FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM71-00320-2E

FR FAMILY

IN CONFORMANCE WITH µITRON 3.0 SPECIFICATIONS

SOFTUNE REALOS/FR USER'S GUIDE

FUJITSU

FR FAMILY

IN CONFORMANCE WITH µITRON 3.0 SPECIFICATIONS

SOFTUNE REALOS/FR USER'S GUIDE

FUJITSU LIMITED

PREFACE

■ Objectives

Softune REALOS/FR (REALOS/FR) is a real-time operating system used on Fujitsu FR family 32-bit microprocessors. The specifications of the REALOS/FR kernel comply with µITRON3.0.

This manual provides information required to configure REALOS/FR application systems. The material in this manual covers the configuration and activation of systems, and can be taken as an overall summary of their operation.

This manual assumes the used of one of the following operation environments.

❍ Host computer

IBM PC/AT compatible machine (FUJITSU FMV series):

Windows 95/NT4.0

Operating environment: i486 or higher (recommended: Pentium 150MHz or more), memory (Windows 95: 16MB or greater (recommended: 32MB)/Windows NT4.0: 32MB or greater (recommended: 48MB)), hard disk 14MB or greater

❍ Cross development tools

- fasm911s: Assembler
- flnk911s: Linker
- flib911s: Librarian
- sim911: Simulator Debugger
- eml911a: Emulator Debugger (for MB2197 emulator)
- fs911s: Softune Workbench

■ Trademarks and Abbreviations

TRON is an abbreviation of The Realtime Operating System Nucleus.

ITRON is an abbreviation of Industrial TRON.

µTRON is an abbreviation of Micro Industrial TRON.

■ Intended Readership

This manual is written for engineers actually developing products for use with REALOS/FR, and describes the development of application systems using REALOS/FR as well as debugging methods. Be sure to read the entire manual carefully.

■ Configuration of This Manual

This manual consists of two chapters and an appendix.

Chapter1 Precautionary Information

This chapter describes precautionary information relating to development of applications systems using REALOS/FR.

Chapter 2 Developing Application Systems

This chapter describes methods for developing application systems using REALOS/FR.

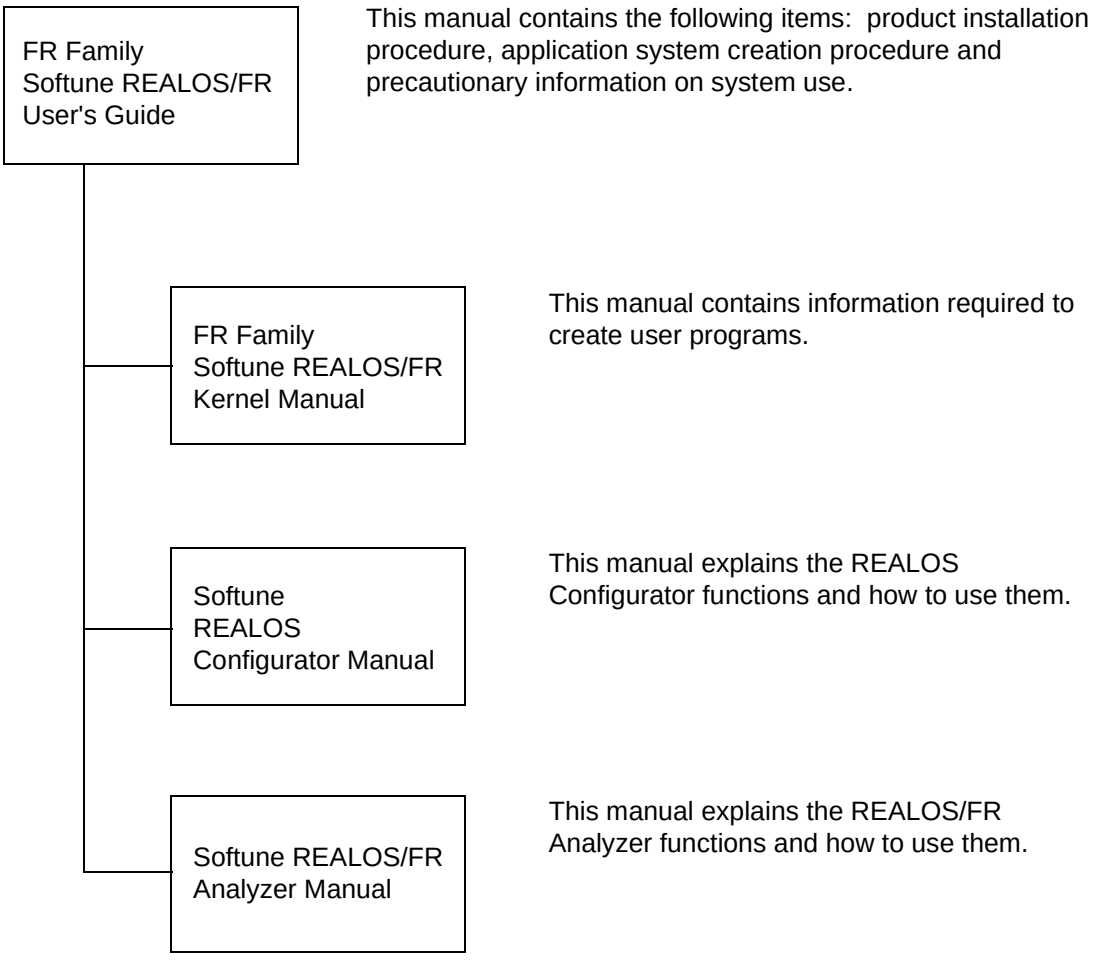
Appendix

The appendix describes sample systems, methods for creating sample I/O drivers, and methods for startup.

1. The contents of this document are subject to change without notice. Customers are advised to consult with FUJITSU sales representatives before ordering.
2. The information and circuit diagrams in this document are presented as examples of semiconductor device applications, and are not intended to be incorporated in devices for actual use. Also, FUJITSU is unable to assume responsibility for infringement of any patent rights or other rights of third parties arising from the use of this information or circuit diagrams.
3. The contents of this document may not be reproduced or copied without the permission of FUJITSU LIMITED.
4. FUJITSU semiconductor devices are intended for use in standard applications (computers, office automation and other office equipments, industrial, communications, and measurement equipments, personal or household devices, etc.).
   CAUTION:
   Customers considering the use of our products in special applications where failure or abnormal operation may directly affect human lives or cause physical injury or property damage, or where extremely high levels of reliability are demanded (such as aerospace systems, atomic energy controls, sea floor repeaters, vehicle operating controls, medical devices for life support, etc.) are requested to consult with FUJITSU sales representatives before such use. The company will not be responsible for damages arising from such use without prior approval.
5. Any semiconductor devices have inherently a certain rate of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
6. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Control Law of Japan, the prior authorization by Japanese government should be required for export of those products from Japan.

Softune REALOS/FR Manual Set

The REALOS/FR manual set consists of four volumes listed below. A first-time user of REALOS/FR should first read the *FR Family Softune REALOS/FR User's Guide*.



READING THIS MANUAL

■ Page Layout

In this manual, an entire section is presented on a single page or spread whenever possible. The reader can thus view a section without having to flip pages.

The content of each section is summarized immediately below the title.
You can obtain a rough overview of this product by reading through these summaries.

Also, higher level section headings are given in lower sections so that you can know to which section the text you are currently reading belongs without referring to the table of contents or the cover of each chapter.

■ Product Names

Product names in this manual are abbreviated as follows:

Microsoft® Windows® 95 operating system: Windows 95

Microsoft® Windows NT® Workstation operating system Version 4.0: Windows NT 4.0

■ Coding Formats

The following table explains the symbols used in this manual for definition statement and other coding formats.

Symbol	Explanation
[ ]	Items enclosed in brackets are optional and may be omitted.
{ }	Items enclosed in braces are alternatives: one must be selected.
|	A vertical bar separates elements in a coding fragment.
. . .	Ellipsis points indicate that the immediately preceding item can be repeated indefinitely.
Note:	This symbol indicates an item requiring special attention. Be sure to read the information provided.

MEMO

CONTENTS

FIGURES

TABLES

MEMO

CHAPTER 1 PRECAUTIONARY INFORMATION

This chapter describes precautionary information relating to development of applications systems using REALOS/FR.

1.1 Precautions for Programming on REALOS/FR Systems

This section describes precautionary information for programming on REALOS/FR systems.

■ Precautions for Programming

Notes:

- The ext_tsk system call cannot be issued from the task independent portion.

  A system call under these conditions will result in an error and the program may not revert to the previous context.

- The ret_int system call cannot be issued from the task portion.

  A system call under these conditions will result in an error and the program may not revert to the previous context.

- The ret_tmr system call cannot be issued from the task portion.

  A system call under these conditions will result in an error and the program may not revert to the previous context.

- The contents of the following registers and memory cannot be directly altered by user programs:

  PS, TBR, SSP, USP registers, bit search module control register, delay interrupt control register, interrupt control register, interrupt vector table, kernel data area.

  These resources are controlled by the kernel.

  Note in particular that setting the PS register I bit to “0” (interrupt disabled) prohibits task switching.

- The EIT vector table is created by the configurator at the time the system is configured, and therefore need not be created by the user.

- At the beginning of the reset entry routine, be sure to set the system stack pointer (R_sstack) created by the Configurator in the SP register and the label (R_vct) for the vector table in the tbr register. Also specify the interrupt mask (strongest) for the ilm register.

- Be sure to jump to the kernel initialization routine (R_init) at the end of the reset entry routine.

  Since the kernel initialization routine uses the stack area of a task, do not initialize the stack area of a task after the sequence jumps to R_init.

- Be sure to use the entry name _uinit to create a kernel initialization routine.

  The user initialization routine (_uinit) is called by the kernel initialization routine as a subroutine.

  Issue only those system calls inside the user initialization routine (_uinit) that can be issued from the task-independent portions.

- If you describe a timer interrupt handler for the system clock, be sure to save each register when the handler starts, and restore each register when the handler ends. Also, be sure to make a R_sys_clock subroutine call in the handler.

- When creating the system down routine, be sure to use the entry name _system_down.

  The _system_down entry is called if the system fails.

- Be sure to specify in RELOCATE_FILE in the configuration file the OS object file that corresponds to the tool being used.

- A task that executes object wait system calls with a time-out must always be specified as TMO in the TSK definition of the Configurator.
- In case of using an idle task in user program, it is necessary to create the idle task by yourself. Define as the idle task a task which has the lowest order of priority and which never makes a transition to the wait state.
- The method of message transmission to and from mailboxes is by means of pointers for message sending, and therefore sending and receiving messages must be paired one-to-one.

  Also, once a message is sent the header area of that message remains in use, and therefore access to header areas of sent messages is prohibited.
- In case of issuing rel_blf with the already released memory block,the OS become unusual movement.
- Because free memory blocks in a variable-size/fixed-size memorypool are managed by using that memory block area, memory block areas must not be accessed after they are returned.
- In processing of system calls having address values as parameters, the address values are not subject to boundary checking.
- Symbol names beginning with ”R_” are reserved by REALOS/FR, and therefore may not be used in user programs.
- The ret_int() function is macro-defined in realos.h. Be sure to include realos.h in any system calls written in C.
- The ret_tmr() function is macro-defined in realos.h. Be sure to include realos.h in any system calls written in C.
- The DMA transfer completion interrupt demand is delayed in the same way as other interrupt when it occurred int the interupt mask section inside OS.
- Directory names containing Japanese characters cannot be used.
- File names containing Japanese characters cannot be used.

1.2 Precautions for REALOS/FR Hardware Design

This section describes precautionary information for REALOS/FR hardware design.

■ Precautions for Hardware Design

Notes:

- The REALOS/FR operating system operates only on FR family microcontrollers with built-in delay interrupt module and bit search module.
- REALOS/FR requires one interval timer generating a timer interrupt signal at constant intervals (typically 1 ms) for use as a system clock.

  It is also possible to use the FR built-in timer.
- System configuration is possible even without a system clock. However the following system calls are related to timing, and will not operate properly.

  tslp_tsk , twai_sem , twai_flg , trcv_msg , tget_blf, set_tim , get_tim , dly_tsk , def_cyc , def_alm
- The delay interrupt module, bit search module, system clock interval timer and vector 64 EIT are used by REALOS/FR and are not available to the user.

CHAPTER 2 DEVELOPING APPLICATION SYSTEMS

This chapter describes methods for developing application systems using REALOS/FR.

2.1 Products Supplied

A REALOS/FR product consists of the REALOS/FR program on its media plus a set of manuals.

■ Products Supplied

Figure 2.1-1 shows the items included in one REALOS/FR product.

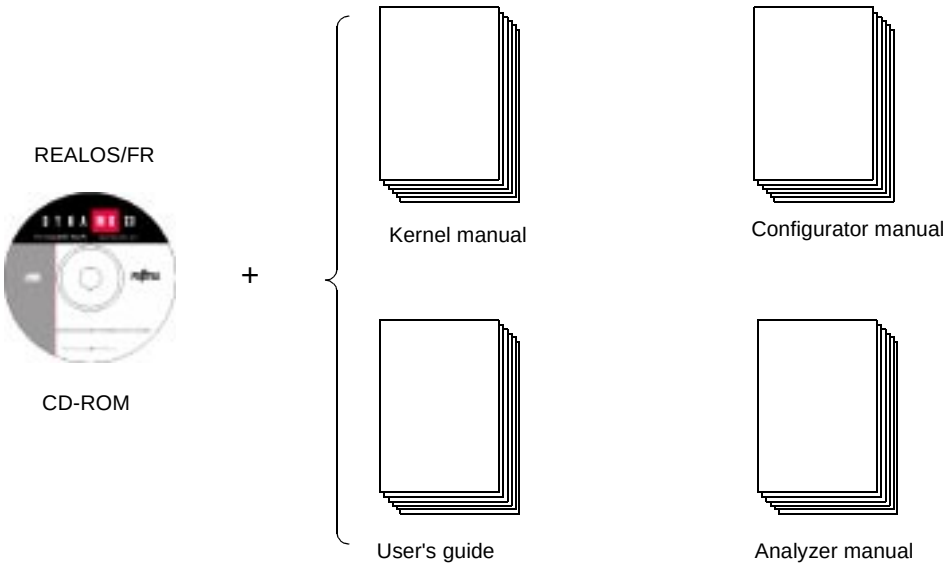


Figure 2.1-1 Products Supplied with REALOS/FR

■ Product Manuals

Table 2.1-1 describes the manuals provided with REALOS/FR products.

Table 2.1-1 REALOS/FR Provided with REALOS/FR Products

Manual	Contents
Kernel manual	Describes items essential for creation of user programs. For reference when developing tasks or handlers.
Configurator manual	Describes the functions and use of the REALOS configurator. For reference when configuring applications systems.
User's guide	This is the manual you are reading now. It describes procedures for creating application systems and precautionary information relating to systems as a whole.
Analyzer manual	REALOS/FR analyzer is a tool that analyzes and displays the state of a task and an object, the transition state of a task, and the use of stack by each task. This manual describes the functions of REALOS/FR analyzer and how to use them.

2.2 Directory Configuration

This section describes the directory configuration of REALOS/FR.

■ Directory Configuration

Figure 2.2-1 show the directory configuration after installation.

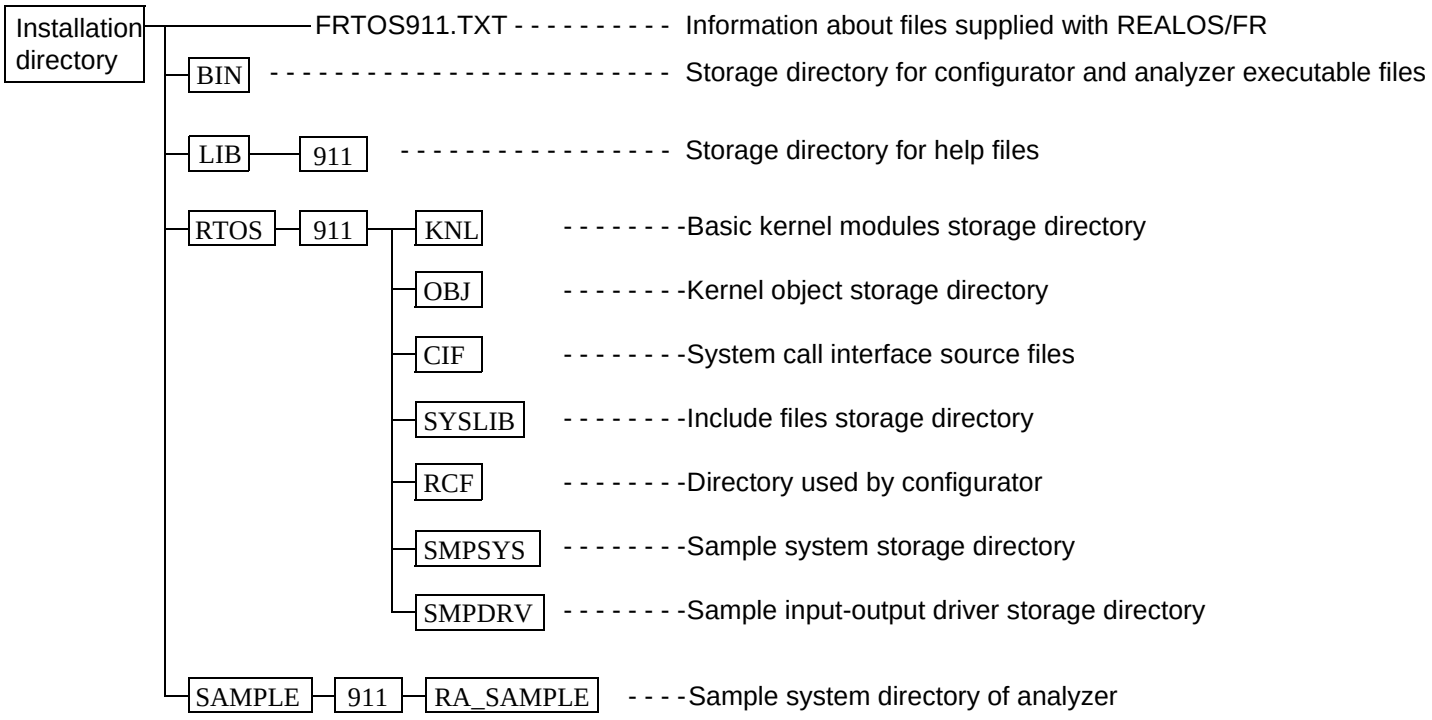


Figure 2.2-1 REALOS/FR Directory Configuration

2.3 Creating the Development Environment

REALOS/FR application programs are developed using cross development tools and an emulator. In order to develop applications programs, it is first necessary to install REALOS/FR and the following cross development tools on the host computer.

■ Support Tools Required for Development

Development of applications programs for REALOS/FR requires the cross development tools and emulator shown in Table 2.3-1.

Table 2.3-1 Support Tools Required for Development

Cross development tools (manufactured by FUJITSU)	C compiler Assembler Linkage kit Softune Workbench (fs911s)
Emulator (manufactured by FUJITSU)	MB2197 emulator

2.4 Installing REALOS/FR

REALOS/FR must be installed on the host computer before REALOS/FR application programs can be developed.

■ Installation procedure

The REALOS/FR installation procedure is explained below:

1) Activate Windows.
2) Insert the CD-ROM. Insert the Softune REALOS/FR CD-ROM into the CD-ROM drive.
3) Activate the installer.

Notes:

- Directory names containing Japanese characters cannot be used.

2.5 Programs in the Configuration of Application Systems

The configuration of an application system using REALOS/FR includes programs for normal processing, programs for processing abnormal situations and programs for expanding system processing.

■ Configuration of an Application Program

Figure 2.5-1 shows the configuration of an application program.

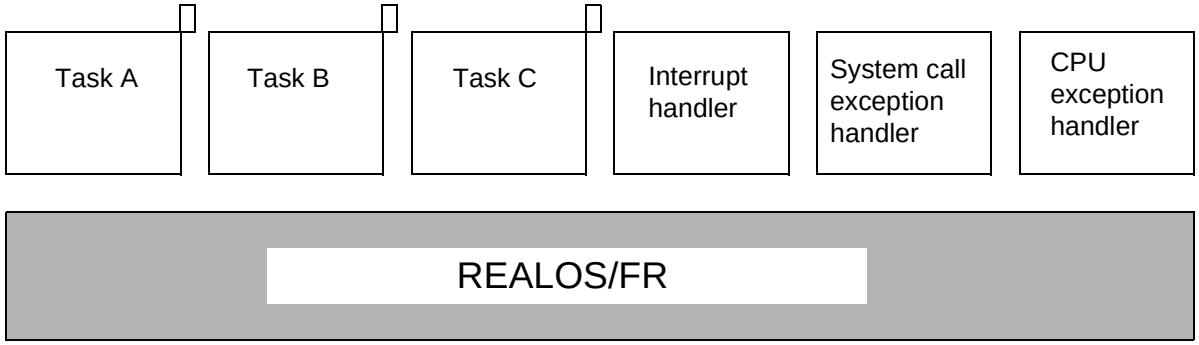


Figure 2.5-1 Configuration of an Application Program

■ Programs for Normal Processing

❍ Tasks

The task is the most fundamental unit in the configuration of an application program. Application system processing is achieved through the coordinated operation of multiple individual tasks.

❍ Interrupt Handlers

An interrupt handler is a program that is executed when an interrupt condition occurs. It is used to receive a processing request from a peripheral device and to notify the appropriate task for processing.

❍ Timer Handlers

This type of program includes cyclic handlers which are activated at regular intervals, and alarm handlers which are activated at designated times. Both function as components of the system clock handler.

Timer handlers form one component of the interrupt handlers.

■ Programs for Processing Abnormal Situations

❍ Exception Handlers

Exception handlers are programs that are activated when certain abnormal conditions occur during the execution of tasks or interrupt handler processing. They are executed following the occurrence of system call exceptions and CPU exceptions.

2.6 Development Tasks

The development of application systems using REALOS/FR requires the execution of tasks shown in Table 2.6-1.
Figure 2.6-1 shows the sequence of development tasks using REALOS/FR.

■ Development Tasks

The following table lists tasks necessary for development, along with the required software.

Table 2.6-1REALOS/FR Development Phases and Software Requirements

Development task	Overview	Software used
System design	Planning the type of system to be developed. Hardware requirements, including the use or no use of REALOS/FR, are determined at this level.	—
Target hardware selection	Determining what hardware the application system will run on. New hardware is developed if needed.	—
Setting up the development environment	The REALOS/FR development environment is configured on the host computer.	—
Program development/debugging	Designing and developing task programs and handler programs based on specifications determined in the system design stage. Each task and handler is debugged individually.	Language processors including C compiler and assembler, simulator debugger
System configuration	Configuring individually created task and handler programs into a REALOS/FR application program.	Configurator linkage kit
Application system startup/debugging	Activating and debugging the configured REALOS/FR application system on the target hardware.	Emulator debugger Analyzer
ROM installation	Using ROM installation tools to load the REALOS/FR application into microprocessor ROM.	Load module converter
Operation	Managing and operating the REALOS/FR application system.	—

■ Development Sequence

Figure 2.6-1 shows the sequence of REALOS/FR development.

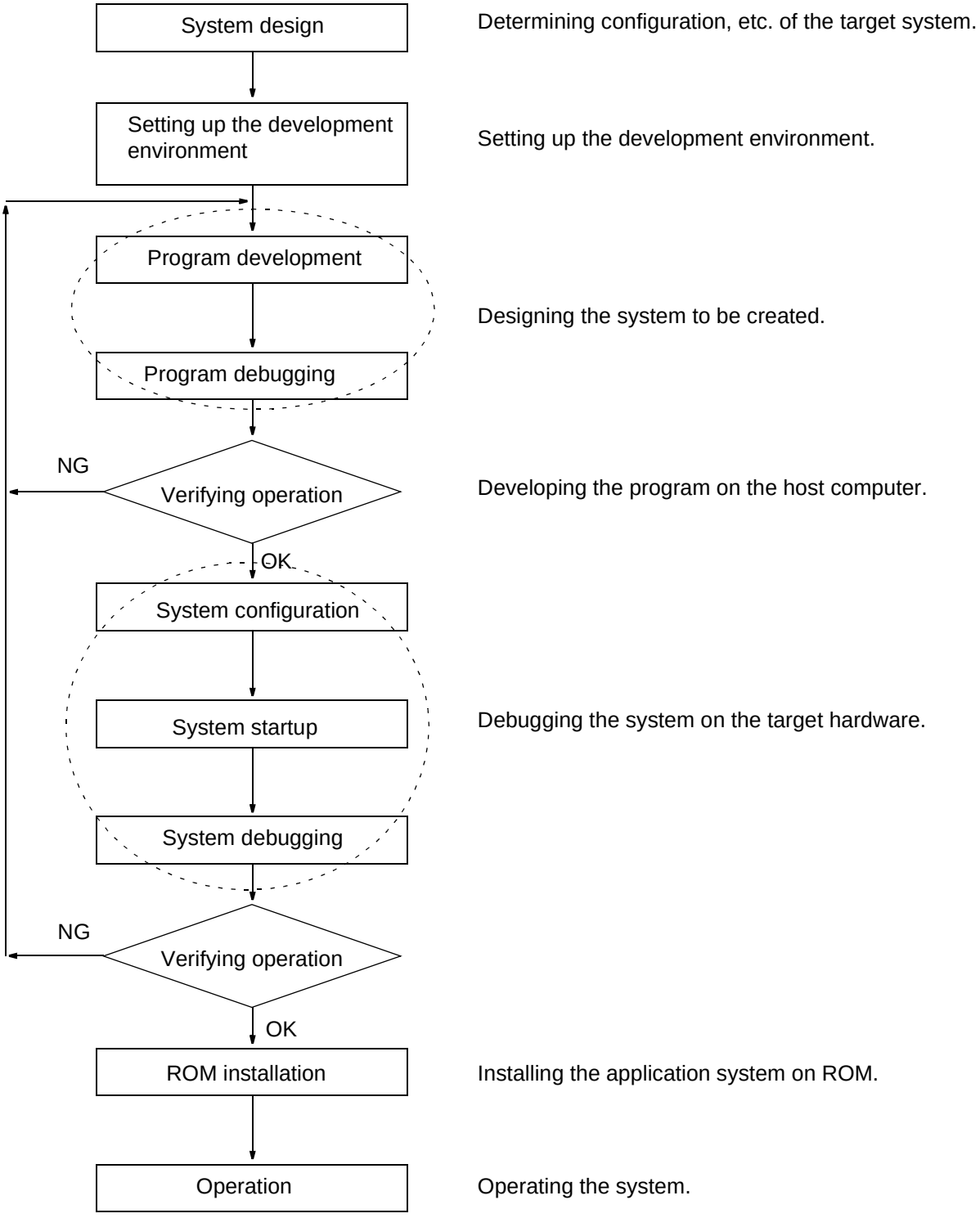


Figure 2.6-1 REALOS/FR Development Sequence

2.7 User Program Development

The system development stage determines three factors:

- **The processing to be achieved**
- **Target hardware**
- **Utilization of REALOS/FR**

Once these three have been determined the actual process of developing the REALOS/FR application system begins. This section describes the process of application system development.

■ Assign Functions to Tasks and Handlers in the User Program

REALOS/FR is a multitask realtime operating system, in which groups of processing functions are assigned to specific tasks and handlers for processing. The individual task and handler programs are designed after the overall configuration is determined, and the system allows the designer to specify methods of synchronization and communication among the task and handler routines.

This process requires an understanding of the functions provided by REALOS/FR.
For details, refer to the *Softune REALOS/FR Kernel Manual*.

- Tasks
- Interrupt handlers
- Exception handlers
- Timer handlers (cyclic handlers, alarm handlers)

■ Create Individual Task and Handler User Programs for Each Function

Individual task and handler programs are created according to the determined functions and configuration of the application. Programs may be written either in C or in assembly language.

■ Determine Objects to be Used

The next step is to determine the objects to be used in common by tasks and handlers. This decision includes the types, numbers, and initial values of each object. Object definitions are entered through the configurator.

For details, refer to the *Softune REALOS Configurator Manual*.

- Semaphore
- Event flags
- Mailboxes
- Variable-size memorypools
- Fixed-size memorypools

Notes:

- The contents of the registers and memory areas listed below must not be changed directly by application programs.
  - PS, TBR, SSP, USP, bit search module control register, delay interrupt control register, interrupt control register, interrupt vector table, and kernel data area. These areas are managed by the kernel.
  - Note in particular that setting the PS register I bit to 0 (interrupt disabled) disables task switching.
- The delay interrupt module, bit search module, system clock interval timer, and vector 64 EIT are used by REALOS/FR; they cannot be used by the user.
- Since the EIT vector table is created by the configurator during system configuration, the user need not do this.
- Because symbol names starting with “R_” are reserved by REALOS/FR, they must not be used in application programs.

2.8 Developing User-Created Modules

The reset entry routine is a user-created module. The user can use the init.asm source files provided as samples to create this module for the user system.
When running, REALOS/FR requires a timer device that can generate interrupt signals at a standard interval of 1 millisecond for use as a system clock. The user should also refer to the supplied files (init.asm) to create timer device initialization routines and interrupt handlers.
The following lists user-created modules:

- **Reset entry routine**
- **User initialization routine**
- **System clock timer interrupt handler**
- **System down routine**

■ Reset Entry Routine (_sys_entry)

The reset entry routine is activated by a reset signal and initializes the processor as well as any peripheral devices requiring initialization after reset.

The entry name of the reset entry routine should be specified using a configurator reset entry definition statement in the system configuration (refer to the *Softune REALOS Configurator Manual* for details).

In a reset entry routine, a system call cannot be issued to the REALOS/FR system. Note that proper system operation cannot be assured if a system call is issued in this way.

A reset entry routine executes the following processing:

❍ Processor initialization

- Sets the stack pointer (SP) (by setting the global symbol R_sstack).
- Sets the table base register (TBR) (by setting the global symbol R_vct).
- For methods used to set the bus interface, clock generator, operating mode, cache etc. see the appropriate *MB911XX Hardware Manual*.

❍ Peripheral device initialization

Initializes all peripheral resources and on-board devices requiring initialization following a reset.

❍ Transfer control to REALOS/FR

Following initialization of the processor and peripheral devices, control is passed to the REALOS/FR system. This is done by jumping directly to the REALOS/FR kernel data initialization routine. The kernel data initialization routine uses the R_init label which is defined and declared globally.

■ User Initialization Routines (_uinit)

User initialization routines called by the kernel act to initialize devices that the kernel uses such as timer devices for system clock timing, as well as devices used by user programs such as input/output devices.

A user initialization routine allows the issue of system calls from task independent areas of REALOS/FR.

REALOS/FR user initialization routines execute the following processes:

❍ Interrupt controller initialization

Initializes the interrupt controller. Initial values for the interrupt controller should be determined according to the configuration of the user's target hardware.

❍ Timer device initialization

Allocates one timer device for the system clock, and makes an initial setting to enable that timer device to generate an interval timer interrupt signal at a standard interval of 1 ms.

❍ User device initialization

Devices utilizing user programs should be initialized. As an alternative, initial settings may be made by software rather than only to hardware.

■ System Clock Timer Interrupt Handler (_timer)

The timer handler used for the system clock is the interrupt handler of the timer device allocated for the system clock that is used by REALOS/FR. The handler clears interrupt factors and performs other functions required for timer device interrupt processing, and initiates subroutine calls for REALOS/FR system clock processing. System clock processing is defined on the R_sys_clock label and declared globally.

This handler should be defined by the configurator at the time the system is constructed, as the interrupt handler having the corresponding interrupt number.

■ System Down Routine (_system_down)

The system down routine is called in the following cases:

- An undefined interrupt occurs.
- A CPU exception occurs when no exception handler has been defined.
- The kernel cannot continue processing because of an abnormal condition.

  _system_down must be specified as the entry name of the system down routine.

Notes:

- In the first step of a reset entry routine, a system stack pointer (R_sstack) created by the configurator must always be set in the SP register. Also, the vector table symbol (R_vct) must always be set in the tbr register. An interrupt mask (highest priority) must also be set for the ilm register.
- In the final step of a reset entry routine, the processing must always be jumped to a kernel initialization routine (R_init).
- A kernel initialization routine uses a task stack area, and so the task stack area must not be initialized after jump to the initialization routine.

- A user initialization routine must always be created using entry name _uinit.
  - A user initialization routine (_uinit) is called from a kernel initialization routine as a subroutine.
  - System calls can be issued only from the task independent portion in a user initialization routine (_uinit).
- A system down routine must always be created using the entry name _system_down.
  - If a system down occurs, processing jumps from the OS to the _system_down entry.
- When specifying a system clock timer interrupt handler, each register must be saved and restored at the top and end of the handler. The R_sys_clock must also be called in the handler as a subroutine.

2.9 System Configuration and Startup

The configurator is used to configure a system by linking the user program with the kernel module.
The system thus generated in absolute format is downloaded into the target system, and then started up. After kernel initialization processing, the user tasks are activated.

System Configuration

The configurator is used to configure a system.

Configuration processing takes place after all the necessary settings are made. Kernel initialization data and EIT vector tables are generated in accordance with the settings that have been entered by the user. Then the REALOS/FR kernel module and user program are linked to form an object program of a REALOS/FR application system in absolute format.

For details, see the *Softune REALOS Configurator Manual*.

System Startup

After the object program in absolute format is generated by the configurator, it is downloaded to the target system, a reset signal is applied, and the REALOS/FR application system is activated.

APPENDIX

The appendix describes programs supplied with REALOS/FR, including memory capacity requirements and typical target systems.

Appendix A Program Descriptions

This appendix describes 12 programs supplied with REALOS/FR.

- **Information about supplied files**
- **Configurators**
- **Kernel base module**
- **User-created module**
- **System call interface source file**
- **System call interface library**
- **User include files**
- **Kernel include files**
- **OS object files**
- **Sample system**
- **Sample drivers**
- **Analyzer**

■ Information about supplied files

The following file provides information about supplied files:

- FRTOS911.TXT: Information about supplied files

This file contains information related to REALOS/FR products.

■ Configurator

There are the following versions of the configurators:

- frcfs.exe: MS-DOS Prompt Version

REALOS provides Softune REALOS configurators as add-in modules to Softune.

The following two files are provided:

- SIRCF.DLL: Configurator DLL (Japanese-language)
- SIRCFENU.DLL: Configurator DLL (English-language)

This is the fundamental part of REALOS/FR. The basic kernel module manages tasks and controls execution of handlers in compliance with µITRON3.0 specifications.

This is a support tool used for REALOS/FR application system configuration.

Refer to the *Softune REALOS Configurator Manual* for details.

■ Basic kernel module

The following two types of basic kernel modules are provided:

- R_entry.asm, etc.: Kernel source (only built-in versions)
- R_entry.obj, etc.: Kernel object (only evaluation versions)

User-Created Module

init.asm provides samples covering processing that varies according to the user's configuration. Specifically, the following items are included:

- Reset entry routine
- User initialization routine
- System clock timer interrupt handler
- System down routine

System Call Interface Source File

The following system call interface source file is provided:

actcyc.asm, etc.: Source program for the REALOS/FR system call interface library, written in C.

System Call Interface Libraries

The following system call interface libraries are provided:

- realos.lib: REALOS/FR system call interface library (for Fujitsu tools)
- realos.a: REALOS/FR system call interface library (for GHS tools)

These are C system call interface libraries that are used each time a system call is executed from a program written in C. Refer to the *Softune REALOS/FR Kernel Manual* for the role of the system call interface libraries.

User Include Files

The following five types of user include files are provided:

- itron.h: User include file (ITRON common constants)
- itron.inc: User include file (ITRON common constants)
- realos.h: User include file (prototype declarations)
- MB911xx.h: User include file (chip-dependent constants)
- MB911xx.inc: User include file (chip-dependent constants)

Kernel Include File

The following kernel include file is provided:

- kernel.inc: Kernel include file
- fj_tool: Kernel include file
- ghs_tool: Kernel include file

This file defines data required for kernel assembly. (Direct user access is not permitted.)

OS object file

The following OS object files are provided:

- dbgfk.obj: OS object file (Fujitsu tools)
- dbgfk.o: OS object file (GHS tools)

An OS object file corresponding to the tool being used must be specified for RELOCATE_FILE in the configuration file.

■ Sample systems

The following seven sample system files are provided:

- realos.rcf: Configuration files for sample system
- smptsk1.c: Sample system task 1
- smptsk2.c: Sample system task 2
- init.asm: User initialization file for sample system
- smpsys.prj: Workbench sample system project file
- smpsys.dat: Workbench data option file
- R_smpsys.h: Sample system include file
- R_smpsys.inc: Sample system include file
- readme.txt: Information about sample system project files

These system configuration files are designed for configuration of sample tasks and sample systems, and are intended for reference purposes when user systems are built.

For details, see Appendix B, “Sample System Startup.”

■ Sample Drivers

The following six sample drivers are provided:

- io.c: Sample I/O interface module
- rsdrv.asm: RS232C driver
- realos.rcf: Configuration file for sample system (using sample drivers)
- smptsk1.c: Task 1 for sample system (using sample drivers)
- smptsk2.c: Task 2 for sample system (using sample drivers)
- init.asm: Sample system (using sample drivers) user initialization file

These files contain sample I/O drivers for the standard target board, which is the FR evaluation board (daughter board: MB91901). Files are presented in source code format. They can be assembled and used without modification, and may also be modified to suit user systems or may be used for reference purposes when drivers are created for other hardware.

For details, see Appendix E, “Sample I/O Drivers.”

■ Analyzer

Refer to the *Softune REALOS/FR Analyzer Manual* for the file related to Analyzer.

Note:

An OS object file corresponding to the tool being used must be specified for RELOCATE_FILE in the configuration file.

Appendix B Sample System Startup

REALOS/FR provides sample programs for reference when creating user programs.

■ Sample System Description

The sample system is intended to illustrate the basic configuration of an application program running on REALOS/FR. The system processes a simple set of instructions using the basic functions of REALOS/FR. The processing itself has no particular significance. Operation of the sample system must be verified by using debugging tools. Figure B-1 and Figure B-2 show the configuration and operation of the sample system.

The idle task is not used in the sample system. Details on describing of idle task, refer to Section 3.3, “Task Creation,” in the *Softune REALOS/FR KERNEL MANUAL*.

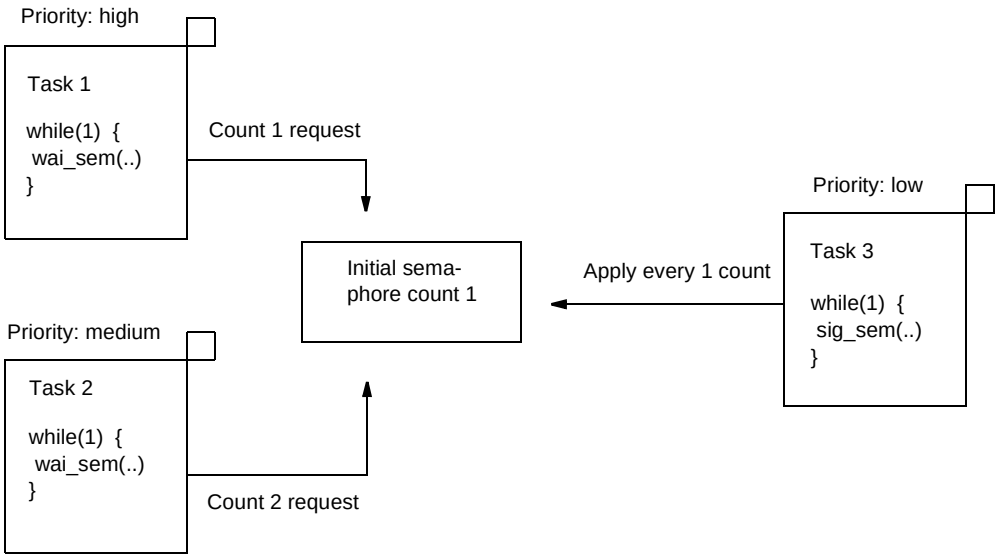


Figure B-1 Sample System Configuration

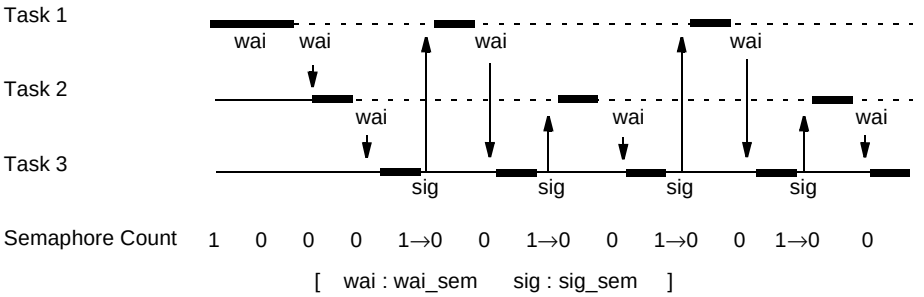


Figure B-2 Sample System Operation

■ Setting up the Development Environment

The development environment must be set up before the sample systems are activated.

When REALOS/FR is installed, the sample systems are stored in the directories shown in Figure B-3.

In this section, sample system startup with the directory configuration in Figure B-3 is assumed.

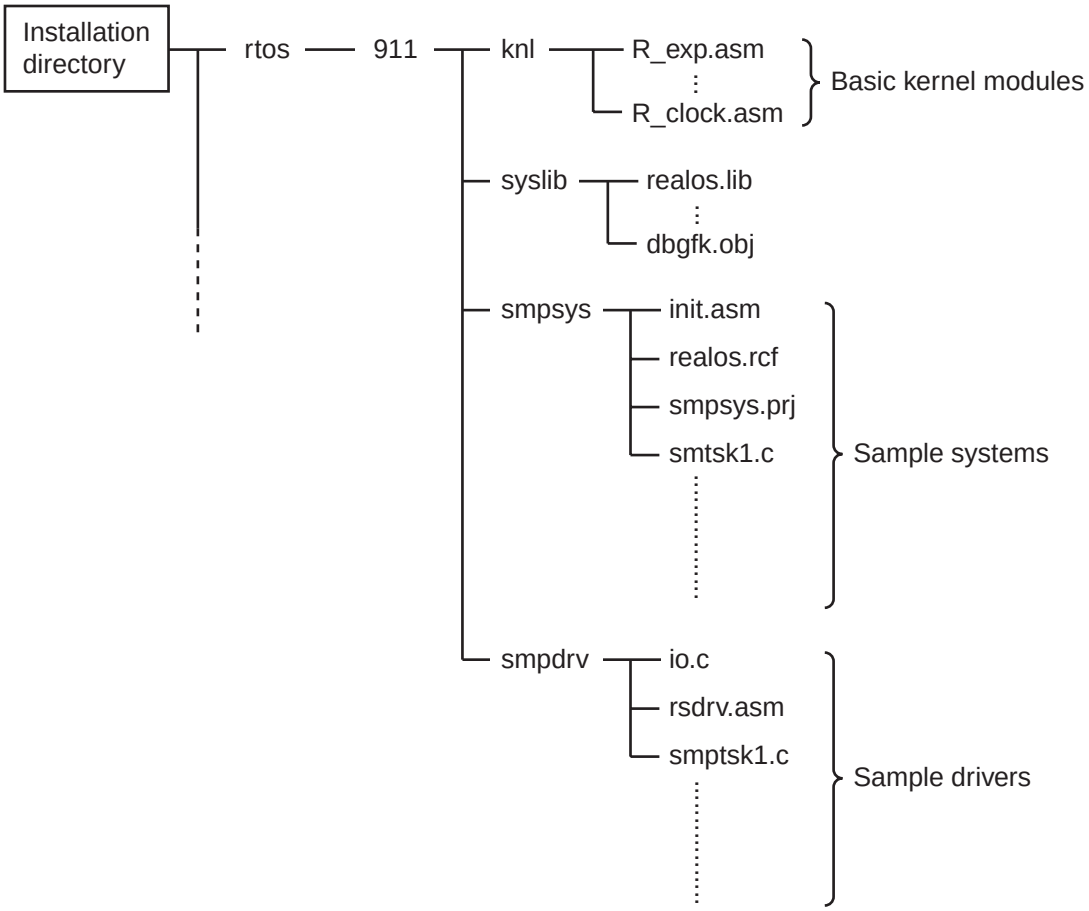


Figure B-3 Sample System Storage Directory

An assembler and linker are required for application system configuration on REALOS/FR. Sample system startup also requires the use of a C compiler because the sample system programs are written in C. A debugger is also required for debugging the configured application systems.

Check whether these cross-language development tools are installed on the host computer.

■ Examples of Creating an Execution Module File and of Creating a Sample System

This section contains examples of configuring a sample system and of creating an execution module file.

In the examples, the MS-DOS prompt command line is used.

❍ Setting the path

Use AUTOEXEC.BAT to set the path to the directory where the compiler, configurator, and other support tools are stored.

Using AUTOEXEC.BAT is the usual way to set the path.

The following example shows how to set the path using the command line:

Example of setting the path

```
C:\>SET PATH = %PATH%;C:\SOFTUNE\BIN
```

Note:

The setting added to AUTOEXEC.BAT takes effect the next time the system is activated.

❍ Setting environment variables

Set the name of the directory where REALOS/FR is installed in environment variable RTOS.

Specify the directory used by the support tools as their work directory in environment variable TMP.

Environment variables are usually defined in AUTOEXEC.BAT.

The following is an example of setting environment variables using the command line:

Example of setting environment variables

```
C:\>SET RTOS = C:\SOFTUNE
C:\>SET TMP = C:\TMP
```

Note:

The setting added to AUTOEXEC.BAT takes effect the next time the system is activated.

❍ Compiling a source program

Compile the sample system source program.

Execute the following command from the directory containing the sample system:

```
>FCC911S -g -c -I ..\syslib -cpu MB91101 -K realos smptsk1.c
>FCC911S -g -c -I ..\syslib -cpu MB91101 -K realos smptsk2.c
```

Note:

The sample system is described for Fujitsu made tool.

Customers using the GHS tools should execute the following commands instead of those above:

```
>CCFR -V -elf -g -c -I ..\syslib -o smptsk1.o smptsk1.c
>CCFR -V -elf -g -c -I ..\syslib -o smptsk2.o smptsk2.c
```

❍ **Assembling the user initialization file**

Use the assembler to assemble the sample system user initialization file init.asm.

Execute the following command:

```
>FASM911S -g -I ..\syslib -cpu MB91101 init.asm
```

Note:

Customers using the GHS tools should execute the following command instead of those above:

```
>ASFR -V -elf -g -I ..\syslib -o init.o init.asm
```

❍ **Configuring the sample system**

Activate the configurator to configure the sample system in the sample system configuration file realos.rcf.

Execute the following command:

```
>frcfs -a -f realos.rcf
```

The sample system execution module file smpsys.abs is created in the current directory.

Note:

Customers using the GHS tools should define the configuration as explained below.

Modify the RELOCATE_FILE specification in realos.rcf as shown in Figure B-4:

```
RELOCATE_FILE init.obj
RELOCATE_FILE ..\syslib\dbgfk.obj
RELOCATE_FILE smptsk1.obj
RELOCATE_FILE smptsk2.obj
```

↓

```
RELOCATE_FILE init.o
RELOCATE_FILE ..\syslib\dbgfk.o
RELOCATE_FILE smptsk1.o
RELOCATE_FILE smptsk2.o
```

Next, change “# LIBRARY_FILE ..\syslib\realos.a” to “LIBRARY_FILE \syslib\realos.a.”

Execute the command below to execute the configuration.

If GHS tools are used, be sure to specify the -GHS option.

```
>frcfs -GHS -f realos.rcf
```

An execution module file is created as same as above.

```
# ***************************
# *   Cpu Type Definition   *
# ***************************
  CPU MB91101

# ************************************
# *   Relocatable Object Definition   *
# ************************************
  RELOCATE_FILE  init.obj
  RELOCATE_FILE  ..\syslib\dbgfk.obj
  RELOCATE_FILE  smptsk1.obj
  RELOCATE_FILE  smptsk2.obj

# ************************
# *   Library Definition   *
# ************************
#  LIBRARY_FILE  ..\syslib\realos.lib
#  LIBRARY_FILE  ..\syslib\realos.a

# ******************************
# *   Output Module Definition   *
# ******************************
  ABSOLUTE_FILE smpsys

# ***************************
# *   Map File Definition   *
# ***************************
  MAP_FILE  LIST

# ******************************
# *   Cylic Handler Definition   *
# ******************************
  CYC_HDR_NUMBER 0

# ******************************
# *   Alarm Handler Definition   *
# ******************************
  ALM_HDR_NUMBER  0

# ***************************
# *   Priority Definition   *
# ***************************
  PRIORITY_LEVEL  32

# ********************************
# *   System Stack Size Definition  *
# ********************************
  SYS_STK_SIZE  0x400
```

Figure B-4 realos.rcf *(Continue)*

```
# ****************************
# *   Memory Map Definition   *
# ****************************
  ROM 0xf0000,0xfffff,romarea
  RAM 0x600000,0x61ffff,ramarea

# ***********************************
# *   Kernel Memory Map Definition   *
# ***********************************
  KNL_RAM 0x00600000
  KNL_ROM 0x000f0000

# *********************************
# *   User Memory Map Definition   *
# *********************************
  USR_SECTION CODE,0x000f8000
  USR_SECTION startcode,0x000f9000
  USR_SECTION R_eit,0x000ffc00

# *********************************
# *   User Initialize Definition   *
# *********************************

# *** task definition ***
  TSK _tsk_1,_task1,10,0x400,READY,1,1,1
  TSK _tsk_2,_task1,20,0x400,READY,2,2,2
  TSK _tsk_3,_task2,30,0x400,READY,3,3,3

# *** semphore definition ***
  SEM _sem_1,1,32767,1,0

# ***********************************
# *   Interrupt handler Definition   *
# ***********************************
  EIT_ENTRY  35,_timer

#
# ** End of Definition **
#
```

Figure B-4 realos.rcf *(Continued)*

■ Sample System Startup Using Softune Workbench

This section explains sample system setup using Softune Workbench.

The explanation covers startup when creating a new project and startup using an existing project file.

❍ Startup when creating a new project

1) Creating a project

Create a project:

Select File and then New on the menu. The New dialog box opens.

Select “Project file” as the file type as shown in Figure B-5, and click the OK button.

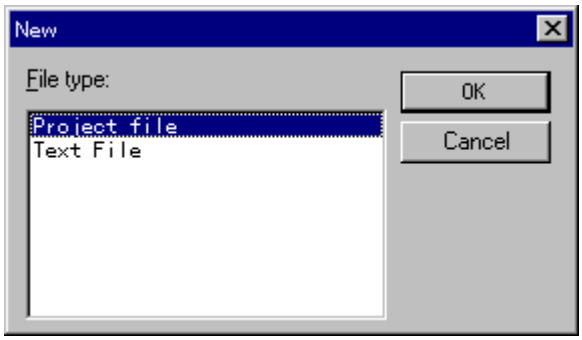


Figure B-5 New Dialog Box

Click the OK button in the New dialog box to close it and open the Create New Project dialog box.

Figure B-6 shows the Create New Project dialog box.

Set data as follows:

- Project Type: REALOS(ABS)
- Project Name: Input a project name (any name).
- Target Filename: Input a target file name (any name).
- Project Directory: Specify a directory (any name) to save the project file.
- Chip Classification: FR
- Target MCU: MB91101

Click the OK button to close the dialog box and open the Create Configuration File dialog box.

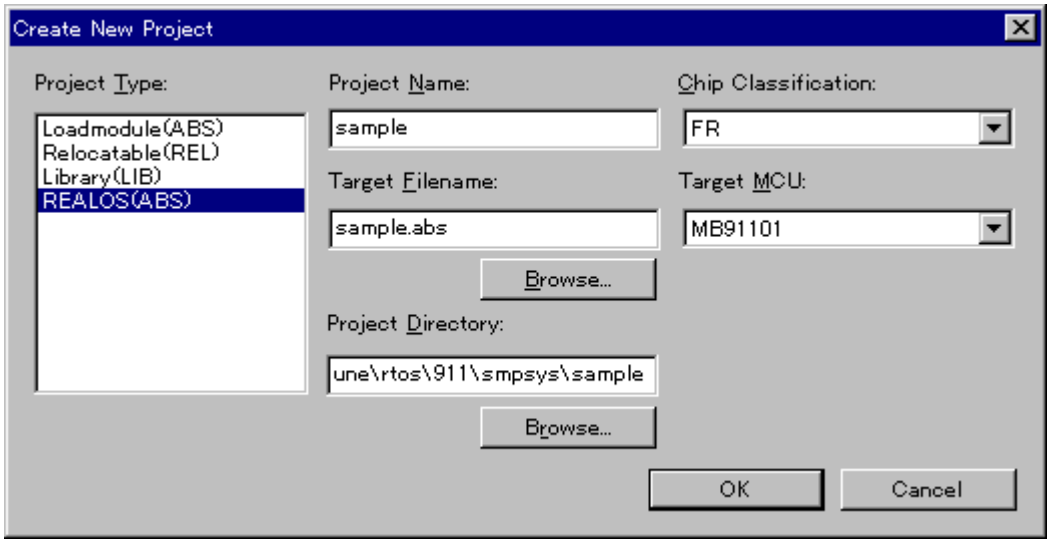


Figure B-6 Create New Project Dialog Box

Figure B-7 shows the Create Configuration File dialog box.

Select “An existing Configuration file” and specify the sample system configuration file realos.rcf. (See Figure B-3 for the directory containing realos.rcf.)

Set data as shown in Figure B-7. Click the OK button to close the dialog box and open the Set Configuration File dialog box.

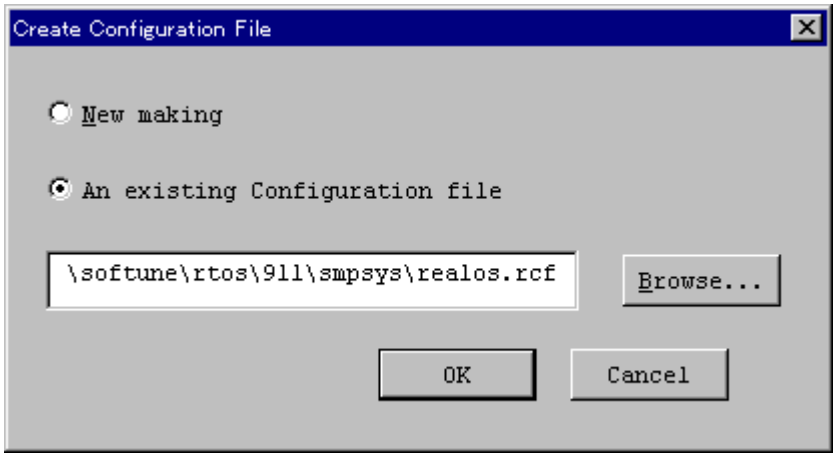


Figure B-7 Create Configuration File Dialog Box

Figure B-8 shows an example of the Set Configuration File dialog box.

Click the tab to check the set data.

Click the OK button to close the dialog box.

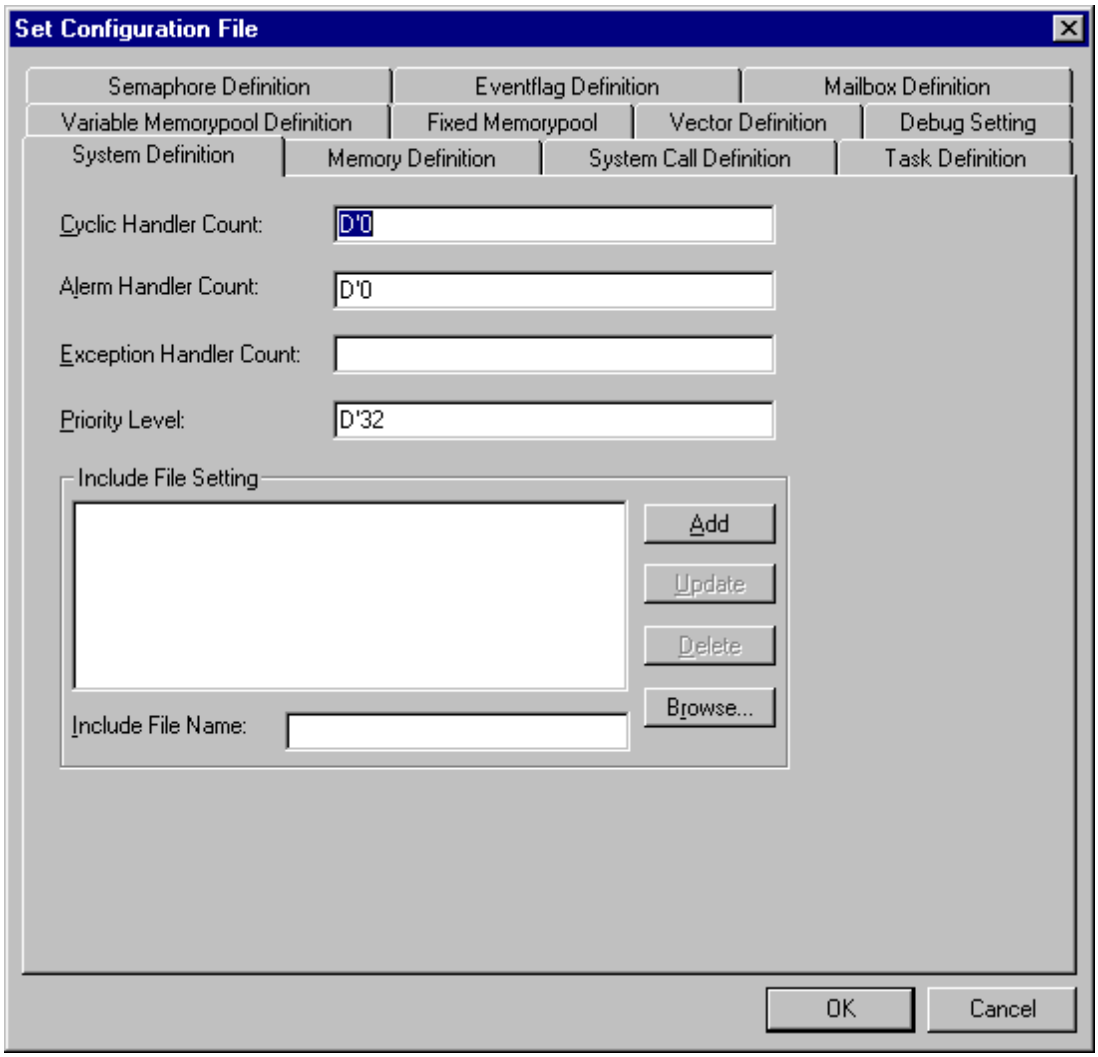


Figure B-8 Set Configuration File Dialog Box.

Note:

When the OK button is clicked to close the Set Configuration File dialog box, the warning message shown in Figure B-9 is displayed. The purpose of the message is to prevent the relocatable object and library files defined in a configuration file from being registered in the project when a configuration file has been loaded.

Click the OK button and register members as explained in Step 2), “Member registration to the project.”

Figure B-9 Warning Message (Member Registration)

2) Member registration to the project

As explained below, register the sample system source file, configuration file, and user initialization file in the created project as members:

Select Project and then Add Member on the menu to open the Add Member dialog box.

Figure B-10 shows an example of the Add Member dialog box.

Select the sample system storage directory in the “Look in” drop-down list box.

As shown in Figure B-10, specify “Source File(*.c;*.asm)” as the file type and select init.asm, smptsk1.c, and smptsk2.c.

Click the OK button to register these files in the project as members.

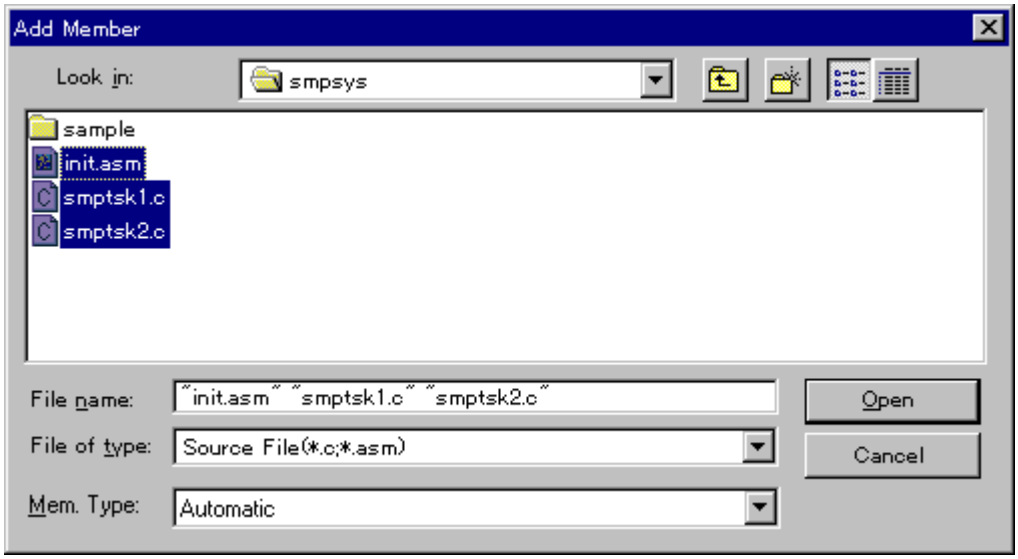


Figure B-10 Add Member Dialog Box (Source Registration)

Figure B-11 shows the project status after member registration.

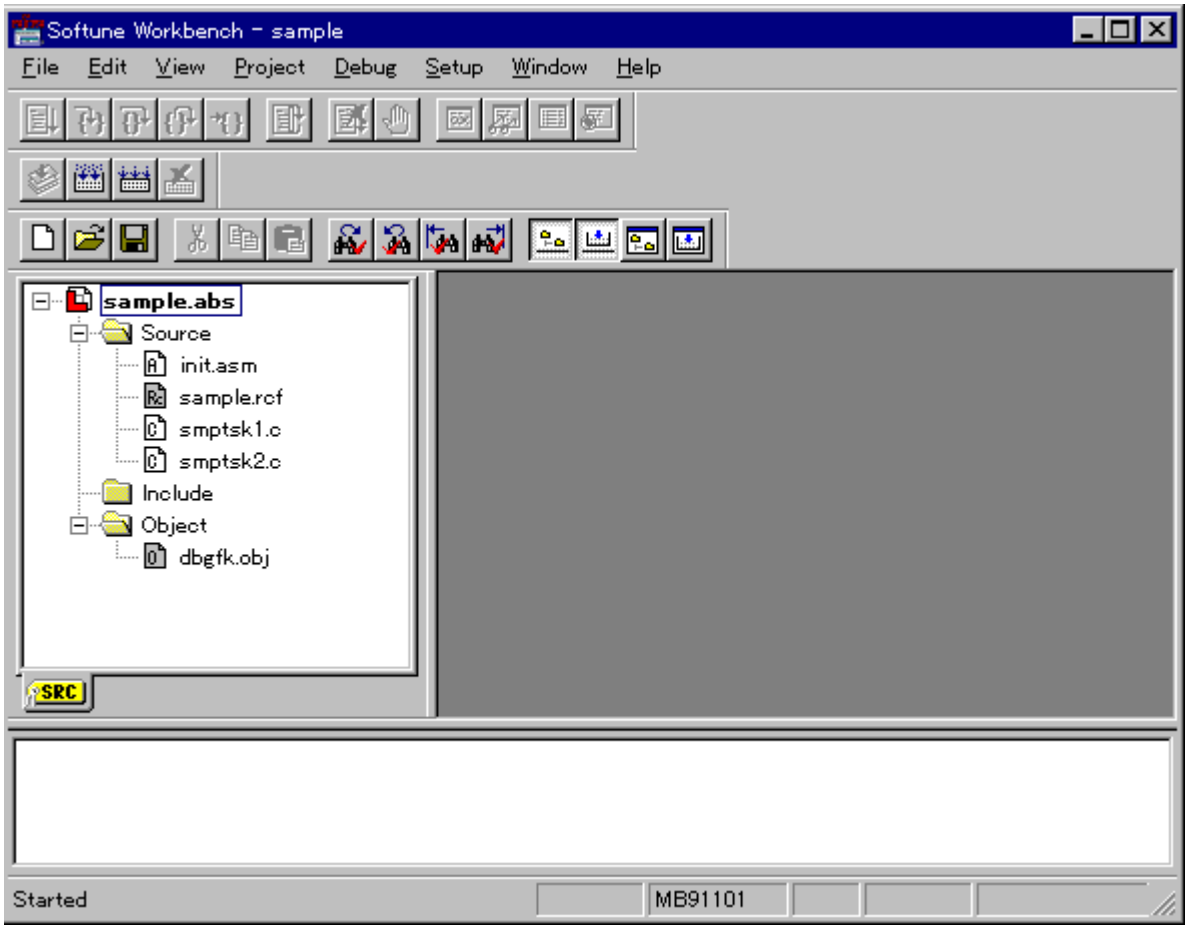


Figure B-11 Project Status after Member Registration

3) Build

Select Project and then Build on the menu to execute build.

All source files registered in the project are compiled/assembled and all objects and libraries are linked to create an absolute format object file (load module file).

When build terminates, a message indicating termination is displayed in the output window as shown in Figure B-12.

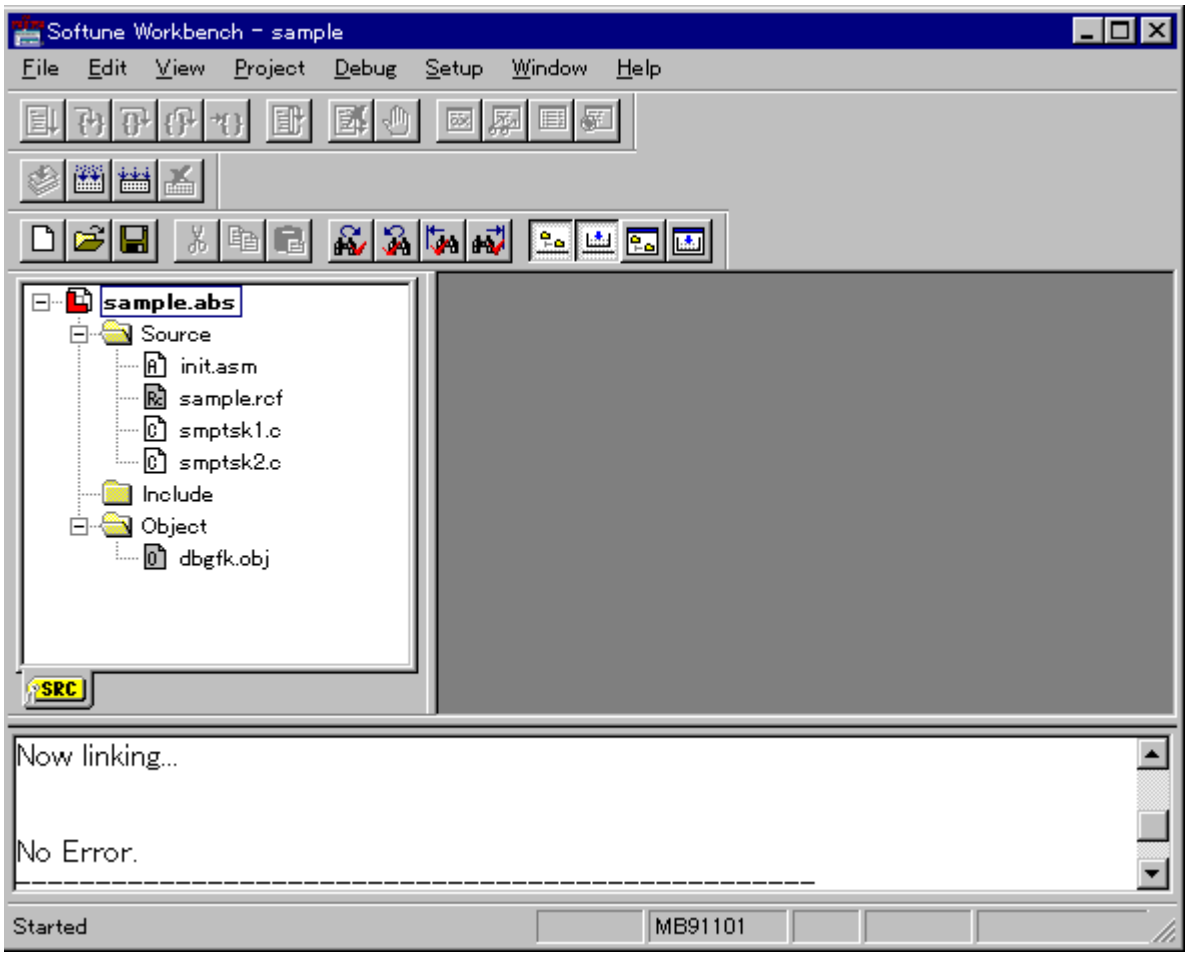


Figure B-12 Display after Build Execution

4) Debugging

Activate the sample system to enable debugging as explained below:

Select Debug and then Start Debug on the menu to switch to debug mode. The debugger setup wizard starts. Execute debugging while referring to the debugger manual.

Notes:

- The setup wizard is not displayed for subsequent debugging sessions after a project has been created.
- As an example, debugging using the simulator debugger is explained below.
- To debug using the simulator debugger, activate the sample system on the FR evaluation board (MB91901 daughterboard:), ROM alternate unit (MB92197-90), and ICE (MB2197).

Load the sample program after setup.

Select Debug and then Load Sample Program on the menu to load the sample program.

Preparation for sample program activation is complete.

An example of a break at the beginning of sample task 1 is shown below:

Double-click smptsk1.c in the Project window to open the Source window. Next, click the circle on the first task 1 line to set a breakpoint, as shown in Figure B-13.

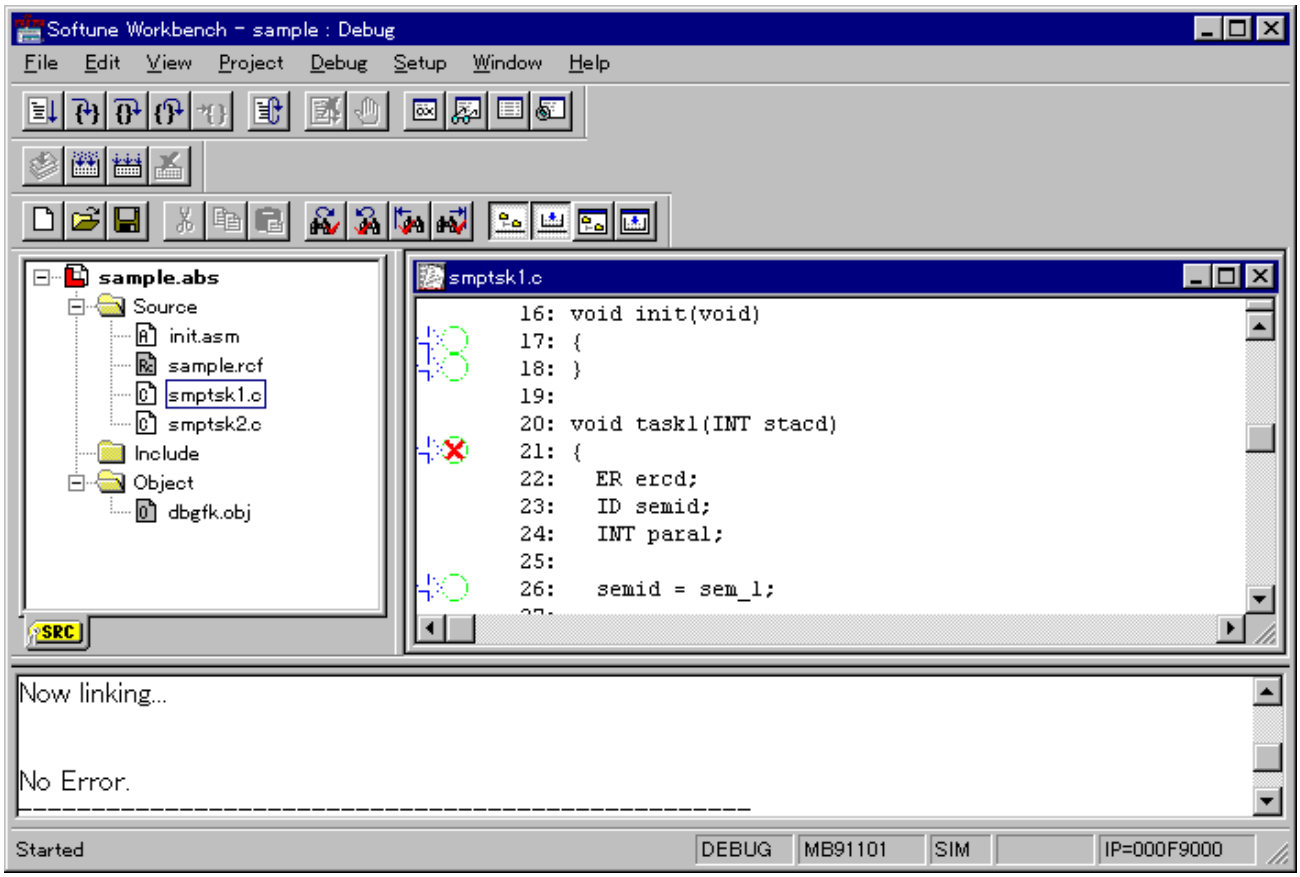


Figure B-13 Setting a Breakpoint

Select Debug, Execute and then Continue on the menu. The system is activated, and the breakpoint set in the Source window is active.

Figure B-14 shows the status.

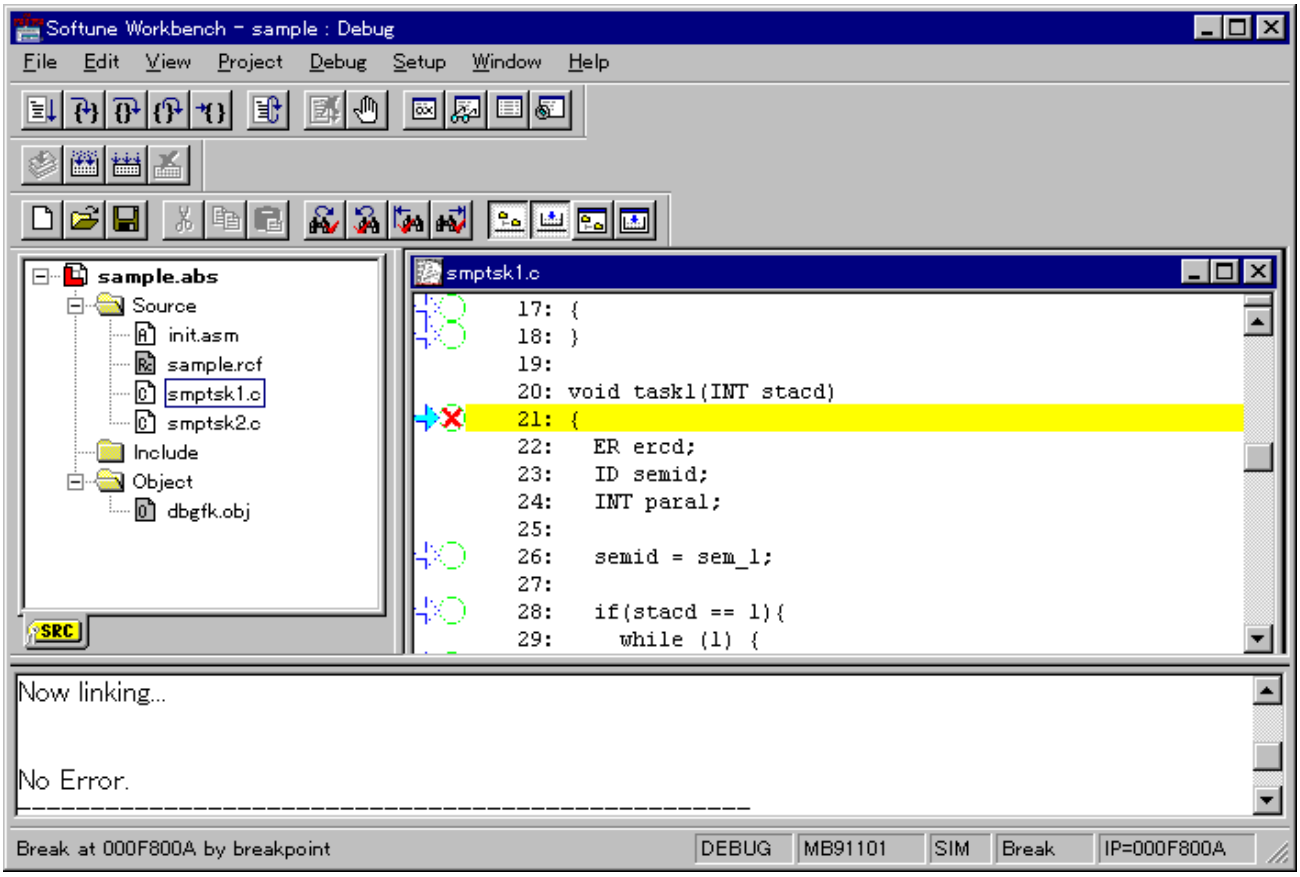


Figure B-14 Broken status

❍ **Startup using sample system project file**

Select “File” and then “Open Project...” on the menu to open the “Open Project” dialog.

Figure B-15 shows the “Open Project” dialog.

Select “smpsys.prj” in the sample system project file storage directory and click the Open button.

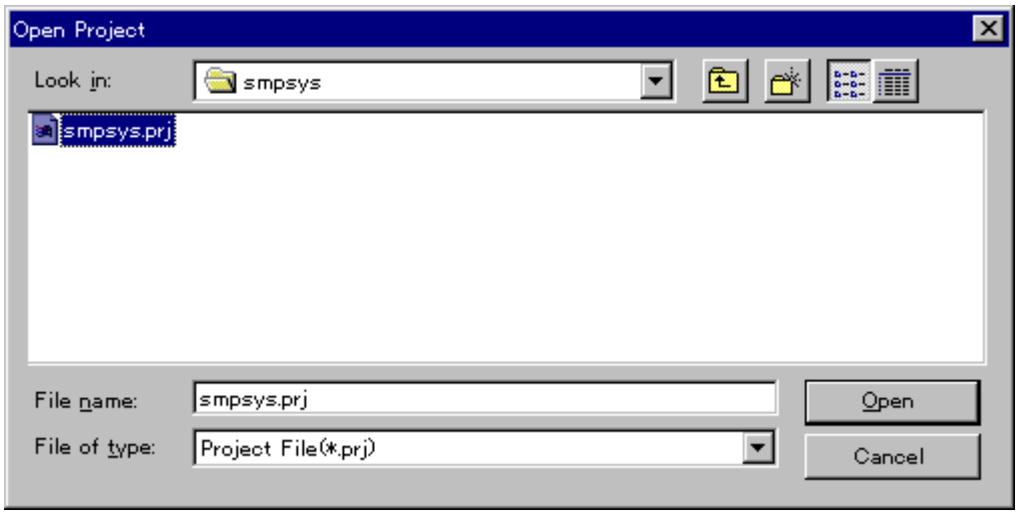


Figure B-15 “Open Project” dialog

Then, the message shown in Figure B-16 is displayed.

Click the Yes button to create the directory.

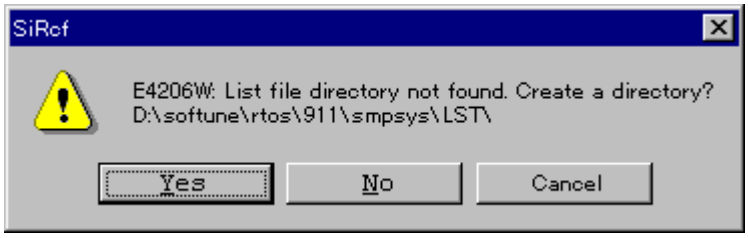


Figure B-16 Warning Message (Creation of LST Directory)

Next, the message shown in Figure B-17 is displayed.

Click the Yes button to create the directory.

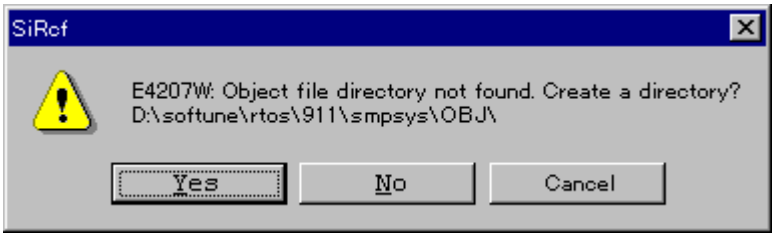


Figure B-17 Warning Message (Creation of OBJ Directory)

Finally, the message shown in Figure B-18 is displayed.

Click the Yes button to create the directory.

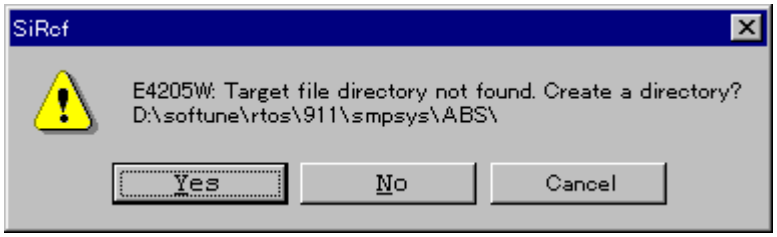


Figure B-18 Warning Message (Creation of ABS Directory)

Next, the file readme.txt is displayed as shown in Figure B-19. Carefully read this file, since it contains information about the sample system project files.

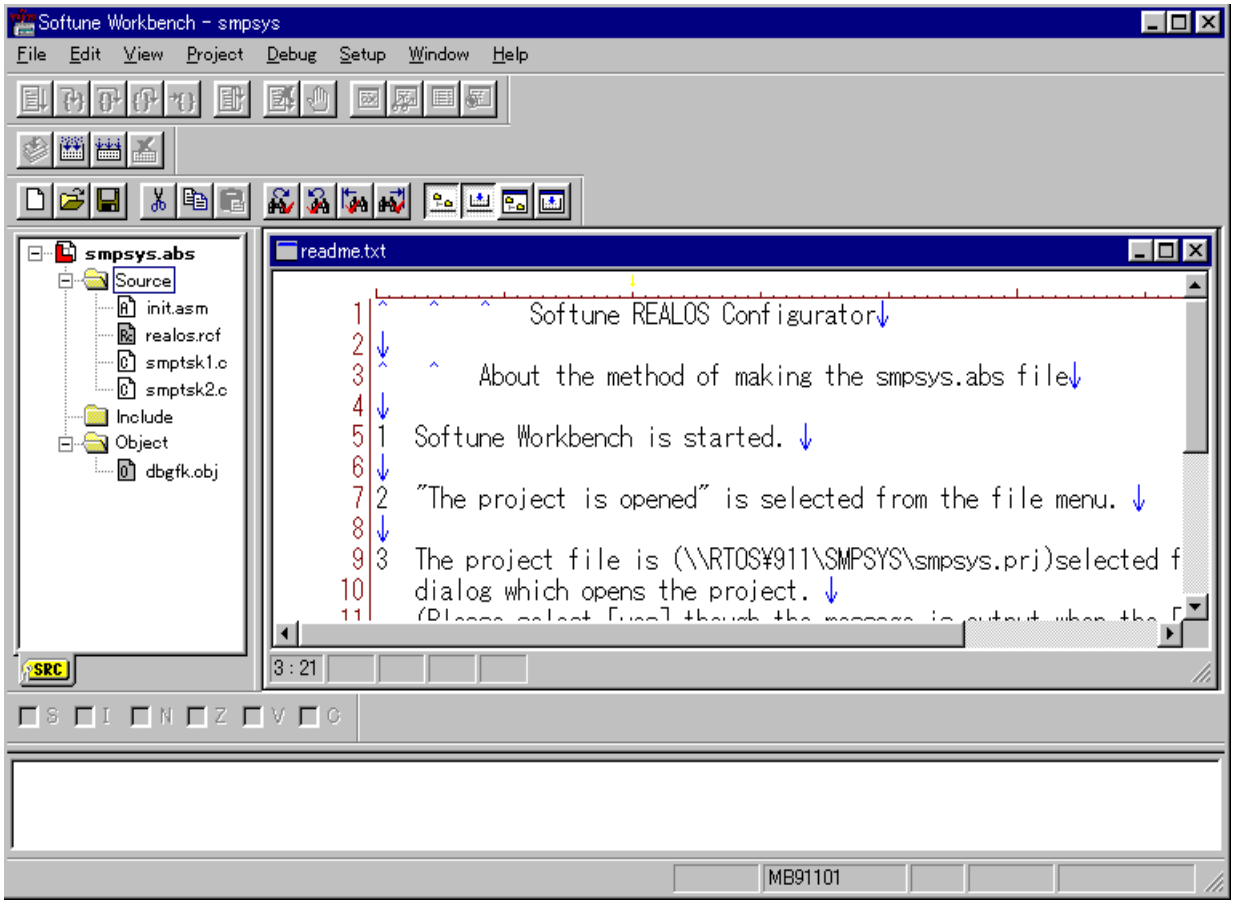


Figure B-19 readme.txt File

The build becomes executable.

For subsequent steps, follow the procedure used to start up a newly created project.

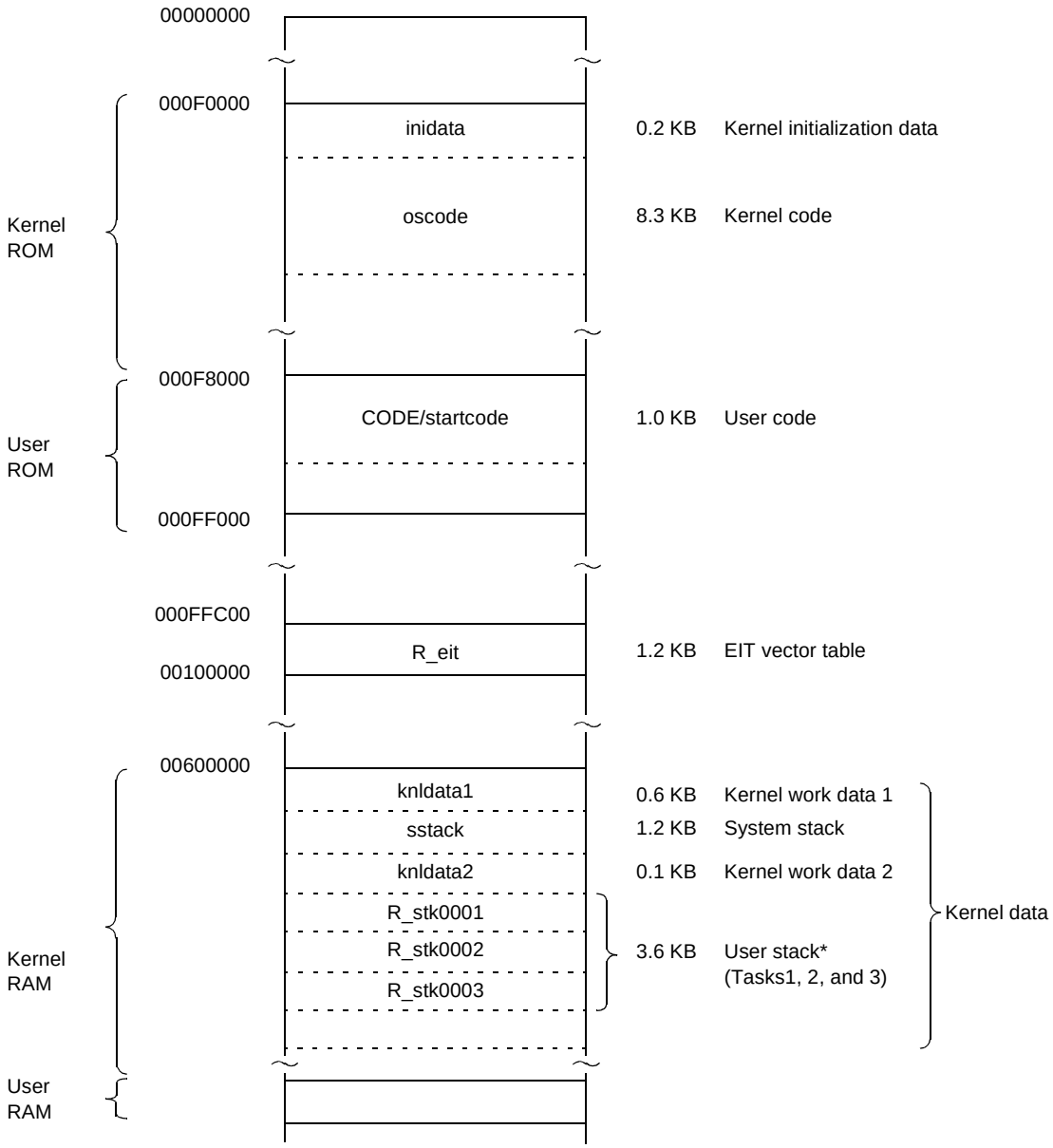
See step 3) and following above.

Appendix C Sample System Memory Map

As an indicator of the memory capacity required to operate REALOS/FR, this appendix describes a map of the memory capacity required by the sample system.

■ Sample System Memory Map

Figure C-1 and Figure C-2 shows a memory map for the sample system.



Note: For methods used to allocate kernel data, see Appendix B, “Kernel Code, Kernel Data” in the *Softune REALOS Configurator Manual.*

Figure C-1 Sample System Memory Map *(Continue)*

*: The OS uses the stack area in each task of user stack as a context area in task. The OS uses about 96-byte stack area for each task.

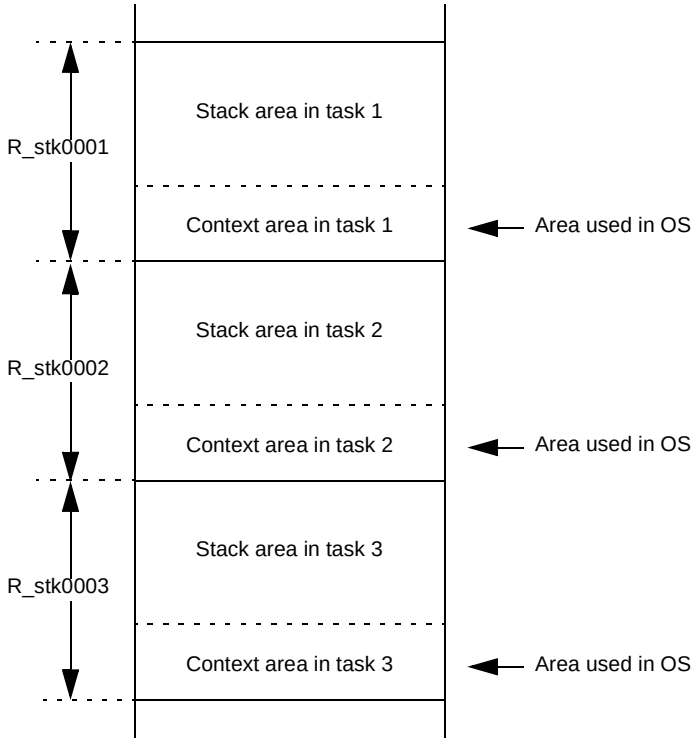


Figure C-2 Sample System Memory Map *(Continued)*

Appendix D Typical Target Systems

This appendix describes the requirements and typical configurations of target hardware. RELOS/FR can accommodate the requirements of basic target hardware configurations.

■ Basic Target Hardware Configuration

Figure D-1 shows the basic target hardware configuration.

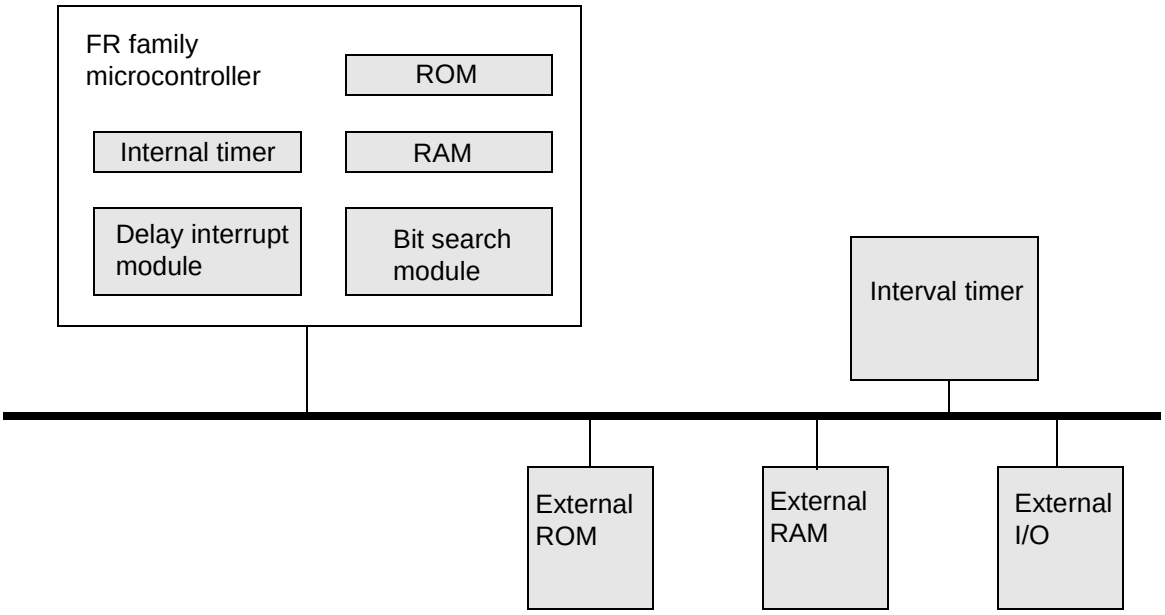


Figure D-1 Basic Target Hardware Configuration

Notes:

- REALOS/FR operation requires an FR family microcontroller with built-in delay interrupt module and bit search module.
- REALOS/FR operation requires one interval timer generating timer interrupt signals at constant intervals (typically 1 ms) for use as a system clock.
  The FR family internal timer may also be used.
- System configuration is possible even without a system clock. However the following system calls are related to timing, and will not operate properly.

  tslp_tsk , set_tim , get_tim , dly_tsk , def_cyc , def_alm, twai_sem, twai_flg, trcv_msg, tget_blf
- The delay interrupt module, bit search module, system clock interval timer and vector 64 EIT are used by REALOS/FR and are not available to the user.

Appendix E Sample I/O Drivers

REALOS/FR provides sample I/O drivers for use as a reference program when creating I/O drivers for user systems. These drivers are designed for an RS232C (UART0) connection to the FR evaluation board (daughter board: MB91901).

■ Sample I/O Driver Configuration

The sample I/O drivers are composed of two parts:

- **Sample I/O interface module (filename: io.c)**

  The interface module provides the read functions and write functions for the RS232C driver.

- **RS232C Driver (filename: rsdrv.asm)**

  The driver controls the RS232C (UART0) chip on the FR evaluation board (daughter board: MB91901), and provides data input and output.

■ Sample I/O Interface Module Functions

The sample I/O interface contains read functions and write functions.

- **read functions**

```
int read(int fileno, char *buf, unsigned int size);
```

A data quantity of “size” bytes is input to the area designated by “buf.”

The “fileno” parameter is ignored.

The 1-character input function of the RS232C driver is used.

- **write functions**

```
int write(int fileno, char *buf, unsigned int size);
```

A data quantity of “size” bytes in the area designated by “buf” is output.

The “fileno” parameter is ignored.

The 1-character output function of the RS232C driver is used.

■ RS232C Driver Functions

The RS232C driver has the following functions.

❍ Driver initialization (entry name: __init_rs)

Initialization includes the following processes:

- Receiving buffer initialization
- UART0 initialization
- Receiving interrupt setting

❍ Single character input (entry name: __getchr)

This function gets one character from the ring buffer which is managed by semaphore.

If there are no characters in the ring buffer, the semaphore remains in wait state until a character is received.

If error occured during input, this function return -1 instead of the character.

❍ Single character output (entry name: __putchr)

This function sends one character to the UART0 chip.

If the system is not ready to transmit, the interface remains in polling state until transmission is enabled.

❍ Receiving interrupt handler (entry name: __inthdr_rs)

This function is activated by an UART0 chip receiving interrupt.

One character is received from the UART0 chip and is stored in the ring buffer. If a task is waiting for the receipt of that character, a wake-up signal is sent to that task.

■ How to Use the Sample I/O Driver

Use of the I/O driver requires the following settings.

❍ RS232C driver settings

The RS232C driver has settings for receiving buffer size. Settings are made by changing the corresponding define statement in the source program.

Also, it is possible to create drivers for different communication conditions by rewriting the driver initialization routine.

❍ Adding Configuration Definitions

The following definitions must be added during configuration.

Relocatable object definitions

Add definition of the sample I/O interface module (filename: io.obj) and RS232C driver (filename: rsdrv.obj) as relocatable objects.

Semaphore definition

Define one semaphore for the use of the sample I/O driver. The semaphore name should be “_RS_Sem”, initial count should be 0 and maximum count value should be one less than the size of the receiving buffer.

Interrupt handler definition

Define the RS232C driver receiving interrupt handler (entry name: __inthdr_rs) as the receiving interrupt (interrupt number: 4) for the UART0 chip on the FR evaluation board (daughter board: MB91901).

If necessary, also define the section array addresses for the sample I/O driver.

In addition, the system calls wai_sem and sig_sem are used by the sample I/O driver and both should be built into the system.

■ Cautionary Information for Use of the Sample I/O Driver

On using the sample I/O driver, use care in the following respects.

❍ Exclusive control of the sample I/O driver

The sample I/O driver does not provide exclusive control. The user must provide exclusive control for each separate read/write function.

❍ Execution from the task independent area

Because the RS232C driver processes single character input by using wai_sem system calls, the sample I/O driver cannot be used from the task independent area. Also, the sample I/O driver cannot be used in dispatch prohibited status. Proper operation is not assured if the driver is used in this manner.

■ Example of Installing the Sample I/O Driver

This section explains how to install the sample I/O driver in the sample system (see Appendix B, “Sample System Startup”). In the example, Softune Workbench is used.

1) Creating a project

Select File and then New on the menu. The New dialog box opens.

Select “Project file” as the file type as shown in Figure E-1, and click the OK button.

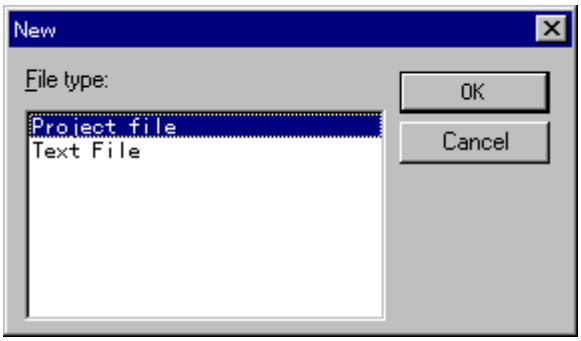


Figure E-1 New Dialog Box

Click the OK button in the New dialog box to close it and open the Create New Project dialog box.

Figure E-2 shows the Create New Project dialog box.

Set data as follows:

- Project Type: REALOS(ABS)
- Project Name: Input a project name (any name).
- Target Filename: Input a target file name (any name).

- Project Directory: Specify a directory (any name) to save the project file.
- Chip Classification: FR
- Target MCU: MB91101

Click the OK button to close the dialog box and open the Create Configuration File dialog box.

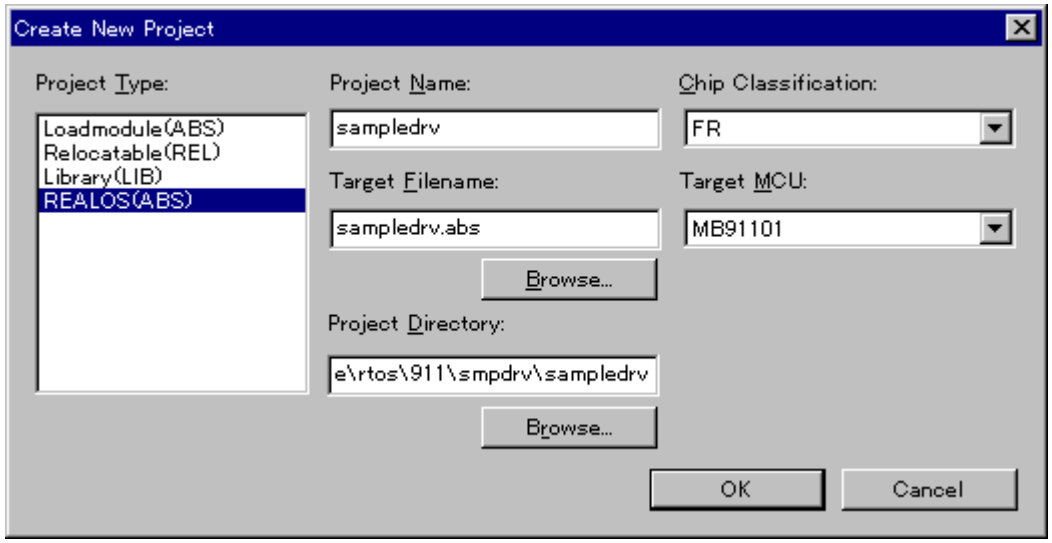


Figure E-2 Create New Project Dialog Box

Figure E-3 shows the Create Configuration File dialog box.

Select “An existing Configuration file” and specify the sample system configuration file realos.rcf. (See Figure B-3 for the directory containing realos.rcf.)

Set data as shown in Figure E-3. Click the OK button to close the dialog box and open the Set Configuration File dialog box.

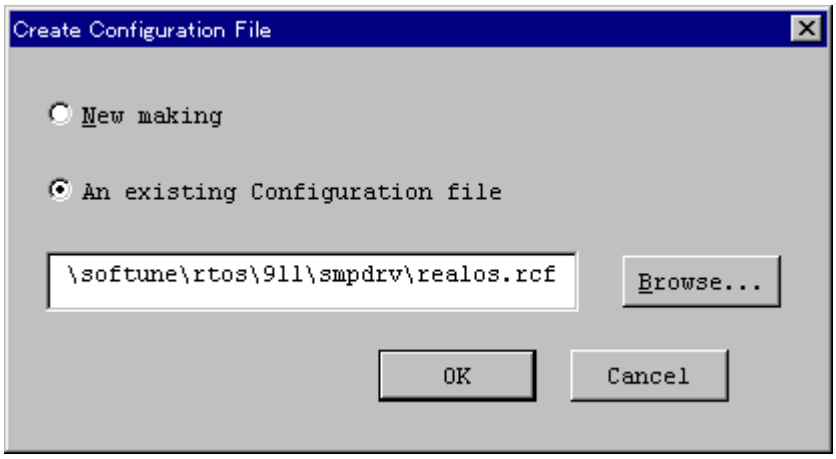


Figure E-3 Create Configuration File Dialog Box

Figure E-4 shows an example of the Set Configuration File dialog box.

Click the tab to check the set data.

Click the OK button to close the dialog box.

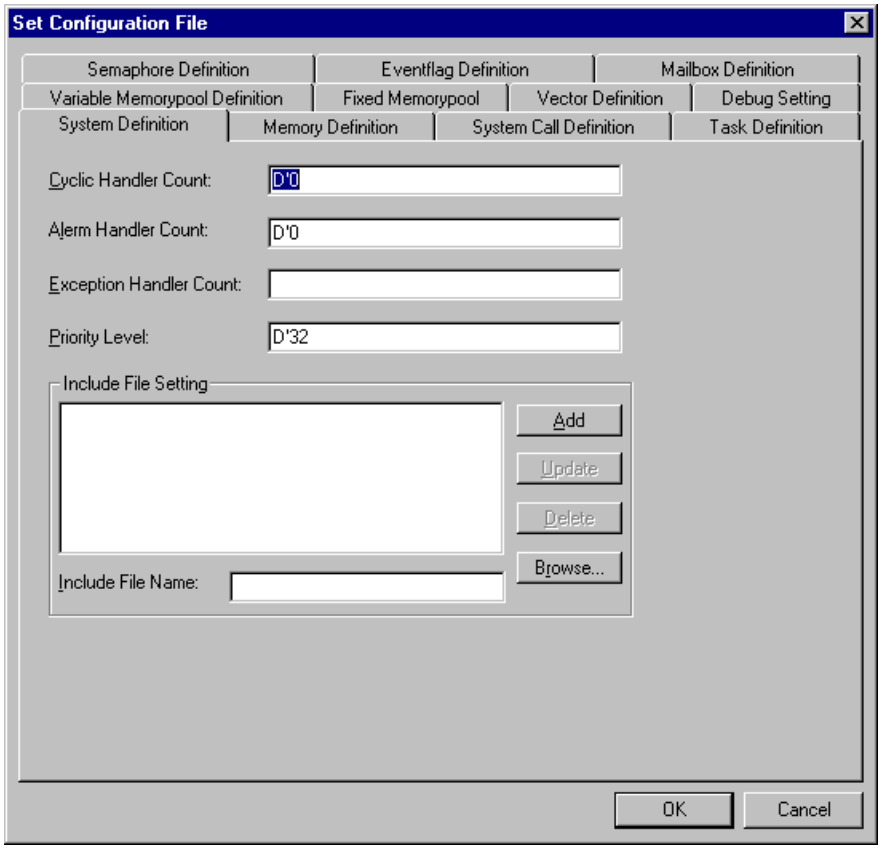


Figure E-4 Set Configuration File Dialog Box.

Note:

When the OK button is clicked to close the Set Configuration File dialog box, the warning message shown in Figure E-5 is displayed. The purpose of the message is to prevent the relocatable object and library files defined in a configuration file from being registered in the project when a configuration file has been loaded.

Click the OK button and register members as explained in Step 2), “Member registration to the project.”

Figure E-5 Warning Message (Member Registration)

2) Member registration to the project

As explained below, register the sample system source file, configuration file, and user initialization file in the created project as members:

Select Project and then Add Member on the menu to open the Add Member dialog box.

Figure E-6 shows an example of the Add Member dialog box.

Select the sample system storage directory in the “Look in” drop-down list box.

As shown in Figure E-6, specify “Source File(*.c;*.asm)” as the file type and select init.asm, smptsk1.c, and smptsk2.c.

Click the OK button to register these files in the project as members.

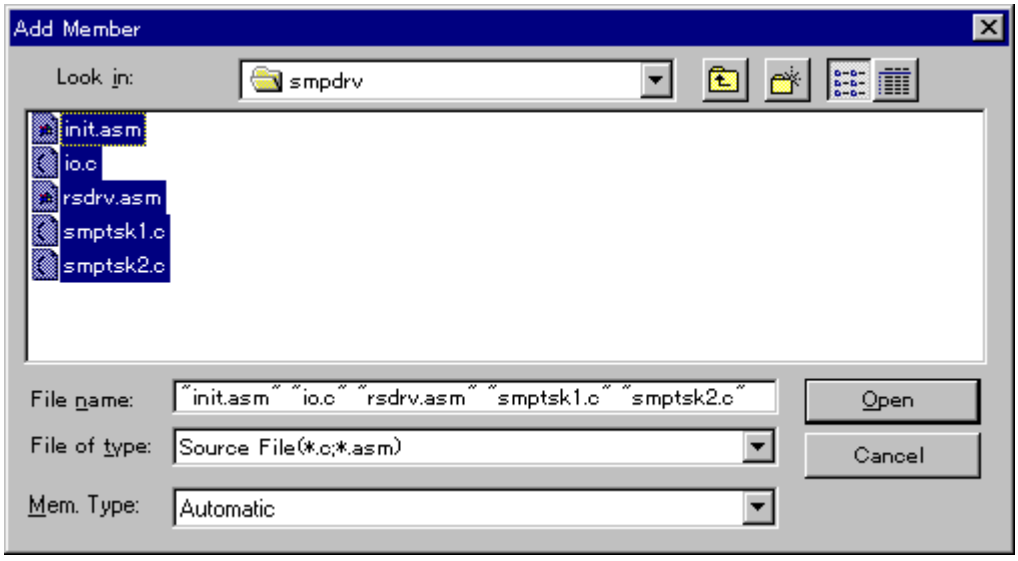


Figure E-6 Add Member Dialog Box (Source Registration)

Figure E-7 shows the project status after member registration.

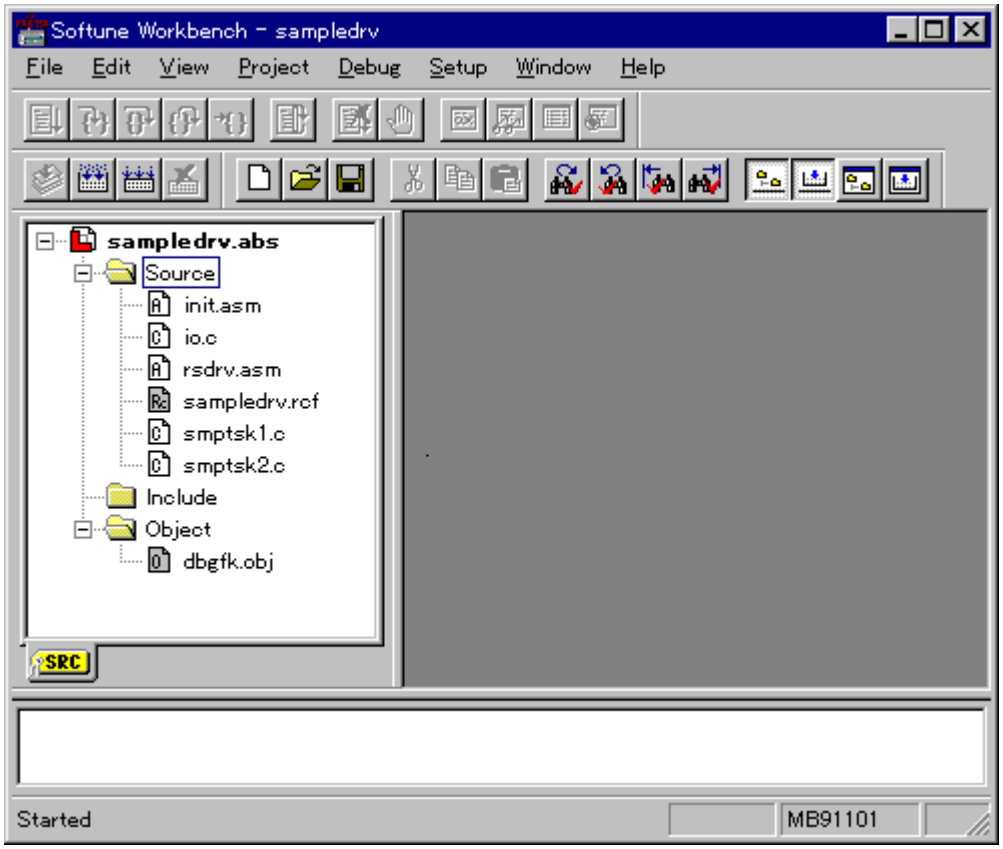


Figure E-7 Project Status after Member Registration

3) Build

Select Project and then Build on the menu to execute build.

All source files registered in the project are compiled/assembled and all objects and libraries are linked to create an absolute format object file (load module file).

When build terminates, a message indicating termination is displayed in the output window as shown in Figure E-8.

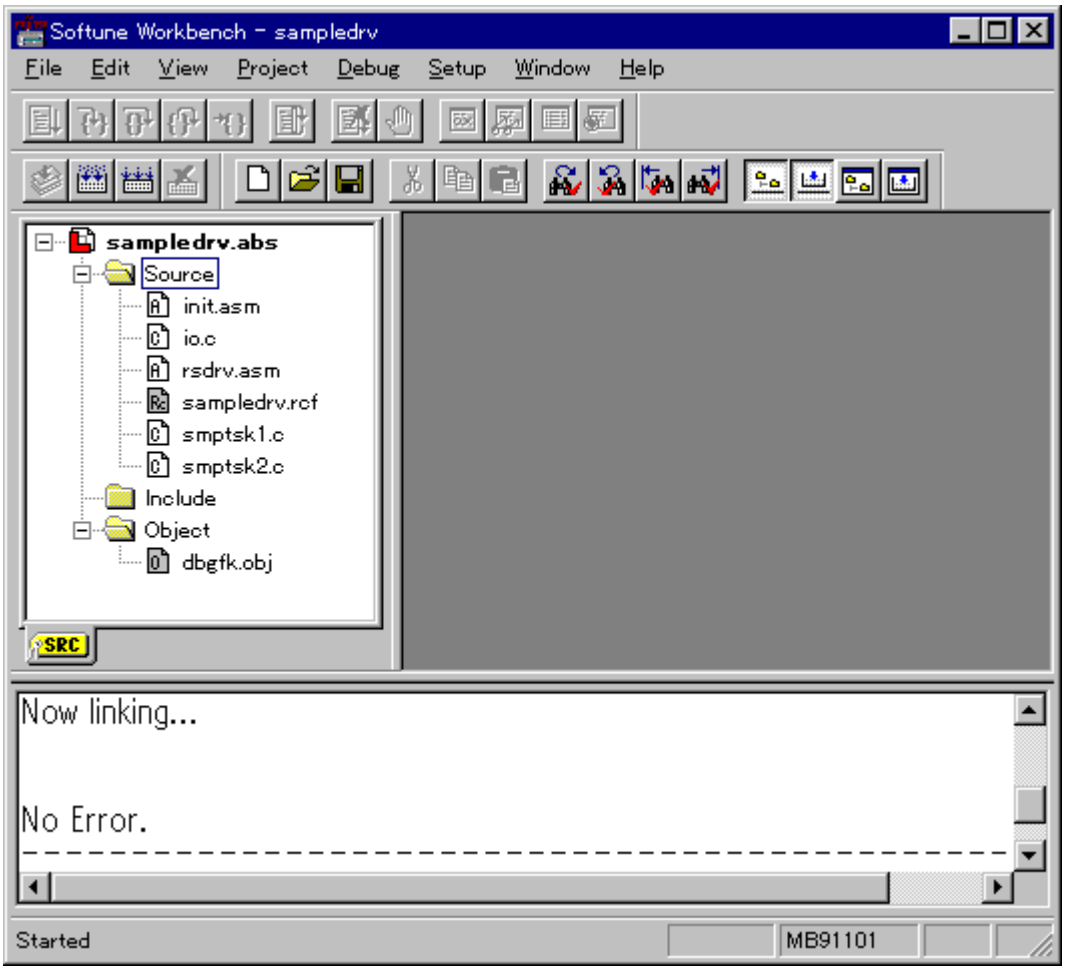


Figure E-8 Display after Build Execution

The sample system embedding the sample I-O has been built. Creating ROM or operation on the evaluation board using debugger can be done with this system.

Appendix F REALOS/FR Upgrade Overview

Function improvements and changes brought about by individual REALOS/FR upgrades are described below.

■ Version 02 Upgrades from Version 01

❍ Default frequency and SRAM wait count change for R_user.asm

The Default frequency is changed from 16 MHz to 25 MHz. The SRAM wait count is changed from 2 waits to 0 wait.

❍ Debugger macro tsk command specification change

The debugger macro tsk command stack display format is changed so as to display the task stack range.

■ Version 03 Upgrades from Version 02

❍ R_user.asm data cache setup process change

The data cache setup process for R_user.asm is changed to a comment.

❍ Configurator function expansion

The configurator is upgraded to support long file names.

❍ REALOS/FR CPU applicability addition

The REALOS/FR is now applicable to FR30 CPUs in addition to the FR20 CPUs.

■ Softune REALOS/FR V30L01 Upgrades from Version 03

❍ REALOS/FR has been upgraded to support Softune.

❍ Change from R_user.asm to R_exp.asm

File name R_user.asm is changed to R_exp.asm. Code corresponding to user initialization in the R_user.asm is required for an application program. An init.asm file is attached to the directory smpsys as a sample of user initialization.

❍ Addition of object wait function with a time-out

The following system calls are added:

twai_sem, twai_flg, trcv_msg, tget_blf

(The tget_blf system call is for the fixed-size memorypool function.)

❍ Fixed-size memorypool function addition

The following system calls are added:

get_blf, pget_blf, tget_blf, rel_blf, ref_mpf

- **GHS tool support**

  GHS tools are supported.

- **Output of information about task stack section**

  The configurator outputs information about task stack sections to a temporary file for each task ID.

- **The debugger macro file is no longer supplied.**

  The debugger macro file rdm911.lst is no longer supplied.

- **Analyzer is added.**

■ REV:300001 Upgrades from V30L01/V02

- **Append PDFs same as manuals.**
- **Change Configurator to execute until "lx" at using GHS tools.**

■ REV:300002 Upgrades from REV:300001

- **Corresponded with FRex Family (MB91V300, MB91V360).**
- **The Debug Setting Dialog is added in Configurator of Softune Workbench.**

  And, the screen of Confugurator GUI version was partly modified together.

■ REV:300003 Upgrades from REV:300002

- **The bug fixed that the program cannot through out from OS inside management rarely.**

■ REV:300004 Upgrades from REV:300003

- **Analyzer upgrades to V30L20.Change user interface.**
- **Upgrade help files.**

INDEX

CM71-00320-2E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

FR FAMILY

IN CONFORMANCE WITH µITRON 3.0 SPECIFICATIONS

SOFTUNE REALOS/FR

USER'S GUIDE

August 1999 the second edition

Published **FUJITSU LIMITED** Electronic Devices

Edited Technical Communication Dept.

FUJITSU

CM71-00320-2E

FUJITSU SEMICONDUCTOR FR FAMILY IN CONFORMANCE WITH µITRON 3.0 SPECIFICATIONS SOFTUNE REALOS/FR USER'S GUIDE