J-Link Real Time Transfer

SEGGER's J-Link Real Time Transfer overcome the limitation of other debug techniques such as Semihosting or SWO, in term of speed and availability. It can run without debugging session, and do not affect the real-time system. However, it only supported by Segger J-Link probes.

#arm #stm32 #debug #rtt

Last update: 2021-06-21 14:40:25

Table of Content

1. Real Time Transfer

Visit the Official J-Link RTT page on SEGGER website for more information.

J-Link Real Time Transfer - Manual

SEGGER's J-Link RTT utilizes the background memory access feature on Debug Access Port (DAP) on Cortex-M and RX MCUs to communicate between the MCU and the PC's host application, through J-Link probes. RTT supports multiple channels in both directions, up to the host and down to the target, which can be used for different purposes and provide the most possible freedom to the user.

The default implementation uses one channel per direction, which are meant for printable terminal input and output. With the J-Link RTT Viewer this channel can be used for multiple "virtual" terminals, allowing to print to multiple windows (e.g. one for standard output, one for error output, one for debugging output) with just one target buffer. An additional up (to host) channel can for example be used to send profiling or event tracing data.

1.1. How RTT Works

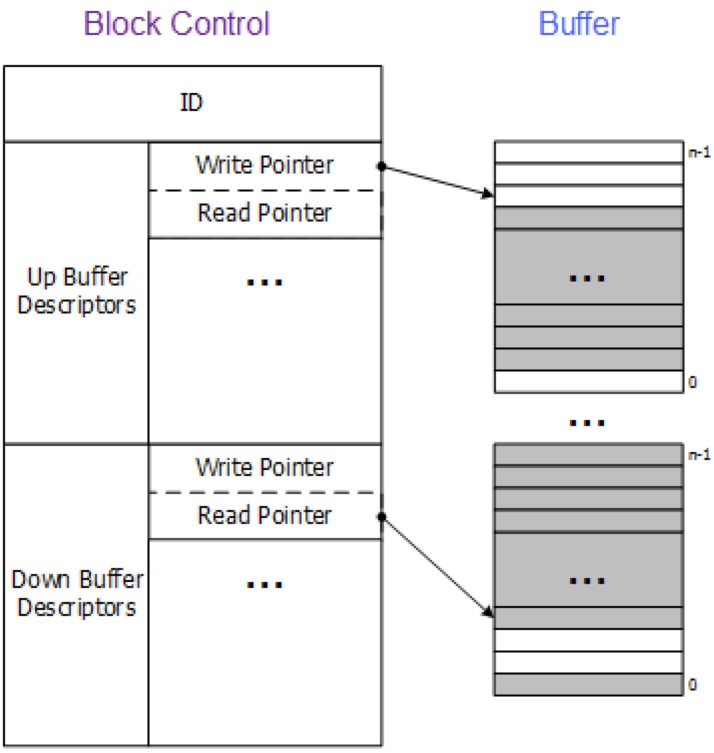
Real Time Transfer uses a SEGGER RTT Control Block structure in the target's memory to manage data reads and writes. The control block contains an ID to make it findable in memory by a connected J-Link and a ring buffer structure for each available channel, describing the channel buffer and its state.

The maximum number of available channels can be configured at compile time and each buffer can be configured and added by the application at run time. Up and down buffers can be handled separately.

Each channel can be configured to be blocking or non-blocking. In blocking mode the application will wait when the buffer is full, until all memory could be written, resulting in a blocked application state but preventing data from getting lost. In non-blocking mode only data which fits into the buffer, or none at all, will be written and the rest will be discarded. This allows running in real time, even when no debugger is connected. The developer does not have to create a special debug version and the code can stay in place in a release application.

When RTT is active on the host computer, J-Link automatically searches for the SEGGER RTT Control Block in the target's known RAM regions. The RAM regions or the specific address of the Control Block can also be set via the host applications to speed up detection or the block cannot be found automatically.

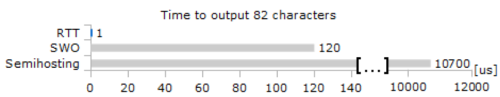
There may be any number of "Up Buffer Descriptors" (Target -> Host), as well as any number of "Down Buffer Descriptors" (Host -> Target). Each buffer size can be configured individually. The gray areas in the buffers are the areas that contain valid data. For Up buffers, the Write Pointer is written by the target, the Read Pointer is written by the debug probe (J-Link, Host). When Read and Write Pointers point to the same element, the buffer is empty. This assures there is never a race condition.



SEGGER RTT does not need any additional pin or hardware, despite a J-Link connected via the standard debug port to the target. It does not require any configuration of the target or in the debugging environment and can even be used with varying target speeds. RTT can be used in parallel to a running debug session, without intrusion, as well as without any IDE or debugger at all.

1.2. RTT Performance

The performance of SEGGER RTT is significantly higher than any other technology used to output data to a host PC. An average line of text can be output in one microsecond or less. Basically only the time to do a single `memcopy()` .



RTT Performance in comparison with Semihosting and SWO

The maximum speed at which output data can be sent to the host depends on the target buffer size and target interface speed. Even with a small target buffer of 512 Bytes an RTT speed of up to

1 MiB/s is possible with a high interface speed and 0.5 MiB/s are possible with a regular J-Link model.

1.3. RTT APIs

The SEGGER RTT implementation is written in ANSI C and can be integrated into any embedded application. RTT can be used via a simple and easy to use API. It is even possible to override the standard `printf()` functions to use RTT. Using RTT reduces the time taken for `printf()` to a minimum and allows printing debug information to the host PC, while the application is performing time critical, real time tasks.

The SEGGER RTT implementation includes a simple implementation of `printf()` which can be used to write a formatted string via RTT. `SEGGER_RTT_Printf()` is smaller than most standard library printf implementations and does not require heap and only a configurable amount of stack. However, it does not support printing double or float numbers.

Function Name	Description
`SEGGER_RTT_Read()`	Read data from an input buffer.
`SEGGER_RTT_Write()`	Write data to an output buffer.
`SEGGER_RTT_WriteString()`	Write a zero-terminated string to an output buffer.
`SEGGER_RTT_printf()`	Write a formatted string to an output buffer.
`SEGGER_RTT_GetKey()`	Get one character from input buffer 0.
`SEGGER_RTT_HasKey()`	Check if a character is available in input buffer 0.
`SEGGER_RTT_WaitKey()`	Wait for a character to be available in input buffer 0 and get it.
`SEGGER_RTT_ConfigUpBuffer()`	Configure an up (output) buffer.
`SEGGER_RTT_ConfigDownBuffer()`	Configure a down (input) buffer.
`SEGGER_RTT_Init()`	Initialize RTT Control Block structure when using RAM only targets.
`SEGGER_RTT_SetTerminal()`	Set the “virtual” Terminal to use for output on channel 0 via Write and WriteString.
`SEGGER_RTT_TerminalOut()`	Send a zero-terminated string via a “virtual” terminal.

1.4. RTT Viewer

J-Link RTT Viewer is the main Windows GUI application to use all features of RTT on the debugging host. RTT Viewer can be used stand-alone, opening an own connection to J-Link and target or in parallel to a running debug session, attaching to it and using this existing J-Link connection.

RTT Viewer supports all major features of RTT:

- Terminal output on Channel 0
- Sending text input to Channel 0
- Up to 16 virtual Terminals with only one target channel
- Controlling text output: Colored text, erasing the console
- Logging data on Channel 1

2. RTT Integration

RTT in the target MCU is provided freely. Firstly, download the J-Link software and install it.

Under the installation folder, the source code of RTT on MCU is found in `Samples\\RTT`. At the time of writing this guide, the version of J-Link is 7.20, therefore, user can find the SEGGER_RTT_V720a.zip file there.

```
| License.txt
| README.txt
|
├─RTT
|   SEGGER_RTT_Conf.h           # Configuration
|   SEGGER_RTT.h                # Main header
|   SEGGER_RTT.c                # Main implementation
|   SEGGER_RTT_printf.c         # Print functions
|   SEGGER_RTT_ASM_ARMv7M.S     # for Cortex-M3/M4
|
├─Syscalls
|   SEGGER_RTT_Syscalls_GCC.c   # redirection for GCC and newlib
|   SEGGER_RTT_Syscalls_IAR.c   # redirection for IAR
|   SEGGER_RTT_Syscalls_KEIL.c  # redirection for KEIL ARM
|   SEGGER_RTT_Syscalls_SES.c   # redirection for Segger Embedded System
|
└─Examples
    Main_RTT_InputEchoApp.c     # echo characters
    Main_RTT_MenuApp.c          # use character to select an option
    Main_RTT_PrintfTest.c       # print log with format
    Main_RTT_SpeedTestApp.c     # measure execution time
```

To integrate RTT into a project, copy the folder `RTT` from the `SEGGER_RTT_V720a.zip` to the project folder. It's recommend to put header files into `Inc` folder, and all source file into `Src` folder, as they will need to add into Build's Path and Symbols.

3. Lab: Print out with RTT

This lab will guide on how to add RTT into the Build's Path and Symbols of the project and print log on the Terminal channel 0.

3.1. Start a new project

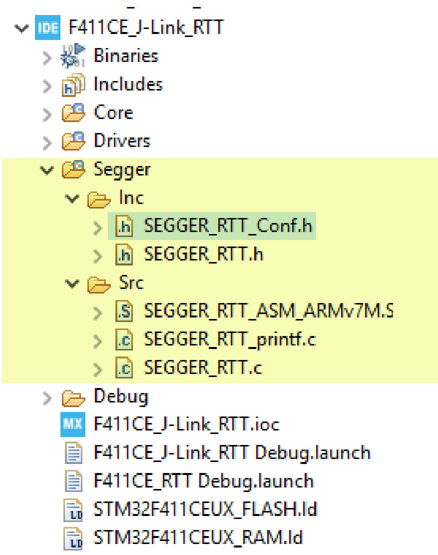
RTT supports all Cortex-M MCUs, in this lab, an F411CE MCU will be used.

To use RTT, just need to enable the SWD interface on pin **PA13** (SWDIO) and pin **PA14** (SWCLK).

3.2. Import RTT files

As mentioned above, the `RTT` source folder can be added into project like below:

```
├─Core
├─Drivers
├─Segger
| ├─Inc
| |  SEGGER_RTT_Conf.h        # Configuration
| |  SEGGER_RTT.h             # Header
| └─Src
|    SEGGER_RTT.c             # Implementation
|    SEGGER_RTT_printf.c      # Print functions
|    SEGGER_RTT_ASM_ARMv7M.S  # for Cortex-M3/M4
```



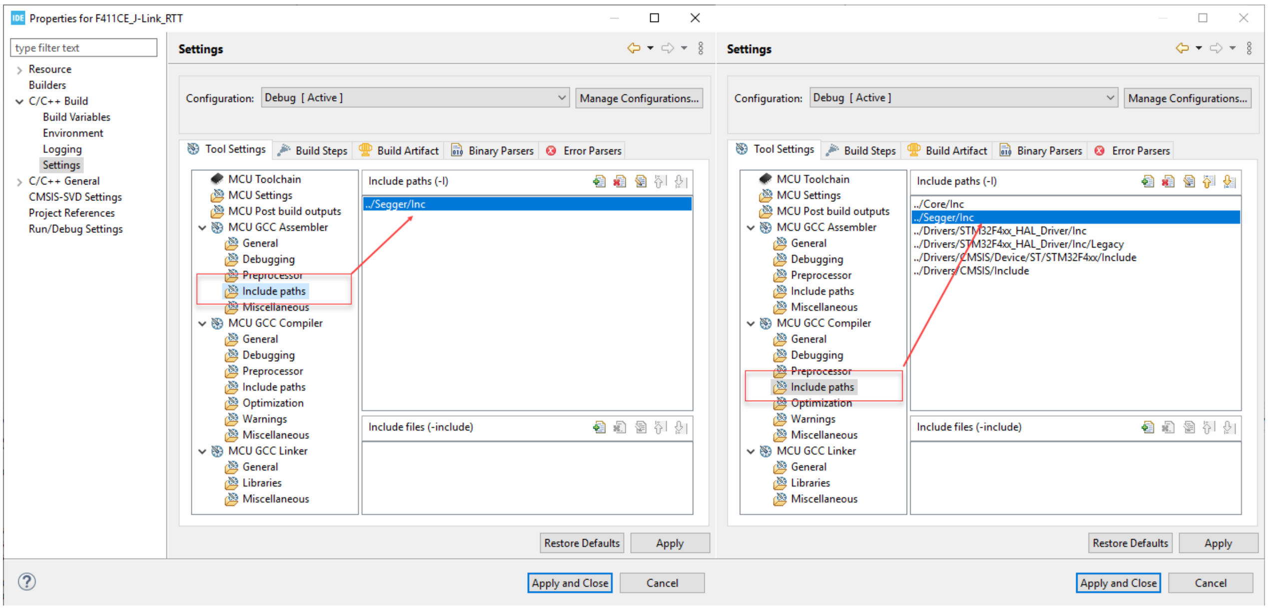
Add RTT files

Add Include Paths

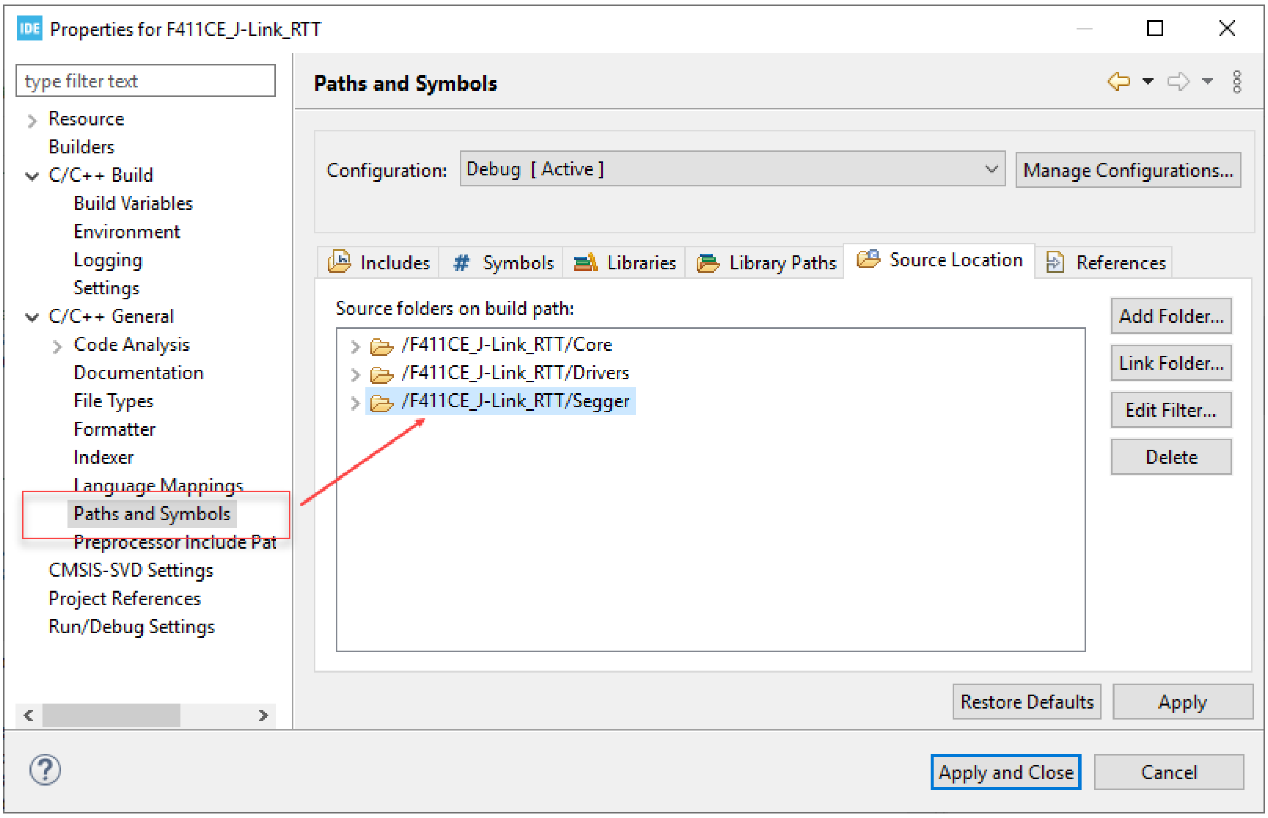
Open the **Project Properties** and select **C/C++ Build** » **Settings**. Then add `../Segger/Inc` into the include paths of both GCC Assembler and GCC Compiler

Add source files

To make RTT source files get compiled, add the folder `/Segger` into the **Source Location** list in the **Paths and Symbols** setting under the **C/C++ General** property.



Add Include path



Add source files

3.3. Add RTT print

In the `main.c` file, include the `SEGGER_RTT.h` firstly to import RTT APIs.

Inside the `int main()` function, call to `SEGGER_RTT_Init()` to initialize the SEGGER RTT Control Block and the Channel 0.

Then, it is ok to start using print API, such `SEGGER_RTT_printf()` to write strings.

main.c

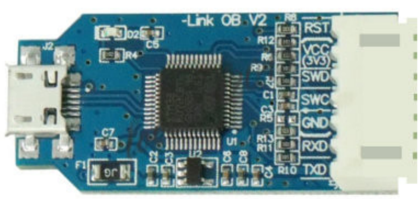
```
#include <stdio.h>
#ifdef CONFIG_USE_RTT
#include "SEGGER_RTT.h"
#endif

unsigned char counter = 0;

int main(void) {
#ifdef CONFIG_USE_RTT
    SEGGER_RTT_Init();
#endif
    while (1) {
        counter++;
#ifdef REDIRECT_TO_RTT
        SEGGER_RTT_printf(0, "S: counter = %d\r\n", counter);
#endif
        HAL_Delay(500);
    }
}
```

That's it. It is very simple to use RTT APIs. Compile and flash the firmware into the target MCU and power it up.

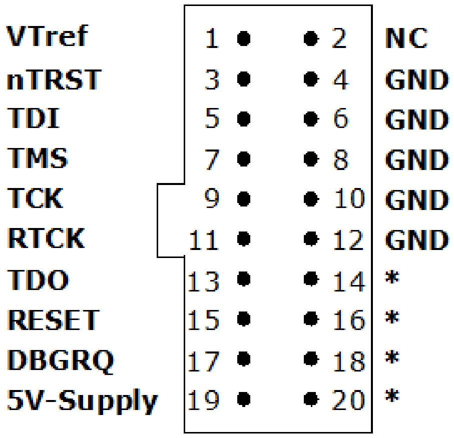
4. Start RTT Viewer

Connect any J-Link probe into the SWD interface of the target MCU. Then start the J-Link RTT Viewer in the J-Link software package.

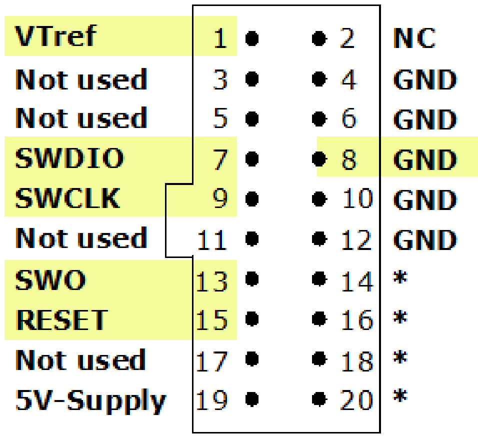
J-Link Pro



Cloned J-Link OB



J-Link JTAG connection

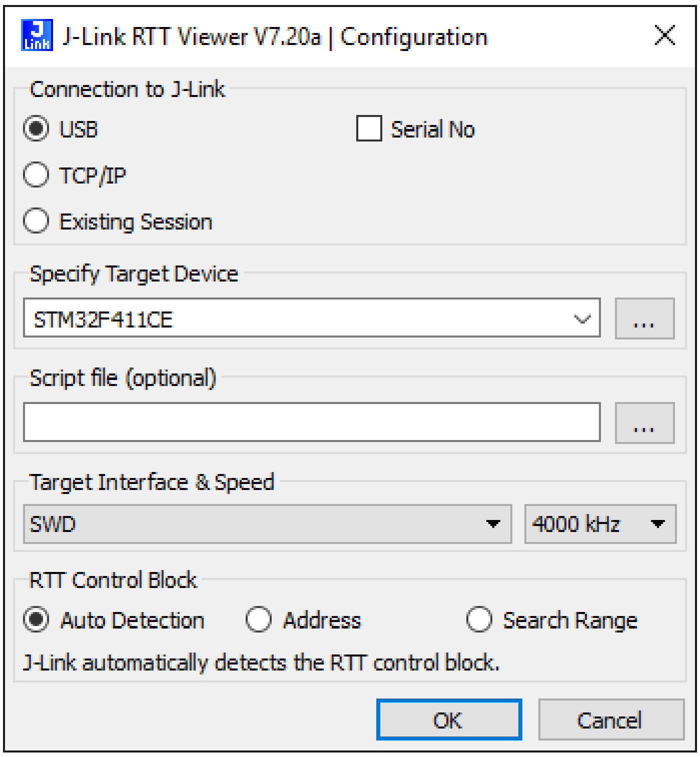


J-Link SWD connection

Convert ST-LINK to J-LINK

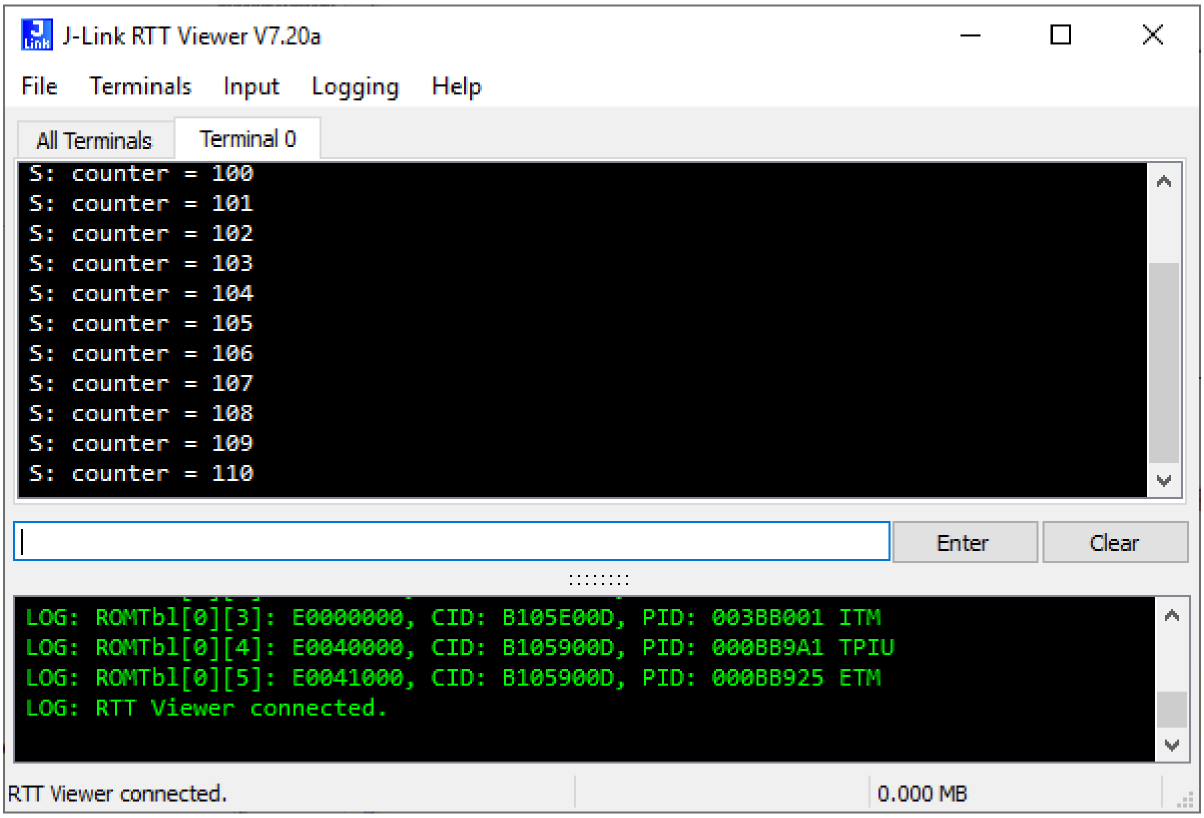
SEGGER offers a firmware upgrading the ST-LINK on-board on the Nucleo and Discovery Boards to a J-LINK On-Board debugger.

The configuration dialog will show up, select **USB** mode, and select the **Target device** from the list of supported devices.



RTT Viewer Configs

Then the viewer will open the Default Channel 0 to display RTT strings.



RTT Viewer

5. Redirect Standard IO to RTT

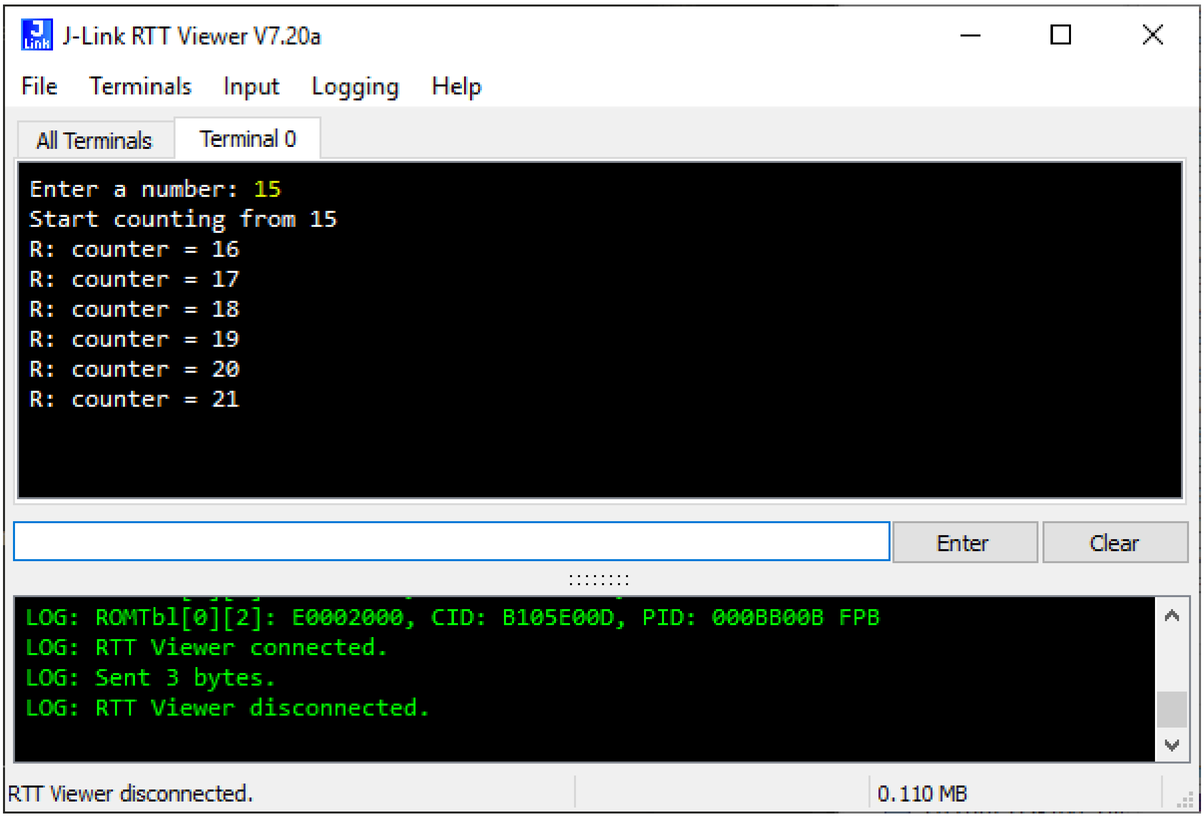
With the same method to redirect standard IO to UART or VCOM, two low-level function `_write()` and `_read()` should be overridden to redirect to RTT.

```
#ifdef REDIRECT_TO_RTT
int _read(int file, char *ptr, int len) {
  *ptr = SEGGER_RTT_WaitKey();
  return 1;
}

int _write(int file, char *ptr, int len) {
  SEGGER_RTT_Write(0, ptr, len);
  return len;
}
#endif
```

Blocking input

The function `SEGGER_RTT_WaitKey()` intensionally block the application to read a character. Once a character is available, it is read and this function returns.

Here is an example of reading a number using RTT redirection:



Use RTT redirection for `scanf`