Renesas Synergy™ プラットフォーム

R11AN0195JU0100
Rev.1.00
2018.06.15

IAR Embedded Workbench® for Renesas Synergy™ スタック使用方法と使用量の監視・解析方法

本資料は英語版を翻訳した参考資料です。内容に相違がある場合には英語版を優先します。資料によっては英語版のバージョンが更新され、内容が変わっている場合があります。日本語版は、参考用としてご使用のうえ、最新および正式な内容については英語版のドキュメントを参照ください。

はじめに（Intoroduction）

本アプリケーションノートは、IAR Embedded Workbench® for Renesas Synergy™におけるスタック（stack）の設定・使用方法、およびランタイム時のスタック使用量（stack usage）の監視・解析方法について説明します。

スタックは、連続アドレスなメモリの固定ブロックであり、静的に割り当てられる必要があります。スタックは、C/C++関数のローカルデータ（local data）に加え、以下のものを含みます。

- レジスタ（register）に格納されないローカル変数（local variable）
- レジスタ（register）に格納されない関数パラメータ（function parameter）
- 式（expression）の一時的な結果
- 関数の戻り値（return value of function）（レジスタを通過した場合を除く）
- 割り込みコンテキスト（Interrupt context）
- 関数（function）が復帰する前に回復する必要があるプロセッサレジスタ（processor register）

スタックは2つの部分に区別されます。関数によって使用される割り当て済みメモリ（allocated memory）と、割り当て可能な空きメモリ（free memory）です。その境界はスタックの先頭（top of stack）と呼ばれ、通常、スタック専用のレジスタで管理されます。関数が使用するメモリは、スタックポインタ（SP：stack pointer）の移動によりその領域が割り当てられます。スタックに割り当てられたメモリは関数の復帰時に解放されます。このため、関数がスタックに割り当てたメモリは、関数の復帰以降、その内容は保持されません。

スタックの主要な利点は、アプリケーション内の複数の部分にある関数が同じメモリ空間を共有して、それぞれのデータを格納できることです。ヒープ（heap）とは異なり、スタックは断片化（fragmented）することもなく、またメモリリーク（memory leak）が発生することもありません。

システムの安定性（stability）と信頼性（reliability）を確保するためには、スタックを適切に構成（configuration）する必要があります。スタックサイズ（stack size）が小さすぎると、SPはスタック領域（stack area）からはみ出し、オーバーフローを発生する可能性があります。SPがスタック領域をはみだした場合（スタックが下方側に伸びる場合）には、実行中のプログラムにより、変数の上書き（overwritten variable）、ワイルドポインタ（wild pointer）、戻りアドレスの破損（corrupted return address）など、深刻な障害を発生させる可能性があります。逆にスタックサイズが大きすぎると、RAMリソースが浪費されることになります。

目次

1. 静的スタック使用量解析（Static stack usage analysis）

リンカ（linker）はコールグラフルート（call graph root）（他の関数から呼び出されない関数）ごとに最大スタック使用量を正確に計算することが可能です。スタック使用量チャプタ（stack usage chapter）はリンカマップファイル（.map）（linker map file）に追加され、そこには、各**コールグラフルート**の最も深度が深いコールチェイン（call chain）の最大深度値と、各**コールグラフルート**カテゴリ（**call graph root** category）の最も深度が深いコールチェインの最大深度の合計値がリストされます。アプリケーションの関数ごとのスタック使用量の情報が十分に得られると、スタック使用量を正確に計算することができます。

一般に、関数ごとのスタック使用量はコンパイラが算出します。ただし、場合によっては、開発者が追加のディレクティブ（directive）を指定して、再帰関数（recrusive function）の間接呼び出し（indirect call）（関数ポインタによる呼び出し）または最大反復回数（maximum number of iteration）をコンパイラに通知する必要があります。これを実行するには、ソースコードで#pragma directiveを使用するか、またはプロジェクトオプションダイアログで個別のスタック使用量制御ファイル（separate stack usage control）を指定します。

2. スタック使用量解析の有効化（Enable stack usage analysis）

以下のように、「**Linker**」オプションの「**Advanced**」タブで「**Enable stack usage analysis**」をチェックします。

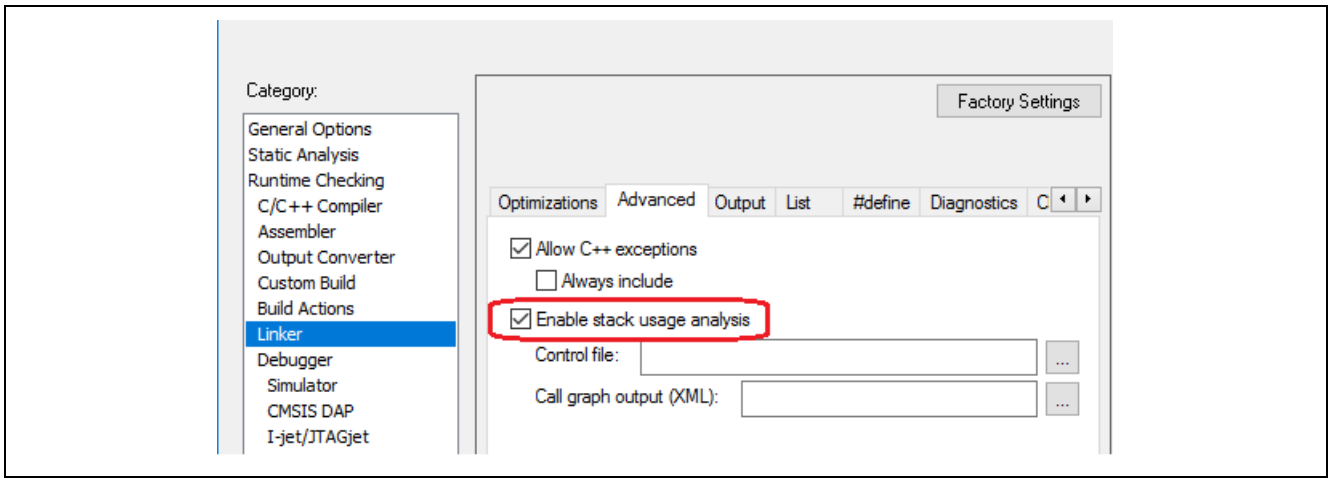

図1　　スタック使用量解析の有効化

リンカマップファイル（**linker map** file）を生成します。このファイルにスタック使用量解析（stack usage analysis）の結果が格納されます。このファイルは、「**Category**」ウィンドウの「**Linker**」オプションにある「**List**」タブで**有効にする**ことが可能です。

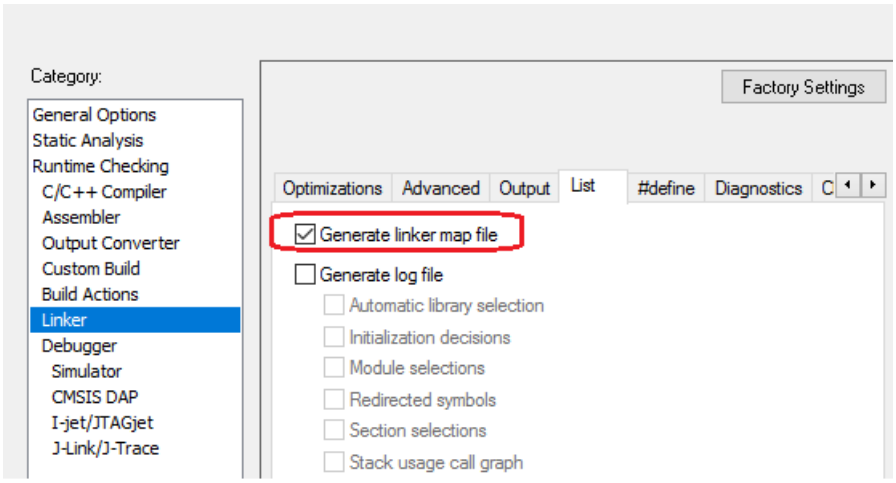

図2　「Generate linker map file」の有効化

簡単なアプリケーションの場合、スタック使用量解析の結果は容易に理解できます。プログラムエントリ（program entry）および割り込みハンドラ（interrupt handler）は、他の関数から呼び出されないため、**コールグラフルート**と見なされます。以下は、Synergyコンフィグレータが S7G2-SK Synergy MCUグループ向けに生成した簡単な**ブリンキー**プロジェクト（**Blinky** project）の例です。プログラムエントリの**コールグラフルート**カテゴリ（Reset_Handler）の最大スタック深度（maximum stack depth）は72バイトであり、呼び出されない関数のコールグラフルートカテゴリ（NMI_Handler、R_CGC_BusClockOutCfgandなど）の最大スタック深度は合計64バイトです。以下の例を参照してください。

```
********************************************************************************
*** STACK USAGE
***

  Call Graph Root Category  Max Use  Total Use
  ------------------------  -------  ---------
  Program entry                  72         72
  Uncalled function              64      1 340


Program entry
  "Reset_Handler": 0x00003fcd

  Maximum call chain                              *?* 72  bytes
    (** call graph contains indirect calls (example: "SystemInit") **)

    "Reset_Handler"                                    8
    "main"                                             8
    "hal_entry"                                       32
    "R_BSP_SoftwareDelay"                             16
    "bsp_cpu_clock_get"                                8

Uncalled function
  "NMI_Handler": 0x00002213

  Maximum call chain                              *?* 16  bytes
    (** call graph contains indirect calls (example: "bsp_group_irq_call") **)

    "NMI_Handler"                                      8
    "bsp_group_irq_call"                               8

Uncalled function
  "R_CGC_BusClockOutCfg": 0x00001129

  Maximum call chain                                32  bytes

    "R_CGC_BusClockOutCfg"                             8
    "HW_CGC_BusClockOutCfg"                           16
    "HW_CGC_HardwareLock"                              8
    "R_BSP_RegisterProtectEnable"                      0
```

図3　　コールグラフルートカテゴリの例

3.　間接呼び出しの指定（Specify indirect calls）

間接呼び出し（indirect call）とは、関数ポインタ（function pointer）を介して関数を呼び出すことを意味します。**呼び出し先**関数（**callee** function）はビルド時点で不明なため、リンカは間接呼び出しに関するスタック使用量情報を自動的に取得できません。間接呼び出しが存在する場合、リンカは以下のような警告メッセージを出力します。

Warning[Ls016]: [stack usage analysis] the program contains at least one indirect call. Example: from "SystemInit". A complete list of such functions is in the map file.

[上記Warning日本語訳]警告[Ls016]：[スタック使用量解析] プログラムに少なくとも1つの間接呼び出しが含まれています。例：「SystemInit」以降。そのような関数の完全なリストはマップファイルにあります。

リンカマップファイルの*Stack Usage*部分の末尾には、以下の記述があります。

The following functions perform unknown indirect calls:

[上記Warning日本語訳]以下の関数は、不明な間接呼び出しを実行します。

```
  "R_CGC_ClocksCfg": 0x000005bb

  "R_CGC_Init": 0x00000529

  "R_ELC_Init": 0x00003927

  "R_IOPORT_Init": 0x000026d9

  "SystemInit": 0x00003b55

  "bsp_clock_init": 0x00001fd1

  "bsp_cpu_clock_get": 0x0000208d

  "bsp_group_irq_call": 0x000021c9
```

この問題を解決するには、開発者が*#pragma calls* ディレクティブ（directive）を使用して、ステートメントで間接的に呼び出される可能性のある関数をリスト化する必要があります。このディレクティブは、間接呼び出しステートメントの直前に挿入し、呼び出し先になり得るすべての関数のリストを指定する必要があります。たとえば、以下のコードは、関数UartRxHandler()、UartTxHandler()、およびUartFaultHandler()が関数ポインタisr()を介して間接的に呼び出される可能性のあることを示しています。

```
void BSP_IntHandler (int int_id) {
void (*isr)(void);
……
    if (int_id < BSP_INT_SRC_NBR) {
        isr = BSP_IntVectTbl[int_id];
#pragma calls=UartRxHandler,UartTxHandler,UartFaultHandler
        isr();
}
……
}
```

4. コールグラフルート情報の指定（Provide call graph root information）

RTOS（リアルタイムOS）を使用するマルチタスク環境（multi-task environment）では、各タスクのルート関数（root function）もコールグラフルートです。リンカ（linker）では、それらを自動的に識別できない場合もあります。それらは他の関数から呼び出されていないように見えるため、代わりにリンカは以下の警告メッセージを生成します。

```
Warning[Lo008]: [stack usage analysis] at least one function appears
to be uncalled. Example: " blinky_thread_func" in blinky_thread.c [1].
A complete list of uncalled functions is in the map file.
```

[上記Warning日本語訳]
警告[Lo008]：[スタック使用量解析] 少なくとも1つの関数が呼び出されないと思われます。例：blinky_thread.c [1]の「blinky_thread_func」。
呼び出されない関数の完全なリストはマップファイルにあります。

リンカマップファイルの*Stack Usage*部分には、以下の記述があります。

```
Uncalled function
```

```
  blinky_thread.o [1]の「blinky_thread_func」 : 0x00004641
……
Uncalled function

  「NMI_Handler」 : 0x00002243
```

この問題を解決するには、#pragma call_graph_rootディレクティブ（directive）を使用して関数をコールグラフルートとして識別します。以下に例を示します。

```
#pragma call_graph_root="task"                          // task category
static void blinky_thread_func (ULONG thread_input) {
{ …… }

#pragma call_graph_root="interrupt"          // task category
void NMI_Handler (void)
{ …… }
```

taskまたは**interrupt**以外の文字列を使用してコールグラフルートカテゴリの名前にすることが可能です。コンパイラにより、**interrupt**および**task**の関数にコールグラフルートカテゴリが自動的に割り当てられます。

5. スタック使用量制御ファイルの使用（Use a stack usage control file）

#pragmaディレクティブ（directive）は、ソースファイル（source file）に挿入される必要がありますが、ソースファイルによっては、**#pragmaディレクティブ**の挿入を許可されないものがあります。ソースコードを変更しない場合は、代わりに別のスタック使用量制御ファイル（stack usage control file）により、同じスタック使用量情報（stack usage information）をコンパイラとリンカに提供します。

スタック使用量制御ファイルは、*.sucという接尾語（suffix）を持つテキストファイルです。スタック使用量制御ファイルのパスは、**「Linker」**オプションの**「Advanced」**タブで設定することが可能です。

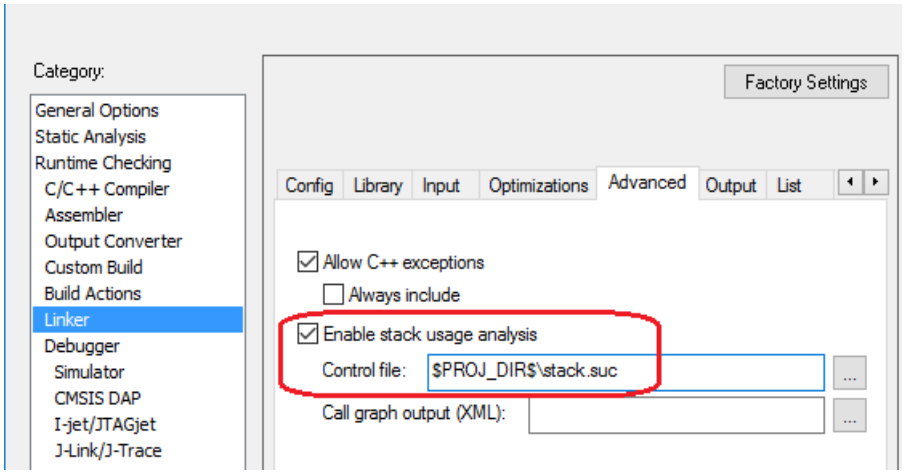

図4　　スタック使用量制御ファイル

スタック使用量制御ファイルで使用できるディレクティブには、いくつかのタイプがあります。たとえば、**function**、**exclude**、**possible calls**、**call graph root**、**max recursion depth**、**no calls from**などです。**possible calls**ディレクティブには**#pragma calls**と同様の効果があります。これは、間接呼び出し（indirect function）の可能性のある**呼び出し先**関数（**callee** function）を指定します。**call graph root**ディレクティブには**#pragma call_graph_root**と同様の効果があります。これは、呼び出されない関数のグループを**コールグラフルート**として識別します。

前述の例で使用された**#pragmaディレクティブ（directive）**をスタック使用量制御ファイル（stack usage control file）の内容に置き換えたものを、以下に示します。

```
call graph root [task] : blinky_thread_func [blinky_thread.o];
call graph root [interrupt] : NMI_Handler;
```

6. 再帰関数の反復の指定（Specify the iteration of recursive functions）

再帰関数は自らを直接または間接的に呼び出します。再帰関数の呼び出しが行われるごとに自らのデータをスタックに格納します。したがって、数回の反復後に復帰する（データを返す）ように適切に設計されていない場合、スタックオーバーフロー（stack overflow）が発生するリスクが高くなります。

実際の反復回数はビルド時点で不明なため、リンカは再帰関数に関するスタック使用量情報を自動的に取得できません。リンカは以下のような警告メッセージを出力します。

```
Warning[Lo010]: [stack usage analysis] the program contains at least
one instance of recursion for which stack usage analysis has not been
able to calculate a maximum stack depth. One function involved is
″_GLCD_SendCmd. A complete list of all recursion nests is in the map file.
```

[上記Warning日本語訳]
警告[Lo010]:[スタック使用量解析] プログラムには、スタック使用量解析で最大スタック深度を計算できなかった再帰のインスタンスが少なくとも1つ含まれています。関連する関数は「_GLCD_SendCmd」です。すべての再帰ネストの完全なリストはマップファイルにあります。

リンカマップファイルの*Stack Usage*部分には、以下の記述があります。

```
The following functions make up recursion nest 0, which has no
maximum recursion depth specified:
```

[上記Warning日本語訳]
以下の関数は再帰ネスト0を構成します。つまり、最大再帰深度が指定されていません。
「_GLCD_SendCmd」：0xffff8aac

この問題を解決するには、**スタック使用量制御ファイル**で**max recursion depth**ディレクティブを使用して、再帰関数ごとに最大再帰深度を指定します。**スタック使用量解析**の結果は、最大反復回数を再帰ネストで最も深いサイクルのスタック使用量で乗じて求められます。以下の例では、関数GLCD_SendCmd()の最大再帰深度が3に設定されています。

```
max recursion depth _GLCD_SendCmd : 3;
```

7. Synergyコンフィグレータにおけるスタックサイズの再定義（Redefining the stack size in the synergy configurator）

ファインアジャストメント（fine adjustment）後の静的スタック使用量解析（static stack usage analysis）から得られた推定されるスタックは、SSP設定の中で使用できます。スタックサイズは、Synergyコンフィグレータ（configurator）によりBSPの設定で定義されます。スタックサイズの設定を変更した場合は、必ずプロジェクトを再生成してください。

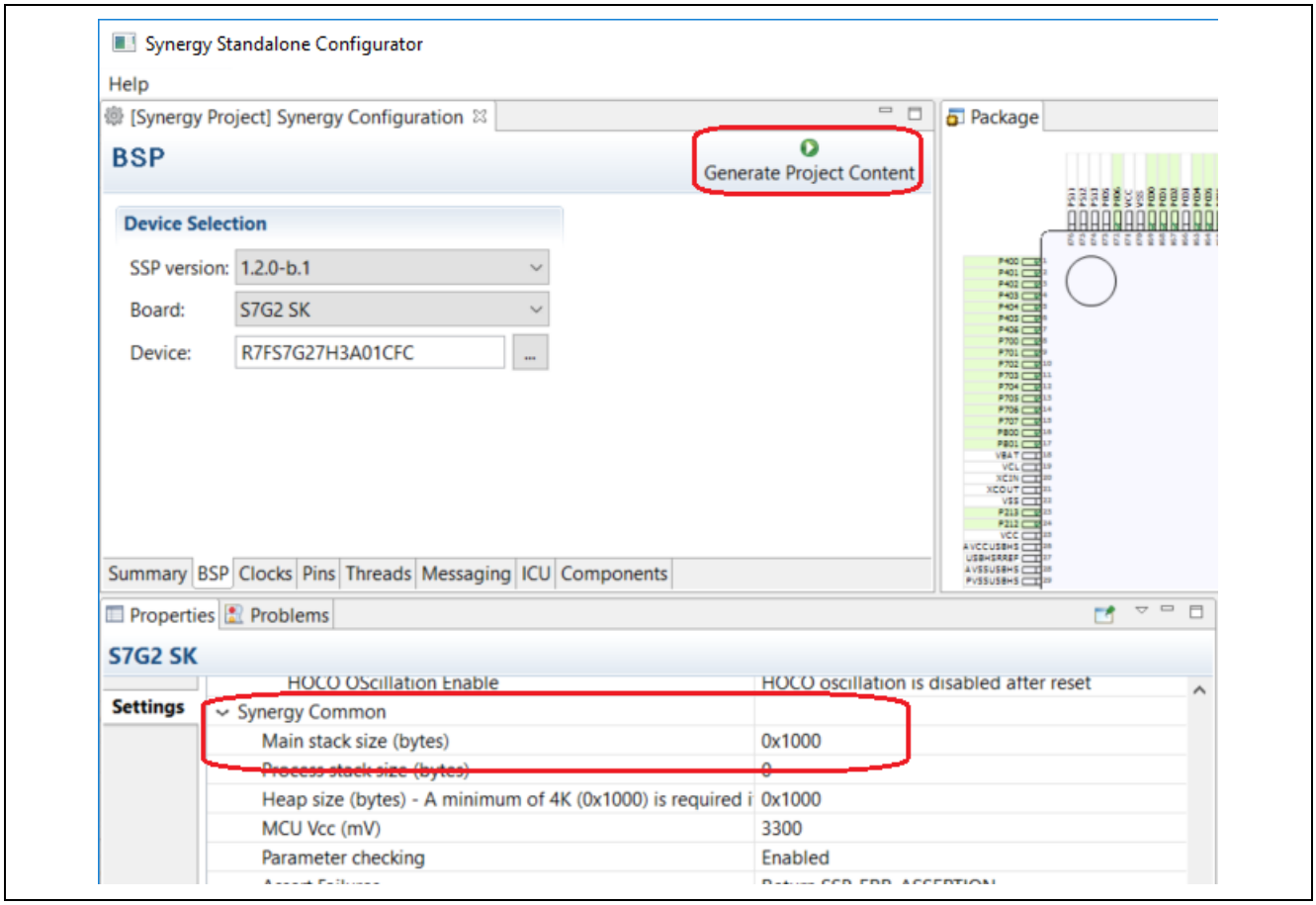

図5　　スタックサイズの再定義

8.　ランタイムスタック使用量の監視（Runtime stack usage monitering）

静的スタック使用量解析では、ビルド時点で理論的な最大スタック必要量が計算されます。実際のスタック消費量は実行中に変動する可能性があります。IAR EW for Synergyは、ランタイムでのスタック使用量を追跡するための別の方法を提供します。これはC-SPYデバッガに実装されています。C-SPYはスタック領域全体をマジックデータパターン（magic data pattern）で埋めることが可能です。たとえば、アプリケーションの実行前に0xCDの値で埋められたスタック領域に対し、アプリケーションの実行後にスタック領域の末端からアドレスの上方に向かって0xCDとは異なる値をサーチすることが可能です。0xCDとは異なる値が出現したアドレスがSPの到達した最も高いアドレスと推定されます。0xCDを格納しているスタックメモリの部分は上書きされないため、その分だけスタックサイズを縮小することが可能です。アプリケーションのテストが十分でない場合、もしくは全てのランタイムの状況を反映しきれていない場合は、スタック領域にゆとりを確保しておくのが賢明です。

IAR EW for SynergyのSSPパッケージによるデバッグ時にグラフィックスタック解析（graphical stack analysis）を有効にするには、「Project」->「Options」->「Debugger」->「Extra Options」->「Use command line options」で、デバッガに対して追加オプション--proc_stack_main=g_main_stack,g_main_stack+sizeof(g_main_stack)を有効にする必要があります。

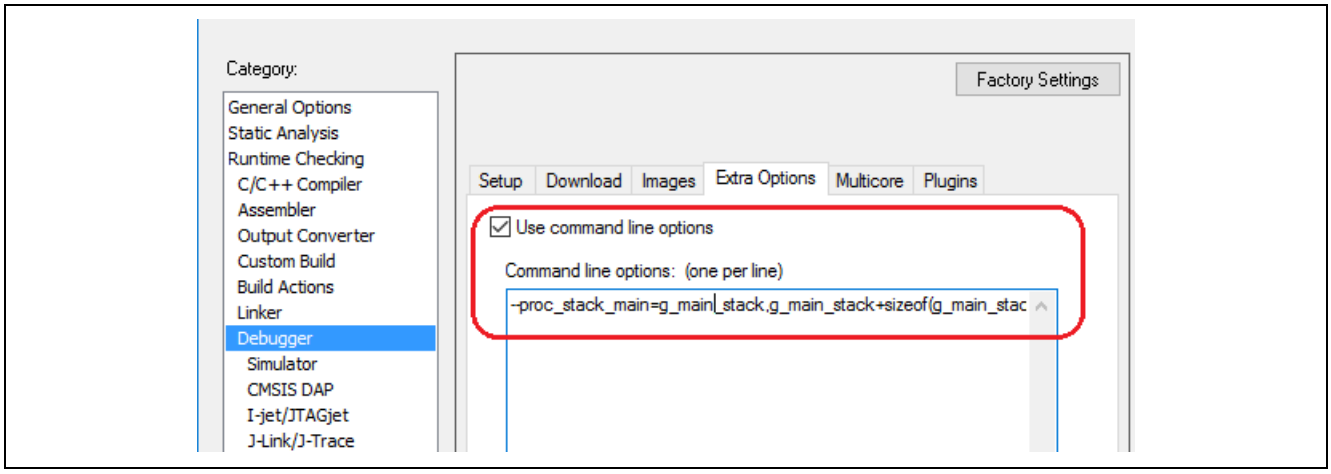

図6　　ランタイムスタック使用量

以下のように、「IDE Options」ダイアログの「Stack」カテゴリの「Enable graphical stack display and stack usage tracking」によって、ランタイムスタック使用量の追跡（runtime stack usage tracking）を有効にします。

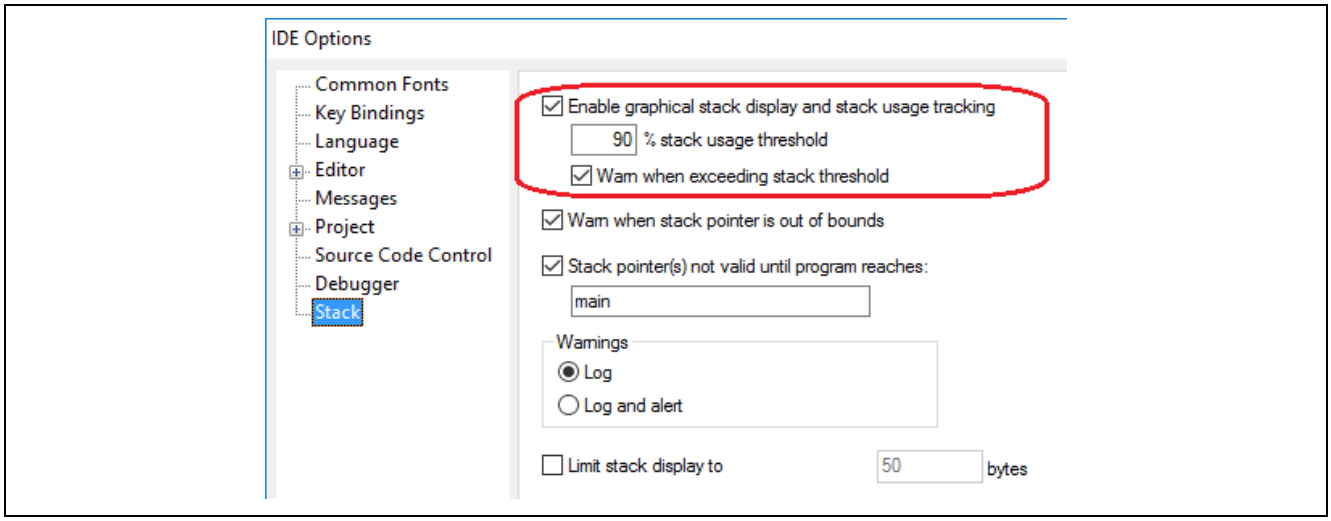

図7　　ランタイムスタック使用量の追跡

「Stack」ウィンドウは「View」メニューから利用できます。実行が停止するたびに、C-SPYは以下に示すウィンドウのスタック使用量のグラフィック表示を更新することが可能です。

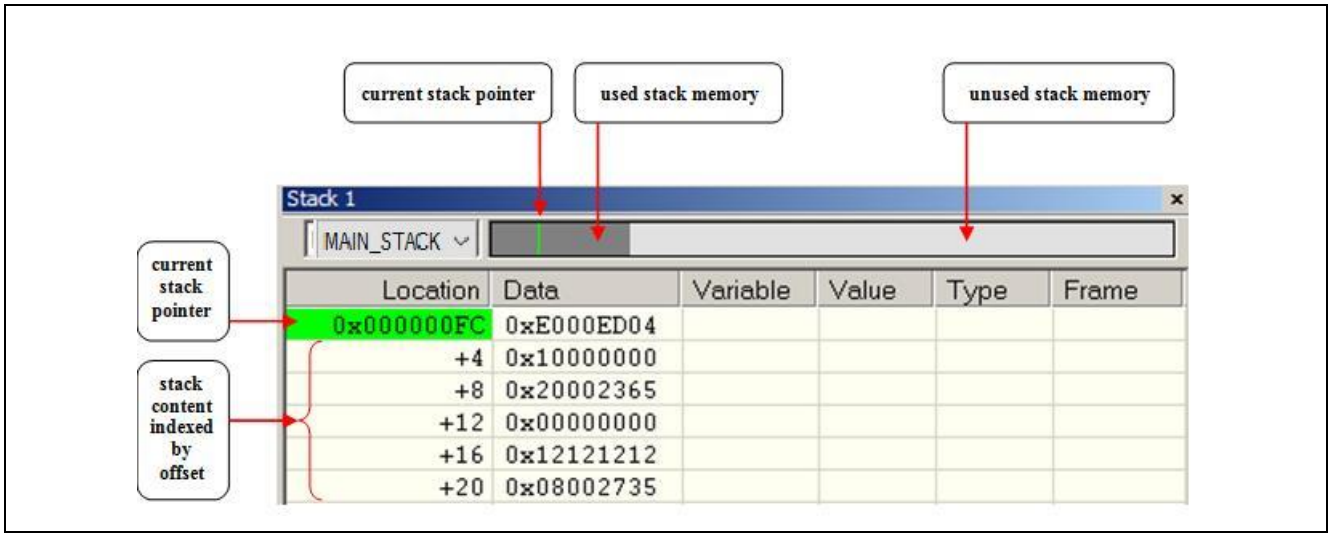

図8　　スタック使用量のグラフィック表示

グラフィックスタックバー（graphical stack bar）の左端はスタックの底（スタックが空の場合のSPの位置）を表します。右端はスタック用に確保されたメモリ空間の最後を表します。濃い灰色の領域は使用済みのスタックメモリを表し、薄い灰色の領域は未使用のスタックメモリを表します。スタック使用量がしきい値を超えると、グラフィックスタックバーは赤色になります。このしきい値は「IDE Options」ダイアログで設定することが可能です。

注：　この機能はスタックオーバーフローが発生した時点では検出できず、その痕跡しか検出できません。これはスタック使用量を確実に追跡する方法ですが、スタックオーバーフローを検出できる保証はありません。たとえば、スタックはその領域外まで誤って増加し、境界付近のバイトを変更することなく、スタック領域外のメモリを変更する可能性もあります。スタック使用量の監視には、データブレークポイント（data breakpoint）の使用を推奨します。データブレークポイントはスタックの最終バイトへのリード／ライトアクセスを監視し、アプリケーションを停止して詳細に解析することが可能です。

参考情報

*SSPユーザーズマニュアル：*SSPディストリビューションパッケージのhtml形式で利用可能であり、SynergyWEBページからpdfとしても利用可能です。

最新の参考資料とその場所を確認するには、Synergyナレッジベースにアクセスしてモジュール名を検索し、**module guide references**を検索に含めます。

たとえば、r_docモジュールの参考情報を探している場合、https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Baseにアクセスし、検索バーに**r_doc module guide references**と入力します。検索により結果のリストが表示され、先頭はモジュールガイドのリファレンスページになります。以下のURLでは、この例の検索結果が直接表示されます。

https://en-us.knowledgebase.renesas.com/Special:Search?fpid=230&search=r_doc%20module%20guide&path=&limit=55&page=1&q=r_doc%20module%20guide%20references&tags

ホームページとサポート窓口

サポート： https://synergygallery.renesas.com/support

テクニカルサポート：

- アメリカ： https://renesas.zendesk.com/anonymous_requests/new
- ヨーロッパ： https://www.renesas.com/en-eu/support/contact.html
- 日本： https://www.renesas.com/ja-jp/support/contact.html

改訂履歴

		改訂内容	
Rev.	発行日	ページ	ポイント
1.00	2018.06.15	-	初版 英文版（資料番号 r11an0195eu0100-synergy-mastering-stack-usage、リビジョンRev1.00、発効日2017年7月10日）を翻訳

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/