# arm

### **Retargeting output to UART**

Version 1.0

Non-Confidential

Copyright  $\ensuremath{\mathbb{C}}$  2021 Arm Limited (or its affiliates). All rights reserved.

**Issue 02** 102440\_0100\_02\_en



### Retargeting output to UART

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

### **Release information**

#### Document history

Issue	Date	Confidentiality	Change
0100-02	1 January 2021	Non-Confidential	Initial 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 <sup>™</sup> 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 © 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-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<sup>®</sup> 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. About semihosting	7
3. Retarget functions to use UART	8
4. Use Telnet to inteface with the UART	10
5. Related information	17
6. Next steps	

# 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:

	03a74: 03a78:	d100c3ff a9027bfd	••••	SUB STP	sp,sp,#0x30 x29,x30,[sp,#0x20]
0x000	03a9c:	d45e0000	••^•	HLT	#0xf000
0x000	03aa8:	d65f03c0	··_·	RET	

The debugger detects this halt as a semihosting operation, and interprets the <u>sys\_write</u> 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 PLO11 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 {
                                                            // +0x00
     volatile unsigned int UARTDR;
     volatile unsigned int UARTECR; // +0x00
volatile unsigned int UARTECR; // +0x04
     const volatile unsigned int unused0[4]; // +0x08 to +0x14 reserved
     const volatile unsigned int UARTFR; // +0x18 - R0
const volatile unsigned int unused1; // +0x1C reserved
   const volatile unsigned int unused1; //
volatile unsigned int UARTILPR; // +0x20
volatile unsigned int UARTIBRD; // +0x24
volatile unsigned int UARTFBRD; // +0x28
volatile unsigned int UARTCR; // +0x30
volatile unsigned int UARTIFLS; // +0x34
volatile unsigned int UARTIFLS; // +0x38
const 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 = 0 \times 0;
     // Set UART 0 Registers
     uart->UARTECR = 0 \times 0; // 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 = 0 \times 0;
                                                                     // 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) != 0 \times 0) {}
```

```
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:

🔝 Import Proje	cts from File System or Archive			—	
	ts from File System or Archive	e file to find projects	and import them in	the IDE.	
Import source:	C:\MyProject		~	Directory	Archive
type filter text				<u>S</u> e	lect All
Folder	-t	Import as		Des	elect All
Use <u>installed pro</u>	pject configurators to:			0 of 1 selecte	d dy open projects
Add projec	t to working sets				Ne <u>w</u>
Working sets:				~	S <u>e</u> lect
?		< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

#### 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 <u>\_\_image.axf</u> 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

		io Workspace - Development Stud Search Project Run Window		lopment Studio IDE			
_							
Project Explore				and Videos 🖾			
✓ ➢ MyProject							
□ _image	avf			DEVELOPM	NENT STUDIC	)	
📄 disasm.t		New	>			Content	upc
le hello_wo		Open	F3	hat's new Ar	ticles, Examples,	Tutorials	De
hello_wo		Open With	>				
pi011_ua		Show in Local Terminal	>	velopment Studio 2018.0		The last fragment from the state of the stat	
🗟 pl011_ua		Copy Paste		hude?		Anne Service - Contract Mitches - Martine - Martine - Contract Victors - Contract	
scatter.b	~	Delete		a almo to backand with "an box ing all words that nois and	And non-collide galaxies. See galaxies hade easier as is trajied a polari-	Constant Show	
startup.c		Move		<ul> <li>I - Mar apply spherical engeneric applications for the online couple of the processed, for it couples applications of the online couple of the processed.</li> <li>I - Marcola - Ma</li></ul>	de l'aggert d'un proposition fon des la confidence de Congle l'aggerte	Balling and Annual States	
Is startup.s		Rename		Proper in const 2 - appel Proper instances at species from an accurate the constraint approximate principal content const at 1 to content principal and 1 instances page.		- Stree - Here - Stree - Here - Stree - Here - Stree - Here - Stree - Here	
	2	Import		<ul> <li>Bendageneti A. An Aparama Sand Sagah and Tana Chair Shake Select 719. To be folgoe Regular and an antisystem part access colors.</li> <li>Bendageneti A. An Aparama Sagah Saga Sagah Sagah Sag Sagah Sagah S Sagah Sagah Sag Sagah Sagah Sag Sagah Saga</li></ul>	2	<ul> <li>Bits ten</li> </ul>	
	2	Export		Excende Maar Infer (NOI) dealer nating NOI dealer Inter Excent	0		
		Refresh	F5	ment Studio Re		Creating	
		Show in Remote Systems view		nt Studio Release N	otes	software	
		Run As	>	1		This tutoria pack, creat	
		Debug As	>	Debug Configu	irations	21-Nov-201	18   A
💸 Debug Contro		Profile As Team	>				
R. 4 -		Compare With	· · ·	Jse Mali Graphics Debug	zzer	Main Bottler	necks
		Replace With	>		t to the		
		Properties	Alt+Enter	raphics Debugger perceive om the beginning area	id problem in your n and pause		
		<del>i k</del>		3) Run a frame capture, overdraw capture, shader map and fragment count	onnect the lon as you rything you eed		
N	lew	Debug Connection	Console	🛿 🗖 Commands 🕅	🕬 Variables 🔟 Registe	ers 🗏 Memory	101
		20	CMSIS Conso		g		
Cor	nec	t with Existing Config		tes are complet Processing com			

- 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:



) 🗈 🗶 🖻 🕸 🗸	Name: FVP Base_Cortex-A76x1	
ype filter text	Connection & Files * Debugger & OS Awareness & Arguments = Environment	
<ul> <li>CMSIS C/C++ Application</li> <li>Generic Arm C/C++ Application</li> <li>New_configuration</li> <li>Java Application</li> </ul>	Select target Select the manufacturer, board, project type and debug operation to use. Currently selected: Arm FVP (Installed with Arm DS) / Base_A76x1 / Bare Metal Debug / Cortex-A76	
a Jython run	Filter platforms	
Launch Group	<ul> <li>&gt; Base_A55x4_A75x2</li> <li>&gt; Base_A55x4_A76x2</li> <li>&gt; Base_A57x1</li> <li>&gt; Base_A57x2_A53x4</li> <li>&gt; Base_A72x1</li> <li>&gt; Base_A72x1</li> <li>&gt; Base_A72x1</li> <li>&gt; Base_A73x1</li> <li>&gt; Base_A73x1</li> <li>&gt; Base_A73x2_A53x4</li> <li>&gt; Base_A75x1</li> <li>~ Base_A76x1</li> <li>~ Bare Metal Debug</li> <li>Cortex-A76</li> <li>&gt; Linux Kernel Debug</li> <li>&gt; Cortex-A9x1 pre-configured to boot Arm Embedded Linux</li> </ul>	
	Arm Debugger will connect to an FVP to debug a bare metal application. The specified FVP is not installed as part of Arm DS. Please ensure it has been installed and is running. Alternatively you can include its location in your PATH environment variable and Arm DS will launch the FVP.	of
	Bare Metal Debug Model parameters	
	DTSL Options Edit Configure trace or other target options. Using "default" configuration options	

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

Debug Configurations     Create, manage, and run configur     A file has been specified to be downly	ations baded to the target, which will require the core to be stopped, but "Connect Only" has also been specified on the Debugger tab.
<ul> <li>A file has been specified to be down</li> <li>Type filter text</li> <li>CMSIS C/C++ Application</li> <li>CMSIS C/C++ Application</li> <li>FVP_Base_Cortex-A76x1</li> <li>Java Application</li> <li>Jython run</li> <li>Launch Group</li> </ul>	Name: FVP_Base_Cortex-A76x1  Connection Target Configuration Application on host to download: CtMyProject_image.axf File System
Filter matched 6 of 19 items	Re <u>v</u> ert Apply
0	Debug Close

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

### Figure 4-5: Debug Configurations - debugging from symbol

Debug Configurations		
reate, manage, and run configur Create, edit or choose a configuration to		Ŕ
1 III X III	Name: FVP_Base_Cortex-A76x1	
ype filter text		
<ul> <li>CMSIS C/C++ Application</li> <li>Generic Arm C/C++ Application</li> <li>FVP_Base_Cortex-A76x1</li> <li>Java Application</li> <li>Jython run</li> <li>Launch Group</li> </ul>	Connection i Files & Debugger OS Awareness M Arguments E Environment A Export      Run control      Connect only O Debug from entry point O Debug from symbol main      Run target initialization debugger script (.ds / .py)      File System      Run debug initialization debugger script (.ds / .py)      File System      Execute debugger commands	
	Host working directory Use default \$(workspace_loc) Paths	Workspace
	Source search directory ~ - File System Workspace +	
ilter matched 6 of 19 items	Reyert	Apply
?	Debug	Close

- 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

Ed Fast Models - CLCD Cortex-A76x1 Base P	VP	- 0 x		
TON USERSA 1	Daughter meaning	Rate Linit ON		Quick Access
+ON BOOTSM 10				St Outline 12 % Breakpoints +
	Total Time: 1s Grab	Nouse: LeftCtrl+LeftNlt		
My			<u>^</u>	□ 1 <sup>2</sup> R x <sup>2</sup> + 8
and the second se				
R hello world.c	4 int main (void) (			
ia helio_world.o	5 uartInit((void*)(0x1C0900)	🕬)); 🚽 Teinet localhost		- 🗉 🗙
helio_wond.o     ji011_uart.c	<pre>6 printf("hello world\n");</pre>			
a pi011_uarth	7 return 0;	hello world		
	8)			
pi011_uart.o	3			
scatter.bit				
startup.o				
startup.s				
bug Control 11 + C				
ouy consol in T				
N 1 + F 8 2 9 2 1 4 + P F				
FVP_Base_Cortex-A76x1 connected				
Cortex-A76 #1 stopped (EL3h)				
	<			
	Console Commands IN Variables	11 Page		-
		a neg		
	S Vext Instruction>			
	Address Opcode			
	EL3:0x00000000000280C A90027E			
	EL3:0x000000000002810 321007E	O OR		
	<ul> <li>EL3:0x00000000002814</li> </ul>	HL.		
	EL3:0x000000000002818 1400000	0.8		
	EL3:0x00000000000281C AA0003E			
	EL3:0x000000000000281C A40003E			
tus: connected				

Copyright  $\ensuremath{\mathbb{C}}$  2021 Arm Limited (or its affiliates). All rights reserved. Non-Confidential

# 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