RX62N Group

OS Adaptor Layer:

User Manual and API Specification for RI600 & FreeRTOS™

R01AN0766ES0100
Rev.1.00
Oct 11, 2011

Introduction

This application note describes the Adaptor Layer in detail. This includes its features and functionalities.

Target Device

- RX62N Group MCU (product number: R5FF562N8BDBG)

Target Board

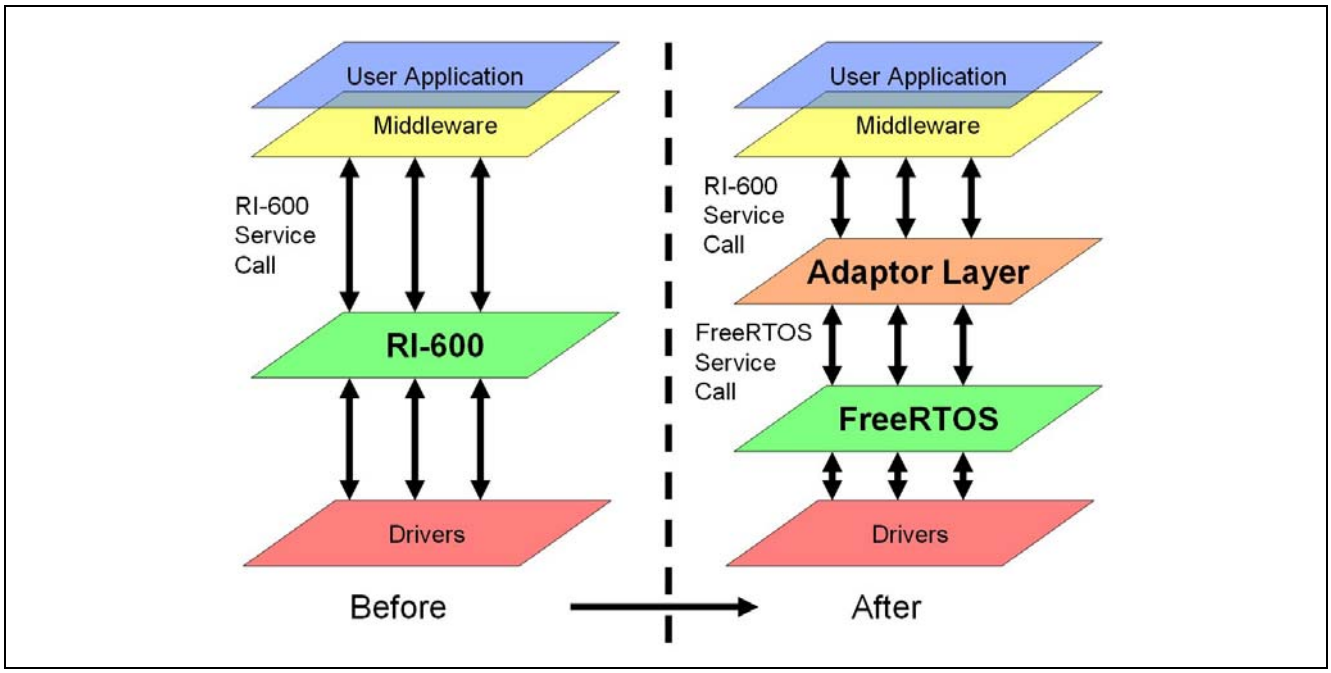
- RX62N Renesas Start Kit + (product number: R0K5562N0C000BE)

Note: FreeRTOS™ is a trademark of Real Time Engineers Ltd

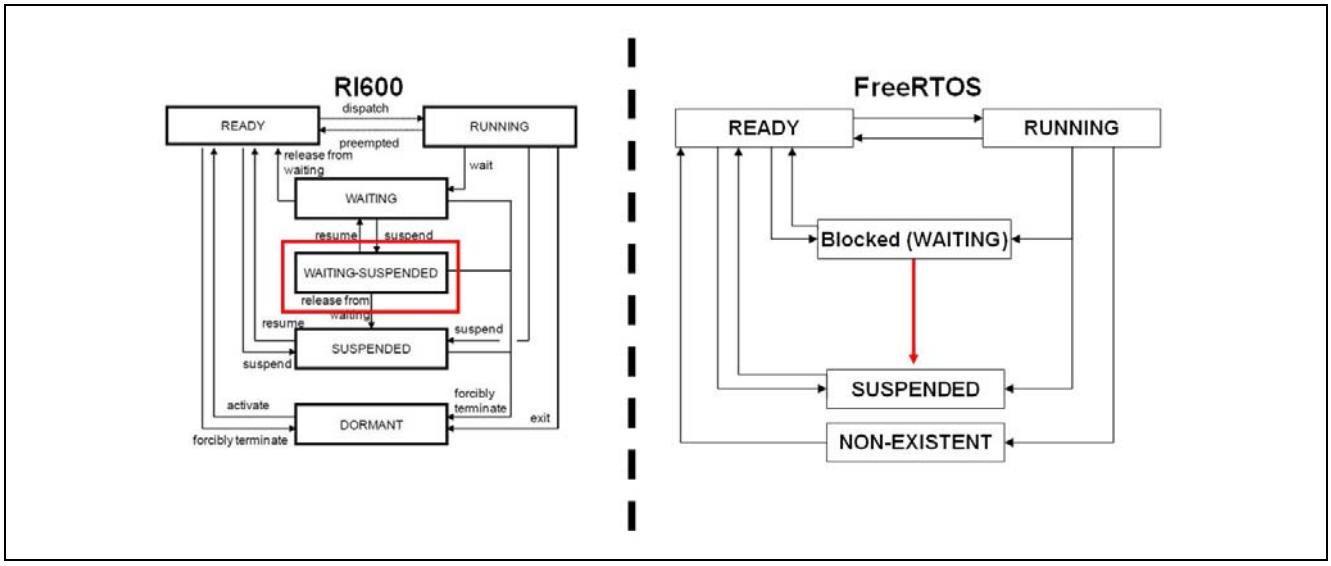
Contents

1. Introduction to the Adaptor Layer

The Adaptor Layer was designed to allow for applications that are targeted to run on RI-600 RTOS to be quickly ported to run on FreeRTOS™ without the need to do any re-programming. An illustration is as follows:



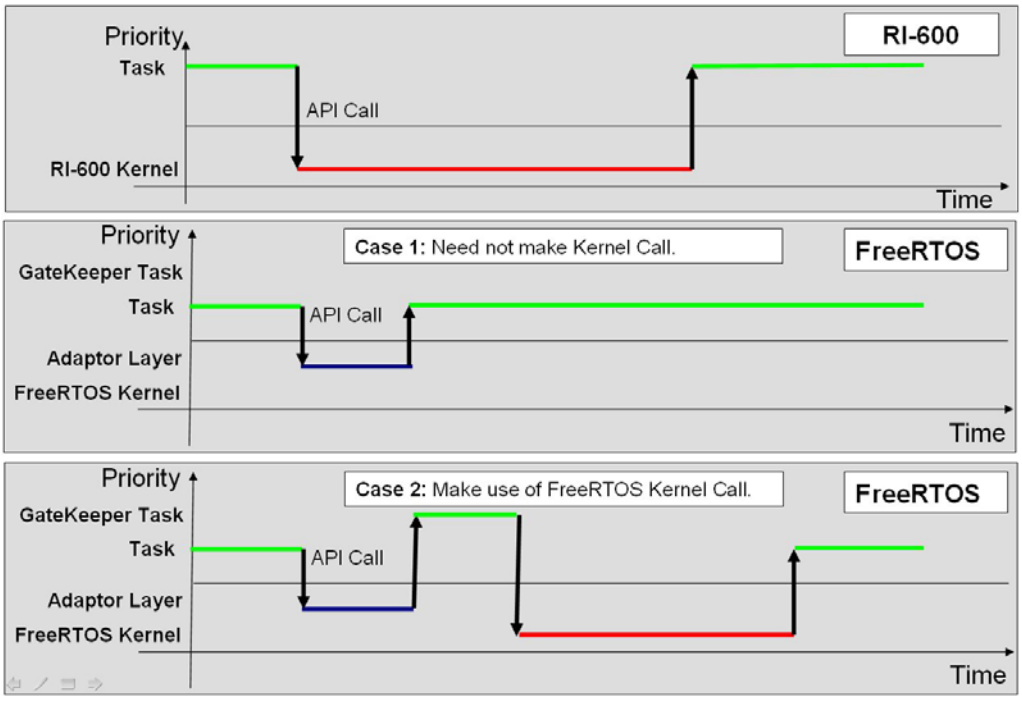
A juxtaposition of RI-600 and FreeRTOS™ Task States shows:



It should be noted that in FreeRTOS™, when a Task is placed in the Blocked (WAITING) State and is subsequently suspended, the resumption of this Task will not place the Task back in the Blocked (WAITING) State as it should have been. Instead, it will be prematurely released from the Blocked (WAITING) state that it was previously in. This can be, at times, deemed as an undesired behavior.

Applications that target the RI-600 platform will not expect such behavioral changes. As such, the Adaptor Layer had to make up for the WAITING-SUSPENDED state that FreeRTOS™ does not provide for. To this end, the Adaptor Layer has to make use of a GateKeeper Task to managethe WAITING-SUSPENDED mechanism. Thus, the compiled Adaptor Layer task states are shown as follows:

Every API that causes a Task State to be changed will have its request latched to the GateKeeper Queue to be processed. However, for APIs that do not require a change of Task State, the Adaptor Layer will take care of its invocation. This can be clearly seen from the following diagram:

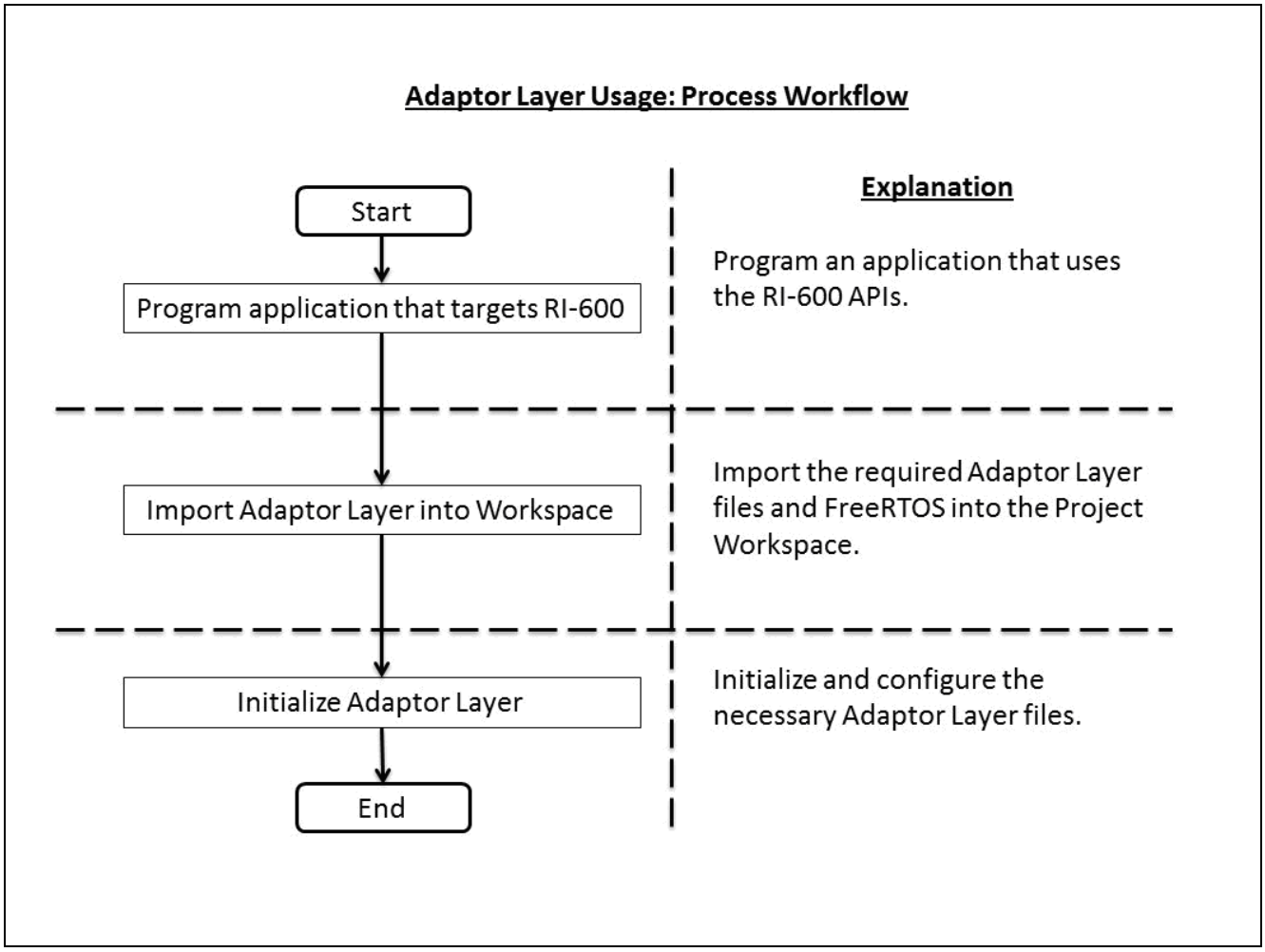


2. Adaptor Layer Process Workflow

This section describes the process workflow to successfully deploy the Adaptor Layer for use.

2.1 Overview

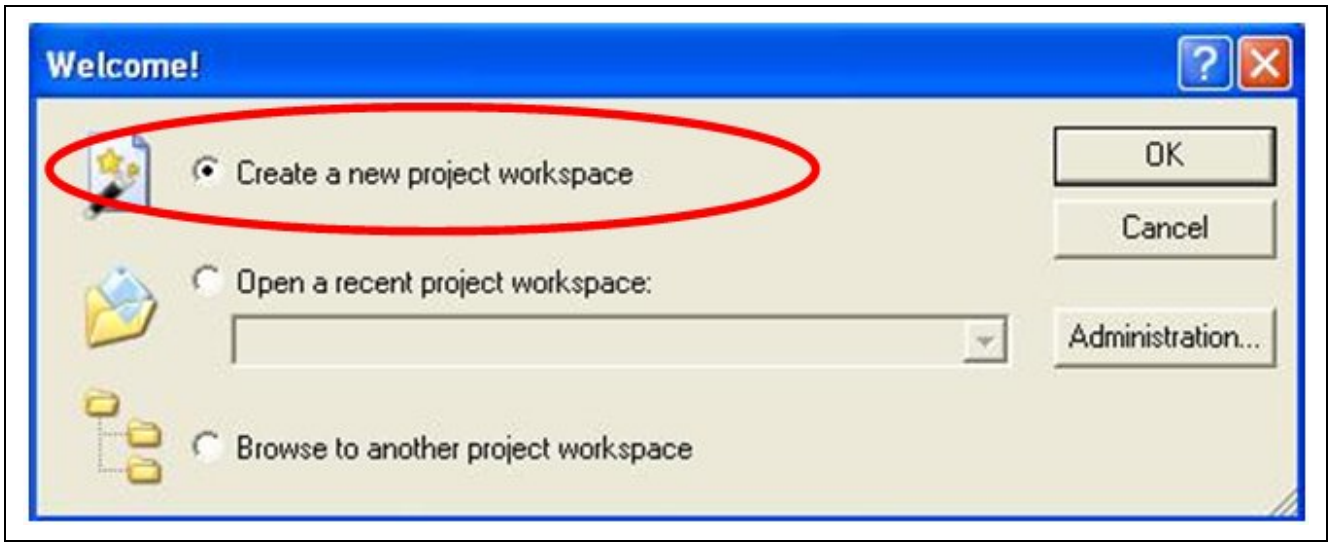
An overview of the Adaptor Layer deployment process can be seen as follows:



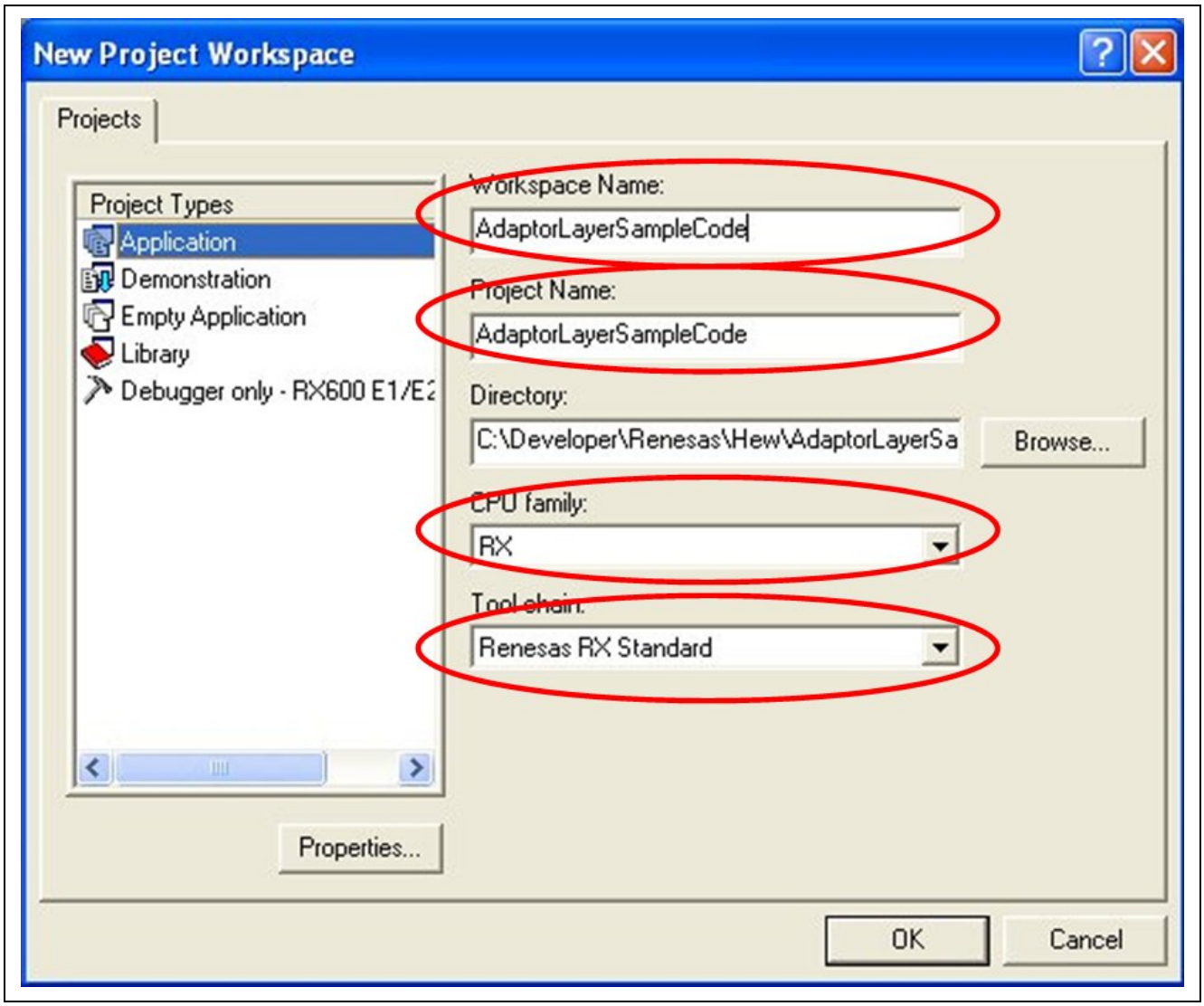
2.2 Guided Example using Renesas HEW IDE

2.2.1 Creating a new RI-600 Project Workspace

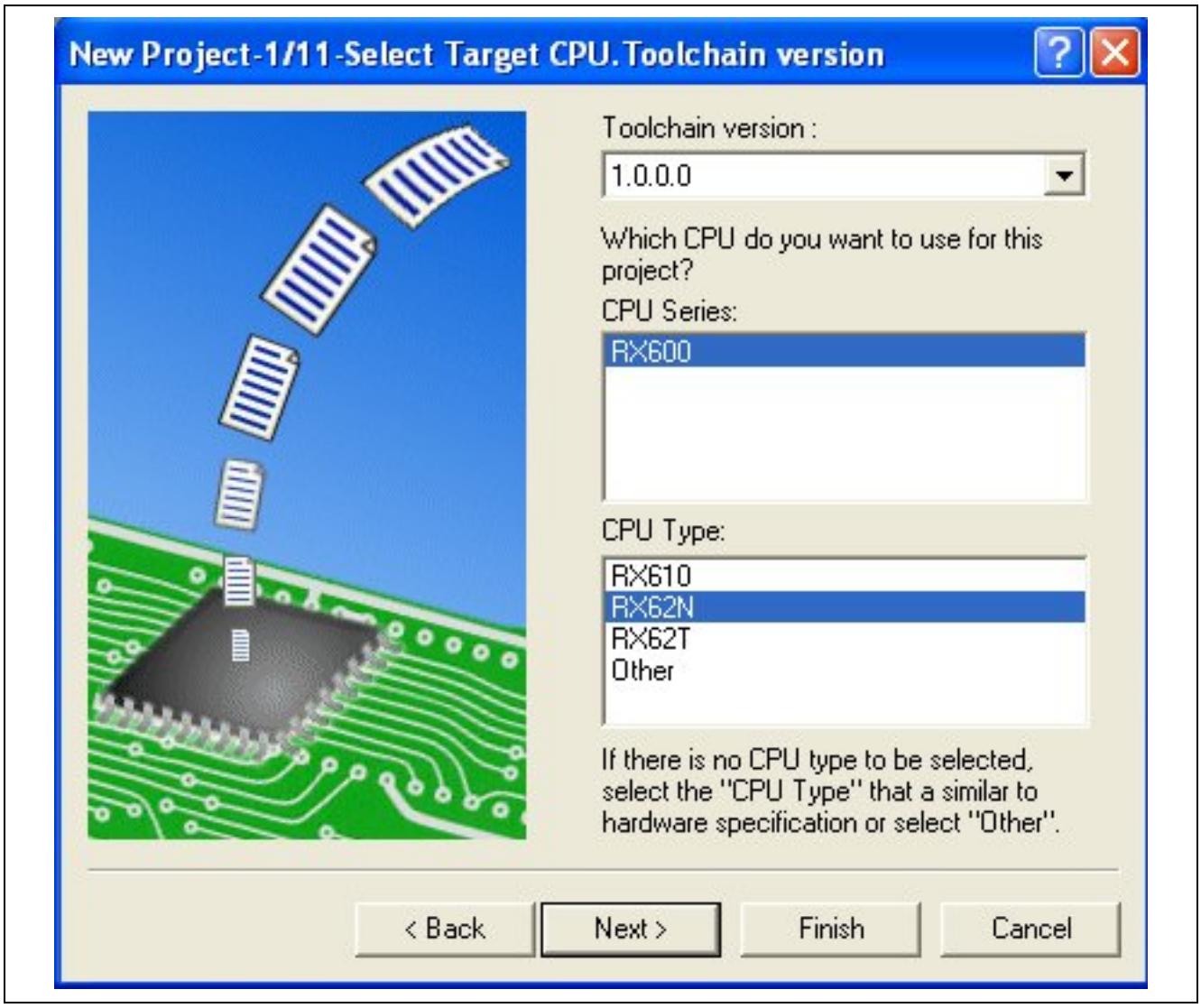
Launch Renesas HEW IDE and create a new project workspace as follows:



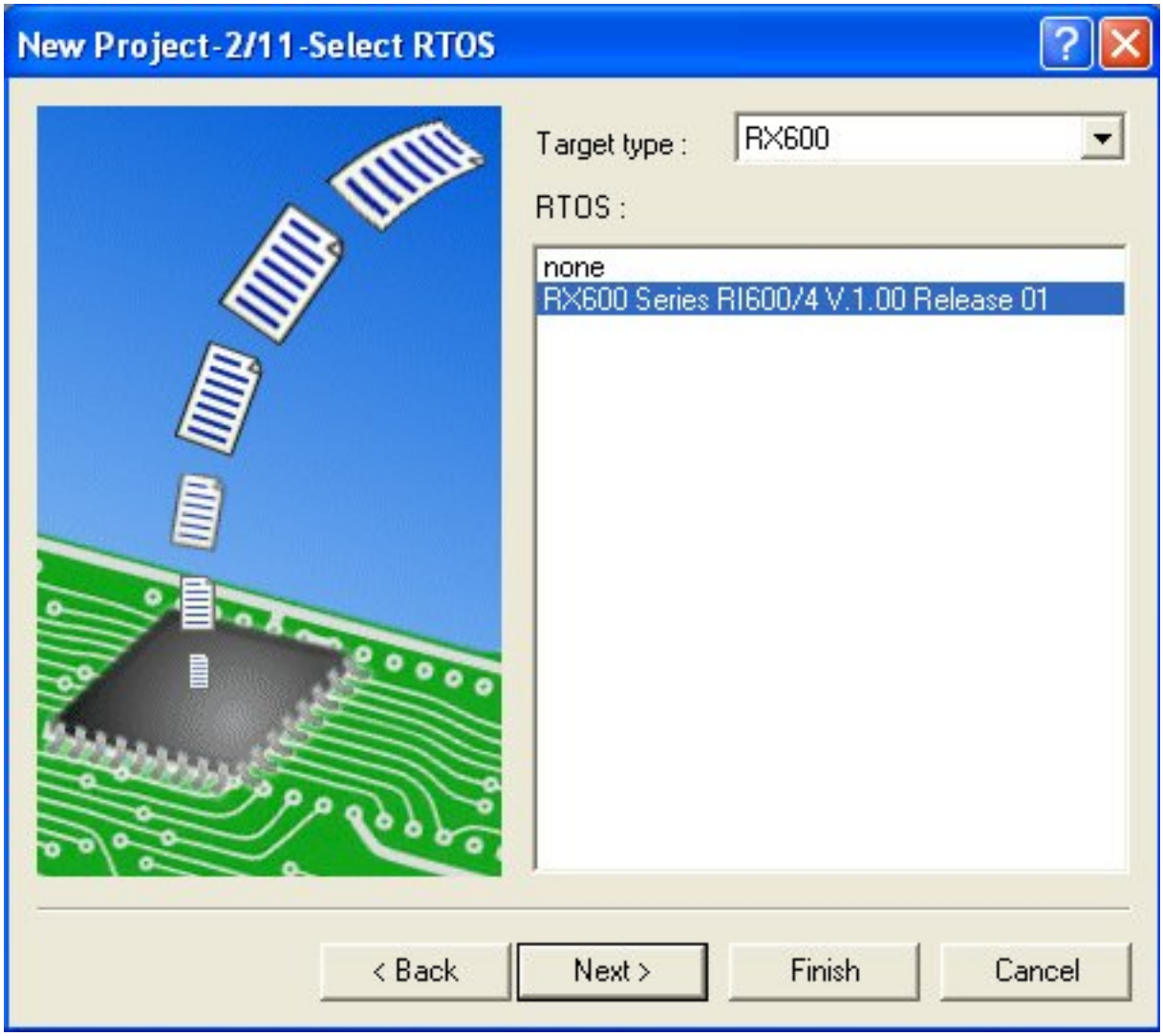
Enter a valid workspace and project name as follows:



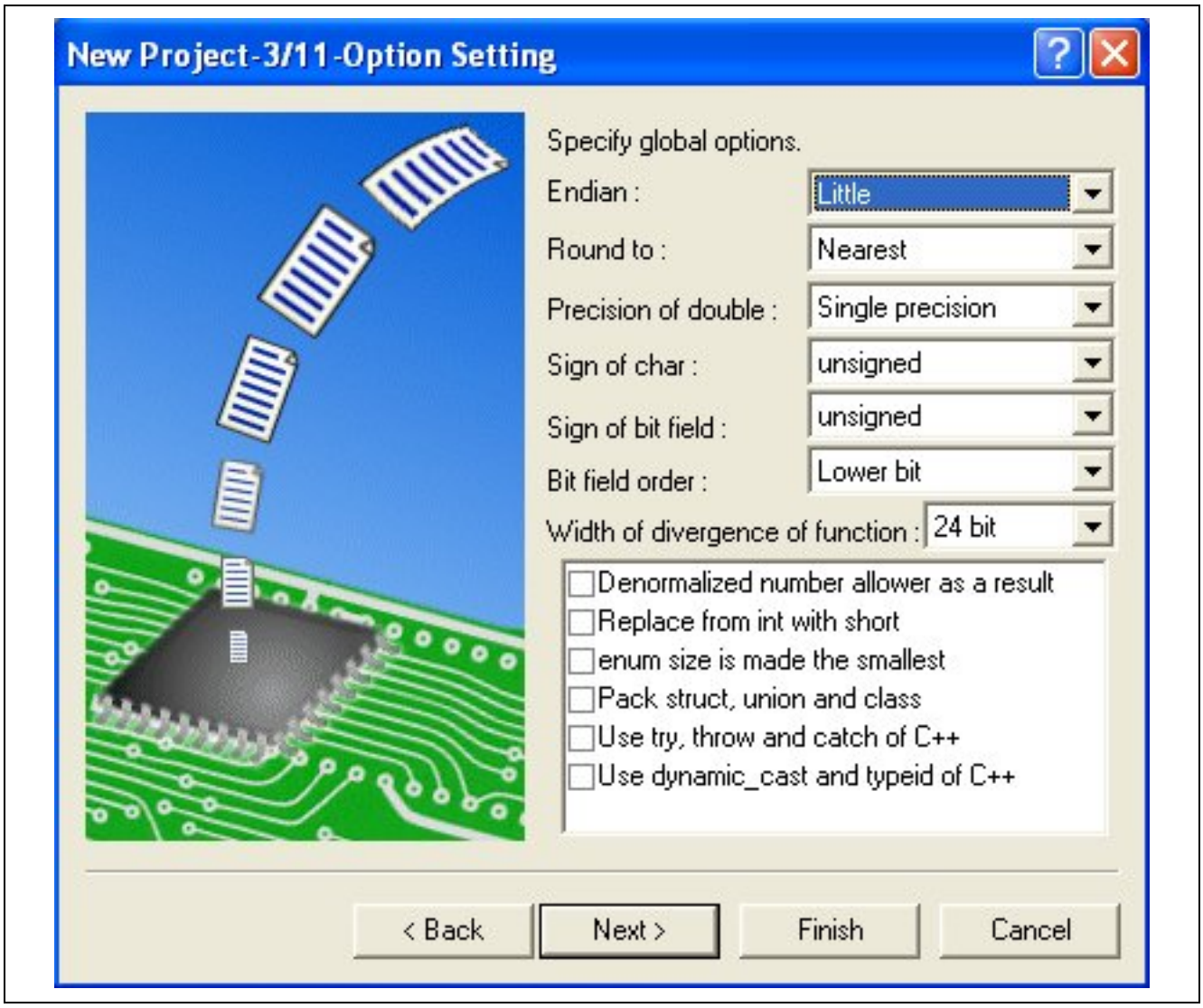
Select the appropriate toolchain and CPU type as follows:



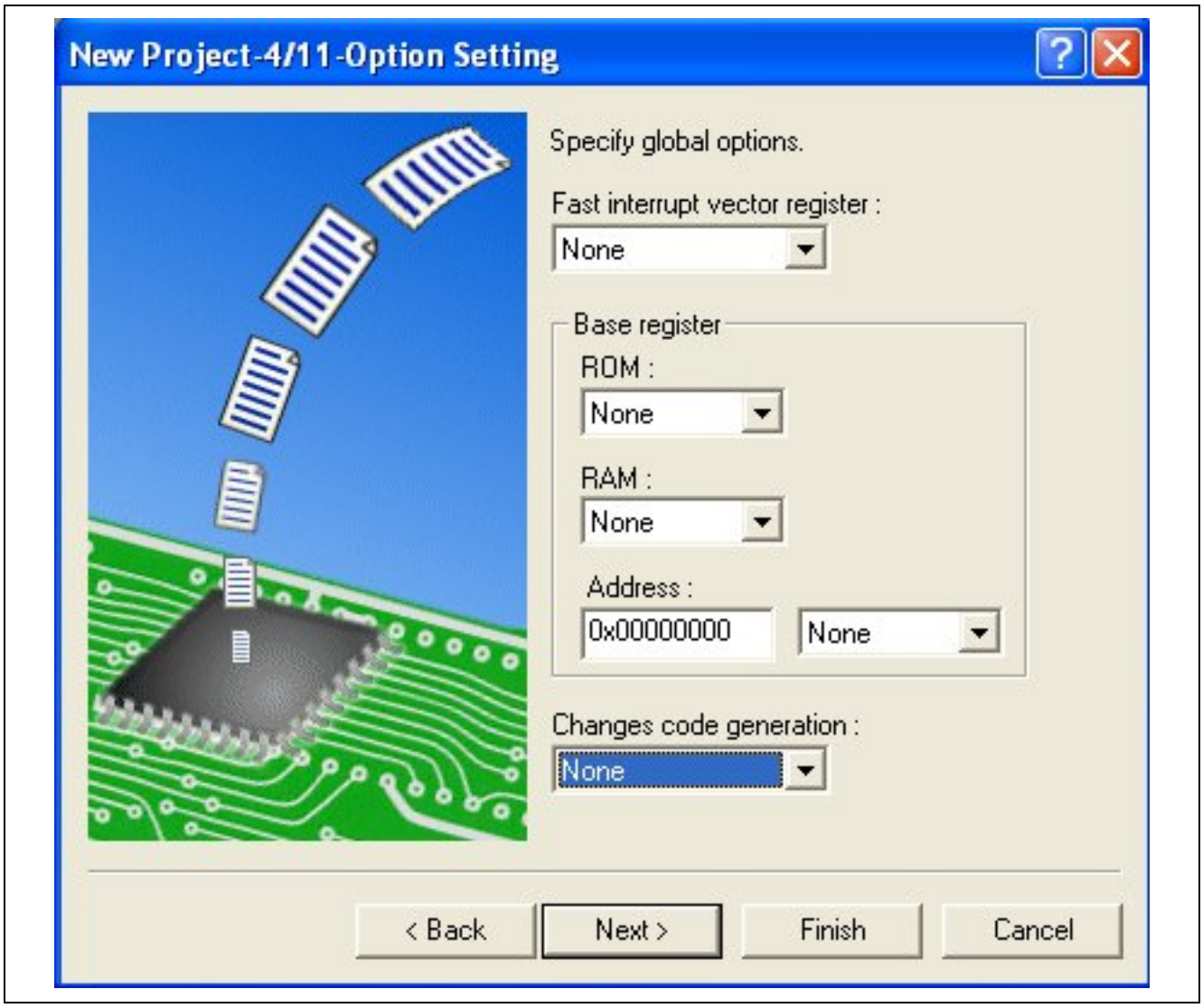
Select RI-600 as the RTOS to be in use:



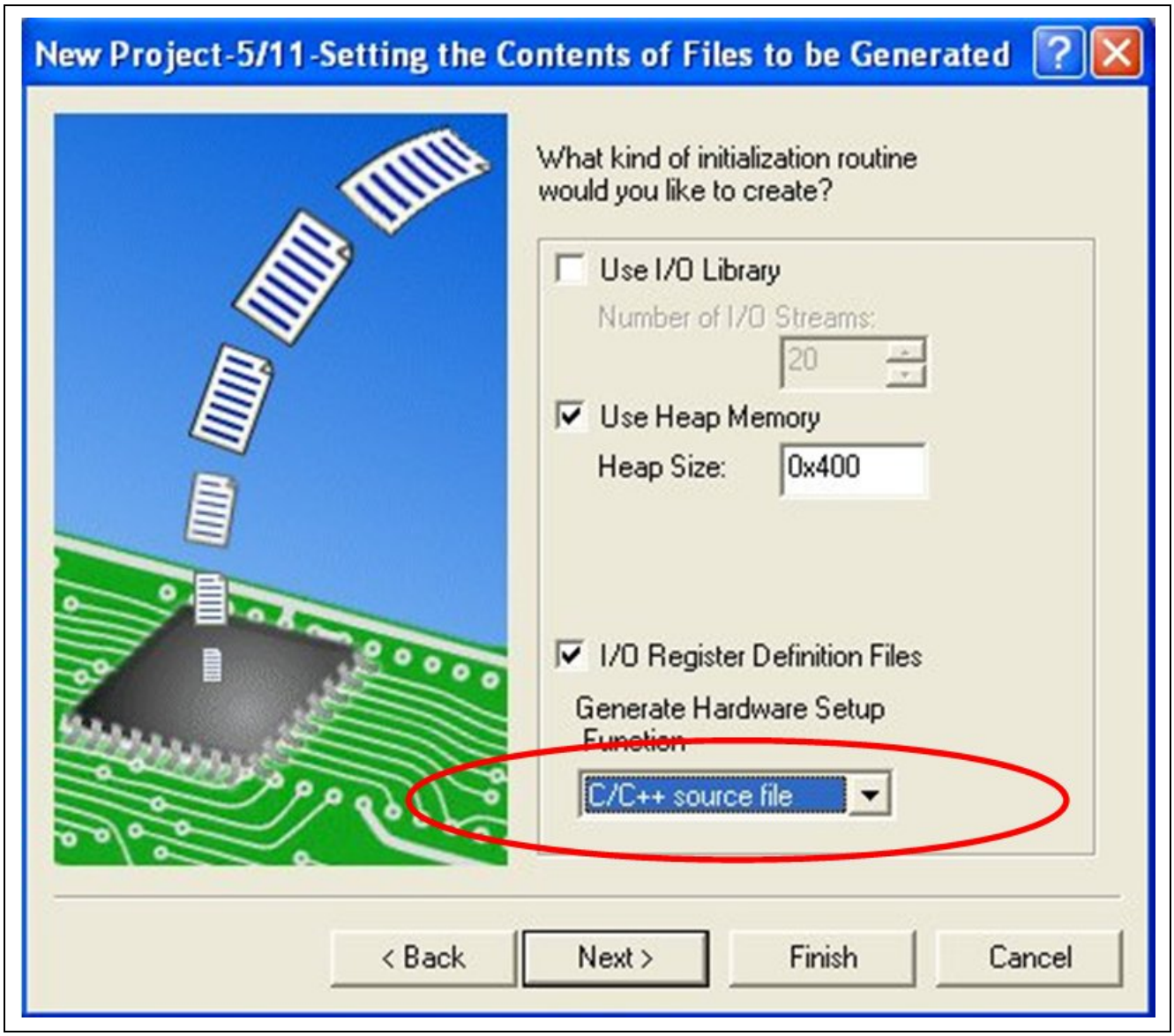
Option settings should be as shown:



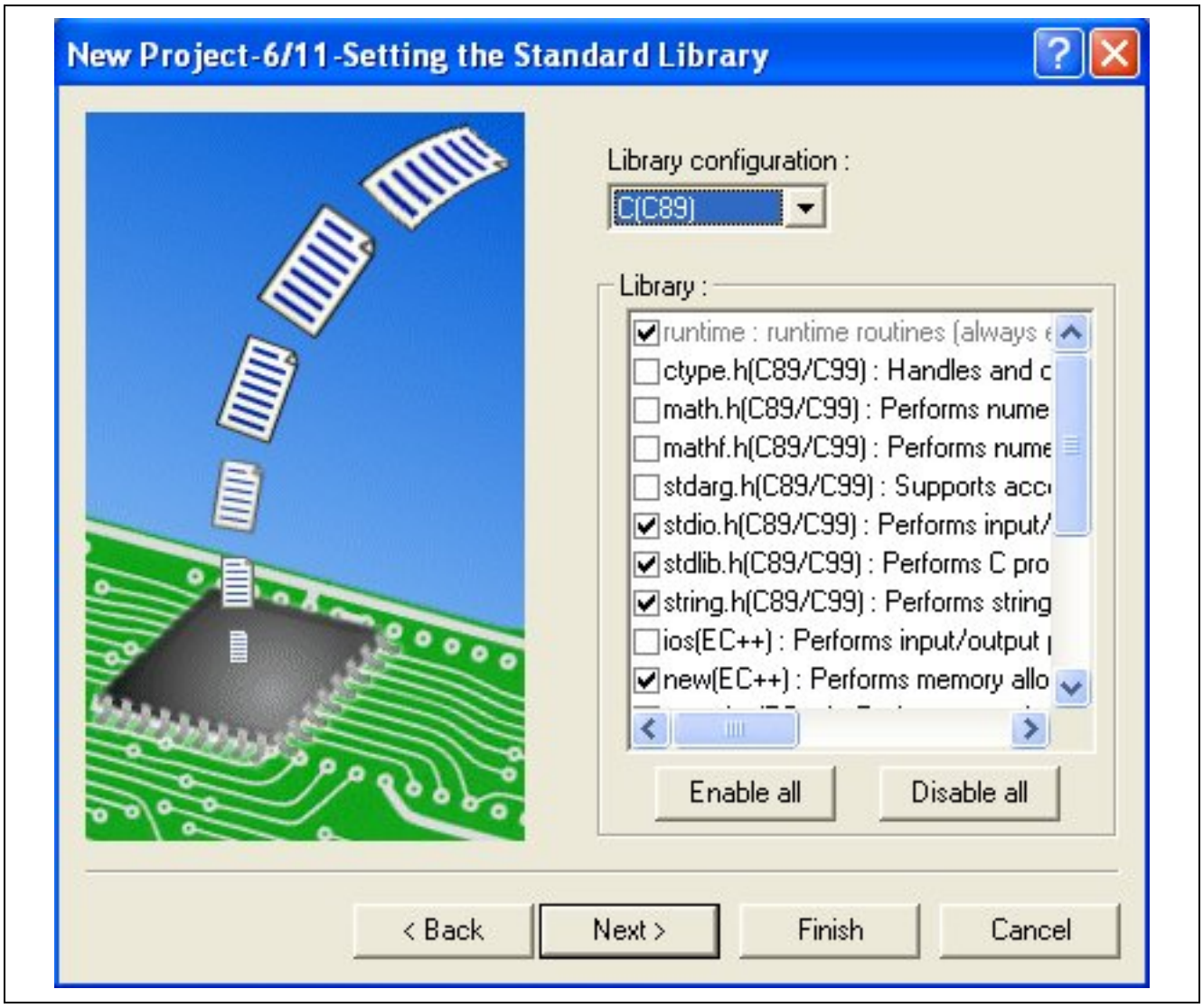
Accept the defaults:



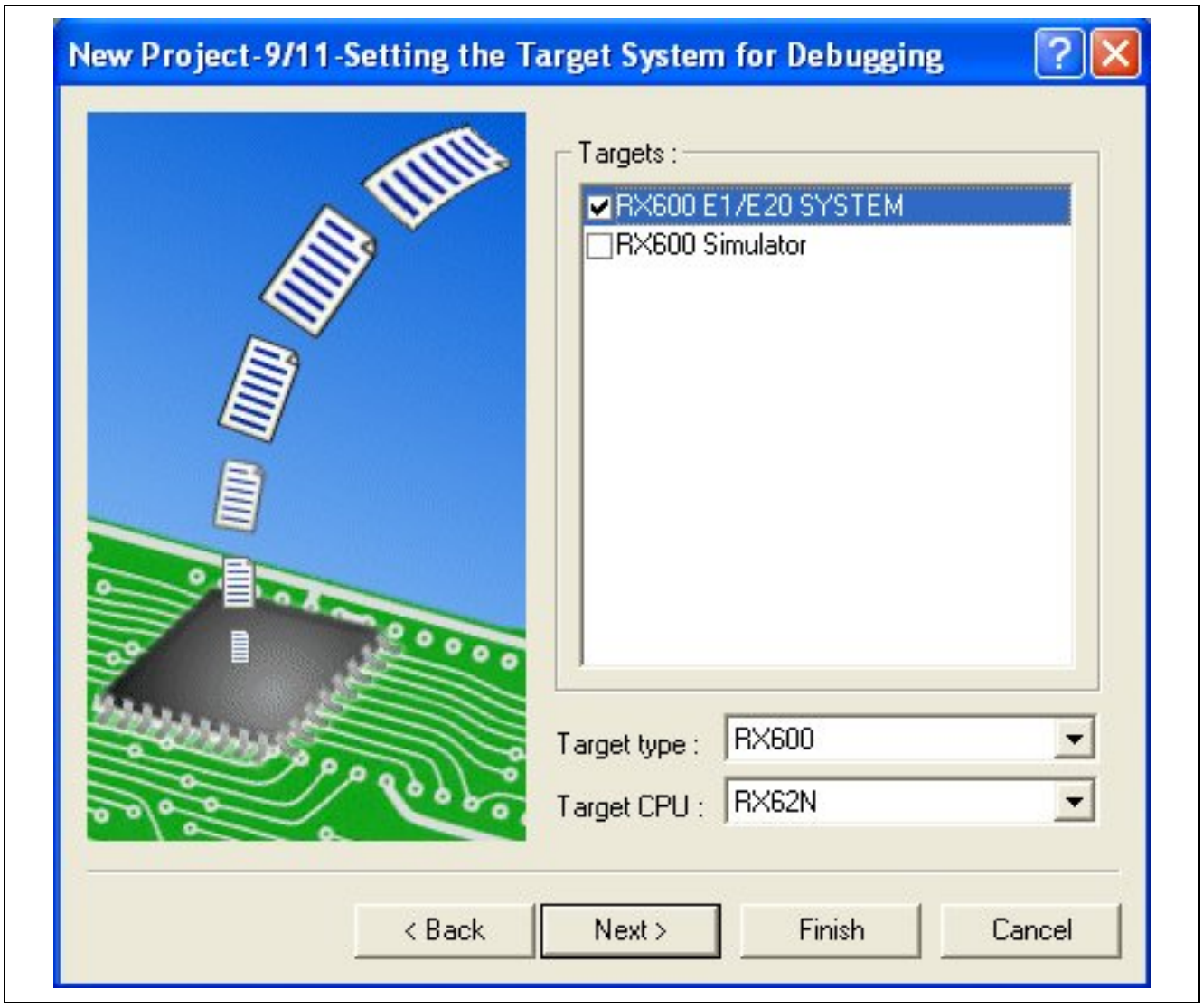
Ensure that the ‘C/C++ Source File’ is selected:



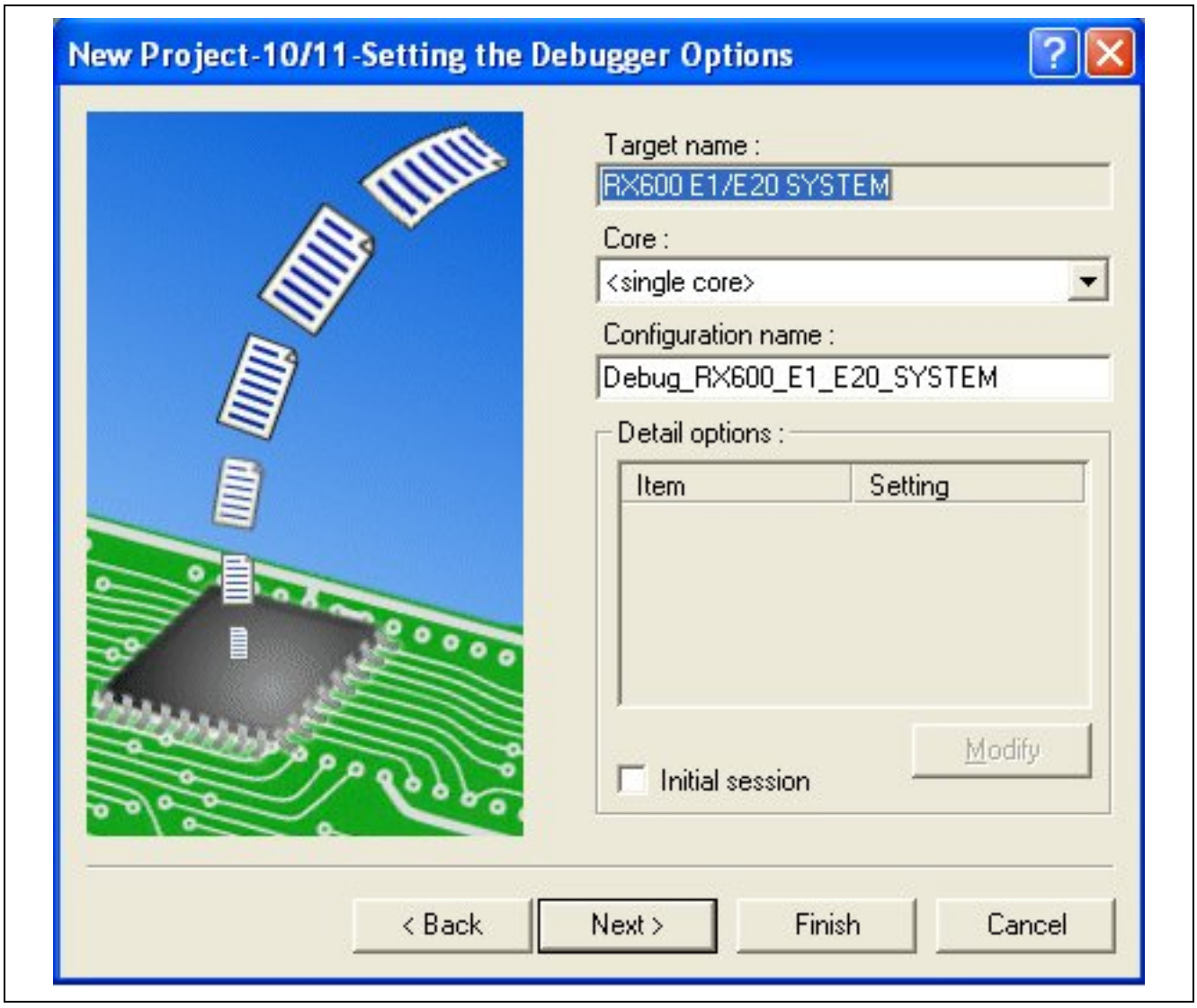
Accept the defaults for the Standard Library configuration:



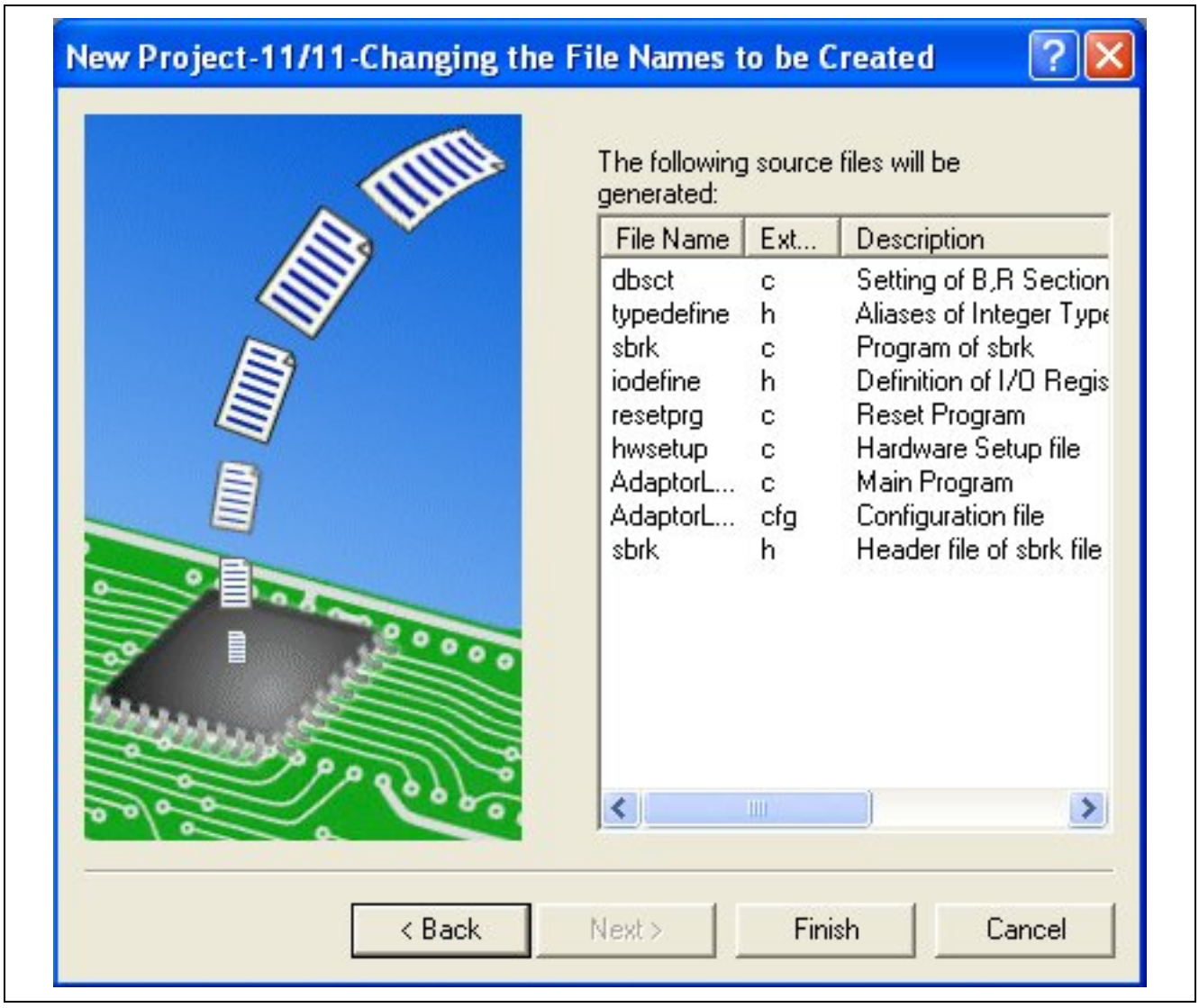
Clicking next moves forward to step 9 for the selection of the emulator to run:



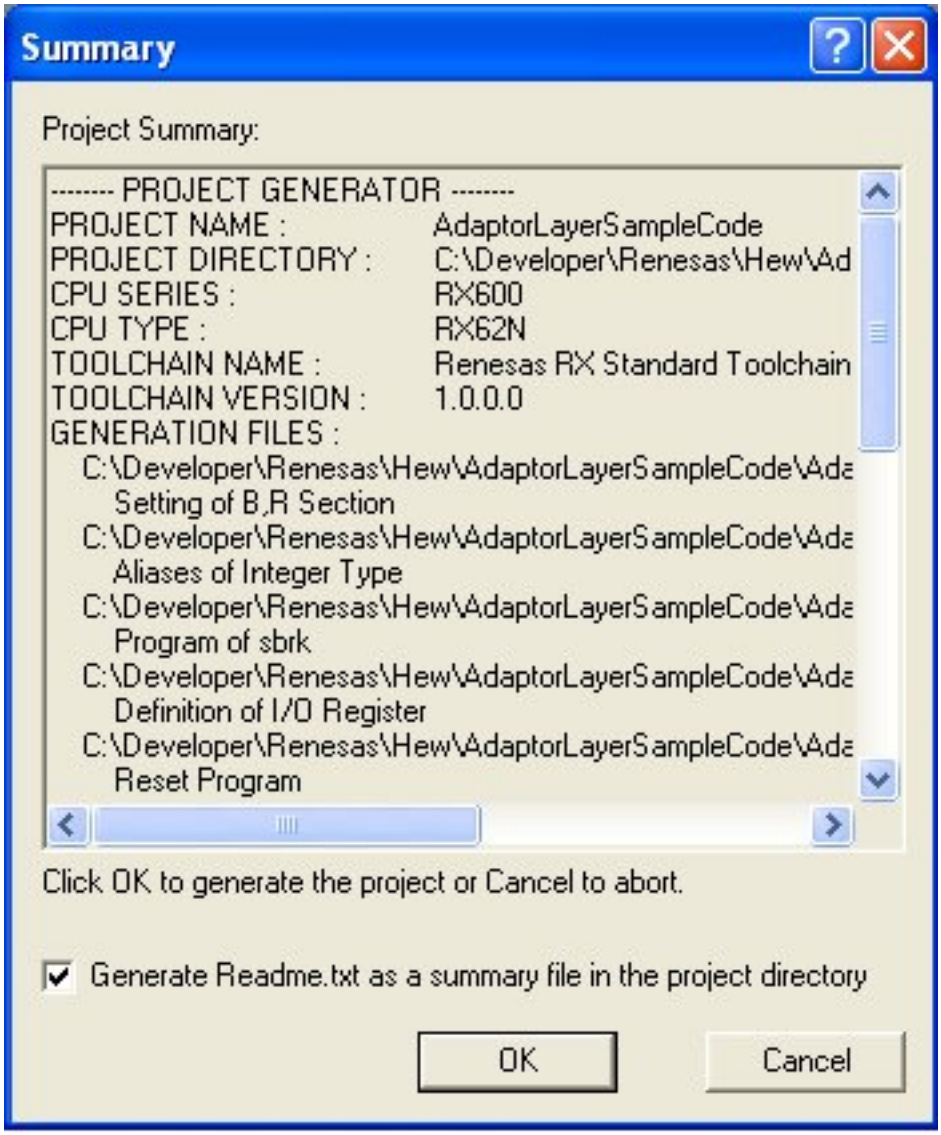
Accept the defaults for step 10 as shown:



Click ‘Finish’ to complete creation of the project workspace:

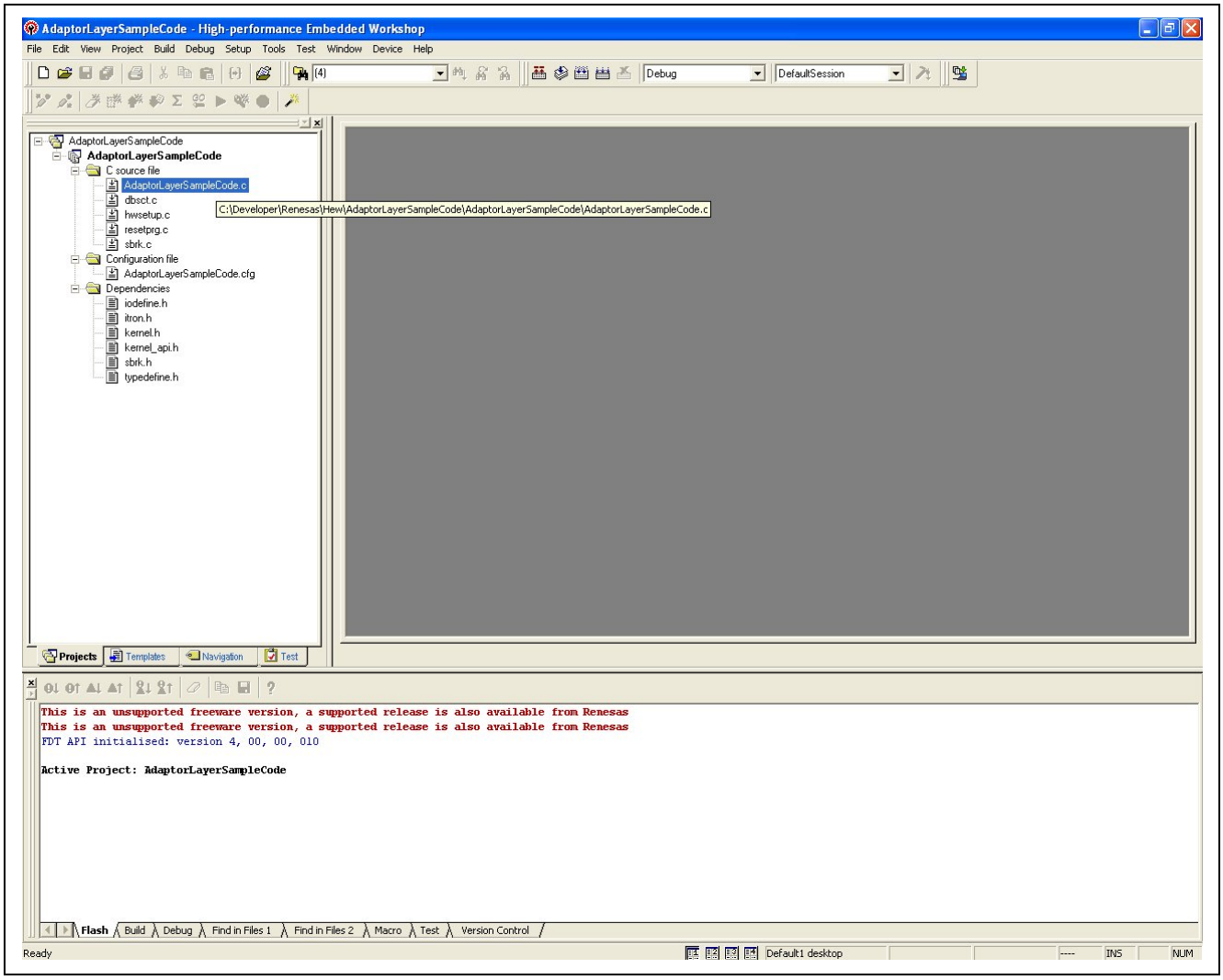


Click ‘OK’ to accept the summary of the project creation workspace:



2.3 Creating a LED Blinky Example Code

Open the file labeled ‘AdaptorLayerSampleCode.c’ as shown in the project workspace:

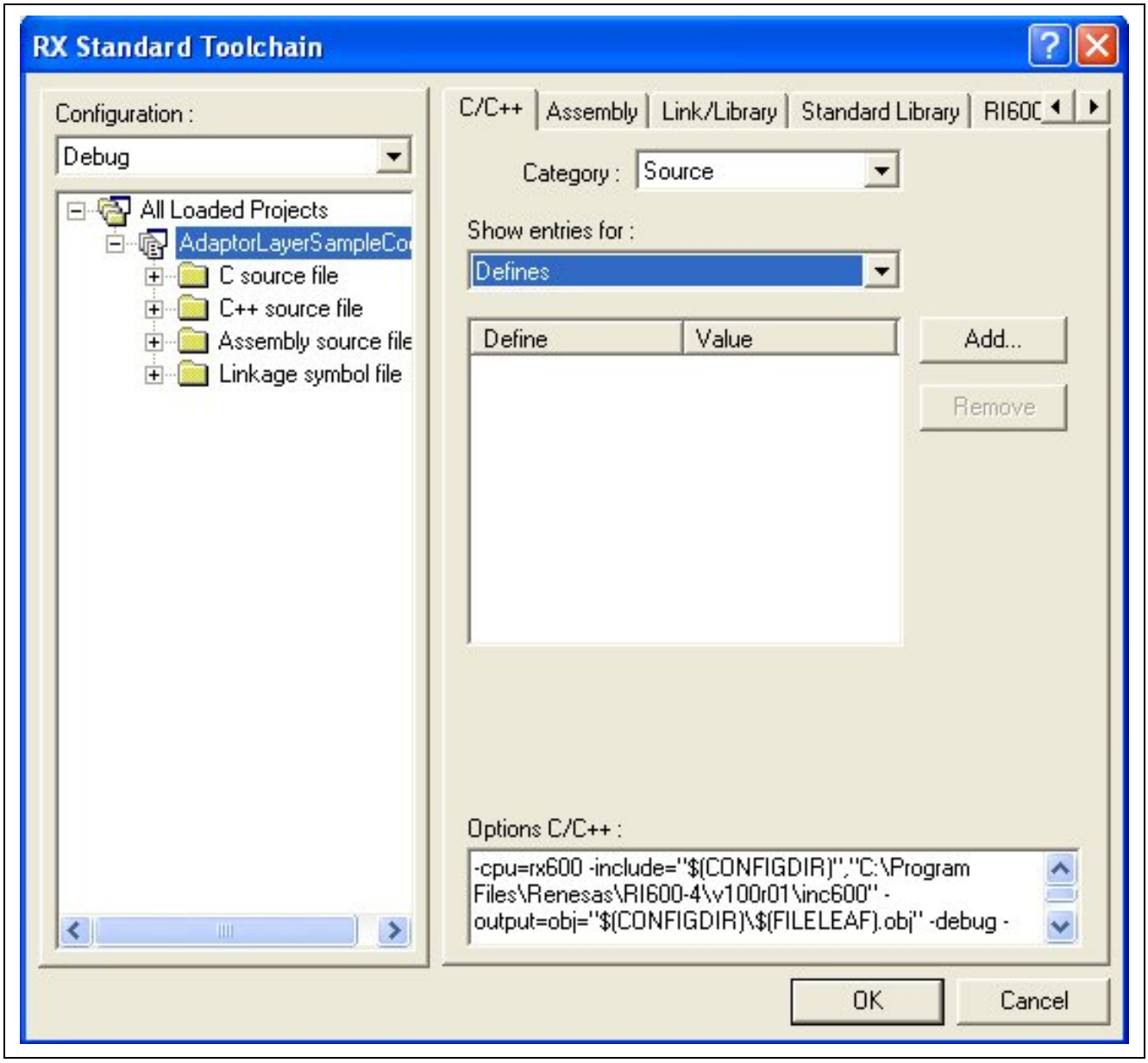


Replace the entire sample code with the following:

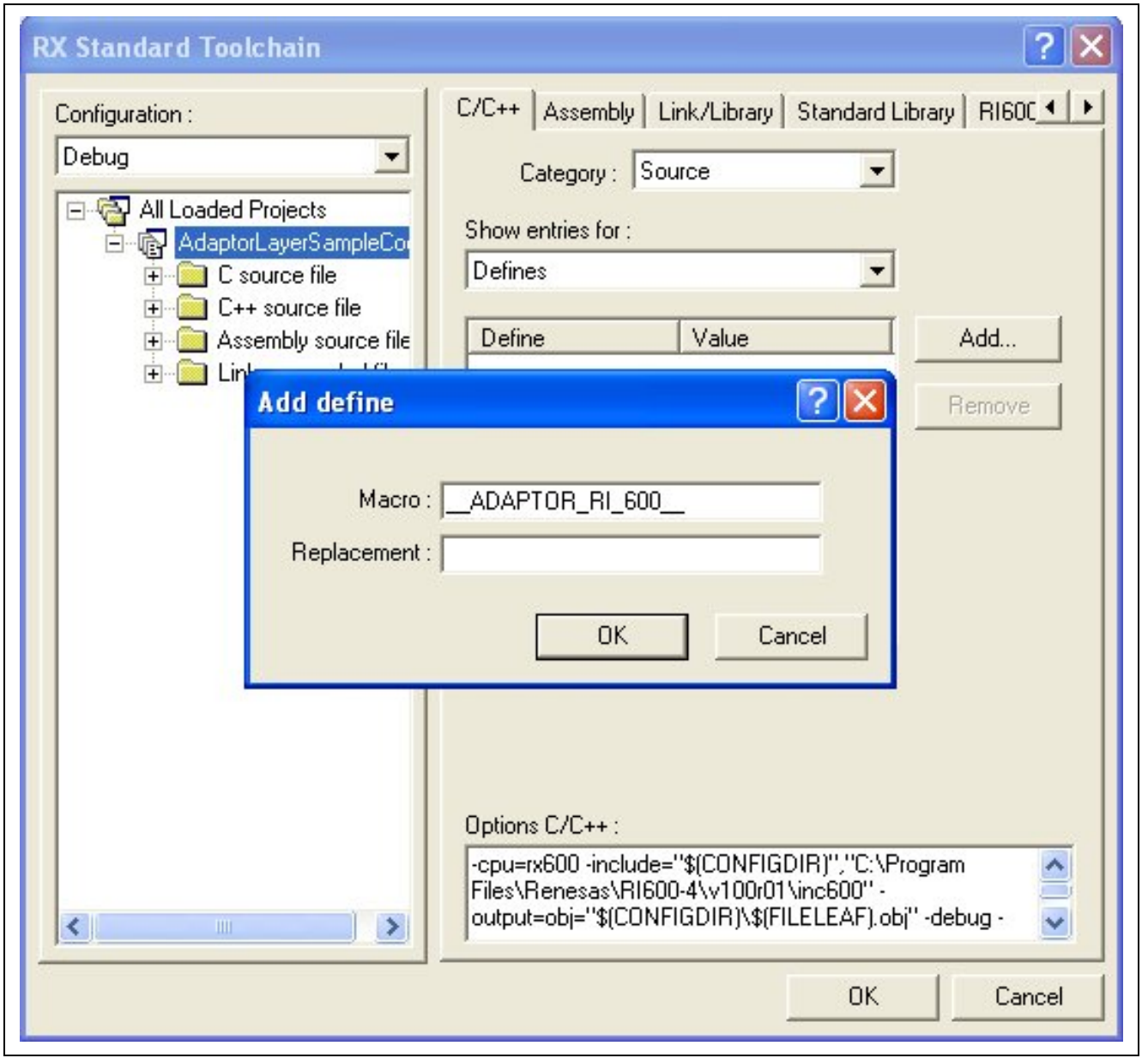
```
#include <machine.h>
#include <stdio.h>
#if defined (__ADAPTOR_RI_600__)
  #include "kernel.h"
  #include "kernel_id.h"
#elif defined (__ADAPTOR_FREE_RTOS__)
  #include "r_FreeRTOSAdaptor.h"
#endif

#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);
  }
}
```

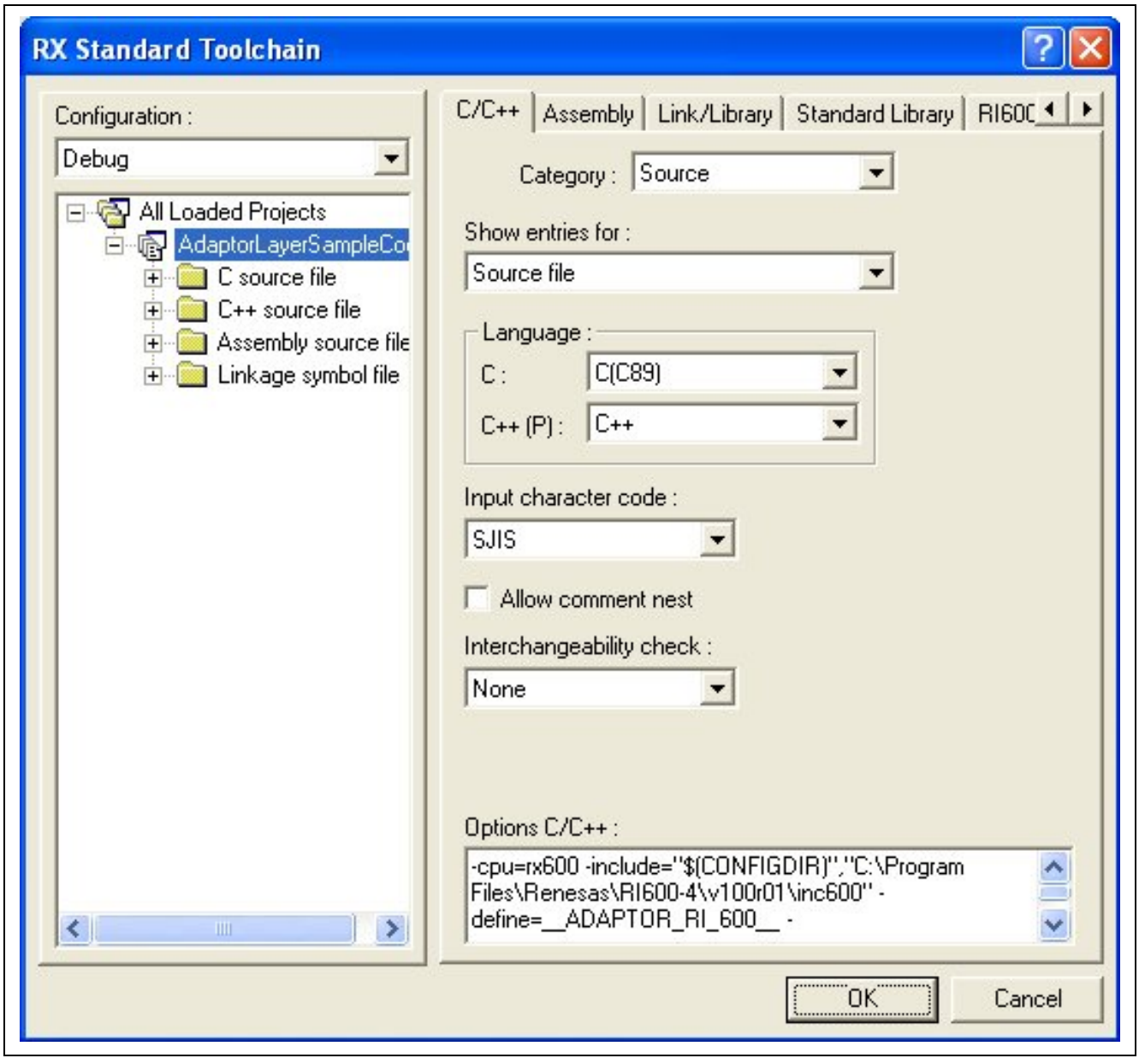
Add the defined macro for the RI-600 Project under Build→RX Standard Toolchain, C/C++ Tab, select: Show entries for: Defines. This is shown as such:



Add the global defined macro __ADAPTOR_RI_600__ as shown:

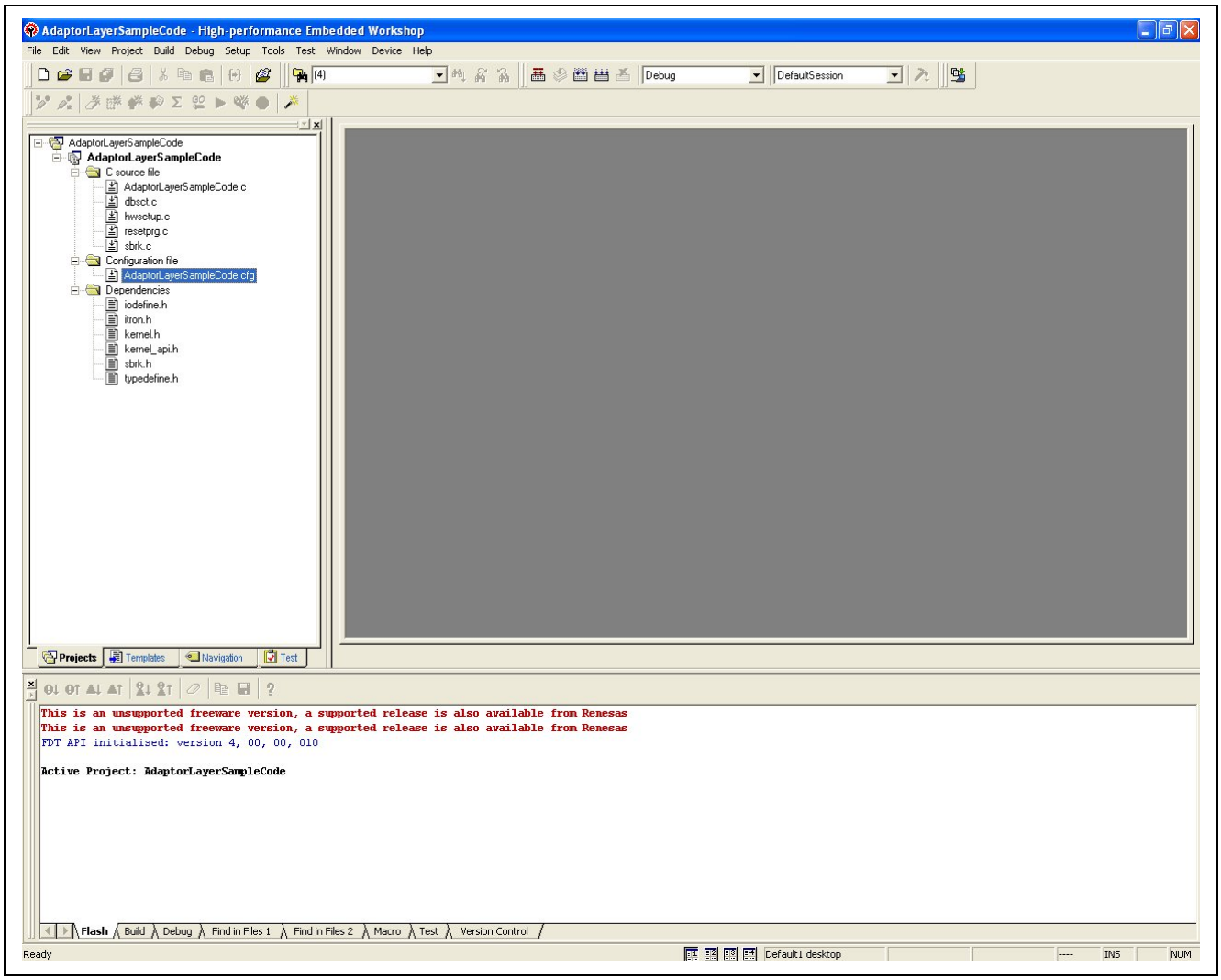


Select ‘OK’ and exit the RX Standard Toolchain options panel.



2.4 Editing the RI-600 Configurator File

Open the ‘AdaptorLayerSampleCode.cfg’ file as shown. This is the configurator file used by RI-600 during the compilation process.



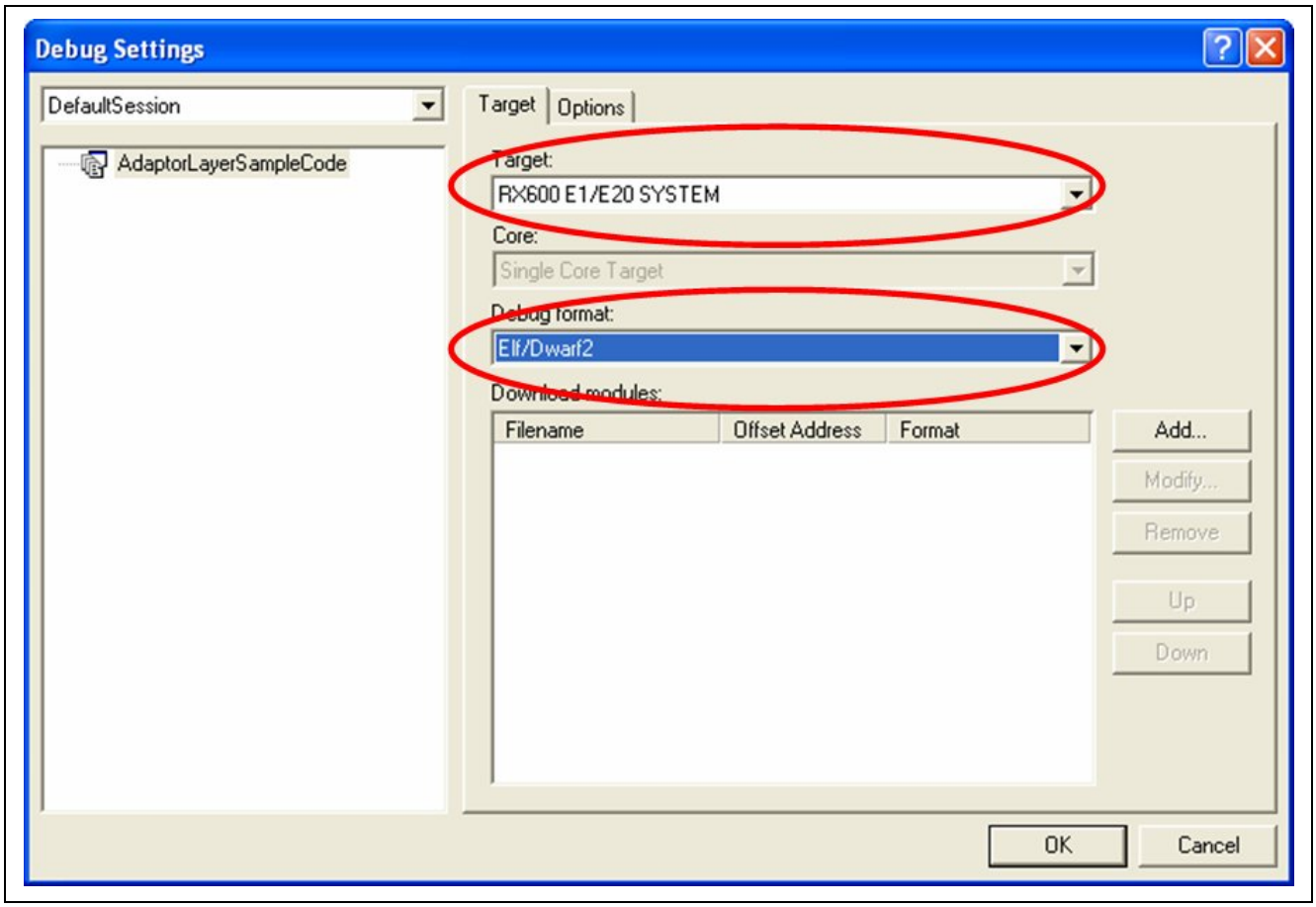
Replace the configuration with the following configuration:

```
//************************************************************
//      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;
};
```

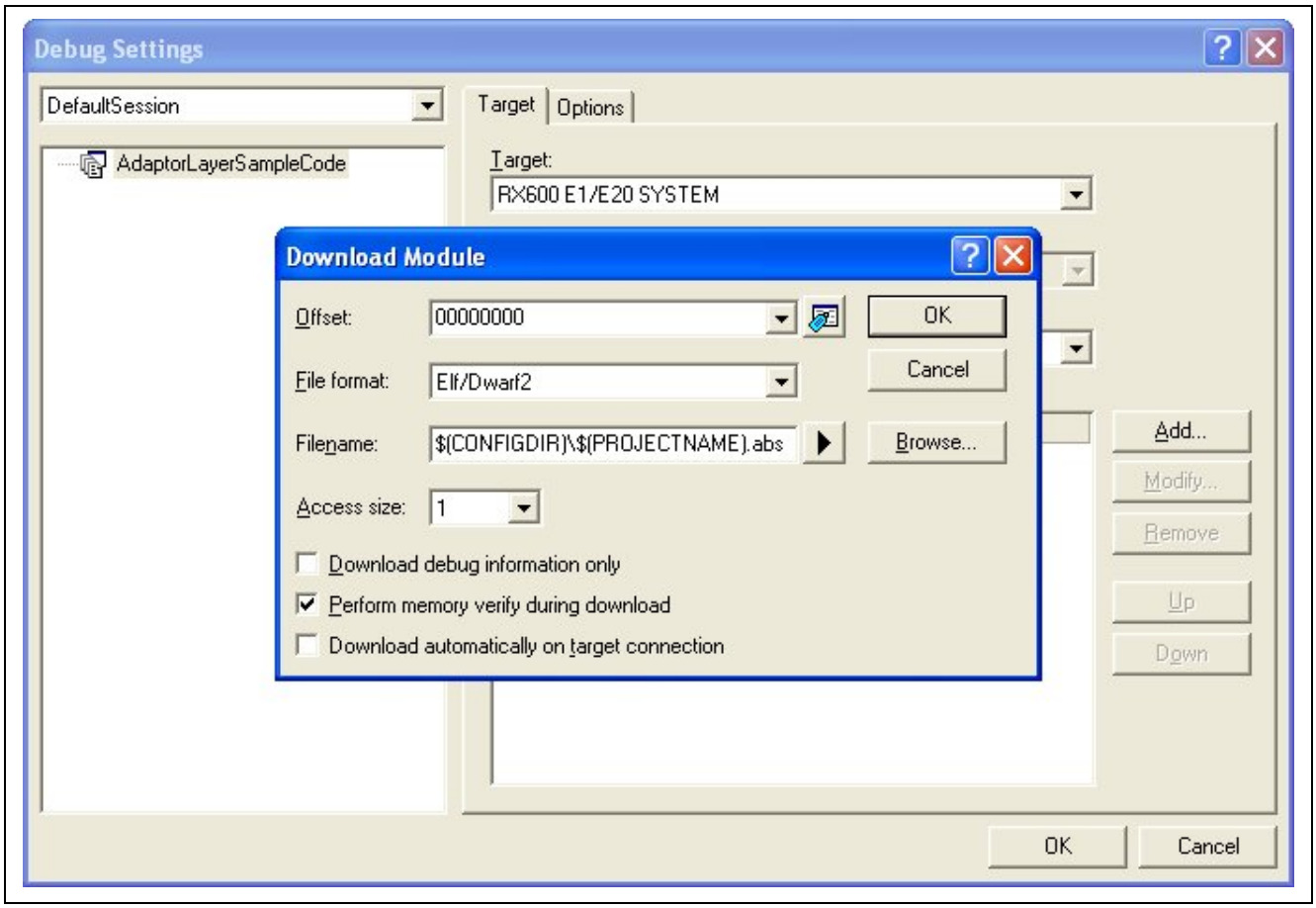
2.5 Connecting to the RX62N Board

Add the emulator to connect to the RX62N RSK Board as shown:

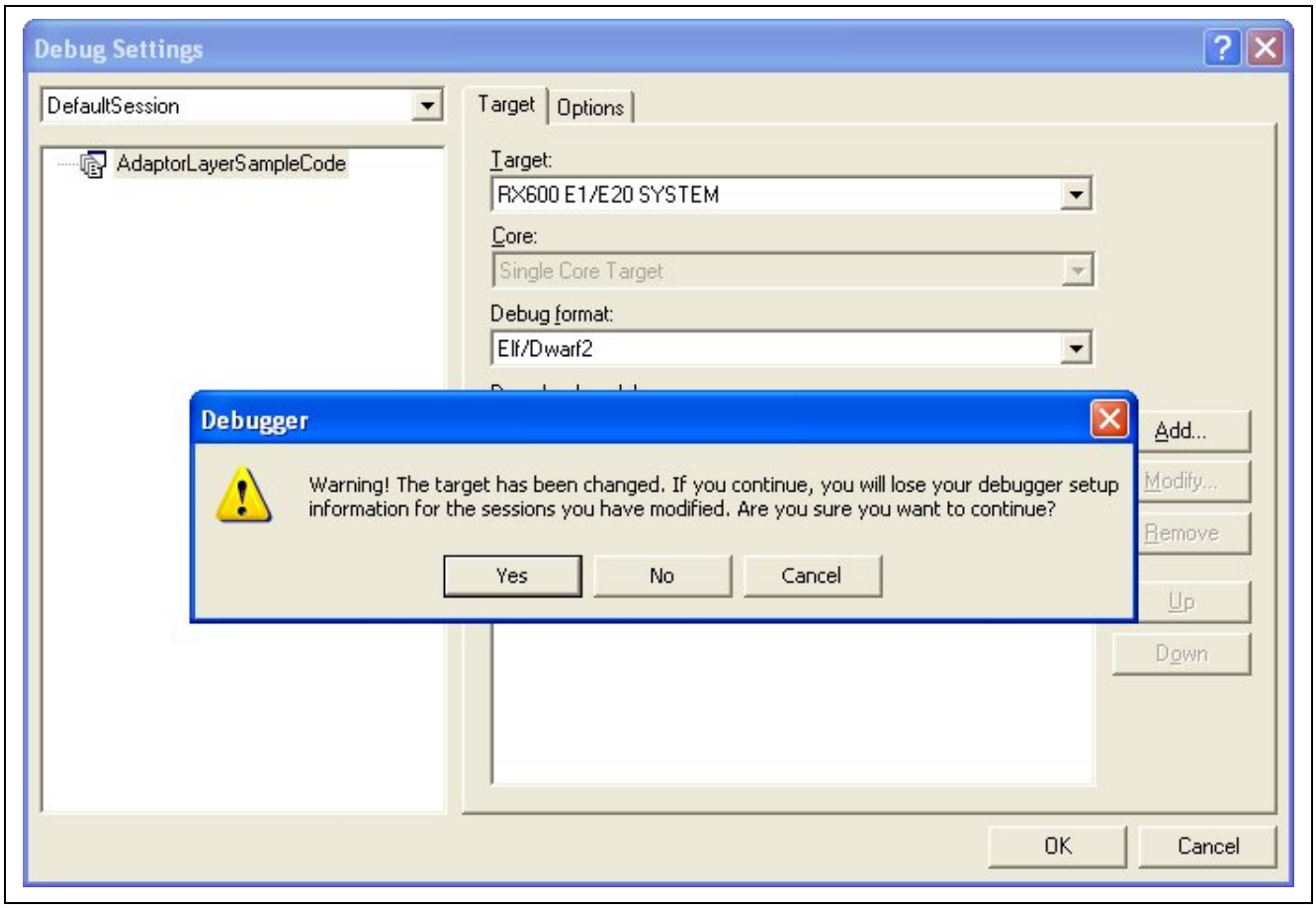
Add the following debug settings as shown:



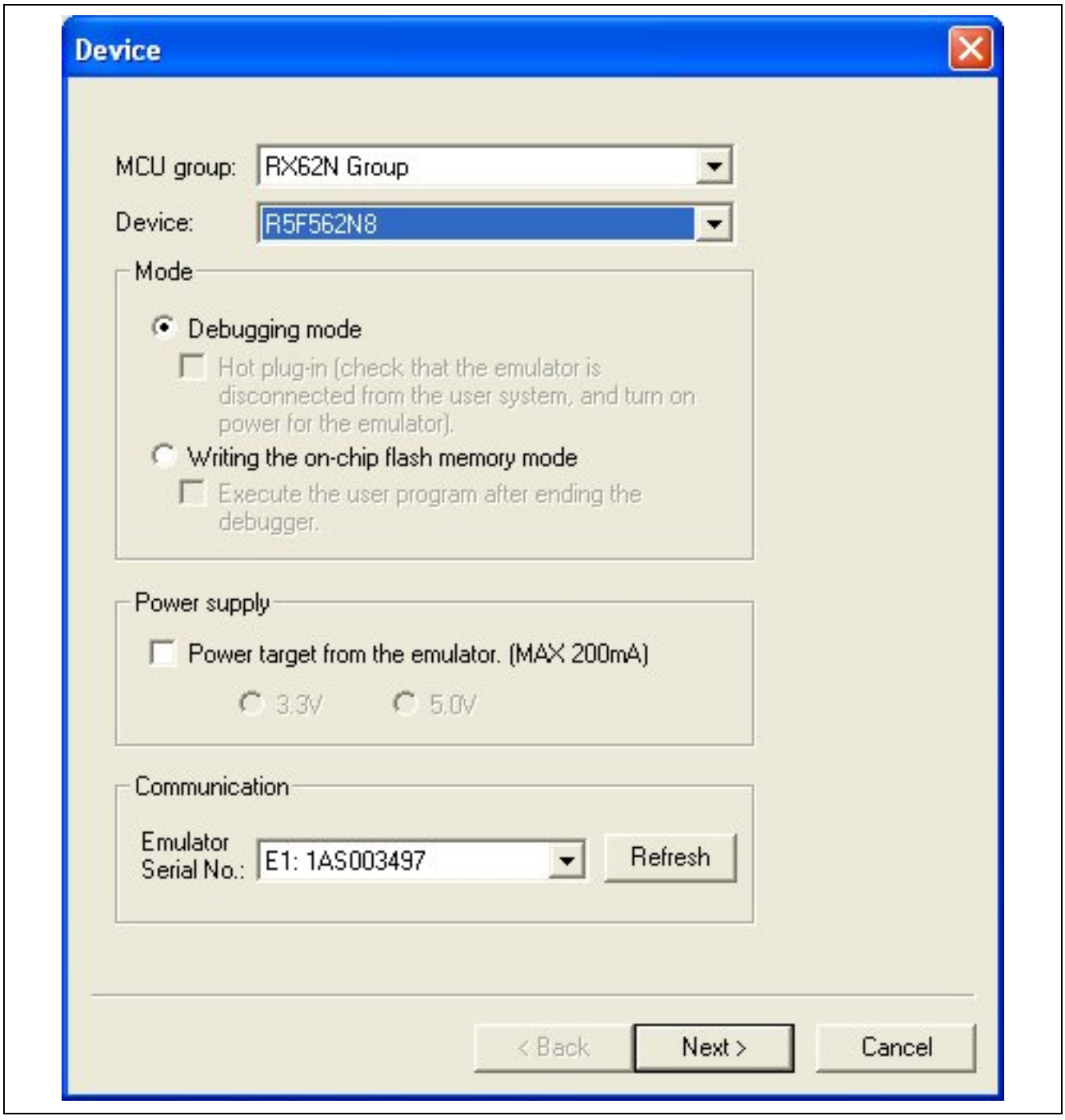
Click ‘Add’ to add the following configuration:



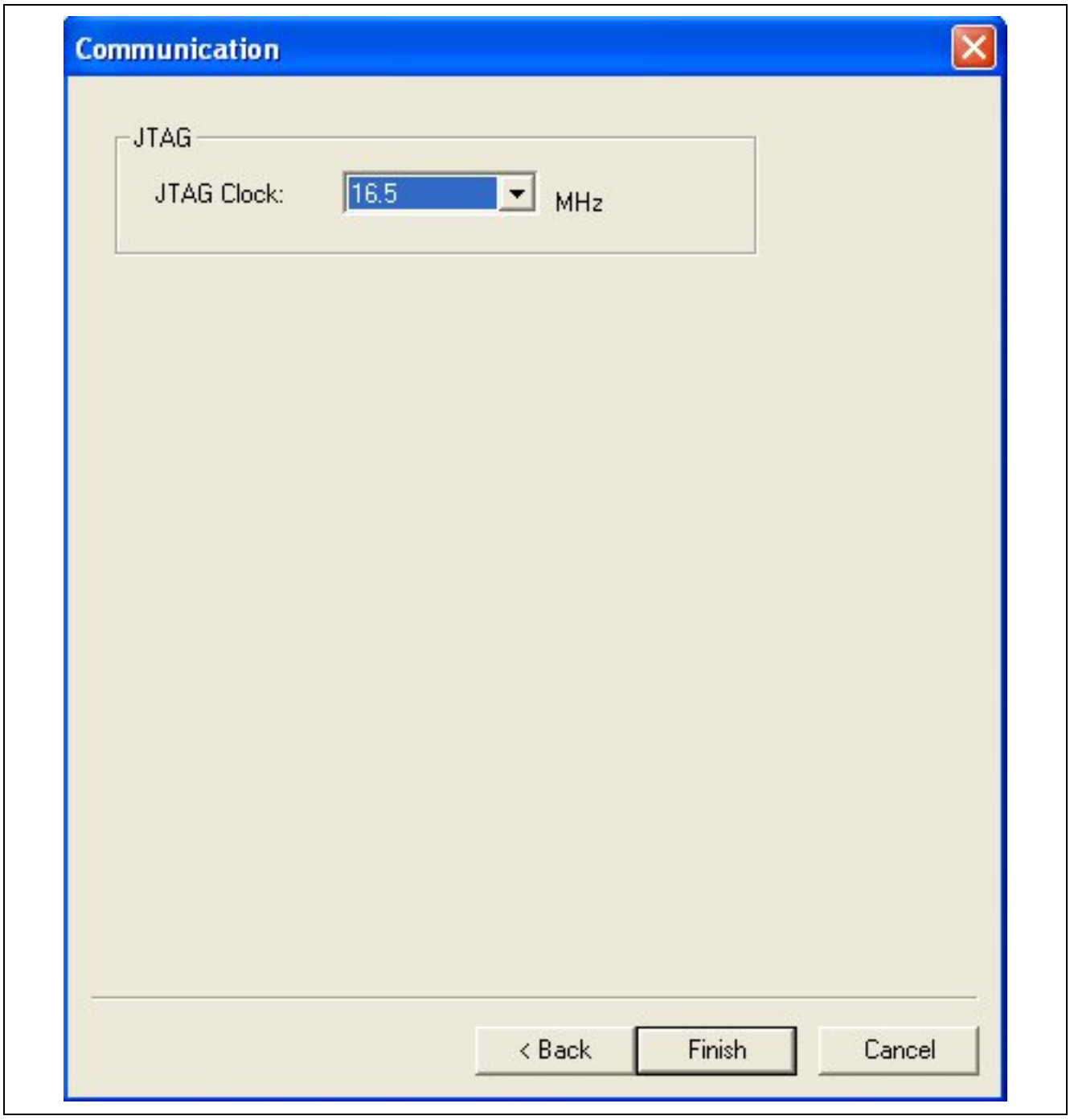
Click ‘OK’ to accept the default settings and ‘Yes’ when this Debugger warning panel is being displayed:



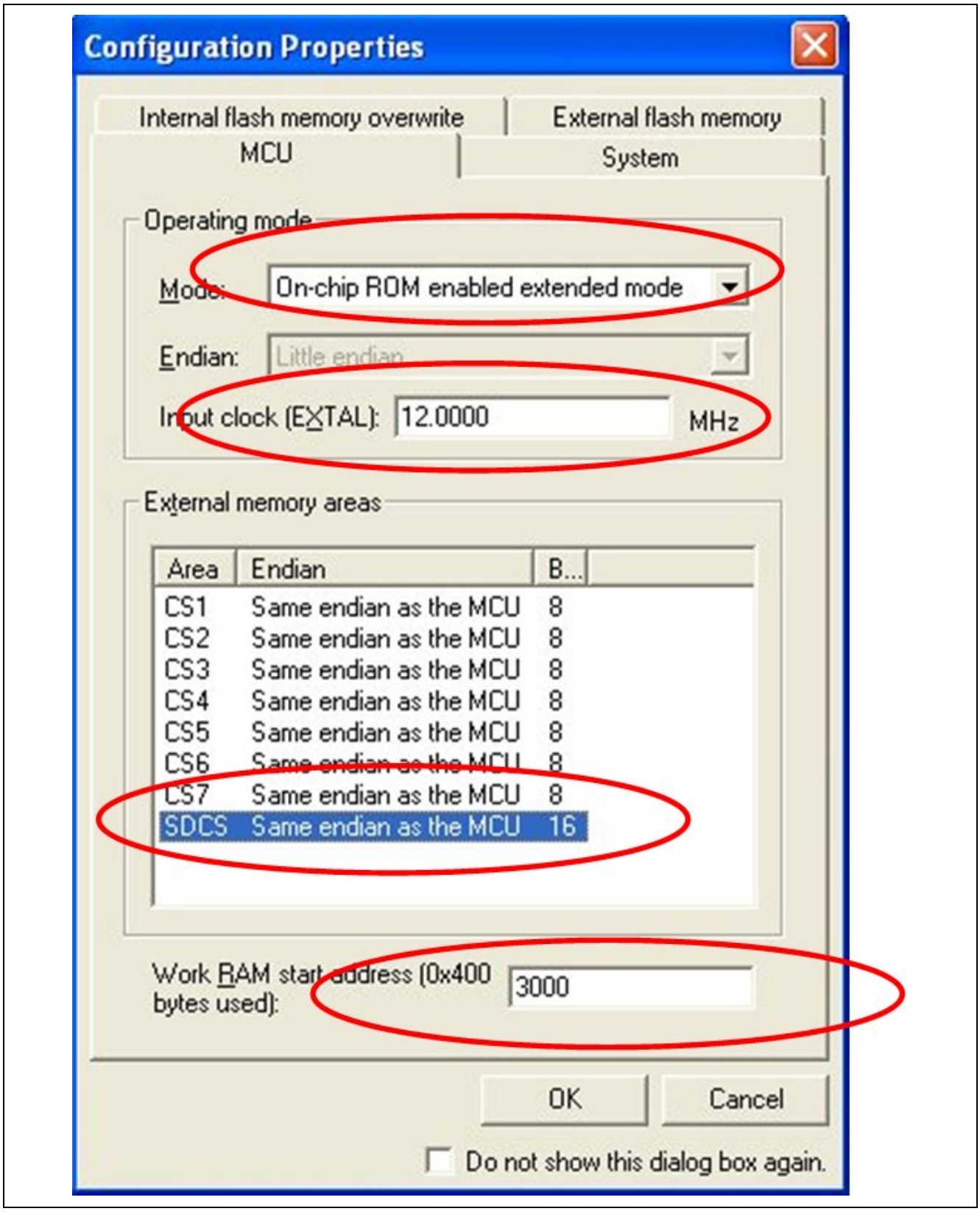
Select the following in the Device window:



Ensure that the board is powered before clicking ‘Next >’:



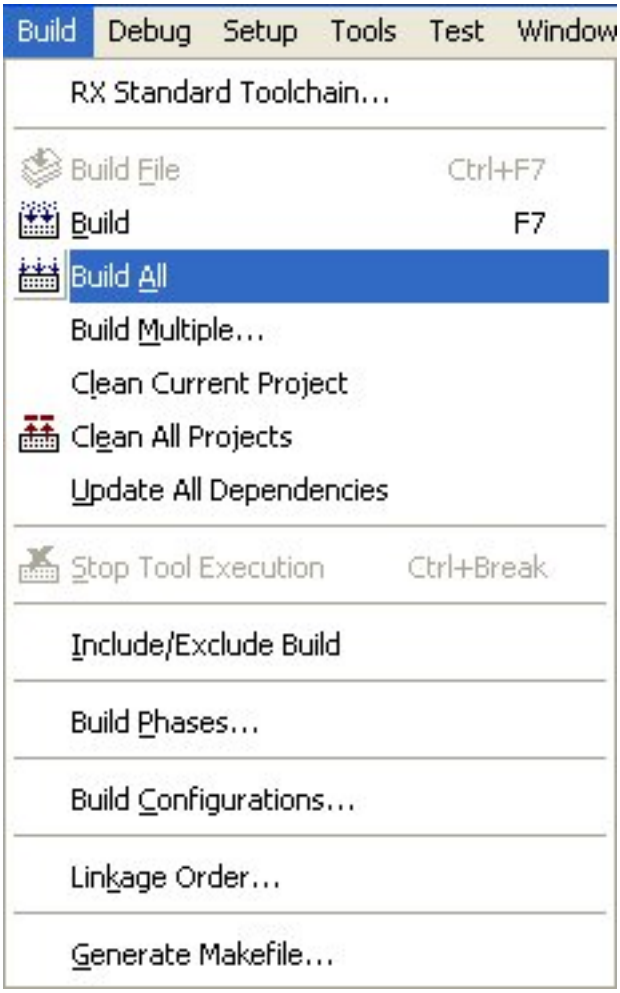
Click ‘Finish’ to begin connection to the device. Select the following settings in the next window:



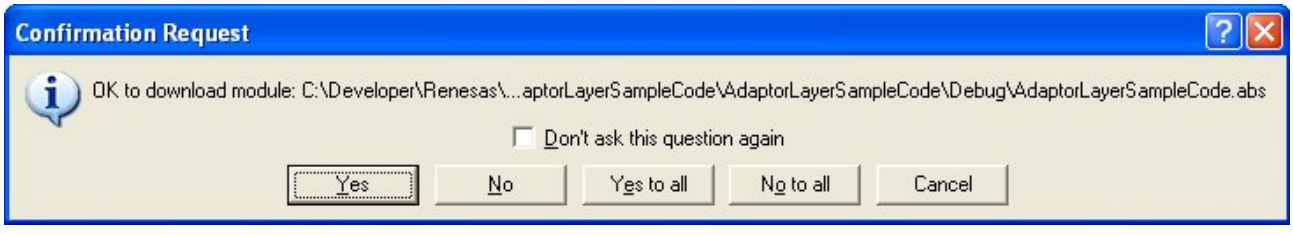
Click ‘OK’ to proceed connecting to the device.

2.6 Compile and Build LED Blinky Example

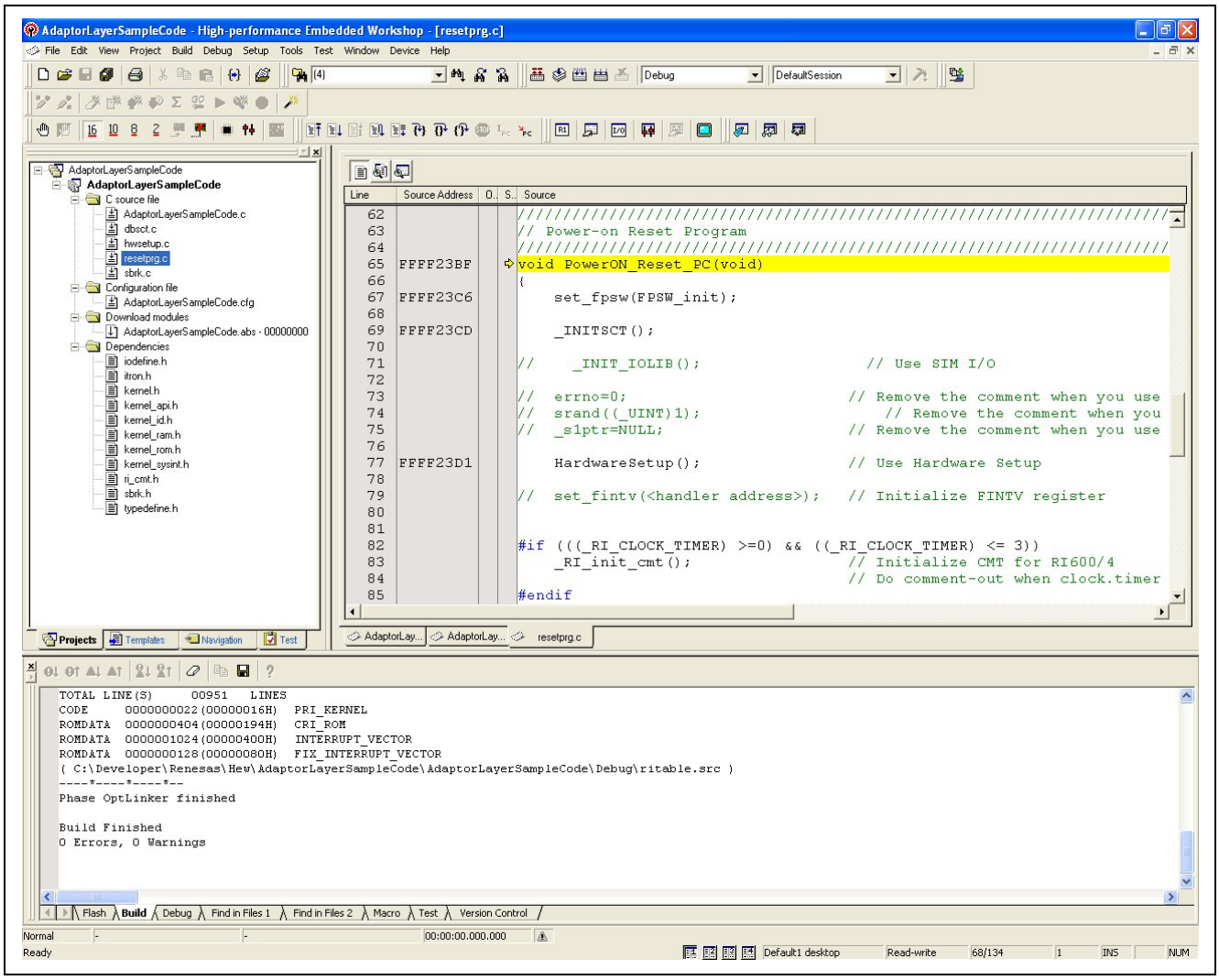
Select Build→Build All to being compilation:



When the compilation is done, select 'Yes to all' to download the built module:

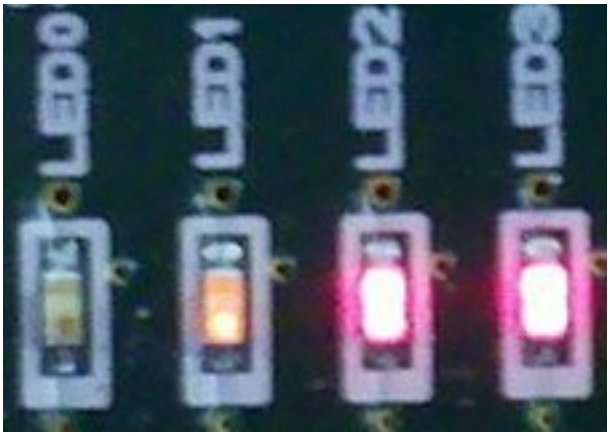


When the module has been downloaded, program is ready for execution as shown:



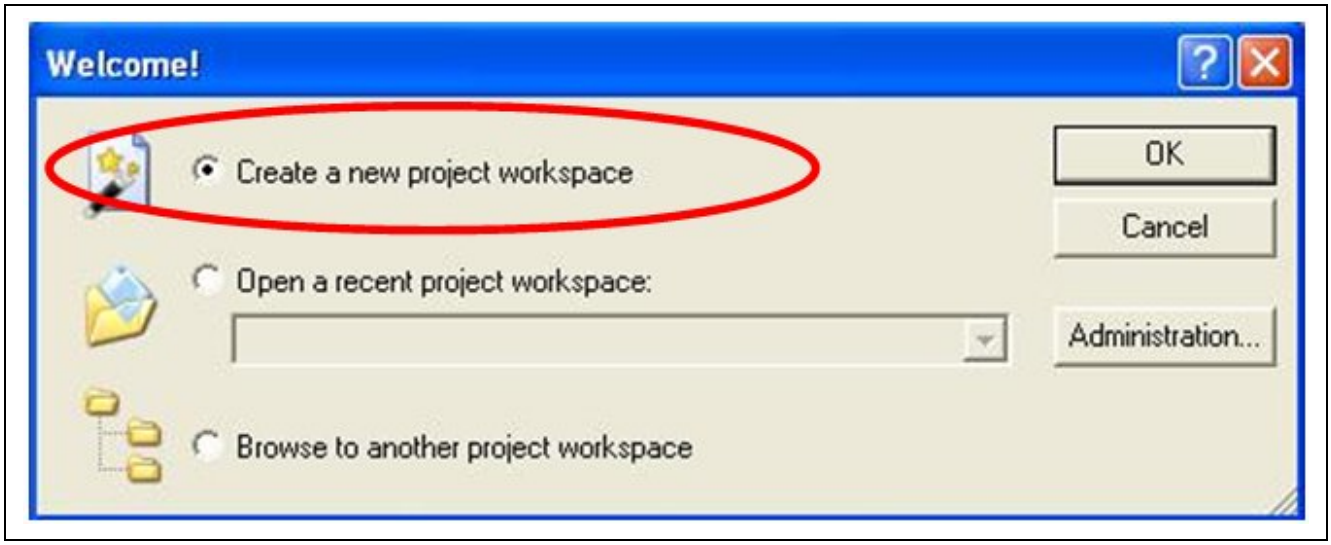
In order to run the program, press F5 or select Debug→Go as shown:

Finally, the LEDs on the RSK should be blinking as such:

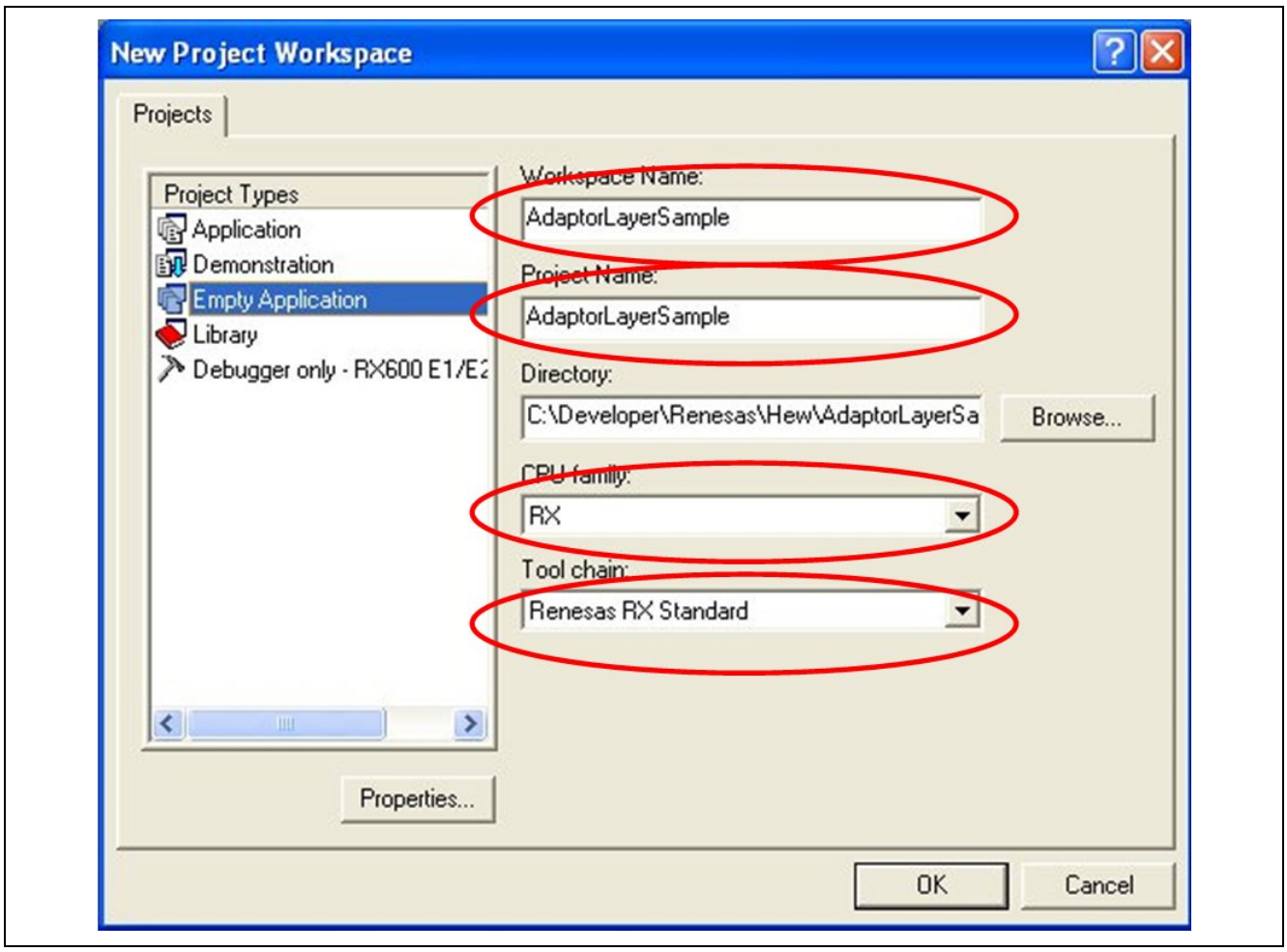


2.7 Creating an Empty Workspace for Adaptor Layer

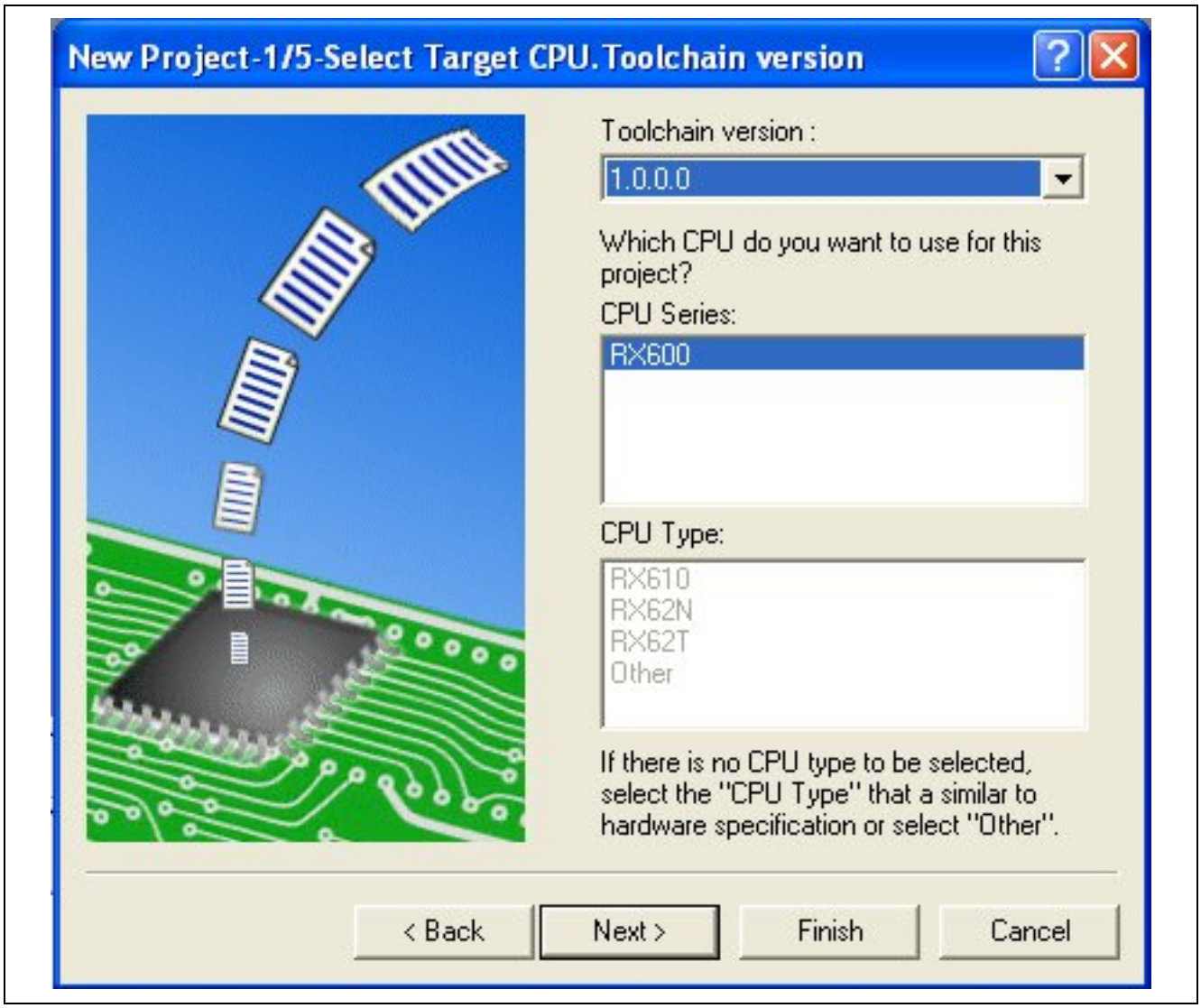
Launch Renesas HEW IDE and create a new project workspace as follows:



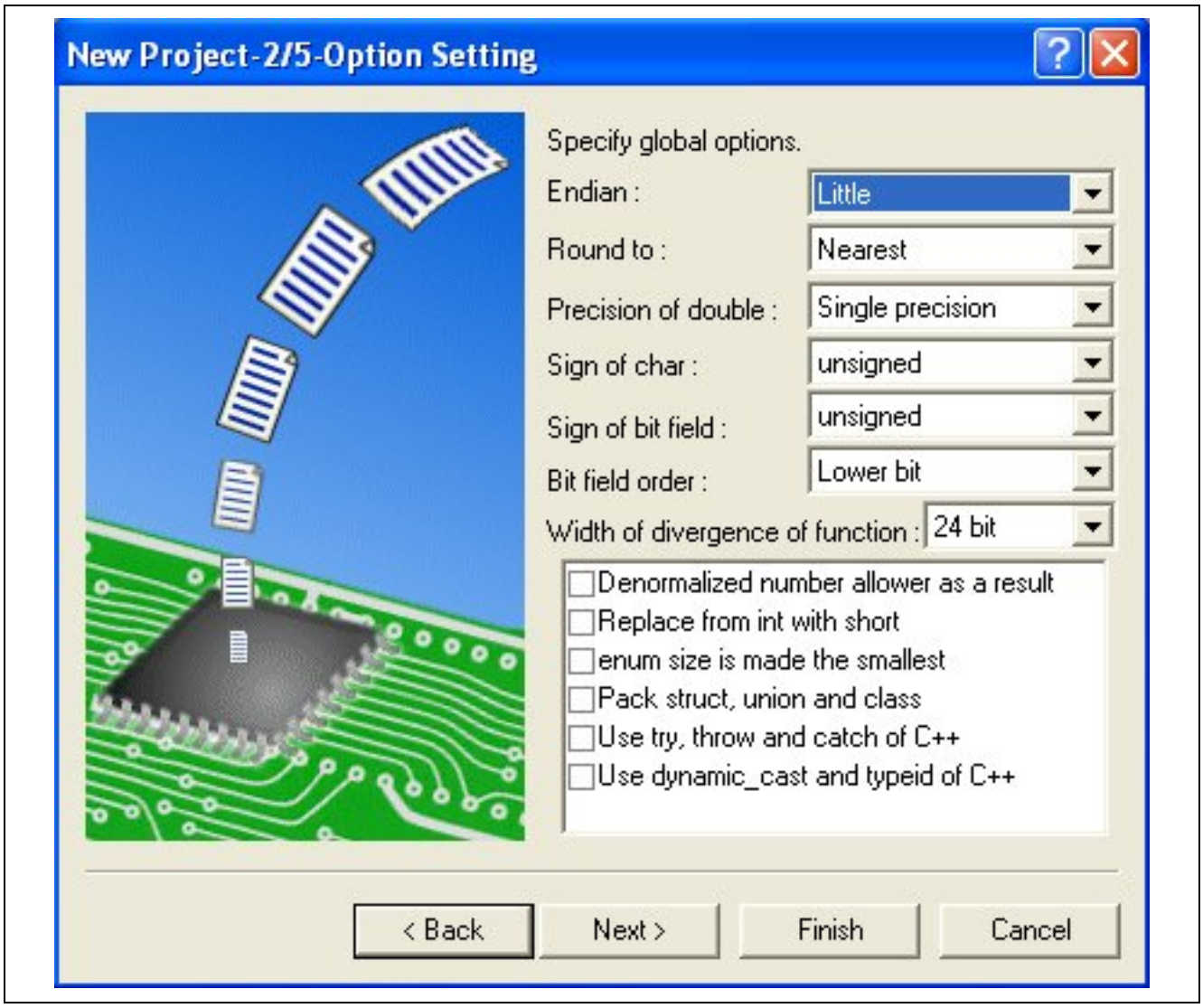
Enter a valid workspace and project name as follows:



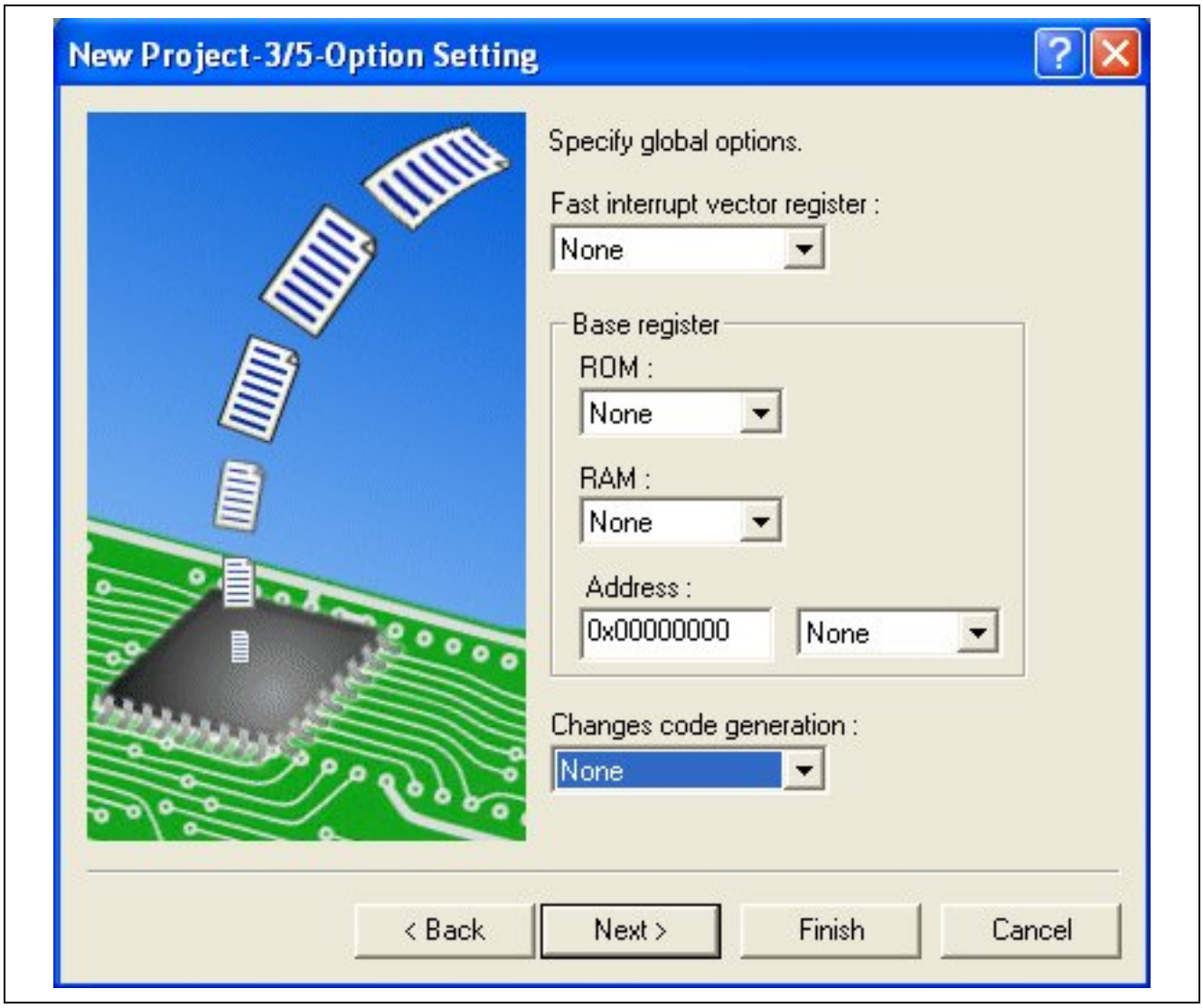
Select the toolchain as shown:



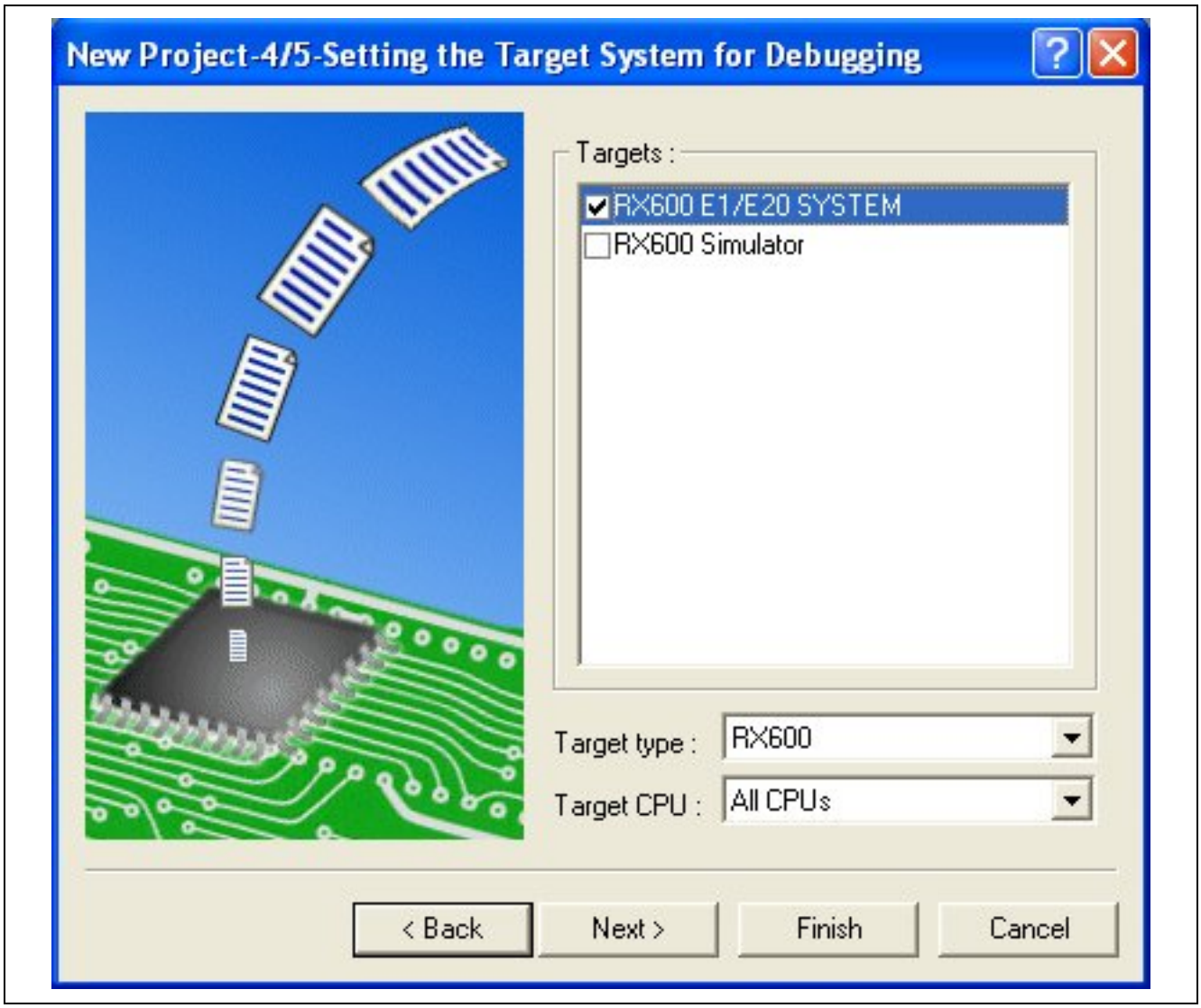
Accept the settings in the following windows:



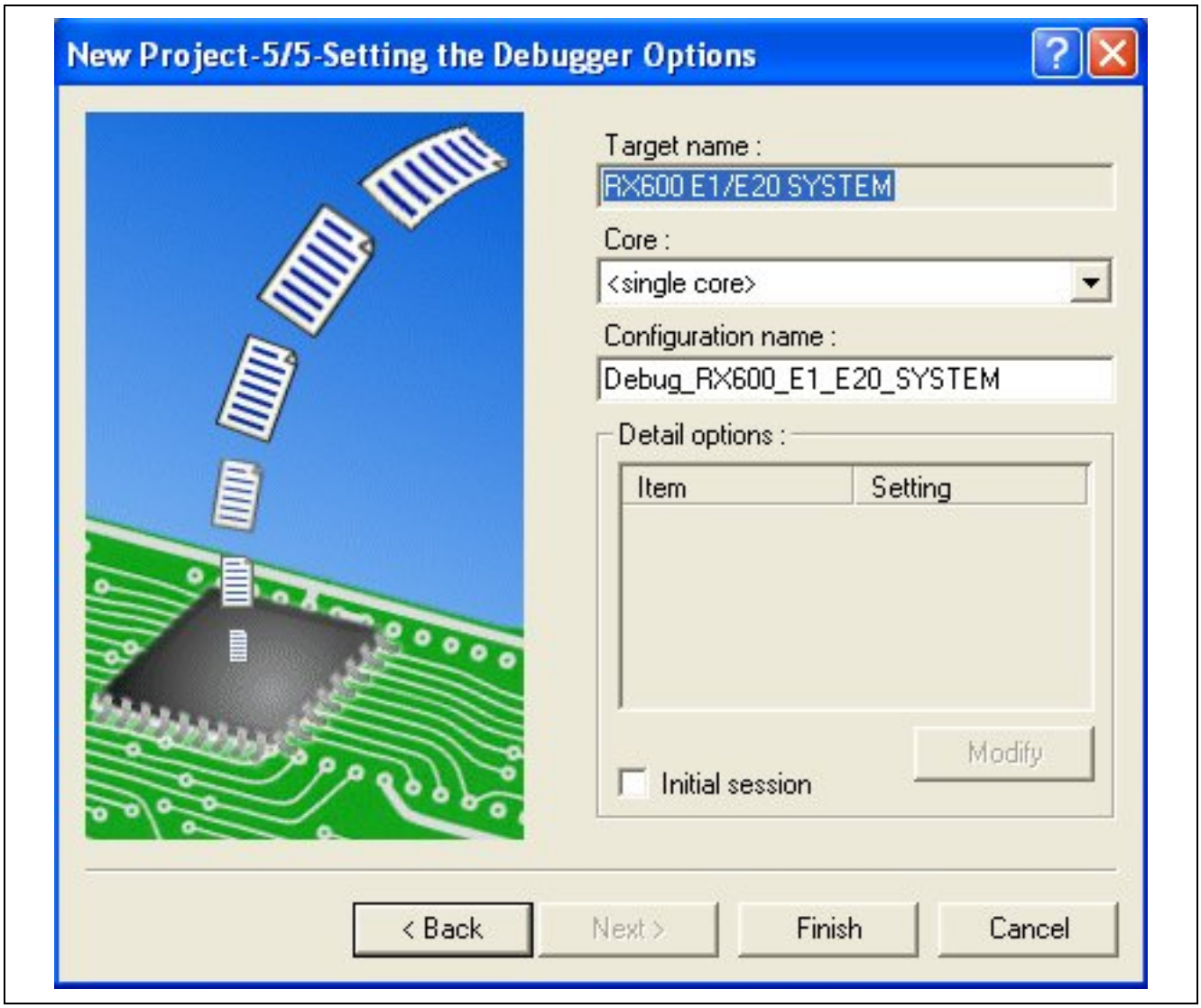
Accept the defaults:



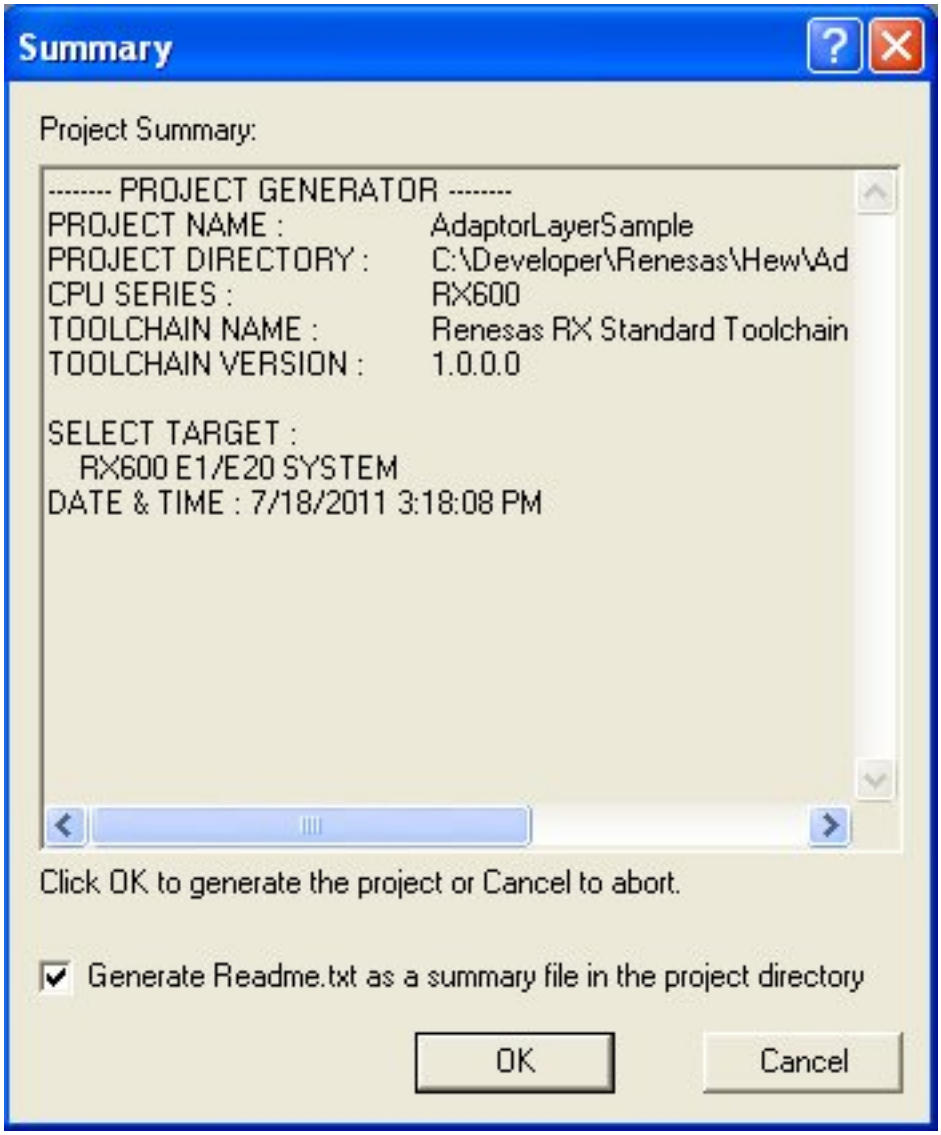
Accept the defaults:



Click ‘Finish’ to generate the project workspace:

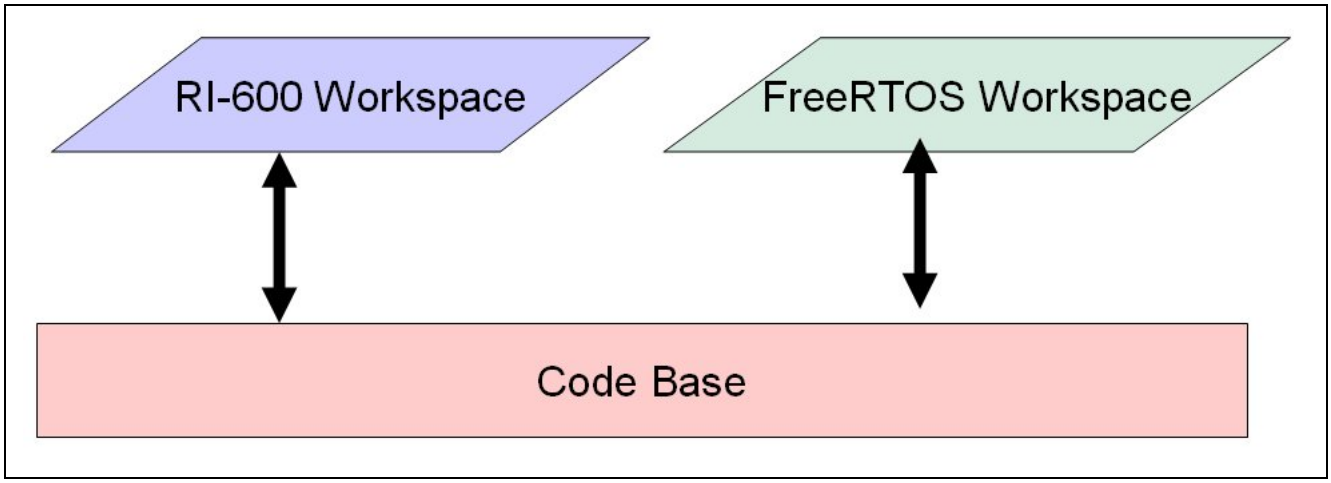


Click ‘OK’ on the prompt as shown:

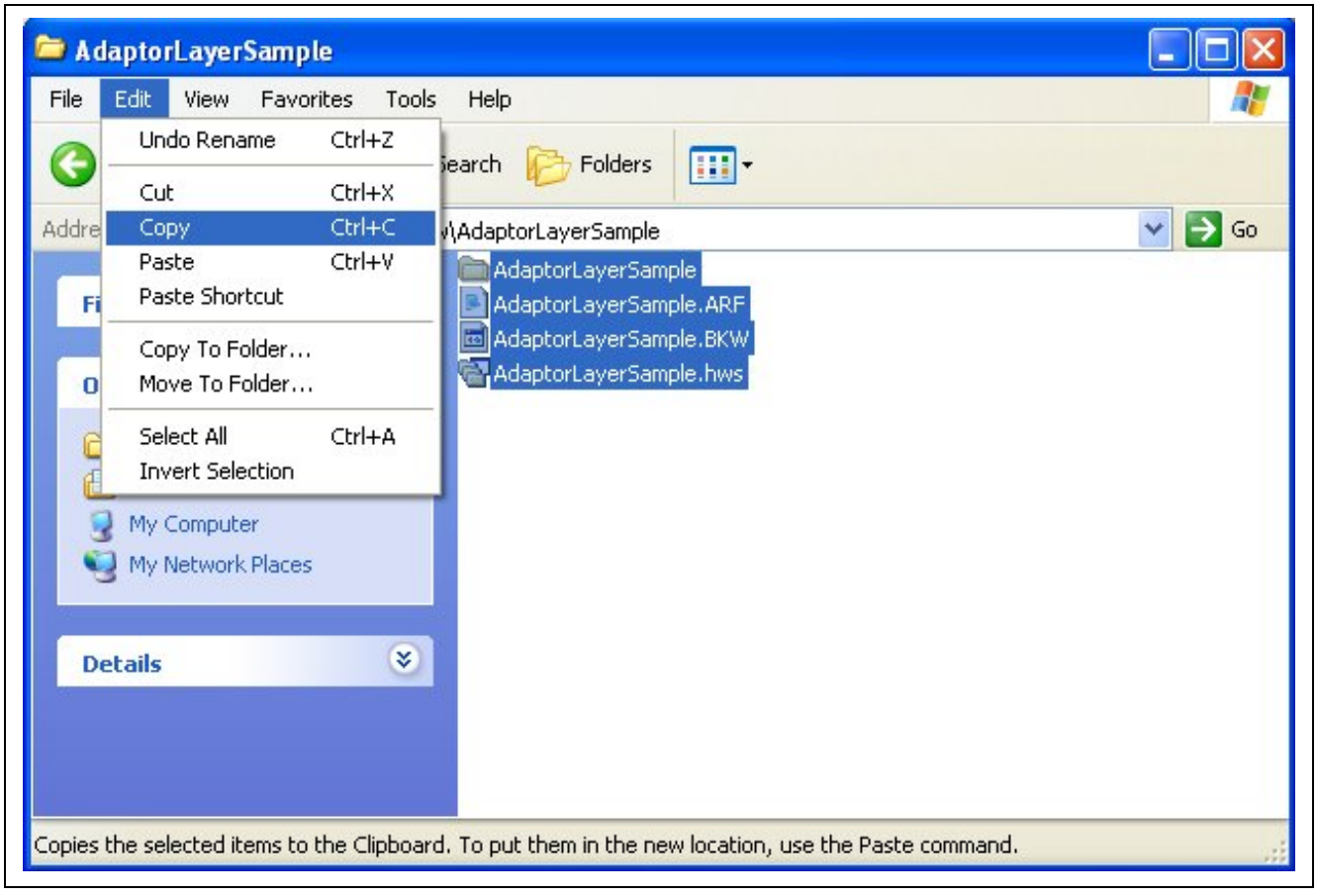


2.8 Combine Workspace Files

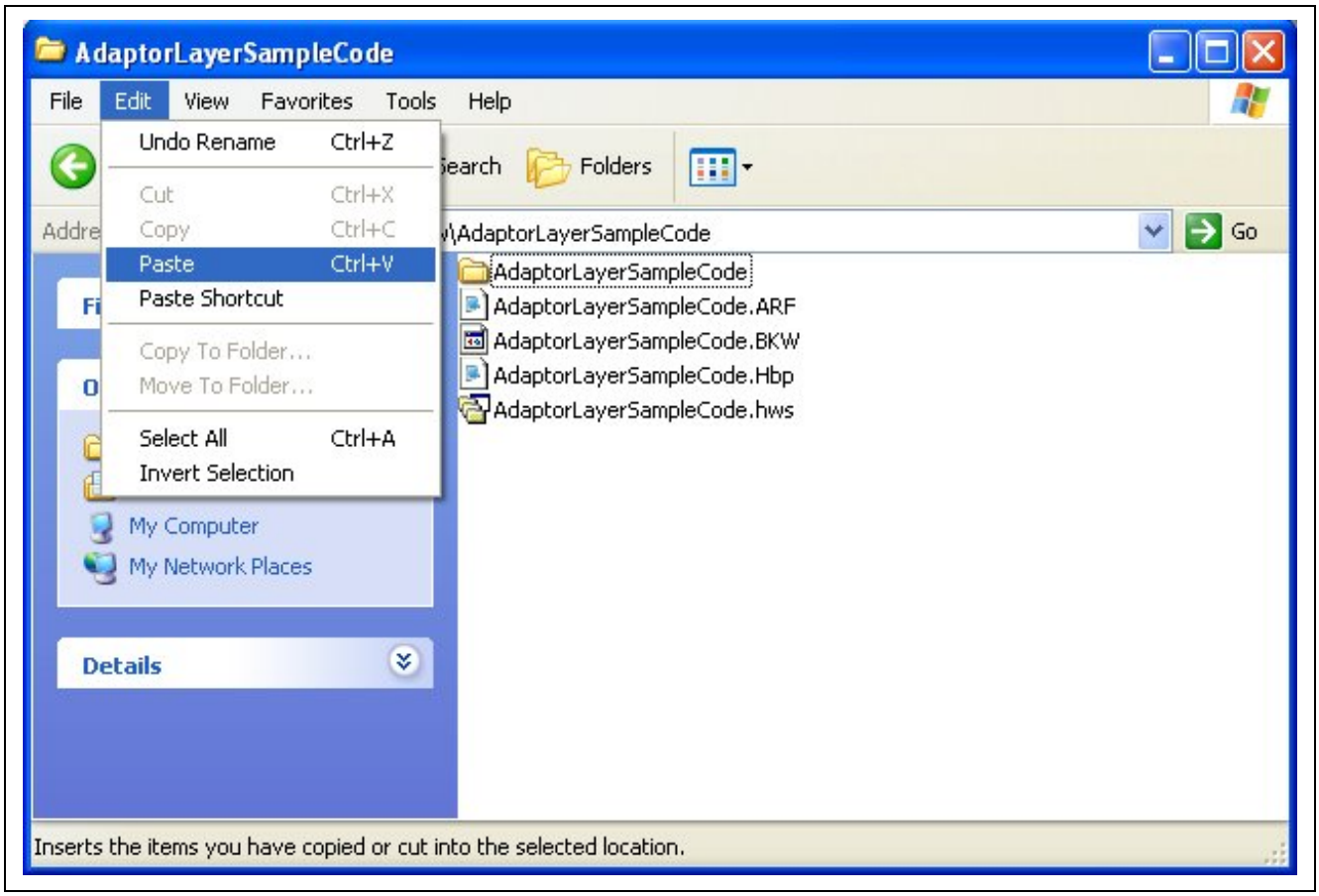
To develop a dual RTOS application, it is best to combine both the workspaces to have files that share the same code base. This can be visualized as follows:



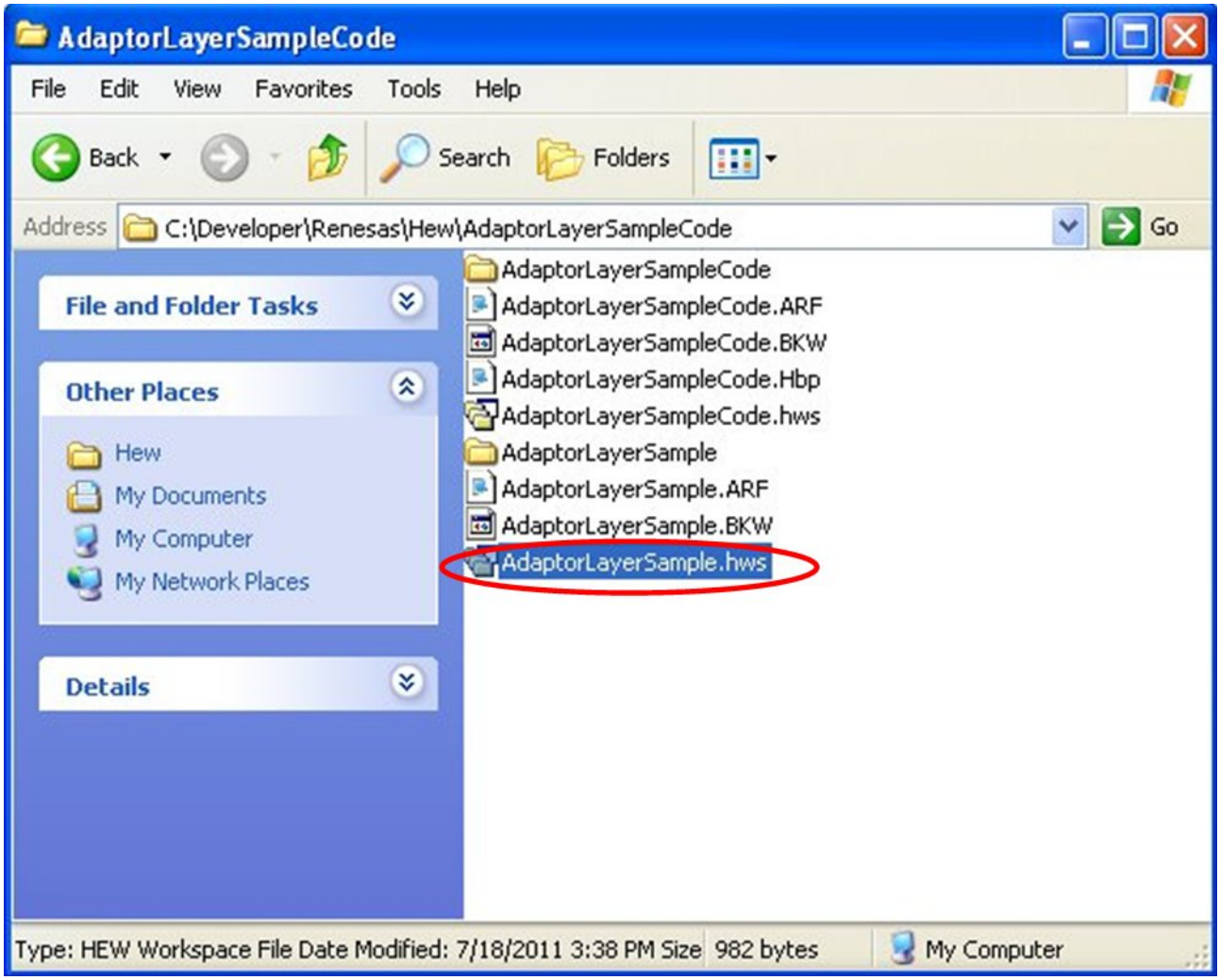
Copy the workspace files from \AdaptorLayerSample to \AdaptorLayerSampleCode:



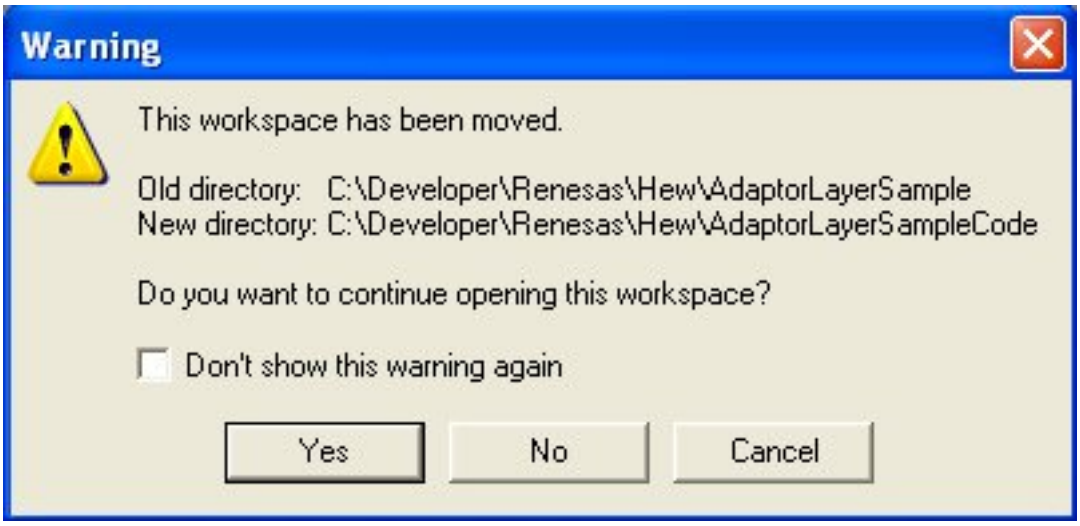
Past in the following directory as shown:



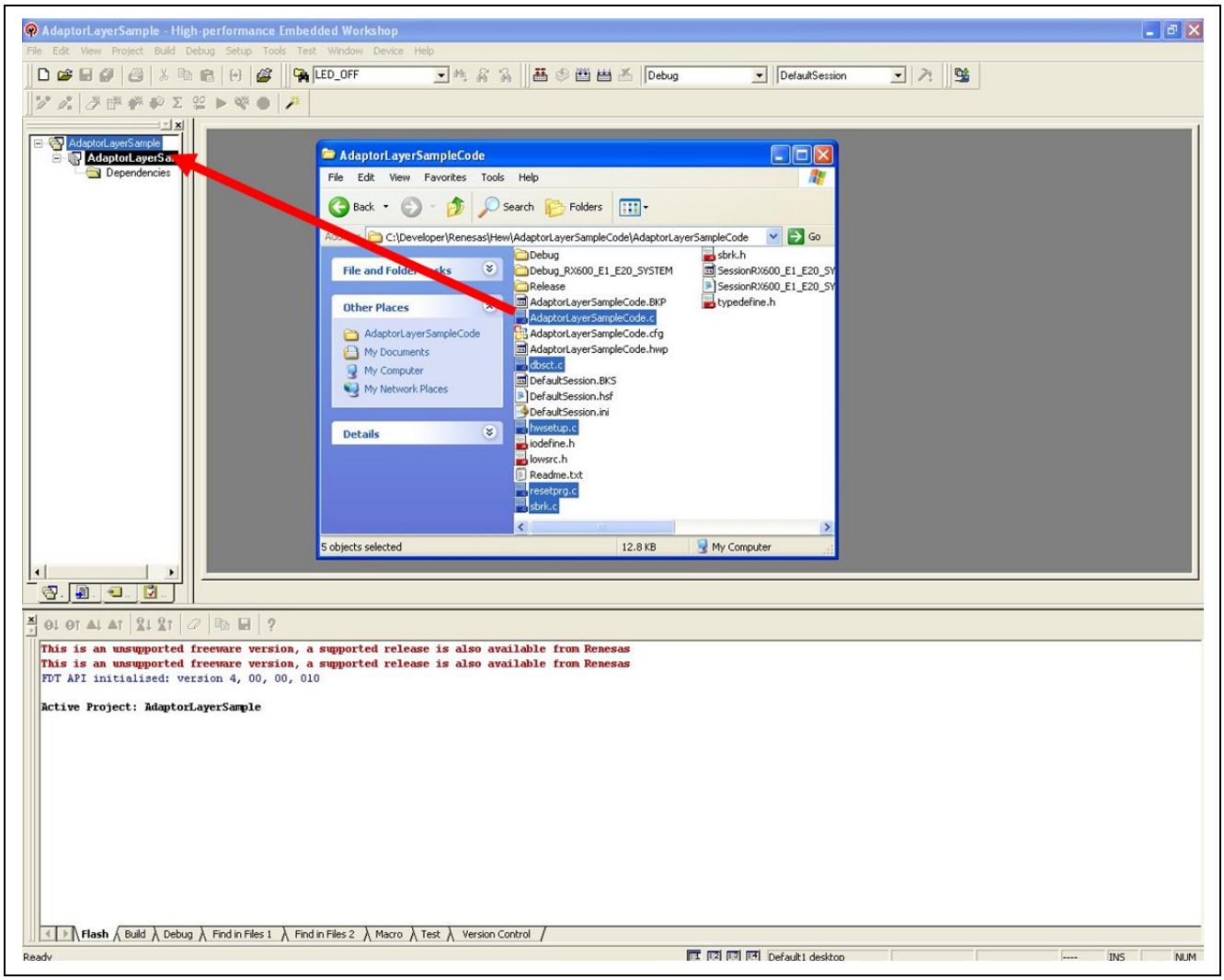
Reopen the new Adaptor Layer Workspace as shown:



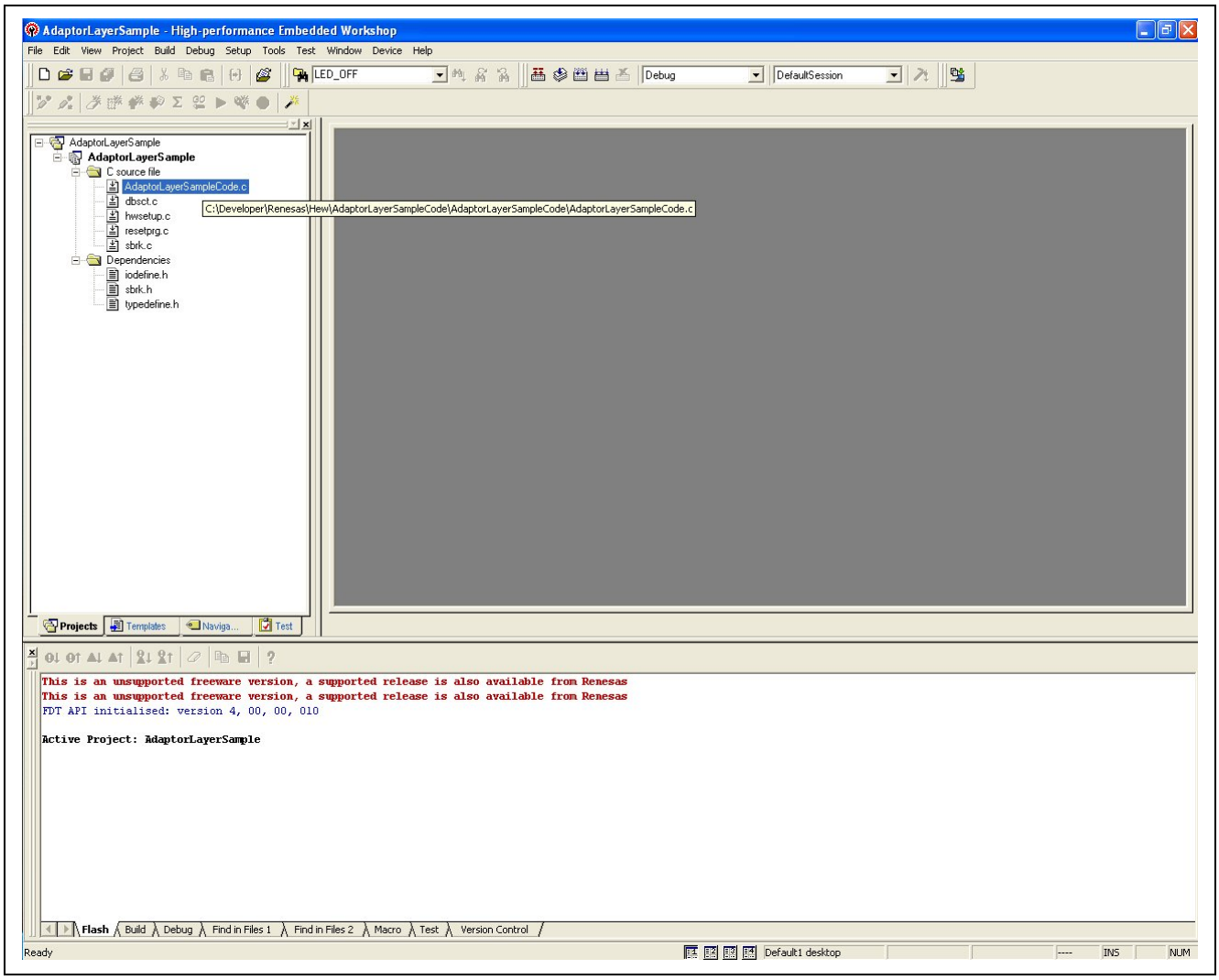
Accept the changes to the workspace by clicking ‘Yes’:



Open the folder \AdaptorLayerSampleCode\AdaptorLayerSampleCode and Drag-and-Drop the files as shown:



Open AdaptorLayerSampleCode.c:



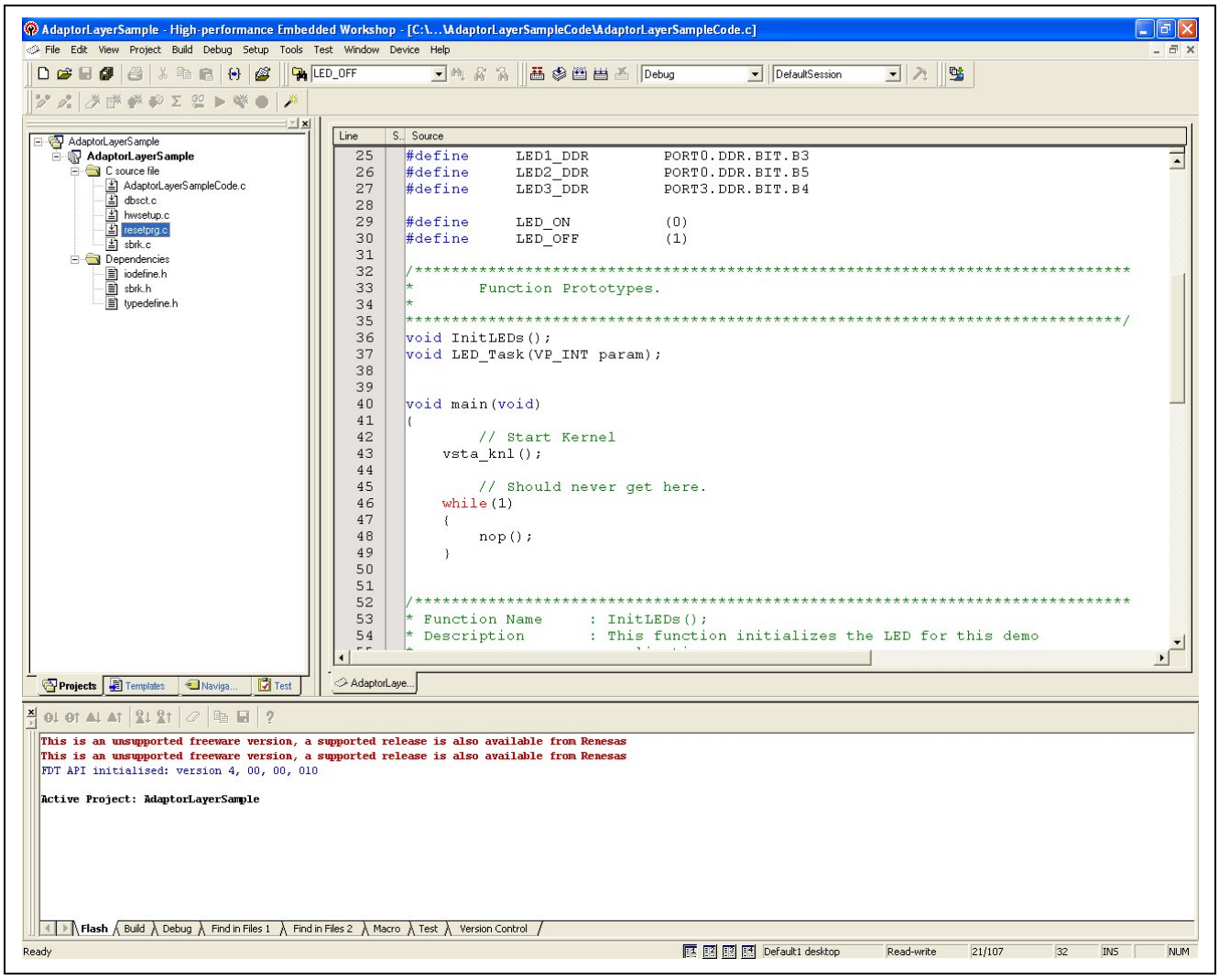
Replace the file's content with the following code:

```
#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);
  }
}
```

Notice that the only difference is the `void main(void)` function that is added.

Open resetprg.c:



Replace the entire code of resetprg.c with the following:

```
#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
```

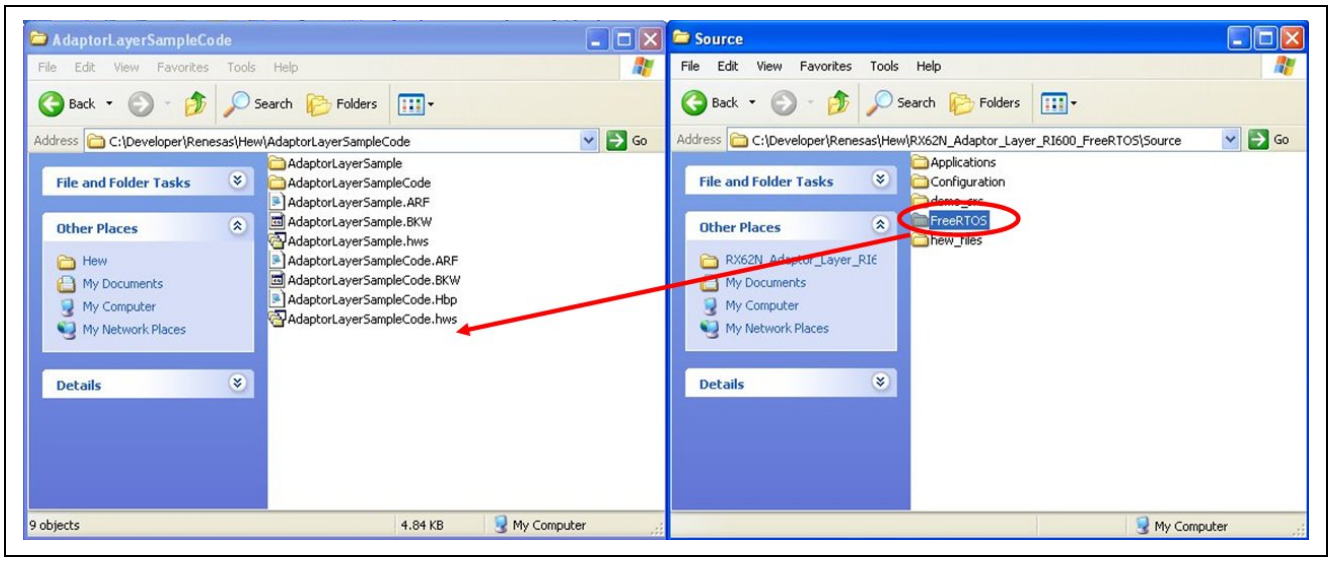
```
//#ifdef __cplusplus       // Remove the comment when you use global class
object
//extern "C" {             // Sections C$INIT and C$END will be generated
//#endif
//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//#endif

#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
```

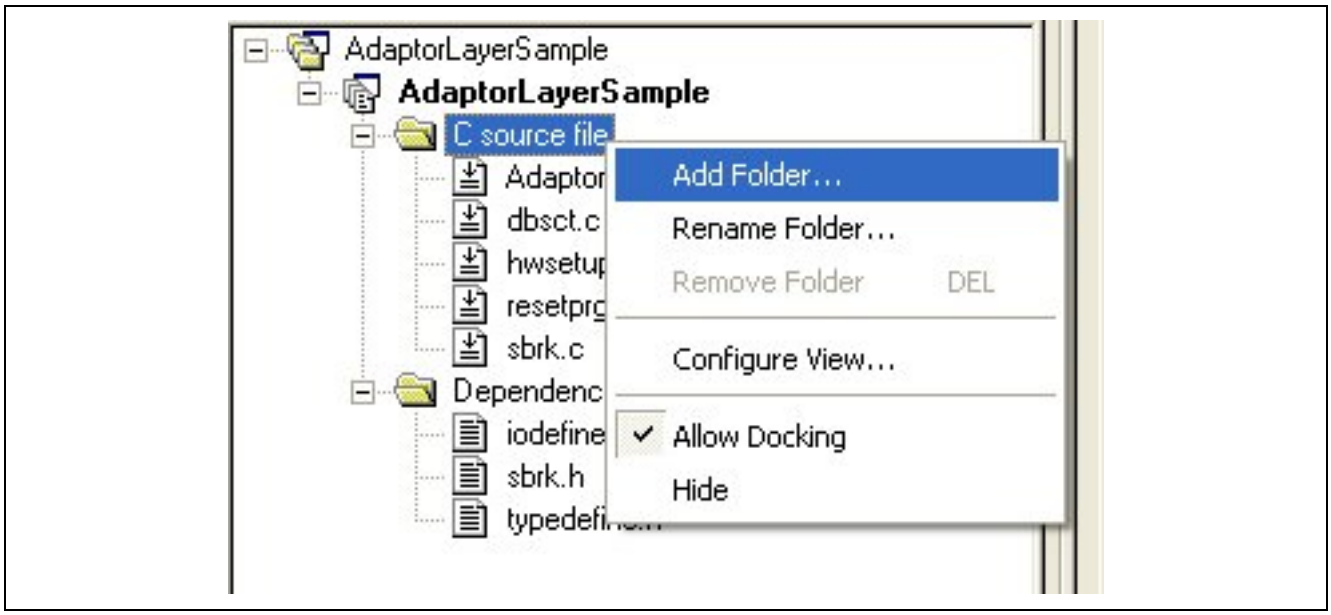
Notice that the parts that were generated by the RI-600 Workspace Generator have been masked out by the conditional compile check of __ADAPTOR_RI_600__ and __ADAPTOR_FREE_RTOS__ .

2.9 Importing FreeRTOS™ Files

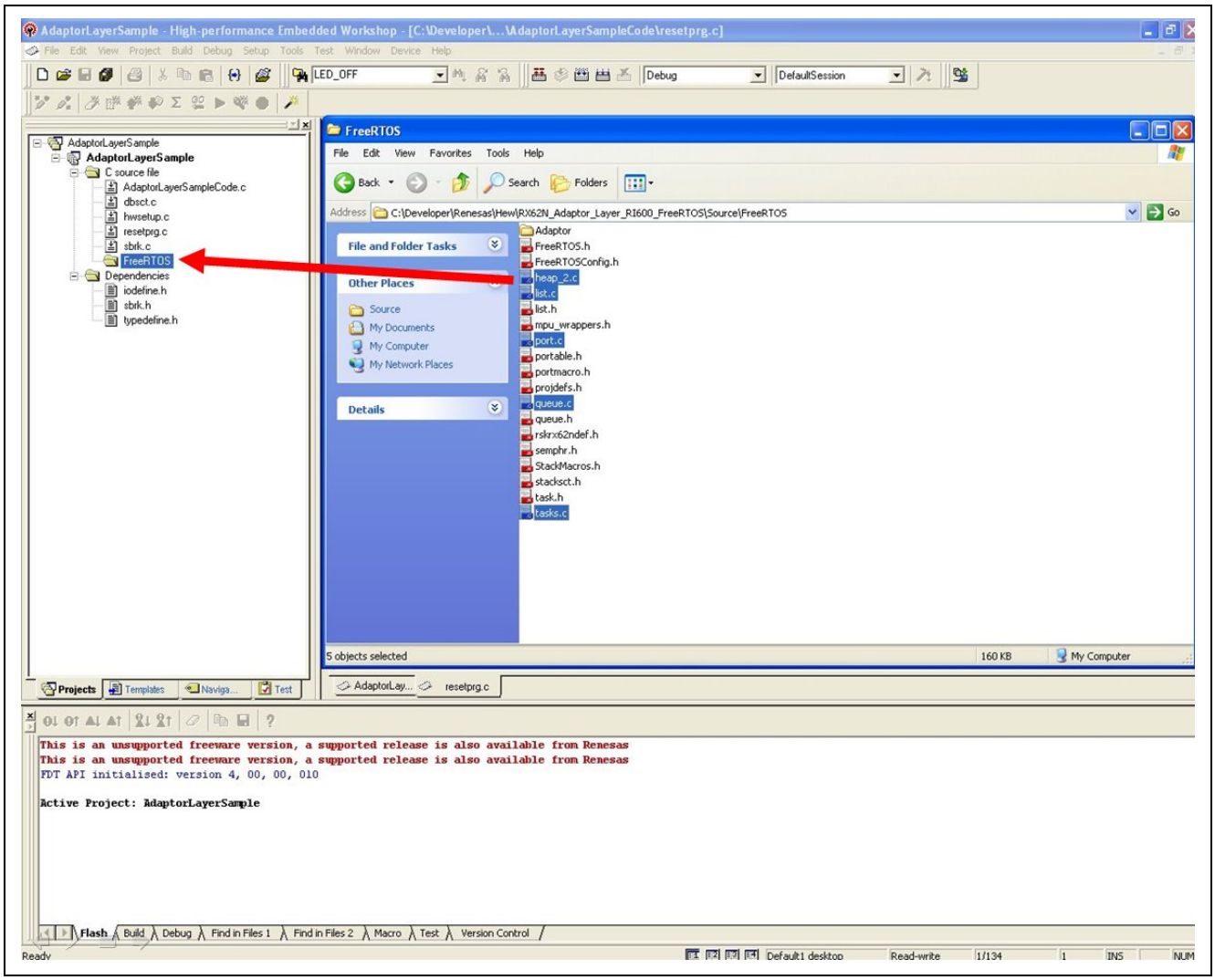
Copy the entire FreeRTOS™ Source Code Folder that came with this documentation into the workspace folder as shown:



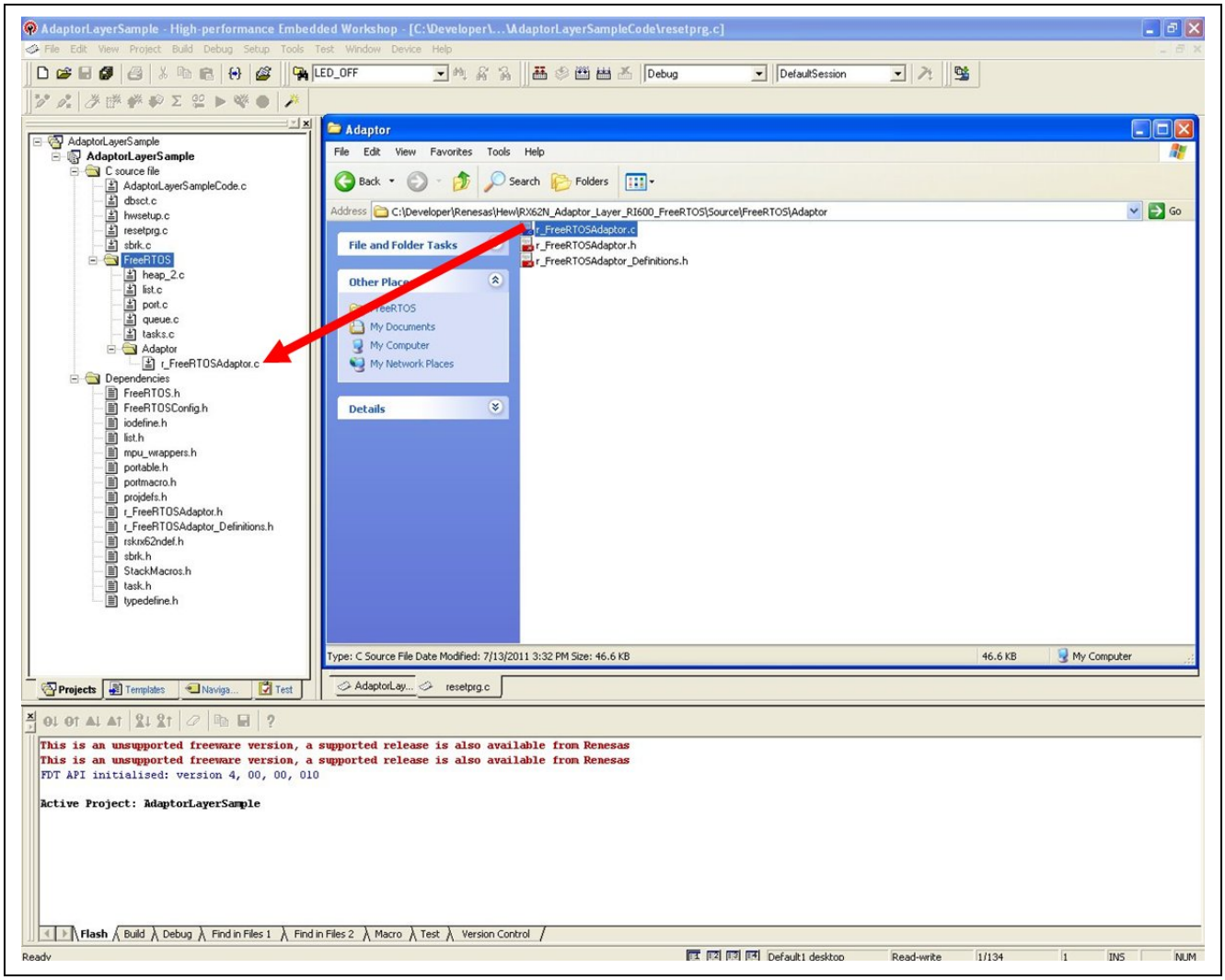
Create a directory in the Workspace for FreeRTOS™:



Drag and drop the FreeRTOS™ files as follows:

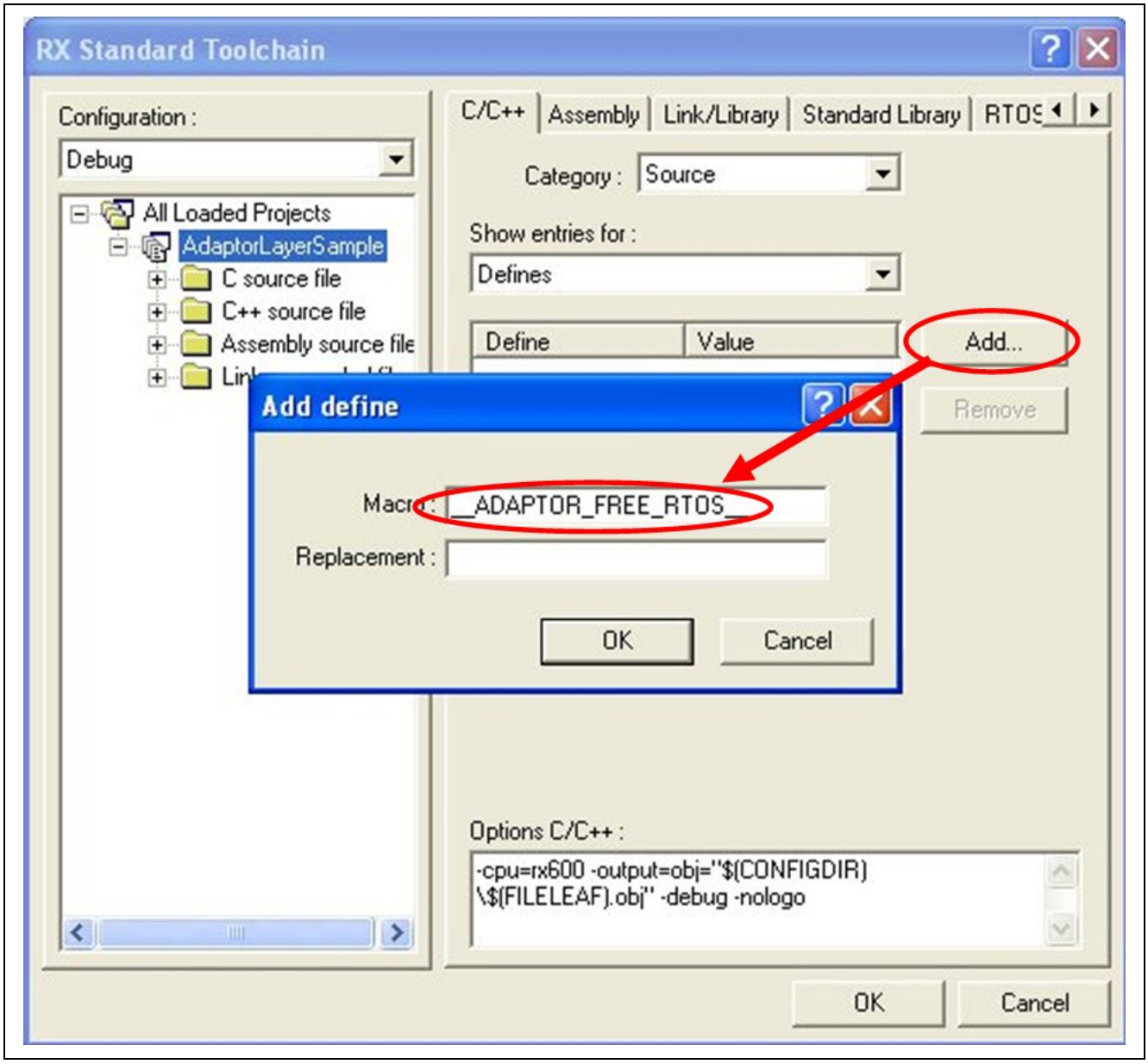


Create a folder called Adaptor under the FreeRTOS™ folder in the Workspace. Drag and drop the required Adaptor C Code file as follows:



2.10 Defining use of Adaptor Layer

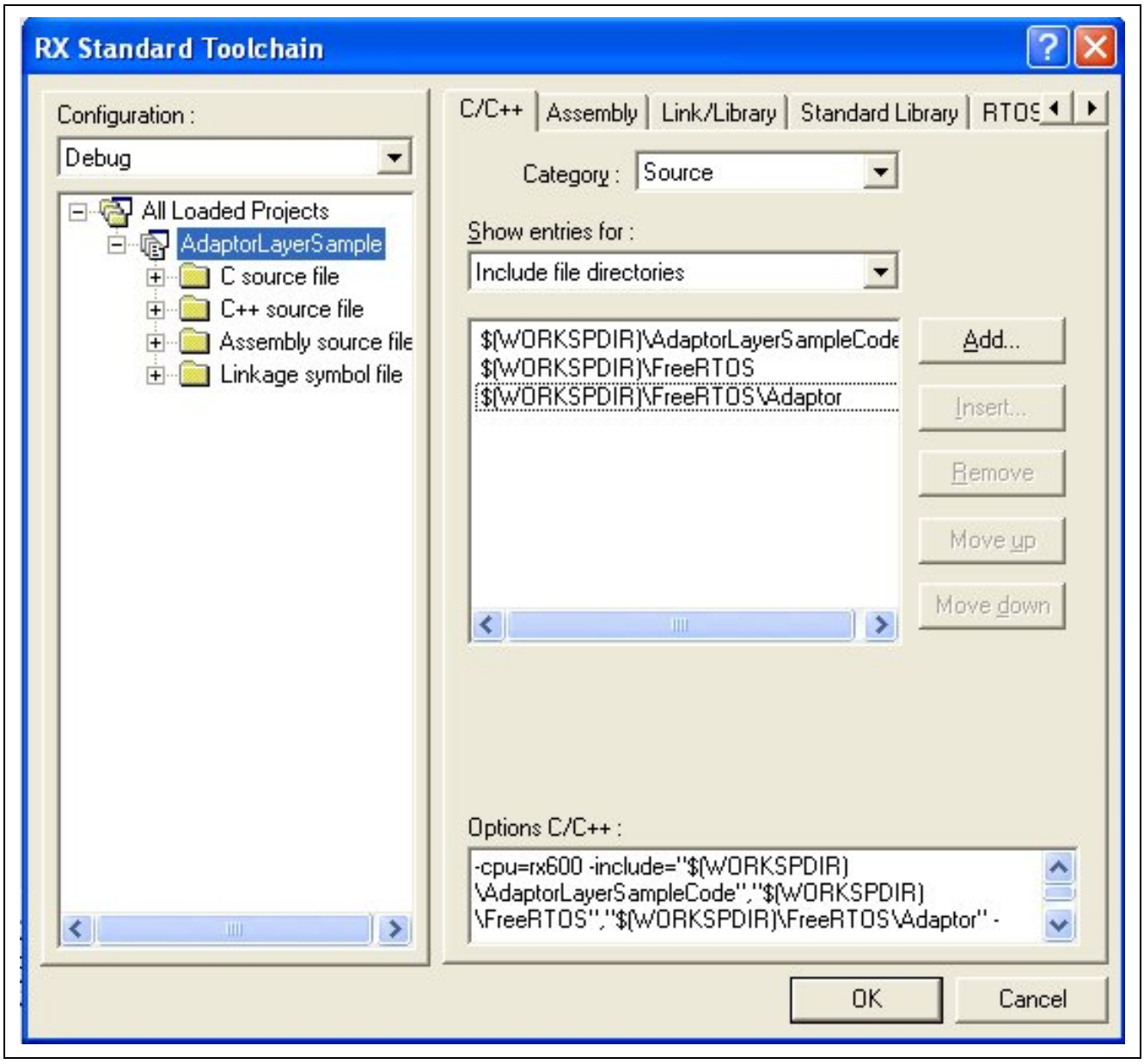
Goto Build→RX Standard Toolchain, under the C/C++ Tab, select: Show entries for: Defines. Add __ADAPTOR_FREE_RTOS__ as shown:



2.11 Including Header Files

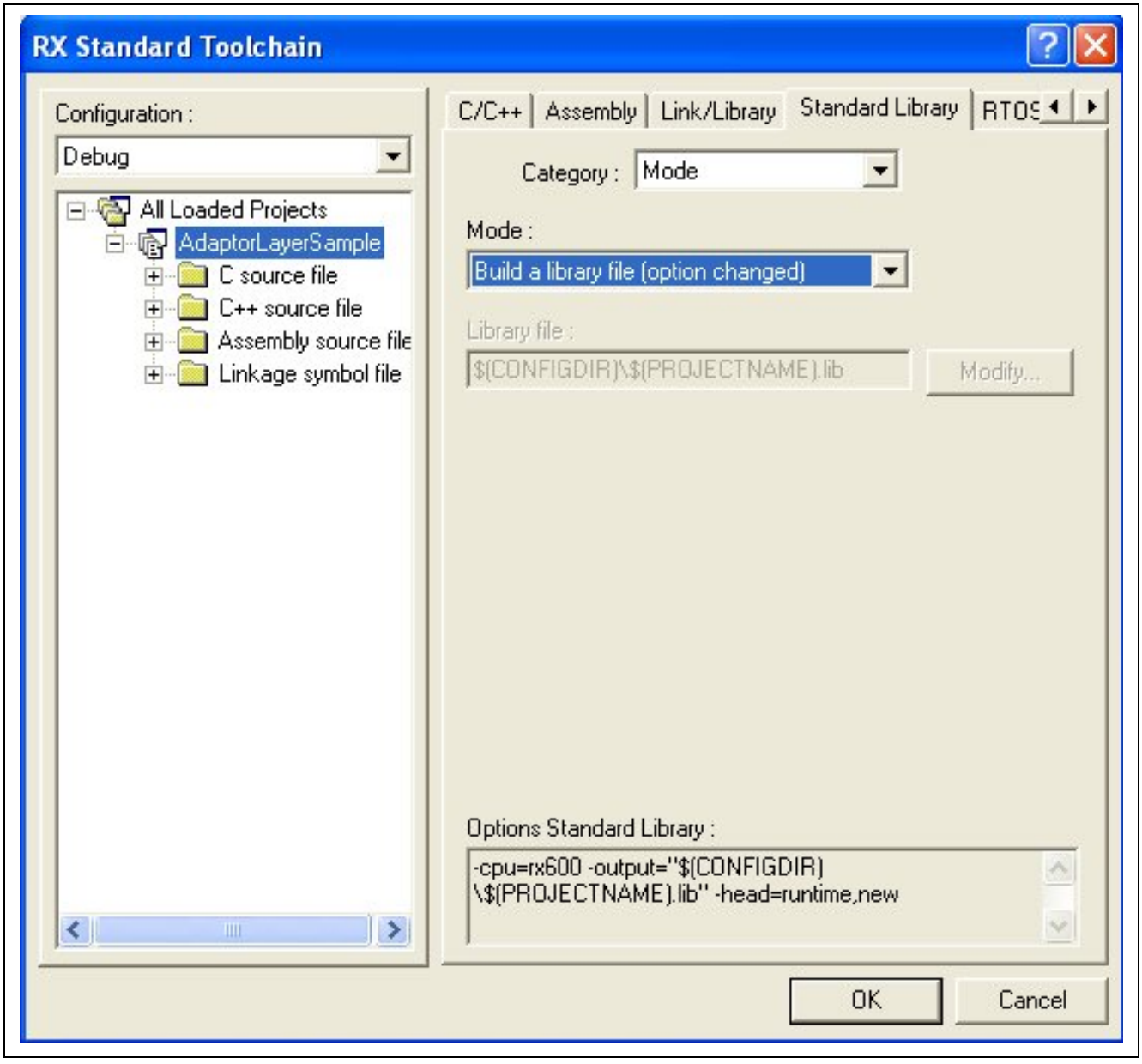
Goto Build→RX Standard Toolchain, under the C/C++ Tab, select: Show entries for: Include file directories, add the following directories as shown:

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



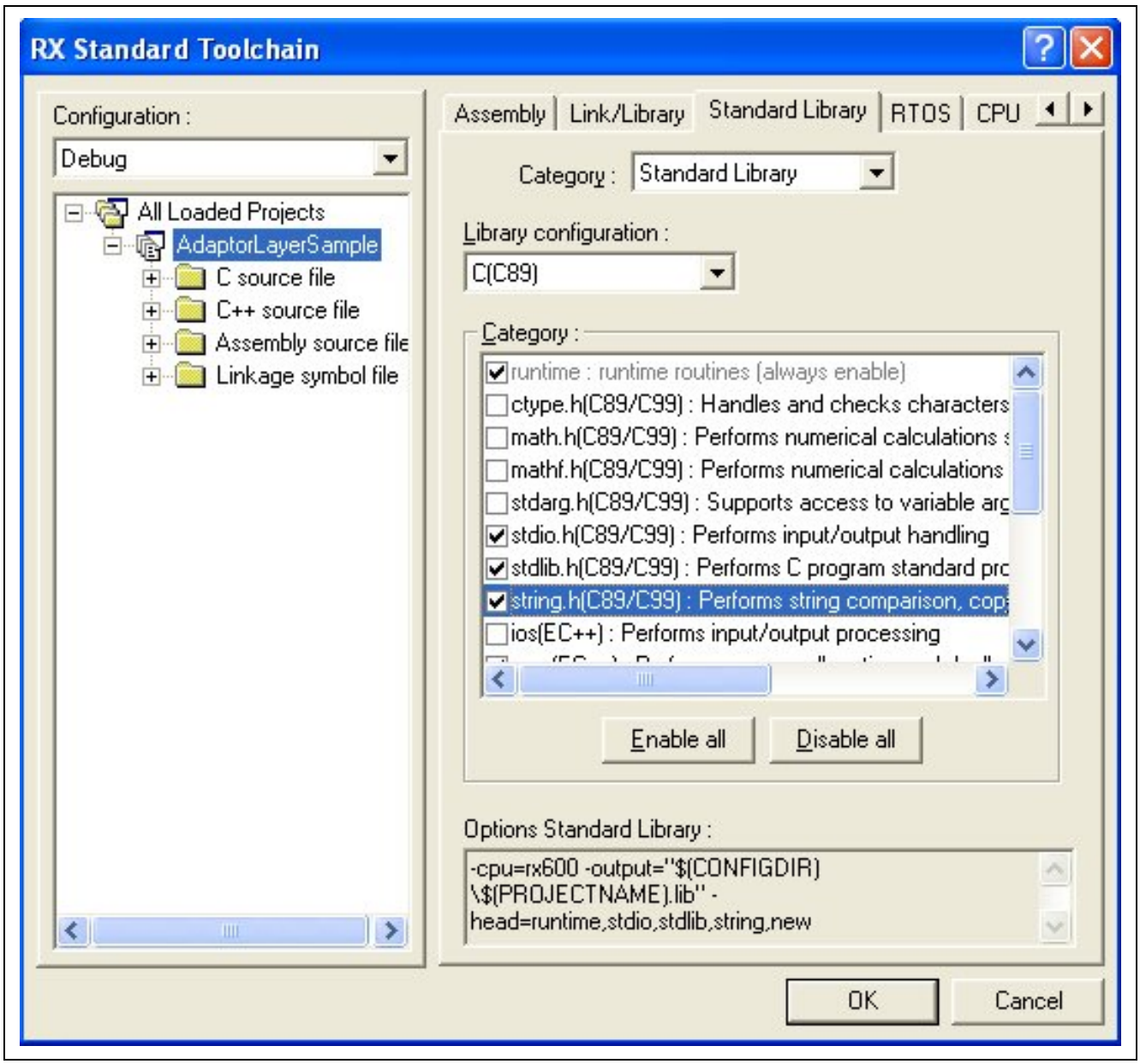
2.12 Setting Library Files

Goto Build→RX Standard Toolchain, under the Standard Library Tab, select Category: Mode and set Mode to have the following value from the drop down menu:



Goto Build→RX Standard Toolchain, under the Standard Library Tab, select Category: Standard Library and select the following header files to be built:

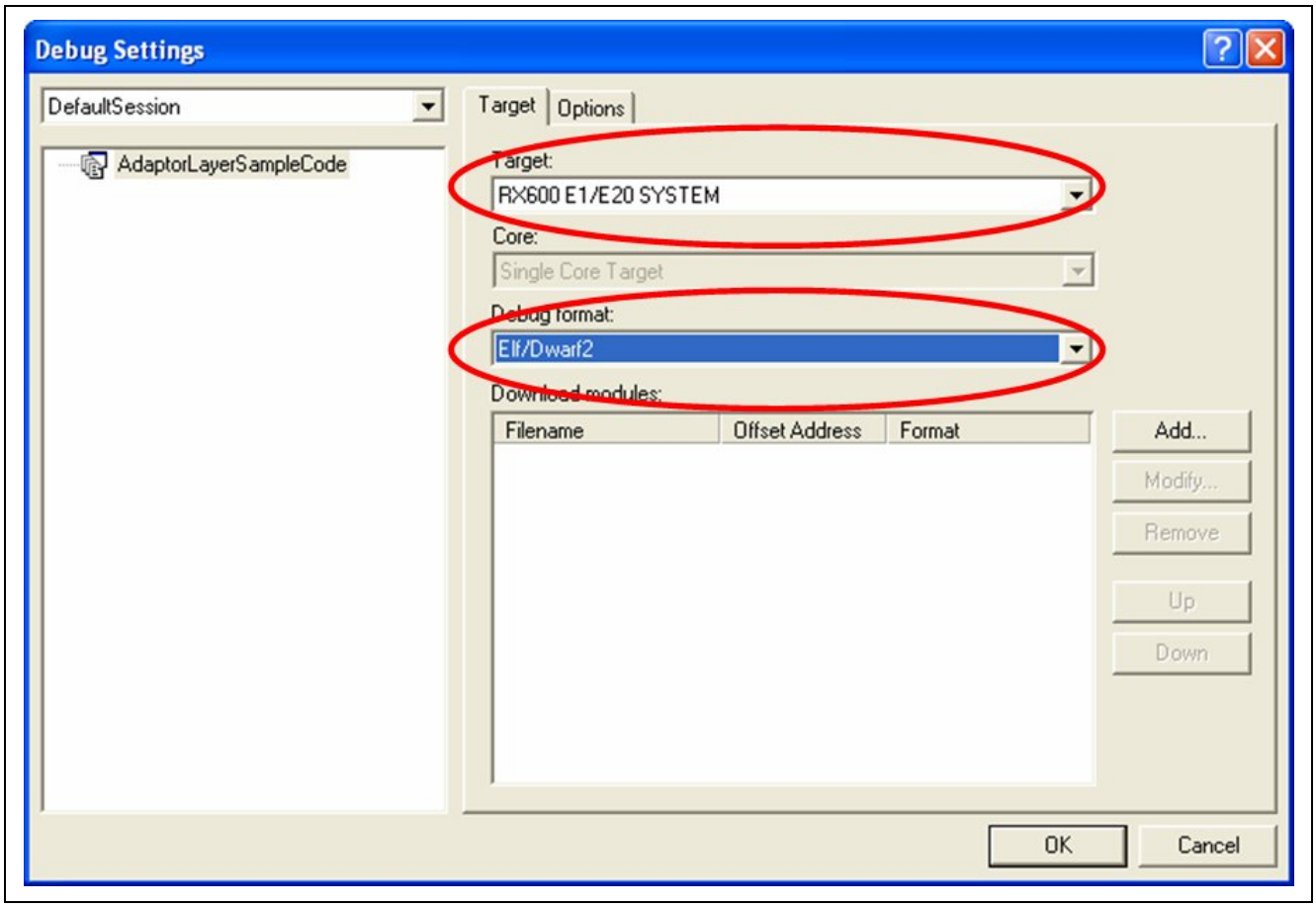
- stdio.h
- stdlib.h
- string.h



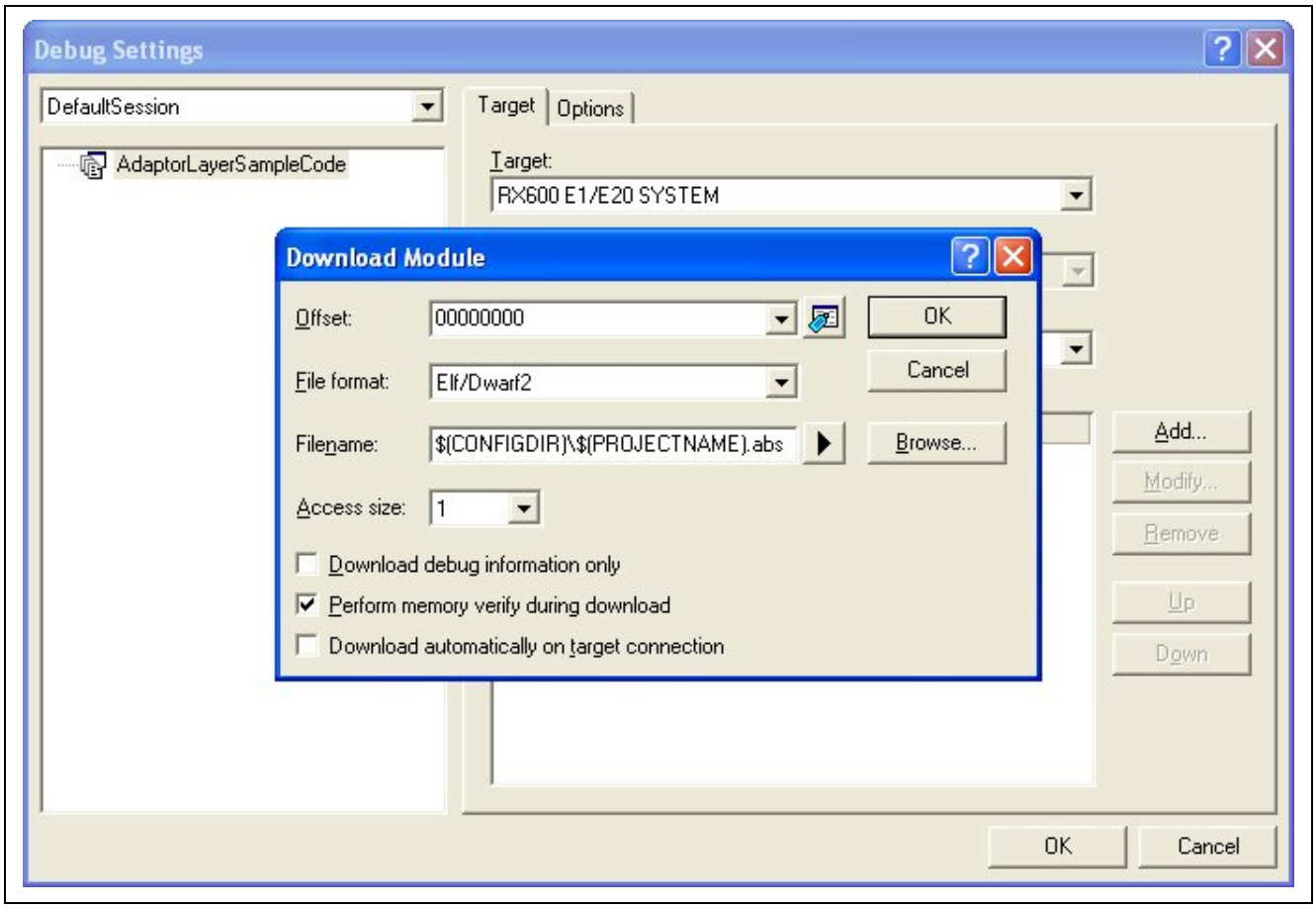
2.13 Connecting to the RX62N Board

Add the emulator to connect to the RX62N RSK Board as shown:

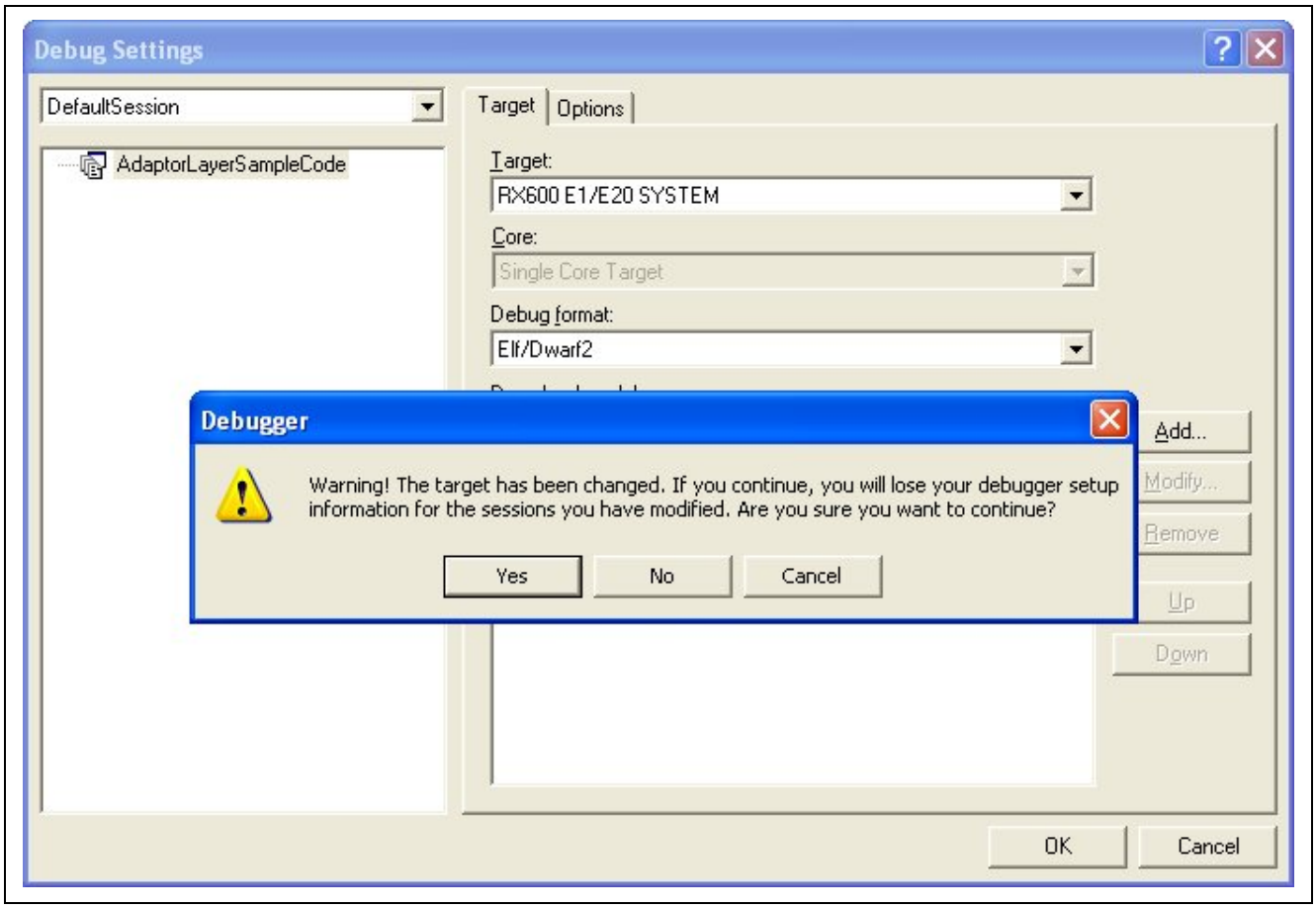
Add the following debug settings as shown:



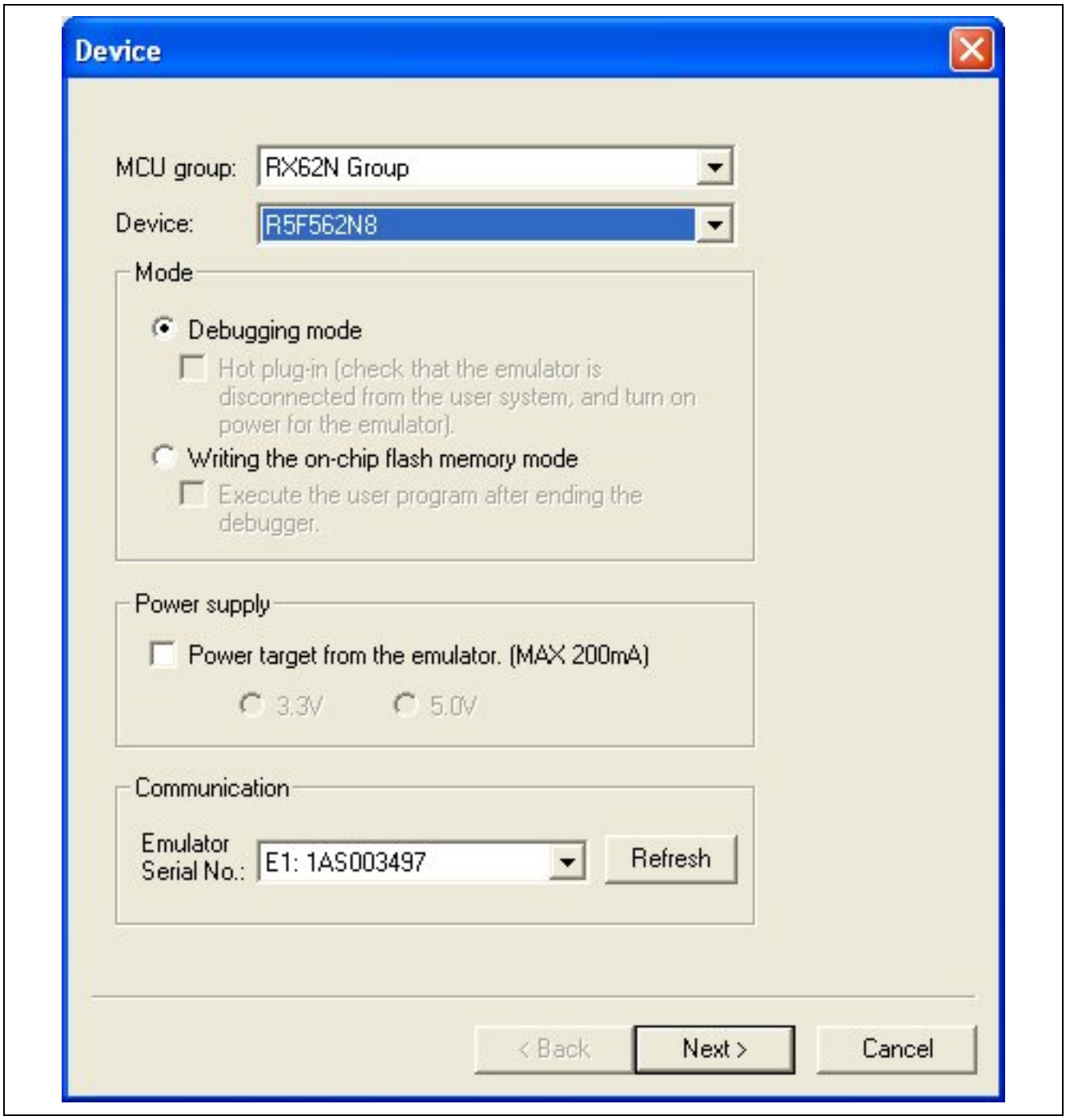
Click ‘Add’ to add the following configuration:



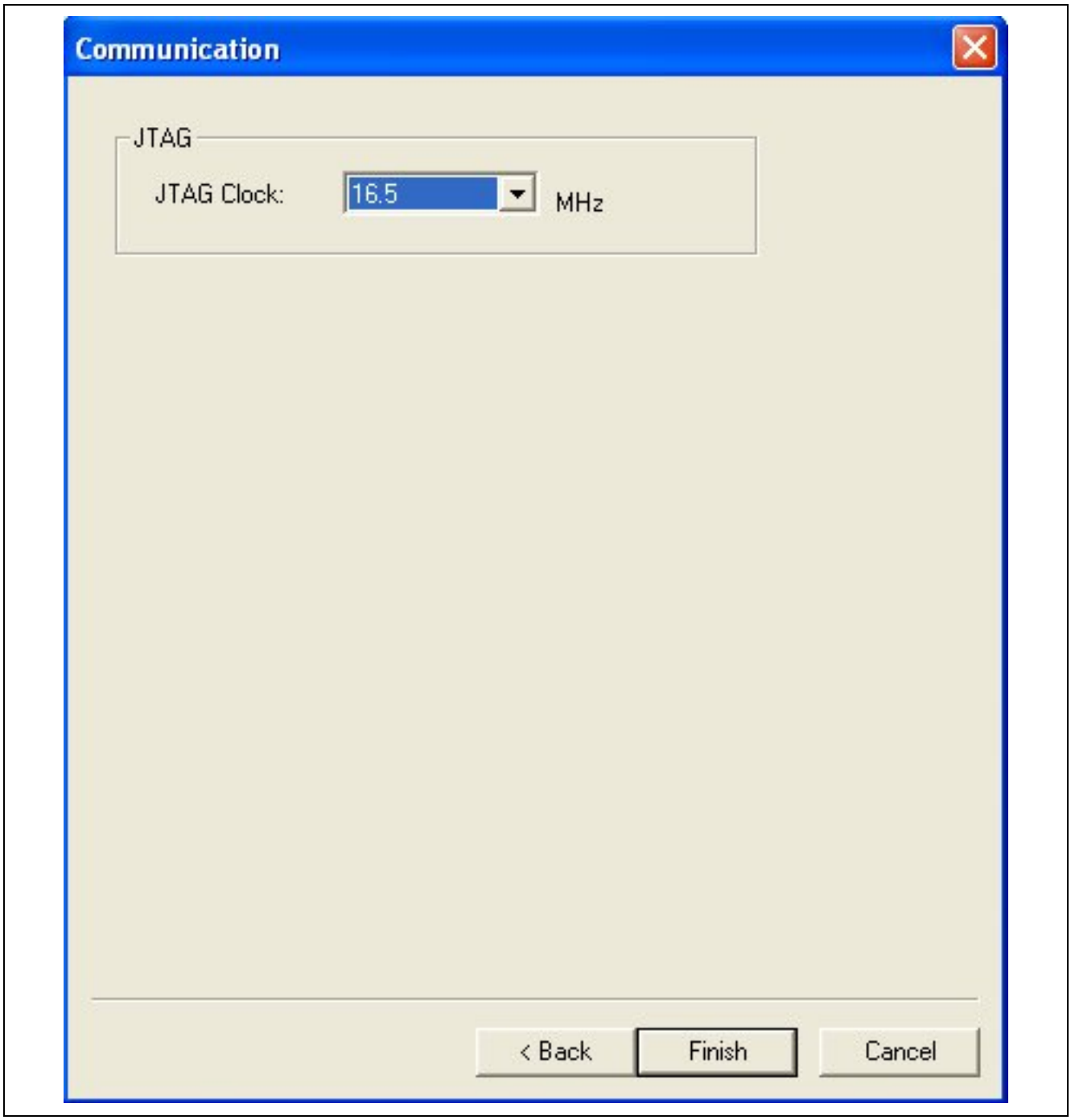
Click ‘OK’ to accept the default settings and ‘Yes’ when this Debugger warning panel is being displayed:



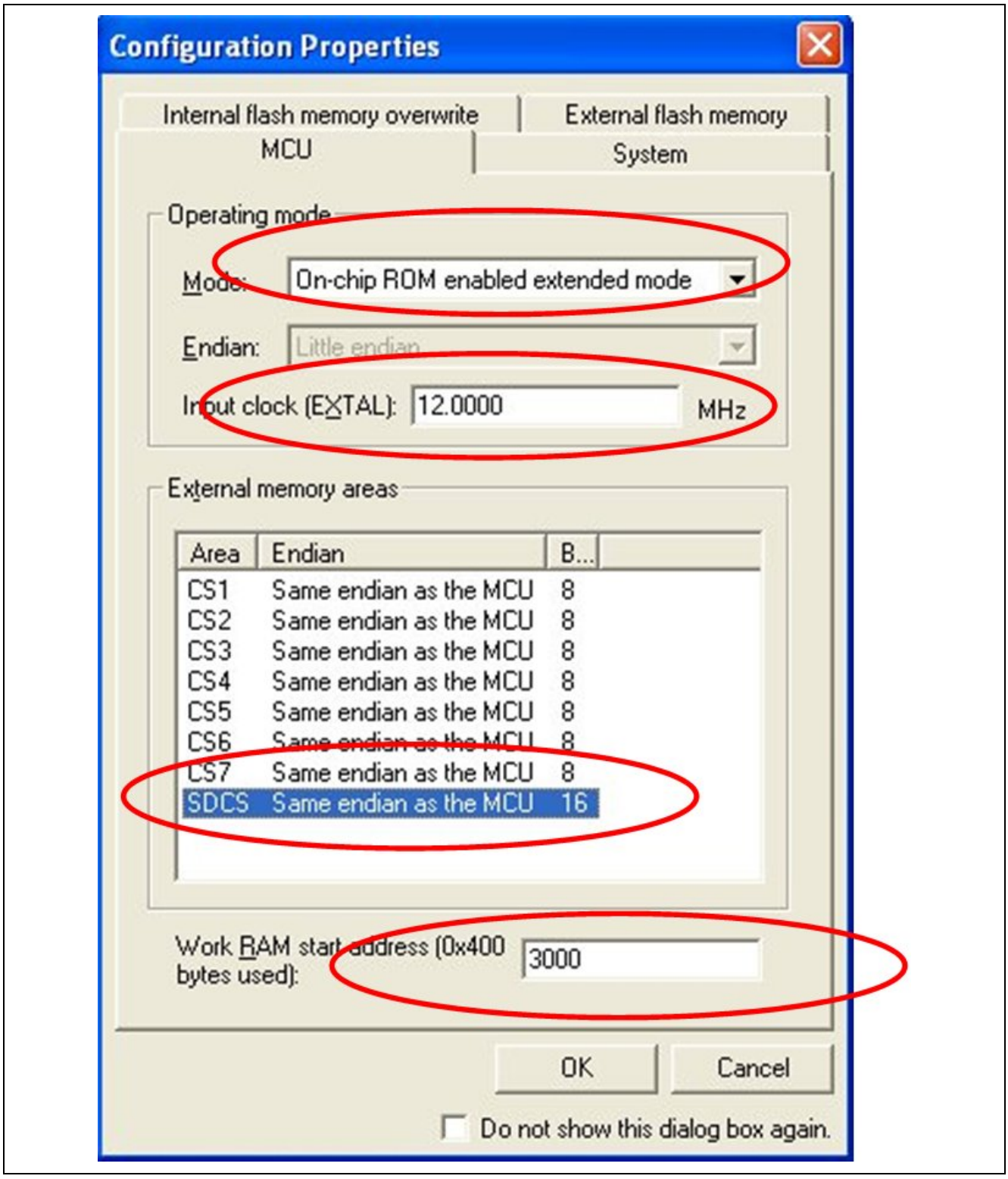
Select the following in the Device window:



Ensure that the board is powered before clicking ‘Next >’:



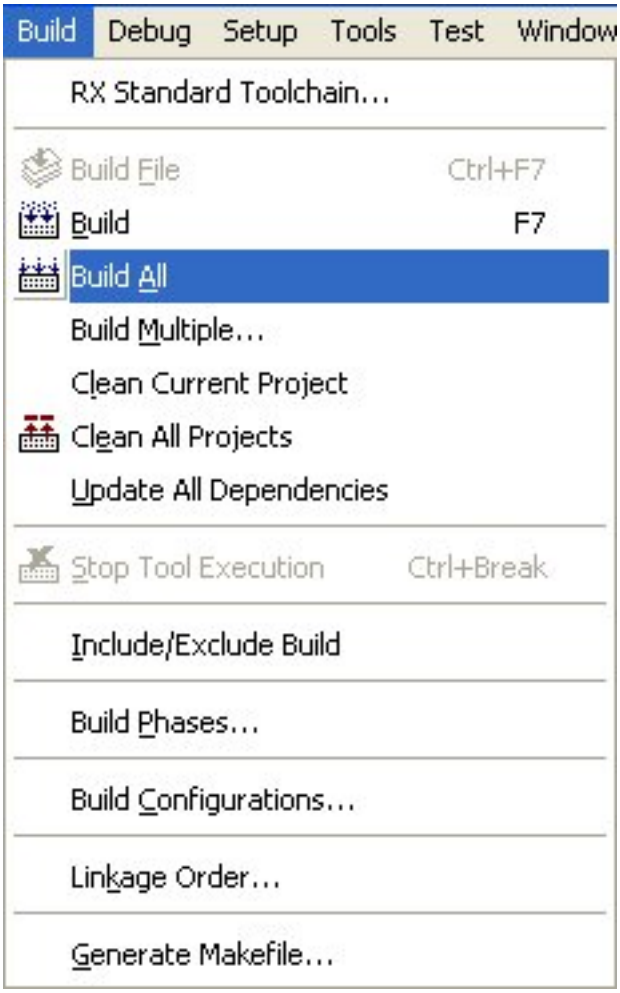
Click ‘Finish’ to begin connection to the device. Select the following settings in the next window:



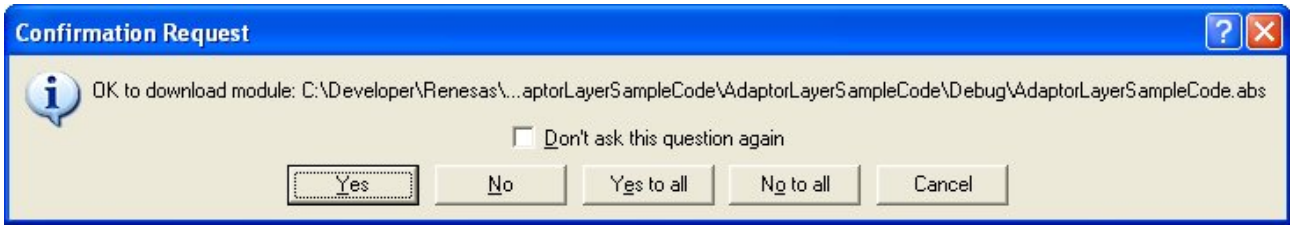
Click ‘OK’ to proceed connecting to the device.

2.14 Compile and Build LED Blinky Example

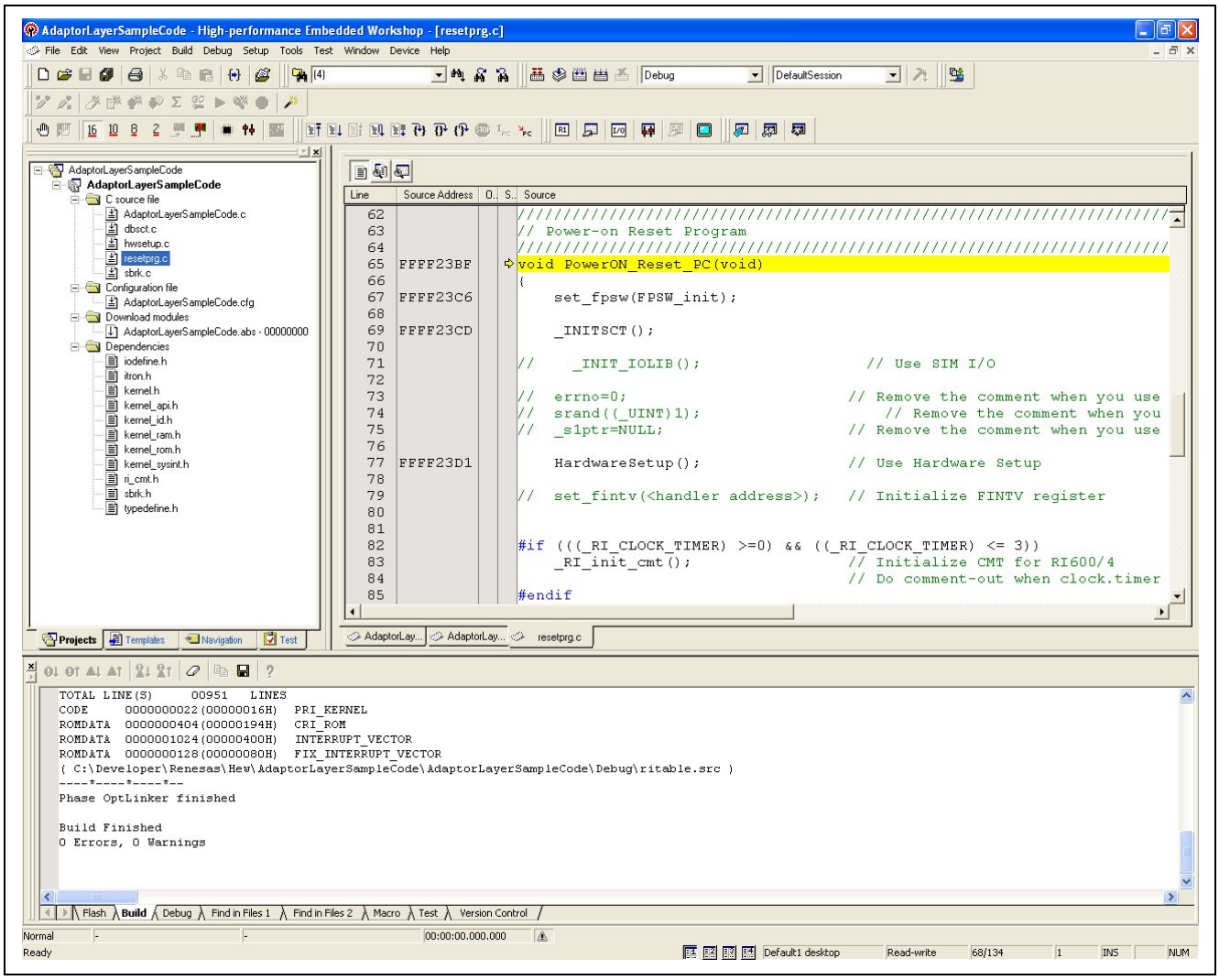
Select Build→Build All to being compilation:



When the compilation is done, select 'Yes to all' to download the built module:

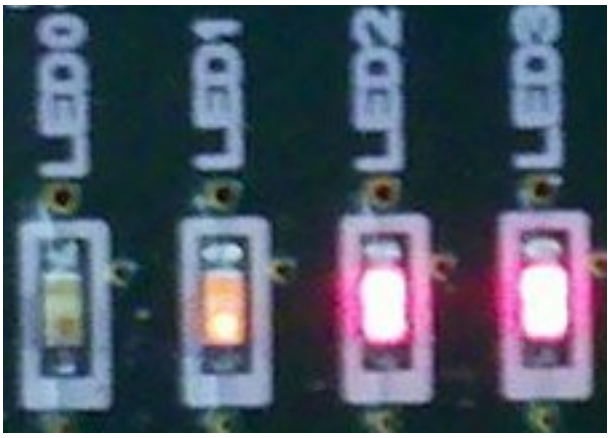


When the module has been downloaded, program is ready for execution as shown:



In order to run the program, press F5 or select Debug→Go as shown:

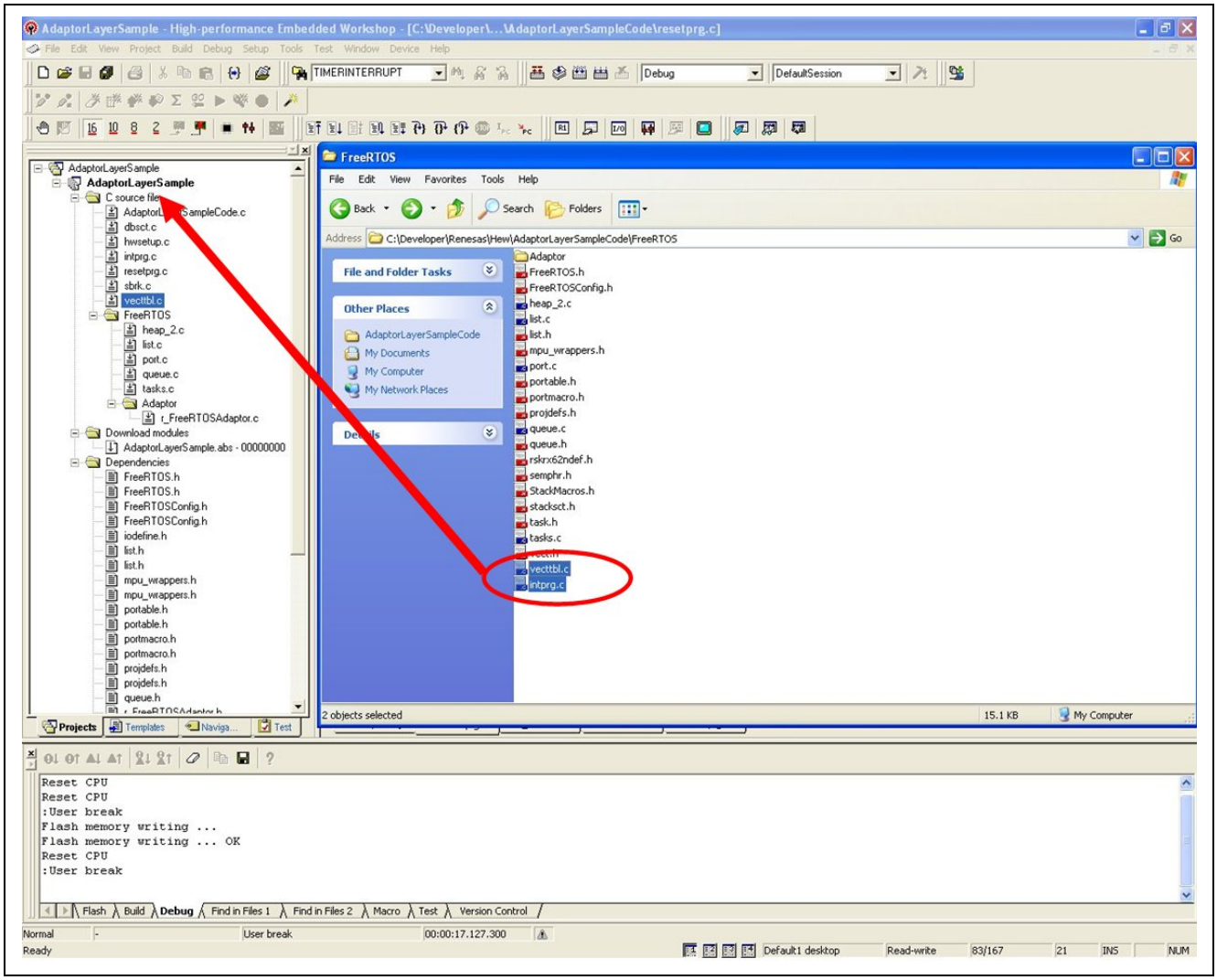
Finally, the LEDs on the RSK should be blinking as such:



2.15 Trouble-Shooting

The default vecttbl.c may not have been used when the project is built, drag and drop to add the following files as shown:

- vecttbl.c
- intprg.c



Compile and build, all should be working fine now.

3. Adaptor Layer Usage

3.1 Creating Tasks

Tasks are managed in the Adaptor Layer by the following C Structure:

```
/* 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;
};
```

This sample shows the creation of Tasks that are to be managed by the Adaptor Layer:

```
/* 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 Creating Mailboxes

Mailboxes are managed in the Adaptor Layer by the following C Structure:

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

This sample shows the creation of Mailboxes that are to be managed by the Adaptor Layer:

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

The `pvMailBox` variable will be initialized by the Adaptor Layer when `vsta_knl()` is invoked.

3.3 Creating Memory Pools

Task structures are managed in the Adaptor Layer by the following C Structure:

```
/* 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;
};
```

This sample shows the creation of Tasks that are to be managed by the Adaptor Layer:

```
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 Creating Cyclic Handlers

Task structures are managed in the Adaptor Layer by the following C Structure:

```
/* 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;
};
```

This sample shows the creation of Tasks that are to be managed by the Adaptor Layer:

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

This structure for Cyclic Handlers is used for recording and updating purposes. It should be noted that Cyclic Handlers are implemented as a high priority Task that will run right after the Gatekeeper Task. This is different from RI-600's implementation of Cyclic Handlers. Cyclic Handlers in RI-600 are considered non-Task state. Implementing Cyclic Handlers in the Adaptor Layer as Task States may cause some issues depending on the real-time necessity of such Cyclic Handlers that must be fired. In the alternative, it is advisable to use a hardware timer interrupt, such as MTU1 or CMT1 on the RX62N, to implement Cyclic Handlers.

In order to implement the Cyclic Handler as a Task, section (4) of the function `vsta_knl()` in `r_FreeRTOSAdaptor.c` should be modified to accommodate the creation of the Cyclic Handler. It is advisable to code the construct in this manner:

```
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) Individual cyclic handler should have its own specific init function.

(2) Reason for individual cyclic handler init function is that stack size needs to be customized.

```
/*********************************************************************/
/* 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. Adaptor Layer Configurations

4.1 Configuration: r_FreeRTOSAdaptor.h

This section describes the available configurations that can be set for the Adaptor Layer in r_FreeRTOSAdaptor.h.

Defined Macro	Description
configADAPTOR_SUPPORT_E_ID	When this is set to 1, the return value of E_ID is supported within the context of RI-600 API Specifications
configADAPTOR_SUPPORT_E_CTX	When this is set to 1, the return value of E_CTX is supported within the context of RI-600 API Specifications.
GATEKEEPER_QUEUE_LEN	This is the Queue Length of the GateKeeper Queue. This value should be judiciously chosen such that the rate of event request arrival will not cause the queue to overrun.

4.2 Configuration: r_FreeRTOSAdaptor_Definitions.h

This section describes the configurations that should be made in r_FreeRTOSAdaptor_Definitions.h.

r_FreeRTOSAdaptor_Definitions.h comprises of definitions required by the use application when using RI-600. Namely, this file should contain definitions from:

- itron.h

This file can be found in the install directory of RI-600 on the local system. This file contains all the definitions of the variable types that are available for use in RI-600. In order for the application to preserve these data types, this file should be updated with the latest RI-600 itron.h definitions used by the application.

- kernel.h

This file can be found in the install directory of RI-600 on the local system. This file contains the required definitions used by RI-600.

- kernel_api.h

This file can be found in the install directory of RI-600 on the local system. This file contains the function prototypes of the RI-600 API.

- kernel_id.h

This file is a generated output file from the RX Configurator. It can be found in the project workspace of the RI-600 configurator output folder.

This is a sample of how r_FreeRTOSAdaptor_Definitions.h should look like:

```
/*****************************************************************/
/* (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 Configuration: `r_FreeRTOSAdaptor.c`

This section describes the configurations that should be done in `r_FreeRTOSAdaptor.c`.

These configurations will be presented as follows:

Section Title

Section Label:

Description:

Directions:

Sample Configuration:

There are 2 primary categories to configure the Adaptor Layer for full integration with the RI-600 conforming application. They are:

- Section (A) through (B)

These sections contain pre-configurations that are required. These features are normally taken care of by the RI-600 Configurator. However, the Adaptor Layer requires such information of the application as well. This section has to be manually programmed in order to use the Adaptor Layer effectively.

- Section (1) through (4)

These are the necessary configurations for the Adaptor Layer to managed: Tasks, Mailboxes, Memory Pools as well as Cyclic Handlers.

4.3.1 Priority Map

Section Label

(A) Priority Map Size and Priority Map

Description

The Adaptor Layer requires a priority look up table for the translation mapping of RI-600 priorities to FreeRTOS™ equivalent priorities. This is where this priority map comes into play. This priority map will be looked up whenever the RI-600 `chg_pri()` API is invoked. If a FreeRTOS™ equivalent priority cannot be found in this table, the Task that invoked the API will be trapped in an endless for-loop.

Directions

The RI-600 target application will have static priorities assigned to the tasks that are present. These priorities can be found in the RI-600 Configuration file. It should be noted that the smaller the numerical value of priority that is assigned to the task in the RI-600 Configuration file, the higher it is the task priority. However, in FreeRTOS™, the larger the numerical value of priority that is assigned to the task, the higher the task priority.

A systematic approach has been developed to convert the RI-600 priorities to equivalent FreeRTOS™ task priorities.

Take for example, an RI-600 Configuration of Tasks as such:

```
.
.
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;
};
.
.
.
```

The 2 step sequential process to convert RI-600 Priorities to FreeRTOS™ priorities is outlined as follows:

- Arrange RI-600 Priorities in ascending order in a Table from top to bottom
- Assign FreeRTOS™ Priority numbers, beginning with 2, in an increasing manner from bottom to top of the Table. The IDLE Task in FreeRTOS™ has a priority of 1. In order not to share CPU time with the IDLE Task, Task priority assignment in FreeRTOS™ should being at 2.

Task Name	RI-600 Task Priority	FreeRTOS™ Priority
TASK_1	3	4
TASK_2	4	3
TASK_3	20	2

Sample Configuration

The above translates to the follow C Code to be added in the Adaptor Layer:

```
/*********************************************************************
*           *** 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 Global Variables and Task Functions

Section Label

(B) Import of global variables and functions

Description

This section imports all the required Task application function which the Adaptor Layer will use to create Tasks.

Directions

Consolidate all Task function prototype that are present in the application layer and import them into the Adaptor Layer with the `extern` keyword modifier.

Sample Configuration

A sample C Code to be added in the Adaptor Layer under section (B) is as follows:

```
/************************************************************
*           *** 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 Adaptor Layer: Task Definitions

Section Label

(1) Task Definitions

Description

This section registers all the application Tasks for the Adaptor Layer to create and manage.

Directions

This section comprises of two steps. Firstly, define the total number of Tasks in the application layer. Secondly, initialize the `fr_TASKS[]` array of Structures.

Sample Configuration

A sample C Code to be added in the Adaptor Layer under section (1) is as follows:

```
/*********************************************************
*           *** 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 Adaptor Layer: Mailbox Definitions

Section Label

(2) Mailbox Definitions

Description

This section defines all the required mailboxes for use by the application Tasks.

Directions

Consolidate all mailbox definitions from the application configurator file and set them accordingly in the Adaptor Layer.

It should be noted that RI-600 supports mailboxes of unlimited length. However, FreeRTOS™ does not have such a feature. As such, the maximum length of the mailboxes should be judiciously chosen.

Sample Configuration

A sample from the RI-600 Configurator file may look as follows:

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

A sample C Code to be added in the Adaptor Layer under section (2) is as follows:

```
/*************************************************************
*           *** 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 Adaptor Layer: Memory Pool Definitions

Section Label

(3) Memory Pool Definitions

Description

This section defines all Memory Pools that are managed by the Adaptor Layer.

Directions

Consolidate all Memory Pools that are used by the application Tasks.

Sample Configuration

A sample from the RI-600 Configurator file may look as follows:

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

A sample C Code to be added in the Adaptor Layer under section (3) is as follows:

```
/************************************************************
*           *** 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 Adaptor Layer: Cyclic Handler Definitions

Section Label

(4) Cyclic Handler Definitions

Description

This section defined all the Cyclic Handler for application use.

Directions

Cyclic Handlers in the Adaptor Layer can be implemented as Tasks or periodic interrupts form a hardware timer such as CMT1 or MTU1.

For harder Real-Time System requirements, it is strongly encouraged that these cyclic handlers be implemented via means of hardware. However, the option is open for the developer to make use of a Task level Cyclic Handler as implemented in the Adaptor Layer.

Sample Configuration

A sample from the RI-600 Configurator file may look as follows:

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

A sample C Code to be added in the Adaptor Layer under section (4) is as follows:

```
/************************************************************
*           (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
};
```

However, it should be noted that an additional 2 functions are required to start and initialize the Cyclic Handlers as Tasks in the Adaptor Layer. These 2 sample functions are provided:

```
/************************************************************
* 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 );
      }
}
```

Subsequently, the `fr_CycHdl_T4_CYC_init()` needs to be placed in the `vsta_knl()` function to create the Cyclic Handler.

A sample C Code in vsta_knl() can be seen as follows:

```
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. Adaptor Layer conformance with RI-600 Specifications

5.1 Document Format

This section will have the following document format:

[Brief Task Descrption] : [API Function]

C Language API

[API Function Usage]

Parameter

[API Parameters]

Return Value

[Common return value]

Error Code

Name	Description	Support
[Defined return code]	[Description]	Yes/No/Configurable

Note: Configurable support can be enabled in `r_FreeRTOSAdaptor.h` as shown:

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

5.2 Adaptor Layer: API Specification

5.2.1 Activates Task with Start Code: sta_tsk()

C Language API

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

Parameter

```
tskid       Task ID
stacd       Start code
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_OBJ	Object state error (task indicated by tskid is not in the DORMANT state)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.2 Terminate Current Task: ext_tsk()

C Language API

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

Parameter

None.

Return Value

None.

Error Code

None.

5.2.3 Terminates Other Task: ter_tsk()

C Language API

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

Parameter

```
tskid       Task ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_OBJ	Object state error (task indicated by tskid is not in the DORMANT state)	N
E_ILUSE	Illegal use of service call (task indicated by tskid is current task)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.4 Changes Task Priority: chg_pri()

C Language API

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

Parameter

tskid	Task ID
tskpri	New base priority of the task

Return Value

E_OK for normal completion or error code (below)

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_ID	Invalid ID number	C
E_ILUSE	Illegal use of service call (task indicated by tskid is current task)	N
E_OBJ	Object state error (task indicated by tskid is not in the DORMANT state)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.5 Refers to Task Status: ref_tst()

C Language API

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

Parameter

```
tskid       Task ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_CTX	Context error (invoked from an unallowed system state)	C

[1] Please refer to RI-600 User Manual for Data Structure

5.2.6 Puts Task to Sleep: slp_tsk()

C Language API

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

Parameter

None.

Return Value

E_OK for normal completion or error code (below)

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_RLWAI	Forced release from the WAITING state (accpt rel_wai while waiting)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.7 Wakeup Task: iwup_tsk()

C Language API

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

Parameter

```
tskid       Task ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_OBJ	Object state error	N
E_QOVR	Queuing overflow	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.8 Suspend Task: isus_tsk()

C Language API

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

Parameter

```
tskid       Task ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_OBJ	Object state error (task indicated by tskid is in the DORMANT state)	N
E_QOVR	Queuing overflow (overflow of suspension request nesting count)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.9 Resume Suspended Task: irsm_tsk()

C Language API

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

Parameter

```
tskid       Task ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_OBJ	Object state error (task indicated by tskid is not in the SUSPENDED state)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.10 Delay Task: dly_tsk()

C Language API

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

Parameter

```
dlytim            Delay time in (ms)
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_RLWAI	Forced release from the WAITING state (accept rel_wai while waiting)	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.11 Send to Mailbox: snd_mbx(), isnd_mbx()

C Language API

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

Parameter

```
mbxid       Mailbox ID
pk_msg      Start address of the message to be sent
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_ID	Invalid ID number	C
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.12 Receive from Mailbox: rcv_mbx(), prcv_mbx(), trcv_mbx()

C Language 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);
```

Parameter

mbxid	Mailbox ID
ppk_msg	Pointer to the storage to which the start address of the received message is retruned.
tmout	Timeout (milliseconds)

Return Value

E_OK for normal completion or error code (below)

Error Code

Name	Description	Support
E_PAR	Parameter Error	N
E_ID	Invalid ID number	C
E_RLWAI	Forced release from the WAITING state	Y
E_TMOUT	Polling failure or timeout	Y
E_CTX	Context error (invoked from an unallowed system state)	C

[2] Please refer to RI-600 User Manual for Data Structure

5.2.13 Acquire fixed-sized memory block: get_mpf(), pget_mpf()

C Language API

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

Parameter

```
mpfid       Fixed-sized memory pool ID
p_blk       Pointer to the storage to which the start address of
      the memory block is returned
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_ID	Invalid ID number	C
E_RLWAI	Forced release from the WAITING state (accept rel_wai while waiting)	Y
E_TMOUT	Polling failure or timeout	Y
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.14 Release fixed-sized Memory Pool: rel_mpf()

C Language API

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

Parameter

```
mpfid       Fixed-sized memory pool ID
blk         Start address of the memory block to be released
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_ID	Invalid ID number	C
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.15 Refer to System Time: get_tim()

C Language API

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

Parameter

```
P_systim    Pointer to the storage to which the system time is
      returned
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_CTX	Context error (invoked from an unallowed system state)	C

[3] Please refer to RI-600 User Manual for Data Structure

5.2.16 Starts Cyclic Handler Operation: sta_cyc()

C Language API

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

Parameter

```
cycid       Cyclic handler ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.17 Release fixed-sized Memory Pool: stp_cyc()

C Language API

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

Parameter

```
cycid       Cyclic handler ID
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_ID	Invalid ID number	C
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.18 Rotates Task Precedence: irot_rdq()

C Language API

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

Parameter

```
tskpri      Task priority
```

Return Value

```
E_OK for normal completion or error code (below)
```

Error Code

Name	Description	Support
E_PAR	Parameter error	N
E_CTX	Context error (invoked from an unallowed system state)	C

5.2.19 Start Kernel: vsta_knl()

C Language API

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

Parameter

None.

Return Value

None.

Error Code

None.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Oct.11.11	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141