RX62N シリーズ

RI600 と FreeRTOS™ 用のユーザーマニュアルおよび API 仕様

R01AN0766JS0100
Rev.1.00
2012.03.05

要旨

本アプリケーションノートでは、アダプタレイヤーについて詳細に説明します。これには、アダプタレイヤーの特性と機能が含まれます。

動作確認デバイス

- RX62N グループ MCU（製品番号: R5FF562N8BDBG）

動作確認ボード

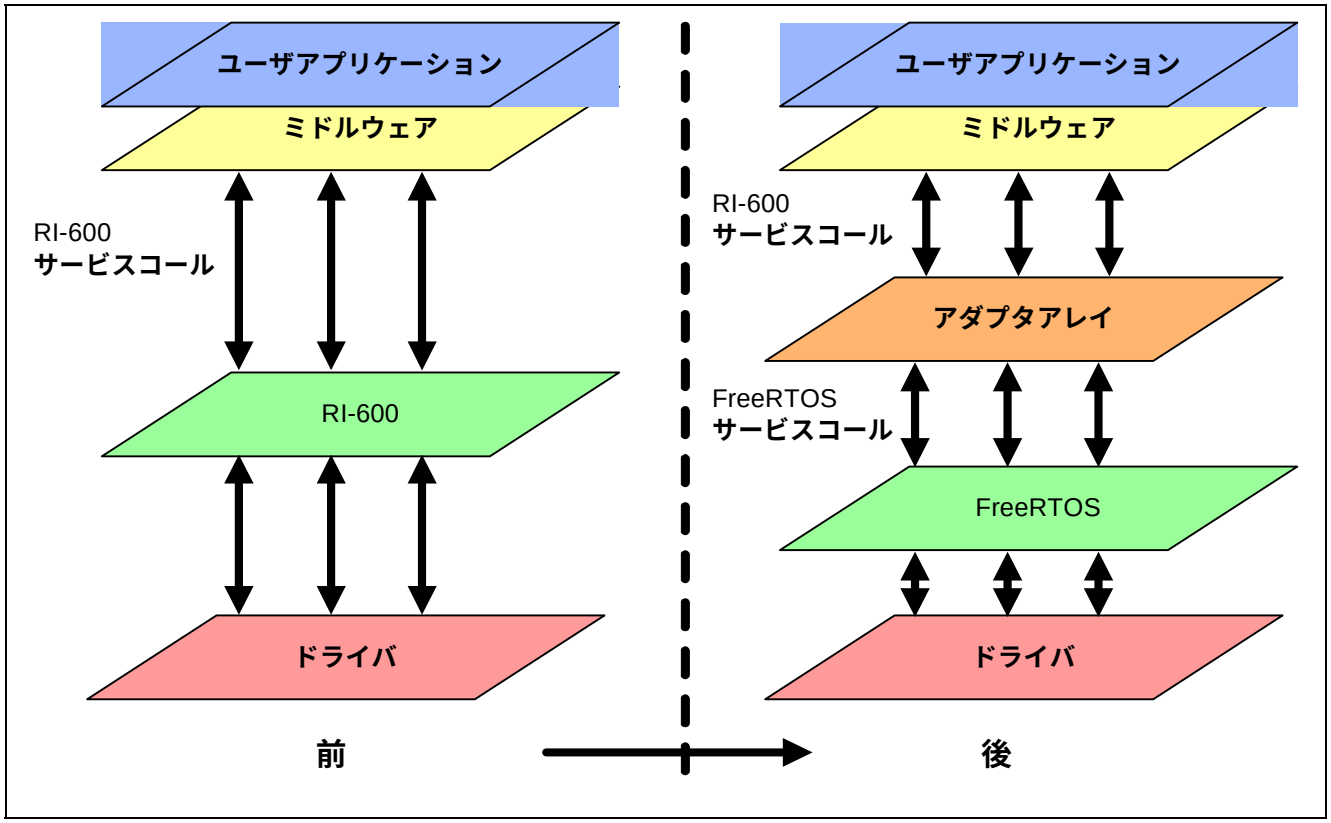
- RX62N ルネサススタータキット +（製品番号: R0K5562N0C000BE)

【注】 FreeRTOS™は Real Time Engineers Ltd の商標です。

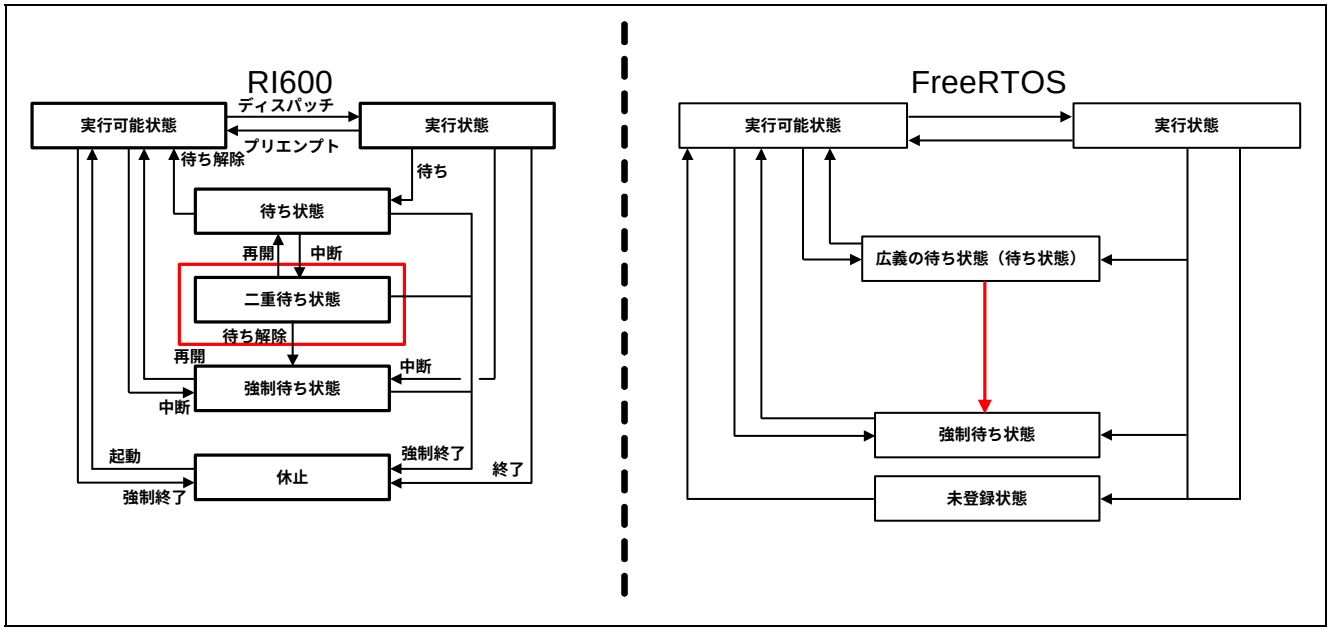
目次

1. アダプタレイヤーの概要

アダプタレイヤーは、RI-600 RTOS 上での動作を目的としたアプリケーションを、再プログラミングの必要なしに、すばやく移植して FreeRTOS™で動作できるように設計されたものです。図で示すと次のようになります。

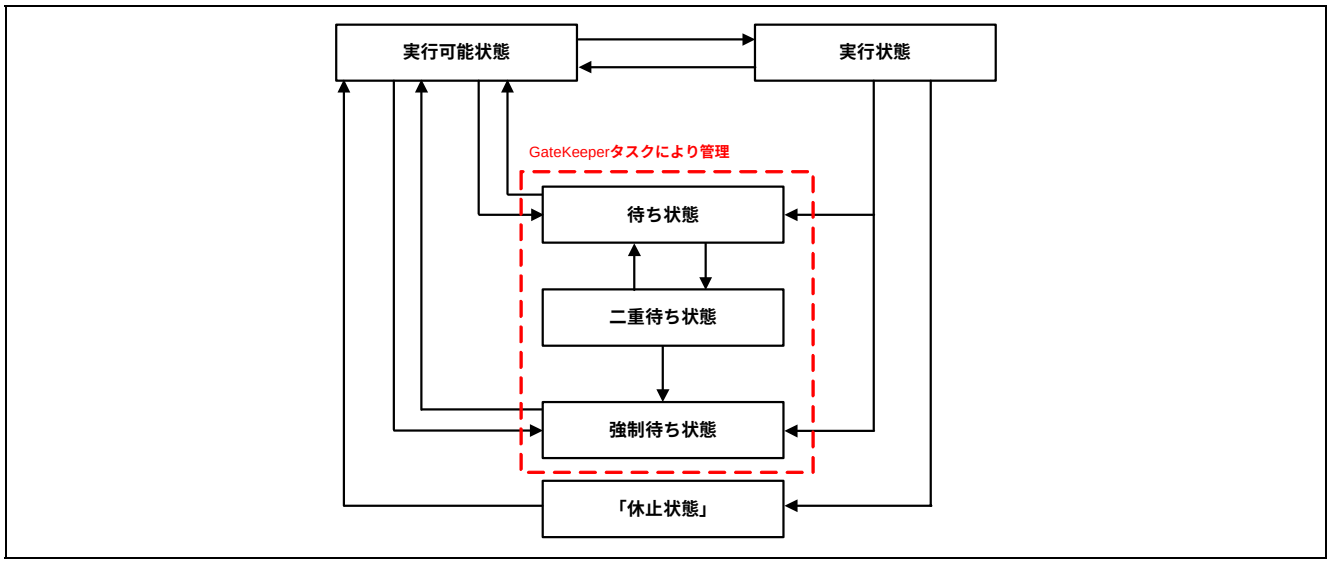


RI-600 と FreeRTOS™タスク状態を並べると、次のようになります。

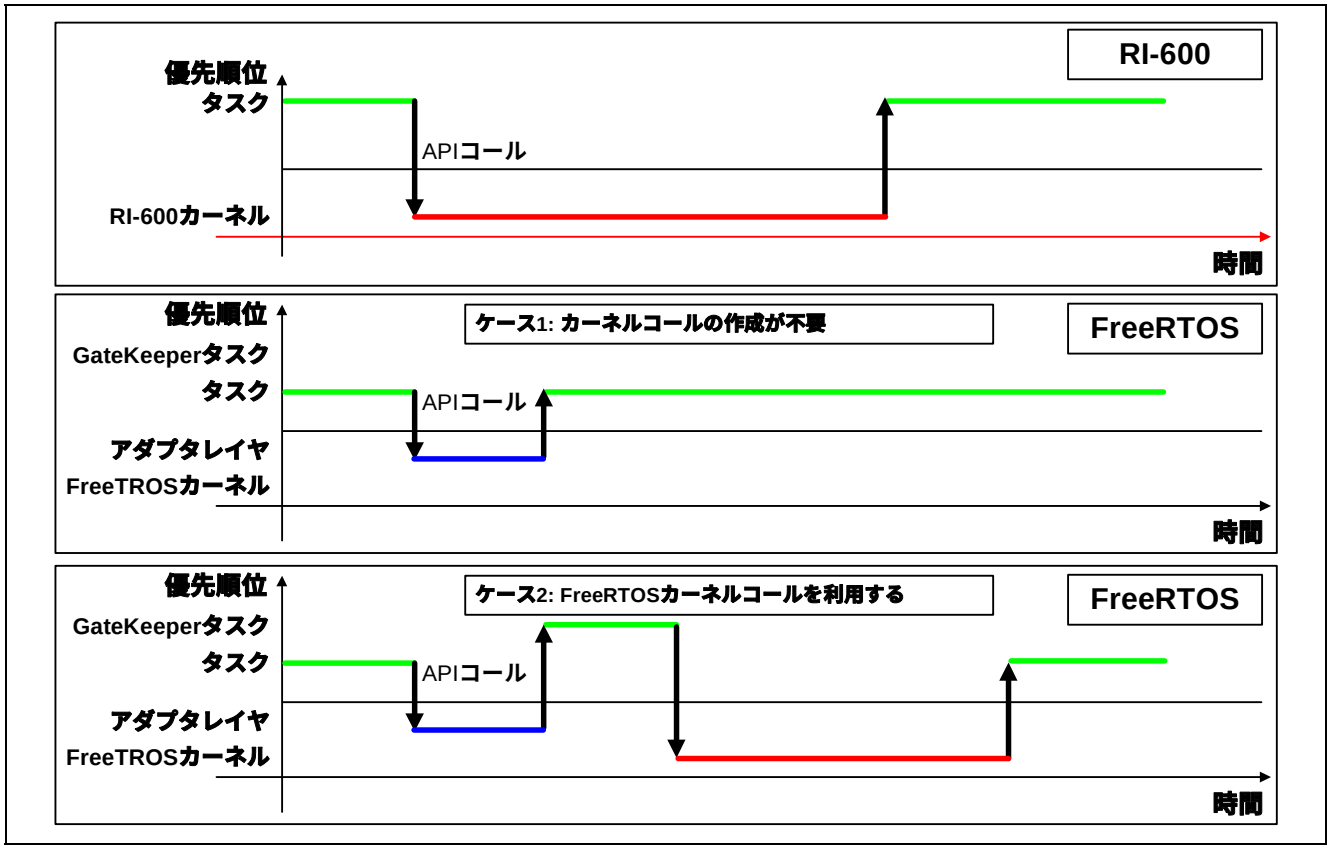


FreeRTOS™では、タスクが広義の待ち状態（待ち状態）に置かれ、次に強制待ち状態になるとき、このタスクの再開によってタスクは広義の待ち状態（待ち状態）に戻るべきですが、そうならないことに注意してください。その代わり、前の広義の待ち状態（待ち状態）から早く解放されることになります。これは時には、望ましくない動きと考えられます。

RI-600 プラットフォームを対象としたアプリケーションは、このような動作の変化を想定していません。このためアダプタレイヤーは、FreeRTOS™が提供しない二重待ち状態を補う必要がありました。この目的のため、アダプタレイヤーは GateKeeper タスクを利用して二重待ちメカニズムを管理する必要があります。従って、コンパイルしたアダプタレイヤータスク状態は、次のようになります。



タスク状態の変更を引き起こすいずれの API も、そのリクエストを処理するため GateKeeper キューにリクエストをラッチします。ただし、タスク状態の変更を要求しない API については、アダプタレイヤーが起動を管理します。このことは、次の図で明らかです。

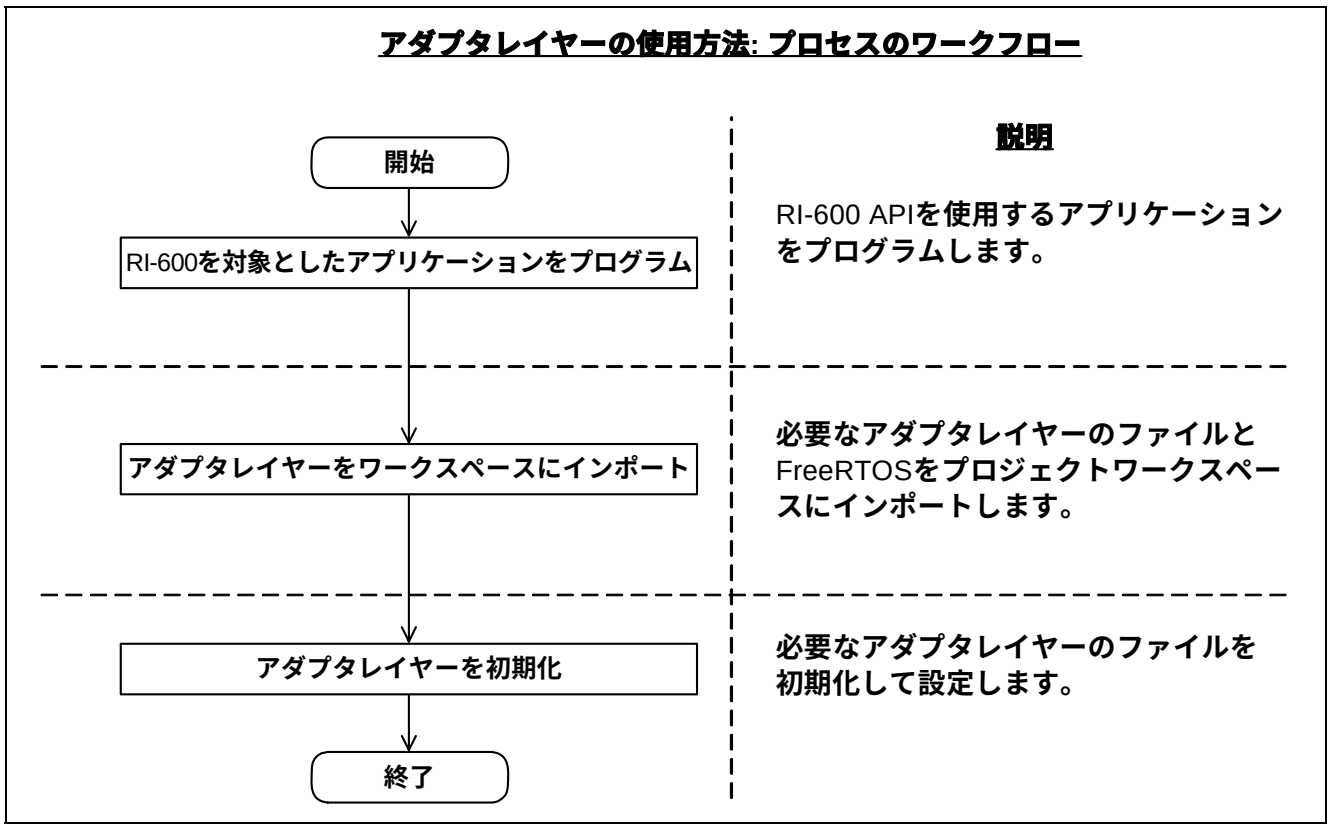

RI-600
優先順位
タスク
APIコール
RI-600カーネル
時間
FreeRTOS
ケース1: カーネルコールの作成が不要
優先順位
GateKeeperタスク
タスク
APIコール
アダプタレイヤ
FreeTROSカーネル
時間
FreeRTOS
ケース2: FreeRTOSカーネルコールを利用する
優先順位
GateKeeperタスク
タスク
APIコール
アダプタレイヤ
FreeTROSカーネル
時間

2. アダプタレイヤープロセスのワークフロー

この節では、アダプタレイヤーを使用できるように正しく実装するプロセスのワークフローについて説明します。

2.1 概要

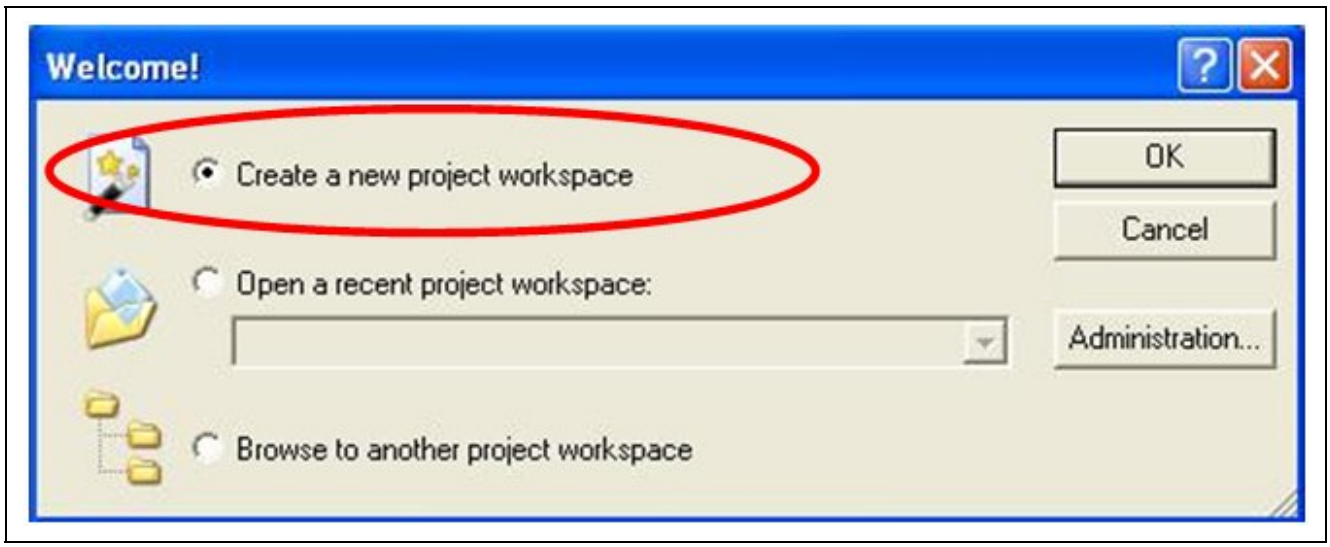
アダプタレイヤー実装プロセスの概要は、次のとおりです。



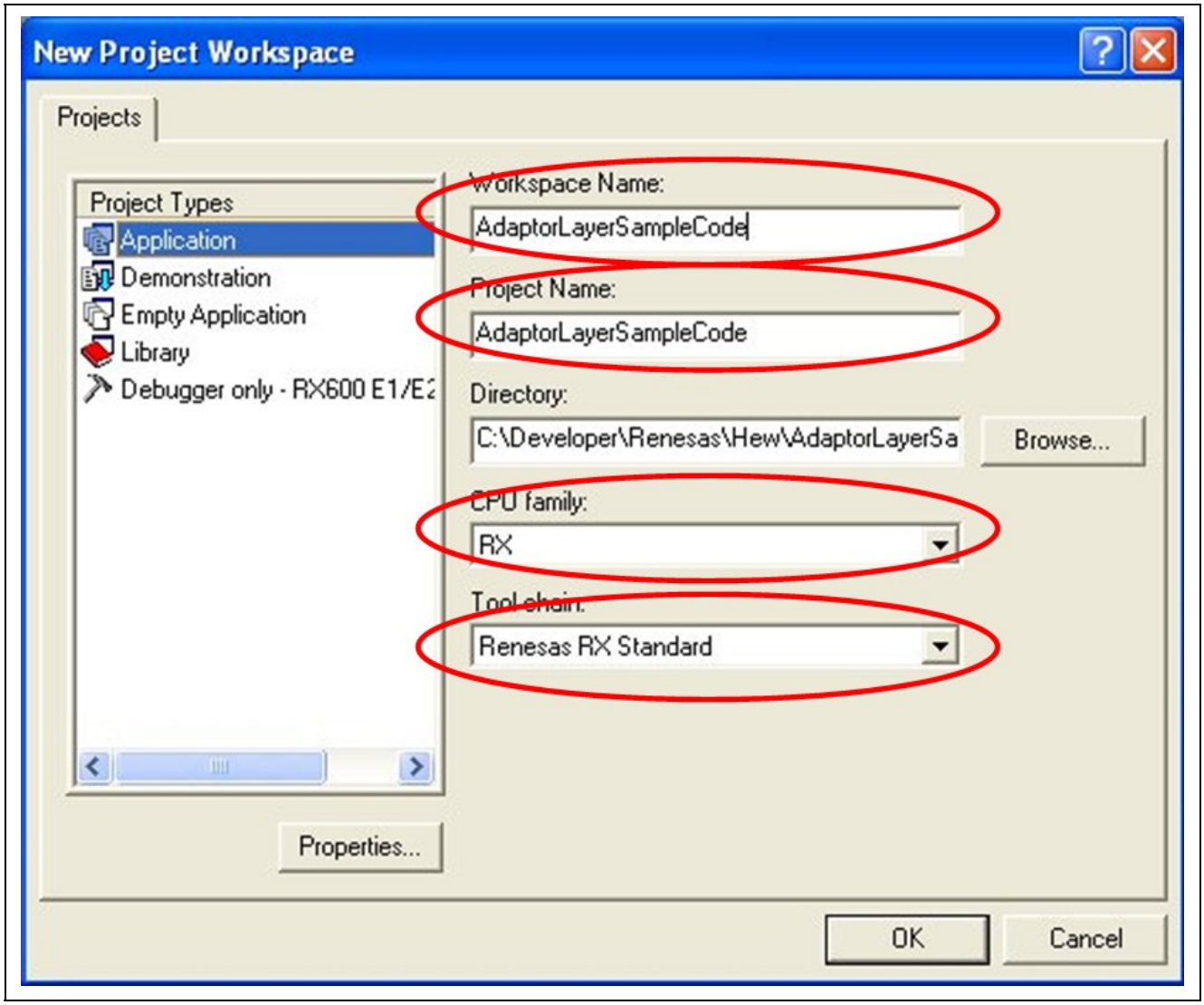
2.2 ルネサス HEW IDE を使用したガイド付きサンプル

2.2.1 新しい RI-600 プロジェクトワークスペースの作成

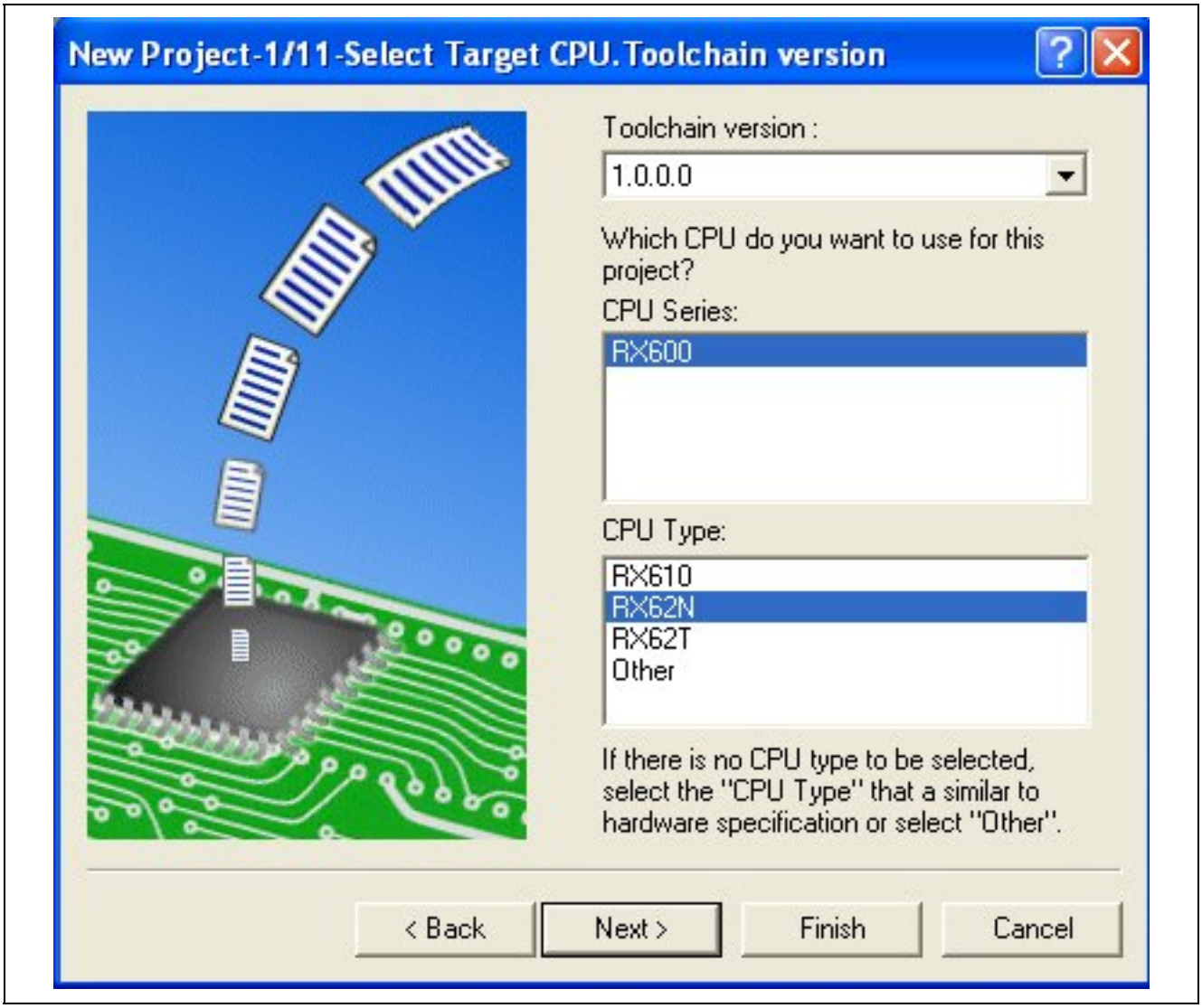
ルネサス HEW IDE を起動し、以下のように新しいプロジェクトワークスペースを作成します。



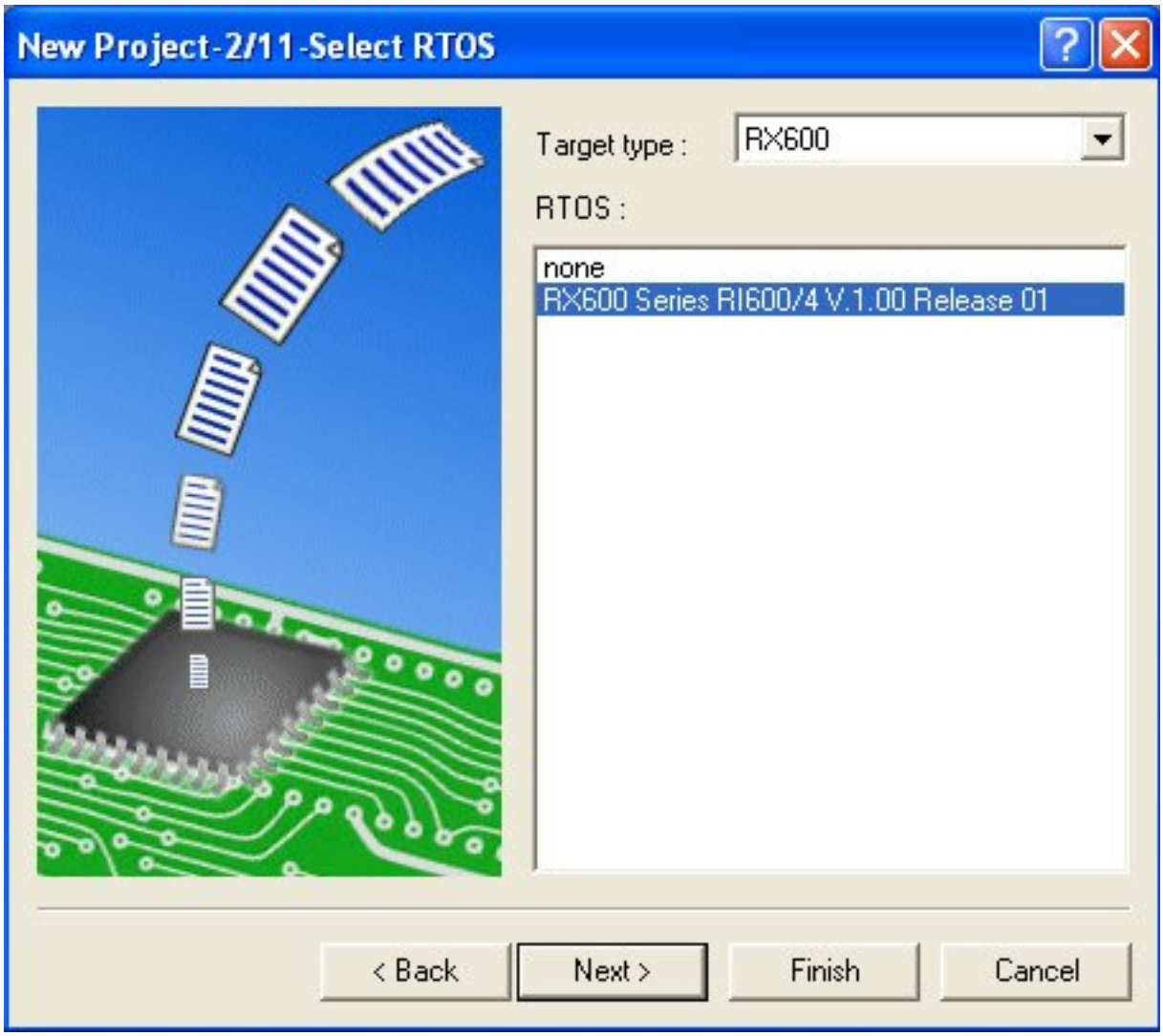
以下のように、有効なワークスペース名とプロジェクト名を入力します。



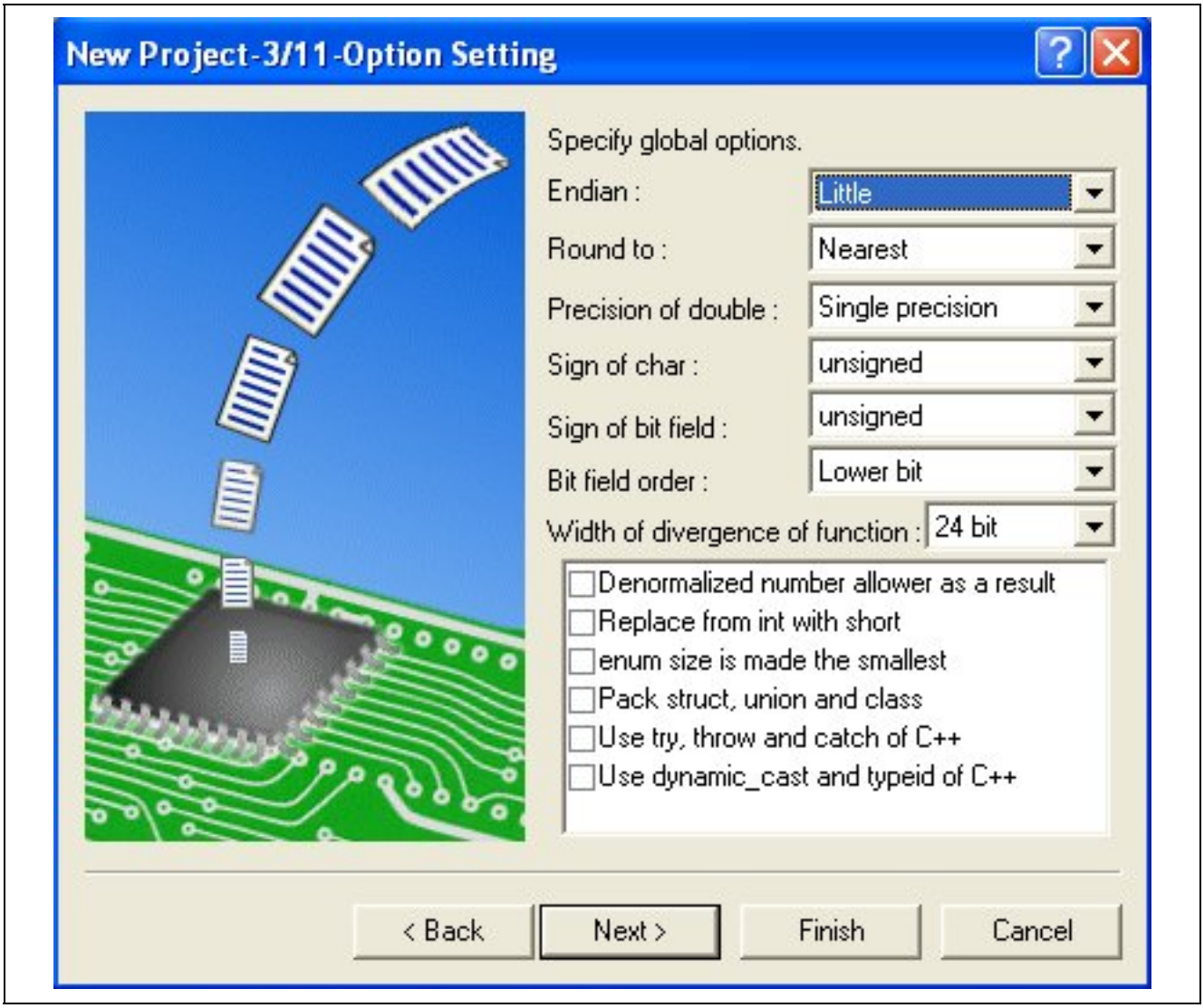
以下のように、適切なツールチェーンと CPU タイプを選択します。



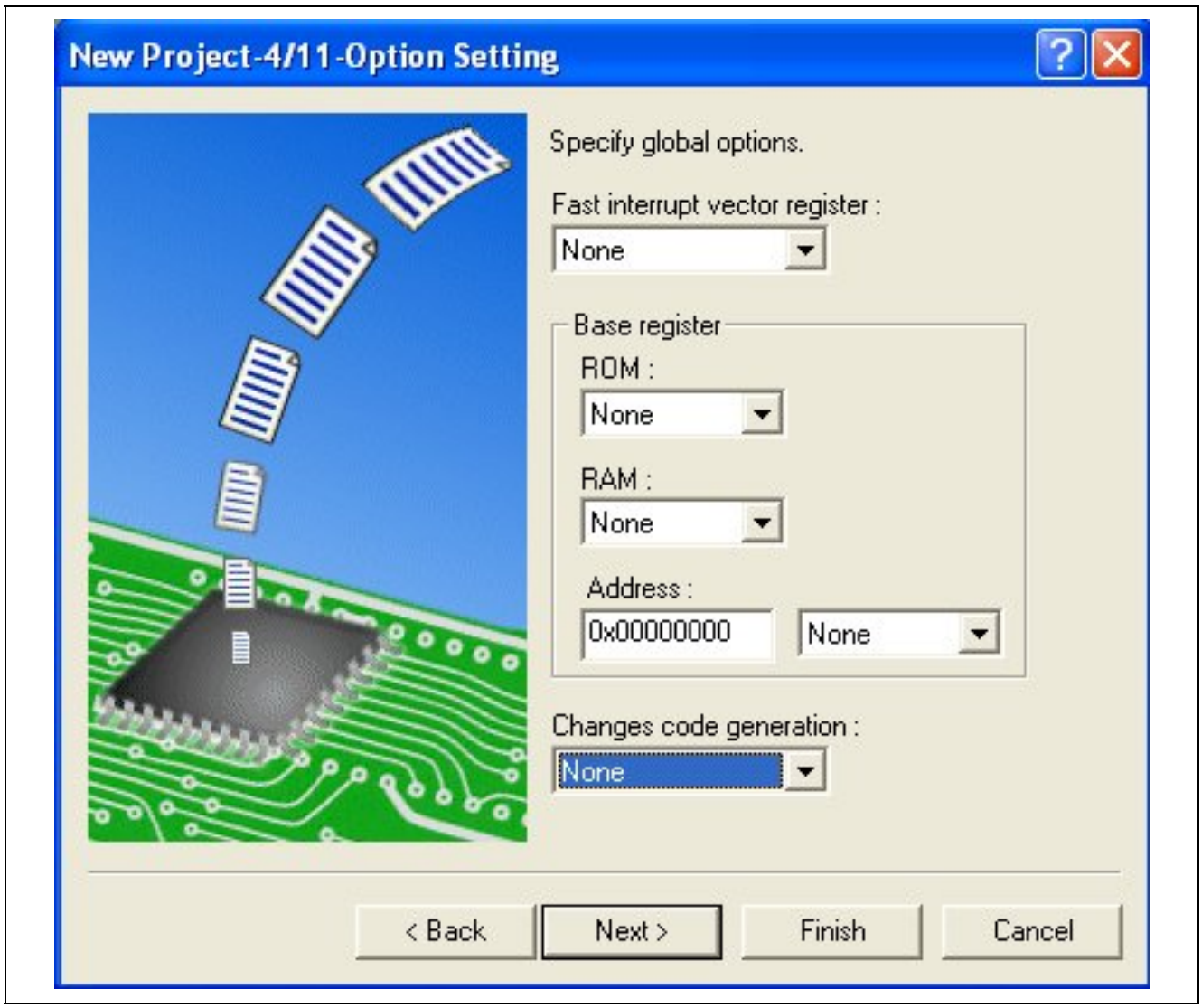
使用する RTOS として RI-600 を選択します。



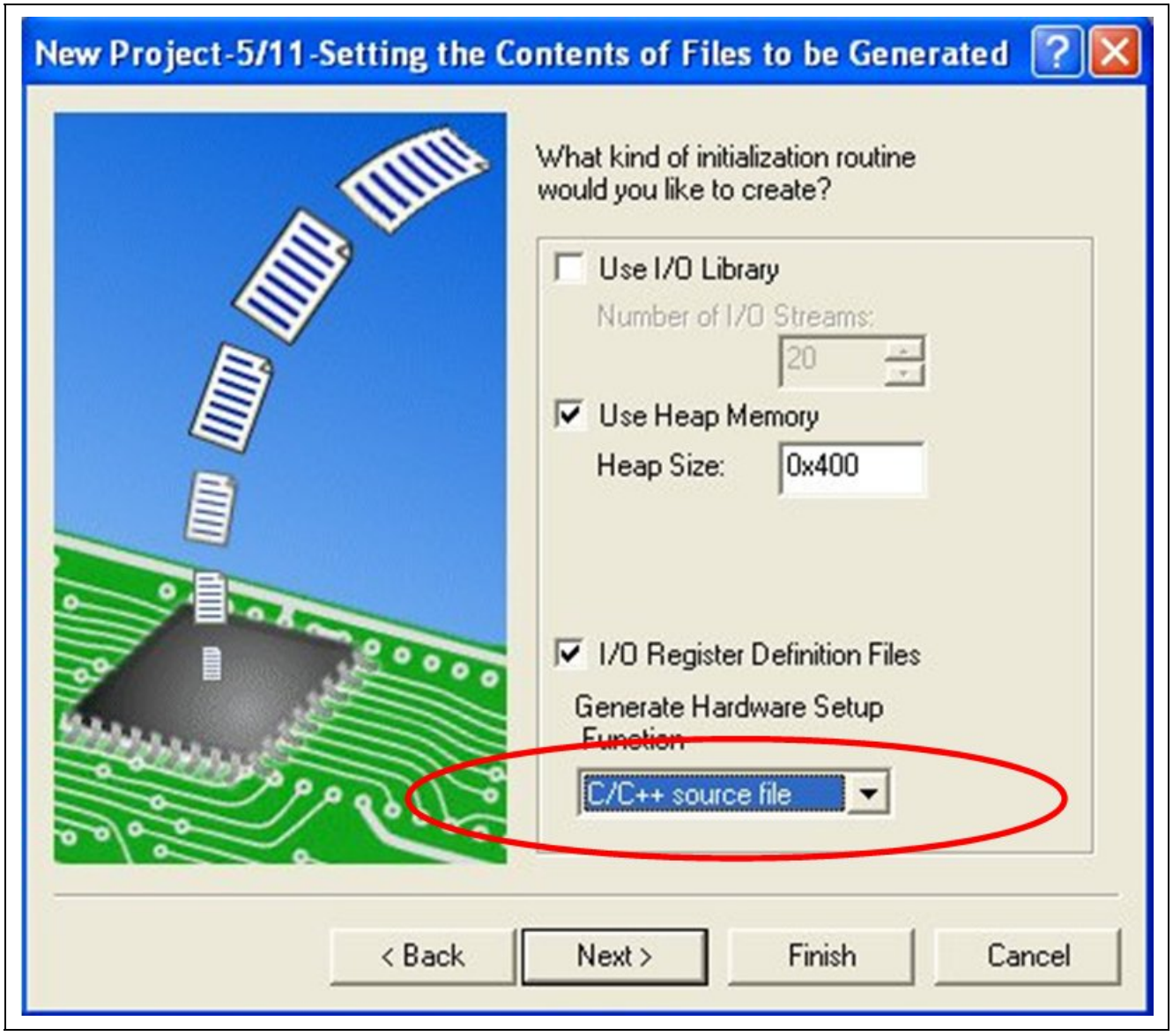
オプション設定は、以下に示すとおりです。



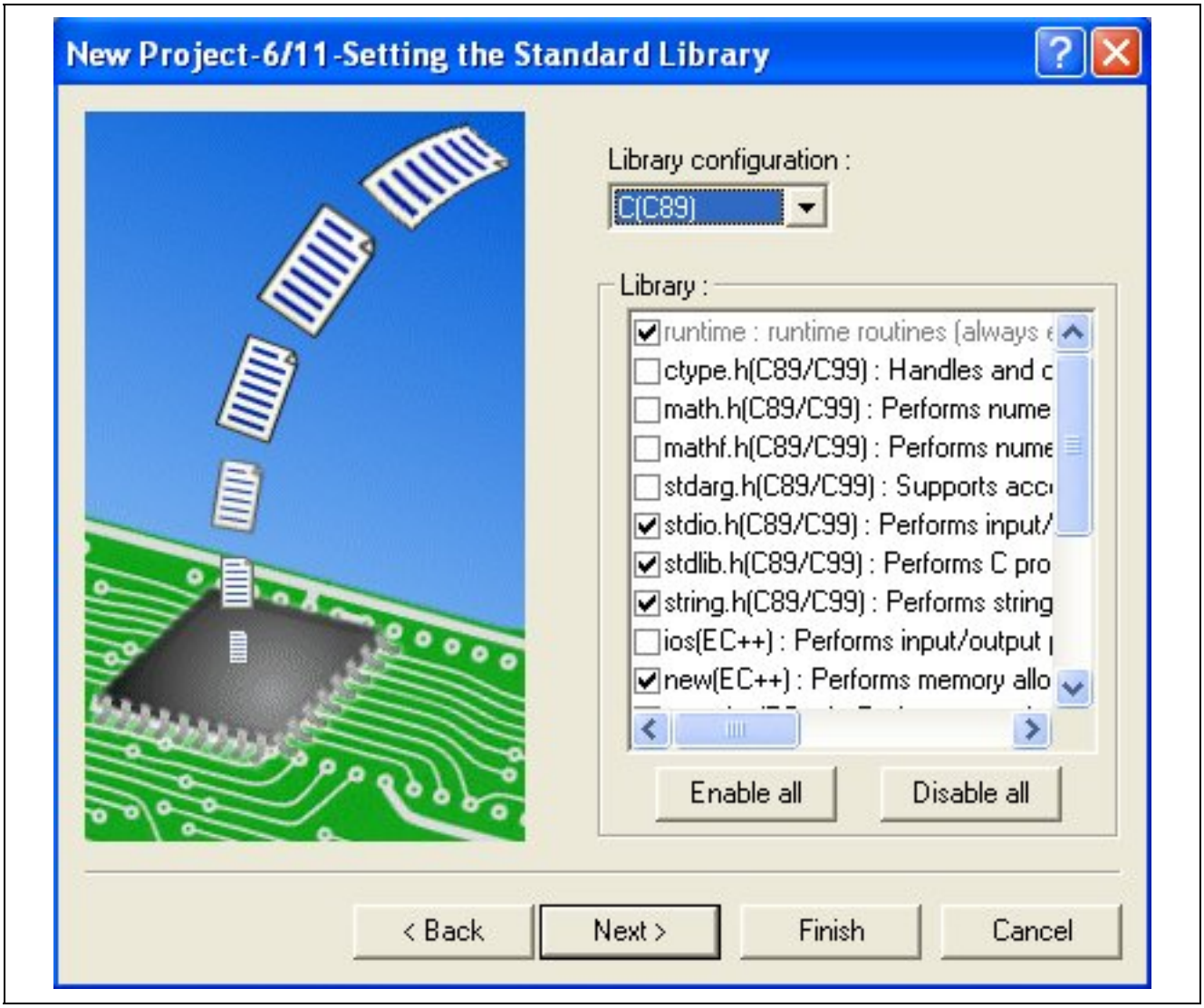
デフォルトを受け入れます。



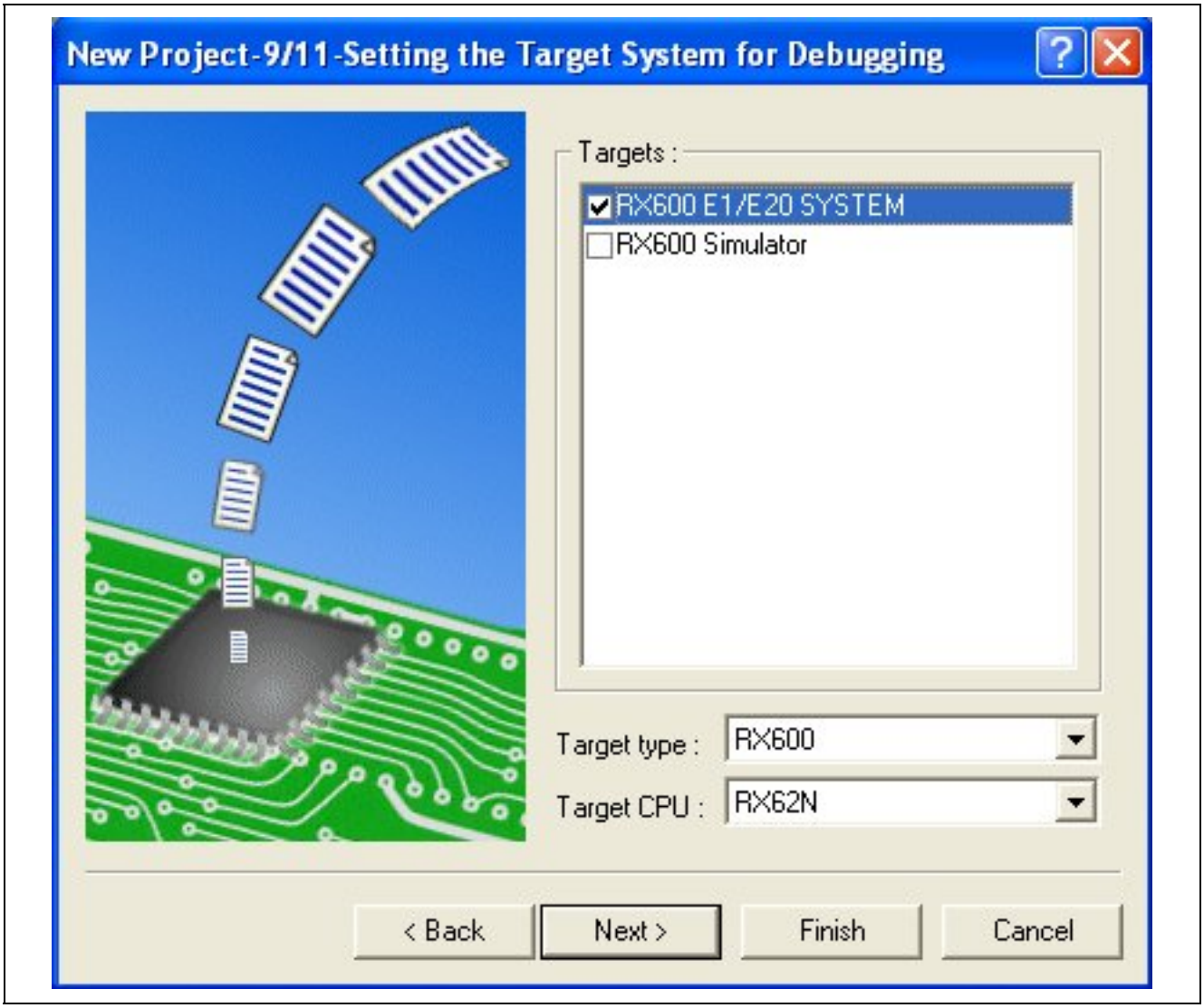
「C/C++ source file」が選択されていることを確認します。



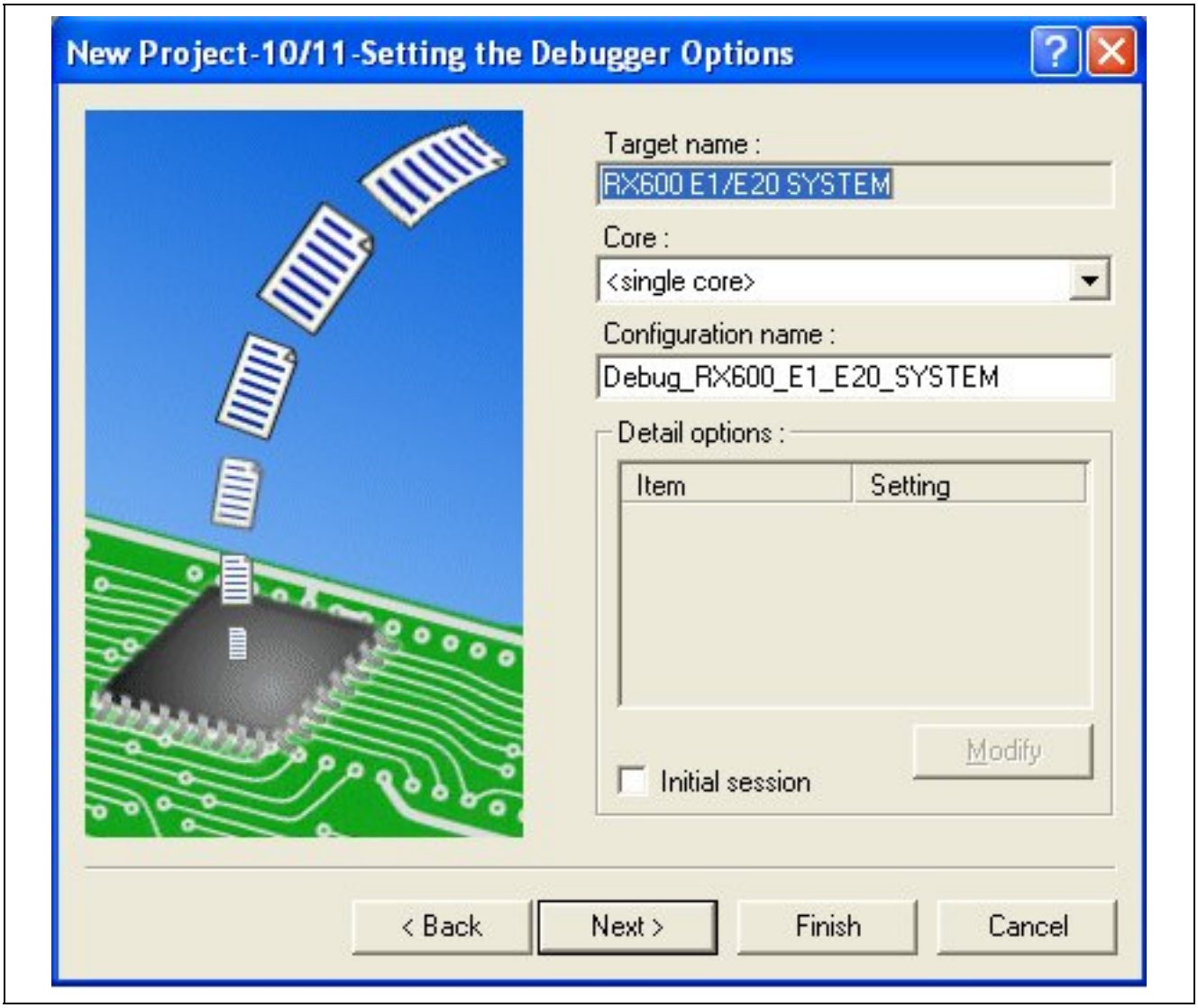
標準ライブラリコンフィグレーションのデフォルトを受け入れます。



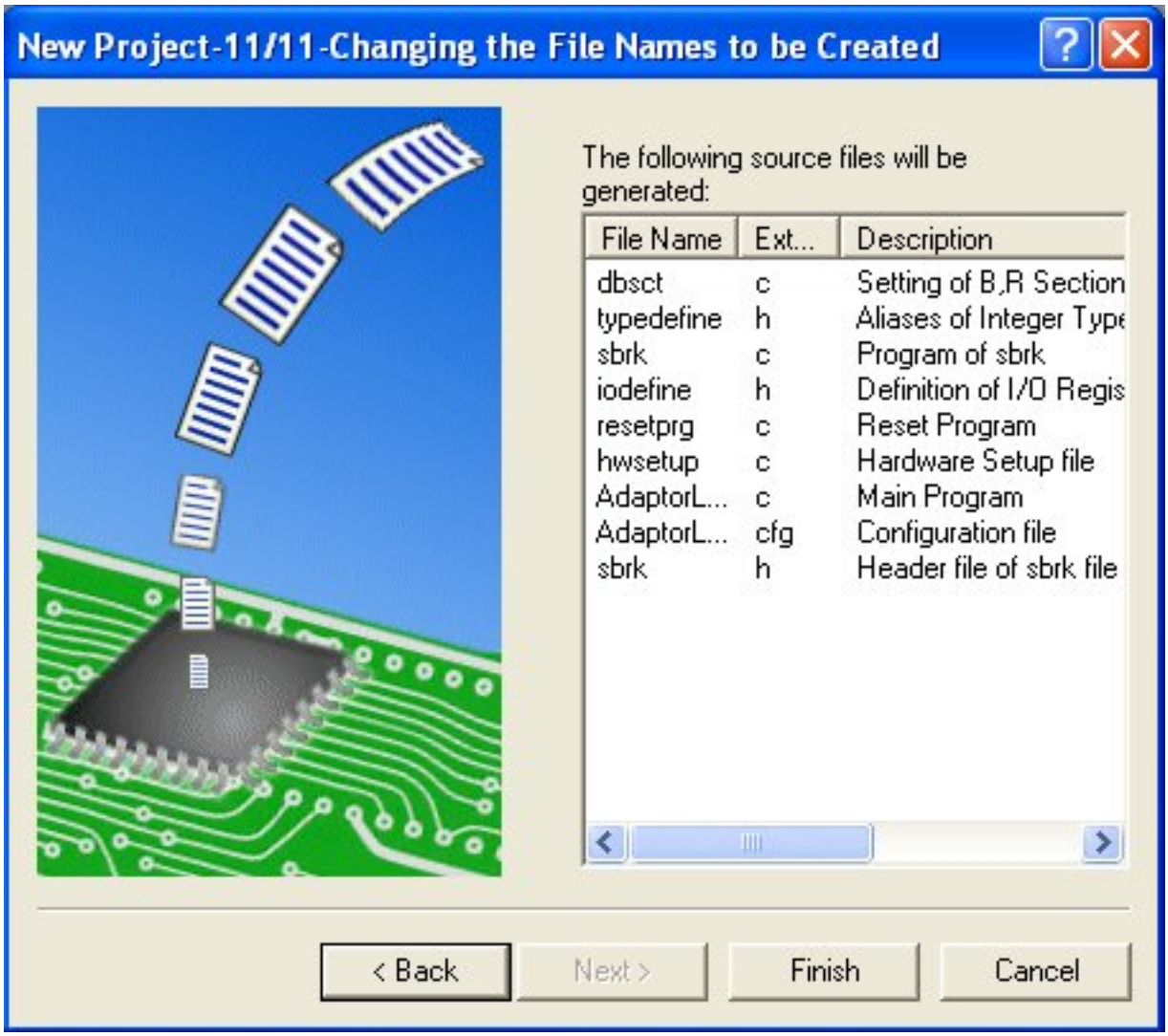
[Next]をクリックしてステップ 9 に進み、実行するエミュレータを選択します。



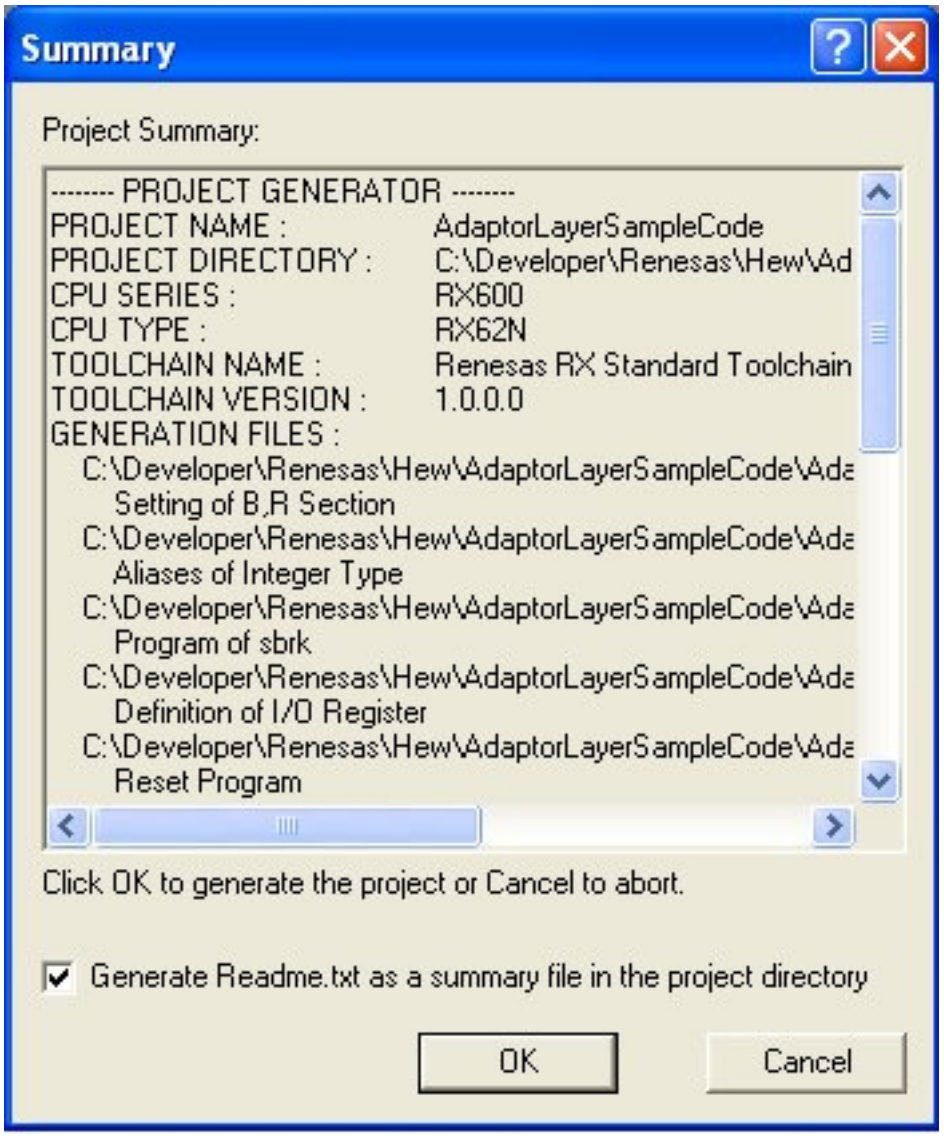
以下に示すように、ステップ 10 のデフォルトを受け入れます。



[Finish]をクリックしてプロジェクトワークスペースの作成を完了します。

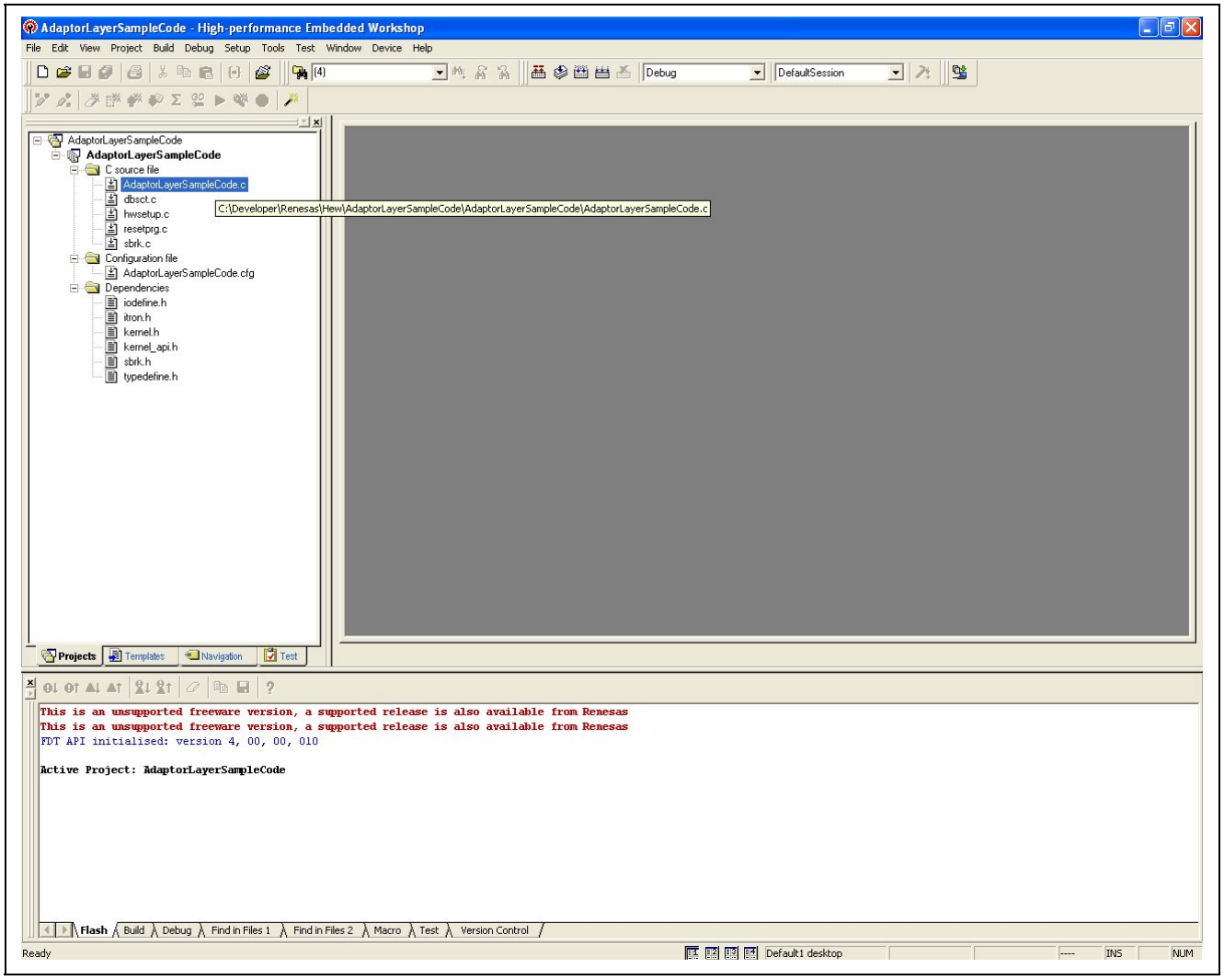


[OK]をクリックしてプロジェクト作成ワークスペースの概要を受け入れます。



2.3　LED 点滅サンプルコードの作成

プロジェクトワークスペースに表示される「AdaptorLayerSampleCode.c」という名前のファイルを開きます。

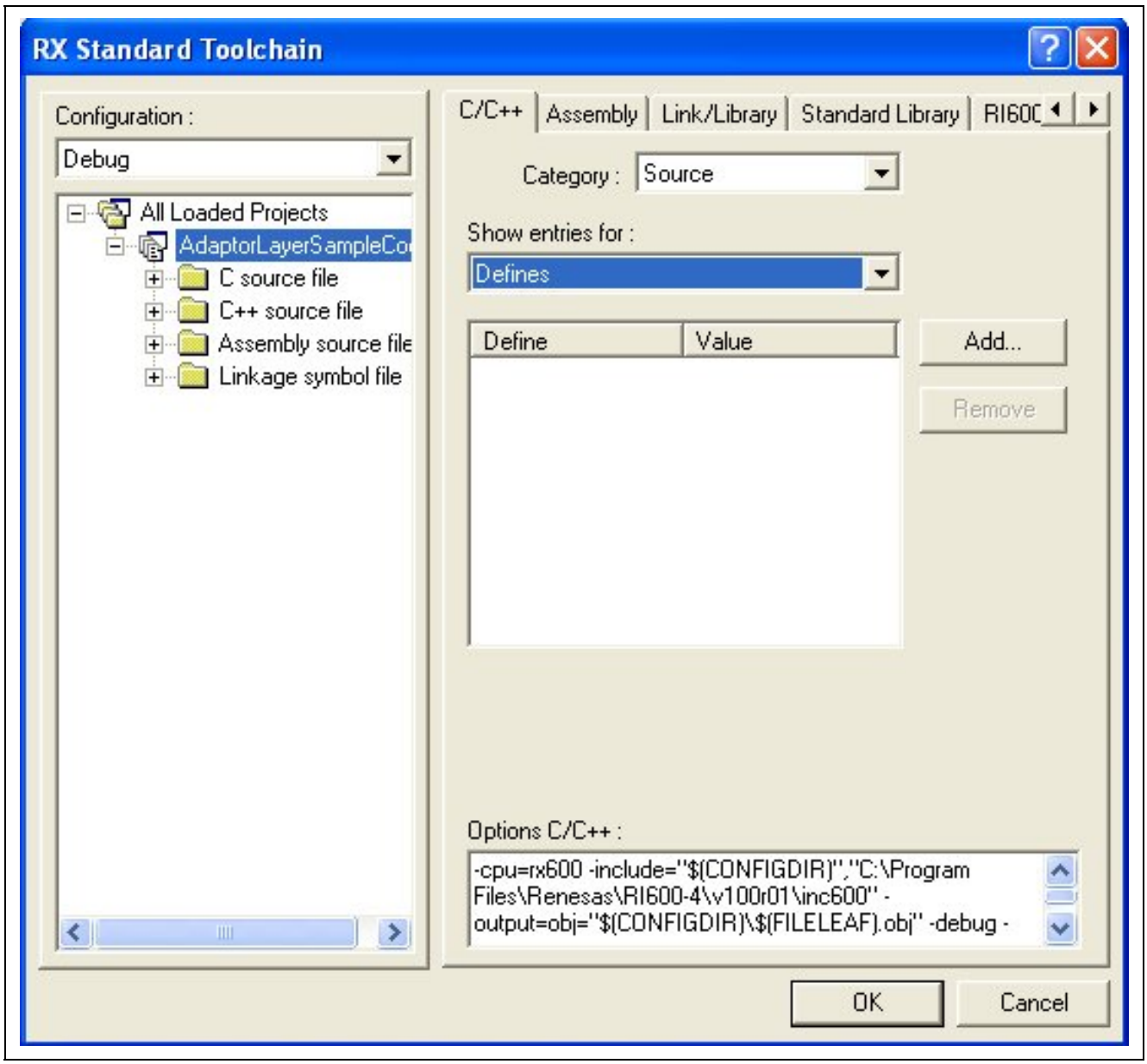


サンプルコード全体を以下のコードに置き換えます。

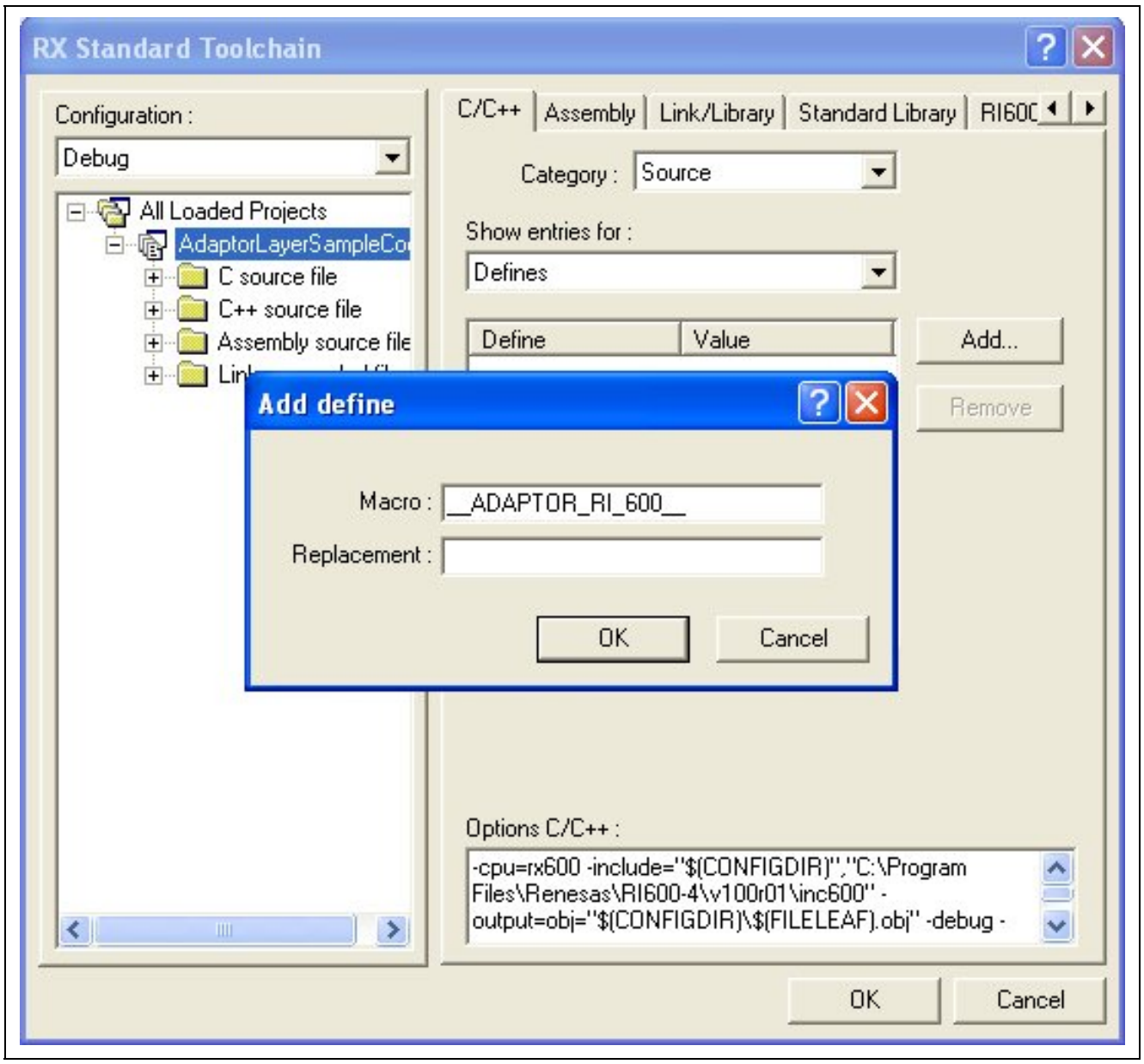
```
#include <machine.h>
#include <stdio.h>
#if defined (__ADAPTOR_RI_600__)
  #include "kernel.h"
  #include "kernel_id.h"
#elif defined (__ADAPTOR_FREE_RTOS__)
  #include "r_FreeRTOSAdaptor.h"
#endif

#include "iodefine.h"
/***********************************************************************
*  Defines.
*
***********************************************************************/
/* LEDs */
#define     LED0        PORT0.DR.BIT.B2
#define     LED1        PORT0.DR.BIT.B3
#define     LED2        PORT0.DR.BIT.B5
#define     LED3        PORT3.DR.BIT.B4
#define     LED0_DDR        PORT0.DDR.BIT.B2
#define     LED1_DDR        PORT0.DDR.BIT.B3
#define     LED2_DDR        PORT0.DDR.BIT.B5
#define     LED3_DDR        PORT3.DDR.BIT.B4
#define     LED_ON         (0)
#define     LED_OFF        (1)

/***********************************************************************
*  Function Prototypes.
*
***********************************************************************/
void InitLEDs();
void LED_Task(VP_INT param);

/***********************************************************************
* Function Name   : InitLEDs();
* Description     : This function initializes the LED for this demo
*             application.
* Argument        : None.
* Return Value    : None.
***********************************************************************/
void InitLEDs(){
  LED0_DDR = 1;
  LED1_DDR = 1;
  LED2_DDR = 1;
  LED3_DDR = 1;

  LED0 = LED_OFF;
  LED1 = LED_OFF;
  LED2 = LED_OFF;
  LED3 = LED_OFF;
}
```

```
/***********************************************************************
*
* Function Name   : LED_Task();
* Description     : This is the LED Task that will execute the Application
*             code.
* Argument        : None.
* Return Value    : None.
***********************************************************************
/
void LED_Task(VP_INT param)
{
   // Local Variables
  static unsigned char toggle = 0;

   InitLEDs();

  for (;;){

   switch (toggle){
     case 0:
         LED0 = ~LED0;
       break;
     case 1:
         LED1 = ~LED1;
       break;
     case 2:
         LED2 = ~LED2;
       break;
     case 3:
        LED3 = ~LED3;
       break;
   }

   toggle = (toggle == 3) ? 0 : ++toggle;

   dly_tsk(500);
  }
}
```

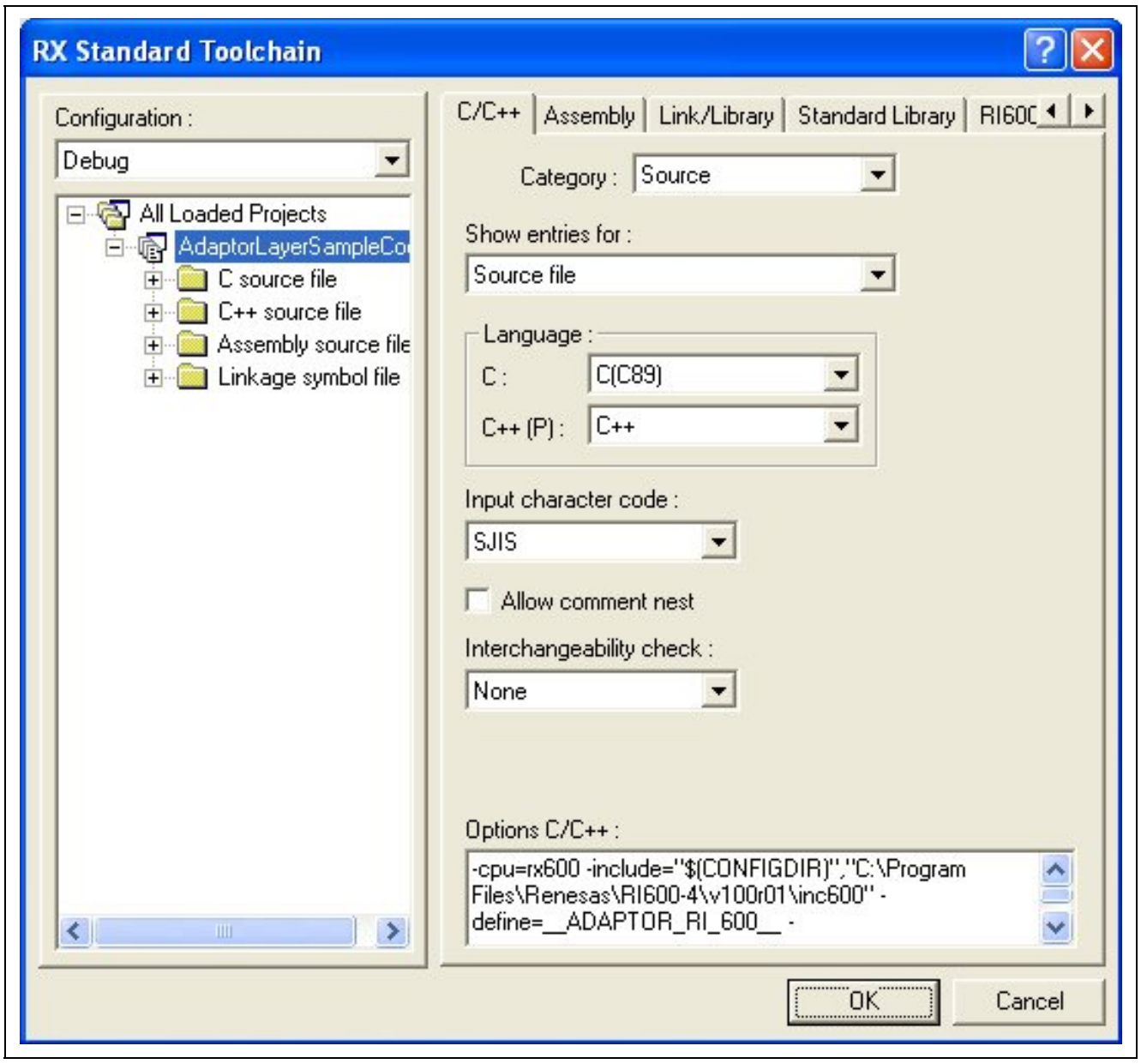
[Build]→[RX Standard Toolchain]の[C/C++]タブで、RI-600プロジェクト用に定義したマクロを追加し、[Show entries for:]に[Defines]を選択します。これを以下に示します。



以下に示すように、グローバルに定義したマクロ__ADAPTOR_RI_600__を追加します。

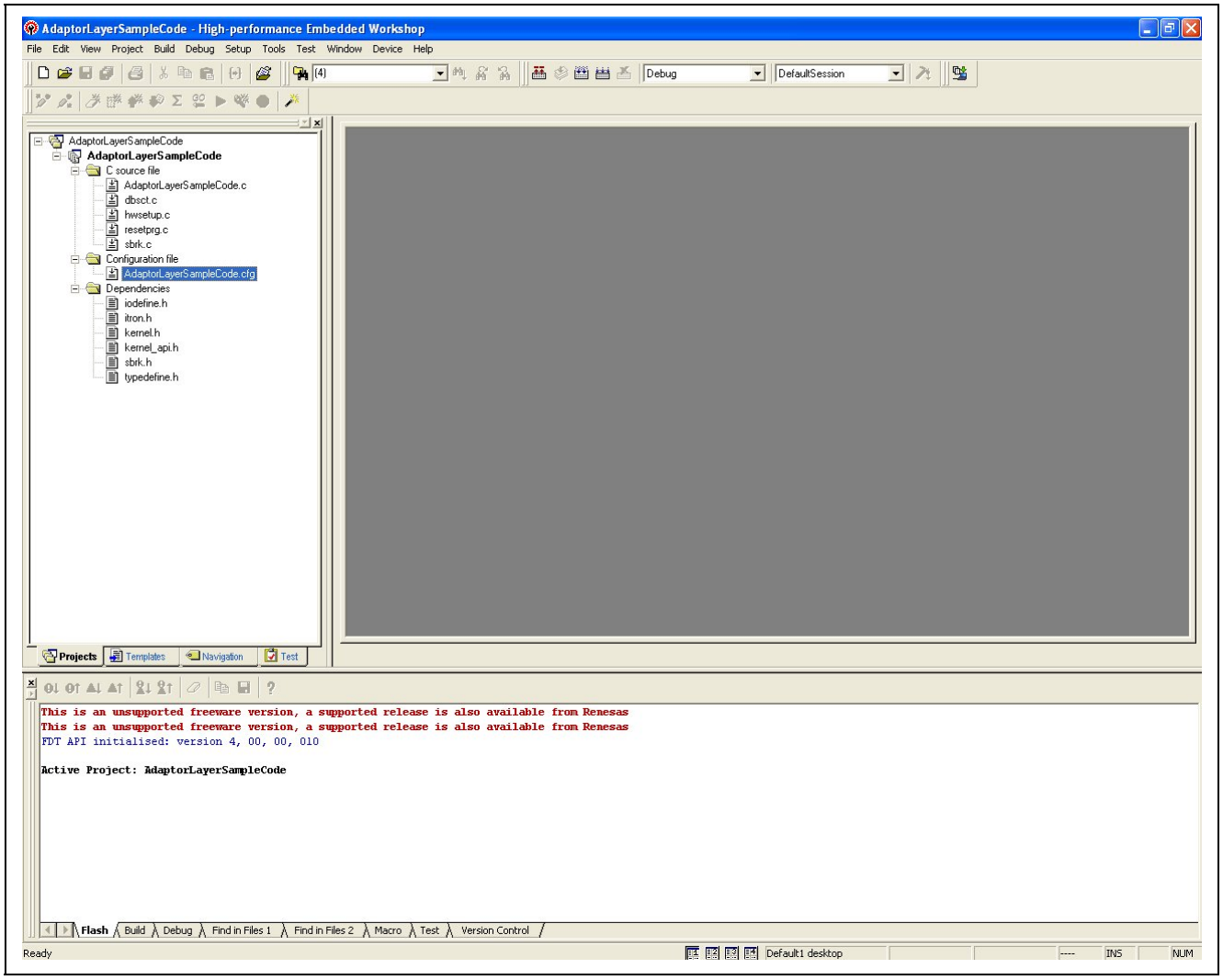


[OK]を選択し、[RX Standard Toolchain]オプションパネルを終了します。



2.4　RI-600 コンフィグレータファイルの編集

以下に示すように、「AdaptorLayerSampleCode.cfg」ファイルを開きます。これはコンパイルプロセス中に RI-600 が使用するコンフィグレータファイルです。



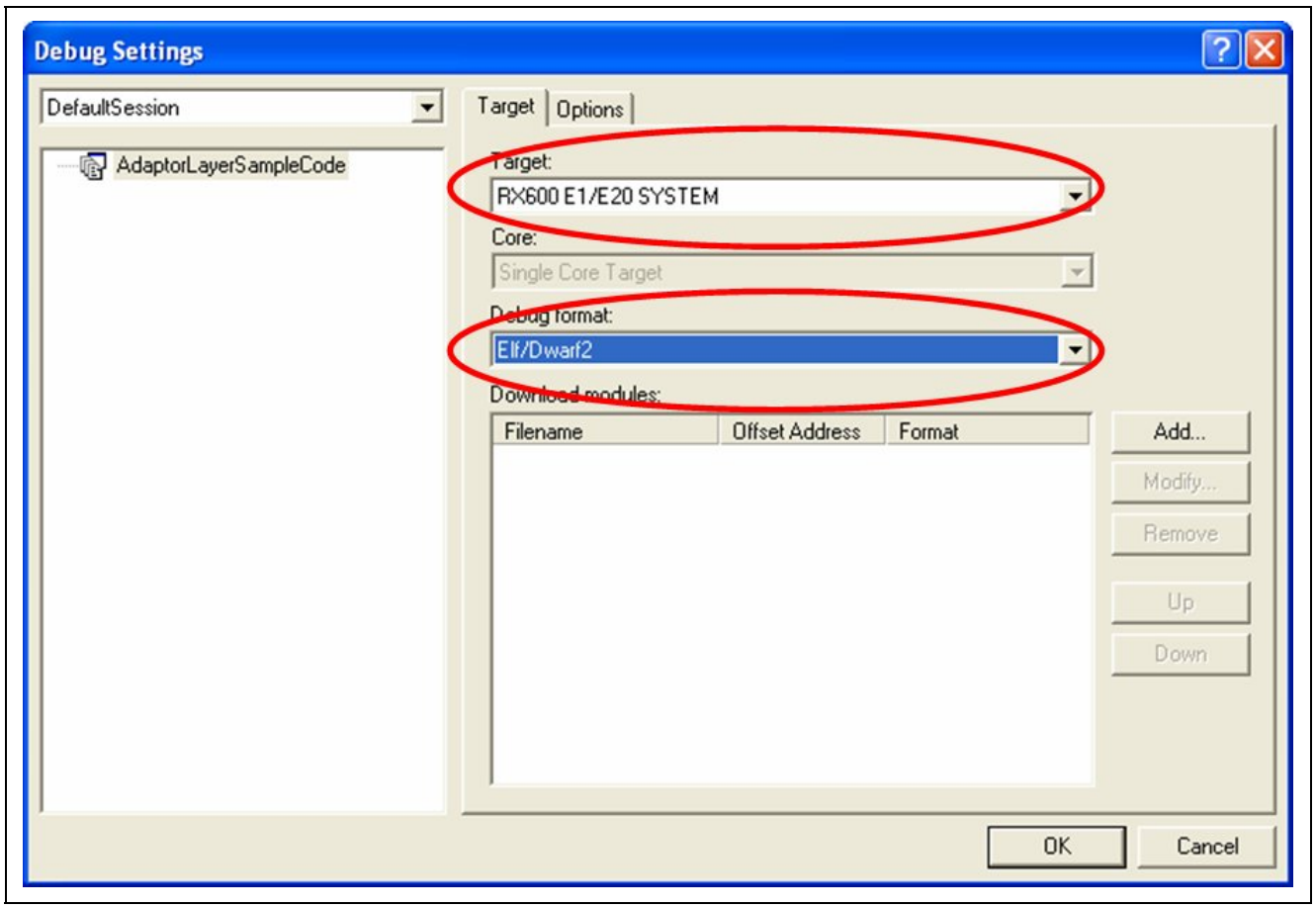
コンフィグレーションを以下のコンフィグレーションに置き換えます。

```
//************************************************************
//      RI600/4 System Configuration File
//        FPU:Yes, DSP:Yes
//************************************************************
// System Definition
system{
    stack_size  = 1024;
    priority     = 10;
    system_IPL  = 4;
    message_pri = 1;
    tic_deno    = 1;
    tic_nume    = 1;
    context     = FPSW,ACC;
};

//System Clock Definition
clock{
    timer       = CMT0;
  template   = rx610.tpl;
    timer_clock = 25MHz;
    IPL         = 3;
};

//Task Definition
task[]{
    name            = LED_TSK;
    entry_address   = LED_Task();
    initial_start   = ON;
    stack_size      = 512;
    priority        = 1;
//  stack_section   = STK1;
    exinf           = 1;
};
```

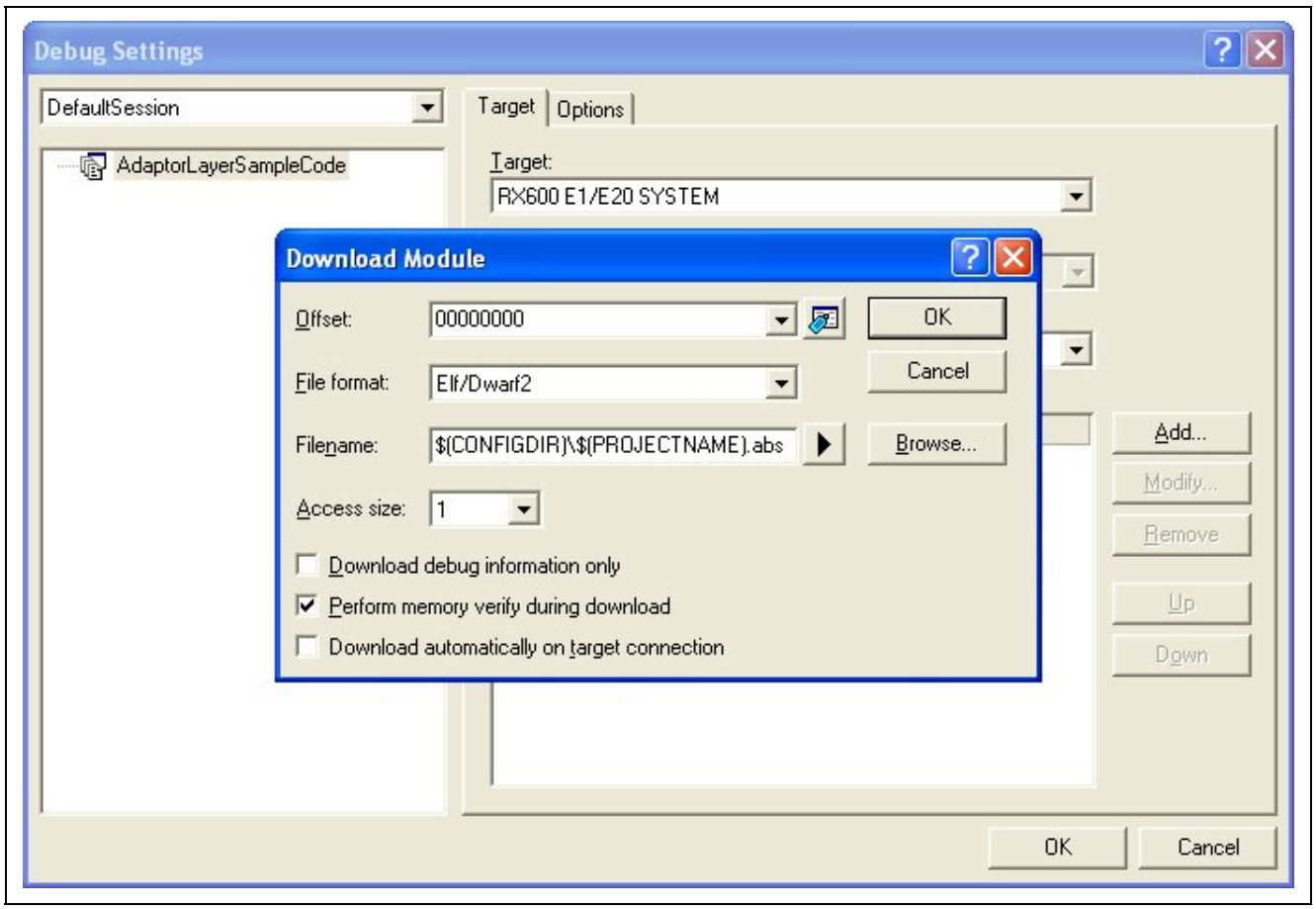
2.5　RX62N ボードへの接続

以下に示すとおり、エミュレータを追加して RX62N RSK ボードに接続します。

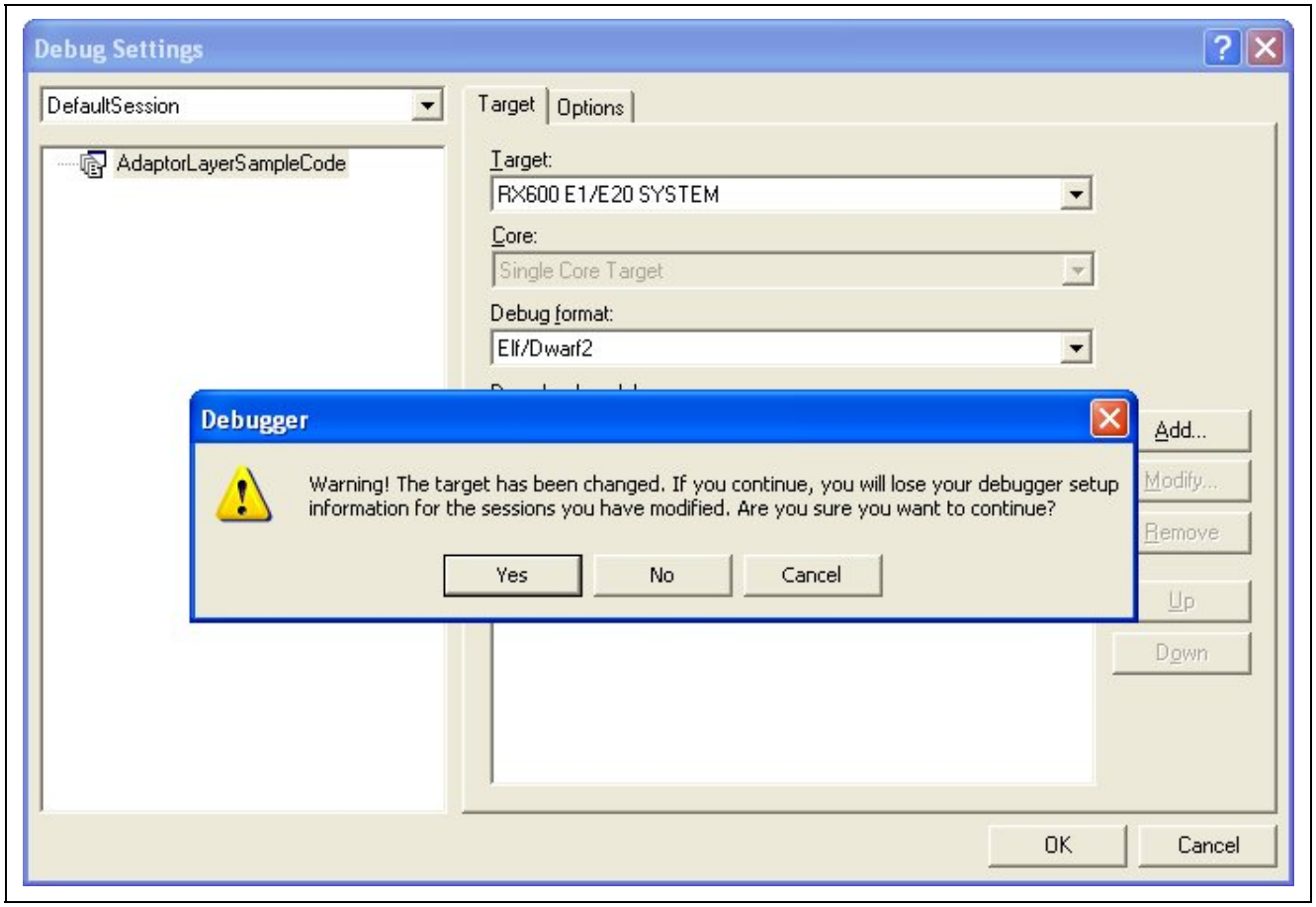
以下に示すように、次のデバッグ設定を追加します。



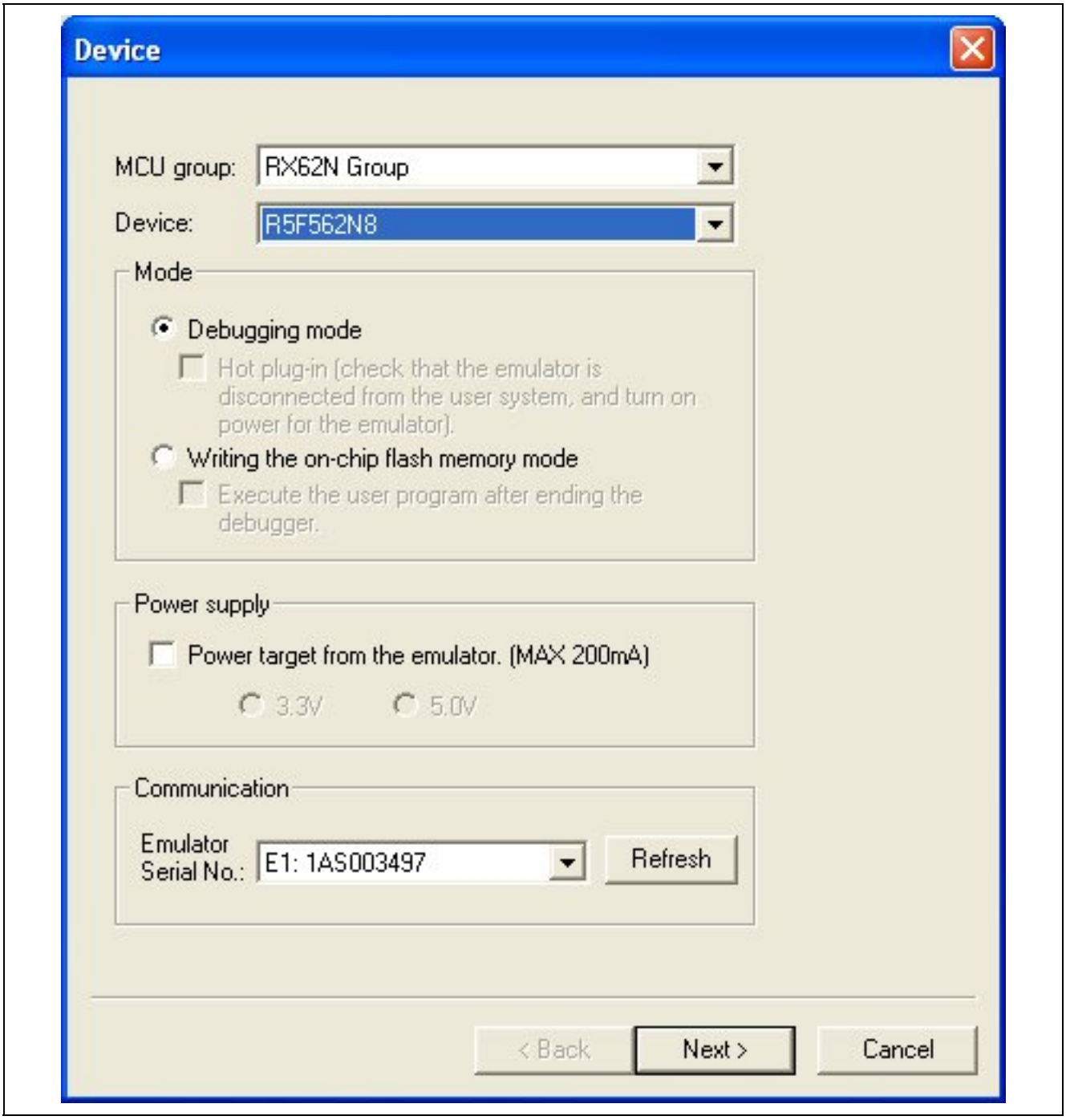
[Add]をクリックして以下のコンフィグレーションを追加します。



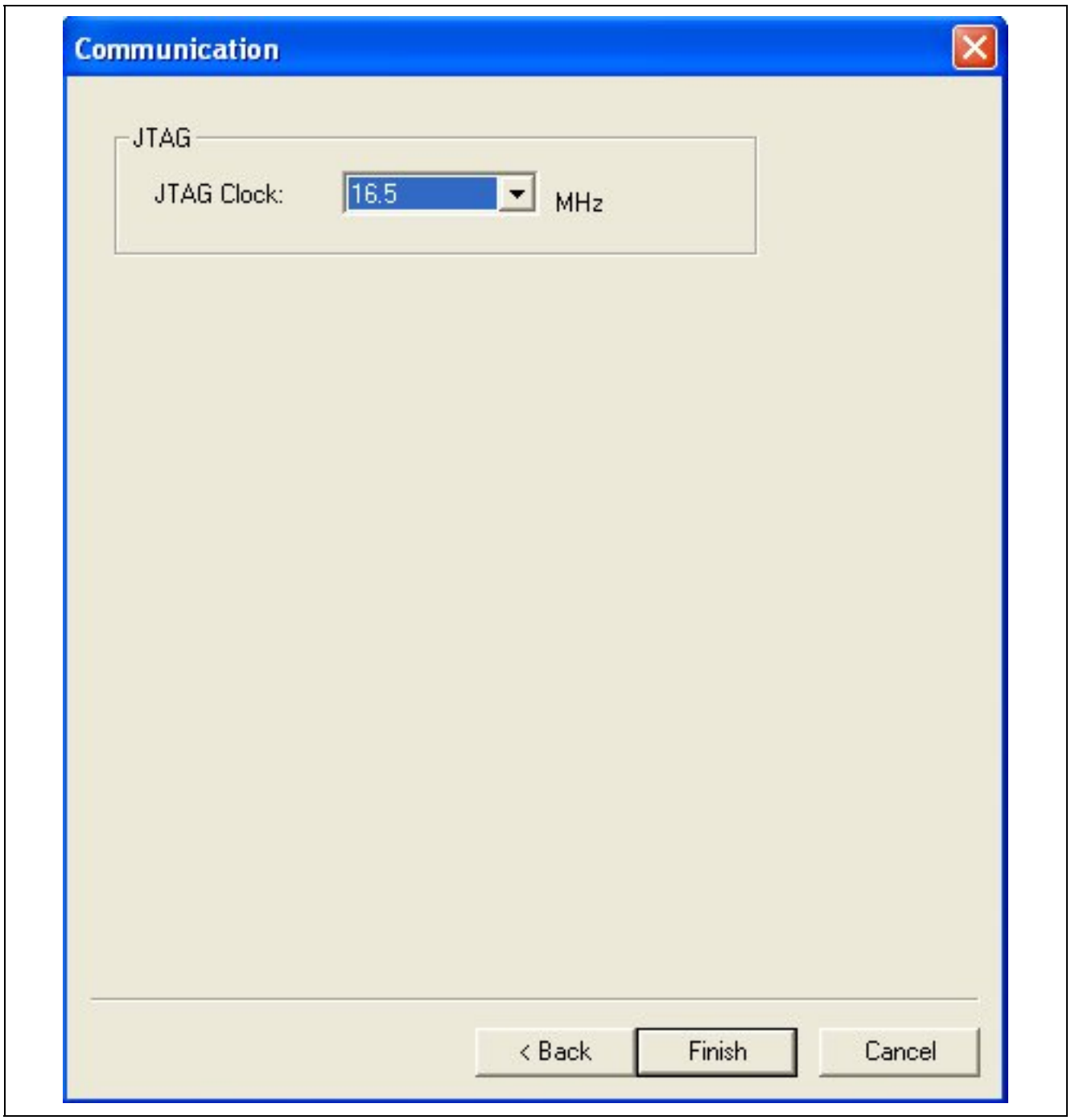
[OK]をクリックしてデフォルト設定を受け入れ、デバッガ警告パネルが表示されているときに[Yes]をクリックします。



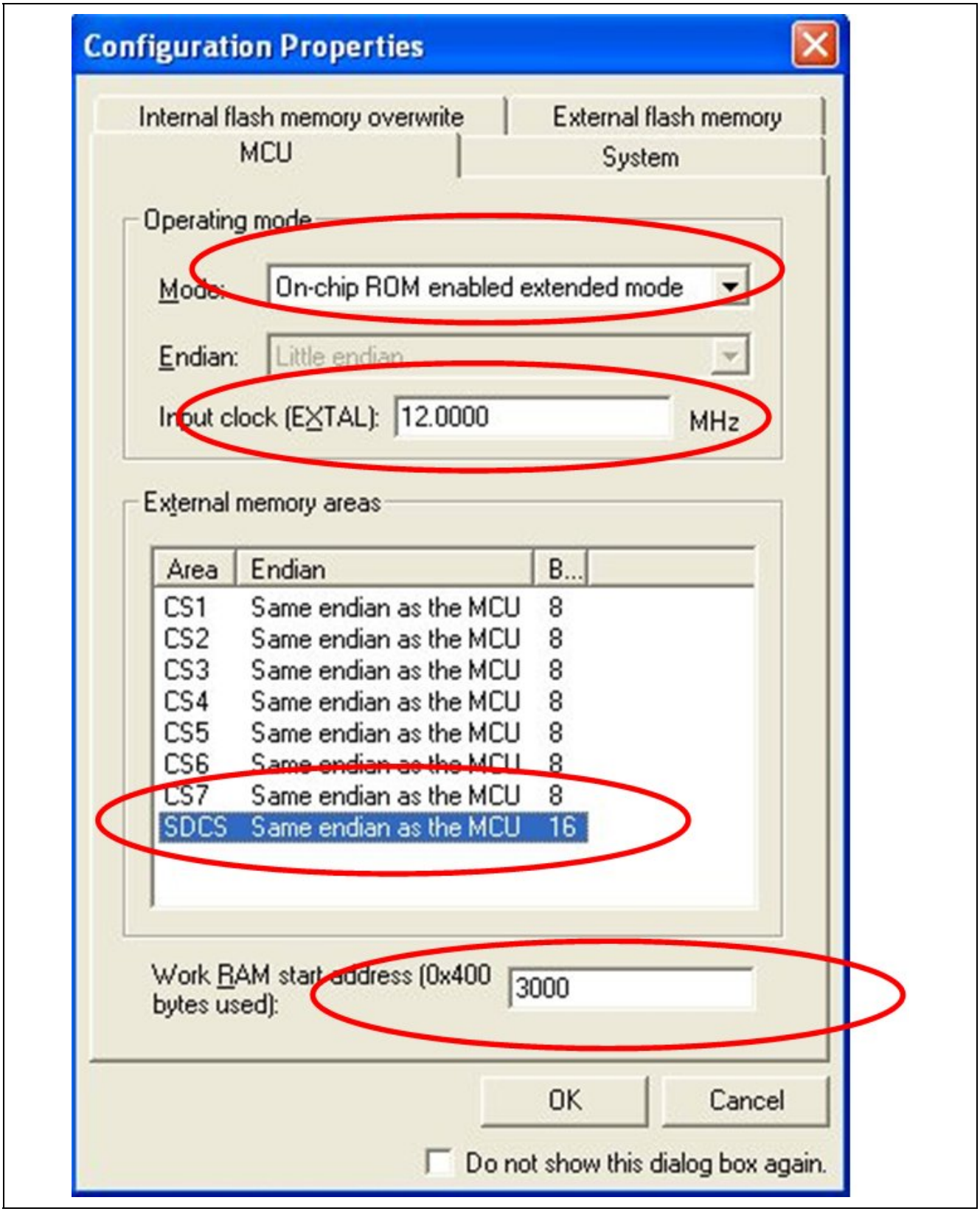
[Device]ウィンドウで以下を選択します。



ボードに電源が投入されていることを確認してから[Next >]をクリックします。



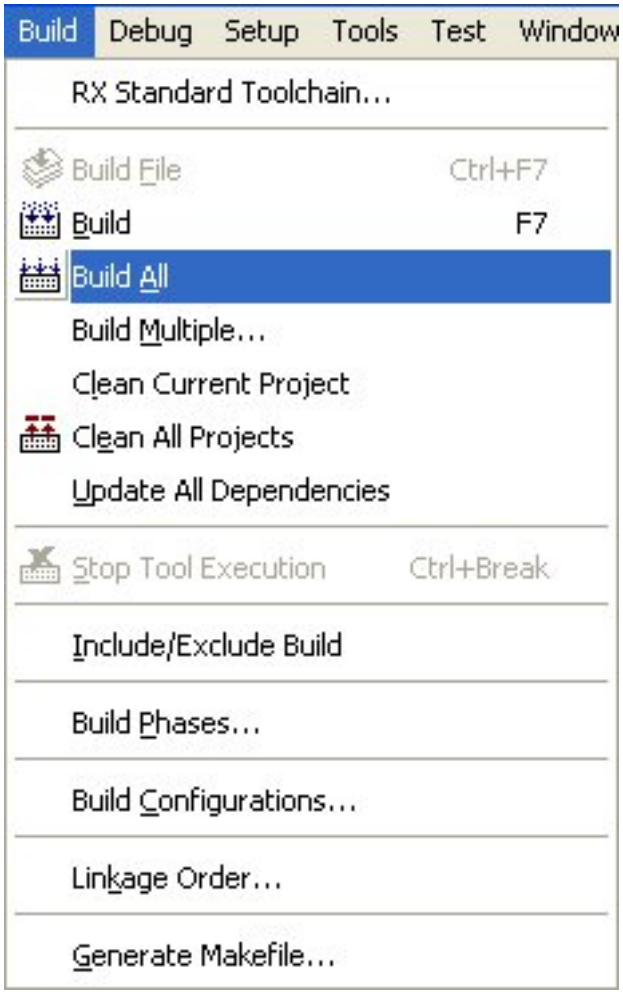
[Finish]をクリックしてデバイスへの接続を開始します。次のウィンドウで以下の設定を選択します。



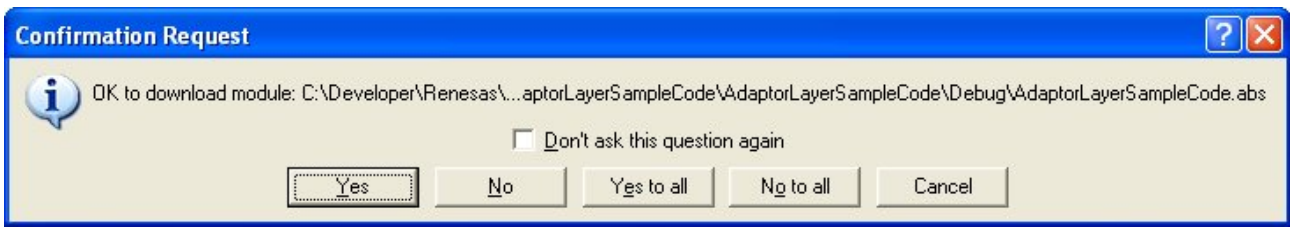
[OK]をクリックしてデバイスへの接続を進めます。

2.6　LED 点滅サンプルのコンパイルとビルド

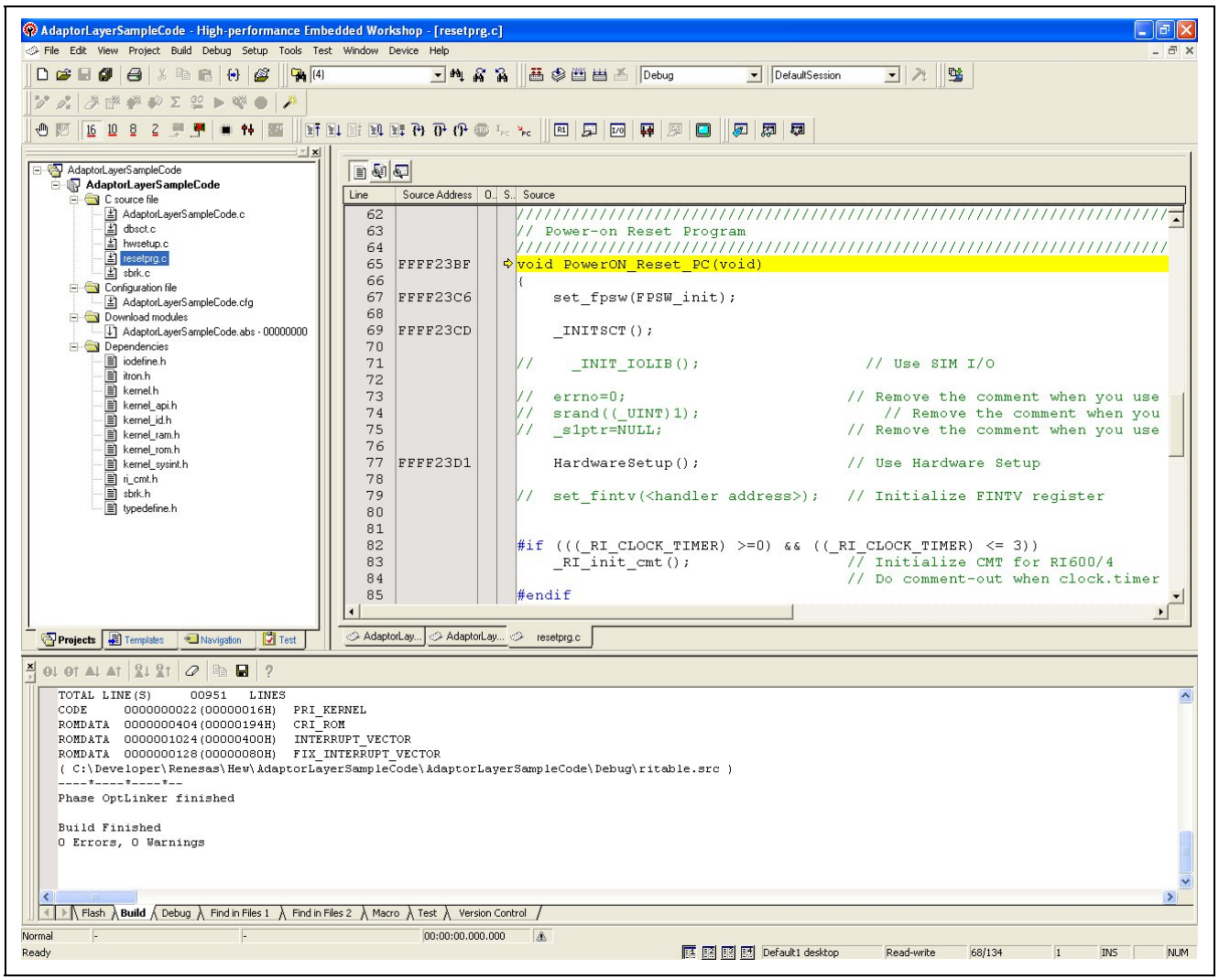
[Build]→[Build All]を選択してコンパイルを開始します。



コンパイルが完了したら、[Yes to all]を選択し、ビルドしたモジュールをダウンロードします。

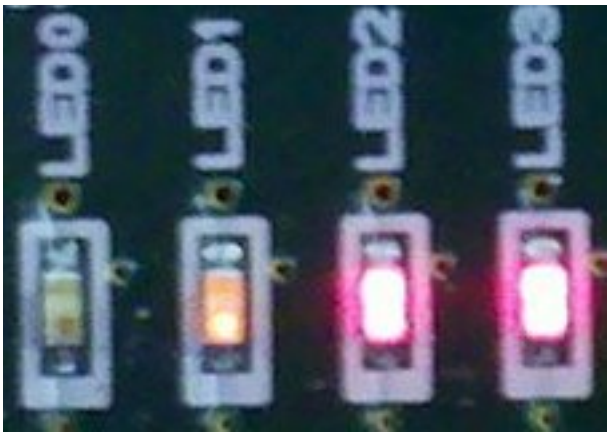


モジュールをダウンロードしたら、以下に示すとおり、プログラムの実行準備が完了です。



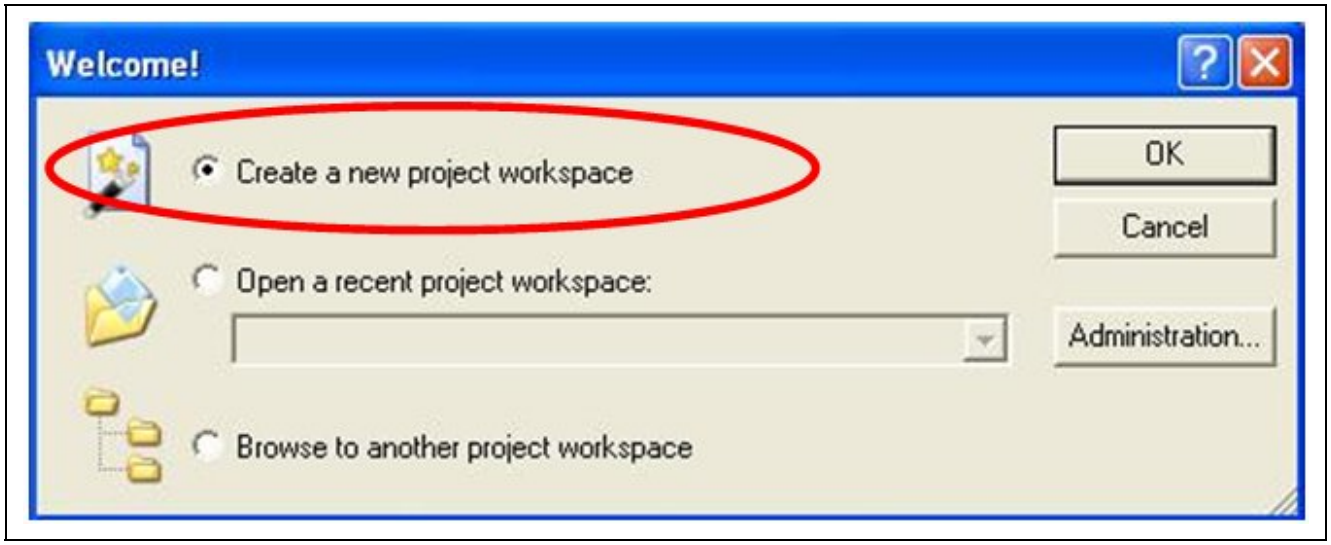
プログラムを実行するには、F5 を押すか、以下に示すとおり[Debug]→[Go]を選択します。

最終的に、RSK の LED の点滅は、次のようになります。

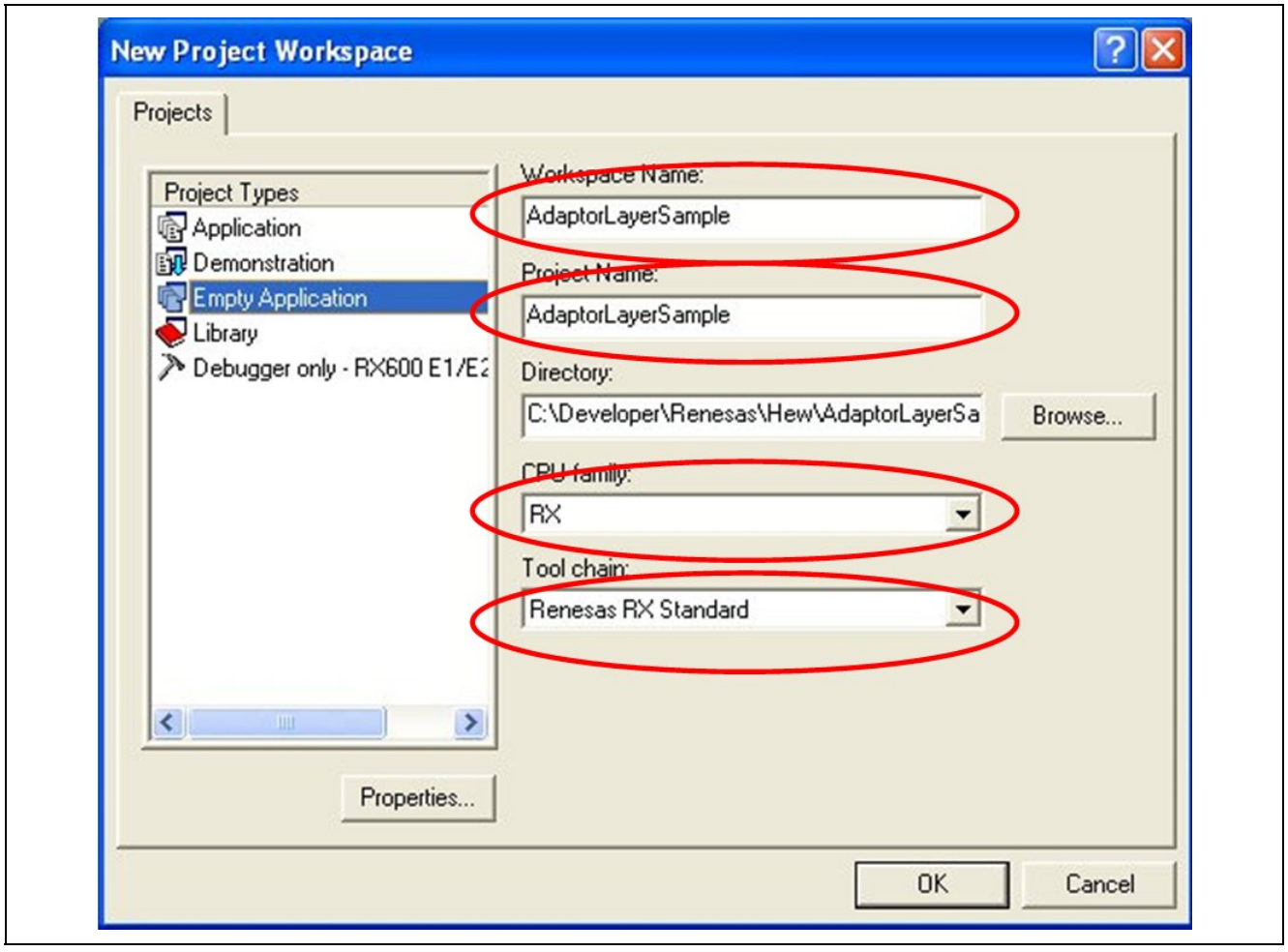


2.7　アダプタレイヤー用の空のワークスペースの作成

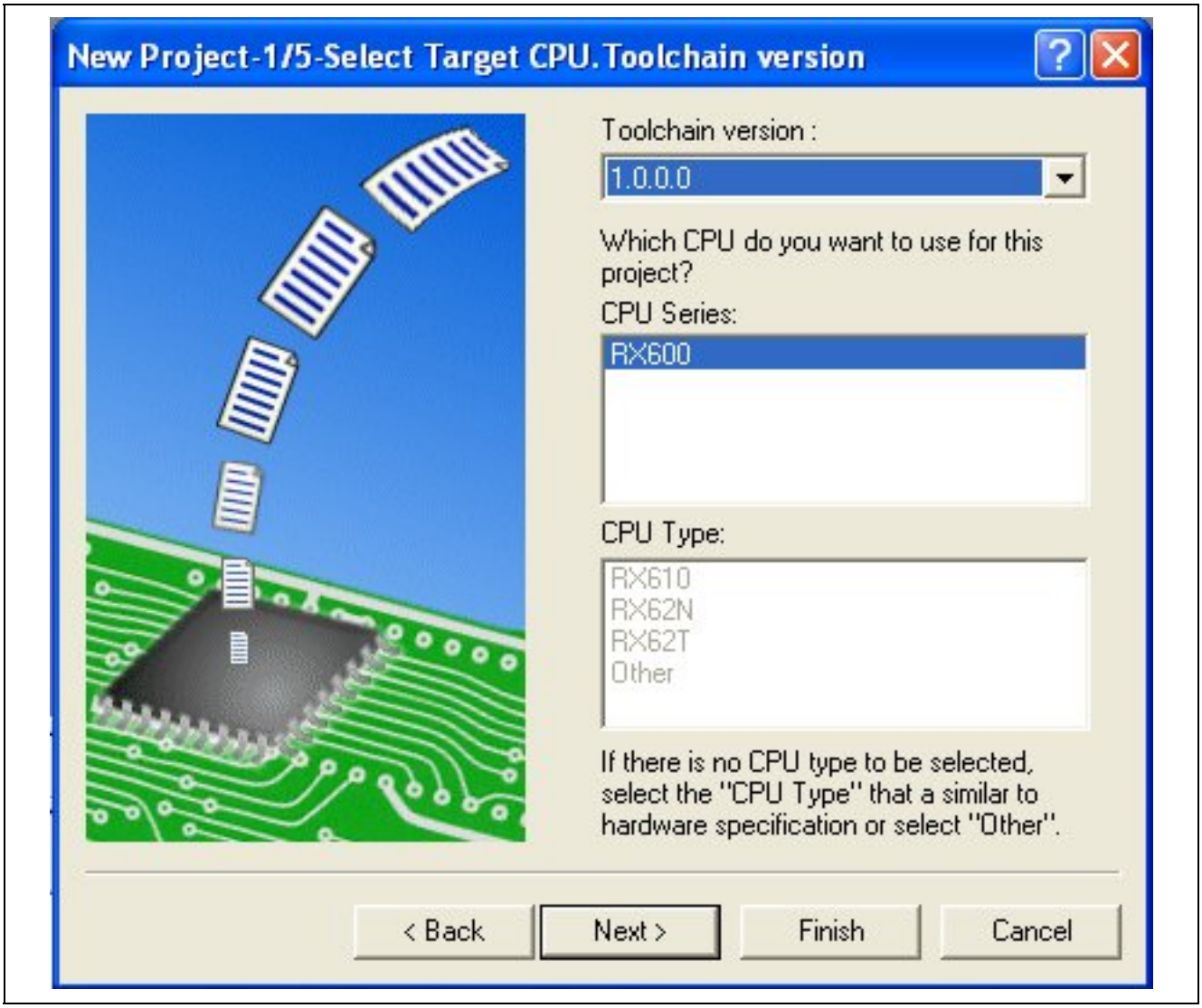
ルネサス HEW IDE を起動し、以下のように新しいプロジェクトワークスペースを作成します。



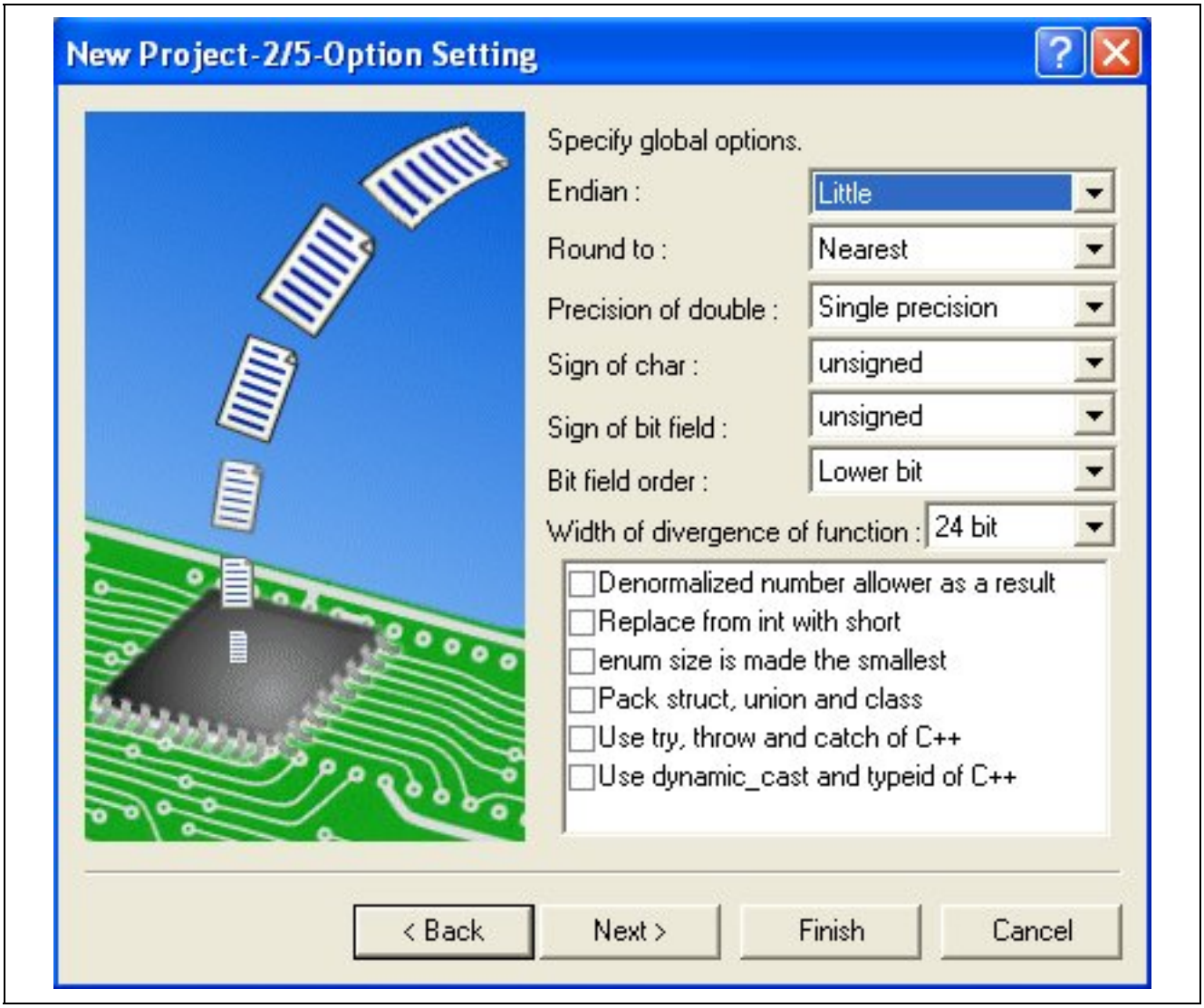
以下のように、有効なワークスペース名とプロジェクト名を入力します。



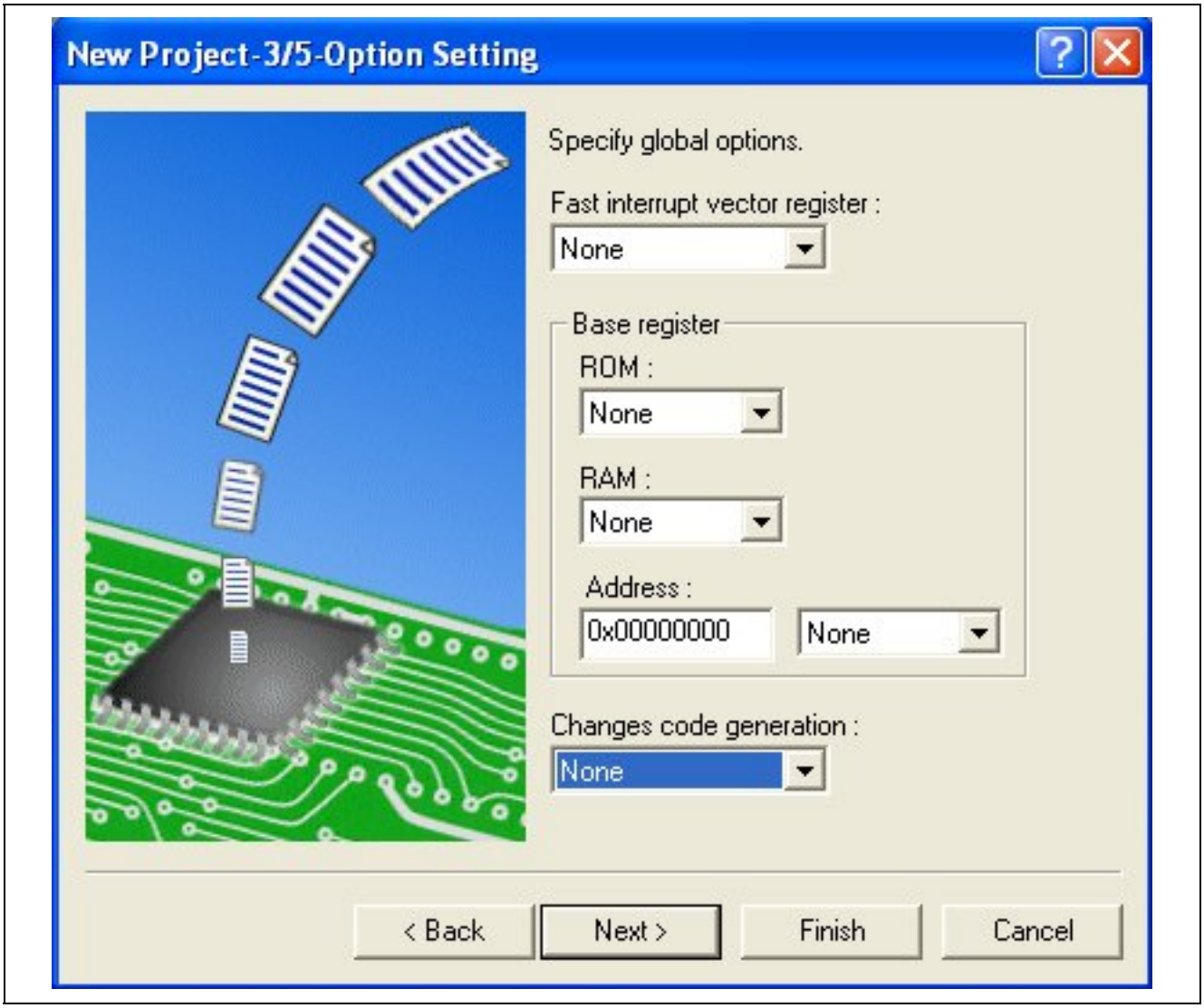
以下に示すとおり、ツールチェーンを選択します。



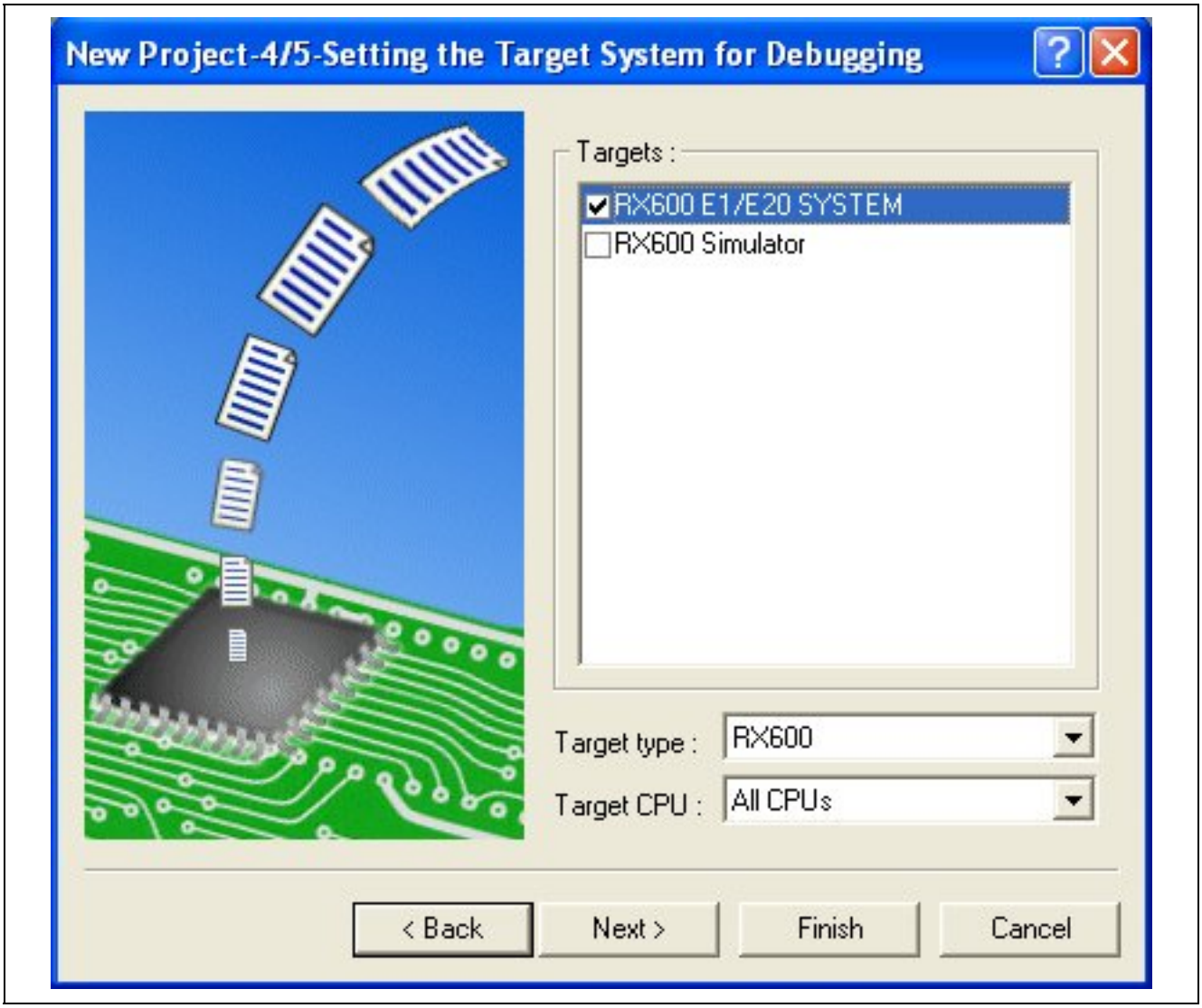
以下のウィンドウで設定を受け入れます。



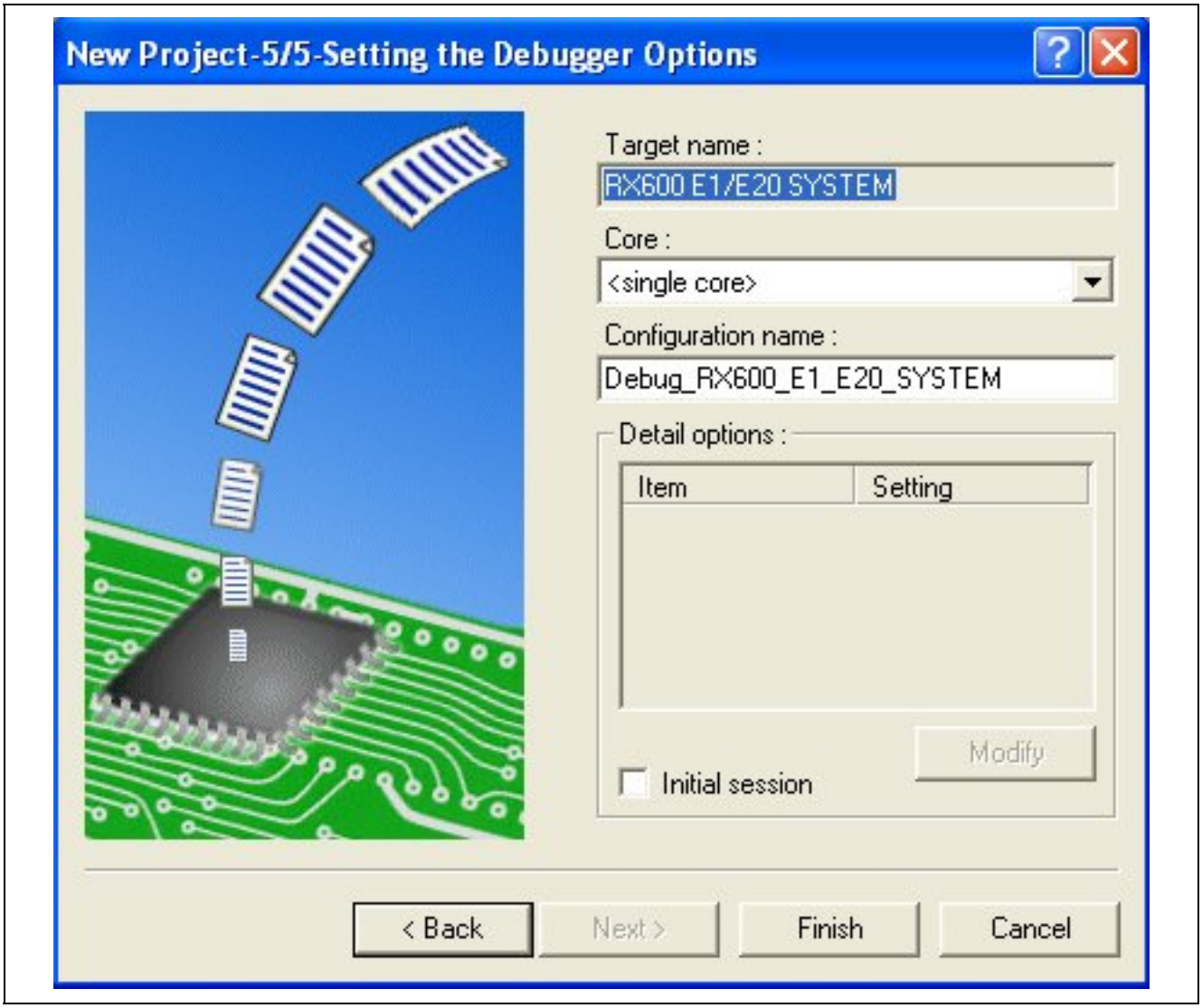
デフォルトを受け入れます。



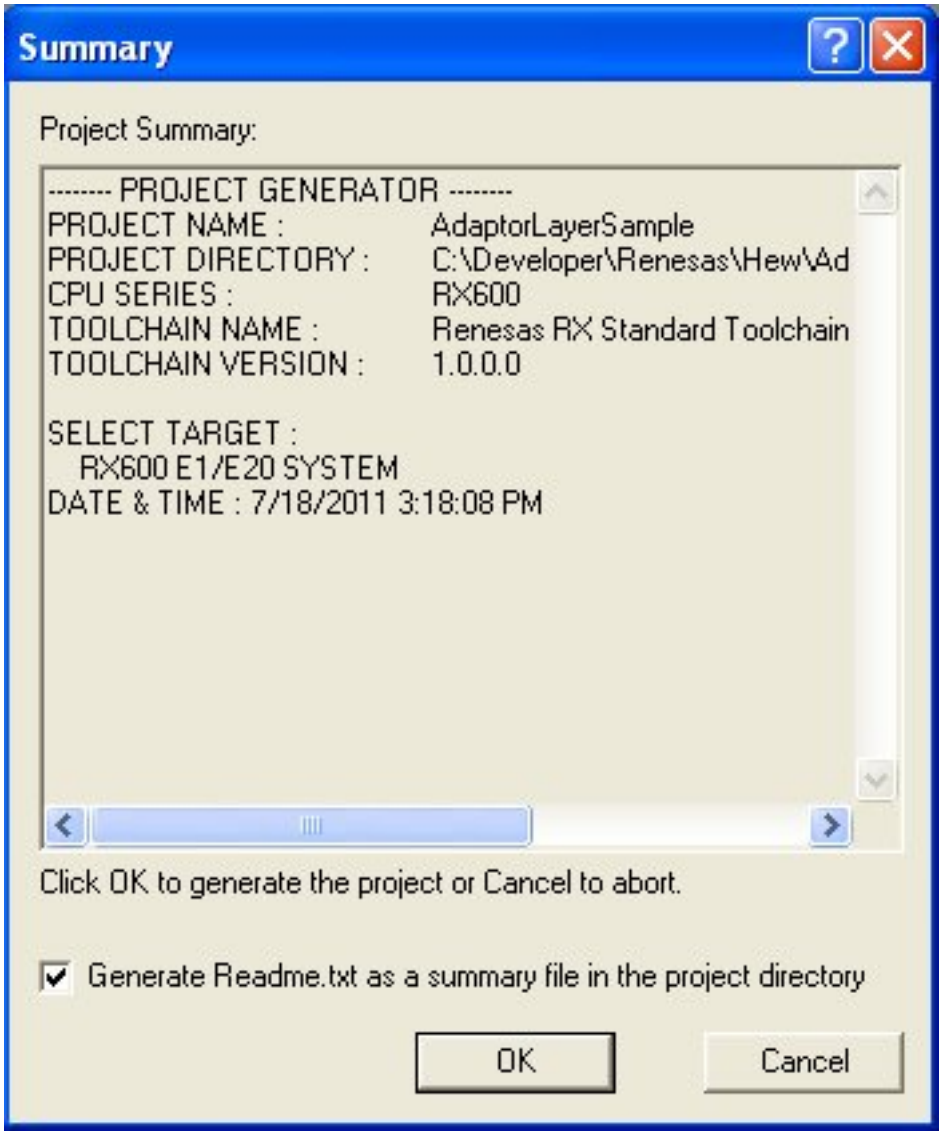
デフォルトを受け入れます。



[Finish]をクリックしてプロジェクトワークスペースを生成します。

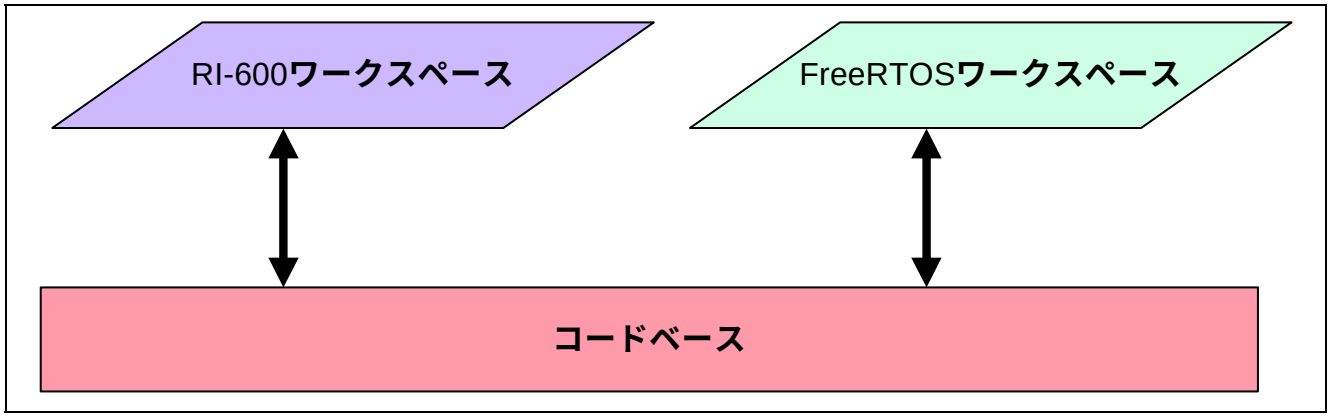


以下に示すプロンプトで[OK]をクリックします。

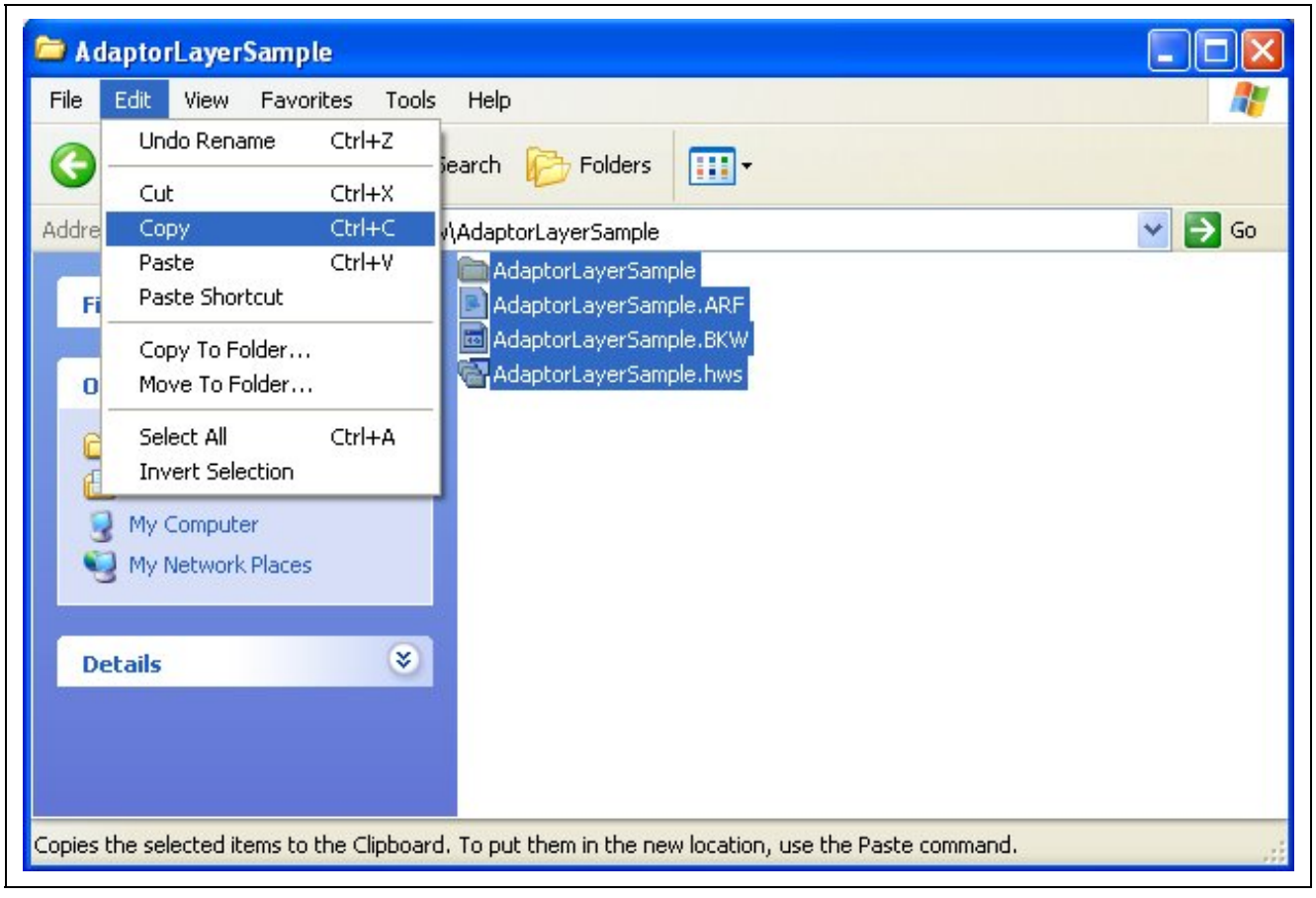


2.8　ワークスペースファイルの結合

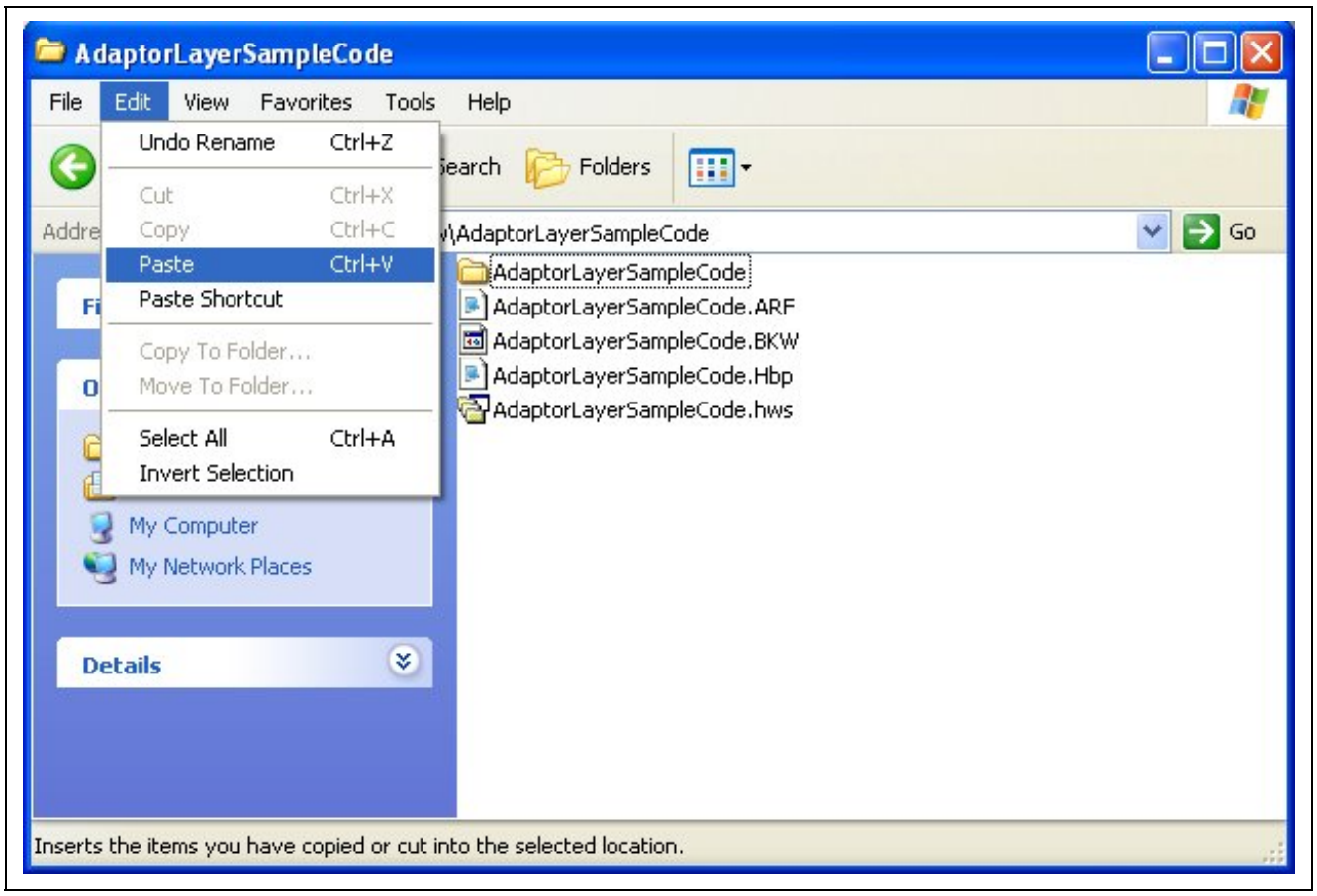
デュアル RTOS アプリケーションを開発するには、同じコードベースを共有するファイルを持つ 2 つのワークスペースを結合することが最良です。これは、次の図で表すことができます。



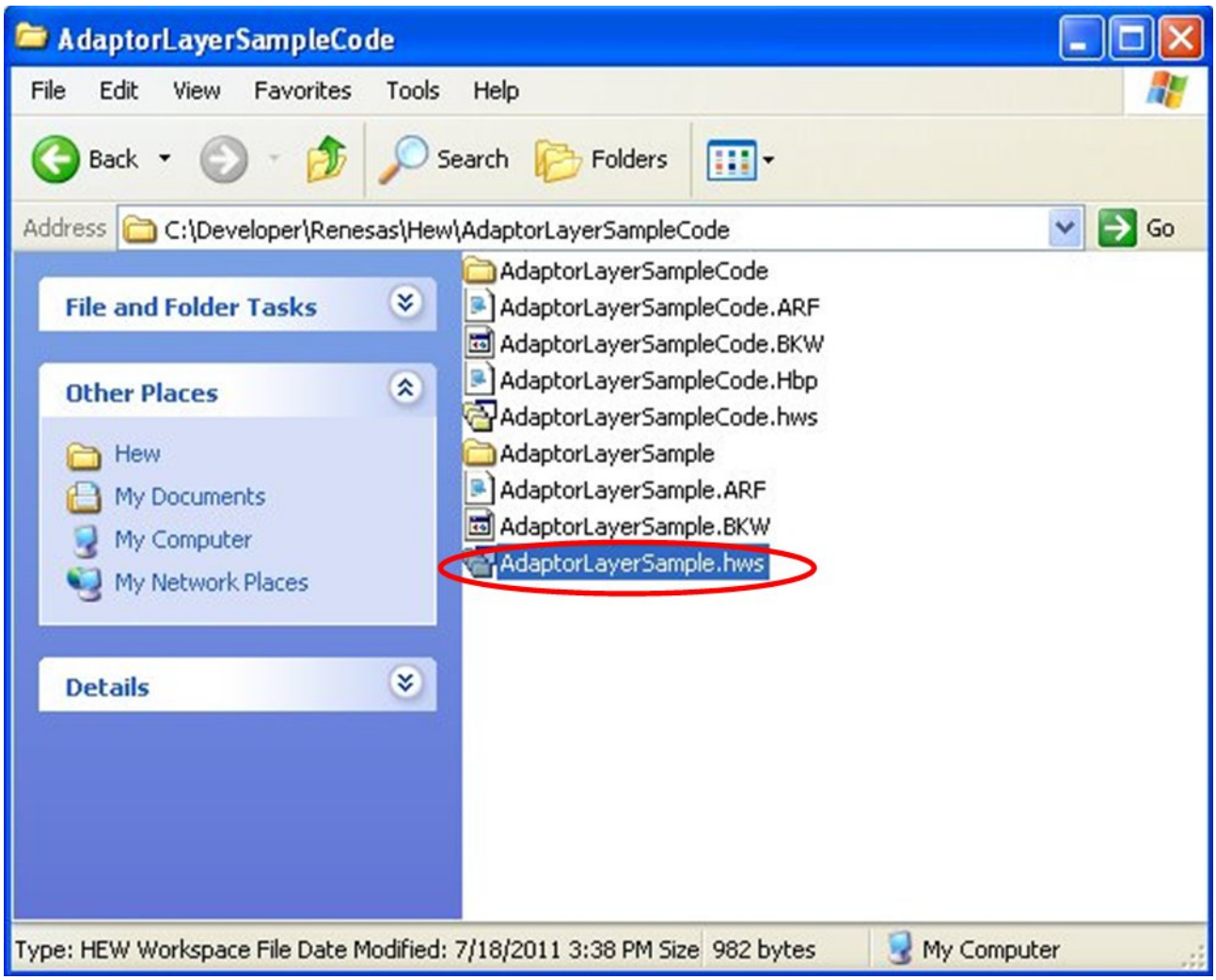
¥AdaptorLayerSample から¥AdaptorLayerSampleCode にワークスペースファイルをコピーします。



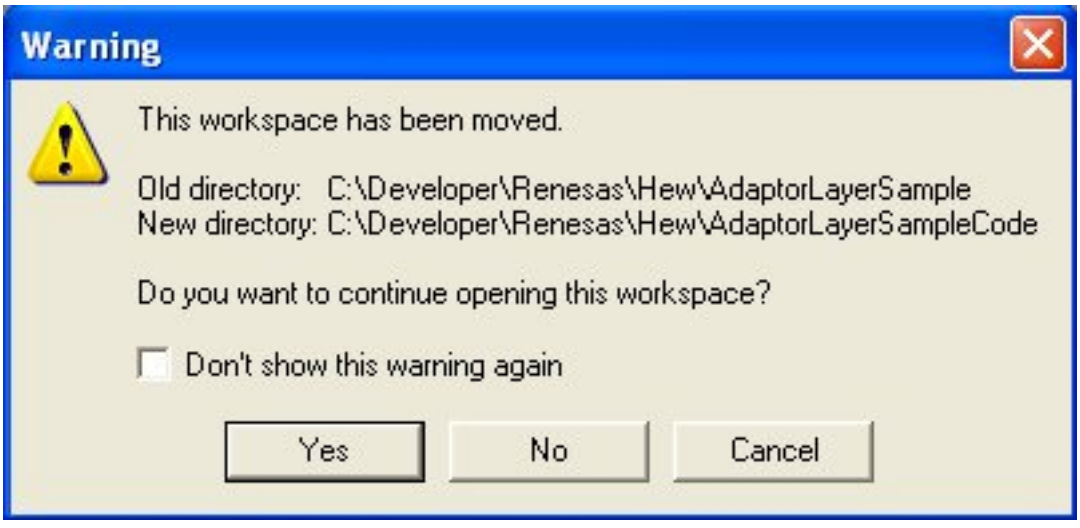
以下に示すとおり、次のディレクトリにペーストします。



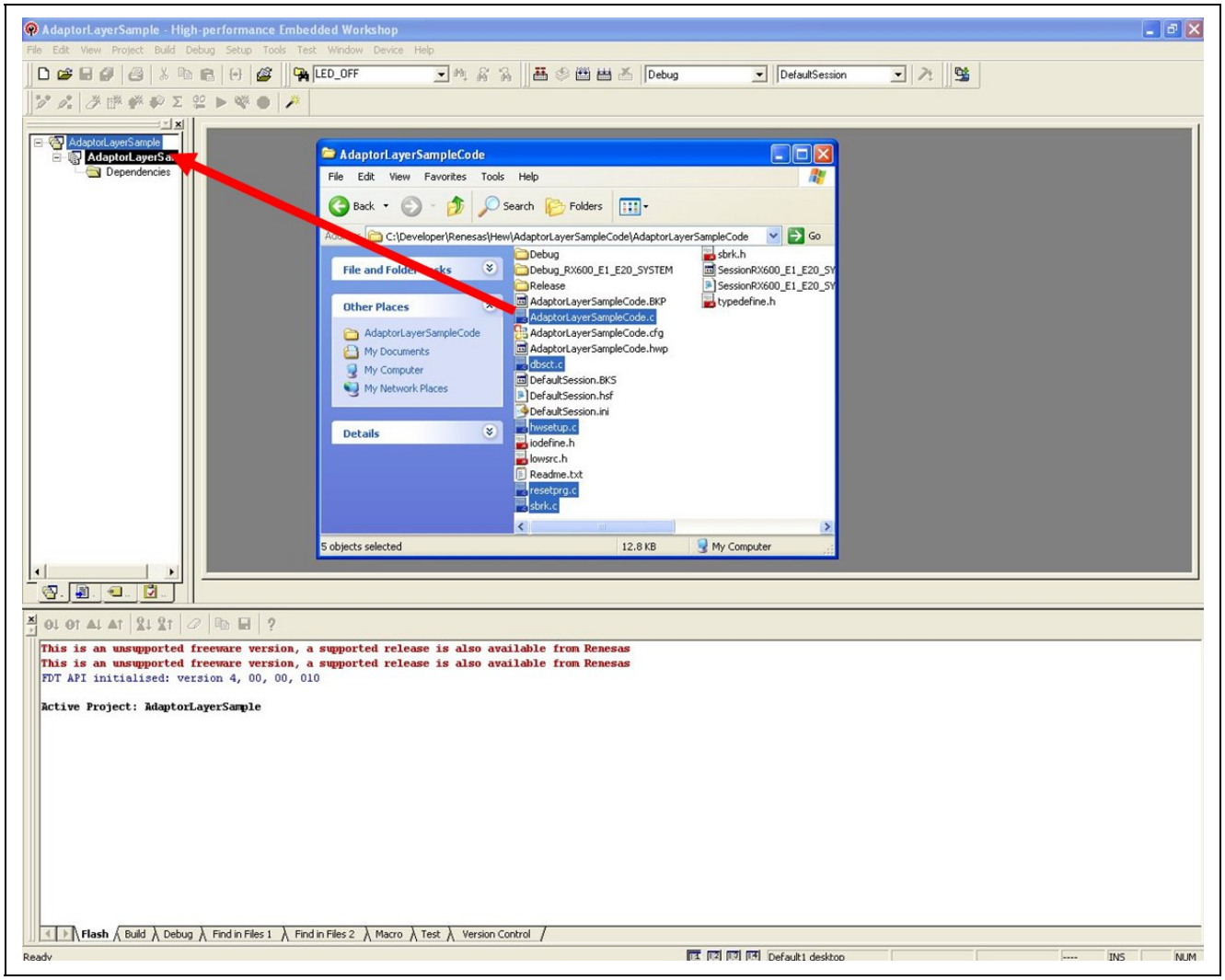
以下に示すとおり、新しいアダプタレイヤーのワークスペースを再度開きます。



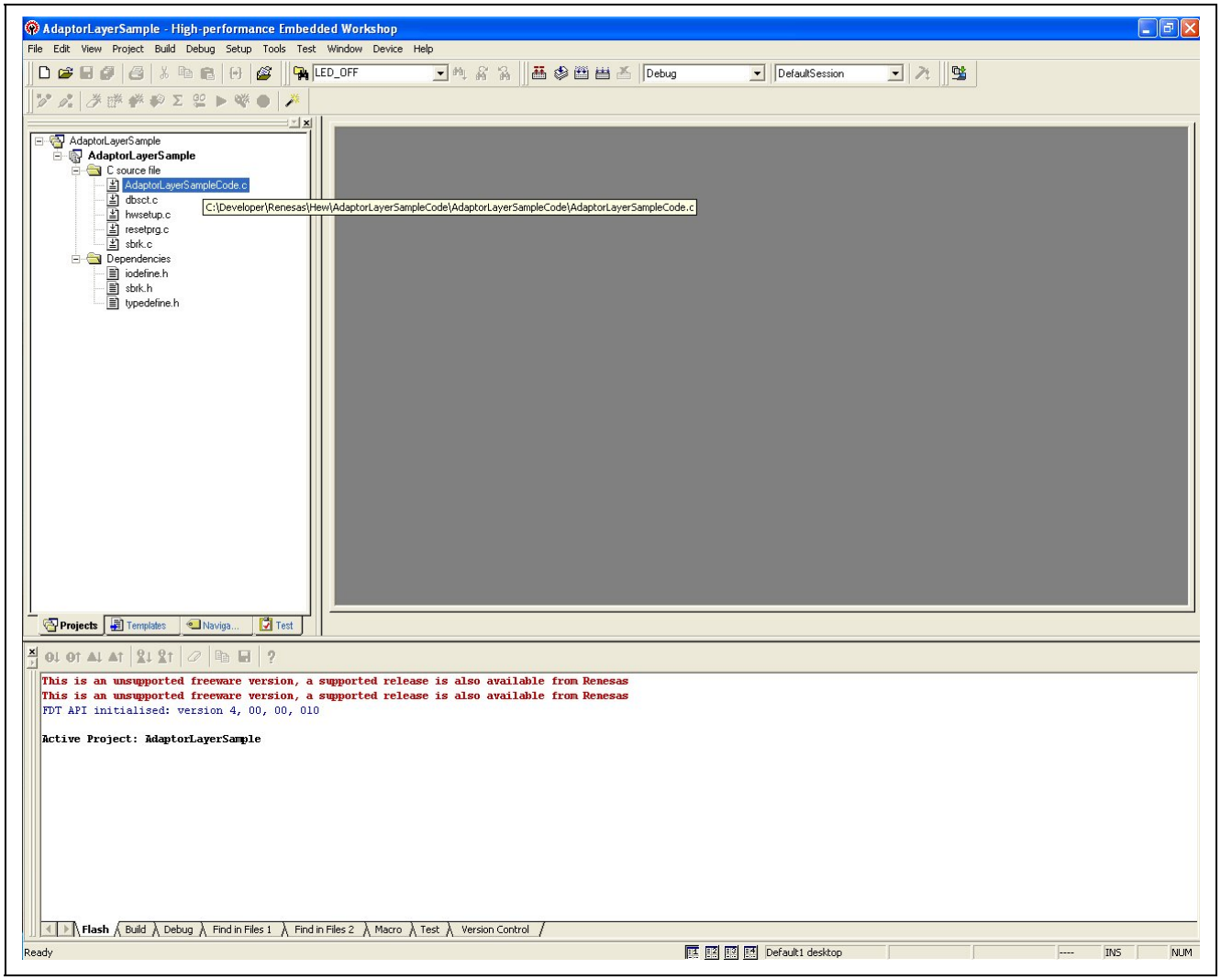
[Yes]をクリックしてワークスペースへの変更を受け入れます。



フォルダ¥AdaptorLayerSampleCode¥AdaptorLayerSampleCode を開き、以下に示すとおり、ファイルをドラッグ＆ドロップします。



AdaptorLayerSampleCode.c を開きます。



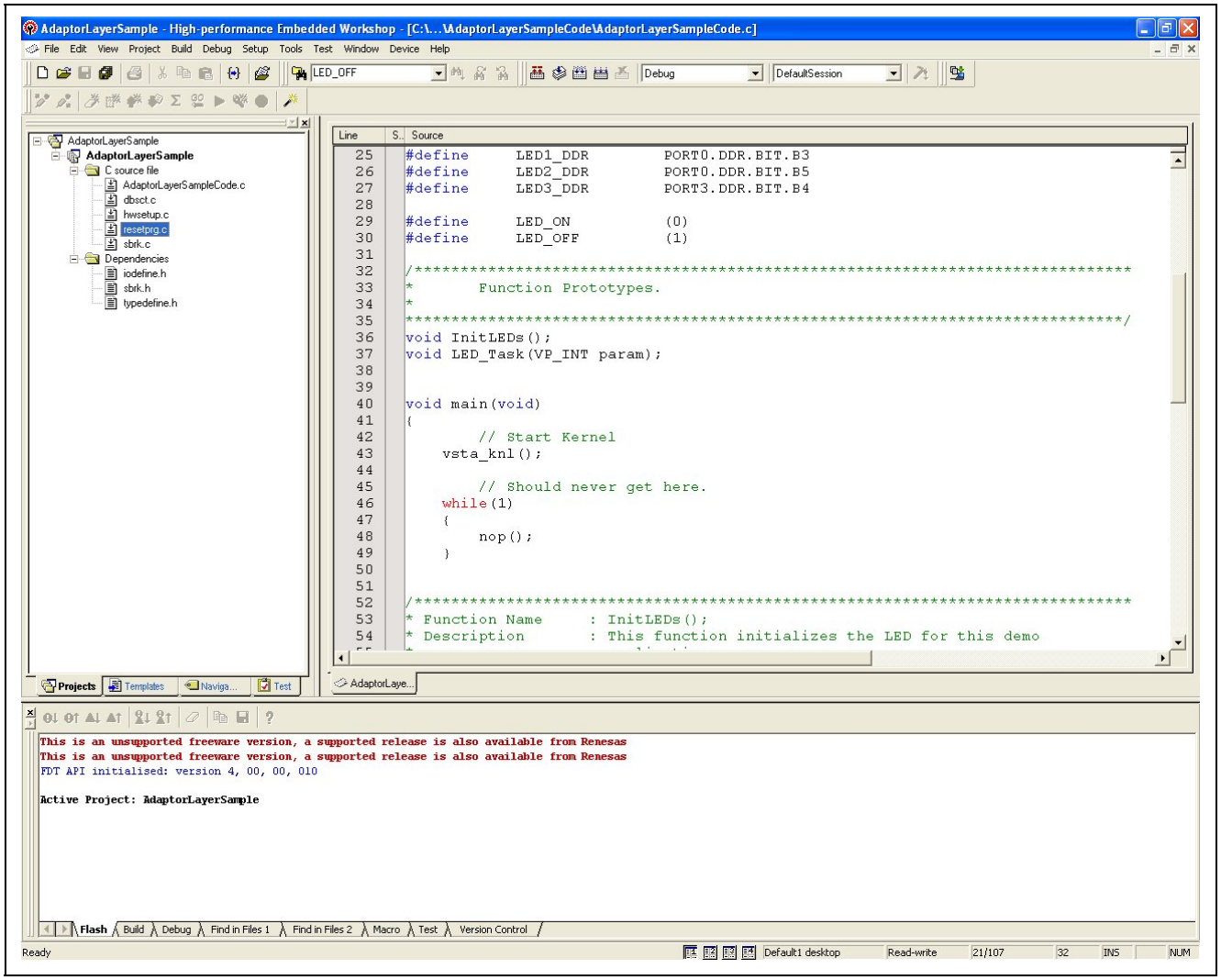
ファイルの内容を以下のコードに置き換えます。

```
#include <machine.h>
#include <stdio.h>
#if defined (__ADAPTOR_RI_600__)
  #include "kernel.h"
  #include "kernel_id.h"
#elif defined (__ADAPTOR_FREE_RTOS__)
  #include "r_FreeRTOSAdaptor.h"
#endif

#include "iodefine.h"

/***********************************************************************
*  Defines.
*
***********************************************************************/

/* LEDs */
#define     LED0        PORT0.DR.BIT.B2
#define     LED1        PORT0.DR.BIT.B3
#define     LED2        PORT0.DR.BIT.B5
#define     LED3        PORT3.DR.BIT.B4
#define     LED0_DDR        PORT0.DDR.BIT.B2
#define     LED1_DDR        PORT0.DDR.BIT.B3
#define     LED2_DDR        PORT0.DDR.BIT.B5
#define     LED3_DDR        PORT3.DDR.BIT.B4

#define     LED_ON      (0)
#define     LED_OFF        (1)

/***********************************************************************
*  Function Prototypes.
*
***********************************************************************/
void InitLEDs();
void LED_Task(VP_INT param);


void main(void)
{
   // Start Kernel
  vsta_knl();

   // Should never get here.
  while(1)
  {
   nop();
  }
}
```

```
/*****************************************************************************
* Function Name   : InitLEDs();
* Description     : This function initializes the LED for this demo
*             application.
* Argument        : None.
* Return Value    : None.
*****************************************************************************/
void InitLEDs(){
  LED0_DDR = 1;
  LED1_DDR = 1;
  LED2_DDR = 1;
  LED3_DDR = 1;

  LED0 = LED_OFF;
  LED1 = LED_OFF;
  LED2 = LED_OFF;
  LED3 = LED_OFF;
}

/*****************************************************************************
* Function Name   : LED_Task();
* Description     : This is the LED Task that will execute the Application
*             code.
* Argument        : None.
* Return Value    : None.
*****************************************************************************/
void LED_Task(VP_INT param)
{
   // Local Variables
  static unsigned char toggle = 0;

   InitLEDs();

  for (;;){

   switch (toggle){
     case 0:
         LED0 = ~LED0;
       break;
     case 1:
         LED1 = ~LED1;
       break;
     case 2:
         LED2 = ~LED2;
       break;
     case 3:
        LED3 = ~LED3;
       break;
   }

   toggle = (toggle == 3) ? 0 : ++toggle;

   dly_tsk(500);
  }
}
```

違いは追加される void main(void)関数のみであることに注意してください。

resetprg.c を開きます。



resetprg.c のコード全体を以下に置き換えます。

```
#include    <machine.h>
#include    <_h_c_lib.h>
//#include  <stddef.h>          // Remove the comment when you use errno
//#include  <stdlib.h>          // Remove the comment when you use rand()
#include    "typedefine.h"

#if defined (__ADAPTOR_RI_600__)
  #include "kernel.h"
  #include "kernel_id.h"
#elif defined (__ADAPTOR_FREE_RTOS__)
  #include "r_FreeRTOSAdaptor.h"
#endif

#pragma stacksize su=0x300
#pragma stacksize si=0x100

#if defined (__ADAPTOR_RI_600__)
  #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
  #include    "ri_cmt.h"                  // Generated by cfg600
  #endif
#endif   /* __ADAPTOR_RI_600__ */

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 );
#ifdef __cplusplus
}
#endif

//#ifdef __cplusplus          // Use SIM I/O
//extern "C" {
//#endif
//extern void _INIT_IOLIB(void);
//extern void _CLOSEALL(void);
//#ifdef __cplusplus
//}
//#endif

#define FPSW_init 0x00000100

//extern void srand(_UINT);   // Remove the comment when you use rand()
//extern _SBYTE *_s1ptr;         // Remove the comment when you use strtok()

#ifdef __cplusplus         // Use Hardware Setup
extern "C" {
#endif
extern void HardwareSetup(void);
#ifdef __cplusplus
}
#endif
```

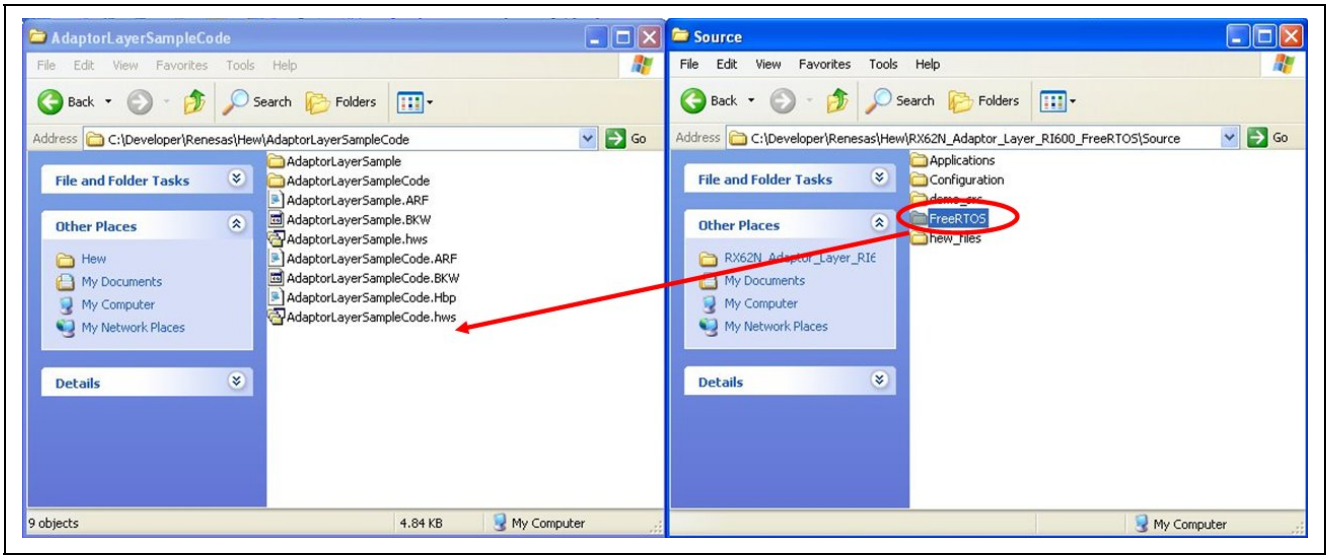
```
//#ifdef __cplusplus       // Remove the comment when you use global class object
//extern "C" {             // Sections C$INIT and C$END will be generated
//#endif
//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//#endif

#pragma section ResetPRG

#pragma entry PowerON_Reset_PC

////////////////////////////////////////////////////////////////////////////
// Power-on Reset Program
////////////////////////////////////////////////////////////////////////////
void PowerON_Reset_PC(void)
{
  #if defined (__ADAPTOR_FREE_RTOS__)
    set_intb((unsigned long)__sectop("C$VECT"));
  #endif

  set_fpsw(FPSW_init);

     _INITSCT();

//    _INIT_IOLIB();                  // Use SIM I/O

//  errno=0;                         // Remove the comment when you use errno
//  srand((_UINT)1);                   // Remove the comment when you use rand()
//  _s1ptr=NULL;                     // Remove the comment when you use strtok()

    HardwareSetup();                 // Use Hardware Setup

//  set_fintv(<handler address>);    // Initialize FINTV register


#if defined (__ADAPTOR_RI_600__)
  #if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
      _RI_init_cmt();                  // Initialize CMT for RI600/4
                                     // Do comment-out when clock.timer is either
NOTIMER or OTHER.
  #endif
#endif  /* __ADAPTOR_RI_600__*/


    nop();

//  _CALL_INIT();                 // Remove the comment when you use global class
object

    //vsta_knl();                    // Start RI600/4
                                   // Never return from vsta_knl
  main();

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

//  _CALL_END();                  // Remove the comment when you use global class
object
```

```
    brk();

}
/*****************************************************************************
*  Compilation differences between RI600 and FreeRTOS.
*
*  The following sections are required for RI600 to function.
*****************************************************************************/

////////////////////////////////////////////////////////////////////////////
// System-down routine for RI600/4
////////////////////////////////////////////////////////////////////////////
#if defined (__ADAPTOR_RI_600__)
  #pragma section P PRI_KERNEL
  #pragma section B BRI_RAM

struct SYSDWN_INF{
  W type;
  VW  inf1;
  VW  inf2;
  VW  inf3;
};

volatile struct SYSDWN_INF _RI_sysdwn_inf;

void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 )
{
  // Now PSW.I=0 (all interrupts are masked.)

  _RI_sysdwn_inf.type = type;
  _RI_sysdwn_inf.inf1 = inf1;
  _RI_sysdwn_inf.inf2 = inf2;
  _RI_sysdwn_inf.inf3 = inf3;

    while(1)
    ;
}
#endif

////////////////////////////////////////////////////////////////////////////
// RI600/4 system data
////////////////////////////////////////////////////////////////////////////
#if defined (__ADAPTOR_RI_600__)
  #include "kernel_ram.h"     // generated by cfg600
  #include "kernel_rom.h"     // generated by cfg600
#endif
```

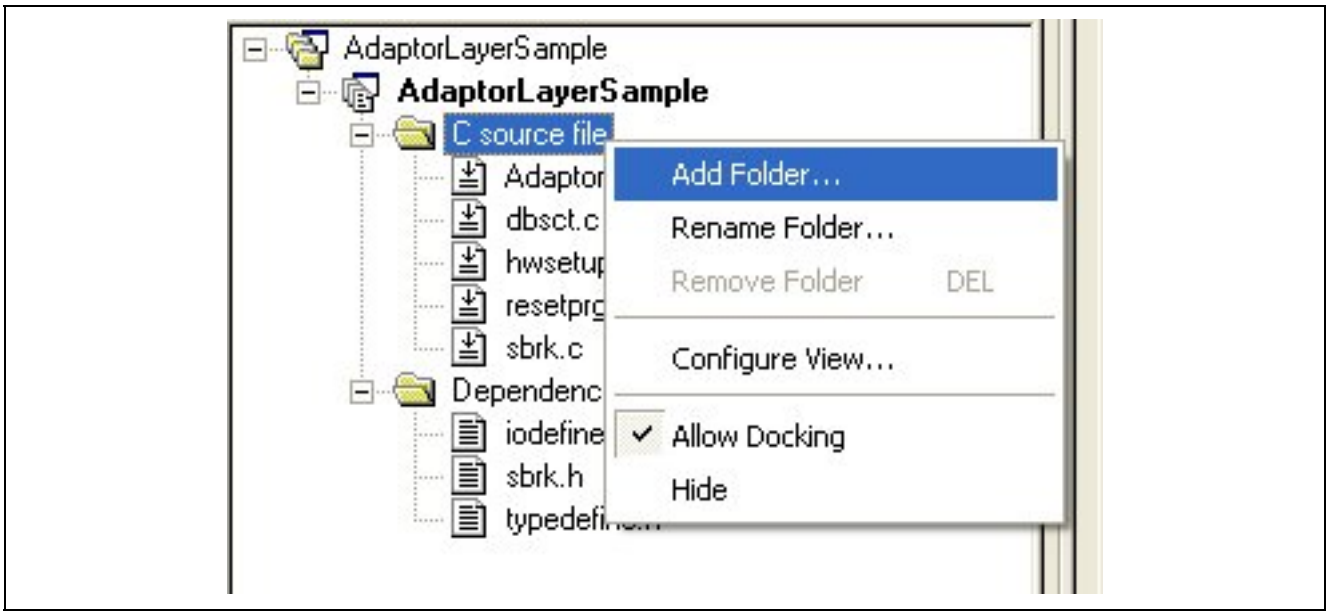
RI-600 ワークスペースジェネレータが生成した部分は__ADAPTOR_RI_600__および__ADAPTOR_FREE_RTOS__の条件コンパイルチェックによってマスキングされていることに注意してください。

2.9　FreeRTOS™ファイルのインポート

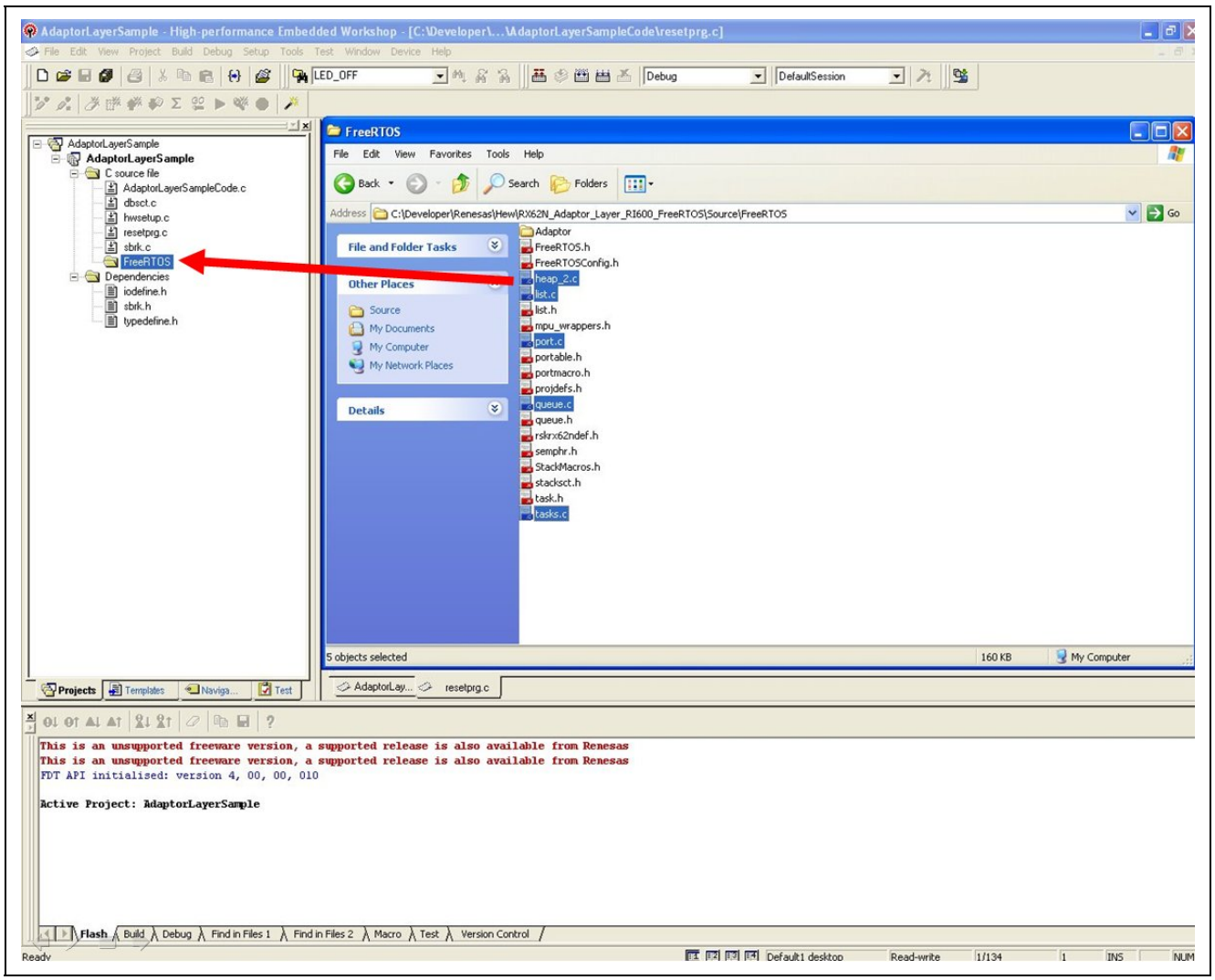
以下に示すとおり、このマニュアルに付属の FreeRTOS™ソースコードフォルダ全体をワークスペースフォルダにコピーします。



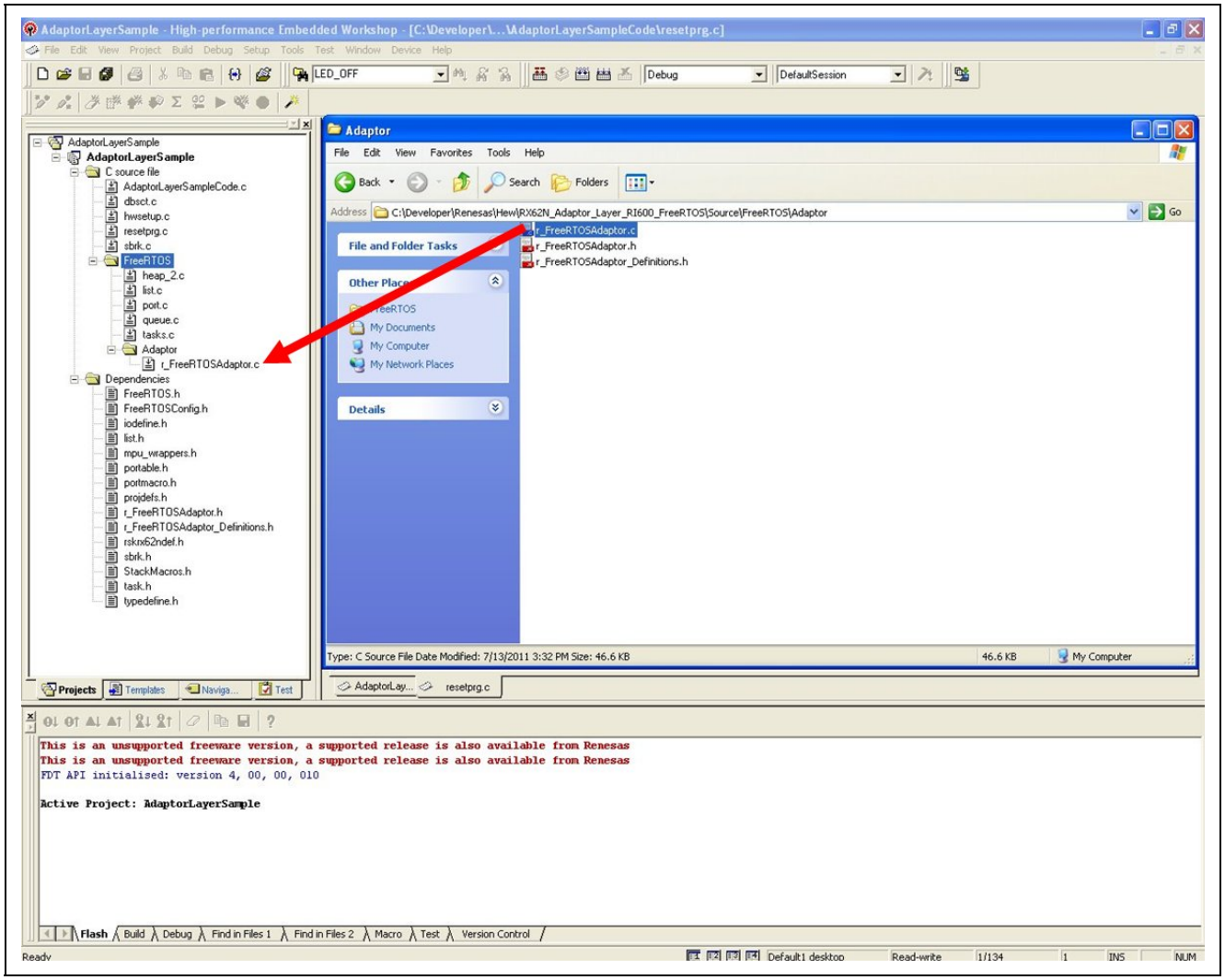
FreeRTOS™のワークスペースにディレクトリを作成します。



以下のように、FreeRTOS™ファイルをドラッグ＆ドロップします。

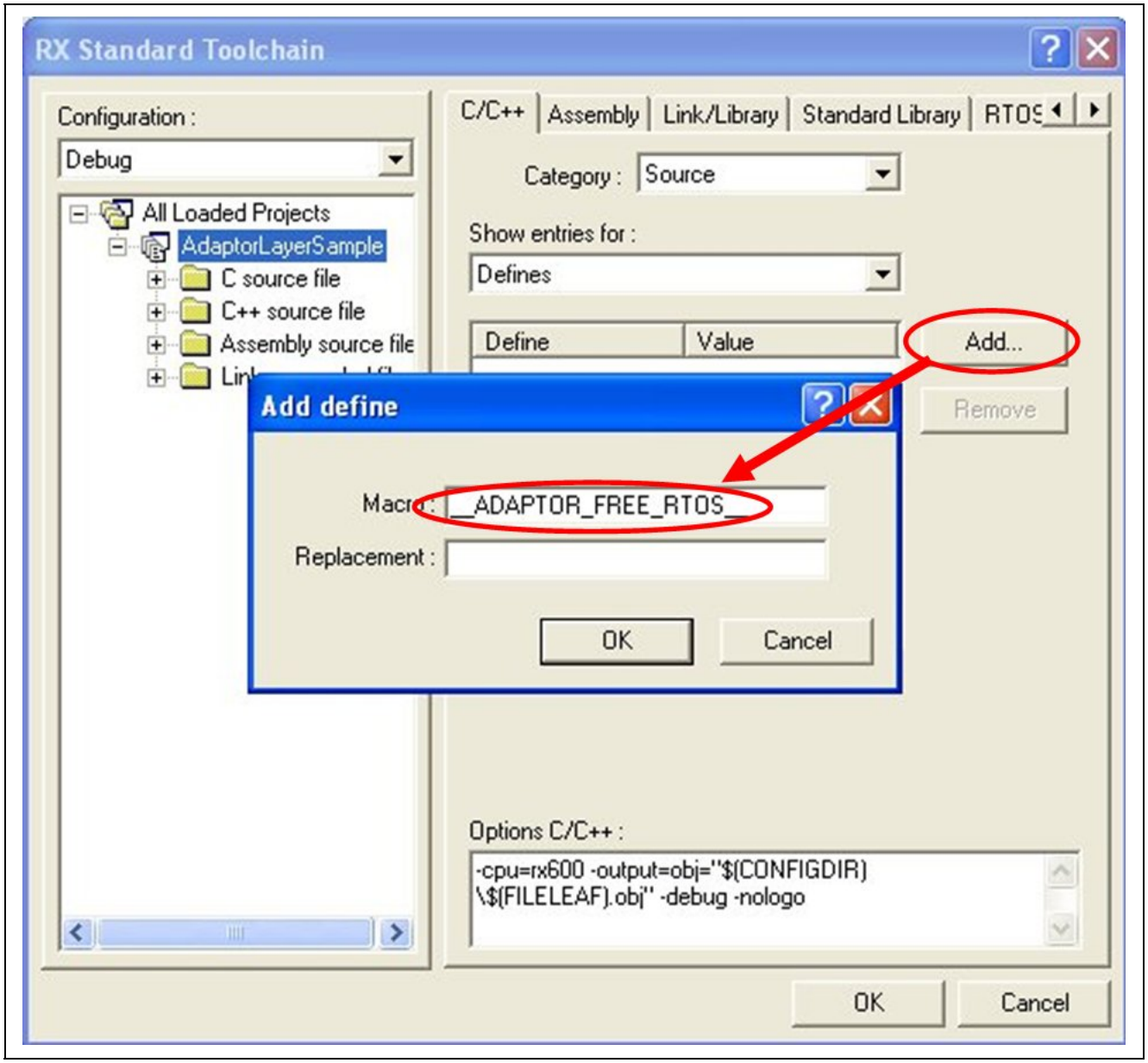


ワークスペースの FreeRTOS™フォルダの下に Adaptor というフォルダを作成します。以下のように、必要な Adaptor C コードファイルをドラッグ&ドロップします。



2.10　アダプタレイヤーの使用の定義

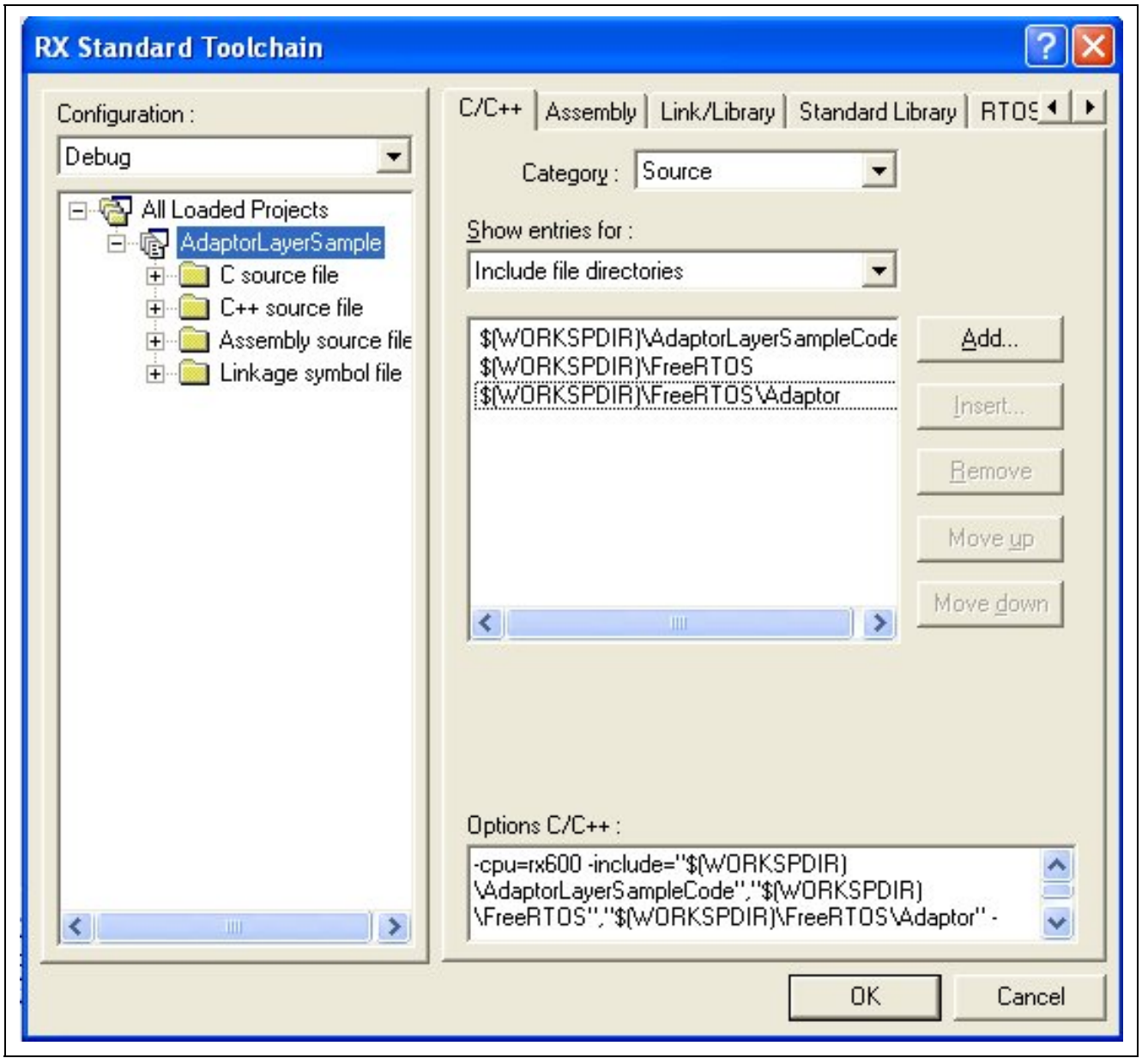
[Build]→[RX Standard Toolchain]に進み、C/C++タブで、[Show entries for:]に[Defines]を選択します。以下に示すとおり、__ADAPTOR_FREE_RTOS__を追加します。



2.11　ヘッダーファイルのインクルード

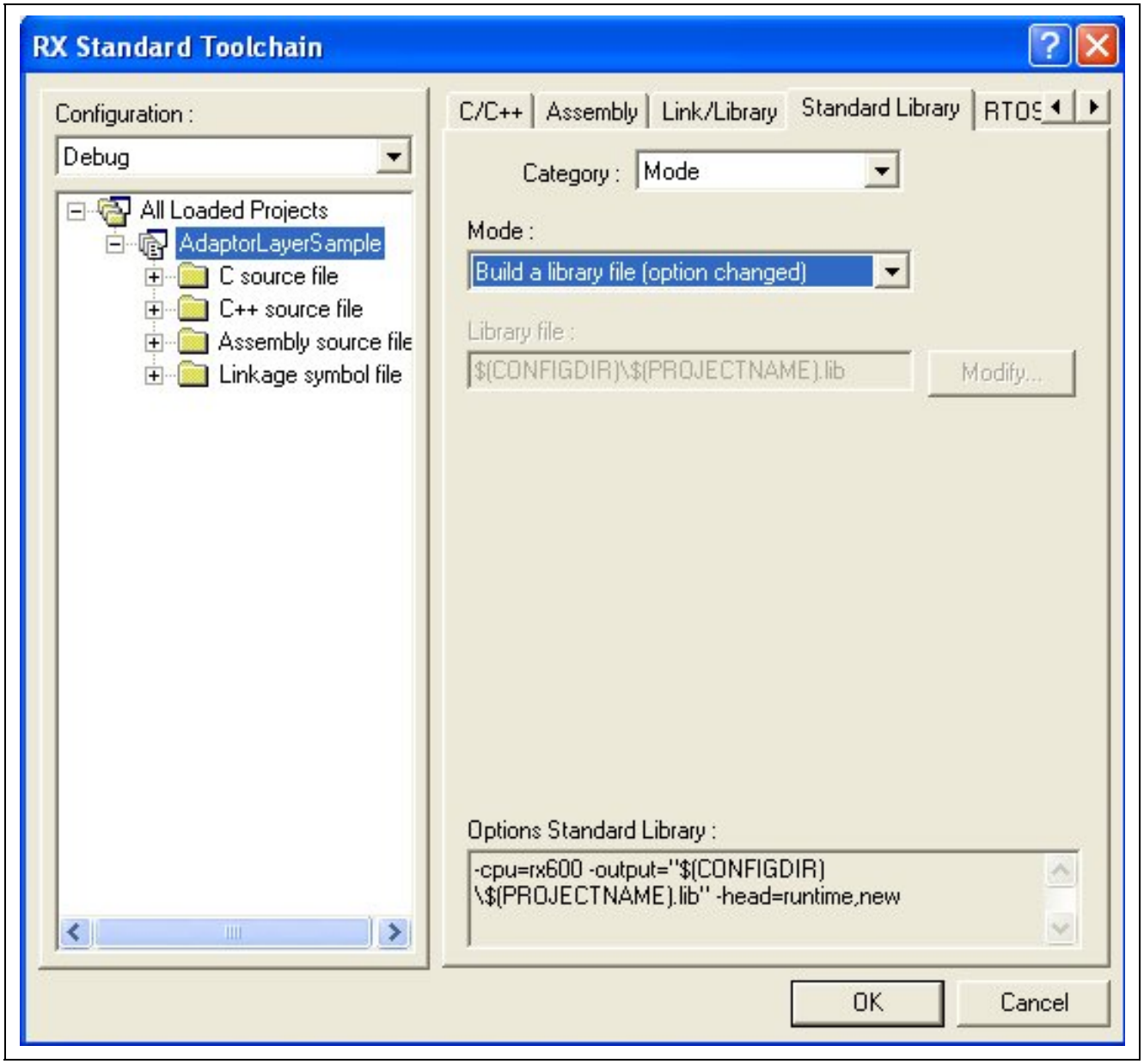
[Build]→[RX Standard Toolchain]に進み、C/C++タブで、[Show entries for:]に[Include file directories]を選択します。以下に示すとおり、次のディレクトリを追加します。

- $(WORKSPDIR)¥AdaptorLayerSampleCode
- $(WORKSPDIR)¥FreeRTOS
- $(WORKSPDIR)¥FreeRTOS¥Adaptor



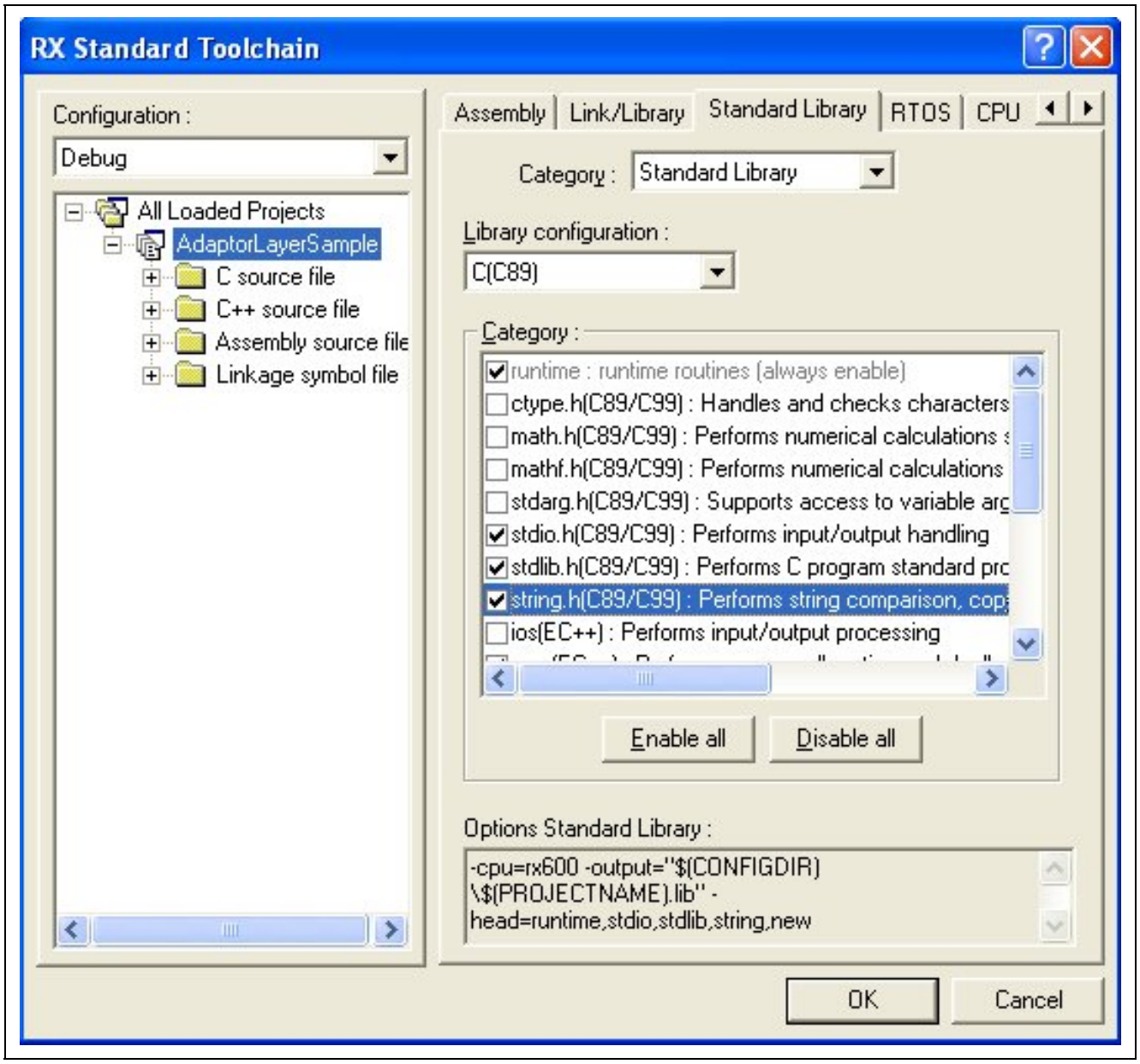
2.12　ライブラリファイルの設定

[Build]→[RX Standard Toolchain]に進み、[Standard Library]タブで、[Category:]に[Mode]を選択し、ドロップダウンメニューから[Mode:]に以下に示す値を設定します。



[Build]→[RX Standard Toolchain]に進み、[Standard Library]タブで、[Category:]に[Standard Library]を選択し、次のヘッダーファイルを選択してビルドします。

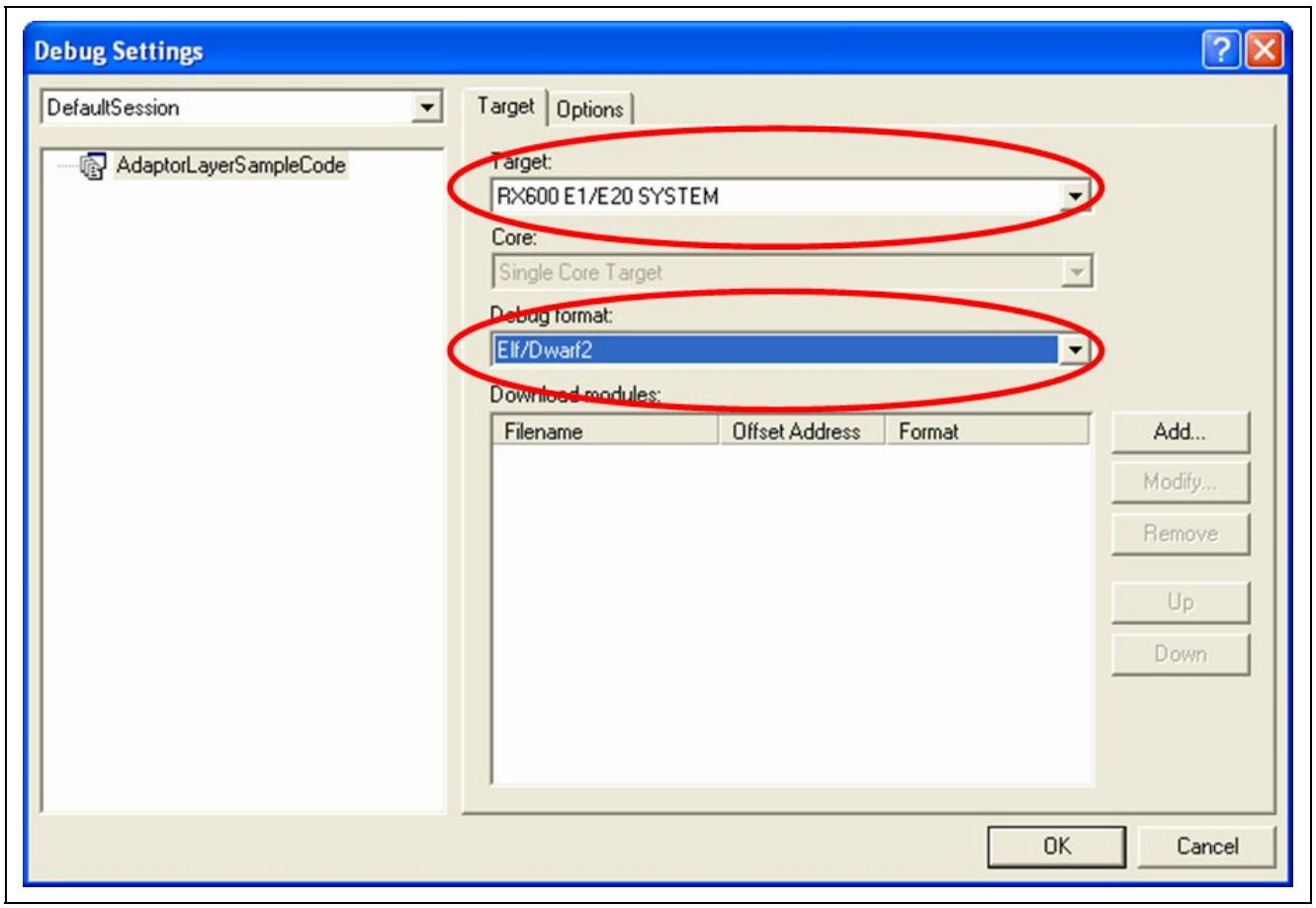
- stdio.h
- stdlib.h
- string.h



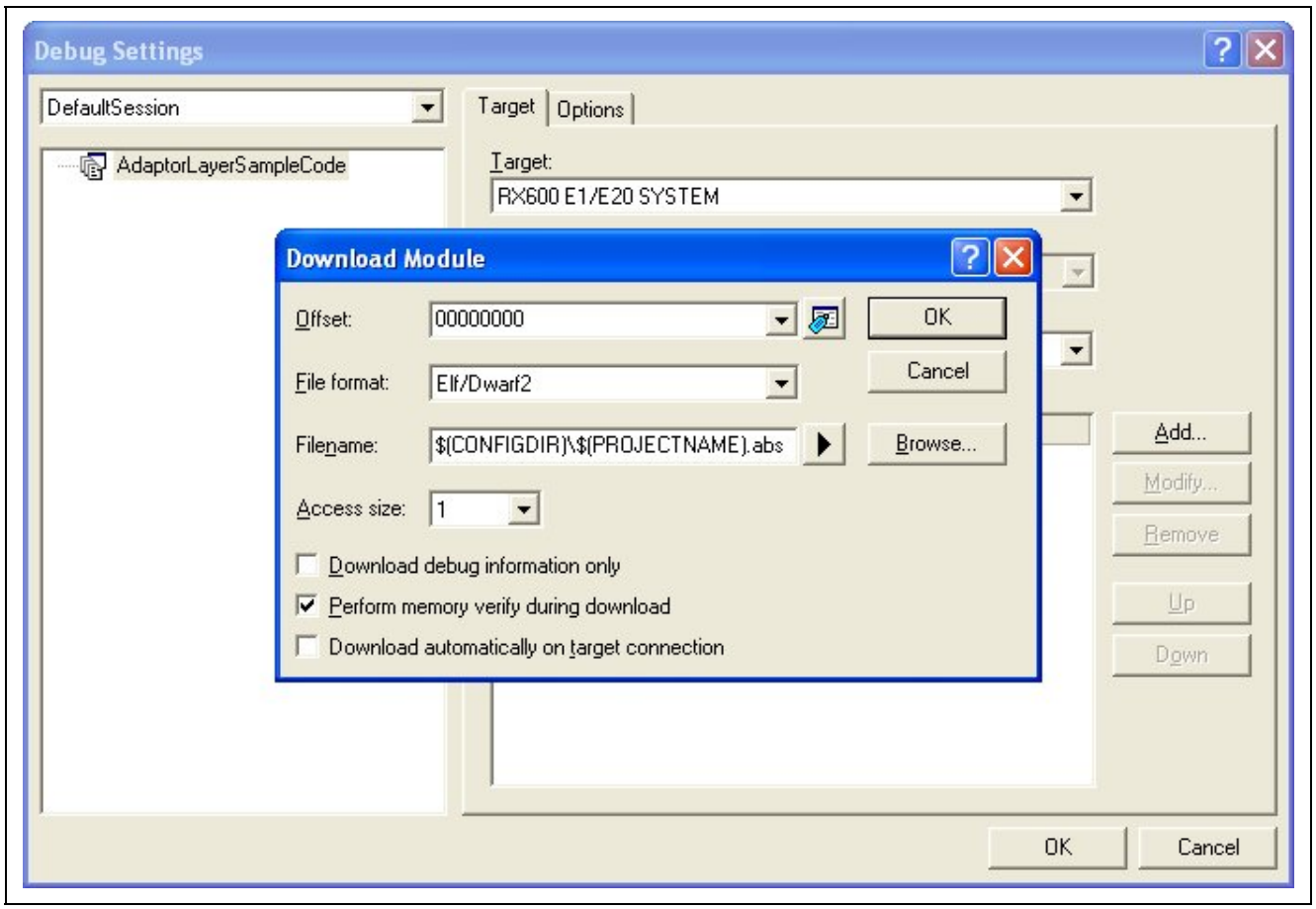
2.13　RX62N ボードへの接続

以下に示すとおり、エミュレータを追加して RX62N RSK ボードに接続します。

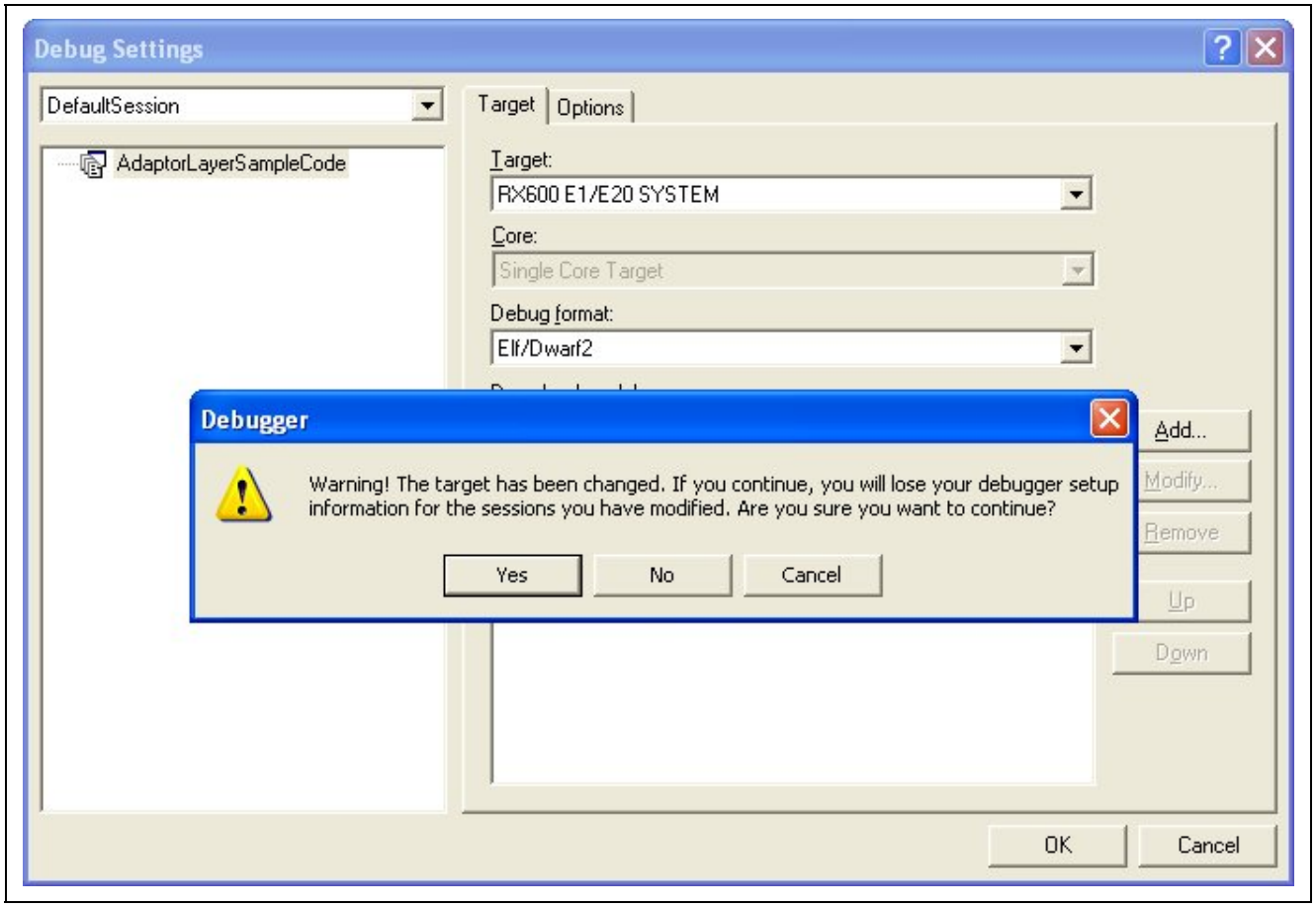
以下に示すように、次のデバッグ設定を追加します。



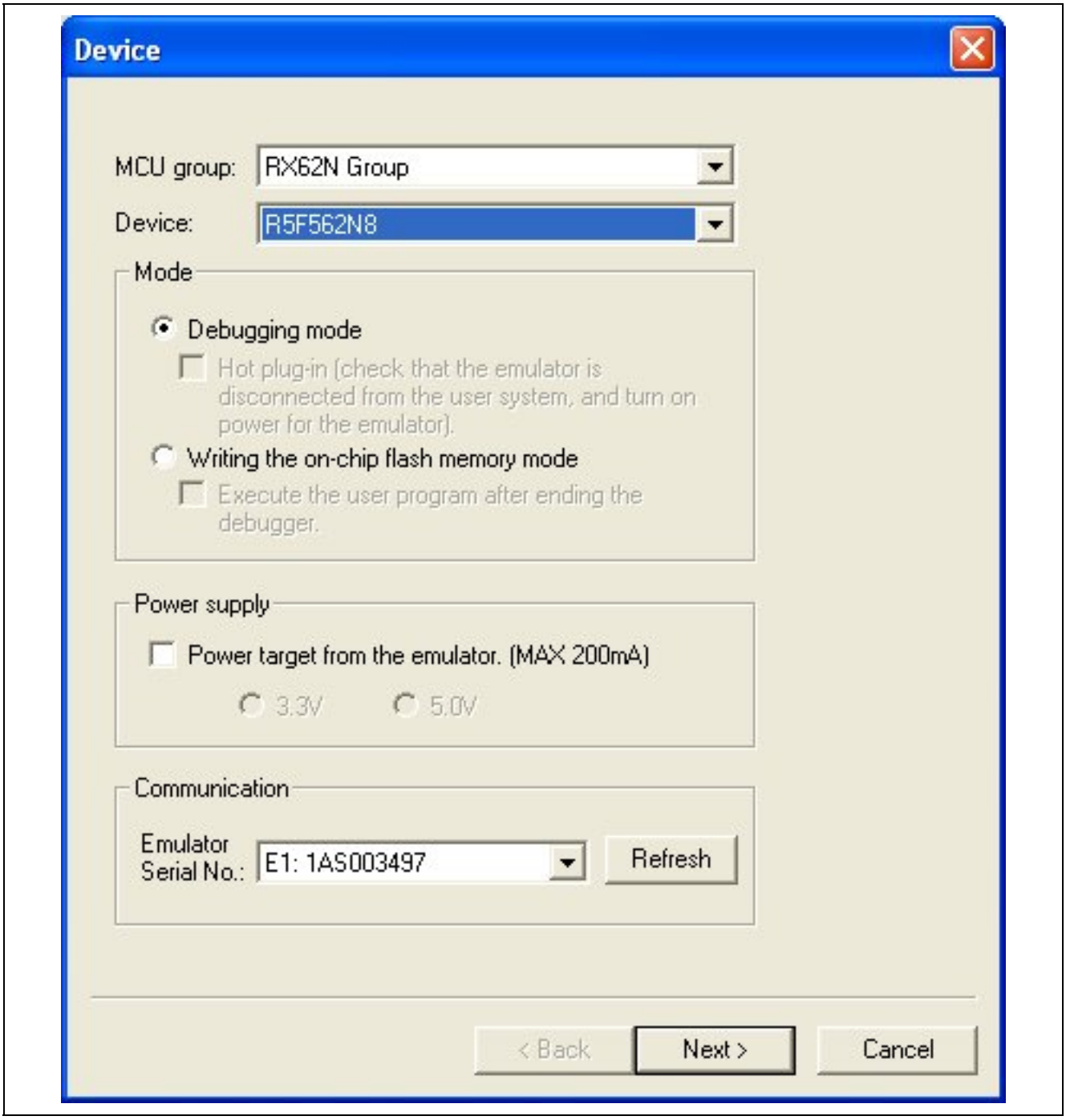
[Add]をクリックして以下のコンフィグレーションを追加します。



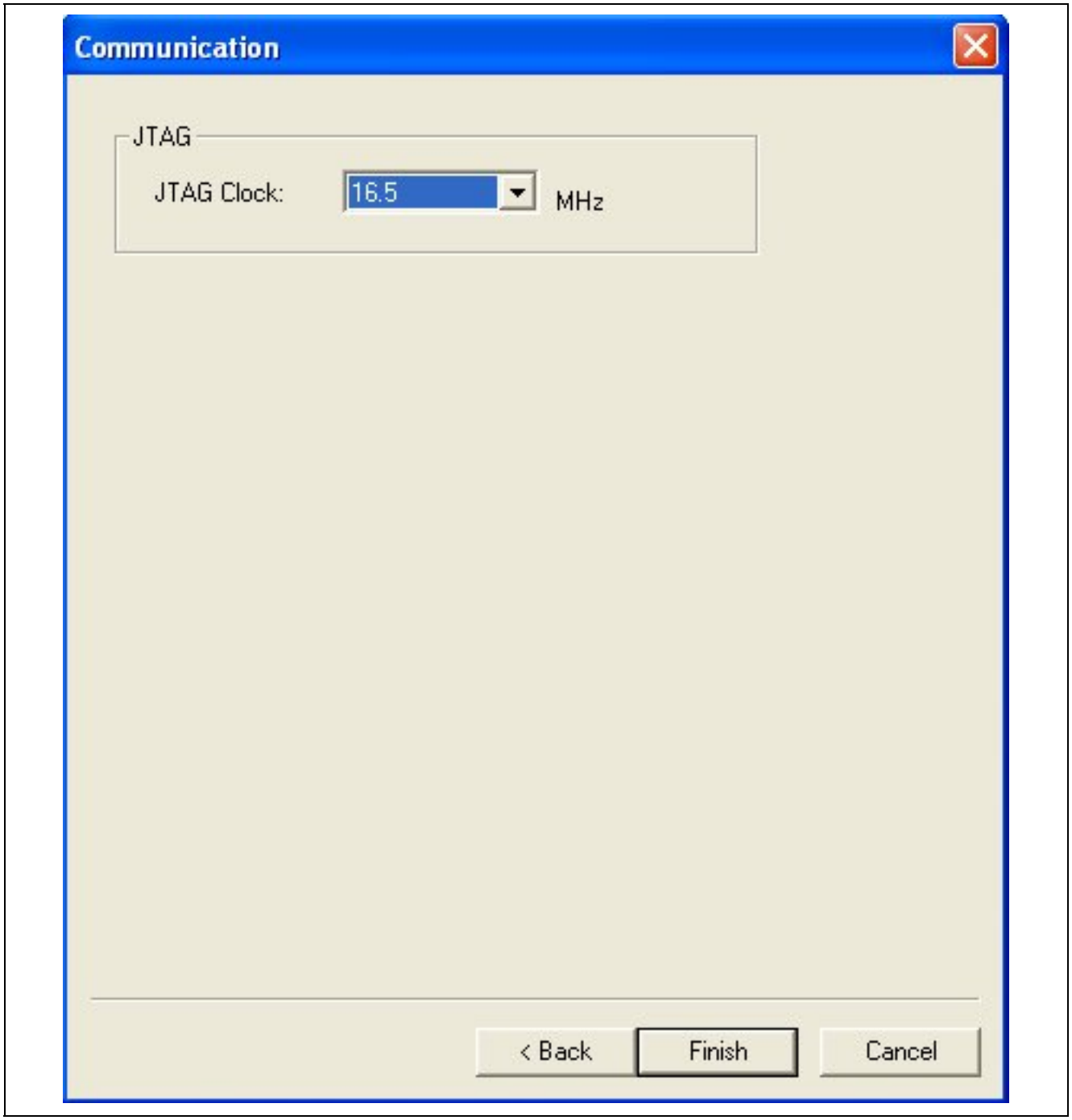
[OK]をクリックしてデフォルト設定を受け入れ、デバッガ警告パネルが表示されたときに[Yes]をクリックします。



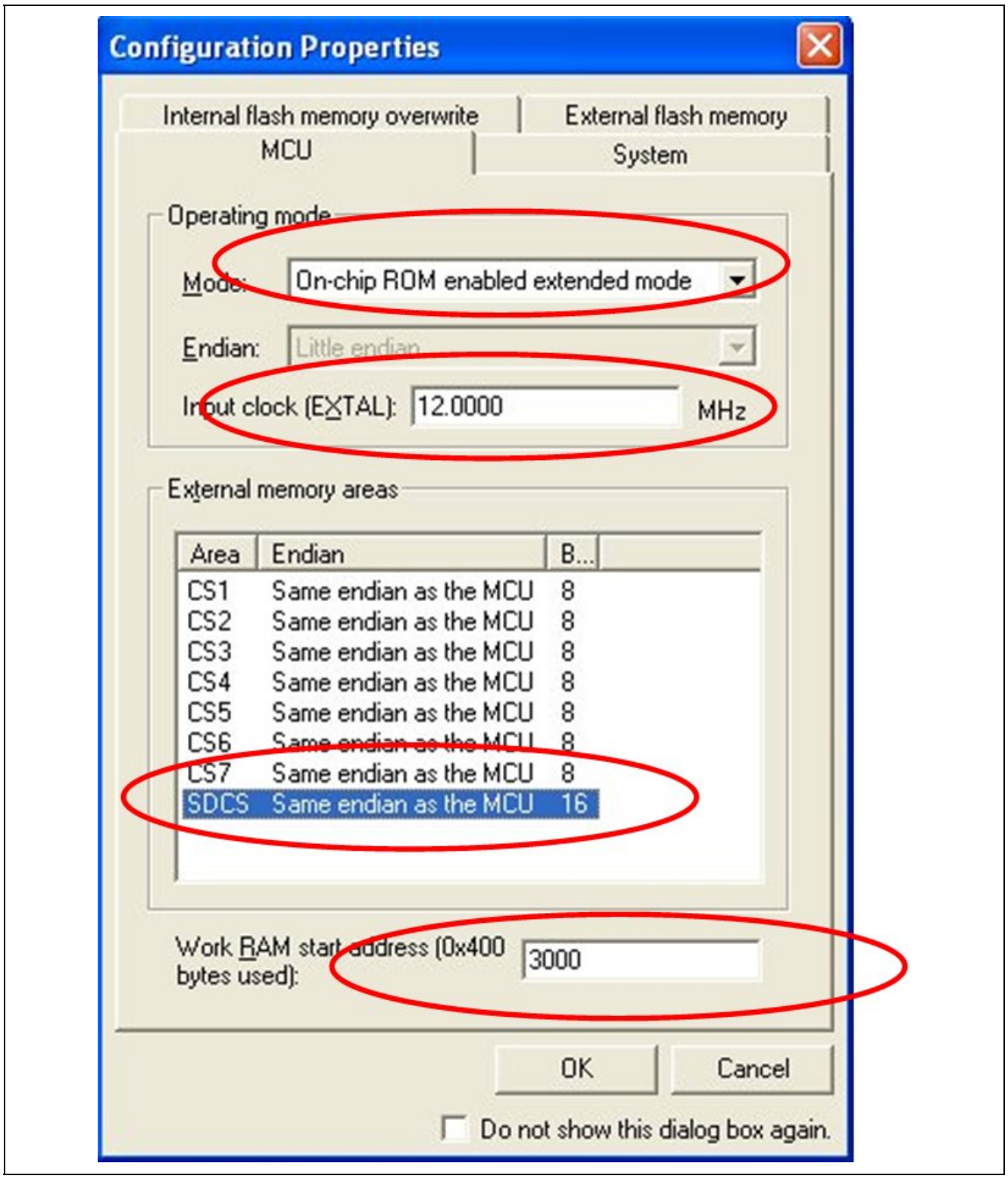
[Device]ウィンドウで以下を選択します。



ボードに電源が投入されていることを確認してから[Next >]をクリックします。



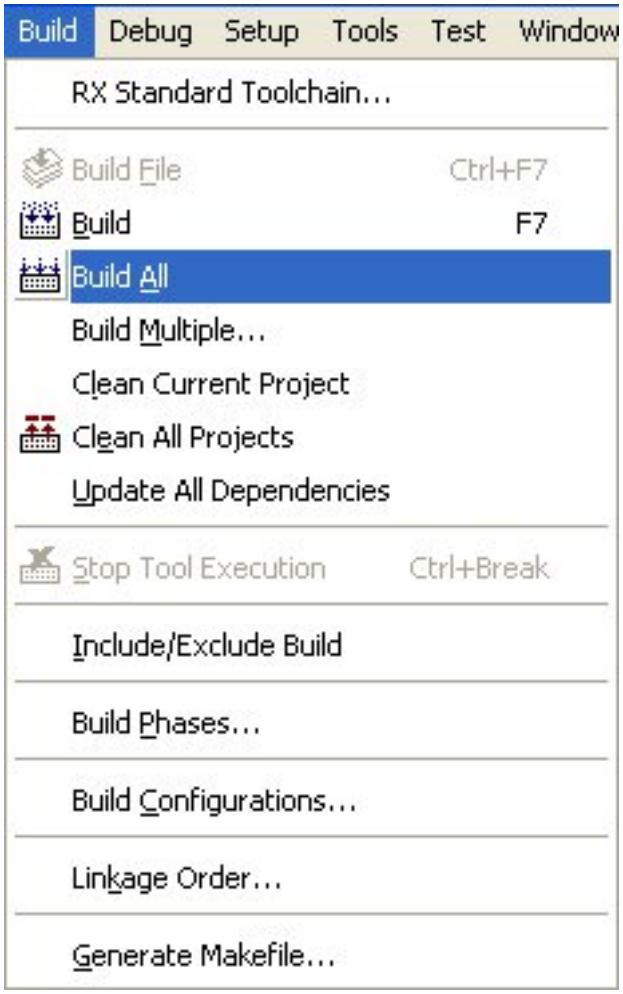
[Finish]をクリックしてデバイスへの接続を開始します。次のウィンドウで以下の設定を選択します。



[OK]をクリックしてデバイスへの接続を進めます。

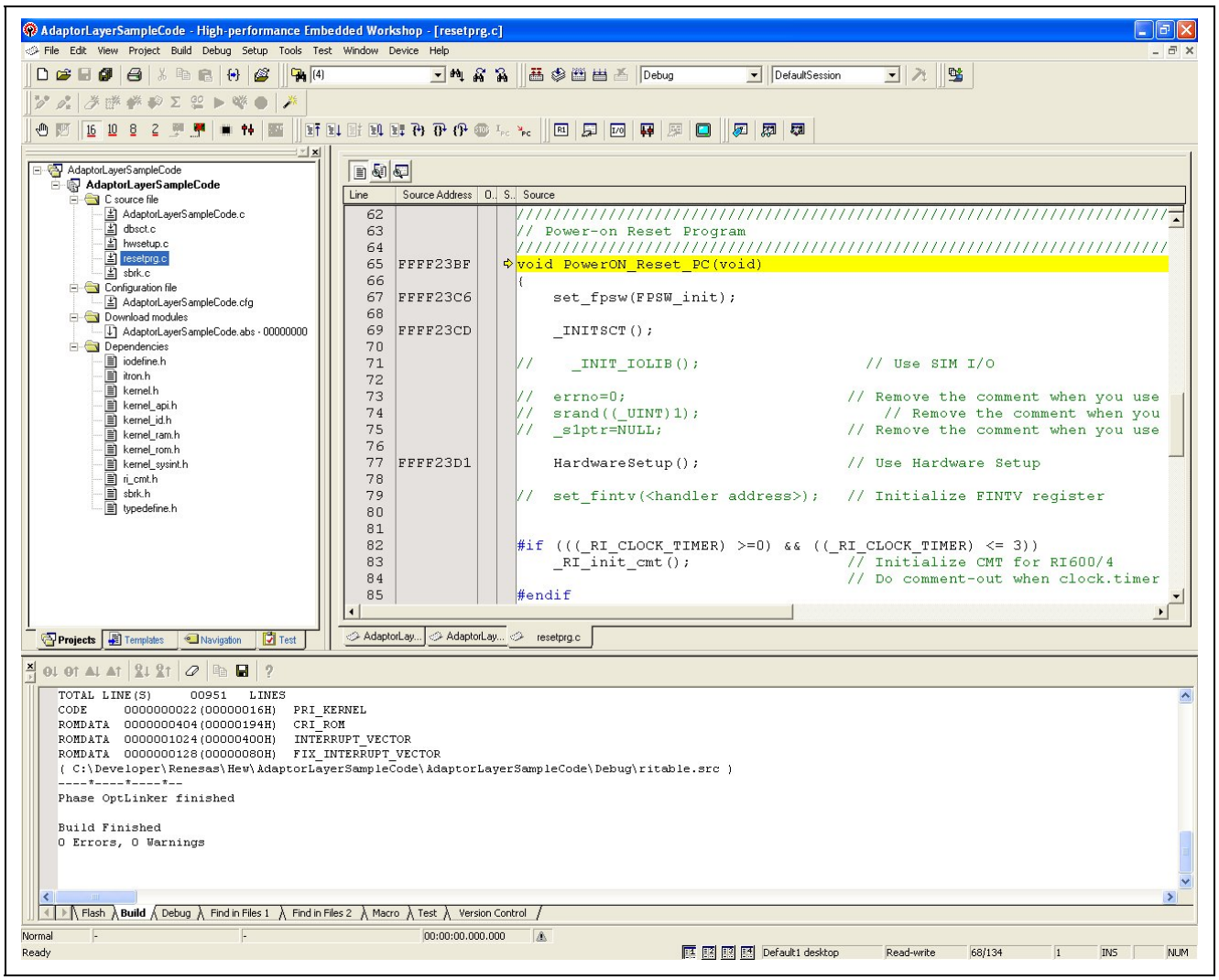
2.14　LED 点滅サンプルのコンパイルとビルド

[Build]→[Build All]を選択してコンパイルを開始します。



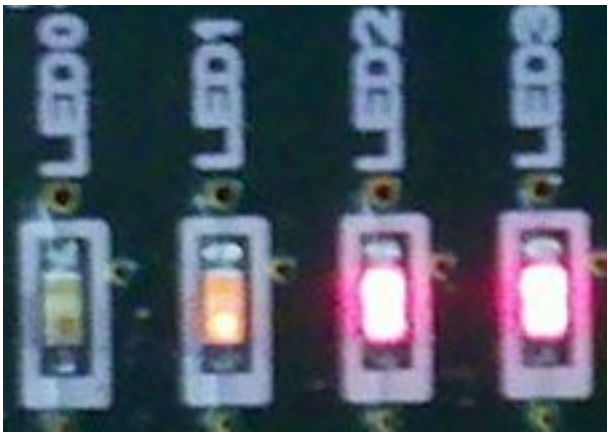
コンパイルが完了したら、[Yes to all]を選択し、ビルドしたモジュールをダウンロードします。

モジュールをダウンロードしたら、以下に示すとおり、プログラムは実行準備が完了です。



プログラムを実行するには、F5 を押すか、以下に示すとおり[Debug]→[Go]を選択します。

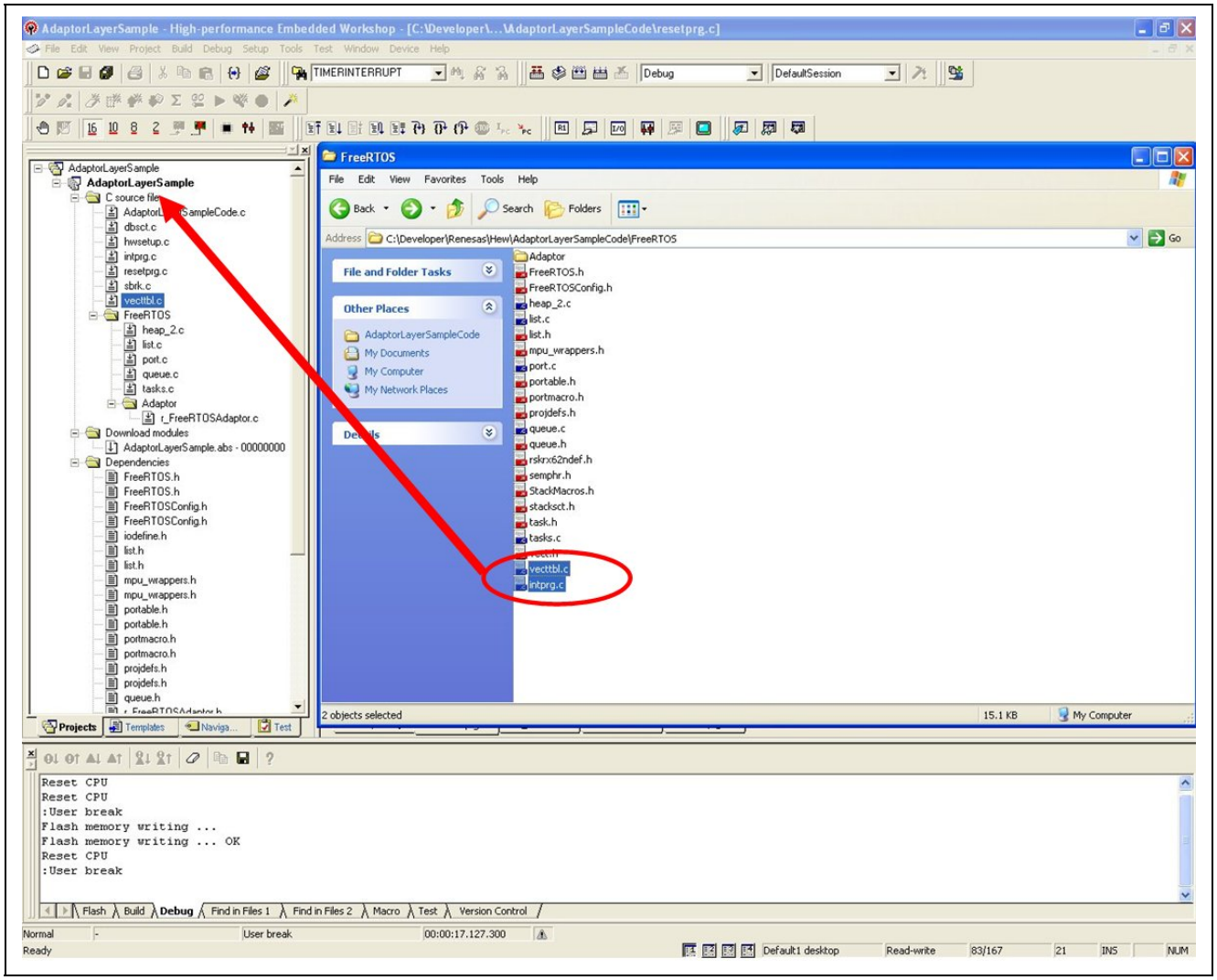
最終的に、RSK の LED 点滅は、次のようになります。



2.15　トラブルシューティング

デフォルトの vecttbl.c はプロジェクトのビルド時に使用していない場合があります。以下に示すとおり、ドラッグ&ドロップして次のファイルを追加します。

- vecttbl.c
- intprg.c



コンパイルしてビルドします。これですべてが正常に動作するはずです。

3. アダプタレイヤーの使用

3.1 タスクの作成

タスクは、以下の C 構造体によってアダプタレイヤーで管理されます。

```
/* Structure to keep track of Tasks in Adaptor Layer */
struct fr_TASK_BLOCK
{
  pdTASK_CODE     pvTaskCode;
  const portCHAR *       const pcName;
  unsigned portSHORT  usStackDepth;
  portBASE_TYPE       uxPriority;
  void             *pvParameters;
  xTaskHandle     xHandle;

    // Implement WAIT-SUSPEND State
  unsigned char       TaskState;
  signed char         SLP_WUP_COUNT;
  signed char         SUS_RSM_COUNT;
};
```

このサンプルは、アダプタレイヤーによる管理対象のタスクの作成を示します。

```
/* Task creation for FreeRTOS */
struct fr_TASK_BLOCK fr_TASKS[fr_MAX_TASKS+1] =
{
  {(pdTASK_CODE) NULL,
    "",
     0,
     0,
     NULL, NULL, 0, 0, 0 },  /* Offset 0; Not used. */
  {(pdTASK_CODE) fr_LED_Task,
    "LED_Task",
    128,
    3,
    NULL, NULL, DORMANT, 0, 0 },
};
```

3.2 メールボックスの作成

メールボックスは、以下の C 構造体によってアダプタレイヤーで管理されます。

```
/* Structure to keep track of MailBoxes in FreeRTOS */
struct fr_MAILBOX_BLOCK
{
  unsigned char    ucMailBoxID;
  xQueueHandle  pvMailBox;
};
```

このサンプルは、アダプタレイヤーによる管理対象のメールボックスの作成を示します。

```
/* MailBox creation for FreeRTOS */
struct fr_MAILBOX_BLOCK fr_MAILBOXES[fr_MAX_MAILBOXES+1] =
{
  {0, NULL},  /* ID:0 Not Used. */
  {MAIN_MBX, NULL}
};
```

pvMailBox 変数は、vsta_knl()を呼び出すときアダプタレイヤーによって初期化されます。

3.3 メモリプールの作成

タスク構造体は、以下の C 構造体によってアダプタレイヤーで管理されます。

```
/* Structure to keep track of Memory Pools in FreeRTOS */
struct fr_MEMORY_POOL_BLOCK
{
  unsigned char    ucMemoryPoolID;
  unsigned int     siz_block;
  unsigned int     num_block;
  void *           xQueueHandle;
};
```

このサンプルは、アダプタレイヤーによる管理対象のタスクの作成を示します。

```
struct fr_MEMORY_POOL_BLOCK fr_MEMORY_POOLS[fr_MAX_MEMORY_POOLS+1] =
{
  {0, 0, 0, NULL},       /* ID:0 Not used. */
  {TOUCH_MPL, 16, 2, NULL},
};
```

3.4　周期ハンドラの作成

タスク構造体は、以下の C 構造体によってアダプタレイヤーで管理されます。

```
/* Structure to keep track of Cyclic Handlers in FreeRTOS */
struct fr_CYCLIC_HANDLER_BLOCK {
  unsigned char       ucCyclicHandlerID;
  void *        pvCyclicHandler;
  unsigned portLONG      interval;/* In milliseconds */
  xTaskHandle      xHandle; /* Keep track of the Task */
  unsigned int     TaskPriority;
};
```

このサンプルは、アダプタレイヤーによる管理対象のタスクの作成を示します。

```
struct fr_CYCLIC_HANDLER_BLOCK
     fr_CYCLIC_HANDLERS[fr_MAX_CYCLIC_HANDLERS+1] = {
  {0, NULL, 0, NULL, 0},
  {T4_CYC, TimerInterrupt, 0xA, NULL, 14},
};
```

周期ハンドラ用のこの構造体は、記録と更新の目的で使用します。周期ハンドラは Gatekeeper タスクの直後に動作する、優先度の高いタスクとして実装されることに注意してください。これは RI-600 の周期ハンドラの実装とは異なります。RI-600 の周期ハンドラは非タスク状態と考えられます。アダプタレイヤーで周期ハンドラをタスク状態として実装すると、起動させることが必要な周期ハンドラではリアルタイム性が要求されるため、そのリアルタイム性によっては問題が生じる可能性があります。これに代わる方法として、RX62N 上の MTU1 または CMT1 などのハードウェアタイマ割り込みを使用して周期ハンドラを実装することをお勧めします。

周期ハンドラをタスクとして実装するためには、`r_FreeRTOSAdaptor.c` の関数 `vsta_knl()` のセクション（4）を変更して周期ハンドラの作成に対応する必要があります。以下の方法で構造体をコーディングすることをお勧めします。

```
void vsta_knl(void)
{
  /* (1) Create all Tasks */
  .
  .

  /* (2) Create MailBoxes */
  .
  .

  /* (3) Create Memory Pools */
  .
  .

  /* (4) Create Cyclic Handlers */
  . fr_CycHdl_T4_CYC_init( (VP_INT) NULL );
  .
}
```

（1）個々の周期ハンドラには、固有の init 関数がなければなりません。

（2）個々の周期ハンドラに init 関数がある理由は、スタックサイズをカスタマイズする必要があるからです。

```
/*********************************************************************/
/* These 2 functions can be in r_FreeRTOSAdaptor.c or in a     */
/* file.                                  */
/*********************************************************************/

/*********************************************************************/
/* Initialize a Cyclic Handler by creating a Task           */
/*********************************************************************/
void fr_CycHdl_T4_CYC_init(VP_INT a)
{
   xTaskCreate(
         (pdTASK_CODE) fr_CycHdl_T4_CYC,
         (void *) "T4_CYC",
         160,
      NULL,
      configMAX_PRIORITIES - ( unsigned portBASE_TYPE ) 2U,
         &fr_CYCLIC_HANDLERS[T4_CYC].xHandle
   );
}


/*********************************************************************/
/* Task implementation of Cyclic Handler              */
/*********************************************************************/
void fr_CycHdl_T4_CYC(VP_INT a)
{
  unsigned portLONG interval = fr_CYCLIC_HANDLERS[T4_CYC].interval;

  for (;;)
  {
   TimerInterrupt(a);

   vTaskDelay( interval / portTICK_RATE_MS );
  }
}
```

4. アダプタレイヤーコンフィグレーション

4.1 コンフィグレーション: r_FreeRTOSAdaptor.h

この節では、r_FreeRTOSAdaptor.h でアダプタレイヤー用に設定できる、利用可能なコンフィグレーションについて説明します。

定義するマクロ	説明
configADAPTOR_SUPPORT_E_ID	これを 1 に設定すると、E_ID の戻り値は RI-600 API 仕様のコンテキスト内でサポートされます。
configADAPTOR_SUPPORT_E_CTX	これを 1 に設定すると、E_CTX の戻り値は RI-600 API 仕様のコンテキスト内でサポートされます。
GATEKEEPER_QUEUE_LEN	これは GateKeeper キューのキュー長さです。イベントリクエストの到着の速度によってキューのオーバーランが発生しないよう、この値は慎重に選択する必要があります。

4.2 コンフィグレーション: r_FreeRTOSAdaptor_Definitions.h

この節では、r_FreeRTOSAdaptor_Definitions.h で実施する必要のあるコンフィグレーションについて説明します。

r_FreeRTOSAdaptor_Definitions.h は、利用するアプリケーションが RI-600 使用時に必要とする定義で構成されます。つまり、このファイルには、以下の定義が含まれています。

- itron.h
  このファイルは、ローカルシステムの RI-600 インストールディレクトリにあります。このファイルには、RI-600 で利用できる変数タイプのすべての定義が含まれます。アプリケーションがこれらのデータタイプを保存するには、アプリケーションが使用する最新の RI-600 itron.h 定義にこのファイルを更新する必要があります。

- kernel.h
  このファイルは、ローカルシステムの RI-600 インストールディレクトリにあります。このファイルには、RI-600 が使用する必要な定義が含まれます。

- kernel_api.h
  このファイルは、ローカルシステムの RI-600 インストールディレクトリにあります。このファイルには、RI-600 API の関数プロトタイプが含まれます。

- kernel_id.h
  このファイルは RX コンフィグレータから生成される出力ファイルです。これは、RI-600 コンフィグレータ出力フォルダのプロジェクトワークスペースにあります。

これは r_FreeRTOSAdaptor_Definitions.h がどのようなものかを示すサンプルです。

```
/***********************************************************************/
/* (1) Extract from itron.h                    */
/***********************************************************************/

#ifndef __ITRON_H
#define __ITRON_H

/***********************************************************************/
/*                              type definition                         */
/***********************************************************************/
typedef signed char         B; /* signed 8 bit integer              */
typedef signed short        H; /* signed 16 bit integer             */
typedef signed long         W; /* signed 32 bit integer             */
typedef signed long long    D; /* signed 64 bit integer             */

typedef unsigned char       UB; /* unsigned 8 bit integer           */
typedef unsigned short      UH; /* unsigned 16 bit integer          */
typedef unsigned long       UW; /* unsigned 32 bit integer          */
typedef unsigned long long  UD; /* unsigned 64 bit integer          */
.
.
/***********************************************************************/
/*                               fixed number                          */
/***********************************************************************/
/*---- object attribute ----*/
#define TA_NULL         0  /* no object attribute specify      */

/*---- time out specify ----*/
#define TMO_POL         0L              /* polling                  */
#define TMO_FEVR        (-1L)           /* forever wait             */
#define TMO_NBLK        (-2L)           /* non blocking             */

/***********************************************************************/
/*                           function macros                           */
/***********************************************************************/
.
.
#endif      /* end of __ITRON_H */
```

```
/*****************************************************************/
/* (2) Extract from kernel.h                                     */
/*****************************************************************/
#ifndef __KERNEL_H
#define __KERNEL_H

typedef UW              FLGPTN;/* bit pattern of eventflag         */
typedef UH              IMASK; /* interrupt mask                   */

typedef struct  t_msg  {
    VP  msghead;                /* message header           */
} T_MSG;
typedef struct t_msg_pri {
    T_MSG       msgque;         /* message header           */
    PRI     msgpri;             /* message priority         */
    H   dmmy;
} T_MSG_PRI;
typedef struct t_rver {                 /*A0*/
    UH  maker;
    UH  prid;                           /*A0*/
    UH  spver;
    UH  prver;
    UH  prno[4];
} T_RVER;                               /*A0*/
.
.
#endif  /* end of __KERNEL_H */

/*****************************************************************/
/* (3) Extract from kernel_api.h                                 */
/*****************************************************************/

#ifndef __RX_ITRON_KERNEL_API_H
#define __RX_ITRON_KERNEL_API_H
/*******************************
 * Systemcall Prototype        *
 *******************************/

#ifdef __cplusplus
extern "C"{
#endif

ER act_tsk(ID tskid);
ER iact_tsk(ID tskid);
ER_UINT can_act(ID tskid);
ER_UINT ican_act(ID tskid);
.
.
.

#ifdef __cplusplus
}
#endif

#endif  /* end of __RX_ITRON_KERNEL_API_H */
```

```
/*********************************************************************/
/* (4) Extract from kernel_id.h                                      */
/*********************************************************************/

#ifndef __KERNEL_ID_H
#define __KERNEL_ID_H

/*-------------------------------------------------*/
#define LED_TASK  1
#define _RI_MAX_TSK   1
#define  VTMAX_TSK    _RI_MAX_TSK
.
.
.
#endif  /* __KERNEL_ID_H */
```

4.3 コンフィグレーション: r_FreeRTOSAdaptor.c

この節では、r_FreeRTOSAdaptor.c で実施する必要のあるコンフィグレーションについて説明します。

これらのコンフィグレーションを以下の形式で示します。

4.3.n セクションタイトル

セクションラベル:

説明:

指示:

サンプルコンフィグレーション:

RI-600 適合アプリケーションと完全に統合できるようにアダプタレイヤーを設定するためには、2 つの主要なカテゴリがあります。これらは以下のとおりです。

- セクション（A）～（B）
  これらのセクションには、必要なプレコンフィグレーションが含まれます。これらの特性は通常、RI-600 コンフィグレータが管理します。ただし、アダプタレイヤーは、アプリケーションに関するこのような情報も必要とします。アダプタレイヤーを効果的に使用するためには、このセクションを手動でプログラムする必要があります。
- セクション（1）～（4）
  これらはアダプタレイヤーがタスク、メールボックス、メモリプール、および周期ハンドラを管理するために必要なコンフィグレーションです。

4.3.1 優先度マップ

セクションラベル

（A）優先度マップサイズと優先度マップ

説明

アダプタレイヤーは、RI-600 の優先度を FreeRTOS™の等価優先度に変換するマッピング用の優先度ルックアップテーブルを必要とします。この変換で優先度マップが効力を発揮します。この優先度マップは、RI-600 chg_pri()API が呼び出されると必ず参照されます。このテーブルで FreeRTOS™の等価優先度が見つからない場合、API を呼び出したタスクはエンドレスの for ループに陥ることになります。

指示

RI-600 の対象アプリケーションは、実在するタスクに静的な優先度を割り当てています。これらの優先度は RI-600 コンフィグレーションファイルにあります。RI-600 コンフィグレーションファイルのタスクに割り当てた優先度の数値が小さいほど、タスクの優先度は高くなることに注意してください。ただし、FreeRTOS™では、タスクに割り当てた優先度の数値が大きいほど、タスクの優先度は高くなります。

RI-600 の優先度を FreeRTOS™タスクの等価優先度に変換するための体系的な手法が開発されています。

例えば、RI-600 のタスクのコンフィグレーションは、以下のとおりです。

```
.
.
task[]{
  entry_address = task1();
  name       = TASK_1;
  stack_size    = 256;
  stack_section = SURI_STACK;
  priority      = 3;
  initial_start = OFF;
  exinf         = 0x0;
};
task[]{
  entry_address = task2();
  name       = TASK_2;
  stack_size    = 256;
  stack_section = SURI_STACK;
  priority      = 4;
  initial_start = OFF;
  exinf         = 0x0;
};
```

```
task[]{
  entry_address = task3();
  name       = TASK_3;
  stack_size    = 1024;
  stack_section = SURI_STACK;
  priority      = 20;
  initial_start = OFF;
  exinf         = 0x0;
};
.
.
.
```

RI-600 の優先度を FreeRTOS™の優先度に変換する 2 ステップのシーケンシャルプロセスを以下で説明します。

- RI-600 の優先度をテーブルの上から下に昇順で並べます。
- FreeRTOS™の優先度番号を、2 から始めて、テーブルの下から上へ増加するように割り当てます。FreeRTOS™の IDLE タスクは優先度 1 です。IDLE タスクと CPU 時間を共有しないためには、FreeRTOS™でのタスク優先度の割り当てを 2 から始める必要があります。

タスク名	RI-600 タスク優先度	FreeRTOS™優先度
TASK_1	3	4
TASK_2	4	3
TASK_3	20	2

サンプルコンフィグレーション

上記の結果、以下の C コードがアダプタレイヤー内に追加されます。

```
/**********************************************************************
*           *** CONFIGURATIONS ***
*
*           (A) Priority Map Size and Priority Map.
**********************************************************************/

#define PRIORITY_MAP_SIZE           3

struct _fr_TASK_PRIORITIES_MAP fr_TASK_PRIORITIES_MAP [PRIORITY_MAP_SIZE] = {
      {  3,  4 }, // RI_PRIORITY, FR_PRIORITY
      {  4,  3 },
      { 20,  2 },
};
```

4.3.2 グローバル変数とタスク関数

セクションラベル

（B）グローバル変数と関数のインポート

説明

このセクションは、アダプタレイヤーがタスクの作成に使用するために必要なすべてのタスクアプリケーション関数をインポートします。

指示

アプリケーションレイヤーに存在するすべてのタスク関数プロトタイプを統合し、extern キーワード修飾子を用いてアダプタレイヤーにインポートします。

サンプルコンフィグレーション

セクション（B）の下でアダプタレイヤー内に追加するサンプル C コードは以下のとおりです。

```
/*************************************************************
*           *** CONFIGURATIONS ***
*
*           (B) Import of global variables and functions
*
*           This section should contain all the Task Function name to be used.
*************************************************************/

      /* Application Layer functions */
extern void Task_1( VP_INT );
extern void Task_2( VP_INT );
extern void Task_3( VP_INT );
```

4.3.3 アダプタレイヤー: タスクの定義

セクションラベル

（1）タスクの定義

説明

このセクションは、アダプタレイヤーが作成して管理するすべてのアプリケーションタスクを登録します。

指示

このセクションには 2 つのステップが含まれます。まず、アプリケーションレイヤー内のタスクの総数を定義します。次に、構造体 fr_TASKS[]の配列を初期化します。

サンプルコンフィグレーション

セクション（1）の下でアダプタレイヤー内に追加するサンプル C コードは以下のとおりです。

```
/***********************************************************
*           *** CONFIGURATIONS ***
*
*           (1) Task Definitions
*
*           Task creation for Adaptor Layer to manage.
***********************************************************/
#define fr_MAX_TASKS                        3     /* 3 Tasks */

struct fr_TASK_BLOCK fr_TASKS[fr_MAX_TASKS+1] =
{     /* Offset 0; Not used. */
      {(pdTASK_CODE) NULL, "", 0, 0, NULL, NULL, NULL, 0, 0 },
      {(pdTASK_CODE) task1,
        "TASK_1",
        128,
        4,
        NULL, NULL, DORMANT, 0, 0 },
      {(pdTASK_CODE) task2,
        " TASK_2",
        128,
        3,
        NULL, NULL, DORMANT, 0, 0 },
      {(pdTASK_CODE) task3,
        " TASK_3",
        256,
        2,
        NULL, NULL, DORMANT, 0, 0 },
};
```

4.3.4 アダプタレイヤー: メールボックスの定義

セクションラベル

（2）メールボックスの定義

説明

このセクションは、アプリケーションタスクが使用するすべての必要なメールボックスを定義します。

指示

アプリケーションコンフィグレータファイルのすべてのメールボックスの定義を統合し、アダプタレイヤー内で適宜設定します。

RI-600 は無限長のメールボックスをサポートすることに注意してください。ただし、FreeRTOS™にはこのような特性はありません。このため、よく考えてメールボックスの最大長を選択する必要があります。

サンプルコンフィグレーション

RI-600 コンフィグレータファイルのサンプルは、以下のようになります。

```
.
.
.
mailbox[]{
      name              = TASK_1_MBX;
      wait_queue        = TA_TFIFO;
      message_queue     = TA_MFIFO;
      max_pri           = 1;
};
.
.
.
```

セクション（2）の下でアダプタレイヤー内に追加するサンプル C コードは以下のとおりです。

```
/************************************************************
*           *** CONFIGURATIONS ***
*
*           (2) MailBox Definitions
*
*           MailBox creation for Adaptor Layer to manage.
*
************************************************************/
#define fr_MAX_MAILBOXES                1

#define fr_MAILBOX_QUEUE_LENGTH         5

struct fr_MAILBOX_BLOCK fr_MAILBOXES[fr_MAX_MAILBOXES+1] =
{
      {0, NULL},  /* ID:0 Not Used. */
      {TASK_1_MBX, NULL},
  };
```

4.3.5 アダプタレイヤー: メモリプールの定義

セクションラベル

（3）メモリプールの定義

説明

このセクションは、アダプタレイヤーが管理するすべてのメモリプールを定義します。

指示

アプリケーションタスクが使用するすべてのメモリプールを統合します。

サンプルコンフィグレーション

RI-600 コンフィグレータファイルのサンプルは、次のようになります。

```
.
.
.
memorypool[]{
      name        = TASK_1_MPL;
      wait_queue  = TA_TFIFO;
      section     = BRI_HEAP;
      siz_block   = 64;
      num_block   = 10;
};
.
.
.
```

セクション（3）の下でアダプタレイヤー内に追加するサンプル C コードは以下のとおりです。

```
/************************************************************
*           *** CONFIGURATIONS ***
*
*           (3) Memory Pool Definitions
*
*           Memory Pool creation for Adaptor Layer to manage.
************************************************************/

#define fr_MAX_MEMORY_POOLS          1

struct fr_MEMORY_POOL_BLOCK fr_MEMORY_POOLS[fr_MAX_MEMORY_POOLS+1] =
{
      {0, 0, 0, NULL},              /* ID:0 Not used. */
      {TASK_1_MPL, 64,  10, NULL},
  };
```

4.3.6 アダプタレイヤー: 周期ハンドラの定義

セクションラベル

（4）周期ハンドラの定義

説明

このセクションでは、アプリケーションで使用するすべての周期ハンドラを定義します。

指示

アダプタレイヤーの周期ハンドラは、タスクとして、あるいは CMT1 や MTU1 などのハードウェアタイマからの定期割り込みとして実装することができます。

より難しいリアルタイムシステムの要件については、ハードウェアを用いてこれらの周期ハンドラを実装することを強くお勧めします。ただし、開発者は、アダプタレイヤー内に実装されたタスクレベル周期ハンドラを利用するという選択肢があります。

サンプルコンフィグレーション

RI-600 コンフィグレータファイルのサンプルは、次のようになります。

```
.
.
.
cyclic_hand[]{
      entry_address     = TimerInterrupt();
      name              = T4_CYC;
      exinf             = 0x0;
      start             = OFF;
      phsatr            = OFF;
      interval_counter  = 0xa;
      phs_counter       = 0x0;
};
.
.
.
```

セクション（4）の下でアダプタレイヤー内に追加するサンプル C コードは以下のとおりです。

```
/************************************************************
*           (4) Cyclic Handler Definitions
*
*           Cyclic Handler creation
************************************************************/
#define fr_MAX_CYCLIC_HANDLERS            1

struct fr_CYCLIC_HANDLER_BLOCK
         fr_CYCLIC_HANDLERS[fr_MAX_CYCLIC_HANDLERS+1] = {
      {0, NULL, 0, NULL, 0},
      {T4_CYC, TimerInterrupt, 0xA, NULL}, // 10ms Interval
  };
```

ただし、周期ハンドラをアダプタレイヤーのタスクとして起動して初期化するには、さらに 2 つの関数が必要であることに注意してください。これらの 2 つのサンプル関数を以下に示します。

```
/************************************************************
* Function Name         : fr_CycHdl_T4_CYC_init
* Description           : This function creates the Cyclic Handler as a Task.
* Argument              : a:VP_INT (Not in use.)
* Return Value          : None.
************************************************************/
      /* -=] T4_CYC :: Cyclic Handler [=- */
void fr_CycHdl_T4_CYC_init(VP_INT a)
{
      xTaskCreate(
            (pdTASK_CODE) fr_CycHdl_T4_CYC,
            (void *) "T4_CYC",
            160,
            NULL,
            configMAX_PRIORITIES - (unsigned portBASE_TYPE) 2U,
            &fr_CYCLIC_HANDLERS[T4_CYC].xHandle
            );
}


/************************************************************
* Function Name         : fr_CycHdl_T4_CYC
* Description           : Task code of the Cyclic Handler.
* Argument              : a:VP_INT (Not in use.)
* Return Value          : None.
************************************************************/
void fr_CycHdl_T4_CYC(VP_INT a)
{
      unsigned portLONG interval = fr_CYCLIC_HANDLERS[T4_CYC].interval;

      for (;;)
      {
            TimerInterrupt(a);

            vTaskDelay( interval / portTICK_RATE_MS );
      }
  }
```

その後、fr_CycHdl_T4_CYC_init()を vsta_knl()関数に配置して、周期ハンドラを作成する必要があります。

vsta_knl()のサンプル C コードは、次のようになります。

```
void vsta_knl()
{
.
.
.
      /*    *** EDIT HERE WHERE NECESSARY ***      */
      /* (4) Create Cyclic Handler for :: T4_CYC */
      /*    *** EDIT HERE WHERE NECESSARY ***      */
            fr_CycHdl_T4_CYC_init( (VP_INT) NULL );
.
.
.
}
```

5. RI-600 仕様へのアダプタレイヤーの適合性

5.1 文書フォーマット

この節は、以下の文書フォーマットになります。

[タスクの簡単な説明]:[API 関数]

C 言語 API

[API 関数の使用方法]

パラメータ

[API パラメータ]

戻り値

[共通の戻り値]

エラーコード

名前	説明	サポート
[定義した戻り値]	[説明]	あり/なし/設定可能

注記: 設定可能なサポートは、以下に示すとおり、r_FreeRTOSAdaptor.h で有効にすることができます。

```
/**************************************************************************
*  Configurations for Adaptor Layer
*  To Enable: Set macro to 1
**************************************************************************/
#define configADAPTOR_SUPPORT_E_ID     1
#define configADAPTOR_SUPPORT_E_CTX    1
```

5.2 アダプタレイヤー: API 仕様

5.2.1 タスクの起動（起動コード指定）: sta_tsk()

C 言語 API

```
ER ercd = sta_tsk(ID tskid, VP_INT stacd);
```

パラメータ

tskid	タスク ID
stacd	起動コード

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_OBJ	オブジェクト状態エラー（tskid のタスクが休止状態（DORMANT）ではない）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.2 自タスクの終了: ext_tsk()

C 言語 API

```
Void ext_tsk(void);
```

パラメータ

なし

戻り値

なし

エラーコード

なし

5.2.3 タスクの強制終了: ter_tsk()

C 言語 API

```
ER ercd = ter_tsk(ID tskid);
```

パラメータ

tskid	タスク ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_OBJ	オブジェクト状態エラー（tskid のタスクが休止状態（DORMANT）ではない）	なし
E_ILUSE	サービスコール不正使用（tskid に自タスクを指定）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.4 タスク優先度の変更: chg_pri()

C 言語 API

```
ER ercd = chg_pri(ID tskid, PRI tskpri);
```

パラメータ

tskid	タスク ID
tskpri	タスクの新しいベース優先度

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	パラメータエラー	なし
E_ID	不正 ID 番号	設定可能
E_ILUSE	サービスコール不正使用（tskid に自タスクを指定）	なし
E_OBJ	オブジェクト状態エラー（tskid のタスクが休止状態（DORMANT）ではない）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.5 タスクの状態参照: ref_tst()

C 言語 API

```
ER ercd = ref_tst(ID tskid, T_RTSK[1] *pk_rtsk);
```

パラメータ

tskid	タスク ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

[1]データ構造体については、RI-600 ユーザーマニュアルを参照してください。

5.2.6 起床待ち: slp_tsk()

C 言語 API

```
ER ercd = slp_tsk();
```

パラメータ

なし

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	パラメータエラー	なし
E_RLWAI	待ち状態強制解除（待ち状態中に rel_wai が発行された）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.7 タスクの起床: iwup_tsk()

C 言語 API

```
ER ercd = iwup_tsk(ID tskid);
```

パラメータ

tskid	タスク ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_OBJ	オブジェクト状態エラー	なし
E_QOVR	キューイングのオーバーフロー	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.8 強制待ち状態への移行: isus_tsk()

C 言語 API

```
ER ercd = isus_tsk(ID tskid);
```

パラメータ

tskid	タスク ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_OBJ	オブジェクト状態エラー（tskid のタスクが休止状態（DORMANT）ではない）	なし
E_QOVR	キューイングのオーバーフロー（強制待ち要求ネスト数のオーバーフロー）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.9 強制待ち状態からの再開: irsm_tsk()

C 言語 API

```
ER ercd = irsm_tsk(ID tskid);
```

パラメータ

tskid	タスク ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_OBJ	オブジェクト状態エラー（tskid のタスクが強制待ち状態（SUSPENDED）ではない）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.10 タスクの遅延: dly_tsk()

C 言語 API

```
ER ercd = dly_tsk(RELTIM dlytim);
```

パラメータ

dlytim	遅延時間（ミリ秒）

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	パラメータエラー	なし
E_RLWAI	待ち状態強制解除（待ち状態中に rel_wai が発行された）	なし
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.11　メールボックスへの送信: snd_mbx()、isnd_mbx()

C 言語 API

```
ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg);
ER ercd = isnd_mbx(ID mbxid, T_MSG *pk_msg);
```

パラメータ

mbxid	メールボックス ID
pk_msg	送信メッセージの先頭アドレス

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	**パラメータエラー**	**なし**
E_ID	**不正 ID 番号**	**設定可能**
E_CTX	**コンテキストエラー（許可されていないシステム状態からの呼び出し）**	**設定可能**

5.2.12 メールボックスからの受信: rcv_mbx()、prcv_mbx()、trcv_mbx()

C 言語 API

```
ER ercd = rcv_mbx(ID tskid, T_MSG[2] **ppk_msg);
ER ercd = prcv_mbx(ID tskid, T_MSG **ppk_msg);
ER ercd = trcv_mbx(ID tskid, T_MSG **ppk_msg, TMO tmout);
```

パラメータ

mbxid	メールボックス ID
ppk_msg	受信メッセージの先頭アドレスを返す領域へのポインタ
tmout	タイムアウト（ミリ秒）

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	パラメータエラー	なし
E_ID	不正 ID 番号	設定可能
E_RLWAI	待ち状態強制解除	あり
E_TMOUT	ポーリング失敗、またはタイムアウト	あり
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

[2]データ構造体については、RI-600 ユーザーマニュアルを参照してください。

5.2.13 固定長メモリブロックの獲得: get_mpf()、pget_mpf()

C 言語 API

```
ER ercd = get_mpf(ID mpfid, VP *p_blk);
ER ercd = pget_mpf(ID mpfid, VP *p_blk);
```

パラメータ

mpfid	固定長メモリプール ID
p_blk	メモリブロックの先頭アドレスを返す領域へのポインタ

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	パラメータエラー	なし
E_ID	不正 ID 番号	設定可能
E_RLWAI	待ち状態強制解除（待ち状態中に rel_wai が発行された）	あり
E_TMOUT	ポーリング失敗、またはタイムアウト	あり
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.14 固定長メモリプールブロックの解放: rel_mpf()

C 言語 API

```
ER ercd = rel_mpf(ID mpfid, VP blk);
```

パラメータ

mpfid	固定長メモリプール ID
blk	返却するメモリブロックの先頭アドレス

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	**パラメータエラー**	**なし**
E_ID	**不正 ID 番号**	**設定可能**
E_CTX	**コンテキストエラー（許可されていないシステム状態からの呼び出し）**	**設定可能**

5.2.15　システム時刻の参照: get_tim()

C 言語 API

```
ER ercd = get_tim(SYSTIM[3] *p_systim);
```

パラメータ

P_systim　　システム時刻を返す領域へのポインタ

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_CTX	**コンテキストエラー（許可されていないシステム状態からの呼び出し）**	**設定可能**

[3]データ構造については、RI-600 ユーザーマニュアルを参照してください。

5.2.16　周期ハンドラの動作開始: sta_cyc()

C 言語 API

```
ER ercd = sta_cyc(ID cycid);
```

パラメータ

cycid	周期ハンドラ ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.17　周期ハンドラの動作停止: stp_cyc()

C 言語 API

```
ER ercd = stp_cyc(ID cycid);
```

パラメータ

cycid	周期ハンドラ ID

戻り値

正常終了については E_OK、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_ID	不正 ID 番号	設定可能
E_CTX	コンテキストエラー（許可されていないシステム状態からの呼び出し）	設定可能

5.2.18 タスク優先順位の回転: irot_rdq()

C 言語 API

```
ER ercd = irot_rdq(PRI tskpri);
```

パラメータ

`tskpri`　タスク優先度

戻り値

正常終了については `E_OK`、またはエラーコード（下記）

エラーコード

名前	説明	サポート
E_PAR	**パラメータエラー**	**なし**
E_CTX	**コンテキストエラー（許可されていないシステム状態からの呼び出し）**	**設定可能**

5.2.19　カーネルの起動:vsta_knl()

C 言語 API

```
Void vsta_knl(void);
```

パラメータ

なし

戻り値

なし

エラーコード

なし

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2012.03.05	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。お客様が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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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その総株主の議決権の過半数を直接または間接に保有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。

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・電話番号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 エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■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：http://japan.renesas.com/contact/

Colophon 1.1