J-Link RTT

Real Time Transfer テクノロジーについて

1. J-Link RTTとは？

本書は、SEGGER社技術WIKI情報を日本語訳、再編集したものとなります。
原文内容は、以下URLを参照ください。

https://wiki.segger.com/RTT

J-Link RTTについて

Real Time Transfer テクノロジーについて

Real Time Transfer（RTT）は、組み込みアプリケーションにおけるインタラクティブなユーザー通信のテクノロジです。
非常に高いパフォーマンスで SWO とセミホスティングの長所を兼ね備えています。

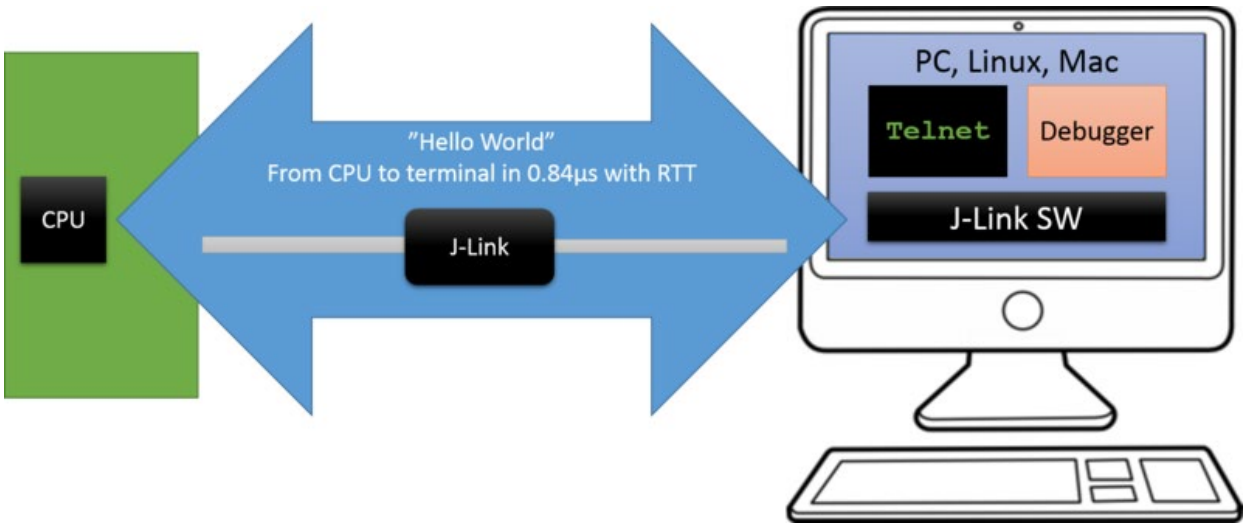
1

1. J-Link RTTとは？

高速・マイコン負荷の少ない独自データ取得送信インターフェース

J-Link RTTを使用すると、ターゲットマイクロコントローラから情報を出力することができます。ターゲットのリアルタイム動作に影響を与えることなく、入力情報をアプリケーションに高速送信します。

J-Link RTTは、任意のJ-Linkモデル及びサポート対象のターゲットプロセッサで使用できます。Cortex-M及びRXターゲットでバックグラウンドメモリアクセスを可能にします。



J-Link RTTは、複数のチャネルを両方向（上限のホストから下限のターゲットまで）でサポートします。これらのチャネルは、様々な目的に使用でき、ユーザーに可能な限り自由度を提供します。

デフォルトの実装では、方向ごとに一つのチャネルが使用されます。これは、プリント可能なターミナルの入出力を目的としています。　J-Link RTT ビューアでは、複数の「仮想」端末にこのチャネルを使用することができ、一つのターゲットバッファで複数のウィンドウ（例：標準出力用、エラー出力用、デバッグ出力用）にプリントできます。（ホストへの）追加チャネルは、例えばプロファイリング又はイベントトレースデータの送信に使用できます。

対応CPUコア
Cortex-M
Cortex-R
Cortex-A
RISC-V
Renesas RX

J-Link RTTの仕組み

2

2. J-Link RTTの仕組み

2.1 J-Link RTTの実装構成

J-Link RTTは、ターゲットのメモリ内のSEGGER RTTコントロールブロック構造を使用して、データの読み書きを管理します。コントロールブロックには、接続されたJ-Linkがメモリ内で認識するためのID、チャネルバッファとその状態を記述する使用可能な各チャネルのリングバッファ構造が含まれています。

使用可能なチャネルの最大数はコンパイル時に構成でき、各バッファは実行時にアプリケーションによって構成及び追加できます。アップバッファとダウンバッファは別々に扱うことができます。 各チャネルは、ブロック又は非ブロックに構成することができます。

ブロッキングモードでは、バッファがいっぱいに埋まるまでアプリケーションが待機し、すべてのメモリが書き込まれ、アプリケーション状態はブロックされますが、データが失われることはありません。

ノンブロッキングモードでは、バッファに収まるデータだけが書き込まれるか、全くデータが書き込まれず、残りのデータは破棄されます。これにより、デバッガが接続されていなくても、リアルタイムで実行することができます。開発者は特別なデバッグバージョンを作成する必要はなく、リリースアプリケーションでコードをそのまま使用できます。

2.2 制御ブロックを検索

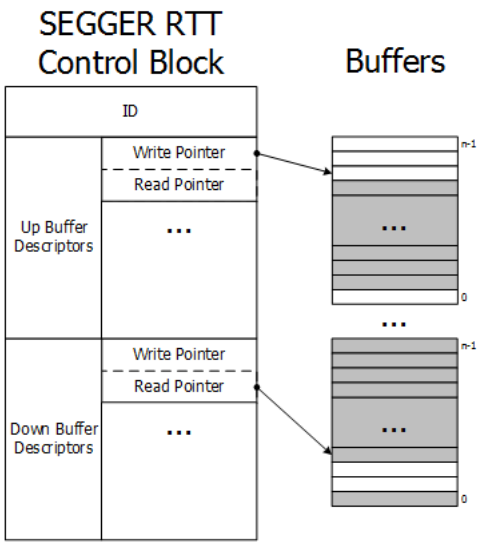
J-Link RTTがホストコンピュータ上で有効な場合、 RTTビューアのようなアプリケーション経由で直接RTTを使用するか、J-Linkを使用しているアプリケーションにTelnet経由で接続することで、 J-Link はターゲットの既知の RAM 領域で制御ブロックを自動的に検索します。RAM 領域または制御ブロックの特定のアドレスは、ホストアプリケーションを介して設定して、検出を高速化したり、制御ブロックを自動的に検出できない場合に設定することもできます。

2.3 制御ブロック位置のマニュアル設定

RTT制御ブロックの位置の自動検出は、ほとんどのターゲットで正常に動作しますが、制御ブロックの正確な位置、J-Linkが制御ブロックを検索する特定のアドレス範囲、いずれかをいつでも手動で指定できます。これは、次のJ-Linkコマンド文字列を利用して行われます。

- **SetRTTAddr**
- **SetRTTSearchRanges**

2.4 内部構造



この図は、ターゲット上の単純化された構造を示しています。

任意の数の「アップバッファ記述子」(ターゲット->ホスト)と「ダウンバッファ記述子」(ホスト->ターゲット)が存在する可能性があります。各バッファー・サイズは個別に構成できます。バッファー内の灰色の領域は、有効なデータを含む領域です。

アップバッファーの場合、書き込みポインターはターゲットによって書き込まれ、読み取りポインターはデバッグ・プローブ (J-Link、ホスト) によって書き込まれます。

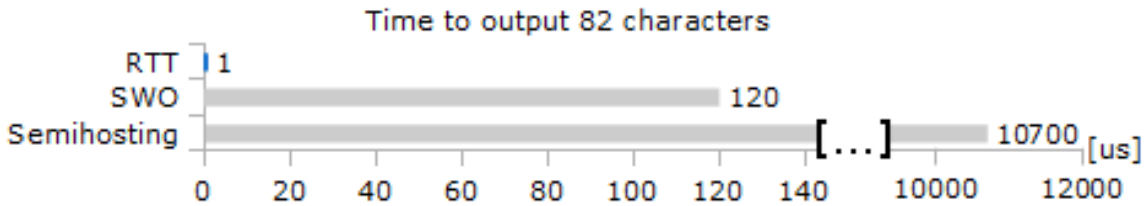
読み取りポインターと書き込みポインターが同じ要素を指している場合、バッファーは空です。これにより、競合状態が発生しないことが保証されます。

2.5 必要要件

RTTは、標準のデバッグポートを介してターゲットに接続されたJ-Linkにもかかわらず、**追加のピンやハードウェアを必要としません。**ターゲット環境やデバッグ環境の構成は不要で、さまざまなターゲット速度で使用することもできます。
RTT Viewerなどのソフトウェアを利用して、IDE やデバッガーで実行中のデバッグセッションと並行して使用できます。

2.6 パフォーマンス

RTTのパフォーマンスは、ホストPCにデータを出力するために使用される他のどの技術よりも大幅に高くなっています。テキストの平均行は、1マイクロ秒以内に出力することができます。基本的に、一つのmemcopy()を行う時間だけです。

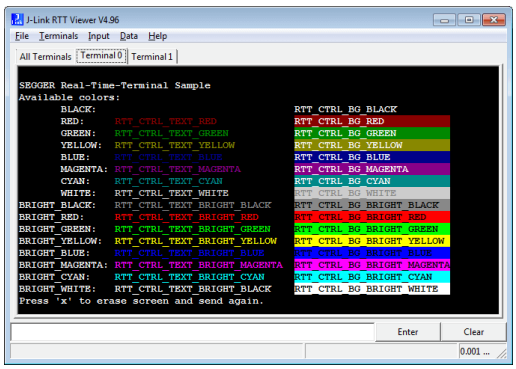


2.7 メモリフットプリント

RTT実装コードは、RAMの制御ブロック用に、500バイトまでのROMと24バイトのID + 24バイトのチャネルを使用します。各チャネルにはバッファ用のメモリが必要です。推奨されるサイズは、入出力チャネルの負荷に応じて、アップチャネルの場合は1kByte、ダウンチャネルの場合は16から32Byteです。

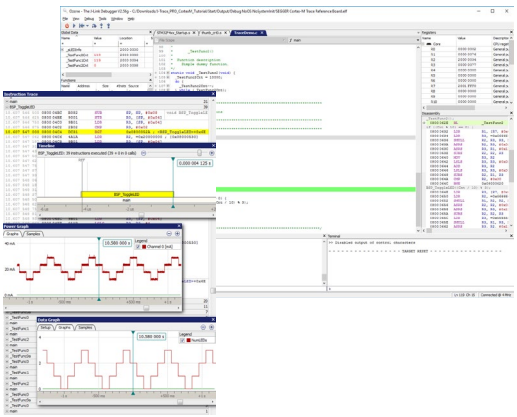
RTT対応ソフトウェアツール

RTTは、様々な対応ソフトウェアで利用可能です。



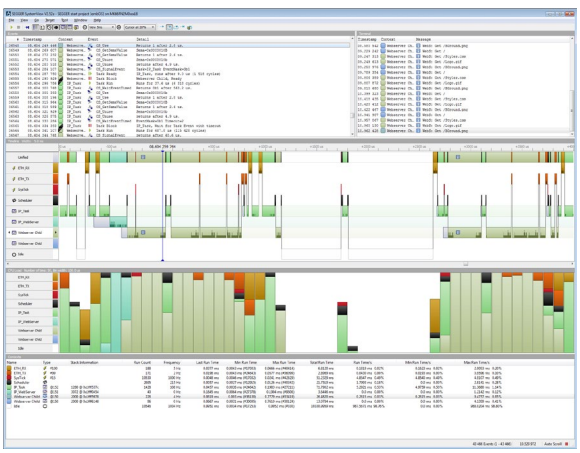
J-Link RTT Viewer（無償）

RTTのフル機能をサポート
J-Link・ターゲットとの接続、
デバッグセッションの開始、
接続の終了までカバーします。



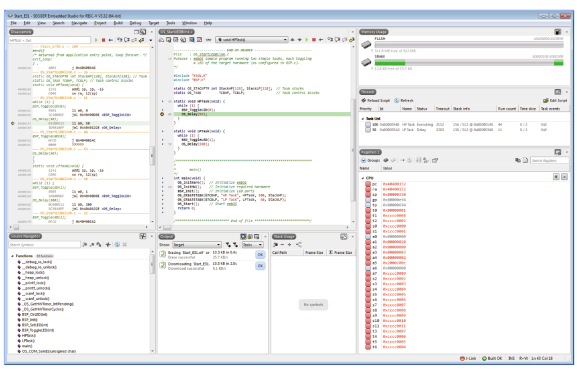
J-Linkデバッガ「OZONE」

(J-Link PLUS以上無償)
OZONEは様々なコンパイラに対応し、
RTTを利用する事でパフォーマンス分析
消費電力分析
RAMモニタリングなど可能な
高性能デバッガソフトウェア



SystemView（有償）

組込ソフトウェア開発者向けの
リアルタイムシステム分析
（視覚化・記録)ツール



SEGGER EmbeddedStudio（有償）

Arm Cortexシリーズをサポートする
統合開発環境

RTTモード

3

3. RTTモード

3.1 RTTモード一覧：
RTTは、次の様な動作モードを持っています。

モード	概要	対応製品
バックグラウンドモード	最大2MB/secの転送速度の最速モード	最新のJ-Linkシリーズ 最新のJ-Trace PROシリーズ
バックグラウンドモード（レガシー）	バックグラウンドモードのレガシーバージョンで、ファームウェア側での処理が対応していないため、転送速度が低下します。	最新のFlasherシリーズ J-Link BASE/PLUS v9未満 J-Link ULTRA+/PRO v4未満 J-TRACE PRO v2未満
ストップモード	疑似 RTT モード。CPU はデータを読み取るために停止します。バックグラウンドアクセスをサポートしていないCPU向け	すべてのJ-Link / J-Trace PRO / Flasherシリーズ

3.2 バックグラウンドモード

次項「3.3 バックグラウンドモード（レガシー）」と同じ動作モードですが、最新のJ-Link / J-Trace PROファームウェアの機能により転送速度が最大2MB/secと高速になっています。

3.3 バックグラウンドモード（レガシー）

このモードでは、アプリケーションの実行中に、J-Linkがターゲットシステムのメモリにアクセスでき(バックグラウンドメモリアクセス)、アプリケーションのリアルタイム動作に実質的に影響しません。このモードを使用するには、ターゲットMCUがバックグラウンドメモリアクセスをサポートしている必要があります。

バックグラウンドモード対応CPUコア

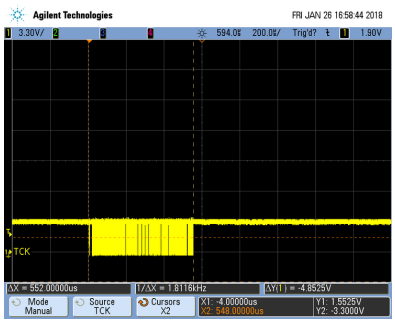
- Cortex-M0 / M0+ / M1 / M3 / M4 / M7 / M23 / M33 / M55
- Renesas RX
- Cortex-A 32-bit (ARMv7-A) ※一部のデバイスでは利用頂けません。Cortex-A項を参照
- Cortex-R 32-bit (ARMv7-R) ※一部のデバイスでは利用頂けません。Cortex-R項を参照
- RISC-V ※一部のデバイスでは利用頂けません。RISC-V項を参照

3.4 ストップモード

J-LinkはCPUを一時的に停止(対象アプリケーションの実行を中断)してメモリにアクセスし、メモリアクセスが完了した後も自動的に動作を継続します。リアルタイム動作に対する実際の影響(停止時間)は、セットアップ(使用されるターゲットインタフェース速度、使用されるターゲットインタフェース、JTAGチェーンの長さ、実際に使用されるコアなど)によって異なります。

対応CPUコア
Cortex-A / Cortex-R / RISC-V

このモードでは、リアルタイム動作への影響をできるだけ小さくするために、ターゲットインターフェイス速度が25MHz以上(J-Link ULTRA+、J-Link PRO)でのみ使用することをお勧めします。



リアルタイム動作への影響

	J-Link ULTRA+ (50MHz)	J-Link PLUS (15MHz)
256 bytes データ	- 552 us	- 1.02 ms
4 bytes データ	- 418 us	- 718 us

CPUコア詳細

4

4. CPUコア詳細

4.1 Cortex-M

CPU にキャッシュを実装している場合:

- RTT 制御ブロックとすべての RTT バッファは、キャッシュラインのアライメントを開始する必要があります。
- RTT 制御ブロック、すべての RTT バッファは、サイズをキャッシュラインの倍数としてください。
- システムが複数のキャッシュ・レベルを提供する場合、制御ブロックおよびバッファのアラインメントとサイズは、最大の領域を持つキャッシュラインをリファレンスとして取る必要があります。
- 制御ブロックおよびアップ/ダウン・バッファ0は、SEGGER_RTT.cによって定義されます。したがって、これらがアライメント要件を満たしていることを確認するために、使用RTT_CACHE_LINE_SIZEシステムでキャッシュラインサイズが< = 32バイトでない場合に、キャッシュラインを指定する必要があります。
- 初期化が完了した後、RTT が初めて使用される前に、RTT 制御ブロックでキャッシュクリーン + 無効化 + すべての RTTバッファを呼び出すのは、ユーザー アプリケーションが担います。それらはmain()で実施することを推奨します。これにより、キャッシュラインに余計なRTTデータ/情報が含まれていないことが保証され、それ以外の場合は実行時のある時点でメモリに削除される可能性があります。
  embOS が使用されている場合、通常、OS_InitHW() が呼び出された後に RTT が使用されていることを確認し、すべてのキャッシュラインのクリーン + 無効化を実行します。
- 実行時に指定および初期化される RTT バッファー (インデックス 0 >) の場合、AllocBuffer() / ConfigBuffer() 関数のいずれかを呼び出す直前に、キャッシュのクリーン + 無効化を実行する必要があります。
- アプリケーションでは、RTT制御ブロック、バッファ、およびその名前へのポインタは、仮想アドレス=物理アドレスとリンクされている必要があります。
- アプリケーションは、制御ブロック + バッファーが配置されているメモリにキャッシュされていないアドレス別名を提供する必要があります。制御ブロックとすべてのバッファは、そのキャッシュされていないアドレス別名を介してアクセス可能である必要があります。

4.2 Cortex-A/R

- RTTバックグラウンドモードは、AHB-AP またはAXI-AP経由で動作します。
- AHB-AP、AXI-AP はオプション実装のため、搭載の有無はデバイスにより異なります。搭載していないCPUデバイスではバックグラウンドモードの利用はできません。
- J-Link が「RTTで使用可能なAP」を認識していないデバイスの場合、バックグラウンドメモリアクセスに使用する AP は、J-Link スクリプトファイルを使用して指定する必要があります。J-Link スクリプトファイルの詳細については、「J-Link スクリプトファイル」の記事を参照してください。
- RTT 制御ブロックとバッファは、外部メモリではなく内部メモリに配置してください。Cortex-Aベースのシステムでは、外部アドレス空間は一般的に非常にデリケートであり、初期化される前にアクセスされた場合(例えば、DDRコントローラの初期化が行われたなど)、ハングアップなどを簡単に引き起こす可能性があります。J-Link には外部メモリ空間の初期化がいつ行われるかについての情報がないため、外部メモリが利用可能になる前に J-Link 側から RTT アクセスが行われる可能性があります。
- AHB/AXI-AP アクセスは DMA アクセスのように動作し、CPU L1/L2 などのキャッシュをバイパスします。
- CPUがキャッシュを実装する場合は、「4.1 Cortex-M」を参照ください。

すぐに動作するCPU
- Xilinx Zynq 7000 series, Cortex-A9 based (J-Link Software V6.85b以降)
- NXP i.MX6Solo series, Cortex-A9 based (J-Link Software V6.85d以降)

サンプルプロジェクト
emPower Zynq board：Download
動作環境：SEGGER Embedded Studio 5.10以降
J-Link Software V6.85d以降

4.3 Cortex – バックグラウンドメモリアクセス

ARM Cortexターゲットでは、RTTに必要なバックグラウンドメモリアクセスは、DMAに似ていますが、デバッグプローブによって排他的にアクセスできる、いわゆるAHB-APを介して実行されます。

Cortex-M ターゲットには常に AHB-AP が存在しますが、Cortex-A および Cortex-R ターゲットでは、これはオプションのコンポーネントです。Cortex-A/R ターゲットは複数の AP を実装する場合があるため(AHB-AP でなくても)、Cortex-A/R ターゲットで RTT を使用するには、RTT バックグラウンド メモリ アクセスに使用される AHB-AP である AP のインデックスを手動で指定する必要があります。

これは、次のJ-Link コマンド文字列 「CORESIGHT_SetIndexAHBAPToUse」 を使用して行われます。

4.4 RISC-V

- RTT バックグラウンド メモリ アクセスは、RISC-V システム バス アクセス (SBA) または AHB/AXI-AP 経由で実行されます。
- SBA サポートはオプションであり、実際の MCU によって異なります。
- AHB/AXI-AP アクセスは、RISC-V コアが ARM CoreSight DAP の背後にあるセットアップでのみ可能です。
- **SBAによるサポート**
  特別な設定は必要ありません。J-Link は SBA サポートを自動的に検出し、RTT をすぐに使用できます。
- **AP経由のサポート**
  AHB-AP と AXI-AP の存在はオプションであり、MCU によって対応が異なります。
- J-Linkソフトウェア V7.50よりこのアクセス方式のサポート開始
- J-Link が「RTT で使用可能な AP」を認識していないデバイスの場合、バックグラウンド メモリ アクセスに使用する AP は、J-Link スクリプトファイルを使用して指定する必要があります。J-Link スクリプトファイルの詳細については、「J-Link スクリプトファイル」の記事を参照してください。
- RTT 制御ブロックとバッファは、外部メモリではなく内部メモリに配置してください。
  Cortex-Aベースのシステムでは、外部アドレス空間は一般的に非常にデリケートであり、初期化される前にアクセスされた場合(例えば、DDRコントローラの初期化が行われたなど)、ハングアップなどを簡単に引き起こす可能性があります。J-Link には外部メモリ空間の初期化がいつ行われるかについての情報がないため、外部メモリが利用可能になる前に J-Link 側から RTT アクセスが行われる可能性があります。
- AHB/AXI-AP アクセスは DMA アクセスのように動作し、CPU L1/L2 などのキャッシュをバイパスします。
- CPUがキャッシュを実装する場合は、「4.1 Cortex-M」を参照ください。

RTT通信

5

5 RTT通信

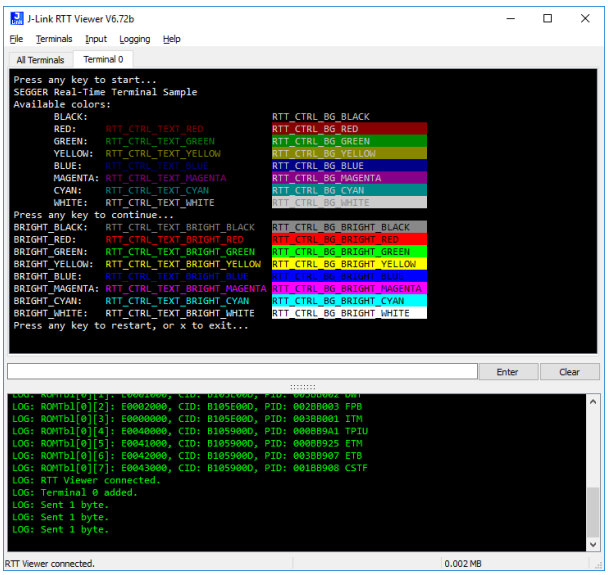
ターゲット上のRTT実装との通信は、様々なアプリケーションで実行できます。この機能は、JLink SDKを使用してカスタムアプリケーションに統合することもできます。

ターゲットアプリケーションでRTTを使用することは簡単です。実装コードは自由にダウンロードでき、既存のアプリケーションに統合できます。 RTTを介して通信するには、任意のJ-Linkを使用できます。

端末（チャネル0）を介して通信する簡単な方法は、J-Linkへの接続（例えば、デバッグセッション経由）が有効な場合にTelnetクライアントなどでlocalhost：19021への接続を作成する方法です。

J-Link Software Packageには、いくつかのアプリケーションが付属しています。様々な目的のためにRTTの機能を利用して実現しています。

5.1 RTT Viewer



J-Link RTT Viewerは、Windows、MacOS、Linuxで利用可能なGUIアプリケーションであり、J-Link Software & documentation packの一部としてダウンロードできます。これにより、RTTのすべての機能を1つのアプリケーションで使用できます。

- Channel 0の端末出力を表示します。
- Channel 0に最大 16 個の仮想端末。
- Channel 0へのテキスト入力の送信。
- 色付きのテキストのテキスト制御コードを解釈し、ターミナルを制御する。
- 端末データをファイルに記録する。
- Channel 1のデータをログに記録します。

詳細については、「RTT Viewer」の記事を参照ください。

5.2 RTT Client

J-Link RTT ClientはTelnetクライアントとして機能しますが、デバッグセッションが終了すると自動的にJ-Link接続に再接続を試みます。J-Link RTTクライアントは、Windows、Linux、OS X用のJ-Link Software & documentation packに含まれており、簡単なRTTユースケースに使用できます。

コマンドラインオプション

コマンド	解説
-?	コマンドラインオプションを表示
-LocalEcho	1 =有効-/ 0 =ローカルエコーを無効
-rtttelnetport <ポート>	RTT telnetポート番号を設定します

J-Link RTT Clientは単独では、 J-Link 経由でデバイスへデバッグ接続行いません。RTT が機能するには、アクティブなデバッグ接続が必要です。したがって、J-Link RTT Clientは、アクティブなデバッグ接続を確立する J-Link ツール (J-Link Commander や J-Link GDB Server など) と組み合わせて利用します。

5.3 RTT Logger

J-Link RTT Loggerを使用すると、アップチャネル1のデータを読み取り、ファイルに記録することができます。このチャネルは、例えば、パフォーマンス分析データをホストに送信するために使用することができます。J-Link RTT LoggerはJ-Link専用の接続を開き、デバッガを実行せずにスタンドアロンで使用することができます。
このアプリケーションは、Windows、Linux、及びOS X用のJ-Link Software & documentation packの一部です。J-Link RTTロガーのソースは、デバッガなどの他のPCアプリケーションでRTTを統合するための出発点として使用でき、J-Link SDKの一部となっています。

コマンド	解説
-?	コマンドラインオプションを表示
-Device <DeviceName>	ターゲット・デバイスを<DeviceName>に設定します。
-if <Interface>	ターゲット インターフェイスを <Interface> に設定します。
-Speed <SpeedInKHZ>	速度を <SpeedInKHZ> に設定します。
-USB <SN>	USB経由でシリアル番号<SN>のJ-Linkに接続します(0 == 自動検出/デバイス選択ダイアログ)。
-IP <IPAddr>	TCP/IP 経由で IP <IPAddr> を使用して J-Link に接続します。
-RTTAddress <RTTAddress>	RTT アドレスを <RTTAddress> に設定
-RTTSearchRanges "<Ranges>"	RTT 検索範囲を <Ranges> に設定します。
-RTTChannel <RTTChannel>	RTT チャネルを <RTTChannel>に設定

5.4 その他のホストアプリケーション

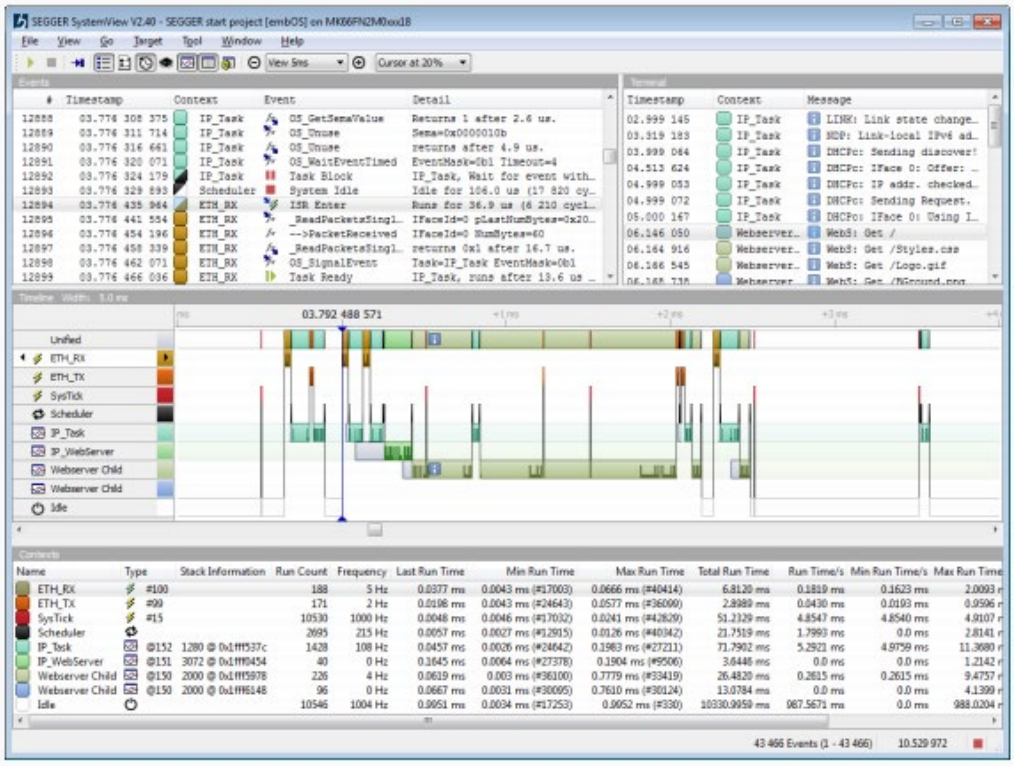
RTTは、デバッガやデータビジュアライザのような他のPCアプリケーションにも、2通りの方法で統合することができます。

- アプリケーションは、J-Link接続が有効なときにlocalhost：19021で開かれるRTT Telnetサーバーへのソケット接続を確立できます。「J-Link ソフトウェアの TELNET チャネル」記事を参照ください。
- アプリケーションはJ-Linkとの独自の接続を作成し、J-Link SDKの一部であるJ-Link RTT APIを使用して、RTTを直接設定して使用します。

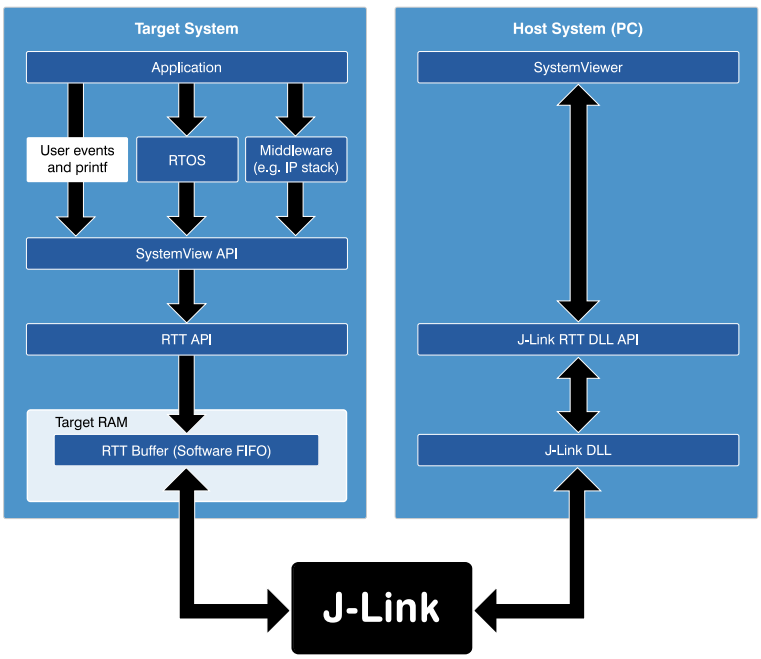
参考：J-Link SDK

5.5 SystemView

組込ソフトウェア開発者向けのリアルタイムシステム分析ツール



アプリケーションの実行時の状態を表示し、開発者は、デバッガから出力されアプリケーションの状態をより詳細に確認できます。
SystemViewを使うことにより、アプリケーションが設計通りに動作し、非効率な動きをしていないか、意図しないリソース競合などを起こしていないか、などを確認知ることができます。実行中のターゲットボードからリアルタイムに記録し、取得したデータは各種分析画面で視覚化されます。



■ 連続データ記録

RTTインターフェースを利用することで、リアルタイムにデータ取得、連続記録を実現します。

実装

6

6.0 実装

SEGGER RTT実装コードはANSI Cで記述されており、使用可能なソースを追加するだけで、組み込みアプリケーションに統合可能です。RTTは、シンプルで使いやすいAPIを介して使用できます。RTTで使用する標準のprintf()関数をオーバーライドすることもできます。RTTを使用すると、出力に要する時間が最小限に抑えられ、アプリケーションがタイムクリティカルなリアルタイムタスクを実行している間にホストコンピュータにデバッグ情報を出力します。

実装コードには、RTTを介して書式設定された文字列を書き込むために使用できるprintf()のシンプルなバージョンも含まれています。これは、ほとんどの標準ライブラリprintf()実装よりも小さく、ヒープを必要とせず、設定可能な量のスタックしか必要としません。

SEGGER RTTの実装は、コンパイル時にプリプロセッサ定義ですべて構成可能です。チャネル数、デフォルトチャネルのサイズを設定することができます。

読み取りと書き込みは、定義可能なLock()ルーチンとUnlock()ルーチンを使用して、タスクセーフにすることができます。

6.1 API関数

RTT実装では、次のAPI関数を使用できます。「SEGGER_RTT.h」を呼び出し元にインクルードする必要があります。

API関数	解説
SEGGER_RTT_ConfigDownBuffer()	名前、サイズ、フラグを指定して、ダウンバッファを設定又は追加します。
SEGGER_RTT_ConfigUpBuffer()	名前、サイズ、フラグを指定して、アップバッファを設定又は追加します。
SEGGER_RTT_GetKey()	SEGGER RTTバッファ0から1文字を読み込みます。ホストには既にデータが格納されています。
SEGGER_RTT_HasKey()	SEGGER RTTバッファで少なくとも1文字の読み込みが可能かどうかをチェックします。
SEGGER_RTT_Init()	RTTコントロールブロックを初期化します。
SEGGER_RTT_printf()	書式設定された文字列をホストに送信します。
SEGGER_RTT_Read()	ホストによって既に格納された任意のRTTダウンチャネルから文字を読み込みます。
SEGGER_RTT_SetTerminal()	チャネル0の次のデータを送信するように「仮想」ターミナルを設定します。
SEGGER_RTT_TerminalOut()	一つの文字列を特定の「仮想」ターミナルに送信します。
SEGGER_RTT_WaitKey()	SEGGER RTT バッファ0 で少なくとも 1 文字が使用可能になるまで待機します。文字が使用可能になると、その文字が読み取られて戻されます。
SEGGER_RTT_Write()	RTTチャネルでホストにデータを送信します。
SEGGER_RTT_WriteString()	RTTを使用して、末尾が0の文字列をアップチャネルに書き込みます。
SEGGER_RTT_GetAvailWriteSpace()	循環バッファで使用可能なバイト数を返します。

6.1.01 SEGGER_RTT_ConfigDownBuffer()

名前、サイズ、フラグを指定して、ダウンバッファを設定又は追加します。

構文

int SEGGER_RTT_ConfigDownBuffer (unsigned BufferIndex, const char* sName, char* pBuffer, int BufferSize, int Flags);

パラメータ

パラメータ	説明
BufferIndex	設定するバッファのインデックス。 SEGGER_RTT_MAX_NUM_DOWN_CHANNELSより低い必要があります。
sName	チャネルの名前として表示される、末尾が0の文字列へのポインタ。
pBuffer	チャネルによって使用されるバッファへのポインタ。
BufferSize	バイト単位のバッファサイズ。
Flags	チャネルのフラグ（ブロック又は非ブロック）。

戻り値

値	説明
>=0	O.K.
< 0	エラー

例

```
//
// Configure down channel 1
//
SEGGER_RTT_ConfigDownChannel(1, "DataIn", &abDataIn[0], sizeof(abDataIn),
SEGGER_RTT_MODE_NO_BLOCK_SKIP);
```

6.1.02 SEGGER_RTT_ConfigUpBuffer()

名前、サイズ、フラグを指定して、アップバッファを設定又は追加します。

構文

int SEGGER_RTT_ConfigUpBuffer (unsigned BufferIndex, const char* sName, char* pBuffer, int BufferSize, int Flags);

パラメータ

パラメータ	説明
BufferIndex	設定するバッファのインデックス。 SEGGER_RTT_MAX_NUM_DOWN_CHANNELSより低い必要があります。
sName	チャネルの名前として表示される、末尾が0の文字列へのポインタ。
pBuffer	チャネルによって使用されるバッファへのポインタ。
BufferSize	バイト単位のバッファサイズ。
Flags	チャネルのフラグ（ブロック又は非ブロック）。

戻り値

値	説明
>=0	O.K.
< 0	エラー

例

```
//
// Configure up channel 1 to work in blocking mode
//
SEGGER_RTT_ConfigUpChannel(1, "DataOut", &abDataOut[0], sizeof(abDataOut),
SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL);
```

6.1.03 SEGGER_RTT_GetKey()

SEGGER RTTバッファ0から1文字を読み込みます。ホストには既にデータが格納されています。

構文

```
int SEGGER_RTT_GetKey (void);
```

戻り値

値	説明
>=0	読み込まれた文字（0～255）
< 0	使用可能な文字無し（空バッファ）

例

```
int c;
c = SEGGER_RTT_GetKey();
if (c == 'q') {
exit();
}
```

6.1.04 SEGGER_RTT_HasKey()

SEGGER RTTバッファで少なくとも1文字の読み込みが可能かどうかをチェックします。

構文

```
int SEGGER_RTT_HasKey (void);
```

戻り値

値	説明
1	このバッファでは最低1文字が使用可能です。
0	読み込みに使用可能な文字はありません。

例

```
if (SEGGER_RTT_HasKey()) {
  int c = SEGGER_RTT_GetKey();
}
```

6.1.05 SEGGER_RTT_Init()

RTTコントロールブロックを初期化します。

構文

```
void SEGGER_RTT_Init (void);
```

6.1.06 SEGGER_RTT_printf()

書式設定された文字列をホストに送信します。

構文

```
int SEGGER_RTT_printf (unsigned BufferIndex, const char * sFormat, ...)
```

パラメータ

パラメータ	説明
BufferIndex	文字列を送信するアップチャネルのインデックス
sFormat	書式文字列へのポインタ、変換のための引数

戻り値

値	説明
>=0	送信されたバイト数
< 0	エラー

例

```
SEGGER_RTT_printf(0, "SEGGER RTT Sample. Uptime: %.10dms.", /*OS_Time*/
890912);
// Formatted output on channel 0: SEGGER RTT Sample. Uptime: 890912ms.
```

■ 追加情報

1.変換指定子の書式

%[flags][FieldWidth][.Precision]ConversionSpecifier%

2. サポートしているフラグ

フラグ	説明
-	フィールド幅内で左揃え
+	署名された変換のために常に署名拡張機能を常に出力する
0	スペースの代わりに 0 でパッドします。'-'-フラグまたは精度を使用する場合は無視されます。

3. サポートしている変換指定子

変換指定子	説明
c	引数を1文字として出力します
d	引数を符号付き整数として出力します
u	引数を符号なし整数として出力します
x	引数を16進整数として出力します
s	引数が指す文字列を出力します
p	引数を8桁の16進整数として出力します

6.1.07 SEGGER_RTT_Read()

ホストによって既に格納された任意のRTTダウンチャネルから文字を読み込みます。

構文

```
unsigned SEGGER_RTT_Read (unsigned BufferIndex, char* pBuffer, unsigned BufferSize);
```

パラメータ

パラメータ	説明
BufferIndex	読み込むダウンチャネルのインデックス
pBuffer	読み込んだ文字を格納する文字バッファへのポインタ
BufferSize	バッファで使用可能なバイト数

戻り値

値	説明
>=0	読み込まれたバイト数

例

```
char acIn[4];
unsigned NumBytes = sizeof(acIn);
NumBytes = SEGGER_RTT_Read(0, &acIn[0], NumBytes);
if (NumBytes) {
  AnalyzeInput(acIn);
}
```

6.1.08 SEGGER_RTT_SetTerminal()

チャネル0の次のデータを送信するように「仮想」ターミナルを設定します。

構文

```
void SEGGER_RTT_SetTerminal(char TerminalId);
```

パラメータ

パラメータ	説明
TerminalId	仮想ターミナルのID（0～9）

例

```
//
// Send a string to terminal 1 which is used as error out.
//
SEGGER_RTT_SetTerminal(1); // Select terminal 1
SEGGER_RTT_WriteString(0, "ERROR: Buffer overflow");
SEGGER_RTT_SetTerminal(0); // Reset to standard terminal
```

6.1.09 SEGGER_RTT_TerminalOut()

一つの文字列を特定の「仮想」ターミナルに送信します。

構文

int SEGGER_RTT_TerminalOut (char TerminalID, const char* s);

パラメータ

パラメータ	説明
TerminalId	仮想ターミナルのID（0～9）
s	送信される末尾が0の文字列へのポインタ

戻り値

値	説明
>=0	ターミナルに送信されたバイト数
< 0	エラー

例

```
//
// Sent a string to terminal 1 without changing the standard terminal.
//
SEGGER_RTT_TerminalOut(1, "ERROR: Buffer overflow.");
```

6.1.10 SEGGER_RTT_Write()

RTTチャネルでホストにデータを送信します。

構文

unsigned SEGGER_RTT_Write (unsigned BufferIndex, const char* pBuffer, unsigned NumBytes);

パラメータ

パラメータ	説明
BufferIndex	データを送信するアップチャネルのインデックス
pBuffer	送信されるデータへのポインタ
NumBytes	送信するバイト数

戻り値

値	説明
>=0	送信されたバイト数
< 0	エラー

例

```
//
// Sent a string to terminal 1 without changing the standard terminal.
//
SEGGER_RTT_TerminalOut(1, "ERROR: Buffer overflow.");
```

6.1.11 SEGGER_RTT_WaitKey()

SEGGER RTT バッファ0 で少なくとも 1 文字が使用可能になるまで待機します。文字が使用可能になると、その文字が読み取られて戻されます。

構文

```
int SEGGER_RTT_WaitKey (void);
```

戻り値

値	説明
>=0	読み込まれた文字（0～255）

例

```
int c = 0;
do {
  c = SEGGER_RTT_WaitKey();
} while (c != 'c');
```

6.1.12 SEGGER_RTT_WriteString()

RTTを使用して、末尾が0の文字列をアップチャネルに書き込みます。

構文

```
unsigned SEGGER_RTT_WriteSting (unsigned BufferIndex, const char* s);
```

パラメータ

パラメータ	説明
BufferIndex	文字列を送信するアップチャネルのインデックス
s	送信される末尾が0の文字列へのポインタ

戻り値

値	説明
>=0	送信されたバイト数

例

```
SEGGER_RTT_WriteString(0, "Hello World from your target.¥n");
```

6.1.13 SEGGER_RTT_GetAvailWriteSpace()

循環バッファで使用可能なバイト数を返します。

構文

```
unsigned SEGGER_RTT_GetAvailWriteSpace (unsigned BufferIndex);
```

パラメータ

パラメータ	説明
BufferIndex	スペースをチェックする必要があるアップ・チャネルのインデックス

戻り値

値	説明
>=0	選択されたアップバッファ内で解放されているバイト数

例

```
unsigned NumBytesFree;
NumBytesFree = SEGGER_RTT_GetAvailWriteSpace(0);
```

6.2 コンフィギュレーション定義

6.2.1 RTTコンフィギュレーション

定義	解説
SEGGER_RTT_MAX_NUM_DOWN_BUFFERS	（ターゲットへの）ダウンチャネルの最大数
SEGGER_RTT_MAX_NUM_UP_BUFFERS	（ホストへの）アップチャネルの最大数
BUFFER_SIZE_DOWN	デフォルトダウンチャネル0のバッファサイズ
BUFFER_SIZE_UP	デフォルトアップチャネル0のバッファサイズ
SEGGER_RTT_PRINT_BUFFER_SIZE	charをバルク送信するためのSEGGER_RTT_printf用のバッファのサイズ
SEGGER_RTT_LOCK()	RTT操作内からの割り込み及びタスクスイッチを防ぐロックルーチン
SEGGER_RTT_UNLOCK()	RTT操作後に割り込み及びタスクスイッチを許可するルーチンのロック解除
SEGGER_RTT_IN_RAM	initセグメント内のRTTコントロールブロックを誤って識別することを防ぐために、アプリケーション全体がRAMにあることを1として定義します。

6.2.2 チャンネルバッファコンフィギュレーション

定義	解説
SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL	アップバッファが埋まると、書き込み関数の呼び出しがブロックされます。
SEGGER_RTT_MODE_NO_BLOCK_SKIP	アップバッファにすべての受取データを格納するための十分なスペースがない場合、バッファには何も書き込まれません。
SEGGER_RTT_MODE_NO_BLOCK_TRIM	アップバッファにすべての受取データを保持するための十分なスペースがない場合、余分なデータを破棄しながら、使用可能なスペースを受け取りデータで埋めます。

SEGGER_RTT_TerminalOutは、ノンブロッキングモードを使用している間でも、暗黙的にターミナルを切り替え、コマンドが常に送信されるようにします。

バッファ構成はターゲット／デバイスのバッファにのみ影響し、ホストのバッファには影響しません。

6.2.3 カラーコントロールシーケンス

設定	解説
RTT_CTRL_RESET	テキストの色と背景色をリセット
RTT_CTRL_TEXT_*	テキストの色を以下の色いずれかに設定します。 •BLACK •RED •GREEN •YELLOW •BLUE •MAGENTA •CYAN •WHITE (light grey) •BRIGHT_BLACK (dark grey) •BRIGHT_RED •BRIGHT_GREEN •BRIGHT_YELLOW •BRIGHT_BLUE •BRIGHT_MAGENTA •BRIGHT_CYAN •BRIGHT_WHITE
RTT_CTRL_BG_*	背景色を以下の色いずれかに設定します。 •BLACK •RED •GREEN •YELLOW •BLUE •MAGENTA •CYAN •WHITE (light grey) •BRIGHT_BLACK (dark grey) •BRIGHT_RED •BRIGHT_GREEN •BRIGHT_YELLOW •BRIGHT_BLUE •BRIGHT_MAGENTA •BRIGHT_CYAN •BRIGHT_WHITE

サンプルコード

7　サンプルコード

7

7 サンプルコード

```
/*********************************************************************
*                    SEGGER Microcontroller GmbH                     *
*       Solutions for real time microcontroller applications         *
**********************************************************************
*
*                                                                    *
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
*                                                                    *
*      www.segger.com Support: support@segger.com                    *
*                                                                    *
**********************************************************************
*


---------------------------------------------------------------------
File    : RTT.c
Purpose : Simple implementation for output via RTT.
It can be used with any IDE.
---------------------------- END-OF-HEADER ---------------------------
*/

#include "SEGGER_RTT.h"

static void _Delay(int period) {
  int i = 100000*period;
  do { ; } while (i--);
}

int main(void) {
  int Cnt = 0;

  SEGGER_RTT_WriteString(0, "Hello World from SEGGER!¥n");
  do {
    SEGGER_RTT_printf(0, "%sCounter: %s%d¥n",
                      RTT_CTRL_TEXT_BRIGHT_WHITE,
                      RTT_CTRL_TEXT_BRIGHT_GREEN,
                      Cnt);
    if (Cnt > 100) {
      SEGGER_RTT_TerminalOut(1, RTT_CTRL_TEXT_BRIGHT_RED"Counter overflow!");
      Cnt = 0;
    }
    _Delay(100);
    Cnt++;
  } while (1);
  return 0;
}

/************************** End of file ****************************/
```

J-Linkソフトウェアの疑似Telnetチャンネル

8

8 J-Linkソフトウェアの疑似Telnetチャンネル

J-Linkソフトウェアは、ポート19021にTELNETに似たチャネルを提供し、デバッグセッションと並行して実行できる別のサードパーティ製アプリケーションからRTTデータに簡単にアクセスできるようにします。TELNET チャネルは非常に使いやすく、基本的に必要なのは、ポート 19021 へのローカル TCP/IP 接続を開いてデータの受信を開始することだけです。

このチャネルはあくまで「TELNET ライク」です。

- TELNET プロトコルは完全には実装されていないため、すべての機能がサポートされているわけではありません。
- 出力可能文字のみで構成される RTT データに対して TELNET に準拠しています。
- (文字以外の) Raw RTT データは、rawバイトとして解釈する必要があります。

8.1 SEGGER Telnet設定文字列

TELNET経由で接続を確立した後、ユーザーはホストシステムから(例えば、J-Link RTTクライアントまたはTelnet client Puttyなどを介して)SEGGER TELNET設定文字列の送信を100ms待機します。
TELNET 接続が確立されてから 100msが経過した後に SEGGER TELNET 設定文字列を送信しても有効にならず、ホストから送信された RTT データの場合と同様に扱われます。
SEGGER TELNET 設定文字列の送信はオプションであるため、RTT はそのような設定文字列を送信しなくても正しく機能します。

設定の構文は、以下の通りです。
$$SEGGER_TELNET_ConfigStr=<Command>[;<Command>][...]$$

それぞれの<Command>の設定は以下の通りです。
<CommandName>;[<Value1>;<Value2>;...]

8.1.1 SEGGER Telnet設定文字列

コマンド	解説
RTTCh;<ChannelNo>	RTT チャンネルを <ChannelNo>に設定します。
SetRTTAddr;<Addr>	RTT制御ブロックのアドレスを<CtrlBlkAddr>に設定します。
SetRTTSearchRanges;<Range>	RTT制御ブロックの検索範囲を<Range(s)>に設定します。

8.1.2 設定文字列コマンドリスト

RTTCh;<ChannelNo>

構文

RTTCh;<ChannelNo>

例

```
$$SEGGER_TELNET_ConfigStr=RTTCh;1$$
```

SetRTTAddr;<Addr>

構文

SetRTTAddr;<Addr>

例

```
$$SEGGER_TELNET_ConfigStr=SetRTTAddr;0x20000000$$
```

SetRTTSearchRanges;<Range>

構文

SetRTTSearchRanges;<Range(s)>
<Range(s)> = <RangeStartInHex> <RangeSizeInHex>[, <Range1StartInHex> <Range1SizeInHex>, ...]

例

```
$$SEGGER_TELNET_ConfigStr=SetRTTSearchRanges;0x10000000 0x1000,
0x20000000 0x1000$$
```

複数コマンドの例

$$SEGGER_TELNET_ConfigStr=RTTCh;1;SetRTTAddr;0x20000000;$$

8.2 サンプルコード

J-Link SoftwareのTELNETチャネルに接続するTELNETクライアントのサンプルコードです。

8.2.1 ブロッキングでの利用

特別なオプションがない場合、ホストOSのSocket API “connect()” の呼び出しは、接続が確立されるかタイムアウトに達するまでブロックされます。
これはデフォルトの方法ですが、カウントダウン中にアプリケーションを閉じる要求を受け取った場合に備えて、connect() を呼び出すスレッドがブロックされ、タイムアウトカウントダウン中に接続自体が終了するという欠点があります。

```
Complete contents of the SYS_ functions can be retrieved from the source code delivered with the J-Link SDK
void TELNET_Client(void) {
  int t;
  int r;
  SYS_SOCKET_HANDLE hSock;
  char ac[512];

  do {
    r = -1;
    printf("Waiting for connection...¥n");
    do {
      hSock = SYS_SOCKET_OpenTCP();
      if (hSock == SYS_SOCKET_INVALID_HANDLE) {     // Failed to open socket? => Something basic in the OS failed. Done...
        goto ErrHandler;
      }
      r = SYS_SOCKET_Connect(hSock, 0x7F000001, 19021); // Connect to 127.0.0.1, port 19021
      if (r == 0) {
        break;
      }
      //
      // We need to close a socket and reopen a new one if a timeout occurs,
      // otherwise the OS sometimes does not recognize that
      // the port became available in between
      //
      SYS_SOCKET_Close(hSock);
    } while (r <= 0);
    //
    // Read from TELNET connection
    //
    do {
      r = SYS_SOCKET_Receive(hSock, ac, sizeof(ac) - 1);
      if (r <= 0) {                                 // 0 == Connection gracefully closed by server. < 0 == Some other error
        break;
      }
      ac[r] = 0;
      printf("%s", ac);
    } while (1);
    SYS_SOCKET_Close(hSock);
  } while (1);
ErrHandler:
}
```

8.2.2 ノンブロッキングでの利用

ソケットを非ブロッキングに構成して、connect() の呼び出しがすぐに戻り、スレッドが応答を維持できるようにすることができます。接続が正常に確立されたかどうかを確認するには、ソケットの状態を定期的にチェックする必要があります。

```
Complete contents of the SYS_ functions can be retrieved from the source code delivered with the J-Link SDK

void TELNET_Client(void) {
  int t;
  int r;
  SYS_SOCKET_HANDLE hSock;
  char ac[512];

  do {
    r = -1;
    printf("Waiting for connection...¥n");
    do {
      hSock = SYS_SOCKET_OpenTCP();
      if (hSock == SYS_SOCKET_INVALID_HANDLE) {     // Failed to open socket? => Something basic in the OS
failed. Done...
        goto ErrHandler;
      }
      SYS_SOCKET_SetNonBlocking(hSock);            // Make sure that connect() returns immediately
      SYS_SOCKET_Connect(hSock, 0x7F000001, 19021); // Connect to 127.0.0.1, port 19021
      r = SYS_SOCKET_IsWriteable(hSock, 10);        // Returns if socket is ready to send data (connection
established)
      if (r > 0) {
        SYS_SOCKET_SetBlocking(hSock);              // Make sure it is blocking again, to calls to recv() and send()
block
        break;
      }
      //
      // We need to close a socket and reopen a new one if a timeout occurs,
      // otherwise the OS sometimes does not recognize that
      // the port became available in between
      //
      SYS_SOCKET_Close(hSock);
    } while (r <= 0);
    //
    // Read from TELNET connection
    //
    do {
      r = SYS_SOCKET_Receive(hSock, ac, sizeof(ac) - 1);
      if (r <= 0) {                                 // 0 == Connection gracefully closed by server. < 0 == Some other
error
        break;
      }
      ac[r] = 0;
      printf("%s", ac);
    } while (1);
    SYS_SOCKET_Close(hSock);
  } while (1);
ErrHandler:
}
if (r <= 0) {                                  // 0 == Connection gracefully closed by server. < 0 == Some other error
        break;
      }
      ac[r] = 0;
      printf("%s", ac);
    } while (1);
    SYS_SOCKET_Close(hSock);
  } while (1);
ErrHandler:
}
```

トラブルシューティング

9.1　省電力モードによるデバッグユニットの無効化
9.2　その他の問題

9

9. トラブルシューティング

9.1 省電力モードによるデバッグユニットの無効化

多くの CPU コアは、イベントまたは割り込みの待機中に電力を節約するために、低電力モードをサポートしています。たとえば、Cortex-Mでは、これはWFI命令とWFE命令を実行することによって実現されます。低電力モードでの正確な動作は実際のデバイスによって異なります(同じCortex-Mコアを組み込んだ異なるベンダーの2つのデバイスは動作が異なる場合があります)。

一部のデバイスは、低電力モード中にデバッグユニット全体を無効にし、J-Linkがデバイスへの接続を失う原因となります。
デバッグユニットを有効にしたまま、内部フラッシュとRAMを無効にするものもあり、ターゲットが低電力モードのときにJ-LinkがRTT要求を実行することが事実上不可能になります。

ほとんどの場合、ターゲットはアイドル状態になるため、低電力モードを使用すると、アプリケーションの実行中にターゲットメモリに99%の時間アクセスできないため、RTTの使用が事実上不可能になります。ターゲットアプリケーションを停止すると、デバイスがメイクアップされ、RAMなどが再びアクセス可能になります。
これにより「RTT出力はターゲットが停止する場合にのみ表示される」などの効果と見なすことができます。

解決：RTT を使用する場合は、低電力モードが使用されていないことを確認してください。

9.2 その他の問題

RTT をセットアップで動作させるのに問題がある場合は、次のことを試してください。

1. ターゲット・デバイスがRTTをサポートしていることを確認ください。「2.RTTモード」を参照してください。

2. アプリケーションが各バッファに対して正しい RTT モードを使用していることを確認ください。注: SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL を除くすべてのモードは、ターゲットがホストが取得するよりも速くRTTデータを書き込むと、RTT データが失われる可能性があります)。

3. RTT データをフェッチするアプリケーションがホスト側に 1 つだけあることを確認ください。複数のアプリケーション(J-Link RTTビューアやJ-Link RTTクライアントなど)を並行して実行すると、一方が他方からデータを取得する可能性があり、RTTログが断片化する可能性があります。

4. RTT 制御ブロックが J-Link DLL によって検出できるか確認ください。
   自動検出を機能させるには、J-Linkに正しいターゲット・デバイス名を渡す必要があります(ターゲット・コア名だけでは自動検出には不十分です)。
   注: 多くのデバイスでは、使用可能な RAM の一部のみが J-Link DLL で指定されます。RTT 制御ブロックがこの指定された領域の外側にある場合、自動検出機能は機能せず、RTT 制御ブロックが配置されている RAM 領域のアドレスまたは検索範囲を指定する必要があります。
   RTT 制御ブロックのアドレスは、J-Link コマンド文字列を使用するか、J-Link RTT ビューアを使用する場合は、設定ダイアログで設定できます。
   J-Link が RTT 制御ブロックを検索するためのアドレス範囲は、J-Link コマンド文字列を使用するか、J-Link RTT ビューアを使用する場合は、設定ダイアログで設定できます。

FAQ

よくあるご質問と回答

10

J-LinkはRTTバッファをどのように見つけますか?

【回答】
2通りの方法がございます。デバッガ（IDE）がSEGGER RTT Control Blockのアドレスを認識している場合、RTTバッファをJ-Linkに渡すことができます。SEGGER RTTに対応していない別のアプリケーションが使用されている場合、J-Linkはバックグラウンドでアプリケーションを実行中に既知のターゲットRAM内のIDを検索します。このプロセスは通常、ほんの1秒に満たない時間内で完了し、プログラムの実行が遅延することはございません。

RAMのみのアプリケーションをデバッグしています。J-LinkはRTTバッファを見つけますが、出力はありません。このような場合、できることは何ですか？

【回答】
アプリケーションのinitセクションがRAMに格納されている場合、J-Linkはデータセクションの実際のものではなく、initセクションのブロックを誤って識別することがあります。これを防ぐには、define SEGGER_RTT_IN_RAMを1に設定します。J-Linkはすぐに正しいRTTバッファを見つけますが、アプリケーション内の最初のSEGGER_RTT関数を呼び出した後でなければなりません。アプリケーションの開始時にSEGGER_RTT_Init()を呼び出すことをお勧めします。

RTTはSWOピンを持たないターゲットでも使用できますか？

【回答】
はい、デバックインターフェースが使用されます。これは、ほとんどのCortex-Mデバイスの JTAG又はSWD（2ピンのみ）、一部のルネサスデバイスのFINEインターフェース、Infineon SPDインターフェース（シングルピン）とも、同じです。

RTTはCortex-M0とM0 +でも使用できますか？

【回答】
はい。

SWOが初期化されていないことから、デバッガ又は停止せずにハードフォールトをトリガするソフトウエアブレイクポイントを使用するため、デバッグ環境外で実行すると、一部のターミナル出力（printf）ソリューションは、プログラムの実行を「クラッシュ」します。そのため、スタンドアロンモードでデバックビルドを実行することは不可能です。SEGGER-RTTはどうですか?

【回答】
SEGGER-RTTはデフォルトで非ブロッキングモードを使用します。つまり、デバッガが存在せず、J-Linkが接続されていない場合でもプログラムの実行を停止しません。 アプリケーションプログラムは引き続き動作します。

アプリケーションでRTTを使用することは間違っていないのですが、出力はありません。このような場合、何を調べますか？

【回答】
場合によっては、J-Linkは既知のRAM領域にRTTバッファを配置できません。この場合、可能な領域又は正確なアドレスは、J-Link実行コマンドによって手動で設定することができます。

- RTTバッファを検索する範囲を設定します。
  SetRTTSearchRanges <RangeStart [Hex]><RangeSize >[, <Range1Start [Hex]> <Range1Size>, …] （例 “SetRTTSearchRanges 0x10000000 0x1000, 0x2000000 0x1000”）
- RTTバッファのアドレスを設定します。
  SetRTTAddr <RTTBufferAddress [Hex]> （例“SetRTTAddr 0x20000000”）
- J-Linkコントロールパネル→ターミナル経由でRTTバッファのアドレスを設定します。

注意
J-Link実行コマンドは、ほとんどのアプリケーションで実行できます。
例えば、J-Link Commanderは「exec <Command>」経由、J-Link GDB Serverは「monitor exec <Command>」経由、IAR EWは「__jlinkExecCommand(”<Command>“);」経由でマクロファイルから実行できます。

J-Link製品購入

エンビテックウェブショップ

https://www.embitek.shop/

J-Link / J-Trace PRO各種を**2-3**営業日以内の発送にて対応

■J-Link製品選択について

J-Linkシリーズの製品選択については、以下PDF資料、Youtube動画を用意していますのでご参照ください。

J-Link製品資料(PDF)：
https://www.embitek.co.jp/download/ps/jlink.pdf
J-Link製品選択について(Youtube動画)：
https://www.youtube.com/watch?v=uCUI-4c3YHo