FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM42-00325-2E

F²MC-16L/16LX/16/16H/16F

µITRON 2.01 SPECIFICATIONS COMPLIANT

SOFTUNE REALOS/907

USER'S GUIDE

FUJITSU

F²MC-16L/16LX/16/16H/16F

µITRON 2.01 SPECIFICATIONS COMPLIANT

SOFTUNE REALOS/907

USER'S GUIDE

FUJITSU LIMITED

PREFACE

■ Purpose of manual and target readers

Thank you for purchasing Softune REALOS/907 (REALOS/907 in this document).

REALOS/907 is a realtime operating system for Fujitsu's F^{2}MC-16L, 16, 16H, and 16F 16-bit microprocessors. The REALOS/907 kernel conforms to the µITRON 2.01 specifications. As the eventflag function supports both single word eventflags and single bit eventflags, either eventflag function can be selected. This manual uses the following notation to clarify the number of bits.

µITRON 2.01 specification		REALOS/907 manual
Single word eventflag	→	16-bit eventflag
Single bit eventflag	→	1-bit eventflag

This manual contains information required when configuring a REALOS/907 system. The manual describes how to configure and run the system. Refer to this manual for operations relating to the overall system.

The manual also describes the debugger macro commands used when debugging REALOS/907 applications using the Fujitsu debugger.

This manual is intended for engineers who are using REALOS/907 to develop actual products. The manual describes application system development, debugging, and the REALOS/907 libraries. Please take time to read through the manual.

The operations described in this manual assume the following operating environment.

- ❍ Host computer
  - IBM PC/AT compatible (Fujitsu FMV series) with Widows 95/Windows NT 4.0 installed, and with following conditions satisfied:

    Operating environment: Pentium 150 MHz or higher (recommended: Pentium 200 MHz or higher)

    Memory: Windows 95 48 Mbytes or more (recommended: 64 Mbytes or more)

    Windows NT 4.0 48 Mbytes or more (recommended: 64 Mbytes or more)

    Free space on hard disk: At least 50 Mbytes
- ❍ Cross development tools
  - fasm907s Assembler
  - flnk907a Linker
  - flib907s Librarian
  - fs907s Softune Workbench

❍ Trademarks

TRON is an abbreviation of "The Realtime Operating System Nucleus".

ITRON is an abbreviation of "Industrial TRON".

µITRON is an abbreviation of "Micro Industrial TRON".

Softune is a trademark of Fujitsu Limited.

REALOS (REALtime Operating System) is a trademark of Fujitsu Ltd.

F^2MC is an abbreviation of "Fujitsu Flexible Microcontroller" and is a product of Fujitsu Ltd.

MS-DOS is a trademark of Microsoft Corporation.

i486 is a trademark of Intel Corporation.

Windows and Windows NT are registered trademarks of Microsoft Corporation in the United States and other countries.

Other system names and product names in this manual are the trademarks of their respective companies or organizations. The symbols ™ and ® are sometimes omitted in the text.

■ Structure of this manual

This manual consists of the following 4 chapters and an Appendix.

Chapter 1 Notes on Operation

Explains a number of important points on using REALOS/907.

Chapter 2 Developing Application Systems

Describes how to develop application systems using REALOS/907.

Chapter 3 Overview of the Standard I/O Library

Introduces the standard I/O library.

Chapter 4 Program Reference for the Standard I/O Library

Lists standard I/O library details in reference format.

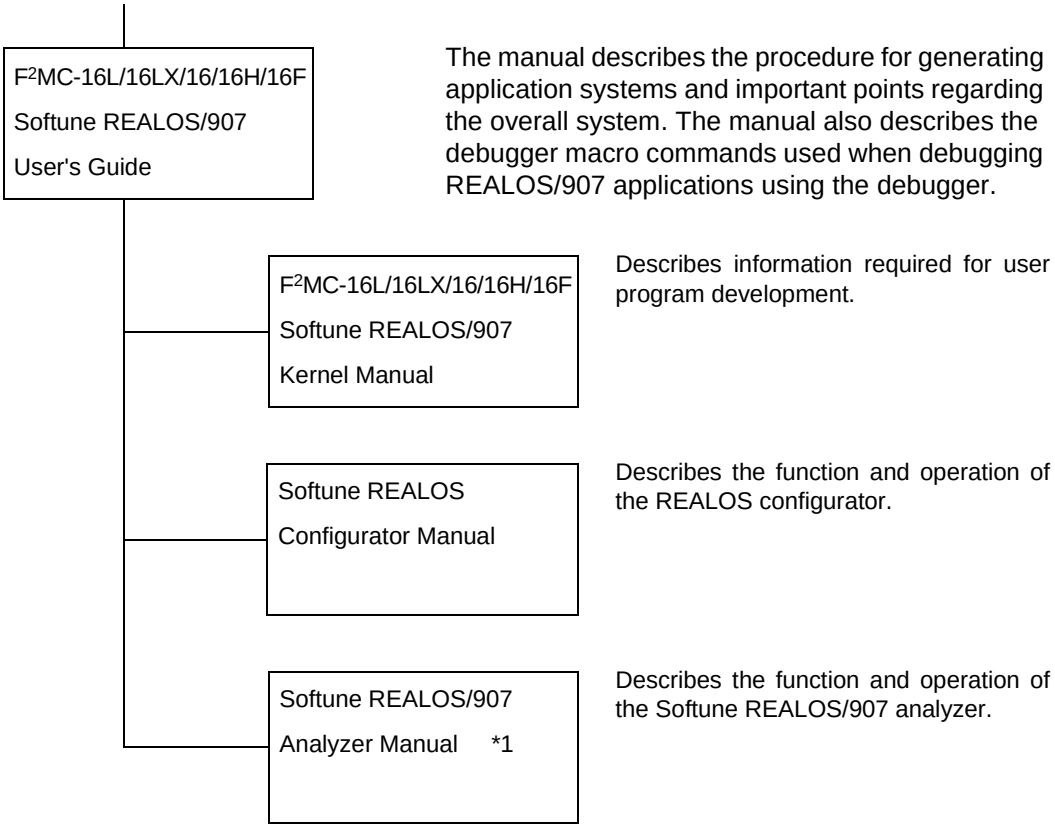
Appendix

Summarizes the REALOS/907 upgrade and describes how to generate and run the sample system.

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 equipment, industrial, communications, and measurement equipment, 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 an inherent chance 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 Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.

REALOS/907 Manual Set

The REALOS/907 manual set consists of the following four manuals. Users who are new to REALOS/907 should read the "F^{2}MC-16L/16LX/16/16H/16F Softune REALOS/907 User's Guide" first.



*1 : Included in the basic (for PC version) version.

USING THIS MANUAL

■ Page layout

In general, each section of this manual is presented on a single page or two facing pages. This allows you to view the contents of a section without having to flip pages.

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

Main section titles are also given in sub-sections to help clarify the context of the section you are currently reading.

■ Meaning of symbols

Note: Special attention is required. Always read these notes.

MEMO

CONTENTS

FIGURES

TABLES

CHAPTER 1 Notes on Operation

This chapter explains a number of important points on using Softune REALOS/907.

1.1 Notes on Programming

Describes important points on programming for Softune REALOS/907.

■ Notes on programming

Note:

- Do not call ext_tsk from task independent portions.

  Calling ext_tsk from a task independent portion generates an error and enters an infinite loop on returning to the task independent portion.

- Do not call ret_int from task portions.

  The previous context may not be able to be restored if ret_int is called from a task portion.

- Call ext_tsk to terminate a task.

  Tasks do not terminate on the ret_int or return instructions (such as the retp and ret assembler instructions or the C language return statement). The program will not function correctly if these instructions are used instead of calling ext_tsk.

- Do not write directly to the following registers and memory areas from within user programs.

  Special registers (PS, SSB, SSP, USB, USP), register memory banks used by the kernel (between 190H and 19fH in the general-purpose register area), delayed interrupt control register, interrupt control register, interrupt vector table, and OS data area.

  These areas are used by the kernel.

- The user does not need to create a program containing the interrupt vector table and OS initial data. These are generated as programs by the configurator during system configuration.

- After a reset, system calls cannot be used before execution of the OS entry routine. The first system call that can be used after the completion of OS initialization is the initialize handler routine.

- The initialize handler routine can only use system calls that can be used in task independent portions. The initialize handler program must specify use of memory areas other than the memory register bank used by the kernel.

- As message passing using the mailboxes is performed by passing pointers, there must be a one-to-one relationship between send and receive operations.

- The message header area is used when passing messages using the mailboxes. Therefore, do not write to the message header after sending the message.

Note:.

- Rigorous error checking is not performed on addresses returned to a memorypool. Therefore, you must store the address of memory reserved from the pool until the memory is released. System operation may be corrupted if the address specified when releasing memory is different to the reserved address.
- Symbol and label names starting with the labels listed below are reserved by the OS. Do not use in user programs.

  “r97_”, “_r97_” (Used in programs)

  “R97_”, “_R97_” (Used in programs)

  “R_”, “r_” (Used in programs)
- The system stack is used by the OS and interrupt program (handlers).
- Except when using a common stack, separate stacks must be provided for each task.
- Task stack areas must be maintained at all times except when the task is in the DORMANT state. Do not manipulate stack areas belonging to other tasks or use as the stack for different tasks.
- All tasks go to the DORMANT state after the OS is initialized. To set tasks to the READY state, tasks must be started (by calling the sta_tsk system call) in the initialize handler or in an interrupt handler.
- Do not use double-byte codes or Japanese characters for a directory name.
- Do not use double-byte codes or Japanese characters for a file name.

1.2 Notes on Hardware

Describes important points relating to the hardware.

■ Notes on hardware

Note:

- Softune REALOS/907 requires a 16-bit microcontroller (F^2MC-16L/16LX/16/16H/16F series) with an internal delayed interrupt module and bit search module.
- One interrupt is required to operate the system clock. Normally, an interval timer is used to generate fixed-period interrupts. An interval timer internal to the MCU can also be used.
- The OS uses the delayed interrupt module. Do not specify a user program in the delayed interrupt vector table.
- The performance of the OS improves if the system stack and kernel management data areas are located in high-speed memory.
- Do not use the internal EI²OS in the microcontroller.

CHAPTER 2 Developing Application Systems

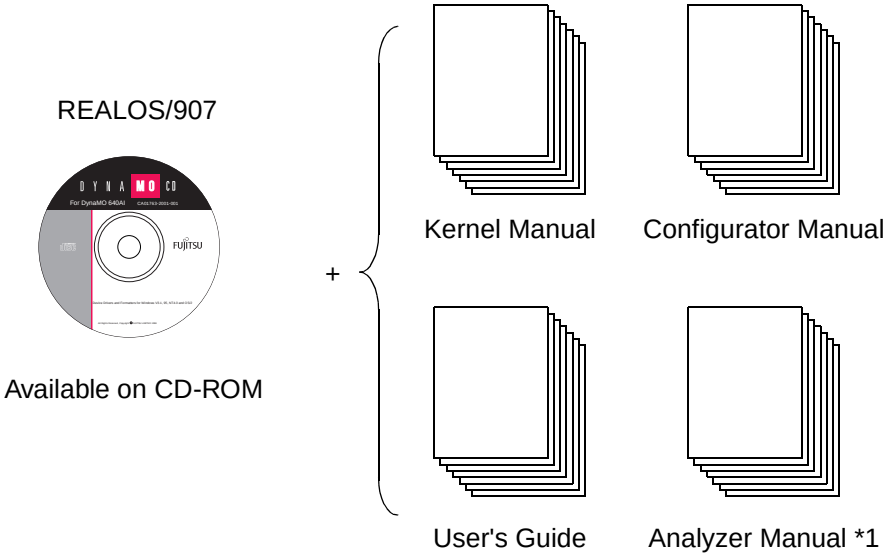
This chapter describes how to develop application systems using Softune REALOS/907.

2.1 REALOS/907 Products

The REALOS/907 products consist of the REALOS/907 realtime OS programs (included in provided media) and the manual set.

■ REALOS/907 products

Figure 2.1 shows the REALOS/907 products.



*1 : Included in the basic (for PC version) version.

Figure 2.1 REALOS/907 Products

■ Manuals

Table 2.1 lists the REALOS/907 manuals.

Table 2.1 REALOS/907 Manuals

Manual	Contents
Kernel Manual	Describes information required for developing user programs. Refer to this manual when developing tasks and handlers.
Configurator Manual	Describes the function and operation of the Softune REALOS/907 configurator. Refer to this manual when configuring an application system.
User's Guide	This manual. The manual describes the procedure for generating application systems and important points regarding the overall system.
Analyzer Manual *1	Describes the function and operation of the Softune REALOS/907 analyzer.

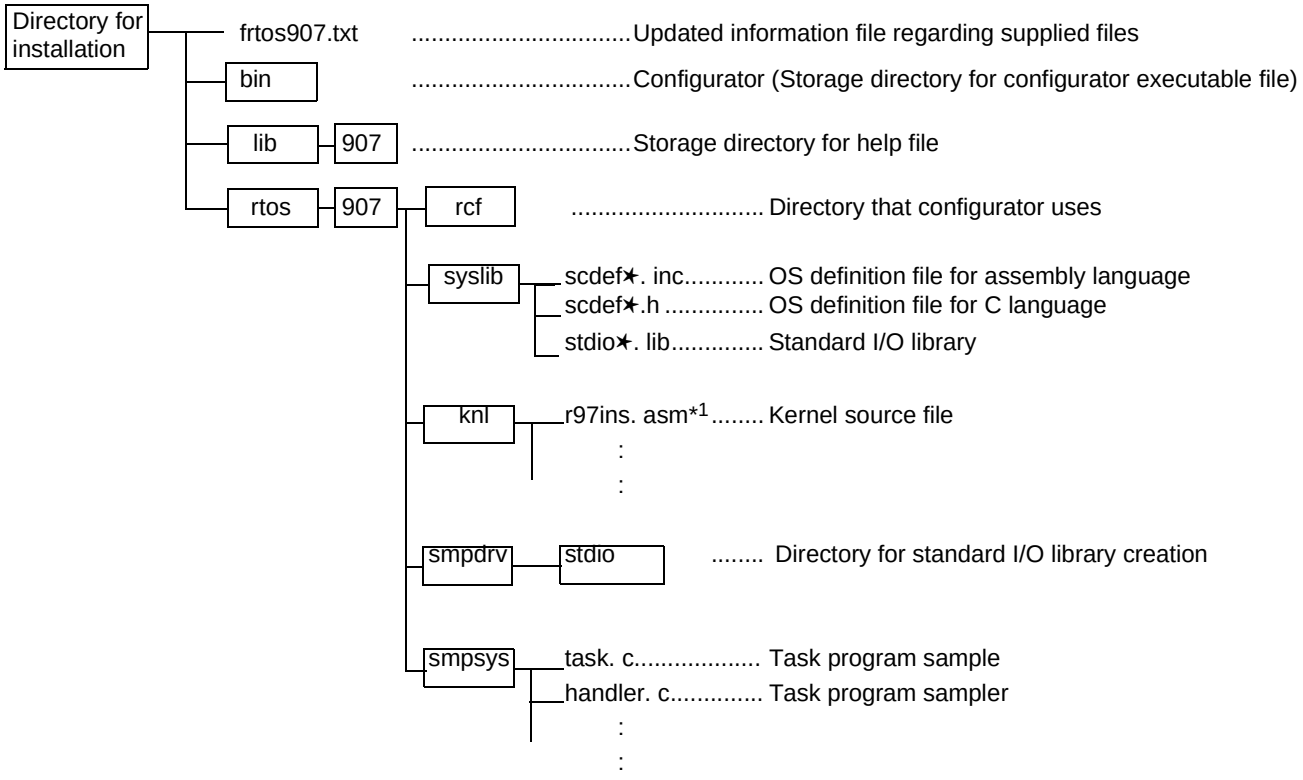
*1 : Included in the basic (for PC version) version.

2.2 Directory Configuration

Figure 2.2 lists the programs that appear in the directory after installation.

■ **Files**

Figure 2.2 shows the structure of the realos directory after installing REALOS/907.



*1: Not included in the evaluation version.

Figure 2.2 REALOS/907 File Structure

2.3 Preparing a Development Environment

REALOS/907 application programs are developed using the cross-development tools and emulator. To develop application programs, you need to install REALOS/907 and the following cross-development support software on the host computer.

■ Support tools required for development

Table 2.3 lists the cross-development tools and emulator required for developing REALOS/907 application programs.

Table 2.3 Support Tools Required for Development

Tool	Tool name
Cross-development tools (Fujitsu)	C compiler (fcc907s) Assembler (fasm907s) Linkage kit: Linker (flnk907s) Librarian (flib907s) Object tool (f2ms, f2is, f2es, m2bs, m2es, m2is, m2ms) Softune Workbench (fs907s)
Emulator (Fujitsu)	MB2140 Series Emulator

2.4 Installing REALOS/907

You must install REALOS/907 on the host computer before you can start developing REALOS/907 application programs.

The following describes the procedure for installing REALOS/907.

■ Installation

Run SETUP.EXE from FRTOS907 of the installation disks on Windows. Follow the instructions displayed by the installation program and specify the directory for installing REALOS/907. The installation program decompresses the files and installs the system under the specified directory. Installing all files requires about 14MB of disk space.

```
D:\Frtos907> SETUP
```

Specify the drive containing the disk in the underlined section.

Note: Do not use double-byte codes or Japanese characters for the installation directory name.

2.5 Programs Required for an Application System

A REALOS/907 application system consists of a program for starting tasks and initializing the system clock, programs for handling normal processing, programs for handling abnormal conditions, and programs for extending system processing.

■ **Application program structure**

Figure 2.5 shows the structure of application programs.

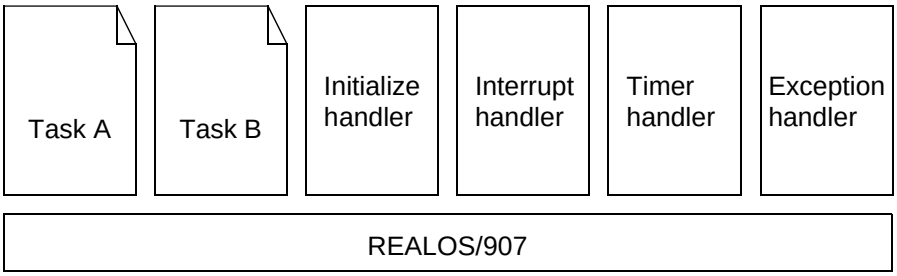


Figure 2.5 Application Program Structure

■ **Program for starting Tasks and initializing the system clock**

❍ Initialize handler

This is the first handler executed after starting the kernel. Performs the initialization required for the application system and starts the initial tasks.

■ **Programs for handling normal processing**

❍ Tasks

Tasks are the basic units of an application program. Application processing is performed by the operation of multiple tasks.

❍ Interrupt handler

Executed on generation of an interrupt. Interrupt handlers are used to receive processing requests from peripheral devices and notify these requests to tasks.

❍ Timer handlers

These consist of cyclic handlers which are activated at fixed intervals and alarm handlers which are activated at a specified time. The handlers are executed as part of the system clock handler.

These handlers can be treated as interrupt handlers.

■ **Programs for handling abnormal conditions**

❍ Exception handlers

Exception handlers are programs activated when an abnormal condition occurs during execution of a task or interrupt handler. An exception handler is executed when a system call exception or CPU exception is generated.

2.6 Development Steps

Table 2.6 lists the steps required to develop an application system for REALOS/907.

Figure 2.6 shows the development procedure for REALOS/907.

■ Development steps

Table 2.6 lists the application development steps and the required software.

Table 2.6 REALOS/907 Development Steps and Required Software

Development Steps	Description	Software
System design	Design the system. Determine the required hardware and whether or not to use REALOS/907 in this step.	-
Select target hardware	Determine the hardware on which the application system is to run. New hardware may need to be developed.	-
Prepare development environment	Configure the REALOS/907 development environment on a host computer.	-
Develop and debug programs	Design and develop the task and handler programs based on the system design specifications Perform independent debugging of tasks and handlers.	C compiler, assembler, linkage kit, simulator debugger
Configure system	Configure the independently developed task and handler programs as a REALOS/907 application program.	Configurator, C compiler, assembler, linkage kit
Run and debug application system	Run the resulting REALOS/907 application system on the target hardware and perform system debugging.	Emulator debugger, Monitor debugger
Load in ROM	Load the REALOS/907 application system into ROM using the ROM loading tools.	Communications program for the ROM programmer, Object tool
Operation	Manage and operate the REALOS/907 application system.	-

■ Development procedure

Figure 2.6 shows the procedure for REALOS/907 development.

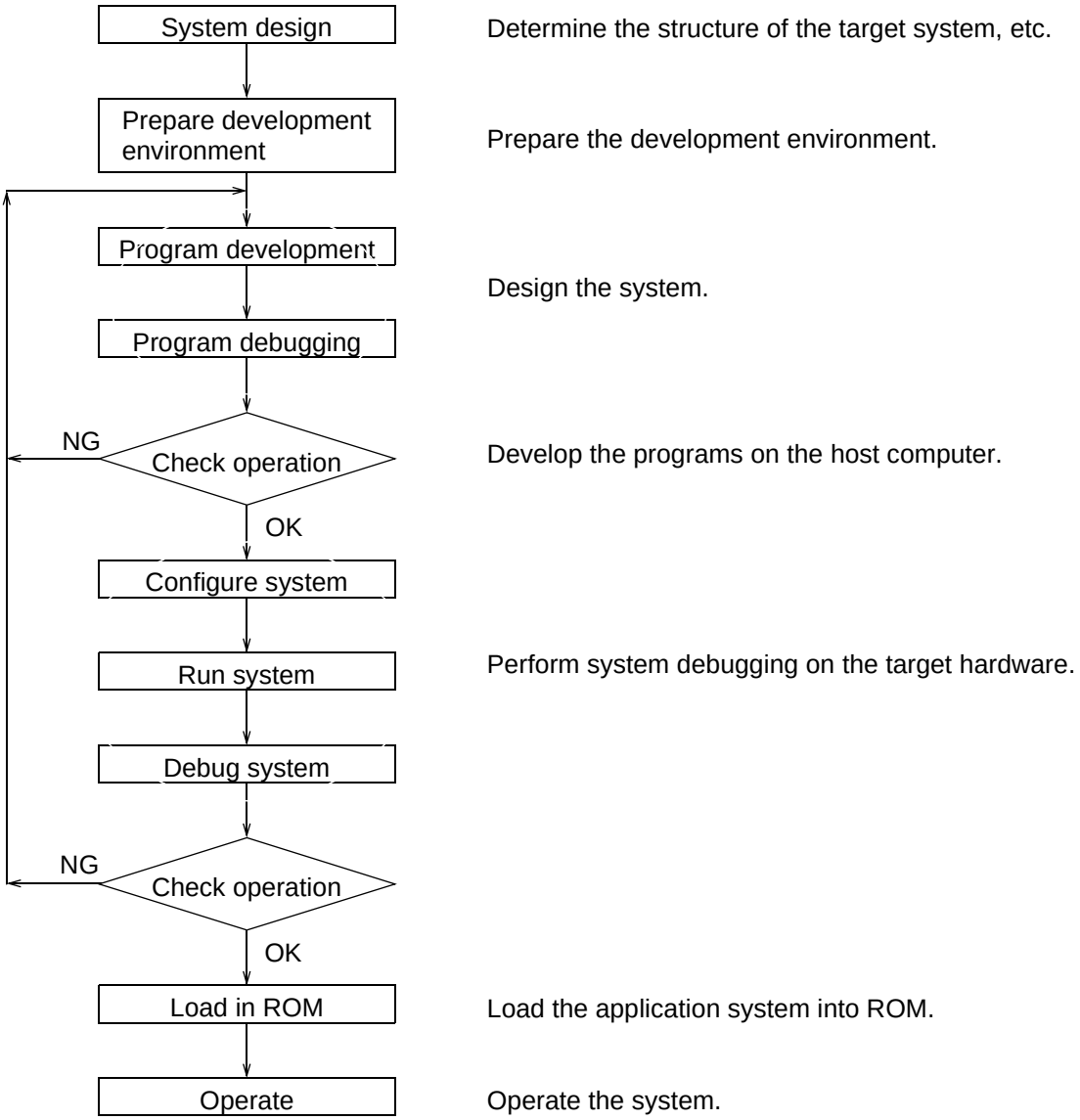


Figure 2.6 REALOS/907 Development Procedure

2.7 Developing User Programs

The following three items are determined in system design.

- **The processing to be executed**
- **The target hardware**
- **Whether or not to use REALOS/907**

Actual development of the REALOS/907 application system starts after the above items have been decided. The following describes the steps for developing an application system.

■ **Split the system functions into tasks and handlers, then develop the user program**

REALOS/907 is a multitasking realtime operating system. The processing to be performed by the application is divided into tasks and handlers. After determining the overall structure, design the individual task and handler programs. Also consider how synchronization and communication will be performed between the tasks and handlers.

To undertake these steps, you need to understand the functions provided by REALOS/907. For more information, see the “Softune REALOS/907 Kernel Manual”.

- Tasks
- Initialize handler
- Interrupt handlers
- Exception handlers (CPU exception handler, system call exception handler)
- Timer handler (cyclic handlers, alarm handlers)

■ **Develop the task and handler programs for each function**

Develop the programs for each task and handler in accordance with the functions and structure specified in the design. Programs can be written in assembler or C.

■ **Determine the objects to use**

Determine the objects to use between the various tasks and handlers. Determine the type, number, initial settings, and other parameters for the objects to be used. Define the objects using the configurator.

See "Softune REALOS Configurator Manual" for details.

- Semaphores
- Eventflags
- Mailboxes
- Memorypools

MEMO

2.8 Developing a User-Defined Module

The kernel module is a user-defined module. You can refer to the sample source files provided and produce a module that fits your specific requirements.

User-defined modules consist of the following.

- **Reset entry routine**
- **Initialize handler**
- **System clock timer interrupt handler**

■ Reset entry routine

The reset entry routine is started by a reset. The routine initializes the processor and initializes those peripheral devices that require initialization after a reset.

REALOS/907 system calls cannot be used in the reset entry routine. The system is not guaranteed to operate correctly if system calls are used.

The entry name of the reset entry routine is specified by the definition statement in the configurator. (Refer to *Softune REALOS Configurator Manual* for more information.)

The reset entry routine performs the following processing.

❍ Initialize the processor

- Set the operating mode
- Set the stack pointer (SP) (When using an external bus, set temporarily in internal RAM until the operating mode is set.)

❍ Initialize the peripheral devices

Initializes those peripheral resources and internal devices that require initialization after a reset.

❍ Pass control to REALOS/907

Passes control to REALOS/907 after completing initialization of the processor and peripheral devices.

Control is passed to REALOS/907 by jumping directly to the REALOS/907 kernel data initialization routine. The r97_entry label is defined for the kernel data initialization routine and is declared globally. Use this label to jump to the routine. REALOS/907 performs initialization of the OS management data and of the delay interrupt.

Initialize handler

The initialize handler is called by the kernel. The handler initializes the devices used by the kernel (such as the timer device used for the system clock) and the devices used in the user program (such as I/O devices).

The initialize handler can use those system calls that can be called from REALOS/907 task independent portions.

The REALOS/907 initialize handler performs the following processing.

- Initialize the interrupt controller

  Initializes the interrupt controller. Initialize the interrupt controller according to the structure of the target hardware.

- Initialize the timer device

  Initializes the interval timer that generates the fixed period interrupts used by the system clock (set the period to suit wai_tsk and the cyclic handler). In general, set a period of 1ms (at CPU operation frequency of 16MHz) or more.

  In the configurator, assign the interval timer interrupt handler to the corresponding interrupt vector number.

- Initialize user devices

  Initialize the devices used by the user program. You can also use this routine to perform system initialization by using software as well as hardware.

- Start tasks

  As all tasks are initially in the DORMANT state, start the initial tasks from this routine.

System clock timer interrupt handler

The timer interrupt handler for the system clock is the system clock interrupt handler used by REALOS/907. The handler performs the timer interrupt processing required by the user target hardware (such as clearing the interrupt) then calls the REALOS/907 system clock processing. The r97_sclk_hdr label is defined for the system clock processing and is declared globally. Use this label to jump to the system clock processing.

In the configurator, assign the label for the system clock timer interrupt handler to the interrupt number corresponding to the timer interrupt on the user target hardware.

2.9 Configuring and Running a System

Generate the system by linking the user program and kernel module using the configurator.

Download the resulting absolute format system to the target system, then run the system. The user tasks start after kernel initialization.

■ Configuring the system

Configure the system using the configurator.

Execute configuration processing after defining the configuration settings. This generates the kernel initialization data and vector table based on these settings. Next, the REALOS/907 kernel module and user program are linked and the absolute format object program is generated for the REALOS/907 application system.

See "Softune REALOS Configurator Manual" for details.

■ Running the System

Download the absolute format object program generated by the configurator to the target system then trigger a reset to start the REALOS/907 application system.

■ **Processing steps from starting the system until the user tasks start**

Figure 2.9 shows the processing steps when REALOS/907 starts.

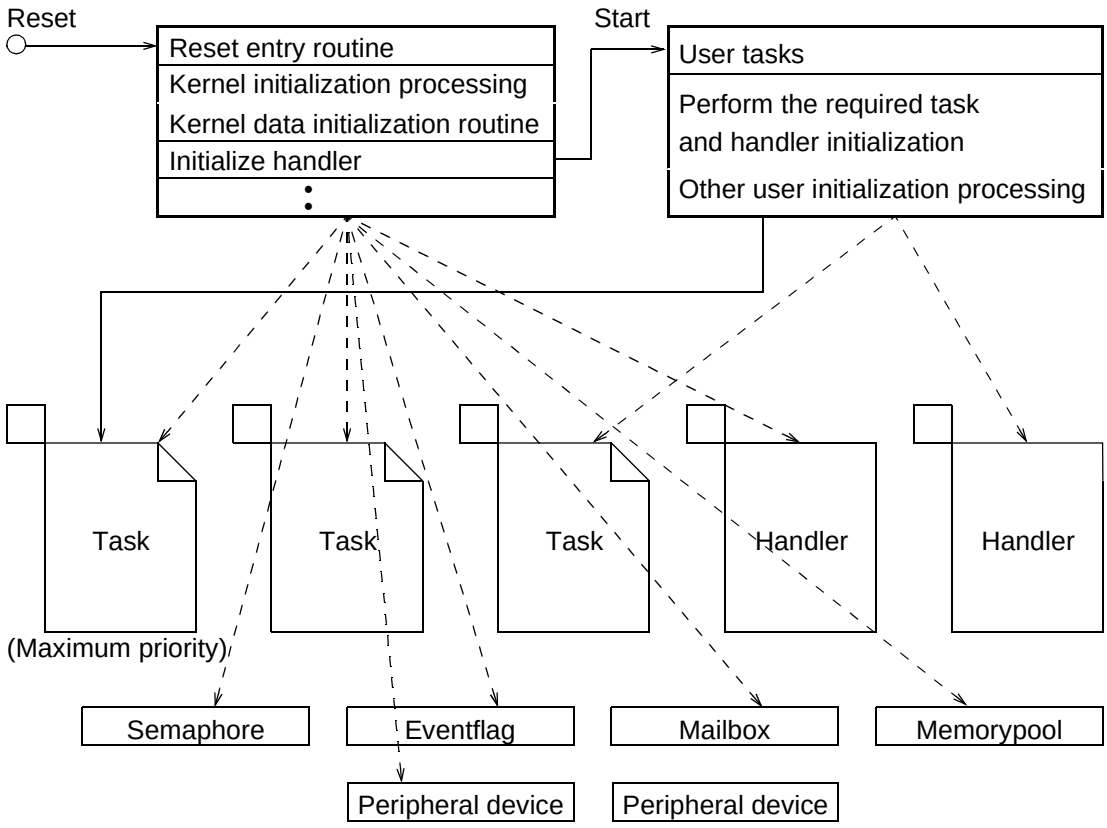


Figure 2.9 Processing Steps When REALOS/907 Starts

MEMO

CHAPTER 3 Overview of the Standard I/O Library

The REALOS/907 system provides a standard I/O library to support development for the user's target system. The standard I/O library uses standard I/O devices such as consoles and provides an interface for data communication and character data I/O (for debug information, for example).

This chapter describes the structure of the standard I/O driver, how to use the driver, and related information.

3.1 Structure of the Standard I/O Library

The standard I/O library consists of the standard text I/O functions putchar, getchar and printf.

■ Structure of the standard I/O library

The standard I/O library consists of the following functions.

❍ Standard I/O functions

Function	Description
putchar	Output a single character.
getchar	Input a single character.
printf	Output a formatted text string.

■ File format of the standard I/O library

The standard I/O library files are located in the syslib directory. Table 3.1a lists the files.

Table 3.1a Standard I/O Library Files

File	Contents
stdiol.lib	Standard I/O Library (Large model)
stdiom.lib	Standard I/O Library (Midium model)
stdioc.lib	Standard I/O Library (Compact model)
stdios.lib	Standard I/O Library (Small mode)

■ Segments used by the standard I/O library

Table 3.1b lists the output segments for the standard I/O library.

Table 3.1b Standard I/O Library Segments

Area	Segment	Segment Attribute	Remarks
Variable data area (including initial values)	DCONST	DSEG	Initial value data
	DINIT	DSEG	Entity
Variable data area (no initial values)	DVAR	DSEG	
Code area	CSEG	CSEG	

3.2 Incorporating the Standard I/O Driver

When using the standard I/O library, you must develop your own code (I/O driver) for the target system. The driver inputs and outputs individual characters using the actual hardware.

■ Incorporating the standard I/O driver

Follow the specifications listed below when creating an I/O driver.

❍ Input driver interface

Item	Description
Input	None
Output	AL: Input character code In case of an error, return EOF (-1).
Registration method	Store the entry address of the input driver (24-bit address) in the four bytes commencing from label “_stdin”.
Register saving	Save all except the AL and AH registers and T, N, Z, V, and C.

❍ Output driver interface

Item	Description
Input	@sp+4 (2 bytes) Output character code
Output	AL: Output character code In case of an error, return EOF (-1).
Registration method	Store the entry address of the output driver (24-bit address) in the four bytes commencing from label "_stdout".
Register saving	Save all except the AL and AH registers and T, N, Z, V, and C.

Programs in the standard I/O library do not issue system calls for exclusive control or waiting on events. If required, issue system calls from the program that calls the function or from the I/O driver.

MEMO

3.3 Example of Creating a Standard I/O Driver

Figure 4.3 shows an example of how to incorporate a standard I/O driver.

■ **Example of creating a standard I/O driver**

```
                    .program   mb89371
                    .title     mb89371
                    #include scdef_s.inc

IO                  .section   IO_mb89371, IO, LOCATE=H'C0
                                          ;MB89371

RS_DATA             .res.b     1          ;h'C0  data register
                    .res.b     1
RS_CTRL             .res.b     1          ;h'C2  control register
                    .res.b     1
RS_BOUD             .res.b     1          ;h'C4  371 baud rate register
                    .res.b     1
RS_CTRL1            .res.b                ;h'C6  371 mode register

CSEG                .section   CODE_mb89371, CODE, ALIGN=2
;---------------------------------------
;   Define entry
;---------------------------------------
                    .export    _stdout, _stdin
_stdout             .data.l    chrout
_stdin              .data.l    chrinp

;-------------------------------------------------
; Serial data transmitter/receiver_iSDTR_FMB89371)
RCLR                .equ       H'40                ; error reset
RMODE               .equ       B'01001111          ; mode (8251 compatible)
RMODE1              .equ       B'01000010          ; mode (8251 compatible)
RBOUD               .equ       B'00000010          ; 9600bps
RCOMM               .equ       B'00010101          ; receive&transmit
RSERR               .equ       B'00111000          ; pattern of error
RXRDY               .equ       1                   ; status bit of receiver
TXRDY               .equ       0                   ; status bit of transmission
UsartAccessDelayTime .equ10                        ; wait time
;-------------------------------------------------
;  _uart_init(MB89371 initialize)
;
;             *PARAMETER*
;                    None
;             *RETURN*
;                    None
;             *ERROR CODE*
;                    None
;-------------------------------------------------
                    .export    _uart_init
_uart_init:
                    mov        i:RS_CTRL, #0       ; MB89371 reset
                    call       delay
                    mov        i:RS_CTRL, #0
                    call       delay
                    mov        i:RS_CTRL, #0
                    call       delay
```

Figure 3.3 Example of Creating a Standard I/O Driver (Continued...)

```
                        mov       i:RS_CTRL, #0
                        call      delay
                        mov       i:RS_CTRL, #RCLR
                        call      delay
                        mov       i:RS_CTRL1, #RMODE1; mode (MB89731 original)
                        call      delay
                        mov       i:RS_BOUD, #RBOUD; baud rate
                        call      delay
                        mov       i:RS_CTRL, #RMODE; mode (8251 compatible)
                        call      delay
                        mov       i:RS_CTRL, #RCOMM; commond
                        call      delay
                        retp
                        EVEN
delay:                  mv        a,#UsartAccessDelayTime
delay01:        decw    a
                        bnz       delay01
                        ret
                        ALIGN 2

;-----------------------------------------------------------------------
;               chrout (output 8bit data )
;
;                *PARAMETER*
;                       R0        output data
;                *RETURN*
;                       None
;                *ERROR CODE*
;                       None
;-----------------------------------------------------------------------
                .export chrout
chrout:
                        bbc       i:RS_CTRL:TXRDY,chrout;   Wait transmit ready
                        mov       a,r0
                        mov       i:RS_DATA,a            ;  Write data
                        retp
                        ALIGN 2

;-----------------------------------------------------------------------
;               chrinp (input 8bit data )
;
;                *PARAMETER*
;                       None
;                *RETURN*
;                       AL        h'0000 - h'00FF : input data
;                                 h'FFFF               : no data
;                *ERROR CODE*
;                       None
;-----------------------------------------------------------------------
                        .export   chrinp
chrinp:
                        bbc       i:RS_CTRL:RXRDY,chrinp_nod; Wait receive ready
                        mov       a,i:RS_DATA                ; Read data
                        retp
chrinp_nod:     movw    a,#h'ffff
                        retp
                        ALIGN 2

                        .end
```

Caution! The above example is for a suppert board for the F²MC-16LX. When using different target board, create a program to match the requirements of the I/O resource being used.

Figure 3.3 Example of Creating a Standard I/O Driver (Continued)

3.4 How to Use the Standard I/O Library

In general, the programs contained in the standard I/O library are written using the C interface. Accordingly, you must use the function execution interface for the C language when using the library from an assembly language program.

When 0ng a function written in C, the calling program pushes the parameters being passed onto the stack. After the function completes, the calling program must restore the stack to its previous level.

■ Program interface

In general, the programs contained in the standard I/O library are written using the C interface. Accordingly, you must use the function execution interface for the C language when using the library from an assembly language program.

When using the standard I/O library from a C program, use the functions in the same way as standard C functions.

■ Function call interface

When executing a function written in C, the calling program pushes the parameters being passed onto the stack. After the function completes, the calling program must restore the stack to its previous level. When there is more than one argument, push the arguments onto the stack in order from the right. For value arguments, push the 2-byte or 4-byte value onto the stack. The number of bytes is determined by the type (two bytes for the char and short (int) types and four bytes for the long type). For address arguments, push the 4-byte address for the large model or 2-byte address for the small model.

The function return value is returned in the AL register.

Figure 4.4a shows an example of calling the "chrcnt = printf("%d %ld \n", 100, 5000L);" function from assembly language. The example is for the large model library.

```
      .import _printf
      .section     CSEG,CODE,ALIGH=2
format .data.b"%d %|d", 10, 0   ; \n = 10, string terminator = 0
        :        :
      movl    a,#5000   ; Push long data (5000) onto stack
      pushw   ah   ;
      pushw   a         ;
      mov     a,#100    ; Push int data (100) onto stack
      pushw   a         :
      movl    a,#format ; Push address of format string onto stack
      pushw   ah        :
      pushw   a         :
      callp   _printf   ; Execute printf. Add _ in front of the entry label
      addsp   #10       ; Release stack
                        ; AL=chrcnt
```

Figure 3.4a Example of Calling the printf Function

■ **Linkage specification**

Add the standard I/O library file to the member when linking applications that use the standard library. Refer to the *Softune Workbench Operation Manual* for the procedure for adding the file to the member.

■ **When using with other libraries provided with the C compiler**

The libraries provided with the C compiler include similar I/O functions. When using the I/O functions provided with REALOS/907, use the librarian to exclude modules with the same name so as not to link the libraries provided with the C compiler by mistake.

Caution! The application will not run if you try to use both the I/O functions provided with REALOS/907 and the I/O functions provided with the C compiler. As the printf module calls the putchar module, always exclude both modules.

Also take care with putchar and getchar as it is possible to call these functions from separate libraries.

MEMO

CHAPTER 4 Program Reference for the Standard I/O Library

This chapter provides a detailed description of three types of standard I/O functions.

4.1 Standard I/O Function Interface Putchar

Outputs one character to the standard I/O device.

■ Putchar

[Interface**]**

Format	int ercd = putchar(int chrcd);		
Parameter	chrcd	:Output character code	
Return parameter	ercd	:Output character code	:Normal completion
		EOF	:Error

[Description**]**

Outputs one character to the standard I/O device. The function returns the output character if the character is output normally. If an error occurs, the function returns EOF (-1).

[Examples**]**

❍ Assembler interface

```
#include scdef.inc
.import _putchar

mov     a, "*"
pushw   a
callp   _putchar
popw    a
```

❍ C interface

```
#include "scdef.h"
prog()
{
  putchar('*');
}
```

4.2 Standard I/O Function Interface Getchar

Inputs one character from the standard I/O device.

■ Getchar

[Interface**]**

Format	int chrcd = getchar();	
Parameter	None	
Return parameter	chrcd	:Input code
	EOF	:Error

[Description**]**

Inputs one character from the standard I/O device. The function returns the input character if input completes normally. If an error occurs, the function returns EOF (-1).

Whether or not getchar echoes input characters and whether or not it waits for key input are determined by the operation of the input driver.

[Examples**]**

❍ Assembler interface

```
#include scdef.inc
.import_getchar

callp_getchar
cbne a,#h'1a,endcode
     :    :
```

❍ C interface

```
#include "scdef.h"
prog()
{
   int c ;
   c = getchar() ;
   :  :
}
```

4.3 Standard I/O Function Interface Printf

Outputs a character string to the standard output device in accordance with the specified format.

■ Printf

[Interface**]**

Format	int chrcnt = printf(B *format [,data1 [,data2 ...]]);	
Parameters	format	:Pointer to the output format string
	data1, ...	:Output data
Return parameter	chrcnt	:Number of characters output

[Description**]**

Outputs a character string to the standard output device in accordance with the specified format.

The control specifiers in the output format string specify the type and output format of the parameters following the format parameter. Characters other than control specifiers in the format are output directly to the standard I/O device. Control specifiers start with the percent (%) character and specify the format as follows.

%[control][field][l]type

❍ control: A control character that specifies the output format. The following can be specified.

Control Character	Format	Processing When Omitted
-	Left justify	Right justify
+	Output the sign	No sign
#	Output the radix	No radix
␣	Add leading spaces	No leading spaces

❍ field..... A numeric value that specifies the number of characters to output in the output data. The output data is not truncated if it requires more than the specified number of characters. If the number starts with a zero (0), the leading spaces of the output data are filled with zeros.

If the number of output characters is omitted, the output data is output in the required number of characters.

❍ l..... Indicates that the type of the corresponding data is long (4-byte data). Such data must be pushed onto the stack as four bytes (for a C program, the data type must be long). If omitted, the stack data is treated as 2-byte data.

❍ type..... A control character that specifies the type of the output data. The following types are available.

Control Character	Type
d	Signed decimal
u	Unsigned decimal
o	Octal
x	Hexadecimal (lower case letters)
X	Hexadecimal (upper case letters)
c	Character
s	Character string

Use "%%" to specify a "%" character as an output character rather than as a control specifier.

The printf function outputs characters using putchar.

The printf function uses approximately 80 bytes of stack as a work area (excluding the arguments and stack used by putchar).

[Examples**]**

- Assembler interface

```
        #include scdef.inc
        .import_printf

string  .data.b"data=%02X",10,0

        mov    a,dtb
        pushw  a
        movl   a,#string
        pushw  ah
        pushw  a
        callp  _printf
        addsp  #6
               :    :
```

- C interface

```
        #include "scdef.h"
        prog()
        {
           int c ;

           printf("data=%d\n",data);
           :    :
        }
```

MEMO

Appendix

The appendices summarize the REALOS/907 upgrade and describe how to generate and run the sample system.

Appendix E Program Description

This appendix describes the nine programs provided by REALOS/907.

- **Information on supplied files**
- **Configurator**
- **Softune REALOS configurator add-in**
- **Kernel basic module**
- **User-created module**
- **Include file for users**
- **Object file for OS**
- **Standard I/O library (subset)**
- **Sample system**

■ **Information on supplied files**

Information on the supplied files is provided in the following file:

❍ frtos907.txt: Information on the supplied files

frtos907.txt is the file containing information on REALOS/907 products.

■ **Configurator**

The following configurator is provided:

❍ frcfs.exe : DOS prompt version of the configurator

■ **Softune REALOS configurator add-in**

REALOS provides the Softune REALOS configurator as an add-in to Softune.

The following files are supplied:

❍ SIRCF.DLL: configurator DLL (Japanese)

❍ SIRCFENU.DLL: configurator DLL (English)

■ **Kernel basic module**

The following are supplied in the kernel basic module:

❍ r97ins.asm and other files: Kernel source (basic version only)

❍ rcf.lib and other files: Kernel library (evaluation version only)

These are the basic part of REALOS/907 for controlling the execution of a task conforming to the µITRON2.01 specification and the execution of the handler.

These are the support tools to be used for constructing a REALOS/907 application system.

Refer to the *Softune REALOS Configurator Manual* for more information.

■ **User-created module**

The following module is supplied as a user-created module:

- ❍ Reset entry routine

Processes that depend on the CPU or the target structure, and the initialization process for C programs are described in 1st.asm (1st2.asm), which is provided as a sample. The following processes are included:

- ❍ Reset entry routine
- ❍ C program initialization routine
- ❍ Initialize handler
- ❍ Low level I/O routine

■ **Include file for users**

The following four files are supplied in the include file for users:

- ❍ scdef_b.h: Header file for system calls

  (For 1-bit eventflag)
- ❍ scdef_w.h: Header file for system calls

  (For 16-bit eventflag)
- ❍ scdef_s.inc: Include file for system calls

  (For F²MC-16L/16LX/16/16H family)
- ❍ scdef_f.inc: Include file for system calls

  (For F²MC-16F family)

There are definitions that describe system calls.

■ **Object file for OS**

The following OS object file is supplied:

- ❍ dbgfk.obj: Object file for OS

This file must be registered in a member.

■ **Standard I/O library (subset)**

The following four files are supplied in the standard I/O library.

- ❍ stdiol.lib: For large C model
- ❍ stdiom.lib: For medium-size C model
- ❍ stdioc.lib: For compact C model
- ❍ stdios.lib: For small C model

These are the library for printf, putchar, and getchar.

■ **Sample system**

The following two project files are provided in the sample system:

- ❍ smp1.prj: Project file of sample system 1 for Workbench

- smp2.prj: Project file of sample system 1 for Workbench

  These are the project files for constructing sample tasks and their system.

  Refer to Appendix B, "Running the Sample system" for the details.

Appendix F Running the Sample System

REALOS/907 includes a sample program that you can refer to when developing user programs.

■ Sample system

The aim of the sample system is to show the basic structure of application programs running on REALOS/907. The sample system contains some simple processing that uses basic REALOS/907 functions. The system does not perform any particularly meaningful operations. The emulator debugger and support board are required to view the operation of the sample system. This appendix describes an example of the sample system that runs under Windows.

■ Configuration of the sample system

Configure the sample system by making the connections and settings shown in Figure Ba below.

Instead of Hyperterm.exe, you can use any other communications software that can send and receive ASCII characters.

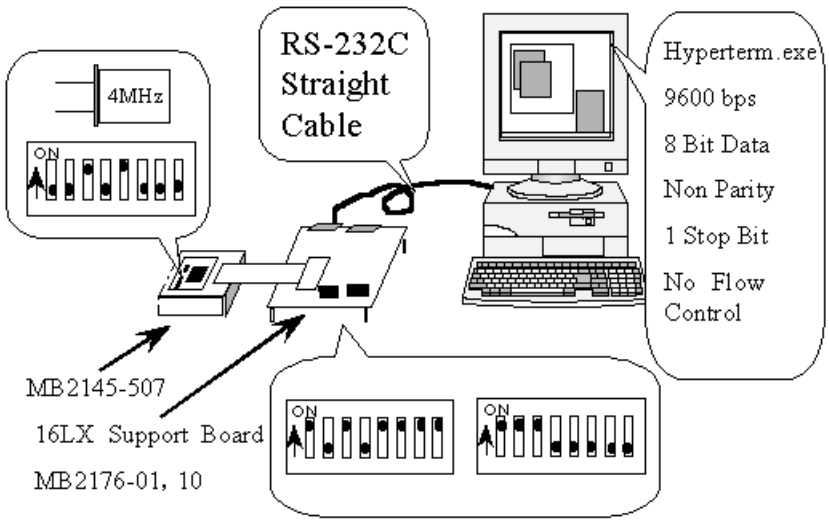
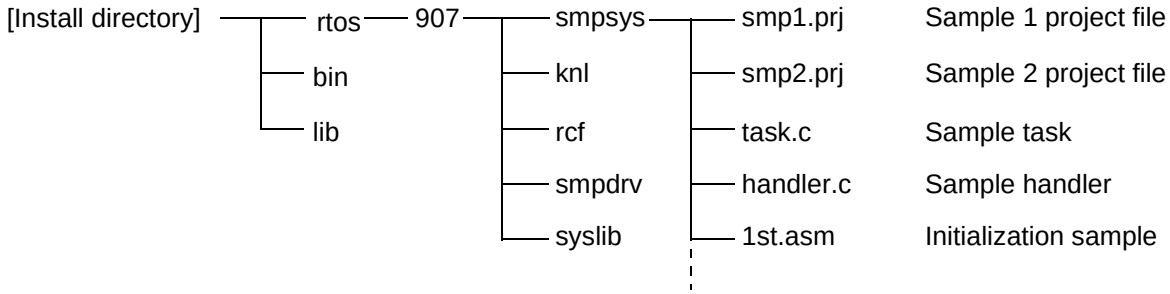


Figure Ba Sample System Environment

There are two sample programs provided: sample 1 using the internal serial interface (UART) for the MB90550 and sample 2 for the serial interface (MB89371) mounted on the support board. The sample programs are stored in the rtos\907\smpsys directory under the REALOS/907 install directory

.



■ Operations of Sample Programs

Figure Bb shows the sample program listings.

The behaviors of samples 1 and 2 are slightly different. This is because, while the serial driver in sample 1 is implemented based on interruption, the one in sample 2 is implemented based on polling. The remainder of this appendix applies mainly to sample 1 whose operation is more difficult to understand than sample 2.

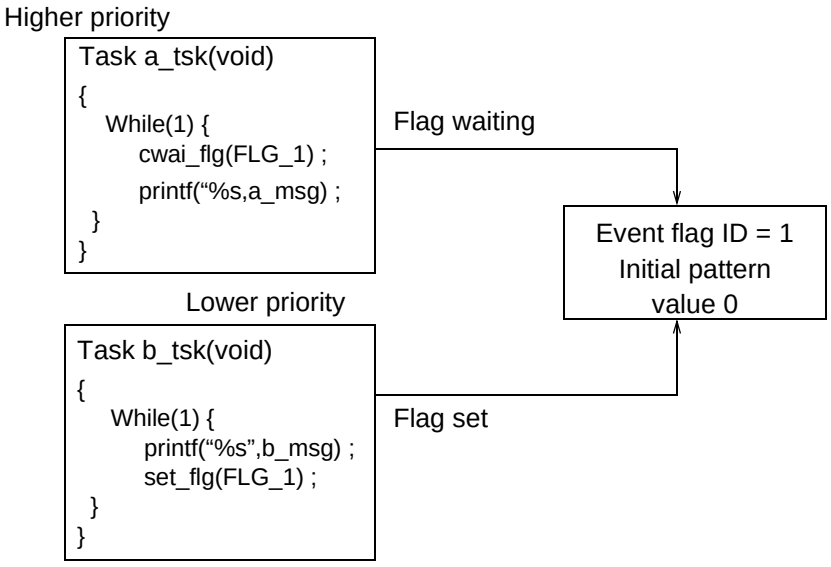


Figure Bb Sample Programs

During sample program operation, the tasks are executed alternately depending on the ID1 eventflag as shown in Figure Bc.

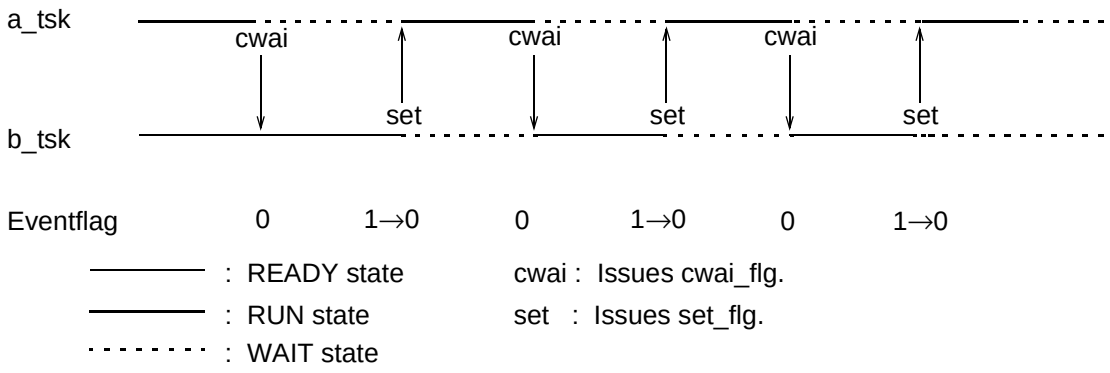


Figure Bc Sample Program Operation (1)

■ Sample system project file

Two types of sample programs are provided for the 16LX support board.

❍ smp1.prj

The internal MB90V550 UART and internal 16 bit timer are used. Two methods are described for the UART (uart.asm). The first uses polling for data communication when there is a call from the interrupt handler. The second uses interrupts for data communication for a call from a task. The internal 16 bit timer (timer16.asm) is used for the interval interrupt for the system clock count.

When a simulator is used, each task is switching to the standby state in order unless simulation of the UART send enable interrupt has been set.

The sample program is recommended for operation with the emulator.

❍ smp2.prj

The MB89371AH serial controller on the 16LX support board and MB89254H timer are used.

MB89371AH (mb89371.asm) communicates using the polling method. The MB89254H (mb89254.asm) is used as the interval interrupt for the system lock count.

This sample operation is easily checked with the simulator. If the simulator is used for checking, address C0h can be used for output to the terminal when the I/O port is set up.

■ Sample activation

The sample system activation with Softune Workbench is described below.

❍ Open project file

Select [Open project...] from [File] menu. The [Open Project] dialog box opens.

The [Open Project] dialog box is shown in Figure Bd.

When the directory containing the sample system is accessed, two project files are displayed. Select smp1.prj, then click the [Open] button.

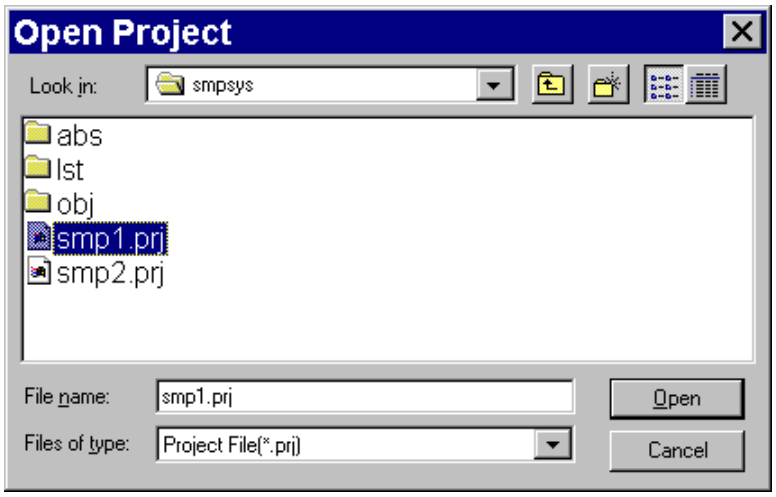


Figure Bd [Open Project] dialog box

❍ Checking the configuration file

To check or change the contents of the configuration file, double-click smp1.rcf, which is registered as a project member. The [Set Configuration File] dialog box opens.

The project members are listed as shown in Figure Be.

Figure Bf shows the [Set Configuration File] dialog box.

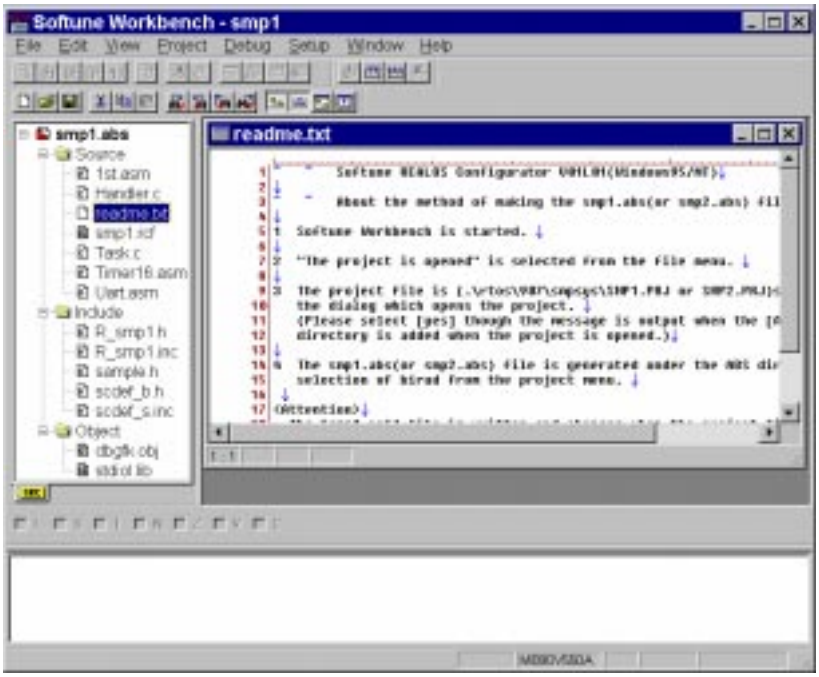


Figure Be Project Member List

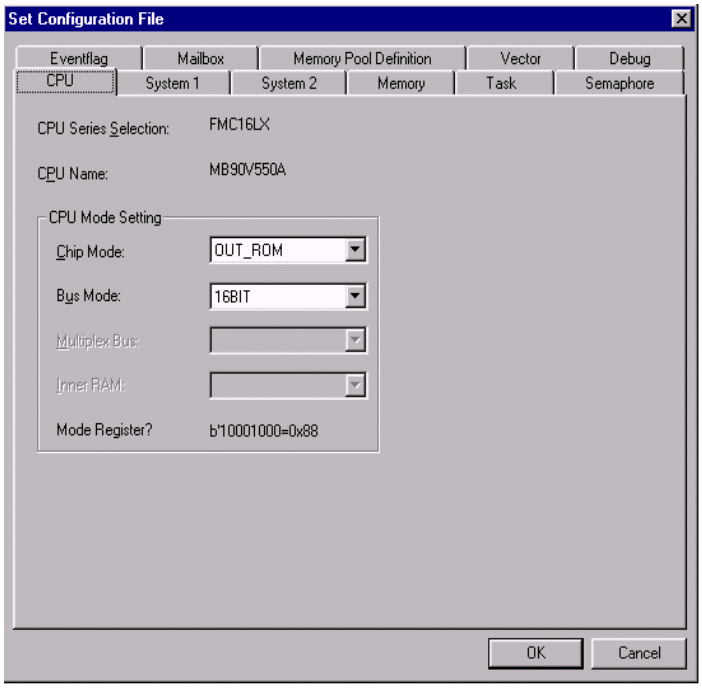


Figure Bf [Set Configuration File] Dialog Box

❍ Creating a load module file

Create a load module file (absolute file). Select [Build] from the [Project] menu to execute the build.

When the build is complete, the completion message shown in Figure Bg is displayed in the output window.

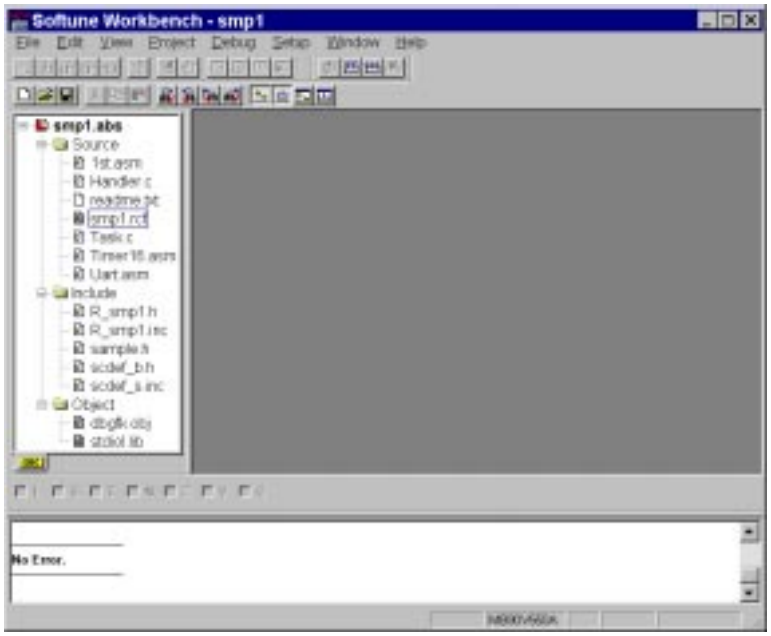


Figure Bg Status when Build Complete

❍ Checking the memory map

To check the memory map, select smp1.abs, then right-click to select, from the context menu, [Open List File] -- [smp1.mp1].

Figure Bh shows the [Open List File] dialog.

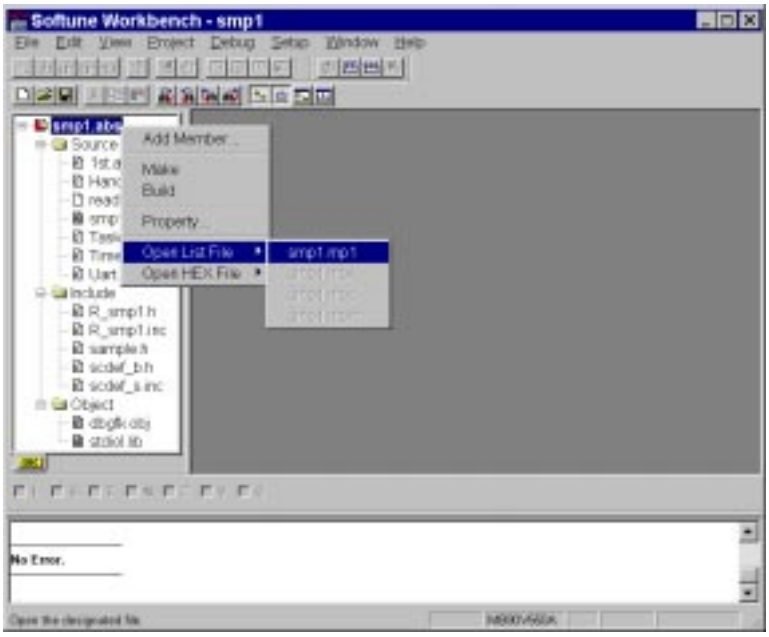


Figure Bh [Open List File]

❍ Debugger activation

Activate the debugger and load the load module.

Select [Start Debug] from the [Debug] menu to switching to debugging mode. The debugger setup wizard starts. Using the debugger manual, set up the debugger.

Note:

- The setup wizard appears only the first time debugging is started.
- An example of debugging with the emulator debugger is described here.
- Debugging with the emulator debugger requires the 16LX evaluation board, a daughterboard, and an ICE (MB2140 series).
- A memory map must have been specified for the emulator debugger.

Select [Load target file] from the [Debug] menu to load the sample program if it has not already been loaded.

Then, select [Reset of MCU] from the [Debug] menu. Preparations for sample system activation are complete.

An example of breaking at the beginning of task 1 (a_tsk) is described below.

Double-click task.c in the project window to open the source window. Then, click ❍ on the first line of a_tsk as shown in Figure Bi to set the breakpoint.

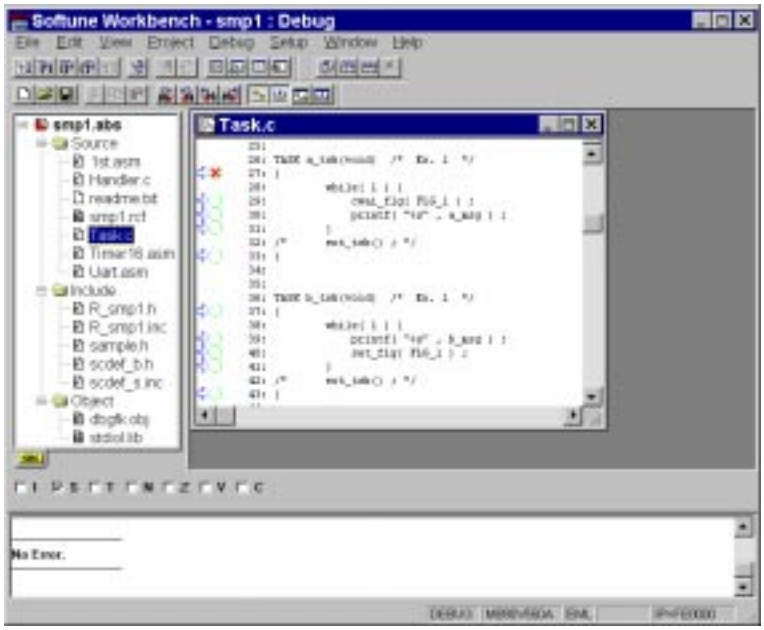


Figure Bi Setting a Breakpoint

When [Run] -- [Go] is selected from the [Debug] menu, a break occurs at a_tsk.

Note: When execution is with the simulator, the operation continues, since the I/O status in A1h does not change. Change the I/O status from the debugger to advance the program.
Select [Watch] from the [View] menu to open the watch window. In the watch window, right-click to select [Set..] from the context menu to open watch setup. Set the variable to h'A1 and click the OK button.
In the watch window, right-click to select [Edit...] from the context menu to open variable editing. Change the value to h'8B and click the [OK] button. As soon as OK button is clicked, a break occurs at a_tsk.

The status when the break occurs is shown in Figure Bj.

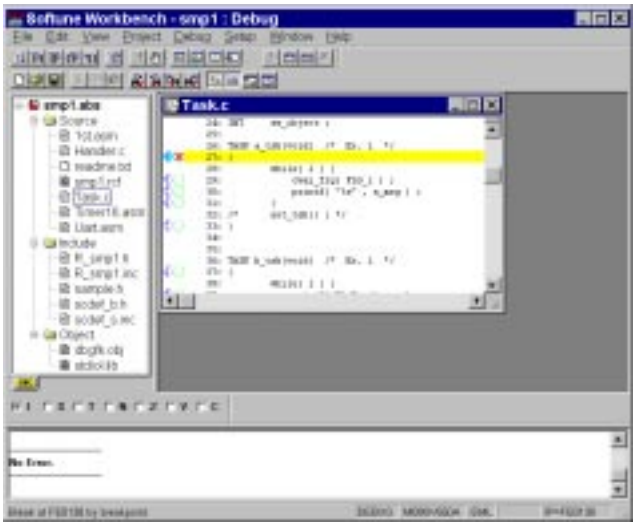


Figure Bj Break at a_tsk

■ Executing the sample system

Execution breakpoints are expressed as bp1 and bp2, as illustrated in Figure Bk.

(1) Run the sample system to the point immediately before cwai_flg of a_tsk (hereafter expressed as the point immediately before bp1). Run the sample system by clicking on the bp1 point shown in Figure Bk).

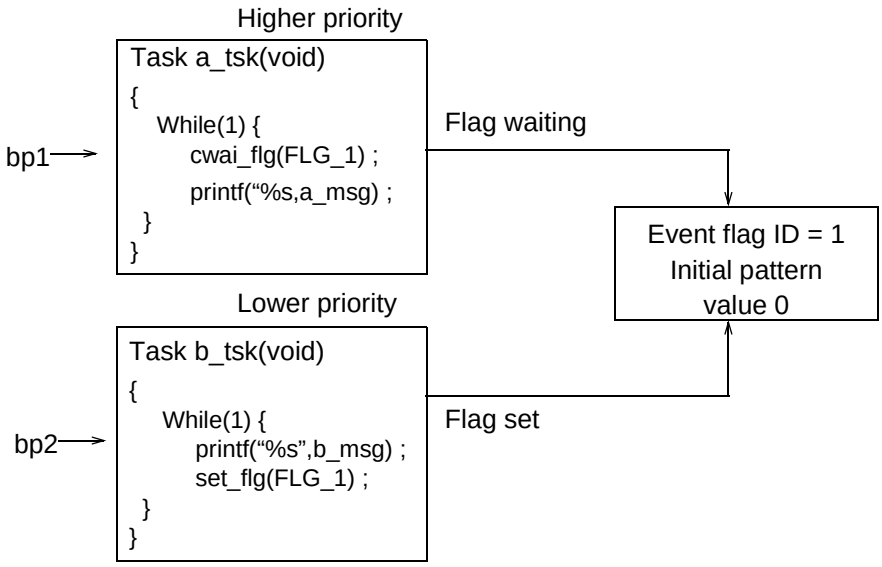


Figure Bk Execution Breakpoints

(2) Run the sample system to the point immediately before set_flg of b_tsk (hereafter expressed as the point immediately before bp2). Run the sample system by clicking on the bp2 point shown in Figure Bk). HyperTerminal will then display " hello, i am b_task. "

(3) Run the sample system to the point immediately before bp1 again. HyperTerminal will then display " HE Lh Le01,1 oI, AiM aAm_ TbA_StKa.s ".

(4) Run the sample system to the point immediately before bp2 again. HyperTerminal will then display "k. ".

(5) Repeat steps (3) and (4) above for output operation.

The data output by each task is set as follows:

`a_tsk:` " HELLO, I AM A_TASK. "

`b_tsk:` " hello, i am b_task. "

In step (5), the tasks look as if they are performing output operation alternately as shown in Figure Bl.

```
cwai_flg(FLG_1);      /* a_tsk */
printf("%s", b_msg);  /* b_tsk */
set_flg(FLG_1);       /* a_tsk */
printf("%s", b_msg);  /* b_tsk */
cwai_flg(FLG_1);      /* a_tsk */
```

Figure Bl Task Operation

However, HyperTerminal displays strings as shown in Figure Bm. From this output, the tasks do not seem to be working as in Figure Bl.

```
hello,i am b_task. HE LhLe01,1 oI, AiM aAm_TbA_Stka.s k.
HE LhLe01,1 oI, AiM aAm_TbA_StKa.s k.HE LhLe01, 1 oI, AiM a
Am_TbA_TbA_StKa.s k.
```

Figure Bm Output at HyperTerminal

This is because the serial driver is implemented as described below. The serial driver in sample program 1 invokes the interrupt handler on occurrence of a transmission enable interrupt and uses the eventflag to notify the task of completion of preparation for output. That is, when the printf function is execution, the cwai_flg system call is issued each time a character is output. Figure Bn below illustrates this situation, focusing on a_tsk and b_tsk.

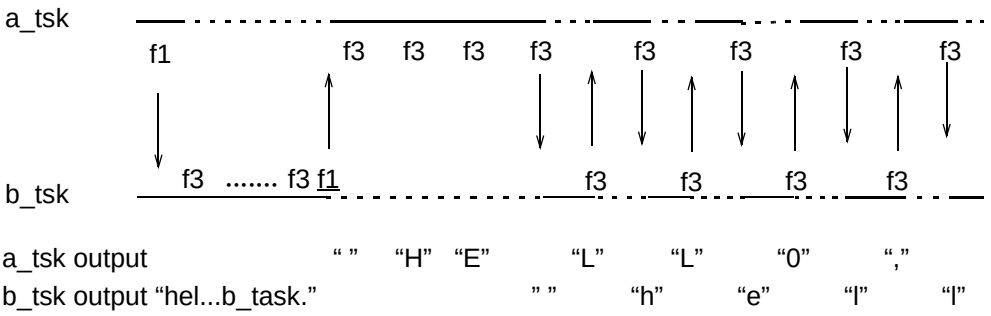
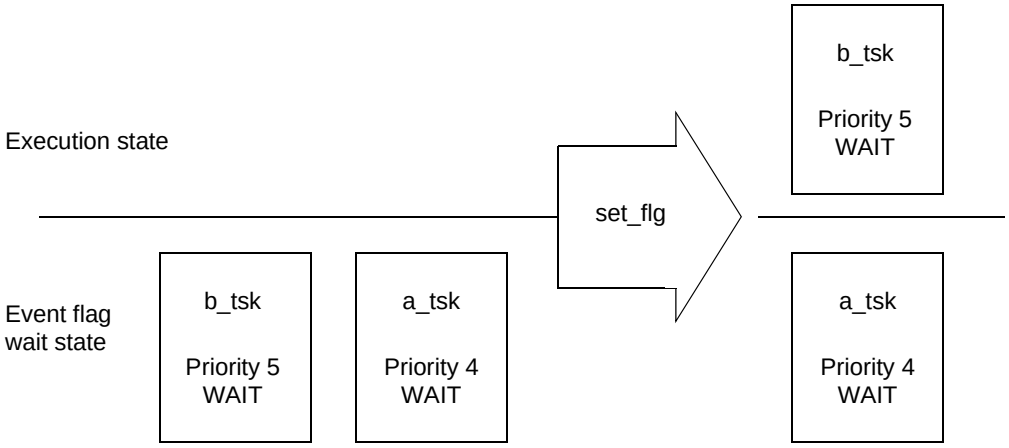


Figure Bn Sample Program Operation (2)

Even though a_tsk has a higher task priority than b_tsk, the tasks work as shown in Figure Bi depending on how the drivers have been designed. This is because, although the ID3 eventflag places the two tasks different in task priority in the wait state at the same time, the tasks are scheduled by the FCFS (First Come First Service) system. If a_tsk enters the wait state after b_tsk enters the wait state, b_tsk restarts working first when the conditions for releasing the tasks from the wait state are satisfied. This symptom is called priority inversion.

a_tsk outputs a series of three characters " HE " in Figure Bn because the serial driver has a single-character buffer.



When only one of the two tasks waiting for the same object is released from the wait state, the FCFS causes the task with a lower priority to enter the execution state.

Figure Bo Priority Inversion

Since the serial driver in sample 2 is implemented based on polling, is does not cause priority inversion. During the serial device transmission wait time, however, only loop processing is executed, preventing any other task from being executed. The transmission rate of the serial interface has been set to 9600 pbs, wasting about 1 ms as the character output interval when characters are output continuously. Since the sample programs require set_flg and cwa_flg processing time of about 62 μs in total, allowing for task switching for processing.

The sample programs handle only two processing tasks, a_tsk and b_tsk. The programs are meaningless if the tasks result in mixed output. That is why the serial driver based on polling is acceptable in the same program. If any other processing task is executed, however, the serial driver based on polling cannot use remaining time efficiently. (If the driver processing speed is so fast that task switching takes longer for processing. the driver must be based on polling.)

To practice programming, use sample 1 to write a program which uses a system call with tasks and handlers. Also, try writing a program which does not cause priority inversion without changing the serial driver. Note that there are more than one solution. If your program is a solution, the character string output by each task will appear continuous as shown in figure Bp or Bq.

```
HELLO, I AM A_TASK. hello, i am b_task. HELLO, I AM A_TASK.
hello, i am b_task. HELLO, I AM A_TASK. hello, i am b_task. H
ELLO, I AM A_TASK. hello, i am b_task. ...
```

Figure Bp HyperTerminal Output Example (1)

```
hello, i am b_task. HELLO, I AM A_TASK. hello, i am b_task.
hello, i am b_task. HELLO, I AM A_TASK. hello, i am b_task. h
ello, i am b_task. HELLO, I AM A_TASK. ...
```

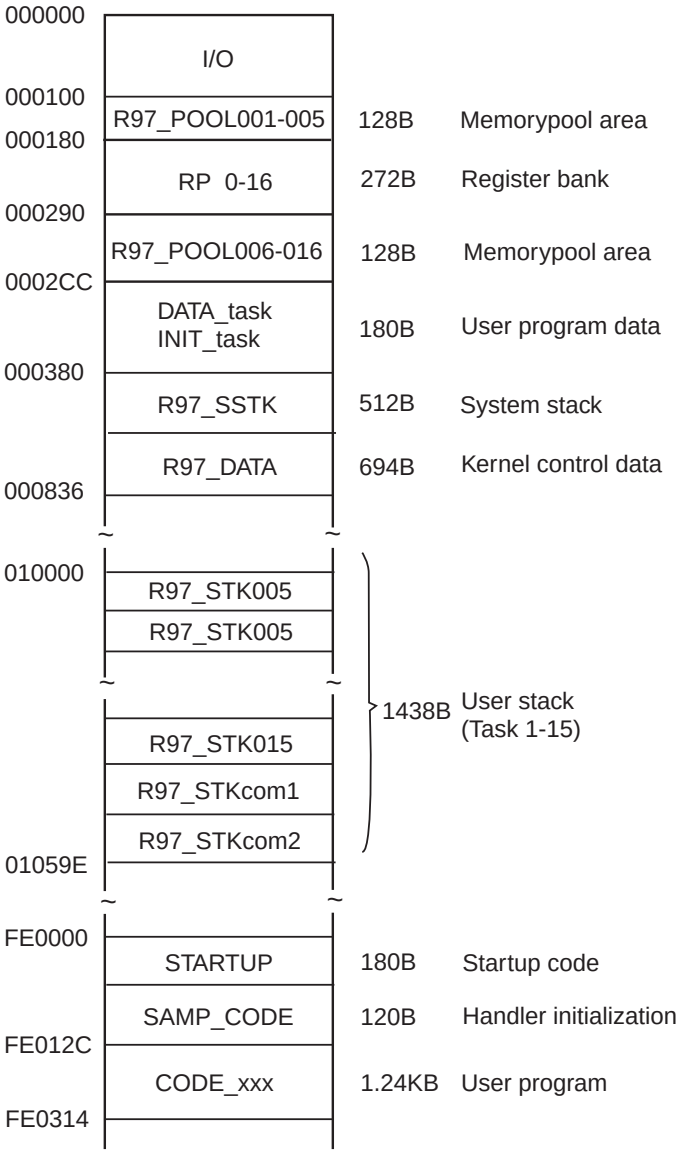
Figure Bq HyperTerminal Output Example (2)

Appendix C Sample System Memory Map

The memory capacity required for the sample system is shown below. This must be used for a reference for the memory capacity required for REALOS/907 system operation.

■ Sample system memory map

Figures Ca and Cb show the sample system memory maps.



Note: Refer to Appendix B, “Kernel Code and Kernel Data,” in the S*oftune REALOS Configurator Manua*l for the procedure for allocating kernel data.

Figure Ca Sample System Memory Map

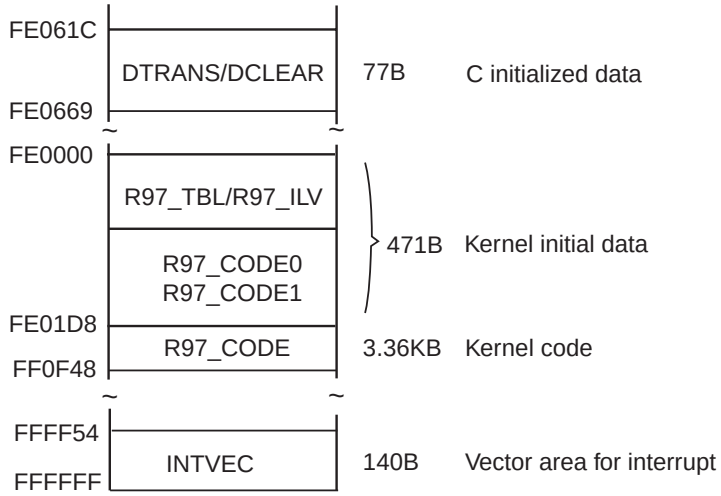


Figure Cb Sample System Memory Map (Continued)

Appendix D Summary of the REALOS/907 Upgrade

This appendix lists the function enhancements and modifications for each REALOS/907 upgrade.

■ Modifications for the V01 to V02 upgrade

❍ Support for the entire F^{2}MC-16L/16/16H/16F series

Object code can be output (in library format) for either the F^{2}MC-16L/16/16H series or F^{2}MC-16F series by making a selection in the kernel generation tool.

❍ Increased maximum number of tasks

The maximum number of tasks that can be registered is increased to 255.

Object Name	V01	V02
Maximum number of tasks	127	255
Maximum task priority levels	7	←
Maximum number of eventflags	127	←
Maximum number of semaphores	127	←
Maximum number of mailboxes	127	←
Maximum number of memorypools	127	←

❍ 16-bit eventflag function added

The 16-bit eventflag function was added to the previous 1-bit eventflag function. The kernel generation tool selects which eventflag function to use.

- Improved system call exception function

  The system call exception is a debugging function that, when selected, calls the system call exception handler when a system call returns an error. However, depending on the system, some returned error codes represent normal operation rather than an error. To allow the system call exception function to be used in such systems, calling of the system call exception handler can be disabled for timeout and polling-failed errors.

  This function is selected by specifying TYPE2 when selecting the system call exception function in the kernel generation tool. Specifying TYPE1 selects the previous function.

- Changes to C type declarations

  The data length of the types listed below have changed from 8-bit to 16-bit.

  This change in the OS type declarations prevents OS definition data structures from having odd-numbered sizes.

  To operate F^{2}MC-16L/16/16H/16F series chips at high speed, multi-byte data must be located at even-numbered addresses. However, structures defined for the OS in V01 have some structure elements located at odd-numbered addresses. This results in multi-byte data being unable to be located at even-numbered addresses, even when the user application is coded optimally.

  The changes to the type declarations in V02 enable all multi-byte structure elements in data structures defined by the OS to be located at even-numbered addresses*1. Also, as data structure sizes are always an even number of bytes*2, programs can be optimally coded without needing to take account of the internal structure of OS-defined structures.

*1: When the structure is located at an even-numbered address.

*2: Excludes the T_MSG structure as the message size in this structure is defined temporarily.

Type	V01	V02
ID	B	H
HNO	B	H
FPTN	UB	UH
SCNT	UB	UH

■ Modifications for the V02 to V03L01 upgrade

❍ Configurator support

V03 uses the configurator in place of the previous kernel generation tool to configure an optimized kernel for the system.

The configurator creates a file containing the initial data required by the OS based on the specified parameters and links this file with the object file created by the user. By specifying the size of task-specific stack areas, the areas are reserved and registered in the OS initial data.

❍ Increased maximum number of task priorities

The maximum number of task priorities that can be registered is increased to 16.

Object Name	V02	V03
Maximum number of tasks	255	←
Maximum task priority levels	7	16
Maximum number of eventflags	127	←
Maximum number of semaphores	127	←
Maximum number of mailboxes	127	←
Maximum number of memorypools	127	←

❍ The initial pattern of eventflags can be specified

Along with configurator support, the initial pattern of eventflags can now be specified.

❍ Support for the REALOS-specific debugger

Two types of debugging tools are supported.

- Debugger macros

  A function to specify and debug OS objects has been added to the Fujitsu debugger (emulator and simulator).

- Monitor-type debugger

  Performs debugging by linking the target and host via RS-232C or similar.

  The state of OS objects can be dumped and system calls issued while the target is executing.

  Improved debugging efficiency when used with the emulator.

■ Modification for the V03L01 to V03L02 upgrade

❍ Windows configurator support

On Windows you can create the configuration files via GUI interface. With the former OS you have to created the files beforehand using some editor. This facilitates definition of configuration.

❍ Multitask debugger support

The debugger supported by V03L01 has been enhanced as a multitask debugger. It has a program break function and can choose whether to break the program at task dispatching or after executing the specified system call.

The multitask debugger is available on the kernel version V03L02 and higher. (The multitask debugger is attached to the Windows basic product.)

■ Modification for the V03L02 to V04L01 upgrade

❍ Support for the F^2MC-16LX

V04L01 supports the F^2MC-16LX family.

This version of REALOS/907 now supports all of F^2MC-16L/16LX/16/16H/16F families.

❍ New C compiler

The applicable C compiler has been changed to the new "optimizing C compiler (CC097)". Compared to the conventional C compiler, the optimizing C compiler generates system call invoke codes further optimized for REALOS using the -K REALOS option.

❍ Omission of the C interface library

The optimizing C compiler is used with the -K Softune REALOS option specified. The option allows the C compiler to generate code optimized more than the code generated from the C interface library. Therefore the V04L01 package does not contain the C interface library. When compiling, use the -K Softune REALOS option.

❍ Support for all compile models of the C compiler

Formerly, the standard-model kernel supports the large model as a C compiler memory model and the single-chip model kernel supports the small model. As the marketplace has been demanding for the support for the medium model to extend the line of CPUs, V04L01 supports all compile models instead of focusing on a single kernel model.

The standard-model kernel now supports four compile models (large, medium, compact, small) of the optimizing C compiler.

❍ Omission of the single-chip model kernel

The former versions of REALOS/907 contained the single-chip model kernel. The single-chip model kernel was used in combination with the C compiler small model. The former versions assume the compile model (generally called the tiny model) smaller than the C compiler small model as the practical memory model, with the RAM size in the kernel management area limited to a maximum of 256 bytes.

In actual development, applications grow to the size of the small or medium model for function enhancements when ported from the 8-bit CPU to 16-bit CPU environment. Currently, the minimum C compiler memory model required is the small model and the single-chip model kernel is not demanded. Therefore the V04L01 packages does not contain the single-chip model kernel.

In the V04L01 version, in contrast, the standard-model kernel supports the C compiler medium and compact models as well as the large and small models, or four compile models in total, providing a wider range of memory model options available for applications.

❍ Omission of the multitask debugger

V04L01 does not contain the multitask debugger formerly bundled with the Windows version of REALOS/907 V03L02, Basic Edition.

❍ Improvements to configuration definition statements

- TSK definition statement

  Formerly, the TSK definition statement had a problem that the DTB register cannot be specified when a task is invoked. A different DTB register could not therefore be specified for each task.

  To solve the problem, the TSK definition statement has been improved to specify the DTB register.

  The configuration definition statement has been changed as follows:

  Old version (V03L02): TSK name,[tskid],entry[,[itskpri][,[stksz][,[bank_reg][,[ADB][,[DPR][,[CCR]]]]]]]

  New version(V04L01): TSK name,[tskid],entry[,[itskpri][,[stksz][,[bank_reg][,[DTB][,[ADB][,[DPR][,[CCR]]]]]]]

  If you specify the statement without ADB, DPR, and CCR which are usually not required when tasks are generated by the C compiler, you can specify the DTB register only by adding “DTB”.

  The TSK definition statement can also be used in the old format. Since the statement is interpreted with CCR set to the default value in that case, be sure to check the configuration file.

- MODEL definition statement

  The MODEL definition statement was provided to select one of the standard-model and single-chip model kernels. V04L01 supports only the standard-model kernel, eliminating the need for selection. The MODEL definition statement was therefore omitted from the V04L01 package. Note that the MODEL definition statement if included in the configuration file results in an error.

❍ External reference declaration of the unnecessary symbol (r97_sclk_hdr) output when the system clock is not incorporated

Formerly, the configurator output r97_tbl containing an external reference declaration symbol (r97_sclk_hdr) in some cases. In that case, the linker outputs warning messages W0101L and W0102L each once to the symbol r97_sclk_hdr during configuration.

If the system clock is not incorporated (with no SYS_CLOCK definition statement included in the configuration file), the configurator generates r97_tbl without external reference declaration of the symbol r97_sclk_hdr.

❍ Sample programs

The sample programs provided for the former version were designed for the MB90704 training board. They were modified for use with the F²MC-16L support board (MB90670/5). To use the serial interface and timer on the support board, you can port the program easily for the F²MC-16LX support board.

■ Modification for V04L01 to V04L02 upgrade

❍ Windows version of the Configurator

A malfunction in scrolling the output window has been corrected.

❍ Kernel

Malfunctions in the set_flg system call in the 16-bit eventflag function have been corrected. A malfunction occurred under the conditions below.

1) 16-bit eventflag was selected.
2) A set_flg was issued during task A was waiting with a wai_flg. In this case, a condition that did not make task A to wait was specified.
3) While the above set_flg was being executed, an interrupt occurred, causing handler H to operate.
4) A set_flag was issued from the above handler H. In the set_flg parameter, a condition that caused waiting task A to be canceled was specified.
5) The interrupt activating the above handler H occurred between symbol r97_flg_set and r97_self_chk in the kernel.

When this sequence occurred, this program went into a loop. The program has been modified to operate correctly under these conditions.

■ Modification for V04L02 to V30L01 upgrade

❍ The program has been modified for Softune Workbench.

❍ The debugger macro file is no longer supplied.

❍ The debugger macro file rdm907a.lst is no longer supplied.

❍ Sample program

The sample program has been modified for the F²MC-16LX support board (MB90V550). The timer and serial interface on the support board can be easily modified for the F²MC-16L support board.

❍ Hook routine for debugger

dbgfk.obj is provided for the interface with the debugger. Since this program is a part of the kernel, always create a link. When r_d_dbg.obj instead of dbgfk.obj is specified for linkage for the other support tool, link with r_d_dbg.obj. Since the interface with the debugger has been changed, the kernel process time is different from that for V04L02.

■ Modification for the V30L01/L02 to REV:300001 upgrade

❍ Append PDFs same as manuals.

❍ Kernel [V30L02 to V30L03]

For cooperation with Softune REALOS Analyzer(under development) requirements.

1) E Modify "set_tim" system call to able to analyze.
2) E Newly establish the hook routine at caused time out.
3) E Prepare distinguishable variable for kernel initializing phase.
4) Formulas for caluculating kernel management areas was changed.

Management area name	Memory size used
Task	4 + 12 × (number of tasks) + 4 × (priorty level)

↓

Management area name	Memory size used
Task	5 + 12 × (number of tasks) + 4 × (priorty level)

5) Prepare distinguishable symbols for kernel code area.

Modify "set_flg" and "wai_tsk" system call to be able to analyze at task debugging.

❍ Configurator Windows version [V01L02 to V01L03]

Modify illegal displaying in "CPU" dialog when CPU mode changed to chips only having "SINGLE" mode.

Modify illegal limitaion of usable character strings number in dialogs that was 31.

❍ Configurator console version PC version [V30L01 to V30L03]

Modify illegal characters outputting in header file that is not allowable formats to compiler.

■ **Modification for the REV:300001 to REV:300002 upgrade**

❍ Kernel [V30L03 to V30L04]

We corrected a problem where the system call processing canceled of top of handler if the interrupt occured and the handler executes during system call executing of "set_flg", "pol_flg", and "wai_tsk".

We corrected a problem where the higher 8 bit and lower 8 bit initialized by only lower 8 bit if choiced 16 bit eventflag and setting all same pattern of initial value.

❍ Configurator Windows version [V01L03 to V01L04]

The task priority level can be setting both dialogs of "System 1 Definition" and "Task Definition".

Changed of the part of Configurator GUI.

We corrected a problem where the string charactars of A to F deal as 16 bits value.

We corrected a problem where the forcibly terminates of application if input very long strings in text box.

We corrected a problem where "Register Bank Switching Method" is not selected if RW register checked at "Save and Restore in Stack".

Added "Debug Setting" dialog on Configurator dialogs of Softune Workbench.

Added function of kernel code and kernel data address setting.

Corresponded to ZIPC6.0 of CATS Corporation.

Addeed Message dialog displaying if the Configuration file can not edit.

- Configurator console version [V30L03 to V30L04]

  We corrected a problem where the eventflag initial value forcibly changes to 0 except eventflag ID for 1, 9, 17... if 1 bit eventflag selected and all initial value setting 1.

- Softune REALOS/907 ANALYZER Windows version [V30L20]

  Softune REALOS/907 Basic version for PC version included Softune REALOS/907 ANALYZER.

INDEX

<Alphabetic>

INDEX

CM42-00325-2E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

F^2MC-16L/16LX/16/16H/16F

µITRON 2.01 SPECIFICATIONS COMPLIANT

SOFTUNE REALOS/907

USER'S GUIDE

July 1999 the first edition

Published **FUJITSU LIMITED** Electronic Devices

Edited Technical Communication Dept.

FUJITSU

CM42-00315-2E

FUJITSU SEMICONDUCTOR F²MC-16L/16LX/16/16H/16F µITRON 2.01 SPECIFICATIONS COMPLIANT SOFTUNE REALOS/907 USER'S GUIDE