Retargeting output to UART

Version 1.0

Non-Confidential

Issue 02

102440_0100_02_en

Retargeting output to UART

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	1 January 2021	Non-Confidential	Initial release

Proprietary Notice

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

This guide is the second in a collection of related guides:

- Building your first embedded image
- Retargeting output to UART (this guide)
- Creating an event-driven embedded image
- Changing Exception level and Security state in an embedded image

This guide shows you how to modify the output mechanism to use the UART capability of the target system.

In Building your first embedded image, we relied on semihosting to handle the output from our embedded image. In this guide, you will modify the output mechanism to send output to a UART serial port. This is useful to know, because embedded systems often have limited display capabilities, or no display capabilities. However, during the debug process it is often useful to be able to print diagnostic messages while a program is running.

Before you begin

To complete this guide, you will need to have Arm Development Studio Gold Edition installed. If you do not have Arm Development Studio, you can download a 30-day free trial.

Arm Development Studio Gold Edition is a professional quality tool chain developed by Arm to accelerate your first steps in Arm software development. It includes both the Arm Compiler 6 toolchain and the Base_A76x1 model used in this guide. We will use the command-line tools for most of the guide, which means that you will need to configure your environment in order to run Arm Compiler 6 from the command-line.

The individual sections of this guide contain some code examples. These code examples are available to download as a ZIP file:

- CommonTasks-RetargetToUART.zip

2. About semihosting

Semihosting enables code running on a target system, the model, to interface with a debugger running on a host system, the computer, and to use its input and output (I/O) facilities. This means that you can interact with a model or microcontroller that may not possess I/O functionality.

In Building your first embedded image, we used a `printf()` call in the code to display the "Hello World" message. This `printf()` call triggers a request to a connected debugger through the library function `_sys_write`. To see how this works, you can use `fromelf` to inspect the compiled code, as shown in the following instruction:

```
$ fromelf --text -c __image.axf --output=disasm.txt
```

This command generates a disassembly of `__image.axf` in the file `disasm.txt`. Within the disassembly, look at `_sys_write`, which contains a HLT instruction:

```
_sys_write
    0x00003a74:    d100c3ff    ....    SUB     sp,sp,#0x30
    0x00003a78:    a9027bfd    .{..    STP     x29,x30,[sp,#0x20]
    ...
    0x00003a9c:    d45e0000    ..^.    HLT     #0xf000
    ...
    0x00003aa8:    d65f03c0    .._.    RET
```

The debugger detects this halt as a semihosting operation, and interprets the `_sys_write` as a request to output to the console.

You can check if you are using semihosting by adding `__asm(".global __use_no_semihosting\n\t");` to `main()`. Linking the image will now throw an error for any functions that use semihosting.

3. Retarget functions to use UART

Real embedded systems operate without sophisticated debuggers, but many library functions depend on semihosting. You must modify, or retarget, these functions to use the hardware of the target instead of the host system.

To retarget `printf()` to use the PL011 UART of the model:

1. Write a driver for the UART. Copy and paste the following code into a new file with the filename `pl011_uart.c`:

```
struct pl011_uart {
    volatile unsigned int UARTDR;        // +0x00
    volatile unsigned int UARTECR;       // +0x04
    const volatile unsigned int unused0[4];    // +0x08 to +0x14 reserved
    const volatile unsigned int UARTFR;        // +0x18 - RO
    const volatile unsigned int unused1;       // +0x1C reserved
    volatile unsigned int UARTILPR;      // +0x20
    volatile unsigned int UARTIBRD;      // +0x24
    volatile unsigned int UARTFBRD;      // +0x28
    volatile unsigned int UARTLCR_H;     // +0x2C
   volatile unsigned int UARTCR;        // +0x30
    volatile unsigned int UARTIFLS;      // +0x34
    volatile unsigned int UARTIMSC;      // +0x38
    const volatile unsigned int UARTRIS;       // +0x3C - RO
    const volatile unsigned int UARTMIS;       // +0x40 - RO
    volatile unsigned int UARTICR;       // +0x44 - WO
    volatile unsigned int UARTDMACR;     // +0x48
};

// Instance of the dual timer
struct pl011_uart* uart;

// ---------------------------------------------------------------
void uartInit(void* addr) {
    uart = (struct pl011_uart*) addr;
    // Ensure UART is disabled
    uart->UARTCR  = 0x0;
    // Set UART 0 Registers
    uart->UARTECR   = 0x0;  // Clear the receive status (i.e. error) register
    uart->UARTLCR_H = 0x0 | PL011_LCR_WORD_LENGTH_8 | PL011_LCR_FIFO_DISABLE |
 PL011_LCR_ONE_STOP_BIT | PL011_LCR_PARITY_DISABLE | PL011_LCR_BREAK_DISABLE;
    uart->UARTIBRD = PL011_IBRD_DIV_38400;
    uart->UARTFBRD = PL011_FBRD_DIV_38400;
    uart->UARTIMSC = 0x0;                     // Mask out all UART interrupts
    uart->UARTICR  = PL011_ICR_CLR_ALL_IRQS;  // Clear interrupts
    uart->UARTCR  = 0x0 | PL011_CR_UART_ENABLE | PL011_CR_TX_ENABLE |
 PL011_CR_RX_ENABLE;
    return;
}

// ---------------------------------------------------------------
int fputc(int c, FILE *f) {
    // Wait until FIFO or TX register has space
    while ((uart->UARTFR & PL011_FR_TXFF_FLAG) != 0x0) {}
    // Write packet into FIFO/tx register
    uart->UARTDR = c;
    // Model requires us to manually send a carriage return
    if ((char)c == '\n') {
        while ((uart->UARTFR & PL011_FR_TXFF_FLAG) != 0x0){}
```

```
        uart->UARTDR = '\r';
    }
   return 0;
}
```

2. Modify `hello_world.c` to use the UART driver, so that the updated file contains:

```
#include <stdio.h>
#include "pl011_uart.h"

int main (void) {
  uartInit((void*)(0x1C090000));
  printf("hello world\n");
  return 0;
}
```

   By redefining `fputc()` to use the UART you have retargeted `printf()`. This is because `printf()` ultimately calls `fputc()`.

3. Rebuild the image:

```
$ armclang -c -g --target=aarch64-arm-none-eabi startup.s
$ armclang -c -g --target=aarch64-arm-none-eabi hello_world.c
$ armclang -c -g --target=aarch64-arm-none-eabi pl011_uart.c
$ armlink --scatter=scatter.txt --entry=start64 startup.o pl011_uart.o
 hello_world.o
```

4. Disassemble the image:

```
$ fromelf --text -c __image.axf --output=disasm.txt
```

   The disassembly in `disasm.txt` now shows no calls to `_sys_write` (although other semihosting functions such as `_sys_exit` will be present).

4. Use Telnet to inteface with the UART

All output is now directed to the model's UART serial port.

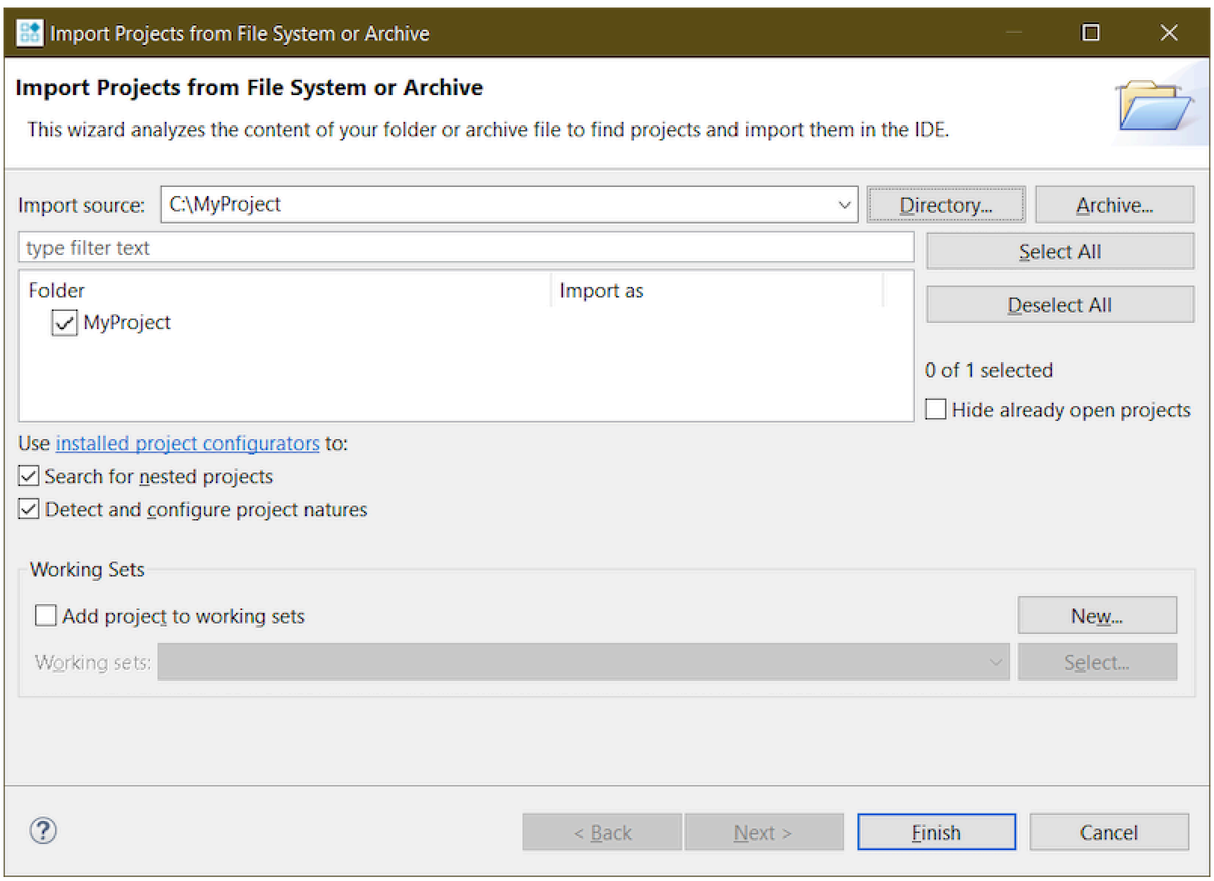
To see this output, we are going to use a Telnet client to connect to the UART. We will use Arm Development Studio to help here, because it automatically starts the Telnet client and connects it to the model.

Note If you want to start a Telnet client and connect to the model manually, you will need to use port 5000 instead of the default port 23. Timing the connection can be difficult, because you must start the client just before the server in the model starts listening.

To interface with the UART using Telnet:

1. Import your executable into Arm Development Studio. Click File > Open Projects from File System...
2. Click Directory in the Import Projects from File System or Archive dialog to select the folder containing your executable, as this screenshot shows:

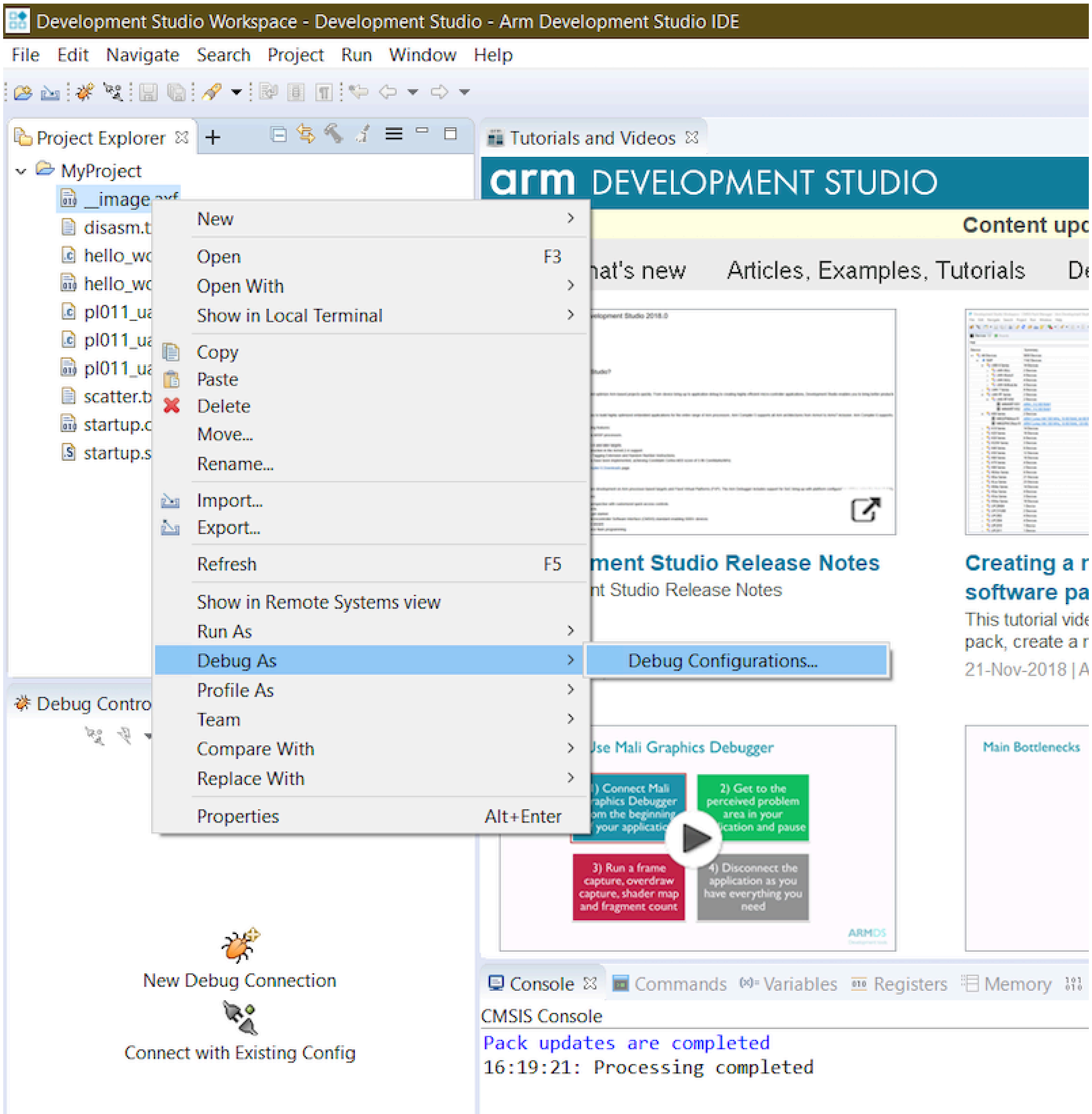
Figure 4-1: Import Projects from File System or Archive



3. Click Finish. Your project files should appear in the Project Explorer tab.

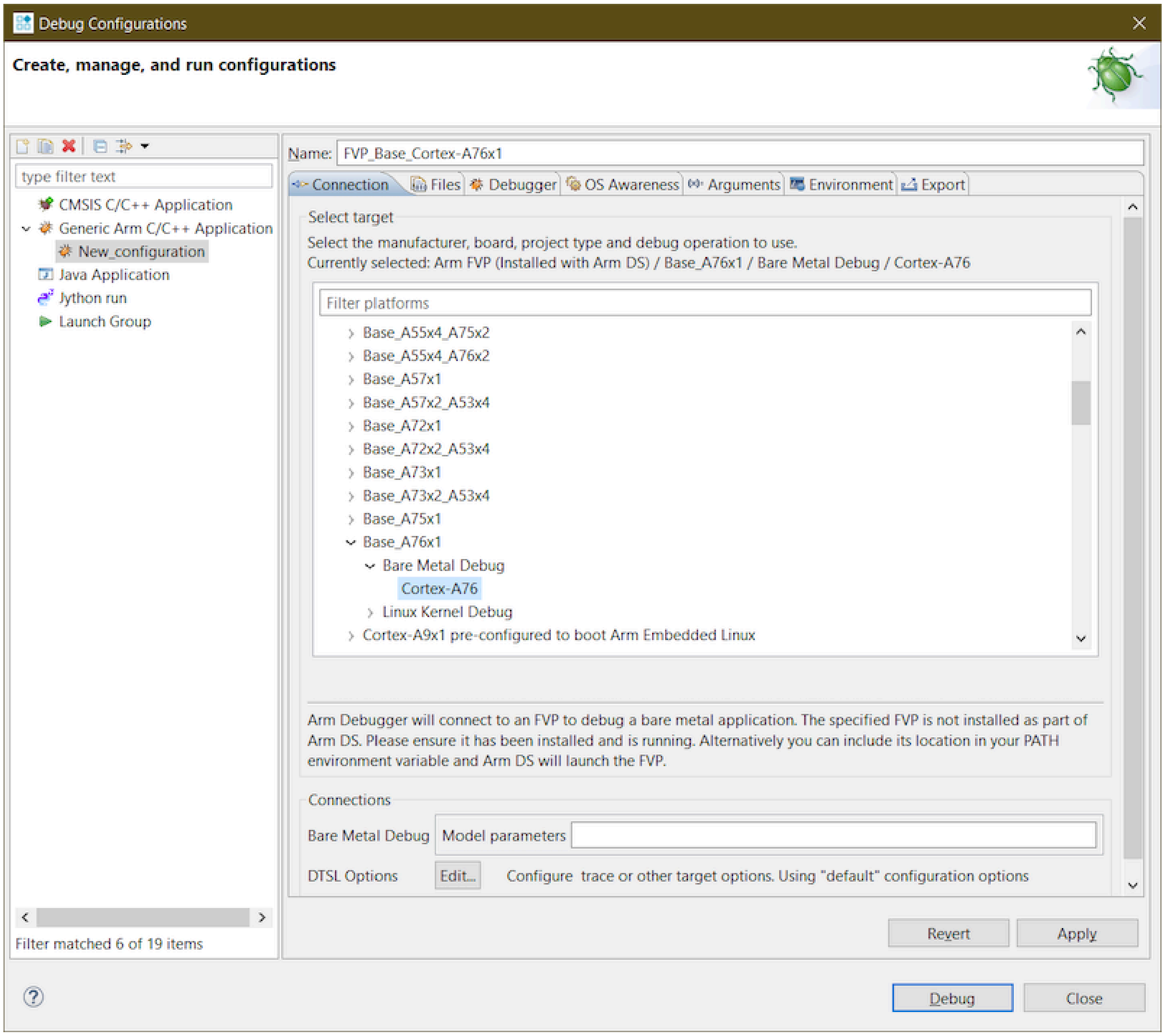
4. Right-click the `__image.axf` file, then select Debug As > Debug Configurations to display the Debug Configuration dialog box. You can see the dialog box in the following screenshot:

Figure 4-2: Opening Debug Configurations



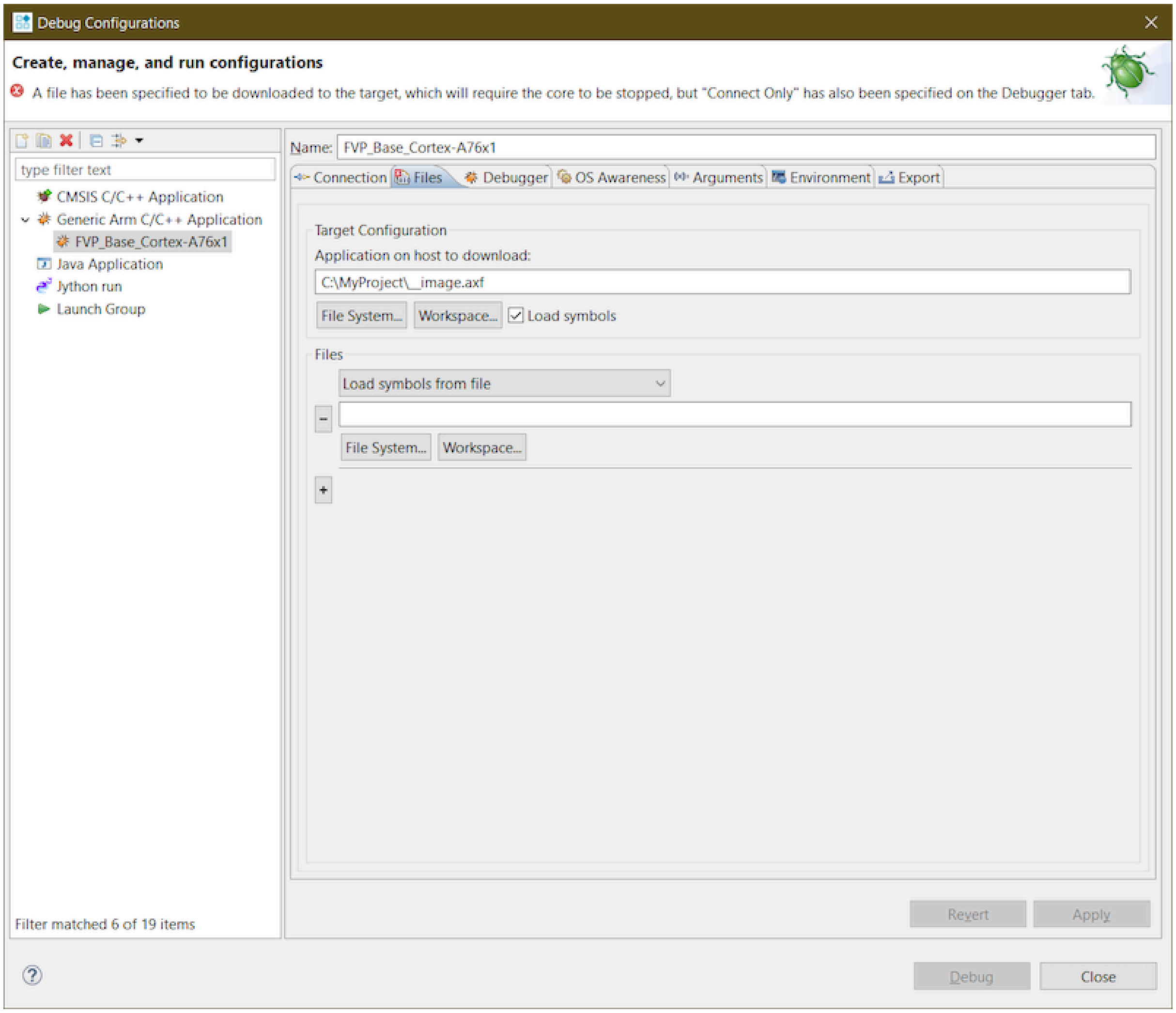
5. Select Generic Arm C/C++ Application, then click the New launch configuration button to create a new debug configuration.
6. In the Name field, give your debug configuration a name, for example `FVP_Base_Cortex-A76x1`.
7. On the Connection tab select ARM FVP (Installed with Arm DS) > Base_A76x1 > Bare Metal Debug > Cortex-A76, as this screenshot shows:

Figure 4-3: Debug Configurations - selecting a connection



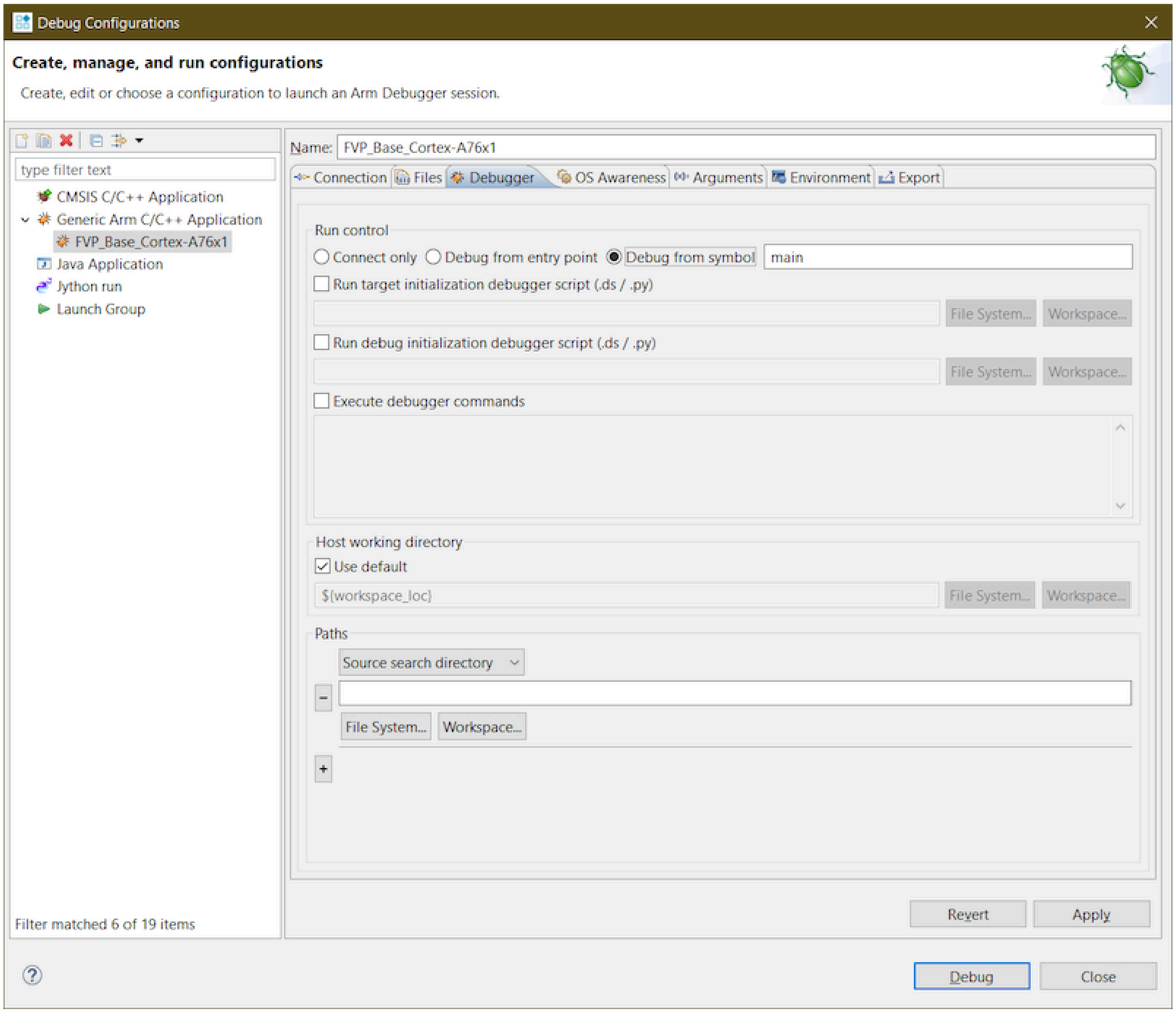
8. On the Files tab, click File System and select your `__image.axf` file, as this screenshot shows:

Figure 4-4: Debug Configurations - selecting an application on host to download



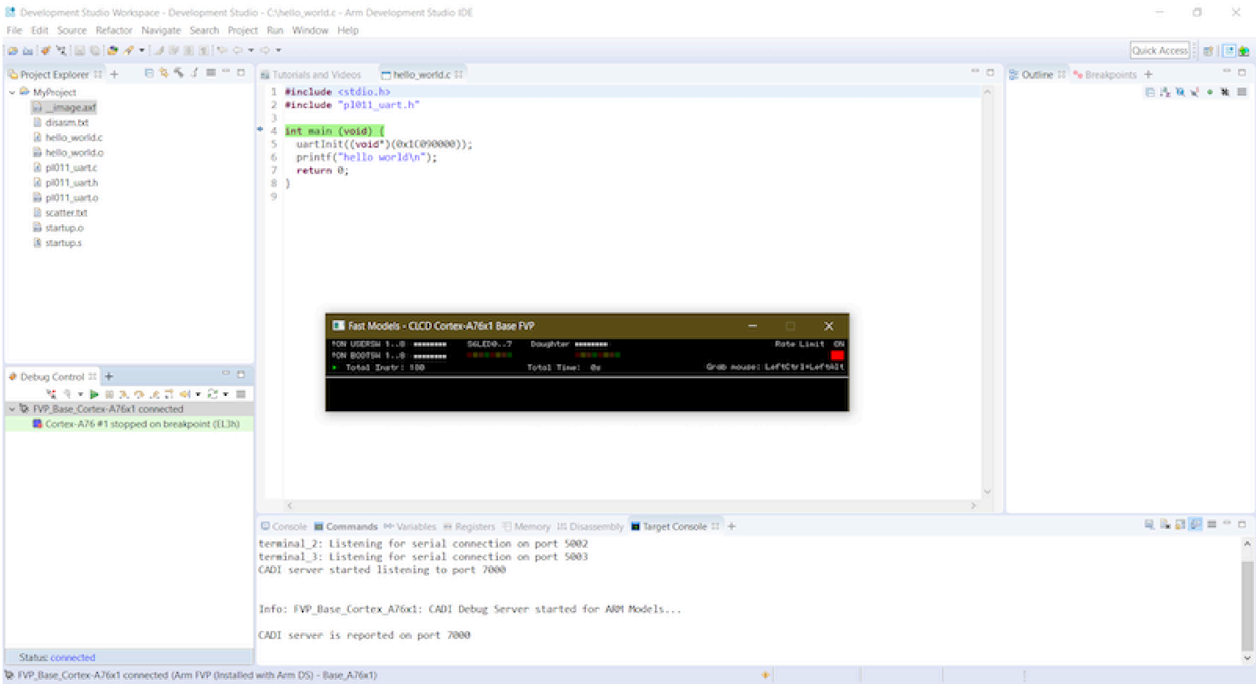
9. On the Debugger tab, select Debug from symbol: main as this screenshot shows:

Figure 4-5: Debug Configurations - debugging from symbol



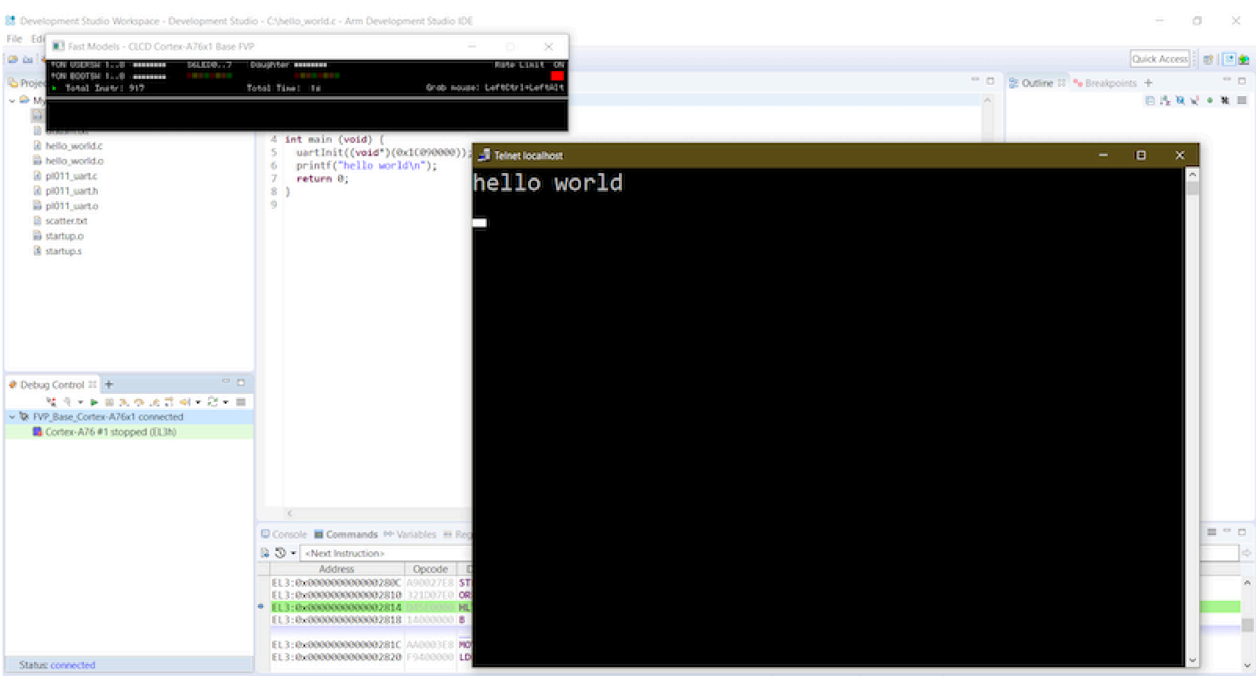
10. Click Apply, then Close.
11. On the Debug Control tab, double-click your debug configuration to start the model and run your application. Execution pauses on entry to `main()`, and you should see the Fast Models window appear, as this screenshot shows:

Figure 4-6: Start the model and run your appliaction



1. Click the Continue toolbar button to continue execution. Arm Development Studio automatically starts the Telnet client and connects to the model. The application output, hello world, will appear in the Telnet client window after it has been sent over the UART serial interface, as this screenshot shows:

Figure 4-7: Telnet client showing UART output



5. Related information

Here are some resources related to material in this guide:

- Arm Community
- Arm Compiler 6 documentation
- Arm Development Studio downloads
- Arm Development Studio documentation
- Armv8-a Learn the Architecture series of guides

6. Next steps

This guide is the second in a series of four guides on the topic of building an embedded image. In this guide, you learned about semihosting, how to retarget functions to use UART and you to use Telnet to interface with the UART.

You can continue learning about building an embedded image in the next guides in the series:

- Creating an Event-Driven Embedded Image
- Changing Exception Level and Security State in an Embedded Image

In case you missed it, the first guide in the series is:

- Building your First Embedded Image