C166 Monitor

Version 18-Nov-2020

```
E::w.d.l
 addr/line  code              label      mnemonic                    comment
   P:0000B8 0000                         add      r0,r0
   P:0000BA 0000                         add      r0,r0
   P:0000BC A55AA5A5          ?C_START..:diswdt
   P:0000C0 0A863F1E                     bfldl    syscon,#1E,#3F
   P:0000C4 1A86007B                     bfldh    syscon,#7B,#0     ; syscon,#7B,#
   P:0000C8 E60A0CFA                     mov      stkov,#0FA0C
   P:0000CC E6010000                     mov      dpp1,#0           ; dpp1,#RESET
   P:0000D0 E6020100                     mov      dpp2,#1
   P:0000D4 DFE2                         bset     p3.0D
```

```
E::w.r
N   _  R0      0  R8      0  DPP0    0
C   _  R1      0  R9      0  DPP1    1
V   _  R2      0  R10     0  DPP2    2
Z   _  R3      0  R11     0  DPP3    3
E   _  R4      0  R12     0  SP   0FC00
MIP _  R5      0  R13     0  MDH     0
USR _  R6      0  R14     0  MDL     0
IEN _  R7      0  R15     0  MDC     0
ILV 0  PSW     0  CP  0FC00  CSP     0
```

```
E::w.per
PROCESSOR CONFIGURATION
SYSCON                 RDYEN dis SGTDIS
            BTYP  w/nomux  ROMEN  dis
BUSCON1                RDYEN dis
            BTYP  b/nomux

System Configuration
SYSCON04C0 STKSZ 256 RDYEN dis SGTDIS
            BTYP w/nomux MTTC 1w RWDC
```

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Basics - Training"** (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

NOTE:	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

General Note

This document describes the processor specific settings and features of the ROM monitor. You can find the description of the JTAG-166CBC Debugger for the C166CBC family at **"XC2000/XC16x/C166CBC Debugger"** (debugger_166cbc.pdf).

Quick Start of the C166 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

```
sys.cpu 167
```

4. Map the EPROM simulator. The mapping of the EPROM simulator is described in the section "**Mapping the EPROM simulator**".

5. Load the application program **(Data.LOAD)**.

```
d.load.o keilc1 /p
```

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 address 0 in the ROM area.

```
d.load.b rom166w.bin 0x400 /offset 0x400 /ny
d.a 0 jmps 0,440
d.a 20 jmps 0,450
d.a 28 jmps 0,470
```

7. Configure the Monitor program. The processor type is required for startup. Other parameters are optional.

```
d.s 0x400 1                    ; for C167 processor
```

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

```
x.respol -
x.nmipol -
x.nmibreak on
```

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

```
SYStem.Up
```

10. A typical start sequence is shown below:

```
; for this example the Ertec C167 Evaluation board is used
; the EPROM is in the addressrange 0x0--0x3ffff
; the RAM is in the addressrange 0x40000--0x7ffff

b:                                  ; select the debugger device
sys.d                               ; switch the system down
winclear                            ; clear all windows
sys.cpu 167                         ; set the CPU type for the user
                                    ; interface
map.res                             ; map the EPROM simulator
map.frag 0x1 0x0 0x0--0x0f9ff       ; part of EPROM below internal
                                    ; registers
map.gap 0x1 0x0fa00--0x0ffff        ; internal registers overlap EPROM

; part of EPROM after internal registers
map.frag 0x1 0x10000 0x10000--0x3ffff

map.bus16                           ; we use a 16 bit wide EPROM bus
d.load.o keilc1 /p                  ; load the application
d.load.b rom166w.bin 0x400          ; load the monitor
/offset 0x400 /ny

; setup vectors for monitor
d.a 0x0 jmps 0,440                  ; reset vector
d.a 0x20 jmps 0,450                 ; breakpoint entry (vector #8)
d.a 0x28 jmps 0,470                 ; bus error entry

; initialize the monitor configuration table

d.s sp:0x400 %byte 1                ; set the processor type
x.respol -                          ; adapt the polarity of RES and NMI
x.nmipol -
x.nmibreak on                       ; enables the connection of the NMI
                                    ; signal
SYStem.Up                           ; power the system up
```

Starting up the ROM Monitor is done as follows:

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

```
b:
```

12. Power the system down **before** pressing the reset button.

```
sys.d
```

This instruction is necessary when the system is restarted. When the system is active while you try to restart the bootstrap loader it will fail.

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

```
sys.cpu 167
```

14. Define the bootstrap loader. This program (binary version required) loads.

```
sys.bootldr2 myloader.bin
```

15. Define the ROM monitor.

```
sys.monitor ~~/demo/c166/monitor/rom166s.bin
```

16. Activate the bootstrap logic.

```
sys.o bootstrap on
```

17. Define the communication parameters.

```
sys.port COM2 baud=38400
```

18. Press the reset button on the target to activate the boot loader.
19. Activate the ROM monitor

```
SYStem.Up
```

20. Load the application program.

```
d.load.o keilcl /p
```

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). If the application program overwrite the vectors for the ROM monitor, they must be restored after the load.

21. Configure the Monitor program. The processor type is required for startup.

```
d.s 400 1                       ; for C167 processor
```

22. Enable the interrupts. This is only required, when the NMI is not.

```
r.s ien 1
```

A typical start sequence is shown below:

```
; for this example the Ertec C167 Evaluation board is used

b:                              ; select the Debugger device
sys.d                           ; switch the system down
winclear                        ; clear all windows
sys.cpu 167                     ; set the CPU type for the user interface
sys.port COM2 baud=38400

sys.bootldr ~~/demo/c166/monitor/ertec167.bin

sys.monitor ~~/demo/c166/monitor/rom166s.bin

sys.u
d.load.o keilcl /p              ; load the application
d.s sp:400 %byte 1              ; initialize the monitor configuration table
r.s ien 1                       ; set the processor type
```

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

Troubleshooting

No information available.

FAQ

EPROM Simulator Error on Data Modification Ref: 0056	**Why does the ROM monitor crash after modification of EPROM?** Check that there is enough space left on the stack. See also "Restrictions for Stack Requirements".
Step or Breakpoint Fails Ref: 0061	**Why does single step or breakpoint not work?** 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.
Stepping Fails when Executing MOV SP,xxx Ref: 0062	**Why does stepping fail, when executing a MOV SP,xxx instruction?** Check that there is enough space left on the stack before and after the execution of the instruction. See "Restrictions for Stack Requirements". 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".
80166 Stepping Fails after Enabling the Interrupts Ref: 0101	**Why does stepping fails after enabling the interrupts?** Check the "disable interrupt" configuration word in the monitor configuration table. For the first try it should be set to 1 to disable all interrupts. Enabling the interrupts in the monitors is only useful in some very specific applications.
80166 Target Peripherals cannot be modified by Data.Set Ref: 0100	**Why can target peripherals not be modified by "Data.Set"?** Make sure that there is no EPROM mapped in that area. Use "MAP.FRAG" and "MAP.GAP" where appropriate. On some processors the write line must be enabled before the CPU can access the external memory.

Monitor Features

The monitor requires no stack during startup and memory operations. A valid stack is only required for modifications in the EPROM while the monitor is running (Hot Patch) and for single step and go commands. This allows to use the monitor even when the stack has overflowed or underflowed. External memory or fixed locations in internal memory are not required. This allows to debug also small applications which run without external memory. The NMI pin of the Eprom Simulator can be used to manually stop the target program.

Monitor Files

The 'rom166' and 'rom166w' monitors are for Eprom Simulator solutions (8bit and 16bit), while the 'rom166e' monitor is used as foreground monitor for Emulators. By using a foreground monitor the target program can be single stepped without stopping the target processors interrupts or PEC transfers. Both monitors have the same source file 'rom166.asm'. This source file should not be modified, it is only included for reference purposes. There are to 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 Rom Monitor and its configuration table is located from address 0400 to 0FFF. The communication area for the Eprom Simulator is located at the fixed address 1000 to 1FFF of the first EPROM. 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

The monitor program consists of three parts:

- Vector Table
- Configuration Table
- Monitor Program Code

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

```
0000--03FF                      ; Vector Table
0400--041F                      ; Configuration Table
0420--0FFF                      ; Monitor Code
```

The alternate monitor version rom166wu is linked to 1400..1FFF.

Vector Table

For the first tests of a software, the '.bin' files can be loaded with vector and configuration table. When the vector table becomes part of the application, it is not loaded with the monitor. Instead the table is setup according to the application (the table may also reside in RAM). Some vectors must be set up to point into the monitor program code. The entry points are located at the beginning of the monitor.

vec	offs	ent	usage
00	000	+20	Reset (optional, can also go to application)
02	008	+30	Manual Break by NMI (optional)
08	020	+30	Breakpoint Trap, (used for breakpoints, can be changed)
0A	028	+50	Bus Error (optional, when Bus Errors should enter monitor)
..	...	+40	Any unused vector may be handled by the monitor

NOTE: The entry point is given relative to the start of the monitor, thus the absolute address for the reset entry point is 440H. The breakpoint trap vector can be configured by the **SYStem.Option BrkVector** command. The default is vector 8 (at location 20). When interrupts are not allowed in the application, a value of 0 will force an undefined opcode to be used for breakpoints.

Configuration

The configuration table of the monitor must always be located directly before the monitor code. The default location used in the binary files is 400 (hex).

- Processor core type (byte at offset 00H):

  00 = 80C166 (default)

  01 = C165, C167

- Watchdog enable/disable (byte at offset 02H):

  00 = Watchdog disabled (default)

  01 = Watchdog enabled

- Monitor Interrupt Level (word at offset 04H)

  0000 = all interrupts enabled in monitor

  0001 = all interrupts locked by clearing the IEN bit

  1000 = interrupt level for PSW register

  …

SYStem.CPU — CPU type

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

<*mode*>: **161** | **163** | **164** | **165** | **166** | **167**

Selects the processor type. The ROM debugger requires also a modification in the debug monitor for different processor types.

SYStem.MemAccess — Real-time memory access (non-intrusive)

Format: **SYStem.MemAccess Enable**| **StopAndGo** | **Denied**<*cpu_specific*>

Enable
CPU (deprecated) — Real-time 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 — Real-time memory access during program execution to target is disabled.

Default: Denied.

Format: **SYStem.CpuAccess Enable** | **Denied** | **Nonstop**

Default: Denied.

Enable Allows intrusive run-time memory access.
In order to perform a memory read or write while the CPU is executing the program, the debugger stops the program execution shortly. Each short stop takes 1 … 100 ms depending on the speed of the debug interface and on the number of the read/write accesses required.
A white S against a red background in the state line of the TRACE32 main window indicates this intrusive behavior of the debugger.

Denied Locks intrusive run-time memory access.

Nonstop Locks all features of the debugger that affect the run-time behavior.
Nonstop reduces the functionality of the debugger to:

- Run-time access to memory and variables
- Trace display

The debugger inhibits the following:

- To stop the program execution
- All features of the debugger that are intrusive (e.g. action **Spot** for breakpoints, performance analysis via **StopAndGo** mode, conditional breakpoints, etc.)

SYStem.Mode Establish the communication with the CPU

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

<*mode*>: **Down**
NoDebug
Go
Attach
Up
StandBy

Default: Down. Selects the target operating mode.

Down	The CPU is in reset. Debug mode is not active. Default state and state after fatal errors.
NoDebug	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.
Go	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.
Attach	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command (only serial monitor).
Up	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.
StandBy	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”.

SYStem.Option BrkVector — Breakpoint trap

Format: **SYStem.Option BrkVector** <*trap*>

Defines the trap number used for breakpoints and single stepping. The default is trap #8.

only serial monitor

SYStem.Option ResVector — Resetvector trap

Format: **SYStem.Option ResVector** <*trap*>

The default is 0x0.

only serial monitor

SYStem.Option BusType — Bus mode

Format: **SYStem.Option BusType** *<mode>*

<mode>:
NOMUX8
MUX8
NOMUX16
MUX16

Selects the bus mode for the processor.

NOMUX8 Non-multiplexed 8 bit bus. Port 0L is data port, Port 1 and 4 are address ports.

MUX8 Multiplexed 8 bit bus. Port 0 is used for address and data.

NOMUX16 Non-multiplexed 16 bit bus. Port 0 is data bus, port 1 and 4 are address signals.

MUX16 Multiplexed 16 Bit bus. Port 0 is address and data bus. The upper address lines (segments) are supported on port 4.

SYStem.Option CS — Chip selects

Format: **SYStem.Option CS** *<size>*

<size>:
0
2
3
5

The reset vector for the chip selects is defined for all C167 probes.

SYStem.Option ADDRSELx — BUSCON settings

Format: **SYStem.Option ADDRSELx** | **BUSCONx** | **SYSCON** *<value>*

Settings only for ICD Trace.

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

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during assembler single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.Option IMASKHLL Disable interrupts while HLL single stepping

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

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.Option SGT Segmentation

Format: **SYStem.Option SGT** <*size*>

<*size*>: **OFF**
256K
1M
16M

The segmentation must be set up for all non-bondout probes. However it is recommended on the C167 bondout probe for correct address mirroring. The setup defines the reset vector in stand-alone mode.

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

Activate the bootstrap logic

only Serial Monitor

SYStem.BOOTLDR2 Bootloader file

Format: **SYStem.BOOTLDR2** <*file*>

Define the BootLoader file

only Serial Monitor

SYStem.MONITOR Monitor file

Format: **SYStem.MONITOR** <*file*>

Define the ROM monitor

only Serial Monitor

SYStem.PORT Set communication parameters

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

<*settings*>: **COM1 BAUD=9600**

Define the communication parameters.

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

DPP(<*offset*>)

Returns the memory addressed by the short pointer argument. The lower 14 bits of the argument hold the offset, the two upper bits the DPP selector.

General Restrictions

Stack Memory

The ROM debugger needs 10 words of memory on the current stack. The stack is not required for starting the Monitor and memory read or modify commands. Modification of the EPROM while the monitor is running (Hot Patch) requires 16 words on the stack.

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

TrOnchip.VarCONVert — Adjust complex breakpoint in on-chip resource

Format: **TrOnchip.VarCONVert** [**ON** | **OFF**] (deprecated)
Use Break.CONFIG.VarConvert instead

The on-chip breakpoints can only cover specific ranges. If you want to set a marker or breakpoint to a complex variable, the on-chip break resources of the CPU may be not powerful enough to cover the whole structure. If the option **TrOnchip.VarCONVert** is set to **ON**, the breakpoint will automatically be converted into a single address breakpoint. This is the default setting. Otherwise an error message is generated.

Format: **TrOnchip.RESet**

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

TrOnchip.state Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.TEnable Set filter for the trace

Format: **TrOnchip.TEnable** <*par*> (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TOFF Switch the sampling to the trace to OFF

Format: **TrOnchip.TOFF** (deprecated)

Refer to the **Break.Set** command to set trace filters.

Format: **TrOnchip.TON EXT** | **Break** (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TTrigger Set a trigger for the trace

Format: **TrOnchip.TTrigger** <*par*> (deprecated)

Refer to the **Break.Set** command to set a trigger for the trace.

Memory Class	Description
D	Data
P	Program
B	Bit
C	Memory access by CPU
E	Emulation memory access
A	Absolute (physical) memory access