J-Link Debugger 「OZONE」

J-Link PLUS以上で利用可能なデバッグソフトウェア

EmbiTeK

J-Link Debugger 「OZONE(オゾン)」

J-Link Debugger「OZONE(オゾン)」はJ-Link PLUS以上の製品に標準バンドルされているデバッグソフトウェアです。

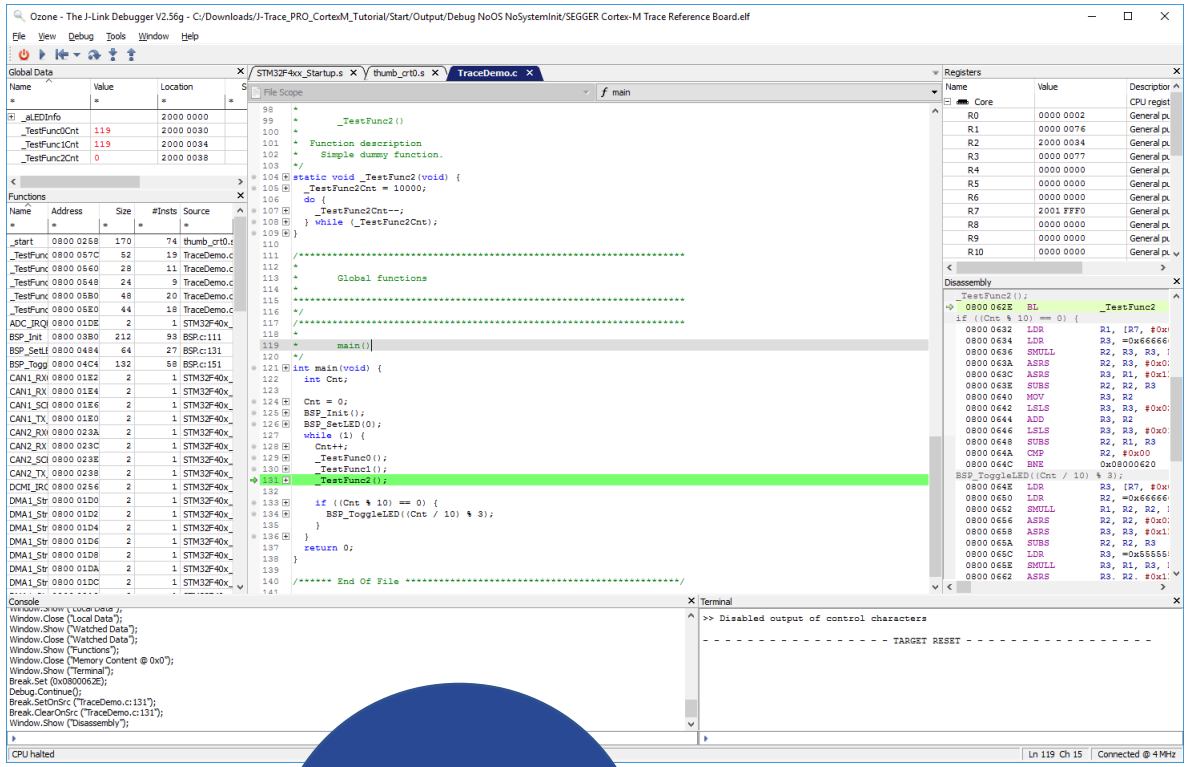
【標準無償バンドル】
J-Link PLUS / J-Link PLUS Compact / J-Link ULTRA+ / J-Link PRO
J-Trace PRO for Cortex / J-Trace PRO for Cortex-M

【別途有償ライセンス】
J-Link BASE / J-Link BASE Compact

デバッグソフトウェア・パフォーマンス分析ツールとして利用

トレース、コードプロファイリング、コードカバレッジ分析などの様々な機能により、
お客様アプリケーションの問題解析から、パフォーマンス分析を行う事で、アプリケーションの非効率な動きと問題点を洗い出し、お客様のソフトウェアをさらに価値のあるものとするサポートができます。

J-Link Debugger 「OZONE(オゾン)」



J-Link RTT
フルサポート

スタンドアロンデバッグソフトウェア

様々なツールチェーン、コンパイラから出力したファイルをデバッグ可能
※GCC / Clang / ARM / IARコンパイラでテストを行っています。

C/C++ソースコードレベルデバッグ
アセンブラ命令デバッグ対応

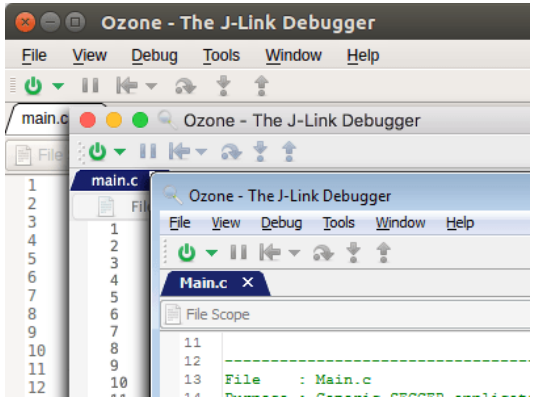
ソースエディタを同梱し、発見したバグを修正可能

J-Linkを利用して、高速書込

J-Linkのフル機能を利用可能

プロジェクトウィザードで簡単に立ち上げ

汎用性の高いデバッグソフトウェア



クライアントOSの依存性なく利用可能

開発者様は、Windows、Linux、MacOSのいずれかのプラットフォーム上で、OZONEを利用する事ができます。

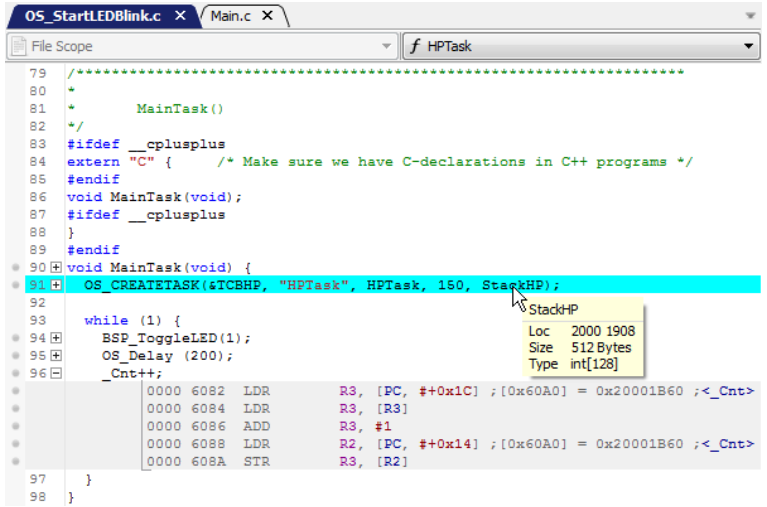
様々なツールチェーン・コンパイラをサポートします。

- **GCC-based IDEs**
  - SEGGER Embedded Studio
  - Eclipse
  - makefile projects
- **clang/LLVM**
  - SEGGER Embedded Studio
- **IAR ICCARM**
  - EWARM V6, EWARM V7, EWARM V8
- **ARM Compiler**
  - Keil uVision V4 / Keil uvision V5

※SEGGER社動作確認済みコンパイラ・IDE

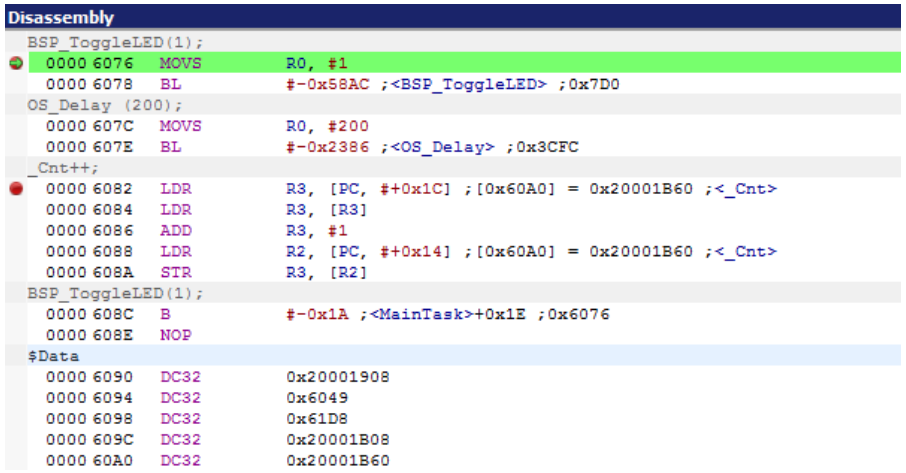
デバッグ機能

デバッグ機能



ソースコードビューワ

ソースコードビューワはターゲットアプリケーションの動きに合わせてナビゲーションします。アクティブなソースコードのハイライト表示。ブレイクポイントの設定、アプリケーション実行開始位置の設定などが可能です。
個々のソースコード行から関連するアセンブリコードを表示する事ができます。



逆アセンブリ

逆アセンブリでステップ実行におけるアセンブリ命令レベルでプログラムの実行追跡ができます。
ソースコードとシンボル情報でアセンブリ命令と関連する情報を表示。双方を個別に表示、非表示を選択することができます。

デバッグ機能

Breakpoints

State	Location	Permitted Impl.	Actual Impl.	Context	Skip #
☑	OS_StartLEDBlink.c:94	Any	Hard	BSP_ToggleLED(1);	0
☑	0000 6076	Any	Hard	MOVS R0, #1	0
☑	0000 608C	Any	Hard	B #-0x1A ;<MainTask	0
☐	OS_StartLEDBlink.c:96	Any		_Cnt++;	0
☐	0000 607C	Any		MOVS R0, #200	0
☑	0000 6088	Any	Hard	LDR R2, [PC, #+0x14]	0

Data Breakpoints

On	Address	Address Mask	Symbol	Access Type	Access Size	Match Value	Value Mask
☐	2000 0EFC	0000 0000	OS_InTimer	Write Only	Auto Access	0x0	FFFF FFFF
☑	2000 1B60	0000 0000	_Cnt	Read Only	Word	0x0	FFFF FFFF
☑	2002 FFFC	0000 0000		Read/Write	Auto Access	0xAC	0000 00FF

Global Data

Name	Value	Location	Size	Type
OS_	*	*	*	*
⊞ OS_COM_pTask	0x0	2000 0EEC	4	volatile struct OS_TASK_STRUCT*
⊞ OS_Global		2000 00C8	72	struct OS_GLOBAL_STRUCT
OS_InitCalled	1 ('\001')	2000 0EFD	1	uchar
OS_InitialSuspendCnt	0 ('\0')	2000 0EF2	1	uchar
OS_InTimer	0 ('\0')	2000 0EFC	1	uchar

Watched Data

Expression	Value	Location	Size	Refresh	Type
OS_Global.Time	25 790	2000 00E8	4	2 Hz	volatile long
OS_Global.Time / 1000	25	const	8	2 Hz	long long
StackHP[3]	CDCD CDCD	2000 1914	4	2 Hz	int
⊟ TCBHP		2000 1B08	88	2 Hz	struct OS_TASK_STRUCT

Local Data @ SetMember2

Name	Value	Location	Size	Type	Access
⊟ this	2000 4374	2000 4354	4	const class Class1*	
⊟ [0]		2000 4374	12	const class Class1	
⊞ pNext	0x0	2000 4374	4	class Class1*	protected
Member1	0xA	2000 4378	4	int	private
Member2	0	2000 437C	1	bool	private
StaticMember		<null>	4	int	private

ブレイクポイント

ソース、命令、データなど様々なタイプのブレイクポイントを設定し、各ブレイクポイントでアプリケーションを制御する事が可能です。
「J-Link Flash Breakpointテクノロジー」を使用することで、ハードウェアの制限を超えて、必要な数だけ、ブレイクポイントを追加できます。RAMで実行するため、アプリケーションコードを変更することはありません。ブレイクポイントはそれぞれで条件設定やマクロで制御する事も可能です。

シンボル監視ウィンドウ

変数と関数パラメータを監視・編集する事ができます。他のウィンドウかやソースコードビューワからシンボルをドラッグアンドドロップするか、手動で入力します。
アプリケーションの実行中に定期的に更新する事ができます。
（ターゲットがサポートしている場合）

デバッグ機能

Registers @ MainTask

Name	Value	Description
Core		CPU registers
R0	0000 0000	General purpose register 0
R1	0000 0000	General purpose register 1
R2	0000 85C4	General purpose register 2
R3	2000 0F0C	General purpose register 3
R4	0000 0000	General purpose register 4
R5	0000 0000	General purpose register 5
R6	0000 0000	General purpose register 6
R7	0000 0000	General purpose register 7
R8	0000 0000	General purpose register 8
R9	0000 0000	General purpose register 9
R10	0000 0000	General purpose register 10
R11	0000 0000	General purpose register 11
R12	2003 0000	General purpose register 12
R13	2002 FFD0	Stack pointer
R14	0000 20BD	Link register
R15	0000 3290	Program counter
xPSR	6100 0000	Program status register
N	b'0	Negative flag
Z	b'1	Zero flag
C	b'1	Carry flag
V	b'0	Overflow flag

CPUレジスタ

基本的なCPUレジスタに加えてメモリマップされたペリフェラルレジスタ（SFR）を表示します。
全てのレジスタはグループ化し表示、アプリケーションが停止された際に、直近変更されたレジスタ値がハイライト表示されます。

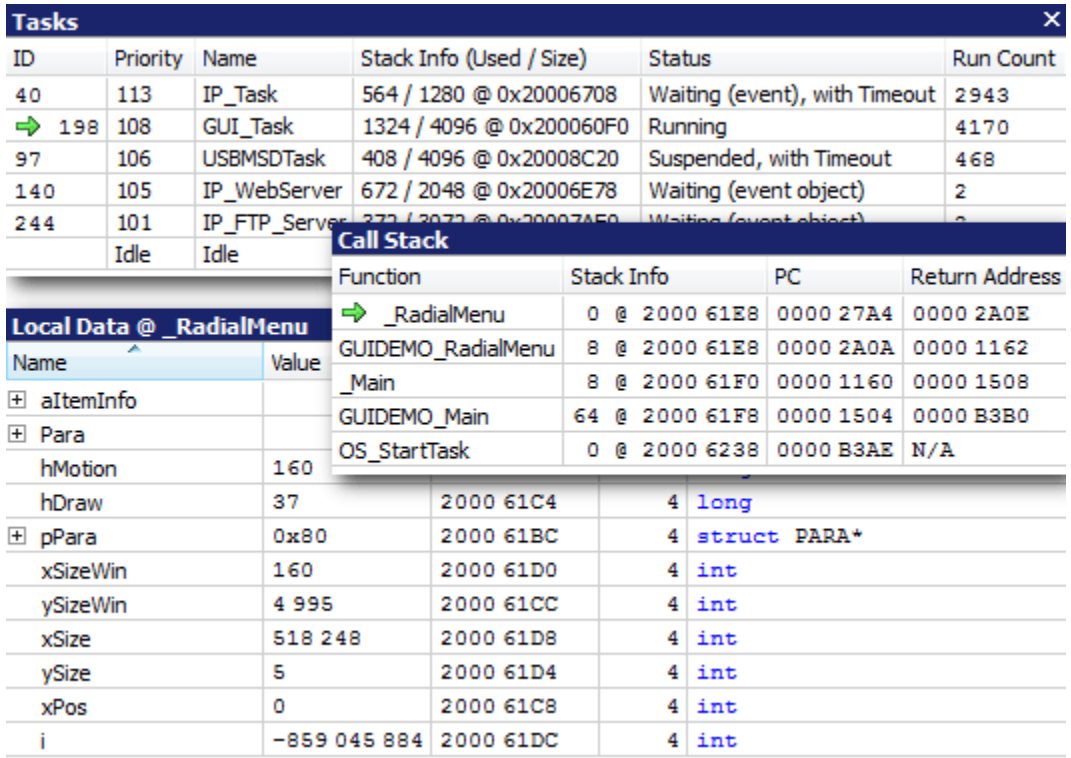
Terminal

```
- - - - - - - - - - - - - - - - - - TARGET RESET - - - - - - - - - - - - - - - - - -
>> Output via SWO active
>> Semihosting IO active
>> RTT active
MainTask   (00.000s) > Booting...
MainTask   (00.002s) >   System Initialization...
MainTask   (00.014s) >   OK.
MainTask   (00.015s) >   Memory Initialization...
MainTask   (00.251s) >   OK.
MainTask   (00.252s) >   Memory Self Test...
MainTask   (01.116s) >   OK.
MainTask   (01.117s) >   System Self Test...
MainTask   (01.350s) >   OK.
MainTask   (01.352s) > System booted.
```

ターミナルI/O

ターゲットアプリケーションから、printf()やJ-Link RTTを使ったI/Oをサポートし、テキストメッセージを送受信します。

RTOS認識デバッグ



RTOS認識デバッグ

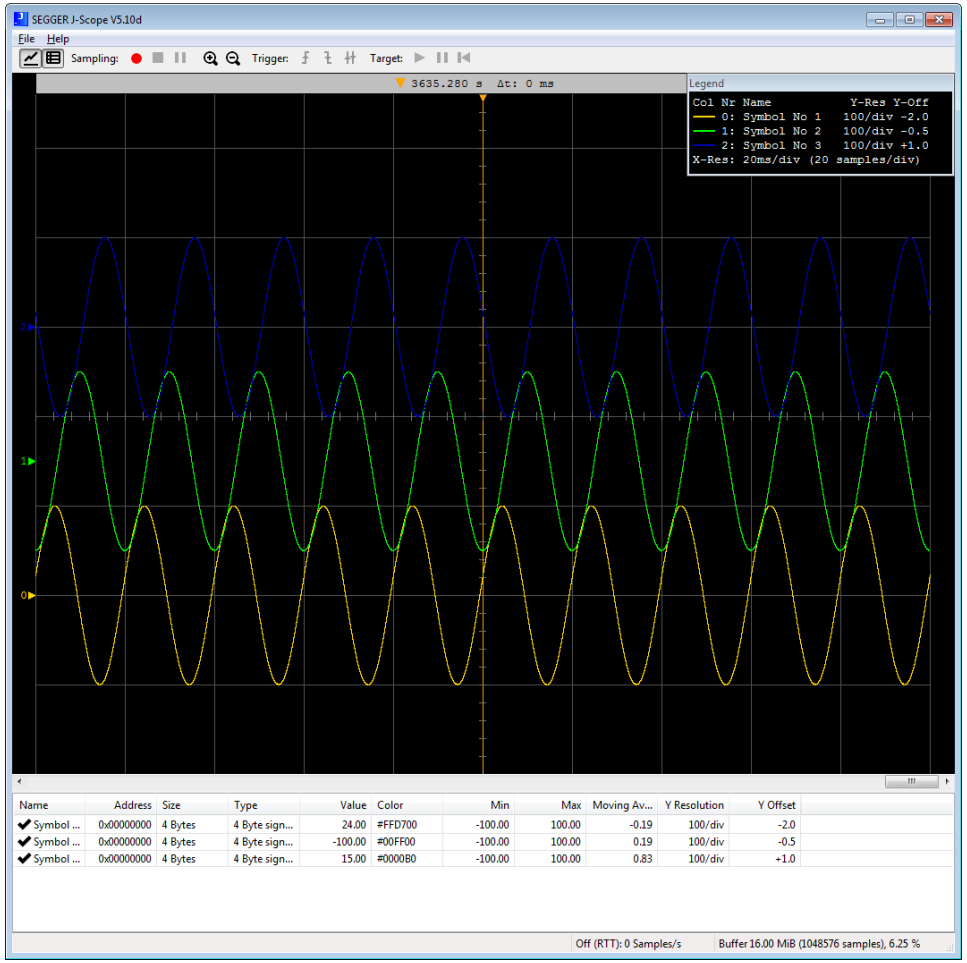
SEGGER embOS、FreeRTOSのOSタスク状態などを
表示することができます。

RTOSプラグイン SDK

他のお客様がご利用中のRTOSで利用される場合、SDKを
利用して、プラグインを開発することも可能です。

https://www.segger.com/products/development-tools/ozone-j-link-debugger/technology/rtos-awareness/

RAMモニタリング（J-Scope機能を同梱）



RAMモニタリング

ターゲットが動作している間、リアルタイムでRAM上のデータを分析し可視化するためのソフトウェアです。
複数の変数の値をオシロスコープのようなスタイルで表示できます。
ELFファイルを読み込み、視覚化する変数の数を選択することができます。

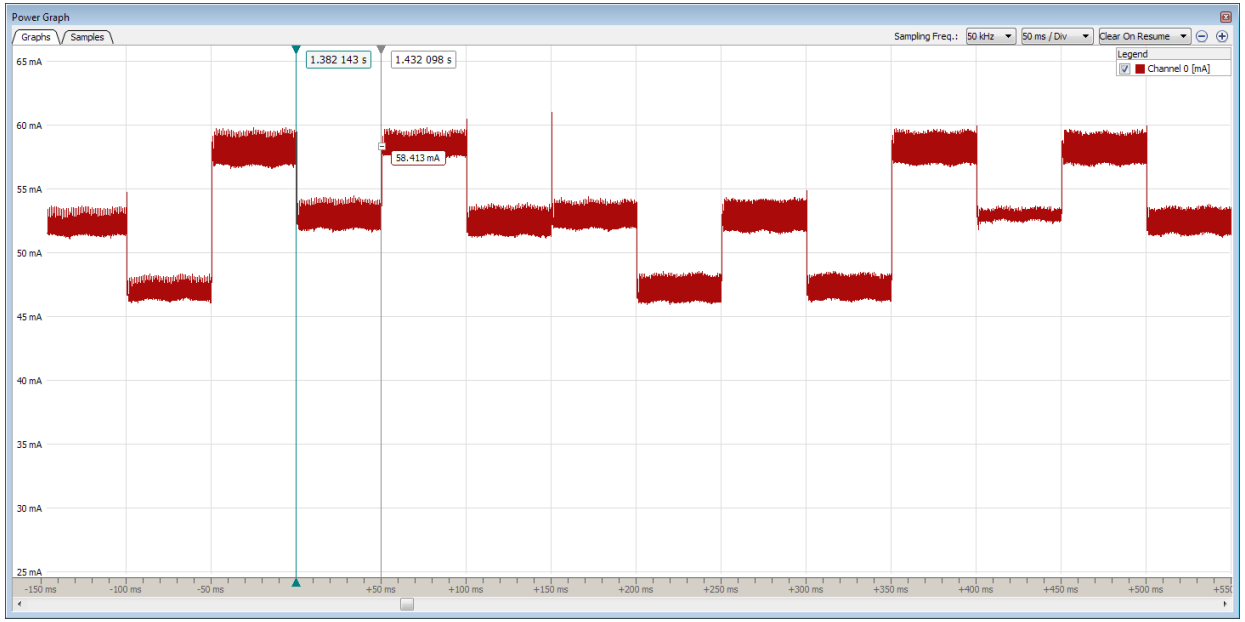
ハードウェアリソース不要

「J-Scope」はJTAG経由で必要な情報を取得します。
そのため特別なハードウェアは不要、新たに情報取得のためのピンを立てる必要はありません。

高速なデータ取得

J-LinkのRTTインターフェースを利用することで、
最小転送周期 ：28.5μSec（2-byte変数x8 (=16byte)の場合/ Cortex-M4 ）を実現します。

消費電力プロファイリング

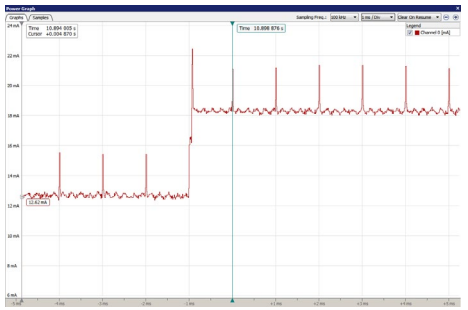


消費電力プロファイリング

J-LinkとOZONEの組合せで消費電力をモニタリングしながら、デバッグする事ができます。

SEGGERハイエンドデバッガ（J-Link ULTRA +、J-Link PRO、J-Trace PRO）は、50μAの解像度で最大100 kHz（200kSa / s）のサンプリングレートを提供します。

LED点滅サンプルアプリケーションの解析結果



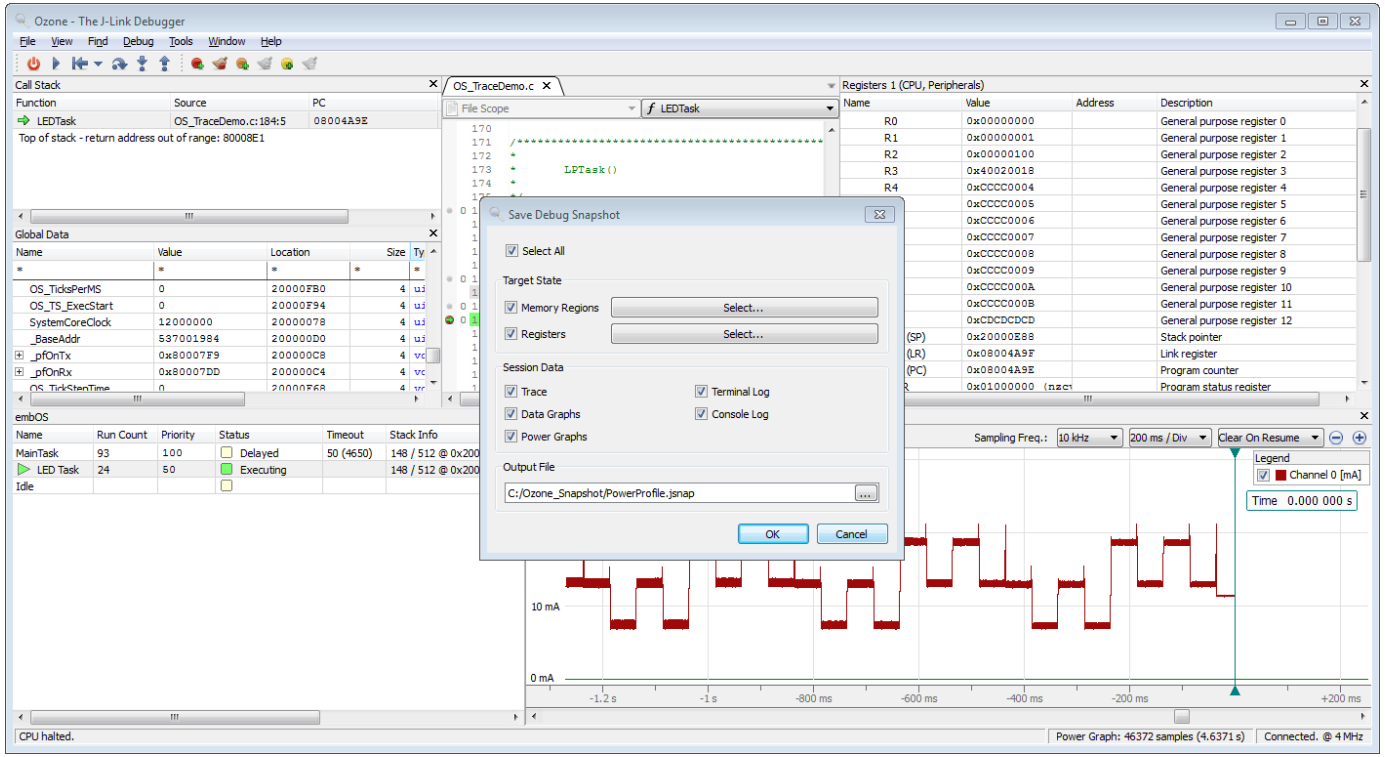
SYSTICK

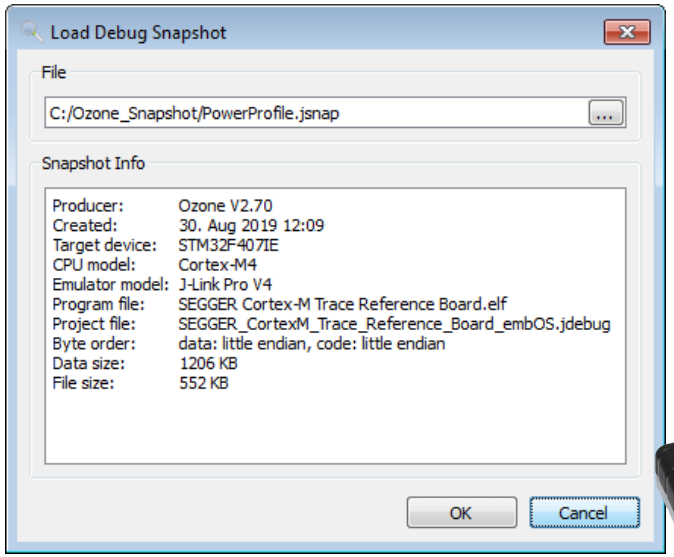
アイドル状態のシステムよりもわずかに多くの電力を必要とするため、消費電力プロファイルにより1msのSysTick割り込みも確認できます。

TASK SWITCH

ズームインすると、タスクがいつ実行され、LEDがオンになったかを特定することもできます。

スナップショット





スナップショット

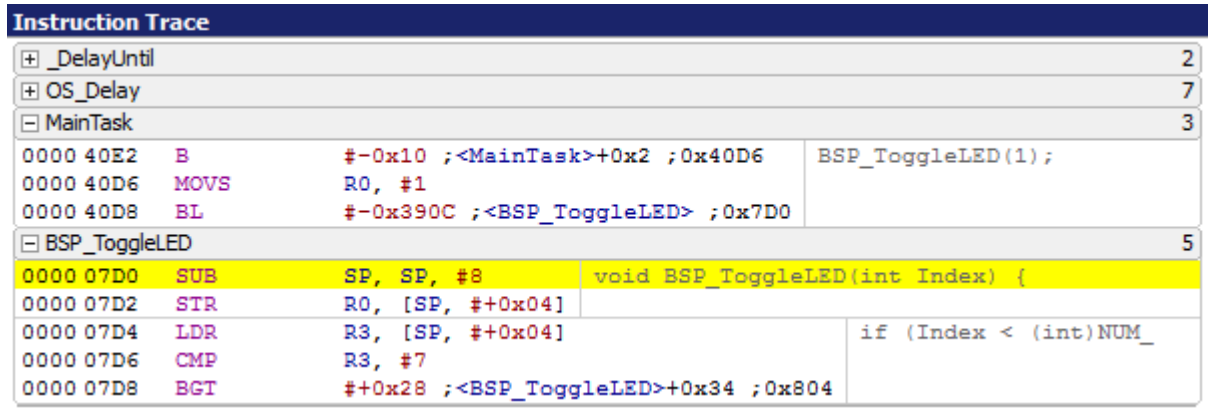
Ozoneのデバッグスナップショット機能を使用すると、システムの状態を任意の時点で保存できます。
スナップショットには、デバイスのROM/RAMの内容、CPUレジスタ値、命令トレースとログ出力、
および周辺レジスタ値などの現在のターゲットハードウェア状態を含めることができます。
スナップショットを保存し、保存したスナップショットをターゲットにリロードすることも可能です。

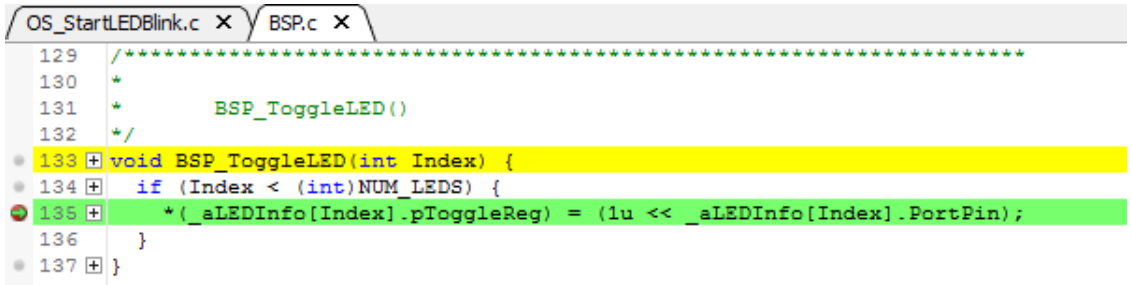
トレース機能

J-Trace PROとの組合せによるトレース機能

J-Trace PROを組み合わせて利用する事により、ターゲットシステムの実行動作検証とパフォーマンス分析をより精緻に行う事ができます。

命令トレース

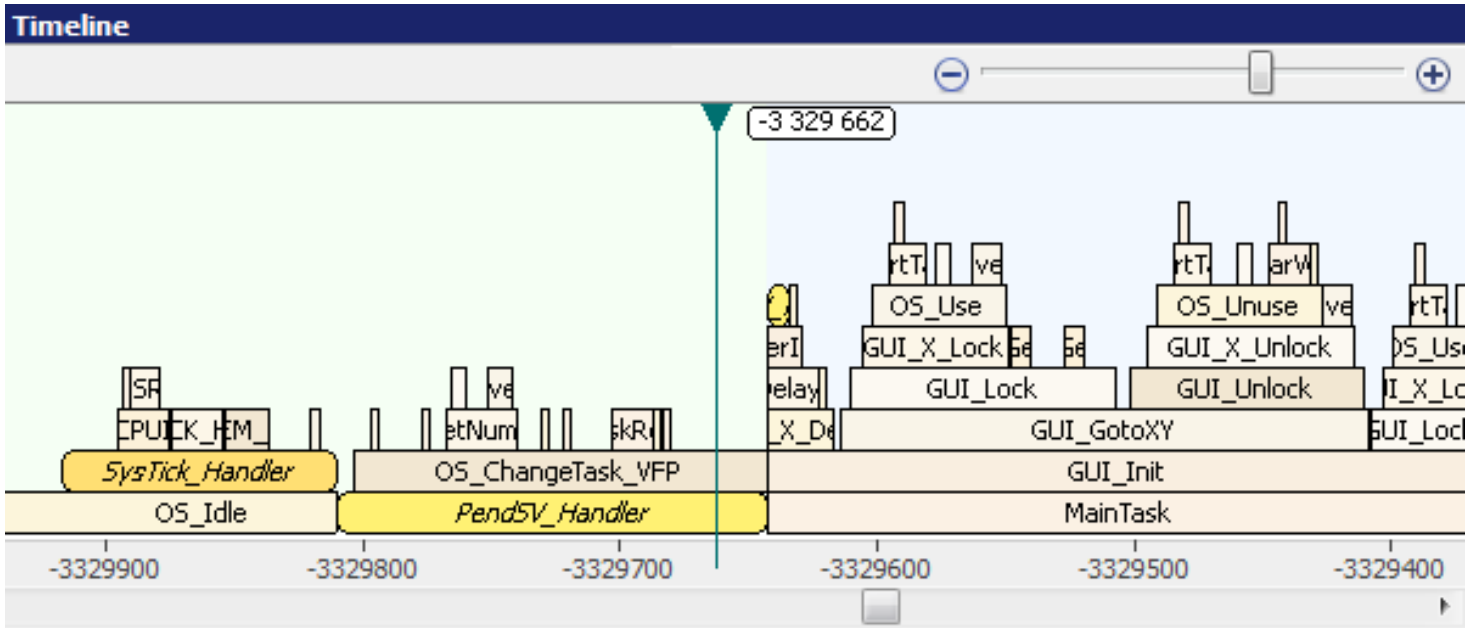




命令トレースでは、ターゲットシステムが最後に実行した命令を表示します。

また全てのプログラムの実行履歴が保存され、検証することができます。各命令は、フレームブロックとして、表示。ソースコードと逆アセンブリコードを並列して表示する事ができます。

タイムライン



タイムラインウィンドウでは、コールスタックを時系列でグラフィカルに表示します。

タイムラインは、ソースコード情報にマッピングされた記録済みトレースデータに基づいて、表示。

関数の呼び出しにかかった時間、呼び出されたサブルーチン、サブルーチンで費やされた時間を計測できます。

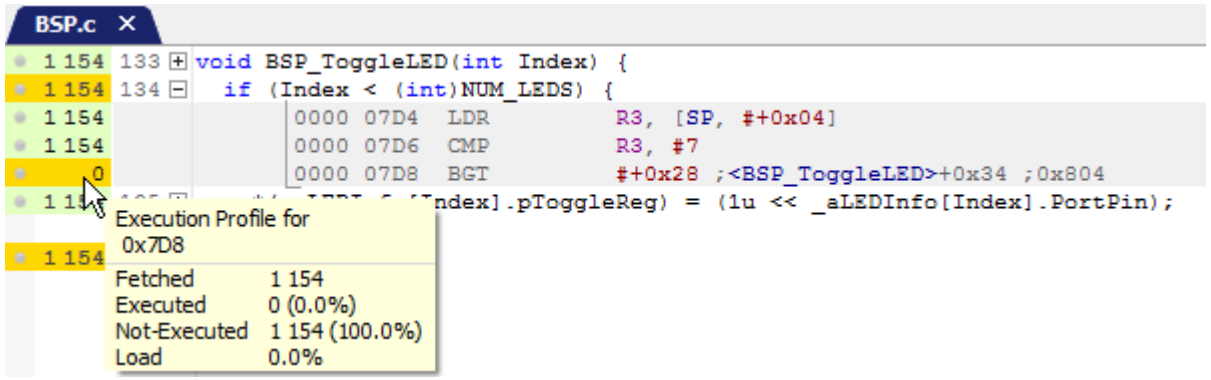
プログラムフローの不連続性（割込やコンテキストスイッチ）は、重要なイベントとして、識別できる様に、色を変えて、強調表示されます。

embOSを使用している場合、OSレベルでタスクスイッチ、タスクの実行状況を確認することができます。

コードプロファイリング

Code Profile

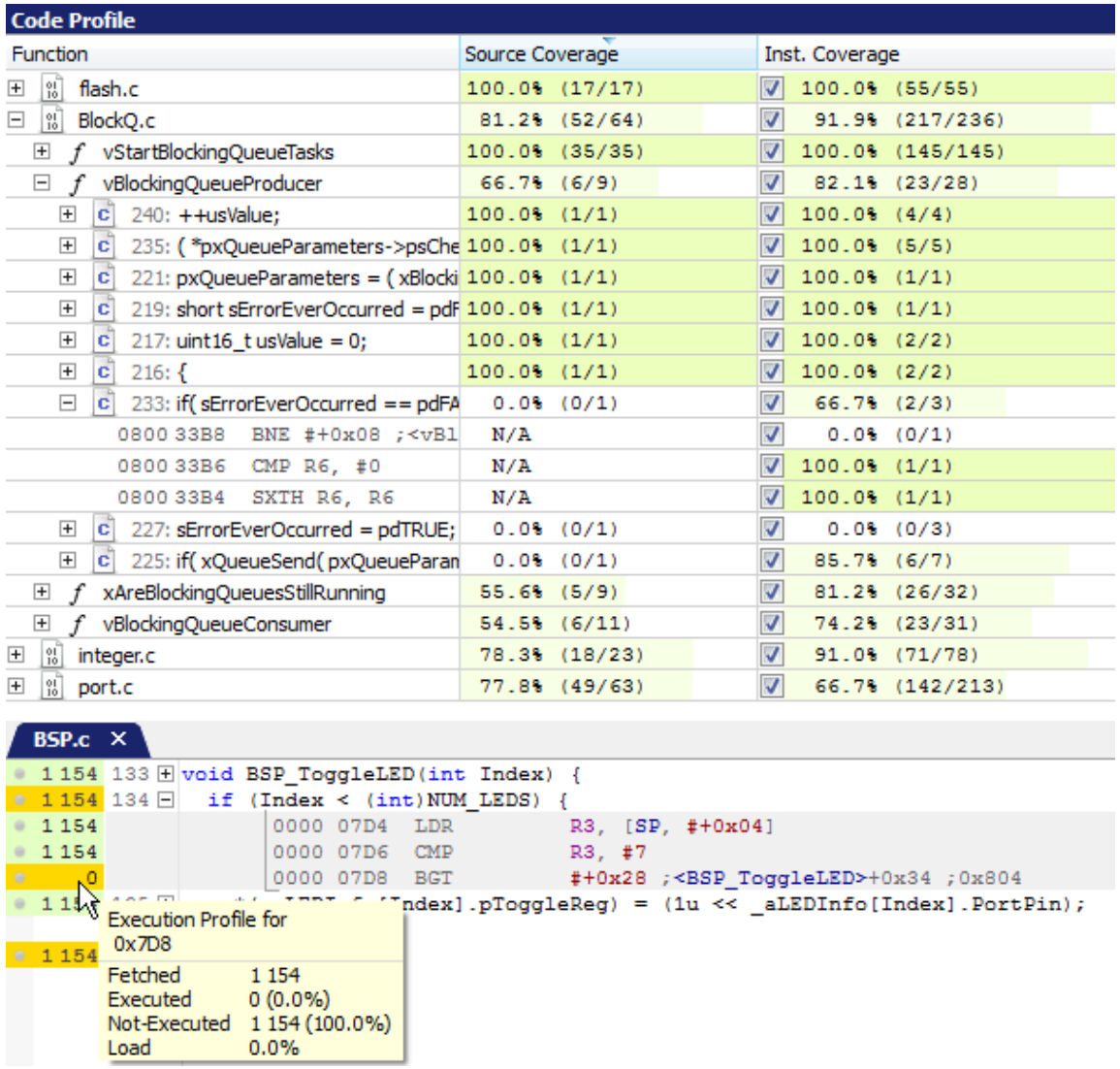
Function	Load	Run Count
sp_flop.c	43.74%	10
vCompetingMathTask3	31.88%	2
vCompetingMathTask4	30.05%	2
vCompetingMathTask1	20.25%	2
vCompetingMathTask2	17.82%	2
xAreMathsTaskStillRunning	0.00%	1
vStartMathTasks	0.00%	1
RegTest.s	18.49%	912 504
port.c	13.38%	2 211 153
queue.c	12.96%	1 167 960
tasks.c	4.79%	1 065 024



J-Trace PROストリーミングトレース機能を使用すると、トレースデータは、リアルタイムで分析され、OZONEで表示されます。ターゲットで実行された命令、条件付き命令が両方のパスを使用したかどうか、各命令の実行頻度などの情報を表示。

プロファイルされたデータを検証して、最適化できる可能性を発見し、アプリケーションの最適化のサポートを行う事ができます。

コードカバレッジ



J-Trace PROとOZONEの組合せで、コードカバレッジ機能を提供します。

コードカバレッジ	対応
C0	標準対応
C1	標準対応
C2	標準対応
MMC	ユーザスクリプトにより
MC/DC	未対応

RTTテクノロジー

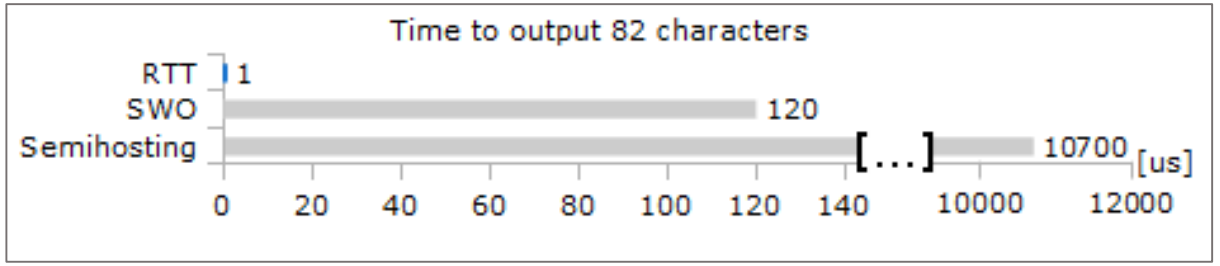
Real Time Transfer（RTT）は、組み込みアプリケーション
におけるインタラクティブなユーザー通信のテクノロジです。
非常に高いパフォーマンスで SWO とセミホスティングの長所を
兼ね備えています。

RTTアウトライン

J-Linkだけが実現する高速転送技術

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

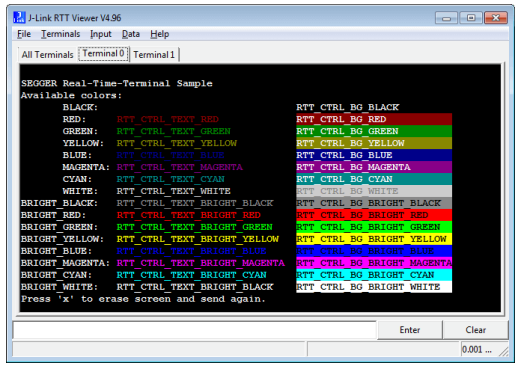
タイムラグのないデータ取得・送信
複数のチャンネルを双方向で利用可能



リソース	使用量
ROM	500 Byte以下
RAM	24Byte + バッファ24Byte /チャネル

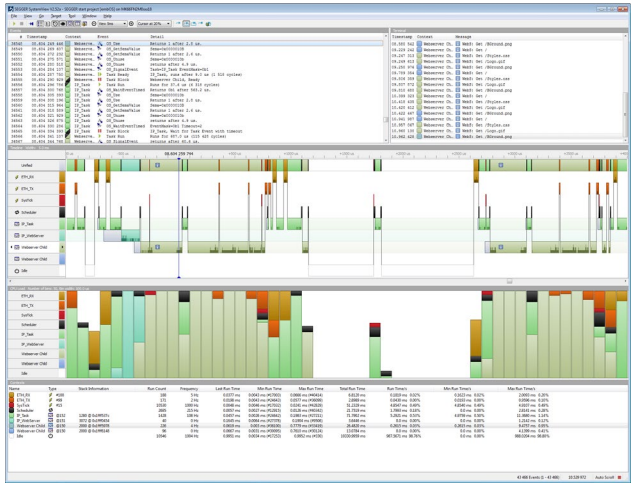
RTT対応ソフトウェアツール

RTTは、OZONE以外にも様々な対応ソフトウェアで利用可能です。



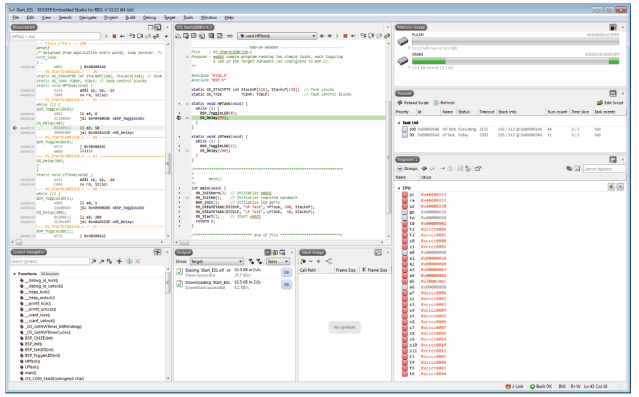
J-Link RTT Viewer（無償）

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。



SystemView（無償）
SystemView PRO（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

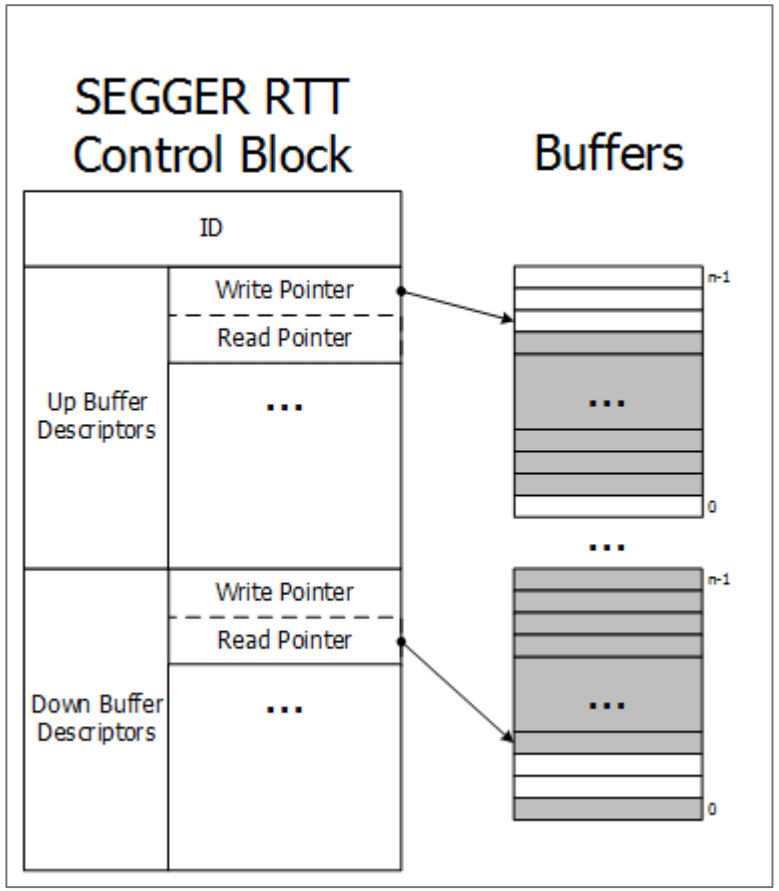


SEGGER EmbeddedStudio（有償）

Arm Cortexシリーズをサポートする
統合開発環境

J-Link RTTの構造

SEGGER RTT Control Block



「アップバッファ記述子」（ターゲット->ホスト）
「ダウンバッファ記述子」（ホスト->ターゲット）
これらの各バッファサイズ、数は個別に設定できます。
バッファ内の灰色の領域は、有効なデータを含む領域です。
アップバッファの場合、書込みポインタはターゲットマイコンによって書込まれ、読取りポインタはデバッグプローブ（J-Link、ホスト）によって書込まれます。

読取りポインタと書込みポインタが同じ要素を指す場合、バッファは空の状態になります。
これにより、競合状態が発生しないことが保証されます。

J-Link RTTは、標準のデバッグポートを介して、ホストPCとターゲットの通信を行います。利用にあたって追加のピンやハードウェアの必要性はありません。デバッグセッションと並列し、RTT通信を実行可能です。

J-Link RTTの実装構成

RTT実装コードはANSI Cで記述されており、簡単なコードを追加するだけで、組込アプリケーションに実装できます。

**RTTはターゲットメモリ内の「SEGGER RTT Control Block」を利用して、
データの読取り、書込みを管理します。
このRTT Control Blockには、接続されたJ-Linkがメモリ内で検索できるようにするIDと
各チャネルのバッファとその状態が記録されます。**

RTTの各チャネルは、使用する局面に応じ、「ブロッキング」または「非ブロッキング」いずれかに設定可能です。

「ブロッキング」モード

バッファがいっぱいになると全てのメモリが書き込まれるまでアプリケーションは待機します。
そのためアプリケーションはブロックされた状態になりますが、データが失われる事はありません。**開発時にのみ利用ください。**

「非ブロッキング」モード

バッファに収まるデータのみ書き込まれ、残りは破棄されます。
そのためJ-Linkが接続されていない状態でもリアルタイム性を損なうことなくアプリケーションは実行されます。
リリース前テスト、リリースアプリケーションにそのまま残しておくことも可能です。

RTTの実装 – printf()の代用

標準printf()関数の代わりに利用

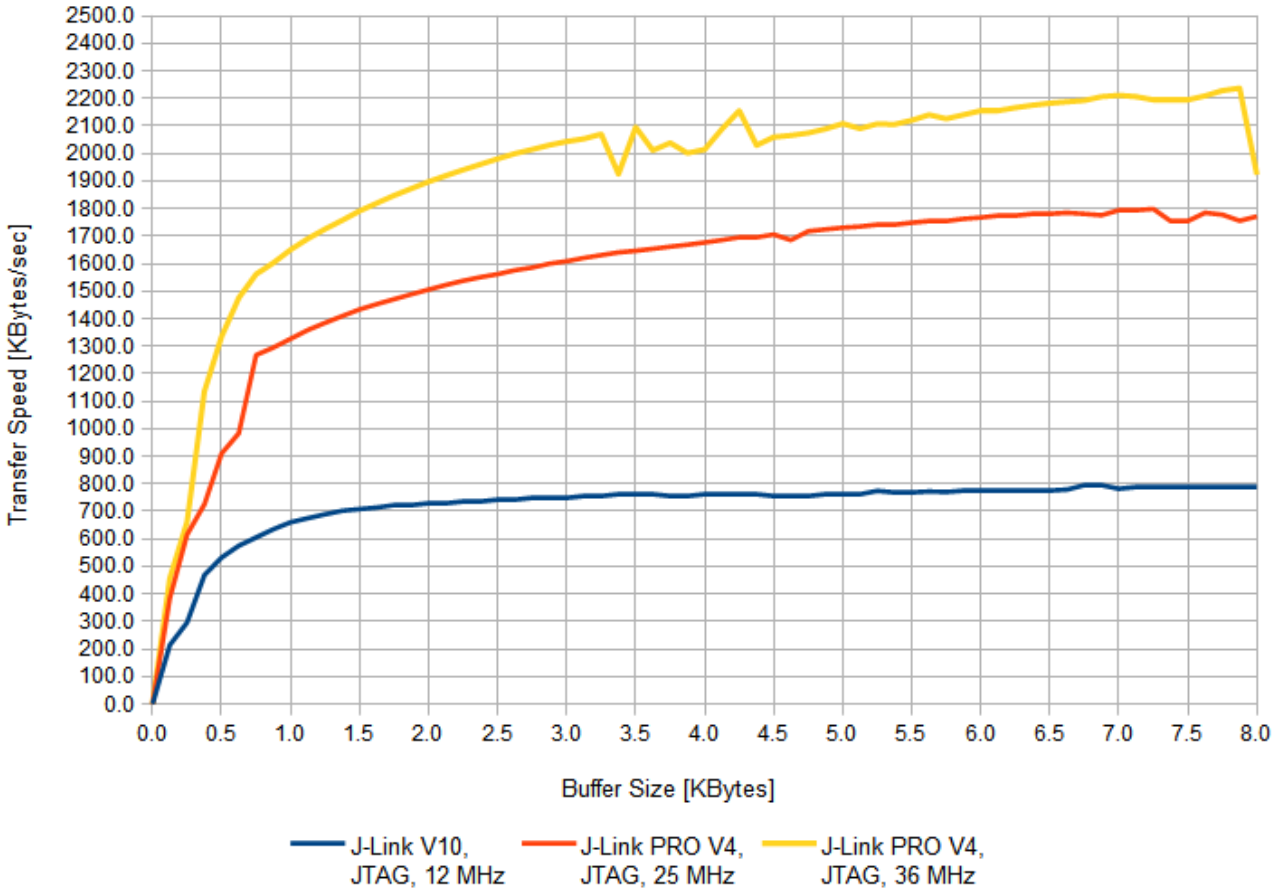
標準のprintf（）関数をオーバーライドしてRTTを使用することもできます。
RTTを使用すると、printf（）にかかる時間が最小限に抑えられ、アプリケーションが精緻なリアルタイムタスクを実行している間に、デバッグ情報をホストPCに出力できます。

SEGGER RTTの実装には、RTTを介してフォーマットされた文字列を書き込むために使用できるシンプルなprintf（）が含まれています。
このSEGGER_RTT_Printf（）は、標準ライブラリのprintf()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のみリソースを必要とします。

SEGGER RTT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。

RTT通信速度

出力データをホストに送信できる最大速度は、ターゲットの「**バッファサイズ**」と「**インターフェイス速度**」によって異なります。 512バイトの小さなターゲットバッファでも、J-Link PRO/Ultra+の高速インターフェースでは最大1 MByte / secのRTT速度が可能。通常のJ-Linkモデルでは0.5 MByte / secが可能です。



RTT API

Function Name	Description
SEGGER_RTT_Read()	インプット（ダウン）バッファよりデータを読み取ります。
SEGGER_RTT_Write()	アッププット（アップ）バッファへデータを書き込みます。
SEGGER_RTT_WriteString()	終端NullのC言語文字列をアッププット（アップ）バッファへ書き込みます。
SEGGER_RTT_printf()	書式設定された文字列をアッププット（アップ）バッファへ書き込みます。
SEGGER_RTT_GetKey()	インプット（ダウン）バッファ０から１文字を読み込みます。
SEGGER_RTT_HasKey()	インプット（ダウン）バッファ０に最低１文字書き込み可能かチェックします。
SEGGER_RTT_WaitKey()	インプット（ダウン）バッファ０に最低１文字書き込み可能になるまで待ちます。
SEGGER_RTT_ConfigUpBuffer()	名前、サイズ、およびフラグを指定して、アッププットバッファを設定又は追加します。
SEGGER_RTT_ConfigDownBuffer()	名前、サイズ、フラグを指定して、インプット（ダウン）バッファを設定又は追加します。
SEGGER_RTT_Init()	RTTコントロールブロックを初期化します。
SEGGER_RTT_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()	終端NullのC言語文字列を特定の「仮想」ターミナルに送信します。

各APIの利用方法については、J-Linkユーザマニュアル（英語）または当社日本語マニュアルを参照ください。

FAQ

Question.1
どのようにJ-Linkは、RTTバッファを取得するのですか？

Answer

J-Linkデバッガ「OZONE」やSEGGER EmbeddedStudioのように、RTTに対応したデバッガであれば、バッファ構造のアドレスをJ-Linkに渡します。RTTに対応していないデバッガであれば、プログラムの実行中にバックグラウンドで、ターゲットRAMからIDを検索特定します。ID文字列を特定するプロセスは一瞬で完了し、プログラムの遅延はほぼ発生しません。

Question.2
一部のターミナル出力、Printf()などは、デバッガが接続されていない環境で利用するとアプリケーションを停止させてしまうことがあります。そのためスタンドアロンデバッグ（リリース前デバッグ）を実行できませんが、RTTも同じでしょうか？

Answer

RTTは、デフォルトでは「非ブロッキングモード」を利用します。そのためデバッガが接続されていない環境、リリースバージョンで利用しても、アプリケーションを止めることはなく、RTTに起因する問題は発生しません。

FAQ

Question.3
RTTは、SWOピンに接続していなくても利用可能ですか？

Answer

はい、デバッグインターフェースピンを利用するので、不要です。Cotex-Mマイコンであれば、JTAG接続またはSWD(2pin)接続で利用可能です。デバイスメーカ専用のインターフェースでは、Infineon SPDインターフェース(1pin)の利用。ルネサスエレクトロニクス社RXの一部の製品では、「FINEインターフェース」を利用します。

Question.4
Cortex-M0/M0+でも利用可能ですか？

Answer

利用可能です。

FAQ

Question.2
RAMのみ利用するアプリケーションでデバッグする場合、J-LinkでRTTバッファの検出は行えましたが、出力を得ることができませんでした。どうすればいいですか？

Answer

アプリケーションのInitセクションがRAMに保存されている場合、J-LinkはInitセクションのブロックをご認識する可能性があります。これを防ぐために「define SEGGER_RTT_IN_RAM」を「１」に設定してください。J-Linkはアプリケーションで最初のSEGGER_RTT関数を呼び出した後にのみ、正しいバッファを検出します。アプリケーションの開始時に「SEGGER_RTT_Init（）」を呼び出すことを推奨します。

サンプル・資料など

RTTサンプル・マニュアル：J-Link Software & documentation Pack内

https://www.segger.com/downloads/jlink/JLink_Windows.exe

RTTサンプル>>インストールフォルダ¥SEGGER¥JLink_Vxxx¥Samples¥RTT

J-Linkマニュアル>>インストールフォルダ¥SEGGER¥JLink_Vxxx¥Doc¥Manuals¥UM08001_JLink.pdf

P.330 – 「RTT」

日本語マニュアルについては、J-Linkシリーズを当社日本語サポート対応サービスありでご購入のお客様はお問い合わせください。

J-Linkデバッガ OZONE：RTT対応デバッガ

https://www.embitek.co.jp/product/ozone.html

SystemView: RTT対応システム分析・記録ツール

https://www.embitek.co.jp/product/systemview.html

J-Link/J-Trace PROはエンビテック オンラインショップ

日本国内で最もローブライスでJ-Link / Flasher シリーズを提供

表示価格は当社サポート対応なし（英語サポート）ローコスト製品モデルです。

クレジットカード（VISA/MASTER/JCB）対応

代金引換（佐川eコレクト）対応

通常翌営業日発送対応

日本語サポート対応サービス

銀行振込請求書対応をご希望の場合は、
通常見積書を発行させて頂きますので、
お問合せください。

>> sales@embitek.co.jp

お問合せ窓口

お気軽に以下窓口へお問い合わせください。

株式会社エンビテック

TEL: 03-6240-2655
FAX　: 03-6240-2656
E-mail：sales@embitek.co.jp
https://www.embitek.co.jp

emWinが利用されています。

MADE IN GERMANY

emPack | embOS/IP | J-Trace | Mobile Networks | embOS-MPU | J-Link | emFile

Flasher ATE | emCompress | emModbus | emLoad | Flasher | emWin | emUSB-Host

embOS-Safe | Wifi Support | MQTT | SystemView | emCrypt | Embedded Studio | Dropbox Client

emSecure | embOS | emUSB-Device | emSSL | emSSH | Flasher Secure | IoT Toolkit

Appendix

Catalog Stand

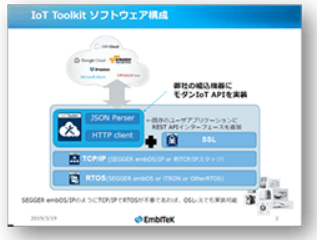
各種資料をダウンロード可能です。

総合カタログ
https://www.embitek.co.jp/download/MB_SolutionGuide.pdf

J-Linkカタログ
https://www.embitek.co.jp/download/MB-CTLG-JLink.pdf

Flasherカタログ
https://www.embitek.co.jp/download/MB-CTLG-Flasher.pdf

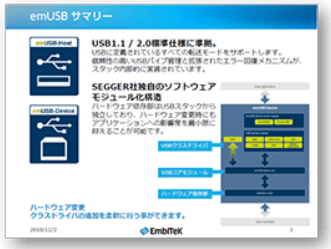
ソフトウェアライセンスについて
https://www.embitek.co.jp/download/MB_SWLicModels.pdf



IoT Toolkit製品資料
https://www.embitek.co.jp/download/ps/IoTtoolbox.pdf

圧縮解凍製品資料
https://www.embitek.co.jp/download/ps/emCompress.pdf

ファイルシステム製品資料
https://www.embitek.co.jp/download/ps/emfile.pdf



USB製品資料
https://www.embitek.co.jp/download/ps/SeggerUSBSolution.pdf