Application Note

Using RTT on Cortex-A/R based devices

Document: AN08005
Software Version: 1.00
Revision: 1
Date: February 15, 2016

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Copyright notice

Trademarks

Contact address

SEGGER Microcontroller GmbH & Co. KG

In den Weiden 11
D-40721 Hilden

Germany

Tel.	+49 2103-2878-0
Fax.	+49 2103-2878-28
E-mail:	`support@segger.com`
Internet:	`www.segger.com`

Manual versions

If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: February 15, 2016

Revision	Date	By	Description
1	160215	AG	Updated references to J-Link manual.
0	151109	AG	Initial Version.

Table of contents

Chapter 1

Using RTT on Cortex-A/R based target devices

This application note describes how to use RTT on Cortex-A/R based target devices. It is assumed that it is already known what SEGGER Real Time Transfer (RTT) is and how it works in general.

For more information about RTT in general, please refer to https://www.segger.com/jlink-rtt.html

1.1 Introduction

In general, the RTT technology is target CPU core independent but some requirements need to be fulfilled to allow RTT to be used on a certain CPU core. While it is possible to use RTT on any ARM Cortex-M based device, for Cortex-A/R things are slightly different. In the following, the requirements as well as limitations that need to be taken into consideration when using RTT on Cortex-A/R based target devices, are explained.

1.2 Background memory access - Target support

In order to use RTT, the target needs to support background memory accesses, meaning J-Link needs to be able to read & write memory while the CPU is running / target application is executing. On Cortex based target devices, this is done via a so-called AHB-AP which is mainly a DMA that can be exclusively accessed by the debug interface. On Cortex-M, the AHB-AP is a mandatory component while for Cortex-A/R it is an optional component, so it is up to the silicon vendor to implement it on a specific device or not. Therefore for Cortex-A/R it depends on the actual device if RTT can be used. For most devices supported by J-Link, it is mentioned in the list of supported devices if background memory access is supported (AHB-AP is present):

https://www.segger.com/jlink_supported_devices.html#DeviceList

Specifying existence and location of an AHB-AP

When using a device that is listed on the page above, there is nothing special that needs to be done by the user. However, when using a device not listed there, the location of the AHB-AP need to be specified manually. For more information about how to do that from within various IDEs etc., please refer to `UM08001 (J-Link User Guide)`, chapter `RTT`, section `ARM Cortex - Background memory access`.

1.3 Background memory access - J-Link support

Background memory access needs to be supported by the J-Link DLL + firmware in order to use RTT on Cortex-A/R based target devices.

Background memory access in the DLL is supported since V5.10m (release) of the J-Link software.

Background memory access in the J-Link firmware is supported in all current J-Link models (Feb. 2016). In case of doubt, please get in touch with `support@segger.com`

1.4 MMU, Caches and Write-buffers

As previously mentioned, AHB-APs are comparable to DMAs. This means memory accesses through the AHB-AP while the CPU is running, bypass all MMUs, caches and write-buffers that may be present on the device. So if a device, with one of the following components being present, is used

- MPU / MMU
- D-cache
- Write-buffer

the RTT control block as well as the RTT buffers need to be placed in a memory section that is:

- Non-cacheable (Not read cachable) / Non-bufferable (not write cachable)
- Specified as strongly-ordered memory

The definition as strongly-ordered memory is also important because the CPU must not be allowed to change the order to memory accesses when accessing the RTT buffers and/ or control block. Otherwise it could for example happen that the write pointer in the RTT control block is incremented before the actual data has been written to the buffer, resulting in the PC application / J-Link DLL to read incorrect data from the buffer.

Configuring MMU in a project using embOS

The following gives and example how to configure a SEGGER Embedded Studio project using embOS, for RTT on Cortex-A/R. The MMU configuration can be found in the `__low_level_init(void);` function in `$PROJ_DIR$\Setup\RTOSInit_<Device>.c`:

```
OS_INTERWORK int __low_level_init(void) {
  [...]
  //
  // Initialize MMU and caches
  //
  //
  // Init MMU and caches. This defines the virtual memory map, which is used during execution.
  // Memory mapping should be complete, meaning 4096 entries.
  // Code below fills in ascending VAddr order
  //
  OS_ARM_MMU_InitTT(_TTbl);
  //                             Mode                       VAddr  PAddr  Size[MB]
  // CS0 space, 64MB NOR Flash (16bit BUS on Eval-Board)
  OS_ARM_MMU_AddTTEntries(_TTbl, OS_ARM_CACHEMODE_C_NB,    0x000, 0x000, 0x040);
  [...]
  // Internal RAM
  OS_ARM_MMU_AddTTEntries(_TTbl, OS_ARM_CACHEMODE_C_B,     0x200, 0x200, 0x009);
  // Internal RAM, 1 MB reserved for RTT control block and RTT buffers
  OS_ARM_MMU_AddTTEntries(_TTbl, OS_ARM_CACHEMODE_NC_NB,   0x209, 0x209, 0x001);
  // Rest of 4 GB addr. space
  OS_ARM_MMU_AddTTEntries(_TTbl, OS_ARM_CACHEMODE_NC_NB,   0x20A, 0x20A, 0xFF6);
}
```

1.5 Virtual vs. Physical Addresses

For RTT on Cortex-A/R all addresses are treated as physical addresses because background memory accesses are like DMA accesses, therefore bypassing any MMU etc. It is strongly recommended to set the linkker file of the IDE accordingly to link the RTT control block as well as the RTT buffers to uncached, strongly-ordered memory.

1.6 Using RTT-block location auto-detection of J-Link

It is not recommended to use the default RTT control block auto-search functionality of the J-Link DLL for RTT on Cortex-A/R based target devices because of the following reasons:

- Data on Cortex-A/R based devices is usually located in external memory and might be quite large so auto-detection may take some time
- The auto-search algorithm of the J-Link DLL searches internal RAM only.
- External memory is usually only available after being initialized by the startup code of the target application. When using RTT control block auto-detection, J-Link may try to access the external memory before it is initialized / available which might lead to data aborts

Manually specifying RTT control block address

The address of the RTT control block can also be manually specified by the user. This is usually done via macro files / ini files that are executed from within the IDE and which pass the address to the J-Link DLL via special commands. For more information how to `UM08001 (J-Link User Guide)`, chapter `RTT`, section `Locating the Control Block`.

1.7 RTT Locking / Unlocking - Target side

By default, the macros for locking and unlocking, when using RTT functions from multiple threads/tasks on the target, are defined to be empty because it is user’s responsibility to define up to which level RTT shall lock for an RTT API call (only task switching disabled, all interrupts disabled, certain interrupt(s) disabled etc.).

The following macros need to be defined in `SEGGER_RTT_Conf.h` accordingly in the user application when using RTT in a multi-threaded context on the target:

- `SEGGER_RTT_LOCK()`
- `SEGGER_RTT_UNLOCK()`

1.8 Example project for Renesas RZ/A1H

For an example project how to use RTT with a Renesas RZ/A1H target, please refer to:

`https://wiki.segger.com/index.php?title=Using_RTT_on_RZ_A1H`