MF1200-01a

EPSON

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER **E0C33 Family**

MON33 Debug Monitor Manual

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

Preface

Written for those who develop applications using the E0C33 Family of microcomputers, this manual describes how to implement the E0C33 Family debug monitor library MON33 and how to debug the target program.

Table of Contents

1 MON33 Package **1**
1.1 Features 1
1.2 Components of MON33 Package 2
1.3 Installation 2

2 Implementing the Debug **4**
2.1 Resources Required for the Debug Monitor 4
2.2 Starting Up the Debug Monitor 5
2.3 Building an Application Program 6
2.4 Creating Communication Control Routines 7

3 Target Program and Debugging **10**
3.1 Notes for Creating Target Programs 10
3.2 Parameter File for Debugging 10
3.3 Starting Up and Terminating Procedure of Debugging 11
3.4 Debugging Method 12
3.5 Precautions for Debugging 14
3.5.1 Restriction on Debugging Command 14
3.5.2 Other Precautions 14

Appendix DMT33MON Board **15**
A.1 Outline of DMT33MON Board 15
A.2 Names and Functions of Each Part 15
A.3 Connecting the System 16
A.4 DMT33MON Block Diagram 17
A.5 Program Debugging with a DMT Board and MON33 18
A.6 Indispensable Signal Pins of DMT33MON 21

1 MON33 Package

The Debug Monitor MON33 is a middleware designed for E0C33 Family single-chip microcomputers. It provides program-debugging functions on the user target board or for the actual product.

1.1 Features

The following lists the features of MON33:

- It is provided as a library file that can be linked to the user program.
  This package also contains source codes of all the modules.
- MON33 uses approx. 10KB ROM, approx. 2.5KB RAM and a channel of serial interface on the E0C33 chip. It allows direct program debugging via the DMT33MON board using the debugger db33 on the personal computer.
- Allows debugging of the target program in the RAM, ROM or Flash memory on the target board.
- Supports the following debugging functions:
  - Successive execution and step execution of the program
  - PC break and data break
  - Memory/register operation
  - Flash memory writing

A configuration of the debugging system is shown in Figure 1.1.1.

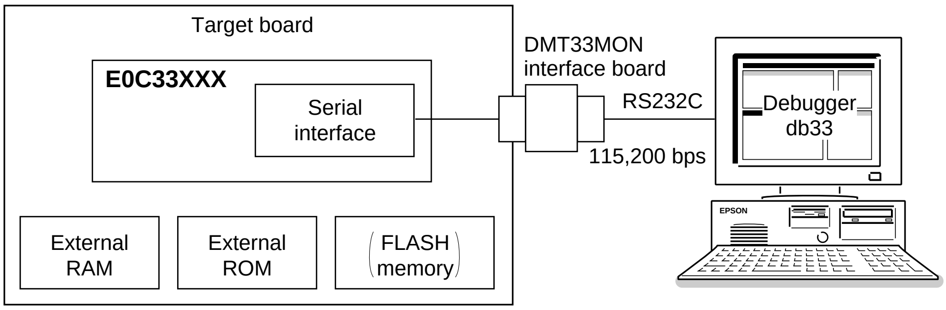


Figure 1.1.1 Configuration of debugging system

1.2 Components of MON33 Package

The following lists the contents of MON33 Package:
When unpacking, make sure that all of the following components are included.

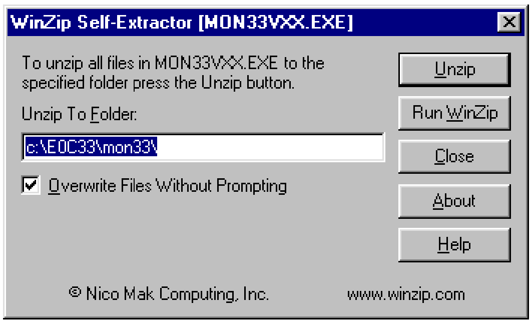
(1) Tool disk (3.5' FD for PC/AT, 1.44MB)	1
(2) E0C33 Family MON33 Debug Monitor Manual (this manual)	2 (1 English/1 Japanese)
(3) Warranty	2 (1 English/1 Japanese)

1.3 Installation

MON33 needs to be linked with the user program as it is implemented. Therefore, make sure all tools of the "E0C33 Family C Compiler Package" have been installed in the personal computer and are ready to run before installing the MON33 files. The basic system configuration is described below.

- Personal computer:
  IBM PC/AT or compatible
  (PC with Pentium 90 MHz or higher and 32MB or more memory recommended)
  One channel of the serial port is used to communicate with the debug monitor.
- OS:
  Windows95, Windows NT 4.0 or higher version (English version or Japanese version)

All the MON33 files are supplied on one floppy disk. Execute the self-extract file "mon33vXX.exe" on the FD to install the files. ("XX" in the file name represents the version number, for example, "mon33v10.exe" is the file name of MON33 ver. 1.0.)
When "mon33vXX.exe" is started up by double-clicking the file icon, the following dialog box appears.


WinZip Self-Extractor [MON33VXX.EXE]

To unzip all files in MON33VXX.EXE to the specified folder press the Unzip button.

Unzip To Folder: c:\E0C33\mon33\

[✓] Overwrite Files Without Prompting

© Nico Mak Computing, Inc. www.winzip.com

Enter a path/folder name in the text box then click [Unzip]. The specified folder will be created and all the files will be copied to the folder.
When the specified folder already exists on the specified path, the folder will be overwritten without prompting if [Overwrite Files Without Prompting] is checked.

The following lists the configuration of directories and files after copying.

(root)\	(default: C:\E0C33\MON33\)		
	readme.txt		Supplementary explanation (in English)
	readmeja.txt		Supplementary explanation (in Japanese)
	lib	..... MON33 library	
		mon33ch0.lib	MON33 library that uses the serial I/F Ch.0 on the E0C33xxx
		mon33ch1.lib	MON33 library that uses the serial I/F Ch.1 on the E0C33xxx
		mon33.lib	MON33 library that does not use a serial I/F on the E0C33xxx ... These libraries cannot be used with the ICE or the ICD. Normally, either "mon33ch0.lib" or "mon33ch1.lib" is used according to the serial I/F channel used. Use "mon33.lib" when providing a serial I/O circuit separately and when not using the DMT33MON board.
		mon33ice.lib	Library that does not use a serial I/F on the E0C33xxx for debugging the MON33 using the ICE or the ICD

mon33ch0.mak	Make file for building mon33ch0.lib
mon33ch1.mak	Make file for building mon33ch1.lib
mon33.mak	Make file for building mon33.lib
mon33ice.mak	Make file for building mon33ice.lib

src\ ..... MON33 source files

m33_def.h	MON33 definition file
m3c_brk.c	C source file for break functions
m3c_exe.c	C source file for program execution
m3c_flsh.c	C source file for Flash memory operation
m3c_main.c	MON33 main C source file
m3c_mem.c	C source file for memory operation
m3c_othe.c	C source file for other functions
m3c_sci.c	C source file for sending/receiving messages
m3s_exe.s	Assembly source file for program execution
m3s_flsh.s	Assembly source file for Flash memory operation
m3s_init.s	Assembly source file for MON33 initial set-up
m3s_mem.s	Assembly source file for memory operation
m3s_sci.s	Assembly source file for sending/receiving messages

dmt33xxx\ ..... Sample source files for the DMT33xxx, MON33 build files and related files

A sample program for blinking the LED on the DMT33xxx, a source for the on-board Flash memory write/erase routines and the make files are included in each dmt33xxx folder. The source files can be modified to use in the application program if necessary. Refer to "readme.txt" or "readmeja.txt" for the contents of the dmt33xxx folder.

2 Implementing the Debug Monitor

This chapter describes how to implement the debug monitor and how to start it from the application program.

2.1 Resources Required for the Debug Monitor

The debug monitor uses the following resources:

- Approximately 10KB of ROM area for the program code of the debug monitor.
- Approximately 2.5KB of RAM area for the work and stack area.
- Since the debug monitor uses the debugging exception of the CPU, addresses 0x0 to 0xF of the built-in RAM area are used as the vector and stack for processing debugging exceptions. Furthermore, addresses 0x10 to 0x2F are reserved for extending functions.
- One channel (Ch.0 or Ch.1) of the serial interface (8-bit asynchronous mode) is used for communicating with the debugger db33 on the personal computer.
  Figure 2.1.1 shows a connection diagram.

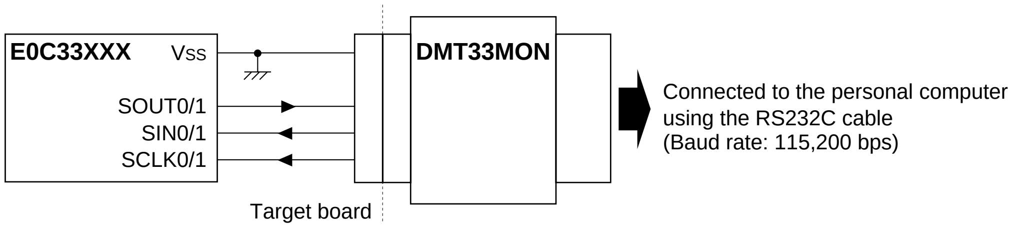


Figure 2.1.1 RS232C connector diagram

- Communication via RS232C is performed by a software polling method without any interrupt related resource.
- A serial I/O device outside the E0C33 chip can be used by customizing the serial I/O routine (see Section 2.4).
- MON33 uses the TXD, RXD and status registers of the corresponding serial interface channel exclusively. Therefore, do not access these control registers.
  Furthermore, fix the I/O pins for the channel at the serial interface pins using the port function select register. For example, the sample program "m3s_sci.s" writes 0x07 to the P0 function select register (0x402D0) when Ch.0 is used or 0x70 when Ch.1 is used to set the serial interface pins.

2.2 Starting Up the Debug Monitor

The debug monitor starts by jumping to m_mon_start( ).
Normally, provide a select switch outside the E0C33 chip for selecting either starting up the debug monitor or a normal execution and create the program that can jump to m_mon_start( ) from the boot routine after an initial reset. Furthermore, start the debug monitor after initializing the BCU if necessary. In case the debug monitor is started before the BCU is initialized, downloaded data cannot be written to a 8-bit device or in DRAM. However, when the MON33 itself is placed on the 16-bit ROM and the 16-bit SRAM is used as a work area, it can be operated even in the default setting (7-wait state) when operating in 20 MHz. In the following example, the initial set-up statements are described as comments so that the BCU operates by default. By decreasing the wait cycle number (2-wait setting in the example below), the file loading and other operations may improve the response time (approximately 5 to 10%).

Example: boot routine of "dmt33004\m3s_boot.s"

```
;******************************************************************************
;
;        BOOT program
;
;******************************************************************************
BOOT:
         xld.w    %r0,0x800
         ld.w     %sp,%r0
;        xld.w    %r5,BCU_A10_ADDR             ;area9-10 (0x800000-0xffffff)
;        xld.w    %r4,0b0000000000010010       ;Device 16 bits,delay 1.5, wait 2
;        ld.h     [%r5]+,%r4
;        xld.w    %r5,BCU_A8_ADDR              ;area8 (0x600000-0x7fffff)
;        xld.w    %r4,0b0000000000010010       ;Device 16 bits,delay 1.5, wait 2
;        ld.h     [%r5]+,%r4
;        xld.w    %r5,BCU_A5_A6_ADDR           ;area5-6 (0x200000-0x3fffff)
;        xld.w    %r4,0b0001001000010010       ;Device 16 bits,delay 1.5, wait 2
;        ld.h     [%r5]+,%r4
         xbtst    [K6XD],0x3                   ;K63 (debug SW check) 0:MON33 1:USER
         xjreq    m_mon_start                  ;MON33 start
         jp       USER
USER:
         xld.w    %r4,TBRP
         xld.w    %r6,0x59
         xld.b    [%r4],%r6                    ;TTBR writable 0x59
         xld.w    %r4,TTBR
         xld.w    %r6,0x200000
         ld.w     [%r4],%r6                    ;TTBR set 0x200000
         ld.w     %r4,[%r6]
         jp       %r4                          ;user application(flash) start
```

2.3 Building an Application Program

The MON33 modules are provided as a library file "mon33*.lib" in the directory "lib\". Link this library to the user modules.
When using the serial interface on the E0C33 chip and the DMT33MON board, link "mon33ch0.lib" (Ch.0 is used) or "mon33ch1.lib" (Ch.1 is used) accordingly. When not using the serial interface on the E0C33 chip and the DMT33MON board, create serial I/O routines separately and link "mon33.lib".
Specify the "lib\" directory of the MON33 as a library path in the linker command file when linking.

Example: "dmt33004\dmt33004.cm"

```
;Map set
-code 0x0c00000             ; set relative code section start address
-bss  0x06ff640             ; set relative bss section start address

;Library path
-l c:\CC33\lib        .... CC33 standard library path
-l ..\lib             .... MON33 library path (c:\E0C33\mon33\lib)

;Executable file
-o dmt33004.srf

;Object files
m3s_boot.o

;Library files
string.lib
ctype.lib
idiv.lib
mon33ch1.lib          .... MON33 library to be linked
```

Since all the source codes of the debug monitor are provided in the directory "src\", it is possible to customize the debug monitor if necessary. In this case, rebuild the library using "mon33*.mak" (make file for creating "mon33*.lib") in the directory "lib\".

MON33 allows the debugger db33 to write the target program to be debugged in the RAM or Flash memory on the target board. In this case, it is not necessary to create a target object linked to the debug monitor. When the DMT33xxx board (DMT33004, DMT33005, etc.) is used, the target program can be downloaded to the RAM or Flash memory and can be debugged by writing MON33 and a simple boot program to the ROM.
When executing the target program in the external ROM on the target board, link the debug monitor to the target program and create an object that is mapped to the external ROM.

2.4 Creating Communication Control Routines

For communicating with the debugger, the debug monitor calls the following four RS232C routines (1 to 4). "mon33chX.lib" that contains these routines can be used when using the serial interface Ch.0 or Ch.1 on the E0C33 chip and the DMT33MON board. Otherwise, it is necessary to create these routines according to the system since "mon33.lib" must be used. The sample program "m3s_sci.s" that uses the serial interface Ch.0/Ch.1 of the E0C33A104/E0C33208 is provided in "src\", so use it after modifying if necessary.

(1) void m_io_init( )

This is an initial set-up routine of the serial interface. Return value is not necessary.
Set up I/O terminals, input clock, baud rate and a data format. Select 8-bit asynchronous mode, no parity and 1 stop bit. The baud rate should be set to a value supported by the debugger db33 and the personal computer.
Refer to the "E0C33XXX Technical Manual" for the serial interface and for setting the clock.
The sample program "m3s_sci.s" assumes use of the DMT33MON board. It sets the baud rate to 115.2 kbps when a 1.843 MHz external clock is used.

Example: "m_io_init( ) of "m3s_sci.s"
```
#define       MON_VER  0x11             ;monitor firm-ware version

#ifdef SIO0
     #define  STDR     0x000401e0       ;transmit data register(ch0)
     #define  SRDR     0x000401e1       ;receive data register(ch0)
     #define  SSR      0x000401e2       ;serial status register(ch0)
     #define  SCR      0x000401e3       ;serial control register(ch0)
     #define  SIR      0x000401e4       ;IrDA control register(ch0)
     #define  PIO_SET  0x07             ;port function register
#else
     #define  STDR     0x000401e5       ;transmit data register(ch1)
     #define  SRDR     0x000401e6       ;receive data register(ch1)
     #define  SSR      0x000401e7       ;serial status register(ch1)
     #define  SCR      0x000401e8       ;serial control register(ch1)
     #define  SIR      0x000401e9       ;IrDA control register(ch1)
     #define  PIO_SET  0x70             ;port function register
#endif

#define       SIR_SET  0x0              ;SIR set(1/16 mode)
#define       SCR_SET  0x7              ;SCR set(#SCLK input 1.843MHz 115200bps)
#define       SCR_EN   0xc0             ;SCR enable
#define       PIO      0x000402d0       ;IO port (P port) register

     .code
;******************************************************************************
;
;    void m_io_init()
;             serial port initial function
;
;******************************************************************************
     .global  m_io_init
m_io_init:
     ld.w     %r0,SIR_SET               ;1/16 mode
     xld.b    [SIR],%r0                 ;SIR set
     ld.w     %r0,SCR_SET
     xld.b    [SCR],%r0                 ;SCR set(#SCLK input 1.843MHz)
     xld.w    %r0,PIO_SET
     xld.b    [PIO],%r0                 ;IO port set
     xld.w    %r0,SCR_EN|SCR_SET
     xld.b    [SCR],%r0                 ;SCR set
     ret
```

In addition to the sample program above, "dmt33001\m3s_sci.s " is provided. This program sets the baud rate to 38,400 bps based on the 20 MHz E0C33A104 internal clock. Refer to it if necessary.

(2) void m_snd_1byte(unsigned char)

This is a routine that sends 1-byte data. This routine receives 1-byte data as the argument (R12 register) and sends it via the serial interface. Return value is not necessary.

Example: m_snd_1byte( ) of "m3s_sci.s"

```
;*****************************************************************************
;
;    void m_snd_1byte( sdata )
;             1 byte send function
;               IN : uchar sdata (R12)  send data
;
;*****************************************************************************
     .global  m_snd_1byte
m_snd_1byte:
     pushn    %r3               ;save r3-r0
snd000:
     xbtst    [SSR1],0x1        ;TDBE1(bit1) == 0(full) ?
     jreq     snd000            ;if full, jp snd000
     xld.b    [STDR1],%r12      ;write data
     popn     %r3               ;restore r3-r0
     ret
```

(3) unsigned char m_rcv_1byte( )

This is a routine that receives 1-byte data. Store received 1-byte data into m_rcv_data. It returns following error codes (unsigned char) as the return value:

0: received normally

1: framing error

2: parity error

3: overrun error

Example: m_rcv_1byte( ) of "m3c_sci.s"

```
;*****************************************************************************
;
;    uchar m_rcv_1byte()
;             1 byte receive function
;               OUT : 0 receive OK
;                     1 receive ERROR (framing  err)
;                     2               (parity   err)
;                     3               (over run err)
;
;*****************************************************************************
     .global  m_rcv_1byte
m_rcv_1byte:
     pushn    %r3               ;save r3-r0
rcv000:
     xbtst    [SSR1],0x0        ;RDBF1(bit0) == 0(empty) ?
     jreq     rcv000            ;if empty, jp rcv000
     ld.w     %r10,0x0
     xbtst    [SSR1],0x4        ;FER1(bit4) == 0 ?
     jreq     rcv010
     xbclr    [SSR1],0x4        ;FER1(bit4) 0 clear
     ld.w     %r10,0x1          ;return 1
rcv010:
     xbtst    [SSR1],0x3        ;PER1(bit3) == 0 ?
     jreq     rcv020
     xbclr    [SSR1],0x3        ;PER1(bit3) 0 clear
     ld.w     %r10,0x2          ;return 2
rcv020:
     xbtst    [SSR1],0x2        ;OER1(bit2) == 0 ?
     jreq     rcv030
     xbclr    [SSR1],0x2        ;OER1(bit2) 0 clear
     ld.w     %r10,0x3          ;return 3
rcv030:
     xld.b    %r0,[SRDR1]       ;read data
     xld.b    [m_rcv_data],%r0  ;read data set
     popn     %r3               ;restore r3-r0
     ret
```

(4) void m_dummy_rd( )

This is a dummy read routine. It reads data from the receive buffer and clears the buffer. Return value is not necessary.

Example: m_dummy_rd( ) of "m3s_sci.s"

```
;*******************************************************************************
;
;    void m_dummy_rd()
;             dummy read function
;
;*******************************************************************************
     .global  m_dummy_rd
m_dummy_rd:
     xld.b    %r4,[SRDR]        ;read data
     ret
```

(5) void m_ver_rd( )

This is a version read routine. Modifying the constant MON_VER can change the MON33 version number. It returns MON_VER as the return value.

Example: m_ver_rd( ) of "m3s_sci.s"

```
;*******************************************************************************
;
;    void m_ver_rd()
;             mon firmware version read function
;
;*******************************************************************************
     .global  m_ver_rd
m_ver_rd:
     xld.w    %r10,MON_VER      ;mon firmware version
     ret
```

(6) Sample command file for building

When the above routines are newly created, create a linker command file as the following example and link "mon33.lib" to the user modules.

Example: dmt330001\dmt33001.cm

```
;Map set
-code 0x0c00000        ; set relative code section start address
-bss  0x081f640        ; set relative bss section start address

;Library path
-l c:\CC33\lib
-l ..\lib

;Executable file
-o dmt33001.srf

;Object files
m3s_boot.o
m3s_bcu.o
m3s_sci.o              .... Originally created SIO routine

;Library files
string.lib
ctype.lib
fp.lib
idiv.lib
mon33.lib              .... Link "mon33.lib" that does not include SIO routine
```

3 Target Program and Debugging

This chapter describes precautions for debugging using the debug monitor.

3.1 Notes for Creating Target Programs

Follow the instructions below when creating the target program to be debugged:

- Since the debug monitor uses the debugging exception of the CPU, addresses 0x0 to 0xF of the built-in RAM area are used as the vector and stack for processing debugging exceptions. Therefore, the target program cannot use this area. Furthermore, do not use addresses 0x10 to 0x2F.
- When debugging the target program by writing in the RAM or Flash memory on the target board, map the program to that address.
- The debug monitor does not allow forced break functions such as key breaks. Forced break functions should be made in the target program using a key input interrupt or an NMI input. Setting a hardware PC break in the interrupt processing routine makes it possible to execute a forced break.

3.2 Parameter File for Debugging

A debug-parameter file is required to start the debugger. Create the file according to the memory configuration of the target system. When using the debug monitor, the debugger ignores all the settings for the ICE emulation memory.
When using a Flash memory, specify it as a RAM.

Example: "dmt33004\sample\33104_m.par"

```
CHIP                 33104              ; chip name (33XXX)
IROM                 1000               ; internal ROM is 80000 to 80FFF
FOPT                 0000               ; f option size
PRC VER              00 ff              ; allow any PRC board
PRC STATUS           ****************   ; allow any PRC board status
MPU                                     ; 0xC00000 external boot address
VER                  1                  ; this file version

; Emulation memory allocation (max 8 areas, 1MB/area, 1MB boundary)
EMROM      c00000    cfffff             ; external ROM 1MB

; Map allocation (max 31 areas, 256bytes boundary)

RAM        0         7FF                ; internal RAM area 2KB
IO         40000     4ffff              ; internal IO area 64KB
RAM        200000    2fffff             ; external FLASH 1MB
RAM        600000    6fffff             ; external SRAM 1MB
EROM       c00000    cfffff             ; external ROM 1MB
; Stack area except internal RAM area (max 8 areas, 256bytes boundary)

STACK      600000    6fffff             ; external stack area 1MB

END
```

3.3 Starting Up and Terminating Procedure of Debugging

Follow the procedure below to start debugging.

1. Make sure the power of the target board and personal computer are off.
2. Connect the DMT33MON board to the target board in which the debug monitor is implemented and connect the DMT33MON board to the personal computer with the RS232 cable.
3. Turn the target board on and then start the debug monitor.
4. Turn the personal computer on and start the debugger db33 in debug monitor mode from the work bench wb33 or DOS prompt.
   Example: `db33 -mon -b 115200 -p 33104_m.par`

Follow the procedure in order from Step 4 to Step 1 to terminate debugging and for power off.

Note: When connecting and disconnecting the RS232C cable, make sure the target board and the personal computer are off.

3.4 Debugging Method

The following three methods are available for debugging the target program.

1. Debugging in the ROM

Map the target program in the ROM after linking to the debug monitor. The target program can be executed and debugged in the ROM.

Since the [Source] window of the debugger displays the disassembled content of the ROM on the target system, it is not necessary to load the target program with the lf command. To display the source, load the same absolute object file as the one written to the ROM. In this case, by using the ld command instead of the lf command, the db33, except for the object code, loads only the debugging information.

After starting the debugger, follow the process below before starting to debug:

1. Load the debugging information (when displaying the source or using symbols).
2. Set up a hardware PC break point for the forced break function.

Example: command file example of the above description

```
ld sample.srf        ; Load the debugging information of the target program
bh ESC               ; Set up a hardware PC break point at the label ESC (for forced break)*
```

* For example, an NMI input switch can be used for the forced break function.

Example: NMI processing routine example for forced break function

```
NMI:                        ; nmi
         nop
         .global  ESC
ESC:                        ; Label set as a hardware PC break point for the forced break function
         reti
```

Note: When debugging the target program in the ROM, the software PC break function (bs command) cannot be used.

2. Debugging in the RAM

Load the target program into the RAM of the target system with the lf command to debug the program in the RAM.

After starting the debugger, follow the process below before starting to debug:

1. Set the trap table base address (only when placing the trap table in the RAM).
2. Load the target program.
3. Reset the CPU.
4. Set up a hardware PC break point for the forced break function.

The following shows a debug-command file example in which the above process is described.

Example: "dmt33004\sample\led.cmd"

```
eb 4812d             ; Set up TBRP (TTBR write protection register)
59                   ; Remove TTBR write protection
q
ew 48134             ; Set up TTBR (trap table base register)
600000               ; Set up the base address to 0x600000 (external RAM)
q
lf led.srf           ; Load the target program (0x600000~)
rsth                 ; Reset the CPU (reset vector at 0x600000 is set to the PC)
bh ESC               ; Set up a hardware PC break at the label ESC (for forced break)
```

3. Debugging in the Flash memory

In the target system in which a Flash memory has mounted, the target program can be debugged by writing it into the Flash memory using the debugger.

Notes: • When debugging the target program in the Flash memory, the software break function (bs command) cannot be used.
• The debugger db33 ver. 1.72 or later version supports data writing to the Flash memory on the target board. To debug using the Flash memory, create Flash erase and Flash write routines in the user program and write the program following the instructions below:

After starting the debugger, follow the process below before starting to debug:

1. Load and initialize the Flash erase/write routines.
2. Erase the Flash memory.
3. Set up the trap table base address.
4. Load the target program.
5. Reset the CPU.
6. Set up a hardware PC break point for the forced break function.

The following shows a debug-command file example in which the above process is described.

Example: "dmt33004\sample\led2.cmd"

```
lf ..\sample\flsh\am29f800.srf   ; Load the Flash erase/write routines to the built-in RAM
fls                              ; Flash set up command
1                                ; 1: Set up    2: Clear
200000                           ; Flash memory start address = 0x200000 (*2)
2fffff                           ; Flash memory end address = 0x2fffff (*2)
FLASH_ERASE                      ; Flash erase routine start address (*1)
FLASH_LOAD                       ; Flash load routine start address (*1)
fle                              ; Flash memory erase command
0x200000                         ; Flash control register = 0x200000
0                                ; Erase start block, 0 = All area, 1–19 = Start section
0                                ; Erase end block, 1–19 = End section, 0 = Ignored
eb 4812d                         ; Set up TBRP (TTBR write protection register)
59                               ; Remove TTBR write protection
q
ew 48134                         ; Set up TTBR (trap table base register)
200000                           ; Set up base address to 0x200000 (Flash memory start address)
q
lf led2.srf                      ; Load the target program (0x200000~)
rsth                             ; Reset the CPU (reset vector at 0x200000 is set to PC)
bh ESC                           ; Set up a hardware PC break at the label ESC (for forced break)
```

*1: "am29f800.srf" is created so as to operate by loading into the built-in RAM (2KB).
When using this source for the E0C33A104 after modifying, use the patch tool "cc33\utility\filter".

*2: This sample ("dmt33004\sample\led2.srf") assumes that a Flash memory of 1MB is located at 0x200000–0x2fffff.

3.5 Precautions for Debugging

3.5.1 Restriction on Debugging Command

When the debug monitor is used for debugging, the following debugging functions/commands are not available or allowed to be used.

When the following commands/functions are used, an error message will be displayed.

```
Error: Command is not supported at present mode.
```

- Trace function (tm, td, ts and tf commands)
- Sequential break function (bsq command)
- ICE Flash memory function (lfl, sfl and efl commands)
- Option related function (lo and od commands)
- ICE emulation memory

The following commands/ functions are not available even though no error message will be displayed.

- File loading via a parallel port (lf and lh commands)
- Map break function
- On-the-fly function
- Execution time measurement function
- Key break function

In addition to the functions above, the following functions cannot be used when the program in the ROM or Flash memory is debugged.

- Software PC break functions (bp, bs and bc commands)
- Commands that use the software PC break function internally (stdin and stdout commands)
- Memory edit functions (eb, eh and ew commands)
- Memory fill functions (fb, fh and fw commands)
- Memory move functions (mv, mvh and mvw commands)

3.5.2 Other Precautions

- The debug monitor uses addresses 0x0 to 0x2F in the built-in RAM and approximately 2.5KB (described later) part of the external RAM. Do not rewrite this area with a memory operation command. When this area is modified, the debug monitor cannot be executed normally.
- The cold reset sequence is the same as the hot reset sequence.
  1) The vector value indicated by TTBR is set to the PC.
  2) Initial setting: general purpose/special registers = 0xAAAAAAAA, PSR = 0x0, SP = 0xAAAAAA8
  In the debug monitor, cold reset is simulated as hot reset.
- The ICE33 and ICD33 halt all the peripheral functions after a break occurs except for the DRAM refresh operation. In the debug monitor, the peripheral functions halt instantaneously when a break occurs or successive/step execution starts, however they restart immediately. Interrupts while the target program is suspended are disabled according to the IE-bit status of the PSR.

Appendix DMT33MON Board

This chapter describes how to use the DMT33MON board.

A.1 Outline of DMT33MON Board

The DMT33MON board provides the interface for the debug monitor to the demonstration tools such as the DMT33004 or the user target board. The DMT33MON allows on-board debugging using the debugger (db33.exe) on a personal computer by connecting it to the target board in which the E0C33 Family debug monitor (MON33) has been implemented.
Two types of boards are available: DMT33MON board for 5 V operation and DMT33MONLV for 3.3 V operation.

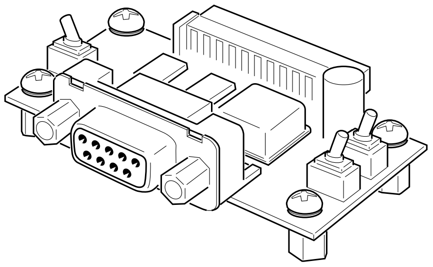


Figure A.1.1 DMT33MON board

A.2 Names and Functions of Each Part

The following describes the parts layout on the DMT33MON board as well as the functions of the connectors and switches:

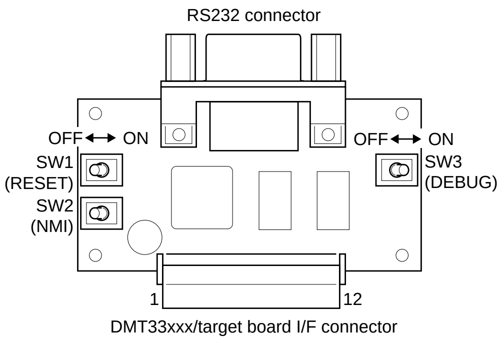


Figure A.2.1 DMT33MON board layout

SW1 (RESET switch)

Sets up the status of the RESET pin at the DMT33xxx/ target board I/F connector. When a DMT33xxx board is connected, this switch can reset the CPU on the board.
ON: RESET = "0" OFF: RESET = "1"

SW2 (NMI switch)

Sets up the status of the NMI pin at the DMT33xxx/ target board I/F connector. When a DMT33xxx board is connected, a NMI request can be input to the CPU on the board.
ON: NMI = "0" OFF: NMI = "1"

SW3 (DEBUG switch)

Sets up the status of the DEBUG pin at the DMT33xxx/target board I/F connector. When the DMT33004/ 33005 board is connected, the switch signal is input to the K63 port and can be used to start up the debug monitor from the boot routine.
ON: DEBUG (K63) = "0" OFF: DEBUG (K63) = "1"
When the switch is ON, the DMT33004/33005 will start the debug monitor. When the switch is OFF, the DMT33004/33005 will start executing the program stored in the Flash memory. The DMT33004/33005 works assuming the debug switch is OFF when the DMT33MON is not connected.

RS232C connector

This is a Dsub 9-pin connector for connecting a personal computer. Use the RS232C cable supplied with the DMT33MON package for connection.

DMT33xxx/target board I/F connector

This connector is used for connecting the DMT33xxx board. The pin layout is as follows:

Table A.2.1 DMT33xxx/target board connector pin layout

No.	Signal name	No.	Signal name
1	Vcc [+5 V, +3.3 V]	7	N.C.
2	Vcc [+5 V, +3.3 V]	8	DEBUG (K63)
3	RESET (#RESET)	9	Vcc [+5 V, +3.3 V]
4	TxD (P05)	10	SCLK (P06)
5	RxD (P04)	11	GND
6	NMI (#NMI)	12	GND

( ) indicates the CPU pin corresponding to the signal when the DMT33004/33005 is connected.
Since the corresponding signals on the DMT33xxx may differ depending on the board, refer to the pin layout table provided in the specifications of each DMT33xxx board.

A.3 Connecting the System

Note: When connecting and disconnecting the system, make sure to turn off the power of the DMT33xxx/ target board and the personal computer.

Connecting to the DMT33xxx board/user target board

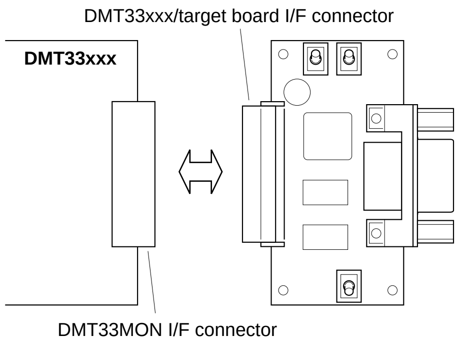


Figure A.3.1 Connecting to the DMT33xxx board

The DMT33xxx board has a connector used for connecting with the DMT33MON.
Connect the DMT33MON to the DMT33xxx board with the DMT33xxx/target board I/F connector.

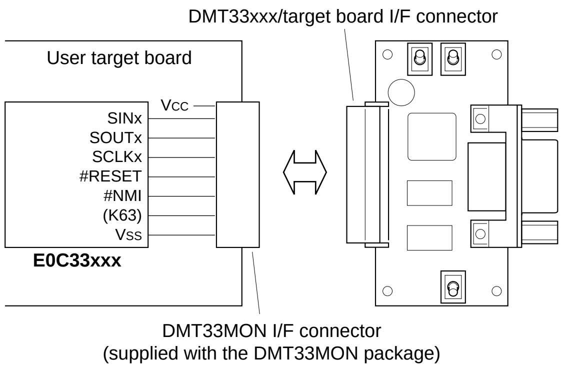
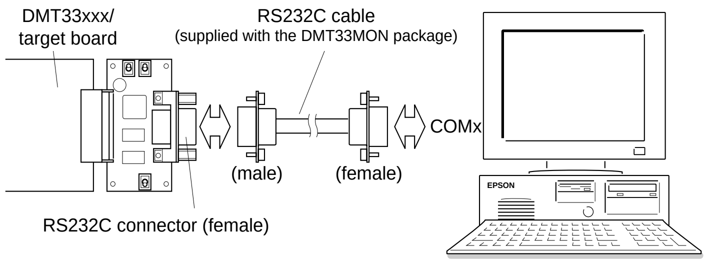


Figure A.3.2 Connecting to the user target board

When connecting to the user target board, attach the DMT33MON I/F connector (supplied with the DMT33MON package) to the target board. See Table A.2.1 for the pin layout of the DMT33xxx/target board I/F connector.

Connecting to a personal computer

Connect the DMT33MON board to the COMx port connector (the port used for debugging) of the personal computer with the RS232C cable supplied with the DMT33MON package.



DMT33xxx/
target board
RS232C cable
(supplied with the DMT33MON package)
COMx
(male)
(female)
RS232C connector (female)
EPSON

Figure A.3.3 Connecting to a personal computer

A.4 DMT33MON Block Diagram

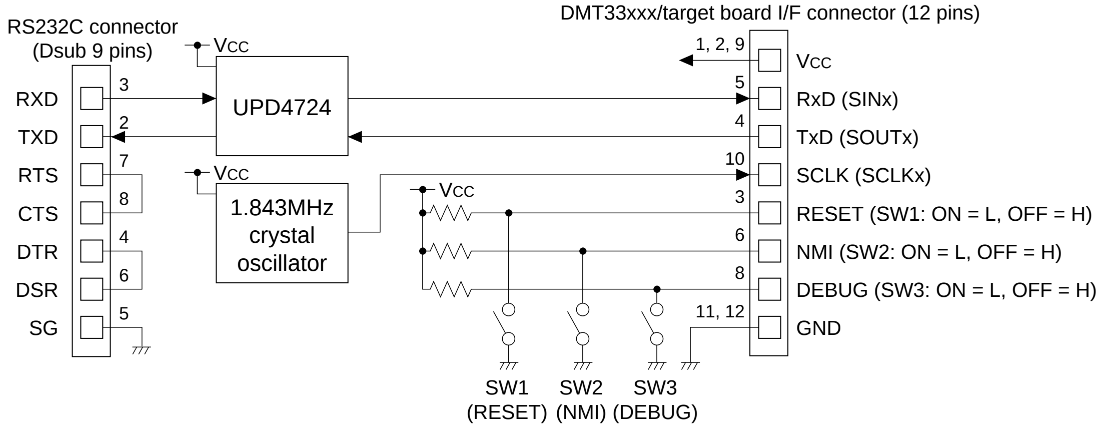


Figure A.4.1 DMT33MON block diagram

A.5 Program Debugging with a DMT Board and MON33

This section describes the debugging procedure of the program on the target system configured with the DMT33MON board and the DMT33004/33005 board using the MON33. The sample program for the DMT33004/33005 is used for the explanation. Further, the development tools in the "E0C33 Family C Compiler Package" including the debugger (db33 ver. 1.72 or later) that supports MON33 should be installed for debugging.
The debugging function of the debug monitor can be tested using the sample file even when using a user target board as well as the DMT board. Use the sample file after modifying the necessary parts such as the mapping condition and the communication routines (refer to Section 2.4) according to the target system.

DMT33004/33005 address map

Figure A.5.1 shows the DMT33004/33005 memory map and the area used by the debug monitor.

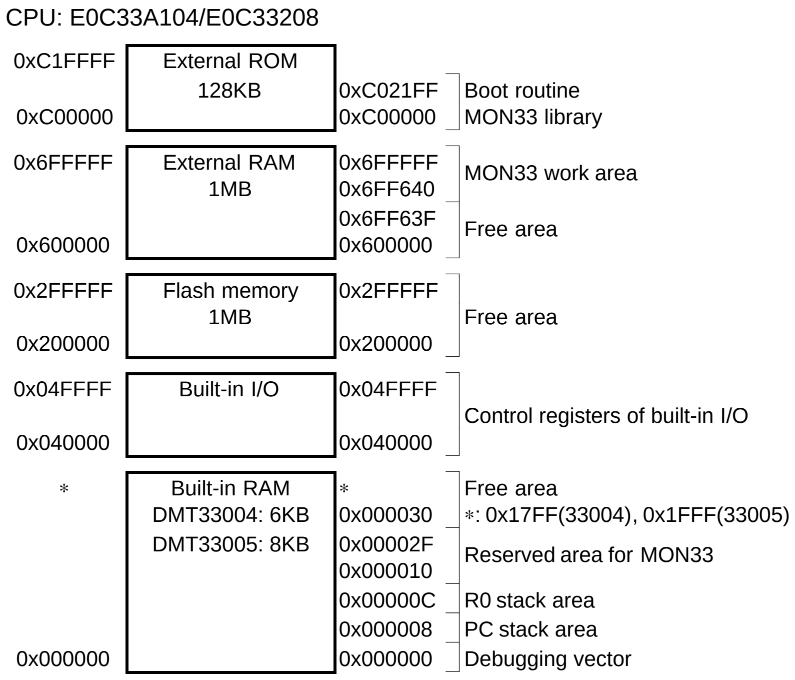


Figure A.5.1 DMT33004/33005 memory map

Sample program

"\dmt33004\sample\led.srf" and "dmt33004\sample\led2.srf" are sample programs for the DMT33004 that blinks the LED on the DMT33004 board. "led.srf" and "led2.srf" are created to be able to debug in the RAM (0x600000~) and in the Flash memory (0x200000~), respectively.
For the contents of the program, refer to the source file (\dmt33004\sample\led.s).
Sample programs for the DMT33005 are also provided in the "\dmt33005\sample\" directory.

Boot routine and implementing the debug monitor

A boot routine and the debug monitor are written in the external ROM (0xC00000~) on the DMT33004/33005 in advance. Therefore, a sample program/target program can be debugged by loading from the debugger db33 to the RAM or the Flash memory on the DMT33004/33005. It is not necessary to link the MON33 library to the program to be debugged.
The MON33 library implemented in the DMT33004 is "mon33ch1.lib" that uses the built-in serial interface Ch.1. The DMT33005 uses "mon33ch0.lib" that supports the built-in serial interface Ch.0.
Refer to "\dmt33004\m3s_boot.s" and "\dmt33005\m3s_boot.s" for the boot routine, "\dmt33004\dmt33004.cm" and "dmt33005\dmt33005.cm" for the linker commands to implement the debug monitor.

Building the sample program

It is not necessary to execute Make when modification of the source is not needed since the executable object files ("led.srf," "led2.srf") are provided in the "sample\" folder.
When the source is modified, execute Make using the make file provided.

Make execution procedure

1) Set "\dmt33004\sample\" (or "\dmt33005\sample\") as the current directory.
2) Enter the following command at the DOS prompt.

```
C:\...\SAMPLE\>make -f led.mak        ... when creating "led.srf"
C:\...\SAMPLE\>make -f led2.mak       ... when creating "led2.srf"
```

Make can also be executed from the work bench wb33 (refer to the "E0C33 Family C Compiler Package Manual").

Starting up the debug monitor

The boot routine mapped from address 0xC00000 on the DMT33004/33005 starts the debug monitor when the K63 input port is set to "0".
Start up the debug monitor following the procedure below after connecting the target system and a personal computer.
1) Turn SW3[DEBUG] of the DMT33MON on.
2) Turn the power of the DMT33004/33005 on.
3) Reset the DMT33004/33005 (DMT33MON SW1 [RESET] ON→OFF).
4) Turn the personal computer on and start up Windows.
5) Start up the debugger db33 (start-up method is described later).

Note: When the power of the DMT33004/33005 is turned on while the SW3 [DEBUG] of the DMT33MON is off, the debug monitor does not start up. The DMT33004/33005 sets TTBR at the beginning of the Flash memory (0x200000~), so the program sequence branches to the boot address. In this case, turn the SW3 [DEBUG] on and reset the DMT33004/33005 with the SW1 [RESET] to start up the debug monitor.

Debugging in the RAM

The sample program for debugging in the RAM (0x600000~) of the DMT33004/33005 is "led.srf". When starting up the debugger, specify the debug command file "led.cmd" with the -c option. "led.cmd" sets the trap table address to the start address of the RAM and loads "led.srf" to the RAM.
Operating procedure is as follows:
1) Start up the debug monitor as described above.
2) Set "\dmt33004\sample\" (or "\dmt33005\sample\") as the current directory.
3) Set a path to db33.exe.
4) Start up the debugger with the following command at the DOS prompt.

```
C:\...\SAMPLE\>db33 -mon -b 115200 -p 33104_m.par -c led.cmd
```

The debugger starts in debug monitor mode and is ready to debug "led.srf". For example, the LED on the DMT33004/33005 board will start blinking by executing the g command.
Refer to "2. Debugging in the RAM" in Section 3.4, "Debugging Method", for the contents of the command file.

Note: The debugger db33 ver. 1.0 does not support the debug monitor. Use ver. 1.72 or a later version.

Debugging in the Flash memory

The sample program for debugging in the Flash memory (0x200000~) of the DMT33004/33005 is "led2.srf". The debugger db33 ver. 1.72 or later version supports debugging in the Flash memory. Refer to the "Debugger" section of the "E0C33 Family C Compiler Package Manual" for details of operations.
To write the sample program to the Flash memory, first load the Flash erase/write routine "am29f800.srf". Then initialize the Flash memory functions using the fls and fle commands and load the sample program into the Flash memory using the lf command. Refer to the sample debug command file "led2.cmd" for executing procedure.

When starting up the debugger, specify the debug command file "led2.cmd" with the -c option. "led2.cmd" contains debug commands for loading the Flash erase/write routine, setting the trap table address and loading "led2.srf" to the Flash memory.
Operating procedure is as follows:
1) Start up the debug monitor as described above.
2) Set "\dmt33004\sample\" (or "\dmt33005\sample\") as the current directory.
3) Set a path to db33.exe.
4) Start up the debugger with the following command at the DOS prompt.

```
C:\...\SAMPLE\>db33 -mon -b 115200 -p 33104_m.par -c led2.cmd
```

The debugger starts in debug monitor mode and is ready to debug "led2.srf". For example, the LED on the DMT33004/33005 board will start blinking by executing the g command.
Refer to "3. Debugging in the Flash memory" in Section 3.4, "Debugging Method", for the contents of the command file.

When debugging in the Flash memory, be aware that the software PC break function (bs command), memory edit/fill/move commands and commands not supported by the debug monitor cannot be used.

Forced break

The debug monitor does not support forced break functions such as key break.
In the sample program, the label ESC is described in the NMI processing routine of the source ("led.s"). When the debug command file ("led.cmd", "led2.cmd") is executed, a hardware PC break point is set at the ESC location after the program has been loaded.
When the SW2 of the DMT33MON is turned on, a NMI is generated and it suspends the program execution forcibly.

Notes on debugging the user program on the DMT33004/33005 board

- When debugging the user program on the DMT33004/33005 board, create the program so that it can be loaded and executed in the free area of the RAM or the Flash memory in the same way as the sample file. (See Figure A.5.1)
- The debug monitor on the DMT33004 has been implemented by linking with the "mon33ch1.lib". Therefore, the built-in serial interface Ch.1 cannot be used from the user program.
- The debug monitor on the DMT33005 has been implemented by linking with the "mon33ch0.lib". Therefore, the built-in serial interface Ch.0 cannot be used from the user program.

A.6 Indispensable Signal Pins of DMT33MON

When using the MON33 it is not absolutely necessary to connect the NMI, RESET and DEBUG switches/signals on the DMT33MON board. If these switches are not used, the target board can be connected to the DMT33MON using only the five signals as shown below.

Example: Connecting the target board to DMT33MON with five wires

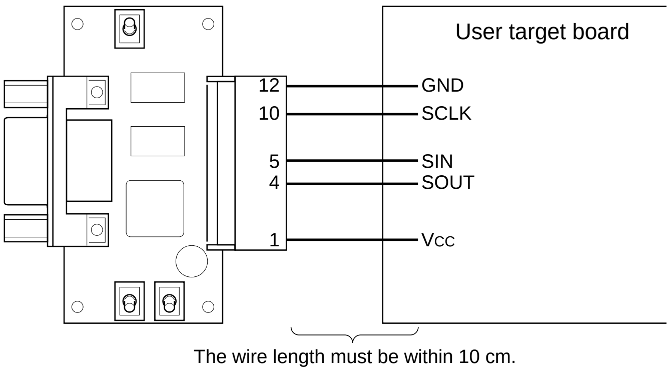


Figure A.6.1 Connection example using indispensable pins

The NMI, RESET and/or DEBUG signals should be connected if necessary.

EPSON **International Sales Operations**

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Muenchen, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)217-15045-0 Fax: +49-(0)217-15045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
2.4 Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -
FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -
EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107320

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -
EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN, R.O.C. -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN, R.O.C.
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN, R.O.C.
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -
EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -
SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-Dong
Youngdeungpo-Ku, Seoul, 150-010, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -
SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department I (Europe & U.S.A.)
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department II (Asia)
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

ENERGY SAVING
EPSON

In pursuit of **“Saving” Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers’ dreams.
Epson IS energy savings.

Issue APRIL 1999, Printed in Japan Ⓜ A