To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

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.

To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

User's Manual

HI7000/4 Renesas Industrial Realtime Operating System

Configuration Guide

Renesas Microcomputer Development Environment System

Renesas Electronics
www.renesas.com

Rev.1.0 2003.03

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

Preface

This guide describes how to configure systems using HI7000/4.

To execute application programs registered as tasks on HI7000/4, the Solution Engine®, the product of Hitachi ULSI Systems Co., Ltd., shall be used as a target board and the HDI of the E10A emulator as a debugger in the initial debug stage. For details about HI7000/4, see the HI7000/4 Series (HI7000/4, HI7700/4, HI7750/4) User's Manual (hereinafter referred to as the HI7000/4 Series User's Manual). To create application programs and link them with HI7000/4, you should use the SuperH™ RISC engine C/C++ compiler package (hereinafter referred to as the SHC/C++ compiler) and the Hitachi Embedded Workshop (HEW), which is an integrated development tool, supplied with the SuperH™ RISC engine C/C++ compiler package.

This guide describes how to change, add and configure programs before executing the start task on multitasking operating system using the above target board, emulator, and compiler.

Related manuals

- HI7000/4 Series (HI7000/4, HI7700/4, HI7750/4) Hitachi Industrial Realtime Operating System User's Manual
- SuperH RISC engine C/C++ Compiler SH-1, SH-2, SH-2E, SH-3, SH3E, SH-4 User's Manual
- SuperH RISC engine C/C++ Compiler Assembler Optimizing Linkage Editor User's Manual
- H Series Linkage Editor, Librarian, and Object Converter User's Manual
- Hitachi Embedded Workshop 2 HEW Debugger User's Manual
- SH7616 Solution Engine™ (MS7616SE01) Overview
- The hardware manual and programming manual of the SuperH microcomputer used

The abbreviation µITRON stands for "Micro Industrial TRON". TRON, in turn, stands for "The Real-time Operating system Nucleus."

Contents

Section 1 Introduction

1.1 Overview

Follow the procedure below to run application programs on HI7000/4:

1. Create application programs.
2. Use the configurator to register the application programs to HI7000/4.
3. Build the executable file using HEW.
4. Install the application programs to the target board, and download and execute them.

This guide describes the above procedure to run the programs on the target board by using a sample program.

1.2 System Configuration

This guide describes how to create sample programs of tasks and an interrupt handler and how to run the programs on the target board.

Figure 1.1 shows an example of a hardware configuration.

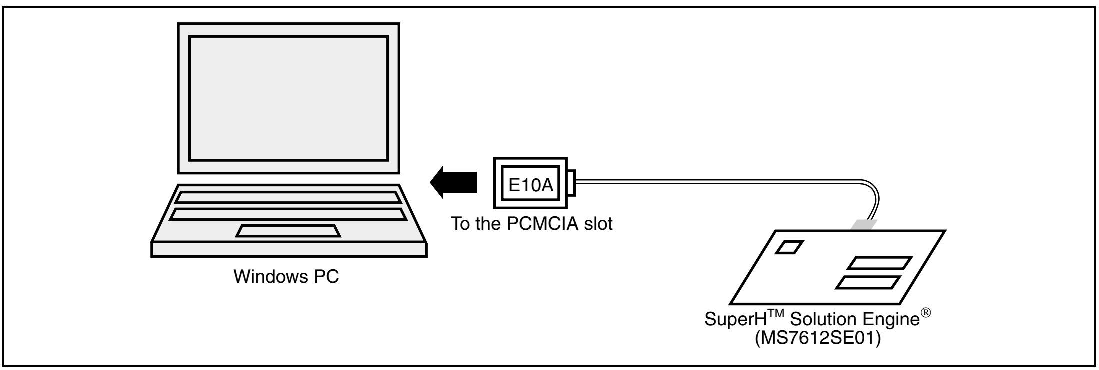


Figure 1.1 Hardware Configuration Example

Table 1.1 lists software configuration.

Table 1.1 Software Configuration

Program	Description	Type	Remarks
CPU initialization routine	Sets the bus controller. Initializes the hardware.	Non-task	
Main task	Initializes the environment. Waits for an event after initialization by setting the wai_flg flag. Cancels the wait status by setting the event flag of the timer interrupt handler and starts the LED task (sta_tsk).	Task	
LED task	Started by the main task to turn the LED on when it is off or turn it off when it is on, and then terminates.	Task	
Timer interrupt handler	Started by the timer interrupt every one second and sets the main task event flag (set_flg).	Non-task	

1.3 Prerequisites

Table 1.2 lists hardware and software required to run the application programs on HI7000/4.

Table 1.2 Required Hardware and Software

Product Name	Product Type	Manufacturer
Windows personal computer	—	Any manufacturer*1
SuperH Solution Engine	MS7612SE01	Hitachi ULSI Systems Co., Ltd.
E10A emulator	HS7612KCM01H	Hitachi, Ltd.
SuperH RISC engine C/C++ compiler	P0700CAS6-MWR	Hitachi, Ltd. *2
HI7000/4	HS0700ITI41SRE	Hitachi, Ltd. *3

Notes: 1. Hardware environment: PC/AT compatible machine with 486DX2/66 MHz or more (Pentium or later recommended)

Operating system: Windows 2000, Window NT 4.0, Windows 98, Windows 95

CD-ROM drive

PCMCIA card slot

Memory: 32 Mbytes or more (For Windows 2000 and Window NT 4.0, memory with 64 Mbytes or more is recommended.)

Free space required on the hard disk: 8 Mbytes or more

2. Version. 6.0 AR2 of the compiler shall be used. You may also use the compilers from Hitachi ULSI Systems Co., Ltd. or Hitachi Software Engineering Co., Ltd.

HI7000/4 with evaluation license (object) shall be used. You may also use HI7000/4 with mass-production license.

The HDI of the E10A emulator, SuperH RISC engine C/C++ compiler package, and HI7000/4 (for SHCV6) must have been installed in the Windows personal computer beforehand. The SH7612 is a target CPU assumed in this manual.

Figure 1.2 shows the folder structure of HI7000/4 that you have just installed.

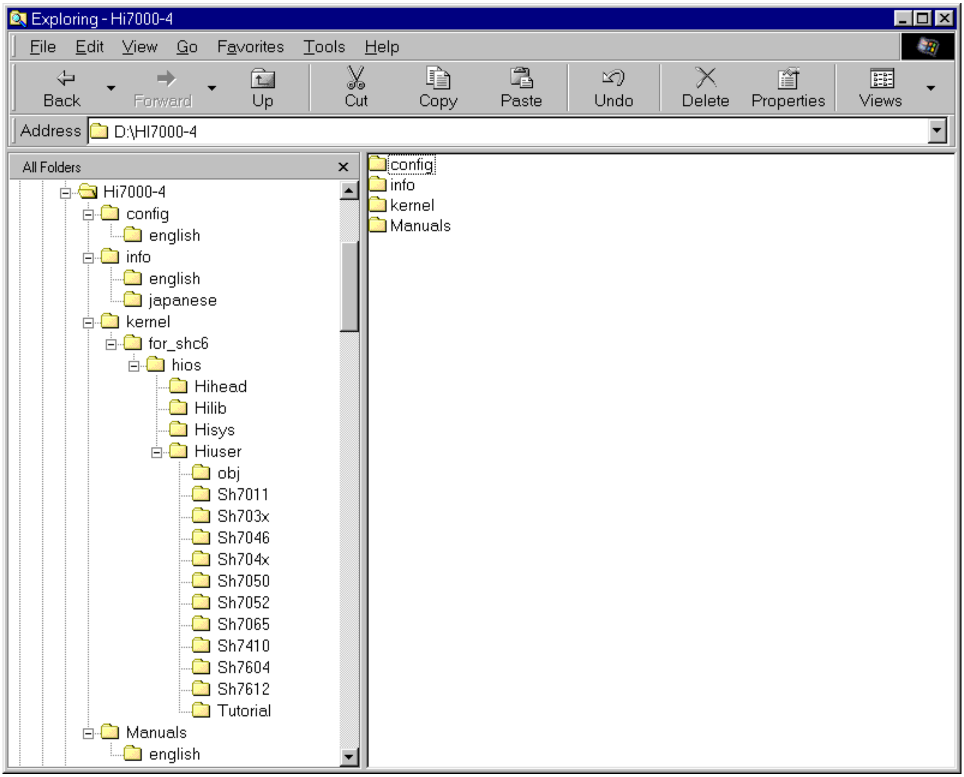


Figure 1.2 Folder Structure of HI7000/4

The install drive is "D" in this guide, but you may use a desired drive for installing HI7000/4. An install folder is represented as the install folder "folder name" in this manual.

Section 2 Creating Application Programs

This section describes how to create application programs that run on HI7000/4. Figure 2.1 shows the relationship among application programs. (The programs in the heavy-outline boxes are created in this guide.)

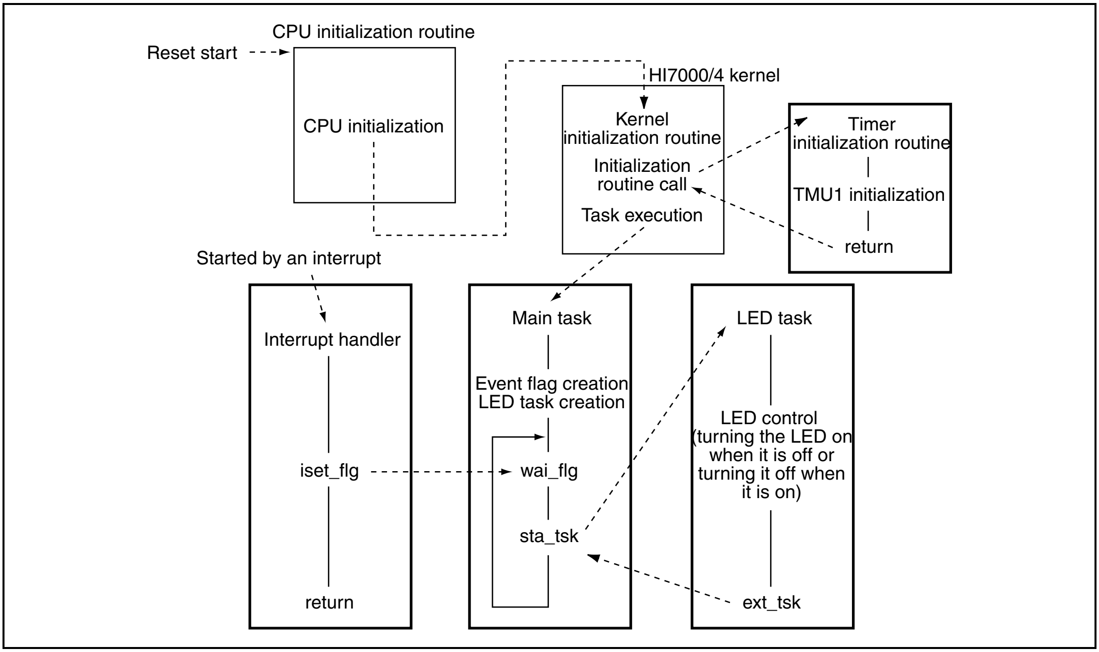


Figure 2.1 Relationship among Application Programs

Figure 2.2 shows the programs to be created in this guide.

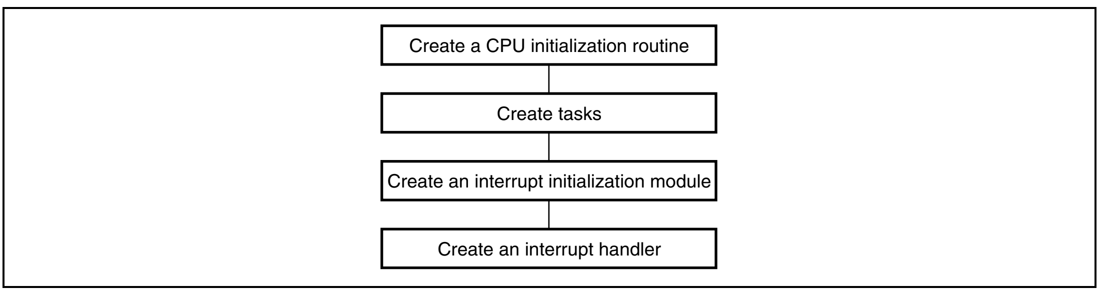


Figure 2.2 Programs to be Created

2.1 Creating CPU Initialization Routine

After the CPU reset, the CPU initialization routine is executed for setting a bus state controller and initializing the hardware.

The ROM monitor supplied with the Solution Engine has already set the bus state controller and initialized the hardware. Thus, this guide omits the description of them.

Figure 2.3 shows the procedure to create the CPU initialization routine.

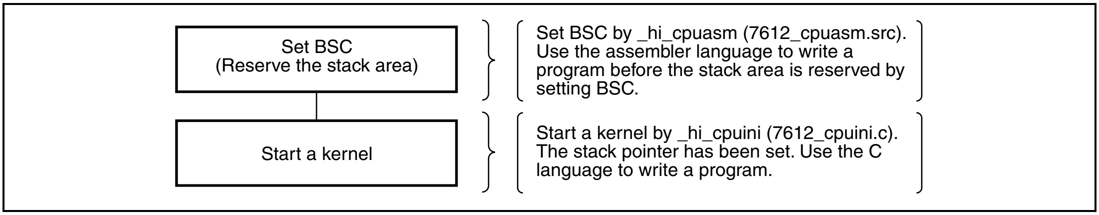


Figure 2.3 Creating a CPU Initialization Routine

In the CPU initialization routine, the stack area must be reserved completely before you attempt to execute any program written in the C language. Because the program created by the compiler may locate the stack frame or work area in a stack, you cannot execute it until the stack area is completely reserved.

Figures 2.4 to 2.6 show the parts to be changed in of _hi_cpuasm (7612_cpuasm.src).

```
;***************************************************************************;
;*              HI7000/4 CPU initialize routine                            ;*;
;*              Copyright (c)  Hitachi, Ltd. 2000.                         ;*;
;*              Licensed Material of Hitachi, Ltd.                         ;*;
;*              HI7000/4(HS0700ITI41SR) V1.0                               ;*;
;***************************************************************************;
;***************************************************************************;
;*  FILE      = 7612_cpuasm.src ;                                          ;*;
;*  CPU type  = SH7612                                                     ;*;
;***************************************************************************;
        .program        _hi_cpuasm
        .heading        "hi_cpuasm : CPU initialize routine"
        .export         _hi_cpuasm
        .import         _hi_cpuini
        .section        P_hicpuasm,code,align=4
;
;***************************************************************************;
;* BSC address                                                             ;*;
;***************************************************************************;
BSC_BASE      .assign  h'ffffffc0          ; BSC   base address (WCR2)
BCR1          .assign  h'ffffffe0-BSC_BASE ; BCR1  address offset
CR2          .assign  h'ffffffe4-BSC_BASE ; BCR2  address offset
BCR3          .assign  h'fffffffc-BSC_BASE ; BCR3  address offset
WCR1          .assign  h'ffffffe8-BSC_BASE ; WCR1  address offset
WCR2          .assign  h'ffffffc0-BSC_BASE ; WCR2  address offset
WCR3          .assign  h'ffffffc4-BSC_BASE ; WCR3  address offset
MCR           .assign  h'ffffffec-BSC_BASE ; MCR   address offset
RTCSR         .assign  h'fffffff0-BSC_BASE ; RTCSR address offset
RTCNT         .assign  h'fffffff4-BSC_BASE ; RTCNT address offset
RTCOR         .assign  h'fffffff8-BSC_BASE ; RTCOR address offset
;
MD_REG_BASE   .assign  h'ffff8000          ; mode register base address of SDRAM
;
CMF_BIT       .assign  h'0080              ; CMF bit in RTCSR
;
```

Figure 2.4   Parts to be Changed in _hi_cpuasm (7612_cpuasm.src)

```
;****************************************************************************;
;* BSC initial data                                                         ;*;
;*  After reset, you must initialize BSC for memory(stack) access at first.;*;
;*  Please modify these definition in order to your hardware.               ;*;
;****************************************************************************;
BCR1_DATA     .assign  h'a55a0000 + h'03f0 ; BCR1  initial data
BCR2_DATA     .assign  h'a55a0000 + h'00fc ; BCR2  initial data
BCR3_DATA     .assign  h'a55a0000 + h'0f00 ; BCR3  initial data
WCR1_DATA     .assign  h'a55a0000 + h'aaff ; WCR1  initial data
WCR2_DATA     .assign  h'a55a0000 + h'000b ; WCR2  initial data
WCR3_DATA     .assign  h'a55a0000 + h'0000 ; WCR3  initial data
MCR_DATA      .assign  h'a55a0000 + h'0000 ; MCR   initial data
RTCSR_DATA    .assign  h'a55a0000 + h'0000 ; RTCSR initial data
RTCNT_DATA    .assign  h'a55a0000 + h'0000 ; RTCNT initial data
RTCOR_DATA    .assign  h'a55a0000 + h'0000 ; RTCOR initial data
;
STP_REFRESH   .assign  h'a55a0000          ; RTCSR initial data(stop count-up)
;
MODE_DATA     .assign  h'0000              ; data of SDRAM mode register
MODE_ADDRESS  .assign  MD_REG_BASE+MODE_DATA ; address to set MODE_DATA
IDLE_TIME     .assign  566                 ; loop counter for idle-time
REFRESH_CNT   .assign  h'8                 ; counter for dummy refresh
;
;****************************************************************************;
;*  NAME      = _hi_cpuasm                                                  ;*;
;*  FUNCTION  = CPU initialize routine ;                                    ;*;
;****************************************************************************;
_hi_cpuasm:
;***** Initialize BSC
;       mov.l   #BSC_BASE,r0            ; set BCR base address to gbr
;       ldc     r0,gbr
;
;       mov.l   #BCR1_DATA,r0           ; initialize BCR1
;       mov.l   r0,@(BCR1,gbr)
;
;       mov.l   #BCR2_DATA,r0           ; initialize BCR2
;       mov.l   r0,@(BCR2,gbr)
;
;       mov.l   #BCR3_DATA,r0           ; initialize BCR3
;       mov.l   r0,@(BCR3,gbr)
;
;       mov.l   #WCR1_DATA,r0           ; initialize WCR1
;       mov.l   r0,@(WCR1,gbr)
;
;       mov.l   #WCR2_DATA,r0           ; initialize WCR2
;       mov.l   r0,@(WCR2,gbr)
;
;       mov.l   #WCR3_DATA,r0           ; initialize WCR3
;       mov.l   r0,@(WCR3,gbr)
;
;       mov.l   #MCR_DATA,r0            ; initialize MCR
;       mov.l   r0,@(MCR,gbr)
;
;       mov.l   @(RTCSR,gbr),r0         ; dummy read for CMF off
;       mov.l   #STP_REFRESH,r0         ; stop refresh
;       mov.l   r0,@(RTCSR,gbr)
;
;       mov.l   #RTCNT_DATA,r0          ; initialize RTCNT
;       mov.l   r0,@(RTCNT,gbr)
;
;       mov.l   #RTCOR_DATA,r0          ; initialize RTCOR
;       mov.l   r0,@(RTCOR,gbr)
;
;       mov.l   #RTCSR_DATA,r0          ; initialize RTCSR
;       mov.l   r0,@(RTCSR,gbr)
```

Change the BSC value according to the hardware

Omit the comment to set BSC

Figure 2.5 Parts to be Changed in _hi_cpuasm (7612_cpuasm.src)

```
;*** Initialize SDRAM
;
;       mov.l   #IDLE_TIME,r0           ; loop for idle-time
;hi_cpuasm010:
;       add     #-1,r0
;       cmp/eq  #0,r0
;       bf      hi_cpuasm010
;
;       mov.w   #MODE_DATA,r0           ; set mode register
;       mov.l   #MODE_ADDRESS,r1
;       mov.w   r0,@r1
;
;       mov.l   #RTCSR_DATA,r0          ; initialize RTCSR
;       mov.l   r0,@(RTCSR,gbr)
;
;       mov     #0,r1                   ; loop for dummy refresh
;       mov.w   #REFRESH_CNT,r2
;hi_cpuasm020:
;       mov.l   @(RTCSR,gbr),r0
;       tst     #CMF_BIT,r0             ; check CMF bit
;       bt      hi_cpuasm020
;
;       add     #1,r1                   ; loop counter up
;       cmp/eq  r1,r2                   ; if end dummy refresh
;       bt      hi_cpuasm030            ; then goto hi_cpuasm030
;       mov.l   #RTCSR_DATA,r0          ; clear CMF bit
;       bra     hi_cpuasm020
;       mov.l   r0,@(RTCSR,gbr)
;
;hi_cpuasm030:
;
        mov.l   #_hi_cpuini,r0          ; get hi_cpuini address
        jmp     @r0                     ; jump to hi_cpuini()
        nop                             ; never return to this point
;
                .pool
;
         .end
```

Omit the comment to set BSC.

Jump to hi_cpuini

Figure 2.6 Parts to be Changed in _hi_cpuasm (7612_cpuasm.src)

Figure 2.7 shows the part to be changed in _hi_cpuini (7612_cpuini.c).

```
/*****************************************************************************/
/*              HI7000/4 CPU initialize routine                              */
/*              Copyright (c)  Hitachi, Ltd. 2000.                           */
/*              Licensed Material of Hitachi, Ltd.                           */
/*              HI7000/4(HS0700ITI41SR) V1.0                                 */
/*****************************************************************************/
/*****************************************************************************/
/*  FILE      = 7612_cpuini.c ;                                              */
/*  CPU type  = SH7612                                                       */
/*****************************************************************************/
#include    <machine.h>
#include    "itron.h"
#include    "kernel.h"

/* extern  void    _INITSCT(void); */   /* section-initialize routine       */

#pragma section _hicpuini
#pragma noregsave(hi_cpuini)

void    hi_cpuini(void)
{

/*** Initialize Hardware Environment ***/

/*** Initialize Software Environment ***/

/*  _INITSCT();    */                   /* Call section-initialize routine  */

    vsta_knl();                         /* Start kernel                     */
}
```

Start a kernel

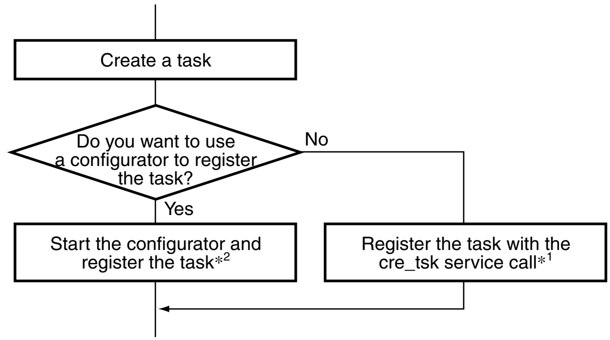
Figure 2.7   Part to be Changed in _hi_cpuini (7612_cpuini.c)

Set a bus state controller and create a hardware initialization routine for the specific hardware.

2.2 Creating Tasks

A task is the main processing of an application program.

Figure 2.8 shows the procedure to create and register a task.



Notes: 1. The cre_tsk service call must be enabled to register the task. For details, see the HI7000/4 Series User's Manual.
2. This procedure is described in section 3, Configurator.

Figure 2.8 Creating and Registering Task

Create a task by changing the sample (task.c) supplied with HI7000/4. The sample is in the install folder “tutorial”.

In this guide, the main task (MainTask) is registered by the configurator and the LED task by the cre_tsk service call.

2.2.1 Main Task

This section describes how to change MainTask contained in the sample program (task.c) supplied with HI7000/4. Figure 2.9 shows the overview of changes made in MainTask. Starting task7 periodically turns the LED on and off.

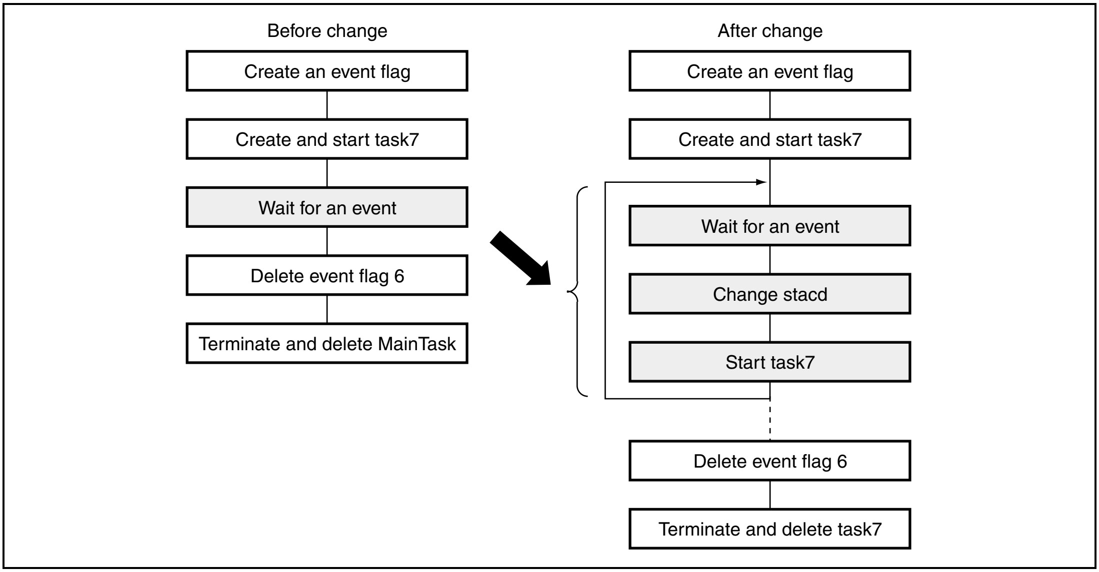


Figure 2.9 Overview of Changes Made in MainTask

Figure 2.10 shows the parts to be changed in MainTask.

```
#include <machine.h>
```

Define the include file supplied with the C compiler.

```
#include "itron.h"
#include "kernel.h"
#include "kernel_id.h"
```

Required when using the HI7000/4 service call.

```
#define LED_ADR (UH *)0x22200000
```

Define the LED output port address. For details, see the Solution Engine Overview Manual.

```
void MainTask(VP_INT exinf);
void task7(VP_INT exinf);
```

MainTask and task7 are the main functions for each task. Another function never calls them. #pragma noregsave is valid to suppress the stack area.

```
#pragma noregsave(MainTask, task7)
```

```
/************************************************************************
 * MainTask()
 * This task is created and activated by Configurator.
 *  tskid : "ID_MainTask" (defined in kernel_id.h as this task's ID.)
 *  itskpri : 6
 ************************************************************************/
void MainTask(VP_INT exinf)
{
 union CrePacket{
     T_CTSK  t_ctsk;  /* Creation info. for task */
     T_CFLG  t_cflg;  /* Creation info. for eventflag */
 } packet;

 ER ercd;
 FLGPTN waiptn, flgptn;

 /*** Create eventflag-6 ***/
 packet.t_cflg.flgatr = TA_TFIFO|TA_WSGL|TA_CLR;
 packet.t_cflg.iflgptn = 0;

 ercd = cre_flg(6, (T_CFLG *)&packet);

 /*** Create task-7 ***/
 packet.t_ctsk.tskatr = TA_HLNG|TA_ACT;
 packet.t_ctsk.exinf = 0;
 packet.t_ctsk.task = (FP)task7;
 packet.t_ctsk.itskpri = 7;
 packet.t_ctsk.stksz = 0x200;
 packet.t_ctsk.stk = (VP)NULL;

 ercd = cre_tsk(7, (T_CTSK *)&packet);

 /*** Wait for eventflag-6 ***/
 waiptn = 0x11111111;
 ercd = wai_flg(6, waiptn, TWF_ANDW, &flgptn);

 /*** Delete eventflag-6 ***/
 ercd = del_flg(6);

 ext_tsk();
}
```

Define the task attribute.
If task7 uses the DSP, use the OR operator to define TA_COP0.
(packet.t_ctsk.tskatr = TA_HLNG | TA_ACT | TA_COP0)
This allows the DSP register to be saved (to guarantee a kernel) when changing a task.

Register the task and start it.

The boxed "Wait for eventflag-6" block is changed to:

```
for(;;) {
 /*** Wait for eventflag-6 ***/
 waiptn = 0x11111111;
 ercd = wai_flg(6, waiptn, TWF_ANDW, &flgptn);
 if(exinf == 0x00000000L) {
      exinf = 0x0000ff00L;
 } else {
      exinf = 0x00000000L;
 }
 ercd = sta_tsk(7,exinf);
}
```

Wait for the event flag to set by the interrupt handler. Change the exinf value and start task7. At this time, exinf is passed to task7 as a start code.

Figure 2.10 Changing MainTask

2.2.2 LED Task

This section describes how to change task7 of the sample program (task.c) supplied with HI7000/4. Figure 2.11 shows the part to be changed in task7.

```
/***************************************************************************
 * task7()
 * This task is created and activated by MainTask.
 *  tskid : 7
 *  itskpri : 7
 ***************************************************************************/
void task7(VP_INT exinf)
{
     ER  ercd;
     ercd = set_flg(6, 0xffffffff);    =>    *LED_ADR = (UH)exinf;
                                             ext_tsk();
     while(1);
}
```

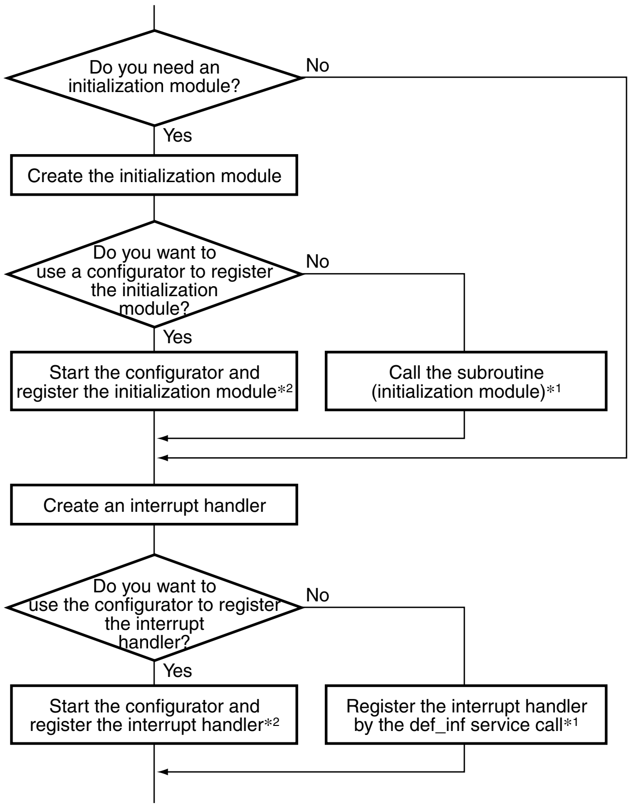
Turn the LED on or off according to the exinf value, and then terminate itself (task7)

Figure 2.11 Changing task7

2.3 Creating an Interrupt Handler

The interrupt handler is started by an external interrupt that suspends another processing.

Figure 2.12 shows the procedure to create and register the initialization module and the interrupt handler.



Notes: 1. The def_inf service call must be enabled to register the interrupt handler. For details, see the HI7000/4 Series User's Manual.
2. This procedure is described in section 3, Configurator.

Figure 2.12 Creating and Registering Initialization Module and Interrupt Handler

This guide describes how to use the on-chip TPU2 in the SH7612 to create the interrupt handler and how to use a configurator to register it.

Create the tpu2.c file for the initialization module and the interrupt handler and store the file in the install folder "tutorial".

Table 2.1 lists the interrupt conditions.

Table 2.1 Interrupt Conditions

Item	Description	Function	File Name
Initialization module	Required. Use the configurator to register the module.	TPU2_ini	tpu2.c
Interrupt handler	Use the configurator to register the handler.	TPU2_int	tpu2.c
Interrupt cycle	An interrupt occurs every one second.	—	—
Interrupt level	1	—	—

2.3.1 Creating Initialization Module

This section describes how to create an initialization module for the on-chip TPU2 in the SH7612. The initialization module initializes the TPU2 and sets the interrupt cycle and level. Figure 2.13 shows the procedure to create the initialization module.

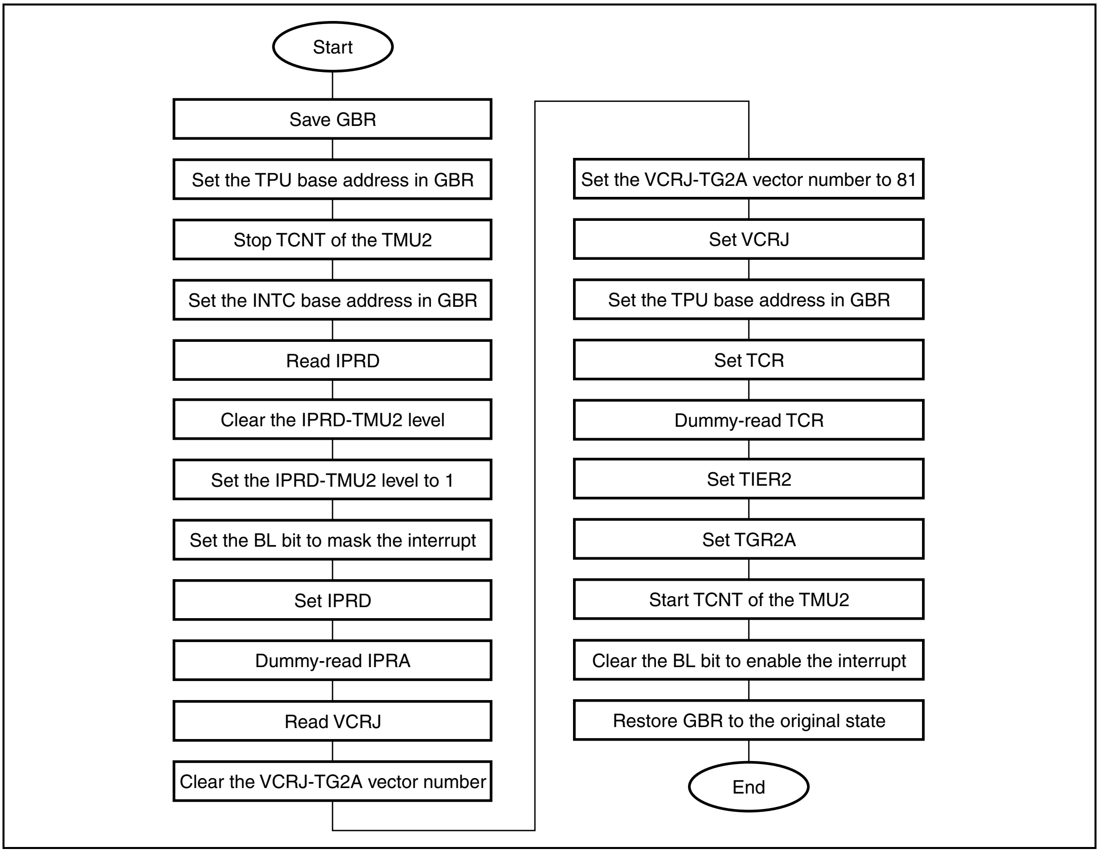


Figure 2.13 Creating Initialization Module

Figures 2.14 and 2.15 show the contents of TPU2_ini (tpu2.c).

```
#include <machine.h>
#include "itron.h"
#include "kernel.h"

#define BL_BIT 0x10000000               /* BL bit pattern                      */

/* peripheral clock (FMR set value = H'0E(CPU:Bus:P=60:60:30MHz))              */
#define PCLK            30000000

/* TSTR set value */
#define TCNT2_STA       0x04            /* Start TCNT of the TPU2              */
#define TCNT2_STP       0xfb            /* Stop TCNT of the TPU2               */

/* TCR2 set value */
#define TCNT_CLR        0x20            /* Clear TCNT by compare match of TGRA */
#define DIV1024         0x07            /* Division ratio: 1/1024              */

/* TIER2 set value */
#define TGFA            0x01            /* Enable an interrupt by the TGFA bit */

/* TGR2A set value */
#define INTERVAL        1000000         /* 1s:1000ms:1000000us                 */
#define DIV             1024            /* Division ratio: 4                   */
#define TCNT2_DAT  (UH)(((double)INTERVAL /(((double)1000000/(double)PCLK)*(double)DIV))-(double)1)
                                        /*(1 second/((1 second/30 MHz)*1024))-1 */
/* IPRD set value */
#define IPRD_CLR_TPU2   0xff0f          /* IPR bit4-7 clear data               */
#define TPU2_LVL        1               /* TPU2 interrupt level = 1            */

/* VCRJ set value */
#define VCRJ_CLR_TG2A   0x00ff          /* VCRJ-TG2A(bit8-15) clear data       */
#define TG2A_VCT        81              /* VCRJ-TG2A vector number : 81        */

/* TPU,IPRD I/O address */
#define INTC_BASE    0xfffffe00                 /* INTC base address           */
#define IPRD        (0xfffffe40 - INTC_BASE)    /* INTC IPR(IPRD:TPU-ch2)      */
#define VCRJ        (0xfffffe4c - INTC_BASE)    /* INTC VCRJ(TPU2-TG2A)        */

#define TPU_BASE     0xfffffc00                 /* INTC base address           */
#define TSTR        (0xfffffc40 - TPU_BASE)     /* TPU TSTR                    */
#define TCR2        (0xfffffc70 - TPU_BASE)     /* TPU TCR (ch2)               */
#define TIER2       (0xfffffc74 - TPU_BASE)     /* TPU TIER (ch2)              */
#define TSR2        (0xfffffc75 - TPU_BASE)     /* TPU TSR (ch2)               */
#define TCNT2       (0xfffffc76 - TPU_BASE)     /* TPU TCNT (ch2)              */
#define TGR2A       (0xfffffc78 - TPU_BASE)     /* TPU TCNT (ch2)              */
```

Figure 2.14 Contents of TPU2_ini (tpu2.c)

```
/****************************************************************************/
/*  NAME      = TPU2_ini                                                    */
/*  FUNCTION  = Initialize TPU2                                             */
/****************************************************************************/
void TPU2_ini(void)
{
    VP      gbrsave;                      /* GBR save area                          */
    UH      iprd;                         /* IPRD retention area                    */
    UH      vcrj;                         /* VCRJ retention area                    */

    gbrsave = get_gbr();                  /* Save GBR                               */
    set_gbr((VP)TPU_BASE);                /* Set the TPU base address in GBR        */

    gbr_and_byte(TSTR,TCNT2_STP);         /* Stop TCNT of the TPU2                  */

    set_gbr((VP)INTC_BASE);               /* Set the INTC base address in GBR       */
    iprd = gbr_read_word(IPRD);           /* Read IPRD                              */
    iprd &= IPRD_CLR_TPU2;                /* Clear the IPRD-TPU2 level              */
    iprd |= TPU2_LVL << 4;                /* Set the IPRD-TPU2 level to 1           */

    set_cr(BL_BIT | get_cr());            /* Set the BL bit to mask the interrupt   */
    gbr_write_word(IPRD,iprd);            /* Set IPRD                               */
    gbr_read_word(IPRD);                  /* Dummy-read IPRD                        */

    vcrj = gbr_read_word(VCRJ);           /* Read VCRJ                              */
    vcrj &= VCRJ_CLR_TG2A;                /* Clear the VCRJ-TG2A vector number      */
    vcrj |= TG2A_VCT << 8;                /* Set the VCRJ-TG2A vector number to 81  */
    gbr_write_word(VCRJ,vcrj);            /* Set VCRJ                               */

    set_gbr((VP)TPU_BASE);                /* Set the TPU base address in GBR        */
    gbr_write_byte(TCR2,TCNT_CLR|DIV1024); /* Set TCR                               */
    gbr_read_byte(TCR2);                  /* Dummy-read TCR                         */

    gbr_write_byte(TIER2,TGFA);           /* Set TIER2                              */
    gbr_write_word(TGR2A,TCNT2_DAT);      /* Set TGR2A                              */

    gbr_or_byte(TSTR, TCNT2_STA);         /* Start TCNT of the TPU2                 */

    set_cr(~BL_BIT & get_cr());           /* Clear the BL bit to enable the interrupt */
    set_gbr(gbrsave);                     /* Restore GBR to the original state      */
}
```

Figure 2.15 Contents of TPU2_ini (tpu2.c)

2.3.2 Creating Interrupt Handler

This section describes how to create an interrupt handler for the on-chip TPU2 in the SH7612. The interrupt handler clears an interrupt source of the TPU2 and issues an event flag to task7. Figure 2.16 shows the procedure to create the interrupt handler.

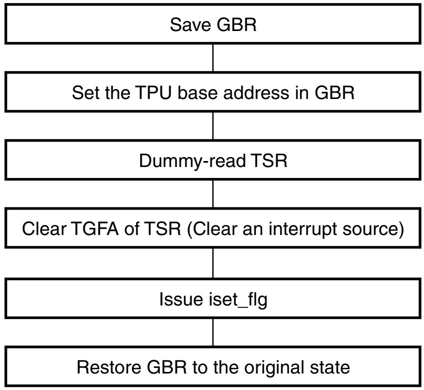


Figure 2.16 Creating Interrupt Handler

Figure 2.17 shows the contents of TPU2_int (tpu2.c).

```
/*************************************************************************/
/*  NAME      = TPU2_int                                                 */
/*  FUNCTION  = TPU2 interrupt handler                                   */
/*************************************************************************/
void    TPU2_int(void)
{
    VP      gbrsave;                        /* GBR save area                       */
    UB      tsr2;                           /* TSR2 retention area                 */

    gbrsave = get_gbr();                    /* Save GBR                            */
    set_gbr((VP)TPU_BASE);                  /* Set the TPU base address in GBR     */

    tsr2 = gbr_read_byte(TSR2);             /* Dummy-read TSR                      */
    gbr_write_byte(TSR2,(tsr2 & ~TGFA)); /* Clear TGFA                          */

    iset_flg(6, 0xffffffff);                /* Set an event flag for task7         */

    set_gbr(gbrsave);                       /* Restore GBR to the original state */
}                                           /* ret_int                             */
```

Figure 2.17 Contents of TPU2_int (tpu2.c)

Section 3 Configuration

Configuration means to register the programs created in section 2 to HI7000/4. HI7000/4 provides a tool that allows easy configuration using GUI and a configurator.

This section describes how to use the configurator to register the application programs.

Figure 3.1 shows the programs to be registered in this guide.

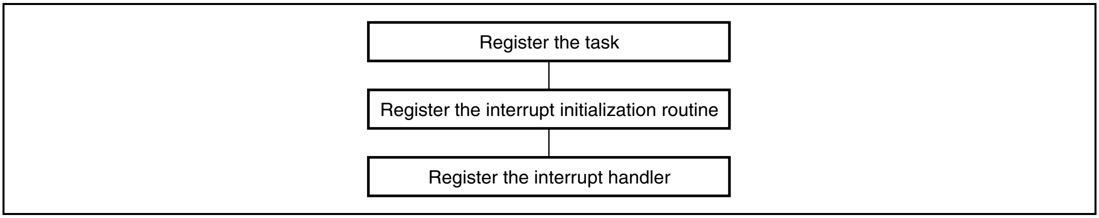


Figure 3.1 Programs to be Registered

The defaults are used for programs other than those above.

For details of each program set by the configurator, see the Configurator Help.

3.1 Starting Configurator

Double-click the configurator set file (7612.hcf) to start the configurator. The 7612.hcf file is in the install folder “sh7612”.

Figure 3.2 shows the Configurator Startup screen.

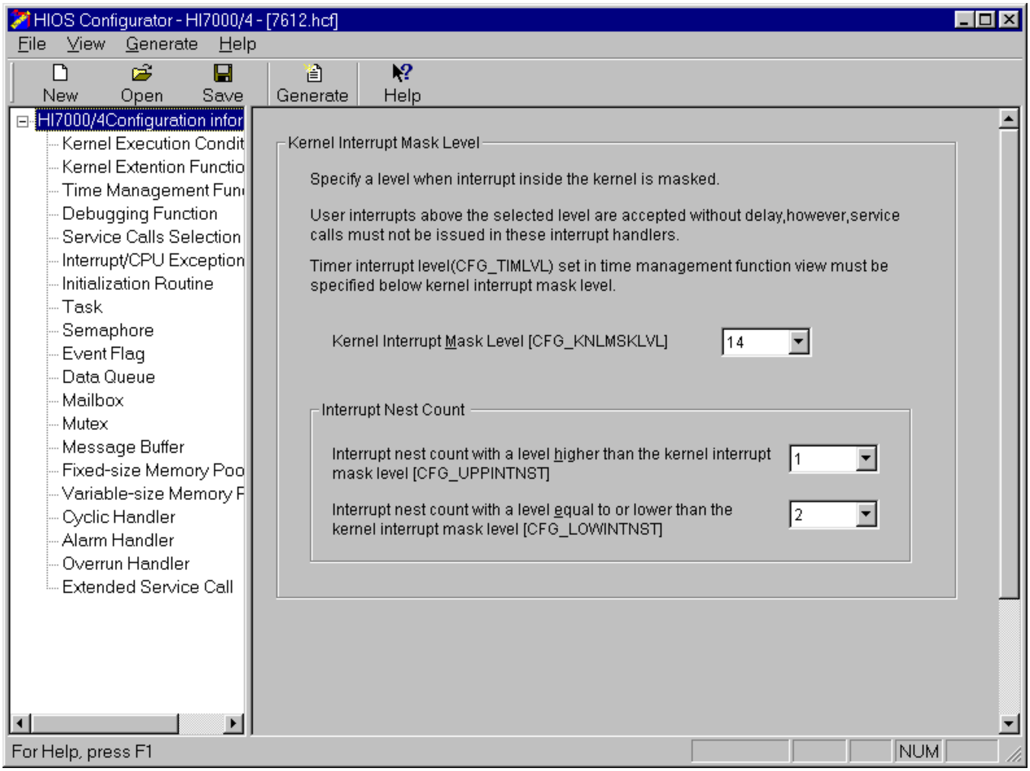


Figure 3.2 Configurator Startup Screen

3.2 Registering Task

Click Task in the HI7000/4 Configuration Information area on the Configuration Startup screen to view the Task Information screen in figure 3.3.

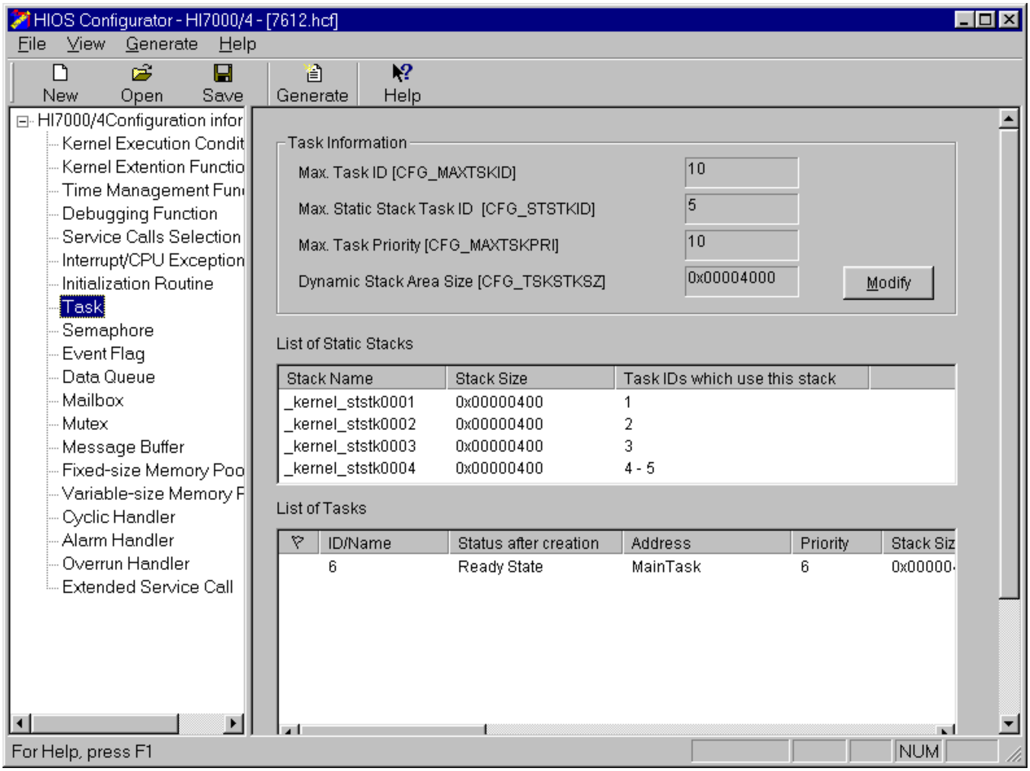


Figure 3.3 Task Information Screen

Click the Change button in the Task Information area in figure 3.3 to view the Modification of Task Information screen in figure 3.4.

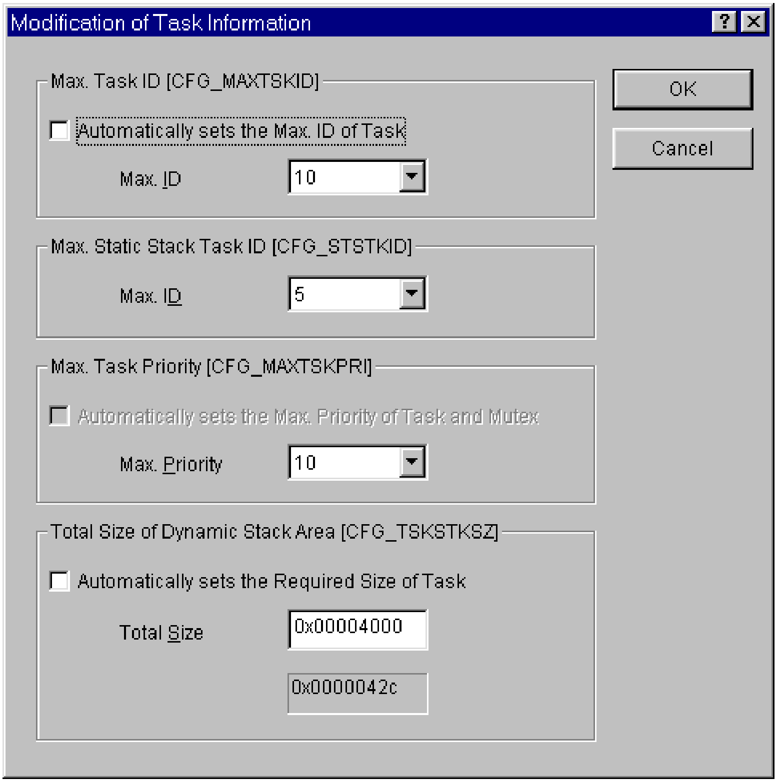


Figure 3.4 Modification of Task Information Screen

On this screen, you can change the maximum task ID, the maximum task ID using static stacks, maximum task priority, and the total size of the dynamic stack area. For details about differences between static stacks and dynamic stacks, see section 2.6.6, Task Stack, in the HI7000/4 Series User's Manual.

For details about how to calculate the task stack size, see Appendix C, Calculation of Work Area Size, in the HI7000/4 Series User's Manual.

In this guide, the defaults are used for registering the task. You do not need to change the task information.

3.3 Registering Interrupt Handler

Click Interrupt and CPU Exception Handler in the HI7000/4 Configuration Information area on the Configuration Startup screen to view the List of Interrupt/CPU/Trap Exception Handlers screen in figure 3.5.

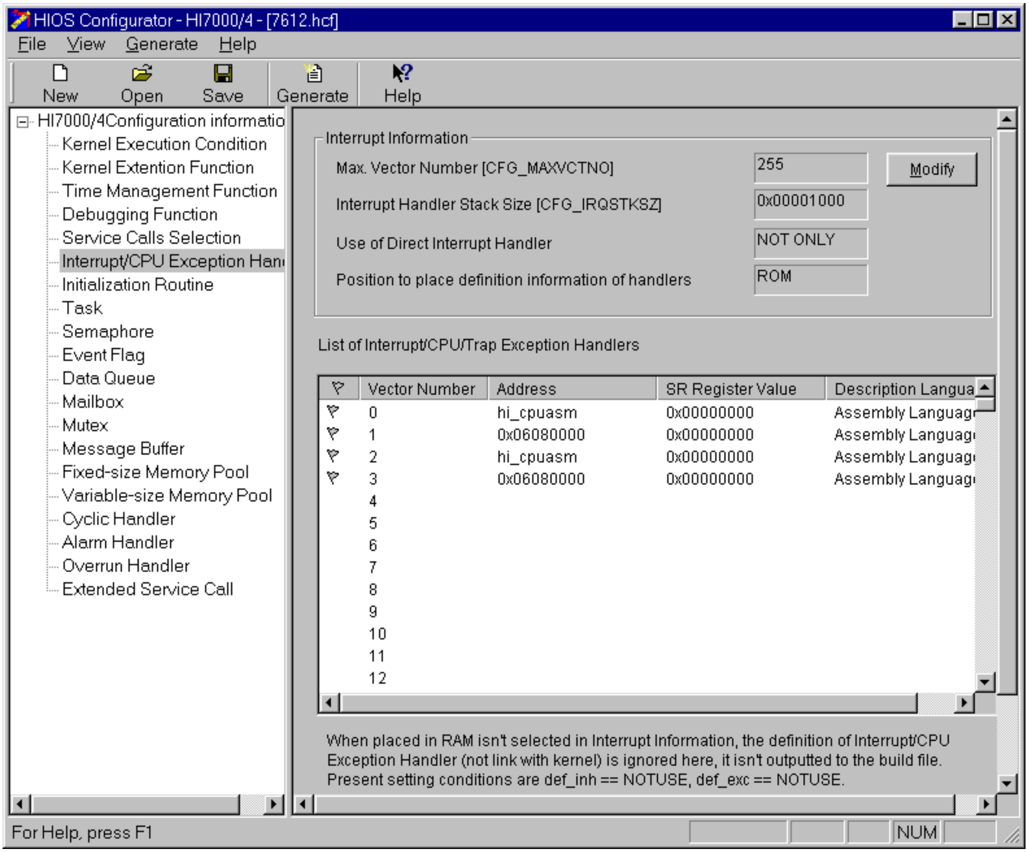


Figure 3.5 List of Interrupt/CPU/Trap Exception Handlers Screen

The following sections describes how to register an interrupt handler including the setting of the stack pointer address to be set at a power-on or a manual reset.

3.3.1 Registering Stack Pointer Addresses for Reset Exception

For the SH-1/SH-2 core CPU, you must set the stack pointer addresses in vector addresses 1 and 3 for reset exception.

The SH7612 Solution Engine is supplied with the 32 Mbytes SDRAM between 0x0600000 and 0x07FFFFFF. In this guide, 16 Mbytes between 0x0600000 and 0x06FFFFFF are used and the end address of the RAM area to be used + 1 address (0x27000000: cache through area) is set for the stack pointer addresses. Figure 3.6 shows the screen for registering the stack pointer addresses for the reset exception.

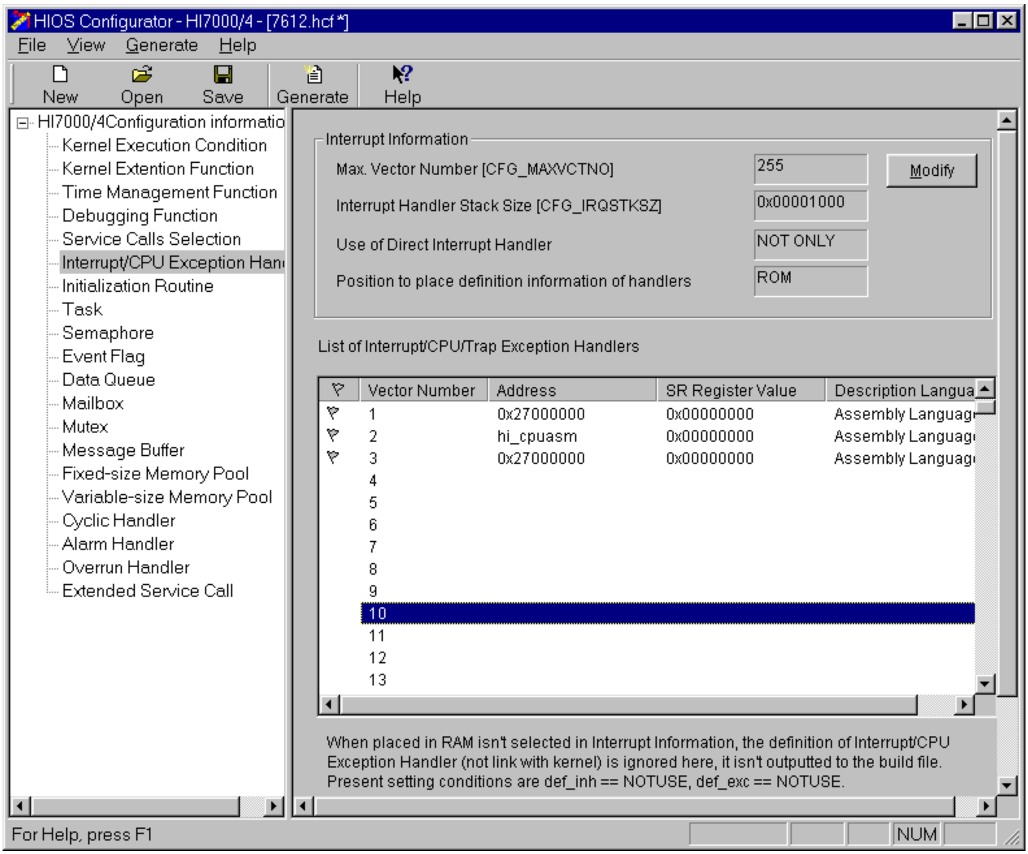


Figure 3.6 Registering the Stack Pointer Addresses for Reset Exception

RENESAS

3.3.2 Registering Interrupt Handler

In this guide, the on-chip TPU2 in the SH7612 is used as an interrupt source.

For the SH-1/SH-2 core CPU, you can freely use the vector numbers between 0 and 127 of the on-chip peripheral module.

The system timer interrupt handler supplied with HI7000/4 uses the FRT and allocates its vector number to 80. The timer interrupt handler implemented in this guide allocates the vector number to 81.

Use the mouse on the scroll bar on the right of the List of Interrupt/CPU/Trap Exception Handlers to specify vector number 81. Double click vector number 81 to view the Defination of Interrupt/CPU/Trap Exception Handler screen in figure 3.7.

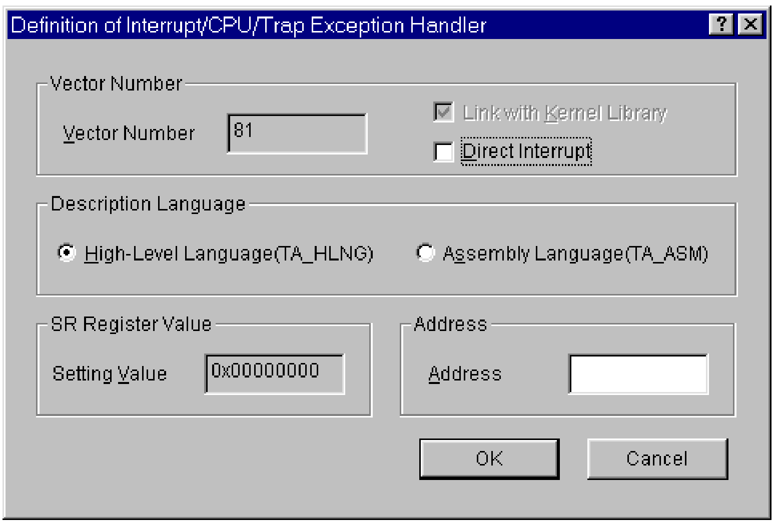


Figure 3.7 Definition of Interrupt/CPU/Trap Exception Handler Screen

Set TPU2_int in the Address box.

Uncheck the Direct Interrupt checkbox to issue the iset_flg service call from the timer interrupt handler.

Table 3.1 lists the type of interrupt (direct interrupt and normal interrupt).

Table 3.1 Direct Interrupt and Normal Interrupt

Type	Description	Note
Direct interrupt	• Directly activates an interrupt handler not via a kernel • Implements a high-speed interrupt response	• The service call of a kernel cannot be used • The handler must be written by #pragma interrupt*
Normal interrupt	• A kernel manages an interrupt • The interrupt handler can be written in the subroutine (function) format • The service call of a kernel can be issued	• The time required for an interrupt response via kernel is longer compared with the direct interrupt

Note: * For an example of the interrupt handler written by #pragma interrupt, see section 4.7.2, Direct Interrupt Handler (HI7000/4), in the HI7000/4 Series User's Manual.

The direct interrupt is generally used to process urgent interrupts if a system error occurs. Use the appropriate type of interrupt depending on the type of processing an interrupt.

The SR register set value is meaningless. The interrupt handler is processed according to the priority that has been set in the CPU interrupt control register.

Figures 3.8 and 3.9 show the Definition of Interrupt/CPU/Trap Exception Handler screen after you made definitions.

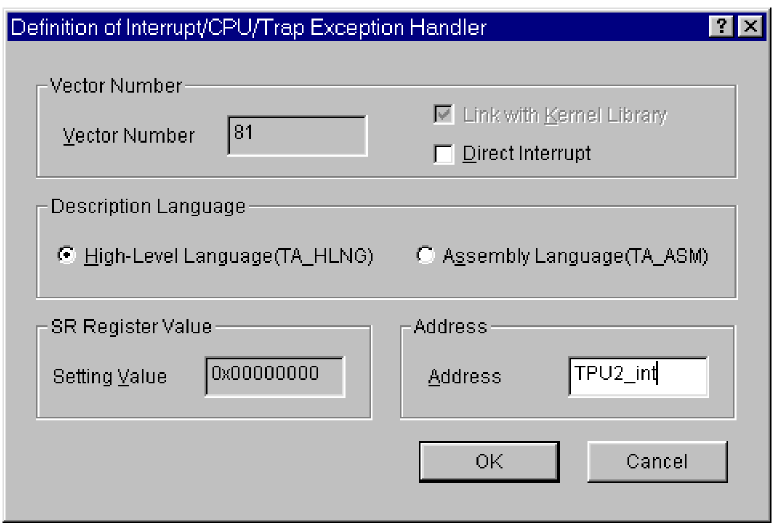


Figure 3.8 Definition of Interrupt/CPU/Trap Exception Handler Screen (after Making Definitions)

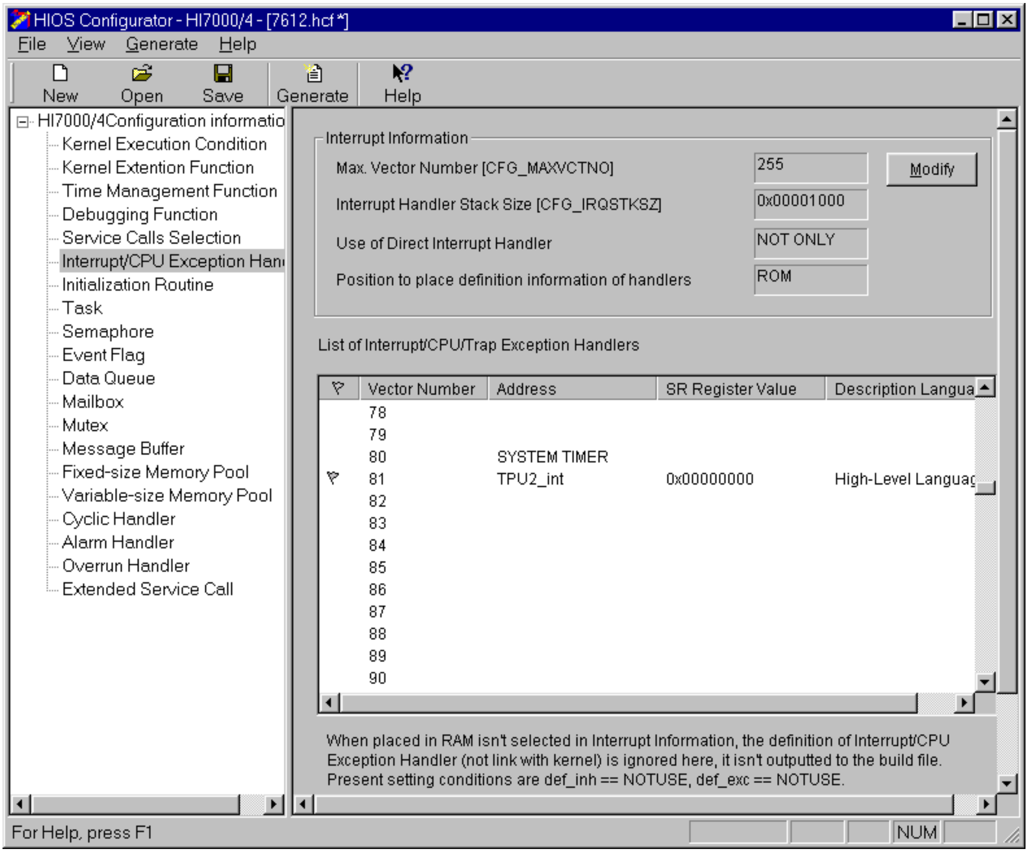


Figure 3.9 List of Interrupt/CPU/Trap Exception Handler Screen (after Making Definitions)

RENESAS

3.4 Registering Initialization Routine

Click Initialization Routine in the HI7000/4 Configuration Information area on the Configurator Startup screen to view the List of Initialization Routines screen in figure 3.10.

The initialization routine that is registered on this screen is called immediately after the kernel startup (setup) completes and executed with the kernel mask level (the value set for the kernel operational conditions in the configuration information). This routine differs from the CPU initialization routine that is executed immediately after a reset.

In the initialization routine, the service call of a kernel can be issued.

The issuable service call is the one that can be called from non-task context (system state: N) described in section 3, Service Calls, in the HI7000/4 Series User's Manual.

The initialization routine is used for the following purposes:

1. Interrupt initialization
2. Initialization routine for task setup
3. Event flag, mailbox, or memory pool of which initial setting is to be completed before passing the control to a task or an interrupt handler

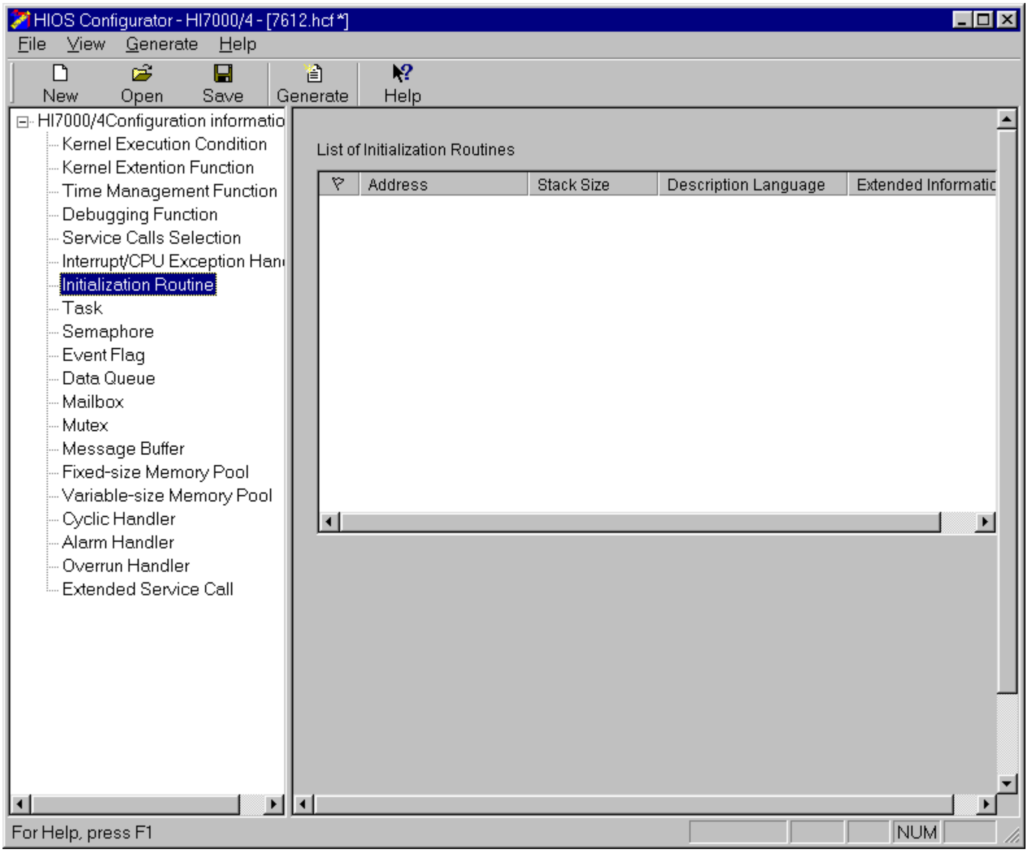


Figure 3.10 List of Initialization Routines Screen

Right click on the blank area of the List of Initialization Routines to view the menu. Then, select Register to view the Registration of Initial Initialization Routine screen in figure 3.11.

The following explains how to register the initial routine.

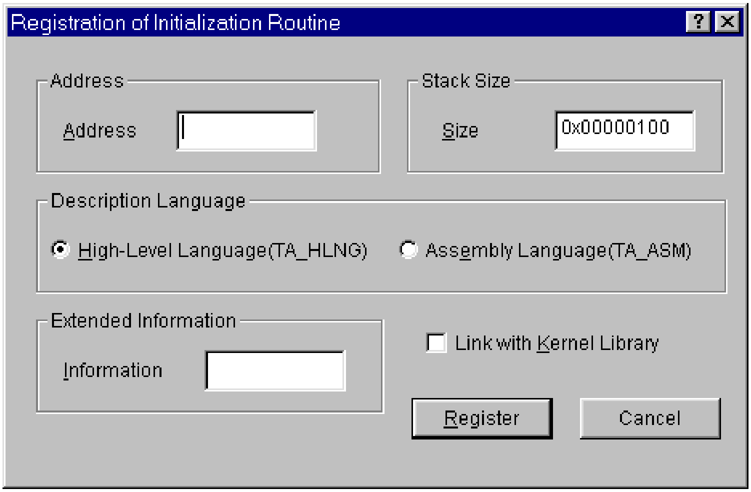


Figure 3.11 Registration of Initialization Routine Screen

Set TMU1_ini in the Address box and click the Register button, and then the Close button. Use the expression below to obtain the stack size.

- TPU2_ini stack frame size: 8 bytes
- Required size for the initialization routine: 184 + 24 bytes

Total: 216 bytes

For details about how to calculate the stack size, see Appendix C, Calculation of Work Area Size, in the HI7000/4 Series User's Manual. Use the default since the calculated stack size is smaller than it.

Figure 3.12 shows the Registration of Initialization Routine screen after registration. Figure 3.13 shows the List of Initialization Routines screen after registration.

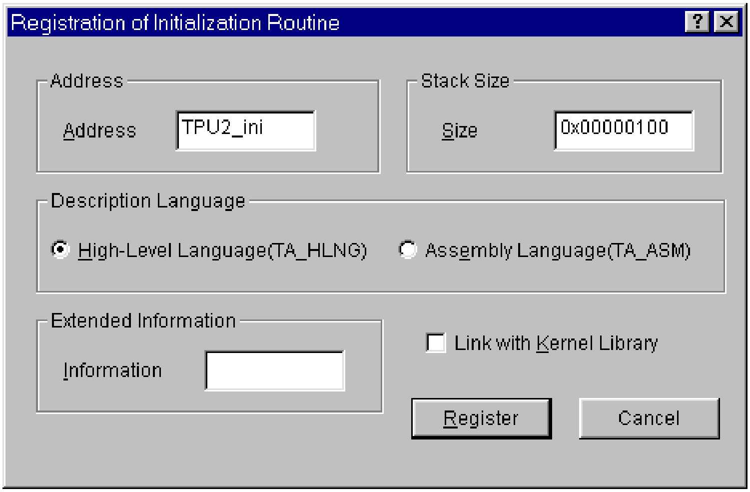


Figure 3.12 Registration of Initialization Routine Screen (after Registration)

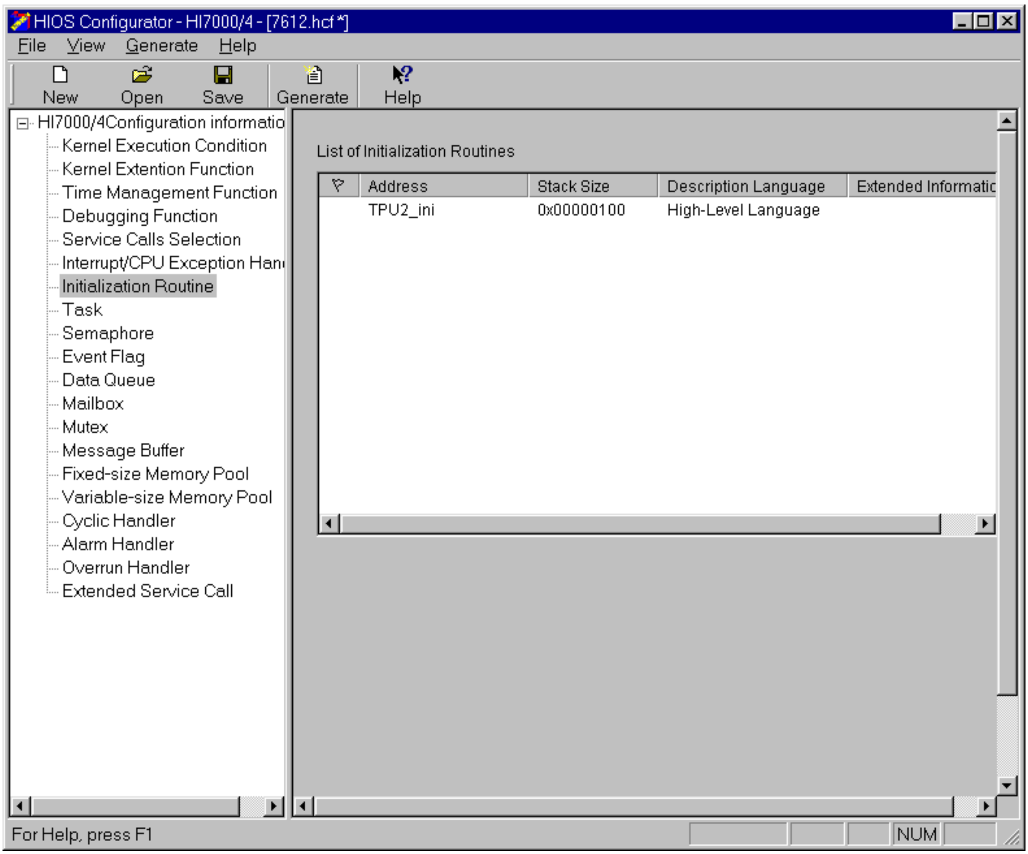


Figure 3.13 List of Initialization Routines Screen (after Registration)

3.5 Registering Event Flag Information

Click Event Flag in the HI7000/4 Configuration Information area on the Configuration Startup screen to view the Event Flag Information screen in figure 3.14.

Click the Change button in the Event Flag Information area to change the maximum event flag ID. Right click on the blank area of the Event Flag List and select Create to view the Creation of Event Flag screen in figure 3.15. For initial creation of an event flag, set the information about the event flag on this screen.

The application implemented in this guide dynamically creates one event flag in the task. Use the default event flag information.

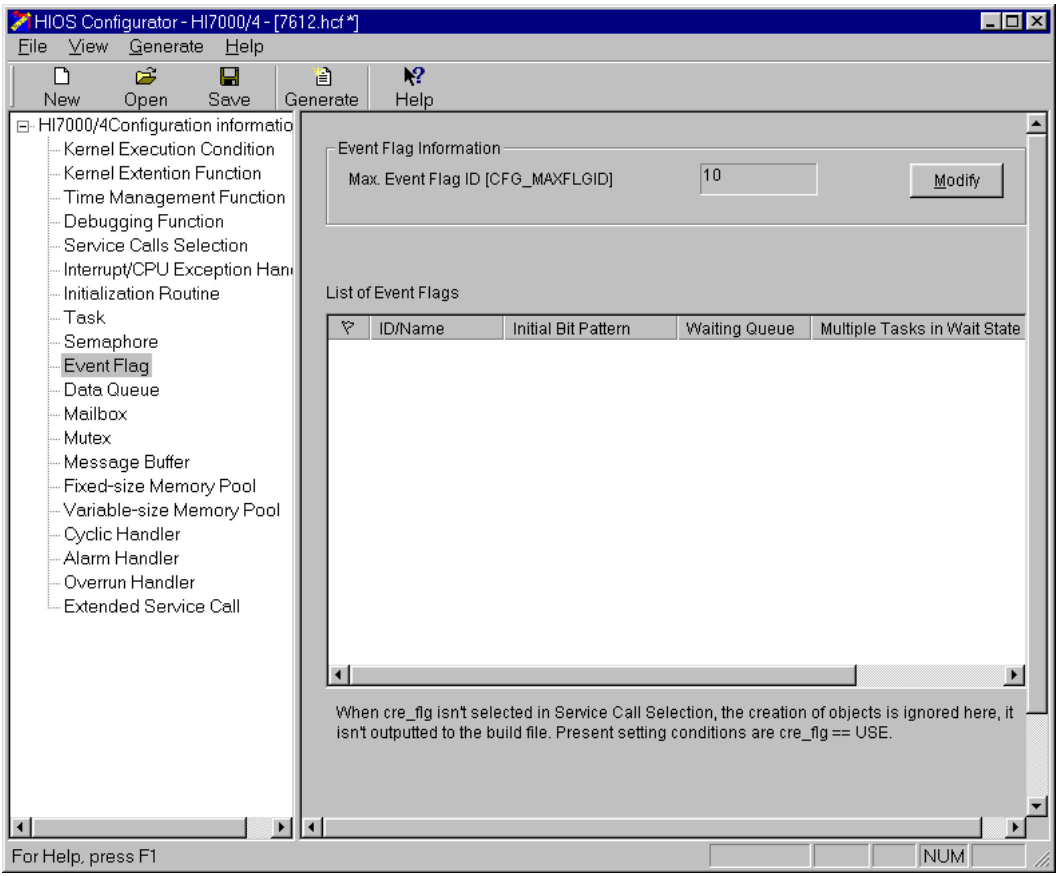


Figure 3.14 Event Flag Information Screen

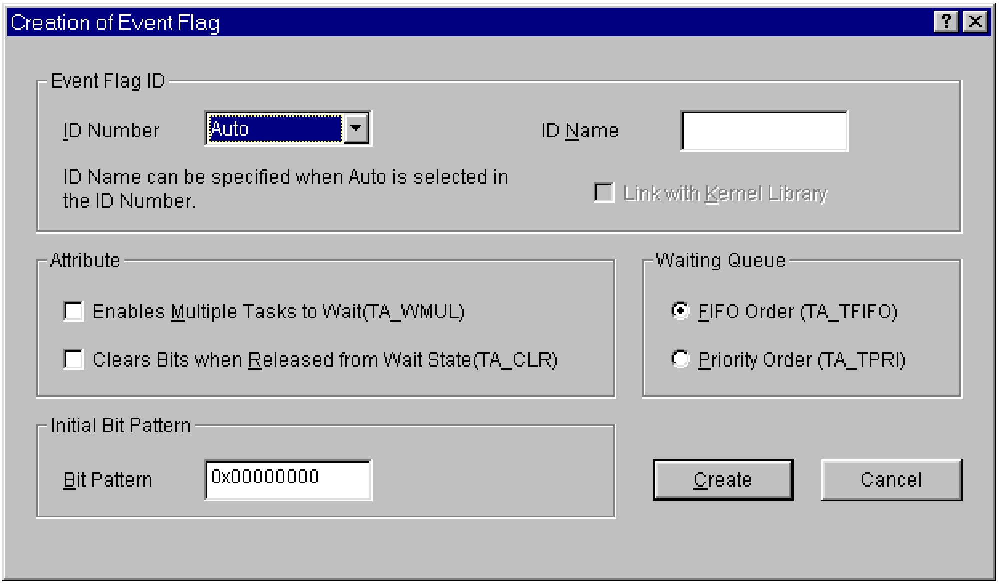


Figure 3.15 Creation of Event Flag Screen

3.6 Creating Configuration Files

Click the Create button on the Configurator Startup screen to create the configuration files required for configuring HI7000/4. For details about the configuration files, see section 5.1.2, Configurator Output File, in the HI7000/4 Series User's manual.

Now, the definition and registration by the configurator are complete. To close 7612.hcf, choose Overwrite or Save As from the File menu to save all the information.

3.7 Building the Executable File by HEW

Compile and link the files created by the configurator using HEW supplied with SHC/C++ compiler to create the executable file to be downloaded. This section describes how to build the executable file by HEW.

There are two methods to configure HI7000/4. Table 3.2 lists the type of links.

Table 3.2 Type of Links

Type	Description
Whole linkage	Links the kernel and all configuration files into a single load module (called a whole load module).
Separate linkage	Links the kernel code portion (called a kernel load module) and the kernel data portion (called a kernel environment load module) into separate load modules. Application files can be included in a kernel load module, a kernel environment load module, or in an independent application load module.

For details, see section 5, Configuration, in the HI7000/4 Series User's Manual.

This guide describes how to use the whole link method to configure the program.

3.7.1 Starting HEW

Double click hios.hws in the install folder "hios" to start HEW to build HI7000/4. Figure 3.16 shows the HEW Startup screen.

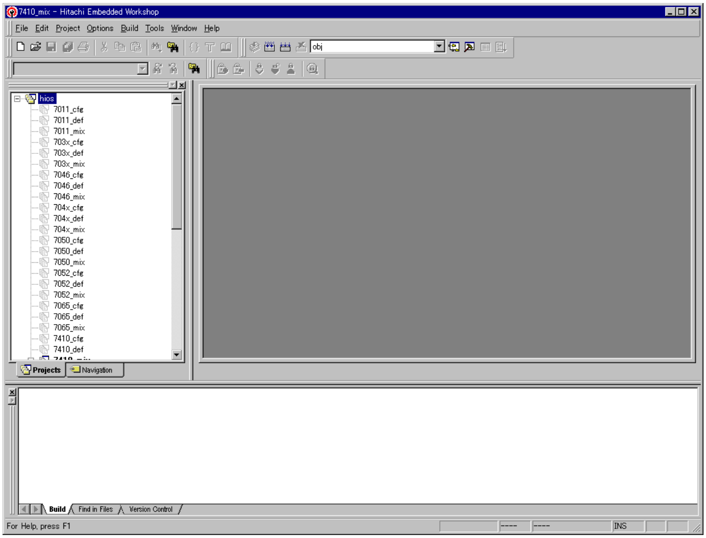


Figure 3.16 HEW Startup Screen

The standard project file hios.hws contains three sub-projects to configure the program for the target CPU. Table 3.3 lists the type of project files.

Table 3.3 Project Files

7612_mix	Project file for creating the whole load module for the whole link method
7612_cfg	Project file for creating the kernel load module for the separate link method
7612_def	Project file for creating the kernel environment load module for the separate link method

Select the project file 7612_mix for creating the whole load module.

Figure 3.17 shows the Set Current Project screen.

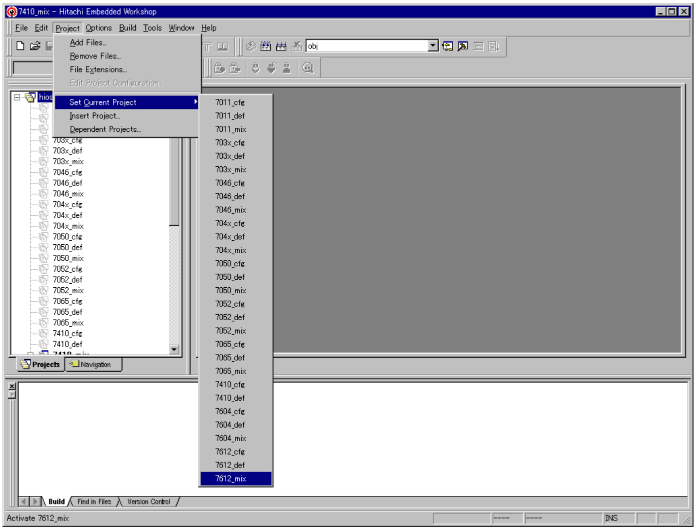


Figure 3.17 Set Current Project Screen

3.7.2 Defining a Configuration File

Define each application program created in section 2 as a project file. Use the default project file configuration and define only the timer driver to implement the sample program operation in this guide.

On the Current Project Set screen, select Add Files... from the Project menu to add tpu2.c as a project file. Figures 3.18 and 3.19 show the screen for adding a file.

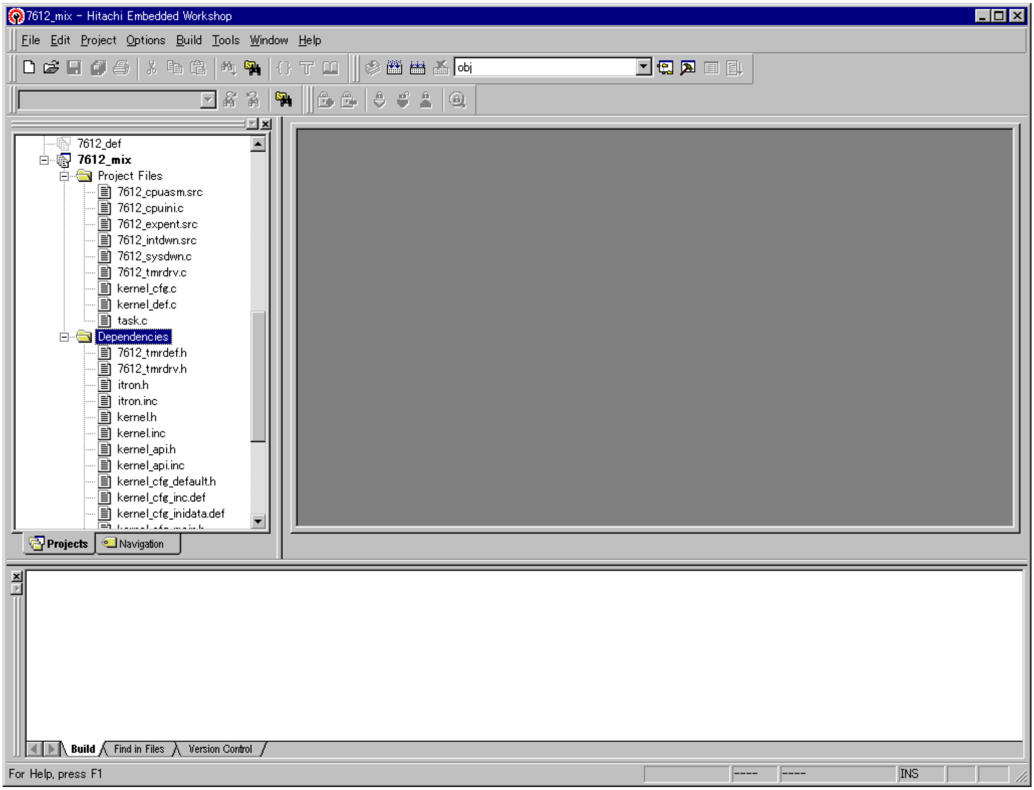


Figure 3.18 Adding a File

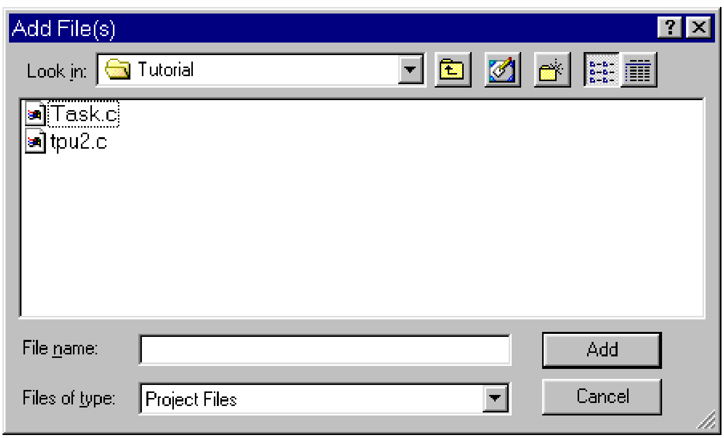


Figure 3.19 Adding a File

Now, defining the configuration files completes.

RENESAS

3.7.3 Changing a Linkage Address

Change the linkage addresses to run the programs on the Solution Engine address map.

The Solution Engine is supplied with 32-Mbyte SDRAM from 0x0600000 to 0x07FFFFFF. In this guide, 16 Mbytes from 0x0600000 to 0x06FFFFFF are used.

Select OptLinker from the Options menu to view the OptLinker Options screen (figure 3.20).

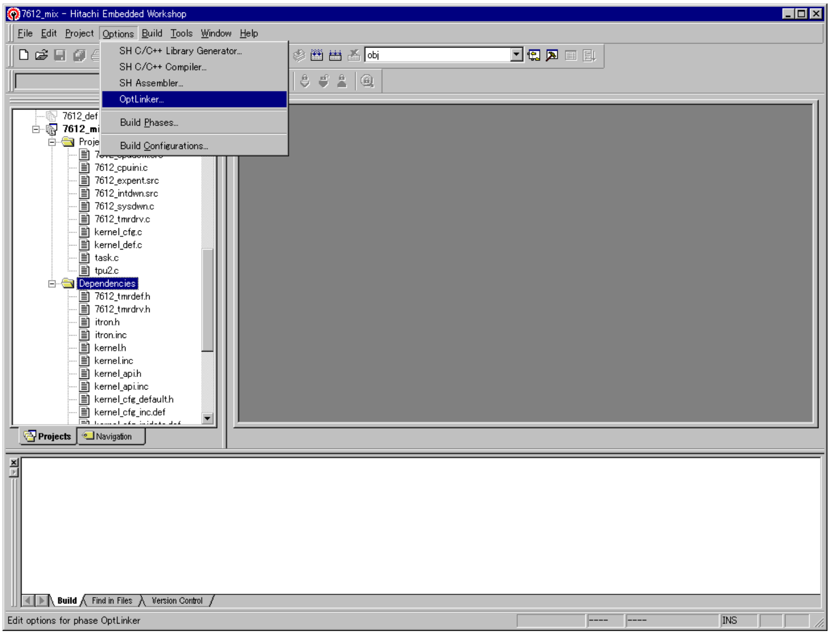


Figure 3.20 Selecting OptLinker

- Changing a section address

Click the Section tab to view the Define Section screen (figure 3.21).

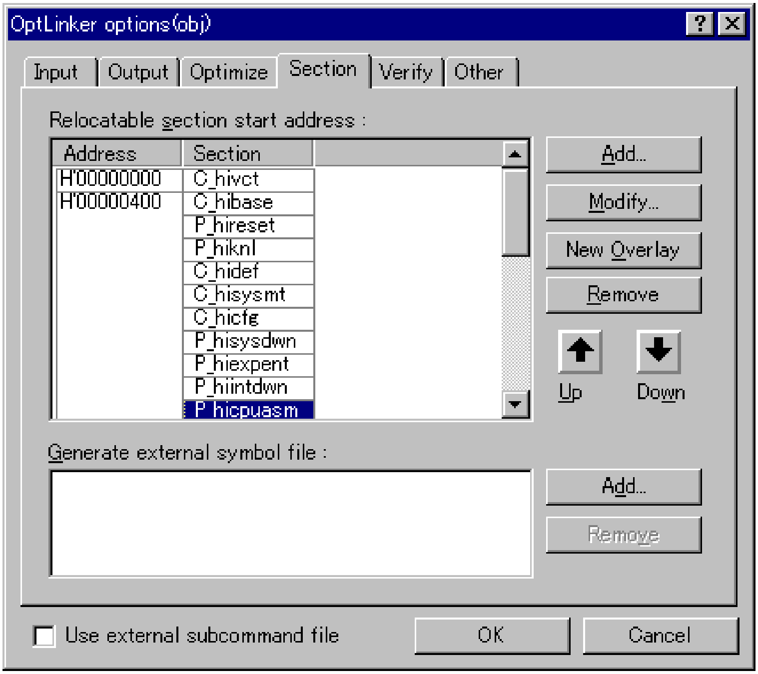


Figure 3.21 Define Section Screen

Click the section P_hicpuasm and then Up button to highlight the first section of address H′00000400 (figure 3.22).

RENESAS

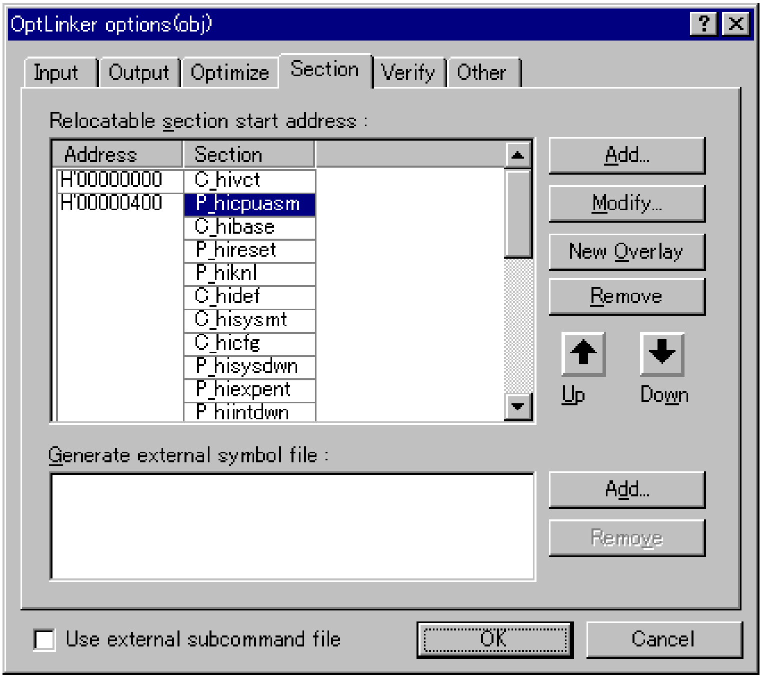


Figure 3.22 Define Section Screen

Click Address for each section to enable the Modify... button. Change the section addresses as listed in table 3.4.

RENESAS

Table 3.4 Section Addresses

Section Name	Before Change	After Change	Section Name	Before Change	After Change
C_hivct	00000000	26000000	B_hiwrk	06000000	26010000
P_hicpuasm	00000400	26000400	B_himpl		
C_hibase			B_hidystk		
P_hireset			B_histstk		
P_hiknl			B_hiirqstk		
C_hidef			B_hitrcbuf		
C_hisysmt			B_hitrceml		
C_hicfg			B		
P_hisysdwn			R		
P_hiexpent					
P_hiintdwn					
P_hicpuini					
P_hitmrdrv					
P					
C					
D					

3.7.4 Build

Execute HEW to build an executable file that can be downloaded to the Solution Engine by the E10A emulator. Select Build from the Build menu. Figure 3.23 shows the screen for selecting Build.

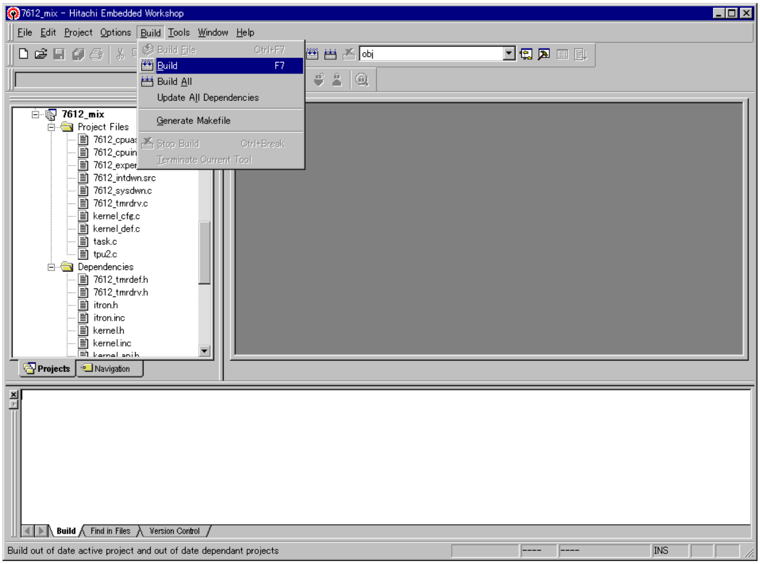


Figure 3.23 Selecting Build

The executable file is created by selecting Build. The result of compilation and linkage is shown at the bottom of the window. If a compile error occurs, correct the applicable source and build the file again. The executable file (with the file extension .abs) is created in the install folder “obj”.

Now you can download the file to the Solution Engine by the E10A emulator and execute it. For details about how to download and execute the file, see section 4, Downloading and Executing Application Programs.

RENESAS

3.8 Disabling Parameter Check Function

When debugging the application programs completes and they are ready to be installed into the product, you can disable the parameter check function. This check function is an unnecessary routine performed in the beginning of the service call, in the HI series operating system.

You can use the configurator to disable the parameter check function. Figure 3.24 shows the screen for disabling the parameter check function.

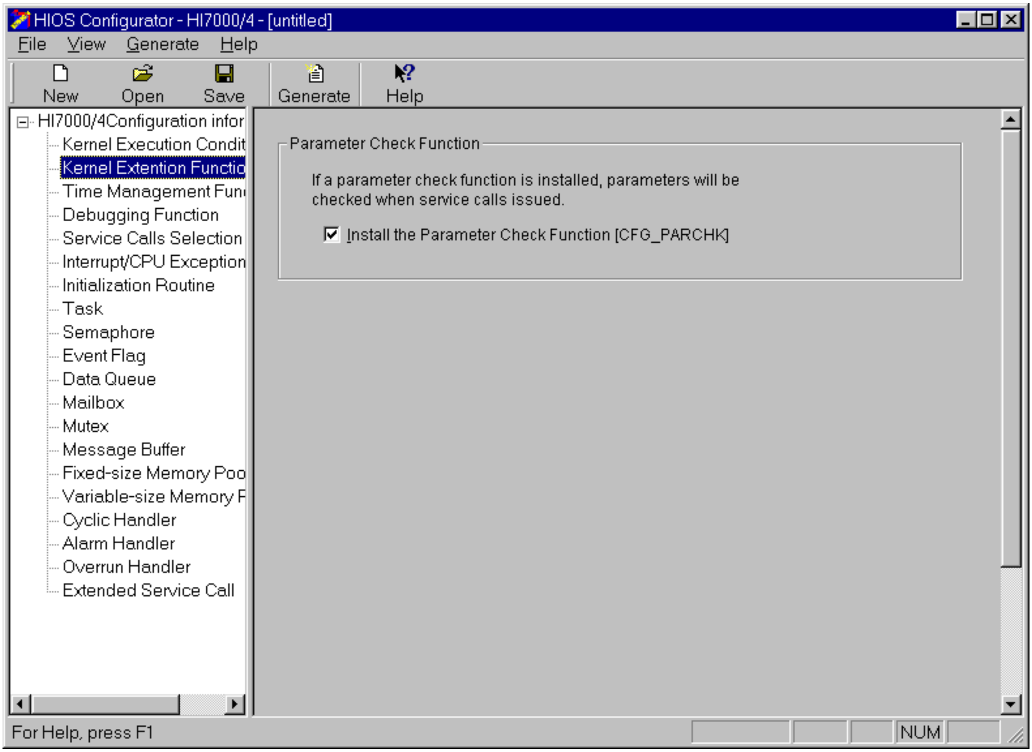


Figure 3.24 Disabling Parameter Check Function

Click Kernel Extended Function on the Configurator Startup screen to view the screen in figure 3.24. Uncheck the Install the Parameter Check Function checkbox and create and build the configuration files. The executable file with the parameter check function disabled is created.

RENESAS

Section 4 Downloading and Executing Application Programs

This section describes how to use the E10A to download the executable file created in section 3, Configuration, and run it on the Solution Engine.

4.1 Initializing Solution Engine

The ROM monitor supplied with the Solution Engine initializes the CPU. In this guide, this monitor is used for the CPU initialization. (When using another board, you must use a specific CPU initialization routine. For details of CPU initialization, see section 2.1, Creating CPU Initialization Routine.)

Configure the system as shown in figure1.1 in section 1, Overview. Start the host computer, turn the Solution Engine on, select HDI for E10A SH7612 from the Windows Start menu to start the HDI. Figure 4.1 shows the HDI Startup screen.

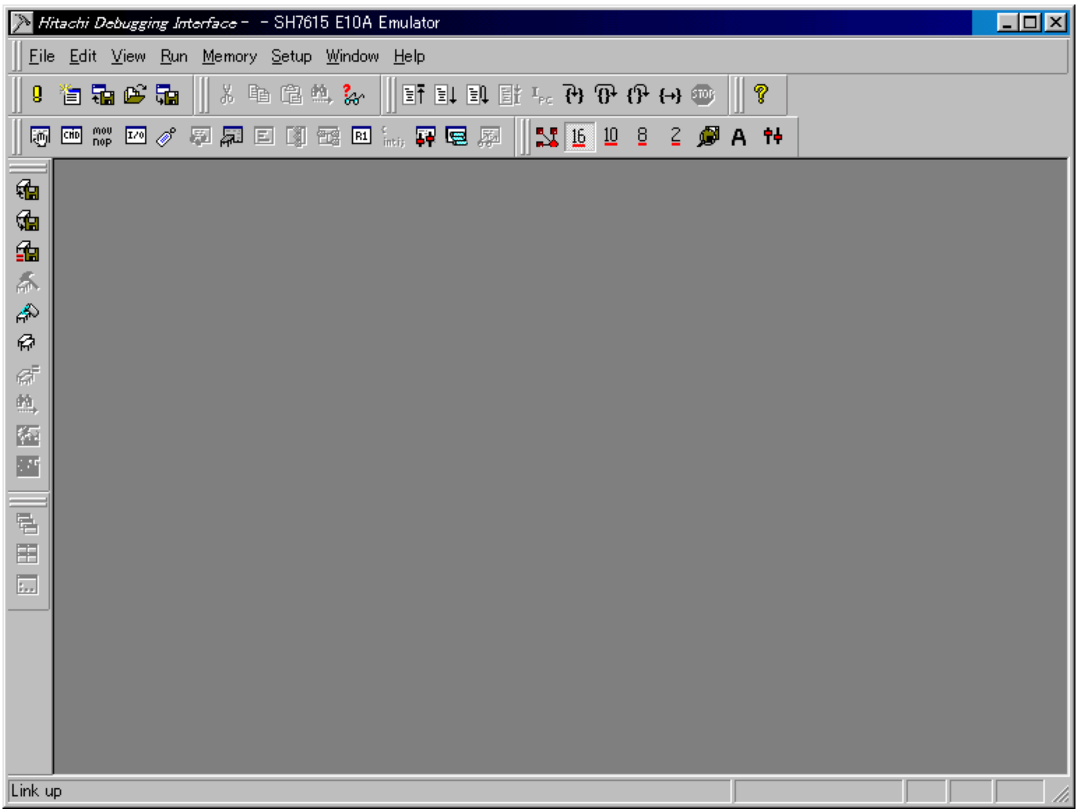


Figure 4.1 HDI Startup Screen

Then, choose Go from the Run menu (figure 4.2).

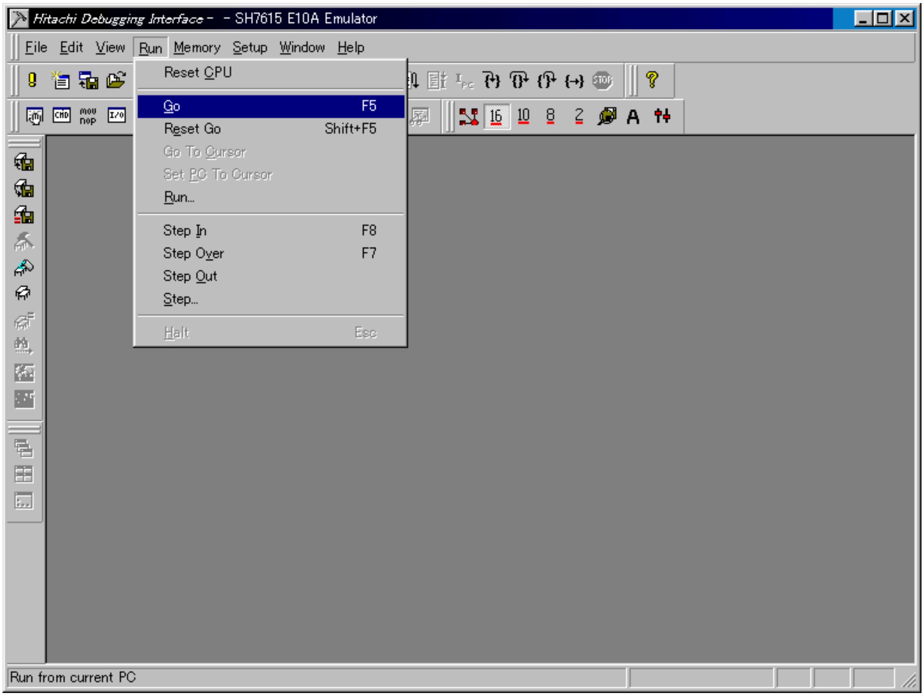


Figure 4.2 Go menu

After one or two seconds, click the STOP button (red) on the menu bar. Now, initializing the Solution Engine completes and this allows reading from or writing to the SDRAM supplied with the Solution Engine.

4.2 Downloading Application Program

Download the executable file created in section 3, Configuration, to the E10A.

Figure 4.3 shows the screen for downloading the executable file.

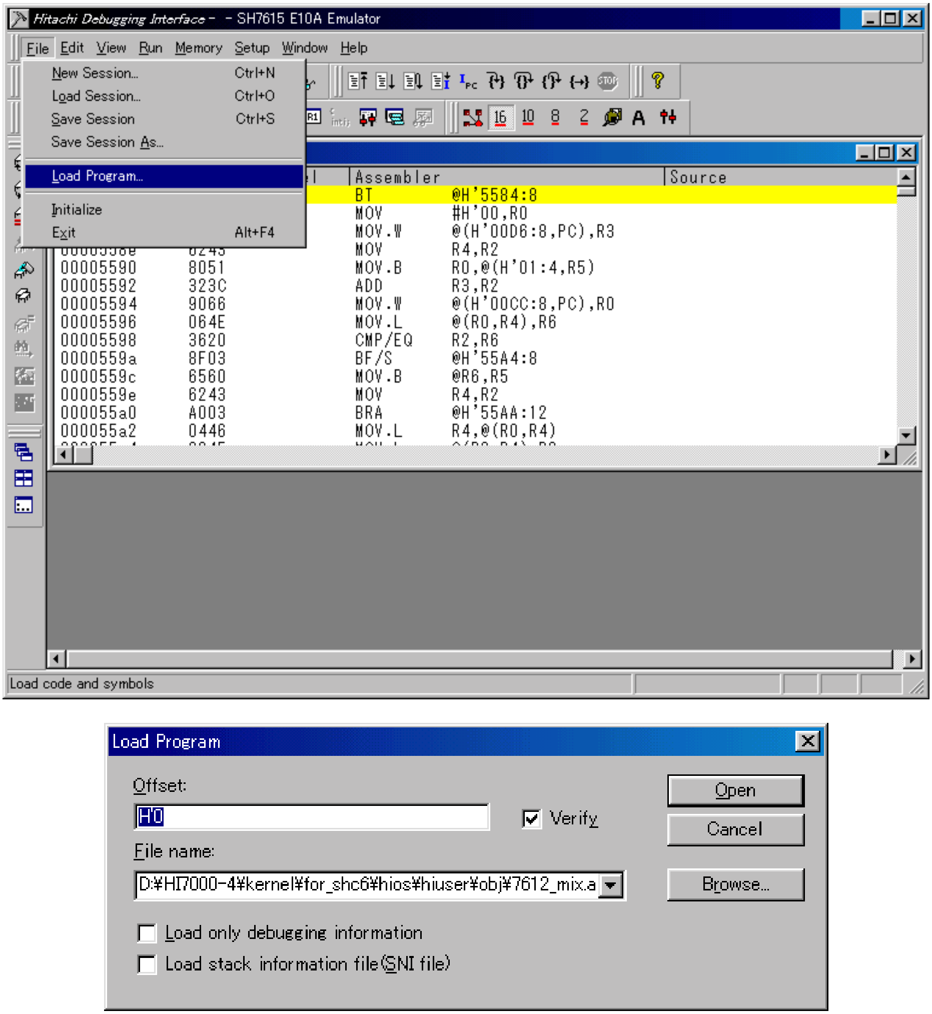


Figure 4.3 Downloading Executable File

Select Load Program... from the File menu on the HDI Startup screen. On the Load Program screen in figure 4.3, enter the name of the executable file to download in the File Name box and click the Open button to download it. The executable file is 7612_mix.abs in the install folder “obj”.

After downloading succeeds, the Complete Download screen in figure 4.4 appears.

RENESAS

Figure 4.4 Complete Download Screen

Click the OK button on the Complete Download screen.

4.3 Executing Application Program

To execute the program, choose Registers from the View menu on the HDI Startup screen to view the register information (figure 4.5).

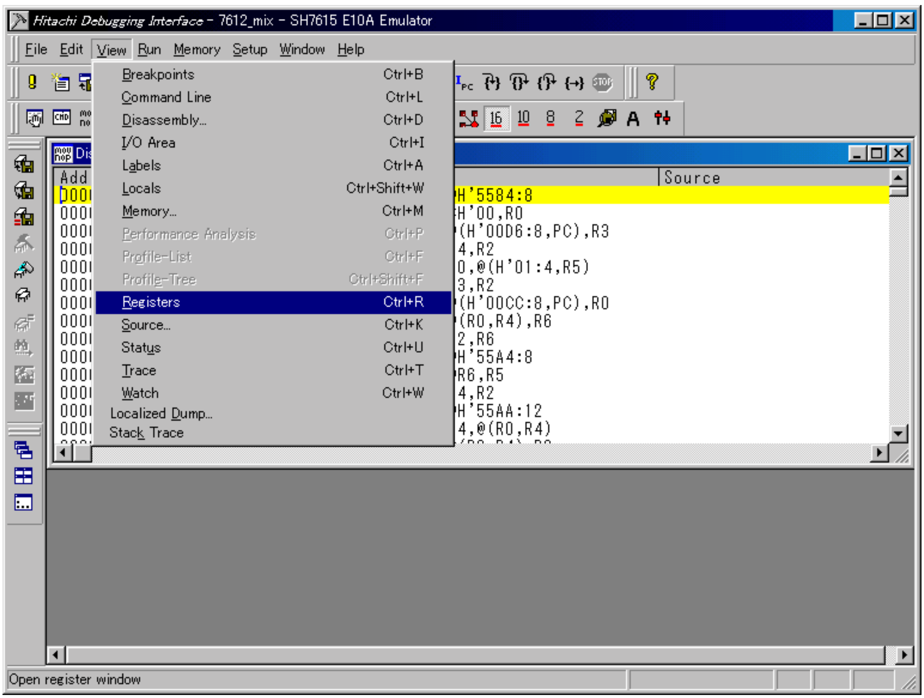


Figure 4.5 Register Information

Then, change the PC value. Double click the PC value on the Register Information screen to view the Change PC Value screen (figure 4.6).

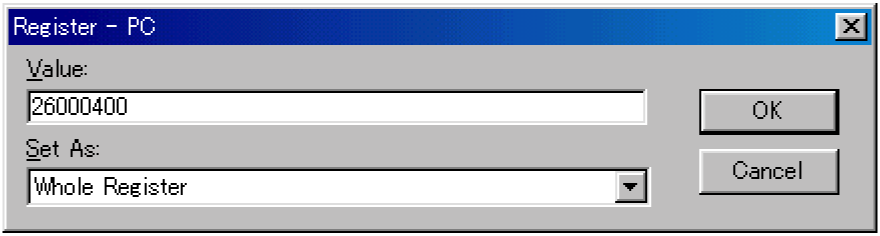


Figure 4.6   Change PC Value Screen

Change the PC value to 26000400 as shown in figure 4.6 and click the OK button. This value is the start address of the CPU initialization routine.

Now, you can execute the program. Select Go from the Run menu to execute the program (figure 4.7).

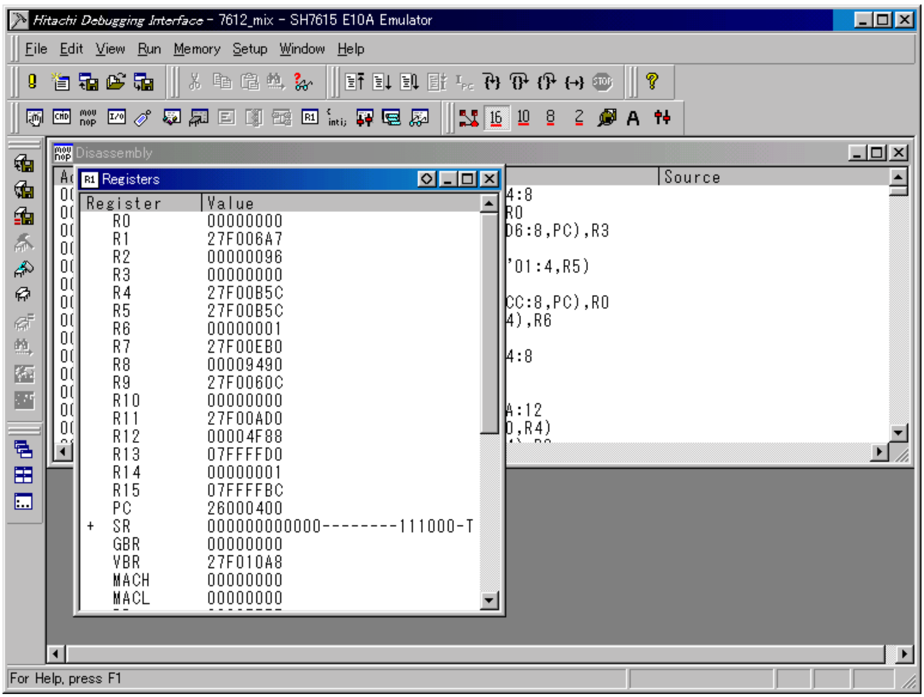


Figure 4.7   Execute Program Screen

HI7000/4 Hitachi Industrial Realtime Operating System Configuration Guide

Publication Date:	1st Edition, March 2003
Published by:	Business Operation Division
	Semiconductor & Integrated Circuits
	Hitachi, Ltd.
Edited by:	Technical Documentation Group
	Hitachi Kodaira Semiconductor Co., Ltd.