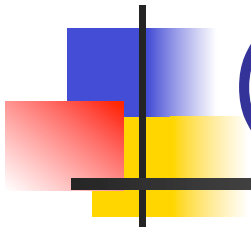
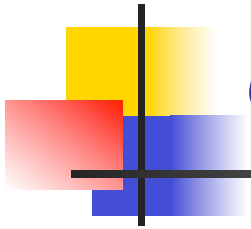


TOPPERS/SSPを動かしてみる（HEWシミュレータ編）

SECTION1
アライブビジョンソフトウェア株式会社
髙橋和浩

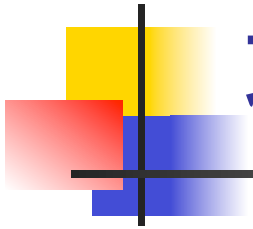


このセクションの目的

- PCにTOPPERS/SSPをインストール、動作確認
- 対象者:

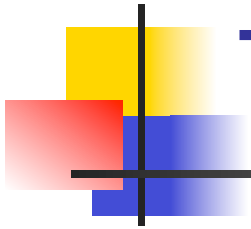
  WindowsPCをお持ちの方

  C言語がある程度わかる方



項目一覧

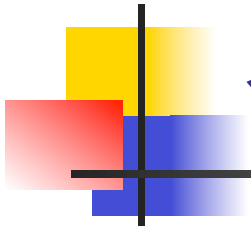
1. TOPPERS/SSPの概要
2. TOPPERS/SSPのインストールするパッケージの説明
3. 作業説明と作業
4. まとめ



TOPPERS/SSPの概要

- 最少セットカーネル
- リアルタイムカーネル(RTOS)
- リアルタイム性能はそのまま、メモリ使用を抑えた
- 特徴的、「待ち状態を持たない」
- スタック共有も可能

- 待ち状態を持たない--> プリエンプトされないので
- コンテキストの保存が不要になり
- シンプルに実装されている



インストールするパッケージ

用意するファイル

1.Interface誌附属RX62Nボード用簡易パッケージ

ssp_frk_rx62n_hew-20111120.lzh

2.統合開発環境 High-Performance Embedded Workshop

【無償評価版】RXファミリ用C/C++コンパイラパッケージ V.1.02 Release 01

ccrxv102r01_ev.exe

必要な環境: WindowsPC 32bit でも　64bitでも

Interface誌附属RX62Nボード用簡易パッケージ

ssp_cq_frk_fm3_gcc-20140307.tar.gz	ARM Cortex-M3	GCC	1.2.1	2014-03-07
ssp_cq_frk_fm3_gcc-20130425.tar.gz	ARM Cortex-M3	GCC	1.2.0	2013-04-25

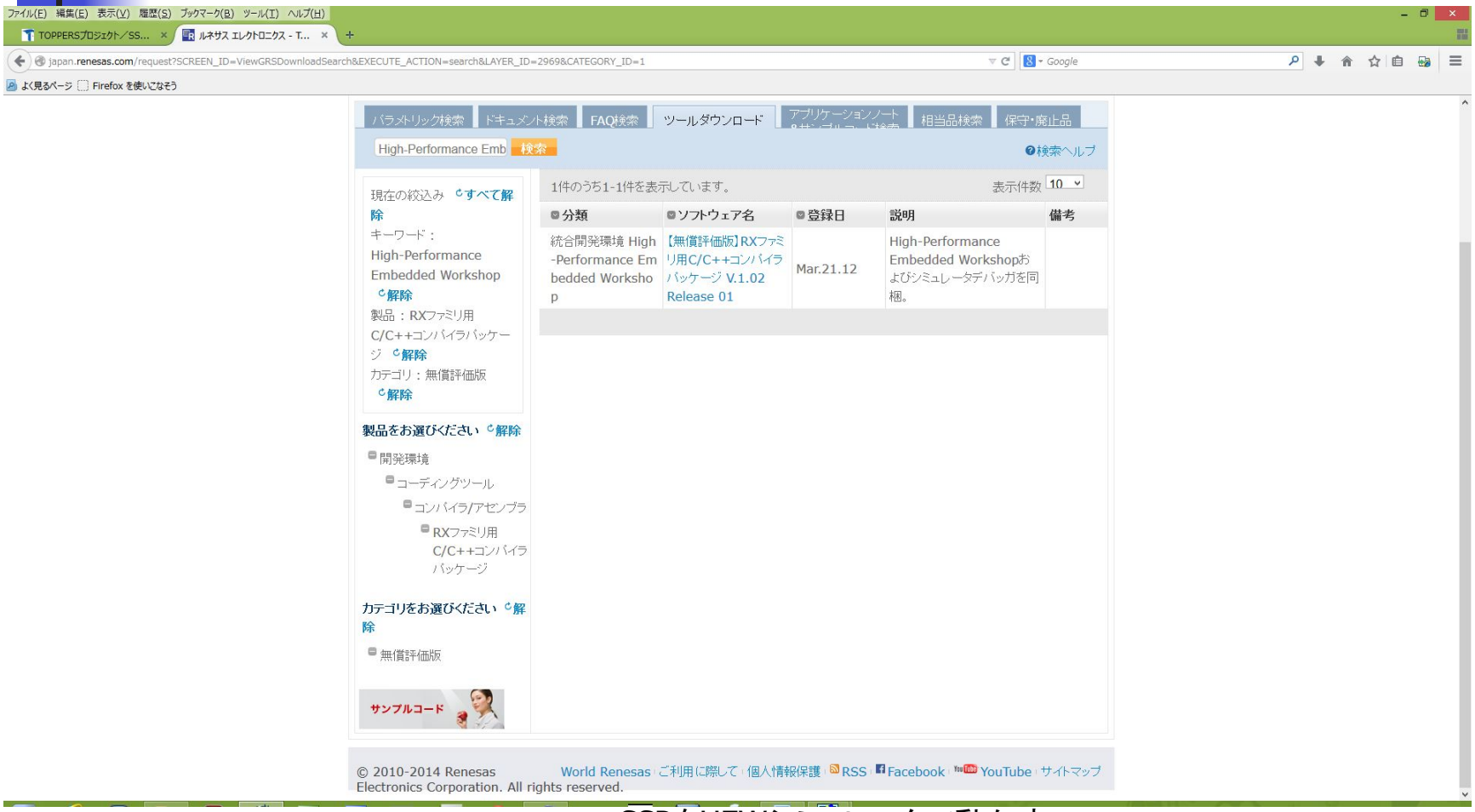
DesignWave誌附属CQ_STARM基板用簡易パッケージ

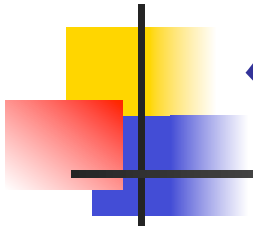
パッケージ	プロセッサ(チップ)	開発環境	非依存部のバージョン	リリース日
ssp_cq_starm_gcc-20120607.tar.gz	STM32F	GCC	1.1.1	2012-06-07
ssp_cq_starm_gcc-20120423.tar.gz	STM32F	GCC	1.1.0	2012-05-07

Interface誌附属RX62Nボード用簡易パッケージ

パッケージ	プロセッサ(チップ)	開発環境	非依存部のバージョン	リリース日
ssp_frk_rx62n_hew-20111120.lzh	RX62N	HEW	1.1.0	2011-11-21

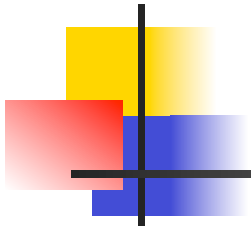
統合開発環境 High-Performance Embedded Workshop





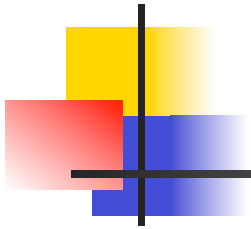
作業説明 大項目

- 1)ダウンロードとセットアップ
- 2)HEWでHELLOWORLD
- 3)SSPをそのままビルド
- 4)HEWのシミュレータ設定
- 5)SSPのコンソール出力のソース修正
- 6)シミュレータ用の調整
- 7)実行



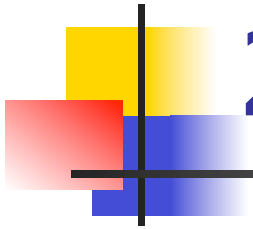
1)ダウンロードとセットアップ

- (1)パッケージのダウンロード
- (2)RXコンパイラのインストール
- すでにHEWがある場合はマルチインストールがお勧め
- (3)TOPPERS/SSPの解凍
- 日本語フォルダ名のパス以外に解凍します。



1)ダウンロードとセットアップ

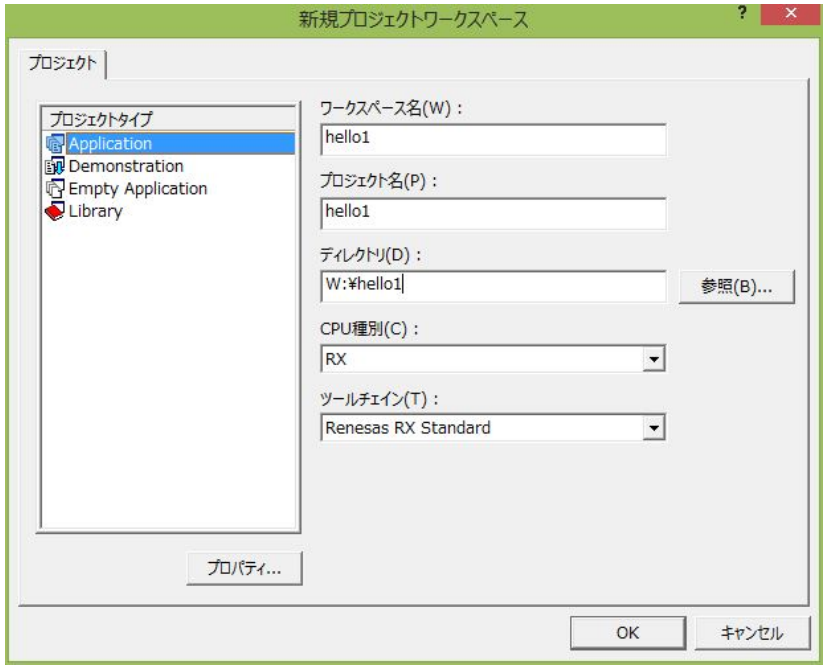
- ディレクトリの例
- 参考として、Windowsのsubstコマンドを紹介します。
- 形式
- C:\>subst ドライブレター: マウントするディレクトリ
- C:\>subst
- でマウント状態が確認できます。こちらの環境は
- W:\>subst
- W:\: => D:\usr\ポリテクITRONセミナ\OS自作セミナ\SWEST\SEC1-WORKSPACE

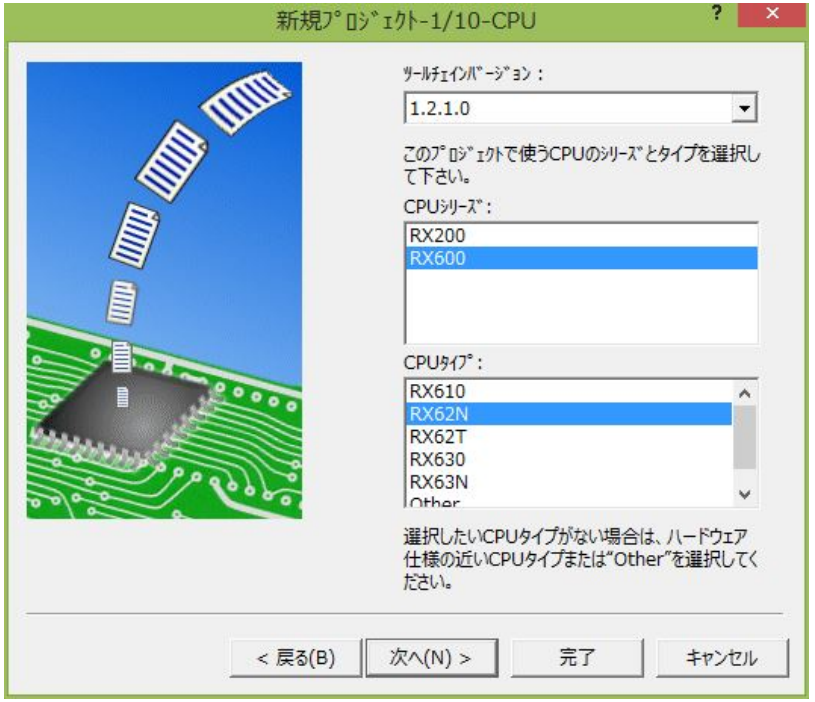


2)HEWでHELLOWORLD

- ウイザードに従って、プログラムを作ります。
- main()のひな形までできます。
- シミュレータ用のシリアルドライバを自動生成します。

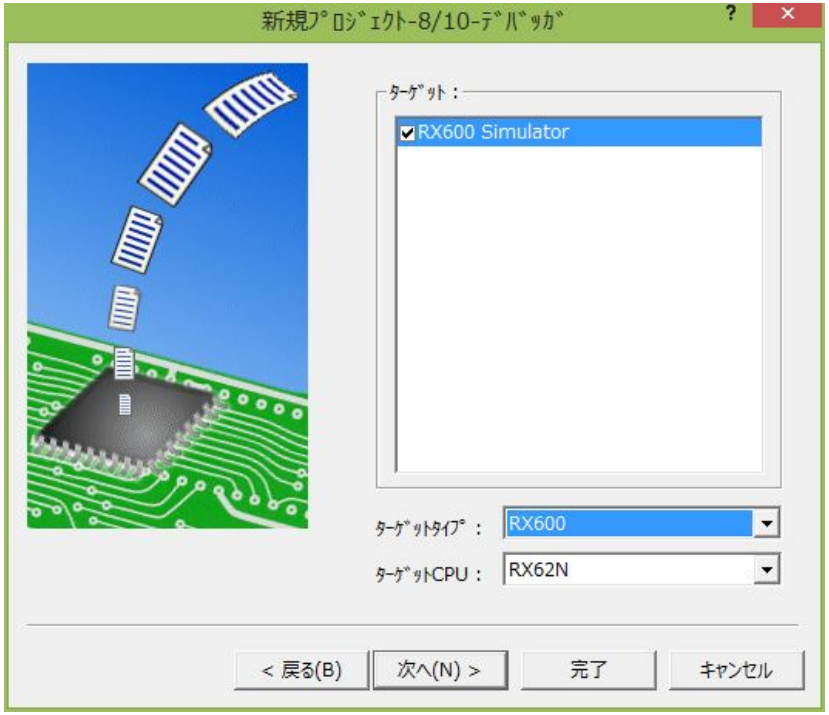
2)HEWでHELLOWORLD

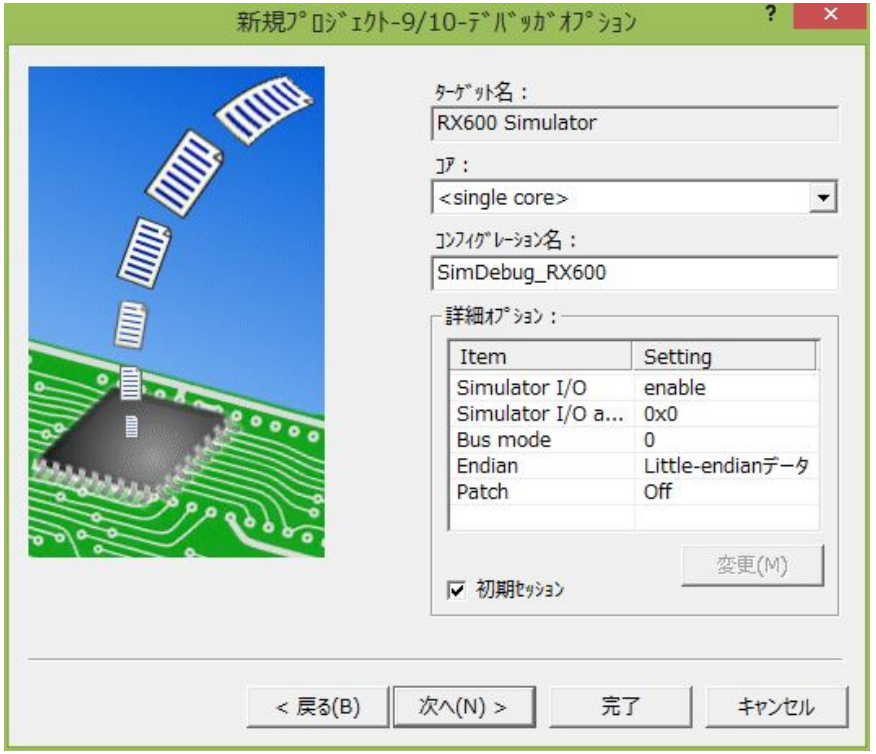




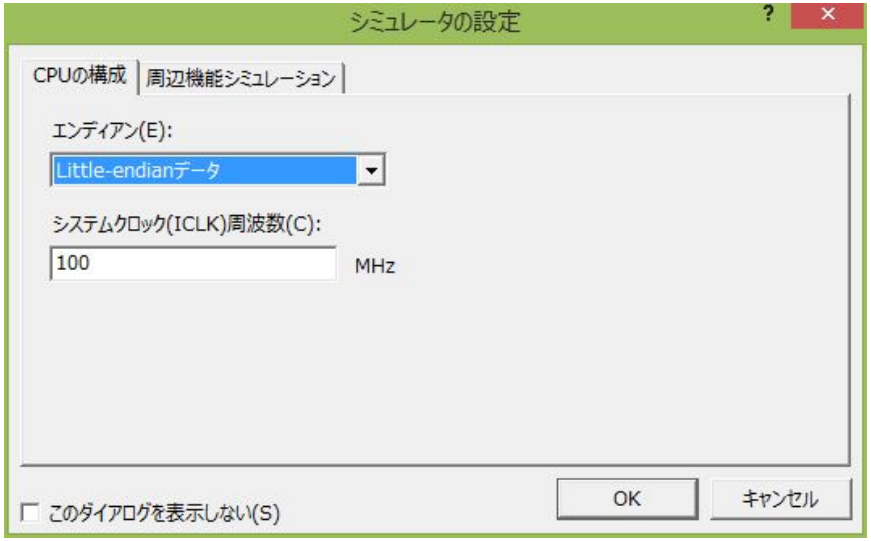
2)HEWでHELLOWORLD

2)HEWでHELLOWORLD





2)HEWでHELLOWORLD

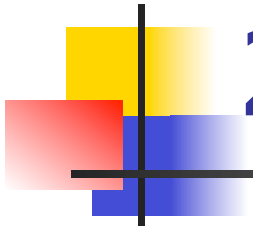


2)HEWでHELLOWORLD

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :hello1.c                                              */
/*  DATE        :Mon, Jul 21, 2014                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :RX62N                                                 */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.4.53).    */
/*  NOTE:THIS IS A TYPICAL EXAMPLE.                                    */
/*                                                                     */
/***********************************************************************/

#include<stdio.h>

//#include "typedefine.h"
#ifdef __cplusplus
//#include <ios>                        // Remove the comment when you use ios
//_SINT ios_base::Init::init_cnt;       // Remove the comment when you use ios
#endif
```

2)HEWでHELLOWORLD

```
void main(void);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif

void main(void)
{
        printf("Hello SWEST\n");
}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```



2)HEWでHELLOWORLD

HEWシミュレータ設定

「表示」「CPU」「デバッグコンソール」

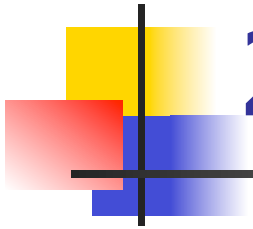
ビルドと実行

「ビルド」「すべてをビルド」

「デバッグ」「ダウンロード」

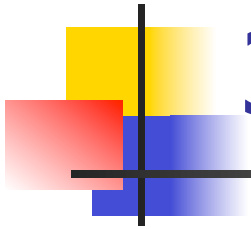
「デバッグ」「CPUのリセット」

「デバッグ」「実行」



2)HEWでHELLOWORLD

- ファイルをコピーします。
- \hello1\lowlvl.src
- これをどこかに置いておきます。
- ここでは仮に　W:\　にコピーします。



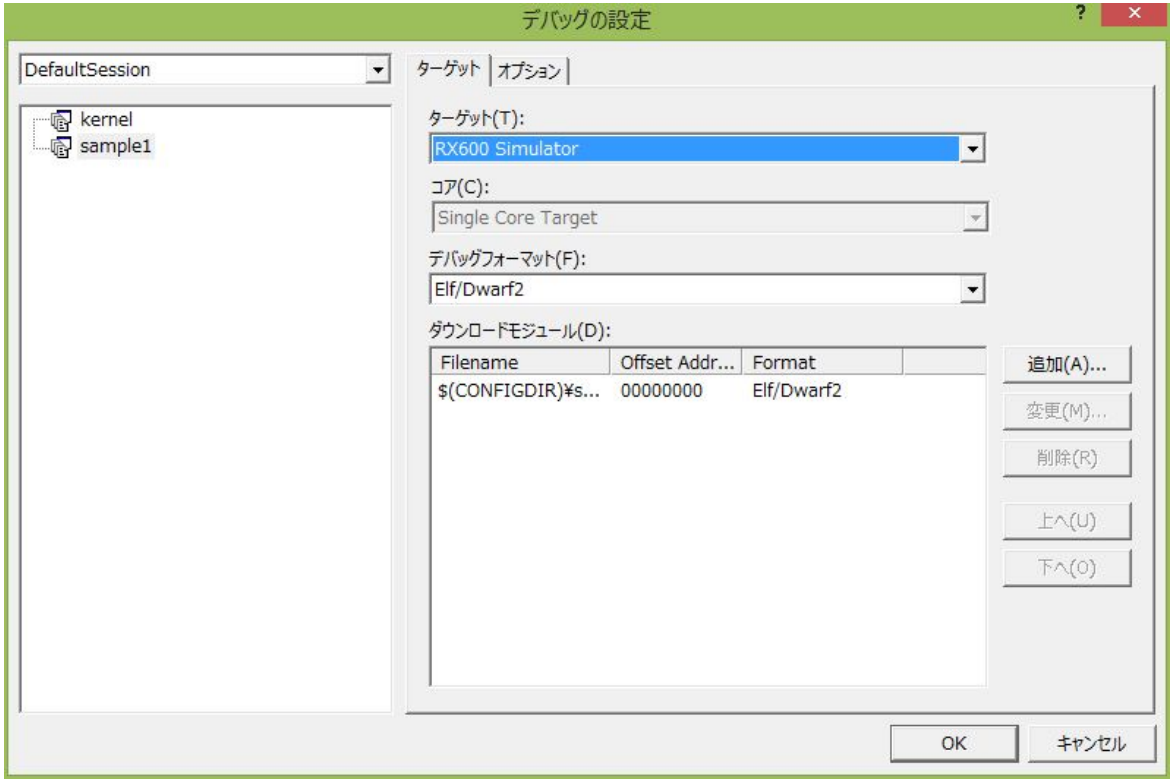
3)SSPをそのままビルド

- （1）解凍
- ssp_frk_rx62n_hew-20111120\target\frk_rx62n_hew\sample_workspace\sample1.hws
- をダブルクリック

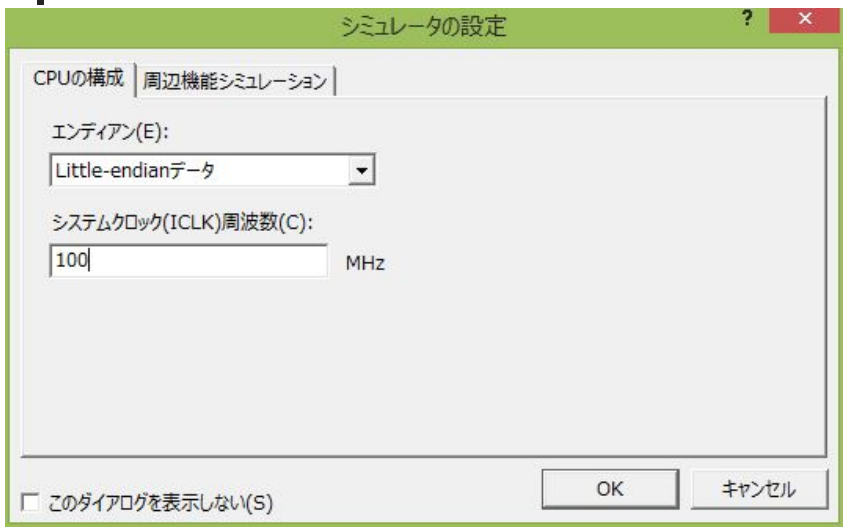
- （2)ビルド
- asm_config
- cfg
- kernel
- sample1
- それぞれアクティブプロジェクトにして
- すべてをビルドします。
- メニュー 「プロジェクト」「アクティブプロジェクトの設定」「該当プロジェクト」
- メニュー「ビルド」「すべてをビルド」

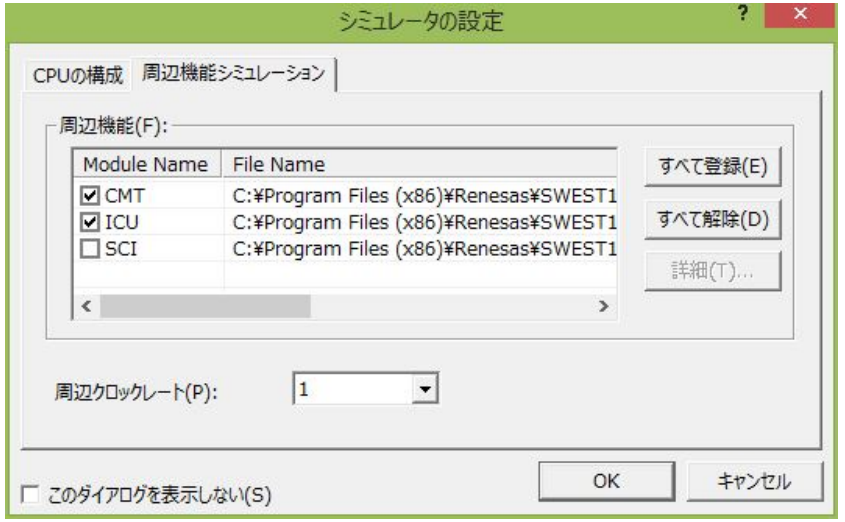
4)HEWのシミュレータ設定

- 「デバッグ」「デバッグの設定」



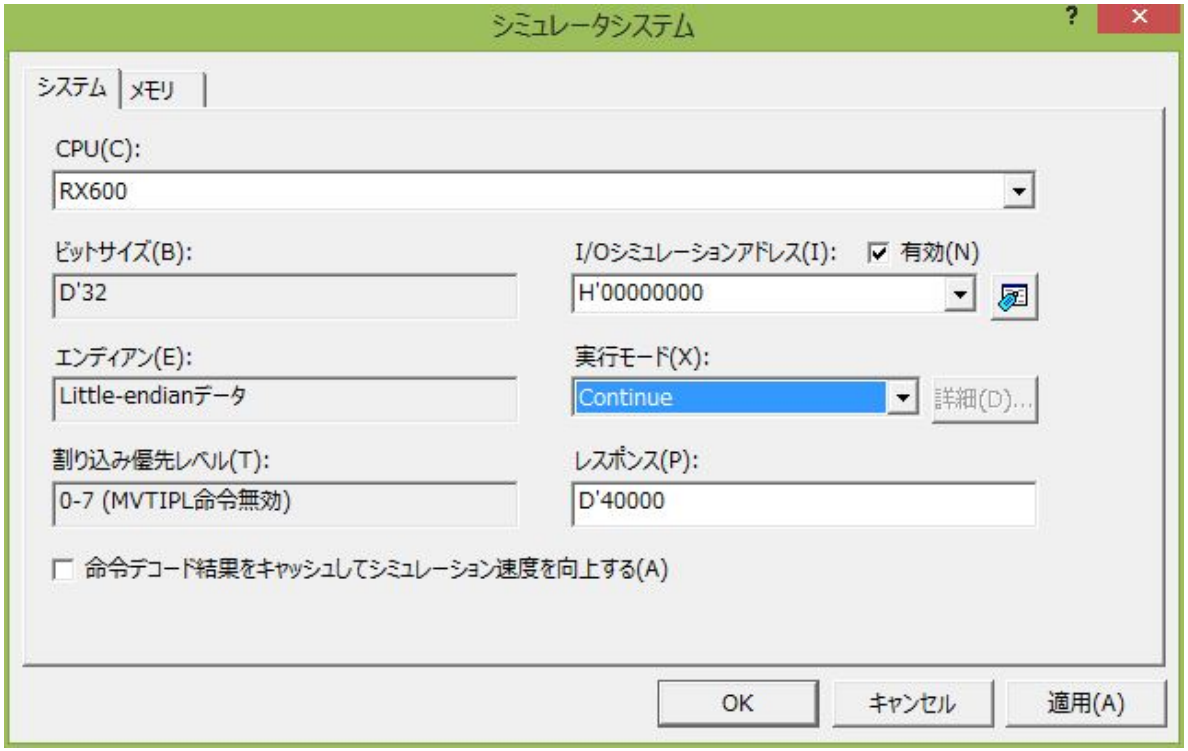
4)HEWのシミュレータ設定





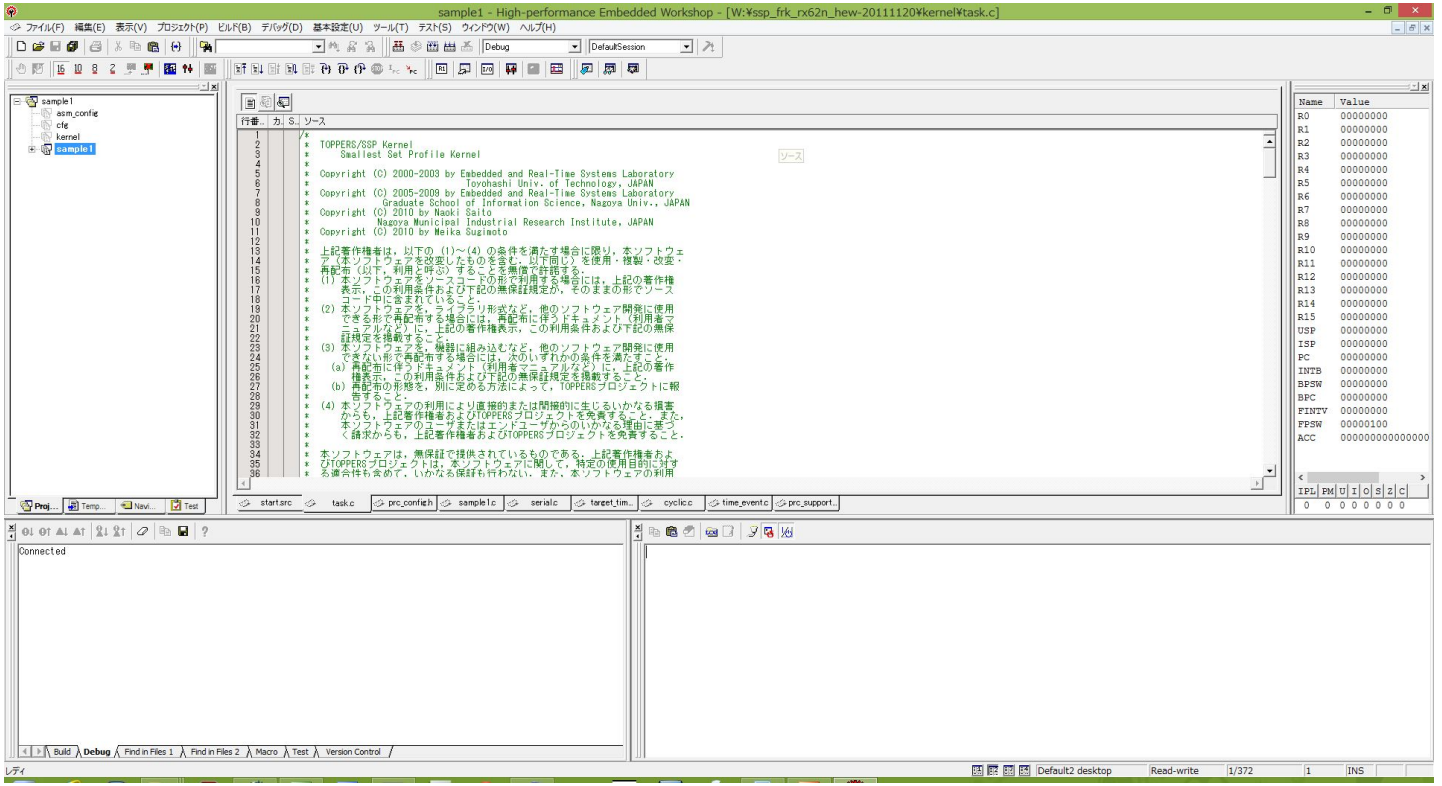
4)HEWのシミュレータ設定

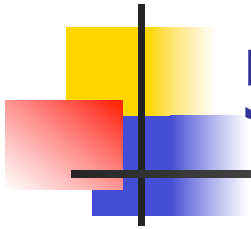
- 「基本設定」「シミュレータシステム」



4)HEWのシミュレータ設定

- 「表示」「CPU」「デバッグコンソール」





5)SSPのコンソール出力のソース修正

(1)lowlvl.src をプロジェクトに追加
　場所は特に指定はありません
　w:\にコピーしたものを　\pdic\rx600 にコピーし、
　メニュー「プロジェクト」「ファイルの追加」 で追加します
(2)上記シリアルドライバを呼び出すソースの修正

- \pdic\rx600\rx600_uart.c: rx600_uart_snd_chr(SIOPCB *p_siopcb, char_t c)
- \pdic\rx600\rx600_uart.c: void　rx600_uart_pol_putc(char_t c, ID siopid)
- \syssvc\serial.c: serial_wri_chr(SPCB *p_spcb, char_t c)

(3)Sample1にログ出力(LOGPUT)追加

5)SSPのコンソール出力のソース修正

- \pdic\rx600\rx600_uart.c:
- /* 標準入力からの1文字入力処理 */
- **extern void charput(unsigned char);**
- /*
- * シリアルI/Oポートへのポーリングでの出力
- */
- void
- rx600_uart_pol_putc(char_t c, ID siopid)
- {
- #if 0
- const SIOPINIB *p_siopinib;
- ----------省略----------
- sil_wrb_mem((void *)p_siopinib->tdreg, (uint8_t)c);
- #endif
- **charput(c);**
- }

5)SSPのコンソール出力のソース修正

```
/*
 *  シリアルI/Oポートへの文字送信
 */
bool_t
rx600_uart_snd_chr(SIOPCB *p_siopcb, char_t c)
{
        bool_t ercd = false;
#if 0
        if((sil_reb_mem(
                (void *)p_siopcb->p_siopinib->ssrreg) & SCI_SSR_TEND_BIT) != 0){
                sil_wrb_mem((void *)p_siopcb->p_siopinib->tdreg, (uint8_t)c);
                ercd = true;
        }
#endif
        charput(c);
        ercd = true;

        return ercd;
}
```

5)SSPのコンソール出力のソース修正

- syssvc\serial.c: serial_wri_chr(SPCB *p_spcb, char_t c)
- /* 標準入力からの1文字入力処理 */
- **extern void charput(unsigned char);**

- /*
- * シリアルポートへの1文字送信
- */
- static ER_BOOL
- serial_wri_chr(SPCB *p_spcb, char_t c)
- {
- #if 0
- bool_t buffer_full;
- ER ercd, rercd;
- /*
- * LFの前にCRを送信する.
- */

----------略----------

- #endif
- **charput(c);**
- return(false);
- }

5)SSPのコンソール出力のソース修正

- サンプルプログラム　sample1.c にプリント文追加
- void main_task(intptr_t exinf)
- {
- 　　static ID tskid = TASK1;
- 　　static uint_t tskno = 1;
- 　　char_t c;
-

- 　　**LOGPUT("Sample program ACTIVE.\n");**

- 　　/* シリアルポートからの文字受信 */
- 　　if(serial_rea_dat(SIO_PORTID , &c , 1) > 0)
- 　　{

-------以下　略----------

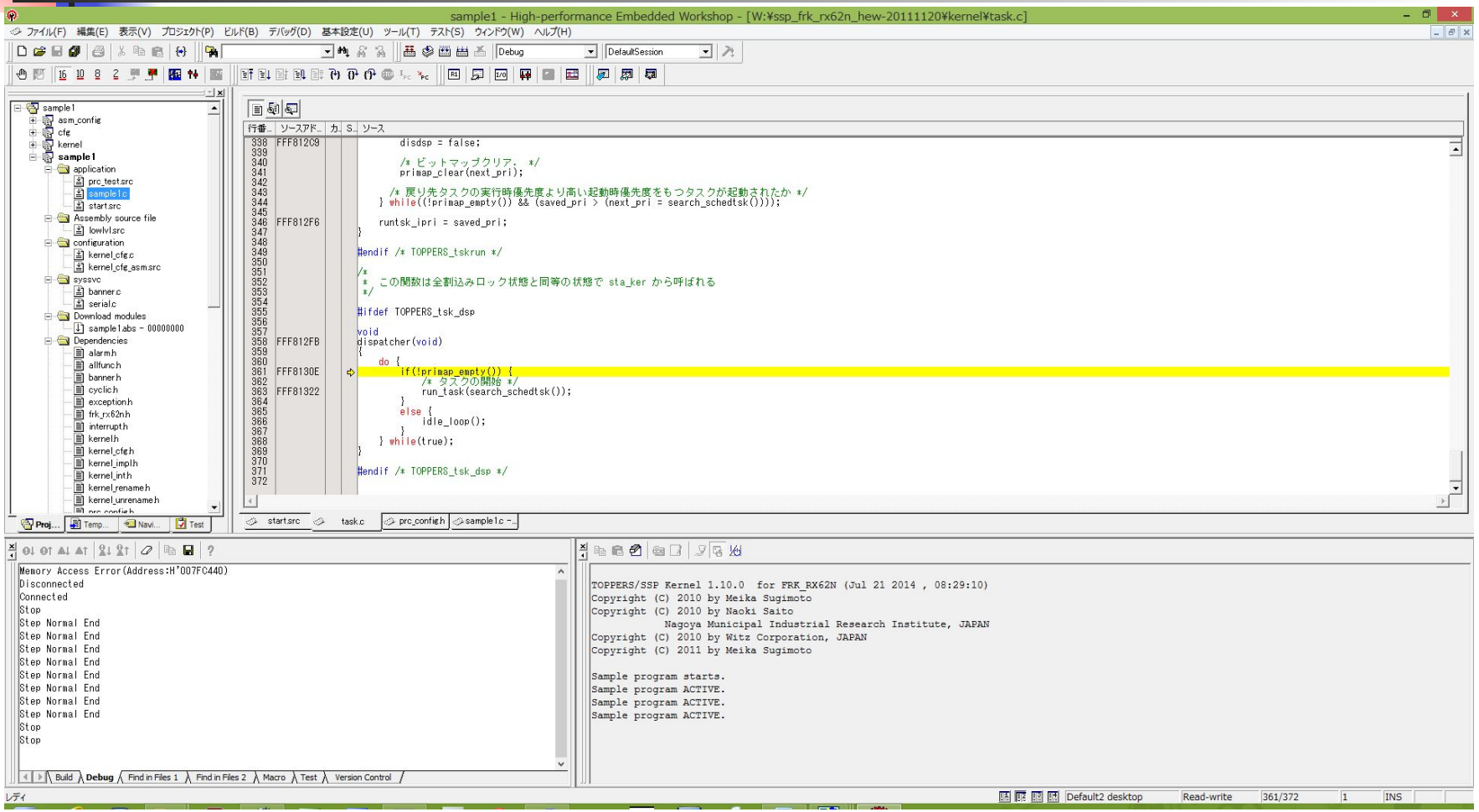
6)シミュレータ用の調整

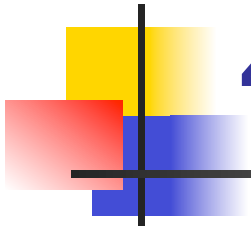
- サンプルプログラム　sample1.c fgのタイマーを10分の1にする
- #include "sample1.h"
- CRE_TSK(INIT_TASK , { TA_ACT , 0 , init_task , INIT_PRIORITY , STACK_SIZE , NULL });
- CRE_TSK(MAIN_TASK , { TA_NULL , 0 , main_task , MAIN_PRIORITY , STACK_SIZE , NULL });
- CRE_TSK(TASK1 , { TA_NULL , 1 , task , TASK1_PRIORITY , STACK_SIZE , NULL });
- CRE_TSK(TASK2 , { TA_NULL , 2 , task , TASK2_PRIORITY , STACK_SIZE , NULL });
- CRE_TSK(TASK3 , { TA_NULL , 3 , task , TASK3_PRIORITY , STACK_SIZE , NULL });

- DEF_EPRI(TASK3 , { TASK3_EXEPRIORITY });

- #ifdef USE_TIMER
- CRE_CYC(MAIN_CYC , { TA_STA , MAIN_TASK , cyclic_handler , **200 , 100** });
- #endif /* USE_TIMER */
- ----------以下　略----------

7)実行


sample1 - High-performance Embedded Workshop - [W:¥ssp_frk_rx62n_hew-20111120¥kernel¥task.c]
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) 基本設定(U) ツール(T) テスト(S) ウィンドウ(W) ヘルプ(H)
Debug
DefaultSession
sample1
asm_config
cfg
kernel
sample1
application
prc_test.src
sample1.c
start.src
Assembly source file
lowlvl.src
configuration
kernel_cfg.c
kernel_cfg_asm.src
syssvc
banner.c
serial.c
Download modules
sample1.abs - 00000000
Dependencies
alarm.h
allfunc.h
banner.h
cyclic.h
exception.h
frk_rx62n.h
interrupt.h
kernel.h
kernel_cfg.h
kernel_impl.h
kernel_int.h
kernel_rename.h
kernel_unrename.h
行番... ソースアド... カ... S... ソース
338 FFF812C9 disdsp = false;
339
340 /* ビットマップクリア。 */
341 primap_clear(next_pri);
342
343 /* 戻り先タスクの実行時優先度より高い起動時優先度をもつタスクが起動されたか */
344 } while((!primap_empty()) && (saved_pri > (next_pri = search_schedtsk())));
345
346 FFF812F8 runtsk_ipri = saved_pri;
347 }
348
349 #endif /* TOPPERS_tskrun */
350
351 /*
352 * この関数は全割込みロック状態と同等の状態で sta_ker から呼ばれる
353 */
354
355 #ifdef TOPPERS_tsk_dsp
356
357 void
358 FFF812FB dispatcher(void)
359 {
360 do {
361 FFF8130E if(!primap_empty()) {
362 /* タスクの開始 */
363 FFF81322 run_task(search_schedtsk());
364 }
365 else {
366 idle_loop();
367 }
368 } while(true);
369 }
370
371 #endif /* TOPPERS_tsk_dsp */
372
start.src
task.c
prc_config.h
sample1.c -...
Memory Access Error(Address:H'007FC440)
Disconnected
Connected
Stop
Step Normal End
Step Normal End
Step Normal End
Step Normal End
Step Normal End
Step Normal End
Step Normal End
Step Normal End
Stop
Stop
Build
Debug
Find in Files 1
Find in Files 2
Macro
Test
Version Control
TOPPERS/SSP Kernel 1.10.0 for FRK_RX62N (Jul 21 2014 , 08:29:10)
Copyright (C) 2010 by Meika Sugimoto
Copyright (C) 2010 by Naoki Saito
Nagoya Municipal Industrial Research Institute, JAPAN
Copyright (C) 2010 by Witz Corporation, JAPAN
Copyright (C) 2011 by Meika Sugimoto
Sample program starts.
Sample program ACTIVE.
Sample program ACTIVE.
Sample program ACTIVE.
レディ
Default2 desktop
Read-write
361/372
INS



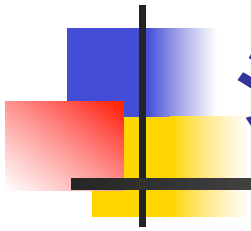
4.まとめ

- Sourceforge
- https://sourceforge.jp/users/alvstakahashi/pf/TOPPERS_ETC/wiki/FrontPage
- Interface誌　RX62N版のHEW実行形式
- E1エミュレータ版、SIOなしコンソール版

- カーネルを実際に動作させること　カーネルを読む理解を高める

- SSPは実装がシンプル　--> カーネル学習に向き

- カーネルカスタマイズセミナーを計画（ポリテクセンター兵庫）

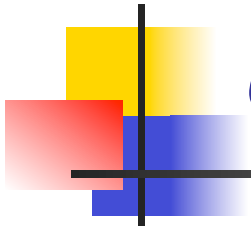


シュリンク版SSPの概要とその組込み

SECTION 2

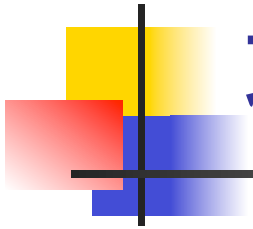
アライブビジョンソフトウェア株式会社

髙橋和浩



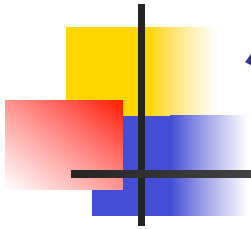
このセクションの目的

- TOPPERS/SSPの派生版として、そのシュリンク版としてshrink-sp-rx62nを開発
- その紹介、組込み手順を紹介



項目一覧

- 1)シュリンク版の紹介
- 2)組込み作業
- 3)まとめ

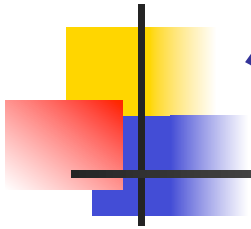


シュリンク版のSSPの紹介

- Interface誌附属RX62Nボード用のみ （現在）
- シュリンク版は、開発ツールをベースにスケジューラ/ディスパッチャ部分のみライブラリとして、利用できるようにしたものです。

- シュリンク版の目的:
- 非RTOS環境からRTOS環境に移行しやすくすることを念頭においています。
- またカーネル学習にもよいと考えています。

- 割り込みの管理、スタートアップおよびデバッグ機能は、開発環境側のものを利用するものとし、SSPはスケジューラ/ディスパッチャのみのものとする。
- ソース量が減り、デバッグ用のドライバも不要になるため移植性が向上すると考えます。

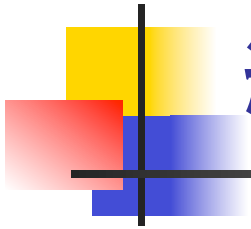


シュリンク版の紹介

機能

- 一部のサービスコール未実装

- TOPPERS/SSPのリアルタイム性能、省メモリ、省スタックはそのまま



組込み作業一覧

1) ツール等のインストール

2)RXコンパイラのセットアップと
　HELLOWORLDの作成

3)シュリンク版SSPの組込み

4)実行

1）ツール等のインストール

必要なもの

・Windows PC

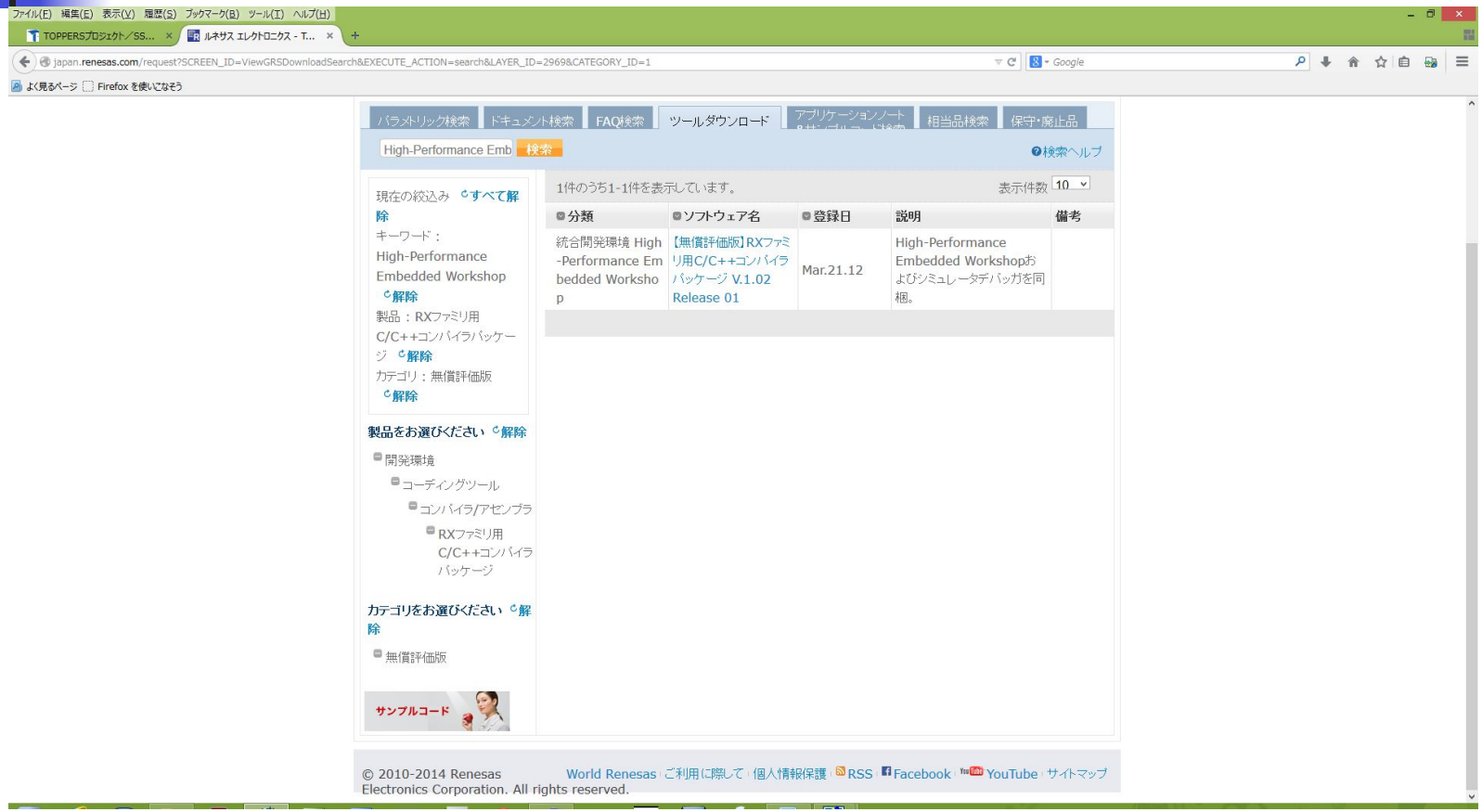
・ 統合開発環境 High-Performance Embedded Workshop

【無償評価版】RXファミリ用C/C++コンパイラパッケージ V.1.02 Release 01

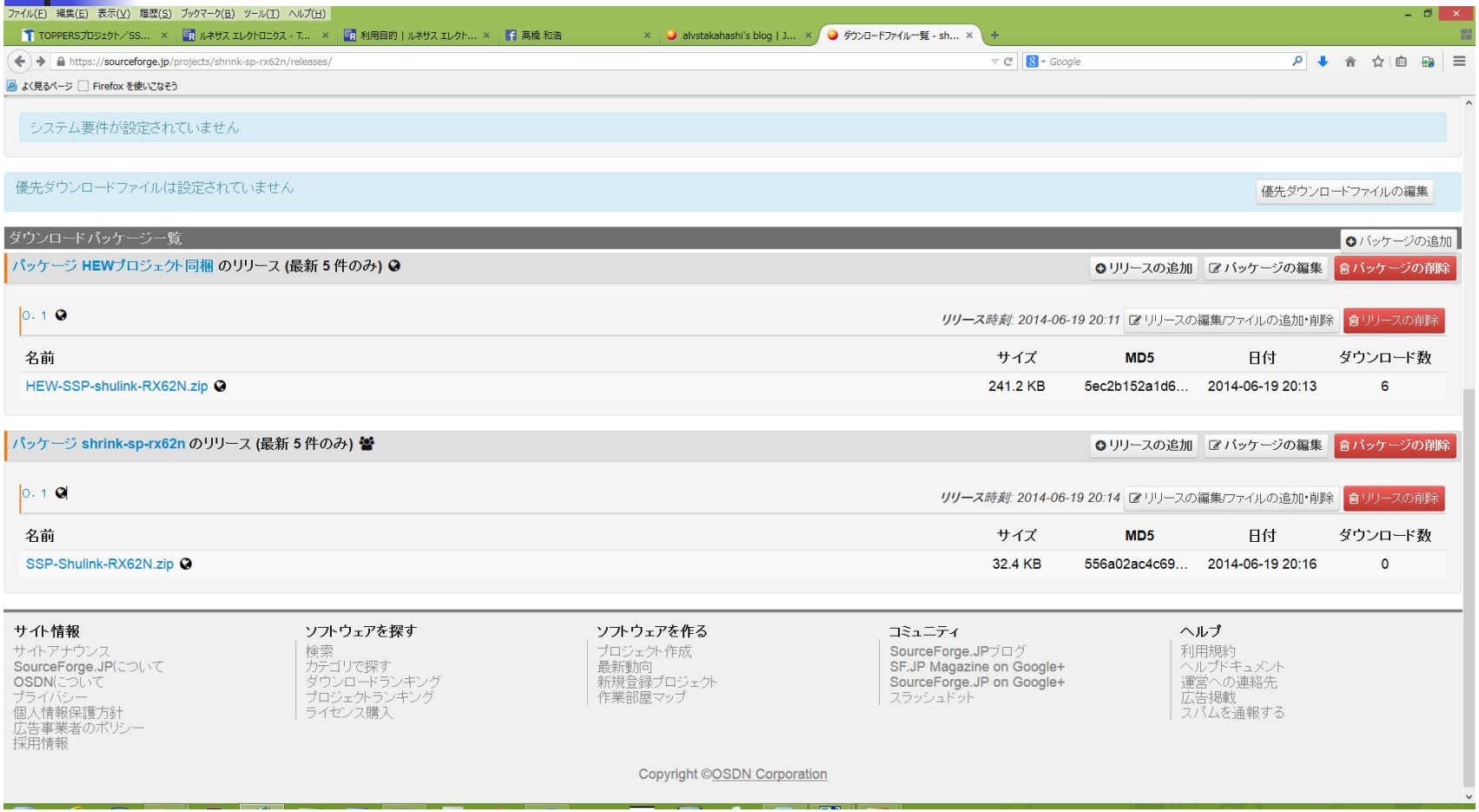
ccrxv102r01_ev.exe

・シュリンク版SSP　shrink-sp-rx62n.zip

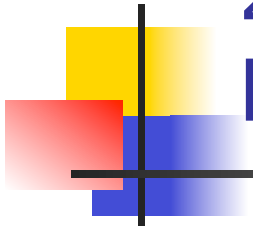
統合開発環境 High-Performance Embedded Workshop



Shrink-SP-RX62N

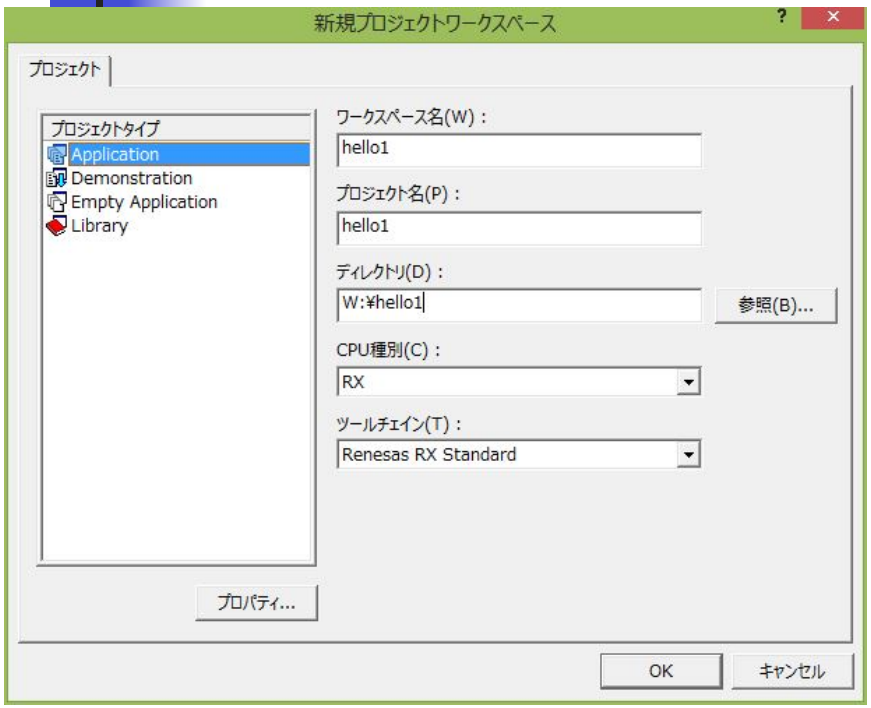


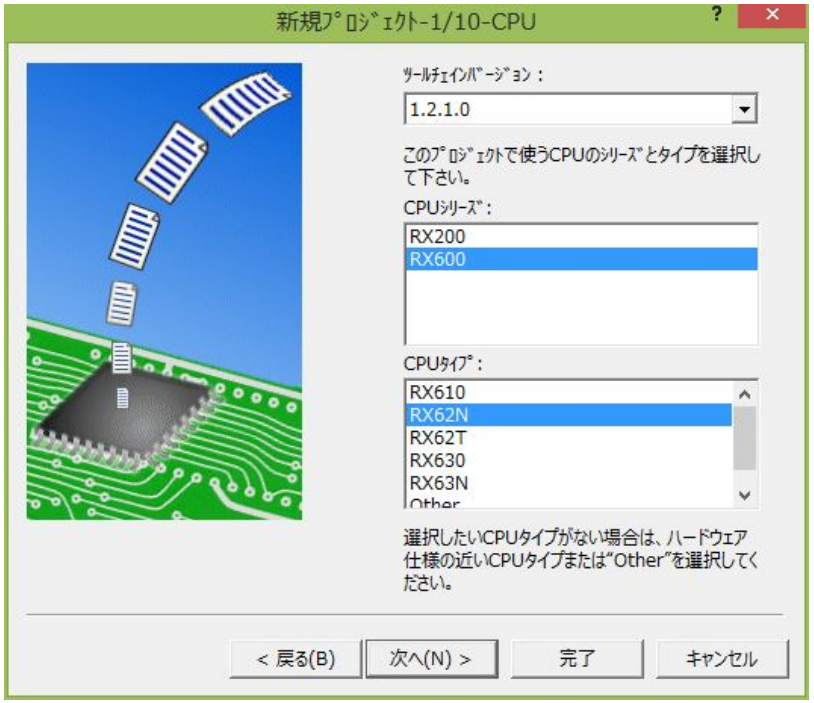
2)RXコンパイラのセットアップとHELLOWORLDの作成



- RXコンパイラ(HEW)をインストーラによってインストール
- HEWのウイザードでmain()まで作ります

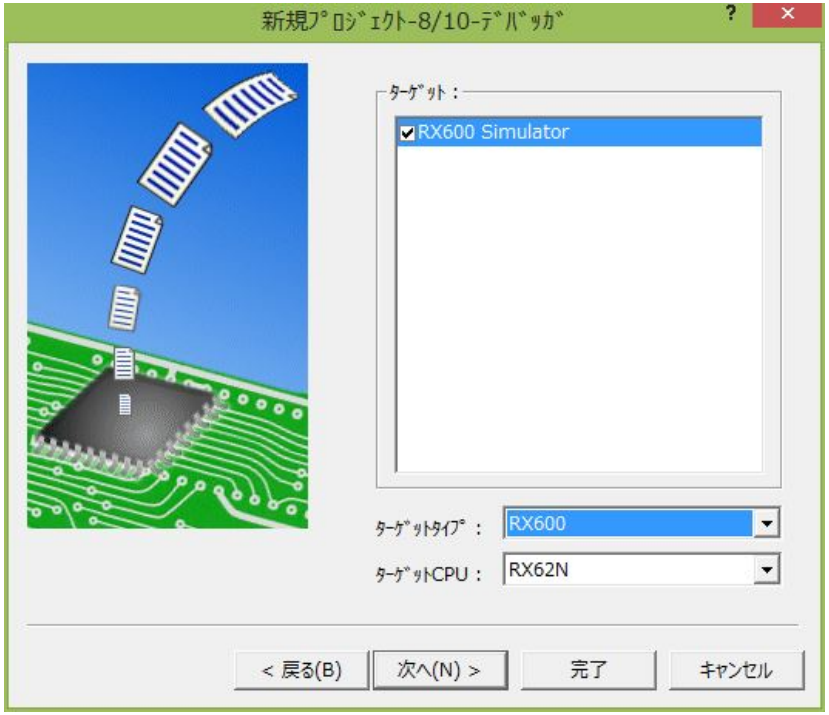
2)RXコンパイラのセットアップと HELLOWORLDの作成

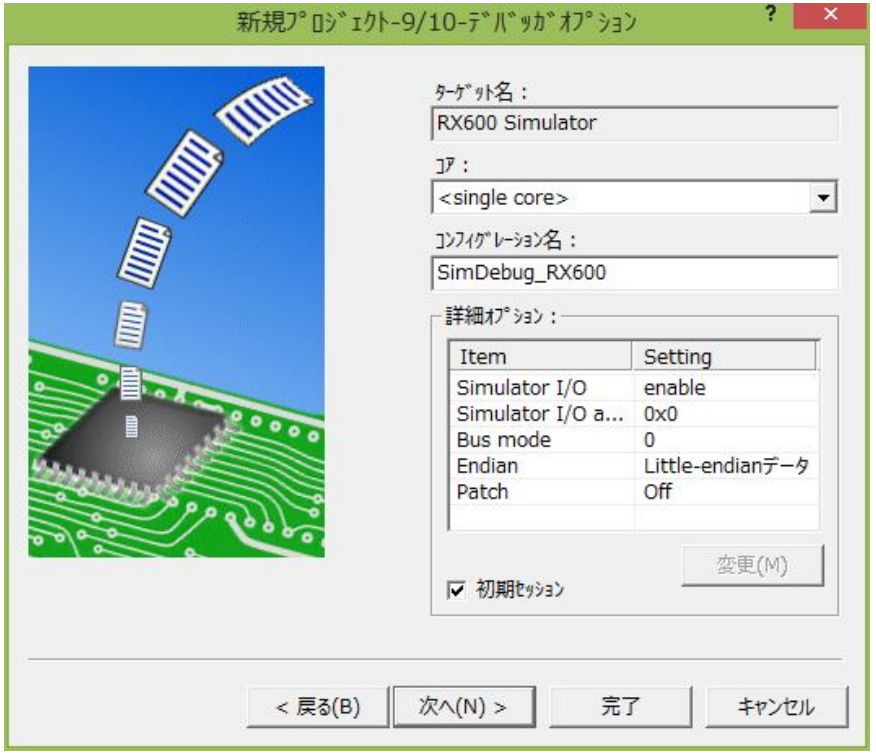




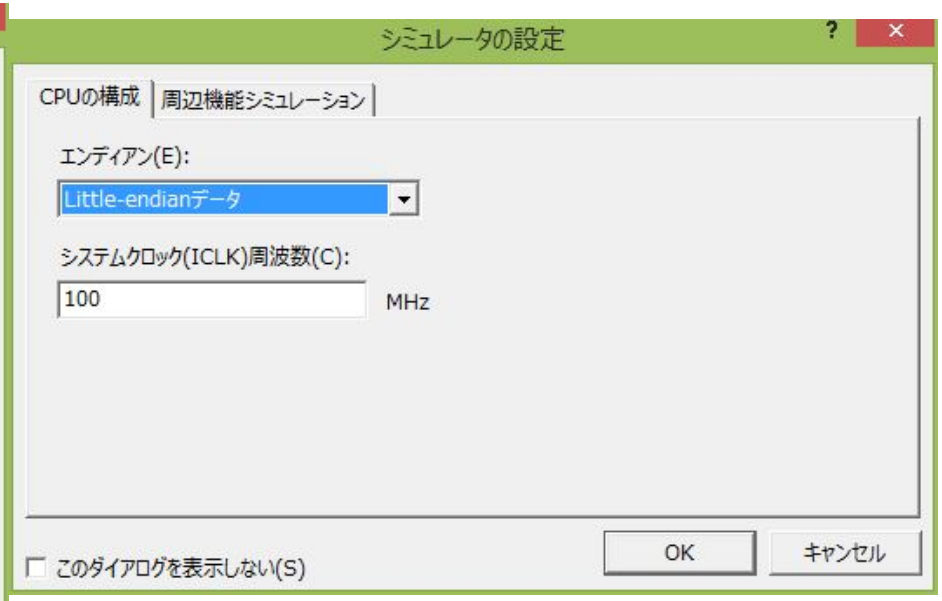
2)RXコンパイラのセットアップと HELLOWORLDの作成

2)RXコンパイラのセットアップと HELLOWORLDの作成





2)RXコンパイラのセットアップと HELLOWORLDの作成

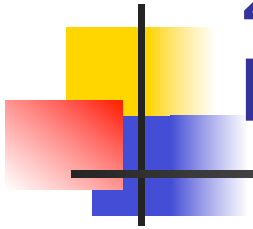


2)RXコンパイラのセットアップと HELLOWORLDの作成

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :hello1.c                                              */
/*  DATE        :Mon, Jul 21, 2014                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :RX62N                                                 */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.4.53).    */
/*  NOTE:THIS IS A TYPICAL EXAMPLE.                                    */
/*                                                                     */
/***********************************************************************/

#include<stdio.h>

//#include "typedefine.h"
#ifdef __cplusplus
//#include <ios>                        // Remove the comment when you use ios
//_SINT ios_base::Init::init_cnt;       // Remove the comment when you use ios
#endif
```

2)RXコンパイラのセットアップとHELLOWORLDの作成

```
void main(void);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif

void main(void)
{
        printf("Hello SWEST\n");
}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```



2)RXコンパイラのセットアップと HELLOWORLDの作成

HEWシミュレータ設定

「表示」「CPU」「デバッグコンソール」

ビルドと実行

「ビルド」「すべてをビルド」

「デバッグ」「ダウンロード」

「デバッグ」「CPUのリセット」

「デバッグ」「実行」

3)シュリンク版SSPの組込み

- (1)解凍とコピー

  解凍すると　SSPフォルダができるのでプロジェクトファイルと同じレベルにSSP/includeフォルダを位置するようにする。

- (2)プロジェクトに追加

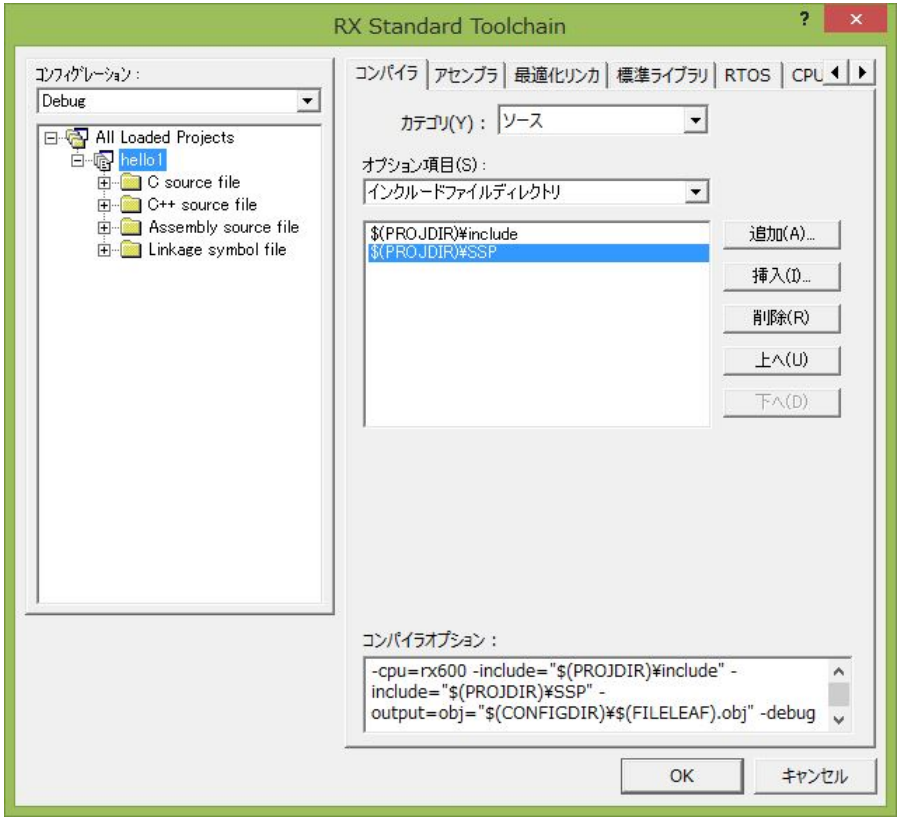
  SSPのCのファイルをプロジェクトに追加。

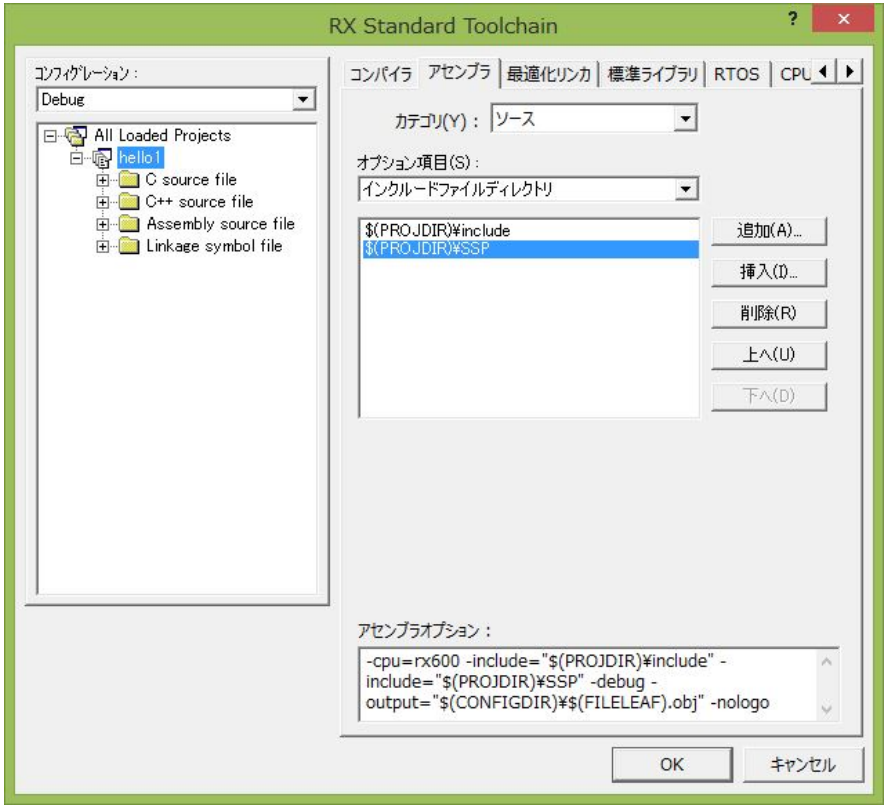
3)シュリンク版SSPの組込み

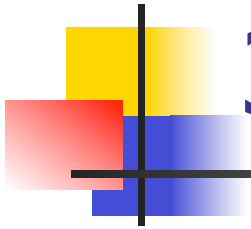
- ヘッダファイルのインクルードディレクトリに設定

「ビルド」「RXstanderd toolchain」

3)シュリンク版SSPの組込み







3)シュリンク版SSPの組込み

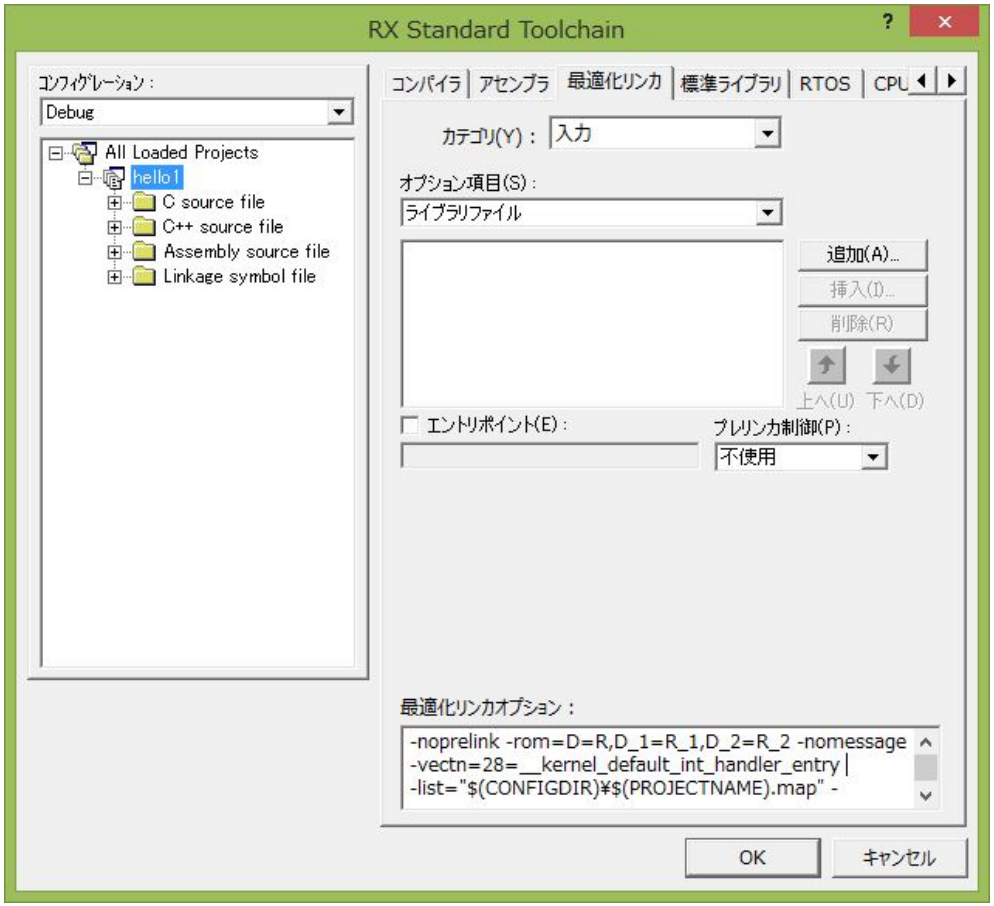
(3)割り込みの設定

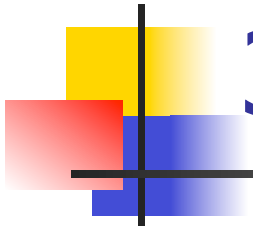
リンカーにタイマー割り込みを指定する。

-vectn=28=__kernel_default_int_handler_entry

「ビルド」「RXstanderd toolchain」「最適化リンカ」

3)シュリンク版SSPの組込み





3)シュリンク版SSPの組込み

- 割り込みベクタ定義を削除する。
- vect.h
  - // CMTU0_CMT0
  - //#pragma interrupt (Excep_CMTU0_CMT0(vect=28))
  - //void Excep_CMTU0_CMT0(void);

3)シュリンク版SSPの組込み

(4)スタートアップの修正

スタートアップからカーネルを起動する。

- resetprg.c

```
        set_psw(PSW_init);              // Set Ubit & Ibit for PSW
//      chg_pmusr();
        sta_ker();

//      main();

        _CLOSEALL();                            // Use SIM I/O
```

3)シュリンク版SSPの組込み

(5)サンプルプログラムを変更する

- #include <kernel.h>
- #include "kernel_cfg.h"

- #include <machine.h> // Include machine.h
- #include "iodefine.h" // Include iodefine.h
-
- #include<stdio.h>

- //#include "typedefine.h"
- #ifdef __cplusplus
- //#include <ios> // Remove the comment when you use ios
- //_SINT ios_base::Init::init_cnt; // Remove the comment when you use ios
- #endif

3)シュリンク版SSPの組込み

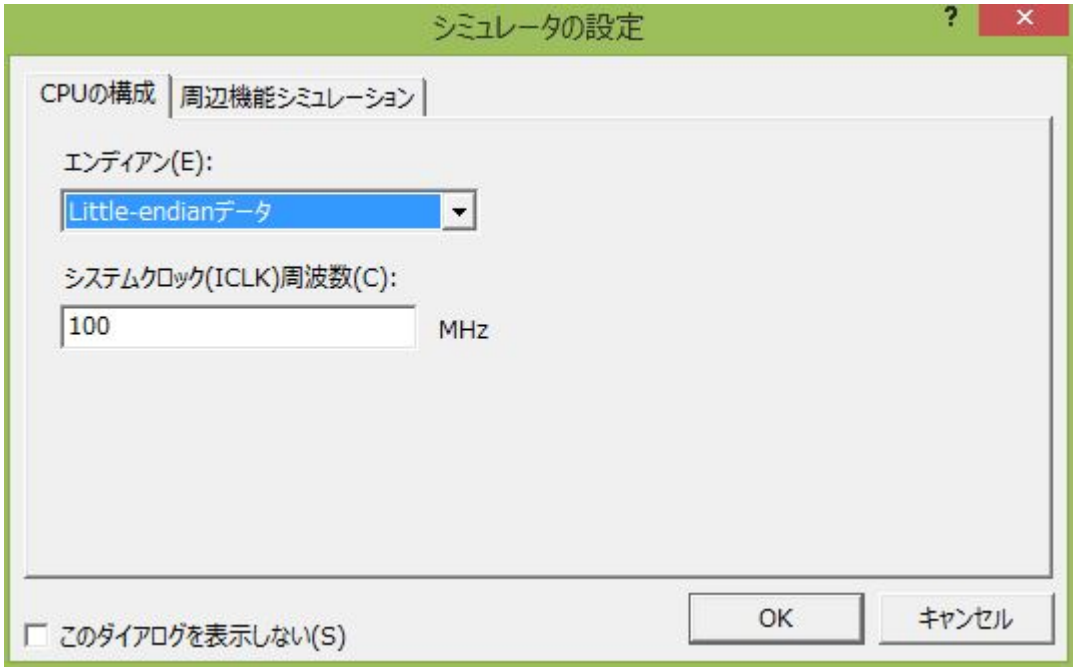
```
void main(intptr_t arg);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif
void main(intptr_t arg)
{

        printf("hello  main\n");
        PORT1.DDR.BIT.B5 = 1;                                   // P15 is Output
        MSTP( CMT0 ) = 0;                               // Wakeup CMT0,CMT1
        CMT0.CMCOR = 4800000/512/2 - 1;                         // CMCOR is 500ms Count
        CMT0.CMCR.WORD = 0x0043;                        // CMIE is Enable,CKS is PCLK/512
        IEN( CMT0, CMI0 ) = 1;                                          // CMI0 Enable
        IPR( CMT0, CMI0 ) = 1;                                          // CMI0
Interrupt Level is 1
//      set_psw( 0x00010000 );                                          // Set I=1,
IPL=0 of PSW
        CMT.CMSTR0.BIT.STR0 = 1;                                // Start CMT0
//      for(;;)
//      ;
}
```

3)シュリンク版SSPの組込み

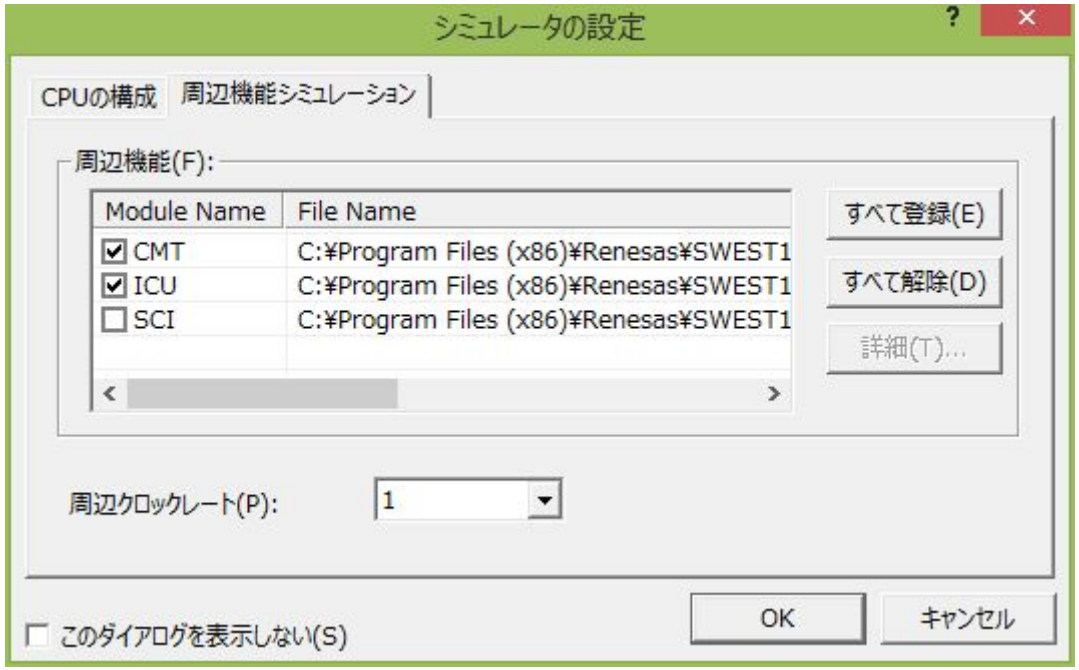
```
void task2(intptr_t arg)
{
        printf("task2\n");
}
void CMI0(void)    // Interrupt Function
{
                                                    // Multiple Interrupt Enable
        PORT1.DR.BIT.B5 ^= 1;                       // Reverse P15(LED)
        iact_tsk(TASK2_ID);
}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```

3)シュリンク版SSPの組込み

- (6)シミュレータの設定

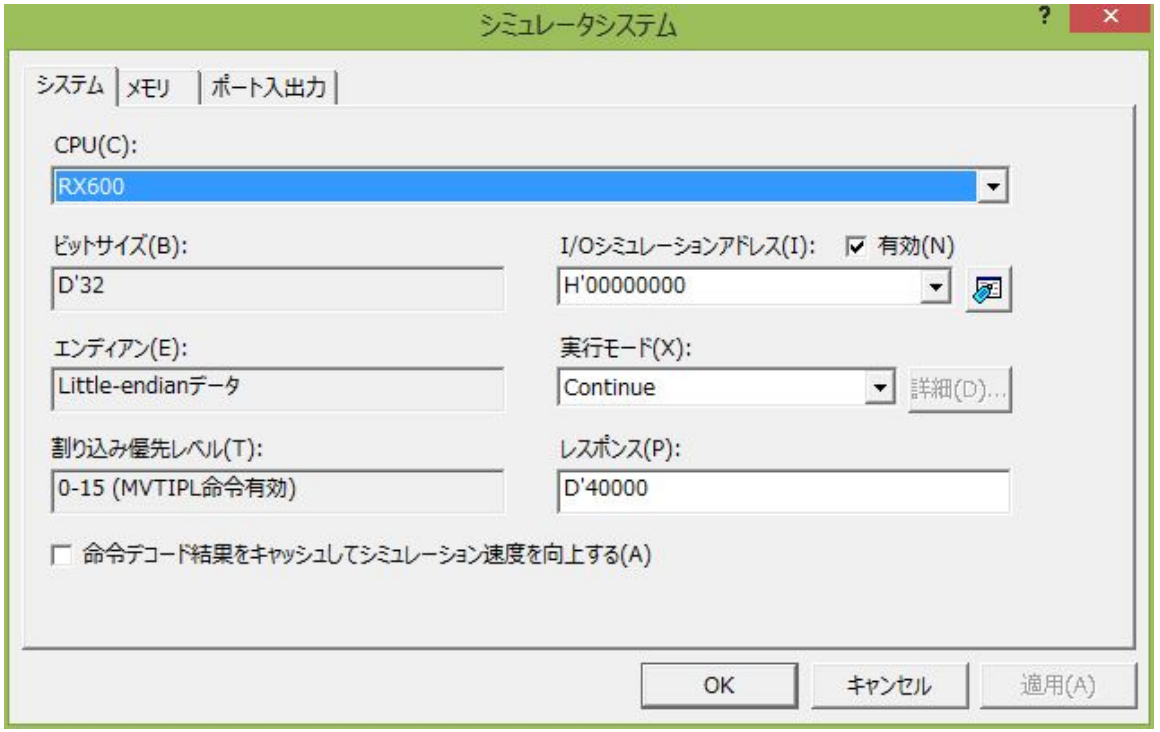


3)シュリンク版SSPの組込み

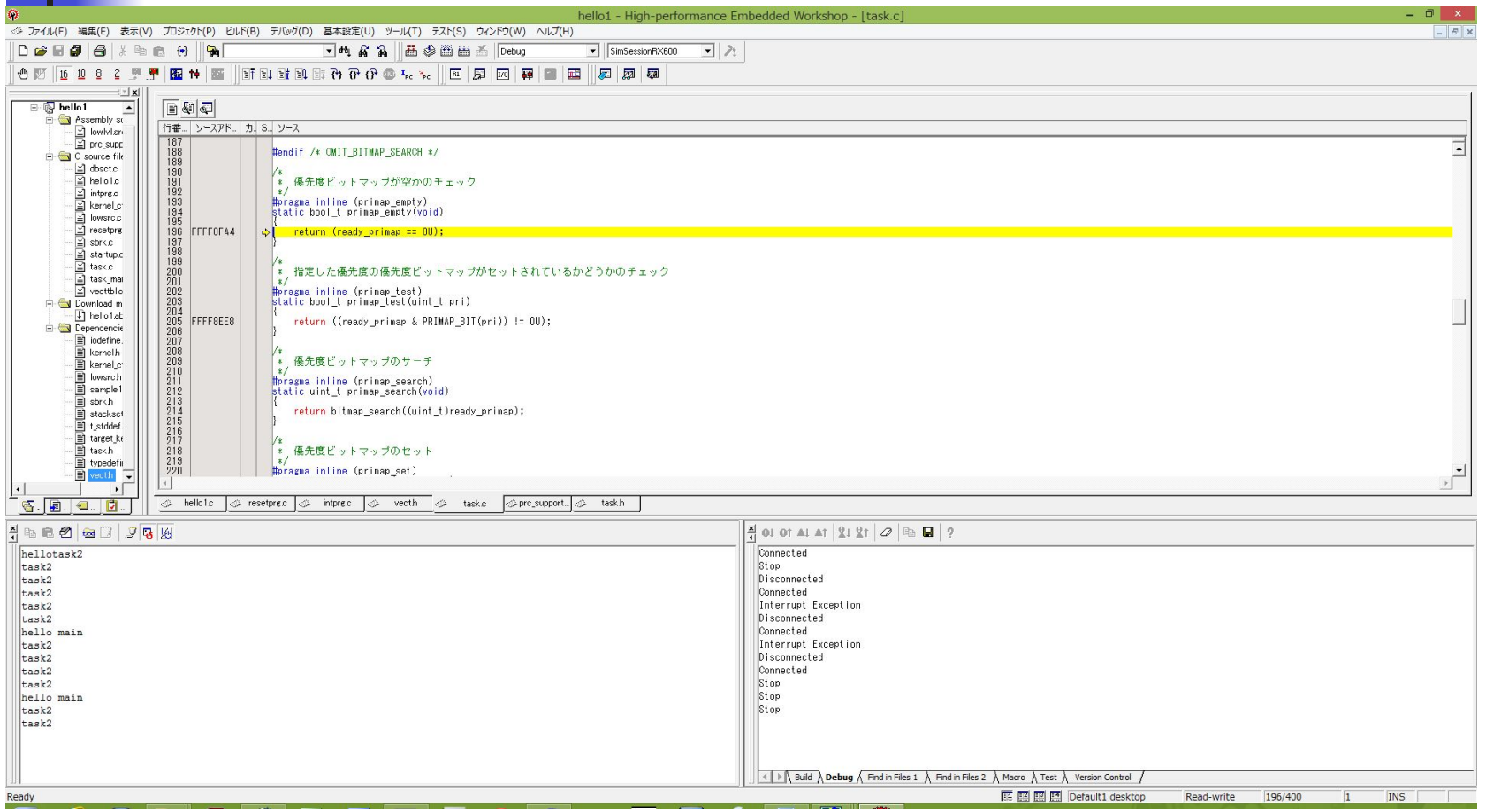


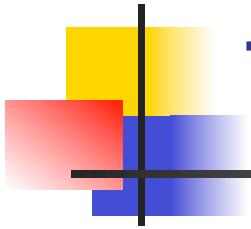
3)シュリンク版SSPの組込み

- 「基本設定」「シミュレータ」「システム」



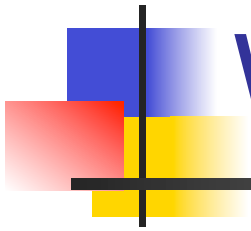
4)実行


hello1 - High-performance Embedded Workshop - [task.c]
hellotask2
task2
hello main
Connected
Stop
Disconnected
Interrupt Exception
Ready



まとめ

- コンフュグレーション　手打ち
- 移植方法
  target_kernel.h
  prc_support.src
  - 割り込みハンドラ関係のみ
  - 多重割り込みの判定をしてスケジューラ
  - ディスパッチャにJUMPするのみ
- TOPPERS/SSPに移植されているもの/いないもの

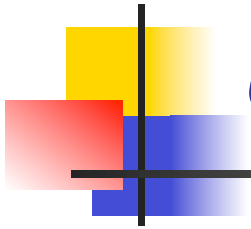


WAIT_SSPの概要とその仕組み

SECTION 3

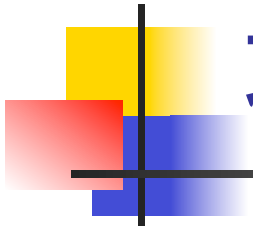
アライブビジョンソフトウェア株式会社

髙橋和浩



このセクションの目的

- WAIT_SSPの紹介とコンテキスト切り替えのアルゴリズムの説明

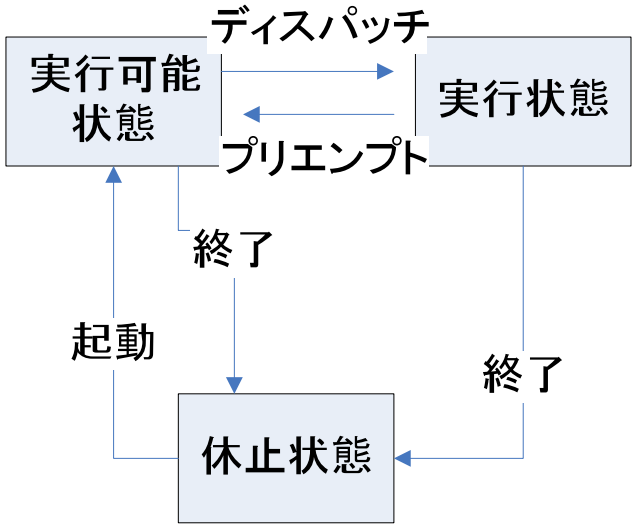


項目一覧

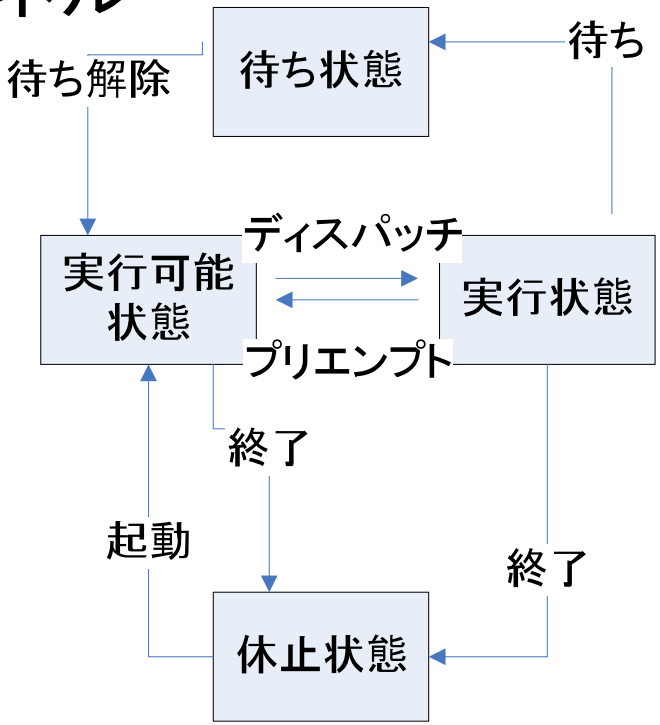
- 1)WAIT_SSPとは
- 2)WAIT_SSPの仕様
- 3)WAIT_SSPの組込み方法
- 4)WAIT_SSPのアルゴリズム
- 5)まとめ　今後の展望

WAIT_SSPとは

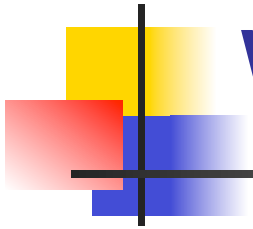
- シュリンク版SSPの改造版
- 待ち状態を持つカーネル



SSP



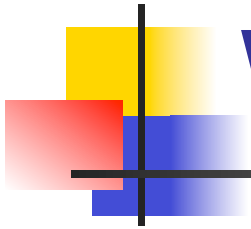
WAIT__SSP



WAIT_SSPとは

- デメリット
  省スタック、省メモリが損なわれる

- メリット
  待ち条件を持つプログラミングスタイルが利用できる。

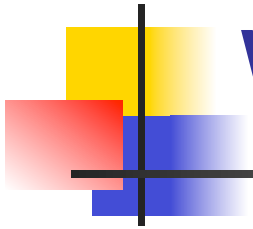


WAIT_SSPの仕様

- wai_tsk() / go_tsk()のサービスコールを追加

  自タスクを待ち状態にする/指定タスクを待ち解除する　のサービスコール

- タスクは優先度8段階の8個登録できる。
- 同一優先度は未対応

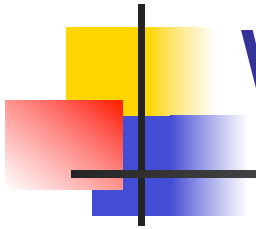


WAIT_SSPの組込み方法

- シュリンク版SSPと同じ。
- https://sourceforge.jp/users/alvstakahashi/pf/WAIT_SSP/files/

- 手打ちのコンフュグレーション

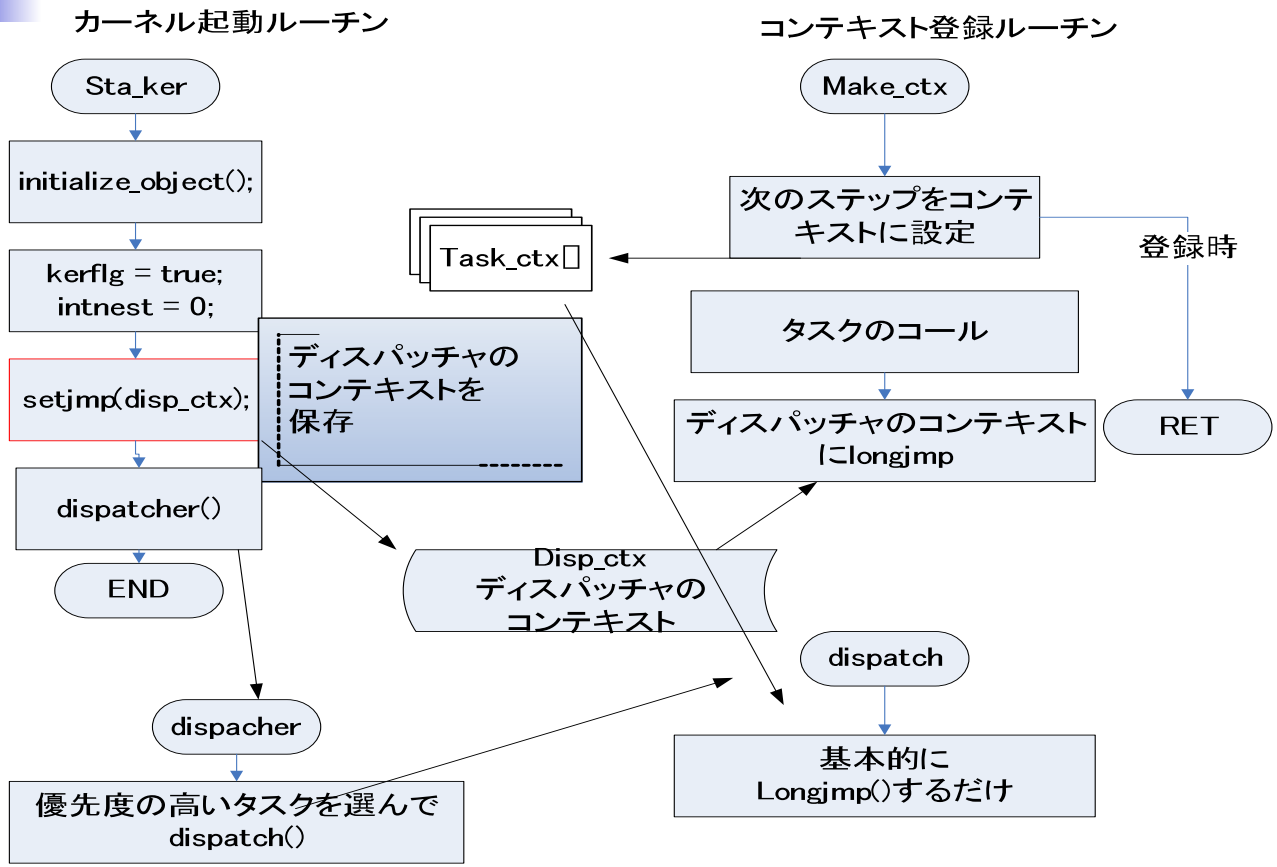
  タスクIDが小さいほど優先度が高いものとして定義

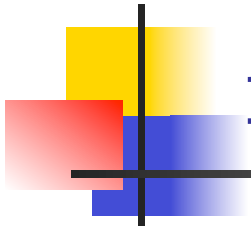


WAIT_SSPのアルゴリズム

- 1)コンテキストの保存/復元を追加
  setjmp/longjmp でコンテキスト切り替え
- 2)割り込み時のレジスタ保存

1)コンテキストの保存/復元を追加

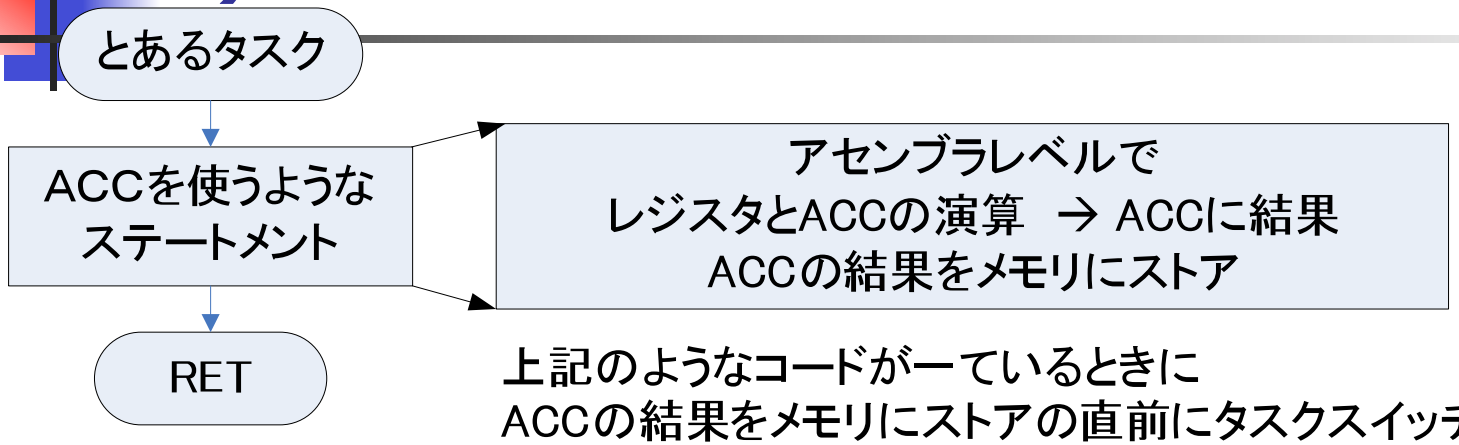




1)コンテキストの保存/復元を追加

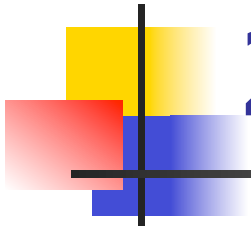
- スケジューラjmp用とタスク別にsetjmp情報を持つ構成
- タスクディスパッチは、タスク別setjmp情報へlongjmp
- タスク終了時は、スケジューラjmp用setjmp情報へlongjmpする

2)割り込み時のレジスタ保存



上記のようなコードが一ているときに
ACCの結果をメモリにストアの直前にタスクスイッチ
した場合に、当然ACCがタスクスイッチする前の
値でないと正しく動作しません

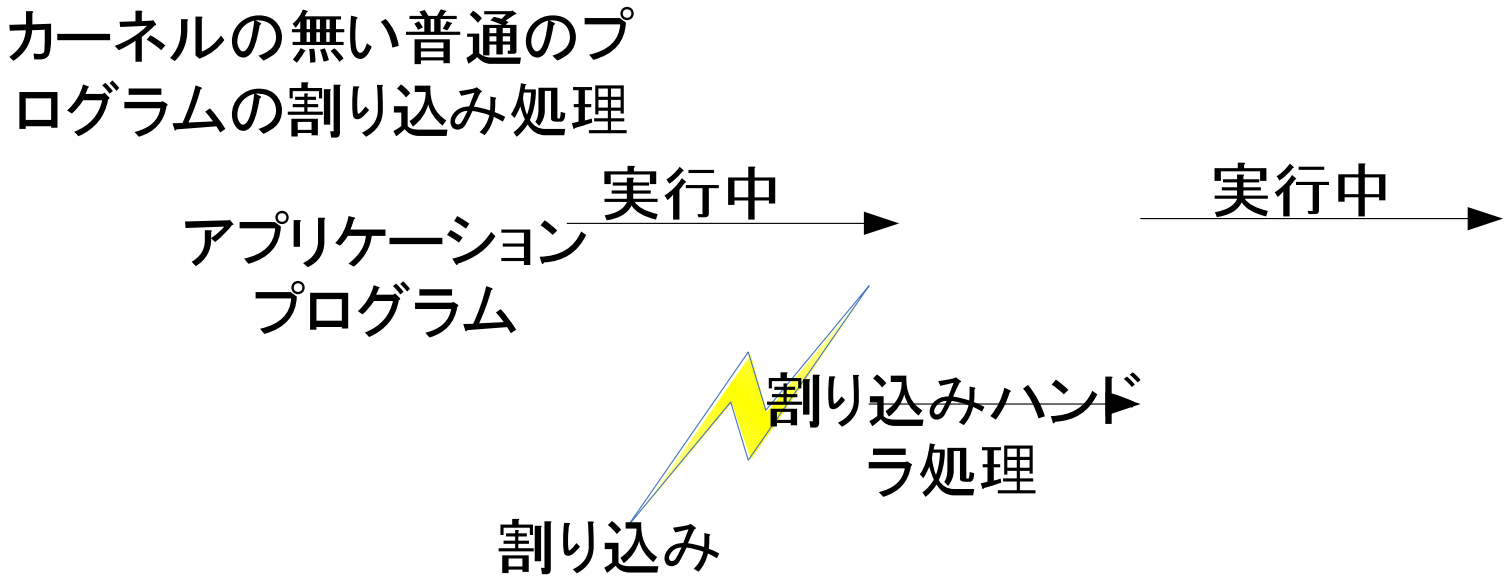
- setjmp/longjmpだけでは、保存不十分
- 割り込みハンドラの入り口と出口で、ACCなどすべてのレジスタを保存/復元する。
- タスクスイッチがなければ、そのまま割りこまれたRUN中のタスクにリターンする。



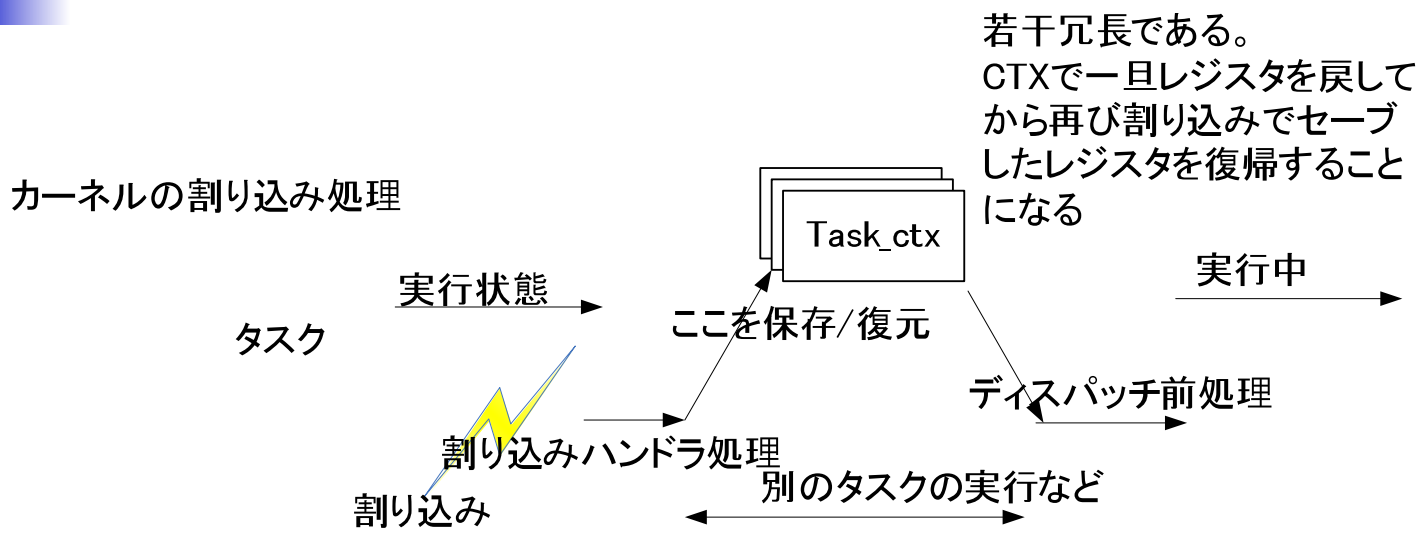
2)割り込み時のレジスタ保存

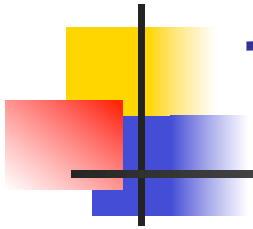
- 実現方法
  RUN中のタスクが、割りこまれてプリエンプトされる場合のみタスク別のsetjmp情報を割り込みハンドラの出口に設定する。つまり、タスクが切り替えられる前のハンドラまでlongjmp()で戻る処理にする。

2)割り込み時のレジスタ保存



2)割り込み時のレジスタ保存

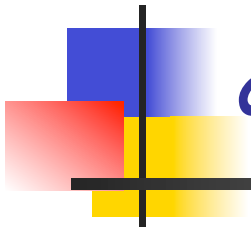




まとめ　展望

- ポリテクセンター兵庫でのセミナー(RTOSセミナ）
- IoT関連への連携　(Ardiuno)
- ニーズ？

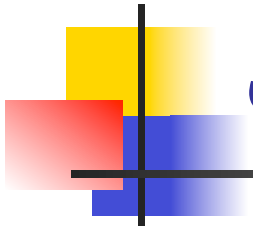
- カーネルとしての完成度
  複数のアーキテクチャ、



シュリンク版SSPの開発過程にみるSSPのソース

SECTION 4

アライブビジョンソフトウェア株式会社
髙橋和浩



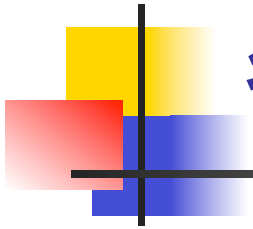
このセクションの目的

- シュリンク版のSSPを作成する過程において、SSPのソースの構成およびソースそのものを理解する必要があり、その中で取捨選択を繰り返した。

  その目線で、どのような取捨選択をしたかをここで説明します

ファイルモジュール表　1

ファイル	場所	モジュール	呼び出し	説明
start.src	arch	_start	_sta_ker	
			_software_init_hook	
			_hardware_init_hook	
			__kernel_istkpt	kernel_cfg.cにて"_kernel_istkpt"の値を決定する.
target_support.src	target	_hardware_init_hook		
		_software_init_hook		
startup.c	kernel	sta_ker	target_initialize();	削除
			initialize_object();	kernel_cfg.c
			call_inirtn();	kernel_cfg.c　削除
			start_dispatch();	
		ext_ker		未実装
		exit_kernel		未実装
target_config.c	target	target_initialize	prc_initialize();	ターゲット固有のSIOドライバ
		target_exit	rx600_uart_init その後SIOの初期化	
		target_fput_log		

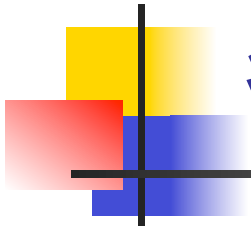


各モジュールの説明　1

- start.src--- 基本的に開発環境のもの（HEWウイザード）のものを利用
- target_support.src--- ターゲット固有のドライバは持たない
- startup.c ----sta_ker()を実装。ターゲット初期化および割り込みベクター初期化は削除
- target_config.c ----ターゲット固有のドライバは持たない

ファイルモジュール表　2

ファイル	場所	モジュール	呼び出し	説明
prc_config.c	arch	prc_initialize		intnest = 1U; のみ
		prc_terminate		割り込みベクター
		xlog_sys		
		x_config_int		
		default_int_handler		
		default_exc_handler		
rx600_usrt.c	target			SIOドライバ
target_serial.c	target			SIOドライバ
target_timer.c	target			タイマードライバ
banner.c	syssvc			バナー
serial.c	syssvc			SIOドライバ
banner.tf	syssvc			ここでバナーの固定テーブルのテンプレート があるので削除した
kerbel_cfg.c		_kernel_initialize_object	_kernel_initialize_task();	
			_kernel_initialize_interrupt();	割り込みベクタの初期化　削除
			_kernel_initialize_exception();	例外ベクタの初期化　削除

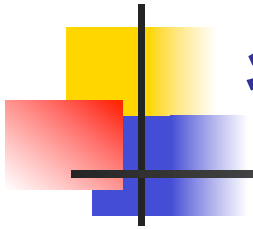


各モジュールの説明　2

- prc_config.c ----ターゲット固有のドライバは持たない
- rx600_usrt.c ----ターゲット固有のドライバは持たない
- target_serial.c ----ターゲット固有のドライバは持たない
- target_timer.c　----ターゲット固有のドライバは持たない
- banner.c ---- バナーはとりあえず無
- serial.c　----ターゲット固有のドライバは持たない
- banner.tf　---- バナーはとりあえず無
- kernel_cfg.c
- 　kernel_initialize_task ---- 残す
- 　kernel_initialize_interrupt()　----割り込みベクタの初期化　削除
- 　kernel_initialize_exception()----例外ベクタの初期化　削除

ファイルモジュール表　3

ファイル	場所	モジュール	呼び出し	説明
task.c	kernel	initialize_task		タスク情報の初期化
		get_ipri_self		
		get_ipri		
		bitmap_search		
		primap_empty		
		primap_serach		
		primap_test		
		primap_set		
		primap_clear		
		swerach_schedtsk		
		test_dormanr		
		make_active		
		run_tsk		
		dispatcher		



各モジュールの説明　3

- task.c --- タスク処理　残す　そのまま

ファイルモジュール表　4

ファイル	場所	モジュール	呼び出し	説明
prc_support.src	arch	_kernel_start_dispatch		起動時のディスパッチャ
		_kernel_call_exit_kernel		カーネル出口処理　未実装
		ret_int		割り込みからのディスパッチャ入口
		ret_int_r_rte:		割り込みで割り込みがひとつネスト を戻して戻る
		_kernel_interrupt:		割り込みハンドラ
		_kernel_exception		CPU例外 削除
interrupt.c	kernel	initialize_interrupt		割り込み情報のテーブル初期化を している この辺は手動にするので、削 除する方向
		dis_int		ユーザーAPI
		ena_int		ユーザーAPI
exception.c	kernel	initialize_exception		CPU例外のベクタ等の設定

各モジュールの説明　4

- prc_support.src
- 　kernel_start_dispatch　----------残す
- 　kernel_call_exit_kernel----------カーネル出口処理　未実装
- 　ret_int　----------残す
- 　ret_int_r_rte:　----------残す
- 　kernel_interrupt:　----------残す
- 　kernel_exception　----------例外処理　削除
- interrupt.c　----------割り込み機能　未実装
- exception.c　----------例外処理　削除