

# x186 Monitor





Release 02.2024

**MANUAL**

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .....	
ICD In-Circuit Debugger .....	
Processor Architecture Manuals .....	
x186 .....	
x186 Monitor .....	1
Brief Overview of Documents for New Users .....	5
Warning .....	5
Quick Start 186 ESI-ROM Monitor .....	6
Troubleshooting .....	9
FAQ .....	9
Basics .....	11
Monitor Features .....	11
Monitor Files .....	11
Address Layout .....	11
Vector Table .....	12
Emulation Modes .....	13
SYStem.Mode .....	Establish the communication with the CPU 13
SYStem.CPU .....	CPU type 14
SYStem.MemAccess .....	Select run-time memory access method 14
SYStem.CpuAccess .....	Run-time memory access (intrusive) 15
SYStem.CpuBreak .....	Master control to deny stopping the target (long stop) 16
SYStem.CpuSpot .....	Master control to deny spotting the target (short stop) 17
General SYStem Settings and Restrictions .....	18
SYStem.Option.REL .....	Relocation register 18
SYStem.Option.NIBBLE .....	Set global nibble flags 18
SYStem.PORT .....	Set communication parameters 18
General Restrictions .....	19
TrOnchip Commands .....	20
TrOnchip.CONVert .....	Adjust range breakpoint in on-chip resource 20
TrOnchip.RESet .....	Set on-chip trigger to default state 20
TrOnchip.state .....	Opens configuration panel 20



PP:00000164 \\SCO386I\func2+15

..... MIX AI

E:w.d.l				
addr/line	code	label	mnemonic	comment
163		autovar = regvar = fstatic;		
PP:0000015B	8B1D6C0C4000		mov ebx, [400C6C]	; ebx, fstat
PP:00000161	895DFC		mov [ebp-4], ebx	
164		autovar++;		
PP:00000164	FF45FC		inc dword ptr [ebp-4]	
166		func1( &autovar );		/ * to force autovar as stack-s
PP:00000167	8D45FC		lea eax, [ebp-4]	

E:w.r						E:w.v.v %c %m ast		
Cy	C	EAX	1	EBX	0	SP	>00000006	ast = (
P	_	ECX	3	EDX	4	-0C	00000007	word = 0x0,
Ac	_	DS	38	ESI	6	-08	00000002	count = 12345,
Zr	_	ES	30	EDI	7	-04	00000000	left = 0x401E14,
S	_	SS	34	ESP	3FBC	FP	>00003FFC	right = 0x0,
T	_			EBP	3FCC	+04	00000C8C	field1 = 1,
I	_	CS	28	EIP	164	+08	00000006	field2 = 2)
D	_	FS	30	TR	40	+0C	00000007	
O	_	GS	30	LDTR	18	+10	00000002	
PL	0	EF	1			+14	00000000	

# Brief Overview of Documents for New Users

---

## Architecture-independent information:

- **“Training Basic Debugging”** (training\_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app\_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general\_ref\_<x>.pdf): Alphabetic list of debug commands.

## Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos\_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

## Warning

---

<b>NOTE:</b>	Do not connect or remove probe from target while target power is ON.  Power up:    Switch on emulator first, then target Power down: Switch off target first, then emulator
--------------	--

# Quick Start 186 ESI-ROM Monitor

---

Starting up the ROM Monitor is done as follows:

1. Select the device **B:** for the ROM Monitor.

```
b:
```

2. Power the system down (optional).

```
sys.d
```

This instruction is necessary when the system is restarted. When the system is active while you try to reinitialize it, you get an error message.

3. Set the CPU type in the ROM Monitor program to load the CPU specific settings.

```
sys.cpu I80186
```

4. Map the EPROM simulator. The mapping of the EPROM simulator is described in the section **“Mapping the EPROM Simulator”**.

```
map.rom a:0x0fc000--0x0fffff  
map.bus16 a:0x0fc000--0x0fffff
```

5. **Load** the application program.

```
d.load.omf mcc.x /mri
```

The format of the Data.LOAD command depends on the file format generated by the compiler. The corresponding options for all available compilers are listed in the compiler list. A detailed description of the Data.LOAD command is given in the Emulator Reference Manual.

**NOTE:** The application must have a gap for the monitor program (see Address Layout below).

6. **Load** the monitor program. Usually the monitor program runs at top of memory in the ROM area. For byte bus width rom186b.bin, for word bus width rom186w.bin must be used.

```
d.load.b rom186w.bin ap:0x0fc000--0x0fffff
```

7. Set the polarity of the Reset and NMI signal according to your target. The NMI signal is optional, it can be use to interrupt the program.

```
x.respol -           Reset and NMI signal should be connected
x.nmipol +          from ESI to target. NMI is used for manual
x.nmibreak on       break.
```

8. Start the ROM Monitor. If the RESET output of the ESI is not connected you must press the RESET button of your target after entering this command.

```
SYStem.Up
```

The start-up can be automated using the programming language PRACTICE.

A typical start sequence is shown below:

```
; the EPROM is in the addressrange    0x0fc000--0x0fffff
; the RAM is in the addressrange      0x000000--0x07ffff
```

```

b: ; select the Debugger device
sys.res
sys.down ; switch the system down
winclear ; clear all windows
sys.cpu i80186 ; set the CPU type for the user
; interface
map.res ; mapper reset
map.rom a:0fc000--0fffff ; map the EPROM simulator
map.bus16 a:0fc000--0fffff ; 2x2764 used (for low byte and high
; byte) from 0fc000--0fffff
d.load.b rom186w.bin ap:0fc000--
0fffff ; load the monitor
x.nmipol + ; adapt the polarity of RES and NMI
x.respol -
x.nmibreak on ; enables the connection of the NMI
signal

d.a a:0ffff0 mov dx,0ffa0
d.a , mov ax,0fc3c ; patch initialization of UMCS
d.a , add ax,1 ; register and jump after RESET
d.a , out dx,ax
d.a , jmp 0fc00:100 ; jump into ROM Monitor

d.s a:0fc00c %w 0a00 0fc00 ; patch offset and segment for
d.load.b ini186.bin a:0fca00 ; peripheral initialization file
; ini186.bin (this file is target
; specific and includes all
; per. ini. which must be done before
; monitor is executed)
; if this ini. is not needed:
; d.s a:0fc00c %w 040 0fc00

sys.u ; power the system up

d.load.o mcc.x /mri ; load application
d.s ad:4 %w 50 ; patch interrupt vector 1 (single
d.s ad:6 %w 0fc00 ; step), 2 (break, NMI), 3
d.copy ad:4--7 ad:8 ; (breakpoint)
d.copy ad:4--7 ad:0c

r.s sp 100 ; set stack pointer to RAM

; Important: ROM Monitor should be located at the base address of ROM!
; (offset: 0). If this is not possible, communication area must be
; removed by using "map.comstart offset". Offset is 1000h default on a
; byte ROM and 2000h on a word ROM.
; Example:
; map.rom a:0e0000--0fffff
; map.comstart 1d000
; d.load.b rom186b.bin ap:0fc000--0fffff
; Offset (byte eprom): 0fc000-0e0000+1000
; Offset (word eprom): (0fc000-0e0000+2000)/2
; For further details regarding address map and initialization of
; peripherals please refer to the file ROM186.ASM.

```



No information available.

## FAQ

---

<p>EPROM Simulator Error on Data Modification</p> <p>Ref: 0056</p>	<p><b>Why does the ROM monitor crash after modification of EPROM?</b></p> <p>Check that there is enough space left on the stack. See also "Restrictions for Stack Requirements".</p>
<p>Step or Breakpoint Fails</p> <p>Ref: 0061</p>	<p><b>Why does single step or breakpoint not work?</b></p> <p>Check that there is enough space left on the stack before and after the execution of the instruction. See "Restrictions for Stack Requirements". Make sure that the single step and INT3 vector (1 + 3) are valid and point to the correct monitor entry.</p>
<p>Stepping Fails when Executing MOV SP,xxx</p> <p>Ref: 0062</p>	<p><b>Why does stepping fail, when executing a MOV SP,xxx instruction?</b></p> <p>Check that there is enough space left on the stack before and after the execution of the instruction. See "Restrictions for Stack Requirements".</p> <p>Check that the value for the CP is within limits for the CPU and that the register space is not being overwritten by the stack. See "Restrictions for Stack Requirements".</p>
<p><b>80186</b></p> <p>186EM/ES/ER: Emulator Crash after RESET</p> <p>Ref: 0063</p>	<p><b>Why does emulator crash after a target RESET?</b></p> <p>PIO29/S6 at the 18xEM/ES/ER must be programmed as S6 signal. Therefore monitor program initializes PIO DIRECTION REGISTER 1 (offset: 78h) to 0dfffh (instead of 0ffffh). It is important to initialize PIO DIRECTION REGISTER 1 with the first executed instructions after RESET.</p>
<p><b>80186</b></p> <p>Manual Break Fails</p> <p>Ref: 0060</p>	<p><b>Why does manual break fail?</b></p> <p>Check that there is enough space left on the stack before and after the execution of the instruction (see Restrictions for stack requirements). Check exception control (x.nmipol +, x.nmibreak on) and NMI connection from EPROM simulator to target. Make sure that the NMI vector (2) is valid and points to the correct Monitor entry.</p>

<p><b>80386</b></p> <p>Manual Break Fails</p> <p>Ref: 0060</p>	<p><b>Why does manual break fail?</b></p> <p>Check that there is enough space left on the stack before and after the execution of the instruction (see Restrictions for stack requirements). Check exception control (x.nmipol +, x.nmibreak on) and NMI connection from EPROM simulator to target. Make sure that the NMI vector (2) is valid and points to the correct Monitor entry.</p>
--	---

## Monitor Features

---

The monitor requires no stack during startup and memory operations. For debugging application a valid stack is always required. The NMI pin of the Eprom Simulator can be used to manually stop the target program.

## Monitor Files

---

The 'rom186b' and 'rom186w' monitors are for Eprom Simulator solutions (8bit and 16bit). The target program can be single stepped without stopping the target processors interrupts. The source file of the monitor is 'rom186.asm'. This source file should not be modified, it is only included for reference purposes. There are two possibilities to include the monitor in the application: loading the '.bin' by the Eprom Simulator or linking the '.src' file together with the application. The '.src' files contain only the monitor code, a corresponding configuration table must be included in the target program.

## Address Layout

---

The boot jump is located at top of memory at address ap:0FFFF0 (boot ROM). The ROM Monitor should be located at address 0 of the first EPROM, communication area is located at the fixed address 1000 to 1FFF (8bit) or 2000 to 3FFF (16bit) of monitor code segment. The CPU address depends on the bus width of the EPROMs. The following table shows the address ranges occupied by the communication port:

bus width	start address	end address
8 bit	1000	1FFF
16 bit	2000	3FFF

**NOTE:**

If the ROM Monitor is not located at the begin of first EPROM the **Map.COMSTART offset** must be used. Offset is set to 1000h on a byte EPROM and 2000h on a word EPROM by default. The offset is calculated to (monitor start address - first EPROM start address + 1000h (byte) or 2000h (word)).

To integrate the monitor program into your application the following parts must be included:

- Startup Code to initialize peripherals and to jump to Monitor Code
- Vector Table (at least for vector 1, 2, 3)
- Monitor Program Code

The '.bin' and '.asm' files contain all parts of the monitor. The address layout of the default monitor is as follows:

```
0x00000--0x009FF      ; Monitor Code
0x00A00--0x00FFF      ; Free for initialization of peripherals
0x01000--0x03FFF      ; Monitor Communication Area (depends on bus
                        width)
```

## Vector Table

---

Some vectors (1, 2, 3) must be set up by the user to point into the monitor program code. The entry points are located at the beginning of the monitor.

vec	offs	ent	usage
00	000	+40	Reset (optional, can also go to application)
01	004	+50	Single Step Break
02	008	+50	Manual Break by NMI (optional)
03	00C	+50	Breakpoint Trap (used for breakpoints)
04	010	+60	Any unused trap maybe handled by monitor
..	...	...	"
17	044	+60	"

<b>NOTE:</b>	The entry point is given relative to the start of the monitor.
--------------	--

Format:	<b>SYStem.Mode</b> <mode>
<mode>:	<b>Down</b> <b>NoDebug</b> <b>Go</b> <b>Attach</b> <b>Up</b> <b>StandBy</b>

Default: Down. Selects the target operating mode.

<b>Down</b>	The CPU is in reset. Debug mode is not active. Default state and state after fatal errors.
<b>NoDebug</b>	The CPU is running. Debug mode is not active. Debug port is tristate. In this mode the target should behave as if the debugger is not connected.
<b>Go</b>	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command or if any break condition occurs.
<b>Attach</b>	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command (only serial monitor).
<b>Up</b>	The CPU is not in reset but halted. Debug mode is active. In this mode the CPU can be started and stopped. This is the most typical way to activate debugging.
<b>StandBy</b>	This mode is used to start debugging from power-on. The debugger will wait until power-on is detected, then bring the CPU into debug mode and start the CPU.

If the mode “Go” is selected, this mode will be entered, but the control button in the SYStem window jumps to the mode “UP”.

Format: **SYStem.CPU** *<mode>*

*<mode>*: **I8086 | I80186 | I80186EA | I80186EB | I80186EC | AM186EM | AM186ES | AM186ER | AM186ED | AM186CC**

Selects the processor type.

## SYStem.MemAccess

Select run-time memory access method

Format: **SYStem.MemAccess Enable | StopAndGo | Denied** *<cpu\_specific>*

### **Enable**

**CPU** (deprecated)

Memory access during program execution to target is enabled.

### **StopAndGo**

Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

### **Denied**

Memory access during program execution to target is disabled.

Default: Denied.

Format: **SYStem.CpuAccess** <sub\_cmd> (deprecated)

<sub\_cmd>: **Enable** (deprecated)  
**Use [SYStem.MemAccess StopAndGo](#) instead.**

**Denied** (deprecated)  
**There is no need to use a successor command (default setting).**

**Nonstop** (deprecated)  
**Use [SYStem.CpuBreak Denied](#) instead.**

Default: Denied.

Configures how memory access is handled during run-time.

<b>Enable</b>	Allow intrusive run-time memory access.
<b>Denied</b>	Lock intrusive run-time memory access.
<b>Nonstop</b>	Lock all features of the debugger that affect the run-time behavior.

Format:	<b>SYStem.CpuBreak</b> [ <i>&lt;mode&gt;</i> ]
<i>&lt;mode&gt;</i> :	<b>Enable   Denied</b>

Default: Enable.

<b>Enable</b>	Allows stopping the target.
<b>Denied</b>	<p>Denies stopping the target. This includes manual stops and stop breakpoints. However, short stops, such as spot breakpoints, may still be allowed.</p> <p><b>SYStem.CpuBreak Denied</b> can be used to protect a target system which does not tolerate that the program execution is stopped for an extended period of time; for example, a motor controller which could damage the motor if the motor control software is stopped.</p> <p>For more information, see <a href="#">SYStem.CpuSpot</a>, <a href="#">SYStem.MemAccess</a>.</p>

### Example:

```
SYStem.CpuBreak Denied

Break.Set main          ; stop breakpoint results in an error message

Break.Set main /Spot    ; spot breakpoint may be allowed
```



Format:	<b>SYStem.CpuSpot</b> [ <i>&lt;mode&gt;</i> ]
<i>&lt;mode&gt;</i> :	<b>Enable   Denied   Target   SINGLE</b>

Default: Enable.

Spotting is an intrusive way to transfer data periodically or on certain events from the target system to the debugger. As a result, the program is not running in real-time anymore. For more information, see [SYStem.CpuBreak](#) and [SYStem.MemAccess](#).

<b>Enable</b>	Allows spotting the target.
<b>Denied</b>	Denies spotting the target. Stopping the target may still be allowed.
<b>Target</b>	Allows spotting the target controlled by the target. This allows target-stopped <b>FDX</b> and <b>TERM</b> communication. All other spots are denied.
<b>SINGLE</b>	Allows single spots triggered by a command. This includes spotting for changing the breakpoint configuration and the <b>SNOOPeR.PC</b> command. This setting also allows target-stopped <b>FDX</b> and <b>TERM</b> communication. All other spots are denied.

**Example:**

```
SYStem.CpuSpot Denied
Break.Set main /Spot ; spot breakpoint results in an error message
```

# General SYStem Settings and Restrictions

---

## SYStem.Option.REL

Relocation register

---

Format:            **SYStem.Option.REL** <value>

**REL** option must be set to the same value the user program write to the REL register.

The adjusted I/O base address can be read back with the functions **IOBASE()** and **IOBASE.ADDRESS()**. They return the offset or the complete address (offset and access mode) for the I/O area.

## SYStem.Option.NIBBLE

Set global nibble flags

---

Format:            **SYStem.Option.NIBBLE** [ON | OFF]

Only ROM monitor with ESI

## SYStem.PORT

Set communication parameters

---

Format:            **SYStem.PORT** <settings>

<settings>:        **COM1 BAUD=9600**

Define the communication parameters.

```
SYStem.PORT COM1 baud=9600
```

## **Stack Memory**

The ROM debugger needs memory on the current stack. For only starting the Monitor and memory read or modify commands no is used. To start application, 22 bytes of stack are used. Modification of the EPROM while the monitor is running (Hot Patch) requires 52 bytes (at all) on the stack.

# TrOnchip Commands

---

## TrOnchip.CONVert

Adjust range breakpoint in on-chip resource

---

Format: **TrOnchip.CONVert [ON | OFF] (deprecated)**  
**Use [Break.CONFIG.InexactAddress](#) instead**

The on-chip breakpoints can only cover specific ranges. If a range cannot be programmed into the breakpoint, it will automatically be converted into a single address breakpoint when this option is active. This is the default. Otherwise an error message is generated.

```
TrOnchip.CONVert ON
Break.Set 0x1000--0x17ff /Write      ; sets breakpoint at range
Break.Set 0x1001--0x17ff /Write      ; 1000--17ff sets single breakpoint
...                                   ; at address 1001

TrOnchip.CONVert OFF
Break.Set 0x1000--0x17ff /Write      ; sets breakpoint at range
Break.Set 0x1001--0x17ff /Write      ; 1000--17ff
Break.Set 0x1001--0x17ff /Write      ; gives an error message
```

## TrOnchip.RESet

Set on-chip trigger to default state

---

Format: **TrOnchip.RESet**

Sets the TrOnchip settings and trigger module to the default settings.

## TrOnchip.state

Opens configuration panel

---

Format: **TrOnchip.state**

Control panel to configure the on-chip breakpoint registers.

# Memory Classes

---

Memory Class	Description
D	Data
P	Program
IO	I/O
C	Memory access by CPU
E	Emulation memory access
A	Absolute (physical) memory access