システムマクロトレース
ポーティングガイド
RTOS　SoftwareModel 編

株式会社ＤＴＳインサイト

システムマクロトレースポーティングガイド

発行：株式会社ＤＴＳインサイト

改訂履歴

第 1 版	2016 年 6 月	新規発行
第 2 版	2016 年 8 月	誤記修正

目次

1. はじめに

1.1. システムマクロトレースとは

システムマクロトレース（以下 SMT）は、一般的な Print 文によるデバッグ手法と関数／タスクの実行履歴機能を融合したデバッグ手法です。システムマクロトレースを使用することで、以下のような解析ができます。

- 関数／タスクの実行履歴を可視化
- 関数の処理時間が見える
- システム全体の動作からボトルネックを発見できる
- 関数履歴と print 文を同時に記録/表示可能

1.2. SMT API ライブラリとは

SMT API ライブラリとは、システムマクロトレース対象のアプリケーションからシステムマクロトレース出力 I/F 専用デバイスドライバを利用し、関数トレース情報や、デバッグプリント文等の出力を行うためのライブラリです。
システムマクロトレースをご使用いただく際には、必ずシステムマクロトレース API ライブラリ（以下、API ライブラリ）をユーザープログラムに組み込んでいただく必要があります。

1.3. ポーティングガイドとは

ポーティングガイドは、RTOS システムに SoftwareModel 用の API ライブラリを組み込む方法について、実際の手順を追って解説するものです。

[ポーティング内容]

- API ライブラリのインストール
- API ライブラリについて
- API ライブラリのビルド(システムマクロトレースドライバの実装)
- システムマクロトレースの取得方法
- コンテキスト切り替え情報の取得
- Cpe/ make_cp での固有設定
- macroTRACE-VIEWER での固有設定

1.4. API ライブラリ出力仕様/制限事項

1.4.1. 出力情報について

システムマクロトレースの出力を行うためには、「4.API ライブラリのインストール」から「9.テスト出力 API 関数の挿入」までの作業が必須になります。

「4.API ライブラリのインストール」から「9.テスト出力 API 関数の挿入」を実施するとシステムマクロトレースで下記情報出力の確認が出来ます。

・ テストアプリケーションからのシステムマクロトレース

1.4.2. タスク遷移情報について

タスク遷移切り替え情報を取得するためには、タスク切り替え情報が取得可能なリアルタイム OS が必要です。

1.4.3. MPU の動作モードについて

SMT データ出力を実施する場合に、MPU の動作モードを変更する必要がある場合は、
割り込み禁止・解除関数（_SMT_InterruptDisable/_SMT_InterruptRestore）を使用して動作モードの変更を行う必要があります。

1.5. 各ツールのマニュアルについて

本ドキュメント内で使用する各ツールの詳細についてはそれぞれ下記マニュアルをご覧ください。
・macroTRACE-VIEWER は『macroTRACE-VIEWER ユーザーズマニュアル』

・Cpe は『システムマクロトレースチェックポイントエディタ(Cpe)ユーザーズマニュアル』

・make_cp は『システムマクロトレースチェックポイント挿入コマンドラインツール(make_cp)ユーザーズマニュアル』

・API ライブラリ関数の詳細は『システムマクロトレース API ライブラリ リファレンスマニュアル』

1.6. 表記規則

本書では以下の表記規則を使用しています。

monospace
　　コマンド、ファイル名、フォルダ、関数名、ソースコードを示しています。

monospace italic
　　コマンド、ファイル名、フォルダ、関数名、ソースコードで、説明の都合上、指定しており、実際には、お客様の開発環境に依存するものを示しています。

gothic
　　GUI 上で指定するメニュー名や、項目名を示しています。

gothic italic
　　GUI 上で指定するメニュー名や、項目名で、説明の都合上、指定しており、実際には、お客様の開発環境に依存するものを示しています。

1.7. フォルダ表記について

API ライブラリインストールフォルダを $ (API_LIBRARY)と表記します。
システムマクロトレースを出力するインターフェス名を$(SMTIF)と表記します。

1.8. バージョン表記について

API ライブラリは APILib-RTOS-REV.tar.gz 形式で提供します。
REV は、API ライブラリバージョンを示し、バージョン 1.00 の場合は
APILib-RTOS-100.zip となります。

1.9. お問い合わせについて

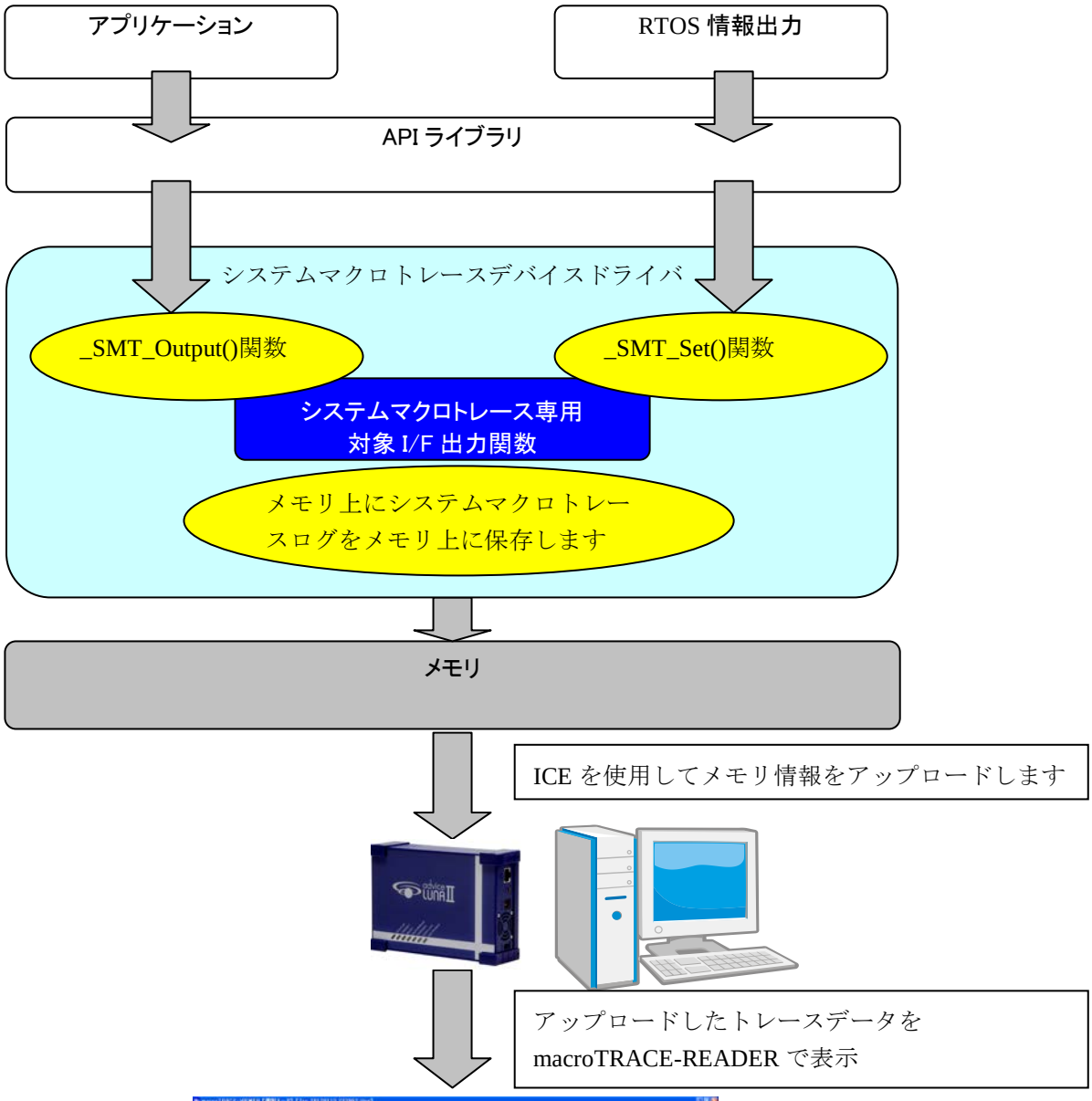
マニュアルや製品に関する質問に関しましては弊社 WEB サイトに FAQ がありますので、
はじめに弊社 WEB サイトの FAQ をご覧ください。
http://www.dts-insight.co.jp/support/support_advice/luna/faq/

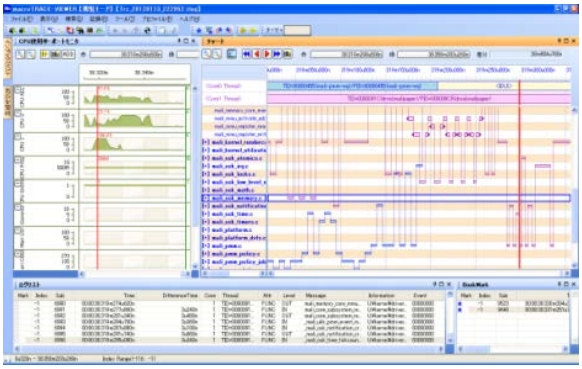
弊社 WEB サイトの FAQ を探しても解決しない場合、下記お問い合わせフォームよりお問い合わせください。
https://www.dts-insight.co.jp/support/support_advice/

2. ご利用の前に

2.1. API ライブラリの機構

デバイスコントローラはユーザーシステムごとに異なり、実装も変わるため、専用のシステムマクロトレース出力 I/F デバイスドライバを環境にあわせてポーティングしていただく必要があります。

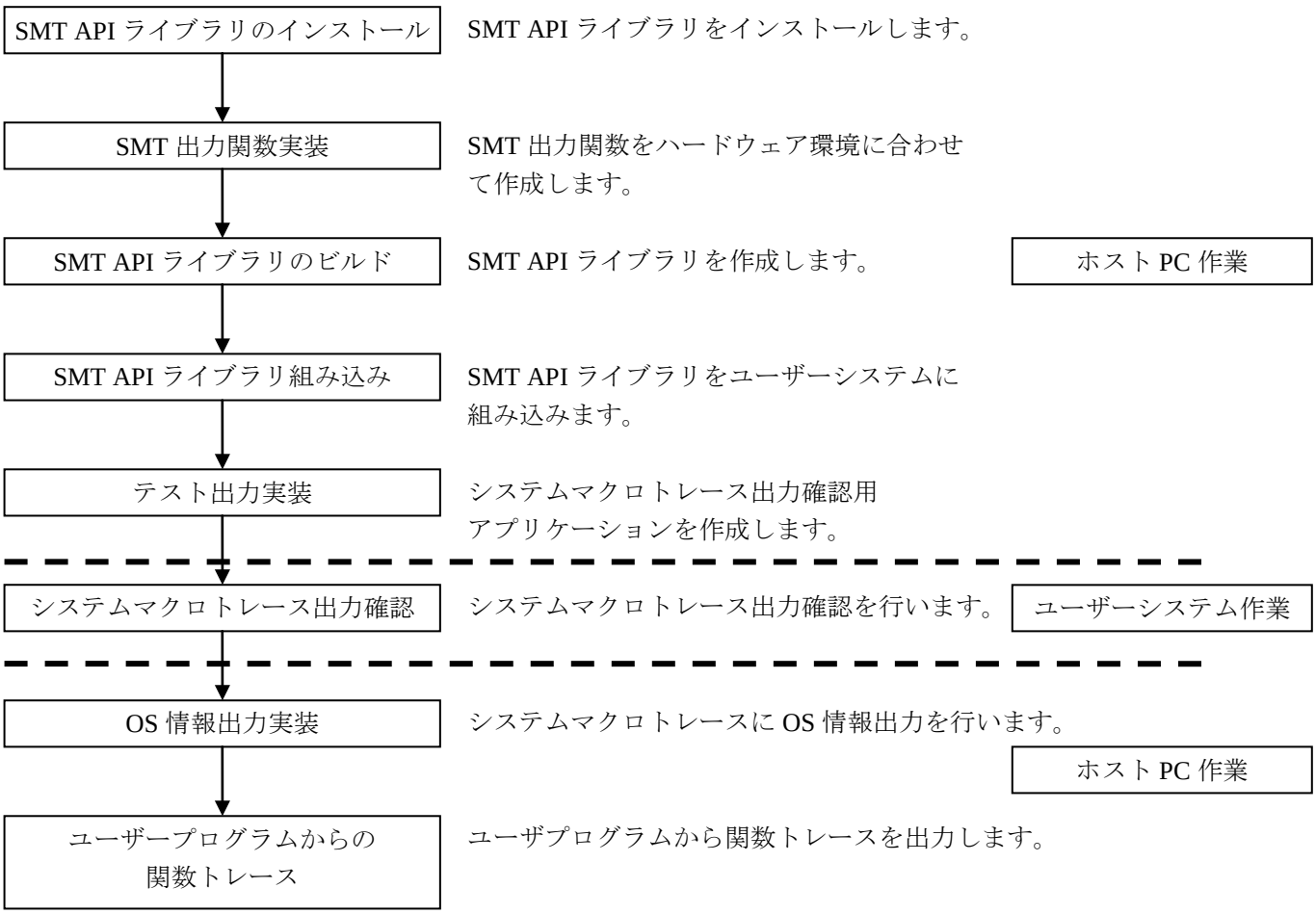




3. 作業の流れについて

システムマクロトレース出力を行うための作業フローは下記のようになっています。

- SMT データ出力関数の実装
- SMT API ライブラリをユーザーシステムに組み込む
- SMT API ライブラリを組み込んだサンプルプログラムを実行する

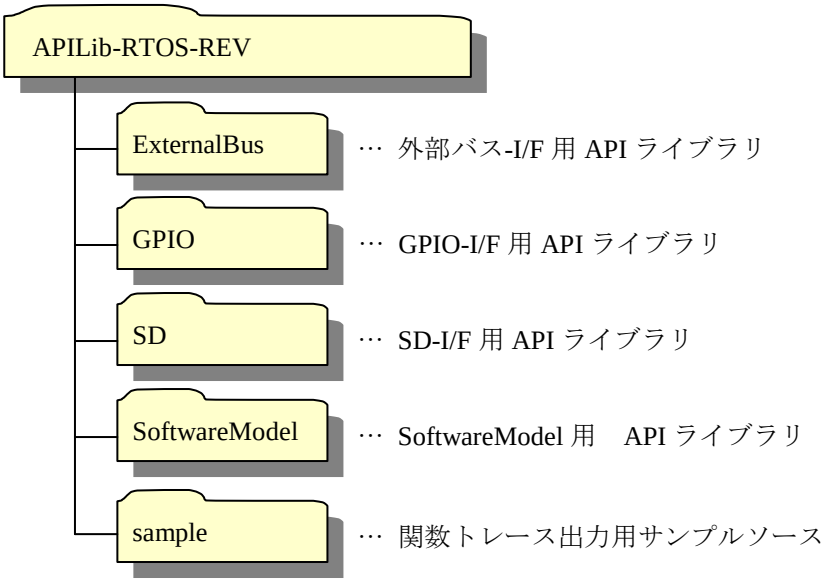


4. API ライブラリのインストール

4.1. インストール

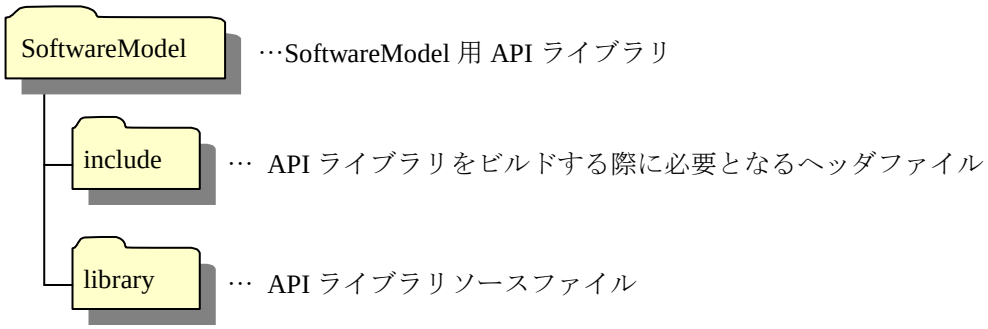
アーカイブファイルを解凍ソフトを使用して展開してください。

C:/YDC/SMT に展開した場合下記のようになります。



4.2. フォルダ構成

SoftwareModel のフォルダ構成は以下のようになっています。



4.3. API ライブラリの注意事項

4.3.1. C コンパイラ標準ライブラリのリンク

API ライブラリ関数で提供する_SMT_Printf 関数は、標準ライブラリ関数の `vsprintf` を使用するように設計されています。
このため、_SMT_Printf 関数を使用する場合は、標準ライブラリのリンクが必要となります。
もしくは、標準ライブラリを使用せずに `vsprintf` 関数が使用できるように、`vsprintf` 関数を再実装する必要があります。

詳しくはご使用になる開発環境のユーザーズマニュアルをご覧ください。

5. システムマクロトレースデータ出力関数実装

SoftwareModel システムマクロトレースを出力するために必要な出力関数と割り込み制御関数の実装について説明します。
ソースファイルは以下のフォルダにインストールされています。
$(API_LIBRARY)/SoftwareModel/library/smtoutput.c 内にひな型が定義してあります。

5.1. トレースデータ出力先設定について

トレースデータは BASE_ADDR に指定したアドレスからトレースデータサイズ指定サイズ保存します。

5.2. トレースデータサイズ指定について

$(API_LIBRARY)/SoftwareModel/library/SMTOutput.c の TRACE_SIZE の値を変更することでトレースデータのサイズを変更することができます。（初期値 100）
確保するメモリサイズは初期値 100 の場合 unsigned long(4byte)*100 で 400byte です。
ご利用の環境に合わせて変更してください。

5.3. システムマクロトレース出力関数の実装

_SMT_GetTimeStamp 関数、_SMT_GetCoreID 関数、_SMT_InterruptDisable 関数、_SMT_InterruptRestore 関数をユーザーシステムに合わせて実装してください。

5.3.1. _SMT_GetTimeStamp 関数

■機能

システムのタイムスタンプを取得し unix 時刻の秒単位を sec 変数に、ナノ秒単位の情報を nano_sec 変数に格納してください。
※ユーザーシステムに合わせて実装してください。

■呼び出し手順

ユーザーから呼び出しません。API ライブラリ内部からコールされます。

■引数

No.	名前	型	意味
1	sec	unsigned long	システムの秒情報を格納してください。
2	nano_sec	unsigned long	システムのナノ秒情報を格納してください。

■戻り値

なし

5.3.2. _SMT_GetCoreID 関数

■機能

コア ID を返します。
シングルコア環境の場合は core_id には常に 0 を設定してください。
マルチコア環境の場合はコア ID を取得処理を実装して core_id 変数にコア ID を代入してください。

※ユーザーシステムに合わせて実装してください。

■呼び出し手順

ユーザーから呼び出しません。API ライブラリ内部からコールされます。

■引数

■戻り値

No.	値	意味
1	core_id	コア ID

5.1. 割り込みマスク制御関数の実装

トレース情報出力中に、割り込み処理関数内で新たなトレース情報出力が発生した場合、出力中のトレース情報が一部欠損する可能性があります。

必ず、トレース情報の欠損を抑止する為に、割り込み禁止・解除関数を実装してください。
また、トレース情報出力ポートへの書き込みに、MPU の動作モード変更が必要な場合、MPU の動作モード変更も本関数内で実施します。

マルチコア環境の場合にはデータ出力中のコア以外からのデータ出力を行わないようにスピンロックの獲得、開放処理も本関数内で実施してください。

ソースファイル「SMTOutput.c」

関数名	処理内容
_SMT_InterruptDisable	スピンロックの獲得、割り込み禁止
_SMT_InterruptRestore	スピンロックの開放、割り込み禁止解除

5.1.1. _SMT_InterruptDisable 関数

■機能
現時点での割り込み禁止状態を退避(保持)し、
割り込み禁止処理（割り込みマスク）を実施します。
ステータスレジスタの割り込みマスクビットを操作します。
マルチコア環境の場合にはスピンロックの獲得処理を実施します。
※ユーザーシステムに合わせて実装してください。

■引数
なし

■戻り値

変数名	説明
ALREADY_IRQ_DISABLED	割り込み禁止状態の場合にはこの変数を戻り値としてください。
SET_IRQ_DISABLE	割り込み禁止状態にした場合にはこの変数を戻り値にしてください。

5.1.2. _SMT_InterruptRestore 関数

■機能
割り込み禁止状態の復元処理（割り込みマスク解除）を実施します。
_SMT_InterruptRestore 関数で退避したステータスレジスタの割り込みマスクビットに戻します。
マルチコア環境の場合にはスピンロックの開放処理を実施します。

※ユーザーシステムに合わせて実装してください。

■引数
なし

■戻り値
なし

5.2. _SMT_get_SP_address 関数実装（API ライブラリバージョン 3.10 以降）

■機能

スタックポインタ値の取得を実施します。
※スタック解析機能を使用する場合にはユーザーシステムに合わせて実装してください。

■呼び出し手順

ユーザーから呼び出しません。API ライブラリ内部からコールされます。

■戻り値

型 ： `_SMT_UNSIGNED_32BIT_INTEGER`
戻り値には取得したスタックポインタ値を指定してください。

5.3. 使用上の注意事項

5.3.1. MPU の動作モードについて

トレースデータ出力を実施する場合に、MPU の動作モードを変更する必要がある場合は、
割り込み禁止・解除関数（_SMT_InterruptDisable /_SMT_InterruptRestore）を使用して動作モードの変更を行う必要があります。

5.3.2. トレースデータサイズ指定について

トレースデータサイズは 34 以上を設定してください。
34 未満の場合トレースを開始できません。

5.3.3. タイムスタンプについて

タイムスタンプ精度によっては同一タイムスタンプが付与される可能性があります。
また、SMT 取得中にはタイムスタンプを変更しないでください。

6. スタック解析機能を有効にする

スタック解析機能は初期状態では出力されません。
1.$(API_LIBRARY)/SoftwareModel/include/user/SMTAPI.h の_SMT_OUTPUT_SP_ADDR の定義値を_SMT_ON にしてください。
2.$(API_LIBRARY)/SoftwareModel/library/SMTOutput.c の_SMT_get_SP_address()にスタックポインタアドレス値を取得するための実装を行ってください。

6.1. 注意点

・スタック解析機能使用するためには関数トレース情報を出力する必要があります。
・スタックポインタのアドレスが 64bit の場合は使用できません。

7. API ライブラリのビルド

デバッグ情報を出力するには、API ライブラリをビルドして組み込む必要があります。
API ライブラリはユーザーシステムに合わせて実装して組み込む必要があります。
この章では SoftwareModel からシステムマクロトレースを出力するのに必要な出力関数、デバイスドライバのビルド方法について説明します。

事前に「5.システムマクロトレースデータ出力関数実装」を実施してください。

7.1. API ライブラリファイル一覧

API ライブラリのビルドに必要なファイルについて示します。
ソースファイルは以下のフォルダにインストールされています。
`$(API_LIBRARY)/SoftwareModel/`

フォルダ	ファイル名	説明
library	SMTDebugLevel.c	API ライブラリソースファイルです。
library	SMTOsCall.c	API ライブラリソースファイルです。
library	SMTOsSwitch.c	API ライブラリソースファイルです。
library	SMTPortOut.c	API ライブラリソースファイルです。
library	SMTPrintf.c	API ライブラリソースファイルです。
library	SMTUsrMsgTag.c	API ライブラリソースファイルです。
library	TRQHook.c	API ライブラリソースファイルです。
library	SMTCP.c	API ライブラリソースファイルです。
library	SMTUser.c	システムマクロトレース出力レベル定義ファイルです。
library	SMTOutput.c	SoftwareModel トレースデータ出力関数・割り込みマスク制御関数ソースファイルです。
include/library	SMTDef.h	トレースフォーマット定義ヘッダファイルです。
include/library	TRQHookc.h	チェックポイント関数のライブラリヘッダファイルです。
include/library	SMTUser.h	デバッグプリント出力レベルの設定ファイルです。
include/library	SMTAPI_CP.h	API 関数のライブラリヘッダファイルです。
include/user	SMTAPI.h	API 関数のライブラリヘッダファイルです。

7.2. ビルド設定

API ライブラリをビルドする為に、$(API_LIBRARY)/SoftwareModel/library/Makefile 先頭部分に定義された変数を、ご利用の環境に合わせて変更してください。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のプレフィックスを設定してください。
TARGET	ユーザーランド API ライブラリファイル名を設定してください。
SMT_CONFIG_SMP	ユーザーシステムがマルチコア環境の場合 yes にしてください。 シングルコア環境の場合には no にしてください。

```
# Set  prefix for a cross compiler
#
CROSS_COMPILE=arm-elf-

#
# Set the SMT API Library file name
#
TARGET = libsmt.a

# Selects yes or no for whether there is Single Core mode setting
# function of a page table entry.
SMT_CONFIG_SMP=yes
export SMT_CONFIG_SMP

#
#      You don’t need to make change hereafter.
#
```

7.3. ビルドの実行

ご利用環境に合わせてビルドを実行してください。

7.4. 実施項目の確認

ビルドが完了後、ユーザーシステムで動作させる前に次の項目をご確認ください。

7.4.1. 実施項目の確認

ビルドが完了後、ユーザーシステムで動作させる前に次の項目をご確認ください。

番号	確認項目	チェック
1	トレースデータ出力先アドレスを設定しましたか？ 「5.1 トレースデータ出力先設定について」	□
2	トレースデータ出力サイズは設定しましたか？ 「5.2 トレースデータサイズ指定について」	□
3	タイムスタンプ取得処理実装をしましたか？ 「5.3.1_SMT_GetTimeStamp 関数」	□
4	コア ID 取得関数を実装しましたか？ 「5.3.2_SMT_GetCoreID 関数」	□
5	割り込みマスク制御関数の実装は行いましたか？ （_SMT_InterruptRestore,_SMT_Interrupt Disable) 「5.1 割り込みマスク制御関数の実装」	□
6	API ライブラリのビルドは問題なく出来ましたか？ 「7.3.ビルドの実行」	□
7	API ライブラリをユーザーシステムに組み込みましたか？ 「8.API ライブラリの組み込み	□

各項目をご確認後「8.API ライブラリの組み込み」へ進んでください。

8. API ライブラリの組み込み

「7.API ライブラリのビルド」で対象 I/F ソースから生成された API ライブラリの名称を下表の名称で作成されたと仮定して説明します。

ファイル名	内容
libsmt.a	SMT API ライブラリ

8.1. API ライブラリのリンク

API ライブラリファイルをリンカ入力に追加してください。
詳しくはご使用になる開発環境のユーザーマニュアルをご覧ください。

9. テスト出力 API 関数の挿入

システムマクロトレース API 関数に、テスト出力のための関数はありません。
代わりに、API 関数の_SMT_Puts をユーザーソースに追記して、データが取得できるかを確認します。
下のように、API 関数を追加します。API ライブラリヘッダファイルのインクルードも追加します。

```
#include "SMTAPI.h"

void main()
{
    /* 省略 */

    _SMT_Start();
    _SMT_Puts(0, " System Macro Trace StartUp!");
    _SMT_Stop();

    /* 省略 */
}
```

作成したプログラムをユーザーシステムに組み込んでください。

10. システムマクロトレースの記録方法について

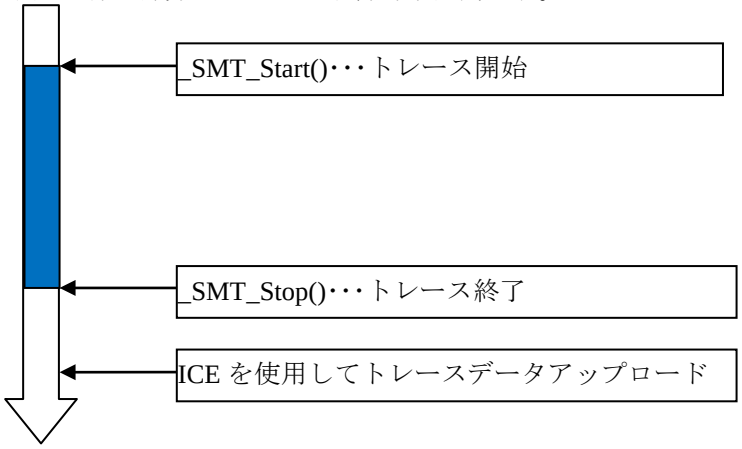
システムマクロトレースの記録方法について記載します。

10.1. システムマクロトレース記録方式について

トレース開始（_SMT_Start 関数コール）から TRACE_SIZE で設定されたサイズになるかトレース停止（_SMT_Stop 関数がコール）されるまでの間のトレースを行います。

10.2. システムマクロトレースの記録方法について

システムマクロトレース記録方法フローは下記のとおりです。
下図の青色部分がトレース取得対象区間です。



10.2.1. トレース開始方法について

トレース開始したい場所に__SMT_Start 関数を実行してください。

10.2.2. トレース終了方法について

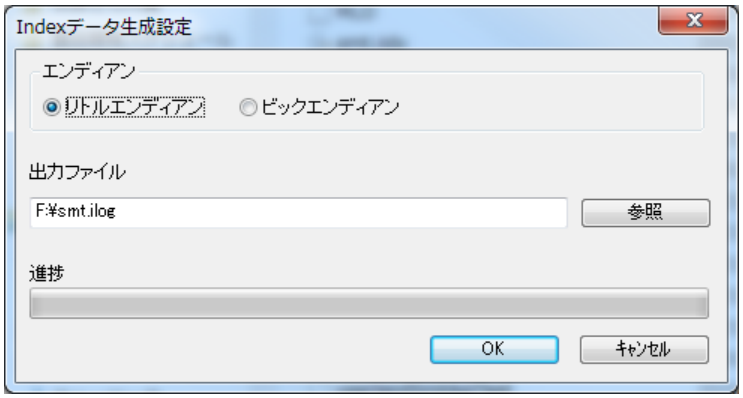
トレース終了したい場所に_SMTStop 関数を実行してください。

10.3. トレースデータのアップロードについて

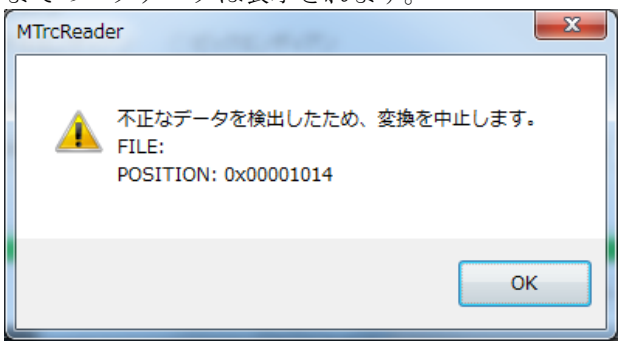
トレース終了後 BASE_ADDR アドレスから TRACE_SIZE 分のトレースデータをバイナリ形式でアップロードしてください。
アップロードしたバイナリデータのファイル拡張子は.slog にしてください。

11. システムマクロトレース出力確認

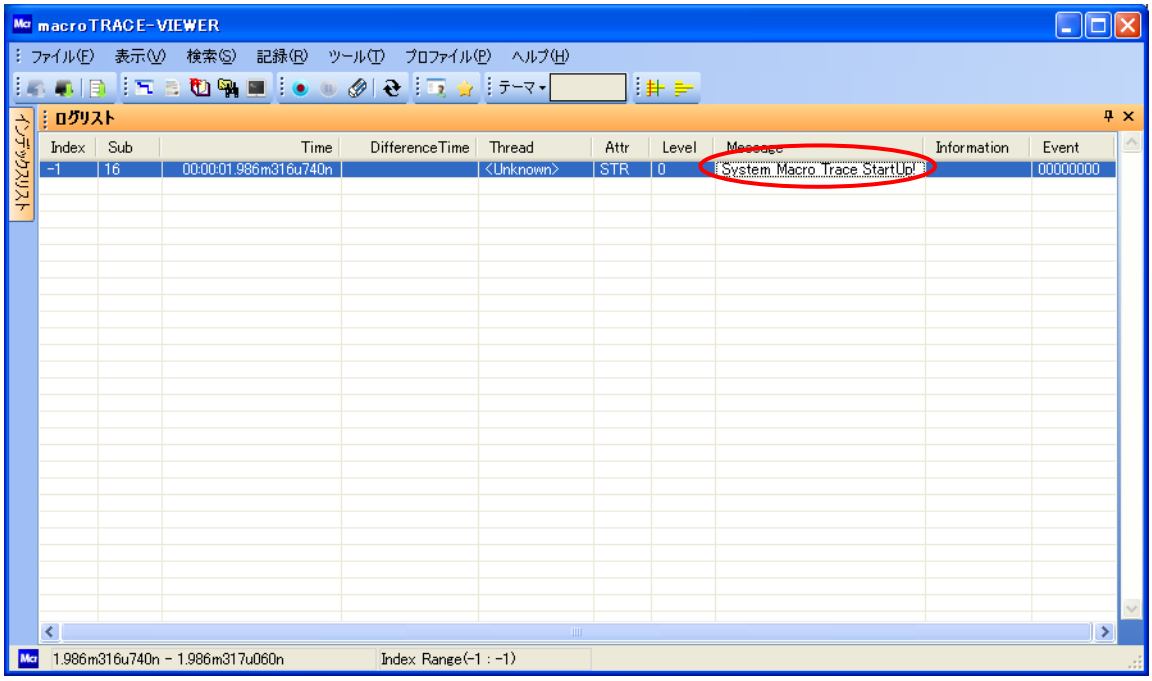
アップロードしたバイナリデータファイル(拡張子.slog)をダブルクリックしてください。
エンディアンはユーザーシステムのエンディアンを選択してください。
出力ファイルにはトレースデータを変換後の保存先とファイル名称(拡張子 ilog)を設定します。



トレースデータのサイズが小さい場合には、変換途中で以下の様な警告が出ますが、変換出来たところまでのログデータは表示されます。



macroTRACE-VIEWER でのトレース取得例を示します。

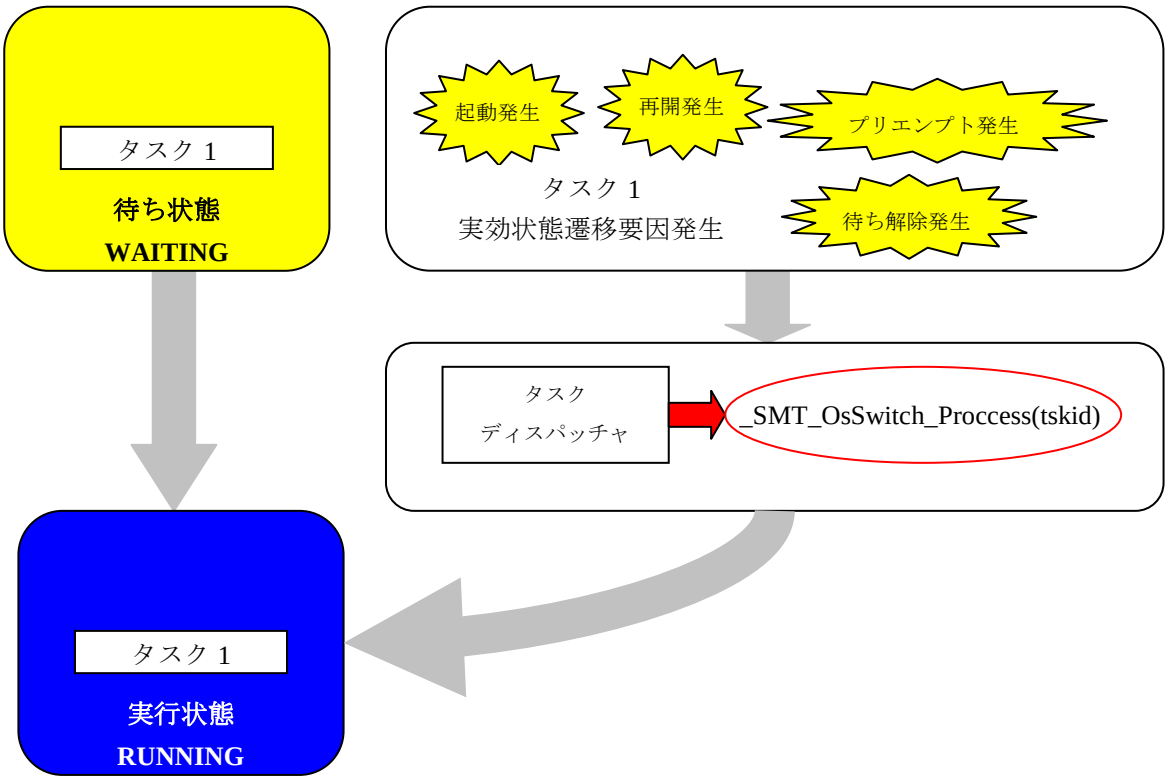


12. タスク切り替えの情報の取得方法例

API ライブラリを OS が搭載されたシステムでご使用された場合、ディスパッチ（タスク切り替え）情報を取得する事が必要です。
ディスパッチ情報取得を実施せずに複数のタスクに対して関数トレースを実施すると、ディスパッチ情報を判別できず、関数チャートが正しく表示されない場合があります。
※ディスパッチ処理は OS ごとに異なり。不明な場合は、OS ベンダーにご確認ください。

12.1. タスク切り替え発生時

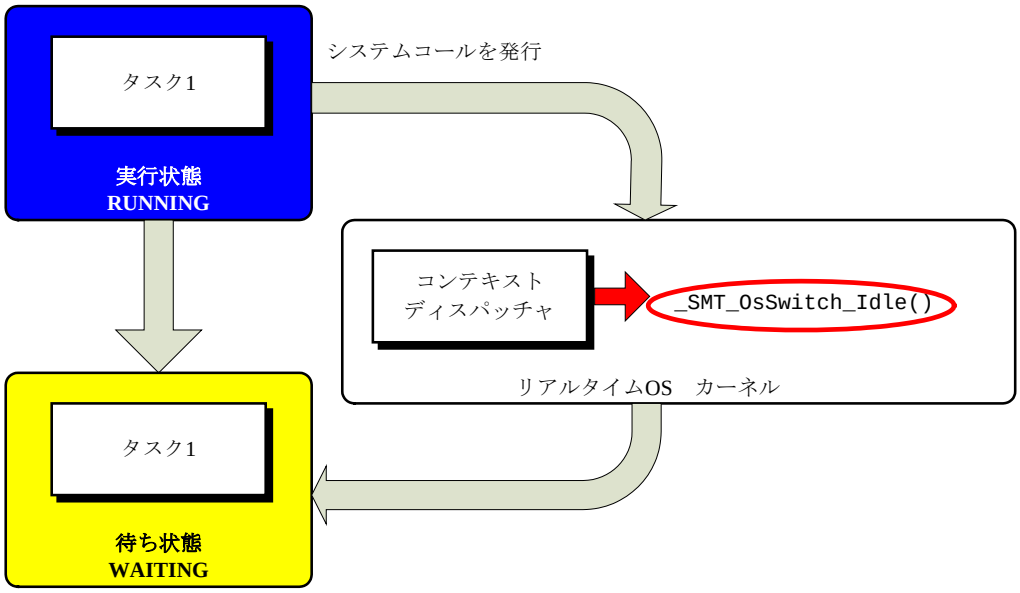
タスク切り替えが発生した場合は、タスク切り替えを実施する関数（タスクディスパッチャ関数）内で、_SMT_OsSwitch_Process 関数を実行状態に遷移する前に、タスク番号を設定して呼び出しを実施します。



例では、タスク 1 が待ち状態から実行可能状態に遷移しますので、
_SMT_OsSwitch_Process 関数にタスク 1 のタスク ID を設定してから呼び出しを実施します。

12.2. 実行タスクが無い場合

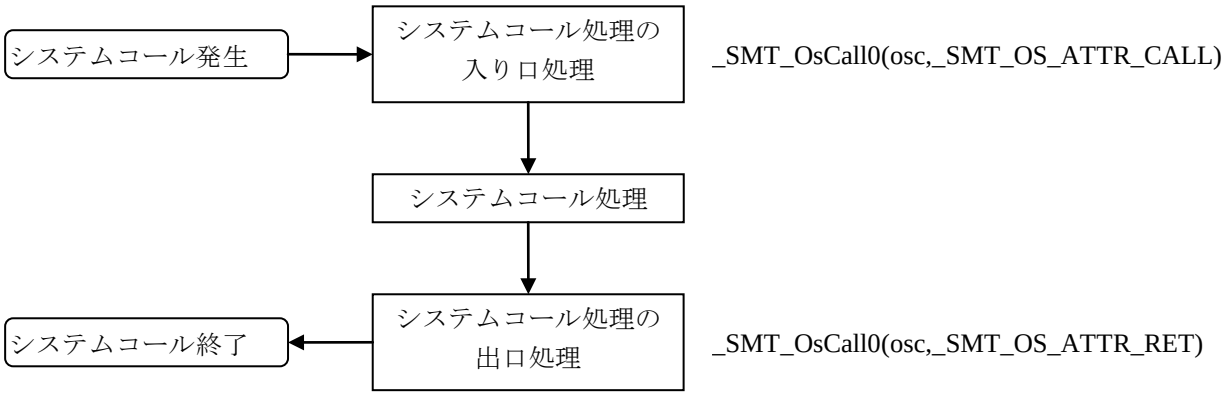
タスク切り替えを実行後に、実行状態のタスクが存在しない場合は、
_SMT_OsSwitch_Idle 関数を呼び出します。



13. システムコール情報の取得方法例

システムコールを取得する場合は、システムコール関数の開始と終了時に
_SMT_OsCall0～16 関数を呼び出します。

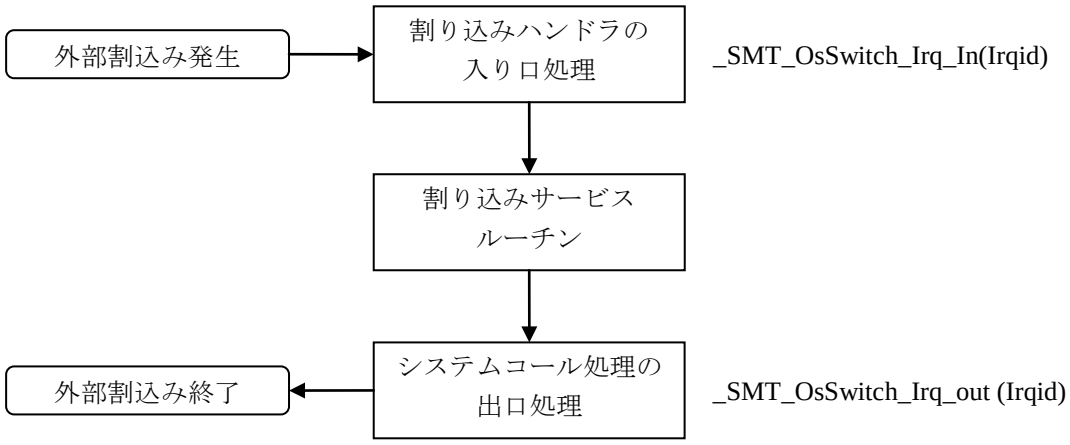
システムコール情報取得の使用イメージ



14. 割り込み情報の取得方法例

割り込み動作を取得する場合は、割り込みハンドラの開始時に_SMT_OsSwitch_Irq_in 関数を呼び出し、終了時に_SMT_OsSwitch_Irq_out 関数を呼び出します。

割り込み情報取得の使用イメージ



15. ユーザープログラムからの関数トレース

システムマクロトレースの機能を使用するには、ユーザープログラムにシステムマクロトレース API ライブラリをリンクして、API 関数を呼び出す必要があります。
この章では Cygwin 環境で専用コンパイラ版を使用してサンプルプログラムからシステムマクロトレースを行う手順について説明します。
専用コンパイラ/汎用コンパイラについての詳細は『Cpe マニュアル』を参照してください。

15.1. サンプルソースプログラムのビルド実施

15.1.1. サンプルソースコード一式を展開

サンプルソースコードは`$(APILIBRARY)/startup/src` にあります。

コピー後、ホームフォルダに project フォルダを作成し、サンプルソースコード一式を展開します。

```
[HOST]$ cd ~
[HOST]$ mkdir project
[HOST]$ cd project
[HOST]$ cp –a $(APILIBRARY)/startup .
[HOST]$ ls
startup
```

15.1.2. ソースコードの確認

`~/project/startup/src` に移動してください。次のサンプルソースファイルがあります。

ファイル名	処理概要
`hellowold.c`	メイン関数
`Makefile`	ビルド用 makefile

```
[HOST]$ cd ~/project/startup/src
[HOST]$ ls
Makefile helloworld.c
```

15.1.3. Makefile を編集

サンプルソースをビルドするために Makefile 先頭部分の以下に示す変数を、ご利用の環境に合わせて適宜エディタで編集してください。
ここでは、ユーザーランド API ライブラリファイル名は、libsmt.a で作成されたと仮定します。

変数名称	説明
CROSS_COMPILE	クロスコンパイラ使用時のプレフィックスを設定します。
SMT_INCDIR	ユーザーランド API ライブラリヘッダファイルのサーチパスを設定します。
SMT_LIBDIR	ユーザーランド API ライブラリのサーチパスを設定します。
SMT_LIB	ユーザーランド API ライブラリファイルを設定します。

編集例

```
CROSS_COMPILE = linux-gnu-
SMT_INCDIR = ~/APILib-RTOS-REV/SoftwareModel/include/user
SMT_LIBDIR = ~/APILib-RTOS-REV/SoftwareModel/library
SMT_LIB = libsmt.a
```

15.1.4. ビルド実施

編集した Makefile を使用して make をしてください。

① コマンドプロンプトにて、コマンドを入力してください。

```
[HOST]$ make
```

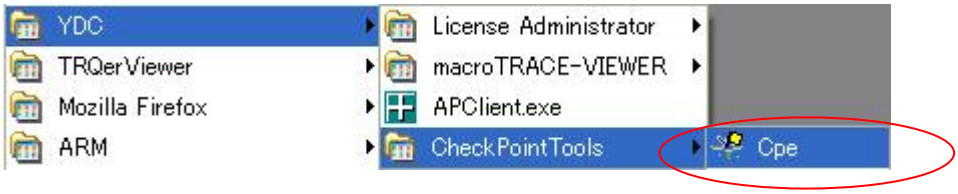
② 実行後、helloworld ファイルが生成されていることを確認してください。

```
[HOST]$ ls
Makefile    helloworld.c
 helloworld  helloworld.o
```

15.2. Cpe ツール設定

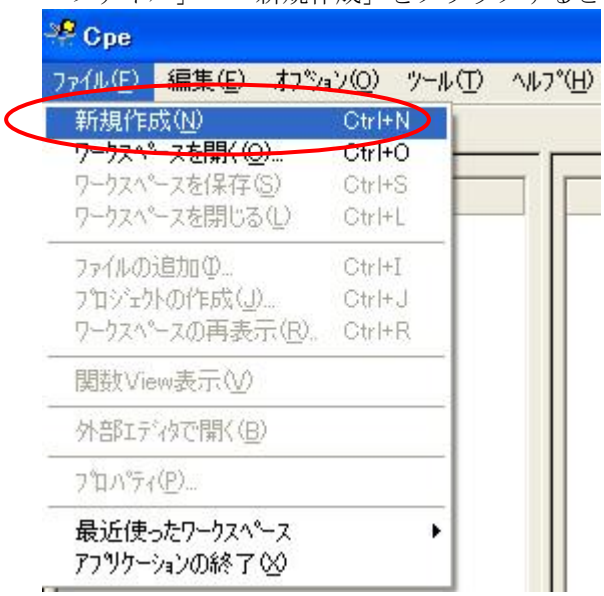
15.2.1. Cpe 起動

「スタートメニュー」→「プログラム」→「YDC」→「CheckPointTools」→「Cpe」をクリックし、Cpe を起動します。



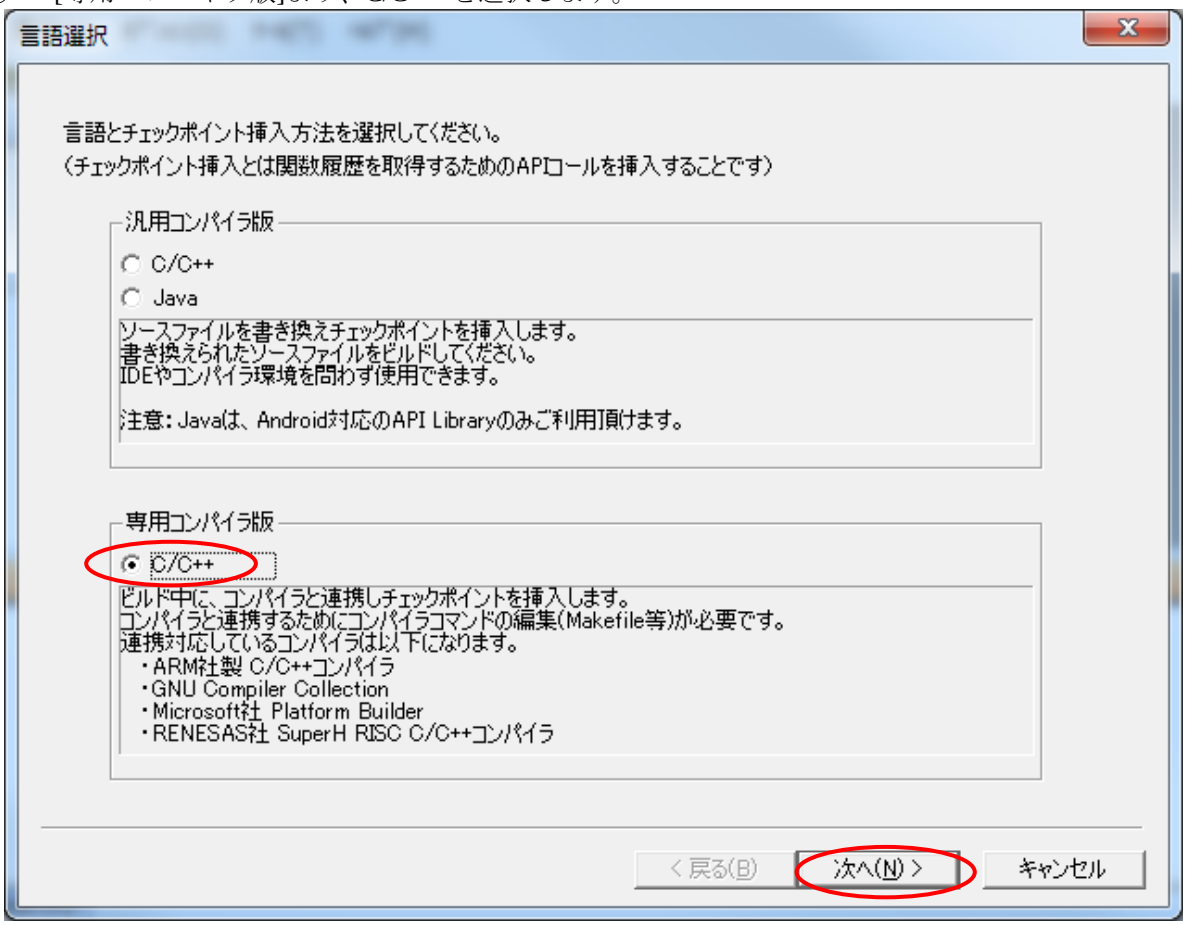
15.2.2. ワークスペース新規作成

「ファイル」→「新規作成」をクリックすると、「新規プロジェクト」ダイアログが開きます。



15.2.3. 言語の選択

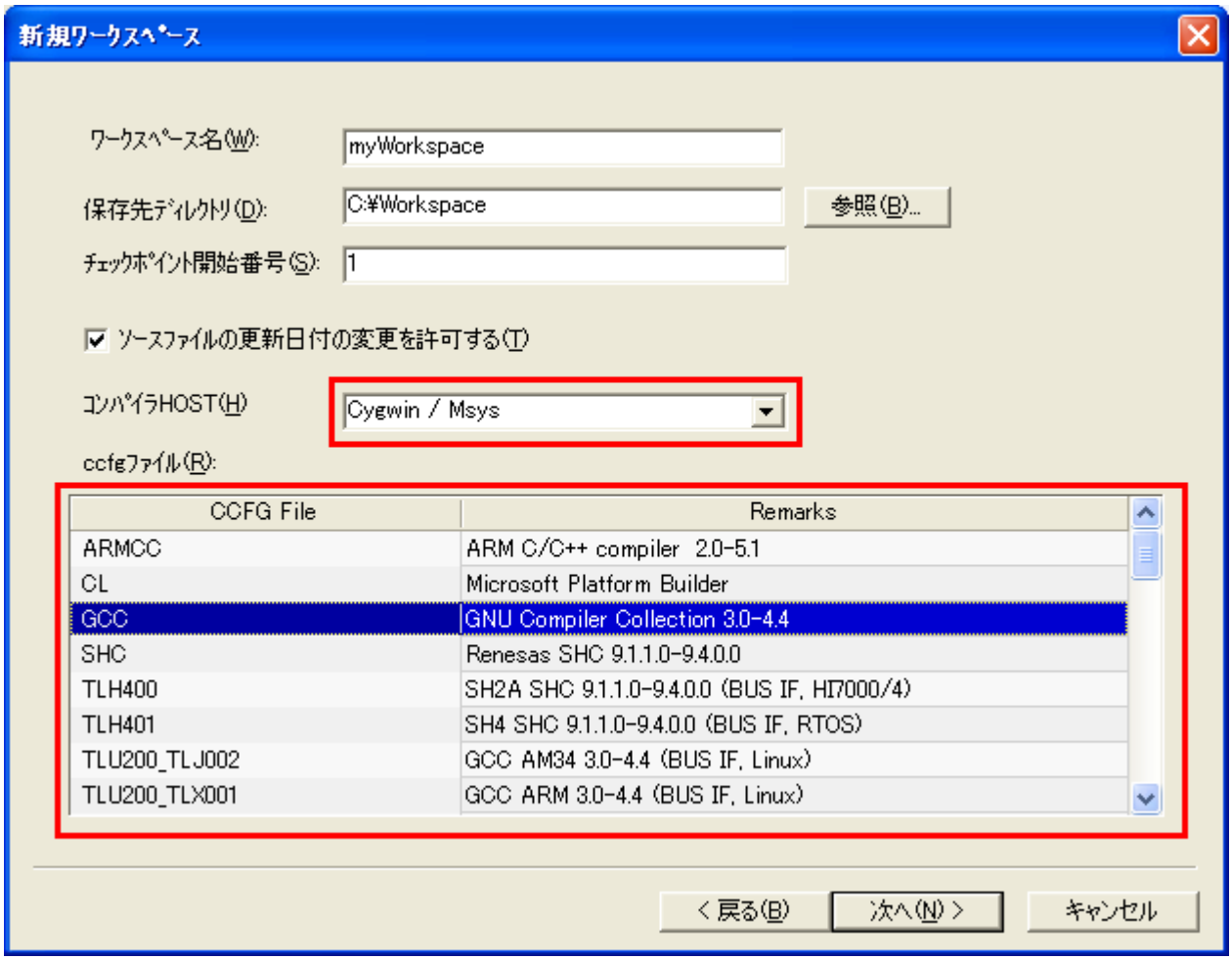
① [専用コンパイラ版]より、C/C++ を選択します。



② 設定が完了したら、「次へ」をクリックします。

15.2.4. 新規ワークスペース

設定項目	設定内容
ワークスペース名(W)	myWorkSpace
保存先フォルダ(D)	C:/WorkSpace
コンパイラ HOST（H）	Cygwin/Msys
CCFG ファイル(R)	使用しているコンパイラを選択してください

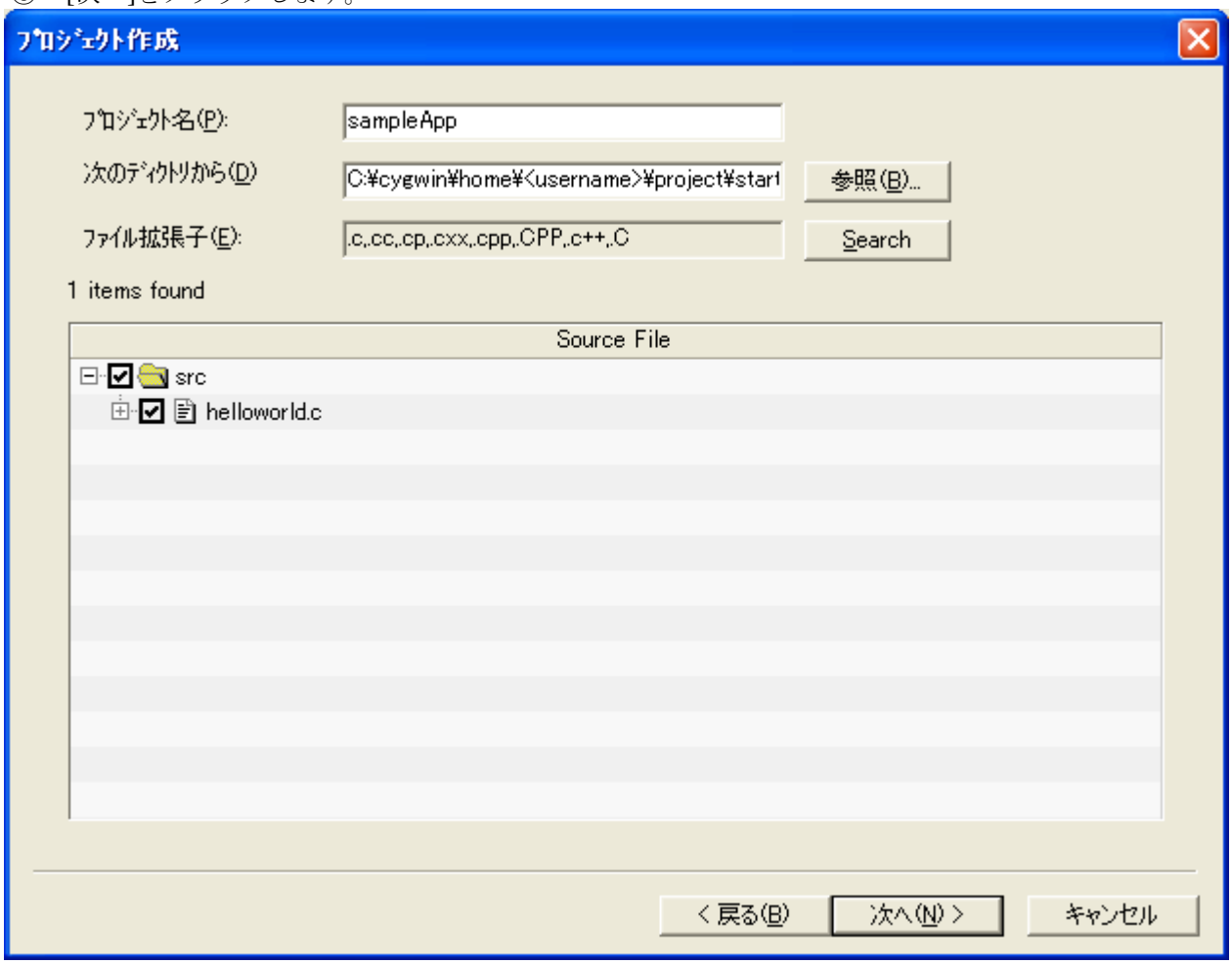


設定が完了したら、「次へ」をクリックします。

15.2.5. プロジェクト作成

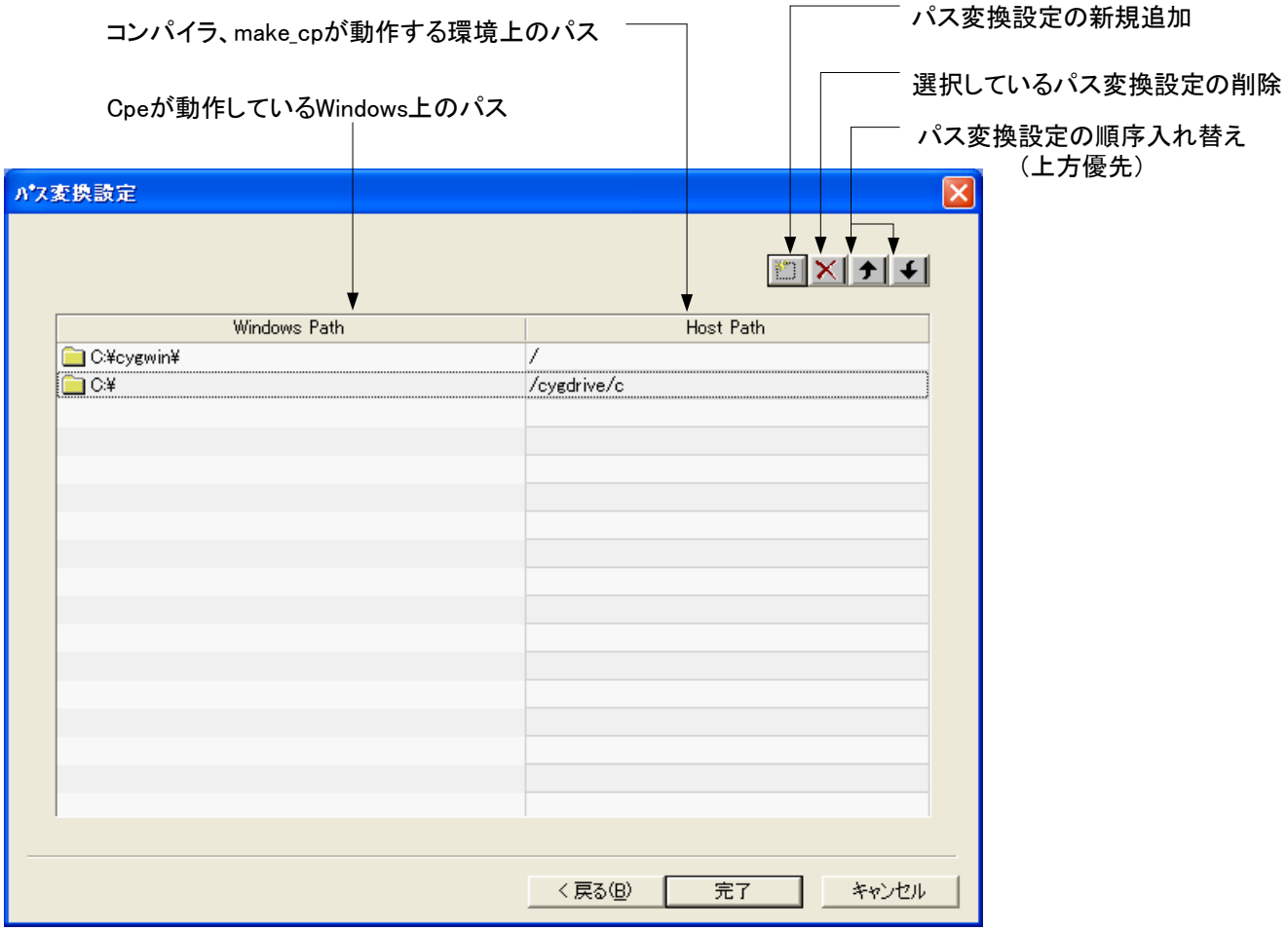
ここではプロジェクト名を"sampleApp"として作成した例を紹介します。

① [プロジェクト名]に、新規プロジェクト名「sampleApp」を入力してください。
② [次のディレクトリから]に、ソースファイルを検索するフォルダを入力してください。
③ [Search] を押してソースファイルの検索を開始してください。
④ [Source File]に見つかったソースファイルが表示されます。
⑤ プロジェクトに登録するファイルにチェックを入れます。
⑥ [次へ]をクリックします。

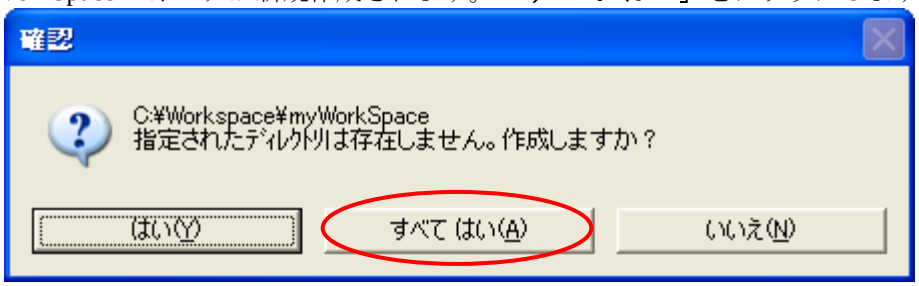


15.2.6. パス変換設定

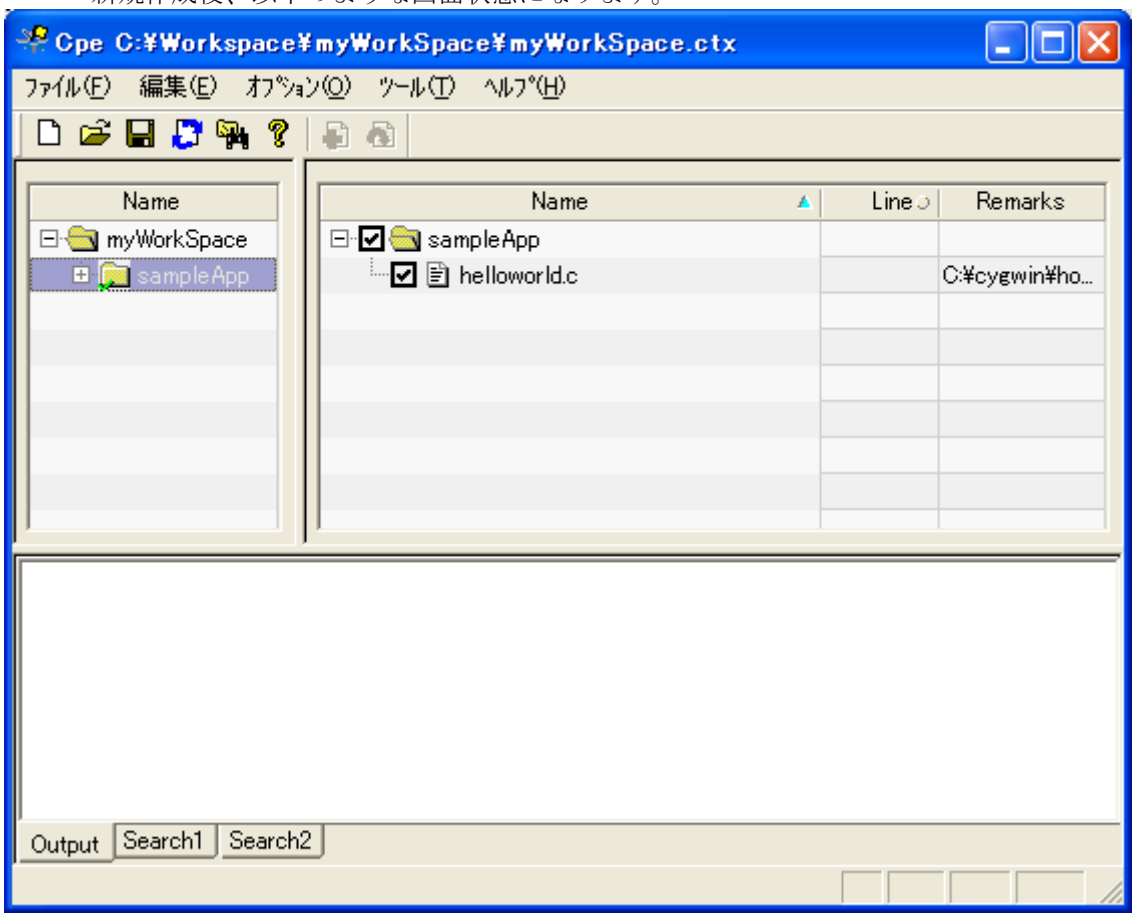
ワークスペースに登録したソースファイルのパスは Windows パスで管理されます。この Windows パスと Cygwin のパスを対応させるための変換条件設定です。
複数の条件を設定した場合、上方の設定が優先されます。



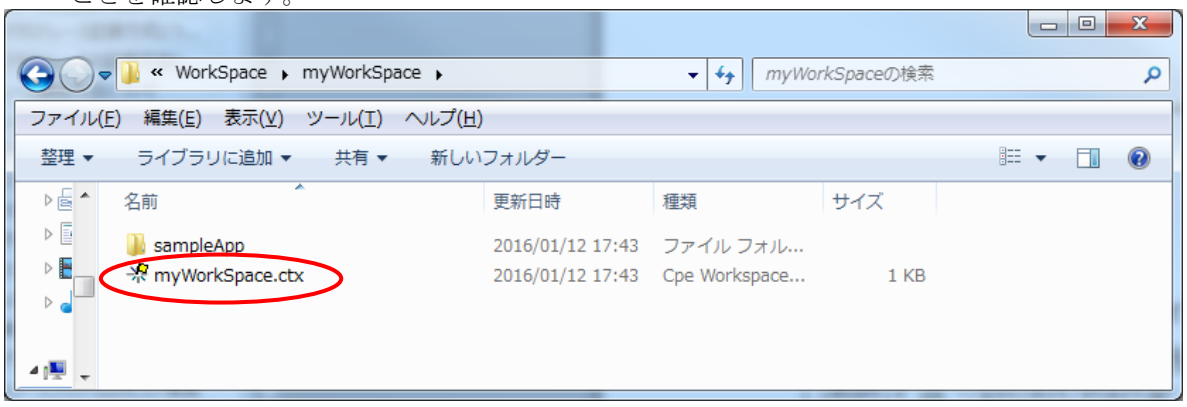
WorkSpace フォルダが新規作成されます。「すべて はい」をクリックします。



新規作成後、以下のような画面状態になります。

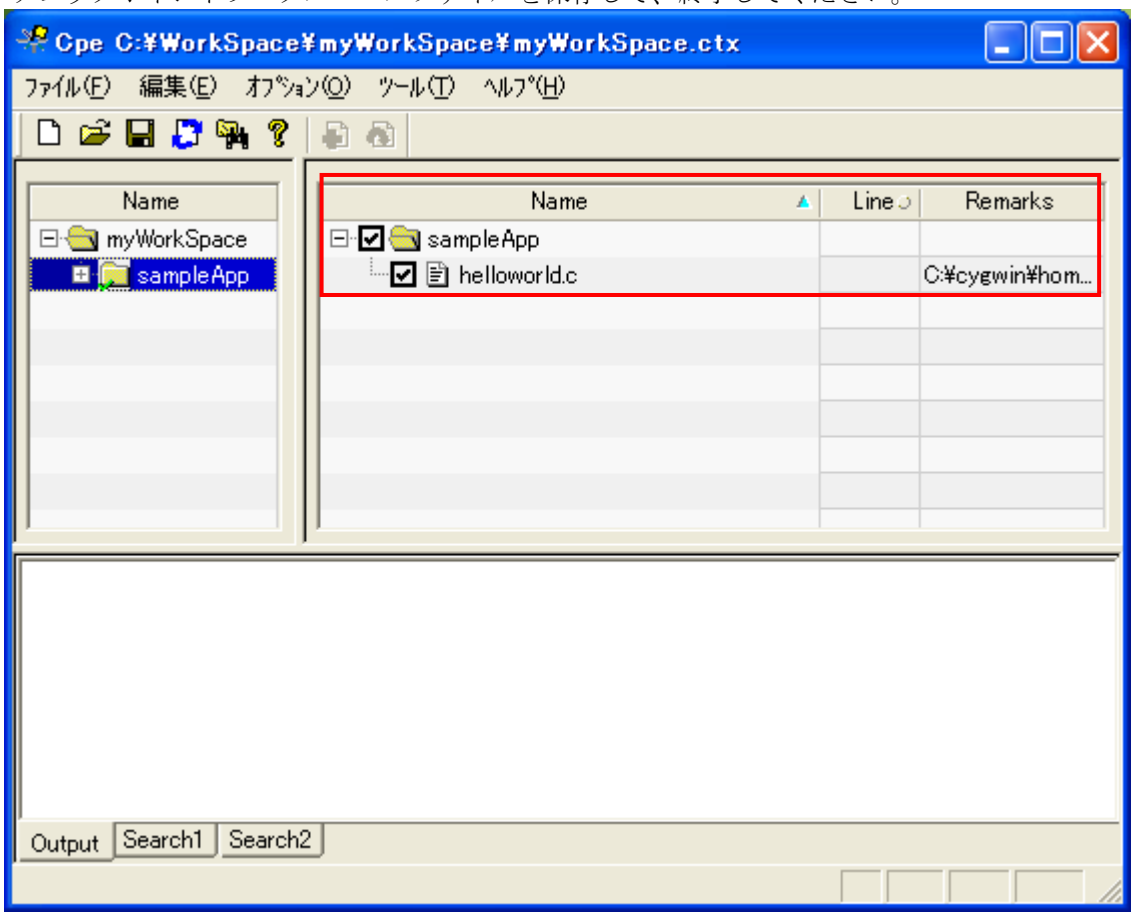


新規ワークスペース作成時に指定したフォルダにワークスペースファイル（拡張子.ctx ）が出来ていることを確認します。



15.2.7. チェックポイント挿入ソース選択の完了

チェックポイント対象のソースファイル登録が完了すると、対象のプロジェクト、ソースファイルが一覧表示されます。
チェックポイントワークスペースファイルを保存して、終了してください。



15.3. make_cp ツール設定

15.3.1. オリジナルの Makefile をコピーしましょう

Makefile をコピーし、Makefile.smt の名前で保存してください。

```
[HOST]$ cd ~/project/startup
[HOST]$ cp Makefile Makefile.smt
```

15.3.2. Makefile.smt を編集

変数 MAKE_CP、MAKE_CP_CFG、CC を修正し、make_cp(チェックポイント挿入ツール)を呼び出すようにします。

変数名称	説明
MAKE_CP	make_cp 実行ファイルを設定します。
MAKE_CP_CFG	make_cp 設定ファイルを設定します。
CC	make_cp の動作設定値を追加設定します。

Makefile.smt 修正前

```
MAKE_CP =
MAKE_CP_CFG =
CC = $(CROSS_COMPILE)gcc
```

Makefile.smt 修正後

```
MAKE_CP = C:/YDC/CheckPointTools/make_cp
MAKE_CP_CFG = -ctx C:/WorkSpace/myWorkSpace/myWorkSpace.ctx -cc
CC = $(MAKE_CP) $(MAKE_CP_CFG) $(CROSS_COMPILE)gcc
```

15.3.3. make の実行

① 一度 clean をしてください。

```
[HOST]$ make clean
rm -f helloworld helloworld.o
```

② Makefile.smt を指定して make をしてください。

```
[HOST]$ make –f Makefile.smt
```

③ 新しいオブジェクトが生成されたことを確認

make の実行結果にエラーがないことを確認し、helloworld が生成されたことを確認してください。

```
[HOST]$ ls helloworld
helloworld
```

15.4. 新しいオブジェクトを実行する

15.4.1. オブジェクトの組み込み

ユーザー環境に合わせて helloworld の組み込みを行ってください。

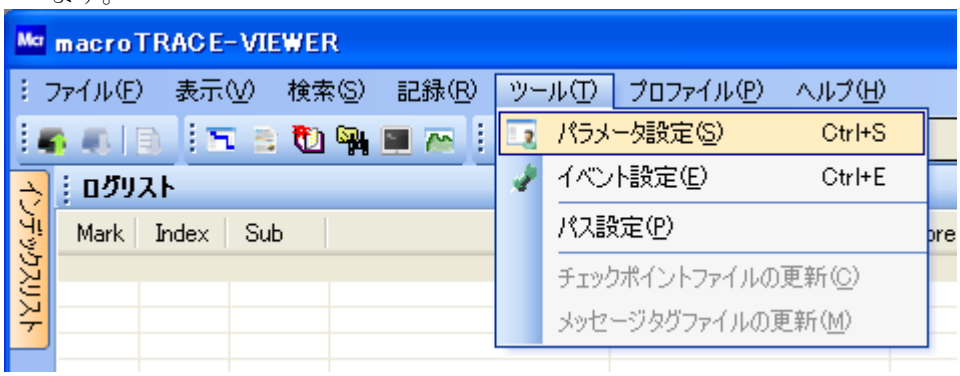
15.4.2. サンプルプログラムの実行

記録を開始しユーザーシステムを起動して helloworld を実行してください。

15.5. macroTRACE-VIEWER の設定

デバッグ情報を記録する際に、macroTRACE-VIEWER で必要な設定について説明します。

(1) **「ツール」→「パラメータ設定」**または、  を選択し、**「パラメータ設定」**ダイアログを表示します。

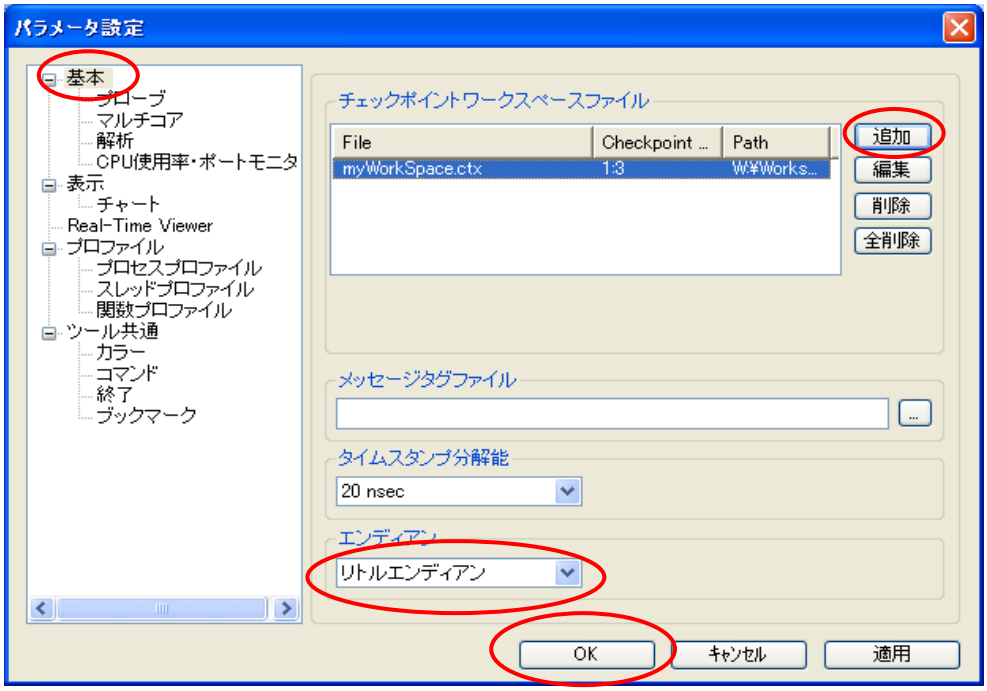


(2) 左のツリーから**「基本」**を選択します。

(3) **「チェックポイントワークスペースファイル」**に、「15.2.Cpe ツール設定」で作成した「*W:/Workspace/myWorkSpace/myWorkSpace.ctx*」を指定します。

(4) **「エンディアン」**をお使いの環境に合わせて選択します。

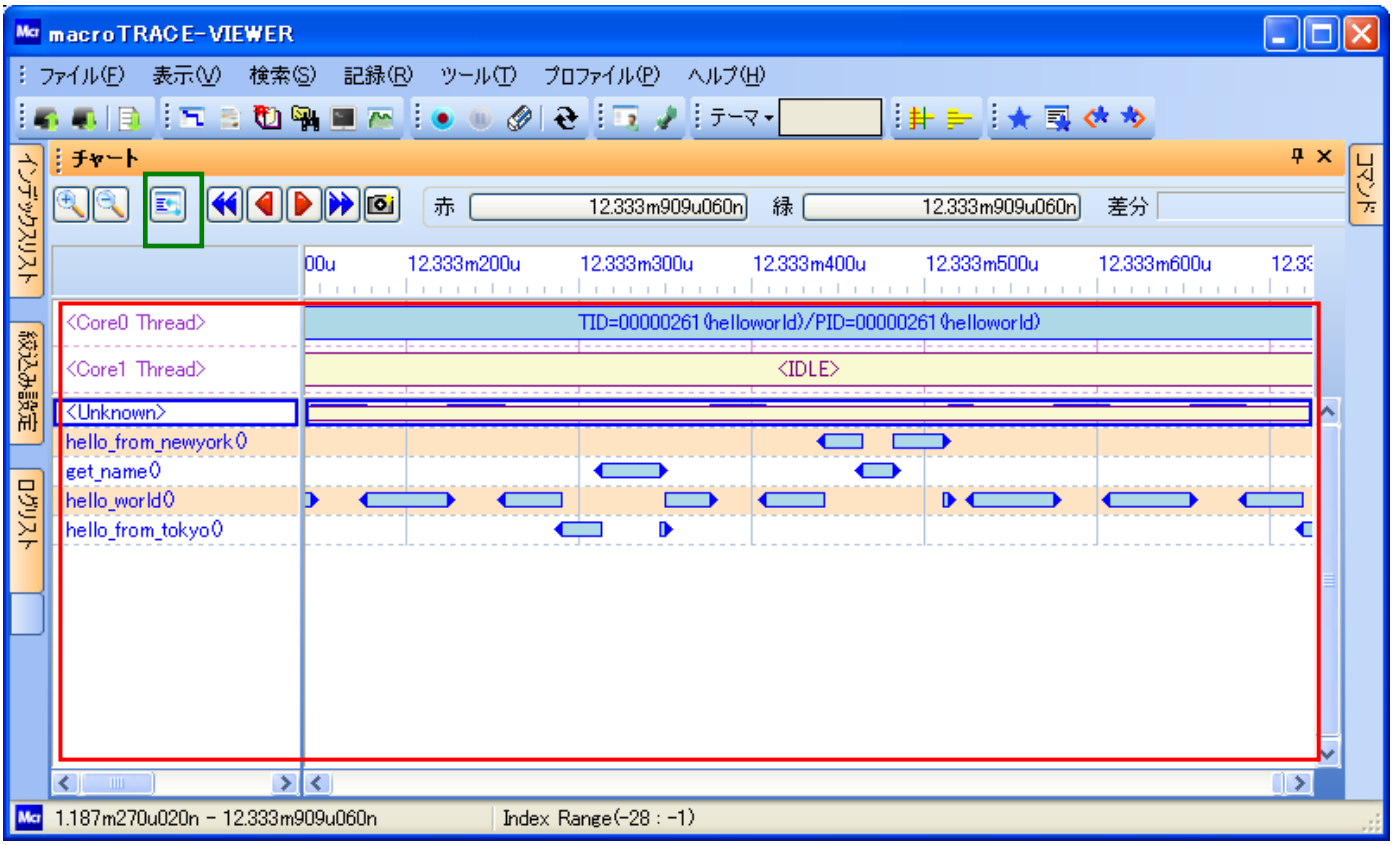
(5) 「OK」をクリックして、変更を保存します。



デバッグ情報を記録するための設定が完了しました。

15.6. システムマクロトレース取得例

取得例は下図のようになります。
関数チャートが表示されない場合、
緑枠のボタンが関数チャート/スレッドチャートの切り替えボタンですので、
ボタンでチャート表示を切り替えて関数チャートが表示されるか確認してください。



16. 付録

16.1. 実施項目の確認事項

実施項目の確認をまとめて記載します。

16.1.1. API ライブラリ関連の確認事項

番号	確認項目	チェック
1	トレースデータ出力先アドレスを設定しましたか 「5.1 トレースデータ出力先設定について」	□
2	トレースデータ出力サイズは設定しましたか？ 「5.2 トレースデータサイズ指定について」	□
3	タイムスタンプ取得処理実装をしましたか？ 「5.3.1_SMT_GetTimeStamp 関数」	□
4	コア ID 取得関数を実装しましたか？ 「5.3.2_SMT_GetCoreID 関数」	□
5	割り込みマスク制御関数の実装は行いましたか？ （_SMT_InterruptRestore,_SMT_Interrupt Disable) 「5.1 割り込みマスク制御関数の実装」	□
6	API ライブラリのビルドは問題なく出来ましたか？ 「7.3.ビルドの実行」	□
7	API ライブラリをユーザーシステムに組み込みましたか？ 「8.API ライブラリの組み込み	□

16.1.2. 関数トレース出力関連

番号	確認項目	チェック
1	Cpe でパス変換設定は正しく実施しましたか？ 「15.2Cpe ツール設定」	□
2	Makefile を編集して make_cp 経由でビルドするようにしましたか？ 「15.3make_cp ツール設定」	□
3	macroTRACE-VIEWER で作成したワークスペースファイルの設定をしましたか？ 「15.5.macroTRACE-VIEWER の設定」	□