

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

<a href="#">TRACE32 Documents</a> .....	
<a href="#">ICD In-Circuit Debugger</a> .....	
<a href="#">Processor Architecture Manuals</a> .....	
<a href="#">DSP56K</a> .....	
<a href="#">DSP56K Debugger</a> .....	<b>1</b>
<a href="#">Brief Overview of Documents for New Users</a> .....	<b>4</b>
<a href="#">Warning</a> .....	<b>5</b>
<a href="#">Quick Start</a> .....	<b>6</b>
<a href="#">Troubleshooting</a> .....	<b>9</b>
<a href="#">SYStem.Up Errors</a> .....	9
<a href="#">FAQ</a> .....	<b>10</b>
<a href="#">Configuration</a> .....	<b>13</b>
<a href="#">On-chip Flash Programming and Debugging on 56F8xxx Derivatives</a> .....	13
<a href="#">General SYStem Settings and Restrictions</a> .....	<b>16</b>
SYStem.CPU .....	Select the used CPU 16
SYStem.CpuAccess .....	Run-time memory access (intrusive) 16
SYStem.LOCK .....	Lock and tristate the debug port 17
SYStem.MemAccess .....	Real-time memory access (non-intrusive) 17
SYStem.Mode .....	Establish the communication with the target 18
SYStem.CONFIG .....	Configure debugger according to target topology 19
<a href="#">Daisy-chain Example</a> .....	21
<a href="#">TapStates</a> .....	22
SYStem.CONFIG.CORE .....	Assign core to TRACE32 instance 23
SYStem.Option COP .....	Enable WATCHDOG 24
SYStem.Option DE .....	Enable DE line 24
SYStem.Option IMASKASM .....	Disable interrupts while single stepping 24
SYStem.Option IMASKHLL .....	Disable interrupts while HLL single stepping 25
SYStem.Option SoftBreakFix .....	Enables "SoftBreakFix" patch 25
SYStem.JtagClock .....	Define JTAG clock 26
<a href="#">General Restrictions</a> .....	27
<a href="#">FPU</a> .....	<b>30</b>

<b>TrOnchip Commands</b> .....	<b>31</b>
TrOnchip.view	Opens configure panel 31
TrOnchip.A	Trigger cycle 31
TrOnchip.AANDB	Triggers if event occurs on unit A and unit B 32
TrOnchip.AAFTERB	Triggers if event occurs first on unit A and then on unit B 32
TrOnchip.AORB	Triggers if event occurs on unit A or unit B 32
TrOnchip.B	Trigger cycle 32
TrOnchip.BAFTERA	Triggers if event occurs first on unit B and then on unit A 33
TrOnchip.CONVert	Automatically convert range to single address 33
TrOnchip.Count	Delay counter 33
TrOnchip.DMA	Trigger on DMA access 33
TrOnchip.Mode	Defines used triggers 34
TrOnchip.OFF	Disable on-chip trigger unit 34
TrOnchip.RESet	Resets settings 34
<b>Floating Point Formats</b> .....	<b>35</b>
<b>Integer Access Keywords</b> .....	<b>35</b>
<b>ONCE Connector (56002/56100)</b> .....	<b>36</b>
<b>JTAG Connector (56300, 56800, 56800E)</b> .....	<b>37</b>
<b>Memory Classes</b> .....	<b>39</b>
<b>Support</b> .....	<b>40</b>
Available Tools	40
Compilers DSP56000	42
Compilers DSP56300	42
Compilers DSP56800/DSP56800E	42
Realtime Operation Systems	43
3rd Party Tool Integrations	43
<b>Products</b> .....	<b>45</b>
Product Information	45
Order Information	45

The screenshot displays the TRACE32 DSP56K Debugger interface with the following components:

- TRACE32 DSP56 [POWER TRACE ETHERNET]** window title.
- Menu Bar:** File, Edit, View, Var, Break, Run, CPU, Misc, Trace, Perf, Cgv, Window, Help.
- Toolbar:** Standard debugger controls including Step, Over, Next, Return, Up, Go, Break, Mode, Find, and Stop.
- B::Register:** A table showing register values for C, V, Z, N, R0-R7, VBA, PABEX, PABD, and PABF.
- B::Data.dump (X:0x0):** A memory dump table with columns for address and data.
- B::SYSTEM:** Configuration panel for Mode (Up), MemAccess (Denied), CpuAccess (Enabled), Option (IMASKASM, IMASKHLL, DE, COP), JtagClock (5.0MHz), and CPU (MC56303).
- [B::Data.List]:** A list of assembly instructions with addresses and comments, including 'loop\_1' and 'process\_stereo'.
- B::Break.List:** A table of breakpoints with columns for address, types, and impl.
- Bottom Panel:** Buttons for emulate, trigger, devices, trace, Data, Var, PERF, SYSTEM, Step, Go, other, and previous. The status bar shows 'P:000139 \pass\pass\loop\_1+0x6 stopped at breakpoint HLL UP'.

## 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**.
- **"RTOS Debugger"** (rtos\_<x>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate RTOS manual informs you how to enable the OS-aware debugging.

**NOTE:**

To prevent debugger and target from damage it is recommended to connect or disconnect the debug cable only while the target power is OFF.

Recommendation for the software start:

1. Disconnect the debug cable from the target while the target power is off.
2. Connect the host system, the TRACE32 hardware and the debug cable.
3. Power ON the TRACE32 hardware.
4. Start the TRACE32 software to load the debugger firmware.
5. Connect the debug cable to the target.
6. Switch the target power ON.
7. Configure your debugger e.g. via a start-up script.

Power down:

1. Switch off the target power.
2. Disconnect the debug cable from the target.
3. Close the TRACE32 software.
4. Power OFF the TRACE32 hardware.

Starting up the debugger is done as follows:

Select the device prompt for the ICD Debugger and reset the system.

```
b: :  
RESet
```

The device prompt `B : :` is normally already selected in the command line. If this is not the case enter `B : :` to set the correct device prompt. The `RESet` command is only necessary if you do not start directly after booting the TRACE32 development tool.

5. Specify the CPU specific settings.

```
SYStem.CPU <cpu>
```

The default values of all other option are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

6. Set the JTAG shift frequency

```
SYStem.JtagClock <frequency>
```

Normally the default value is 1.0 MHz, but the 56800E requires a lower value in the starting process.

7. Inform the debugger about read only address ranges (ROM, FLASH).

```
MAP.BOnchip <range>
```

The B(reak)Onchip information is necessary to decide where on-chip breakpoints must be used. On-chip breakpoints are necessary to set program breakpoints to read-only memories. The sections of FLASH and ROM depend on the specific CPU and its chip selects.

8. Enter debug mode.

```
SYStem.Up
```

This command resets the CPU and enters debug mode. After this command is executed it is possible to access memory and registers.

9. Configure chip according application.

```
Register.Set OMR 3 ; 56800: Development mode
```

Before loading binary data into the processor memory, the memory should be made writable. Therefore processor configuration registers have to be set e.g. OMR, SR or chip select register. The flash of the 56F8300 derivatives should be initialized here, too.

10. Load the program.

```
Data.LOAD.Elf program.elf ; ELF specifies the format,  
; program.elf is the file name
```

The format of the Data.LOAD command depends on the file format generated by the compiler. Refer to [Supported Compilers](#) to find the command, that is necessary for your compiler.

A detailed description of the [Data.LOAD](#) command and all available options is given in the “[General Commands Reference](#)”.

A typical start sequence for the DSP56858 is shown below. This sequence can be written to an ASCII file (script file) and executed with the command **DO** <filename>. Other sequences can be found on the CD in the DEMO directory.

```
b::                                ; Select the ICD device prompt
WinCLEAR                            ; Clear all windows
SYStem.CPU 56858                    ; Select CPU (56800E class here)
SYStem.JtagClock 687000.            ; Choose JTAG frequency
MAP.BOnchip 0x1f000..1f03ff         ; Specify where read-only memory is
SYStem.Up                            ; Reset the target and enter debug
                                      ; mode
Register.Set PP 0x1F000              ; Set the extended program counter PP
                                      ; (not PC!) to the begin of the boot
                                      ; flash. The statement is redundant in
                                      ; this case, but remember the
                                      ; execution without loading a program
                                      ; starts here.
Register.Set OMR 0x0                ; Prepare access to memory by using
                                      ; operating mode 0
Data.LOAD.Elf                       ; Load the application with option
ldm_external_memory.elf /LARGE      ; large memory model and verify the
/VERIFY                              ; process
Go main                              ; Run and break at main()
Data.List                            ; Open source window
Register /SpotLight                 ; Open register window
Var.Local                            ; Open window with local variables
```

## SYStem.Up Errors

---

The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

- The JTAG lines are not connected correctly.
- The target has no power.
- The pull-up resistor between the JTAG[VCCS] pin and the target VCC is too large.
- The target is in reset:

The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up. Therefore no external R-C combination or external reset controller is allowed.

- There is logic added to the JTAG state machine:

By default the debugger supports only one processor in one JTAG chain. If the processor is the only one member of a JTAG chain the debugger has to be informed about the target JTAG chain configuration. Use the SYStem.CONFIG command to specify the position of the device in the JTAG-chain. Debuggers for DSP56000 and DSP56100 do not support the SYStem.CONFIG options! For the DSP56800 chips the support depends to the license. There is a license upgrade available which also allows to debug 56800E core based chips. For Multicore DSP56300 systems e.g. DSP56720 or DSP56721 a Multicore License is necessary.

- Wrong CPU is selected
- JTAG clock is too high, especially for 56800E core based processors
- CPU executed illegal code and is in a bad state that can be only be reverted by re-powering the target. To avoid this situations first plug the debugger to the target, then power the target. The debugger will keep the target in RESET state until the command **SYStem.Up** was successful.

There are additional loads or capacities on the JTAG lines

Debugging via  
VPN

**The debugger is accessed via Internet/VPN and the performance is very slow. What can be done to improve debug performance?**

The main cause for bad debug performance via Internet or VPN are low data throughput and high latency. The ways to improve performance by the debugger are limited:

in practice scripts, use "SCREEN.OFF" at the beginning of the script and "SCREEN.ON" at the end. "SCREEN.OFF" will turn off screen updates. Please note that if your program stops (e.g. on error) without executing "SCREEN.OFF", some windows will not be updated.

"SYStem.POLLING SLOW" will set a lower frequency for target state checks (e.g. power, reset, jtag state). It will take longer for the debugger to recognize that the core stopped on a breakpoint.

"SETUP.URATE 1.s" will set the default update frequency of Data.List/Data.dump/Variable windows to 1 second (the slowest possible setting).

prevent unneeded memory accesses using "MAP.UPDATEONCE [address-range]" for RAM and "MAP.CONST [address--range]" for ROM/FLASH. Address ranged with "MAP.UPDATEONCE" will read the specified address range only once after the core stopped at a breakpoint or manual break. "MAP.CONST" will read the specified address range only once per SYStem.Mode command (e.g. SYStem.Up).

<p>Setting a Software Breakpoint fails</p>	<p><b>What can be the reasons why setting a software breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to implement the wanted breakpoint.</p> <p>Possible reasons:</p> <p>The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller.</p> <p>Example: Read, write and access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</p> <p>TRACE32 can not change the memory.</p> <p>Example: ROM and Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All type of memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</p> <p>Contrary settings in TRACE32.</p> <p>Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</p> <p>RTOS and MMU:</p> <p>If the memory can be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</p>
<p>Cannot release from Software Breakpoint</p>	<p><b>Cannot make an HLL-step from a software breakpoint or the context switches while debugging</b></p> <p>Possible reason: Interrupts are pending</p> <p>In HLL-stepping mode always debug with SYStem.Option IMASKASM ON, when interrupts are pending.</p> <p>Possible reason: Routine is called again</p> <p>Debugging in recursive routines can have strange side effects. The debugger uses breakpoints to perform HLL single steps. These breakpoints will place the processor in debug state also when you try to step over a self call. So the stackframe/context can switch or in case of fixed user breakpoints it seems that the debugger cannot release from a breakpoint. Similar behavior can be expected for debugging in nested interrupts or real time operating systems with software interrupts.</p>
<p><b>56800/56800E</b></p> <p>PC Register is not equal to Program Counter</p>	<p><b>How can I set the program counter to including the extension bit from the SR register?</b></p> <p>Use the register PP e.g. Register.Set PP 0x1F0000.</p>

<p><b>56800E</b></p> <p>AnySymbol Option for Metrowerks 800E</p>	<p><b>Debugger does not step correctly into routine or debugger does not load HLL code for routine.</b></p> <p>Use the /AnySymbol paramater when you load Metrowerks ELF files.</p>
<p><b>56800E</b></p> <p>Pointer has wrong Width</p>	<p><b>Pointer type has wrong width.</b></p> <p>The width of the pointer type depends on the memory model. The large memory model requieres the parameter /LARGE for the command Data.LOAD.ELF.</p>

The processor type must be selected by the **SYStem.CPU** command before issuing any other target related commands.

## On-chip Flash Programming and Debugging on 56F8xxx Derivatives

---

TRACE32 offers target based flash programming for the internal flash on the 56F8xx and 56F8300 derivative. Before accessing the flash the device has to be configured. Example scripts for programming and debugging can be found in */demo/dsp56800/flash* and */demo/dsp56800e/compiler/mwerks/56f8323*.

### Configuration for flash programming:

- Select the CPU with **SYStem.CPU** and use **SYStem.Up** to enable debug mode.
- Optional: Adjust the processors system clock (SYS\_CLK) to allow a faster JTAG communication and shorter flash algorithm runtime.

Consult the processors architecture manual for the right PLL and Clock settings. An example for the 56F8323 can be found under *demo/dsp56800e/hardware/dsp568323demo/system.up*

- Set the chip configuration register and peripherals to enable access to the memory sections with flash.

Check the processor manual for correct setting of SR, OMR registers (especially EX bit and MODE bits) and the chip select peripheral register. The exact configuration depends on your application. Use the commands **Register.Set** and **Data.Set** to modify these registers.

- Configure the Flash programming

Use the TRACE32 commands **FLASH.Create** and **FLASH.TARGET** to inform the debugger about Flash memory sections and the used flash algorithm. TRACE32 provides example scripts for all known derivatives with the configuration EX=0 and Mode=0. Look in the *demo/dsp56800/flash* and *demo/dsp56800e/flash* directories for these scripts.

The implementation of the **FLASH.Create** command differs from the standard:

- The physical range addresses are counted in words.
- The sector size is passed in bytes.
- The bus width is fixed to "Word".
- Additional access class parameter for 56800 and 56800E family.

**56800 family:** An additional parameter after the access size parameter is necessary. The 32 bit parameter tells the target program the address of the flash controller base register and the flash memory class to use. Bit 0..15 of this parameter give the base address of the flash controller registers, bit 16..31 specify the access class. Access class:

- 0 : Program flash memory or boot flash memory
- 1 : Data flash memory

Example:

```
; Program flash, control base is 0x1020  
FLASH.Create 1. P:0x0000--0x7bff 0x200 TARGET Word 0x01020  
; Data flash, 0x10000 + control base 0x1060  
FLASH.Create 2. X:0x1800--0x1fff 0x200 TARGET Word 0x11060
```

**56800E family:** An additional parameter after the access size parameter is necessary, that tells the target program about the flash memory class. Access class:

1 : Boot flash  
2 : Program flash  
3 : Data flash.

Example:

```
; Boot Flash  
FLASH.Create 1. P:0x020000--0x020FFF 0x200 TARGET Word 1  
; Program Flash  
FLASH.Create 2. P:0x000000--0x003FFF 0x400 TARGET Word 2  
; Data Flash  
FLASH.Create 3. X:0x001000--0x001FFF 0x200 TARGET Word 3
```

- Enable Flash programming and download application  
Use **FLASH.AUTO ALL** to enable cached write access to the flash memory and download your application with **Data.Load**. Alternatively it is also possible to use **FLASH.Erase** and **FLASH.Program**, especially when large memory blocks have been changed.
- Disable Flash programming with **FLASH.AUTO OFF** or **FLASH.Program OFF**.

## Configuration for debugging in flash:

- Select the CPU with **SYStem.CPU** and use **SYStem.Up** to enable debug mode.
- Load Symbols

Assuming that the application is already programmed into flash, load the symbols with the help of the TRACE32 command **Data.Load** with the additional parameters /NoCODE, /NOREG, /AS and optional /LARGE if you use the large memory model.
- Configure the flash programming
- Execute start-up code to configure the device

The start-up code of your application includes normally instructions to configure the chip registers and peripherals. You can execute the start-up code with the TRACE32 instruction “**Go main /ONCHIP**”. The command lets the processor execute the code and breaks at *main* with the help of an on-chip breakpoint.
- Adjust JTAG-Clock and enable flash memory for write access.

Assuming that the PLL is initialized correctly by the start-up code, the JTAG frequency can be optimized to allow faster communication. The TRACE32 command **FLASH.AUTO ALL** enables the flash for writing. This is necessary for debugging in flash. The executed program code should not change the system clock (SYS\_CLK) otherwise the flash clock divider (can only be set one time after reset) and JTAG frequency can become invalid.
- Debug your application

The step-over function uses asm single steps to perform, because this safes flash life cycles. For faster operation it is better to use break and go commands.
- Shutdown the processor with **FLASH.AUTO OFF** and **SYStem.Down** to replace the software breakpoints with the original application code.

## SYStem.CPU

Select the used CPU

Format:	<b>SYStem.CPU</b> <i>&lt;cpu&gt;</i>
<i>&lt;cpu&gt;</i> :	<b>56002   56004   56005   56006   56007 (56000 processors)</b> <b>56301   56302   56303   56307   56309   56311   56321   56362   56364   56366</b> <b>  56367   56371 (56300 processors)</b>  <b>56801   56803   56805   56807   56809   56811   56827 (56800 processors)</b>  <b>56852   56853   56854   56855   56857   56858 (56800E 5685x processors)</b>  <b>56F8322   56F8323   56F8345   56F8346   56F8347   56F8355   56F8356  </b> <b>56F8357   56F8365   56F8366   56F8367 (56800E 56F83xx processors)</b>  <b>56F8122   56F8123   56F8145   56F8146   56F8147   56F8155   56F8156  </b> <b>56F8157   56F8165   56F8166   56F8167 (56800E 56F81xx processors)</b>  <b>56F8013   56F8014 (56800E 56F80xx processors)</b>

Selects the processor type.

## SYStem.CpuAccess

Run-time memory access (intrusive)

Format:	<b>SYStem.CpuAccess</b> <b>Enable   Denied   Nonstop</b>
---------	--

Default: Denied.

### Enable

Allow 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 red S in the state line of the TRACE32 screen indicates this intrusive behavior of the debugger.

**Denied** Lock intrusive run-time memory access.

**Nonstop** Lock 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 break-points, performance analysis via StopAndGo mode, conditional break-points etc.)

## SYStem.LOCK

Lock and tristate the debug port

Format: **SYStem.LOCK [ON | OFF]**

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the lock command is to give debug access to another tool.

## SYStem.MemAccess

Real-time memory access (non-intrusive)

Format: **SYStem.MemAccess CPU | Denied**<cpu\_specific>  
**SYStem.ACCESS** (deprecated)

**CPU** Real-time memory access during program execution to target is enabled.

**Denied** Real-time memory access during program execution to target is disabled.

Default: Denied.

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

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

<b>Down</b>	Disables the debugger (default). The state of the CPU remains unchanged. The JTAG port is tristated.
<b>NoDebug</b>	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
<b>Go</b>	Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or external trigger.
<b>Attach</b>	User program remains running (no reset) and the debug mode is activated. After this command the user program can be stopped with the break command or if any break condition occurs.
<b>Up</b>	Resets the target, sets the CPU to debug mode and stops the CPU. After the execution of this command the CPU is stopped and all register are set to the default level.
<b>StandBy</b>	Not available for DSP56K.

Format:	<b>SYSystem.CONFIG</b> <parameter> <number_or_address> <b>SYSystem.MultiCore</b> <parameter> <number_or_address> (deprecated)
<parameter> (General):	<b>state</b> <b>CORE</b> <core>
(JTAG):	<b>DRPRE</b> <bits> <b>DRPOST</b> <bits> <b>IRPRE</b> <bits> <b>IRPOST</b> <bits> <b>TAPState</b> <state> <b>TCKLevel</b> <level> <b>TriState</b> [ON   OFF] <b>Slave</b> [ON   OFF]

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. ARM + DSP). The information is required before the debugger can be activated e.g. by a **SYSystem.Up**. See **Daisy-chain Example**.

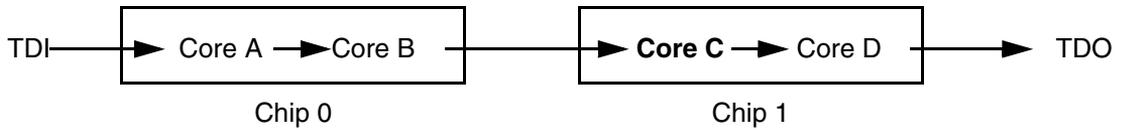
For some CPU selections (**SYSystem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

TriState has to be used if several debuggers (“via separate cables”) are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode. Please note: nTRST must have a pull-up resistor on the target, TCK can have a pull-up or pull-down resistor, other trigger inputs needs to be kept in inactive state.

	<p>Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).</p>
---	--

<b>state</b>	Show multicore settings.
<b>CORE</b>	For multicore debugging one TRACE32 GUI has to be started per core. To bundle several cores in one processor as required by the system this command has to be used to define core and processor coordinates within the system topology. Further information can be found in <a href="#">SYStem.CONFIG.CORE</a> .
<b>DRPRE</b>	(default: 0) <number> of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.
<b>DRPOST</b>	(default: 0) <number> of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.
<b>IRPRE</b>	(default: 0) <number> of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.
<b>IRPOST</b>	(default: 0) <number> of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.
<b>TAPState</b>	(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.
<b>TCKLevel</b>	(default: 0) Level of TCK signal when all debuggers are tristated.
<b>TriState</b>	(default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.
<b>Slave</b>	(default: OFF) If more than one debugger share the same debug port, all except one must have this option active. JTAG: Only one debugger - the "master" - is allowed to control the signals nTRST and nSRST (nRESET).

## Daisy-chain Example



Below, configuration for core C.

Instruction register length of

- Core A: 3 bit
- Core B: 5 bit
- Core D: 6 bit

```
SYStem.CONFIG.IRPRE 6 ; IR Core D
SYStem.CONFIG.IRPOST 8 ; IR Core A + B
SYStem.CONFIG.DRPRE 1 ; DR Core D
SYStem.CONFIG.DRPOST 2 ; DR Core A + B
SYStem.CONFIG.CORE 0. 1. ; Target Core C is Core 0 in Chip 1
```

0	Exit2-DR
1	Exit1-DR
2	Shift-DR
3	Pause-DR
4	Select-IR-Scan
5	Update-DR
6	Capture-DR
7	Select-DR-Scan
8	Exit2-IR
9	Exit1-IR
10	Shift-IR
11	Pause-IR
12	Run-Test/Idle
13	Update-IR
14	Capture-IR
15	Test-Logic-Reset

```
Format:          SYStem.CONFIG.CORE <coreindex> <chipindex>
                SYStem.MultiCore.CORE <coreindex> <chipindex> (deprecated)

<chipindex>:    1 ... i

<coreindex>:    1 ... k
```

Default *coreindex*: depends on the CPU, usually 1. for generic chips

Default *chipindex*: derived from CORE= parameter of the configuration file (config.t32). The CORE parameter is defined according to the start order of the GUI in T32Start with ascending values.

To provide proper interaction between different parts of the debugger the systems topology must be mapped to the debuggers topology model. The debugger model abstracts chips and sub-cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected a generic chip or none generic chip is created at the default *chipindex*.

### None Generic Chips

None generic chips have a fixed amount of sub-cores with a fixed CPU type.

First all cores have successive chip numbers at their GUIs. Therefore you have to assign the coreindex and the chipindex for every core. Usually the debugger does not need further information to access cores in none generic chips, once the setup is correct.

### Generic Chips

Generic chips can accommodate an arbitrary amount of sub-cores. The debugger still needs information how to connect to the individual cores e.g. by setting the JTAG chain coordinates.

### Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a none generic chip, two GUI are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is value since every new GUI uses a new *chipindex* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

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

Default: OFF.

The watchdog remains active when this option is set to ON. This option is not necessary for the 56800E processors because their watchdog is disabled automatically in debug mode.

## SYStem.Option DE

## Enable DE line

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

Default: ON.

Enables the use of the /DE line on the JTAG connector. This increases the speed of the debugger for 56300 and 56800 processors. The 56800E processors can perform the **SYStem.Up** command faster when DE is activated.

## SYStem.Option IMASKASM

## Disable interrupts while single stepping

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. For 56800E processors IMASKASM ON is necessary for HLL stepping and stepping from software breakpoints.

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.

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

Default: OFF.

If enabled an experimental patch gets active, which allows to set breakpoints in delay slots of conditional branches (only 56800E processors). The patch is only useful, if the "pad pipeline" options for compiler and assembler in the Metrowerks Codewarrior are disabled. The patch can have strange side effects e.g. in code which reads data from p memory (case statements) or the debugger halts near a breakpoint.

Format:	<b>SYStem.JtagClock</b> <rate> <b>SYStem.BdmClock</b> <rate> (deprecated)
<fixed>:	<b>976. ... 15 000 000.</b>

Selects the frequency for the debug interface.

- For a fast setup of the clock speed the pre-configured buttons can be used to enter the clock speed. These are the most used frequencies fixed. The default frequency for the fixed clock is 1 MHz.
- The clock speed depends on the speed of the processor. Especially the 56800E processors need a frequency lower or equal to 500000.Hz at SYStem.UP. After initializing the PLL to a core clock of 120 MHz the JTAG clock can be increased up to 15 MHz. The other processors can operate with the default JTAG clock setting at SYStem.UP.

**NOTE:**

Buffers, additional loads or high capacities on the JTAG/COP lines reduce the debug speed.

## **SYS\_CLK cannot be changed while flash-programming is active**

The flash clock divider depends on the sys\_clk. Since the flash clock diver can only be set one time after system up, the flash clock divider cannot be adapt to a modified system clock (SYS\_CLK). The restriction can cause a “flash timing error”

## **Program Modifications**

When the program is modified, the contents of the PIL and PDB registers are not changed. If the modified address is already fetched, the processor will execute the old instructions. Modifying the PIL or PDB register, or setting the PC will cure this problem. Program modifications by the debugger, like software breakpoints, consider all pipeline effects.

## **Setting the PC**

In cases where the program counter consists of the PC register and extension bits in SR register, the program counter can be set by the register PP.

Setting the PC causes the execution of a jump instruction. Pending REP instructions will be canceled.

## **Breakpoints in L: memory**

Breakpoints in L: memory will be set to the Y: memory class.

## **Breakpoints on second XAB access on 56100**

Must be set by using the Y: memory access class.

## **Program counter after SYStem.UP**

In some cases the program counter after System.Up is not placed at the begin of the boot flash.

## **Debugging with interrupts**

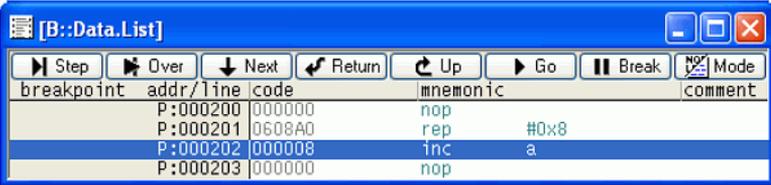
When IMASKHLL or IMASKASM is enabled the debugger won't update correctly the interrupt level bits in the SR register in case the core is placed into the highest priority.

## **JTAG Multi Core configuration**

The settings in the **SYStem.CONFIG** window are only active for 56300 with Multicore License or 56800 and 56800e core based chips with the license type DSP56800. You can view your license type with license.list.

## **Software Breakpoints in hardware supported REP-loops**

For the DSP56300 family setting a program software breakpoint on the instruction following a REP instruction and using the Go-command to continue will result in invalid loopcounter register LC. Code example:



breakpoint	addr/line	code	mnemonic	comment
P:000200		000000	nop	
P:000201		0608A0	rep	#0x8
P:000202		000008	inc	a
P:000203		000000	nop	

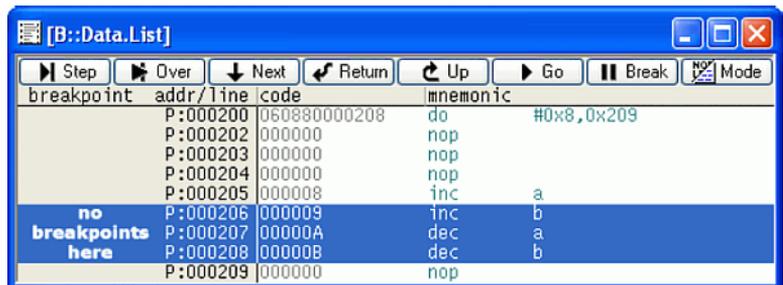
If using software breakpoints the original instruction on the breakpoint is replaced (invisible to the user) by a *debug* instruction. The DSP executes the *debug* instruction, decrements the loop counter register LC and then halts the system by entering debug mode. Using the Go-command the *debug* instruction is replaced by the original instruction which is then executed. This will decrement the loop counter register LC one more time which is invalid. To avoid this behavior do not use breakpoints within a hardware supported REP-loop.

### Software breakpoints in hardware supported DO-loops

For the DSP56300 family proper operation of DO-loop, DOR-loop or DO-forever-loop is only guaranteed if no software breakpoints are set near the end of the loop at loopaddress-2, loopaddress-1 or loopaddress.

In HLL debugging mode setting software breakpoints near the end of HLL-loops of any kind must be handled with care. To avoid this behavior use on-chip breakpoints for critical applications.

Code example:



If using software breakpoints the original instruction on the breakpoint is replaced (invisible to the user) by a *debug* instruction. The DSP halts after debug is executed. Prior to the next Go- or Step-command *debug* is replaced by the original instruction and the program counter is moved back onto the original instruction. Setting the program counter on instructions near the loop-end will decrement the loop counter register LC.

### Possible realtime violation using onchip program breakpoints

Using on-chip breakpoints of the DSP56300 family may violate realtime execution under the following conditions:

- Setting on-chip breakpoints on the next three instructions following conditional branches, jumps, loop-breaks or condition calls to subroutines.
- Setting on-chip breakpoints on the instruction that is a branch target of conditional branches, jumps, loop-breaks or conditional calls to subroutines if the conditional branch, jump, break or call is not taken.
- Setting on-chip breakpoints on instructions that are executed in parallel.
- Setting on-chip breakpoints on the instruction following a hardware supported REP-loop.

An onchip program breakpoint may get triggered even if the program counter is not directly on the given program memory location of the breakpoint. This is caused by the pipeline decode and prefetch mechanism and the parallel execution capabilities of the DSP.

If the DSP is halted by an on-chip breakpoint the debugger spots the current program counter and compares it with all on-chip breakpoint addresses. This is to decide which breakpoint has triggered. Under the above conditions sometimes no match can be found and the debugger continues execution. This behavior violates realtime execution and is signaled to the user in the state line by setting the spot breakpoint active flag. This behavior also applies to debugging in HLL mode.



To avoid realtime violation set on-chip breakpoints on instructions which not meet the above conditions or try using software breakpoints instead.

### **Range of fractional numbers**

For the DSP56300 family the range of fractional numbers which can be entered by using the command FPU.SET is limited to:

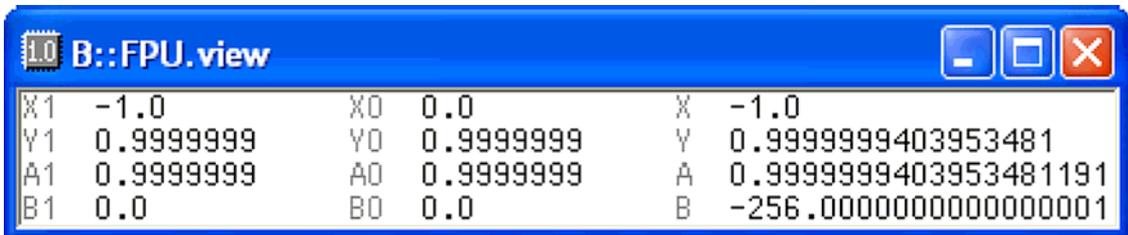
- -1.0 to 0.9999999999999929 for the registers X and Y.
  - -256.0 to 255.999999999999997 for the registers A and B.
- Entering numbers outside this ranges is not supported by the DSP and will lead to invalid displayed fractional numbers.

Format: **FPU.view**

Format: **FPU.Set** <register> <value>

**view** Display accumulator registers as fractional numbers.

**FPU.Set** Changes accumulator registers in fractional number format.



# TrOnchip Commands

The breakpoint registers on the ONCE debugger can be used to monitor data or program access in real-time. They are automatically set, when a read or write breakpoint is set. On the 56300 family it is possible to trigger on access sequences to two different addresses.

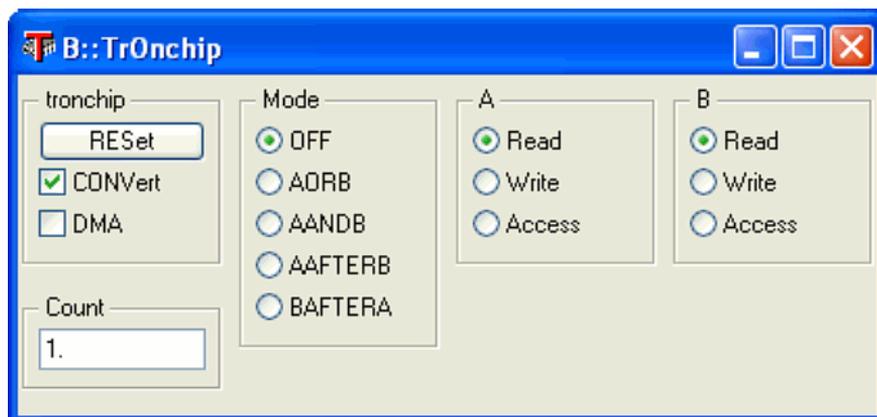
## TrOnchip.view

Opens configure panel

Format: **TrOnchip.view**

Control panel to configure the on-chip breakpoint registers.

DSP56300:



## TrOnchip.A

Trigger cycle

Format: **TrOnchip.A** <cycle>  
**TrOnchip.B** <cycle>

<cycle>: **Read**  
**Write**  
**Access**

Defines on which cycle the trigger system triggers (only 56300).



## **TrOnchip.BAFTERA** Triggers if event occurs first on unit B and then on unit A

---

Format: **TrOnchip.BAFTERA**

The on-chip breakpoint triggers, if an event occurs first on trigger unit B and then on trigger unit A (only 56300)

## **TrOnchip.CONVert** Automatically convert range to single address

---

Format: **TrOnchip.CONVert [ON | OFF]**

When enabled (default) the on-chip breakpoints are automatically converted from a range to a single address if required. If the switch is off, the system will only accept breakpoints which exactly fit to the on-chip breakpoint hardware.

## **TrOnchip.Count** Delay counter

---

Format: **TrOnchip.Count <count>**

Defines the delay counter for the trigger system. A value of 1 means no delay (only 56300).

## **TrOnchip.DMA** Trigger on DMA access

---

Format: **TrOnchip.DMA [ON | OFF]**

Trigger on DMA access instead of regular memory access (only 56300).

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

*<mode>*:  
**OFF**  
**AORB**  
**AANDB**  
**AAFTERB**  
**BAFTERA**

Defines which triggers are used and in what combination (only 56300). In **OFF** mode the triggers are used for the regular read/write breakpoints. In the other modes the **Alpha** and **Beta** breakpoints are used to define the memory addresses.

## TrOnchip.OFF

## Disable on-chip trigger unit

Format: **TrOnchip.OFF**

Disables on-chip trigger unit.

## TrOnchip.RESet

## Resets settings

Format: **TrOnchip.RESet**

Resets the trigger system to the default state.

## Floating Point Formats

---

<b>F24</b>	Fractional fixed point 24 bit
<b>F48</b>	Fractional fixed point 48 bit
<b>F16</b>	Fractional fixed point 16 bit
<b>F32</b>	Fractional fixed point 32 bit
<b>M56</b>	Floating point format (56002)
<b>leeeS</b>	Floating point format (56100)

**NOTE:** Fractional floating point numbers are always displayed with a fixed precision, i.e. a fixed number of digits. Small fractional numbers can have many non relevant digits displayed.

## Integer Access Keywords

---

<b>Word</b>	Word (16 bit)
<b>TByte</b>	Triple byte (24 bit)
<b>Long</b>	Double Word (32 bit), upper and lower word swapped
<b>HByte</b>	Hexabyte (48 bit)
<b>Quad</b>	Tertiary Word (64 bit), upper and lower word swapped

# ONCE Connector (56002/56100)

---

This connector is obsolete.

Signal	Pin	Pin	Signal
DSI	1	2	GND
DSO	3	4	GND
DSCK	5	6	GND
DR-	7	8	VCC
RESET-	9	10	GND

# JTAG Connector (56300, 56800, 56800E)

Signal	Pin	Pin	Signal
TDI	1	2	GND
TDO	3	4	GND
TCK	5	6	GND
N/C	7	8	KEY
RESET-	9	10	TMS
VCCS	11	12	N/C
DE-	13	14	TRST-

Pins	Con- nection	Description	Recommendations
1	TDI	Test Data In	Not 56300: If there are multiple chip devices on the JTAG chain, connect TDI to the TDO signal of the previous device in the chain.
2,4,6	GND	System Ground Plan	Connect to digital ground.
3	TDO	Test Data Out	Not 56300: If there are multiple chip devices on the JTAG chain, connect TDO to the TDI signal of the next device in the chain.
5	TCK	Test Clock	Add 10 k $\Omega$ pull-up resistor to VCC.
7, 12	NC	No Connect	Leave unconnected.
8	KEY	Mechanical Keying	Pin should be removed.
9	$\overline{\text{RESET}}$	Reset	May be tied to HRESET.
10	TMS	Test Mode Select	None.
11	VCCS	VCC Sense	Connect to Chip I/O voltage VDDH through a 10 $\Omega$ current limiting resistor.

14	$\overline{\text{TRST}}$	Test Reset	TRST has an internal pull-up resistor, so no external pull-up or pull-down resistor is required. However, a 10 k $\Omega$ pull-down resistor should added to GND on this signal to keep the JTAG in reset mode while the device is operating regularly. When using more than one debug dongle driving this signal it is not recommended to pull down the signal in debug mode, because during the dongle source switch the signal output is set to tristate.
13	$\overline{\text{DE}}$	Debug Enable	Add 10 k $\Omega$ pull-up resistor to VCC. This Signal is not needed. If not available, set SYSTEM.Option.DE OFF.

# Memory Classes

---

<b>Memory Class</b>	<b>Description</b>
X	X: data memory space
Y	Y: data memory space or second XAB access on 56100 for breakpoints (not 56800, not 56800E)
L	L: data memory space which is X:Y chained data memory (not 56100, not 56800, not 56800E)
P	P: programm memory space

## Available Tools

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
DSP56002			YES				
DSP56004			YES				
DSP56005			YES				
DSP56156			YES				
DSP56166			YES				
MC68356	YES		YES	YES			YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
DSP56301			YES				
DSP56303			YES				
DSP56307			YES				
DSP56309			YES				
DSP56311			YES				
DSP56321			YES				
DSP56362			YES				
DSP56364			YES				
DSP56366			YES				
DSP56367			YES				
DSP56371			YES				
DSP56374			YES				
DSP56720			YES				
DSP56721			YES				

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
DSP56824			YES				
DSP56852			YES				
DSP56853			YES				
DSP56854			YES				
DSP56855			YES				
DSP56857			YES				
DSP56858			YES				
DSP56F801			YES				
DSP56F802			YES				
DSP56F803			YES				
DSP56F805			YES				
DSP56F807			YES				
DSP56F8122			YES				
DSP56F8123			YES				
DSP56F8135			YES				
DSP56F8145			YES				
DSP56F8146			YES				
DSP56F8147			YES				
DSP56F8155			YES				
DSP56F8156			YES				
DSP56F8157			YES				
DSP56F8165			YES				
DSP56F8166			YES				
DSP56F8167			YES				
DSP56F826			YES				
DSP56F827			YES				
DSP56F8322			YES				
DSP56F8323			YES				
DSP56F8335			YES				
DSP56F8345			YES				
DSP56F8346			YES				
DSP56F8347			YES				
DSP56F8355			YES				
DSP56F8356			YES				
DSP56F8357			YES				
DSP56F8365			YES				
DSP56F8366			YES				
DSP56F8367			YES				

## Compilers DSP56000

---

Language	Compiler	Company	Option	Comment
ASM	GAS	NXP Semiconductors	MCOFF	Source level debugging
C	GCC56K	NXP Semiconductors	MCOFF	
C	C56	TASKING	IEEE	

## Compilers DSP56300

---

Language	Compiler	Company	Option	Comment
C	GCC56300	NXP Semiconductors	MCOFF	
C	C563	TASKING	IEEE	

## Compilers DSP56800/DSP56800E

---

Language	Compiler	Company	Option	Comment
C	CODEWARRIOR	NXP Semiconductors	ELF/DWARF	
C	CODEWARRIOR	NXP Semiconductors	ELF/DWARF	/LARGE for LDM (800E only) /AS

No operation systems supported.

## 3rd Party Tool Integrations

CPU	Tool	Company	Host
ALL	ADENEO	Adeneo Embedded	
ALL	X-TOOLS / X32	blue river software GmbH	Windows
ALL	CODEWRIGHT	Borland Software Corporation	Windows
ALL	CODE CONFIDENCE TOOLS	Code Confidence Ltd	Windows
ALL	CODE CONFIDENCE TOOLS	Code Confidence Ltd	Linux
ALL	EASYCODE	EASYCODE GmbH	Windows
ALL	ECLIPSE	Eclipse Foundation, Inc	Windows
ALL	RHAPSODY IN MICROC	IBM Corp.	Windows
ALL	RHAPSODY IN C++	IBM Corp.	Windows
ALL	CHRONVIEW	Inchron GmbH	Windows
ALL	LDRA TOOL SUITE	LDRA Technology, Inc.	Windows
ALL	UML DEBUGGER	LieberLieber Software GmbH	Windows
ALL	ATTOL TOOLS	MicroMax Inc.	Windows
ALL	VISUAL BASIC INTERFACE	Microsoft Corporation	Windows
ALL	LABVIEW	NATIONAL INSTRUMENTS Corporation	Windows
ALL	CODE::BLOCKS	Open Source	-
ALL	C++TEST	Parasoft	Windows
ALL	RAPITIME	Rapita Systems Ltd.	Windows
ALL	DA-C	RistanCASE	Windows
ALL	TRACEANALYZER	Symtavision GmbH	Windows
ALL	SIMULINK	The MathWorks Inc.	Windows

<b>CPU</b>	<b>Tool</b>	<b>Company</b>	<b>Host</b>
ALL	TA INSPECTOR	Timing Architects GmbH	Windows
ALL	UNDODB	Undo Software	Linux
ALL	VECTORCAST UNIT TESTING	Vector Software	Windows
ALL	VECTORCAST CODE COVERAGE	Vector Software	Windows
ALL	WINDOWS CE PLATF. BUILDER	Windows	Windows

## Product Information

---

OrderNo Code	Text
<b>LA-7730</b> BDM-DSP56K	<b>BDM Debugger for DSP56K (ICD)</b> supports DSP56002/4/5, DSP561xx and DSP-part of 68356 (only for chips with 5V tolerant IO's) includes software for Windows, Linux and MacOSX requires Power Debug Module debug cable with 10 pin connector

OrderNo Code	Text
<b>LA-7735</b> BDM-DSP56300	<b>BDM Debugger for DSP56300 (ICD)</b> supports DSP5630x includes software for Windows, Linux and MacOSX requires Power Debug Module not supported by PowerDebug PRO debug cable with 14 pin connector

OrderNo Code	Text
<b>LA-7738</b> BDM-DSP56800	<b>BDM Debugger for DSP56800 and DSP56800E (ICD)</b> supports DSP568xx and DSP5683xx includes software for Windows, Linux and MacOSX requires Power Debug Module Not supported by PowerDebug PRO debug cable with 14 pin connector

## Order Information

---

Order No.	Code	Text
LA-7730	BDM-DSP56K	BDM Debugger for DSP56K (ICD)

Order No.	Code	Text
LA-7735	BDM-DSP56300	BDM Debugger for DSP56300 (ICD)
<b>Additional Options</b>		
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging

Order No.	Code	Text
LA-7738	BDM-DSP56800	BDM Debugger for DSP56800 and DSP56800E (ICD)