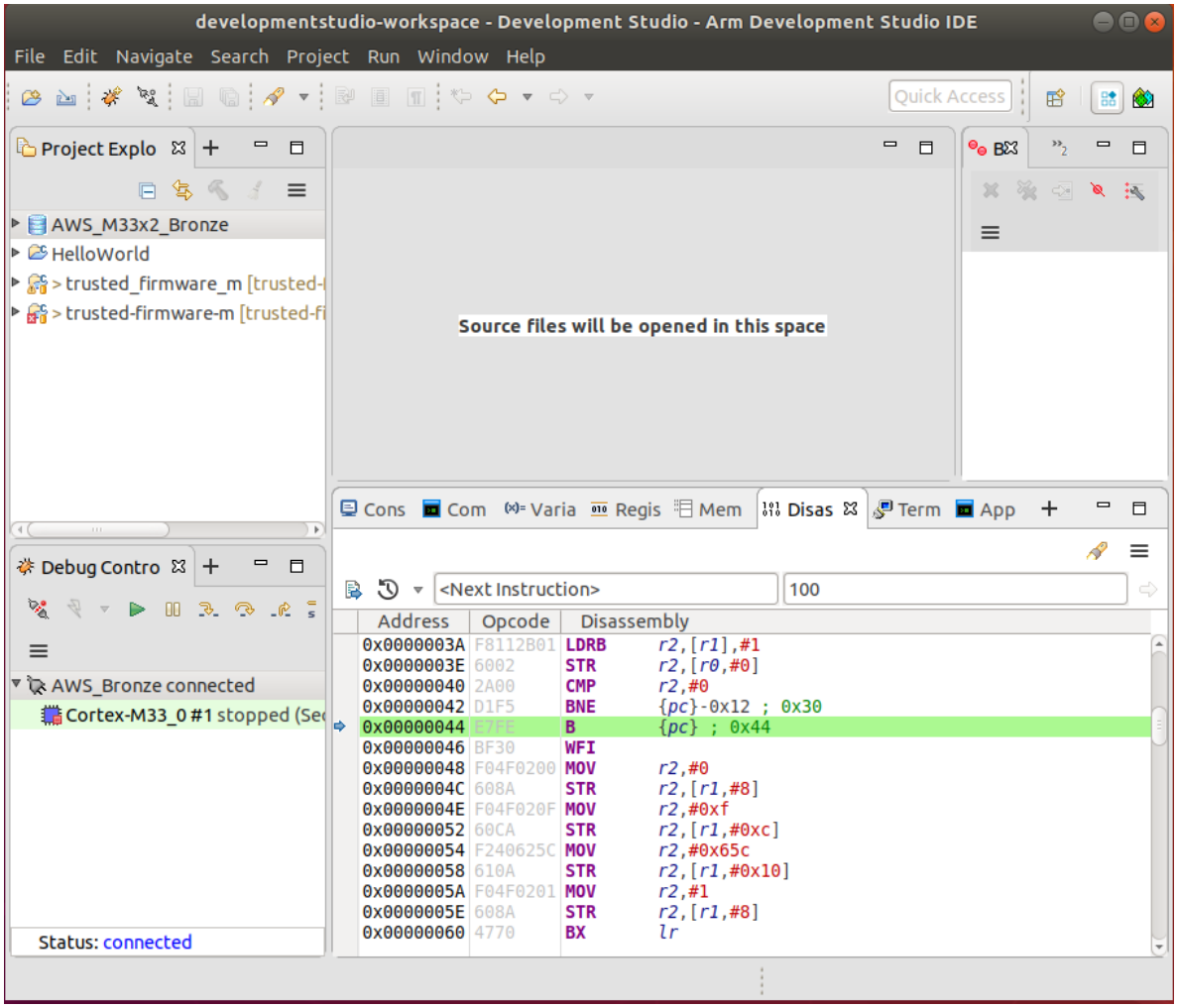


Figure 11 Connected to target on AWS FPGA cloud

4 Building, running, and debugging test applications

This section of the guide shows the process of building, running, and debugging using a simple Hello World example and a more complicated trusted firmware example. The software will be built by Arm DS on a local machine and then run on an AWS instance with the local Arm DS debugger attached.

4.1 Hello World Example

4.1.1 Creating the Arm DS project

Here are the steps to create a simple "Hello World" project.

1. Create a new C project:

   If you have no projects open, click **New Project** in the Project Explorer view.

   Otherwise, select: **File** > **New** > **Project**.
2. Expand the C/C++ menu, and select C project, then click **Next**.
3. In the C Project dialog:

   In the Project name field, enter HelloWorld.

   Under Project type, select **Executable** > **Hello World ANSI C Project**.

   Under Toolchains, select **Arm Compiler 6**.

   Click **Finish**.

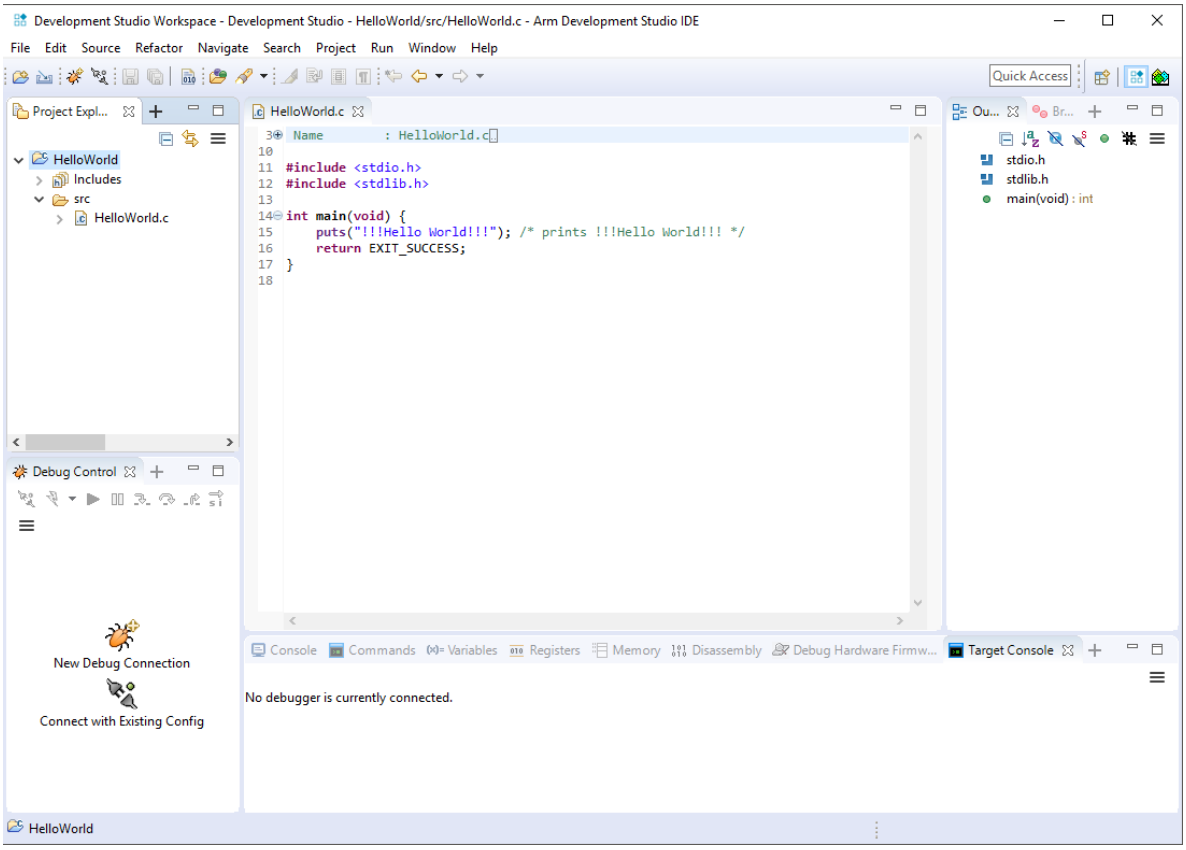


Figure 12 Hello World Project creation

4. In the Project Explorer view, right-click the HelloWorld project and select **Properties**. The Properties for HelloWorld dialog box opens.
5. Add debug symbols into the image file:

   Expand C/C++ Build, and select **Build Variables**

   Set Configuration to **Debug [Active]**.
6. Set the linker base RAM address, under C/C++ Build select **Settings**

   In the Tool Settings tab, select **All Tools Settings** > **Target**.

   From the Target CPU dropdown, select **Cortex-M33**.

   From the Target FPU dropdown, select **No FPU**.
7. Select **Tool Settings** > **Arm Linker 6** > **Image Layout**.

   In the **RO base address** field, enter 0x80000000.

   Click **OK**.
8. If you are prompted to rebuild the index, click **Yes**.
9. In the Project Explorer view, **right-click** the HelloWorld project, and select **Build Project**.

4.1.2 Configure your debug session

A debug configuration was created in Section 3.5. The next step is to select the AWS instance and enter the correct server address, which is available from AWS console.

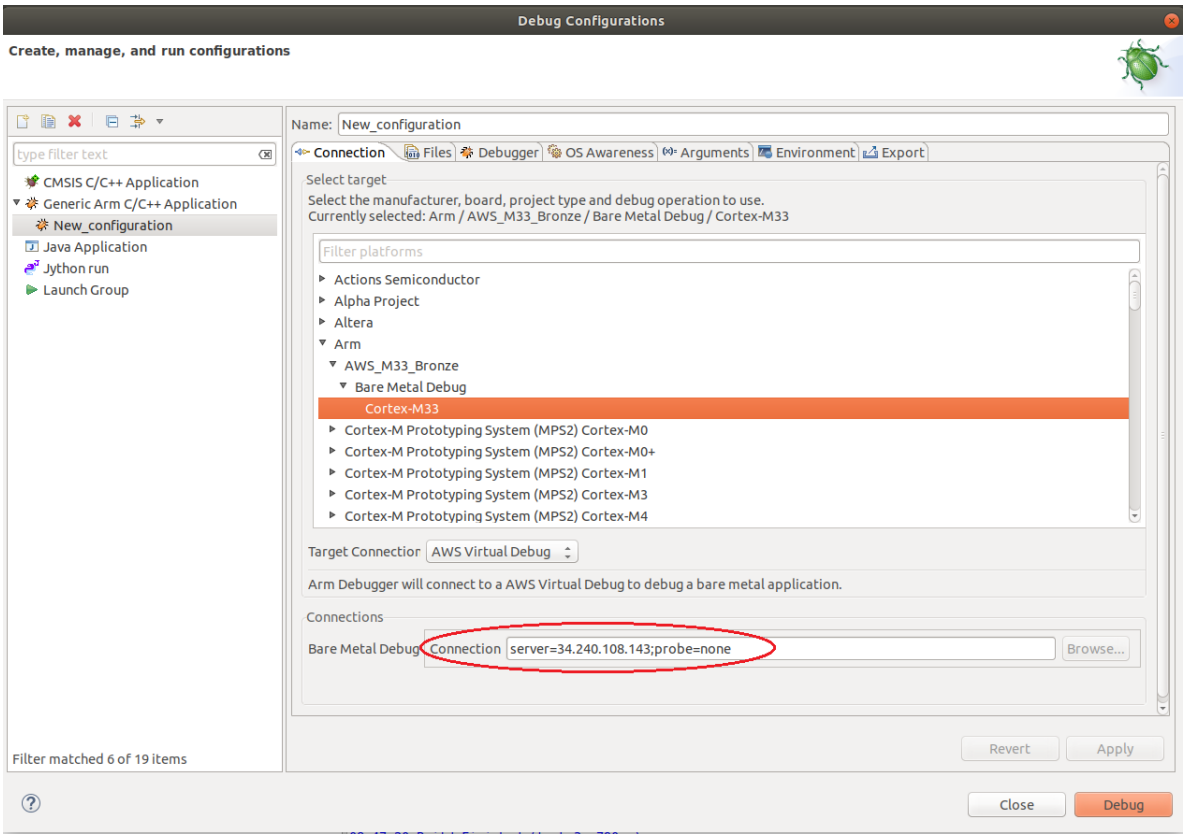


Figure 13 Target selection

1. Select the **Files** tab and click the **Workspace** button to find the file HelloWorld.axf

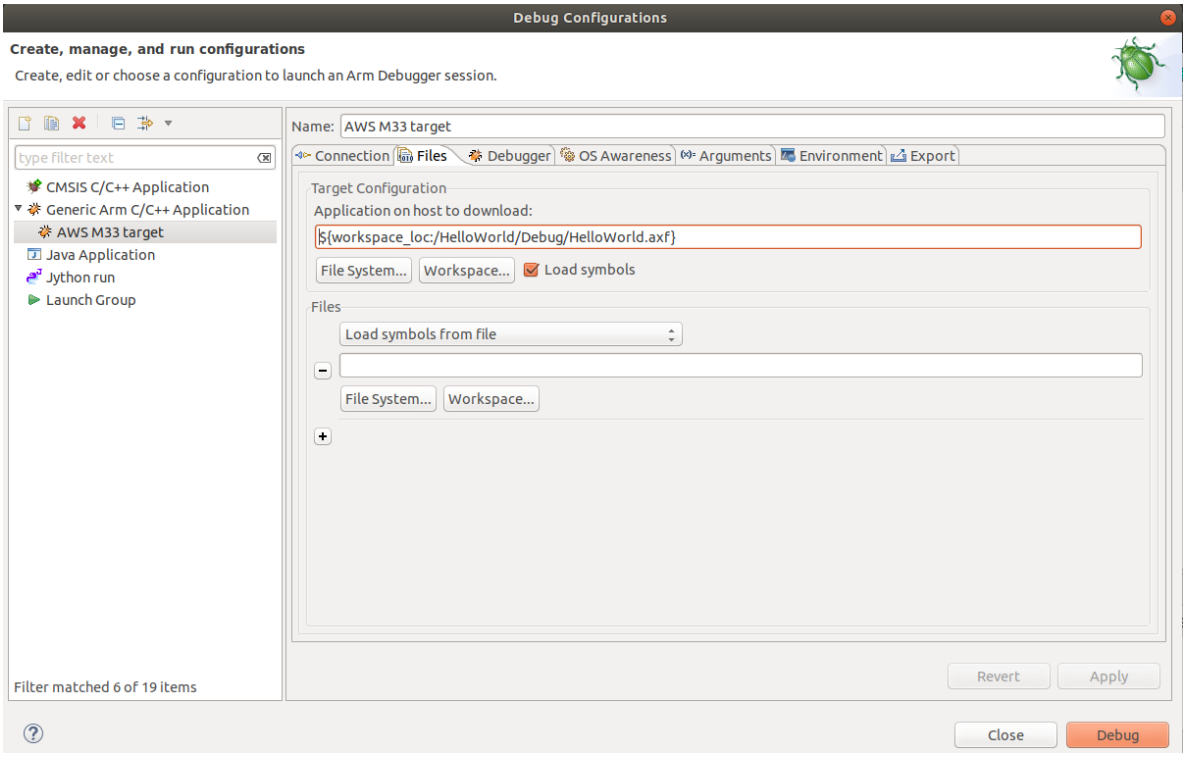


Figure 14 Program selection

2. Next step is to move to **Debugger** tab and select "**Debug from symbol**" option and click **Debug**

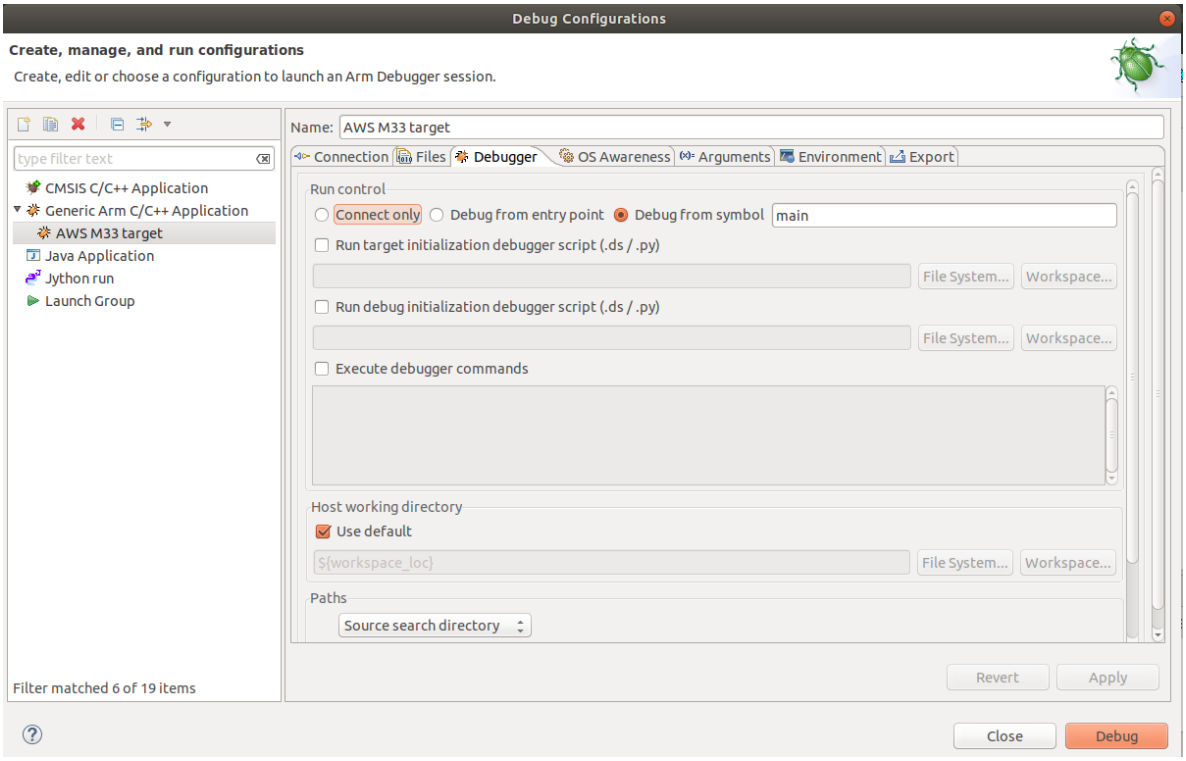


Figure 15 Debug choice

If for some reason you fail to connect, ensure the debugger server is running on the instance as in section Using Arm DS with DesignStart FPGA on cloud. Typically, this might be caused if you stop or pause the instance.

3. Once connected, click the **green play button** to run the program and present the output in the console window.

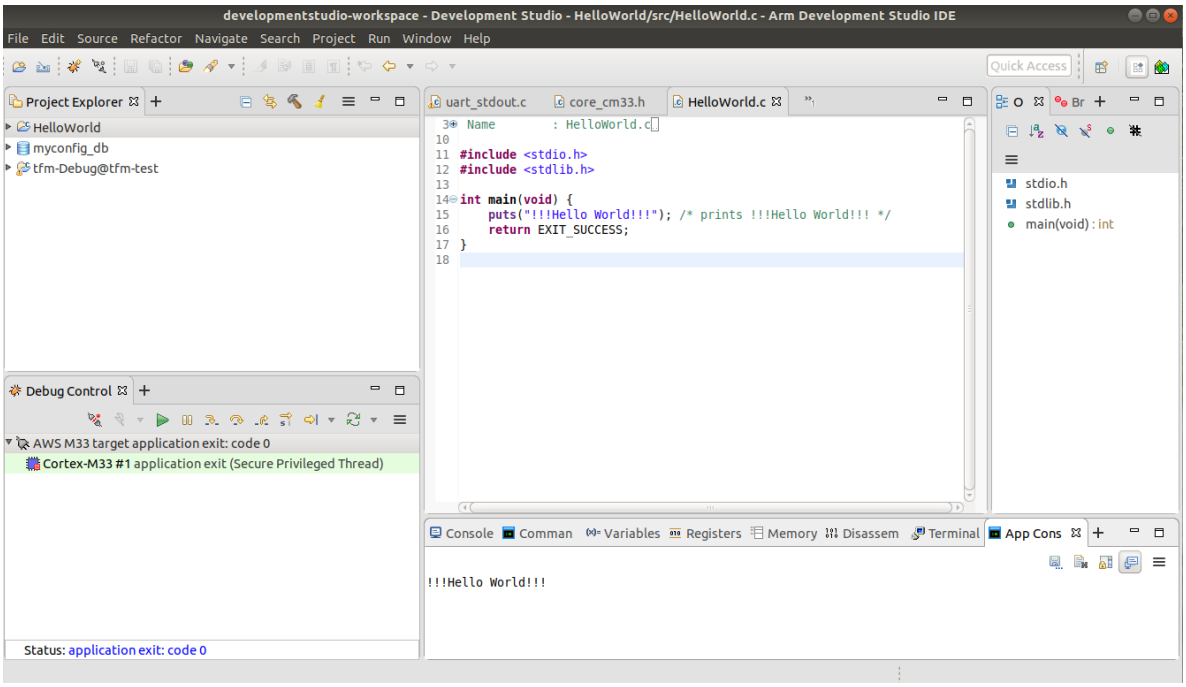


Figure 16 Program output

4.2 Arm Trusted Firmware M (TF-M) Example

This example is based on Arm Trusted firmware M, which provides a reference implementation of secure world software for ARMv8-M. For the purpose of this example Ubuntu 18.04 is used as the operating system.

4.2.1 Dependencies

Please review the **TF-M Software requirements** before you begin to ensure you have the correct build tools installed.

At this point of time TF-M does not support ARM Compiler V6.13 which comes with Arm DS version 2019.1, so ARM Compiler V6.12, this needs to be downloaded and installed. The download is available from **developer.arm.com**

Once installed, this tool chain needs to be added to Arm DS as shown in Figure 17 Arm DS Tool Chain addition

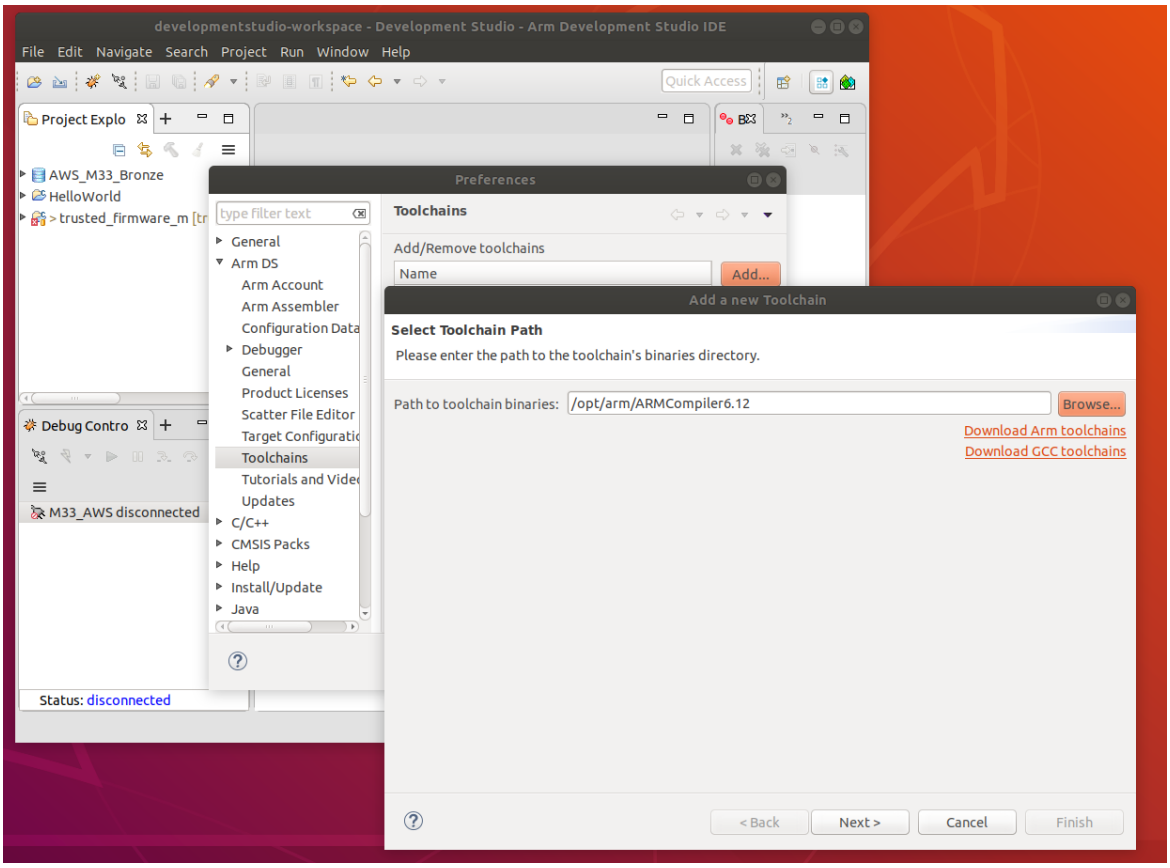


Figure 17 Arm DS Tool Chain addition

4.2.2 Building TF-M

1. Install **Git LFS**. Copy the link to the latest version for `Linux AMD64`. Then run the following commands:

```
wget https://github.com/git-lfs/git-lfs/releases/download/v2.9.0/git-lfs-linux-amd64-v2.9.0.tar.gz # Replace this link with the one you copied from the webpage

tar xf git-lfs-linux-amd64-v2.8.0.tar.gz # Replace this name with the name of the file you downloaded

sudo ./install.sh
```

2. Clone the various git repositories; this example uses Linux for the sample, **instructions for Windows** are also available.

```
mkdir tfm

cd tfm

git clone https://git.trustedfirmware.org/trusted-firmware-m.git -b TF-Mv1.0-RC2

git clone https://github.com/ARMmbed/mbedtls.git -b mbedtls-2.7.9

git clone https://github.com/ARMmbed/mbed-crypto.git -b mbedcrypto-1.1.0

git clone https://github.com/ARM-software/CMSIS_5.git -b 5.5.0

pushd CMSIS_5

git lfs install

git lfs pull

popd
```

3. It is possible to build TF-M just using the command line, detailed instructions are available on git.trustedfirmware.org under **build instructions**. However, in this example we are using Arm DS to build and compile the project.
4. Launch Arm DS
5. As a first step the project needs to have access to TF-M code, so select **File** > **Import**
6. On the import dialogue select "Existing Code as Makefile Project"
7. On the "New Project" dialogue create a project name then browse to the root of the TF-M repository, select the Arm Compiler 6.12 tool chain and then press "Finish". This will create a new project.

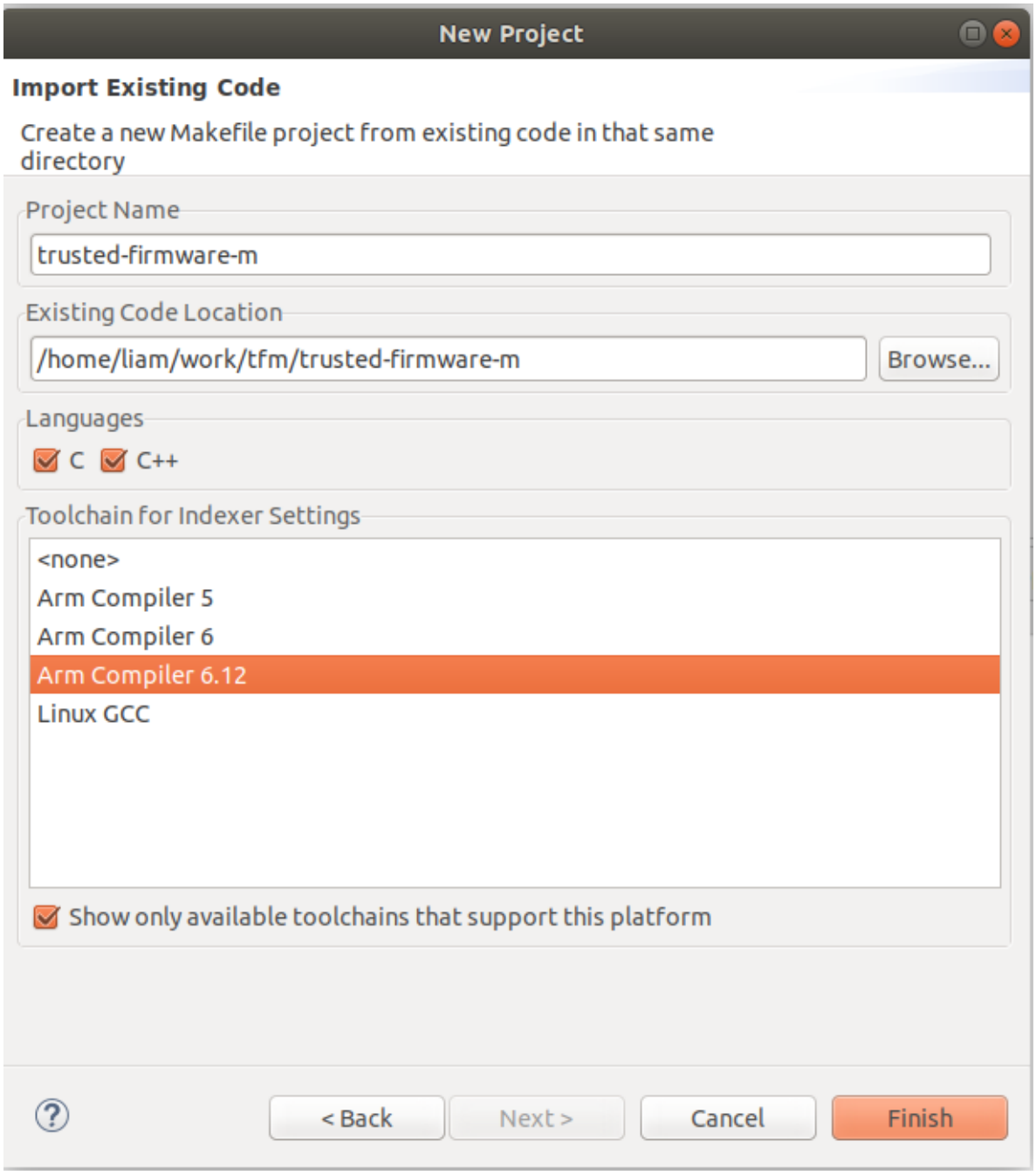


Figure 18 New Target creation

8. It is necessary to make some changes to the project to run on AWS due to the AWS instance of AN521 running faster than other platforms.

   Change the platform frequency to match the AWS target in project trusted-firmware-m by navigating to platform>ext>target>mps2>an521>cmsis_core>system_cmsdk_mps2_an521.c and editing the following defines to match below and save.

```
#define  SYSTEM_CLOCK    (62500000UL)
#define  PERIPHERAL_CLOCK     (62500000UL)
```

9. The UART BAUD rate in platform>ext>common>uart_stdout.c, needs to change to the following and save.

```
ret = TFM_DRIVER_STDIO.Control (ARM_USART_MODE_ASYNCHRONOUS, 38400);
```

10. Now the project has been updated to support AWS, the following steps create the build targets
11. Right click the project and select "Build Targets" then "Create…"
    a. On the "Create Build Target" dialogue set:
    b. The target name to: gen-<config>-<build-type>-<platform>-<compiler> where:
    c. config is the name of the TF-M configuration the target builds
    d. build-type is either debug or release
    e. platform is the name of the target platform (i.e an521)
    f. compiler is the name of the compiler to be used (i.e armclang) For example for the "Regression Test" build configuration a debug build targetting an521 compiled with armclang the name will be: "gen-regression-debug-an521-clang"
    g. Untick the "Same as target name" checkbox.
    h. Clear the "Build target" edit box.
    i. Untick the "Use builder settings" checkbox and enter the following command, then click OK

```
bash -c "{ cmake -E remove_directory build && cmake -E make_directory build && cd
build && cmake -G'Unix Makefiles' -DCMAKE_BUILD_TYPE=Debug -
DPROJ_CONFIG=<path>/tfm/trusted-firmware-m/configs/ConfigRegression.cmake  -D
TARGET_PLATFORM=AN521 -D COMPILER=ARMCLANG ../ ;}"
```

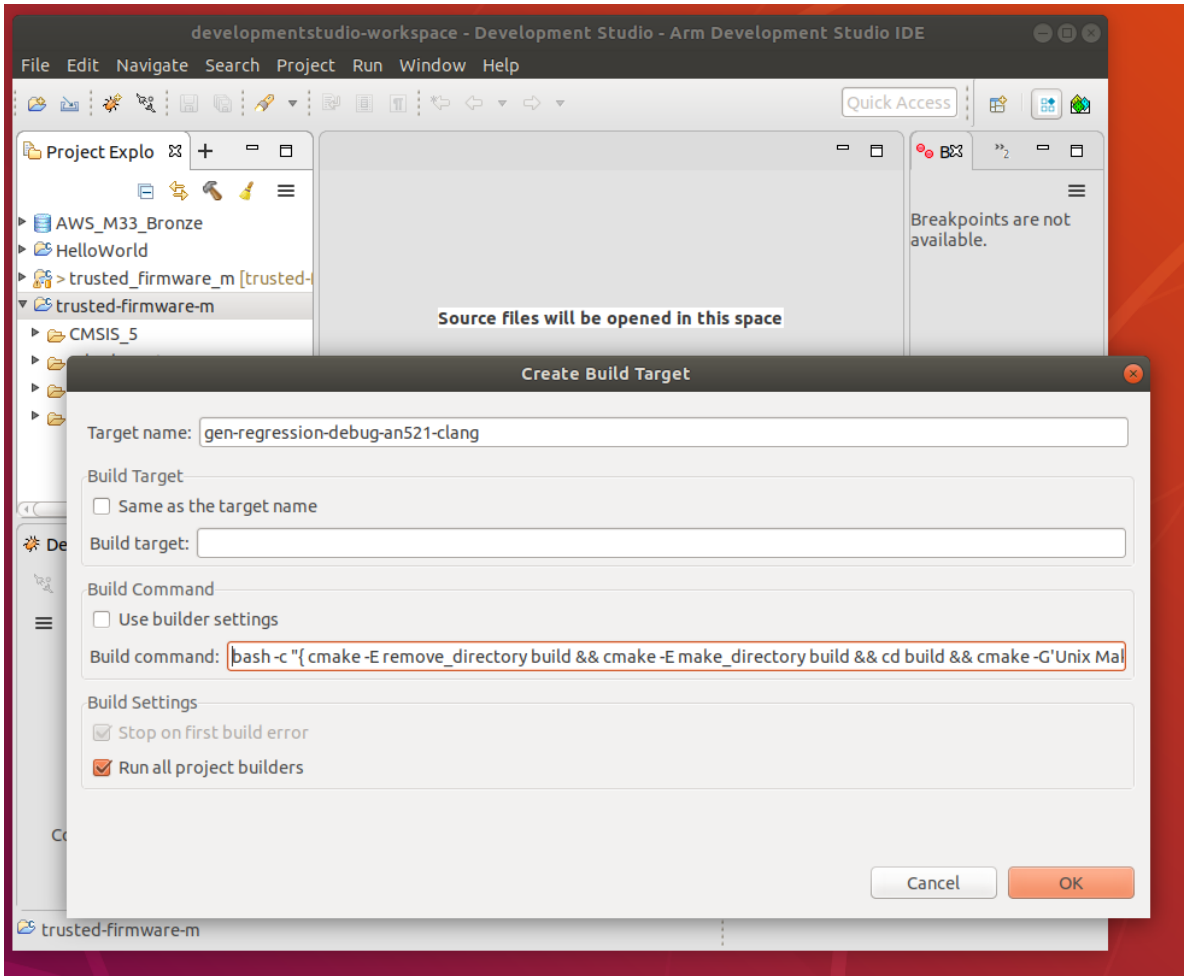


Figure 19 Build regression target

12. Repeat step 8 by right click the project and select "Build Targets" then "Create…" once complete click OK
    a. Set the "Target name" to build
    b. Clear the "Build target:" box
    c. Set the "Build command:" box to

```
cmake--build ./build
```

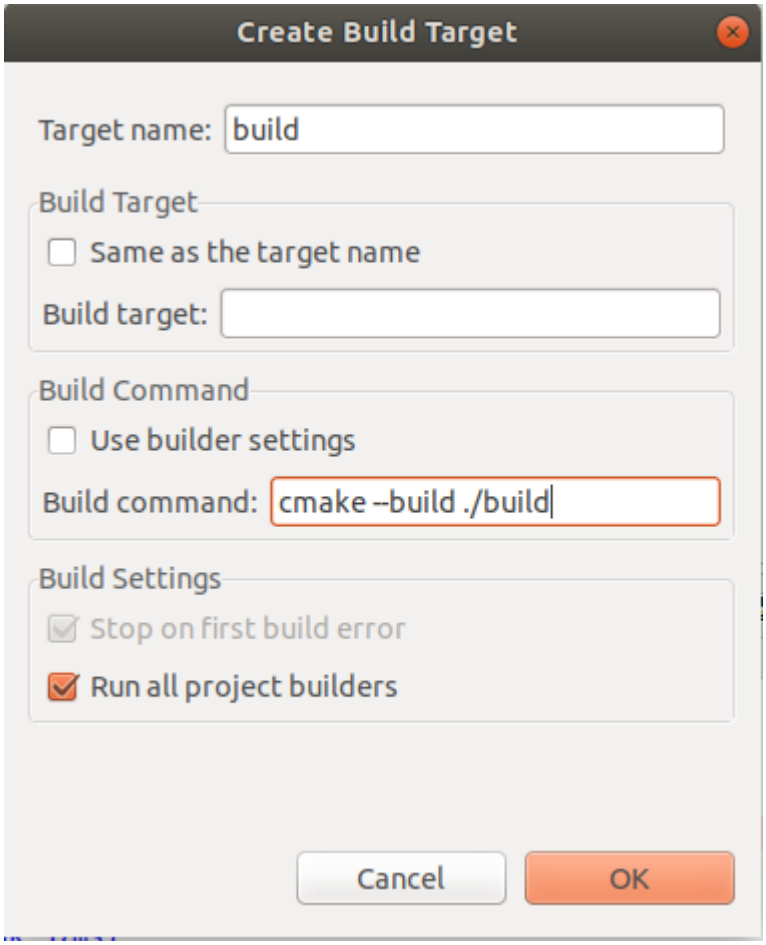


Figure 20 Create Build Target

13. Under the project "trusted-firmware-m" expand the "Build Targets" and right click **gen-regression-debug-an521-clang** target and select "Build Target"
14. Repeat step 10 this time right clicking **build** target and select "Build Target"
15. Example binaries will be created

- `mcuboot.bin (under tfm/trusted-firmware-m/build/bl2/ext/mcuboot/)`
- `tfm_sign.bin (under tfm/trusted-firmware-m/build)`

16. These binaries needed to be copied to the AWS instance and preloaded into memory, this will be detailed in section Loading and running the binaries on AWS

4.2.3 Loading and running the binaries on AWS

1. Use scp or another tool, copy the resulting binary file to the AWS instance. See below for an example running from the build directory, note the IP address will be different and will be available from "AWS Connect to Your Instance dialog box"

```
scp -i "arm_ds.pem" mcuboot.bin ec2-user@ec2-34-240-108-143.eu-west-1.compute.amazonaws.com:

scp -i "arm_ds.pem" ./bl2/ext/mcuboot/mcuboot.bin ec2-user@ ec2-34-240-108-143.eu-west-1.compute.amazonaws.com:
```

2. SSH into the instance

```
ssh -i "arm_ds.pem" ec2-user@ ec2-34-240-108-143.eu-west-1.compute.amazonaws.com
```

3. The FPGA module must be cleared and loaded with an image before we run our payload. To do this, run the following code on the AWS instance:

```
sudo fpga-clear-local-image -S 0

sudo fpga-load-local-image -S 0 -I agfi-0b63123c24cc6f8df
```

4. If the debug server is not already running, start it and put it in the background using the following commands.

```
pushd /opt/arm/developmentstudio-2019.1/sw/debughw/debug_server

sudo LD_LIBRARY_PATH=/opt/arm/developmentstudio-2019.1/sw/debughw/debug_server ./rddidap_serverd -log -rddi_dap_dll ./librddi_3rd_party_probe_aws.so -debuglog &

popd
```

5. To capture the output of the payload, open a virtual UART receiver in another terminal window with the following commands.
   a. Open a new shell and SSH in again

```
ssh -i "arm_ds.pem" ec2-user@ ec2-34-240-108-143.eu-west-1.compute.amazonaws.com
```

   b. In this shell start a virtual UART

```
sudo ~/vUART/uart 0x0 0x4 0x8 ~/uart_output 0
```

6. Switching back to the first shell, load the binary files into the FPGA with the following command. The program will begin executing immediately.

```
sudo ~/preloader/preload_software -S 0 -b mcuboot.bin 0x10000000 -b tfm_sign.bin 0x10080000
```

The output of the project is visible on the virtual UART and captured in file called "uart_output"

4.2.4 Debugging TF-M example

These steps take you through the creation of a new Arm DS debug configuration a shown in

Click **Run** > **Debug Configurations…** on the Arm DS menu bar, and create a new configuration for Generic Arm C/C++ Application.

1. Set the target to **Arm** > **AWS FPGA** > **Bare Metal Debug** > **Cortex-M33** under the Connection tab.

a. Select AWS_M33_Bronze>Bare Metal Debug>Cortex-M33

b. Set the Target Connection to AWS Virtual Debug

c. Then set the connection to:

```
server=<AWS IP>;probe=none
```

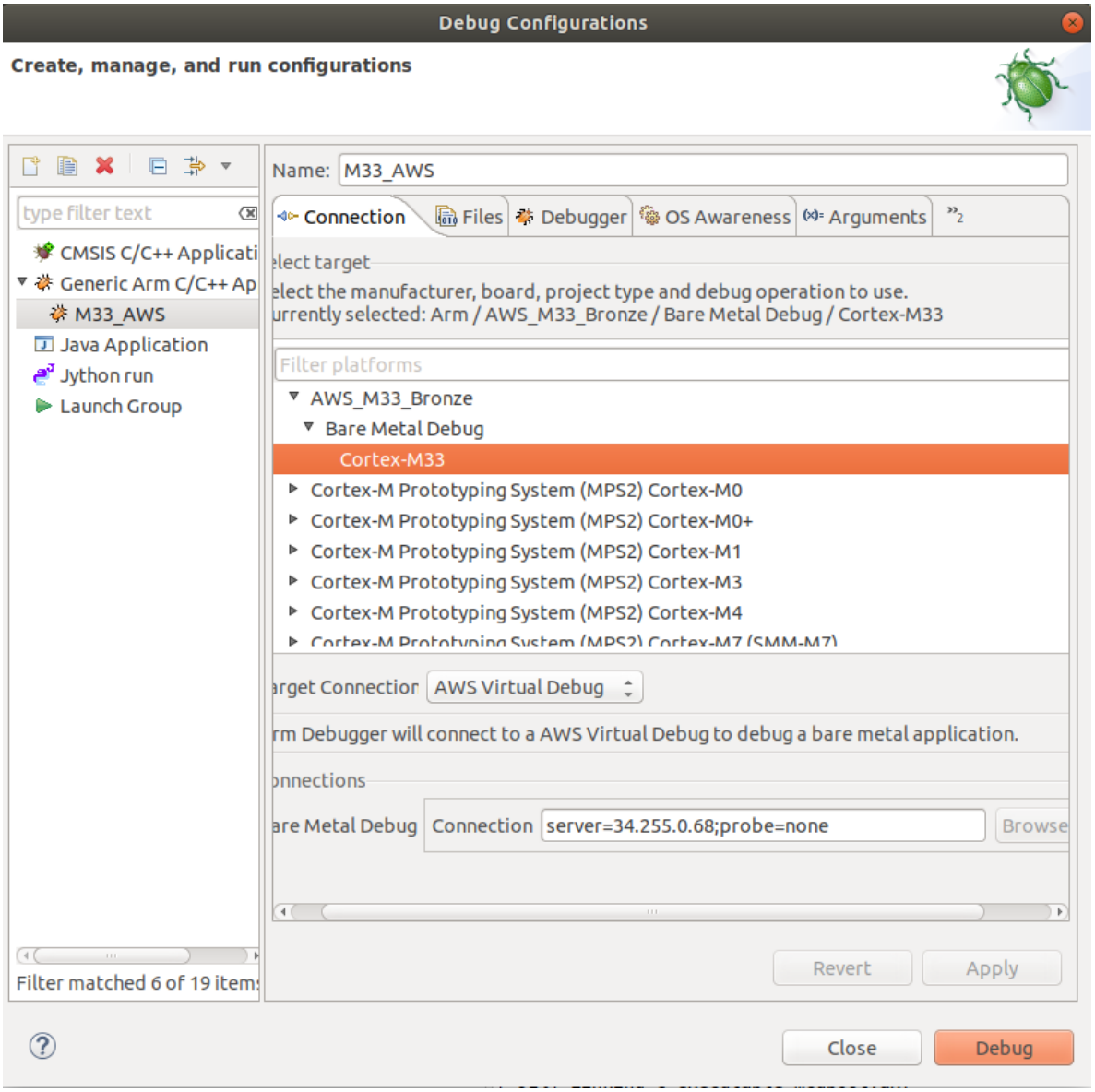


Figure 21 Debug Configuration

2. Switch to the Files tab and set the location of the debug symbols

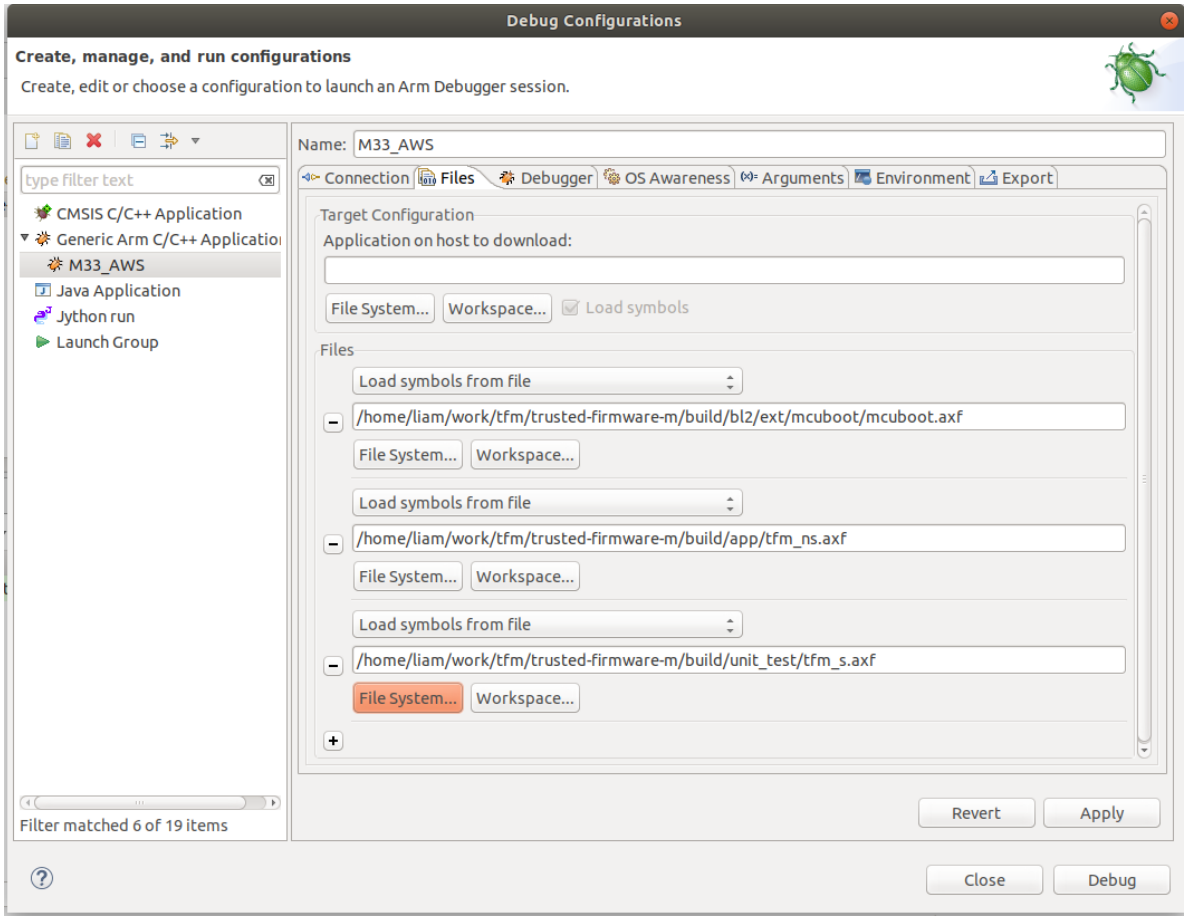


Figure 22 Debug symbols

3. Select **Debugger** tab and select **Debug from symbol:** main **as** shown in Figure 23 Debug from main and click **Debug**

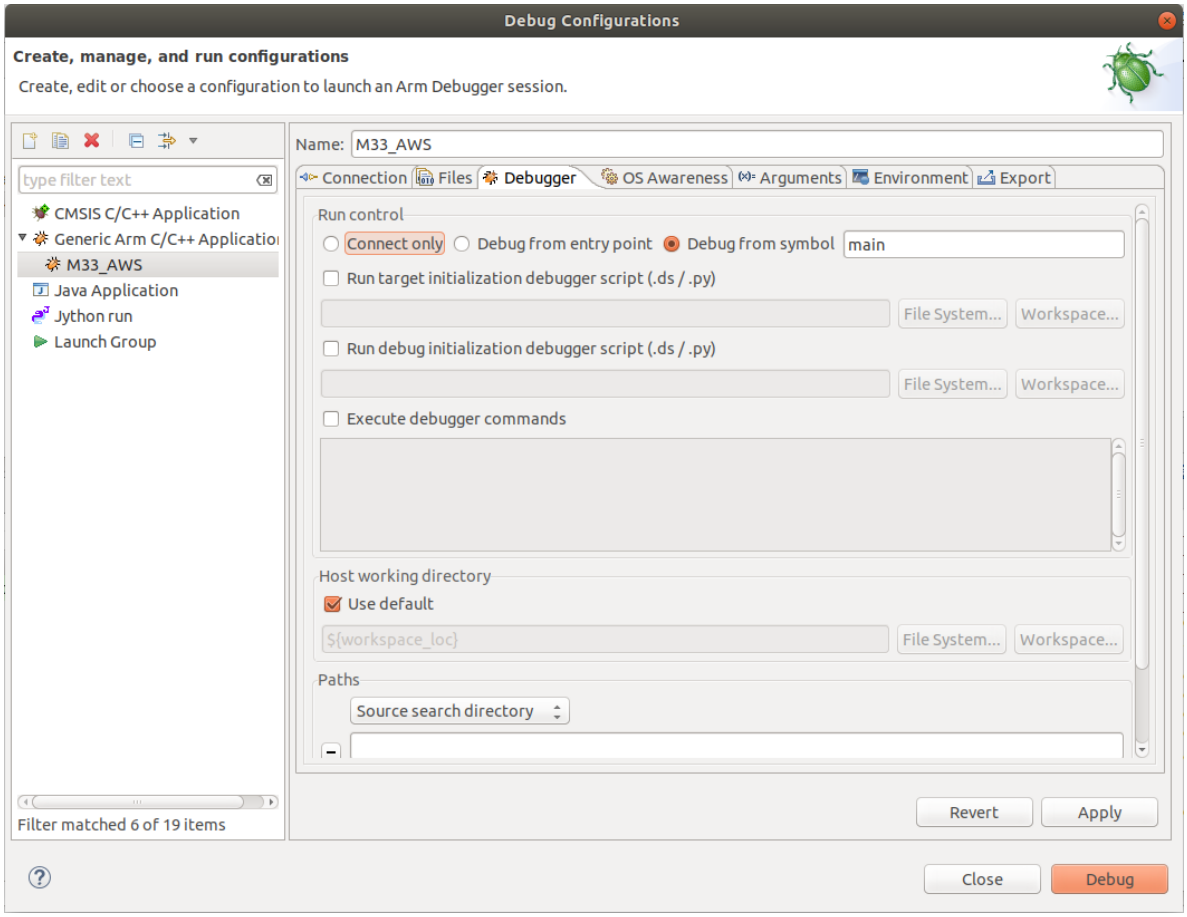


Figure 23 Debug from main

4. Now the connection is made, disassembled code, memory and registers can be viewed

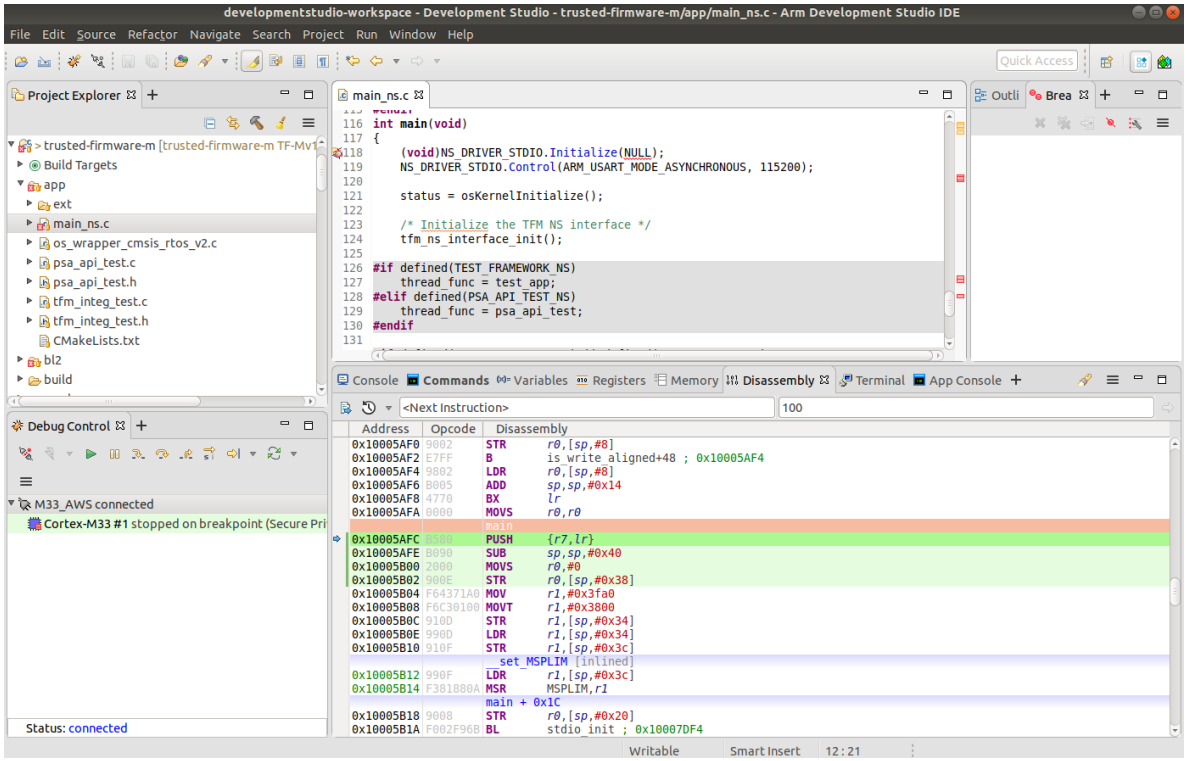


Figure 24 Debugger connection

5 Troubleshooting

5.1 How do I start debugging from main?

Insert the following code at the beginning of the `main` function under `tfm-test/[Source directory]/bl2/ext/mcuboot/bl2_main.c`:

```
{
        volatile int i = 1;
        while (i) {}
}
```

Initially, the payload will be stuck in an infinite loop. Connect the debugger and modify the value of variable `i` to `0` to resume execution.

The debugger cannot connect to the AWS server.

Check the following configurations:

- The AWS firewall must be configured to allow inbound connections to certain ports. Refer to Section 3 Setting up AWS and Arm DS.
- The debug server must be running on the AWS instance. Refer to Section 3 Setting up AWS and Arm DS.
- The IP address of the AWS instance changes across reboots. Check that the current IP address is used in the debug configuration.

5.2 The UART output on virtual UART is garbled

- Check that you have changed both the SYSTEM_CLOCK and PERIPHERAL_CLOCK in system_cmsdk_mps2_an521.c and UART BAUD rate uart_stdout.c and rebuilt the project. See Building TF-M for more information.

5.3 Can't debugging both CPUs

In order to debug both CPUs in the system, you first need to bring both CPUs out of reset, you can do this by loading a software payload as below. If you load a single payload you will not able to connect to the second CPU as it will be held in reset.

```
./preload_software -S 0 -p ./TestPayloads/single_file_test/MultiCore_Bringup.elf
```

This will print the following in the vUART

```
Cortex-M33 0
Cortex-M33 1
```

Assuming the debug server is up and running, you can use select which CPU to connect to as in Figure 10 Creating a new debug configuration.

Appendix A : Terminology

Arm DS

: Arm Development Studio

AWS

: Amazon Web Services

FPGA

: Field Programmable Gate Array

AFI

: Amazon FPGA Image

AMI

: Amazon Machine Image

TF-M

: Trusted Firmware M